

# Приключение в стиле Python



От основ Python к  
работающему веб-приложению

Павел Анни

# *A Pythonic Adventure*

ОТ ОСНОВ PYTHON К  
РАБОТАЮЩЕМУ ВЕБ-ПРИЛОЖЕНИЮ

ПАВЕЛ АННИ



MANNING

SHELTER ISLAND

Чтобы получить онлайн-информацию и заказать эту и другие книги Мэннинга, посетите сайт [www.manning.com](http://www.manning.com). Издательство предлагает скидки на эту книгу при заказе большого количества. Для получения более подробной информации, пожалуйста, свяжитесь с

Special Sales Department  
Manning Publications Co.  
20 Baldwin Road  
PO Box 761  
Shelter Island, NY 11964  
Email: [orders@manning.com](mailto:orders@manning.com)


©2024, Manning Publications Co. Все права защищены.

Никакая часть этой публикации не может быть воспроизведена, сохранена в поисковой системе или передана в любой форме или с помощью электронных, механических, фотокопировальных или иных средств без предварительного письменного разрешения издателя.

Многие обозначения, используемые производителями и продавцами для обозначения своей продукции, заявлены как товарные знаки. Если эти обозначения встречаются в книге и Manning Publications было известно о претензии на товарный знак, обозначения были напечатаны заглавными буквами или полностью.

© Признавая важность сохранения написанного, политика Мэннинга заключается в том, чтобы книги, которые мы публикуем, печатались на бескислотной бумаге, и мы прилагаем все усилия для достижения этой цели. Признавая также нашу ответственность за сохранение ресурсов нашей планеты, книги Мэннинга печатаются на бумаге, которая как минимум на 15 процентов переработана и обработана без использования элементарного хлора.

Автор и издатель приложили все усилия, чтобы информация в этой книге была верной на момент публикации. Автор и издатель не принимают на себя и настоящим отказываются от какой-либо ответственности перед какой-либо стороной за любые убытки, ущерб или сбой, вызванные ошибками или упущениями, независимо от того, являются ли такие ошибки или упущения результатом небрежности, несчастного случая или любой другой причины, или любого использования информация здесь.

 Manning Publications Co.  
20 Baldwin Road  
PO Box 761  
Shelter Island, NY 11964

Development editor: Toni Arritola  
Technical editor: Marie-Therese Smith  
Review editor: Adriana Sabo  
Production editor: Andy Marinkovich  
Copy editor: Andy Carroll  
Proofreader: Mike Beady  
Typesetter: Gordan Salinovic  
Cover designer: Monica Kamswaag

ISBN 9781633438897

Напечатано в Соединенных Штатах Америки

*Посвящается моему папе*

## *краткое содержание*

---

1	■ Кофе для друзей: Первые шаги	1
2	■ Списки: Что в меню?	19
3	■ Функции: Не повторяйтесь!	34
4	■ Ошибки пользователя: все совершают ошибки	46
5	■ Работа с файлами: Быть менеджером кофейни	62
6	■ Главное меню: Следующий клиент!	73
7	■ Создание функций: Получить заказ и распечатать его	86
8	■ Работаем с JSON: сохраняем заказ	95
9	■ Дополните меню: Настоящая программа	107
10	■ Изучаем Flask: ваше первое веб-приложение	116
11	■ Веб-форма для заказов: Кофейня в Интернете	136
12	■ База данных: Нам нужно хорошее хранилище	148
13	■ Стили: Делаем красиво	166
14	■ Помощь искусственного интеллекта: улучшение нашего кода	188
15	■ Следующие шаги: Планы на будущее	197

# содержание

---

предисловие	viii
слова благодарности	x
об этой книге	xii
об авторе	xvi

<b>1</b>	<b>Кофе для друзей: Первые шаги</b>	<b>1</b>
	Прежде всего: установка	4
	Как общаться с компьютером	7
	Что такое программа?	17
	Новое, что вы узнали сегодня	17
	Код для этой главы	18
	Больше идей для ваших проектов	18
<b>2</b>	<b>Списки: Что в меню?</b>	<b>19</b>
	Новое, что вы узнали сегодня	33
	Код для этой главы	33
<b>3</b>	<b>Функции: Не повторяйтесь!</b>	<b>34</b>
	Новое, что вы узнали сегодня	45
	Код для этой главы	45

4	<i>Ошибки пользователя: все совершают ошибки</i>	46
	Новое, что вы узнали сегодня	61
	Код для этой главы	61
5	<i>Работа с файлами: Быть менеджером магазина</i>	62
	Новое, что вы узнали сегодня	71
	Код для этой главы	72
6	<i>Главное меню: Следующий клиент!</i>	73
	Новое, что вы узнали сегодня	85
	Код для этой главы	85
7	<i>Создание функций: Получить заказ и распечатать его</i>	86
	Каков ваш выбор?	89
	Распечатайте это!	92
	Новое, что вы узнали сегодня	94
	Код для этой главы	94
8	<i>Работаем с JSON: сохраняем заказ</i>	95
	Новое, что вы узнали сегодня	106
	Код для этой главы	106
9	<i>Дополните меню: Настоящая программа</i>	107
	Новое, что вы узнали сегодня	115
	Код для этой главы	115
10	<i>Изучаем Flask: ваше первое веб-приложение</i>	116
	Новое, что вы узнали сегодня	134
	Код для этой главы	135
11	<i>Веб-форма для заказов: Кофейня в Интернете</i>	136
	Новое, что вы узнали сегодня	147
	Код для этой главы	147
12	<i>База данных: Нам нужно хорошее хранилище</i>	148
	Новое, что вы узнали сегодня	164
	Код для этой главы	165

13	<i>Стили: Делаем красиво</i>	166
	Новое, что вы узнали сегодня	186
	Код для этой главы	187
14	<i>Помощь ИИ: улучшение нашего кода</i>	188
	Новое, что вы узнали сегодня	196
15	<i>Следующие шаги: Планы на будущее</i>	197
	Новое, что вы узнали сегодня	207
	Код для этой главы	207
	<i>приложение А</i>	
	<i>Идеи для вашего первого приложения</i>	209
	<i>приложение В</i>	
	<i>Как установить редактор Ми и среду Python</i>	212



## предисловие

---

Эта книга основана на реальной истории. Мой сын Эрик любил ходить в Starbucks и пробовать напитки с разными вкусами и топингами. Однажды он решил сам приготовить напитки и угостить друзей. Он взял свой планшет, чтобы собрать у них заказы, но я предложил ему создать для этого простую программу. Раньше он пытался изучать программирование, но большинство упражнений были для него скучными. На этот раз он увидел реальную проблему, которую можно было решить с помощью программирования, и заинтересовался.

Так началась эта книга. Я надеюсь, что вы найдете свою интересную задачу, которую можно решить с помощью программирования, и надеюсь, что эта книга вам в этом поможет.

Вот несколько советов:

- *Не торопитесь* — я понимаю ваше желание сразу перейти к последней главе, загрузить код финальной версии программы и запустить ее. Не делай этого. Идите шаг за шагом, напишите код самостоятельно (пожалуйста, не копируйте его и не вставляйте!), попробуйте его и двигайтесь вперед. Иногда вам, возможно, придется вернуться и перечитать главу. Иногда вам понадобится перерыв. Не волнуйтесь! Сделайте перерыв и повторите главу — только не бросайте.
- *Делайте ошибки* — Вы не учитесь, когда все идет идеально. Единственный способ учиться — это совершать ошибки. Не бойтесь ошибок. Экспериментируйте с кодом, меняйте что-то, получайте сообщения об ошибках и читайте о них. Найдите сообщение об ошибке в Интернете и найдите тысячи других людей, совершивших ту же ошибку. Узнайте, как они ее исправили, а затем исправьте свою. Двигайтесь вперед и не бросайте.

- *Задавайте вопросы* — Спрашивайте своих друзей, родителей, бабушек и дедушек, спрашивайте в Интернете. Объясните кому-нибудь свою проблему — иногда этого достаточно, чтобы найти решение самостоятельно. Не бывает «глупых вопросов» — не стесняйтесь. Задавайте вопросы и не опускайте руки.
- *Идите дальше* — Модифицируйте приложение, которое вы создадите с помощью этой книги. Измените что-либо, чтобы оно больше походило на ваше собственное приложение. Подумайте о других приложениях, которые вы можете создать. Посмотрите вокруг: что можно автоматизировать? Можете ли вы создать приложение, похожее на знакомое вам приложение или веб-сайт? Расскажите о своих идеях друзьям — возможно, вы вместе что-нибудь создадите. Программирование — это круто. Не бросайте учебу.
- *Высказывайтесь* — Поделитесь своим опытом на дискуссионном форуме liveBook. Поделитесь своими мыслями и идеями с друзьями. Обсуждение решенных вами проблем сделает ваше собственное приключение в стиле Python намного интереснее.

## *слова благодарности*

---

Это моя первая книга, и я не смог бы написать ее без неоценимой поддержки и советов со стороны многих людей.

В первую очередь хочу выразить благодарность моим родителям. Мой отец, твердо веривший в важность STEM-образования, наполнил наш дом научно-популярными книгами и посоветовал мне изучать программирование. Он проницательно предсказал, что это станет второй грамотностью, и он не мог быть более точным. Моя мама, пожизненная учительница физики, щедро передала мне свои педагогические гены.

Я также обязан поблагодарить своих сыновей Саймона и Эрика за предоставленную мне возможность научить их Python. Этот бесценный опыт позволил мне учиться и расти как педагогу и программисту.

Моей жене Татьяне я выражаю сердечную признательность за ее огромное терпение и непоколебимую поддержку. Она стойко переносила многие выходные, когда я был сосредоточен на письме на клавиатуре, вместо того, чтобы наслаждаться ее восхитительной компанией. На протяжении всего процесса она выслушивала мои мысли и идеи, предлагая бесценные советы и свою точку зрения.

Я выражаю благодарность моим редакторам из Мэннинга: Трою Дрейеру, Тони Арритолу и Мари-Терез Смит, которые сыграли важную роль в доведении этой книги до нужной кондиции. Трой осознал потенциал моего раннего черновика и принес его на рассмотрение в Мэннинг, без чего эта книга никогда бы не была начата. Мягкая, но настойчивая поддержка Тони обеспечила своевременное завершение книги. Без нее эта книга никогда бы не была закончена. Мария-Тереза, работавшая преподавателем, инженером и специалистом по обработке данных, тщательно проверяла код и технические детали, не позволяя читателям наткнуться на самый первый пример кодирования.

Я хотел бы выразить признательность Николасу Х. Толлерви, основному автору редактора `Mu`, и всем участникам этого фантастического проекта с открытым исходным кодом. Эта удобная для начинающих среда разработки `Python` предоставила мне все необходимое для путешествия по этой книге: от первых шагов в `Python` до разработки полнофункционального веб-приложения.

Всем рецензентам: Афифу Хайдеру, Аль Пежевски, Алану Хил Форза, Алексису Перье, Амог Рагунату, Эндрю Р. Фриду, Арсалан Хан, Арья Арун Кумар, Жан-Батист Банг Нтеме, Бен Макнамара, Бернард Фуэнтес, Крис Карделл, Кристиан Саттон, Клара МакАдамс, Клеменс Баадер, Дэвид Кухта, Дивья Сивасубраманиан, Элона Виалатте, Файяз Эван Хейдер, Ганеш Свамнатан, Джордж Томас, Грег Фрид, Ханна Чунг, Хасин Хейдер, Джексон МакНамара, Яна Драгосавлевич, Джанит Анджария, Химена де Хесус Мата Кобиан, Жоао Динис Феррейра, Кит Ким, Кевин Чунг, Мафинар Хан, Марк-Энтони Тейлор, Мэри Энн Тигесен, Мигель Эдуардо Хиль Биро, Натан-Стивен Тейлор, Ник Пипенбрейер, Нинослав Черкез, Ор Голан, Павел Шимон, Филипп Виалат, Регина де Хесус Мата Кобиан, Роберт Кулаговски, Рупа Лахири, Сатадуру Рой, Шаурья Дара, Шиванш Батра, Шион Рой, Шраванти Редди, Шриша Редди, Шриша Тиммаредди, Судип Батра, Сумит Бхаттачарья, Тарун Ганеш, Тавхида Хусейн, Уолтер Александр Мата Лопес, Уилл Пежевски и Вольфи Баадер, ваши предложения помогли сделать эту книгу лучше.

Наконец, я выражаю сердечную благодарность всему сообществу `Python`. Богатство знаний, которое я получил за эти годы из подкастов, видео, выступлений на конференциях, книг и онлайн-форумов, неизмеримо. Гостеприимство и поддержка этого сообщества, несомненно, являются важной причиной того, почему разработчики так ценят `Python`. Спасибо вам всем!

## об этой книге

---

Я написал эту книгу для подростков, которые хотят изучать программирование с помощью практических проектов. В книге вы найдете серию неформальных диалогов между двумя братьями, вместе работающими над приложением Coffee Shop. Следите за их беседами и вместе с ними разрабатывайте собственное приложение.

### ***Кому стоит прочитать эту книгу?***

Возможно, вы уже посещали уроки программирования — онлайн или в школе. Вы знаете основы, но еще не пробовали создать реальный проект.

Возможно, вы еще не начали программировать, но умеете пользоваться компьютером и хотите узнать, как работает программирование.

Возможно, вы использовали некоторые онлайн-приложения и хотите создать свое собственное.

Возможно, ваш старший брат или сестра уже создают компьютерные программы и приложения, и вы тоже хотите научиться программированию.

Если вы являетесь кем-то из вышеперечисленных — отлично! С помощью этой книги вы шаг за шагом вместе с Эриком и Саймоном за пару недель создадите собственное приложение.

Если вы (бабушка)родитель и вам нужно простое руководство по изучению программирования вместе с (внуками) детьми, эта книга тоже для вас! Даже если вы хотите научиться программированию самостоятельно, не думайте, что эта книга предназначена только для детей. Приглашаем всех изучать Python и создавать приложения!

## *Как организована эта книга: план действий*

В каждой главе этой книги рассказывается о том, что Эрик и Саймон (а позже и Эмили) создали за один день. Они сидели вместе около часа каждый день, обсуждали свое приложение и писали код. Вы можете сделать то же самое — читать по главе в день и писать код для этой главы. Если у вас есть вопросы, задавайте их на форуме книги или поищите ответ на него в Интернете.

В книге 15 глав.

Часть 1 состоит из первых 9 глав, в которых Эрик и Саймон работают над первой версией своего текстового приложения. Это означает, что оно работает как чат, спрашивая клиентов, что они хотят заказать, и ожидая их ответы.

В главе 1 у Эрика рождается идея создания своего приложения. Он хочет приготовить кофейные напитки для своих друзей, и ему приходится собирать от них заказы. Саймон помогает ему создать первую версию приложения.

В главе 2 Эрик узнает о списках. Списки — хороший способ организовать меню с различными вкусами и топингами.

В главе 3 Эрик замечает, что повторяет свой код несколько раз. Саймон предлагает использовать функции, чтобы избежать этого. Код приложения становится короче и легче читается.

В главе 4 Эрик и Саймон работают над ошибками пользователей. Что, если клиент допустит ошибку при вводе заказа? Их приложение должно позаботиться об этом.

В главе 5 Саймон учит Эрика пользоваться файлами. Теперь менеджер магазина может редактировать текстовые файлы, добавляя или удаляя вкусы или топинги из меню.

В главе 6 братья создают главное меню приложения. Это позволяет им непрерывно запускать приложение и обслуживать одного клиента за другим.

В главе 7 Эрик и Саймон создают дополнительные функции. Они используют функции для получения заказа и его вывода на печать.

В главе 8 Эрик узнает о JSON. Саймон объясняет, как использовать файл JSON для хранения заказов.

В главе 9 братья завершают текстовую версию приложения. Они обсуждают, как сделать его веб-версию.

Во второй части (главы 10–15) Эрик, Эмили и Саймон работают над веб-версией своего приложения. Они создают веб-меню, подключают базу данных и работают над стилями и цветами. Наконец, они тестируют свое приложение на смартфонах и планшетах.

В главе 10 к братьям присоединяется Эмили. Саймон показывает Эмили и Эрику, как создать очень простое веб-приложение. Они учатся использовать веб-формы и меню.

В главе 11 Эмили и Эрик работают над созданием веб-форм и меню для своей кофейни.

В главе 12 Саймон объясняет Эмили и Эрику, как использовать базы данных для хранения заказов. Они учатся использовать для этого язык структурированных запросов (SQL).

В главе 13 друзья начинают делать свое веб-приложение красивым. Они учатся использовать на веб-страницах шрифты, цвета и изображения.

В главе 14 Саймон показывает Эмили и Эрику, как использовать искусственный интеллект (ИИ) для улучшения кода приложения, написания комментариев и объяснения сообщений об ошибках.

В главе 15 Эмили, Эрик и Саймон делают приложение доступным для своих смартфонов и планшетов и тестируют его. Оно работает! Они обсуждают свои следующие шаги: как сделать приложение доступным в облаке и как добавить в него больше функций. Они решают продолжить работу над приложением, но использовать веб-сайт для обмена идеями.

Я предлагаю читать эту книгу как рассказ, главу за главой. Пожалуйста, не пропускайте части, помеченные как «ВАШ ХОД». Вам придется написать код, который написали Эрик, Эмили и Саймон, но по-своему. Используйте свое воображение, меняйте вещи и создайте свое собственное уникальное приложение! Иногда вам захочется сделать перерыв. Иногда вам захочется перечитать главу, чтобы лучше ее понять. Это нормально. Читайте книгу в своем темпе — только не бросайте!

Если вы застряли на каком-то фрагменте кода, попробуйте скопировать его из нашего репозитория кода (обсуждается далее). Если что-то не понятно, задайте вопрос на форуме liveBook (обсуждается чуть позже).

## О коде

Исходный код этой книги можно найти на GitHub: <https://github.com/pavelanni/pythonicadventure-code>. В конце каждой главы есть ссылка на код этой главы. Вы также можете получить исполняемые фрагменты кода из liveBook (онлайн) версии этой книги по адресу <https://livebook.manning.com/book/a-pythonic-adventure>.

Эта книга содержит множество примеров исходного кода, как в виде нумерованных списков, так и в виде встроенного в обычный текст. В обоих случаях исходный код форматируется таким шрифтом фиксированной ширины, чтобы отделить его от обычного текста.

Во многих случаях исходный код был переформатирован; мы добавили разрывы строк и переработали отступы, чтобы они соответствовали доступному пространству страницы в книге.

Кроме того, комментарии в исходном коде часто удаляются, когда код описывается в тексте. Многие листинги сопровождаются аннотациями к коду, подчеркивающими важные концепции.

## Дискуссионный форум liveBook

Покупка *Приключение в стиле Python* включает бесплатный доступ к liveBook, платформе онлайн-чтения Мэннинга. Используя эксклюзивные функции обсуждения liveBook, вы можете добавлять комментарии ко всей книге, либо к ее определенным разделам или абзацам. Делать заметки для себя, задавать технические вопросы и отвечать на них, а также получать помощь от автора и других пользователей — совсем несложно. Чтобы получить доступ к форуму, перейдите по адресу <https://livebook.manning.com/book/a-pythonic-adventure/discussion>. Вы также можете узнать больше о форумах Мэннинга и правилах поведения по адресу <https://livebook.manning.com/discussion>.

Обязательство Мэннинга перед нашими читателями состоит в том,



чтобы предоставить место, где может состояться содержательный диалог между отдельными читателями, а также между читателями и автором. Это не обязательство к какому-то конкретному объему участия со стороны автора, чей вклад в форум остается добровольным (и неоплачиваемым). Предлагаем вам попробовать задать автору несколько сложных вопросов, чтобы его интерес не угас! Форум и архивы предыдущих обсуждений будут доступны на сайте издателя, пока книга находится в печати.

## **Другие онлайн-ресурсы**

У книги есть сопутствующий сайт: <https://pythonicadventure.com>. На этом сайте мы собрали некоторые идеи для других подобных проектов (на случай, если вам не понравится проект кофейни), информацию по устранению неполадок и идеи по улучшению приложения.

Лучшим источником информации о Python является официальный сайт: <https://www.python.org>. Там вы найдете всю необходимую документацию, блоги, статьи и учебные пособия. Одним из самых полезных сайтов для новичков является коллекция ссылок на другие ресурсы: <https://wiki.python.org/moin/BeginnersGuide>.

## **Идите дальше**

Прочитав эту книгу, вы будете готовы более серьезно работать с Python. Продолжая свое приключение в стиле Python, прочтите следующие книги:

- *Tiny Python Projects*, автор Ken Youens-Clark  
(<https://www.manning.com/books/tiny-python-projects>)
- *Python Workout*, автор Reuven M. Lerner  
(<https://www.manning.com/books/python-workout>)
- *Practices of the Python Pro*, автор Dane Hillard  
(<https://www.manning.com/books/practices-of-the-python-pro>)

## *об авторе*

---



**ПАВЕЛ АННИ** — главный инженер по работе с клиентами в SambaNova Systems. Прежде чем присоединиться к этому стартапу в области искусственного интеллекта, Павел работал в Sun Microsystems, Oracle и Red Hat. В своей карьере его основной ролью было обучение людей и популяризация новых технологий. Он разработал и провел курсы по языкам программирования, операционным системам Unix и Linux, Kubernetes и другим темам.

# Кофе для друзей: Первые шаги

---

## **В этой главе**

- Эрику приходит в голову идея
- Эрик и Саймон обсуждают будущее приложение
- Эрик устанавливает редактор кода и пытается запустить свою первую программу на Python
- Саймон объясняет, как использовать переменные
- Эрик пишет свой первый диалог на Python

Все началось в один из солнечных летних дней. Эрик пришел домой с идеей: он захотел приготовить кофейные напитки для своих друзей. Кто знал, что для этого он создаст собственное онлайн-приложение?

«Сделаю, как в Старбаксе, со множеством вкусов и топингов», — подумал он. «Думаю, у меня есть все, что мне нужно: кофе, три или четыре вкуса, которые можно добавить, и немного шоколадного крема для топинга. Великолепно!»

«Где мой iPad?» он спросил своего старшего брата Саймона.

«Там, где ты его оставил. А что?»

«Он мне нужен, чтобы собирать заказы для моей кофейни!»

Через несколько минут он вернулся с заметками на своем iPad, приготовил четыре напитка для друзей и снова ушел.

«Разве это не удачная идея?» спросил он Саймона, когда тот пришел домой с четырьмя пустыми пластиковыми стаканчиками.

«Да, отличная идея», — сказал Саймон. "Но . . .»

«Какое «но»?» — спросил Эрик. Он чувствовал, что его старший брат как всегда хочет испортить ему день.

«Ты использовал iPad для приема заказов, но использовали его просто как обычный блокнот. Ты можешь создать простое приложение для своей кофейни и использовать его для приема заказов»

«Ты имеешь в виду, как в интернет-магазине? С меню и всем этим?» — Эрик уже представлял себе собственный интернет-магазин с огромным заголовком сверху: «Erik's Coffee Shop».

«Да, именно так. Ты немного знаешь Python из того онлайн-курса, который ты прошел, не так ли?»

«Да, но я мало что помню. Мы сделали всего несколько упражнений. . . Думаю, будет сложно сделать его похожим на настоящий интернет-магазин.»

«Не волнуйся», — сказал Саймон. «Мы будем делать это шаг за шагом. Я сделал несколько подобных проектов для своей школьной команды по робототехнике.»

**ПРИЧАНИЕ** Не волнуйтесь, если у вас вообще нет опыта программирования. Эрик и так мало что помнил из занятий, поэтому начнем с самого начала.

Саймон учится в последнем классе средней школы. Он выучил Python несколько лет назад и использовал его в школьном клубе компьютерных наук, а совсем недавно — в своей команде по робототехнике.

«Так ты утверждаешь, что мы можем создать настоящее онлайн-приложение?» — Эрика это не убедило.

«Да, конечно. Если ты не бросишь учебу», — улыбнулся Саймон, — «ты сделаешь его за пару недель. Тогда твои клиенты смогут выбрать любой напиток, который они хотят, и добавить вкусы.»

«И топинги!», — добавил Эрик.

«Да, и топинги. А после того, как они подтвердят заказ, ты увидишь его на странице заказов. Ты будешь знать, что готовить и для кого. Что-то вроде этого», — и Саймон взял лист бумаги и начал рисовать простую веб-страницу. «Это будет страница вашего заказа.»

Erik's Coffee Shop

Enter your name:

Choose your drink Choose your flavor Choose your topping

☐ Coffee  
☐ Chocolate  
☒ Decaf

☐ Caramel  
☐ Marshmallow  
☒ Strawberry

☐ Chocolate  
☐ Caramel  
☒ Cinnamon  
☐ Vanilla

Order

Cancel

«А это будет список твоих заказов».

Erik's Coffee Shop

Orders

Name	Product	Flavor	Topping
Alex	Coffee	Caramel	Vanilla
Drew	Decaf	Strawberry	Chocolate

"Круто! Как ты думаешь, мы сможем это сделать?», — Эрик все еще не мог поверить своему брату.

"Конечно! Как я уже сказал, просто не бросай учебу. У тебя будет достаточно времени, чтобы закончить это во время летних каникул»

**ПРИМЕЧАНИЕ** Есть несколько других идей проектов, которые вы можете использовать, если идея с кофейней вам не нравится. Некоторые из них будут обсуждаться, когда подруга Эрика Эмили присоединится к осуждению в последующих главах. Подробности ищите в приложении А.

## Прежде всего: Установка

«Начнем с простых вещей. Ты очень быстро запомнишь Python. Он установлен у тебя на ноутбуке?» — спросил Саймон.

«Нет, я так не думаю».

«Вот отличный редактор Python, созданный специально для таких новичков, как ты. Он называется Му. Попробуй найти его и установить. Ты сможешь это сделать сам, я уверен».

Эрик нашел сайт, где можно было его скачать: <https://codewith.mu/>.




Он скачал программу-установщик со страницы Download: <https://codewith.mu/en/download>.

Он нажал кнопку Instructions и нашел страницу инструкций со всеми шагами для своего компьютера.

«Не волнуйся, это не игрушка. Это идеальный редактор», — сказал Саймон. «Мы используем его в нашей команде по робототехнике для работы с микроконтроллерами. Как видишь, есть версии для Windows, macOS и Linux. Я со своей командой использую версию Linux».


**ПРИМЕЧАНИЕ** Все необходимые ссылки и инструкции вы найдете в приложении В.


 [Download](#) [About](#) [Tutorials](#) [How to..?](#) [Discuss](#) [Developers](#) [Language ▾](#)


## Download Mu

The simplest and easiest way to get Mu is via the official installer for Windows or Mac OSX (we no longer support 32bit Windows). We also have an experimental Appliance for Linux users running on Intel based hardware.

The current recommended version is Mu 1.2.0. We advise people to update to this version via the links for each supported operating system. All previous beta versions of Mu [can be downloaded from here](#).

**Windows Installer**  
[Download](#) [Instructions](#)

**Mac OSX Installer**  
[Download](#) [Instructions](#)

**Linux Appliance Package (Experimental)**  
[Download](#) [Instructions](#)

«Есть ли другие редакторы для Python?», — Эрик не хотел просто следовать указаниям брата.

«Да, конечно, их много. Еще один хороший вариант для новичков — Thonny. Посмотри здесь: <https://thonny.org/>».

"Мне нравится!", — сказал Эрик. — И название забавное».

«И, конечно же, есть и другие редакторы кода, которые работают на любой платформе:

- VS Code (<https://code.visualstudio.com/>)
- Sublime Text (<http://www.sublimetext.com/>)

«Все они прекрасно работают с Python. Даже очень старые редакторы, такие как Vim (<https://www.vim.org/>) и Emacs (<https://www.gnu.org/software/emacs/>), поддерживают Python, но нужно быть *очень серьезным* программистом, чтобы их использовать», — и Саймон подмигнул брату.

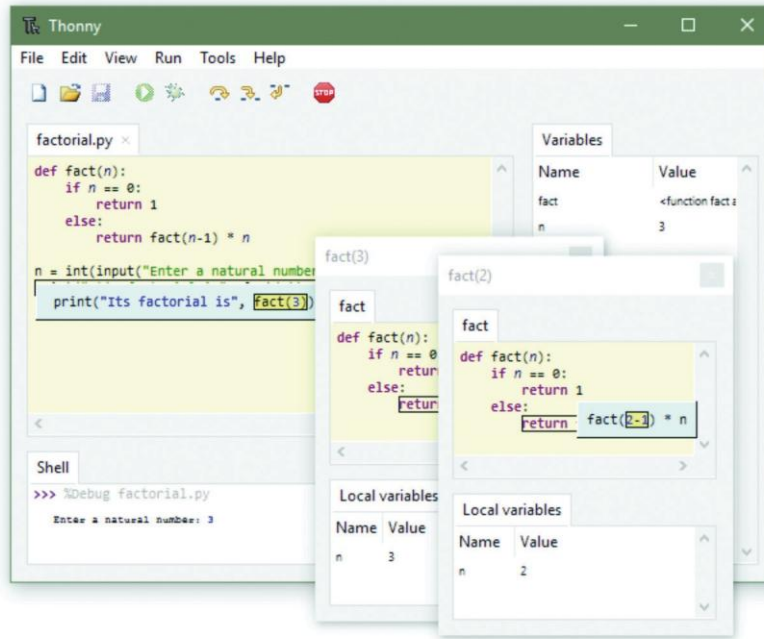
«И Mu, и Thonny», — продолжил Саймон, — «*включают* в установочный пакет Python. Чтобы использовать Python с *другими* редакторами, тебе необходимо будет сначала его *установить*. В некоторых системах, таких как Linux и macOS, Python уже присутствует с самого начала. В случае Windows тебе необходимо его установить. Если хочешь, позже я могу показать тебе, как это сделать.

# Thonny

Python IDE for beginners



Download version **4.0.1** for  
Windows • Mac • Linux



## **Ваш ход! Установите вашу среду Python**

Теперь ваша очередь. Откройте свой ноутбук или настольный компьютер и установите редактор Му. Полные инструкции для разных платформ вы найдете в приложении Б (они доступны для Windows, macOS и Linux).

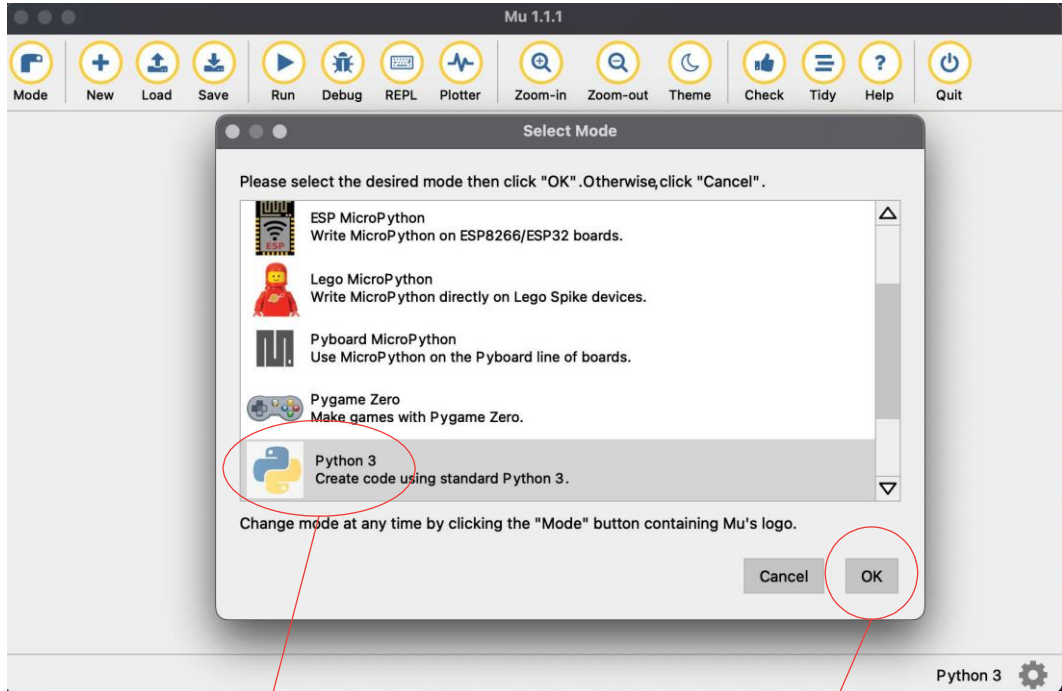
Если вы предпочитаете какой-то другой редактор, смело устанавливайте его вместо Му. Не бойтесь экспериментировать!



## Как общаться с компьютером

«Давай запустим Му и начнем писать программу для твоей кофейни», — сказал Саймон.

Эрик запустил Му и увидел его главное окно.



1. Выберите Python 3

2. Нажмите OK

«Выбери Python 3 в меню и нажми «ОК», — предложил Саймон.

Эрик сделал то, что сказал Саймон. «С этого момента», — продолжил Саймон, — «Му запомнит, что ты предпочитаешь использовать Python 3. Возможно, ты заметил, что есть еще какие-то режимы, которые можно использовать для работы с микроконтроллерами, создания веб-приложений и других дел. Но сейчас нам нужен только режим Python 3».

Теперь перед Эриком открылось окно редактора.



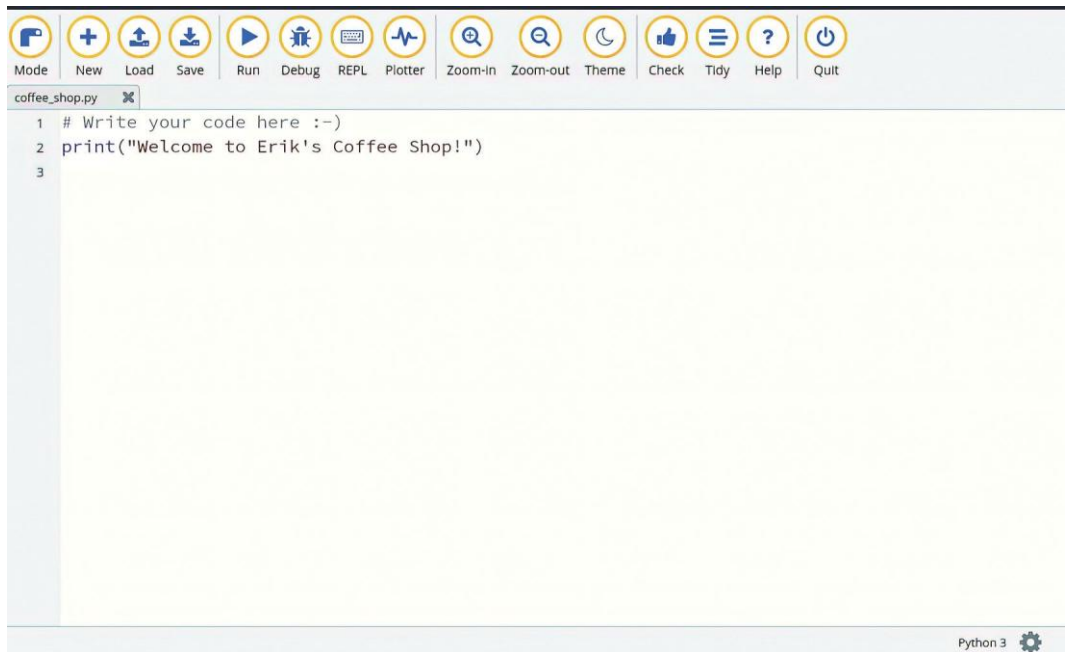
«Что мне здесь написать?» – спросил Эрик.

«Что ты хочешь, чтобы твоя программа делала в первую очередь?»

«Она должна вывести на экран: «Welcome to Erik’s Coffee Shop!»»

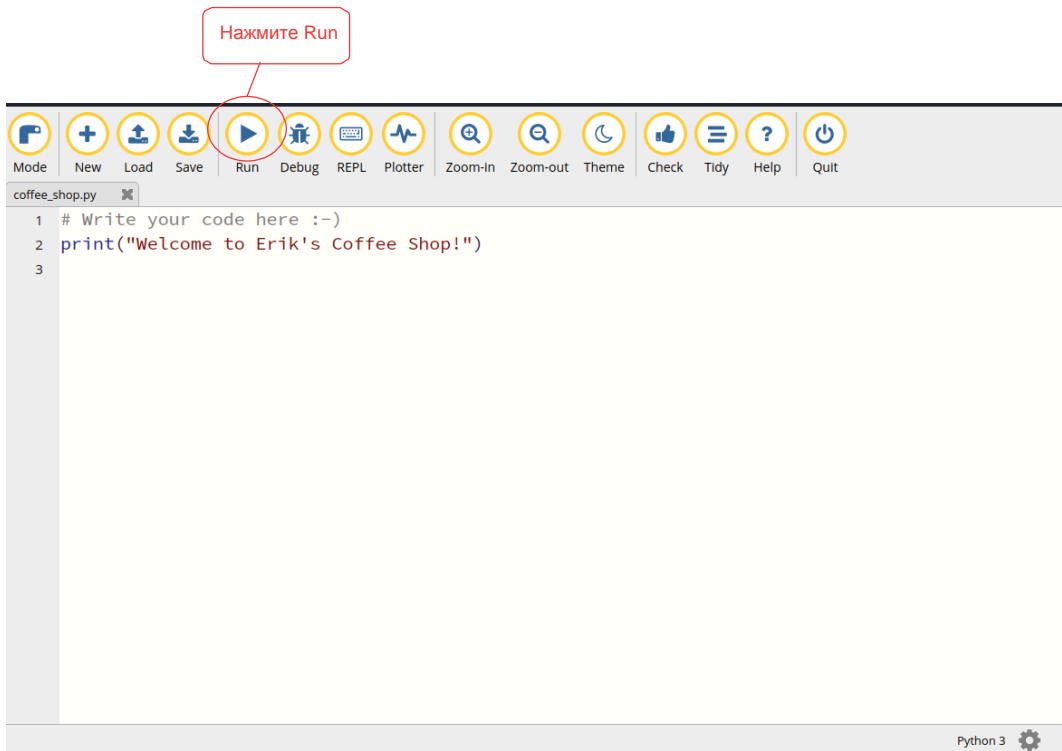
"Прекрасно! Давай напишем для этого код. Помнишь функцию `print()` в языке Python?"

Эрик начал печатать. Этот первый шаг был весьма легким.



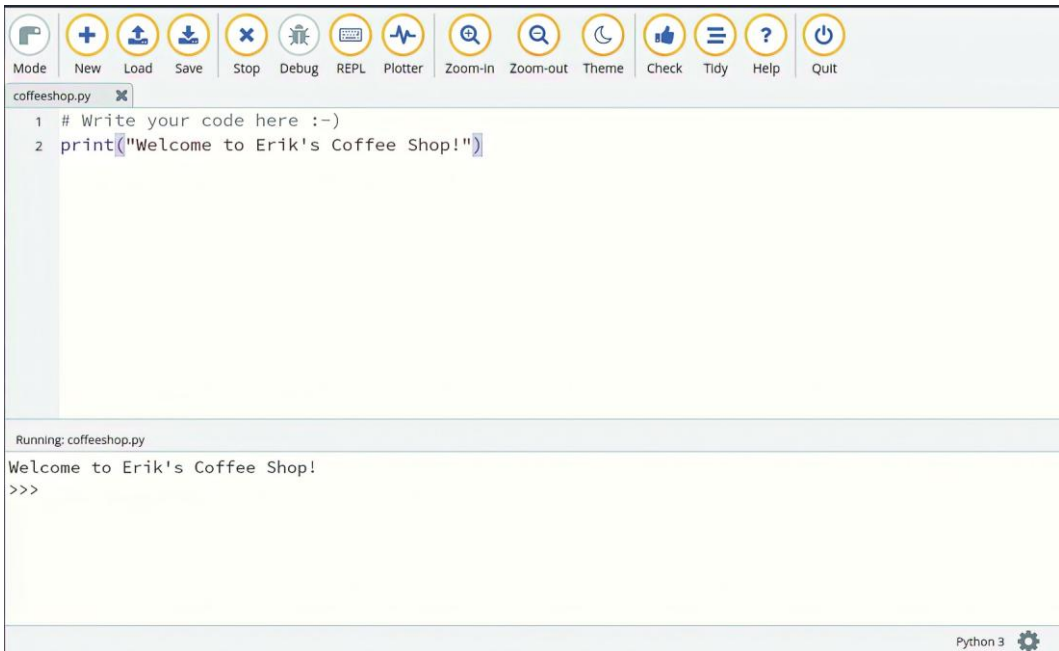
«Что дальше?»

«Теперь ты можешь его запустить. Нажми «Run».



Эрик нажал кнопку, и внезапно появилось другое окно с вопросом, хочет ли он сохранить программу. Это было легко. Эрик ввел имя файла, `coffeeshop`, и уже был готов нажать Enter, чтобы сохранить файл, когда Саймон сказал: «Подожди, подожди... Не забудь добавить расширение `.py` к имени файла. Ты должен сообщить своему текстовому редактору, что это программа на Python. Ми добавляет это расширение автоматически, а другие редакторы — нет, поэтому убедись, что все твои файлы на Python имеют расширение `.py` после имени файла».

Эрик добавил `.py` к имени файла и сохранил его. Сразу после этого он заметил еще одно окно внизу, под окном редактора. Приветствие выглядело именно так, как он хотел!



«Это работает!» – Эрик был в восторге

«Конечно, это работает. Почему бы и нет?» – Ответил Саймон. — «Но ты же хотел собирать заказы, не так ли?»

«Да, я бы спросил имя моего клиента и то, что он хочет».

«А потом?» – Саймон, очевидно, знал ответ, но он хотел, чтобы Эрик нашел его сам.

«А потом я вывел бы «Привет! Вот ваш заказ:» и показал бы его название, вкус и топинг, как в настоящем рецепте».

«Хорошая идея», — сказал Саймон. «Но посмотри: когда ты пишешь свою программу, ты не знаете, что хочет заказать твой друг, верно? Поэтому ты не можете написать в своей программе «Вы заказали карамель». Кроме того, разные клиенты заказывают *разное*. Для Алекса это будет карамель, а для Эмили — клубника. Как видишь, вкус меняется от заказа к заказу, как и имя клиента. Помнишь, как в программировании называется эта штука?»

«Это *переменная!*», — Эрик был рад, что вспомнил курс Python, который он прошел несколько месяцев назад.

"Верно!" — Саймон тоже был рад, — «Переменная подобна коробке: в нее можно положить что-то, а затем открыть и посмотреть, что в коробке. Кроме того, ты можешь заменить то, что находится в коробке, на что-нибудь другое».

«В нашем случае», — продолжил Саймон, — «давай начнем с поля под названием `answer` и будем хранить в нем все, что ты услышишь от своего клиента.

Ты спрашиваешь у клиента имя, и он отвечает, например, Alex. Ты помещаешь этот ответ в поле `answer` и хранишь его там. Если ты хочешь его распечатать, ты говоришь Python: «Пожалуйста, распечатайте то, что сейчас находится в поле под названием `answer`.» Имя следующего клиента может быть Эмили, поэтому вы помещаете в поле `Emily`. И в следующий раз Python выведет `Emily`, а не `Alex`, потому что именно это *сейчас* находится в поле под названием `answer`. Давайте напишем для этого код».

«Прямо здесь, в этом же файле?» – спросил Эрик.

«Конечно, продолжай в том же файле. Чтобы получить информацию от клиента, мы используем функцию `input()`. Когда мы ее вызываем, она ждет, когда пользователь что-нибудь введет. Итак, пользователь вводит что-то на клавиатуре и нажимает Enter. Затем функция *возвращает* все, что ввел пользователь».

«Подожди, подожди», — остановил Саймона Эрик. — «Что ты имеешь в виду под словом «*возвращает*»? А еще ты говоришь о функциях. Конечно, я знаю, что это такое, но можешь ли ты рассказать мне, что *ты* подразумеваешь под «функциями»?» — Эрик не хотел показать, что он *почти ничего* не помнит о функциях из пройденного курса.

«Функция — это фрагмент кода, который что-то *делает*. Почти каждый фрагмент кода что-то делает, но некоторые фрагменты кода мы используем чаще других. Позже мы создадим свои собственные функции, а сейчас мы будем использовать функции, написанные кем-то другим. Есть операции, которые люди используют часто, например, вывод на печать чего-либо. Ты этого не заметил, но ты уже использовал функцию, когда написал `print()` в своей предыдущей программе. В программировании мы говорим, что мы *вызываем* функцию».

«Ага, я понимаю», — сказал Эрик, — «То, что содержит круглые скобки, называется «функцией».

«Верно. И ты можешь поместить что-то в эти круглые скобки, и функция что-то с этим *сделает*. Например, она выведет на печать наше сообщение. То, что ты передаешь в функцию, называется аргументами. Иногда аргументом является строка, иногда число, а иногда аргументов несколько».

«Что такое «строка»?» – спросил Эрик.

«В данном случае строка — это слово или несколько слов. Мы будем ими часто пользоваться, так что хорошо, что ты спросил об этом», — сказал Саймон.

«Мы говорим, что «*передаем аргументы*» в функцию», — продолжил Саймон, — «Функция что-то делает с аргументами и что-то получает в *результате*. Например, она может что-то вычислить или сделать что-то с переданной вами строкой, например, преобразовать ее в ЗАГЛАВНЫЕ БУКВЫ или зашифровать ее. А затем она возвращает этот результат в нашу основную программу».

«Но как я увижу результат?» – спросил Эрик, — «Функция выведет его?»

«Нет, не выведет. Здесь нам нужны *переменные*. Мы говорим Python: «Пожалуйста, вызови эту функцию с этими аргументами и помести все, что она

возвращает, в это поле, извини, в эту переменную». Все это делается с использованием простого знака «равно», т.е.: `=`. Например, если ты хочешь вызвать функцию `input()` и поместить в переменную `answer`, то, что она возвращает, просто напиши следующее:

```
answer = input()
```

«А после того, как ты сохранишь ответ клиента, ты сможешь вывести его на печать. Для этого нужно вызывать функцию `print()` и передать в нее переменную в качестве аргумента».

«Отлично», — сказал Эрик. — «Теперь я понимаю, как это записать». Он начал печатать в редакторе. Через минуту он показал следующее:

```
print("Welcome to Erik's Coffee Shop!")

answer = input()
print(answer)
```

«Должен ли я запустить этот код?» — спросил он Саймона.

«Конечно, вперед, нажми Run .» — Эрик нажал Run .

«Там написано: 'Welcome to Erik's coffee shop', а потом ничего».

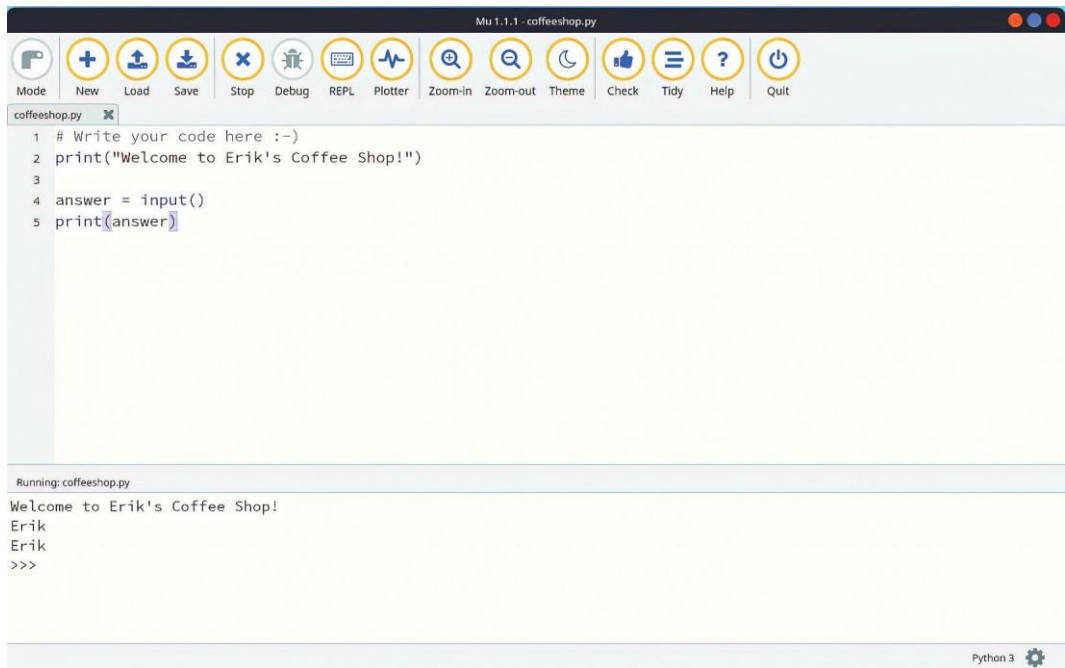
«Чего ты ожидал? » — спросил Саймон.

«Что он спросит мое имя».

«Но ты не сказал Python, что он должен что-либо спрашивать. Теперь он ждет твоего ответа. Введи что-нибудь».

Эрик ввел "Erik" и нажал Enter Enter.

Питон напечатал Erik.



«Это работает!» - сказал Эрик.



**Ваш ход! Напишите свой первый диалог**

Напишите программу диалога, которую только что написал Эрик. Это короткая программа, поэтому я рекомендую вам набрать ее самостоятельно, а не копировать из книги. Придумайте название для своей кофейни и используйте его в первом приветственном сообщении. Либо вы можете создать другой магазин, если хотите. Что он будет продавать? Мороженое? Пиццу? Цветы? Игрушки для домашних животных?

Попробуйте запустить вашу программу. Делает ли она то, что вы от нее ожидаете? Если нет, скопируйте код из книги или с веб-сайта книги (<https://github.com/pavelanni/pythonicadventure-code/tree/main/ch01>) и запустите программу снова. Она должна работать.





«Да, это работает», — сказал Саймон, — «но давай сделаем программу более удобной для пользователя. Помнишь, ты был сбит с толку, когда там не было ничего, кроме ‘Welcome’. Ты должен сообщить пользователю, чего ты от него ожидаешь. И еще, вместо того, чтобы выводить только Erik, ты можешь добавить что-то вроде Here is your order, Erik (Вот ваш заказ, Эрик).

«Ты можешь передать эту строку в функцию `input()` в качестве аргумента. Мы называем это строкой *подсказки*. Она объясняет пользователю, чего мы от него ожидаем. А в функции `print()` ты можешь добавить строку, которую ты хочешь вывести, перед переменной `answer`. Давай я тебе помогу.»

Саймон помог Эрику добавить эти строки в код, и вот как код стал выглядеть после этого:

```
print("Welcome to Erik's Coffee Shop!")

answer = input("Please enter your name: ")
print("Here is your order, ", answer)
```

Саймон заметил, что Эрик искал кнопку Run , и объяснил: «Прежде чем снова нажать Run , тебе необходимо остановить предыдущий сеанс Python. Видишь три угловые скобки >>> ? Они означают, что Python запущен и ожидает твоего ввода. Я объясню тебе, как использовать Python данным образом, позже, а сейчас просто нажми Stop , а затем снова нажми Run ».

Теперь программа запросила ввод и ответила точно так, как запрограммировал Эрик:

```
Welcome to Erik's Coffee Shop!
Please enter your name: Erik
Here is your order, Erik
>>>
```

**Ваш ход! Сделайте вашу программу более удобной для пользователя**

Добавьте приглашение и строку вывода в вашу первую программу. Создайте подсказку, спрашивающую у клиента имя, например: «Рад вас видеть! Как вас зовут?»

«Это выглядит гораздо более удобным для пользователя, не так ли?» — спросил Саймон. «Всегда думай о своих пользователях и спрашивай себя: «Понятно ли, чего я ожидаю от пользователя? Могут ли они здесь сделать ошибку?»»

«Теперь», — продолжил Саймон, — «нам нужно спросить твоего клиента о его заказе. Ты сказал, что у тебя есть кофе и шоколад. И ты говорил что-то о вкусах и топингах?»

«Да», — сказал Эрик, — «я хочу спросить его, какую начинку и вкус он желает».

«Ну, давай, спроси у него. Ты можешь просто повторить тот же код, но не забудь изменить строку подсказки. И я думаю, что тебе следует вывести весь заказ в самом конце, а не после каждого вопроса. Попробуй это сделать.»

Эрик написал этот код и остановился на последней строке:

```
print("Welcome to Erik's Coffee Shop!") answer

= input("Please enter your name: ")
answer = input("Please enter your drink: ")
answer = input("Please enter your flavor: ")
answer = input("Please enter your topping: ")
print("Here is your order: ", answer)
```

«Ты сказал мне поместить ответы в переменную `answer`. Но как мне узнать, какой вкус и топинг?» — Эрик был в замешательстве.

«Да, я сказал тебе поместить ответы в переменную, называемую `answer`», — ответил Саймон. «Здесь мы подходим к одной из самых сложных проблем в информатике: присвоению имен переменным», — улыбнулся он. — «Конечно, мы не храним все ответы в переменной с наименованием `answer`. Давай использовать разные переменные для разных ответов и зададим им осмысленные имена. В качестве имени клиента мы будем использовать переменную `name`—это легко. Если ты спрашиваешь об основном напитке, укажи ответ в переменной `drink` или `product`. Для ответов о вкусе и топпинге используй переменные `flavor` и `topping`.»

«В конце», — продолжил Саймон, — «выведи каждую переменную в отдельной строке, используя несколько функций `print()`. Вперед, не бойся! Я помогу тебе, если понадобится».

Эрик поработал над своим кодом и наконец тот стал выглядеть так:

**Листинг 1.1 Первая версия приложения для кофейни**

```
print("Welcome to Erik's Coffee Shop!")

name = input("Please enter your name: ")
drink = input("Please enter your drink: ")
flavor = input("Please enter your flavor: ")
topping = input("Please enter your topping: ")
print("Here is your order, ", name)
print("Main product: ", drink)
print("Flavor: ", flavor)
print("Topping: ", topping)
print("Thanks for your order!")
```

Эрик нажал Run, и его программа начала диалог. Эрик ответил на все вопросы и получил следующий результат:

```
Welcome to Erik's Coffee Shop!
Please enter your name: Erik
Please enter your drink: coffee
Please enter your flavor: caramel
Please enter your topping: chocolate
Here is your order, Erik
Main product: coffee
Flavor: caramel
Topping: chocolate
Thanks for your order!
>>>
```

Саймон заметил последнюю строчку и похвалил Эрика за инициативу: «Всегда принято благодарить своих клиентов».

«Да, я видел это на нескольких чеках в кофейнях», — сказал Эрик. Он был рад, что сделал что-то самостоятельно, помимо того, что сказал ему старший брат.

**Ваш ход! Добавьте больше вариантов в диалог. Используйте переменные.**

Отредактируйте свою предыдущую программу и добавьте новые строки в диалог. Опять же, не стесняйтесь изменять выводимые подсказки и строки, чтобы они больше подходили для вашего проекта.

Измените результат печати. Посмотрите чеки из мест, которые вы посещаете (кафе, ресторанов, продуктовых магазинов и т.д.). Постарайтесь, чтобы распечатка выглядела похоже. Используйте текстовые символы, такие как |, \_, =, + и другие, чтобы ваш вывод выглядел интересно.

Саймон решил, что на сегодня пора подводить итоги.

«Я думаю, что мы сегодня хорошо поработали», — сказал он. — «Давай посмотрим, что мы уже сделали. Сначала мы развернули нашу среду программирования».

«Да», — сказал Эрик. «Мне нравится редактор Ми. Он использует цвета, чтобы выделить для меня различные части программы. Моя строка также отображается красным цветом, пока я не поставлю кавычки в конце. И у него

есть темный режим! Я знаю, что настоящие программисты всегда используют темный режим! Думаю, я продолжу его использовать».

«Во-вторых», — сказал Саймон, — «мы впервые использовали *функцию*. Какая это была функция?»

«Это была функция `print()`», — сказал Эрик. — «Я сказал ей, что нужно вывести, и она это вывела».

"Верно. Мы *вызвали* функцию и *передали ей аргумент*».

«В-третьих», — продолжил Саймон, — «мы использовали другую функцию для получения информации от пользователя».

«Это была функция `input()`», — сказал Эрик. — «Она сохраняла ответы клиента в переменных».

"Великолепно!" — Саймон действительно гордился своим братом. — «Ты делаешь большие успехи».

### ***Ваш ход!* Объясни это сам**

Попробуйте объяснить это своими словами.

- Что такое функция? Приведите несколько примеров.
- Что такое аргументы функции? Приведите несколько примеров.
- Как вызвать функцию?
- Что такое переменная? Зачем они вам нужны?
- Как следует называть переменные? Почему?



## Что такое программа?

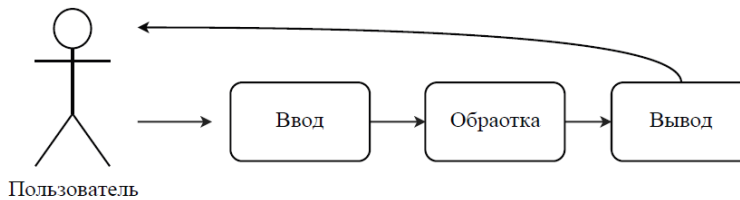
«Наконец, давай добавим еще немного теории», — сказал Саймон. — «Мы только что создали очень простую программу, но в ней присутствуют все основные компоненты любой другой программы. Мы попросили пользователя *ввести* некоторую информацию. После того как мы получили данные от пользователя, мы что-то сделали с этими данными. Обычно мы называем это *обработкой*. В нашем случае мы просто сохранили данные, но могли бы сделать с ними что-нибудь еще, верно?»

"Например?" — спросил Эрик.

«Например, ты ввел слово 'coffee' строчными буквами, но мы можем захотеть начинать все названия продуктов и вкусов с заглавных букв. Для этого в Python есть специальная функция. Поэтому мы можем *обработать* данные после их получения».

«Отличная идея, хочу ее использовать!» — сказал Эрик.

«Конечно, мы это сделаем. И, наконец, после того, как мы обработали данные, мы их *вывели*. Другими словами, мы получили некоторый результат. Смотри сюда.



«Ввод не всегда исходит от пользователя», — продолжил Саймон. — «Иногда взаимодействия с пользователем нет, и программа берет данные откуда-то еще. Например, из Интернета, или последних спортивных результатов. А иногда от датчиков, как в робототехнике. Либо из документов и изображений».

«Часто результат не просто распечатывается. Моя команда по робототехнике получает входные данные с датчиков, мы обрабатываем их в микроконтроллере, а на выходе мы получаем сигналы управления двигателями, например «повернись налево, двигайся вперед». Но структура при этом остается той же: ввод → обработка → вывод».

«Хватит теории», — сказал Саймон. — «Завтра мы поработаем над улучшением нашей программы».

«Улучшением?» — Эрик был удивлен. — «Но ведь она уже и так работает нормально, не так ли?»

«Что, если твой пользователь введет что-то, чего нет в твоей кофейне?» — спросил Саймон. — «Как, например, «кленовый сироп»? Что ты будешь тогда делать? Ты должен рассказать пользователю, что есть в твоей кофейне и что он может заказать. Итак, завтра мы поработаем над меню. Также посмотрим, что можно сделать в случае возникновения ошибок».

## Новое, что вы узнали сегодня

- *Переменная*—место (ящик), где мы можем хранить значения. Например, мы можем хранить числа, буквы или строки. Переменная может хранить одновременно только одну вещь.
- *Функция*—фрагмент кода, который делает то, что мы хотим иметь возможность выполнить снова и снова.
- *Аргументы функции*—информация, необходимая функции для выполнения ее работы.
- *Вызов функции*—напишите наименование функции с круглыми скобками и аргументами внутри этих скобок.

## Код для этой главы

Вы можете найти код этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch01>.

## Больше идей для ваших проектов

Дополнительные идеи для ваших приложений можно найти на сопутствующем сайте книги: [https:// pythonicadventure.com/projects/](https://pythonicadventure.com/projects/). Пока не беспокойтесь о Flask, SQLite и веб-приложениях (которые там упоминаются). Мы вернемся к ним позже.

# 2

## Списки: Что в меню?

---

### **В этой главе**

- Эрик начинает использовать меню в своей кофейне
- Эрик использует списки Python для своих напитков, вкусов и топпингов
- Эрик использует циклы `for` для *вывода* своих списков
- Саймон объясняет, как компьютеры хранят числа и строки
- Эрик узнает важную вещь об *индексах* списков

На следующий день Эрик был готов продолжить работу над приложением Coffee Shop. Он вспомнил, что Саймон сказал что-то об отсутствующих продуктах, о которых клиенты могли бы упомянуть в диалоге. Он подошел к брату и спросил: «Ты вчера сказал, что покупатели могут ввести то, чего нет у меня в кофейне. Как мне в таком случае быть?»

«Вспомни, когда ты в последний раз были в кафе или ресторане. Как ты узнал, что можно заказать?»

«У них было меню со списком блюд, которые готовятся в ресторане.»



«Верно!» — сказал Саймон. — «Меню! Это как раз то, что мы собираемся создать сегодня. Как выглядит меню кофейне?»

«Это списки. Перечень напитков, таких как кофе, шоколад и кофе без кофеина. И перечень вкусов, которые я могу добавить. Например, карамель, мята и другие. А также перечень топпингов.»

«Правильно, это списки!» — Саймон был очень рад, что Эрик использовал это слово. — «Вот так, да?»

Он быстро набросал что-то похожее на меню.

```
Menu

Drinks
Coffee      1.00
Chocolate   1.50
Decaf       1.20
  Flavors
Caramel     0.50
Vanilla     0.45
...
  Toppings
Chocolate   0.30
Cinnamon    0.30
....
```

«Списки — это то, что нам нужно! У нас есть списки в Python — возможно, вы это помните. Списки в Python очень полезны. Они могут содержать числа, строки и даже другие списки. Например...» — Саймон взял еще один лист бумаги и написал несколько примеров.

```
fruits = ['apple', 'peach', 'banana']

numbers = [42, 256, 1000]

constants = [3.1416, 2.718, 1.4142]
```

«Давай создадим списки для твоего меню. Мы просто зададим для каждого списка наименование, например, `flavors`, а затем запишем наши вкусы в квадратных скобках. Затем сделаем то же самое с топпингами и основными напитками. Не забывай, что вкусы, топпинги и напитки представляют собой строки, поэтому их следует заключать в кавычки. Ты можешь создать новый файл в своем редакторе и назвать его, например, `menu.py`.»

Эрик открыл редактор и начал писать код. Вот что у него получилось за несколько минут:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]
```


«Очень хорошо», — сказал Саймон. — «Теперь давайте выведем их как меню.»

«Просто `print(drinks)`?» — предположил Эрик.

«Ты можешь это сделать, но это будет некрасиво. Попробуй это.»

Эрик добавил оператор `print()` в конце:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]
print(drinks)
```

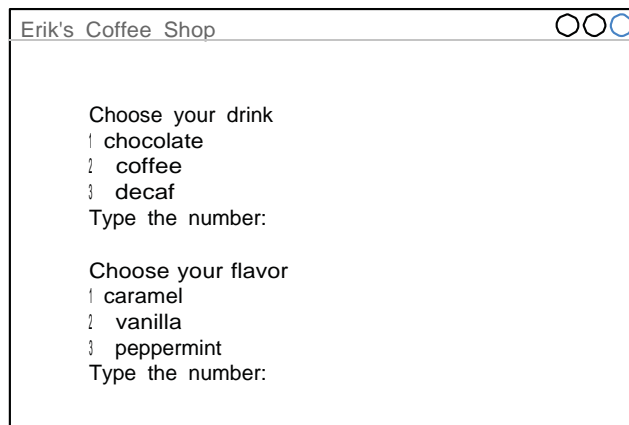
«Нажми Run », — сказал Саймон.

Эрик щелкнул мышью и увидел результат под окном редактора:

```
['chocolate', 'coffee', 'decaf']
>>>
```

«Если наше меню представляет собой список, нам следует и вывести его как список», — сказал Саймон. — «И ты должен предоставить пользователю возможность выбора из списка. Например, ты можешь попросить его ввести букву. Но здесь у нас есть `chocolate` (шоколад) и `coffee` (кофе), поэтому вы не можете просто использовать первую букву слова, ‘C’, для обоих напитков. Вместо этого давай будем использовать цифры. Для каждого пункта меню мы выведем число. Затем наш пользователь введет номер в соответствии со своим выбором. Например, 1 для шоколада, 2 для кофе. Что-то вроде этого.»

Саймон взял лист бумаги и нарисовал простое меню.



«Да, я видел такое в китайском ресторане — у каждого блюда есть номер», — вспомнил Эрик. — «Но как мне это сделать на Python?»

«У тебя есть список из нескольких элементов», — начал объяснение Саймон. «Тебе придется вывести каждый элемент, добавив перед ним число. Когда нам нужно что-то *повторить* на Python, мы используем *цикл*. В данном случае это будет цикл `for`. Мы сообщаем Python, что для каждого элемента списка он должен что-то сделать. Например, распечатать.»

«Напиши простой цикл `for`», — продолжил Саймон. — «Позволь мне написать первый цикл».

Саймон взял клавиатуру Эрика и добавил пару строк в его код:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

for d in drinks:
    print(d)
```

Саймон нажал `Run`, и они увидели на экране результат:

```
chocolate
coffee
decaf
>>>
```

«Теперь твой ход», — сказал Саймон. — «Напиши такой же код для двух других списков. Обрати внимание на то, что я использовал другую переменную для печати элементов списка. Список называется `drinks`. Я использовал только букву `d` для каждого напитка в списке. Обычно лучше всего использовать осмысленные наименования для переменных, например, `drinks` для списка напитков. Но если переменная будет использоваться только в одном цикле для просмотра списка, наименование может быть коротким, например, в одну или две буквы. Это не правило, но его легче ввести.»

«Еще одна важная вещь», — продолжил Саймон, — «это то, что в Python пробелы имеют *большое* значение. Ты видишь, что вызов функции `print()` сдвинут на четыре пробела вправо? Так мы сообщаем Python, *что* следует повторить в цикле. Та часть, которая смещена вправо, называется *блоком*. Все, что вы поместите в этот блок, будет повторяться для каждого элемента списка. Сейчас у нас в блоке только один вызов функции, но позже мы добавим что-нибудь еще.

«Ты еще заметили, что я не вводил с клавиатуры четыре пробела. Наш редактор сделал это за меня автоматически. Все редакторы кода, которые мы используете для программирования на Python, имеют эту функцию. Когда они видят двоеточие (:), они автоматически сдвигают следующую строку. Это называется «начало блока». Теперь продолжай писать циклы.»

Эрик создал еще два цикла для вывода двух других списков. Ему понравилась идея использования более коротких имен переменных (меньше нажимать на кнопки!). Он также заметил, что редактор автоматически сдвинул

строку после двоеточия. Очень полезная функция! Вот что он получил в результате:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

for d in drinks:
    print(d)

for f in flavors:
    print(f)

for t in toppings:
    print(t)
```

Он сохранил программу, а затем запустил ее:

```
chocolate
coffee
decaf
caramel
vanilla
peppermint
raspberry
plain
chocolate
cinnamon
caramel
>>>
```

### ***Ваш ход!* Создайте свой список продуктов и выведите его**

Откройте редактор кода и создайте программу, похожую на ту, которую только что создал Эрик. Она должна содержать три или более списков предметов. Затем используйте циклы для распечатки содержимого этих списков.

Вы можете использовать пункты меню Эрика или создать свои собственные. Мороженое со вкусом, бублики, минифигурки — все, что пожелаете!

«Очень хорошо», — сказал Саймон. «Но у нас пока нет цифр. Мы должны это исправить. Помнишь, я говорил тебе, что мы добавим в блок еще кое что? Вот что я предлагаю. Мы создадим переменную, которая будет хранить номер элемента в списке. Каждый раз, когда мы переходим к следующему элементу, мы добавляем единицу к этой переменной. В этом случае chocolate (шоколад) будет номером один, coffee (кофе) номером два и так далее.

«Позволь мне показать», — и Саймон снова взял клавиатуру Эрика:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

i = 1
for d in drinks:
    print(i, d)
    i = i + 1

for f in flavors:
    print(f)

for t in toppings:
    print(t)
```

Он запустил программу, и они увидели следующий результат:

```
1 chocolate
2 coffee
3 decaf
caramel
vanilla
peppermint
raspberry
plain
chocolate
cinnamon
caramel
>>>
```

«Видите: я добавил переменную `i`. Для каждого элемента списка теперь я вывожу не только его значение, но и его номер. А затем я прибавляю единицу к числу, чтобы перейти от 1 к 2, затем от 2 к 3 и так далее. Теперь иди и измени остальное», — сказал Саймон.

Эрик внес изменения:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

i = 1
for d in drinks:
    print(i, d)
    i = i + 1

for f in flavors:
    print(i, f)
    i = i + 1

for t in toppings:
    print(i, t)
    i = i + 1
```

Когда он запустил программу, он увидел следующий результат:

```
1 chocolate
2 coffee
3 decaf
4 caramel
5 vanilla
6 peppermint
7 raspberry
8 plain
9 chocolate
10 cinnamon
11 caramel
>>>
```

«Но это не то, чего я хотел!» — сказал Эрик. — «Я думаю, что это должно быть 1, 2, 3 для напитков, затем 1, 2, 3 для вкусов и снова 1, 2, 3 для топпингов».

«Верно!» — Саймон согласился. — «Как бы ты это сделал?» «Использовать другую переменную?»

«Да, это возможно. Но вы можете использовать ту же переменную `i`. Важно установить значение в 1 перед каждым циклом. Мы называем это *инициализацией* переменной».

Эрик добавил `i = 1` перед каждым циклом и получил вот это:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

i = 1
for d in drinks:
    print(i, d)
    i = i + 1

i = 1
for f in flavors:
    print(i, f)
    i = i + 1

i = 1
for t in toppings:
    print(i, t)
    i = i + 1
```

### ***Ваш ход!* Выведите три (или более) меню с цифрами**

Измените предыдущую программу, чтобы добавлять номера к пунктам меню. Используйте циклы. Не забудьте сбросить переменную `i` (обычно она называется счетчиком элементов) для каждого нового списка.

Он нажал Run и получил следующий результат:

```
1 chocolate
2 coffee
3 decaf
1 caramel
2 vanilla
3 peppermint
4 raspberry
5 plain
1 chocolate
2 cinnamon
3 caramel
>>>
```

«Теперь давай сделаем его немного красивее», — сказал Саймон. «Добавь заголовки типа ‘Our drinks’ («Наши напитки») перед каждым списком. Помни, мы должны сообщить пользователю, что он видит и что ему следует делать».

Эрик добавил несколько заголовков. Он даже добавил дополнительную строку под каждым заголовком. Он был уверен, что это будет похоже на настоящее меню:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

print("Erik's Coffee Shop drinks")
```

```

print("-----")
i = 1
for d in drinks:
    print(i, d)
    i = i + 1

print("Erik's Coffee Shop flavors")
print("-----")
i = 1
for f in flavors:
    print(i, f)
    i = i + 1

print("Erik's Coffee Shop toppings")
print("-----")
i = 1
for t in toppings:
    print(i, t)
    i = i + 1

```

### **Ваш ход! Добавьте заголовки в свои меню**

Добавьте заголовки в свои меню, чтобы получить красивый результат. Используйте название вашего магазина в заголовках. Попробуйте использовать другие символы, а не только тире.

Результат был прекрасен, как он и ожидал:

```

Erik's Coffee Shop drinks
-----
1 chocolate
2 coffee
3 decaf
Erik's Coffee Shop flavors
-----
1 caramel
2 vanilla
3 peppermint
4 raspberry
5 plain
Erik's Coffee Shop toppings
-----
1 chocolate
2 cinnamon
3 caramel
>>>

```

«Выглядит хорошо», — сказал Саймон. — «Что еще хорошо в этом формате, так это то, что теперь у тебя есть три списка в меню и три списка в твоей программе».

«Теперь», — продолжил Саймон, — «для каждого списка в меню ты должен попросить пользователя выбрать элемент, а затем получить от него эту информацию. Как получить информацию от пользователя? Ты делал это



вчера, помнишь?»

«С помощью `input()`?» — спросил Эрик.

«Конечно!» — Саймон был рад, что Эрик вспомнил предыдущий урок. — «Ты ведь можешь написать это сам, не так ли?»

«Позволь мне попробовать», — сказал Эрик и начал редактировать свой код. Он вспомнил, что ему следует использовать функцию `input()`. Затем он поместил подсказку в круглые скобки, а слева использовал переменную. Он вспомнил, что не следует использовать одну и ту же переменную для разных вопросов.

Вот что он написал:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

print("Erik's Coffee Shop drinks")
print("-----")
i = 1
for d in drinks:
    print(i, d)
    i = i + 1
drink = input("Choose your drink: ")

print("Erik's Coffee Shop flavors")
print("-----")
i = 1
for f in flavors:
    print(i, f)
    i = i + 1
flavor = input("Choose your flavor: ")

print("Erik's Coffee Shop toppings")
print("-----")
i = 1
for t in toppings:
    print(i, t)
    i = i + 1
topping = input("Choose your topping: ")
```

### **Ваш ход! Добавьте пользовательский ввод в свои меню**

Добавьте функции `input()` в свои меню. Используйте соответствующие наименования переменных для хранения ответов пользователя.

«Что теперь?» — спросил он Саймона.

«Теперь твой пользователь вводит номер, и ты используешь его, чтобы найти элемент. В Python мы называем это число *индексом* списка. Если ты поместишь это число в квадратные скобки рядом с названием списка, ты получишь этот элемент. Вот так», и он написал пример:

```
drinks[drink]
```

«‘Here is your order’ («Вот ваш заказ»), как и вчера, но теперь ты получишь эти позиции из меню. Попробуй это. Я помогу тебе, если понадобится».

Это было немного сложнее. Эрик просмотрел свою вчерашнюю программу и скопировал строки из нее в конец этой программы. Затем он заменил переменные, такие как `drink`, элементами списка, как это предложил Саймон.

Вот его код:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

print("Erik's Coffee Shop drinks")
print("-----")
i = 1
for d in drinks:
    print(i, d)
    i = i + 1
drink = input("Choose your drink: ")

print("Erik's Coffee Shop flavors")
print("-----")
i = 1
for f in flavors:
    print(i, f)
    i = i + 1
flavor = input("Choose your flavor: ")

print("Erik's Coffee Shop toppings")
print("-----")
i = 1
for t in toppings:
    print(i, t)
    i = i + 1
topping = input("Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drinks[drink])
print("Flavor: ", flavors[flavor])
print("Topping: ", toppings[topping])
print("Thanks for your order!")
```

«Теперь запусти программу и посмотри, что она нам даст», — сказал Саймон.

Эрик нажал `Run`, программа распечатала меню напитков и попросила его сделать выбор. Пока программа сработала. Эрик быстро ввел числа для всех трех меню и увидел вот этот результат:

```
Here is your order:
Traceback (most recent call last):
  File "/home/erik/mu_code/menu.py", line 30, in <module>
    print("Main product: ", drinks[drink])
TypeError: list indices must be integers or slices, not str
```

>>>

«Что это?» — он был озадачен.

«Поздравляю!» — сказал Саймон.

«Чему ты так рад? Что моя программа не работает?» — Эрик начал злиться на своего брата.

«Вовсе нет!» — сказал Саймон. — «Ты получил первое сообщение об ошибке от Python, и это хороший знак! Делать ошибки и исправлять их — единственный способ учиться. У тебя появилось сообщение об ошибке — теперь давай попробуем решить проблему. Обычно Python объясняет причину, почему это произошло. Начни с прочтения последнего сообщения. Что там написано?»

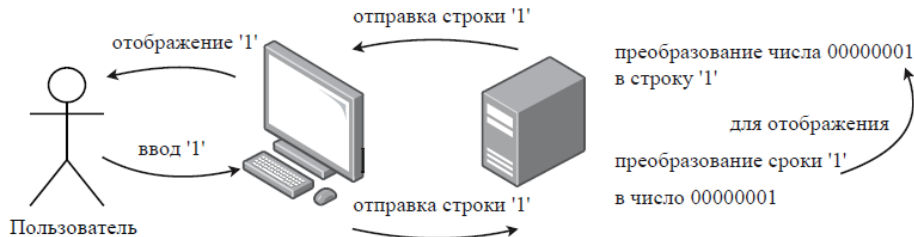
«Что-то насчет ломтиков... Должны быть целыми числами, а не str. Что это такое?»

«Я согласен, чтобы научиться читать сообщения об ошибках Python, нужна определенная практика. Здесь говорится, что когда вы используете переменную в качестве индекса списка, эта переменная должна быть целым числом, например 1, 2, 3».

«Но я же ввел цифры!» — Эрик все еще был в замешательстве.

«Да, ты набирал цифры на клавиатуре. Но для Python все, что ты вводишь с клавиатуры, является строкой. Python различает строку, содержащую число «1», и целое число 1.»

«Это связано с тем, как компьютеры хранят информацию в памяти», — продолжил Саймон. — «Компьютер хранит *число* 1 в памяти, но когда он показывает его тебе, он преобразует его в *строку* «1». Компьютеры также преобразуют числа, полученные с клавиатуры. Ты набираешь «123» на клавиатуре, а компьютер получает эту *строку* и *преобразует* ее в *число* 123. Посмотрите сюда». — Саймон нарисовал изображение компьютера, клавиатуры, дисплея и пользователя.



«Нам нужно указать Python, чтобы он преобразовывал строки, которые ты вводишь на клавиатуре, в целые числа. Для этого существует специальная функция `int()`. Позволь мне показать тебе, как ей пользоваться.»

Саймон изменил программу Эрика в одном месте и позволил ему сделать то же самое в двух других местах. Вот как после этого стала выглядеть программа Эрика:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]
```

```
print("Erik's Coffee Shop drinks")
print("-----")
i = 1
for d in drinks:
    print(i, d)
    i = i + 1
drink = input("Choose your drink: ")

print("Erik's Coffee Shop flavors")
print("-----")
i = 1
for f in flavors:
    print(i, f)
    i = i + 1
flavor = input("Choose your flavor: ")

print("Erik's Coffee Shop toppings")
print("-----")
i = 1
for t in toppings:
    print(i, t)
    i = i + 1
topping = input("Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drinks[int(drink)])
print("Flavor: ", flavors[int(flavor)])
print("Topping: ", toppings[int(topping)])
print("Thanks for your order!")
```

Преобразует  
строку в число

Эрик запустил программу, ввел свой выбор (coffee, caramel, chocolate) и получил такой результат:

```
Here is your order:
Main product: decaf
Flavor: vanilla
Topping: cinnamon
Thanks for your order!
>>>
```

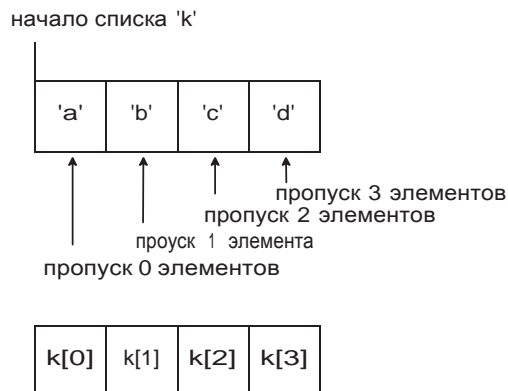
«Что? Это все неправильно!» — воскликнул Эрик. — «Это не то, что я выбрал!»

«Я не рассказал тебе одну важную вещь о списках Python», — сказал Саймон. — «Их индексы начинаются с нуля, а не с единицы. Поэтому, если ты хочешь получить первый элемент списка, тебе следует поставить в квадратные скобки 0. Если же тебе нужен второй элемент, используй в качестве индекса 1».

«Но почему?» — Эрик был шокирован такой странной вещью.

«Это долгая история», — ответил Саймон. — «Это связано с тем, как компьютеры хранят списки в памяти. Индекс, который ты используешь, — это количество элементов, которые тебе следует пропустить от начала списка, чтобы получить нужный элемент. Если тебе нужен первый элемент списка, то тебе не нужно пропускать какие-либо элементы. Ты просто берешь его в самом начале списка. Значит, количество элементов, которые тебе следует пропустить, равно *нулю*, верно? Вот почему индекс первого элемента равен 0. Посмотри сюда».

Саймон нарисовал еще одну картинку.



«Так что мне теперь делать?» — спросил Эрик. Он думал, что понял объяснение Саймона, но его все равно раздражало это неудобство.

«Я вижу, что ты раздражен», — сказал Саймон. — «Не волнуйся, ты очень быстро к этому привыкнешь. И ты, как все настоящие программисты, начнешь

считать все с нуля», — улыбнулся Саймон. — «Пока ты просто вычитай единицу из каждого индекса в квадратных скобках. Но будь осторожен: тебе нужно добавить - 1 *после* того, как ты преобразуешь ввод в целое число, но не раньше. Вот так: `drinks[int(drink) - 1]`.»

Эрик исправил свой код, и теперь он выглядел так:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

print("Erik's Coffee Shop drinks")
print("-----")
i = 1
for d in drinks:
    print(i, d)
    i = i + 1
drink = input("Choose your drink: ")

print("Erik's Coffee Shop flavors")
print("-----")
i = 1
for f in flavors:
    print(i, f)
    i = i + 1
flavor = input("Choose your flavor: ")

print("Erik's Coffee Shop toppings")
print("-----")
i = 1
for t in toppings:
    print(i, t)
    i = i + 1
topping = input("Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drinks[int(drink) - 1])
print("Flavor: ", flavors[int(flavor) - 1])
print("Topping: ", toppings[int(topping) - 1])
print("Thanks for your order!")
```

### ***Ваш ход!* Измените свою программу так, чтобы вывести заказ**

Измените свою программу так, как это только что сделал Эрик, и выведите свой заказ. Не забудьте преобразовать входные строки в числа. И не забудьте вычесть единицу (1) из каждого числа — индексы списка начинаются с нуля, помните?

Он запустил программу, ввел 2, 1, 1 и наконец получил то, что хотел:

```
Here is your order:
Main product: chocolate
Flavor: caramel
Topping: chocolate
Thanks for your order!
>>>
```

«Круто! Она работает!» — Эрик был определенно счастлив. — «Мне нравится программа моей кофейни! Мы закончили на этом?»

«Почти», — ответил Саймон. — «Послушай, ты три раза написал почти один и тот же фрагмент кода».

«Что в этом плохого?»

«Представь, что ты хочешь что-то изменить в своем коде. Например, ты можешь изменить способ вывода пунктов меню. Тебе придется изменить его во всех трех местах. Или в еще большем количестве мест, если ты решишь добавить другие списки меню. Представь, что ты хочешь добавить десерты в меню кофейни. Это означает, что тебе придется скопировать этот код еще раз. А что делать, если ты допустил ошибки в коде? Программисты называют их *багами* (позже расскажу почему). Тогда тебе придется исправлять эти ошибки в четырех местах, повторяя все действия. Программистам нравится принцип DRY: Don't Repeat Yourself (не повторяйся)».

«Но я не понимаю, как я могу это сделать», — растерялся Эрик. — «Если у меня есть три списка меню, мне придется вывести их *три* раза. И мне придется трижды просить пользователя вводить данные».

«Здесь мы можем использовать *функцию*», — объяснил Саймон. — «Помнишь, когда мы начали использовать функцию `print()`, я говорил тебе, что для операций, которые мы хотим повторять снова и снова, мы используем функции. До сих пор мы использовали функции, написанные кем-то другим. Теперь мы создадим нашу собственную функцию и будем ее использовать».

"Это круто. Мне это нравится», — сказал Эрик.

«Отлично, мы сделаем это завтра. Я думаю, что на сегодня мы сделали достаточно. Ты отлично сегодня поработал, Эрик», — Саймон действительно был рад, что его брат добился успехов. — «Давай подведем итог тому, что мы узнали сегодня. Что было первым?»

«Сначала мы создали списки», — сказал Эрик. — «Мы внесли в списки все наши напитки, вкусы и топпинги».

«Хорошо, а что было дальше?»

«Затем мы вывели списки с помощью циклов. А затем мы вывели цифры рядом с каждым напитком или вкусом».

«Да, именно так», — подтвердил Саймон. — «Давай, а что было после этого?»

«Затем я попытался вывести напитки из списка, но получил от Python сообщение об ошибке. Ты объяснил мне, как в памяти компьютера хранятся числа. Потом мы преобразовали цифры, и я снова попробовал вывести свой заказ. И только потому, что ты не сказал мне, что индексы начинаются с нуля (Эрик этого не забыл!), в моем заказе были выведены не те позиции!»

«Пожалуйста, прости меня», — улыбнулся Саймон. — «Но теперь ты запомнишь это гораздо лучше, я в этом уверен!»

«Наконец-то я все исправил, и теперь все работает хорошо!» — закончил Эрик.

"Отличная работа!" — Саймон показал Эрику большой палец. — «Завтра мы продолжим и напишем нашу первую функцию».

## Новое, что вы узнали сегодня

- *Список*—коллекция элементов в Python. Элементами списка могут быть строки или числа или даже их сочетание.
- *Индекс списка*—номер, который мы используем для извлечения элемента из списка. Индексы в списках всегда начинаются с нуля и увеличиваются на единицу для каждого следующего элемента: 0, 1, 2, 3 и так далее.
- *Числа и строки* —это разные *типы* переменных в Python. Когда вы печатаете что-то на экране или вводите данные с клавиатуры, вы всегда используете строки. Если вы хотите выполнить какие-либо математические операции с числами, полученными от пользователя, вам необходимо преобразовать их из строк в числа.

## Код для этой главы

Вы можете найти код этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch02>.



# Функции: Не повторяйтесь!



## **В этой главе**

- Саймон объясняет, как не повторяться (и почему)
- Эрик пишет свою первую функцию на Python
- Эрик начинает использовать интерактивный инструмент Python
- Эрик совершенствует свою функцию, чтобы рецепт выглядел профессионально

«Где мы вчера остановились?» — спросил Саймон у Эрика на следующий день

«Ты собирался рассказать мне, почему ошибки называют багами», — сказал Эрик.

"Верно!" — сказал Саймон. — «Хочешь верь, хочешь нет, но первые компьютеры были собраны на электромеханических устройствах, называемых реле, с контактами, похожими на выключатели света. Однажды инженеры обнаружили ошибку при выполнении своей программы. Код был правильным, но они обнаружили жучка (или *баг* по-английски), застрявшего между контактами одного из реле. Вот почему программисты называют ошибки багами. Ошибки обычно скрыты в вашем коде, и чтобы ваша программа заработала, вы должны устранить их. Что еще?"

«Ты сказал, что мне не следует повторяться. И ты сказал, что сегодня мы собираемся написать собственную функцию».

"Верно! Сначала расскажи мне, что ты уже знаешь о функциях».

«Мы уже использовали пару функций», — ответил Эрик. — «Мы использовали `print()` и `input()`. Ты сказал, что кто-то написал их, чтобы мы могли их использовать. Мы также можем с функциями использовать аргументы функций. Для этого нам просто нужно поместить аргументы в круглые скобки, и функция что-нибудь с ними сделает, например напечатает».

«Это верно! Ты отличный ученик!» — Саймон улыбнулся. — «В функциях важно то, что они могут каждый раз делать одно и то же, но с разными аргументами. Поэтому, если ты видишь, что одно и то же выполняется несколько раз, тебе следует подумать, можно ли превратить это в функцию. Чтобы принять решение, тебе следует посмотреть на повторяющийся код и спросить себя, какие его части одинаковые, а какие различаются. Где твоя вчерашняя программа?»

"Здесь." Эрик открыл в редакторе файл `menu.py`:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

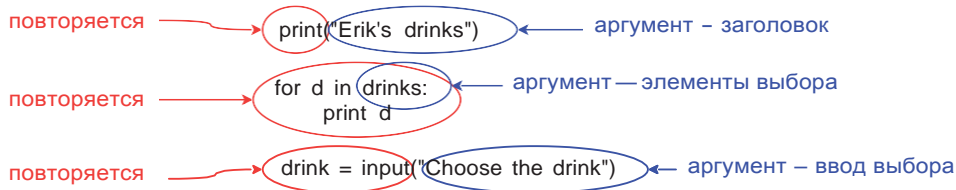
print("Erik's Coffee Shop drinks")
print("-----")
i = 1
for d in drinks:
    print(i, d)
    i = i + 1
drink = input("Choose your drink: ")

print("Erik's Coffee Shop flavors")
print("-----")
i = 1
for f in flavors:
    print(i, f)
    i = i + 1
flavor = input("Choose your flavor: ")

print("Erik's Coffee Shop toppings")
print("-----")
i = 1
for t in toppings:
    print(i, t)
    i = i + 1
topping = input("Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drinks[int(drink) - 1])
print("Flavor: ", flavors[int(flavor) - 1])
print("Topping: ", toppings[int(topping) - 1])
print("Thanks for your order!")
```

«Смотри, что здесь повторяется?» — спросил Саймон и начал рисовать диаграмму с фрагментом кода Эрика.



«Цикл `print()` в начале и `input()` в конце.»

«И что же в этих трех случаях отличается?»

«Разный ввод выбора для `input()`», — ответил Эрик. — «Кроме того, немного отличается заголовок».

«И ты запускаешь циклы `for` для различных списков, таких как `drinks`, `flavors` и `toppings`, верно?» — Саймон решил помочь Эрику. — «Поэтому мы будем передавать в качестве аргументов то, что *отличается*. В нашем случае это будет список элементов выбора, заголовок меню и ввод выбора.»

«Давай создадим новый файл в редакторе и назовем его `menu_function.py`. Мы напишем в нем нашу функцию».

Эрик нажал в редакторе New, а затем Save. Затем он ввел `menu_function.py` и был готов писать код.

«Функции в Python начинаются со слова `def`, за которым следует наименование функции», — продолжил Саймон. — «Давай присвоим нашей функции наименование `menu`. Затем ты откроешь скобки иставишь свои аргументы».

Эрик написал `def menu()` и не знал, что делать дальше.

Саймон помог: «Мы только что говорили о твоих аргументах. Я вижу, ты думаешь, как их назвать. Помни, что присвоение имен переменным и аргументам — одна из самых сложных проблем в информатике. Но ты не один. Давай назовем их `choices`, `title` и `prompt`. Просто введи их после открывающей скобки и поставьте двоеточие после закрывающей скобки».

Эрик напечатал следующее:

```
def menu(choices, title, prompt):
```

Он заметил, что после того, как он нажал Enter, курсор переместился на следующую строку, но на четыре пробела вправо. «Мне здесь писать?» — спросил он Саймона.

"Да, конечно!" — Ответил Саймон. — «Видишь ли, редактор помогает тебе написать код твоей функции! Теперь посмотри на свой вчерашний код и начни копировать то, что ты хочешь поместить в функцию. Сначала мы напечатали заголовок, давайте сделаем так и здесь, но вместо самой строки мы просто напечатаем *аргумент* с названием `title`. Ты даже можешь поставить после него строку из тире, как ты это делал раньше».

Эрик написал следующее:

```
def menu(choices, title, prompt):
    print(title)
    print("-----")
```

«Теперь напиши цикл», — продолжил Саймон. — «Но вместо `drinks` или `flavors` твой список теперь называется `choices`. И ты можешь использовать в цикле переменную `c`, так как это первая буква слова ‘choices’.”

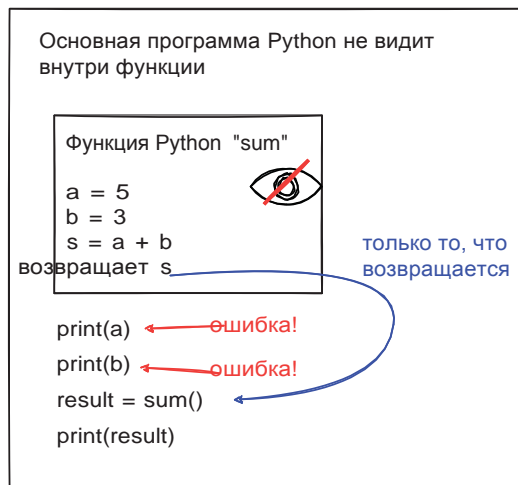
Эрик скопировал цикл из своей предыдущей программы. Он добавил функцию `input()` с `prompt`, не спрашивая брата:

```
def menu(choices, title, prompt):
    print(title)
    print("-----")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    choice = input(prompt)
```

"Великолепно!" — сказал Саймон, когда увидел код Эрика. — «Теперь нам предстоит *вернуть* сделанный клиентом выбор».

«Разве мы не можем просто напечатать переменную `choice` в нашей основной программе?» — спросил Эрик.

"Нет! И это очень важная вещь в функциях», — Саймон был рад, что Эрик задал этот вопрос. — «Переменные, которые находятся внутри твоей функции, *видны* только внутри этой функции. Смотри сюда, я нарисую картинку».



«Это означает, что вы не можете видеть, что находится внутри этих переменных, когда вы не находитесь *внутри функции*. Поэтому, если мы хотим, чтобы наша основная программа видела эти значения, мы должны *вернуть* эти переменные. Обычно внутри функции имеется много переменных, но в результате мы хотим вернуть только одну или две. В этом случае мы можем вернуть число, введенное пользователем, которое сохраняется в переменной `choice`».

Саймон на мгновение остановился и задумался. «Но мы можем добиться большего», — сказал он. — «Посмотри еще раз на свой вчерашний код. Что еще повторяется?»

Эрик посмотрел и сказал: «Это функции `int()`, и нам пришлось трижды использовать `- 1`. Это раздражало». Он до сих пор не привык к тому, что индексы списков начинаются с нуля.

«Хорошо, давай добавим их тоже в функцию», — предложил Саймон. — «Мы преобразуем ответ пользователя в целое число, получим элемент из списка и вернем сам элемент, а не его индекс. Это сделает нашу функцию еще более полезной. Основная программа, которая ее вызывает, получит выбор, сделанный пользователем, а не просто какой-то номер. Позволь мне показать тебе это», и Саймон добавил в функцию операции преобразования и оператор `return` :

```
def menu(choices, title, prompt):
    print(title)
    print("-----")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    choice = input(prompt)
    answer = choices[int(choice) - 1]

    return answer
```

«Теперь давай проверим ее», — сказал Саймон. — «В нашей основной программе, которая находится чуть ниже функции, мы вызываем функцию и выводим полученный ответ. Но сначала нам нужны списки с напитками и всем остальным. Скопируй их из начала вчерашней программы».

Эрик добавил три списка чуть ниже функции. На этот раз строки не были смещены вправо и начались с первой позиции:

```
drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]
```

«Хорошо», — сказал Саймон. — «Теперь ты готов вызвать свою функцию. Передай заголовок, список напитков и запрос на ввод. Затем присвой результат функции переменной. Назови ее, например, `choice`. А потом выведи ее содержимое».

Эрик последовал инструкциям Саймона, и вот что он получил:

```
def menu(choices, title, prompt):
    print(title)
    print("-----")
```

```

i = 1
for c in choices:
    print(i, c)
    i = i + 1
choice = input(prompt)

answer = choices[int(choice) - 1] return

answer

drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]
choice = menu(drinks, "Erik's drinks", "Choose your drink: ")
print(choice)

```

### ***Ваш ход!*** Создайте свою собственную функцию

Создайте свою собственную функцию, аналогичную той, которую только что создал Эрик. Используйте списки меню, заголовки и подсказки. Попробуйте запустить свою программу (раньше Эрика!).

«Должен ли я теперь запустить мою программу?» – спросил Эрик. «Да, теперь она готова. Вперед, запускай!»

Эрик нажал Run, и программа спросила его о напитке, точно так же, как и раньше. Эрик ответил и получил ожидаемый результат:

```

Erik's drinks
-----
1 chocolate
2 coffee
3 decaf
Choose your drink: 2 coffee
>>>

```

«Оно работает!» — сказал он. — «Я добавлю сюда и другие меню», — и он начал писать. Менее чем за 10 минут он был готов протестировать всю программу. Теперь его программа выглядит так:

```

def menu(choices, title, prompt): print(title)
    print("-----")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    choice = input(prompt)
    answer = choices[int(choice) - 1]

    return answer

drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

```

```
drink = menu(drinks, "Erik's drinks", "Choose your drink: ")
flavor = menu(flavors, "Erik's flavors", "Choose your flavor: ")
topping = menu(toppings, "Erik's toppings", "Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drink)
print("Flavor: ", flavor)
print("Topping: ", topping)
print("Thanks for your order!")
```

### **Ваш ход! Добавьте другие меню**

Добавьте другие меню в вашу программу. Они будут использовать одну и ту же функцию, но с разными аргументами — списками вариантов выбора, заголовками и подсказками. Попробуйте запустить и протестировать свою программу, используя пункты меню.

Это сработало, как и ожидалось! Эрик запустил программу; снова ввел 2, 2, 1; и получил свой заказ:

```
Here is your order:
Main product:  coffee
Flavor:  vanilla
Topping:  chocolate
Thanks for your order!
>>>
```

Саймон сказал: «Обрати внимание на то, что твоя программа стала короче. Теперь, если тебе нужно что-то изменить, то нужно изменить это только в одном месте».

«Почему я захотел бы что-нибудь изменить в программе? Она уже работает хорошо», — сказал Эрик.

«О, всегда есть способы улучшить код!» Ответил Саймон. «Давайте немного улучшим название. Ты заметил, что линия тире теперь длиннее заголовка?»

«В самом деле? О да, ты прав», — сказал Эрик. — «Это потому, что я изменил название на простое 'Erik's drinks'. Я могу сделать его короче, это легко.»

«Конечно, можно сделать короче, но посмотрите — в меню `drinks` заголовок короче, а в меню `toppings` — длиннее. Линия тире должна быть такой же длины, как и строка заголовка».

«Но как я могу это сделать? Может быть, лучше убрать эти черточки?» — Эрик был в замешательстве.

«Нет, мне нравится эта линия; она сделает твою квитанцию более реальной. Я хочу, чтобы ты ее оставил. Но нам нужно вычислить длину заголовка и сделать линию одинаковой с ним длины. Здесь ты изучишь одну новую функцию и одну новую операцию».

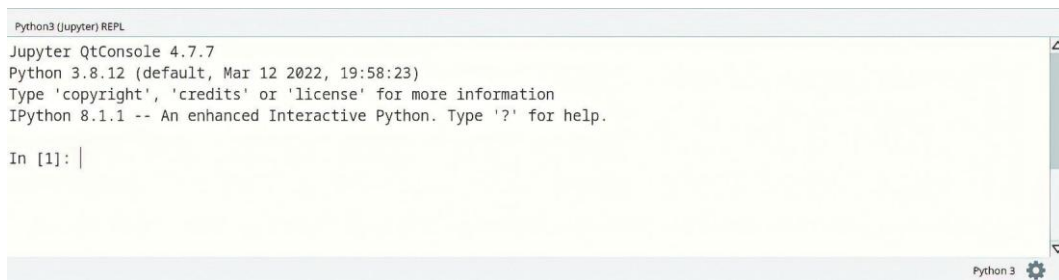
«Функция, которую мы собираемся использовать, называется `len()`. Ты просто помещаешь строковый аргумент в круглые скобки, и он возвращает длину строки. Позволь мне показать тебе. Мы воспользуемся еще одной функцией редактора Му, называемой *REPL*. Это означает *Read-Eval-Print-Loop*, и позволяет

использовать Python в интерактивном режиме. Обычно я использую ее, когда хочу быстро что-то протестировать. Или показать что-нибудь кому-нибудь, как сейчас», — и Саймон улыбнулся.

«Нажми REPL », — продолжил Саймон.

Эрик щелкнул на пиктограмме, и в нижней части окна редактора Mu открылось еще одно окно.

«Видишь, у нас здесь появился интерактивный Python», — сказал Саймон. — «Ты можешь ввести здесь любой код на Python, и он будет выполнен. Ты даже можешь использовать его как калькулятор. Попробуй ввести `print('hello world')` или `2 + 2` сразу после `In [1]:`. В REPL, `In` означает ввод, а число после него — это номер вашей команды. REPL выведет `Out[1]:` и ответ. Ты это увидишь.»



Эрик напечатал, Саймон действительно оказался прав:

```
In [1]: print('hello world') hello
world
```

```
In [2]: 2 + 2
Out[2]: 4
```

«Теперь давай посчитаем длину строки. Введи `len('abcd')` ».

Эрик напечатал и получил следующее:

```
In [3]: len('abcd')
Out[3]: 4
```

«Теперь ты видишь, что длина строки `abcd` равна 4», — сказал Саймон. — «Ты можешь сделать то же самое и со строковыми переменными. Используй переменную `s`, помести в нее строку `'hello'` и вычисли ее длину. Я уверен, что теперь ты знаешь, как это сделать».

Эрик напечатал в окне REPL и получил такой результат:

```
In [4]: s = 'hello'

In [5]: len(s)
Out[5]: 5
```

«Хорошо», — сказал Саймон. — «Теперь ты знаешь, что если у тебя есть строка, ты всегда можешь узнать ее длину. Более того, аналогичным образом ты можешь



получить длину списка. Создай список из чисел 1, 2, 3 и узнай его длину. Назови его `n`, например».

Эрик напечатал:

```
In [6]: n = [1, 2, 3]
In [7]: len(n)
Out[7]: 3
```

«Мы воспользуемся этим позже, а сейчас позволь мне показать тебе один трюк», — продолжил Саймон. — «Что мне выдаст Python, если я попрошу его взять число 2 и умножить его на 2?»

«4?» Эрик не был уверен, что это не уловка. Вопрос был слишком простым. "Верно. Что мне выдаст Python, если я возьму букву «А» и умножу ее на 2?" "Я не знаю? 2А, может быть?"

«Давай, попробуйте это с интерактивным Python!» — предложил Саймон. Эрик напечатал и получил результат:

```
In [8]: 2 * 'A'
Out[8]: 'AA'
```

«Интересно!» — Эрик был удивлен.

«А что, если вместо «А» взять тире и умножить его на 10?»

Эрик начал догадываться, куда его направляет Саймон, и напечатал:

```
In [9]: 10 * '-'
Out[9]: '-----'
```

«Теперь замени число 10 на длину строки `'hello'`».

Теперь Эрик уловил идею Саймона:

```
In [10]: len('hello') * '-'
Out[10]: '-----'
```

### **Ваш ход!** Используйте REPL и поэкспериментируйте с функцией `len()`

Запустите REPL, щелкнув соответствующий значок в редакторе. Затем повторите все эксперименты Эрика.

Попробуйте умножить число на строку из двух или трех букв. Сможете ли вы угадать, каким будет результат?

«Теперь я вижу это!» — сказал он. — «Мы берем аргумент `title`, вычисляем его длину и выводим строку из тире точно такого же размера!»

«Можешь ли ты изменить свою функцию сейчас?» — спросил Саймон.

«Да, конечно, я знаю, что делать!» — Эрик уже начал печатать. Он изменил только третью строку, и теперь его функция выглядела так:

```
def menu(choices, title, prompt):  
    print(title)  
    print(len(title) * '-')  
    i = 1  
    for c in choices:  
        print(i, c)  
        i = i + 1  
  
    choice = input(prompt)  
    answer = choices[int(choice) - 1]  
  
    return answer
```

Повторяет тире (-)  
несколько раз

### **Ваш ход! Измените свою функцию, чтобы вывести корректную строку тире**

Внесите изменения в свою программу, чтобы она выводила правильное количество тире, аналогично тому, что только что сделал Эрик. Попробуйте использовать другой символ — знак равенства (=), подчеркивание (\_) или что-то еще.

Эрик протестировал основную программу, и теперь все строки тире были точно такой же длины, как и их заголовки.

Саймон прокомментировал: «Теперь ты видишь, что не только результат, возвращаемый вашей функцией, зависит от аргументов, но и то, что она выводит на печать, также может зависеть от аргументов».

«Всегда полезно анализировать аргументы, которые вы получаете в своей функции», — продолжил Саймон. — «В данном случае мы проверили длину заголовка. Что бы сделала наша функция, если бы она получила пустую строку нулевой длины?»

«Я не знаю», — ответил Эрик. — «Я думаю, что она напечатает пустую строку — ничего».

«Верно», — сказал Саймон. — «Но, возможно, мы все же сможем вывести что-нибудь разумное, даже если заголовок будет пустым. Может быть, просто 'Erik's Menu' и черточки. Для таких случаев в Python у нас есть значения, *принимаемые по умолчанию* для аргументов функции. В твоей функции я бы изменил первую строку на вот такую». Саймон отредактировал файл Эрика:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
```

«В этом случае мы говорим Python: «Если *есть* аргумент `title`, примите его значение». Если ты не передал `title` при вызове функции, функция будет использовать значение по умолчанию, то есть `Erik's Menu`. Обычно рекомендуется устанавливать значения, принимаемые по умолчанию. Ты всегда можешь изменить их на что-то другое при вызове функции».

«Давай проверим», — предложил Саймон. — «При первом твоём вызове `drinks` удали и заголовок, и запрос. Оставь только напитки в качестве единственного аргумента».

Эрик сделал то, что предложил Саймон, и первый вызов функции выглядел так:

```
drink = menu(drinks)
```

Он запустил программу еще раз и увидел первое меню:

```
Erik's Menu
-----
1 chocolate
2 coffee
3 decaf
Choose your item:
```

Вот полная программа Эрика.

### Листинг 3.1 Полная программа меню с использованием функции

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)

    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    choice = input(prompt)
    answer = choices[int(choice) - 1]

    return answer

drinks = ["chocolate", "coffee", "decaf"]
flavors = ["caramel", "vanilla", "peppermint", "raspberry", "plain"]
toppings = ["chocolate", "cinnamon", "caramel"]

drink = menu(drinks)
flavor = menu(flavors, "Erik's flavors", "Choose your flavor: ")
topping = menu(toppings, "Erik's toppings", "Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drink)
print("Flavor: ", flavor)
print("Topping: ", topping)
print("Thanks for your order!")
```

#### **Ваш ход! Используйте в своей функции аргументы, принимаемые по умолчанию**

Добавьте значения, принимаемые по умолчанию, для ваших аргументов `title` и `prompt`. Попробуйте вызвать функцию без этих аргументов и убедитесь, что она использует значения, принимаемые по умолчанию.

«Конечно, теперь пользователю не сообщается, что это меню *напитков*, но это все равно лучше, чем пустая строка. Это полезно, если ты хочешь быстро что-либо протестировать. Ты всегда можешь добавить более описательные заголовки и подсказки позже».

«Думаю, на сегодня достаточно», — сказал Саймон. — «Давай подведем итог тому, что ты узнал. Что было сегодня первым?»

«Мы просмотрели программу, которую я написал вчера, и обнаружили вещи, которые повторялись трижды. И ты сказал мне, что мы можем написать свою собственную функцию. Ты рассказал мне о слове `def` и аргументах».

«Отлично! Давай дальше», — подбадривал Эрика Саймон. — «А что насчет этих аргументов?»

«В своей функции в качестве аргументов я использовал перечень возможного выбора, заголовков и подсказку».

Эрик любит рассказывать о *своей* функции — он впервые написал ее сам!

«Потом ты показал мне эту штуку *REPL* в редакторе», — продолжил Эрик. — «Мне она понравилась! А затем мы определили длину строки».

«И для чего мы ее использовали?» — спросил Саймон.

«Мы использовали ее для печати наших чеков, и теперь они выглядят красиво. А затем мы попытались использовать для аргументов значения, принимаемые по умолчанию. Это было немного скучно, но это сработало».

Саймон сказал: «Ты только что сказал очень важную вещь. Хорошие программные решения часто выглядят скучно, но они работают. Программирование – это не всегда хитрости и выдумки. Большую часть времени тебе придется делать очень скучные вещи, например проверять ввод пользователя, проверять наличие ошибок и так далее. Но если выполнение этих скучных вещей заставит твою программу работать, оно того стоит. Завтра мы посмотрим, что тебе нужно сделать, чтобы твоя программа работала, даже если твой пользователь вводит неправильные значения. А пока — отдохни! Ты сегодня отлично поработал!»

### Новое, что вы узнали сегодня

- *Функция* — фрагмент программного кода, который можно использовать (вызывать) повторно. Функция может быть написана вами или кем-то другим. Если он написан кем-то другим, обычно он является частью *библиотеки* или модулем Python.
- *Аргументы* — переменные, которые мы передаем функции при ее вызове. Функция принимает аргументы и использует их для подготовки выходных данных. Выходные данные можно распечатать или *вернуть* в основную программу.
- *REPL* — Read–Eval–Print–Loop, способ интерактивного запуска Python. Это очень полезно для быстрого тестирования функций.

### Код для этой главы

Вы можете найти код этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch03>.



# Ошибки пользователя: все делают ошибки



## **В этой главе**

- Эрик обнаруживает, что пользователи не всегда делают то, что вы им говорите
- Эрик учится использовать циклы, чтобы повторить свой вопрос пользователю
- Саймон помогает Эрику сделать меню более надежным

«Вчера ты проделал отличную работу, Эрик», — сказал Саймон на следующий день. — «Ты написал очень хорошую функцию, добавил аргументы, принимаемые по умолчанию, и протестировал ее».

«Да», — ответил Эрик. — «Я считаю, что это хорошая программа. Я хочу показать ее своим друзьям!»

«Подожди, подожди», — сказал Саймон. — «Я не думаю, что она еще готова к использованию».

«Почему? Я думаю, она работает прекрасно!»

"Да неужели? Дай мне попробовать», — Саймон выглядел так, будто он что-то задумал. Он снова запустил программу Эрика и в первом меню ввел `coffee`:

Erik's drinks

-----

```
1 chocolate
2 coffee
3 decaf
```

Сообщение об ошибке от Python

Choose your drink: **coffee**

Traceback (most recent call last):

File "/home/erik/mu\_code/menu\_function.py", line 18, in <module>

drink = menu("Erik's drinks", drinks, "Choose your drink: ")

File "/home/erik/mu\_code/menu\_function.py", line 9, in menu

answer = choices[int(choice) - 1]

ValueError: invalid literal for int() with base 10: 'coffee'

>>>

«Что ты делаешь?!» — Эрик был в ярости. — «Вводить следует только цифры, а не слова!»

«Но ты дал мне список и спросил, чего я хочу. Мне захотелось кофе, поэтому я и ввел `coffee`. В чем дело?» — Саймон пытался выглядеть невинным, но не смог скрыть улыбку.

«Ну, для таких *тупых* пользователей, как ты, я напишу ЗАГЛАВНЫМИ БУКВАМИ, что нужно вводить *цифры*!» — проворчал Эрик.

«Ладно, ладно, позволь мне попробовать еще раз», — сказал Саймон. Он запустил программу еще раз и в первом меню ввел 42:

Choose your drink: **42**

Сообщение об ошибке от Python

Traceback (most recent call last):

File "/home/erik/mu\_code/menu\_function.py", line 18, in <module>

drink = menu("Erik's drinks", drinks, "Choose your drink: ")

File "/home/erik/mu\_code/menu\_function.py", line 9, in menu

answer = choices[int(choice) - 1]

IndexError: list index out of range

>>>

"Снова?! Ты снова сломал ее?!» — Эрик был готов дать Саймону по рукам. — «Разве ты не видел, что есть только три варианта? Почему ты ввел 42?»

«Во-первых, потому что 42 — мое любимое число. Во-вторых, да, это была моя ошибка. Пользователи совершают ошибки, ты знаешь. Seriously, я хотел показать тебе, что твоя программа должна быть к этому готова. Ты можешь написать заглавными буквами все, что хочешь, но найдутся пользователи, которые этого не прочитают. Будут встречаться пользователи, которые будут совершать ошибки».

«И что мне с этим делать?» — Эрик все еще злился на своего брата, но был склонен с ним согласиться. Он и сам допускал ошибки с программами.

«Ты должен проверить, что вводит пользователь, и сообщить ему, если ввод неправильный. Давай подумаем, что здесь можно сделать».

«Что пользователи должны вводить в первом меню?» — спросил Саймон.

«Они должны вводить 1, 2 или 3», — ответил Эрик.

«Хорошо, мы можем проверить, вводили ли они 1, 2 или 3, а затем передать этот ввод в функцию и выбрать соответствующий элемент из списка вариантов. Но если это не так, мы должны сказать пользователю об ошибке».



«Да, я помню. Мы можем использовать в Python if-else», — предложил Эрик.

«Давай попробуем», — сказал Саймон. — «Как ты это сделаешь? Попробуй объяснить мне это так, как если бы я не знал об if-else».

«Я добавлю к своей функции: Если выбор пользователя — 1, или 2, или 3, тогда использовать его. В противном (else), вывести напоминание, что пользователь должен ввести одно из этих чисел».

«А как проверить, какой выбор сделал пользователь — 1, 2 или 3? Как это сделать на Python?» — спросил Саймон.

«Я не помню...» — сказал Эрик.

«Тебе нужно использовать оператор *сравнения*. Здесь используются два знака равенства, например: ==. Например, чтобы проверить, равен ли choice 1, вы должны написать choice == '1'».

«Но я хочу проверить все три номера», — сказал Эрик. — «Должен ли я написать if choice == '1' or '2' or '3'?»

«Ты почти прав!» — Ответил Саймон. — «Да, вы можете использовать or в Python, но в этом случае вам следует записать это в виде трёх *отдельных* сравнений и поместить между ними or . Вот так», — и он написал этот код на листке бумаги:

```
if choice == '1' or choice == '2' or choice == '3':
```

«Хорошо, понял», — сказал Эрик.

«Хорошо, давай закодируем это», — сказал Саймон.

Эрик открыл свой редактор и добавил четыре строки, до возврата return к его функции:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    choice = input(prompt)

    if choice == '1' or choice == '2' or choice == '3':
        answer = choices[int(choice) - 1]
    else:
        print("Enter number 1, 2, or 3!")

    return answer
```

«Теперь давай попробуем», — сказал Саймон.

Эрик запустил программу и в первом меню ввел 2.

«Почему ты ввел 2?» — спросил Саймон.

«Потому что я захотел кофе», — ответил Эрик.

«Но нам надо проверить вашу программу на предмет *неправильных* ответов!» — сказал Саймон. — «Да, я знаю, это твоя программа и ты не хочешь ее ломать, но как разработчик ты *должен* попытаться сломать свою программу. Ты должен представить себе все возможные неправильные способы использования твоей программы пользователями. Я знаю, это тяжело и очень неудобно, но тебе придется преодолеть это и попытаться ввести все возможные неправильные значения».

«Хорошо, хорошо», — сказал Эрик и перезапустил свою программу. В первом меню он ввел 42 как это сделал Саймон в прошлый раз (ввод Эрика выделен жирным шрифтом):

```
Erik's Menu
-----
1 chocolate
2 coffee
3 decaf
Choose your item: 42
Enter number 1, 2, or 3!
Traceback (most recent call last):
  File "/home/erik/mu_code/menu_function.py", line 21, in <module>
    drink = menu(drinks)
  File "/home/erik/mu_code/menu_function.py", line 14, in menu
    return answer
UnboundLocalError: local variable 'answer' referenced before assignment
>>>
```

Python показывает, где ошибка.

Python сообщает, что за ошибка.

**ПРИМЕЧАНИЕ** Вы можете узнать больше о сообщениях об ошибках Python на сопутствующем сайте книги:

<https://pythonicadventure.com/troubleshooting/index.html>.

«Давайте посмотрим, что здесь происходит», — сказал Саймон. — «Во-первых, когда ты ввел 42, твоя программа вывела сообщение, что пользователю следует ввести только 1, 2 или 3. Это хорошо. Но потом что-то пошло не так. Посмотри, там написано, что на переменную `answer` была сделана ссылка до присвоения:

```
UnboundLocalError: local variable 'answer' referenced before assignment
```

«Проще говоря, это означает, что ты не создал `answer`, но попытался ее использовать. И Python показывает тебе, где именно: ты пытался вернуть переменную `return answer`, но Python ничего не знал о переменной `answer`.»

"Но почему?" — сказал Эрик. — «У меня в коде есть строка `answer =`».

«Да, она у тебя есть, но важно то, *где* эта линия используется. В твоём коде ты создаешь переменную `answer` *только* тогда, когда пользователь вводит правильный выбор. Если пользователь вводит что-то другое, `answer` даже не создается».

«Другими словами», — продолжил Саймон, — «даже когда пользователь отвечает неверным номером или даже словом, тебе все равно придется вернуть *какой-то* ответ. Это очень важное правило: никогда не используйте переменную до того, как вы ее создадите и не присвоите ей *какое-либо* значение. Какое значение мы можем присвоить `answer` в нашем случае, если пользователь допустит ошибку? Здесь должна работать пустая строка типа `' '`. Добавь ее в свою функцию и проверь, помогло ли это».

Эрик изменил функцию следующим образом:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    choice = input(prompt)

    if choice == '1' or choice == '2' or choice == '3':
        answer = choices[int(choice) - 1]
    else:
        print("Enter number 1, 2, or 3!")
        answer = ' '

    return answer
```

Пустая строка, присвоенная `answer`.

### **Ваш ход!** Добавьте код для проверки ответа пользователя

В функцию меню добавьте код, который только что добавил Эрик. Убедитесь, что вы присвоили пустую строку переменной `answer` на случай неверного ввода. Проверьте, действительно ли он проверяет ваши ответы.

Он еще раз протестировал программу, и на этот раз она не выдала ошибки.

Программа вывела сообщение `Enter number 1, 2, or 3!` и перешла к следующему меню.

«Как ты думаешь, твоя программа справилась с этим?» — спросил Саймон.

«Она вывела сообщение о том, что следует ввести 1, 2 или 3», — ответил Эрик. — «Я думаю, это правильно».

«Но ты не получил данные о выборе напитка пользователем. Если он ввел неправильный номер, ты должен дать ему возможность ввести правильный номер. Это не школьный тест, где у тебя есть только один шанс ответить. Ты должен продолжать спрашивать пользователя, пока не получишь один из правильных ответов».

«Как мне это сделать?» — спросил Эрик.

«Для этого существует еще один вид *цикла*», — объяснил Саймон своему брату. — «Он называется *циклом while*. Он повторяется снова и снова и при каждой итерации проверяется *условие*. Если условие истинно, выполнение цикла продолжается. Если оно становится ложным, цикл завершается».

«Иногда мы проверяем условие в начале цикла, если оно нам уже известно. В этом случае мы говорим: «Пока условие верно, выполняй это». Но иногда, как в нашем случае, у нас нет ответа, когда мы запускаем цикл, потому что мы еще не спросили нашего пользователя об его выборе. Вот почему мы запускаем *бесконечный цикл* и проверяем условие *внутри* цикла — после того, как получим ответ от пользователя. Мы выходим из цикла, если условие становится истинным. Мы называем это *выходом* из цикла. Давай я покажу на схеме», — сказал Саймон и начал рисовать.

Саймон объяснил: «В данном примере мы уже знаем условие до того, как запустим цикл. Проверяем его состояние и решаем, стоит ли начинать. Состояние должно быть истинным или True, иначе цикл даже не запустится. Для этой проверки я использовал *ромбовидную* фигуру; именно так программисты обычно рисуют точки принятия решений».

«Если условие истинно, то мы что-то сделаем. Важно то, что это «сделай что-нибудь» должно, среди прочего, изменить состояние условия. В противном случае цикл будет продолжаться вечно, а мы этого не хотим».



## ПРИМЕР

`n = 5`

```
while n > 0:
    n = n - 1
```

«После того, как мы сделали это «что-то», мы переходим к началу цикла и *снова проверяем условие*. Если оно по-прежнему истинно, мы повторяем фразу «делаем что-то». Если нет, мы выходим из цикла и продолжаем работу нашей программы».

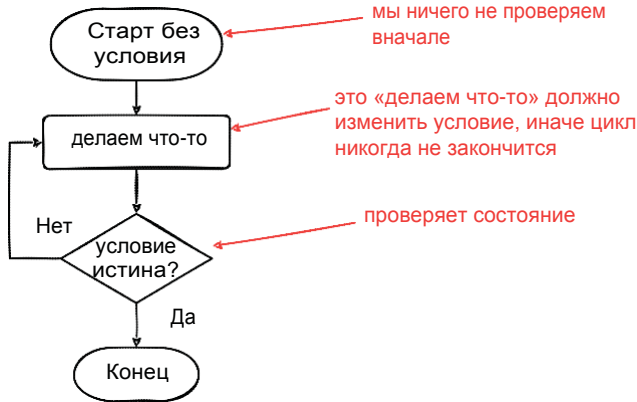
«Посмотри на этот простой пример: мы хотим вести обратный отсчет от 5 до 0. Сначала мы устанавливаем переменную `n` равной 5, а в начале цикла проверяем, больше ли она 0. Да, она больше, поэтому мы начинаем цикл. Помните, что в какой-то момент нам придется изменить условие, иначе цикл никогда не закончится. В нашем примере мы вычитаем 1 из `n` каждый раз, когда проходим цикл. В конце концов переменная `n` станет равной 0, и цикл остановится».

«Важно то, что в данном случае, прежде чем мы начнем цикл, мы уже знаем значение переменной `n`, и мы знаем, что оно больше 0».

«Что, если мы не знаем значение переменной, либо если переменная вообще не существует? Как и в нашем случае, мы можем проверить ответ пользователя *только после* того, как попросим его выбрать один из пунктов меню. Поэтому мы должны это сделать», — и Саймон нарисовал еще одну диаграмму.

Саймон продолжил свое объяснение: «Здесь мы начинаем цикл без проверки каких-либо условий. Сначала мы что-то делаем, и *только после этого* проверяем условие. Иногда этот цикл называется «do-until», что означает «делать что-то, пока условие не станет истинным». Когда условие истинно, мы выходим из цикла и продолжаем выполнение остальной части программы».

«В приведенном здесь примере я использовал твою ситуацию с `input()`. Ты запрашиваешь ввод данных, а затем проверяешь эти вводимые данные. Это твое условие: если введенные данные верны, следует выйти из цикла. В Python для этого мы используем оператор `break`».



## ПРИМЕР

```

while True:
    answer = input()
    if answer == '1':
        break
  
```

Эрик был немного ошеломлен этим длинным объяснением, но чувствовал, что знает, что делать. Он спросил своего брата: «Значит, ты хочешь сказать, что мне следует просто вставить `while` перед строкой `input()` и добавить `break` после того, как я получу правильный ответ?»

«Да, — ответил Саймон, — совершенно верно! Только не забудь об отступе всех строк, входящих в цикл, на четыре пробела вправо. Ваш редактор поможет вам, не волнуйтесь».

Эрик начал работать над своей программой. Через несколько минут его функция выглядела так:

```

def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    while True:
        choice = input(prompt)
        if choice == '1' or choice == '2' or choice == '3':
            answer = choices[int(choice) - 1]
            break
        else:
            print("Enter number 1, 2, or 3!")
            answer = ''

    return answer
  
```

"Верно?" — спросил Эрик своего брата.

«Я говорил тебе, что редактор может тебе помочь. Нажмите Check 🍌».

Эрик сделал, как посоветовал брат, и увидел следующее:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i += 1
    while True:
        choice = input(prompt)
        ↑ Syntax error. Python cannot understand this line. Check for missing characters!
        ↑ Expected an indented block
        if choice == '1' or choice == '2' or choice == '3':
            answer = choices[int(choice) - 1]
            break
        else:
            print("Enter number 1, 2, or 3!")
            answer = ''

    return answer
```

«Теперь ты видишь?» — сказал Саймон. — «Ты забыл сделать отступ строк вправо. Вот почему он говорит, что ожидает здесь нахождение блока с отступом. Смести вправо все строки, входящие в цикл».

Эрик изменил свою функцию и еще раз проверил код:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    while True:
        choice = input(prompt)
        if choice == '1' or choice == '2' or choice == '3':
            answer = choices[int(choice) - 1]
            break
        else:
            print("Enter number 1, 2, or 3!")
            answer = ''

    return answer
```

Цикл while начинается здесь.

Если ответ правильный, выходим из цикла.

Конец блока с отступом; конец цикла

### **Ваш ход! Добавьте цикл while в вашу функцию**

Добавьте цикл, как только что это сделал Эрик. Попробуйте воспользоваться кнопкой Чекс, чтобы узнать, есть ли ошибки в вашем коде. Сделайте несколько ошибок и посмотрите, сможет ли их найти редактор.

Тут же на кнопке Check на мгновение появился зеленый палец вверх 👍, и Эрик понял, что все в порядке. Он нажал Run, чтобы протестировать программу. В первом меню он ввел 42, как это сделал Саймон в прошлый раз. Программа сообщила, что ему следует выбрать 1, 2 или 3, и снова вернулась к подсказке. Она сработала без ошибок!

«Что еще Саймон делал не так с моей программой?» — Эрик попытался вспомнить. «Правильно, он пытался ввести слово «кофе»! Давай я тоже попробую».

Он ввел `coffee`, и его программа снова отреагировала так, как он и ожидал! Наконец Эрик ввел 2 и получил следующее меню. Вот что он увидел в окне редактора:

```
Erik's Menu
-----
1 chocolate
2 coffee
3 decaf
Choose your item: 42
  Enter number 1, 2, or 3!
Choose your item: coffee
  Enter number 1, 2, or 3!
Choose your item: 2
  Erik's flavors
-----
```

В следующем меню он ввел 4, соответствующее малине, и получил то же сообщение:

```
Erik's flavors
-----
1 caramel
2 vanilla
3 peppermint
4 raspberry
5 plain
Choose your flavor: 4
  Enter number 1, 2, or 3!
Choose your flavor:
```

«Почему это так?» — спросил он Саймона.

«Твоя программа ведет себя точно так, как ты ее написал», — ответил Саймон. Конечно, он знал, в чем проблема.

«Как ты написал свое условие?» — спросил он.

Эрик сказал: «Если ответ 1, 2 или 3 — ага, теперь я понимаю! Я ввел 4, поэтому программа считает, что это неправильный ответ! Но как мне это исправить?»

«Похоже, нам нужен список действительных ответов для каждого списка меню», — сказал Саймон. «Можно передать это как еще один аргумент. Но я думаю, ты знаешь уже достаточно, чтобы найти более лучшее решение».



"Что это?" — спросил Эрик. «Опять что-нибудь с циклами?»

«Не только циклы», — ответил Саймон. «Ты также узнаешь кое-что новое о списках. Да, я знаю, что ты устал от моих лекций, но давай сегодня покончим с этим. Это заставит твою программу снова работать правильно — разве это того не стоит?» — и он подмигнул брату.

Эрик устал, но этот баг (да, он выучил это слово!) в меню вкусов очень раздражал, и он хотел его исправить. — «Ладно», — вздохнул он, — «давай исправим это. Что ты хотел мне рассказать о списках?»

«Посмотри на свои условия», — сказал Саймон. — «Ты использовал простую проверку if-else и проверили введенные данные по трем допустимым ответам: 1, 2 и 3. Но что, если твой список элементов будет длинным, например, из 20 элементов? Твой блок if-else будет слишком длинным. В Python есть другой способ. Мы можем проверить, присутствует ли определенный элемент в списке. В нашем случае мы можем проверить, находится ли ответ, полученный от пользователя, в списке 1, 2, 3. Мы можем попробовать это для первого меню. Позволь мне показать тебе."

Саймон взял клавиатуру у Эрика и изменил строку `if choice == . . .` в его функции к следующим образом:

```
. . .
while True:
    choice = input(prompt)
    if choice not in ['1', '2', '3']:
        answer = choices[int(choice) - 1]
        break
    else:
        print("Enter number 1, 2, or 3!")
        answer = ''
. . .
```

«Это требует некоторого объяснения», — сказал он. «Смотри, пользователь вводит строку, которая может иметь значение 1 или 42. Мы проверяем, находится ли эта строка в списке допустимых ответов (1, 2 или 3). Если ее нет в списке, мы печатаем сообщение об ошибке и продолжаем цикл. Если она есть в списке допустимых ответов, мы конвертируем ее и выбираем этот пункт из списка меню».

«Да, я понимаю», — сказал Эрик. — «А как насчет второго меню, где у меня возникла ошибка? Она не позволила мне выбрать номер 4, потому что его не было в моем if. С твоим списком у меня будет та же проблема. У меня есть пять вкусов, поэтому мне нужен еще один список ответов, верно?»

«Отличный вопрос!» — сказал Саймон. — «Я как раз собирался спросить об этом сам. И ты прав. Каждый список меню должен иметь свой собственный список допустимых ответов. Это не большая проблема; мы можем создать его, если знаем, что находится в нашем списке меню. Давай я сначала напишу, а потом объясню все шаг за шагом».

Саймон добавил следующие строки непосредственно перед блоком `if`, который он добавил ранее:

```
...
while True:
    choice = input(prompt)
    allowed_answers = []
    for a in range(1, len(choices)+1):
        allowed_answers.append(str(a))
    if choice not in allowed_answers:
        answer = choices[int(choice) - 1]
        break
    else:
        print("Enter number 1, 2, or 3!")
        answer = ''
...

```

Пустой список  
допустимых ответов

Вычисляет  
длину меню

Добавляет  
допустимый  
ответ

«Сначала мы создаем пустой список разрешенных ответов. Затем мы измеряем длину списка меню с помощью функции `len()`. Для твоих напитков результат будет три, а для вкусов — пять. Затем мы используем функцию `range()` для создания *последовательности* чисел от единицы до длины меню. Для напитков последовательность будет 1, 2, 3. Для вкусов — 1, 2, 3, 4, 5. Ты понял идею. Обрати внимание, что в функции `range()` нам следует использовать не *последний* элемент последовательности, а тот, который находится *после последнего*, который не входит в последовательность. Вот почему нам нужно добавить единицу к длине меню, вот так:

```
len(choices)+1
```

«Наконец, в цикле `for` мы конвертируем каждое число из последовательности в строку и добавляем его в конец списка допустимых ответов. Эта функция называется `append()`».

Саймон закончил свое объяснение и добавил: «Да, для первого раза это немного сложно, но попробуй прочитать код Python самостоятельно, и ты поймешь его, как если бы это был простой текст на английском языке.

«Теперь нам нужно изменить наш список с 1, 2, 3 на список допустимых ответов, который мы только что создали», — Саймон внес это изменение, посмотрел на код и хлопнул себя по лбу. — «О, я только что заметил!»

«Что?» — Эрик думал, что они уже закончили, но, похоже, осталось еще что-то.

«Мы также должны изменить наше сообщение», — сказал Саймон. — «Поскольку наша функция теперь может принимать списки меню любой длины, мы должны сказать пользователю что-то вроде: «Введите число от 1 до 6» или «от 1 до 12», в зависимости от длины нашего меню. Помнишь, как мы можем получить длину списка?»

«С помощью функции `len()` ?» — спросил Эрик.

«Конечно!» — сказал Саймон и внес последнее изменение в функцию. Теперь она выглядела так:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    while True:
        choice = input(prompt)
        allowed_answers = []
        for a in range(1, len(choices)+1):
            allowed_answers.append(str(a))

        if choice in allowed_answers:
            answer = choices[int(choice) - 1]
            break
        else:
            print("Enter number from 1 to ", len(choices))
            answer = ''

    return answer
```

`len()` используется  
для расчета  
количества  
вариантов выбора.

**Ваш ход! Добавьте список `allowed_answers`**

Добавьте список допустимых ответов в вашу функцию. Проверьте, позволяет ли она использовать списки меню разной длины.

«Теперь мы защищены от ошибок пользователей!» — сказал Саймон. «Попробуй и посмотри, сработает ли это!»

Эрик снова запустил программу и ввел 42, coffee и weryiuryt587456, но программа не вылетела, как раньше. Каждый раз он напоминал ему, что ему следует использовать число, и оно должно быть от 1 до 3 или 5, в зависимости от меню.

«Это здорово! Она работает и не вылетает!» — Эрик был очень рад, что создал такую надежную программу.

«Еще одна вещь», — сказал Саймон. — «Она будет последней на сегодня, я обещаю!»

«Хорошо», — сказал Эрик. Ему начало нравиться программирование. Ему нравилось, что его программа теперь выглядела как настоящая — и она работала! Даже если это еще не было онлайн- или мобильное приложение, оно работало как чат с кофейней. Эрик представил, что он переписывается со своей любимой кофейней, заказывает напитки, а затем приходит за ними.

Саймон сказал: «Твоя программа не позволяет мне вводить ничего, кроме цифр от 1 до 3 или 5. Но что, если я захочу что-то пропустить? Типа, я не хочу, чтобы в моем напитке были какие-то топинги?»

«В моем меню вкусов есть слово «plain», что означает «без вкуса». Я могу добавить то же самое в топинги», — ответил Эрик.

«Это сработает», — сказал Саймон. — «Но в целом в каждом меню ты должен предоставить пользователю возможность выйти из меню. Обычно для этого используют что-то вроде «Нажмите X, чтобы выйти из этого меню». Я думаю, нам следует добавить это и в нашу функцию меню».

«Как мы сделаем это?» — спросил Эрик. Он подумал немного и сказал: — «Я знаю! Мы добавим «X» в список допустимых ответов! Я прав?»

"Абсолютно!" — Саймон был очень рад видеть, что его брат добился таких успехов. — «Помнишь, мы использовали функцию `append()` для добавления элементов в список? Мы можем использовать ее здесь, сразу после того, как закончим добавлять номера в `allowed_answers`».

«Дай мне попробовать», — сказал Эрик и начал печатать.

«Конечно, давай», — подбодрил брата Саймон. — «Просто не забудь добавить ее после цикла `for`. Лучше добавить после нее пустую строку; таким образом ты сможешь увидеть, что это не часть цикла».

Вот новая версия функции меню Эрика:

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "): print(title)
    print(len(title) * "-")
```

```
i = 1
for c in choices: print(i, c)
    i = i + 1
while True:
    choice = input(prompt)
    allowed_answers = []
    for a in range(1, len(choices)+1):
        allowed_answers.append(str(a))

    allowed_answers.append('X')           <— 'X' теперь является допустимым ответом.
```

```

    if choice in allowed_answers:
        answer = choices[int(choice) - 1]
        break
    else:
        print("Enter number from 1 to ", len(choices))
        answer = ''

return answer

```

«Прекрасно», — сказал Саймон. — «Я бы еще добавил строчную букву 'х', потому что именно ее напечатает большинство людей. Что же нам делать, если пользователь вводит 'х'?»

«Выйти из цикла меню?» — спросил Эрик.

«Верно! Но что мы собираемся вернуть в основную программу? Обычно мы возвращаем выбор пользователя из меню: кофе, шоколад или что-то еще. Что, если пользователь введет 'х'? Что нам следует вернуть в таком случае?»

«Ничего?» — предположил Эрик.

«Да, мы возвращаем пустую строку», — сказал Саймон. — «Если пользователь вводит 'х', мы присваиваем пустую строку (' ') переменной answer и возвращаем answer так же, как мы это делаем, если ответ находится в меню».

Саймон продолжил: «Важная вещь: тебе следует выполнить эту проверку до того, как ты попытаешься произвести преобразование в число, но после того, как ты проверишь, есть ли оно в списке allowed\_answers. Ты видишь, куда поместить эту проверку?»

«Да, сразу после этой строки: if choice in allowed\_answers».

«Великолепно! Добавь ее. В этом случае у тебя будет вложенный оператор if — один if внутри другого if. Это довольно распространенное явление — иногда ты можешь увидеть три уровня вложенности if или даже больше. Просто убедись в правильности отступов. Именно так Python сообщает компьютеру, что делать, если условие истинно или ложно».

Эрик еще немного поработал над своим кодом и наконец получил следующее:

#### Листинг 4.1 Финальная версия меню Эрика

```

def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    print(title)
    print(len(title) * "-")
    i = 1
    for c in choices:
        print(i, c)
        i = i + 1
    while True:
        choice = input(prompt)
        allowed_answers = []
        for a in range(1, len(choices)+1):

```

```

        allowed_answers.append(str(a))

    allowed_answers.append('X')
    allowed_answers.append('x')

    if choice in allowed_answers:
        if choice == 'X' or choice == 'x':
            answer = ''
            break
        else:
            answer = choices[int(choice) - 1]
            break
    else:
        print("Enter number from 1 to ", len(choices))
        answer = ''

    return answer

```

### **Ваш ход! Добавьте опцию выхода**

Добавьте опцию 'x' в списки допустимых ответов. Затем добавьте вложенный `if`, чтобы проверить ее. Проверьте, работает ли это. Что, если вы введете 'x' во всех меню? Какой заказ вы получите в таком случае?

Он протестировал программу, введя x во всех трёх меню, и получил то, что ожидал:

```

Erik's Menu
-----
1 chocolate
2 coffee
3 decaf
Choose your item: x
Erik's flavors
-----
1 caramel
2 vanilla
3 peppermint
4 raspberry
5 plain
Choose your flavor: x
Erik's toppings
-----
1 chocolate
2 cinnamon
3 caramel
Choose your topping: x
Here is your order:
Main product:
Flavor:
Topping:
Thanks for your order!

```

«Пустой заказ!» — воскликнул он.

«Верно, так и должно быть», — подтвердил Саймон. — «Мне нравится твоя программа», — продолжил он. «Она работает — это первая и самая важная задача. Она удобна для пользователя и дает ему инструкции о том, что делать — это вторая задача. Также программа проверяет ввод и не позволяет пользователю вводить неверные значения — это третья задача».

«Давай подведем краткий итог тому, что ты узнал сегодня», — сказал Саймон. — «Что было первым?»

«Сначала ты снова сломал мою программу!» — ответил Эрик. На этот раз он не очень злился, потому что знал, что вместе с Саймоном исправил программу. — «И ты сказал мне, что я всегда должен думать о том, как мои пользователи могут использовать программу не так».

«Да, иметь дело с упрямыми пользователями, которые не хотят следовать твоим инструкциям, — это часть работы программиста», — сказал Саймон.

«Затем мы написали цикл меню, в котором проверяли, что ввел пользователь, и не позволяли ему использовать недопустимые ответы. Потом ты рассказал мне о функции `append()`, которая добавляет в список элементы».

Эрик продолжил: «Затем я написал проверки вложенных `if`, и теперь мой код выглядит как настоящие программы, которые показывают в фильмах».

Саймон улыбнулся: «Поверь мне, программы, которые показывают в фильмах, очень редко являются настоящими программами. Но ты прав, твоя программа становится сложнее. Она использует разные операторы Python, все эти циклы, `if` и списки».

«А потом мы добавили в меню опцию 'х', и теперь любой пользователь может получить пустой заказ!» — хихикнул Эрик.

«Да, а почему бы и нет?» — спросил Саймон. — «Ты не должен заставлять своих пользователей всегда что-то заказывать, но ты должен дать им возможность отменить заказ или выйти из меню».

Саймон продолжил: «Программа сейчас действительно хороша. Завтра я попрошу тебя перестать быть программистом и стать менеджером кофейни».

«Разве я уже менеджер?» — спросил Эрик.

«Да, это так», — улыбнулся Саймон. — «А теперь представь, что ты, менеджер кофейни, только что получил новый вкусовой компонент для своих кофейных напитков. И ты хочешь добавить эту изюминку в меню, а также пару новых топингов. Что бы ты сделал?»

«Я бы добавил эти топинги в список топингов. Ничего страшного», — ответил Эрик.

«Верно, но что, если ты *всего лишь* менеджер, а не программист? Ты ничего не знаешь об этой программе и не знаешь Python, но хочешь добавить в меню эти вкусы и топинги. Ты, как программист, должен предоставить менеджеру простой способ добавлять что-либо в меню».



«Как ты предлагаешь мне это сделать?» – спросил Эрик. Он уже знал, что Саймон что-то задумал.

«Я думаю, нам следует поместить меню в файлы и читать наши списки из этих файлов».

«Как в документах Word?» — спросил Эрик.

«Да, почти так», — ответил Саймон. — «Твоя программа откроет эти файлы и прочитает их. Я думаю, проще всего будет иметь по одному файлу для каждого меню. В одном файле будут храниться все напитки, в другом — все вкусы, в третьем — все топинги. В таком случае ваш менеджер сможет просто редактировать эти файлы вместо того, чтобы редактировать ваш код Python. Звучит неплохо?»

«Да, неплохо», — сказал Эрик. Он задавался вопросом, как его программа на Python будет открывать файлы так же, как это делает Word.

«Великолепно», — сказал Саймон. — «Это то, что мы собираемся сделать завтра. А сейчас отдыхай».

### **Новое, что вы узнали сегодня**

- *Пользователи совершают ошибки* — ты узнал, что пользователи не всегда следуют указаниям, которые ты им даешь в своей программе. Ты должен быть к этому готов и должен проверять их ввод на предмет ошибок, некорректность типов и так далее.
- *Отступы* — Когда вы создаете *блок* в Python (например, `while`), вы должны убедиться в том, что весь код в блоке имеет *отступы*, то есть он смещен вправо.
- *Как выйти из меню* — ты должен предоставить пользователям возможность выхода из каждого меню. Например, если они не хотят заказывать топинг, они должны иметь возможность пропустить это меню.

### **Код для этой главы**

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch04>.

# 5

## Работа с файлами: Быть менеджером кофейни

---

### В этой главе

- Эрик узнает, что нужно менеджеру кофейни
- Эрик читает свои меню из файлов
- Эрик пишет свою вторую функцию на Python

«В прошлый раз ты сказал что-то о менеджере кофейни», — напомнил Эрик своему брату на следующий день. — «Что-то насчет изменения меню? Я забыл».

«Да, я сказал, что было бы хорошо, если бы менеджер вашей кофейни мог менять меню, не вдаваясь в код Python», — сказал Саймон.

«Да, хорошая идея», — сказал Эрик. — «Не все знают Python».

«Моя идея — создать простые текстовые файлы для каждого меню — назовем их `drinks.txt`, `flavors.txt` и `toppings.txt`. Затем твоя программа сможет читать эти файлы и создавать списки из элементов, содержащихся в файлах».

«Почему ты назвал файлы с расширением `.txt`?» — спросил Эрик. — «Разве они не должны быть в формате `.docx`, чтобы менеджер мог редактировать их в Microsoft Word?»

«Хорошая мысль», — сказал Саймон. — «Да, менеджер, возможно, более знаком с Word, но нам нужен *простой текстовый файл* без шрифтов, заголовков и оглавления. Это похоже на ваш код на Python — эти файлы не должны содержать ничего, кроме строк простого текста, а менеджер должен использовать текстовый редактор для работы с ними. Когда я вызываю их с

расширением `.txt`, я сообщаю операционной системе — будь то Windows, macOS или Linux — что этот файл следует открывать в текстовом редакторе, а не в текстовом процессоре, таком как Word. Во всех этих системах всегда есть текстовый редактор, который может редактировать эти файлы. Вы также можете установить для этого другое приложение, как мы сделали это с редактором `Мн`. Кроме того, Python гораздо проще читать из обычного текстового файла, чем из файла `.docx`».

«Давай создадим эти файлы», — продолжил Саймон. — «Для этого ты можешь использовать свой редактор `Мн`. Только не забудь добавить расширение `.txt` при сохранении файлов. В противном случае автоматически будет добавлено `.py`.

«Создайте новый файл, укажите напитки в отдельных строках и сохраните его под именем `drinks.txt`. Затем сделайте то же самое для вкусов и топингов».

Эрик начал работать. Через несколько минут у него уже было три файла:

#### Листинг 5.1 `drinks.txt`

```
Coffee
Chocolate
decaf
```

#### Листинг 5.2 `flavors.txt`

```
caramel
vanilla
peppermint
raspberry
plain
```

#### Листинг 5.3 `toppings.txt`

```
chocolate
cinnamon
caramel
```

### **Ваш ход! Создайте свои собственные файлы меню**

Создайте текстовые файлы с пунктами меню, как это только что сделал Эрик. Убедитесь, что это *обычные текстовые* файлы. Попробуйте использовать в своем меню свои любимые вкусы и топинги.

«Что теперь?» — спросил он брата.

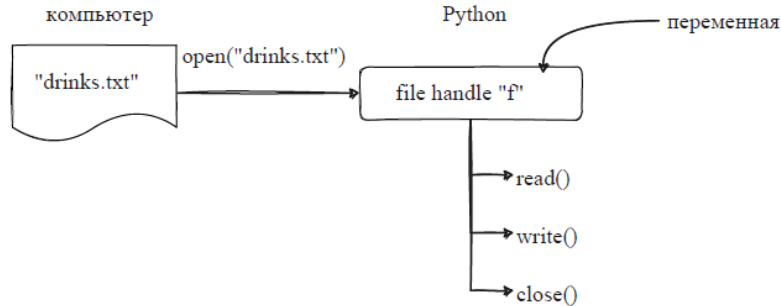
«Теперь давай научимся работать в Python с файлами. Создадим для этого новую программу. Немного потренируемся, а затем добавим этот код в нашу основную программу. Обычно я делаю так, когда изучаю что-то новое — пробую в отдельной простой программе, прежде чем добавлять код в основное приложение».

Саймон продолжил: «Создай еще один файл в редакторе и сохраните его как

files.py ».

Эрик делал это уже несколько раз, так что это заняло у него всего пару секунд.

«Работа с файлами — сложная тема, поэтому я начну с диаграммы», — сказал Саймон.



«Когда вы работаете с файлом на своем компьютере, вы используете его имя. Вы велите программе-редактору *открыть* файл `drinks.txt`. Затем ваша программа-редактор *читает* файл и показывает вам его содержимое. Затем вы редактируете файл и сохраняете его, то есть *записываете* файл на диск компьютера. Пока все идет нормально?» — спросил Саймон.

«Да», — сказал Эрик. — «Но диски используются только в очень старых компьютерах. В моем компьютере это называется SSD-накопитель — это уже не диск. Мне рассказал это мой друг Алекс».

«Ты абсолютно прав!» — Саймон был рад услышать это от брата. — «Да, сейчас в большинстве компьютеров используется SSD — это не диск и у него не подвижных частей».

«Но давайте вернемся к моим файлам», — попросил Эрик. — «Я вижу, что ты написали «компьютер» и «Python» вверху своей картинки. Что это значит?»

«Это означает», — сказал Саймон, — «что в Python, если вы хотите работать с файлом, вам нужно создать специальный *объект*, который обычно называется *дескриптором файла*. Вы должны использовать этот объект для чтения и записи файла. Используйте функцию `open()` для создания такого объекта вы. Вызовите функцию `open()` и передаете имя файла в качестве аргумента. В нашем случае это будет `open("drinks.txt")`. Функция возвращает дескриптор файла, который вы помещаете в переменную. В данном случае переменная называется `f`, но вы можете использовать здесь любое имя».

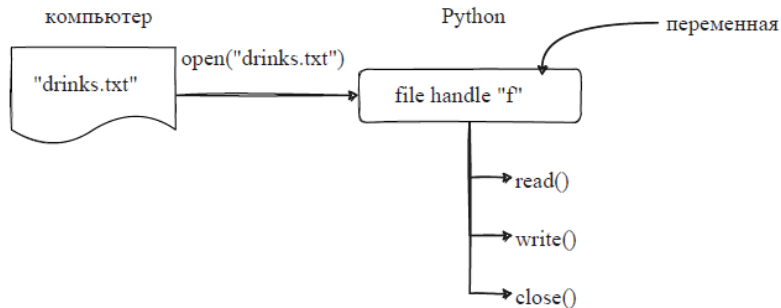
«Почему все так сложно?» — спросил Эрик. — «Почему мы не можем просто использовать имя файла?»

«Да, в первый раз это выглядит немного сложно. Причина в том, что имя файла — это просто строка, помнишь? Когда мы хотим прочитать из файла, мы хотим прочитать из *файла с таким именем*, а не из строки. *Имя файла* и собственно сам файл — это разные вещи. Когда мы используем функцию `open()`, мы создаем связь между именем файла и самим файлом. Мы говорим

Python: «Пожалуйста, найди файл с именем `drinks.txt` в компьютере и используй его *как файл*». Не беспокойся об этом слишком сильно прямо сейчас. Иногда лучший способ понять что-то — начать это использовать».

«Хорошо», — сказал Эрик. Все это его все еще немного смущало, но он хотел посмотреть, как можно читать меню из только что созданных текстовых файлов

«Теперь давай напомним простую программу на Python для работы с файлами», — сказал Саймон. — «Перейди на вкладку в своем редакторе, где открыта программа `files.py`. Посмотри на мою схему: ты должен вызвать функцию `open()`; передать имя файла, например `drinks.txt`; и сохранить результат в переменной `f`. Можешь написать код для этого?»



«Дай мне попробовать», — сказал Эрик и написал следующее:

```
f = open("drinks.txt")
```

"Хорошо!" — сказал Саймон. «Теперь у тебя есть файловый объект с именем `f` и ты можешь читать из него. Для этого ты вызываешь *метод*, называемый `read()`. Методы выглядят и ведут себя во многом как функции, но методы применяются к *объектам*. Мы поговорим об объектах позже, а пока все, что тебе нужно знать, это то, что для вызова метода объекта ты используешь имя объекта, после ставишь точку, а затем имя метода в круглых скобках, аналогично вызову функции, например: `f.read()`. Методы также могут возвращать что-либо, например функции, поэтому вам нужно где-то сохранить результат. Как ты думаешь, что вернет этот метод `read()`, когда ты его вызовешь?»

«Я думаю то, что находится внутри файла», — ответил Эрик

«Абсолютно верно!» — сказал Саймон. — «Сохрани это в переменной `drinks`, а затем попытайся вывести на печать».

Эрик написал следующий код:

```
f = open("drinks.txt")
drinks = f.read()
print(drinks)
```

«А теперь запусти его», — предложил Саймон.

Эрик нажал Run и получил такой результат:

```
Coffee  
Chocolate  
decaf
```

```
>>>
```

**Ваш ход! Прочитайте содержимое файла**

Напишите такую же короткую программу и попробуйте прочитать файл `drinks.txt`. Убедитесь, что вы можете вывести на печать все содержимое файла.

«Она работает!» — Он был очень рад. Его программа на Python открыла файл, прочитала его и распечатала содержимое на странице! — «Теперь я знаю, как распечатать меню в Python. Позволь мне написать то же самое для двух других файлов».

«Да», — сказал Саймон. — «Но это не совсем то, чего мы хотим».

«Почему?» — Эрик не мог понять.

«Помнишь, что в своей программе ты не просто выводил меню. Ты также позволяешь пользователю осуществлять выбор в меню, а затем находишь этот элемент в списке, верно?»

«Да, но разве здесь нет списка?» — спросил Эрик. — «Это же похоже на список».

«Это может выглядеть как список, но это не список. Это строка», — сказал Саймон. — «Когда ты вызвал метод `read()`, ты скопировал все содержимое файла в переменную `drinks`. Итак, эта переменная представляет собой всего лишь одну большую строку. Если ты мне не веришь, ты можешь проверить это прямо здесь. Видишь эти три угловые скобки в окне вывода? Ты можешь ввести здесь любую команду Python, чтобы продолжить работу с программой — так же, как мы это делали с REPL, помнишь? Введи здесь `type(drinks)` и ты увидишь тип этой переменной».

Эрик сделал это и увидел следующее:

```
>>>
type(drinks)
<class 'str'>
>>>
```

«Видишь, Python говорит, что это строка», — сказал Саймон. — «А нам нужен список».

«Что нам нужно сделать?» — спросил Эрик.

«К счастью, разработчики Python знали, что нам это может понадобиться, и создали другой метод для файлового объекта. Он носит наименование `readlines()`. Попробуй сменить `read()` на `readlines()` и посмотри, что произойдет. Не забудь нажать Stop перед повторным запуском программы».

Эрик модифицировал свою программу следующим образом:

```
f = open("drinks.txt")
drinks = f.readlines()
print(drinks)
```

Он нажал Stop, а затем снова Run и получил такой результат:

```
['coffee\n', 'chocolate\n', 'decaf\n']
>>>
```

«Попробуй еще раз проверить тип», — предложил Саймон.

Эрик переключился на окно вывода и снова напечатал `type(drinks)`:

```
>>>
type(drinks)
<class 'list'>
>>>
```

### **Ваш ход! Проверьте типы с помощью Python**

Повторите проверки, которые только что сделал Эрик. Видите разницу между строкой и списком?

«Это список!» — воскликнул он. — «Но что это за символы с косой чертой \n? У меня их нет в моей папке с напитками».

«Ты *не видишь* их в своем файле, но они там есть. Это *невидимые* символы. Когда ты видишь *обратную косую черту-n*, это один символ, называемый символом *новой* строки. Он сообщает компьютеру, что он должен напечатать следующий элемент в начале следующей строки. Без него все ваши напитки были бы напечатаны так: `coffeechocolatedecaf`. Ты же этого не хочешь, верно?» — улыбнулся Саймон.

«Конечно, нет!» — сказал Эрик. — «Но они нам не нужны в списке, верно? Я думаю, что списки меню должны выглядеть так же, как в моей основной программе, верно?»

"Ты абсолютно прав. И снова разработчики Python создали для этого очень полезный метод. Он называется `strip()` и он удаляет невидимые символы с обоих концов строки. Нам просто нужно применить его к каждому элементу в списке. Как ты думаешь, что нам следует использовать для этого?»

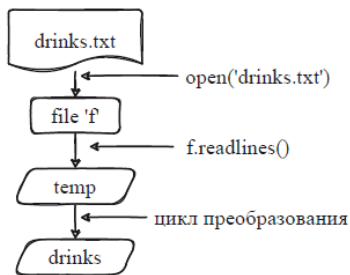
"Цикл?" — предположил Эрик.

«Правильно, цикл!» — сказал Саймон. — «Мы просмотрим список и удалим эти символы новой строки с помощью метода `strip()` ».

Саймон на мгновение остановился, задумавшись. Затем он продолжил: «Есть несколько способов сделать это. Некоторые из них короче, но их труднее понять. Давай воспользуемся тем, который легче читать и которому легче следовать. На самом деле это хорошее правило в программировании: выбирая между различными способами выполнения чего-либо, всегда используйте тот, который легче читать и которому легче следовать. Если кто-то читает твой код, он будет тебе за это благодарен. Даже ты сам, если три месяца спустя будешь читать свой собственный код».

Он быстро нарисовал схему.





«Давай прочитаем временный список из файла. Затем мы пройдемся по этому временному списку, преобразуем каждый элемент и добавим его в новый список. Мы назовем этот новый список `drinks`. Затем повторим те же действия для вкусов и топингов. Мы можем использовать для всех них одну и ту же временную переменную. Позволь мне помочь тебе», — сказал Саймон. Он начал вводить программу Эрика. Вот как это выглядело после того, как он закончил:

```
f = open("drinks.txt")
temp = f.readlines()
drinks = []
for item in temp:
    new_item = item.strip()
    drinks.append(new_item)

print(drinks)
```

Он нажал Run и они увидели результат:

```
['coffee', 'chocolate', 'decaf']
>>>
```

### **Ваш ход! Удалите символы новой строки**

Удалите символы новой строки из пунктов меню с помощью метода `strip()`.

«Выглядит лучше, не так ли?» — спросил он Эрика. — «Теперь сделай то же самое с другими файлами меню».

Эрик начал работать над своей программой, и когда он почти закончил работу с файлом `flavors.txt`, он воскликнул: «Подожди! Я повторяюсь! Ты сказал мне, что мне не следует повторяться».

Саймон улыбнулся. Он был счастлив, что его младший брат так быстро уловил эту концепцию.

«Что нам делать, чтобы не повторяться?» — спросил он.

«Написать функцию?» - спросил в ответ Эрик.

«Да, точно! Посмотри на этот код: что осталось прежним, а что изменилось? Что будет аргументом и что эта функция вернет?»

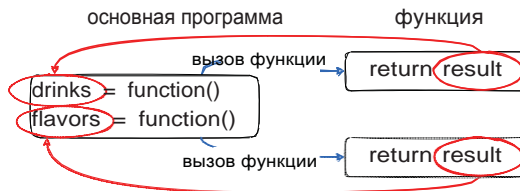
Эрик начал думать вслух: «Я открываю разные файлы, поэтому имя файла

должно быть аргументом, верно?»

«Верно», — подтвердил Саймон. — «Что ты хочешь вернуть?»

«Думаю, я верну список с опциями меню — после того, конечно, как мы удалим символы новой строки. Но как мне назвать этот список?»

«Ты можешь называть его как угодно, потому что снаружи его *не видно*. Назовите его, например, `result`. Затем ты можешь написать `return result` в конце функции. Когда ты вызываешь функцию, этот список `result` *присваивается* переменной в основной программе, например `drinks` или `flavors`, в зависимости от файла, который ты читаешь. *Внутри* функции переменная всегда будет называться `result`, но *вне* функции, в основной программе, ты можешь присвоить результат любой переменной».





«Давай, напиши эту функцию», — подбадривал Саймон своего брата. — «Помнишь, как это сделать? Начните с `def`, назови функцию, например, как `read_menu`, передай `filename` в качестве аргумента и скопируй только что написанный код».

«Хорошо, я попробую», — сказал Эрик. Он начал писать свою вторую функцию на Python. Саймон немного помог брату, и вот что они вместе написали:

#### Листинг 5.4 `files.py`

```
def read_menu(filename):
    f = open(filename)
    temp = f.readlines()
    result = []
    for item in temp:
        new_item = item.strip()
        result.append(new_item)

    return result

drinks = read_menu("drinks.txt")
print(drinks)
flavors = read_menu("flavors.txt")
print(flavors)
toppings = read_menu("toppings.txt")
print(toppings)
```

Эрик сохранил файл и нажал Run. Разумеется, он получил ожидаемый результат:

```
['coffee', 'chocolate', 'decaf']
['caramel', 'vanilla', 'peppermint', 'raspberry', 'plain']
['chocolate', 'cinnamon', 'caramel']
>>>
```

#### **Ваш ход!** Создайте функцию `read_menu`

Создайте функцию `read_menu` так же, как это сделал Эрик. Убедитесь, что в именах файлов нет опечаток, но что, если они есть? Попробуйте изменить имя файла и посмотрите, какую ошибку выдает Python. Не забудьте исправить имя файла, чтобы ваша программа снова работала.

«Это моя вторая функция, и она работает!» — с гордостью сказал он своему брату.

«Да, ты уже создаешь свою собственную библиотеку функций — отлично!» — сказал Саймон. — «Теперь давай скопируем твою новую функцию в основную программу. Не копируй строки `print()` — мы использовали их только для тестирования. Я думаю, этот файл называется `menu_function.py`, верно?»

«Верно», — сказал Эрик. — «Но куда мне поместить свою функцию в этом

файле? В начале или в конце?»

«Правило Python заключается в том, что вы должны определить свою функцию, *прежде* чем начать ее использовать. По этой причине функции обычно размещаются в начале файла, перед основной программой. Вы можете разместить ее сразу после первой функции».

«Хорошо», — сказал Эрик. Он начал работать и вот что получил:

#### Листинг 5.5 Меню Эрика с файлами

```
def menu(choices, title="Erik's Menu", prompt="Choose your item: "):
    . . .
    # this function didn't change
    . . .
    return answer

def read_menu(filename):
    f = open(filename)
    temp = f.readlines()
    result = []
    for item in temp:
        new_item = item.strip()
        result.append(new_item)

    return result

drinks = read_menu("drinks.txt")
flavors = read_menu("flavors.txt")
toppings = read_menu("toppings.txt")

drink = menu(drinks)
flavor = menu(flavors, "Erik's flavors", "Choose your flavor: ")
topping = menu(toppings, "Erik's toppings", "Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drink)
print("Flavor: ", flavor)
print("Topping: ", topping)
print("Thanks for your order!")
```

#### **Ваш ход! Скопируйте свою функцию в основную программу**

Скопируйте новую функцию `read_menu()` в свою основную программу и попробуйте ее.

Он протестировал программу, она работала точно так же, как и раньше.

«Это хорошо», — сказал Саймон. — «Теперь попробуй добавь что-нибудь в файл топингов `toppings`. Посмотри, изменилось ли от этого меню».

Эрик открыл файл `toppings.txt`, добавил в конец файла `'vanilla powder'` и сохранил его. Он снова запустил программу, и в последнем меню появилась дополнительная строка: `4 vanilla powder`.

**Ваш ход! Добавьте еще один элемент**

Добавьте новый пункт в одно из меню. Измените один из элементов. Не забудьте сохранить файлы меню после их изменения. Проверьте, выводит ли ваша программа обновленные меню.

«Это хорошо, мне это нравится!» — сказал Эрик. — «Теперь любой, кто может редактировать текстовый файл, может изменить меню! Подожди . . .» — у него возникла идея. — «Значит я могу поместить в эти меню *что угодно!* Мороженое, бутерброды или что угодно! Круто, мне нравится! Я должен рассказать об этом своему другу Алексу — ему нравятся минифигурки LEGO. Возможно, он сможет использовать эту программу для обмена фигурками с друзьями!»

«Абсолютно верно!» — сказал Саймон. — «Я рад, что у тебя так много идей о том, как использовать свою программу. Отлично! У меня тоже есть кое-какие идеи, но нам лучше поговорить о них завтра. Завтра нам также будет нужно создать главное меню».

«Что ты имеешь в виду?»

«Ну, в настоящее время тебе приходится запускать программу каждый раз, когда ты хочешь принять заказ. Ты принимаешь заказ, выводишь его, а затем твоя программа завершает работу. Было бы лучше, если бы твоя программа могла вернуться к начальному диалогу, в котором ты спрашиваешь имя клиента».

«Да, верно», — согласился Эрик. — «Это должно быть похоже на киоск, где вы что-то заказываете, нажимаете «Готово», и на первом экране появляется надпись «Добро пожаловать в наш магазин». Да, давайте сделаем это!»

«Давай подведем итоги сегодняшнего дня», — предложил Саймон. — «Что мы сделали сегодня?»

«Во-первых, ты сказал, что менеджер кофейни захочет редактировать наши меню в файлах. Затем я написал три файла с меню напитков, вкусов и топингов».

«Очень хорошо, что дальше?»

«Затем я открыл файлы и прочитал их. Сначала я читал построчно, но потом у меня появились эти странные символы «обратная косая черта». Затем мы использовали метод `strip()`, чтобы удалить их».

«Хорошо», — сказал Саймон. — «А ты помнишь, что я тебе говорил об объектах?»

«Не совсем. Ты сказал, что файл — это объект Python, и это не то же самое, что его имя. Еще ты сказал, что функции с объектами называются «методами».

«Да, это правда», — сказал Саймон. — «Объекты — сложные понятия. Сейчас мы просто будем использовать их и их методы, но больше узнаем о них позже. Мы также создадим свои собственные объекты и методы подобно тому, как мы это делали с функциями».

«Верно», — сказал Эрик. — «Ты напомнил мне — я написал свою вторую функцию на Python, и она сработала!»

«Действительно. Теперь ты становишься серьезным программистом», — сказал Саймон и улыбнулся. — «Давай пока отдохнем. Завтра мы сделаем твою

программу еще лучше».

### **Новое, что вы узнали сегодня**

- *Что значит открыть файл* — ты узнал разницу между именем файла и дескриптором файла внутри программы.
- *Что такое `\n` и как его удалить из строк* — ты узнал, что символ `\n` означает «начать новую строку». Он не нужен нам в пунктах меню, поэтому ты использовал функцию `strip()`, чтобы удалить его.
- *То, что вы возвращаете из функции, присваивается переменной в основной программе* — ты узнал, что переменная внутри функции не видна в основной программе. Чтобы передать ее значение, тебе необходимо вернуть эту переменную из функции и присвоить ее значение другой переменной в основной программе.

### **Код для этой главы**

Код для этой главы можно найти здесь:

<https://github.com/pavelanni/pythonicadventure-code/tree/main/ch05>.

# Главное меню: Следующий клиент!

---

## **В этой главе**

- Эрик создает главное меню для обслуживания большого количества клиентов
- Эрик узнает о словарях Python
- Саймон объясняет подход к разработке сверху вниз

«Вчера мы решили, что хотим создать главное меню», — напомнил Саймон Эрику.

«Да, ты сказал, что если бы я хотел использовать эту программу для обслуживания большого количества клиентов, мне пришлось бы повторять меню для каждого клиента — спрашивать их имя и то, что они хотят заказать».

«Абсолютно верно!» — ответил Саймон. — «И что ты собираешься использовать для этого?»

«Может быть, цикл? Как мы это сделали с меню. Мы будем повторять это до тех пор, пока клиент не наберет правильные цифры или не введет x».

«Ты абсолютно прав!» — Саймон был очень рад, что его брат так быстро уловил эту идею программирования. — «Мы спрашиваем у клиентов имя, как в нашей первой программе. Тогда мы получим их заказ со всеми вкусами и топингами».

«Да», — продолжил Эрик, — «и тогда мы предоставим им выбор: совершить



или отменить заказ? Я видел это на некоторых сайтах».

"Верно. Когда они нажимают «Order», мы сохраняем заказ и распечатываем его для бариста. Если они отменят заказ, мы просто забудем об этом. В обоих случаях мы возвращаемся в начало главного меню и спрашиваем имя следующего клиента».

Саймон взял лист бумаги и начал рисовать. «Мы должны сначала спланировать этот алгоритм визуально. Когда мы договоримся, как он должен себя вести, можно приступать к написанию кода. Всегда полезно описывать программу простыми словами и диаграммами, прежде чем приступить к написанию кода».

«Первое, что мы здесь делаем, — это получаем заказ», — сказал Саймон. — «Понимаешь, я указал это здесь как ‘Получаем заказ’».



«А где все наши меню со вкусами и топингами? Почему ты не поместил их туда?» – спросил Эрик.

«Я решил использовать блок «Получаем заказ», который *содержит* все меню. Обычно сначала думают большими блоками, а затем работают над деталями каждого блока отдельно. Это еще одна причина, по которой программисты используют функции. Сначала они думают о программе большими блоками, а затем описывают каждый блок в отдельной диаграмме. Представь себе, если бы мы включили каждую изначальную информацию нашей программы в одну диаграмму. Понять основной алгоритм было бы невозможно!»

«Продолжим», — сказал Саймон. «Мы получаем заказ и просим клиента его подтвердить. Если заказ отменяется, возвращаемся в самое первое меню: «Welcome to Erik’s Coffee Shop». Но если клиент подтвердит заказ, то нам следует его сохранить и распечатать».

«Хорошо, я понимаю, что нам нужно распечатать его, чтобы приготовить напитки», — сказал Эрик. — «Но почему мы должны его сохранять? И что ты

подразумеваешь под *сохранением* заказа?»

«Во-первых, было бы неплохо в конце дня проверить, скольких друзей ты обслужил в своей кофейне, ты не находишь?»

«Я уже это знаю,» — сказал Эрик. — «В тот день я приготовил пять напитков».

«Но мы говорим о *настоящей* кофейне, не забывай это. Она работает каждый день и обслуживает десятки и сотни клиентов. Пара моих друзей работала в разных кофейнях, и могу вас заверить, они прекрасно знают, сколько клиентов они обслуживают каждый день».

«Другая причина», — продолжил Саймон, — «заключается в том, что менеджер кофейни должен знать, что у них есть в продаже и что им нужно дозаказывать. Помнишь, мы говорили об изменении меню? Допустим, вкус карамели не заказали вовремя, и его пришлось убрать из меню. Почему его не заказали? Потому что никто не считал, сколько порций карамельного вкуса заказали клиенты. Если мы сохраним все заказы и проанализируем их, мы сможем заказать вкусы и топинги до того, как нам перестанет их хватать».

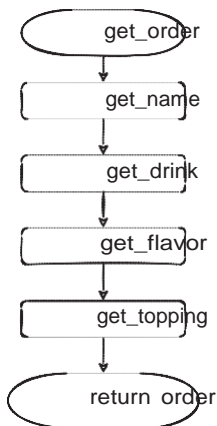
«Я не думал об этом», — сказал Эрик. — «Да, это хорошая идея — сохранять заказы. Но как мы можем это реализовать?»

«Есть несколько способов», — сказал Саймон. — «Мы можем использовать файлы или базы данных. Конечно, все серьезные приложения используют базы данных. Я думаю, нам следует начать с файлов, а затем, если у тебя хватит смелости, мы сможем использовать и базы данных».

«Да, я хочу, чтобы моя программа была похожа на эти *серьезные* приложения!» — сказал Эрик. — «Также я хочу попробовать использовать базы данных!»

«Хорошо», — сказал Саймон, — «но пока давай покончим с главным меню. Скоро мы начнем сохранять заказы».

«Что касается заказов», — сказал Саймон, взяв еще один листок бумаги, — «вот что находится внутри блока «Получаем заказ».



**Ваш ход! Создайте свою собственную диаграмму**

Если вы решили работать с магазином другого типа, создайте диаграмму для *вашей* функции `get_order()`.

«Мы уже написали эту функцию. Мы просто не называли это функцией. Видите ли, мы уже создали диалоги, чтобы спросить у клиента имя, напиток, вкус и топинг. Есть только одна вещь, которую мы еще не сделали. Вы ее видите?»

«Отказ от заказа?» — спросил Эрик. — «Мы этого не делали, но я не знаю, что это здесь значит».

«Вот как выглядит твой заказ, согласен?»

```
Order:
name:
Erik
drink:
decaf
flavor: vanilla
topping:
chocolate
```

«Мы используем функцию `get_order()` для сбора всей этой информации, но вместо того, чтобы возвращать четыре отдельных значения для имени, напитка, вкуса и топинга, я хочу вернуть одно значение, которое я бы назвал заказом. И эта единственная *вещь* содержит в себе несколько ценностей, которые в целом взаимосвязаны».

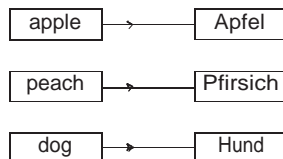
«Я знаю, ты хочешь использовать здесь список!» — поделился своим мнением Эрик.

«Это один из вариантов, но я имею в виду кое-что получше. В Python у нас есть *словари*. Что такое обычный словарь?»

«Это книга со словами и их значениями», — ответил Эрик. — «Или их переводами, если это, например, англо-немецкий словарь».

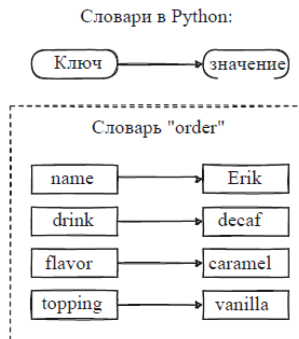
«Верно!» — сказал Саймон. — «У нас есть слово и значение, связанное с этим словом. Это может быть слово и его значение или его перевод, вот так», — и он нарисовал диаграмму.

Что такое словарь?



«В Python словарь использует слова, которые мы называем *ключами*, чтобы получить связанные с ними *значения*. Давай посмотрим на твой заказ. У тебя есть ключ с наименованием `name` и его значение — `Erik`. У тебя есть еще один

ключ с наименованием `drink` и его значение — `decaf` и так далее. Весь словарь называется `order`, и это то, что мы собираемся вернуть в результате выполнения этой функции.



«Давай еще раз попрактикуемся с REPL», — предложил Саймон. — «Нажми REPL, чтобы перейти к интерактивному сеансу».

Эрик щелкнул значок REPL и переключился на окно интерактивного сеанса. У Эрика это выглядело следующим образом, но ваша версия может выглядеть немного иначе — ничего страшного.

```
Jupyter QtConsole 4.7.7
Python 3.8.12 (default, Mar 12 2022, 19:58:23)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.1.1 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]:
```

«Мы начнем с создания пустого словаря под названием `order`. Чтобы создать в Python словарь, мы используем фигурные скобки, `{}`, чтобы они отличались от списков, в которых используются квадратные скобки, `[]`. Введи `order = {}`, а затем нажми Enter. Это будет твой заказ.»

Эрик typed:

```
In [1]: order = {}
```

```
In [2]:
```

“Now we can add items to your order. Let’s start with the name. Type `order['name'] = 'Erik'`. Then try to print the order with a simple `print()` function».

Эрик напечатал:

```
In [2]: order['name'] = 'Erik'
```

```
In [3]: print(order)
{'name': 'Erik'}
```

In [4]:

«Но ты сказал, что словари должны использовать фигурные скобки. Почему мы используем здесь квадратные скобки?» — спросил Эрик.

«Хороший вопрос». — Саймон настолько привык к этой возможности Python, что не мог сразу найти хороший способ объяснить ее. Он начал: «Ну, мы используем фигурные скобки, чтобы *создать* словарь. Но мы используем квадратные скобки для *доступа* к словарю, когда хотим получить из него элемент. В этом смысле он похож на списки: когда вы хотите получить элемент из списка, вы используете квадратные скобки. Разница в том, что в списках вы используете индексы, которые являются целыми числами. В словарях вы используете ключи, которые обычно представляют собой строки. Если бы у вас был обычный словарь и вы использовали его индекс (например, 546) для определения значения или перевода слова, это было бы неудобно, не так ли? Вместо этого вы используете само слово, например «собака», которое найти можно значительно быстрее».

«Да, верно», — сказал Эрик. — «Должен ли я добавить напиток, ароматизатор и топпинг сейчас?»

«Отличная идея, вперед!» - сказал Саймон.

Эрик продолжил свою интерактивную сессию:

```
In [4]: order['drink'] = 'decaf'
```

```
In [5]: order['flavor'] =
```

```
'vanilla'
```

```
In [6]: order['topping'] = 'chocolate'
```

```
In [7]: print(order)
{'name': 'Erik', 'drink': 'decaf', 'flavor': 'vanilla',
 'topping': 'chocolate'}
```

```
In [8]:
```

«Обрати внимание», — сказал Саймон, — «что твои ключи и значения всегда идут парами с двоеточием, :, между ними».

### ***Ваш ход! Изучите словари с помощью REPL***

Откройте REPL и поработайте со словарями. Вы можете повторять команды Эрика или создавать свои собственные словари. Попробуйте использовать разные клавиши. Попробуйте сохранить число вместо строки. Это работает?

«Это круто, мне нравится!» — сказал Эрик. — «Но могу ли я вывести это наилучшим образом, каким я осуществлял вывод раньше?»

«Конечно», — ответил Саймон. — «Я думаю, тебе следует написать для

этого новую функцию. Но нам лучше вернуться к редактору».

«Это будет моя третья функция», — прокомментировал Эрик.

«Ты все еще считаешь?» — Саймон улыбнулся. — «Я уверен, что ты скоро потеряешь счет написанным тобой функциям».

«Теперь давай начнем писать программу главного меню в вашем редакторе», — продолжил Саймон. — «Здесь мы будем использовать подход к проектированию *сверху вниз*».

«А это что такое?» — спросил Эрик.

«Это похоже на то, что я тебе только что показал: сначала мы разрабатываем алгоритм для всей программы. Мы решаем, какими будут большие блоки и как мы будем переходить от одного блока к другому. Обычно они включает в себя такие решения, как разрешение пользователям подтверждать или отменять заказ».

«Мы можем разработать основную программу и использовать такие функции, как `get_order()` или `print_order()`. Неважно, если у нас еще нет этих функций. Пока мы не напишем настоящие функции, мы можем писать очень простые функции, которые просто печатают сообщение типа: ‘Я — функция `print_order()`.’ Некоторые люди называют их *заполнителями*. Затем, когда мы увидим, что главное меню работает корректно и вызывает нужные функции, мы можем написать настоящие функции».

«Давай я тебе помогу.» — Саймон взял клавиатуру. — «Сначала мы создадим новый файл и сохраним его как `main_menu.py`. Затем мы создадим новую функцию под названием `main_menu()` с ключевым словом `def` и круглыми скобками».

«Другая функция?» — спросил Эрик.

«Да, в программировании мы обычно создаем функции для всего. Основная программа обычно очень короткая и вызывает одну из этих функций. Затем эта функция вызывает другие функции и так далее. Хорошей практикой является писать даже главное меню как функцию».

Саймон продолжил: «Посмотри на диаграмму еще раз. Ты видишь эти стрелки, которые возвращаются к пункту «Старт»? Обычно они означают, что в своем алгоритме ты собираешься что-то повторить. Как только мы возвращаемся к пункту «Старт», мы снова и снова проходим один и тот же алгоритм. А чтобы повторить что-то в программе, мы используем . . . что?"



«Цикл!» — ответил Эрик.

«Абсолютно верно!» — подтвердил Саймон. — «Мы уже использовали два типа циклов: цикл `for` и цикл `while`. Какой из них ты собираешься использовать здесь?»

«Я думаю, это должен быть цикл `while`», — сказал Эрик. — «Это главное меню похоже на то, что мы сделали в меню напитков: повторяем вопросы и проверяем, что ответил пользователь».

«Я согласен», — сказал Саймон. — «Посмотри на код, в котором ты написал меню как функцию. Помнишь, мы использовали `while True`: там мы проверяли то, что вводит пользователь. Как ты думаешь, что нам следует здесь проверить? Дам подсказку: на диаграммах те места, где нам предстоит принять решение, обычно изображаются в виде ромба».



«Вижу!» — сказал Эрик. — «Здесь мы спрашиваем пользователя, хочет ли он подтвердить или отменить заказ».

«Хорошо, давай начнем писать», — предложил Саймон.

Эрик написал:

```
def main_menu():
    while True:
```

«Что дальше?» — спросил он.

«Посмотри на диаграмму», — попросил Саймон.

«Навести порядок?» - спросил

Эрик..

«Верно! И помни, что функция `get_order()` вернет словарь с заказом. В словаре будет указано имя заказчика, напиток, вкус и все такое. Мы поместим этот словарь в переменную `order` в нашем главном меню», — Саймон добавил строку в код Эрика:

```
def main_menu():
    while True:
        order = get_order()
```

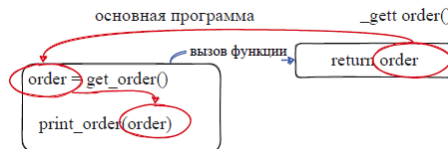
«Что дальше?» — спросил он брата.

«Теперь нам нужно спросить клиента, подтверждает ли он заказ», — сказал Эрик.

"Хорошо. Но я думаю, что прежде чем спрашивать, мы должны показать ему заказ», — и Саймон добавил еще несколько строк:

```
def main_menu():
    while True:
        order = get_order()
        print("Check your order:")
        print_order(order)
        confirm = input("Confirm? Press Y to confirm, N to cancel: ")
```

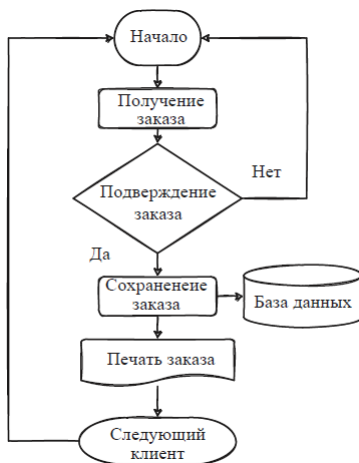
«Послушай», — сказал Саймон Эрику, — я использовал переменную порядка, полученную от `get_order()`, в качестве аргумента для следующей функции, `print_order()`. Это очень распространено в программировании: мы вызываем одну функцию, чтобы сделать что-то, она возвращает результат, а затем мы используем этот результат в качестве входных данных для другой функции».



«Понятно», — сказал Эрик. — «Как в кинотеатре: кассир распечатывает билет и дает его вам. Потом берешь билет, идешь на вход, отдаешь билет служащим на входе, и они его проверяют».

«Да, хорошая аналогия, Эрик! Давай продолжим: мы только что получили ответ на вопрос, хочет ли клиент подтвердить заказ. Теперь, как ты только что сказал, нам предстоит проверить ответ и решить, что делать дальше. Как в кинотеатре, где проверяют, действителен ли билет, и решают, пускать или нет на сеанс. Давай добавим эти строки. Взгляни на схему. Если пользователь ответит «Да», что нам делать?»





«Сохранить и распечатать заказ», — ответил Эрик, глядя на рисунок Саймона.

«Хорошо, а если пользователь захочет отменить заказ и ответит «Нет»?»

«Нам не следует ничего делать, просто вернуться к началу. Но я не знаю, как это сделать. У тебя там просто находится стрелка».

«В Python для этой стрелки есть простое слово — `continue`. Это означает: «Не выполнять оставшуюся часть цикла и продолжить цикл с начала». Довольно просто, да?» — и Саймон добавил в функцию следующие строки:

```
def main_menu():
    while True:
        order = get_order()
        print("Check your order:")
        print_order(order)
        confirm = input("Confirm? Press Y to confirm, N to cancel: ")
        if confirm == "Y" or confirm == "y":
            save_order(order)
            print("Thanks for your order:")
            print_order(order)
        else:
            continue
```

«Я вижу, что ты добавил еще две функции: `save_order()` и `print_order()`», — сказал Эрик. «Но мы их еще не написали».

«Давай напишем их!» — воскликнул Саймон. — «Сейчас мы напишем очень простые функции. Они просто выведут что-то вроде «сохранение заказа. . . », чтобы мы могли видеть, что они были вызваны. Позже мы улучшим их, чтобы они делали больше полезных вещей».

Саймон добавил функции ниже функции `main_menu()`:

```
def get_order():
    return {}

def print_order(order):
    print(order)
    return

def save_order(order):
    print("Saving order...")
    return
```

Он объяснил это Эрику: «Функция `get_order()` — это то, что ты уже написал. Мы скоро перенесем сюда твой код, но пока она ничего не делает. Меню и диалогов нет, но она должна возвращать заказ. Помни, что заказ будет словарем с такими ключами, как `'name'` и `'drink'`, но на данный момент функция просто возвращает пустой словарь, который представляет собой пару фигурных скобок. Все идет нормально?»

«Да», — ответил Эрик. — «Значит, ты подразумеваешь, что мы скопируем мои предыдущие функции из предыдущего файла в данный, верно?»

«Верно», — сказал Саймон. — «Иногда, когда программа становится достаточно большой, полезно сгруппировать функции в отдельные файлы. Но в нашем случае проще все хранить в одном файле.

«Функция `print_order()`», — продолжил Саймон, — «просто выводит заказ, полученный из аргумента. В данном случае мы будем использовать стандартную функцию Python `print()`, но позже сделаем все более красиво. Ты уже это сделал, помнишь?»

«Конечно!» — сказал Эрик. — «Думаю, мы сможем сделать это похожим на настоящий чек из кафе».

«Хорошая идея», — сказал Саймон. — «Функция `save_order()` ничего не делает, кроме вывода «Сохранение заказа...». Пока ничего, напомним позже».

«Теперь мы готовы вызвать функцию `main_menu()` и протестировать наш алгоритм. Добавь в конец вызов `main_menu()` и запусти его. Ваша основная программа будет состоять только из этого вызова функции».

Эрик добавил вызов функции, и вся программа теперь выглядит следующим образом:

#### Листинг 6.1 Главное меню с простыми функциями

```
def main_menu():
    while True:
        order = get_order()
        print("Check your order:")
        print_order(order)
        confirm = input("Confirm? Press Y to confirm, N to cancel: ")
```

```
        if confirm == "Y" or confirm == "y":
            save_order(order)
            print("Thanks for your order:")
            print_order(order)
        else:
            continue

def get_order():
    return {}

def print_order(order):
    print(order)
    return

def save_order(order):
    print("Saving order...")
    return

main_menu()
```

### **Ваш ход! Создайте главное меню**

Напишите функцию главного меню, подобную той, которую только что создал Эрик. Не стесняйтесь изменять диалоговые сообщения.

Он нажал Run и увидел следующий результат:

```
Check your order:
{}
Confirm? Press Y to confirm, N to cancel:
```

Он ввел y и получил следующее:

```
Saving order...
Thanks for your order:
{}
Check your order:
{}
Confirm? Press Y to confirm, N to cancel: y
```

«Почему мне снова выдается сообщение ‘Check your order’?» – спросил Эрик.

«Потому что это цикл!» — сказал Саймон. — «После того, как ты ответил y на запрос подтверждения, алгоритм возвращается в начало цикла. А поскольку мы еще не добавили твой диалог меню, он печатает пустой заказ. Все работает так, как ожидалось. Теперь попробуйте ответить n на вопрос».

Эрик ввел n и получил такой результат:

```
Confirm? Press Y to confirm, N to cancel: n
Check your order:
{}
Confirm? Press Y to confirm, N to cancel:
```

«Ты видишь разницу?» — спросил Саймон.

«Я вижу, что на этот раз он не вывел «Сохранение заказа. . .». Это означает, что он следовал короткой стрелкой в правой части твоей диаграммы».



"Отлично!" — Саймон был рад видеть, что Эрик действительно понял схему своего алгоритма.

«Я думаю, что сегодня мы добились хорошего прогресса — наше главное меню работает. Завтра мы напишем настоящие функции, которые будут делать то, что мы хотим. А пока давай подведем итоги и повторим то, что ты узнал сегодня».

«Мы снова использовали цикл `while`!» — сказал Эрик.

"Верно! И ты использовал то, чему научился, работая над меню», — подтвердил Саймон.

«Также ты узнал о словарях. Они похожи на обычные словари, но хранить там можно что угодно, а не только описания слов».

«Да, точно! В нашем простом словаре `order` мы храним имена, напитки и вкусы. Но в более сложном словаре мы можем хранить числа (например, цены) и даже списки и другие словари. Словари в Python действительно полезны, и мы будем использовать их постоянно».

«Кроме того, мне понравилось, как ты создал простые функции только для проверки главного меню», — сказал Эрик. — «Ты сказал, что это называется «сверху вниз», верно?»

«Именно», — ответил Саймон. — «Как ты мог догадаться, существует также подход «снизу вверх». В этом случае сначала создаются функции, правильно тестируются, а затем объединяются в большую программу. В каком-то смысле мы

тоже использовали этот подход, когда создавалась и тестировалась наша первая функция `menu()`. Теперь мы собираемся использовать нашу функцию в нашей большой программе.

«Теперь пора отдохнуть», — продолжил Саймон. — «Завтра мы поработаем над функциями, которые мы добавили в наше главное меню».

### **Новое, что вы узнали сегодня**

- *Подход сверху вниз* — сначала вы создаете «общую картину» вашего приложения и используете простые функции, которые просто что-то выводят, а не выполняют реальную работу. Когда основной алгоритм работает правильно, вы разрабатываете реальные функции.
- *Словарь* — в Python словари могут хранить пары ключей и значений. Вы можете присваивать значения ключам и быстро находить их по этим ключам.
- *Символы блок-схем* — Программисты обычно используют диаграммы для обсуждения своих алгоритмов, прежде чем приступить к написанию собственно кода. Обычно прямоугольник означает какой-то процесс, а ромбовидная форма означает точку принятия решения с вопросом типа «Да/Нет». Также имеются символы для ввода, использования документов, использования баз данных и другие. Мы познакомимся с ними позже.

### **Код для этой главы**

Вы можете найти код этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch06>.

# Создание функций: Получить заказ и вывести его

---

## **В этой главе**

- Эрик создает реальные функции для получения заказов и их вывода
- Эрик использует словарь для хранения и вывода заказов клиентов
- Теперь программа Эрика работает как надо!
- Эрик и Саймон планируют написать функцию для сохранения заказов

«Вчера мы создали главное меню, да?» — начал разговор Саймон. — «Мы даже протестировали функциональность главного меню».

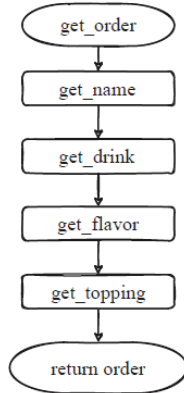
«Да, но это не принесло ничего полезного», — сказал Эрик.

«Верно!» — сказал Саймон. — «Помнишь, мы говорили о подходе «сверху вниз»? Мы создали пустые функции только для проверки главного меню. Теперь пришло время заставить их сделать что-то полезное. Открой файл Python, в котором мы создали меню из файлов. Он называется `menu_files.py`».



Эрик открыл этот файл, и теперь в его редакторе было две вкладки: одна с файлом `main_menu.py`, а другая с файлом `menu_files.py`.

«Переключимся на файл `main_menu.py` и давайте посмотрим, какие функции нам нужно туда добавить», — предложил Саймон. — «Есть три функции: `get_order()`, `print_order()` и `save_order()`. Let's begin with `get_order()`. Где наша блок-схема?»



«Давай начнем с `get_name`. Что мы с ее помощью делаем?» — спросил Саймон.

«Мы просто спрашиваем у пользователя его имя ‘What’s your name?’», — ответил Эрик.

«Верно, а потом?»

«А затем мы сохраняем его в переменной, например `name`».

«Почти верно», — сказал Саймон. — «Помнишь, мы решили, что `order` будет словарем. И мы сохраняем *все*, что связано с этим заказом, в этом словаре. Например, чтобы сохранить имя клиента, вместо `name = 'Erik'`, нам следует написать `order['name'] = 'Erik'`. Только вместо ‘Erik’, мы будем использовать функцию `input()`, как ты это делал в своей первой программе».

«Дай мне попробовать», — сказал Эрик. Он написал следующую функцию:

```
def get_order():
    order['name'] = input("What's your name: ")
    return {}
```

«Теперь попробуй», — сказал Саймон.

Эрик нажал Run. Программа спросила его имя, и он ввел ‘Erik’. Но затем он получил несколько строк сообщений об ошибках:

```
What's your name: Erik
Traceback (most recent call last):
  File "/home/erik/mu_code/main_menu.py", line 41, in <module>
    main_menu()
  File "/home/erik/mu_code/main_menu.py", line 3, in main_menu
    order = get_order()
  File "/home/erik/mu_code/main_menu.py", line 27, in get_order
    order['name'] = input("What's your name: ")
NameError: name 'order' is not defined
>>>
```



"Что это такое?" — спросил он Саймона.

«Послушай, Python сообщит тебе, где возникла проблема. Прочитай последнюю строчку».

«Name 'order' is not defined», — прочитал Эрик..

«Это очень просто», — объяснил Саймон. — «Ты пытался поместить что-то в словарь, но еще не создал его. Это легко исправить. Давай создадим пустой словарь. Напоминаю, для этого мы используем фигурные скобки. Просто напиши `order = {}` перед строкой с `input()`».

Эрик изменил свою функцию следующим образом:

```
def get_order():
    order = {}
    order['name'] = input("What's your name: ")
    return {}
```

Он запустил программу еще раз, и на этот раз она не выдала никаких ошибок:

```
What's your name: Erik
Check your order:
{}
Confirm? Press Y to confirm, N to cancel: y
Saving order...
Thanks for your order:
{}
What's your name:
```

«Сейчас программа стала лучше», — сказал он.

«Да, лучше, но смотри: она все равно выводит пустой заказ. Ты создал заказ и даже ввел свое имя, но твоя функция возвращает пустой словарь. Видишь эту строку: `return {}`?»

«Но ведь это *ты* ее так написал!» — Эрик был уверен, что это не его вина.

«Да, я написал это так, чтобы проверить работу главного меню. Но теперь нам нужно вернуть фактический словарь `order`. Измени его на `return order` и посмотри, выведет ли программа твое имя».

Эрик изменил функцию:

```
def get_order():
    order = {}
    order['name'] = input("What's your name: ")
    return order
```

И он снова запустил программу. На этот раз она вывела его имя:

```
What's your name: Erik
Check your order:
{'name': 'Erik'}
Confirm? Press Y to confirm, N to cancel:
```

«Да, теперь он выводит мое имя!»

«Поздравляю!» — сказал Саймон. — «Теперь ты умеешь работать со словарями».

### **Ваш ход! Создайте функцию `get_order()`**

Начните писать собственную функцию `get_order()` в файле `main_menu.py`. Добавьте первый ввод, чтобы получить имя клиента. Проверьте это, запустив программу главного меню.

### **Каков ваш выбор?**

«Давай продолжим», — предложил Саймон. — «Теперь нам нужно добавить функцию `menu()` для вывода списка напитков и вкусов. Но нам также нужна функция `read_menu()` для чтения твоих меню из файлов. Скопируй и `menu()`, и `read_menu()` из файла `menu_files.py` и вставьте их сюда, в `main_menu.py`. Вставь их прямо перед строкой `def get_order():`».

«Что, если я вставлю их после этой строки?» — Эрик хотел знать, почему его старший брат дал ему такое строгое указание.

«Тогда это не сработает», — просто ответил Саймон и улыбнулся. — «Хорошо, если ты действительно хочешь знать, мы собираемся использовать эти две функции в функции `get_order()`. Сначала нам нужно прочитать из файлов содержимое меню: твои напитки, вкусы и топинги. Затем мы трижды вызываем функцию `menu()`, чтобы получить выбор клиента. И прежде чем мы сможем использовать эти функции, мы должны их *определить*».

«Другими словами, мы должны сообщить Python, что эти функции существуют и что они делают. Вот почему их нужно вставлять именно перед строкой `def get_order():`. Кстати, именно поэтому мы используем слово `def` для запуска функции — мы ее *определяем*».

«Хорошо», — согласился Эрик. Он начал копировать функции. Через пару минут его файл `main_menu.py` выглядел так:

#### **Листинг 7.1 Главное меню готово**

```
def main_menu(): while
    True:
        order = get_order()
        print("Check your order:")
        print_order(order)
        confirm = input("Confirm? Press Y to confirm, N to cancel: ")
        if confirm == "Y" or confirm == "y":
            save_order(order)
            print("Thanks for your order:")
            print_order(order) else:
                continue

def menu(choices, title="Erik's Menu", prompt="Choose your item: "): print(title)
    print(len(title) * "-") i
    = 1
    for c in choices:
        print(i, c)
        i = i + 1
```

```

while True:
    choice = input(prompt)
    allowed_answers = []
    for a in range(1, len(choices) + 1):
        allowed_answers.append(str(a))

    allowed_answers.append("X")
    allowed_answers.append("x")

    if choice in allowed_answers:
        if choice == "X" or choice == "x":
            answer = ""
            break
        else:
            answer = choices[int(choice) - 1]
            break
    else:
        print("Enter number from 1 to ", len(choices))
        answer = ""
    return answer

def read_menu(filename):
    f = open(filename)
    temp = f.readlines()
    result = []
    for item in temp:
        new_item = item.strip()
        result.append(new_item)
    return result

# . . .
# No changes in get_order() and save_order() functions
# . . .
main_menu()

```

«Правильно!» — сказал Саймон. — «Теперь ты знаешь правило: что-либо следует определять прежде, чем использовать это. Ты уже видел ошибки, которые возникали, когда ты не определил словарь `order`».

«Теперь мы готовы использовать эти функции в `get_order`», — продолжил он. — «Посмотри на свою программу `menu_files.py`. Что мы сделали в первую очередь?»

«Мы прочитали меню из файлов», — ответил Эрик.

"Хорошо. Давай сделаем это здесь, но внутри функции».

Эрик добавил в функцию `get_order()` три строки:

```

def get_order():
    order = {}
    order["name"] = input("What's your name: ")
    drinks = read_menu("drinks.txt")
    flavors = read_menu("flavors.txt")
    toppings = read_menu("toppings.txt")
    return order

```

Ему пришлось добавить четыре пробела перед каждой строкой, чтобы все они были на одном уровне. «А теперь, то же самое с тремя функциями `menu()`?» —

спросил он Саймона.

«Конечно, давай!»

Эрик модифицировал свою функцию следующим образом:

```
def get_order():
    order = {}
    order["name"] = input("What's your name: ")
    drinks = read_menu("drinks.txt")
    flavors = read_menu("flavors.txt")
    toppings = read_menu("toppings.txt")
    drink = menu(drinks, "Erik's drinks", "Choose your drink: ")
    flavor = menu(flavors, "Erik's flavors", "Choose your flavor: ")
    topping = menu(toppings, "Erik's toppings", "Choose your topping: ")
    return order
```

Он гордился своей работой и взглянул на Саймона.

«Почти верно», — сказал Саймон. — «Ты все правильно скопировал, но тебе следует немного изменить код, чтобы хранить ответы в словаре `order`. Это изменение — ты знаешь, как это сделать».

«А, понятно», — сказал Эрик. Он изменил функцию. Теперь она выглядела так:

#### Листинг 7.2 main\_menu.py

```
def get_order():
    order = {}
    order["name"] = input("What's your name: ")
    drinks = read_menu("drinks.txt")
    flavors = read_menu("flavors.txt")
    toppings = read_menu("toppings.txt")
    order["drink"] = menu(drinks, "Erik's drinks",
                          "Choose your drink: ")
    order["flavor"] = menu(flavors, "Erik's flavors",
                           "Choose your flavor: ")
    order["topping"] = menu(toppings, "Erik's toppings",
                             "Choose your topping: ")
    return order
```

Саймон подбадривал брата: «Давай, запусти ее!»

Эрик запустил свою программу.

```
What's your name: Erik
Erik's drinks
-----
1 coffee
2 chocolate
3 decaf
Choose your drink: 1
Erik's flavors
-----
1 caramel
2 vanilla
3 peppermint
```

```

4 raspberry
5 plain
Choose your flavor: 2
    Erik's toppings
-----
1 chocolate
2 cinnamon
3 caramel
4 vanilla powder
Choose your topping: 3
    Check your order:
{'name': 'Erik', 'drink': 'coffee', 'flavor': 'vanilla',
'topping': 'caramel'}
Confirm? Press Y to confirm, N to cancel: y
    Saving order... Thanks
for your order:
{'name': 'Erik', 'drink': 'coffee', 'flavor': 'vanilla',
'topping': 'caramel'}
What's your name:

```

«Ух ты!» — Он был действительно счастлив. — «Я написал программу более чем из 70 строк, и она работает!»

«Да, ты это сделал — программа действительно работает», — подтвердил Саймон и улыбнулся.

### ***Ваш ход! Добавьте пункты меню в вашу программу***

Добавьте функции `menu()` и `read_menu()`, как в предыдущей программе, в ваш файл `main_menu.py`. Проверьте программу, запустив ее и введя свой выбор. Попробуйте ввести неправильный выбор и убедитесь, что функция `menu()` не позволяет вам это сделать.

### ***Распечатайте это!***

«Но чего-то все еще не хватает. Заказ выглядит не очень профессионально. Это не как в настоящей кофейне», — продолжил Саймон.

"Я понимаю. Нам следует изменить функцию `print_order()`, верно?" — предположил Эрик.

"Да, верно. Перейдите к функции `print_order()` в своем файле `main_menu.py`",  
Функция `print_order()` Эрика выглядела так:

```

def print_order(order):
    print(order)
    return

```

«Здесь мы используем функцию печати по умолчанию, предоставляемую Python», — продолжил Саймон. «Python *может* вывести твой словарь, но это будет некрасиво. Для отладки это нормально, но вот для реальных заказов и чеков придется все организовать более красиво. Ты уже это сделал, верно?»

«Ты имеешь в виду, когда я выводил линии из тире? Да, это было более красиво, чем этот обычный вывод».

«Давай сделаем что-то похожее на то, что ты сделал в конце файла

menu\_files.py. Ты можешь скопировать оттуда эти строки, начиная с `print`. Просто не забудь сохранить правильный отступ и убедись, что ты используешь словарь, а не простые переменные. И не забывай, что теперь у нас есть имя клиента. Я думаю, тебе следует использовать его в своей функции. Готов?»

«Да», — ответил Эрик. Он начал работать над этой функцией. В итоге у него получилось следующее:

```
def print_order(order):
    print("Here is your order, ", order["name"])
    print("Main product: ", order["drink"])
    print("Flavor: ", order["flavor"])
    print("Topping: ", order["topping"])
    print("Thanks for your order!")
    return
```

### **Ваш ход!** Добавьте функцию `print_order()`

Добавьте функцию `print_order()` в файл `main_menu.py`. Не стесняйтесь использовать такие украшения, как тире (`-`), подчеркивание (`_`) или знаки равенства (`=`), чтобы ваш выведенный на печать заказ выглядел достойно. Попробуйте найти чеки из ресторанов, кафе или магазинов мороженого и посмотрите, сможете ли вы сделать свои чеки похожими.

Он снова запустил программу и получил гораздо более красивый результат:

```
Here is your order, Erik
Main product: coffee
Flavor: vanilla
Topping: caramel
Thanks for your order!
Confirm? Press Y to confirm, N to cancel:
```

«Да, так намного лучше!» — сказал Саймон. — «По своему усмотрению ты можешь добавить такие украшения, как тире и вертикальные линии. Но как программист ты проделал отличную работу. Ты написал несколько очень полезных функций, правильно их организовал и протестировал. Хорошая работа, Эрик. Я действительно горжусь тобой!»

«Кроме того, мы узнали о словарях, и я ими уже пользовался», — Эрик почувствовал, что пришло время подводить итоги, и ему следует упомянуть все, что он узнал и использовал за день.

«Да, верно», — подтвердил Саймон. — «Словари очень важны в Python. Мы постоянно используем их в наших программах. Позже ты узнаешь о них больше».

«Ты сказал, что нам также следует каким-то образом сохранить наши заказы. Сделаем ли мы это завтра?» — спросил Эрик.

«Да, конечно», — сказал Саймон. — «Ты знаешь о JSON?»

«Джейсона? Да, мы вместе ходим на математику. А что такое?»

«Нет, это не тот Джейсон», — засмеялся Саймон. — «JSON — это формат файла, который мы можем использовать для сохранения заказов твоей кофейни. Поговорим об этом завтра, ладно?»

«Хорошо», — сказал Эрик и вышел.

## **Новое, что вы узнали сегодня**

- *Определение переменных и функций* — в Python нам необходимо *определить* переменные и функции, прежде чем мы сможем их использовать. Для переменных это так же просто, как присвоение переменной пустого значения. Для словаря это `order = {}`, а для строки `name = ""`. Функции должны быть определены с использованием ключевого слова `def`.

## **Код для этой главы**

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch07>.

# Работаем с JSON: Сохраняем заказ

## В этой главе

- Эрик узнает о формате и файлах JSON
- Эрик узнает о модулях Python
- Эрик создает список заказов
- Саймон и Эрик пишут функцию для сохранения заказов в файл JSON

«Вчера ты сказал что-то о Джейсоне», — напомнил Эрик Саймону. — «Но ты сказал, что это другой Джейсон, не тот, что ходит со мной на математику».

«Да, это другой Джейсон», — улыбнулся Саймон. — «Это JSON, J-S-O-N, формат файла, который мы используем для хранения данных».

«Как файлы, которые мы использовали для хранения меню?»

«Да, похоже», — ответил Саймон. — «Но этот формат очень хорош для хранения *структурированных* данных».

"Что это такое?" — спросил Эрик.

«Иногда нам нужно сохранить только фрагмент текста или изображения. Обычно они не имеют какой-либо фиксированной структуры. Текст — это просто текст. Причем изображение может быть большим или маленьким, оно может быть черно-белым или цветным, но у него нет никакой структуры — это просто набор пикселей. Это то, что мы называем *неструктурированными* данными».

«Но в твоем случае каждый заказ имеет структуру. В каждом заказе указывается имя клиента и все компоненты напитка, который ты собираешься приготовить —



ни больше, ни меньше. В нем всегда есть основной напиток, вкус и топинг. Поскольку ты написал меню, клиент должен ответить на все его вопросы, прежде чем ты сможешь вывести или сохранить заказ. Кроме того, клиент не может ничего *добавить* к своему заказу».

«Например, другой топинг?» — спросил Эрик.

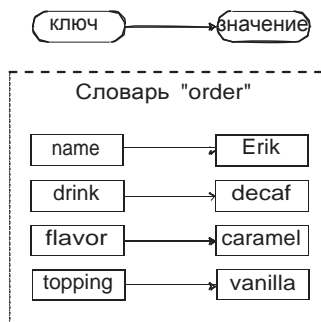
«Да», — сказал Саймон. — «Твой заказ — это пример структурированных данных».

Эрик не думал, что создал что-то с таким серьезным названием.

«Твой заказ хранится в словаре, — продолжил Саймон, — и ты точно знаешь, что для каждого заказа есть словарные ключи (помнишь, что это такое?), называемые `name`, `drink`, `flavor` и `topping`».

Саймон вытащил одну из своих диаграмм.

Словари в Python:



«Формат JSON создан для хранения такого рода структурированных данных. Давай немного потренируемся. Как и раньше, сначала мы создадим простую программу и попробуем выполнить несколько простых операций. Затем мы возьмем то, чему научились в этой простой программе, и воспользуемся этим, чтобы наша основная программа сохраняла заказы в файл. Это много для одного дня, так что, возможно, мы перенесем что-нибудь завтра».

Саймон продолжил: «Теперь открой редактор и создай новый файл. Сохрани его, например, под именем `dict_json.py`».

Эрик открыл окно редактора и нажал New. Затем он нажал Save, ввел `dict_json.py` и снова нажал Save. Он уже был знаком с этой процедурой.

«Теперь», — сказал Саймон, — «создай пример заказа».

«Что такое пример заказа?»

«Твой заказ — это словарь, верно?» — начал объяснять Саймон. — «В своей основной программе ты создал пустой словарь, а затем начал заполнять его значениями, полученными от клиента. В данный момент мы хотим пропустить этот шаг и представить, что наш словарь `order` уже заполнен вариантами выбора клиента. Позволь мне начать вместо тебя», — сказал Саймон.

Он напечатал в редакторе Эрика:

```
order = {  
    "name": "Erik",
```

«Ты можешь продолжать», — сказал он. — «Не забудьте закрыть фигурные скобки».

Эрик закончил словарь `order` и закрыл фигурные скобки. Теперь все выглядело так:

```
order = {  
    "name": "Erik",  
    "drink": "coffee",  
    "flavor": "caramel",  
    "topping": "chocolate"  
}
```

«Я заметил, что ты сделал отступы в строках этого словаря», — сказал он брату. — «Это правило для словарей Python?»

«Нет», — ответил Саймон. — «В данном случае я сделал это просто для того, чтобы код выглядел лучше. И чтобы сделать его более *читабельным*. Я мог бы объединить все элементы в одну строку или начать с начала строки, но мне кажется, так будет лучше».

«Теперь», — продолжил он, — «у нас есть словарь. И мы хотим сохранить его в файле. Думаю, мне стоит напомнить тебе о файловых операциях со словарями.

«Да», — ответил Эрик. — «Это было *так* давно. Я мало что помню».

«Конечно», — сказал Саймон. — «Кроме того, ты узнаешь кое-что новое о файлах. Сначала нам нужно *открыть* файл. Чтобы открыть файл, нам нужно вызвать функцию `open()` — разумеется — и передать имя файла в качестве аргумента. Ты ведь знаешь все о функциях и их аргументах, верно?

«Функция `open()` возвращает *дескриптор* файла. Это специальный объект, который наша программа может использовать для работы с этим файлом».

Саймон написал еще одну строку под словарем, который создал Эрик:

```
order = {  
    "name": "Erik",  
    "drink": "coffee",  
    "flavor": "caramel",  
    "topping": "chocolate"  
}  
  
f = open("orders.json", "w")
```

«Вот и первая новинка. Видишь эту букву в последней строке? Это означает, что мы собираемся произвести запись в файл».

«Но когда мы открывали файлы меню, мы не использовали никаких букв», — напомнил Эрик.

"Ты прав!" — ответил Саймон. — «Мы не использовали никаких букв — кстати, они называются *режимами* — потому что, когда мы не используем никаких букв,

Python открывает файлы для чтения. На этот раз мы хотим выполнить запись в этот файл, поэтому нам нужно уведомить об этом Python».

«И я вижу, что ты назвал файл `'orders.json'`. Это потому, что ты хочешь использовать тот самый формат JSON, о котором говорили?»

«Да, именно так», — ответил Саймон. — «Это не обязательно, но принято добавлять расширение `.json` к файлам JSON. Еще одно отличие режима записи `'write'` заключается в том, что Python создаст файл с таким именем, если его еще не существует».

«Что дальше?» — спросил Эрик. — «Как нам произвести запись в этот файл JSON? В прошлый раз мы использовали «методы» — они ведь так называются?»

«Да, ты правильно помнишь», — сказал Саймон. — «Но на этот раз мы поступим по-другому. Это все потому, что мы собираемся записывать структурированные данные, а не просто текст. Мы собираемся использовать модуль Python под названием `json`».

«Что такое модуль?» — тут же спросил Эрик.

«Я объясню тебе это прямо сейчас», — сказал Саймон. Он улыбнулся. — «Помнишь, совсем недавно ты написал несколько функций на Python. Например ту, которая читает пункты меню из файла и возвращает список. Представь, что один из твоих друзей захотел написать собственную программу для кофейни или чего-то подобного».

«Да», — сказал Эрик. — «Недавно я разговаривал с Эмили, и она сказала, что хочет создать программу для кафе-мороженого».

"Великолепно!" — воскликнул Саймон. — «Возможно, ты захочешь помочь ей и поделиться функциями, которые ты написал. Это сэкономит ей время, так как ее программа будет закончена раньше. Среди программистов принято делиться своей работой, чтобы помочь друг другу. В Python ты можешь сгруппировать функции, которыми хочешь поделиться, в файле и передать его Эмили. Она сможет скопировать этот файл на свой компьютер, а затем *импортировать* его в свою программу. После импорта она сможет использовать твои функции в своем приложении. Этот твой файл с функциями и будет называться *модулем*».

«Что, если я не хочу делиться своими функциями?» — спросил Эрик. — «Я потратил на их написание несколько дней!»

«Да, это так», — сказал Саймон. — «И ты проделал огромную работу. Но помни, что многие люди потратили много дней на написание других функций на Python и даже самого Python. Они поделились своей работой с другими программистами, поэтому ты можешь использовать Python и его функции совершенно бесплатно. Таким образом, мы помогаем друг другу работать над нашими проектами. Было бы гораздо тяжелее, если бы нам с тобой пришлось писать все самим с нуля. Вот почему люди используют чужой код и делятся своим кодом с другими. Обычно такую группу людей называют *сообществом открытого исходного кода*».

«Вернемся к JSON», — продолжил Саймон. — «Мы собираемся использовать модуль `json`, написанный другими людьми. Этот модуль может читать словари Python и преобразовывать их в файлы JSON. Перейди в начало файла и добавь

строку: `import json`. Это должна быть самая первая строка файла».

Эрик сделал это. Вот его обновленный файл:

```
import json

order = {
    "name": "Erik",
    "drink": "coffee",
    "flavor": "caramel",
    "topping": "chocolate"
}

f = open("orders.json", "w")
```

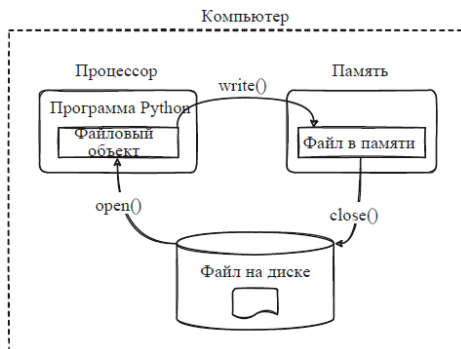
«Теперь нам нужно преобразовать твой пример заказа в JSON и записать его в файл, который мы только что открыли», — сказал Саймон. — «В модуле `json` эта функция называется `dump`. Мы вызовем ее в твоей программе, но нам нужно сообщить Python, что он должен искать эту функцию в модуле `json`, поэтому мы вызываем ее так: `json.dump()`. Тебе просто нужно передать два аргумента: словарь и файловый объект. Добавь эту функцию в конец файла. Твой словарь - `order`, твой файловый объект - `f`».

Эрик добавил эту строку в конец программы:

```
json.dump(order, f)
```

Саймон продолжил: «Есть еще одна вещь, которую мы раньше не делали с файлами. Нам следует *закрывать* файл. Это важно, поэтому давайте рассмотрим другую диаграмму».

«Вот три основных компонента компьютера: процессор, память и диск. Твоя программа Python обрабатывается процессором. Твой файл хранится на диске в файловой системе. Файловая система — это то, что вы видите в Finder на Mac. В Проводнике Windows это папки и файлы. Когда ты хочешь работать с файлом в Python, ты *открываешь* его, как ты это уже делал. Это создает файловый объект в твоей программе. Когда ты производишь запись в файл, ты осуществляешь запись в память компьютера. Затем, когда ты хочешь, чтобы ваш файл *действительно* был записан в файловой системе на диске, ты *закрываешь* его».



«Это так сложно!» — Эрик был в замешательстве. — «Почему бы нам не произвести

запись прямо на диск?»

«Компьютеры *сложны*, ты прав!» — согласился Саймон. — «Причина в том, что компьютерные инженеры пытаются заставить компьютеры работать быстрее. Запись на диск происходит медленно — намного медленнее, чем запись в память. Представь, что ты пишешь программу в текстовом редакторе. Если бы он сразу сохранял каждую нажатую тобой букву на диске, это было бы *очень* медленно. Тебе же не нравится работать на медленных компьютерах, не так ли?»

«Чтобы компьютеры работали быстрее», — продолжил Саймон, — «инженеры решили хранить данные в памяти и сохранять их на диск только при необходимости. Есть много уловок, которые используют, чтобы заставить компьютеры работать быстрее, и, конечно, я знаю не все из них. Возможно, я узнаю о них больше в колледже».

«Давай вернемся к нашей программе и закроем файл. Помни, что файловые объекты используют методы. В этом случае мы осуществляем вызов так `f.close()`. Это обеспечит запись нашего заказа в файл. Добавь его в твою программу после последней строки».

Это было несложно после такого долгого объяснения. Эрик быстро добавил строку и получил следующее:

```
import json

order = {
    "name": "Erik",
    "drink": "coffee",
    "flavor": "caramel",
    "topping": "chocolate"
}

f = open("orders.json", "w")
json.dump(order, f) f.close()
```

«Теперь запустим ее», — сказал Саймон.

Эрик нажал Run и увидел знакомое `>>>` внизу окна.

«Что теперь?» — спросил он Саймона.

«Ничего не произошло?» — Саймон улыбнулся. Он почувствовал замешательство Эрика. — «Это потому, что ты не сказал Python печатать что-либо. Но все же за кулисами кое-что произошло. Python открыл файл `orders.json`, записал в него твой заказ и закрыл его. Теперь нам нужно открыть его и проверить, правильно ли он записан. Чтобы открыть файл, используй простой текстовый редактор. Ты используешь Mac, поэтому это будет TextEdit из папки «Applications». В Windows это Блокнот, а в Linux — gedit или Kate. Запусти редактор и открой файл. Он находится в твоей домашней папке, в разделе `mu_code` и называется `orders.json`».

Эрик запустил TextEdit, нашел файл и открыл его. Действительно, он увидел свой заказ.



```
orders.json
{"name": "Erik", "drink": "coffee", "flavor": "caramel", "topping": "chocolate"}
```

### **Ваш ход! Сохраните пример своего заказа в файле JSON**

Напишите программу, которую только что написал Эрик. Попробуйте создать немного другой пример заказа. Запустите программу и проверьте полученный файл JSON с помощью текстового редактора. Попробуйте создать другой заказ и снова запустите программу. Ваш файл JSON изменился? (Возможно, вам придется переоткрыть файл в текстовом редакторе.)

«Видишь?» — спросил Саймон. — «Это твой пример заказа, хранящийся в файле. Позволь мне добавить кое-что, и ты поймешь, почему файлы JSON идеально подходят для хранения словарей Python».

Саймон взял клавиатуру и изменил вызов `json.dump()` следующим образом:

```
json.dump(order, f, indent=4)
```

Он снова запустил программу и снова открыл файл `orders.json`. Эрику это показалось знакомым.



```
orders.json
{
    "name": "Erik",
    "drink": "coffee",
    "flavor": "caramel",
    "topping": "chocolate"
}
```

### **Ваш ход! Сделайте это красиво**

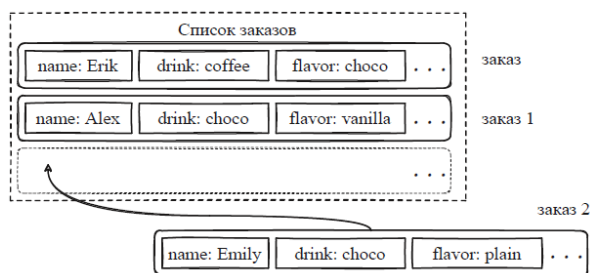
Добавьте аргумент `indent=4` в предыдущую программу, запустите программу еще раз и проверьте, изменился ли ваш файл JSON.

«Он выглядит точно так же, как мой словарь!» — воскликнул Эрик.

«Я говорил тебе», — сказал Саймон. — «Мы будем использовать JSON для хранения твоих заказов. Я сказал «заказов», а это значит, что теперь нам нужно научиться хранить несколько заказов и хранить их в файле. Мы уже знаем, как Python хранит несколько элементов по порядку — ты использовал это для своих меню».

«Список!» — уточнил Эрик.

«Верно! Список в Python может содержать разные вещи: строки, числа и даже словари. В данном случае у нас будет список словарей. Каждый словарь будет содержать заказ, и мы будем добавлять их в список по одному — новый клиент, новый заказ, новый словарь. Я нарисую схему».



«Давайте создадим список заказов», — продолжил Саймон. — «Скопирую существующий заказ в код и назови его, например, `order1`».

«Затем измени содержание заказа: название, напиток и другие пункты».

Эрик некоторое время работал над своим кодом и создал дополнительный заказ, чуть ниже первого:

```
order1 = {
    "name": "Alex",
    "drink": "choco",
    "flavor": "vanilla",
    "topping": "caramel"
}
```

«Хорошо», — сказал Саймон. — «Теперь создай пустой список под названием `orders`. Обрати внимание, что это множественное число — `orders`. Это очень похоже на создание пустого словаря — ты уже это сделал — только вместо фигурных скобок используй квадратные».

Эрик добавил следующую строку под вторым заказом:

```
orders = []
```

«Теперь мы добавим в список оба заказа», — сказал Саймон. — «Хочешь верь, хочешь нет, но список `orders`, который ты только что создал, тоже является объектом. В Python все является объектом, и у каждого объекта есть методы, которые ты можешь использовать. Тебе просто нужно знать, какие методы существуют для каждого объекта. Например, для всех списков существует метод `append()`. Он добавляет элемент, который ты передаешь в качестве аргумента, в конец списка».

«Смотри, я использую его, чтобы добавить `order` и `order1` в список `orders`», — Саймон добавил эти две строки ниже строки, в которой был создан список `orders`:

```
orders.append(order)
orders.append(order1)
```

«Но откуда ты знаешь, что здесь следует использовать функцию `append()`?» — спросил Эрик.

«Я прочитал это в документации Python, когда изучал Python», — ответил Саймон с улыбкой. — «Ты можешь использовать Google для поиска «методов

списков Python», и ты найдешь все, что тебе нужно знать».

«Теперь мы можем попытаться сохранить этот список в формате JSON», — продолжил Саймон. — «Единственное, что нам нужно изменить, это то, *что* мы хотим «сбросить» в файл. Мы использовали `order` в качестве аргумента функции `dump()` для записи одного заказа. Теперь давайте заменим его `orders` (во множественном числе!) и посмотрим, что изменится».

Эрик изменил строку с помощью `json.dump()` на следующую:

```
json.dump(orders, f, indent=4)
```

Он запустил программу и снова открыл файл `orders.json`.



```
[
  {
    "name": "Erik",
    "drink": "coffee",
    "flavor": "caramel",
    "topping": "chocolate"
  },
  {
    "name": "Alex",
    "drink": "chocolate",
    "flavor": "vanilla",
    "topping": "caramel"
  }
]
```

### Ваш ход! Сохраните список

Добавьте еще один пример заказа и назовите его `order2`. Создайте список заказов и сохраните его в том же файле JSON. Проверьте результат с помощью текстового редактора. Добавьте столько заказов, сколько сможете, и запишите их в файл. Есть ли какой-либо предел количества заказов, которые вы можете сохранить?

«Так что ты скажешь?» — спросил Саймон. — «Это похоже на твой заказ?»

«Да, это в точности как на Python!» — сказал Эрик. — «Но почему мы записали мои заказы в отдельный файл? Если это похоже на Python, почему бы нам не записать мои заказы в нашу программу на Python?»

«Отличный вопрос!» — Саймон был очень рад, что Эрик захотел разобраться. — «Прежде всего, мы всегда хотим отделить программы от данных. Помнишь: когда ты запускаешь приложение Word, ты не записываешь свои документы в программу Word. Ты сохраняешь их в отдельных файлах. Это именно то, что мы здесь и делаем. Твоя программа может сохранять заказы в разных файлах, например, соответствующих разным дням. Все, что тебе нужно сделать, это изменить имя выходного файла, например, `orders.Monday.json`, `orders.Tuesday.json` и так далее».

«Вторая причина», — продолжил Саймон, — «заключается в том, что этот формат не просто так называется JSON. Это означает JavaScript Object Notation



или нотацию объектов JavaScript. Его изобрели люди, использовавшие язык программирования JavaScript, а затем его начали использовать и другие языки. Таким образом, ты можешь использовать Python для записи своих заказов в файл JSON, а затем некоторые из твоих друзей могут создать другую программу на JavaScript, которая будет читать данные из этого файла и распечатывать твои заказы, например, на веб-странице».

«Да, я слышал, как некоторые ребята в моем классе говорили, что знают JavaScript», — сказал Эрик.

«Хорошо! Возможно, ты захочешь создать команду программистов и вместе с ними работать над приложениями», — сказал Саймон.

«Но давай продолжим работать с нашей программой. Теперь мы читаем заказы из файла JSON и сохраняем их в новом списке. Назовем его `saved_orders`».

«Зачем мы их читаем, если мы просто записали их в файл?» — Эрик был в замешательстве.

«Может быть, я не так объяснил», — ответил Саймон. — «В этой программе мы отрабатываем некоторые операции с файлами JSON, поэтому хорошо их знаем и можем использовать в нашей основной программе. Программисты делают это часто: они создают простые программы для проверки концепций и идей. Позволь мне показать тебе мой план нашей основной программы, чтобы ты лучше понял, куда мы идем».

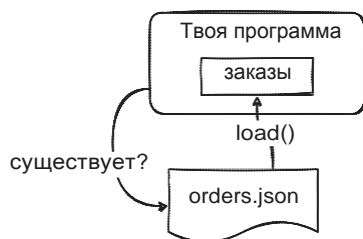
Саймон взял еще один лист бумаги и начал рисовать.

«Сначала мы проверяем, существует ли файл с именем `orders.json`. Если он существует, мы открываем его и читаем из него наши предыдущие заказы».

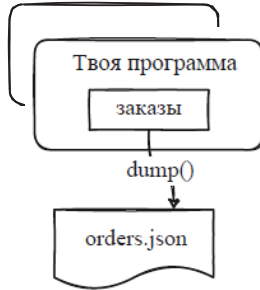
«Зачем нам наши предыдущие заказы? Мы их уже подготовили», — спросил Эрик. — «Да, но помни: мы можем захотеть подсчитать, сколько клиентов мы обслужили сегодня, вчера или в прошлом месяце. Или подсчитать, сколько порций карамели мы использовали, и решить, не пора ли пополнить ее запасы. Тебе нужны все заказы, если ты хочешь серьезно управлять своим бизнесом в кофейне. Вот почему все предприятия хранят такие записи в течение длительного времени».

«А что, если у нас нет этого файла?» — спросил Эрик.

«Это означает, что мы только что открыли свой бизнес и начали работать», — сказал Саймон. — «В таком случае мы создадим пустой список и начнем принимать заказы. Файл будет создан автоматически, когда мы откроем его для записи».



«Посмотри на эту диаграмму: здесь у нас есть список `orders`, либо заполненный заказами за предыдущие дни, либо пустой. Когда мы начинаем получать заказы, мы сохраняем их в этом списке. Закончив день, мы закрываем файл, и все наши заказы сохраняются на диске. На следующий день мы снова открываем файл и продолжаем принимать заказы. Все новые заказы будут добавлены к заказам предыдущего дня».



### **Ваш ход! Нарисуйте свои собственные диаграммы**

Попробуйте нарисовать диаграммы для своей программы, не заглядывая в книгу. Рисование диаграмм поможет вам понять, как работают программы.

«Так работают настоящие кофейни? Типа Старбакс?» – спросил Эрик.

«Да, почти», — сказал Саймон. — «Конечно, они используют базу данных для надежности и безопасности. Их записи заказов также более сложны по сравнению с нашими. Но весь процесс очень похож».

«Теперь, когда ты знаешь главный план, давай продолжим нашу простую программу и прочитаем данные из файла. Мы прочитаем предыдущие заказы в новый список с наименованием `saved_orders`, а затем распечатаем его, чтобы проверить, правильно ли мы его прочитали. Для этого в модуле `json` есть функция `load()`. Она работает так же, как и `dump()`: сначала мы открываем файл, но на этот раз для чтения, а не для записи. Затем мы вызываем `json.load()` и передаем объект файла в качестве аргумента. Функция *возвращает* объект, который она прочитала из файла, и мы присваиваем этот объект переменной. В нашем случае это будет список заказов, которые, как ты помнишь, являются словарями».

«Звучит сложно? Давай я тебе помогу. В Python это выглядит намного короче». — Саймон начал добавлять строки в код Эрика. Вот что он добавил в конце программы:

```
f = open("orders.json", "r")
saved_orders = json.load(f)
print(saved_orders)
```

Он нажал Run и Эрик увидел результат.

```
Running: dict_json.py
```

```
[{'name': 'Erik', 'drink': 'coffee', 'flavor': 'caramel', 'topping': 'chocolate'}, {'name': 'Alex', 'drink': 'chocolate', 'flavor': 'vanilla', 'topping': 'caramel'}]  
>>> |
```

### **Ваш ход! Прочитайте данные из файла JSON**

Добавьте предыдущие строки в свою программу и попробуйте прочитать созданный вами файл JSON. Получаете ли вы те же заказы, что и в вашем примере?

«Сегодня ты многому научился», — сказал Саймон. — «Давай сделаем перерыв до завтра. Затем мы добавим эти функции в нашу основную программу, и она станет настоящим приложением-кофейней. Можешь ли ты кратко повторить, что мы сделали сегодня?»

«Мы создали файл JSON из моего словаря Python, и он очень похож на Python. Затем ты объяснил все о файлах, памяти и дисках. Мы также создали список словарей и тоже сохранили его в файле».

«И мы узнали о модулях Python и о том, как их импортировать», — добавил Саймон.

«Пока все хорошо», — подытожил он. — «Продолжим завтра. Мы очень близки к завершению первой версии твоего приложения».

### **Новое, что вы узнали сегодня**

- *JSON (JavaScript Object Notation)* — формат, который используется для хранения структурированных данных и может использоваться для обмена информацией между программами.
- *Модули Python* — группы функций Python, которые могут использоваться другими программистами. Обычно они сгруппированы в файлы. Вам необходимо импортировать модули перед их использованием.
- *Списки словарей* — списки могут содержать разные типы: строки, числа, словари и даже другие списки.
- *Операции с файлами* — файлы можно открывать для чтения и записи. Записывать данные в файлы можно, но изначально они записываются в память компьютера. Вам следует закрыть файлы, чтобы сохранить данные на диске компьютера.
- *Сообщество открытого исходного кода* — люди, которые делятся программами, которые они пишут, и помогают друг другу писать оптимальный код.

***Код для этой главы***

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch08>.

## Дополните меню: Настоящая программа

### **В этой главе**

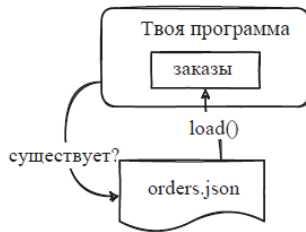
- Эрик и Саймон создают функции загрузки и сохранения в основной программе
- Саймон добавляет функции выхода в главное меню и функцию `get_order()`
- Саймон объясняет, почему, по его мнению, Эрик только что создал настоящую программу
- Братья обсуждают планы на будущее

«А теперь давай посмотрим правде в глаза», — сказал Саймон. — «Вчера мы поиграли с учебными заказами и простыми программами. Сегодня пришло время использовать то, что ты узнал, в твоей реальной программе».

«Да, давай сделаем это!» — ответил Эрик.

«Открой файл `main_menu.py`, в котором ты написал главное меню. Нам следует добавить туда пару функций для работы с JSON-файлом. Давай вспомним, что нам нужно сделать в первую очередь». — Саймон вытащил свою вчерашнюю диаграмму.





«Нам нужно написать функцию, которая будет загружать список заказов из файла JSON. Но сначала она должна проверить, существует ли этот файл. Если файл не существует, мы создадим пустой список и вернем его из этой функции. Если файл существует, мы прочитаем его, преобразуем файл JSON в список Python и вернем этот список».

«Давай я тебе помогу», — Саймон чувствовал, что Эрика это может немного сбить с толку. — «Обычно в функциях, работающих с файлами, в качестве аргумента мы передаем имя файла».

Саймон начал функцию в конце файла, прямо перед последней строкой с `main_menu()`:

```
def load_orders(filename):
```

«Теперь нам нужно проверить, существует ли файл. Для этого есть специальная функция, и мы можем найти ее в модуле `os`».

«Что такое `os`?» — спросил Эрик.

«OS означает *operating system* или операционная система. Операционная система компьютера управляет всеми файлами и программами. Она работает с экраном, клавиатурой, динамиками и видеокамерой. На типичном компьютере может быть установлена ОС Windows, macOS или Linux».

«Мы собираемся спросить операционную систему, существует ли файл с таким именем на этом компьютере». — Саймон добавил строчку:

```
def load_orders(filename):
    if os.path.exists(filename):
```

«Посмотри, здесь мы использовали модуль `os`. Это означает, что нам придется импортировать его так же, как мы вчера импортировали модуль `json` в твой пример программы. В этой программе мы еще не импортировали модули `json` или `os`, поэтому давайте импортируем и тот, и другой».

Саймон переместил курсор в самое начало файла и добавил две строки:

```
import os
import json
```

Он вернул курсор в функцию `load_orders()` в конце файла и продолжил объяснение: «Если файл существует, мы открываем его для чтения. Затем мы используем функцию `json.load()` для чтения из файла списка `orders` и возврата списка».

Он добавил в функцию три строки:

```
def load_orders(filename):
    if os.path.exists(filename):
        f = open(filename, "r")
        orders = json.load(f)
        return orders
```

«Если его не существует, мы просто создаем пустой список и возвращаем его:»

```
def load_orders(filename):
    if os.path.exists(filename):
        f = open(filename, "r")
        orders = json.load(f)
        return orders
    else:
        orders = []
        return orders
```

«Теперь функция загрузки готова!» — сказал он и посмотрел на Эрика.

«Я не думаю, что смог бы написать это сам», — сказал Эрик.

«Конечно, это выглядит сложно, когда делаешь это в первый раз. Но посмотри, ты можешь прочитать этот код, как если бы это был простой текст на английском языке, не так ли?»

Эрик еще раз посмотрел на функцию и попытался прочитать ее: «Если файл filename существует, открываю его. Сохраняю его в объекте с именем f. Затем загружаю предыдущие заказы из этого файла f в orders. Хммм, да, я могу это прочитать. Он был удивлен».

Теперь он мог читать Python и понимать его!

«Следующая функция проще», — продолжил Саймон. — «Думаю, ты сможешь написать ее сам, если согласишься файл, где ты тренировался с образцами заказов. Посмотри, сразу после двух операций append(), нам нужны три строки. У нас уже есть функция save\_order(), которая ничего не делает, кроме вывода 'Saving order . . .'. Давай заменим ее реальной функцией. Я думаю, его следует называть save\_orders() — во множественном числе, потому что теперь мы знаем, как сохранить список заказов в файле JSON, верно?»

Саймон написал начало функции:

```
def save_orders(orders, filename):
```

Он объяснил: «Мы передаем список заказов в качестве первого аргумента. Затем мы передаем имя файла, в котором хотим его сохранить. Теперь ты можешь добавить те три строчки из вчерашней программы».

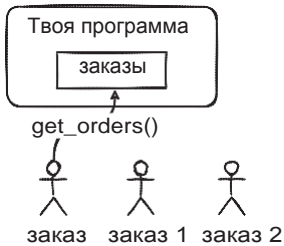
Эрик просмотрел файл dict\_json.py и скопировал из него три строки. Теперь функция стала выглядеть так:



```
def save_orders(orders, filename):
    f = open(filename, "w")
    json.dump(orders, f, indent=4)
    return
```

«Можем ли мы проверить ее сейчас?» — спросил он Саймона.

«Мы почти готовы к этому», — ответил Саймон. — «Посмотрите внизу нашего файла — мы просто вызываем функцию `main_menu()`. Вот что нам нужно сделать в этой функции», — и он быстро нарисовал картинку.



«Это то, что мы делаем в функции `main_menu()`. Нам просто нужно его немного отредактировать, чтобы обслуживать нескольких клиентов и сохранять их заказы в списке `orders`. Нам нужно передать этот список в функцию `main_menu()`, где есть место для сохранения заказов по мере их ввода».

Он переместил курсор в начало файла и добавил аргумент `orders` в определение `main_menu()`:

```
def main_menu(orders):
```

«Теперь», — продолжил он, — «каждый раз, когда клиент вводит новый заказ, он добавляется в список `orders`. До того, как мы добавили список в качестве аргумента, `main_menu()` не знала, куда добавить новый заказ. Теперь мы можем использовать метод `append()` и добавить его в `orders` сразу после того, как клиент подтвердит заказ».

«Мы не будем использовать здесь функцию `save_order()`. Мы сохраним все заказы, когда закроем программу», — Саймон изменил функцию `main_menu()` на следующее:

```
def main_menu(orders):
    while True:
        order = get_order()
        print("Check your order:")
        print_order(order)
        confirm = input("Confirm? Press Y to confirm, N to cancel: ")
        if confirm == "Y" or confirm == "y":
            orders.append(order)
            print("Thanks for your order:")
            print_order(order)
        else:
            continue
```

← Добавляет  
новый заказ в  
список

«Мы также изменим основную программу, чтобы она выполняла три действия: загрузка предыдущих заказов, получение новых заказов (главное меню) и сохранение всех заказов». — Саймон добавил эти три строки в конец файла, и теперь он выглядел так:

```
orders = load_orders("orders.json")
main_menu/orders)
save_orders/orders, "orders.json")
```

«Могу ли я попробовать ее сейчас?» — спросил Эрик.

«Конечно, давай!» — ответил Саймон.

Эрик запустил программу. Он ввел свое имя при появлении первого запроса, а затем выбрал компоненты напитка. Когда программа попросила его подтвердить заказ, он ввел `Y`. Программа вернулась к запросу «What's your name?».

«Отлично!» — сказал Эрик, — «Она работает. Но как я могу проверить свои заказы? Они сохранены в файле?»

«Давай проверим», — сказал Саймон.

Братья открыли файл `orders.json` и с удивлением увидели, что он все еще содержит старые заказы из вчерашних экспериментов — даже Саймон был сбит с толку.

«Посмотрим», — сказал он. «Открываем файл, читаем из него, получаем заказ. . . но мы ниразу не производим запись в файл, потому что все еще находимся в главном меню и принимаем заказы! Мы никогда не доберемся до функции `save_orders()`! Хммм, дай мне подумать, как это исправить.»

Эрик улыбнулся. Его всезнающий брат не знал, что делать. Но это длилось всего мгновение.

Саймон сказал: «Понятно. Мы не дали пользователю возможности выйти из главного меню. Мы продолжаем спрашивать у пользователя его имя, но что, если мы захотим завершить программу?»

«Я видел, что ты нажал клавиши `Control-C`, когда захотел остановить мою программу», — заметил Эрик.

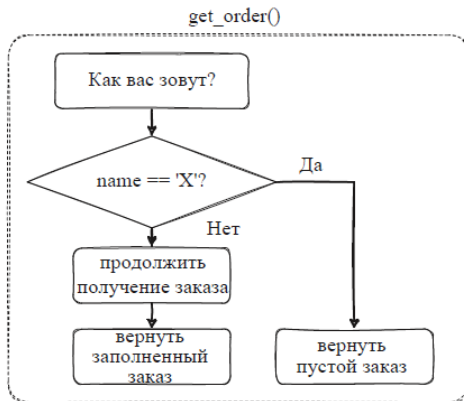
«Да, я так и сделал, но это *не обычный* способ завершения программ. Когда я это сделал, программа была *прервана*. Обычно при этом Python выдает сообщение об ошибке. Когда программа прерывается, она больше ничего не делает — она просто останавливается там, где она находится. Он не запишет наши заказы в файл и не закроет файл. Нажатие клавиш `Control-C` — плохой способ завершения программы».

Саймон немного помолчал, а затем продолжил: «Мы должны предоставить пользователю обычный способ завершения нашей программы».

«Как команда «Выход» в меню текстового процессора Word?»

«Да, вроде того. Давай скажем пользователю, что если он хочет выйти, ему следует ввести `x` и только `x` в ответ на запрос ввода его имени. Вероятность того, что у нас встретится клиент, чье настоящее полное имя — `'X'`, очень мала — фактически почти равна нулю. Итак, давай сделаем так: если клиент введет имя `x` в функцию `get_order()`, мы не будем задавать никаких других вопросов и вернем

пустой заказ, например: `order = {}`. Затем этот заказ передается функции `main_menu()` и она решает, что произойдет дальше: если заказ пуст, он сохранит его в файл и завершит работу. Если он не пуст, то она добавит заказ в список и продолжит работу. Давайте нарисуюем схему».



«Это будет наша обновленная функция `get_order()`. Позволь мне помочь тебе написать ее», — сказал Саймон. Он взял клавиатуру Эрика и начал редактировать функцию `get_order()`:

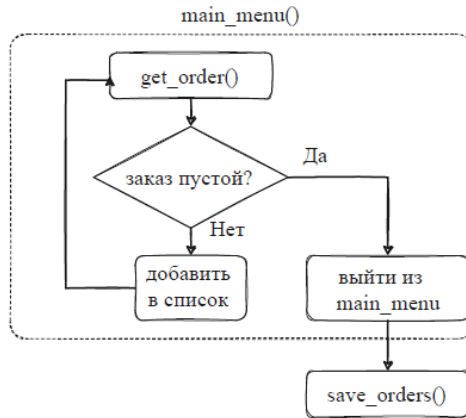
```
def get_order():
    order = {}
    name = input("Enter your name or enter 'X' to exit: ")
    if name == "X" or name == "x":
        return {}
    else:
        order["name"] = name
        drinks = read_menu("drinks.txt")
        flavors = read_menu("flavors.txt")
        toppings = read_menu("toppings.txt")
        order["drink"] = menu(drinks, "Erik's drinks",
                              "Choose your drink: ")
        order["flavor"] = menu(flavors, "Erik's flavors",
                               "Choose your flavor: ")
        order["topping"] = menu(toppings, "Erik's toppings",
                                "Choose your topping: ")
    return order
```

← Проверяет как 'X', так и 'x'

«В этой функции я просто следовал диаграмме, которую мы создали вместе».

«Теперь о главном меню», — продолжил Саймон. — «Вот еще одна диаграмма».

Он начал рисовать.



«Как мы уже говорили, если `get_order()` возвращает пустой заказ, мы выходим из главного меню. После этого наша программа сохраняет заказы в файл». Саймон отредактировал функцию `main_menu()` следующим образом:

```
def main_menu(orders):
    while True:
        order = get_order()
        if order == {}:
            print("You entered 'X', exiting...")
            return
        print("Check your order:")
        print_order(order)
        confirm = input(
            "Confirm? Press Y to confirm, N to cancel, X to finish: ")
        if confirm == "Y" or confirm == "y":
            orders.append(order)
            print("Thanks for your order:")
            print_order(order)
        elif confirm == "X" or confirm == "x":
            return
        else:
            continue
```

Если заказ пуст, нам  
следует выйти

### **Ваш ход! Отредактируйте функцию главного меню**

Отредактируйте функцию главного меню, как это сделал Саймон. Если вам нужна помощь, полный код программы для этой главы находится здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch09>.

«И все», — сказал Саймон. — «Давай проверим. На этот раз просто введи другое имя, чтобы ты мог легко увидеть, что оно было добавлено в файл JSON».

Эрик запустил программу. Он ввел “Jason”, когда программа запросила его имя. Он ввел остальную часть своего заказа и нажал `Y`, чтобы подтвердить заказ. Программа снова запросила его имя.

«Теперь давай введем `x` и посмотрим, правильно ли она завершит работу», — предложил Саймон. Эрик ввел `x` и нажал Enter.

`You entered 'x', exiting . . .` ответила программа. Затем она вернулась к знакомому приглашению `Python >>>`.

«Теперь проверь файл `orders.json`», — сказал Саймон.

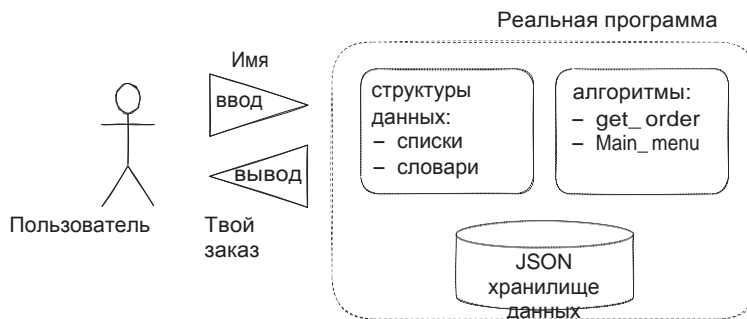
Эрик запустил текстовый редактор и открыл файл JSON. Прямо внизу файла он нашел свой недавний заказ от 'Jason'.

«Она сработала!» — сказал он. — «Программа сохранила все заказы в файле, и теперь я вижу их все!»

«Совершенно верно», — сказал Саймон и улыбнулся. Он был рад увидеть полностью работающую программу, которая принимала заказы и сохраняла их в файл. И это написал его младший брат!

«Поздравляю, Эрик!» — сказал Саймон. — «Думаю, теперь можно сказать, что ты создал настоящее приложение. Посмотри, у него есть входные и выходные данные. Оно имеет структуры данных и алгоритмы. Оно проверяет наличие ошибок. Обладает хранилищем данных. А самое главное, оно работает и является очень полезным — собирает заказы. Я абсолютно серьезно — это хорошая программа. Я очень горжусь тобой.»

Как обычно, Саймон нарисовал диаграмму. На этот раз это было то, что он назвал настоящей программой.



«Да, мне тоже нравится моя программа», — сказал Эрик. — «Она делает то, что я хочу, и выглядит хорошо. Она печатает заказы почти так же, как печатаются чеки в Starbucks. Да, почти. . . возможно, я смогу добавить пару строк или звездочек, чтобы сделать их лучше. У меня есть еще несколько идей по поводу того, что добавить в эту программу».

«Что еще ты хочешь добавить?» — спросил Саймон.

«Прежде всего, я хочу сделать эту программу веб-приложением — с меню и кнопками. Оно должно быть онлайн-приложением, чтобы я мог взять с собой iPad и пользоваться им».

«Отличная идея!» — сказал Саймон. — «Давай начнем работать над этим на следующей неделе. У меня тоже есть пара идей», — сказал он и улыбнулся.

«Почему ты улыбаешься?» — спросил Эрик.

«Я вспомнил, как ты думал, что закончил с программой после нашего первого дня».

«Ха, да, я тоже это помню», — сказал Эрик. — «Конечно, программа тогда была еще не совсем готова. Какие у тебя еще идеи?»

Саймон сказал: «Я бы добавил пару вещей в нашу структуру данных. Например, мы можем добавить дату и время оформления заказа. Таким образом, мы сможем увидеть, сколько клиентов мы обслуживали каждый день или каждый месяц».

«Да, это было бы хорошо», — согласился Эрик.

«Тогда, возможно, нам следует сохранять заказы в хранилище данных сразу после их ввода. Это позволит нам сохранить все предыдущие заказы, даже если программа откажет и будет повреждена».

«Но ты сказал, что это замедлит процесс», — вспомнил Эрик.

«Совсем немного. Но оно того стоит — иначе мы рискуем потерять все свои заказы. Я думаю об использовании для этого базы данных».

«Кроме того», — продолжил Саймон, — «нам нужны такие функции, как «распечатать все заказы» и «подсчитать, сколько порций ванильного ароматизатора мы использовали», если мы хотим сделать твою программу настоящим бизнес-приложением».

«Конечно, я хочу этого», — сказал Эрик. — «Но сначала я хочу сделать это веб-приложением и сделать его красивым».

«Конечно, мы можем начать работать над этим на следующей неделе».

## **Новое, что вы узнали сегодня**

- *Как проверить, существует ли файл* — мы использовали для этого модуль `os` из Python и метод `os.path.exists()`. Мы передаем имя файла, а возвращается `True` или `False`.
- *Нажатие сочетания клавиш Control-C не является хорошим способом завершения программы* — мы используем Control-C, когда хотим остановить программу, которая ведет себя ненормально. Качественные программы всегда должны давать вам возможность завершить их штатно.
- *Что такое реальная программа* — мы узнали, что реальные программы имеют ввод и вывод, структуры данных и алгоритмы, хранилище данных и проверку ошибок.

## **Код для этой главы**

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch09>.

# 10

## *Изучаем Flask: ваше первое веб-приложение*

---

### **В этой главе**

- Эрик создает свое первое простое веб-приложение
- Саймон объясняет, что такое Flask и как он работает
- Эмили и Эрик работают над веб-формой
- Первое интернет-меню кофейни готово!

«Ты сказал, что ты хочешь создать веб-приложение?» — спросил Саймон у Эрика на следующей неделе.

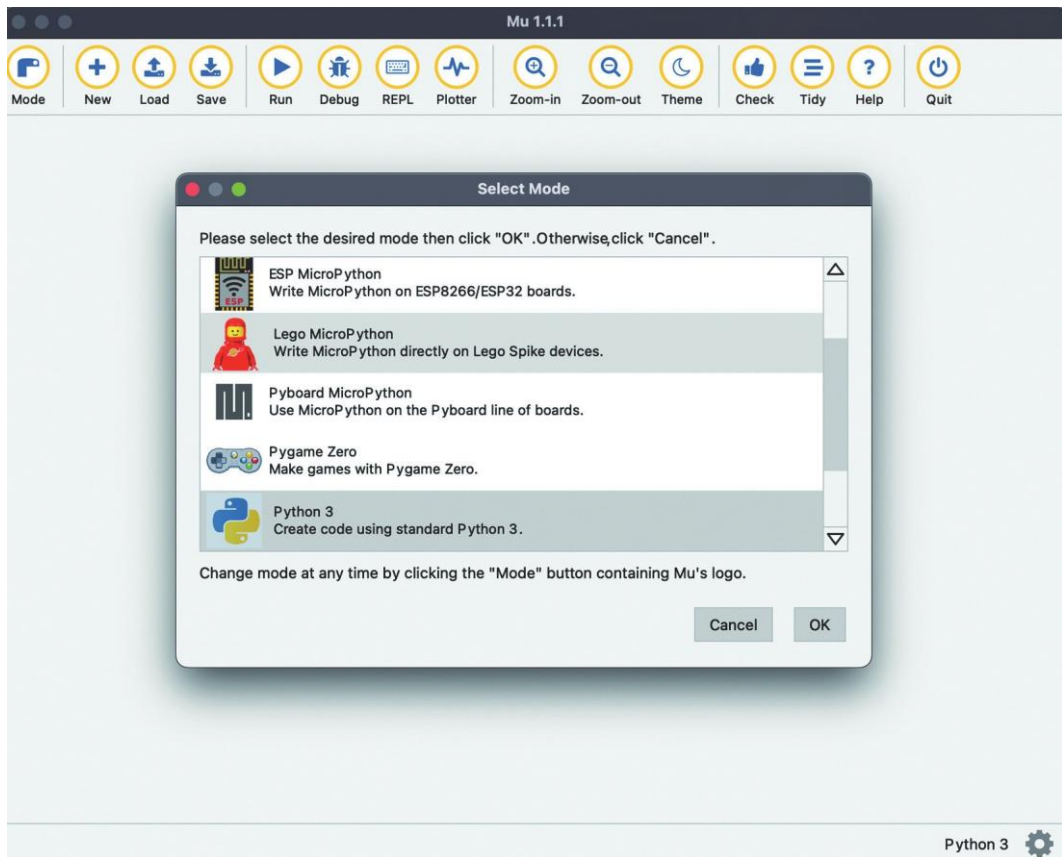
«Да, конечно!» - сказал Эрик. «Иначе как бы я мог использовать свою программу на iPad?»

«Хорошо, но будь готов: это непростая задача. Это потребует всего твоего внимания. Если ты не совсем понимаешь *все*, что мы здесь делаем, не волнуйся. Я помогу тебе, когда тебе это понадобится».

«Я знаю, ты классный брат», — сказал Эрик. В том, как он это сказал, почти не было иронии.

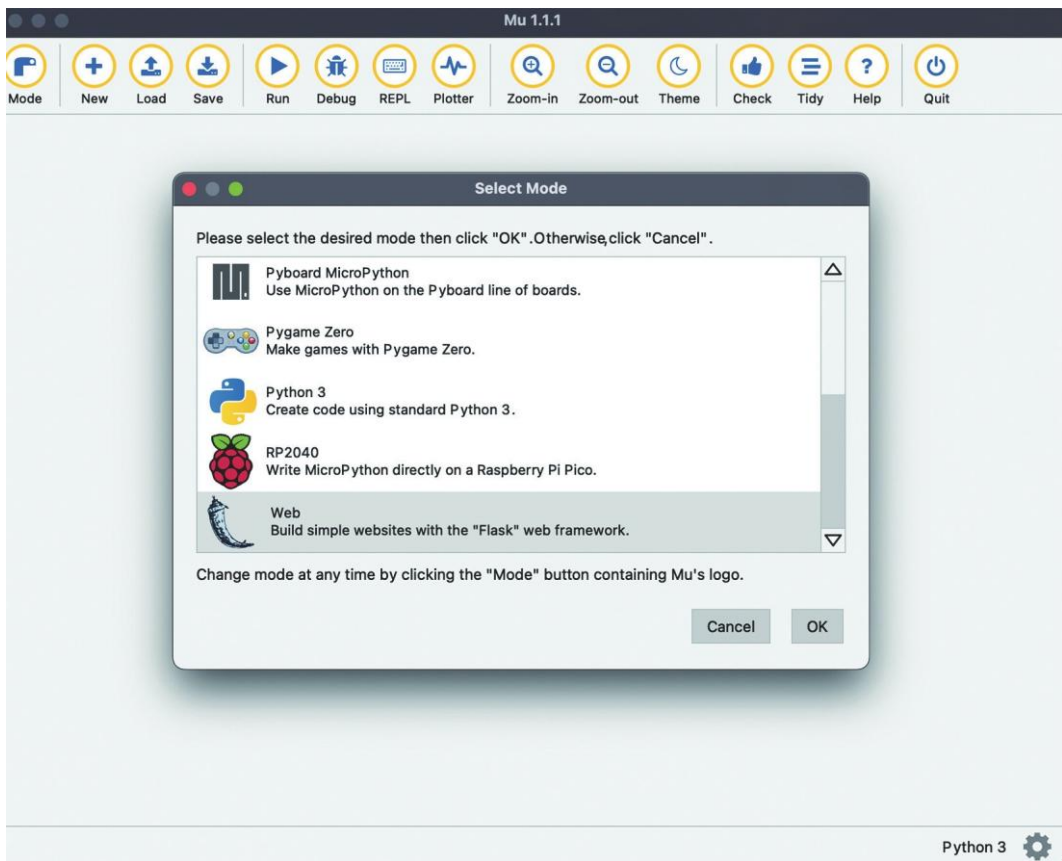
«Для нашего веб-приложения мы будем использовать нашего хорошего знакомого, редактор Ми. Для этого существует специальный режим. Запустите редактор и нажмите Mode в верхнем левом углу».

Эрик сделал это и увидел меню.



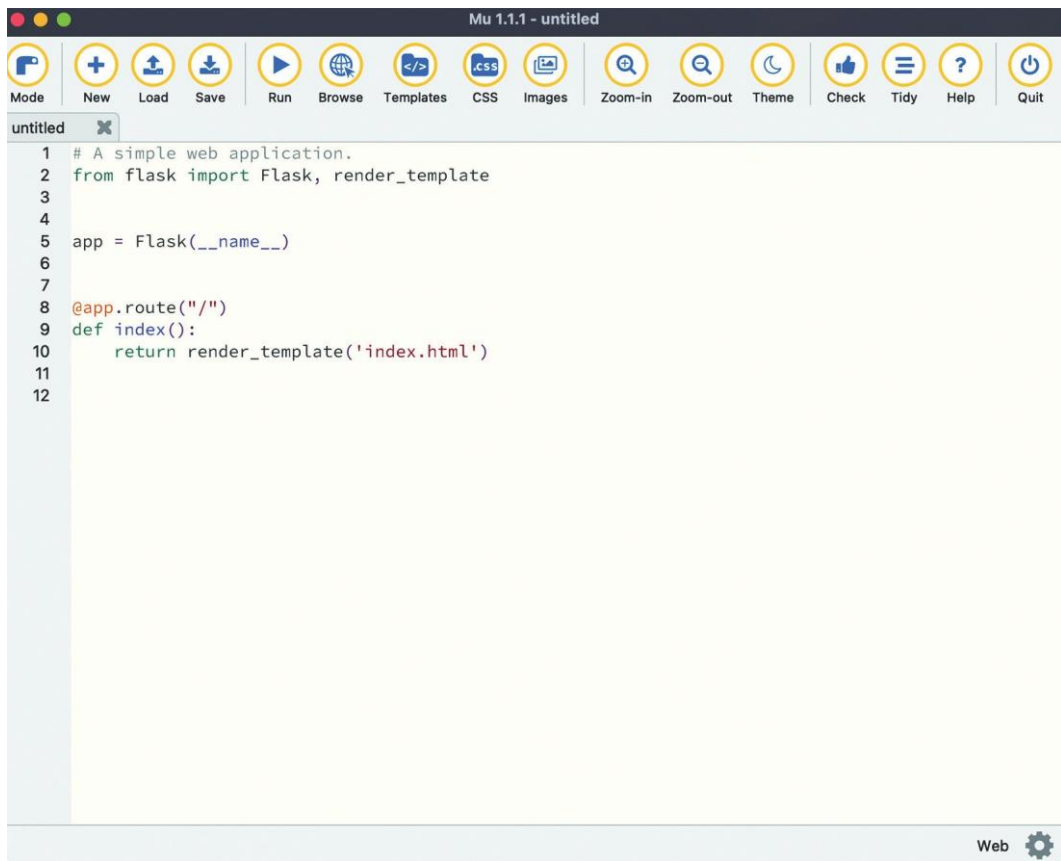
«Прокрути вниз», — сказал Саймон. — «Найдите режим Web и щелкните его. Затем нажмите «OK».





После того, как Эрик сделал это, Саймон указал на правый нижний угол и сказал: «Видишь слово ‘Web’ рядом с штеренкой? Это показывает, что мы перешли в веб-режим. Теперь давай посмотрим, что мы можем с этим сделать — нажми New».

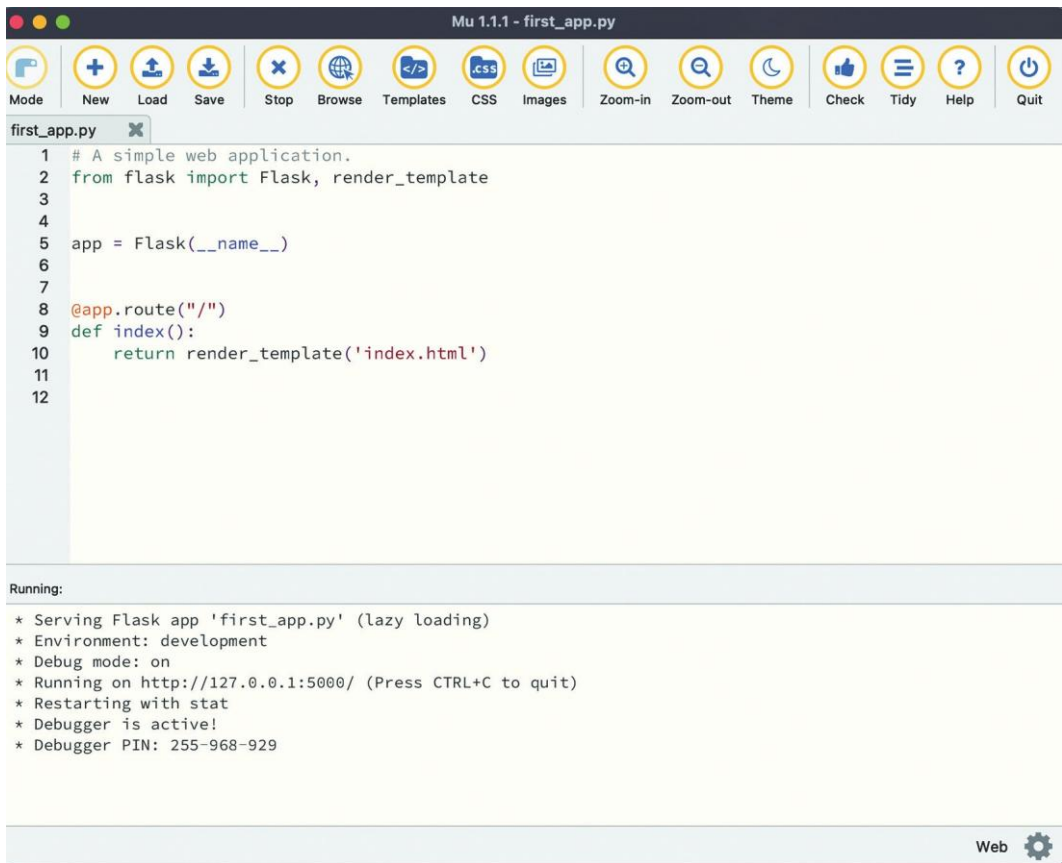
Эрик так и сделал, и код на Python тут же появился в окне редактора.



«Интересно», — сказал он. — «Му уже написал кое-что для меня. Могу ли я запустить его?»

«Конечно, давай. Во-первых, тебе придется это сохранить. Назови этот файл `first_app.py`».

Эрик нажал Run и в диалоговом окне Save ввел `first_app.py`. Он увидел результата вывода в нижней части окна.



The screenshot shows the Mu Python IDE interface. At the top, the title bar reads "Mu 1.1.1 - first\_app.py". Below it is a toolbar with icons for Mode, New, Load, Save, Stop, Browse, Templates, CSS, Images, Zoom-in, Zoom-out, Theme, Check, Tidy, Help, and Quit. The main editor window displays the code for `first_app.py`:

```
1 # A simple web application.
2 from flask import Flask, render_template
3
4
5 app = Flask(__name__)
6
7
8 @app.route("/")
9 def index():
10     return render_template('index.html')
11
12
```

Below the editor, the "Running:" panel shows the following output:

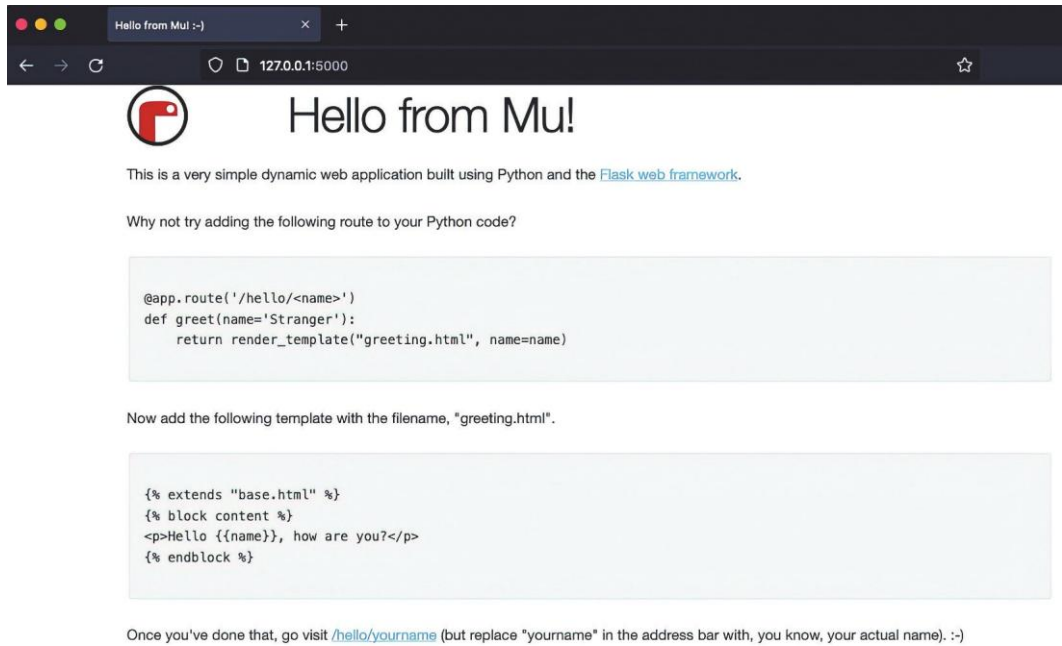
```
* Serving Flask app 'first_app.py' (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 255-968-929
```

At the bottom right of the IDE, there is a "Web" button with a gear icon.

«Что это такое?» — спросил он.

«Это говорит о том, что запущено твое первое веб-приложение. Видишь это сообщение: 'Running on http://127.0.0.1:5000/'? Это означает, что ты можешь зайти в браузер и ввести этот адрес: `http://127.0.0.1:5000/`. Иногда этот адрес называют *URL* или *Uniform Resource Locator* — ты постоянно будешь слышать эти термины при работе в сети Интернет».

Эрик открыл новую вкладку в своем браузере и ввел адрес. Он был удивлен увиденным.



«Подожди, это все написано моим редактором?» — спросил он Саймона.

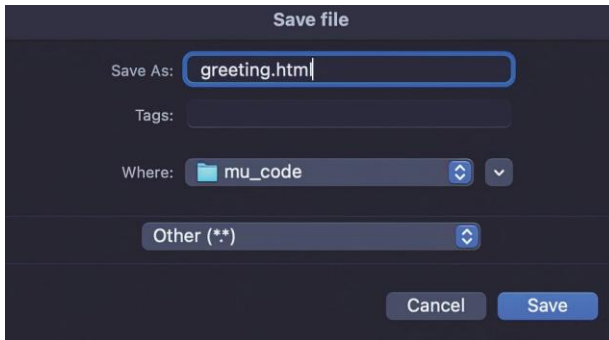
«Да, но Му предлагает тебе написать все остальное самому», — сказал Саймон. Он указал на пример кода на странице. — «Му рекомендует скопировать код из первого серого окна в твою программу. Давай, сделай это».

Это было легко. Эрик быстро скопировал текст и вставил его ниже существующего кода.

«Теперь Му предлагает тебе создать новый файл», — продолжил Саймон. «Скопируй текст из второго окна и сохрани его как новый файл с именем `greeting.html`».

Эрик знал, как это сделать. Он щелкнул «New» в редакторе Му и удалил программу, которую Му поместил в него. Он скопировал и вставил текст из второго серого окна. Затем он нажал Save.

Саймон помог ему: «Используй раскрывающееся меню, чтобы изменить `*.py` на `Other (*.*)`. В противном случае Му подумает, что ты пытаешься сохранить программу на Python. А нам нужно сообщить ему, что это файл другого типа. В поле Save As введи `greeting.html`.

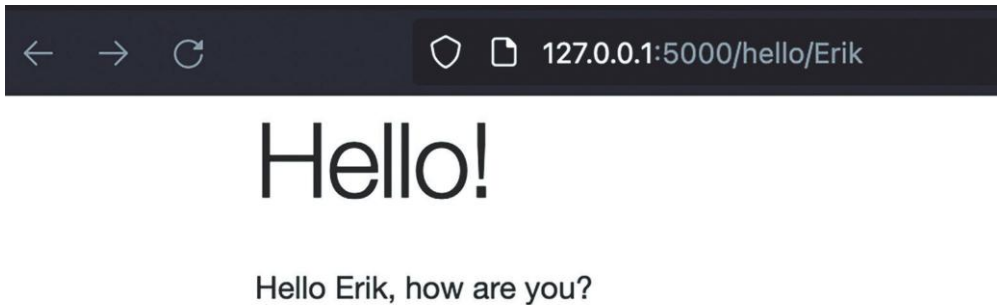


«Теперь останови программу и запусти ее снова», — сказал Саймон. — «Не забудь переключиться на вкладку `first_app.py`».

Эрик переключился на вкладку приложения, нажал **Stop** и снова нажал **Run**. Он увидел тот же результат в нижней части окна.

«Теперь вернись в свой браузер и сделайте то, что он предлагает. Посмотри, что там написано. Там предлагается перейти по адресу `/hello/yourname` и использовать свое имя. Добавь `/hello/Erik` в адресную строку сразу после `5000`».

Теперь это выглядело как настоящий взлом. Эрик ввел то, что предложил Саймон, и нажал **Enter**.



«Ух ты! Оно разговаривает со мной!» — он был действительно впечатлен.

«Это уже с тобой разговаривает твоя программа», — сказал Саймон. — «Это было легко, да?»

«Подожди», — сказал Эрик, — «если мы собираемся работать над этим веб-материалом, мне нужно позвонить Эмили. Она говорит, что изучала HTML, и это то, что нам нужно для Интернета, верно?»

«Совершенно верно», — согласился Саймон. — «Давай, позвони ей. Всегда приятно работать вместе».

**Ваш ход! Создайте свое первое веб-приложение**

Переключитесь в веб-режим в редакторе Му и создайте свое первое веб-приложение, скопировав пример со страницы браузера, как это сделал Эрик. Попробуйте запустить его от своего имени. Попробуйте другие имена. Покажите это своим друзьям и попросите их использовать свои имена.

Эмили жила неподалеку. Она пришла примерно через 15 минут, очень воодушевленная проектом. Она сразу сказала: «Эрик, покажи мне свой HTML-код!»

Эрик показал ей файл `greeting.html`. Он сказал: «Ну, это *не мой* код, это код этого редактора Му».

«Ух ты, интересно», — сказала Эмили. — «Я никогда прежде не видела в HTML этих фигурных скобок».

«Правильно», — сказал Саймон, — «потому что это не чистый HTML — это *шаблон*. Мы используем здесь программу Flask, которая входит в состав Му и которая использует шаблоны для генерации HTML».

«Понятно», — сказала Эмили. — «Но я узнаю эти теги `<h1>` и `<p>`».

«Теги? Что такое «теги»?» — спросил Эрик.

«Теги — это небольшие фрагменты кода, которые ты добавляешь в текст, чтобы изменить его внешний вид. Посмотри, ты помещаешь `<h1>` перед ‘Hello!’, а `</h1>` после, и поэтому текст выглядит больше. Это то, что в HTML называется *заголовком*, как заголовки глав».

«А что насчет `<p>`?» — спросил Эрик.

«Он означает «абзац», — объяснила Эмили. — «В HTML ты можешь писать текст так, как хочешь: в одну длинную строку, во множество коротких строк или по одному слову в строке. Но если у него есть `<p>` в начале и `</p>` в конце, он будет отображаться в браузере как один абзац».

«Существует много других тегов», — продолжила она. — «Ты можешь сделать текст жирным или курсивом, изменить цвета и многое другое».

«Эмили, ты знаешь что-нибудь о HTML-формах?» — спросил Саймон.

«На занятиях нам рассказали, что мы можем создавать формы в HTML для ввода текста или использования меню», — ответила Эмили. — «Но сама я их еще не пробовала».

«Меню — это то, что нам нужно!» — воскликнул Эрик.

«Я помогу тебе», — сказал Саймон. — «Во-первых, нам следует использовать обязательные HTML-теги. Мы всегда должны начинать файл тегом `<html>` и заканчивать тегом `</html>`. Кроме того, нам следует заключить наш текст внутри тегов `<body>`. Аналогично, мы используем тег `<body>` чтобы *открыть* текст и тег `</body>`, чтобы закрыть текст. Вот почему теги с косой чертой / называются закрывающими тегами».

```
<html>
<body>

</body>
</html>
```

«Теги подобны скобкам в списках языка Python», — сказал Эрик. Он хотел показать Эмили, что уже знает Python.

«Ты прав», — подтвердил Саймон. — «Эти теги *заключают* в себя некоторый текст и объясняют его значение. Некоторые фрагменты текста являются заголовками, некоторые — абзацами. Но сейчас мы хотим создать меню. Для этого нам сначала понадобится тег `<form>`, а затем тег `<select>` внутри него. Давай создадим очень простое меню». Он начал писать.

#### Листинг 10.1 Первая форма меню с двумя элементами

```
<html>
<body>
<form>
  <select>
    <option>Coffee</option>
    <option>Decaf</option>
  </select>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

#### **Ваш ход!** Создайте свою первую веб-форму

Создайте файл `forms.html` и сохраните его в папке `mu_code/templates`. Скопируйте приведенный выше код и протестируйте его в своем браузере. Попробуйте изменить параметры и добавить дополнительные параметры.

Саймон закончил писать код и нажал Save. Он сохранил файл как `forms.html` в каталоге `mu_code/templates`. «Сколько элементов, заключенных в теги, ты видишь здесь? Эмили, тебе это должно быть более знакомо».

Эмили начала считать. «Первый тег `<html>`, Затем тег `<body>`. Внутри `body` у нас идет `<form>`, а внутри `form` находится `<select>`. Это для меню, да? И затем в `select` у нас есть два элемента `<option>`».

"Верно. Ты проделала отличную работу, Эмили», — сказал Саймон. — «И не забудь элемент `<input>`, который является частью формы. У него нет закрывающего тега — он существует сам по себе. Мы используем его для создания кнопки отправки Submit».

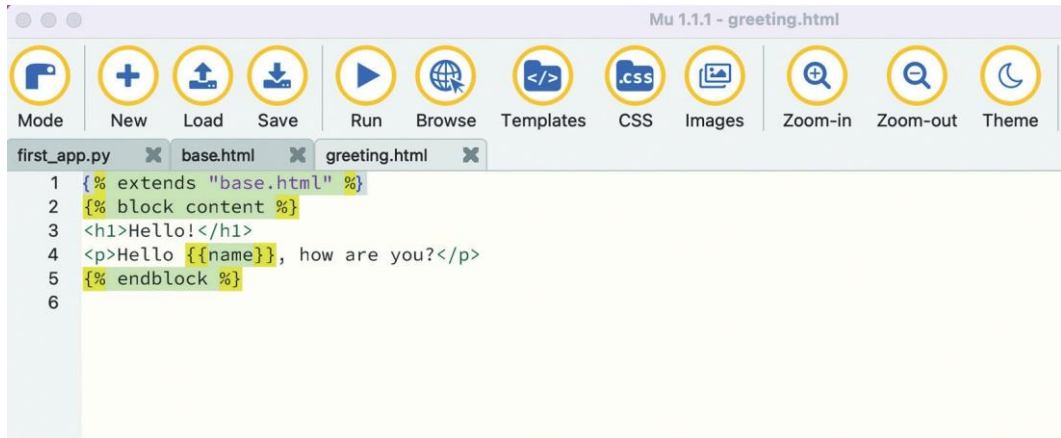
«Давай посмотрим, как это выглядит в браузере», — продолжил Саймон.

«Могу ли я открыть *файлы* в браузере?» — спросил Эрик. — «Я думал, что браузеры предназначены только для веб-сайтов».

«Конечно, можешь», — ответила Эмили. — «Мы постоянно делали это на

уроках HTML! Ты просто используешь меню «Файл» браузера и нажимаешь «Открыть файл». Затем ты найдешь свой файл. Вот и все».

«Эрик сделал то, что только что сказала Эмили, и открыл файл `forms.html`. Он увидел меню, очень похожее на то, что он видел и использовал на многих сайтах. Он щелкнул меню, и оно открылось».



«Я не знала, что можно так легко создавать формы», — сказала Эмили.

«Да, создать такую простую форму довольно легко, но есть некоторые недостающие детали», — сказал Саймон.

«Кажется мне это подходит», — сказал Эрик. — «Чего не хватает?»

«Да, это *выглядит* хорошо, но ничего *не делает*», — сказал Саймон.

«Мы должны получить данные от пользователя, а затем передать их программе. Как мы можем передать данные?» — спросил Саймон. Затем он ответил на свой вопрос. — «Мы должны использовать переменные и значения, очень похожие на используемые в Python. Позволь мне добавить кое-что к этой форме»:

#### Листинг 10.2 Обновленная форма с переменной и целью действия

```

<html>
<body>
<form action="/order" method="post">
  <select name="drink">      </select>
  <input type="submit" value="Submit">
    <option value="">- Choose drink -</option>
    <option value="coffee">Coffee</option>
    <option value="decaf">Decaf</option>
</form>
</body>
</html>

```

Определяет переменную, которую будет возвращать меню

Добавляет опцию, которая сообщает пользователю, что делать

Определяет значения, которые будут возвращать опции меню



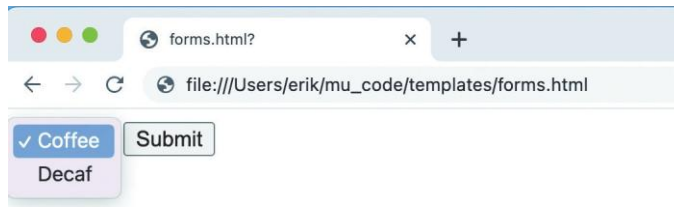
«Сначала», — сказал Саймон, — «мы определяем *переменную*, которую хотим вернуть из этого меню. В этом меню переменная называется `drink`».

«В первой опции мы говорим пользователю, что делать. Эта опция будет отображаться первой в меню и работать как подсказка. Как видите, ее значение пусто. Если пользователь не выбрал напиток, он увидит это приглашение. Без этой информации вы не сможете приготовить заказ, не так ли?»

«Наконец, следующие две опции определяют *значение*, которое может возвращать эта опция меню. Это похоже на то, что мы делали, когда выбирали пункты из текстового меню, помнишь?»

«Могу ли я попробовать?» — Эрик спросил.

«Конечно, продолжай и откройте файл еще раз. Либо просто перезагрузи его в браузере». — Эрик перезагрузил файл и выбрал в меню Decaf. Затем он нажал Submit.



«Что это такое?» — спросил он, явно озадаченный.

«О, я забыл тебе сказать», — сказал Саймон. — «Смотри, я немного изменил тег формы».

### Листинг 10.3 The action target

```
<form action="/order" method="post">
```

«В каждой форме должно быть *действие*», — начал он объяснять. — «Действие — это то, что наше приложение будет делать, когда пользователь отправляет форму. Когда пользователь делает свой выбор — кофе или кофе без кофеина, — он должен *передать* эту информацию какой-либо функции. Эта функция должна знать, что делать с этой информацией: хранить ее в файле или базе данных, распечатать ли заказ и так далее».

«Как то, что мы делали в нашей предыдущей программе?» — спросил Эрик.

«Ты продолжаешь говорить о своей «предыдущей» программе. Можешь мне ее показать?» — попросила Эмили

«Мне очень жаль, Эмили», — ответил Саймон. — «Я должен был рассказать это раньше. Я работал с Эриком над программой, которая собирает заказы в кофейне, похожей на Starbucks, где можно заказать напиток, добавить

ароматизаторы, топинги и все такое. Эрик написал программу, которая показывает меню и спрашивает клиента, что он хочет заказать. Когда он выбирает напиток, вкус и топинг, программа печатает заказ. Но программа пока работает только на терминале, в текстовом режиме. Эрику нужна твоя помощь, чтобы преобразовать ее в веб-приложение».

«Звучит как крутой проект!» — сказала Эмили. — «Надеюсь, Эрик тоже научит меня Python».

«Конечно», — сказал Саймон. — «Обучать кого-то — лучший способ научиться самому».

«Вернемся к нашей форме», — продолжил он. — «Этот атрибут `action` сообщает браузеру: ‘После того, как пользователь отправит форму, открой этот адрес и передайте туда информацию из формы’. В нашем случае адрес называется `/order`».

«Не волнуйтесь, это только звучит страшно — я покажу вам, что с этим делать», — добавил Саймон. Он заметил замешательство на лицах Эмили и Эрика.

«Я до сих пор не понимаю», — сказал Эрик. — «Где тот адрес, о котором ты говоришь?»

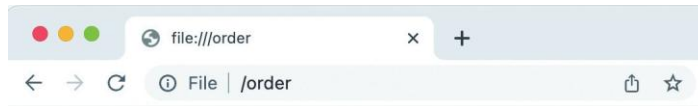
«Посмотри на свое первое приложение», — сказал Саймон. — «Видишь эту функцию `greet` ? Эту функцию для нас написал редактор Му — точнее, его авторы. Ты видишь уже знакомое определение функции, которое начинается с `def`, но прямо над ним находится что-то новенькое: `@app.route('/hello/<name>')`. В Python это называется *декоратор*, но сегодня мы не собираемся изучать декораторы».

```
@app.route('/hello/<name>') def
greet(name='Stranger'):
    return render_template("greeting.html", name=name)
```

«Сегодня для нас важно то, что ты можешь использовать его, чтобы сообщить своей программе, какую функцию использовать для какого адреса».

«Ага, это тот адрес `/hello/Erik`, который я ввёл в браузере! Теперь я понимаю», — сказал Эрик. — «Давайте покажем Эмили, как это работает». Он открыл в редакторе вкладку с файлом `first_app.py` и нажал Run. Затем он нажал Browse. В его браузере открылась страница с поздравлением от Му.

«Эмили, послушай, я могу ввести адрес здесь, сразу после этих цифр, `127.0.0.1:5000`. Посмотри, что он нам дает!» — Эрик набрал `/hello/Emily` и нажал Enter. Браузер показал им результат.



## Your file couldn't be accessed

It may have been moved, edited, or deleted.

ERR\_FILE\_NOT\_FOUND

«Вау, мне нравится!» — сказала Эмили. — «Можно попробовать?»

Она изменила “Emily” на “Erik”. Конечно, браузер показал страницу с надписью “Hello Erik, how are you?”

«Интересно!» — сказала она. — «На нашем уроке по HTML мы могли менять страницы, но нам приходилось редактировать HTML. Это намного проще!»

«Верно», — сказал Саймон. — «Это то, что мы называем *динамическими* страницами — страницами, которые меняются в зависимости от того, что ты вводишь. Ты можешь ввести свою информацию в адрес, например `/hello/Emily`, или использовать формы. Затем страница *создается* с использованием введенной тобой информации. Я уверен, что ты видел такую страницу много раз — например, когда ты оставляешь комментарий или общаешься с кем-то в Интернете. Ты нажимаешь Submit или Enter, и страница обновляется, верно? Сейчас ты научишься делать такие страницы самостоятельно».

«Позволь мне показать это на диаграмме», — сказал Саймон и начал рисовать.



«Эта комбинация букв и цифр в верхней части вашего браузера — это *адрес* или *URL*. Я пометил его синим цветом. Обычно здесь можно увидеть название сайта, например `google.com`. В нашем случае используются цифры, которые являются IP-адресом сайта. Мы используем ваш собственный

компьютер, а для каждого компьютера в мире адрес 127.0.0.1 означает «этот компьютер». Но сейчас не беспокойся об этом».

Саймон указал на первый красный кружок вокруг слова ‘order’ в адресе и сказал: «Это то, что вас должно волновать. Смотри, это часть адреса. Открыв этот адрес, мы увидим форму с меню напитков. Когда ты нажимаешь Submit,» и он проследил за стрелкой вниз по диаграмме, «форма знает, что ей следует найти функцию, отвечающую за адрес /order. Видите, она здесь, в поле `formaction`.

«А затем, — он проследил за стрелкой вверх до блока Python, — форма находит функцию Python, которая может с ней работать, потому что мы использовали декоратор `@app.route('/order')`. Видите ли, эти три вещи связаны; вам просто нужно использовать одно и то же имя в адресе, форме и программе Python».

«Я вижу, что функция также называется ‘order’ — это четвертое место, где мы ее используем?» — спросил Эрик.

«У тебя очень зоркий глаз!» — сказал Саймон и улыбнулся. «Нет, в этом случае функция может иметь другое имя. Я мог бы назвать ее как ‘new\_order’ или ‘get\_order’.

«Но теперь нам нужно написать реальную функцию. Я помогу тебе здесь. Это будет выглядеть немного пугающе, но не волнуйся. Я сам изучаю систему Flask и обычно слеую онлайн-урокам и беру оттуда примеры. Не думай, что я сам все это помню и знаю».

Саймон написал функцию, просматривая пример, который он оставил открытым в браузере. «В первой строке ты видишь слова GET и POST. Это *методы*, которые мы используем с веб-серверами. Мы используем GET когда хотим *получить* что-то с веб-сервера, например веб-страницу. Мы используем POST для *отправки* информации на веб-сервер. Как в нашем случае: мы хотим отправить или POST напиток, выбранный клиентом. Другими словами: когда ты загружаешь страницу в браузере, ты используешь GET; а когда ты нажимаешь Submit в форме, ты используешь POST. Ты поймешь это лучше, когда мы начнем использовать это, не волнуйся».

«Ниже мы начинаем использовать одно из слов, POST: `if request.method == 'POST':`. Посмотри, что там написано: в случае метода POST, , что означает, что кто-то заполнил форму и нажал Submit, мы читаем информацию, которую он ввел в форму, и распечатываем ее».

«Помнишь, мы использовали в форме `<select name="drink">?` Это наименование мы используем в следующей строке, в квадратных скобках: `drink = request.form['drink']`. Позже мы добавим другие меню вкусов и топингов. В форме они будут иметь такие названия, как `flavor` и `topping`. Здесь, в коде, мы будем использовать их как `request.form['flavor']` и `request.form['topping']`».

«В строке `print` мы просто печатаем все, что получили из формы. Ты увидишь это в редакторе».

«В последней строке мы приказываем нашему веб-серверу вывести эту страницу с меню. Это похоже на цикл меню, который мы раньше использовали в нашей программе: мы получаем информацию от клиента, выводим ее и возвращаемся в меню, чтобы получить еще один заказ. Мы повторяем этот цикл, пока не закончится ввод заказов».

В следующем листинге показана функция, которую написал Саймон:

#### Листинг 10.4 Функция обработки заказа

```
@app.route('/order', methods=('GET', 'POST'))  ← Методы, используемые с этой формой
def order():
    if request.method == 'POST':  ← Метод POST означает, что мы отправляем информацию.
        drink = request.form['drink']  ←
        print("Drink: ", drink)  ←
    return render_template("forms.html")  ←
    ↑
    |
    | Отображение шаблона forms.html
    |
    | Выводит полученный нами выбор
    |
    | Получает выбор клиента из
    | поля формы drink
```

«Позволь мне добавить еще одну вещь», — сказал Саймон. Он добавил `request` в первую строку с `import`. — «Этот модуль, называемый запросом `request`, является частью Flask. Если мы его используем, нам придется его импортировать».

Теперь первая строка выглядела так:

```
from flask import Flask, render_template, request
```

### **Ваш ход! Напишите свою собственную функцию `order()`**

Добавьте предыдущую функцию `order()` в свою программу `first_app.py`. Не забудьте изменить строку `import`.

Затем попробуйте запустить ее. Откройте новую вкладку в браузере и используйте адрес `http://127.0.0.1:5000/order`. Если у вас возникли проблемы, продолжайте читать и следите за тем, что делают Эмили и Эрик.

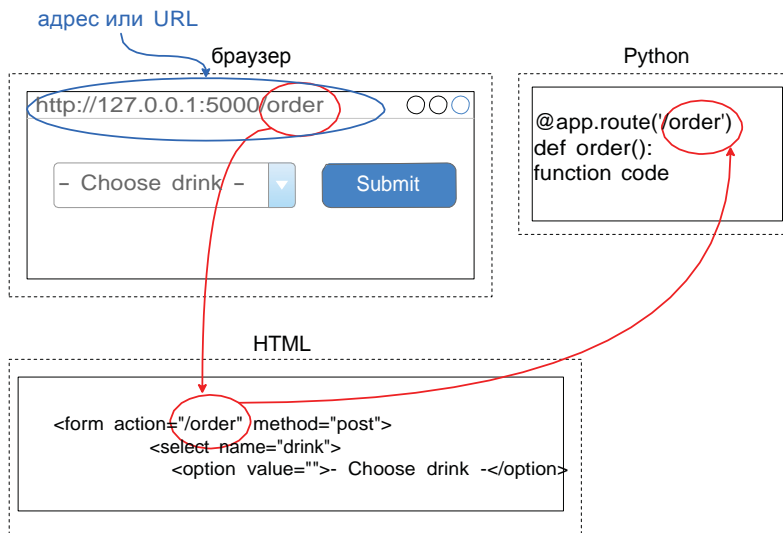
«Могу ли я запустить ее сейчас?» — спросил Эрик. Он немного устал после такого долгого объяснения. Эмили же слушала объяснения Саймона, как будто он был волшебником. Ей понравилась вся эта магия программирования, и ей не терпелось попробовать программу.

«Могу ли я запустить ее?» — спросила Эмили.

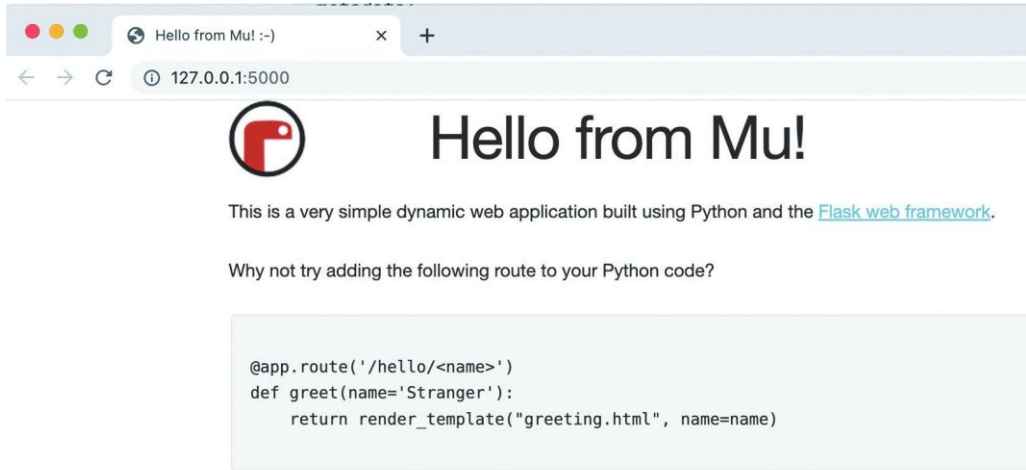
«Конечно», — сказал Эрик. — «Просто нажми Save, а затем Run».

«Теперь нажми Browse», — сказал Саймон.

Эмили так и сделала, и страница открылась в новой вкладке браузера.



«Теперь нам нужно добавить `/order` к адресу, помнишь?» — Саймон помог ей. Эмили так и сделала, и адресная строка в браузере стала такой: `http://127.0.0.1:5000/order`. На странице появилось меню, которое они только что создали вместе.

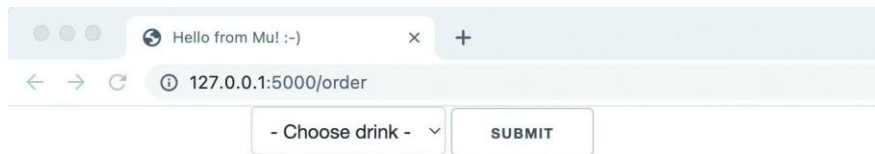


"Вперед, продолжай! Выберите напиток и нажмите Submit», — сказал Саймон.

Эмили выбрала вариант без кофеина Decaf и нажала Submit. Она получила ту же страницу с меню выбора напитка Choose Drink.

«Она сработала?» — спросила она. Эмили выглядела растерянной.

«Давай проверим», — сказал Саймон. — «Вернись в редактор. Посмотри на нижнее окно».



«Видишь эту строку: `Drink: decaf?` Это то, что выводит наша программа», — сказал Саймон. — «Это значит, что она работает!»

«Но я думала, что она напечатает заказ на странице», — сказала Эмили.

«Так и будет, поверьте мне», — сказал Саймон. — «Мы еще не написали эту часть. Мы займемся ею прямо сейчас».

Он взял клавиатуру и изменил файл `first_app.py`, добавив одну строку после строки `print()`. Саймон объяснил: «Когда мы впервые в браузере открываем адрес `/order`, это означает, что мы используем метод `GET`. Мы хотим сначала *получить* страницу, верно? У нас пока нет ничего для метода `POST`. В этом случае мы используем шаблон `forms.html`, который отображает наше меню напитков.

«Но после того, как мы выбрали напиток и нажали `Submit`, мы используем метод `POST`. Мы хотим *отправить* эту информацию в программу. В этом случае мы собираем данные из формы — выбор напитка — и используем *другой* шаблон. Я назвал его `print.html`, потому что мы хотим напечатать заказ».

Функция `order()` теперь стала выглядеть так:

```
@app.route('/order', methods=('GET', 'POST'))
def order():
    if request.method == 'POST':
        drink = request.form['drink']
        print("Drink: ", drink)
        return render_template("print.html", drink=drink)
    return render_template("forms.html")
```

Печатает заказ,  
используя шаблон  
`print.html`



«Но у нас нет файла `print.html`», — заметила Эмили.

«Хорошо, я собираюсь создать его прямо сейчас», — Саймон создал в редакторе еще один файл и сохранил его как `print.html` в разделе шаблонов `templates`:

#### Листинг 10.5 Шаблон для печати заказа (первая версия)

```
{% extends "base.html" %}
{% block content %}
<h1>Thanks for your order!</h1>
<p>Your drink: <strong>{{drink}}</strong></p>
{% endblock %}
```

Эмили посмотрела на его код и сказала: «О, это я могу понять! Ты печатаешь заголовок ‘Thanks for your order’, после открываешь новый абзац и печатаешь ‘Your drink’, а затем жирным шрифтом печатаете сам напиток. И этот `drink` в двойных фигурных скобках работает так же, как он работал с моим именем, когда напечатал ‘Hello Emily’, верно?»

«Абсолютно верно!» — воскликнул Саймон. «Ты абсолютно права, Эмили!»

#### **Ваш ход!** Отредактируйте свою веб-форму, чтобы вывести выбранный напиток на страницу

Скопируйте предыдущий шаблон в файл `print.html`. Не стесняйтесь менять заголовок и текст. Измените файл `first_app.py`, добавив строку с `return`, и тоже сохраните его. Попробуйте запустить вашу программу.

«Могу ли я попробовать?» — спросила Эмили.

«Конечно, нажмите Run», — сказал Саймон.

Эмили нажала Run и выбрала в меню Coffee. Она увидела обновленную страницу.

```
Running:
127.0.0.1 - - [16/Oct/2022 12:22:56] " [36mGET /static/img/logo.png HTTP/1.1 [0m" 304 -
127.0.0.1 - - [16/Oct/2022 12:27:22] "GET /order HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2022 12:27:22] " [36mGET /static/css/normalize.css HTTP/1.1 [0m" 304 -
127.0.0.1 - - [16/Oct/2022 12:27:22] " [36mGET /static/css/skeleton.css HTTP/1.1 [0m" 304 -
Drink: decaf
127.0.0.1 - - [16/Oct/2022 12:30:23] "POST /order HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2022 12:30:23] " [36mGET /static/css/normalize.css HTTP/1.1 [0m" 304 -
127.0.0.1 - - [16/Oct/2022 12:30:23] " [36mGET /static/css/skeleton.css HTTP/1.1 [0m" 304 -
```

«Да, она работает!» — воскликнула она.

«Да, точно!» - сказал Саймон. Это похоже на вызов функции `print_order()` в нашей предыдущей программе», — сказал Эрик.

«Да, точно!» - сказал Саймон.

«Но как мне вернуться на страницу заказа?» – спросил Эрик.

«Ты видишь, что адрес в браузере по-прежнему указывает на /order?» – спросил Саймон.

«Это значит, что если ты щелкнишь мышкой по адресной строке и нажмешь Enter, ты перезагрузишь страницу заказа. Только не нажимай кнопку перезагрузки, иначе будет создан еще один заказ».

Эмили сделала то, что сказал Саймон, и снова увидела страницу заказа.

«Но есть лучший способ», — сказал Саймон. — «Кнопка на странице печати, например «Назад на страницу заказа» или Back To The Order Page, не так ли?»

«Да, это было бы проще», — согласилась Эмили.

«Мы можем использовать для этого другую форму», — сказал Саймон. «Это будет очень просто».

Он добавил несколько строк в файл templates/print.html.

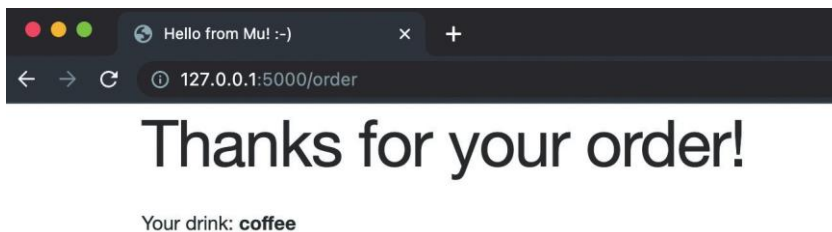
«Смотри, я создал еще одну форму, в которой есть только кнопка Submit. Я просто переименовал его в ‘New order’. Look, Посмотри, она указывает на /order в поле action. Это означает, что когда мы нажмем кнопку New Order, мы отправимся на страницу /order и снова увидим меню напитков. Попробуй!»

#### Стиль 10.6 Обновленный шаблон печати с кнопкой New Order

```
{% extends "base.html" %}
{% block content %}
<h1>Thanks for your order!</h1>
<p>Your drink: <strong>{{drink}}</strong></p>

<form action="/order">
<input type="submit" value="New order" />
</form>
{% endblock %}
```

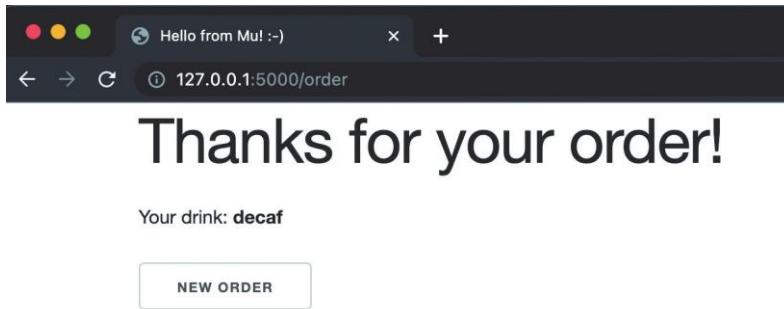
Эмили так и сделала, и после того, как она нажала Submit, она увидела страницу с новой кнопкой.



Она нажала кнопку и вернулась на страницу заказа.

Эрик отметил: «Это похоже на наше главное меню с циклом. Закажи напиток, подтверди заказ, распечатай заказ, а затем вернись в меню заказа».

«Ты прав!», — согласился Саймон. — «Позволь мне показать это на диаграмме».



«Посмотри на схему. Шаг первый: ты выбираешь в меню Decaf. Это присваивает значение `decaf` переменной `drink` в форме».

«Шаг второй: это значение `decaf` передается в нашу программу Python через `request.form`. Теперь переменная `drink` в программе Python имеет значение `decaf`».

«Шаг третий: мы передаем значение переменной `drink` из Python — которое имеет значение `decaf` — в переменную `drink` в шаблоне `print.html`».

«Шаг четвертый: мы вызываем `render_template()` с переменной `drink`, которая заменяется ее значением, `decaf`. И теперь на веб-странице будет напечатано `decaf`».

«Наконец, шаг пятый: нажимаем New Order и возвращаемся на страницу заказа».

### **Ваш ход!** Измените шаблон `print.html`

Добавьте кнопку New Order в шаблон `templates/print.html`. Протестируйте его. Можете ли вы вернуться на страницу заказа?

«Эмили и Эрик, вы сегодня отлично поработали», — сказал Саймон. — «Самое главное, вы не уснули во время всех этих долгих объяснений».

«Я почти уснул», — ответил Эрик.

«Да, я заметил», — улыбнулся Саймон. — «А если серьезно, создавать веб-приложения в 10 раз сложнее, чем работать над текстовыми меню и диалогами. Я восхищаюсь вашим терпением!»

«Но оно того стоило», — заметила Эмили. — «Программа работает!»

«Нам также нужно добавить ароматизаторы и топинги», — добавил Эрик

«Верно!» — сказал Саймон. — «Надо создать все меню, но в веб-форме. Я почти уверен, что Эмили поможет тебе в этом».

«Конечно», — сказала Эмили. — «Похоже, нам нужно добавить в шаблон больше форм `select`. Эрик, ты покажешь мне свою предыдущую программу? Завтра мы можем начать работу над веб-приложением».

«Конечно», — ответил Эрик. — «Давайте встретимся завтра и поработаем над этим».

«Я буду рад помочь», — сказал Саймон. — «Пожалуйста, дайте мне знать, когда вы начнете».

### Новое, что вы узнали сегодня

- *Веб-режим в редакторе Ми* — помимо стандартного режима Python, редактор Ми также имеет веб-режим. Он имеет простой пример веб-приложения.
- *HTML-формы* — это способ передачи информации от пользователей в веб-приложение. У вас могут быть меню, текстовые поля и кнопки. Когда вы нажимаете Submit, форма отправляет информацию на специальный адрес, указанный в поле `action`. Информация с этого адреса может быть обработана программой.
- *Flask* — программа, которая помогает нам создавать веб-приложения. Она разрабатывается сообществом открытого исходного кода и содержит хорошие учебные пособия и примеры. Flask используется многими онлайн-сайтами и веб-приложениями.

### Код для этой главы

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch10>.

# 11

## Веб-форма для заказов: Кофейня в Интернете

---

### В этой главе

- Эмили и Эрик создают настоящую веб-форму для своей кофейни
- Саймон объясняет, как передавать значения из формы в программу
- Эрик и Эмили узнают об элементах формы, таких как `input` и `select`
- Саймон объясняет, как работают файлы

На следующий день Эрик, Эмили и Саймон собрались, чтобы продолжить работу над приложением Coffee Shop.

«Ты готов к серьезному разговору?» — спросил Саймон. — «Сегодня нам нужно поработать над нашей *настоящей* веб-формой. Вчера мы попробовали очень простое меню. Теперь нам нужно добавить еще два меню и текстовое поле, в котором клиент может ввести свое имя».

«Давайте посмотрим на наш вчерашний код. Мы уже создали там одно меню — код между тегами `<select>` и `</select>`. В этом меню уже есть два варианта: `coffee` и `decaf`.



**Листинг 11.1** Первые пункты меню в `templates/forms.html`

```
{% extends "base.html" %}
{% block content %}

<form action="/order" method="post">
  <select name="drink">
    <option value="">- Choose drink -</option>
    <option value="coffee">Coffee</option>
    <option value="decaf">Decaf</option>
  </select>
  <input type="submit" value="Submit">
</form>
{% endblock %}
```

Пункт меню для coffee

Пункт меню для decaf

«Теперь нам нужно добавить еще одну опцию в меню `drink`. Эрик, я думаю, у тебя в меню был `'chocolate'`, верно?»

«Да, кофе, шоколад и кофе без кофеина», — подтвердил Эрик.

«Хорошо. А как насчет вкусов и топингов?»

«Что касается вкусов, у нас есть карамель, ваниль, мята, малина и без ароматизатора», — сказал Эрик. — «Для топинга у нас есть шоколад, корица и карамель».

«Вкусно», — сказала Эмили.

«Вот вам задание», — сказал Саймон. — «Во-первых, добавьте третий вариант в меню `drink`. Во-вторых, добавьте еще два меню для вкусов и топингов. Это потребует выполнения копирования, вставки и набора текста, а также внимания. Кто за клавиатурой?»

«Я», — сказала Эмили. — «Я могу сделать это. Я знакома с HTML и всеми этими тегами».

«Очень хорошо, вперед!»

Эмили начала редактировать файл. Эрик время от времени наблюдал и задавал вопросы.

«Почему ты переместила строку `<option>` вправо? Существуют ли в HTML те же правила в отношении отступов, что и в Python?»

«Нет, в HTML не нужно делать отступы строк», — ответила Эмили. — «Я могла бы написать их все с первой позиции, но так выглядит лучше. Мой преподаватель говорит, что это хорошо для *удобочитаемости*».

«Да, Саймон говорит мне то же самое», — сказал Эрик. — «В Python тебе приходится использовать отступы для блоков, но в некоторых случаях ты можешь выбирать, размещать ли код в одной строке или использовать новые строки и отступы. Саймон сказал, что тебе следует писать *читаемый* код — по крайней мере, для себя».

С помощью Эрика Эмили перечислила все варианты вкусов и топингов и создала следующий HTML-шаблон:

#### Стиль 11.2 Дополнительные пункты меню в `templates/forms.html`

```
{% extends "base.html" %}
{% block content %}

<form action="/order" method="post">
    <select name="drink">
        <option value="">- Choose drink -</option>
        <option value="coffee">Coffee</option>
        <option value="chocolate">Chocolate</option>
        <option value="decaf">Decaf</option>
    </select>
    <select name="flavor">
        <option value="">- Choose flavor -</option>
        <option value="caramel">Caramel</option>
```



```

    <option value="vanilla">Vanilla</option>
    <option value="peppermint">Peppermint</option>
    <option value="raspberry">Raspberry</option>
    <option value="plain">Plain</option>
  </select>
  <select name="topping">
    <option value="">- Choose topping -</option>
    <option value="chocolate">Chocolate</option>
    <option value="cinnamon">Cinnamon</option>
    <option value="caramel">Caramel</option>
  </select>

  <input type="submit" value="Submit">
</form>

{% endblock %}

```

### **Ваш ход! Создайте свои меню**

Создайте новые меню в файле `templates/forms.html`. Смело используйте разные варианты. Возьмите несколько идей из местного кафе или магазина мороженого.

Саймон посмотрел на их код и сказал: «По моему мнению, выглядит хорошо. Но прежде чем мы перейдем с HTML на Python, давайте добавим в форму еще кое-что. Помните, мы также запрашиваем имя клиента».

«Как в Старбаксе!» — отметила Эмили.

— Именно, — сказал Саймон. «Позвольте мне добавить еще одну строку в вашу форму. Она называется `input` и имеет тип `text`. Мы будем использовать его для ввода имени. Видите ли, я добавил ее сразу после строки `form`.

### **Листинг 11.3 Текстовое поле для ввода имени в `templates/forms.html`**

```

{% extends "base.html" %}
{% block content %}

<form action="/order" method="post">
  <input type="text" name="name" placeholder="ENTER NAME">
  <select name="drink">
    <option value="">- Choose drink -</option>
    <option value="coffee">Coffee</option>
  . . .

```

**Ваш ход! Добавьте поле `input`**

Добавьте поле ввода `input` имени клиента. Не забудьте поставить плейсхолдер в качестве подсказки для пользователя.

«Взгляни еще раз на HTML-код», — продолжил Саймон. — «Все три меню, которые добавила Эмили, и строка `input` находятся между открывающим тегом `<form>` и закрывающим тегом `</form>`. Эмили проделала большую работу по созданию отступов для удобства чтения».

«Теперь мы ясно видим, что они находятся внутри формы. Это означает, что все они являются частями *одной и той же формы*. Итак, когда ты нажмешь Submit, они все вместе будут переданы в программу».

«В программе мы сможем читать их с помощью словаря `request.form`, точно так же, как мы это делали в нашей простой программе».

«Как в `request.form['drink']`?» — спросил Эрик.

«Да, верно», — ответил Саймон. — «Послушай, Эмили присвоила им наименования, `flavor` и `topping`, чтобы мы могли использовать их в качестве ключей для словаря в нашей программе. Давайте снова откроем наш `first_app.py` и посмотрим, где нам нужно внести изменения. Помнишь, что мы здесь делаем? Мы получаем словарь формы и копируем значение ключа `drink` в переменную `drink`. Например, если покупатель выбирает шоколад `chocolate`, он будет присвоен переменной `drink`. Эта переменная будет передана в форму с имен `print.html` для отображения — или, как некоторые говорят, *визуализации* — на странице».

**Фрагмент 11.4 Первая версия `order()` in `first_app.py`**

```
...
@app.route("/order", methods=("GET", "POST")) def
order():
    if request.method == "POST":
        drink = request.form["drink"]
        print("Drink: ", drink)
        return render_template("print.html", drink=drink)

    return render_template("forms.html")
```

Саймон продолжил: «Теперь, когда мы получаем больше значений из формы, нам следует собрать их все и передать в шаблон. Эрик, можешь добавить эти строки?»

«Я так думаю, что да», — ответил Эрик.

Эрик добавил строки для вкуса и топинга, когда Саймон сказал: «Не забудь о наименовании `name`!»

Эрик добавил его, и теперь его код выглядел так:

**Листинг 11.5** Дополнительные параметры меню переданы в шаблон печати в `first_app.py`

```
...
@app.route("/order", methods=("GET", "POST")) def
order():
    if request.method == "POST": name=request.form["name"]

        drink=request.form["drink"]
        flavor=request.form["flavor"]
        topping=request.form["topping"]
        return render_template("print.html",
                               name=name,
                               drink=drink,
                               flavor=flavor,
                               topping=topping)

    return render_template("forms.html")
```

**Ваш ход!** Отредактируйте свое приложение на Flask

Отредактируйте свою программу `first_app.py` и добавьте переменные, как это сделал Эрик. Пока не запускайте ее, потому что у вас нет файла `print.html`.

«Отличная работа, Эрик!» — сказал Саймон. — «Ну, возможно, так еще слишком рано говорить, не проверив ее», — и он улыбнулся.

«Теперь осталась только одна маленькая вещь», — продолжил Саймон. — «Нам нужно добавить эти переменные в шаблон `print.html`. Давай откроем его в редакторе. Смотри, помимо основного напитка, нам нужно вывести вкус и топпинг. Да, и не забывай, что теперь мы знаем имя клиента. Было бы вежливо вывести приветствие. Эмили, ты можешь отредактировать этот HTML-код?»

«Конечно», — сказала Эмили. Она начала редактирование, объясняя это Эрику.

«Во-первых, давай заменим ‘Thanks for your order’ на ‘Hello’ и имя покупателя. Я оставлю его здесь как `<h1>` чтобы сделать его жирным и большим».

«Затем мы поместим ‘Here is your order’ в `<h2>` », — продолжила она. — «Он будет немного меньше. После этого выведем в порядке: основной напиток, ароматизатор и топпинг, каждый в отдельном абзаце. Я продолжу использовать тег `<strong>` , чтобы выделить его жирным шрифтом. Внизу страницы у нас есть простая форма с одной кнопкой, позволяющей вернуться на страницу заказа. Надеюсь, ты помнишь это еще со вчерашнего дня».

**Листинг 11.6** Шаблон для печати заказа в `templates/print.html`

```
{% extends "base.html" %}
{% block content %}

<h1>Hello {{name}}!</h1>

<h2>Here is your order:</h2>

<p>Drink: <strong>{{drink}}</strong></p>
<p>Flavor: <strong>{{flavor}}</strong></p>
<p>Topping: <strong>{{topping}}</strong></p>

<form action="/order">
<input type="submit" value="New order" />
</form>
{% endblock %}
```

Эта форма с помощью одной кнопки возвращает нас на страницу заказа.

**Ваш ход! Создайте шаблон** `templates/print.html`

Создайте шаблон для печати, как это только что сделала Эмили. Обратите внимание на переменные: вы должны использовать переменные, которые вы передали из программы. Если вы изменили файл `first_app.py` и использовали другие переменные, измените их и здесь. Затем попробуйте запустить приложение.

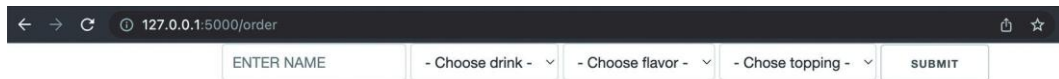
«Можем ли мы попробовать ее сейчас?» — спросил Эрик.

«Ты сохранил все свои файлы?» — спросил Саймон. — «Посмотри на свои вкладки в редакторе. Му показывает их маленьким красным кружком, если ты их не сохранил».

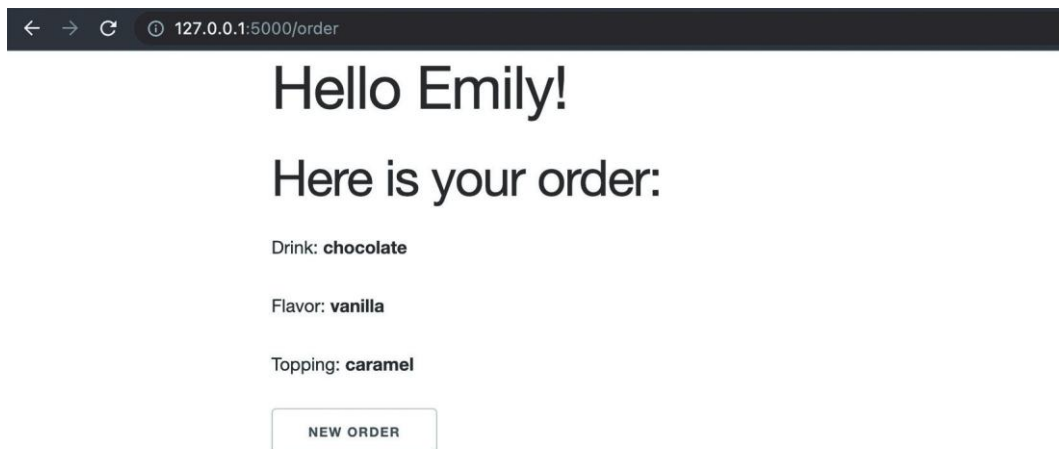
«Все в порядке», — сказал Эрик.

«Теперь переключись на вкладку Python и нажми Run».

Эрик так и сделал, а затем нажал Browse. В его браузере открылась новая вкладка со знакомой страницей “Hello from Mu!”. Эрик добавил `/order` в адресную строку и увидел меню.



«Дай мне попробовать!» — попросила Эмили. Она ввела свое имя и выбрала Chocolate, Vanilla и Caramel. Она нажала Submit и распечатала свой заказ.



«Ух ты, это работает!» — она была рада увидеть настоящее веб-приложение,

которое они только что написали вместе.

«Это здорово», — сказал Саймон. — «Теперь нам нужно добавить функции, которые мы написали для текстовой версии».

«Какие функции?» — Эмили была озадачена. Она думала, что все уже сделано. — «Эрик, ты можешь объяснить?»

«То, что мы здесь написали», — начал он, — «это пользовательский интерфейс для нашего приложения. Мы делали это и раньше, но для текстового режима. Это веб-приложение выглядит намного лучше. Но помимо печати заказа мы еще хотим сохранять его в файл. Например, мы можем посчитать, сколько заказов с вкусом карамели мы обслужили».

«Понятно, чтобы знать, когда купить еще карамели для кофейни», — догадалась Эмили.

«Точно! Ты думаешь как менеджер магазина, Эмили!» — сказал Саймон.

Он продолжил: «Как раз перед тем, как ты присоединилась к нам, мы создали функцию для сохранения заказов в файле. Наша программа сохраняла заказы, когда мы сообщали ей, что закончили. В следующий раз, когда мы запустили программу, она загрузила все предыдущие заказы в список и была готова добавить в него новые заказы».

Саймон остановился на некоторое время, задумавшись: «Но с этим новым веб-приложением нам придется делать это немного по-другому».

«Почему?» — спросил Эрик.

«Потому что наше приложение работает постоянно, и мы не собираемся его останавливать. Такое приложение называется *веб-сервисом*. Он не останавливается. Он работает все время. Более того, если с ним что-то случится и он по какой-то причине выйдет из строя, другая программа перезапустит его, чтобы убедиться, что он всегда готов отвечать на запросы — в нашем случае — принимать заказы».

«Для нас», — продолжил Саймон, — «это означает, что мы должны сохранять каждый заказ сразу после его получения. Позволь мне быстро найти правильный способ сделать это».

Он открыл новую вкладку в браузере и что-то искал пару минут. — «Теперь я знаю», — сказал он наконец. — «Эрик, давай откроем нашу предыдущую программу и скопируем функции `load_orders()` и `save_orders()` в наше веб-приложение».

«Где мы собираемся их использовать? В функции `order()`?» — спросил он Саймона. — «В прошлый раз ты сказал мне, что я должен внести их в код, *прежде* чем использовать, верно?»

«Да, совершенно верно», — сказал Саймон. — «Помести их в самом начале, даже до строки `app = Flask(name)`. Таким образом, мы отделим наши файловые функции от функций веб-приложения. Также не забудь добавить `import json` и `import os`».

Эрик скопировал функции из файла `main_menu.py`, над которым они работали несколько дней назад.

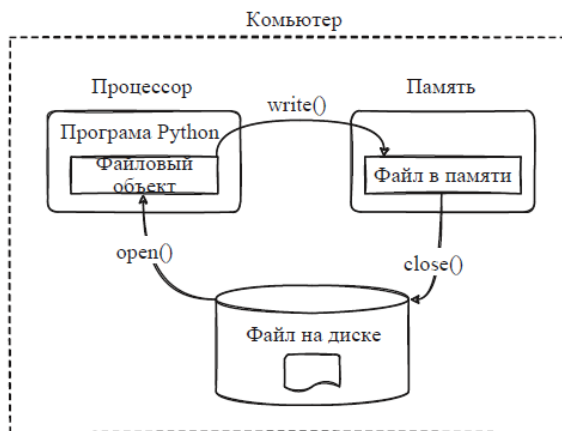
Саймон проверил свою работу и сказал: «Хорошо. Нам просто нужно закрыть файл после загрузки заказов и, самое главное, после их сохранения».

«Но раньше мы этого не делали. Почему мы должны сделать это сейчас? И что именно означает *закрытие файла*?» — спросил Эрик.

Саймон сказал: «Мы говорили об этом некоторое время назад. Давай я найду схему. Эмили, я думаю, ты тоже захочешь это узнать».

«Это что-то вроде закрытия файла в Word?» — спросила Эмили.

«Да, но в данном случае нам следует сделать это из нашей программы на Python. Важно понимать, что пока ты не закроешь файл, Python хранит заказы в памяти. Если что-то случится с нашим компьютером и он выключится, все заказы будут потеряны. Итак, нам нужно закрыть файл, чтобы убедиться, что все заказы сохранены на диске. Посмотри на схему».



### **Ваш ход! Попробуйте объяснить, как работают файлы**

Используйте эту диаграмму и попытайтесь объяснить кому-нибудь, как работают файлы. Вы будете это лучше понимать после того, как объясните кому-нибудь?

«Раньше мы этого не делали, потому что наша программа автоматически закрывала все файлы, когда мы ее завершали. В данном веб-приложении мы будем закрывать файл с заказами каждый раз, когда будем получать новый заказ».

«Но ты сказал, что это замедляет программу», — напомнил Эрик.

«Да, совсем немного. Но инженерное искусство — это компромиссы — это то, что нам постоянно говорит мой преподаватель робототехники. Наша программа становится все более реалистичной, поэтому нам следует подумать о сохранении важных данных, даже если это немного замедлит работу программы».

Саймон добавил метод `f.close()` в обе функции, поэтому теперь функции выглядели следующим образом:

**Листинг 11.7** Функции для загрузки и сохранения заказов в `first_app.py`

```
import os
import json

from flask import Flask, render_template, request

def save_orders(orders, filename):
    f = open(filename, "w")
    json.dump(orders, f, indent=4)
    f.close()
    return

def load_orders(filename):
    if os.path.exists(filename):
        f = open(filename, "r")
        orders = json.load(f)
        f.close()
        return orders
    else:
        orders = []
        return orders

. . . . .
```

Саймон добавил: «Конечно, здесь было бы лучше использовать базу данных. Возможно, мы добавим эту возможность позже».

«Теперь нам нужно загрузить заказы из файла», — сказал Саймон. Он добавил строку непосредственно перед строкой `Flask`:

**Листинг 11.8** Создание списка заказов, используя `load_orders()` в `first_app.py`

```
. . . . .
orders = load_orders("orders.json")

app = Flask(name)
. . . . .
```

«Теперь у нас есть список заказов, к которым мы будем добавлять новые заказы. Это рождает идею», — Саймон начал печатать, объясняя при этом.

«У нас есть список, в котором мы храним заказы, и это список словарей Python, верно? Так что, когда мы получим новый заказ из формы, лучше скопировать его в словарь, а не разделять переменные. Мы добавим этот словарь в список, а затем сохраним его в файл. Более того, мы можем передать словарь в шаблон HTML для его отображения. Это упростит нашу программу!»

Он закончил печатать, и функция `order()` теперь выглядела так:

**Листинг 11.9** Переход в словарь за новым порядком в `first_app.py`

```
@app.route("/order", methods=("GET", "POST"))
def order():
    if request.method == "POST":
```



```

        new_order = {"name": request.form["name"],
                     "drink": request.form["drink"],
                     "flavor": request.form["flavor"],
                     "topping": request.form["topping"]}
    orders.append(new_order)
    save_orders(orders, "orders.json")
    return render_template(
        "print.html", new_order=new_order
    )

return render_template("forms.html")

```

Добавляет новый заказ в список

Создает новый словарь из формы

Сохраняет список в файле JSON

Передает словарь в шаблон

### Ваш ход! Редактирование приложения

Используйте три предыдущих листинга и отредактируйте свою программу `first_app.py`. Убедитесь, что вы понимаете, что происходит, когда вы создаете новый заказ и передаете его в шаблон.

«Наконец, нам нужно изменить шаблон HTML», — сказал Саймон. — «Эмили, ты мне поможешь?»

«Конечно, но я не знаю, что делать», — ответила она.

«Послушай, раньше мы передавали в шаблон переменные, чтобы отображать их на странице: `name`, `flavor` и так далее. Теперь вместо четырех переменных мы передадим только одну — словарь `new_order` — который содержит все четыре. В Python это называется *словарем*. Ты передаешь ему ключ, а он возвращает значение. Например, ты передаешь ему `flavor`, а он возвращает `caramel`».

«Да, Эрик уже рассказывал о словарях в Python. Но как мы можем использовать их в HTML?»

«Точно так же, как мы делаем в Python: с квадратными скобками! Позволь мне показать тебе», — сказал Саймон. Он заменил `name` на `new_order["name"]` в строке `Hello`.

«Теперь я вижу это», — сказала Эмили. Она начала менять другие переменные. В следующем листинге показан шаблон `print.html` после того, как она закончила:

#### Листинг 11.10 Использование словаря в `templates/print.html`

```

{% extends "base.html" %}
{% block content %}

<h1>Hello {{new_order["name"]} }</h1>

<h2>Here is your order:</h2>

<p>Drink: <strong>{{ new_order["drink"]} }</strong></p>
<p>Flavor: <strong>{{ new_order["flavor"]} }</strong></p>
<p>Topping: <strong>{{ new_order["topping"]} }</strong></p>

<form action="/order">

```

```
<input type="submit" value="New order" />
</form>
{% endblock %}
```

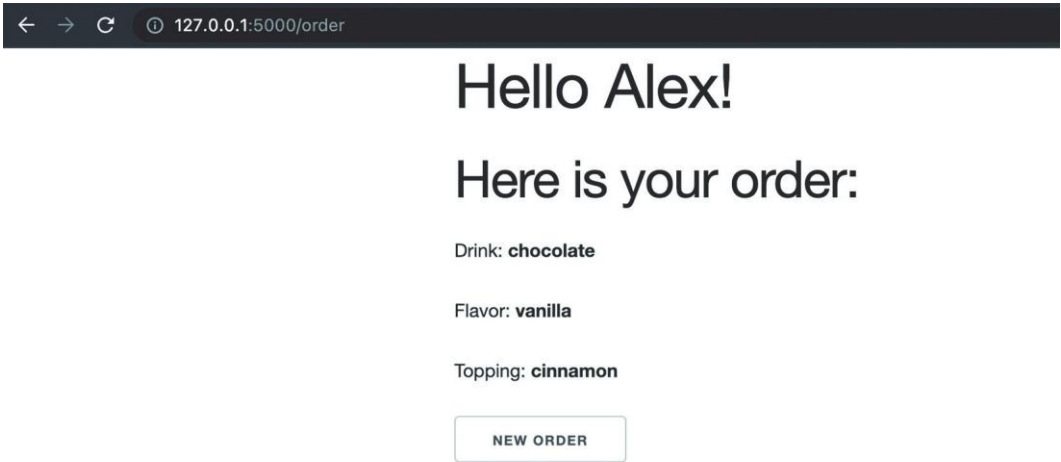
**Ваш ход!** Отредактируйте файл `templates/print.html`

Отредактируйте шаблон печати и перейдите от использования переменных к использованию словаря. Обратите внимание на ключи словаря — если вы меняли их в программе Python, вам следует изменить их и здесь. Попробуйте запустить приложение сейчас.

«Превосходно!» — сказал Саймон. — «Готовы протестировать?»

«Да!» — ответил Эрик. Он открыл обычный адрес, `http://127.0.0.1/order`.

Он ввел Alex и выбрал chocolate, vanilla и cinnamon. Ион получил заказ.



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:5000/order`. The page content is as follows:

# Hello Alex!

## Here is your order:

Drink: **chocolate**

Flavor: **vanilla**

Topping: **cinnamon**

**NEW ORDER**

«Теперь нам нужно проверить, сохранила ли она этот заказ в файле», — сказал Саймон.

Он нашел файл `orders.json` в папке `mu_code` и открыл его в текстовом редакторе.

«Да!» — сказал он. — «Наша программа сохранила заказ в файле! Посмотри — первый заказ сделан в тот день, когда мы тестировали функцию `save_order()`. Но второй заказ — это то, что ты только что добавил. Очень хорошо!»

```
orders.json
{
  {
    "name": "Erik",
    "drink": "coffee",
    "flavor": "caramel",
    "topping": "chocolate"
  },
  {
    "name": "Alex",
    "drink": "chocolate",
    "flavor": "vanilla",
    "topping": "cinnamon"
  }
}
```

«А как насчет файлов меню?» — спросил

Эрик. «Какие файлы меню?»

«Помнишь», — начал Эрик, — «ты сказал мне, что менеджер магазина может не знать Python. И тогда мы создали наши меню в виде простых текстовых файлов. Сейчас наше меню в HTML, но я думаю, что менеджер магазина также может не знать HTML. Можем ли мы сделать то же самое с этим веб-приложением и позволить менеджеру магазина редактировать текстовые файлы вместо HTML, если он захочет добавить что-либо в меню?»

«Отличная идея, Эрик!» — ответил Саймон. — «Я думаю, что это возможно с помощью HTML-шаблонов, но мне придется проверить документацию Flask. Но прежде чем вы уйдете, давай кратко подведем итог тому, что мы узнали сегодня».

«Ты объяснил, как работают файлы на компьютере. Я не знал этого раньше», — сказал Эрик.

«Я узнала о поле ввода `input` в формах HTML», — сказала Эмили. — «И было здорово работать над реальным приложением, а не над учебными упражнениями».

«Я согласен», — сказал Саймон. — «Я думаю, что лучший способ чему-то научиться — это работать над реальным проектом. Сегодня мы многое сделали, так что давайте отдохнем и начнем работать над идеей Эрика завтра».

## Новое, что вы узнали сегодня

- *Веб-сервис* — веб-сервис — это программа, которая работает постоянно и отвечает на запросы, например, на новый заказ в кофейне.
- *Почему важно закрывать файлы* — Когда вы работаете с файлами, Python сохраняет обновления в памяти. Если вы хотите быть уверены, что ваши данные сохранены на постоянном диске, вам следует вызвать `f.close()`.

### *Код для этой главы*

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch11>.

# База данных: Нам нужно хорошее хранилище

## В этой главе

- Саймон предлагает отдельную папку для приложения
- Эрик и Эмили перемещают файлы приложения в новую структуру папок
- Эрик повторно использует функции для чтения текстовых файлов
- Эрик и Эмили узнают о базах данных
- Саймон помогает Эмили и Эрику изменить приложение для использования базы данных

«Давайте сделаем наш проект немного серьезнее», — сказал Саймон, когда друзья встретились на следующий день.

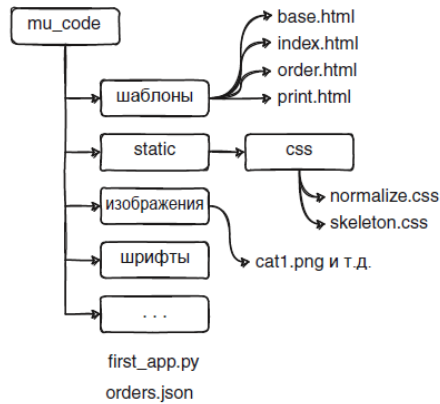
«Вчера ты сказал что-то о базе данных», — сказал Эрик. — «Именно это ты имеешь ввиду?»

«Да, и это тоже», — ответил Саймон. — «Но сначала я предлагаю внести изменения в структуру наших папок. Давай упорядочим наши файлы. Посмотри на наши файлы и папки».

Он взял лист бумаги и начал рисовать и объяснять.

«Когда мы начали работать с редактором Mu, он поместил все наши программы в папку, назначенную по умолчанию — `mu_code`. Затем мы обнаружили, что в папке `templates` внутри `mu_code` также есть все необходимые

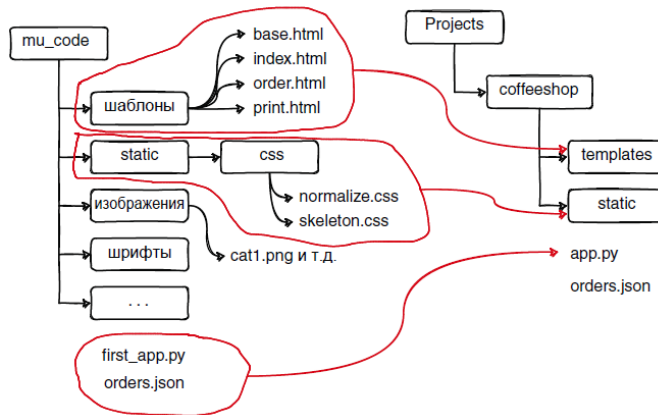
шаблоны для веб-приложений. Если вы посмотрите ее в файловом менеджере — Finder в macOS или Проводник в Windows — вы увидите, что в ней находится больше папок, таких как `static`, `images`, `fonts` и так далее. Помните, мы уже самостоятельно создали пару шаблонов, `order.html` и `print.html`, и поместили их в `templates`. Если вы заглянете в папку `static`, вы обнаружите папку `css` — я думаю, Эмили, ты знаешь, что она предназначена для таблиц стилей — и там находится пара файлов».



«Видите ли, в нашем приложении есть несколько файлов в разных папках. Предлагаю создать отдельную папку для приложения Coffee Shop и скопировать в эту папку все нужные нам файлы. Это также поможет нам отделить наше веб-приложение Coffee Shop от предыдущей версии. Всегда полезно хранить разные проекты в отдельных папках».

Саймон продолжил: «Давайте создадим папку для всех наших проектов, включая будущие проекты. Обычно я называю ее `Projects`, аналогично папкам `Downloads`, `Pictures` и `Documents`, которые у нас уже есть. В папке `Projects` мы создадим еще одну папку и назовем ее `coffeeshop`. Мы переместим в нее все файлы, связанные с нашим проектом Coffee Shop. Я также предлагаю переименовать `first_app.py` просто в `app.py`. Именно так люди обычно называют свои приложения Flask. Позвольте мне показать вам, что я имею в виду».

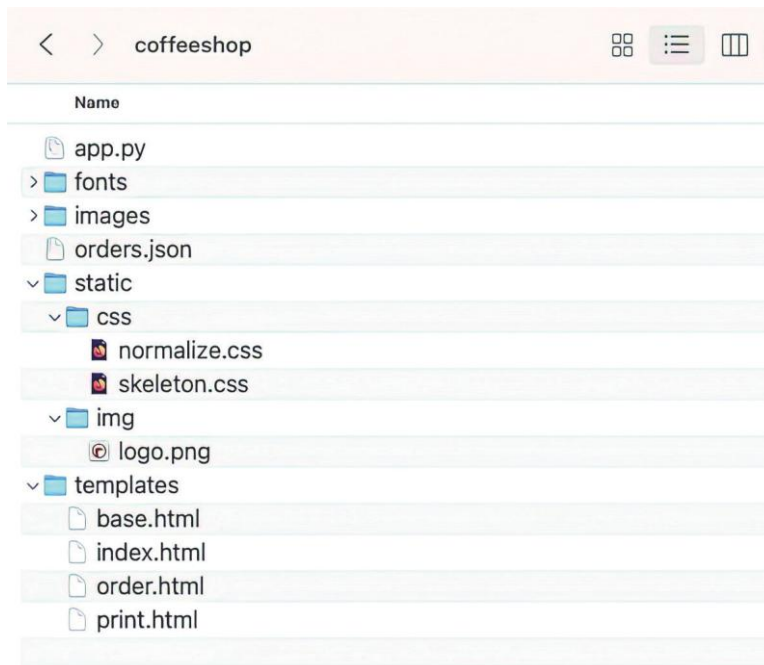
Саймон добавил еще одну диаграмму к той, которую он создал ранее.



«Вы знаете, как это сделать?» — спросил Саймон у Эмили и Эрика.

«Да», — ответила Эмили. Они начали работать над созданием папок и перемещением файлов. Через некоторое время они показали Саймону папку их проекта `coffeeshop`.

«Это хорошо выглядит?» — спросил Эрик.



**Ваш ход! Создайте папку своего приложения**

Создайте папку для своих проектов (вы можете использовать наименование `Projects` или какое-нибудь другое). Создайте еще одну папку для вашего приложения. Назовите ее в соответствии с вашим приложением — это может быть `icecream` или `pizzaplace`. Переместите файлы, связанные с веб-приложением, из `mu_code` в эту новую папку.

«Дай-ка я посмотрю», — сказал Саймон, глядя на их папку. — «Вы переместили свой каталог `static` со всем его содержимым и каталог `templates`, хорошо. Вы даже переместили `fonts` и `images` — мы их сейчас не используем, но ничего страшного».

«У меня вопрос», — сказал Эрик. — «Почему ты иногда говоришь «каталоги», а иногда «папки»? Думаю, они означают одно и то же, верно?»

«Извини, я не придерживаюсь строгой терминологии», — сказал Саймон. — «Ты прав, эти два слова означают одно и то же: что-то, что хранит файлы. Они используются для организации файлов, принадлежащих определенному проекту, например `coffeeshop`; либо они используются для файлов определенного типа, например `templates`. Исторически сложилось так, что в системах Unix и Linux они называются каталогами. MacOS основана на Unix, а в моем клубе робототехники мы используем Linux, поэтому я привык называть их каталогами. В Windows они называются папками — это одно и то же. Просто привыкните к разным терминам, это не имеет большого значения».

«Кроме того, вчера ты сказал, что мы можем хранить наши меню в текстовых файлах, как мы это делали в предыдущей версии», — сказал Эрик. — «Для менеджера кофейни, который не знает HTML, помнишь?»

«Спасибо, что напомнил мне, Эрик», — сказал Саймон. — «Сегодня я собираюсь показать вам, как работать с базой данных. Но это серьезная тема. Давайте сначала разберемся с этими файлами меню. Можете ли вы скопировать их в новый каталог из `mu_code`?»

«Да, секундочку», — сказал Эрик. Он скопировал файлы `drinks.txt`, `flavors.txt` и `toppings.txt` в папку проекта `coffeeshop`.

**Ваш ход! Переместите файлы меню**

Переместите файлы меню в новую папку, которую вы только что создали.

«Теперь взгляни на нашу старую программу и посмотри, как мы работали с этими файлами», — сказал Саймон. — «Открой файл `menu_files.py` и обнови состояние памяти. Попробуй объяснить это Эмили — это лучший способ что-либо лучше понять».

«Да, мне бы хотелось узнать, что вы делали с этой программой раньше», — сказала Эмили.

Эрик посмотрел на код и начал объяснять: «Мы создали функцию `read_menu()`, которая считывала текстовый файл с напитками, вкусами и



топингами и создавала список вариантов меню. Мы передавали этот список функции `menu()`. Функция `menu()` выводила список и предлагала покупателю выбрать напиток, затем вкус и топинг. После этого мы распечатывали заказ».

**Листинг 12.1** `menu_files.py`: прочитайте пункты меню из файлов

```
def
    read_menu(filename)
    e): f =
    open(filename)
    temp =
    f.readlines()
    result = []
    for item in temp:
        new_item = item.strip()
        result.append(new_item)

    return result

drinks = read_menu("drinks.txt")
flavors = read_menu("flavors.txt")
toppings =
read_menu("toppings.txt")

drink = menu(drinks)
flavor = menu(flavors, "Erik's flavors", "Choose your flavor: ")
topping = menu(toppings, "Erik's toppings", "Choose your topping: ")

print("Here is your order: ")
print("Main product: ", drink)
print("Flavor: ", flavor)
print("Topping: ", topping)
print("Thanks for your order!")
```

«Понятно», — сказала Эмили. — «Что такое *список*? Это похоже на *массив* в JavaScript?»

«Да, именно так», — подтвердил Саймон. — «Это то же самое: набор предметов, расставленных по порядку. В разных языках программирования это называется по-разному, но это одно и то же. Самое главное, что вы можете выбирать элементы по их положению в списке: например, элемент номер три или элемент номер пять. Такие числа называются *индексами* списка».

«Можем ли мы использовать эту функцию снова?» — спросил Эрик.

"Конечно!" — сказал Саймон. — «Вот для чего и создаются функции — для повторного использования! Давай скопируем эту функцию `read_menu()` в наш новый файл `app.py`. Поместите ее сразу после функции `load_orders()`. Также скопируйте три вызова функции, которые идут сразу после функции. Нам нужно создать эти списки и здесь».

Эрик скопировал функцию `read_menu()` и три строки после нее из `menu_files.py` в `app.py` в каталоге `coffeeshop`.

«Теперь у нас есть списки вариантов меню из следующих трех файлов: `drinks.txt`, `flavors.txt` и `toppings.txt`. Мы должны отобразить выбранные варианты на странице заказа. Хорошо, что наш файл `order.html` — это не простой HTML-файл, а *шаблон*. Мы можем использовать *переменные* в шаблоне и передавать *значения*, которые хотим отобразить. Помнишь, мы сделали это в шаблоне `print.html` когда передали словарь `new_order`. Позволь мне напомнить тебе. Вот код на Python для этого в `app.py`:

```
return render_template(
    "print.html",
    new_order=new_order
)
```

«А вот часть шаблона `order.html`, которая использует словарь `new_order` », — продолжил Саймон.

```
. . .
<h2>Here is your order:</h2>

<p>Drink: <strong>{{ new_order["drink"] }}</strong></p>
<p>Flavor: <strong>{{ new_order["flavor"] }}</strong></p>
<p>Topping: <strong>{{ new_order["topping"] }}</strong></p>
. . .
```

«Теперь нам следует использовать шаблон `order.html` и передавать ему списки с вариантами выбора. В шаблоне мы будем использовать их для построения меню. Найди вызов функции `render_template()`, которая использует шаблон `order.html`, и передайте эти три списка так же, как вы передали словарь `new_order` парой строк выше».

Эрик нашел эту строку (это была последняя строка программы) и изменил ее на следующую:

```
return render_template("order.html",    drinks=drinks,    flavors=flavors,
toppings=toppings)
```

«Хорошо», — сказал Саймон. — «Теперь, после того как мы передали списки в шаблон, нам нужно изменить шаблон. Эмили, ты будешь удивлена. Это не будет похоже на простой HTML. Мы будем использовать *циклы* в шаблоне».

«Как в Python?» — уточнил Эрик.

«Да, они выглядят почти одинаково и работают одинаково», — сказал Саймон. — «Вместо того, чтобы вывести параметры в нашем меню `select` в виде строк, мы возьмем их из списка. Наши циклы будут выглядеть как `for d in drinks` — очень похоже на то, что ты сделал в функции `menu()`. Я думаю, это потому, что язык шаблонов был создан тем же человеком, который создал Flask, — Армином Ронахером. Так что, как вы понимаете, он очень близок к Python. Язык шаблонов, кстати, называется *Jinja* — да, как ниндзя».

«Позвольте мне помочь вам с шаблоном `order.html` », — продолжил

Саймон. — «Я собрал несколько примеров, поэтому буду использовать их здесь».

«Значит, ты сам не знаешь всего?» — Эрик подмигнул брату.

«Конечно, нет», — серьезно ответил Саймон. — «Я сам учусь. Когда мне что-то нужно, я ищу в Интернете примеры и документацию и использую их. К счастью, документация Flask и Jinja написана хорошо и очень полезна. Кроме того, существует множество примеров и руководств, которыми поделились люди:»

Саймон изменил файл `templates/order.html` следующим образом:

**Листинг 12.2** `order.html`: Используйте списки меню для отображения пунктов меню в шаблоне

```
. . .
<form action="/order" method="post">
  <input type="text" name="name" placeholder="ENTER NAME">
  <select name="drink">
    <option value="">- Choose drink -</option>    ← Пункт инструкции
```

```

    {% for d in drinks -%}
    <option value="{{ d }}">{{ d | capitalize }}</option>
    {% endfor %}
</select>
<select name="flavor">
  <option value="">- Choose flavor </option>
  {% for f in flavors -%}
  <option value="{{ f }}">{{ f | capitalize }}</option>
  {% endfor %}
</select>
<select name="topping">
  <option value="">- Choose topping </option>
  {% for t in toppings -%}
  <option value="{{ t }}">{{ t | capitalize }}</option>
  {% endfor %}
</select>
<input type="submit" value="Submit">
</form>
. . .

```

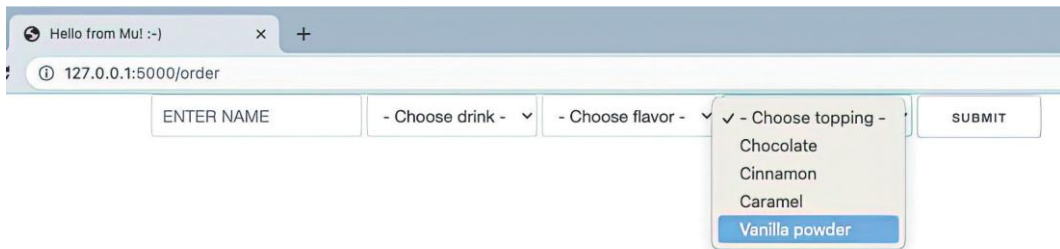
Цикл `for` включает пункты из списка в меню.

### Ваш ход! Измените `order.html`

Таким же образом измените шаблон `templates/order.html`. Будьте осторожны при копировании текста — обратите внимание, что в шаблонах мы используем **двойные** фигурные скобки. Не забудьте все тире и знаки процентов.

Они протестировали страницу, она работала так же, как и раньше. Саймон предложил: «Эмили, попробуй добавить что-нибудь в один из текстовых файлов меню и посмотри, обновится ли страница».

Эмили открыла файл `toppings.txt` и добавила `vanilla powder` в конец файла. Она сохранила файл и перезапустила приложение. Действительно, они увидели в меню новый пункт среди топингов.



### Ваш ход! Протестируйте свою страницу

Протестируйте свою страницу с новым шаблоном. Выглядит ли она так же, как раньше? Добавьте новый элемент в один из файлов с помощью текстового редактора. Протестируйте меню еще раз и проверьте, отображается ли новый элемент.

«Саймон, я вижу, что ты добавил в шаблон слово `capitalize` », — сказала она, — «и теперь там отображается `Vanilla powder` с заглавной буквы ‘V’. Довольно круто!»

«Да», — сказал Саймон, — «с помощью шаблонов можно сделать много интересных вещей. Это только одна из них. Если хочешь, мы можем использовать позже еще что-нибудь. Но теперь мы собираемся изучить базы данных!»

«Знаете, зачем нам нужны базы данных?» — спросил Саймон. — «Мы использовали файл `JSON` для хранения наших заказов, и он работал очень хорошо, верно? Но представьте, что мы хотим найти в нашем списке заказов, например, все заказы со вкусом ванили. Или все заказы от всех клиентов, которых зовут Алексей. Или узнать, использовали ли мы больше мятного аромата в этом месяце или в предыдущем. Мы могли бы написать такие функции сами, верно? Но оказывается, что подобные вопросы настолько распространены, что люди уже давно начали создавать специальные программы, которые не только хранят такие данные, как заказы и клиенты, но и могут отвечать на такие вопросы. Эти вопросы в мире баз данных обычно называются *запросами*. Много лет назад умные люди создали язык под названием *язык структурированных запросов* (*Structured Query Language*), или сокращенно *SQL*».

«Сегодня мы не собираемся изучать *SQL*», — продолжил Саймон, — «но причина, по которой я вам это рассказываю, заключается в том, что мы собираемся использовать базу данных под названием *SQLite* — если хотите, называйте ее эскулайт».

«Мы собираемся использовать для загрузки и хранения заказов вместо файла *JSON* базу данных?» — спросил Эрик.

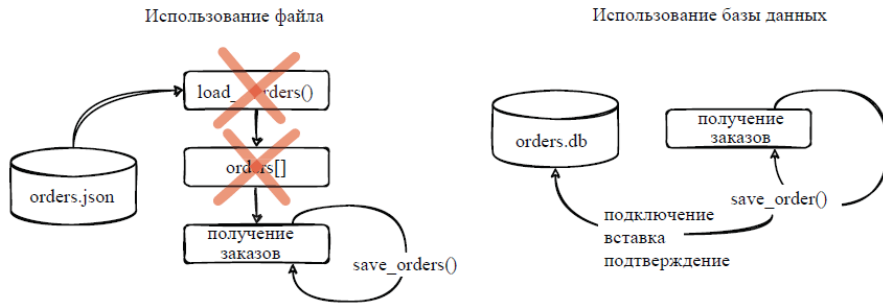
«Да», — ответил Саймон, — «но нам больше не нужно загружать заказы. Помнишь, когда мы закончили нам нужно было загрузить заказы в список, чтобы иметь возможность добавлять в этот список новые заказы и сохранять этот список в файле `JSON`? Теперь база данных будет хранить все наши заказы в собственном файле. База данных знает, как добавить новую запись в список заказов, поэтому нам не придется об этом беспокоиться».

«Мне это нравится», — сказал Эрик. — «Что нам теперь делать?»

Саймон начал рисовать диаграмму. «Мы будем использовать файл базы данных с именем `orders.db`. Нам больше не нужна функция `load_orders()`», — Он зачеркнул ее красным маркером.

«Нам не нужен список заказов — его сохранит база данных. Мы также перепишем нашу функцию `save_orders()`. Нам не нужно сохранять *все* заказы в файле каждый раз, когда у нас появляется новый заказ. Просто сохраним новый заказ в базе данных — все предыдущие заказы уже будут храниться там. Я предлагаю переименовать функцию в `save_order()`».

«Посмотрите на диаграмму и вы увидите разницу между нашим старым способом использования файла `JSON` и новым способом использования базы данных».



«Каждый раз, когда мы хотим сохранить новый заказ, мы создаем *соединение* с базой данных. Затем мы даём специальную команду для *вставки* нашего нового заказа в базу данных. После этого мы вызываем специальную функцию `commit()`, чтобы убедиться, что наш заказ сохранен в базе данных».

«Это похоже на то, что мы сделали с файлом», — сказал Эрик.

«Да, очень похоже», — согласился Саймон. — «Разница в том, что теперь мы обращаемся к базе данных, а не к файлу. Мы работаем над очень простой программой, но что, если бы в нашей кофейне было несколько бариста, принимающих заказы? Если несколько человек одновременно начнут писать в один и тот же файл, это создаст массу проблем. Когда мы работаем с базой данных, база данных позаботится об этом. Более того, если вы научитесь работать с SQLite, используя язык SQL, вы также сможете использовать этот язык со многими другими базами данных. Они прекрасно понимают SQL».

«Хватит теории, давайте напишем немного кода», — сказал Саймон. — «Прежде всего, нам следует создать таблицу в нашей базе данных. В нашем случае таблица довольно простая — в ней указано имя клиента и наши обычные опции: напиток, ароматизатор и топпинг».

Он начал редактировать файл `app.py` и объяснять. «Я добавил строку `import sqlite3` в начале. Затем я удалил вызов функции `orders = load_orders()` и всю функцию `load_orders()`. Непосредственно перед строкой `app = Flask()` я добавил три строки, которые устанавливают соединение с базой данных и создают таблицу под названием `orders`».

```

...
import sqlite3  <— Импортирует пакет sqlite3
...
con = sqlite3.connect("orders.db")  <— Подключается к базе данных
cur = con.cursor()
cur.execute(
    "CREATE TABLE IF NOT EXISTS orders(name, drink, flavor, topping);"  <—
)
app = Flask(name)  Создает таблицу заказов
...

```

### Ваш ход! Добавьте команды для подключения к базе данных и создания таблицы

Измените файл `app.py` и добавьте три строки, как это сделал Саймон. Не забудьте добавить `import sqlite3` в самом начале вашего файла `app.py`.

«Что такое `cur`?» — спросила Эмили.

«В мире баз данных это называется *курсором*. Как и курсор мыши, он может указывать на определенное место в базе данных. Мы не будем вдаваться в подробности — в данном случае я следую примерам из официальной документации Python. В документации сказано, что этот курсор нам нужен для выполнения команд SQL в базе данных.

«Посмотри на следующую строку с помощью `cur.execute()`. Это наша первая инструкция SQL. Она создает таблицу `orders` со *столбцами* `name`, `drink`, `flavor` и `topping`. В базах данных таблицы представляют собой обычные таблицы со строками и столбцами. Каждый новый заказ — это новая строка в таблице. В каждой строке у нас есть четыре столбца с именами, которые я уже упоминал. Легко, да?» — Он нарисовал простую таблицу.

	имя	напиток	вкус	топпинг
заказ 1				
заказ 2				
заказ 3				
...				

«Теперь давайте перепишем функцию `save_orders()`. Во-первых, мы переименуем ее в `save_order()`, поскольку мы собираемся сохранять за один раз только один заказ. Затем мы повторим две строки, которые я написал ранее, чтобы создать соединение и создать курсор. После этого мы можем выполнить команду SQL. В этом случае мы воспользуемся командой `INSERT` и передадим значения из словаря `order[]`. Наконец, мы вызовем функцию `commit()`, чтобы сохранить новый заказ в базе данных.

#### Листинг 12.3 `app.py`: Добавление заказа в таблицу

```
...
from flask import Flask, render_template, request

def save_order(order):
    con = sqlite3.connect("orders.db")
    cur = con.cursor()
    cur.execute(
        "INSERT INTO orders(name,drink,flavor,topping) VALUES(?,?,?,?);",
        (order["name"], order["drink"], order["flavor"], order["topping"]),
```

```

    )
    con.commit()
    return
. . .

```

### **Ваш ход! Создайте функцию `save_order()`**

Скопируйте функцию `save_order()` в свою программу. Обратите внимание на кавычки, квадратные скобки, запятые и круглые скобки.

«Я знаю, это выглядит сложно, но если вы просто прочитаете это, это довольно легко понять. Вы говорите базе данных вставить `INSERT` некоторые значения `VALUES` в базу данных `orders`. Затем вы указываете эти значения в скобках. Вы уже знакомы со словарями и знаете, как получить *значение* для определенного *ключа*».

«Почему ты написал некоторые слова заглавными буквами?» — спросил Эрик.

«Это соглашение в `SQL`. Люди используют все заглавные буквы для команд и ключевых слов `SQL`, таких как `INSERT` и `INTO`. Для переменных, таких как наименование таблиц `orders`, люди используют строчные буквы. Я тоже пробовал использовать строчные буквы для команд, и это сработало. Таким образом, заглавные буквы — это просто соглашение о читабельности, позволяющее отделять слова `SQL` от переменных».

«Давай попробуем!» — сказала Эмили.

"Минуту!" — сказал Саймон. — «Я чуть не забыл — нам нужно удалить вызов `load_orders()` из функции `order()`. А также нам следует переименовать функцию сохранения».

Он отредактировал функцию `order()` так, чтобы она выглядела следующим образом:

#### **Листинг 12.4 app.py: Удаление `load_orders()`**

```

@app.route("/order", methods=("GET", "POST"))
def order():
    if request.method == "POST":
        new_order = {"name": request.form["name"],
                     "drink": request.form["drink"],
                     "flavor": request.form["flavor"],
                     "topping": request.form["topping"]}
        save_order(new_order)
        return render_template(
            "print.html", new_order=new_order
        )

    return render_template("order.html", drinks=drinks,
                          flavors=flavors, toppings=toppings)

```

### **Ваш ход! Отредактируйте функцию `order()`**

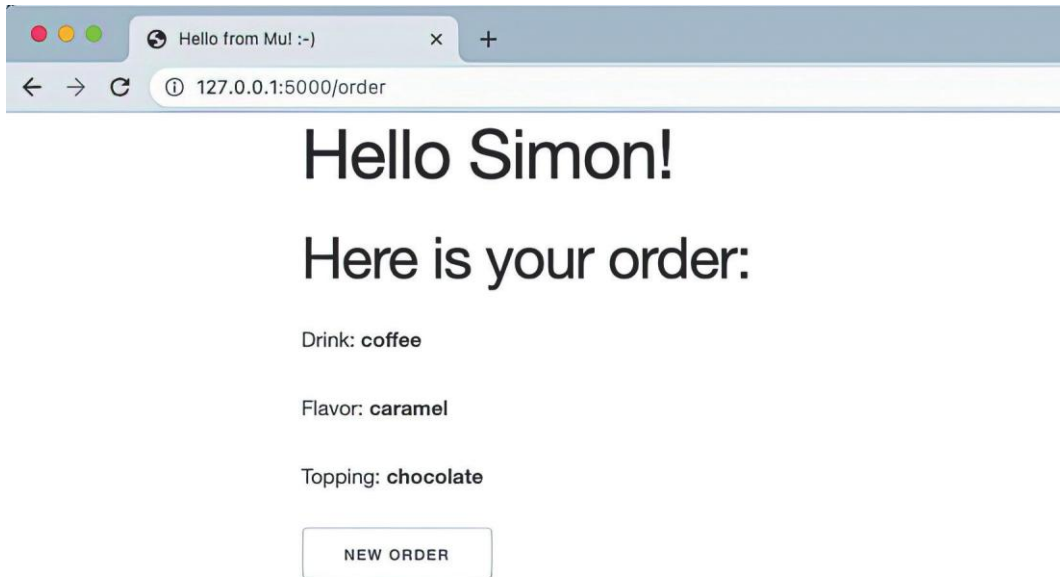
Отредактируйте функцию `order()` так же, как это сделал Саймон. Не забудьте



изменить `save orders()` (множественное число) на `save_order()` (единственное число).

Он сохранил файл и сказал: «Теперь все готово. Эмили, вперед! Но позволь мне заполнить заказ, хорошо?»

Эмили нажала Run и перезагрузила страницу заказа. Саймон вошел в меню.



### **Ваш ход! Выполните тестирование**

Протестируйте свою программу. Теперь она использует базу данных вместо файла JSON, но интерфейс при этом должен выглядеть прежним.

«Она работает!» — сказал Саймон со вздохом облегчения. Он и сам не был уверен, что все сделал правильно, но не хотел этого показывать.

«Можем ли мы проверить, сохранился ли заказ в базе данных, как мы это делали с файлом JSON?» — спросил Эрик.

«Хороший вопрос!» — сказал Саймон. — «Нет, файл базы данных — это не текстовый файл, который мы можем прочитать. Но мы можем добиться большего. Мы можем создать новую функцию, которая будет показывать нам все заказы на веб-странице».

«Это было бы прекрасно!» — сказал Эрик. — «Например, если менеджер магазина хочет узнать, сколько порций карамели мы сегодня израсходовали».

«И чтобы мы могли видеть, что конкретный клиент любит заказывать», — добавила Эмили.

«Отличные идеи», — сказал Саймон. — «Готовы еще немного поработать?»

Нам понадобятся три вещи. Во-первых, помните, что для открытия страницы заказа мы добавляем `/order` к адресу сайта. Во Flask это называется *маршрутом*. Нам следует создать функцию для нового маршрута, которую мы сможем использовать для вывода списка заказов».

«Давайте назовем этот маршрут `/list`. Функция для этого маршрута также будет называться `list()`. Это очень похоже на функцию `order()`. В конце этой функции `list()` мы также вызовем функцию `render_template()` для отображения списка заказов».

«Во-вторых», — продолжил он, — «мы создадим функцию, которая будет обращаться к базе данных и возвращать список заказов в функцию `list()`. Назовем ее `get_orders()`».

«И в-третьих, нам следует создать шаблон для отображения заказов. Эмили, ты научилась создавать таблицы на HTML?»

«Таблицы? Да, я могу их создать», — сказала Эмили. «Мы должны использовать тег `<table>` для создания таблицы, затем тег `<tr>` для каждой строки и тег `<td>` для каждой ячейки в строке».

"Великолепно!" — сказал Саймон. — «Тогда ты создашь шаблон для нашей функции `list()`. Эрик, ты напишешь функции, ладно?»

«Конечно», — ответил Эрик. — «С чего мне начать?»

«Сразу после функции `save_order()`. Создай функцию `get_orders()`. Начни с той же первой строки, что и в функции `save_order()`. Добавь после нее еще одну строку: `con.row_factory = sqlite3.Row`. Это специальная инструкция, которая позволит нам использовать наименования столбцов при работе с результатом. Не беспокойся об этом сейчас — ты увидишь это позже. Затем создай курсор той же строкой, что и раньше: `cur = con.cursor()`. Дай мне посмотреть твой код».

Эрик показал то, что он написал:

```
def get_orders():
    con = sqlite3.connect("orders.db")
    con.row_factory = sqlite3.Row
    cur = con.cursor()
```

«Хорошо», — сказал Саймон. — «Теперь важная часть: нам нужно получить все заказы из таблицы `orders`. Мы будем использовать одну из наиболее распространенных инструкций SQL, которая приказывает базе данных предоставить нам все строки и все столбцы из таблицы. Выглядит это так: `SELECT * FROM orders`; Звездочка `*` означает «все».

«Мы должны использовать эту инструкцию с той же функцией `cur.execute()`, которую мы использовали в `save_order()`. После выполнения инструкции результаты должны быть получены из базы данных. Тебе придется сохранить результаты в списке, называемом `rows`, с помощью функции `cur.fetchall()`. Затем ты вернешь список строк из этой функции».

«Не волнуйся, вот страница документации, с которой я это взял. Используй ее как пример», — Саймон показал вкладку, открытую в браузере: <https://docs.python.org/3.8/library/sqlite3.html>.

Эрик начал печатать. Через несколько минут он показал свой код Саймону:

**Листинг 12.5** app.py: Функция `get_orders()`

```
def get_orders():
    con = sqlite3.connect("orders.db")
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    cur.execute("SELECT * FROM orders;")
    rows = cur.fetchall()

    return rows
```

**Ваш ход!** Создайте фнкцию `get_orders()`

Создайте функцию `get_orders()`, как только что это сделал Эрик.

«Мне кажется, неплохо», — сказал Саймон. — «Посмотрим, когда проверим».

«Теперь нам нужно добавить еще одну функцию, которая обрабатывает маршрут `/list`. Мы должны добавить ее после функции `order()`, которая обрабатывает `/order`. Давай я тебе помогу.» — Он начал писать функцию в конце файла.

«Сначала я указываю маршрут, который мы собираемся обрабатывать, с помощью `@app.route()`. Затем я определяю функцию обычным способом. Сама функция довольно проста: я просто вызываю `get_orders()` и передаю результаты в `render_template()`. Как вы помните, эта функция отображает данные в браузере, используя шаблон, который мы ей передаем».

**Листинг 12.6** app.py: Создание функцию `list()`

```
@app.route("/list", methods=["GET"])
def list():
    orders = get_orders()

    return render_template("list.html", orders=orders)
```

**Ваш ход! Создайте функцию `list()`**

Создайте функцию `list()`, как только что это сделал Эрик.

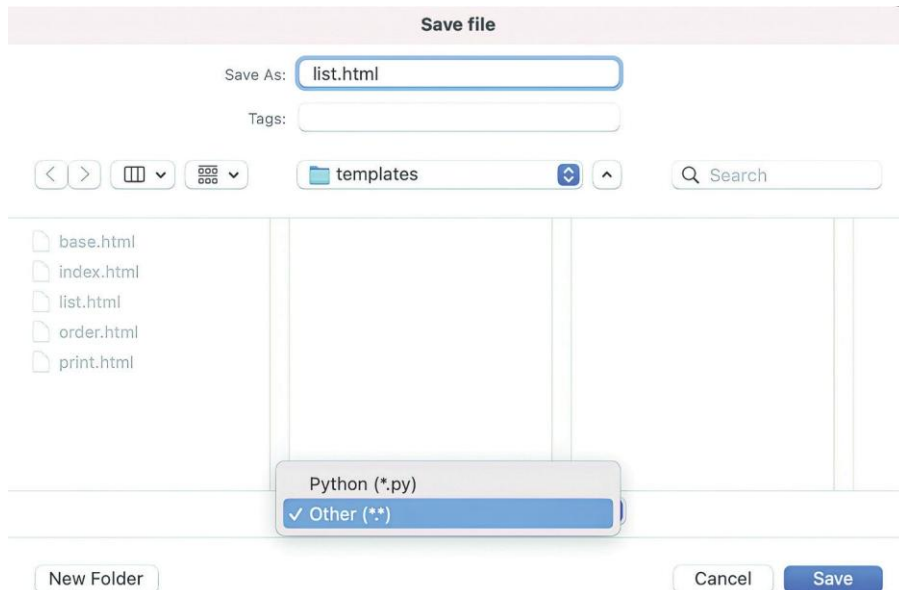
«Почему мы не получили все заказы из базы данных прямо в этой функции?» — спросил Эрик. — «Почему ты попросил меня написать отдельную функцию?»

«Хороший вопрос», — сказал Саймон. — «Я пытаюсь отделить функции, которые работают с *пользователем*, от функций, которые работают с *данными*. Иногда люди называют это *фронтом* и *бэкендом*. Если мы хотим изменить способ работы с данными, нам нужно изменить функцию `get_orders()`. Например, предположим, что мы хотим иметь возможность получать только заказы со вкусом карамели. Мы передадим этот запрос функции, и она вернет список заказов так же, как и раньше. Наша функция `list()` будет отображать все, что она получит от `get_orders()`. Ей неважно, будет ли это полный список заказов или только часть заказов — она их просто отображает. С другой стороны, функцию `get_orders()` не волнует, как будут отображаться заказы. Она просто возвращает список заказов».

«Эту идею в информатике обычно называют *разделением ответственности*. Я дам вам статью из Википедии об этом, чтобы вы прочитали ее позже.

**ПРИМЕЧАНИЕ** Статья из Википедии о разделении ответственности находится здесь: [https://en.wikipedia.org/wiki/Separation\\_of\\_concerns](https://en.wikipedia.org/wiki/Separation_of_concerns).

«Остался только шаблон списка», — сказал Саймон. — «Эмили, давай создадим новый файл в редакторе. Удали пример веб-приложения, созданного для нас Ми. Сохрани новый файл `list.html` в `templates`. Не забудь изменить тип файла с `Python (*.py)` на `Other (*.*)`».



«Начни с тех же строк шаблона, которые мы использовали в шаблоне `print.html`. Первые две строки и последняя строка должны быть одинаковыми».

Эмили скопировала эти три строчки:

```
{% extends "base.html" %}
{% block content %}

{% endblock %}
```

«Теперь создай таблицу с одной пустой строкой и пустыми ячейками. У нас будет четыре ячейки в каждой строке и четыре заголовка столбца. Это будет первый ряд. Ты знаете, как создавать заголовки столбцов?» — спросил Саймон у Эмили.

«Да, с помощью тега `<th>` », — ответила она. Она начала работать над таблицей HTML в файле `templates/list.html`.

«Теперь ты можешь добавить заголовки», — сказал Саймон. — «Это будут `Name, Drink, Flavor` и `Topping`».

#### Листинг 12.7 The `templates/list.html` template with headers

```
{% extends "base.html" %}
{% block content %}

<table>
  <tr>
    <th>Name</th>
    <th>Drink</th>
    <th>Flavor</th>
    <th>Topping</th>
  </tr>
  <tr>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
  </tr>
</table>
{% endblock %}
```

«Отлично», — сказал Саймон. — «Теперь позвольте мне добавить цикл. Здесь мы будем использовать тот же цикл `for`, который мы использовали для отображения меню. Помните, что мы передаем в этот шаблон список `orders`. Цикл будет перебирать этот список, например: `for o in orders`. В каждую ячейку между `<td>` и `</td>` мы вставим значение для каждого из ключей: `name, drink, flavor` и `topping`», — Саймон изменил шаблон:

**Листинг 12.8** Шаблон `templates/list.html` с циклом `for`

```
{% extends "base.html" %}
{% block content %}

<table>
  <tr>
    <th>Name</th>
    <th>Drink</th>
    <th>Flavor</th>
    <th>Topping</th>
  </tr>
  {% for o in orders -%}
  <tr>
    <td>{{ o['name'] }}</td>
    <td>{{ o['drink'] }}</td>
    <td>{{ o['flavor'] }}</td>
    <td>{{ o['topping'] }}</td>
  </tr>
  {% endfor %}
</table>
{% endblock %}
```

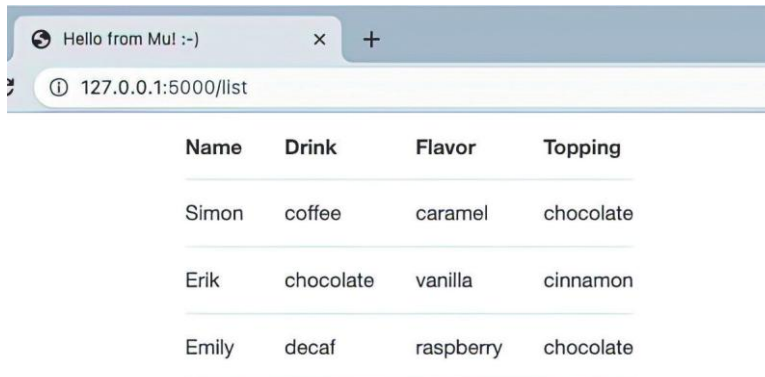
**Ваш ход! Создайте шаблон списка**

Создайте файл `templates/list.html`, как это только что сделала Эмили. Обратите внимание на кавычки, квадратные и двойные фигурные скобки.

«Пришло время попробовать функцию списка!» — сказал Саймон после того, как сохранил файл. — «Запусти приложение и оформи два-три заказа».

Эрик запустил приложение, открыл вкладку с адресом `127.0.0.1:5000/order` и оформил три заказа: для Саймона, для Эмили и для себя.

«Теперь измени маршрут `/order` в адресе на `/list`», — сказал Саймон. Эрик сделал это, и они увидели результат.



The screenshot shows a web browser window with the title "Hello from Mul :-)" and the address bar displaying "127.0.0.1:5000/list". Below the browser window, a table displays the following data:

Name	Drink	Flavor	Topping
Simon	coffee	caramel	chocolate
Erik	chocolate	vanilla	cinnamon
Emily	decaf	raspberry	chocolate

«Выглядит великолепно», — сказала Эмили. — «Очень полезно для менеджера магазина!»

«Да, мне это тоже нравится», — сказал Эрик. — «Я думаю, нам следует добавить в строки цифры, чтобы увидеть, какое количество людей мы обслужили».

«Отличная идея!» — сказал Саймон. — «И ты можешь добавить даты и время, чтобы увидеть, сколько клиентов вы обслуживали каждый день. Все это возможно — нам просто нужно изменить нашу базу данных, чтобы сохранить эту информацию. Но мы сделаем это позже. На завтра у меня есть еще одна идея. Эмили, ты случайно не знаешь CSS?»

«Мы кое-чему научились на занятиях», — ответила Эмили.

«Что такое CSS?» — спросил Эрик.

«Это способ изменить внешний вид вашей HTML-страницы в браузере», — объяснила Эмили. — «Я думаю, это означает *каскадные таблицы стилей*».

«Совершенно верно», — сказал Саймон. — «Завтра давайте поработаем над тем, чтобы наш сайт благодаря CSS выглядел еще лучше. Но прежде чем вы уйдете, давайте подведем итоги того, что мы сделали сегодня».

Эмили начала: «Мы создали новый каталог проекта и переместили в него весь наш контент. Мы переименовали основной файл приложения в `app.py`».

«Мы использовали текстовые файлы для пунктов меню так же, как и в предыдущей версии приложения», — добавил Эрик. — «Теперь менеджер магазина может просто редактировать эти файлы вместо того, чтобы менять код программы».

«А потом мы начали использовать базу данных», — сказала Эмили. — «Это не легко».

«Да, я согласен», — сказал Саймон. — «Но оно того стоит. С базой данных наша программа работает более профессионально. Теперь пришло время сделать ее более красивой. Мы поработаем над этим завтра».

## Новое, что вы узнали сегодня

- *Папки и каталоги* — В разных операционных системах люди используют разные термины для одного и того же. Папки и каталоги представляют собой коллекции файлов, принадлежащих определенному проекту, или файлов определенного типа.
- *Шаблоны Jinja* — Шаблоны Jinja используются Flask для создания HTML-страниц. Вы можете передавать переменные, списки и словари в шаблоны и использовать их так же, как в Python. Вы можете использовать циклы `for` в шаблонах, аналогично Python.
- *База данных* — База данных — это программа, которая может хранить данные и выполнять с ними различные операции. Базы данных могут выполнять поиск записей по определенным критериям или суммировать данные (например, количество заказов на ванильный вкус за этот месяц).

- *SQL* — SQL означает язык структурированных запросов и используется для работы с базами данных.
- *Фронтенд и бэкенд* — Фронтенд-часть приложения работает с пользователями: отображает страницы и получает ввод с клавиатуры и из меню. Бэкенд-часть работает с данными: хранит и извлекает данные из файлов и баз данных. Интерфейсная и серверная части используются для разделения задач при работе над большими приложениями.



### *Код для этой главы*

Вы можете найти код для этой главы здесь: <https://github.com/pavelanni/pythonicadventure-code/tree/main/ch12>.

# 13 Стили: делаем красиво

## В этой главе

- Эрик и Эмили экспериментируют со стилями, цветами и шрифтами
- Эрик и Эмили создают новую домашнюю страницу кофейни
- Эрик и Эмили добавляют фотографии на свои страницы

«Давайте сделаем программу красивой!» — сказал Саймон, когда друзья встретились в следующий раз.

«Ты имеешь в виду использование стилей для наших HTML-файлов?» — спросила Эмили. — «Мы использовали стили, когда мы изучали HTML. Ты можешь изменить цвета, размеры шрифта и фон».

Саймон сказал: «Да, давайте поговорим о стилях. Эмили, можешь немного объяснить?»

«Это называется CSS. И это означает . . .» — Она на мгновение остановилась, вспоминая. — «Это означает *каскадные таблицы стилей*, верно?» — и она посмотрела на Саймона. — «Но, честно говоря, я не помню почему».

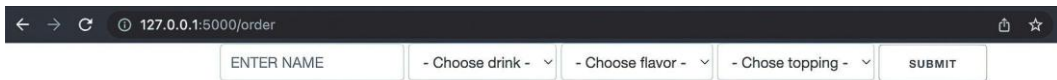
"Ты права!" — сказал Саймон. «Это каскадные таблицы стилей! Они называются *таблицами стилей*, потому что представляют собой обычные текстовые файлы, описывающие стили. Стили определяют, как ваши HTML-элементы будут выглядеть на странице. Например, вы можете сказать, что ваши заголовки `<h2>` должны иметь размер 36 пикселей и быть красными. Либо им следует использовать определенный шрифт».

Он продолжил: «Одна из идей веб-дизайна — отделение контента от стилей. Конечно, мы могли бы установить стиль в HTML-файле. Но это не очень удобно. Если вы хотите изменить размер или цвет шрифта, вам придется просмотреть *все* ваши HTML-файлы и изменить *все* заголовки `<h2>`. Вместо этого мы определяем стили в отдельном файле, где говорим, что все заголовки `<h2>` должны использовать данный цвет и размер. Тогда, если мы захотим их изменить, нам нужно произвести изменение только в одном месте. Очень удобно, да?»

«Понятно», — сказал Эрик. — «Это похоже на функции в Python. Вы пишете код один раз, а затем используете его в других местах».

«Да, верно!» — сказал Саймон. — «Мне нравится твоя аналогия. Итак, мы разобрались, почему их называют таблицами стилей, но почему *каскадными*? Для своего веб-сайта или приложения ты можешь иметь несколько таблиц стилей и комбинировать их одну за другой. Часто люди используют таблицу стилей, созданную кем-то другим, и добавляют или изменяют только те элементы, которые хотят изменить. Это еще одно сходство с функциями».

Он продолжил: «В нашем проекте мы будем использовать таблицы стилей, включенные в редактор Му, и немного изменим их, чтобы сделать их более подходящими для нашего проекта кофейни. Взгляните на страницу заказа и подумайте, что нам следует изменить».



Эрик посмотрел на экран и сказал: «Нам нужно название. Как в нашем терминальном приложении: `WelcometoErik'sCoffeeShop!`»

«Постой!» — сказала Эмили. — «Это больше не просто твоя кофейня. Я думаю, она должна называться `Erik and Emily's Coffee Shop`. Или сокращенно `E&E Coffee Shop`. Согласен?»

«Я думаю, что теперь это командный проект, и `E&E` — отличное название», — сказал Саймон.

«Да, давайте назовем кофейню `E&E Coffee Shop`», — согласился Эрик.

«Позвольте мне добавить это», — сказала Эмили. Она открыла файл `order.html` в папке `templates`. Она добавила заголовок, поэтому начало файла теперь выглядело так:

```
{% extends "base.html" %}
{% block content %}

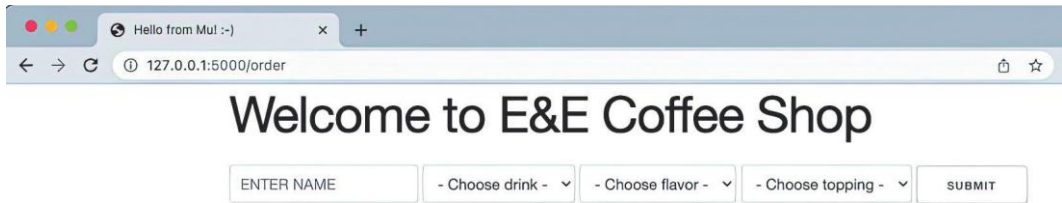
<h1>Welcome to E&E Coffee Shop</h1>

<form action="/order" method="post">
```

```
<input type="text" name="name" placeholder="ENTER NAME">
```

. . .

Эмили сохранила файл, перешла на вкладку `app.py` в редакторе Mu и нажала Run. Она открыла вкладку браузера и снова ввела URL: `http://127.0.0.1:5000/order`.



«Выглядит неплохо!» — оценил Эрик.

«Саймон, ты сказал, что мы можем изменить шрифт в названии», — напомнила Эмили. — «Сейчас это выглядит немного... . . скучно. Я думаю, нам нужно что-то более необычное».

«Я согласен», — сказал Саймон. — «К счастью, существует множество веб-шрифтов, которые мы можем использовать бесплатно. Некоторые шрифты стоят денег, но для данного проекта вполне подойдут бесплатные шрифты. Например, Google Fonts — хороший ресурс со шрифтами».

Он открыл новую вкладку в браузере и набрал адрес: `fonts.google.com`. «Как видите, на выбор предлагается более 1400 шрифтов. Вы также можете найти шрифты для разных языков. Видите это поле с надписью ‘Type something’? Введите заголовок страницы, и вы увидите, как он выглядит при отображении разными шрифтами».

«А как насчет ‘Categories’? Что это значит?» — спросила Эмили.

«Если вы нажмете это меню, вы увидите Serif, Sans Serif, Display, Handwriting и Monospace. Обычно моноширинный шрифт используется для программного кода. Вы можете использовать Handwriting, если хотите распечатать поздравительную открытку для друга. Display означает, что этот шрифт используется только для компьютерных дисплеев, а не для печати. Serif и Sans Serif звучат странно, но на самом деле они представляют два класса шрифтов, с которыми вы хорошо знакомы. Serif означает, что шрифт имеет небольшие штрихи (засечки) на концах букв. Конечно, вы знаете Times New Roman — шрифт по умолчанию для большинства программ и документов. Попробуйте выбрать в меню Categories только Serif, и вы поймете, что я имею в виду».

«Мне нравится этот Playfair Display!» — сказала Эмили.

«Хорошо!» — сказал Саймон. — «Пометь это, мы вернемся к этому позже. Я просто хотел показать вам, как выглядят шрифты без засечек. ‘Sans’ означает «без»; в данном случае это означает, что в шрифтах нет тех засечек, о которых я упоминал ранее. Скорее всего, вы видели Arial из этой категории. Посмотрите сюда», — и он изменил Category на Sans Serif.

«Подобные шрифты используются в меню и других элементах интерфейса приложений. Просто посмотрите вокруг, и вы начнете замечать и узнавать эти две категории шрифтов повсюду в Интернете и в приложениях. Если вы хотите узнать больше о шрифтах, перейдите на вкладку Knowledge. Это очень интересно, поверьте!»

***Ваш ход! Узнайте больше о Google Fonts***

Откройте страницу [fonts.google.com](https://fonts.google.com) и посмотрите. Попробуйте найти интересные шрифты и выполнить поиск по разным категориям. Обратите внимание на то, что вы можете сортировать результаты несколькими способами: Trending (шрифты в тренде), Most Popular (наиболее популярные шрифты), Newest (новые шрифты) и т.д. Откройте свои любимые сайты и попытайтесь выяснить, где они используют шрифты с засечками, без засечек и моноширинные шрифты.

Google Fonts

Fonts Icons Knowledge FAQ

Search fonts Custom Welcome to E&E Coffee Shop 40px

Serif Language Font properties Show only variable fonts Show only color fonts

290 of 1474 families About these results Sort by: Trending

**Noto Serif Nyiakeng Puachue Hmong** Variable  
Google  
Welcome to E&E Coffee Shop

**Roboto Slab** Variable  
Christian Robertson  
Welcome to E&E Coffee Shop

**Playfair Display** Variable  
Claus Eggers Sørensen  
Welcome to E&E Coffee Shop

**Merriweather** 8 styles  
Sorkin Type  
Welcome to E&E Coffee Shop

**Lora** Variable  
Cyreal  
Welcome to E&E Coffee Shop

**PT Serif** 4 styles  
ParaType  
Welcome to E&E Coffee Shop

Google Fonts

Fonts Icons Knowledge FAQ

Search fonts Custom Welcome to E&E Coffee Shop 40px

Sans Serif Language Font properties Show only variable fonts Show only color fonts

546 of 1474 families About these results Sort by: Trending

**Roboto** 12 styles  
Christian Robertson  
Welcome to E&E Coffee Shop

**Open Sans** Variable  
Steve Matteson  
Welcome to E&E Coffee Shop

**Noto Sans Japanese** 6 styles  
Google  
Welcome to E&E Coffee Shop

**Montserrat** Variable  
Julieta Ulanovsky, Sol Matas, Juan Pablo del Peral, Jacques Le Bailly  
Welcome to E&E Coffee Shop

**Lato** 10 styles  
Łukasz Dziedzic  
Welcome to E&E Coffee Shop

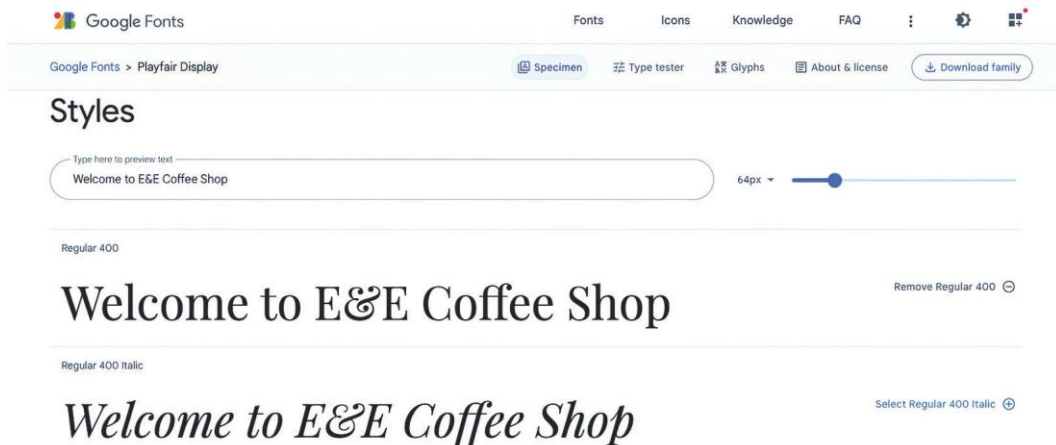
**Poppins** 18 styles  
Indian Type Foundry, Jonny Pinhorn  
Welcome to E&E Coffee Shop

«Эти шрифты выглядят красиво, но мне нравится шрифт Playfair. Я думаю, он будет хорош для заголовка нашей страницы. Эрик, что ты думаешь?» — спросила Эмили.

«Да, мне это тоже нравится. Это не скучно», — сказал Эрик. — «И мне нравится, как он отображает знак &. Да, это хорошо для заголовка».

Саймон снова переключился на категорию Serif и щелкнул шрифт Playfair Display. Открылась Страница шрифтов. Саймон объяснил: «Здесь вы можете найти информацию о том, как использовать этот шрифт на вашей веб-странице или в приложении».

Он прокручивал страницу до тех пор, пока заголовков, который они использовали, не появился чуть ниже заголовка Styles. «Здесь вы можете выбрать одну из толщин шрифта. Как видите, один и тот же шрифт, или, лучше сказать, *гарнитура*, может иметь разную «жирность». Я выбрал только один из них, и это даст нам необходимую информацию. Обратите внимание на маленькую красную точку в правом верхнем углу страницы. Это означает, что мы выбрали что-то для нашей коллекции, и нам нужно щелкнуть по ней, чтобы открыть инструкции».



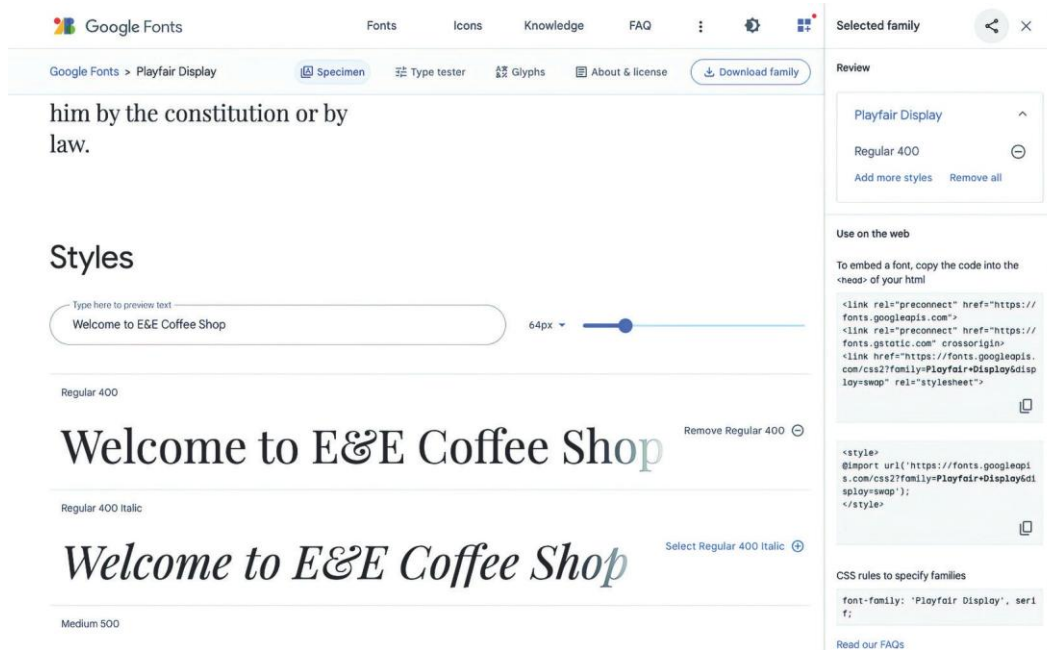
Он щелкнул значок с красной точкой, и открылись необходимые инструкции.

«Посмотрите на блок справа, в разделе Use On the Web (Использование в Интернете). Google предоставляет нам полезные инструкции по использованию данного шрифта. Нам просто нужно скопировать эти строки в наши файлы. Из первых двух фрагментов кода рекомендуется использовать первый, с `<link>`. Третий фрагмент кода, называемый CSS Rules to Specify Families (Правила CSS для указания семейств), демонстрирует, как мы можем использовать данный шрифт в наших файлах CSS. Это может показаться сложным, но не волнуйтесь, я вам помогу».

«Хорошая особенность Flask — то, что мы используем шаблоны», —

продолжил Саймон. — «Посмотрите на свой файл `order.html`: вы увидите, что он начинается с `{% extends "base.html" %}`. Ваш файл `print.html` также начинается с этой строки. Это означает, что мы *повторно используем* `base.html` в наших шаблонах. Нам не нужно менять каждый файл в нашем проекте — нам просто нужно изменить `base.html`. Да, Эрик, это снова похоже на функции», — и Саймон улыбнулся.

«Эмили, не могла бы ты открыть файл `base.html` в каталоге `templates`?»



Когда Эмили открыла файл, Саймон сказал: «Посмотрите на веб-страницу Google Fonts с инструкциями. Скопируйте три строки, начинающиеся с `<link>`, из серого поля. Затем вставьте их в `base.html` перед закрывающим тегом `</head>`».

Эмили скопировала эту строку в файл `base.html`, и теперь он выглядел следующим образом (показываю только часть файла):

### Листинг 13.1 `templates/base.html`: Добавление шрифта

```
...
<meta name="viewport" content="width=device-width, initial-scale=1">

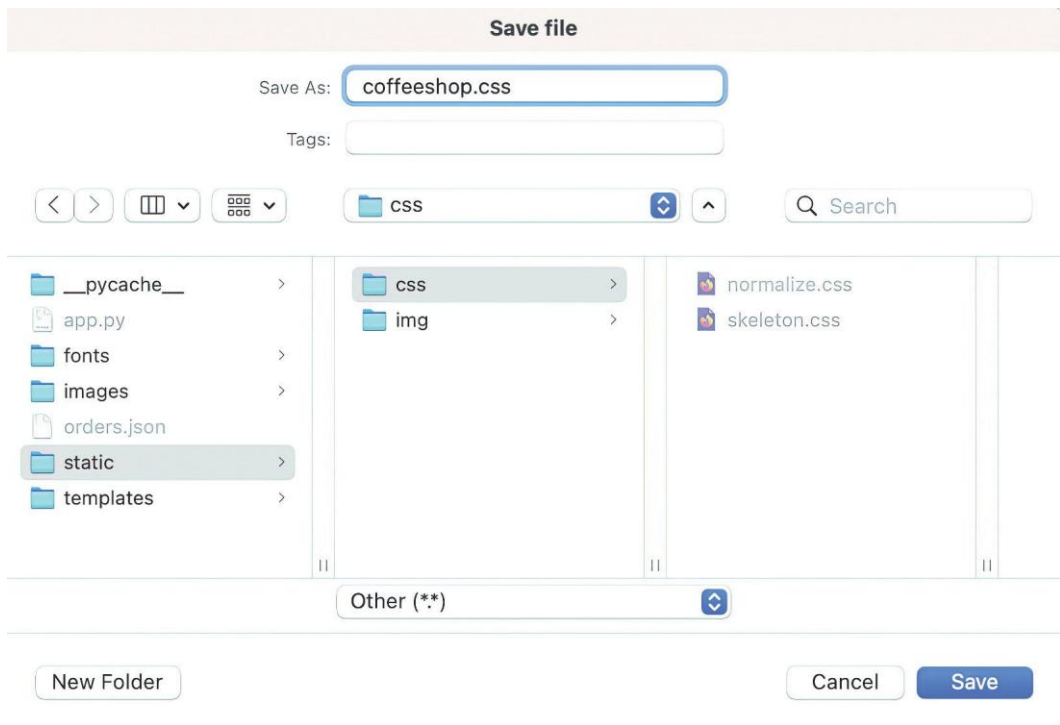
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Playfair+Display&display=swap" rel="stylesheet">
</head>
<body>
...
```



**Ваш ход! Добавьте ссылку на шрифт в `base.html`**

Откройте файл `templates/base.html` в редакторе и внесите изменения, которые только что внесла Эмили.

"Хорошо. Это был шаг 1», — сказал Саймон. — «Шаг 2 — создадим собственную таблицу стилей. Нажмите New (Создать) в редакторе. Будет создан пример веб-приложения, но сейчас он нам не нужен. Удалите из него весь текст и нажмите Save (Сохранить). Вам следует сохранить его в том же каталоге, где хранятся другие файлы CSS, под именем `static/css`. Назовите этот файл `coffeeshop.css`. Не забудьте изменить расширение файла с 'Python (\*.py)' на 'Other (\*.\*)'».



Саймон проверил, что файл `coffeeshop.css` появился рядом с другими файлами CSS. «Хорошо», — сказал он. «Теперь мы можем добавить характеристику шрифта в заголовок `h1`.

«Посмотрите на поле CSS Rules to Specify Families (Правила CSS для указания семейств) на странице шрифтов в браузере. Выделите и скопируйте текст оттуда. Теперь введите `h1 {` и вставьте то, что вы скопировали, после открывающей фигурной скобки. Затем добавьте закрывающую фигурную скобку и сохраните файл».

Это было легко. Эмили отредактировала файл `coffeeshop.css`, чтобы он выглядел так:

**Листинг 13.2** `static/css/coffeeshop.css`: Добавление шрифта в CSS

```
h1 {font-family: 'Playfair Display', serif;}
```

**Ваш ход! Создайте** `coffeeshop.css`

Создайте в редакторе файл `static/css/coffeeshop.css` и добавьте строку для стиля `h1`.

«Великолепно!» — сказал Саймон. — «Теперь у нас есть собственный CSS-файл. Нам просто нужно сказать Flask, что мы хотим его использовать. Это шаг 3. Эмили, полагаю, у тебя все еще открыт файл `base.html` в редакторе? Найди две строки со словом `stylesheet`. Ты узнаешь наименования файлов».

«Да, я их вижу», — сказала Эмили. «Они называются `normalize.css` и `skeleton.css` — те же имена, которые я видела, когда сохраняла нашу таблицу стилей».

«Хорошо, теперь скопируй строку со `skeleton.css` и вставь ее сразу за ней. У тебя должны получиться две абсолютно одинаковые строки. Теперь во второй из них замени `skeleton` на `coffeeshop`».

Эмили так и сделала, и теперь файл `base.html` стал выглядеть так:

**Листинг 13.3** `templates/base.html`: Добавление CSS-файла `coffeeshop`

```
. . .
<!-- CSS
----->
<link rel="stylesheet" href="/static/css/normalize.css">
<link rel="stylesheet" href="/static/css/skeleton.css">
<link rel="stylesheet" href="/static/css/coffeeshop.css">
. . .
```

**Ваш ход! Добавьте таблицу стилей** `coffeeshop.css` **в** `base.html`

Откройте файл `templates/base.html` в редакторе и внесите изменения, которые только что внесла Эмили.

«Теперь вы можете понять, почему эти таблицы стилей называются *каскадными*. У нас есть три таблицы стилей. Первая, называемая `normalize.css`, необходима для того, чтобы наш веб-сайт хорошо выглядел во всех типах браузеров и на разных устройствах».

«Даже на моем iPad?» — спросил Эрик.

«Да, так должно быть», — ответил Саймон. — «Мы проверим это позже».

Он продолжил: «Следующий CSS-файл называется `skeleton.css`. Он был включен в редактор Ми и обеспечивает красивые стили для всех элементов HTML. Нам не нужно создавать стили для отдельных элементов — они и так хорошо выглядят. Нам нужно добавлять стили в наш собственный CSS-файл только в том

случае, если мы хотим что-то изменить, например семейство шрифтов для заголовков. Стили применяются один за другим в том порядке, в котором CSS-файлы перечислены в файле HTML. Поэтому их и называют каскадными.

**ПРИМЕЧАНИЕ** Вы можете прочитать больше про `normalize.css` здесь: <https://nicolasgallagher.com/about-normalize-css/>. Таблица стилей `skeleton.css` описана здесь: <http://getskeleton.com/>.

«Хватит теории, давай вернемся к нашему сайту!» — сказал Саймон.

Эмили и Эрик еще раз проверили, все ли файлы сохранены. Затем они переключились на вкладку `app.py` и нажали Run. Они открыли вкладку браузера со знакомым адресом `http://127.0.0.1:5000/order` и увидели обновленную страницу.



«Ух ты! Мне это нравится!» — воскликнул Эрик.

«Теперь вы знаете, как изменить любой шрифт на вашей веб-странице», — сказал Саймон.

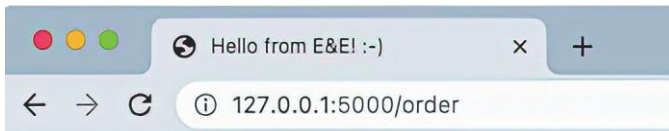
«Но почему там написано ‘Hello from Mu!’?» — заметил Эрик. — «Разве это не должно быть ‘Hello from E&E’?»

"Хорошее замечание!" — сказал Саймон. — «Это довольно легко изменить. Найдите эту фразу в файле `base.html` и измените ее. Она должна находиться очень близко к началу и должна быть заключена в теги `<title>`."

Эрик быстро нашел фразу и изменил ее. «Следует ли мне перезапустить приложение Python?» — спросил он Саймона.

«Нет, ты можешь просто перезагрузить страницу в браузере и увидишь изменения».

Действительно, на странице теперь появился новый заголовок.

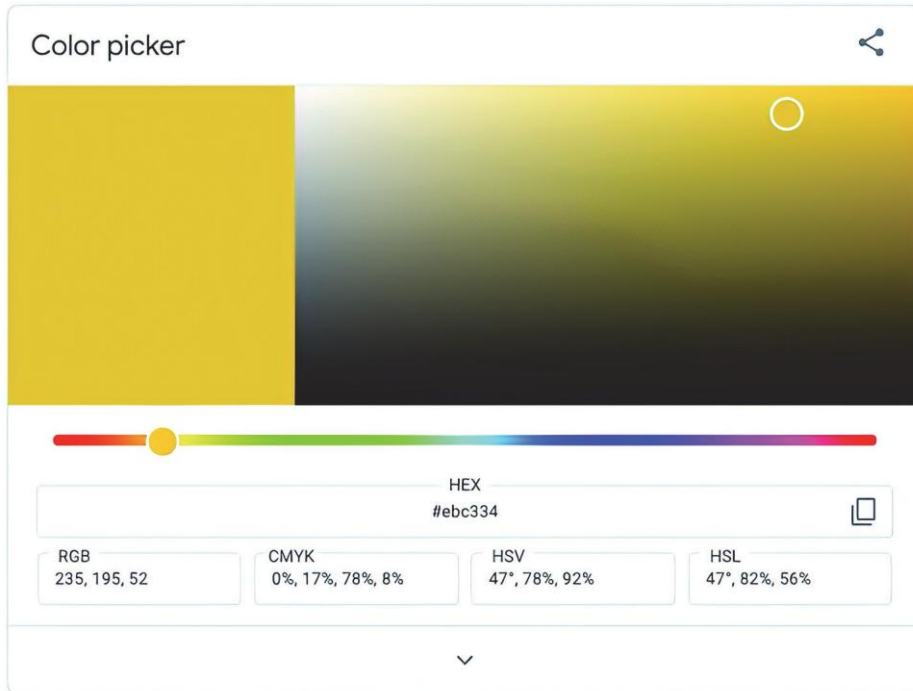


Эмили сказала: «Мы также можем изменить цвет фона. Думаю, нам нужно что-то похожее на кофе или капучино».

«Отличная идея!» — сказал Саймон. — «Во-первых, вам нужно найти

хороший цвет, который нравится вам обоим. Самый простой способ — открыть страницу поиска Google и ввести `color picker`».

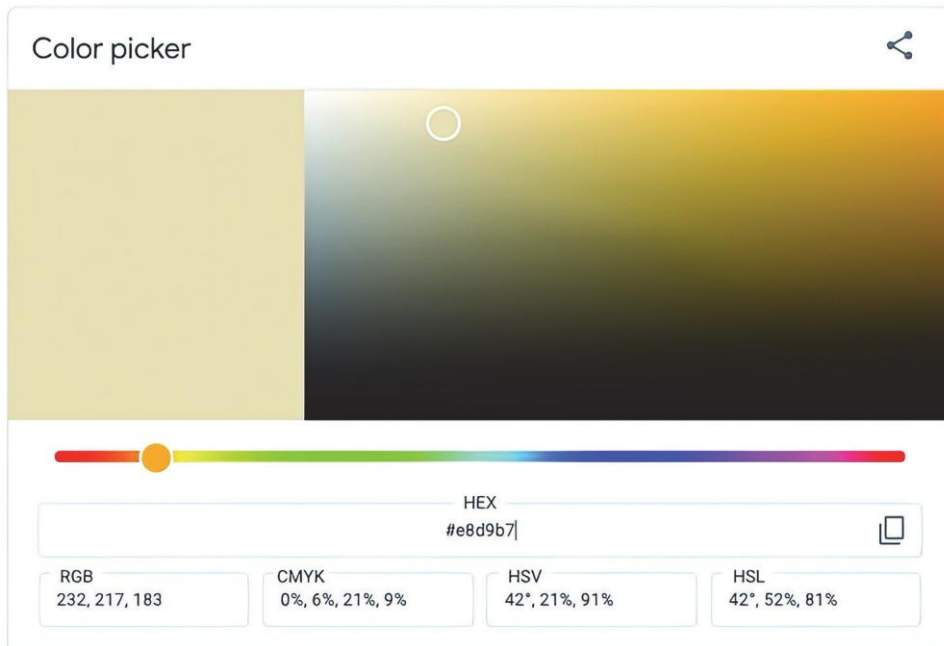
Саймон сделал это, чтобы продемонстрировать что получится, и они увидели Color Picker.



«Вы можете использовать радужную линию, чтобы выбрать базовый цвет, а затем использовать белый кружок, чтобы выбрать оттенок, который вам нравится».

«Есть много других сайтов, которые помогут вам подобрать хорошие сочетания цветов. Это целый мир — такой же, как и в случае шрифтов. Есть чему поучиться. Например», — и он выбрал один из первых результатов на странице поиска: <https://htmlcolorcodes.com/color-picker/>. — «Попробуйте и вы найдете много полезной информации о цветах и о том, как их использовать для веб-страниц».

Эрик и Эмили несколько минут играли с палитрой цветов и наконец придумали цвет.



### **Ваш ход! Поиграйте с палитрой цветов**

Найдите палитру цветов в Google и поиграйте с ней. Выясните, как изменить основной цвет, а затем как изменить его оттенок. Найдите свои любимые цвета для фона.

«Хорошо», — сказал Саймон. — «Теперь скопируйте эти шесть букв и цифр в поле Нех, и мы отредактируем файл `coffeeshop.css`. Кто вводит на этот раз?»

«Я», — сказал Эрик.

«Введи вот это: `body {background-color: .` Затем вставь то, что вы скопировали из палитры цветов. Убедись, что ты вставил знак решетки (#). Затем введи точку с запятой и закройте фигурную скобку».

Эрик добавил эту строку, и теперь CSS-файл стал выглядеть так:

#### **Листинг 13.4 static/css/coffeeshop.css: Добавьте цвет фона в CSS**

```
h1 {font-family: 'Playfair Display', serif;}

body {background-color: #e8d9b7;}
```

### **Ваш ход! Измените фон страницы**

Отредактируйте файл `static/css/coffeeshop.css` и добавьте строку `body` с выбранным вами цветом фона.

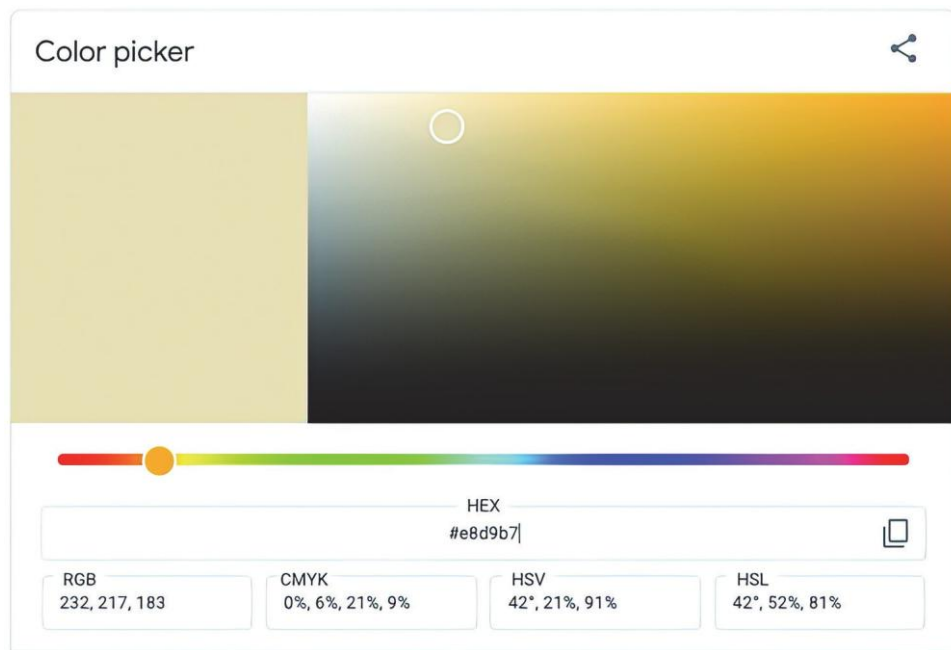
Эрик перезагрузил страницу заказов, и теперь она стала выглядеть совсем по-другому.



«Мне это нравится все больше и больше», — сказала Эмили. — «Можем ли мы сделать эти меню тоже кофейного цвета?»

«Да, конечно», — ответил Саймон. — «Но я предлагаю сделать их немного светлее, а не такого же цвета. Иди и найди что-нибудь».

Это было сделать еще проще. Эмили и Эрик придумали новый тон меню.



"Хорошо. Теперь позвольте мне помочь вам добавить его в таблицу стилей», — сказал Саймон. — «Мы должны применить этот цвет к элементам нашей формы. Элемент меню называется `select`, а поле ввода, в которое вы вводите имя клиента, называется `input[type="text"]`. Давайте добавим их в таблицу стилей и используем выбранный вами цвет фона».

**Листинг 13.5** `static/css/coffeeshop.css`: Смена цвета меню

```
input[type="text"], select {background-color: #ede9df;}
```

«Что это за цифры и буквы? Что они означают?» — спросил Эрик.

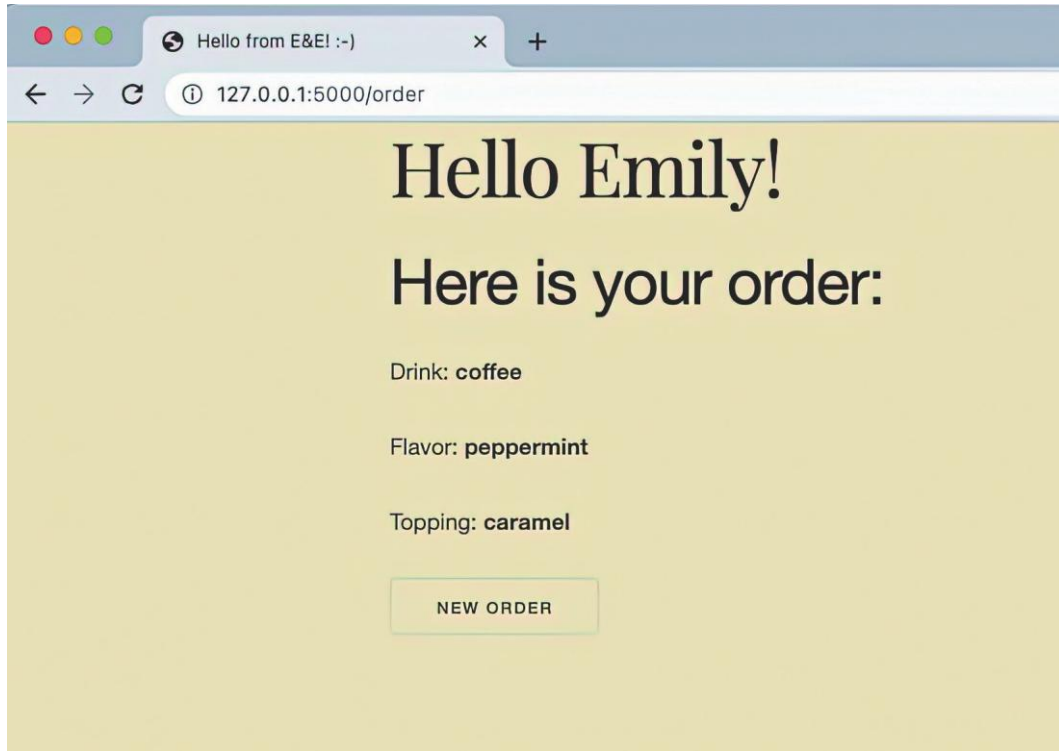
«Они означают, что этими шестью цифрами и буквами можно описать любой цвет, какой пожелаешь», — объяснила Эмили. — «Эта шестёрка состоит из трёх пар трёх цветов: красного, зелёного и синего. Каждая пара показывает, сколько данного цвета содержится в выбранном нами цвете. Это как с красками: можно смешать красный и синий, чтобы получить фиолетовый. Каждая пара символов представляет собой число, но в шестнадцатеричном формате, и это могут быть цифры или буквы от а до f. Я просто не помню, почему они называются «шестнадцатеричными».

«Нех — это сокращение от шестнадцатеричного», — сказал Саймон. — «Это способ представления чисел, похожий на наш обычный способ, но более короткий и более понятный для компьютеров. В нашей обычной системе, называемой «десятичной», мы имеем цифры от 0 до 9, всего 10 цифр. Одной цифрой мы можем описать числа от 0 до 9. Двумя цифрами мы можем описать числа от 0 до 99. Шестнадцатеричная система аналогична, но вместо 10 цифр мы используем 16 цифр. Они идут от 0 до f: мы добавили буквы от а до f и используем их как цифры. Итак, цифра а означает 10, b — 11 и так далее. Цифра f означает 15. Одной шестнадцатеричной цифрой мы можем описать числа от 0 до 15; двумя — от 0 до 255. Такая запись будет значительно короче: 255 требует трёх цифр в десятичной системе, а в шестнадцатеричной — это просто ff».

«Давайте вернемся к цветам», — продолжил Саймон. — «Как сказала Эмили, каждое шестизначное шестнадцатеричное число говорит нам, *сколько* красного, зеленого и синего содержится в определенном цвете. Например, цвет вашего меню #ede9df означает сочетание красного цвета ed, зеленого цвета e9 и синего цвета df. В десятичных числах это означает...» — Саймон посмотрел на страницу выбора цвета и прочитал оттуда цифры, — «это означает 237 красного цвета, 233 зеленого цвета и 223 синего цвета. Числа могут доходить до 255. Если все три из них станут 255, это будет просто белый цвет».

«Совершенно верно», — подтвердил Саймон. — «Если вы хотите узнать больше о веб-цветах, прочитайте о них статью в Википедии». Он быстро нашел страницу веб-цветов и открыл ее в новой вкладке: [https://en.wikipedia.org/wiki/Web\\_colors](https://en.wikipedia.org/wiki/Web_colors).

«Я хочу заказать что-нибудь на странице с нашим новым дизайном», — сказала Эмили. Она вошла в свой заказ. Затем она нажала Submit и увидела новую страницу.



«Почему он не напечатал вторую строку нашим шрифтом?» — она была не очень довольна.

«Может быть потому, что мы просто установили шрифт для `h1`?» — предложил Эрик.

«Ты прав, Эрик», — сказал Саймон. — «Это довольно легко исправить. Откройте файл `skeleton.css` и посмотрите, как там определяются стили. Мы можем многому научиться, глядя на то, как это делается в таблицах стилей других разработчиков. Идите вперед и перейдите к разделу под названием `Typography`. Вы увидите, как стилизовать другие уровни заголовка».

Эрик открыл `skeleton.css` и прокрутил до `Typography`. Он увидел следующее:

#### Листинг 13.6 `static/css/skeleton.css`: Раздел `Typography`

```
/* Typography
----- */
h1, h2, h3, h4, h5, h6 {
  margin-top: 0;
  margin-bottom: 2rem;
  font-weight: 300; }
h1 { font-size: 4.0rem; line-height: 1.2; letter-spacing: -.1rem; } h2
{ font-size: 3.6rem; line-height: 1.25; letter-spacing: -.1rem; } h3 {
font-size: 3.0rem; line-height: 1.3; letter-spacing: -.1rem; } h4 {
font-size: 2.4rem; line-height: 1.35; letter-spacing: -.08rem; } h5 {
```



```
font-size: 1.8rem; line-height: 1.5; letter-spacing: -.05rem; } h6 {  
font-size: 1.5rem; line-height: 1.6; letter-spacing: 0; }
```

Саймон сказал: «Я не буду вдаваться здесь во все детали, но вы видите, что можно перечислить различные уровни заголовков — h1, h2, h3 и т. д. — через запятую, и применить стиль ко всем из них. Также для каждого из уровней заголовка можно задать свой размер шрифта и другие параметры. Вы можете скопировать эту часть в файл `coffeeshop.css` и попробовать изменить различные параметры. Но сейчас самое важное — добавить h2, h3 и т. д. к нашему стилю h1».

«Понял», — сказал Эрик. Он быстро изменил первую строку `coffeeshop.css`:

**Листинг 13.7** `static/css/coffeeshop.css`: Изменение шрифта для всех заголовков

```
h1, h2, h3, h4, h5, h6 {font-family: 'Playfair Display', serif;}
```

Он нажал кнопку `Reload`, и браузер отобразил сообщение с подтверждением.

---

Confirm Form Resubmission

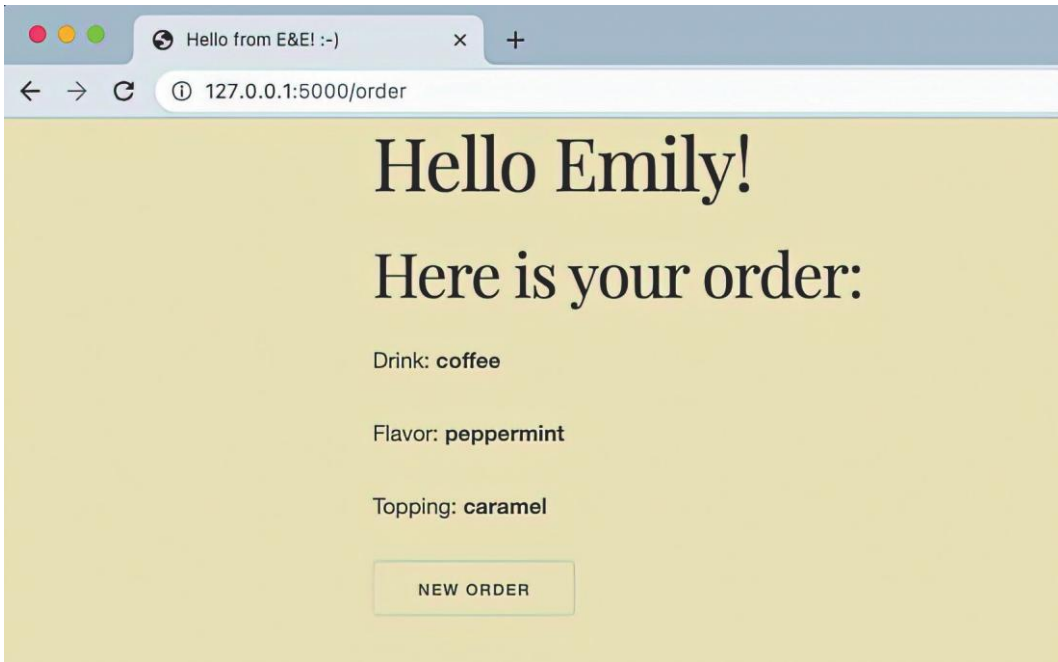
The page that you're looking for used information that you entered.  
Returning to that page might cause any action you took to be  
repeated. Do you want to continue?

Cancel

Continue

«Нажмите `Continue`», — сказал Саймон. — «Это просто снова отправит заказ Эмили. Пока мы производим тестирование, это нормально, но в реальном приложении это создаст еще одну запись в нашей базе данных и новый заказ, поэтому вам следует быть осторожным, когда это приложение работает для реальной кофейни».

Теперь приложение использовало правильный шрифт.

**Ваш ход! Измените шрифт для других заголовков**

Отредактируйте файл `static/css/coffeeshop.css`, чтобы изменить шрифт для всех заголовков с `h1` по `h6`.

«Мы также можем добавить картинку», — сказала Эмили. — «Мы делали это на уроке. Нам просто нужно найти красивую картинку и загрузить ее».

«Да, хорошая идея», — сказал Саймон. — «Однако убедитесь, что вы не загружаете просто любое понравившееся изображение. В Интернете много классных фотографий, но у большинства из них есть владельцы, например фотографы или художники, которым не нравится, когда кто-то берет их фотографии и размещает на своем сайте, даже не упоминая владельца. В этом отношении существует множество правил, и у разных владельцев будут разные требования. Для нас будет безопасным использовать бесплатные картинки, чтобы не нарушать чьи-либо права. Найдите что-нибудь вроде `coffee images free download` и посмотрите, что это нам даст».

Эмили и Эрик искали несколько минут, нажимая на разные картинки и споря о них.

Наконец Эрик сказал: «Это хорошо. Здесь есть разные виды кофейных напитков, в красивых чашках, с топингами». И показал Саймону страницу: [https://www.freepik.com/free-vector/colorful-coffee-cup-set-flat-style\\_1544839.htm](https://www.freepik.com/free-vector/colorful-coffee-cup-set-flat-style_1544839.htm).

«Что ж, давайте посмотрим», — сказал Саймон. — «Посмотрите на этот блок на странице».

🛡️ Free license [More info](#)

✍️ Attribution is required [How to attribute?](#)

📄 File type: AI, EPS, JPG [How to edit?](#)

He clicked “More info” and checked the license terms.

## License summary

Our license allows you to use the content

- ✓ For commercial and **personal projects**
- ✓ On digital or **printed media**
- ✓ For an **unlimited number of times** and without any time limits
- ✓ From **anywhere in the world**
- ✓ To make **modifications** and create derivative works

«Мне кажется, что это хорошо. Нам разрешено его использовать, но мы должны использовать указание авторства — видите эту фразу *Attribution is required*? Давайте посмотрим, что нам следует делать». Он нажал ссылку **Attribute?**

## How to attribute?

Creating content takes a lot of time and effort, which is why we ask you to provide a link to attribute the source. Where will you be using the image?

[Web](#) [Printed items](#) [Video](#) [Apps and games](#)

For example: websites, social media, blogs, ebooks, newsletters, etc.

Copy this link and paste it in a visible place, close to where you’re using the image. If that’s not possible, place it at the footer of your website, blog or newsletter, or in the credits section.

Image by Freepik

Copy 

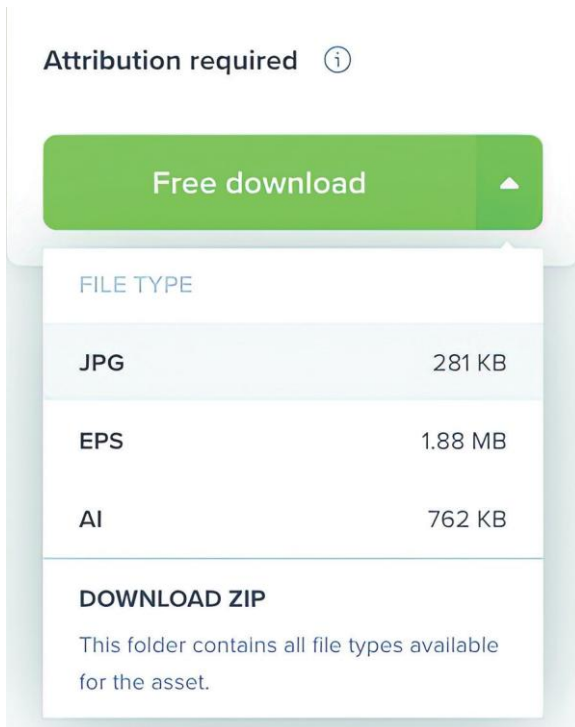
«Нам нужно скопировать текст и добавить его на нашу страницу рядом с картинкой. Без проблем», — сказал он и нажал синюю кнопку Сору.

«Теперь нам нужно добавить его на нашу страницу заказа», — сказал он. Он открыл файл `order.html` и вставил текст внизу страницы, прямо перед строкой `{% endblock %}`.

«Мы перенесем его позже, когда разместим нашу фотографию», — пояснил он.

«Теперь загрузи картинку», — сказал он Эрику

Эрик нажал зеленую кнопку Download, а затем открыл раскрывающееся меню Free Download. Затем он выбрал JPG.



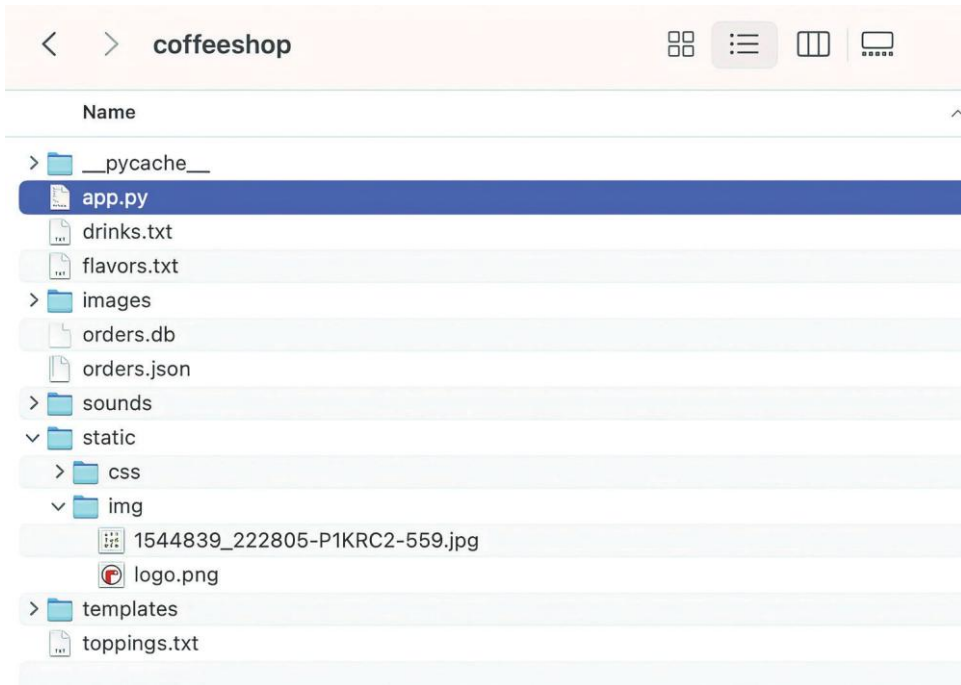
Наименование файла изображения, `1544839_222805-P1KRC2-559.jpg`, появилось в нижней части его браузера.

**ПРИМЕЧАНИЕ** В вашем случае наименование файла, скорее всего, будет другим. Пожалуйста, используйте наименование своего файла в следующих изменениях кода.

Саймон сказал: «Теперь скопируйте этот загруженный файл изображения из папки Downloads в папку вашего проекта. Вам придется скопировать его в папку `coffeeshop/static/img/`. Сможешь ли ты найти ее?»

«Я думаю, что смогу», — сказал Эрик. Он открыл программу файлового

менеджера (Finder на своем Mac-Book). После нескольких движений мышью и щелчков он нашел нужное место. Каталог его проекта теперь содержал файл изображения.



«Отлично», — сказал Саймон. — «Теперь нам нужно разместить его на странице. Откройте файл `order.html` еще раз. Позвольте мне помочь вам».

Саймон добавил строку непосредственно перед строкой с указанием авторства, которую он скопировал ранее.

Нижняя часть файла `order.html` теперь выглядит так:

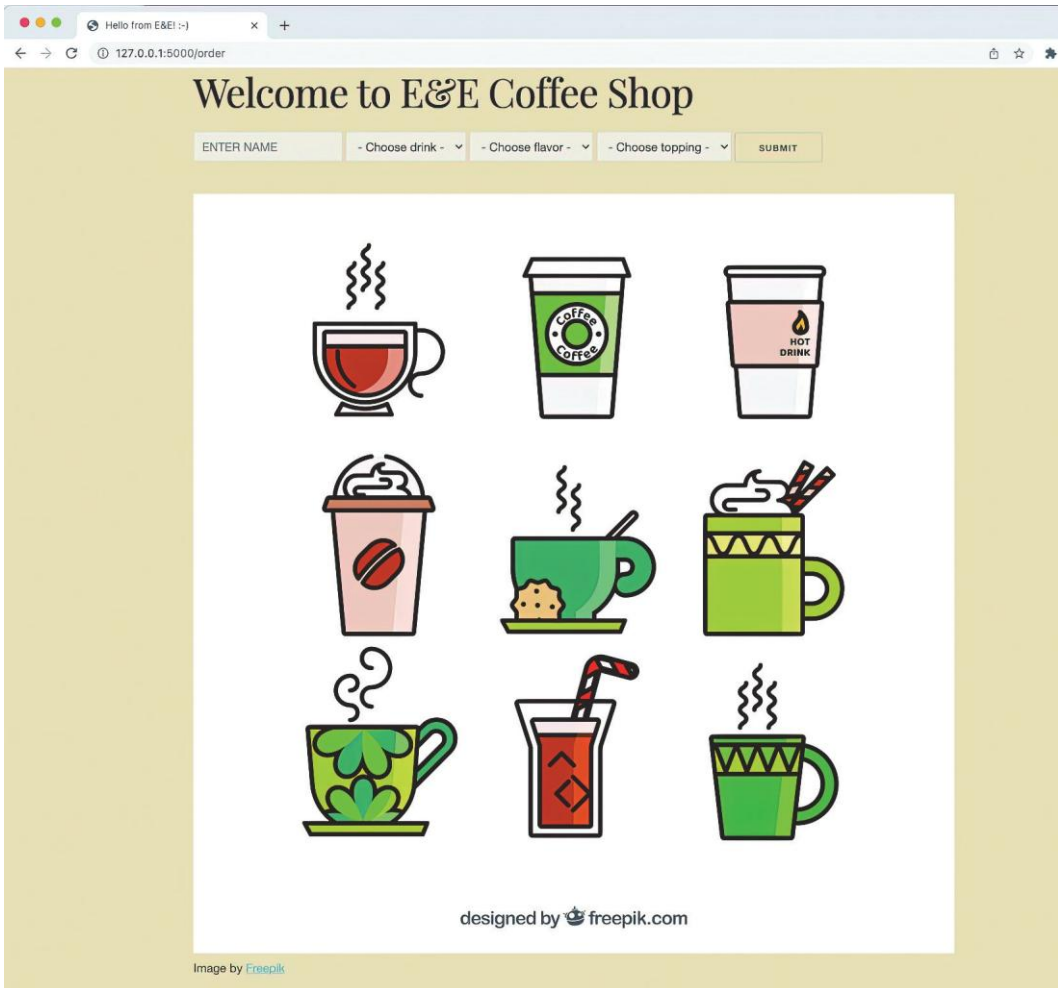
```
...

Image by <a href="https://www.freepik.com/free-vector/ colorful-coffee-cup-set-
flat-style_1544839.htm">Freepik</a>

{% endblock %}
```

Он сохранил файл и сказал Эмили и Эрику: «Теперь перезагрузите страницу, и мы посмотрим, сработает ли она».

Эмили нажала Reload, и они увидели обновленную страницу.



### **Ваш ход! Добавьте изображение на свою страницу**

Найдите понравившуюся картинку и добавьте ее на страницу заказа. Не забудьте проверить лицензию и следовать инструкциям по соблюдению авторских прав.

«Отличная работа!» — сказал Саймон. — «Теперь это выглядит как настоящий интернет-магазин. Важно то, что это не только выглядит великолепно, но и работает!»

Эмили и Эрик были счастливы увидеть результат своей упорной работы. Собрать все мелкие кусочки воедино было непросто, но результат оказался ошеломляющим — это было настоящее веб-приложение, которое они написали сами.

«Как насчет подведения итогов?» — спросил Саймон.

«Мне понравилось работать со шрифтами и цветами. Это было проще, чем работать с базой данных», — сказала Эмили. — «И спасибо за объяснение шестнадцатеричных чисел — я не знала всех подробностей».

«Мне понравилась страница Google Fonts», — сказал Эрик. — «Как много красивых шрифтов! Могу ли я использовать их в своих документах? Я планирую создать плакат».

«Конечно!» — ответил Саймон. — «Вы можете установить их на свой компьютер и использовать в Word. В Google Docs их даже не нужно устанавливать. Просто найдите «Дополнительные шрифты» в меню «Шрифты» и используйте их».

«Кроме того, мы узнали о лицензировании и указании авторства изображений», — добавила Эмили. — «Саймон, ты прав, это целый мир».

«Замечательный итог, отличная работа», — заметил Саймон. — «Теперь у меня есть другая идея. Вы ведь знаете, что такое ИИ или *искусственный интеллект*, верно?»

«Да, все эти глупые чат-боты и программы, которые создают забавные картинки», — ответил Эрик. — «Некоторые из них мы уже опробовали с друзьями».

«Да, но я говорю об искусственном интеллекте, который может помочь нам, разработчикам», — сказал Саймон. — «Некоторые инструменты могут помочь нам писать функции, улучшать наш код и объяснять сообщения об ошибках».

«Да, я видела несколько демо», — сказала Эмили. — «ИИ писал целые рабочие программы. Это было похоже на волшебство. Саймон, значит ли это, что ИИ может написать приложение, подобное тому, которое мы только что написали?»

«Возможно», — сказал Саймон. — «Но он все равно не может сделать это без помощи человека. Сначала вам нужно придумать идею приложения, как это сделал Эрик несколько недель назад. Во-вторых, вам придется объяснить ИИ, чего вы от него ожидаете. Вы должны быть очень конкретными, иначе ИИ сделает что-то, чего вы не ожидаете. В-третьих, вам придется просмотреть код и убедиться, что он работает так, как вы хотите. И, конечно же, никакой ИИ не сможет выбрать цвета, шрифты и изображения для веб-сайта, которые вам нравятся, как вы это только что сделали с Эриком. Это дело вкуса, и, насколько я знаю, ни один ИИ не может этого сделать».

«ИИ может оказать большую помощь», — продолжил он, — «но пока он не может полностью заменить людей-разработчиков и дизайнеров. Давайте попробуем завтра и посмотрим, как ИИ может нам помочь».

### **Новое, что вы узнали сегодня**

- *Различные типы шрифтов* — с засечками, без засечек и моноширинные — это разные типы шрифтов, которые вы можете использовать в своем

приложении. Шрифт с засечками хорош для заголовков, текстов книг и документации. Шрифт без засечек используется для меню, а также для заголовков и текста. Моноширинный формат используется для демонстрации программного кода, имен файлов и примеров команд.

- *Цветовые коды* — цвета можно описывать с помощью *шестнадцатеричных* чисел, например `#a1d4c5`. Они состоят из трех основных цветов: красного, зеленого и синего. Вы можете использовать эти цветовые коды, чтобы указать цвета фона, меню, заголовка и текста для вашей веб-страницы.
- *У изображений есть владельцы и лицензии* — важно знать, что не все изображения, которые вы найдете в Интернете, можно свободно использовать на ваших веб-страницах. У картин и фотографий есть владельцы, которые решают, как вы можете использовать их работы. Часто вы можете использовать их фотографии, не платя им, но вам следует разместить на своей странице указание авторства. Это способ сказать «Спасибо» за их работу.

### ***Код для этой главы***

Вы можете найти код для этой главы здесь:

<https://github.com/pavelanni/pythonicadventure-code/tree/main/ch13>.



# 14

## *Помощь искусственного интеллекта: улучшение нашего кода*

### **В этой главе**

- Эрик и Эмили просят ИИ о помощи
- ИИ проверяет код
- ИИ пишет комментарии и поясняет сообщения об ошибках
- ИИ объясняет сам себя
- ИИ дает другие идеи о том, как использовать ИИ

«Готовы ли вы получить помощь от ИИ?» — спросил Саймон у Эрика и Эмили.

«Конечно», — ответил Эрик. — «Есть ли специальный сайт по искусственному интеллекту для разработчиков?»

«Для разработчиков существует несколько инструментов искусственного интеллекта», — сказал Саймон. — «В некоторых из них вы заходите на сайт и просите о помощи. Например, вы запрашиваете проверку кода или пояснения к коду. В ваш редактор можно установить и другие инструменты, которые помогут вам писать код. Вы попробуете их позже, поэтому сегодня давайте начнем с использования веб-сайта. У меня есть аккаунт в ChatGPT — давайте воспользуемся им».

Саймон открыл веб-страницу <https://chat.openai.com> и ввел свое имя для входа

и пароль. Они увидели простую страницу с подсказкой внизу “Send a message . . .».

ChatGPT PLUS

➤

«Помнишь, что я тебе вчера говорил», — начал он, — «что ты должен объяснить ИИ, чего именно ты от него хочешь? Это не чат-бот, где можно начать с вопроса «Как дела?». Мы хотим получить разбор кода для одной из наших функций, поэтому должны объяснить именно это. Рекомендуется сказать ИИ, что он должен действовать как кто-то. В данном случае мы попросим его выступить в роли рецензента кода. Затем мы скажем ему, чего мы от него ожидаем».

Саймон начал печатать в окне подсказки:

Выступите в роли рецензента кода. Ваша задача — получить от меня функцию и дать отзыв о структуре кода, синтаксисе, возможных ошибках и возможных улучшениях.  
Вот моя первая функция для анализа:

Он остановился и сказал: «Давайте начнем с простой функции. Например, возьмем функцию `read_menu()` ». Он скопировал функцию из редактора:

```
def read_menu(filename): f
    = open(filename) temp =
    f.readlines() result =
    []
    for item in temp:
        new_item = item.strip()
        result.append(new_item)

    return result
```

ИИ почти сразу начал отвечать:

*Спасибо за предоставленный код. Вот мой разбор:*

*Я рассмотрел вашу функцию read\_menu и вот мой отзыв:*

- Структура кода. Общая структура вашего кода проста и понятна. Хорошая работа!
- Синтаксис — синтаксис правильный и соответствует соглашениям Python.
- Потенциальные ошибки. В коде отсутствует надлежащая обработка ошибок, особенно когда речь идет об операциях файлового ввода-вывода. Если данное имя файла не существует или существует проблема с разрешением, ваш код вызовет исключение.

- *Улучшения:*
  - *Для открытия файла можно использовать оператор with, который автоматически обрабатывает закрытие файла даже в случаях исключений. Это гарантирует, что файл будет правильно закрыт после использования.*
  - *Вы можете упростить цикл for, используя упрощенный подход к созданию списков, что делает ваш код короче и более Python-ориентированным.*

*Вот обновленная версия вашей функции с предлагаемыми улучшениями:*

ИИ открыл отдельное окно с кодом Python:

```
def read_menu(filename):
    try:
        with open(filename) as f:
            result = [item.strip() for item in f.readlines()]
    except FileNotFoundError:
        print(
            f"Error: The file '{filename}' could not be found.")
        result = []
    except PermissionError:
        print(f"Error: You do not have permission to read \
the file '{filename}'.")
        result = []
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        result = []

    return result
```

После этого ИИ добавил свои пояснения:

*Эта версия функции включает обработку ошибок для файловых операций, использует оператор with для обработки файла и содержит упрощенный подход к созданию списка результатов.*

«Ух ты!» — одновременно воскликнули Эмили и Эрик.

Саймон прокомментировал: «Все аргументы верные. Да, использование with open — обычная практика при работе с файлами. Раньше я не хотел использовать его, поскольку это потребовало бы дополнительных пояснений, а мы к этому еще не были готовы. Комбинация try - except также очень распространена. В этом случае, если файл меню не найден или у вас нет прав на его чтение, то программа не просто вылетает с ошибками, а вежливо сообщает вам, что произошло. Это очень хорошая практика».

«Что такое упрощенный подход к созданию списка?» — Эрик спросил.

«Это особый способ Python сделать циклы for короче. Посмотрите на код, который написал ИИ».

```
result = [item.strip() for item in f.readlines()]
```

«Это замена этого цикла:»

```
temp = f.readlines() result
= []
for item in temp:
    new_item = item.strip()
    result.append(new_item)
```

«Да, так получилось короче», — согласился Эрик, — «но цикл легче понять».

«Верно!» — сказал Саймон. — «Вот поэтому мы и начали с циклов. Их легче понять. Теперь, когда вы знаете, как это работает, вы можете использовать более компактную версию. Сначала я сам не понимал, что такое список. Но я стал использовать их чаще, и теперь они читаются так же легко, как обычные циклы. И, как правильно заметил ИИ, это более соответствует *стилю Python*. О, мы также можем попросить ИИ объяснить нам это».

Саймон ввел:

Пожалуйста, объясните, что такое упрощенный подход к созданию списка в Python?

ИИ ответил:

*Упрощенный подход к созданию списков — это краткий способ создания списка в Python. Это синтаксическая конструкция, которая позволяет вам создать новый список, указав элементы, которые вы хотите включить, обычно на основе существующей итерации (например, списка, кортежа или диапазона). Понимание списков является более кратким и часто быстрее, чем использование традиционного цикла for и метода append() для построения списка.*

*Базовая структура упрощенного подхода к созданию списка такова:*

```
[expression for item in iterable if condition]
```

- *expression: Выражение, определяющее значение каждого элемента в новом списке. Часто это связано с переменной item.*
- *item: Временная переменная, представляющая каждый элемент в итерируемом объекте iterable.*
- *iterable: Итерируемый объект (например, список, кортеж, диапазон), который вы хотите пройти в цикле.*
- *if condition: Необязательное условие, которое фильтрует элементы из итерируемого объекта. В новый список будут включены только те элементы, для которых условие истинно.*

*Вот простой пример, демонстрирующий, как использовать упрощенный подход к созданию списка:*

```
# Traditional way to create a list of squares using a for loop
squares = []
for i in range(1, 6): squares.append(i**2)
print(squares) # Output: [1, 4, 9, 16, 25]

# Using list comprehension to create the same list of squares
squares = [i**2 for i in range(1, 6)]
print(squares) # Output: [1, 4, 9, 16, 25]
```

*Вы также можете использовать упрощенный подход к созданию списка с условием. Вот пример, который создает список четных чисел от 1 до 10:*

```
even_numbers = [x for x in range(1, 11) if x % 2 == 0]
print(even_numbers) # Output: [2, 4, 6, 8, 10]
```

*Упрощенный подход к созданию списков может сделать ваш код более кратким и читабельным, особенно для простых операций. Однако для более сложных операций или вложенных циклов он может стать менее читабельным, поэтому важно найти баланс между краткостью и читабельностью.*

«Давайте попросим ИИ помочь нам с чем-нибудь еще. Когда мы писали наш код, мы не писали к нему комментарии. Это не очень хорошая практика. Вам следует добавлять комментарии в свой код, чтобы объяснить, что вы делаете. Возможно, кто-то читает ваш код. Или, что чаще, вы прочитаете его через три месяца. Я гарантирую вам, что если вы не прокомментируете свой код, вы спросите себя: «Что это? Почему я написал именно так? Что я имел в виду?» Поверьте мне и моему опыту, — сказал Саймон и улыбнулся.

«Так что это хорошая практика — комментировать свой код. Обычно мы делаем это сами, но давайте посмотрим, сможет ли нам помочь ИИ», — он написал следующее в окне запроса:

Пожалуйста, добавьте комментарии к только что созданной функции.

ИИ ответил сразу же:

*Конечно! Вот обновленная версия функции с добавленными комментариями, объясняющими каждый шаг:*

```
def read_menu(filename):
    """
    Reads the menu from a file and returns a list of items.

    :param filename: The name of the file containing the menu.
    :return: A list of menu items, with leading and trailing
    whitespaces removed.
    """
    try:
        # Open the file using 'with' to ensure it's properly
        # closed after use
        with open(filename) as f:
            # Read all lines from the file and remove whitespaces
            # using list comprehension
            result = [item.strip() for item in f.readlines()]
    except FileNotFoundError:
        # Handle the case when the file does not exist
        print(f"Error: The file '{filename}' could not be found.")
        result = []
    except PermissionError:
        # Handle the case when there is no permission to read
```

```

    # the file
    print(f"Error: You do not have permission to read \
    the file '{filename}'".)
    result = []
except Exception as e:
    # Handle any other unexpected errors print(f"An
    unexpected error occurred: {e}") result = []

return result

```

«Это действительно здорово!» — сказал Саймон. — «Смотрите: в начале функции добавлена строка документации. Это специальная строка, которую можно включить в документацию вашей функции. Затем для каждого исключения было добавлено объяснение того, что оно означает. Я сам не смог бы написать лучший комментарий! Посмотрите на этот пример и всегда добавляйте комментарии и строки документации в свой код!»

«О, я думал, что ИИ сейчас сделает это за нас», — сказал Эрик и улыбнулся.

«Да, ИИ может комментировать простые строки кода, такие как обработка исключений», — ответил Саймон, — «но я уверен, что довольно скоро вы начнете писать функции, которые не так просты, и даже самый умный ИИ не сможет прокомментировать их. Приготовьтесь делать это самостоятельно, если собираетесь писать действительно оригинальный код».

«Может ли ИИ помочь нам с сообщениями об ошибках?» — спросила Эмили.

«Отличная идея!» — сказал Саймон. — «Давайте попробуем — я уверен, ИИ может объяснить некоторые сообщения об ошибках. Эрик, помнишь, в самом начале ты получил первое сообщение об ошибке?»

«Да, я помню, что ты был этому очень рад», — проворчал Эрик.

«Дайте мне найти эту ошибку, я сохранил ее в своих заметках. Вот она», — сказал Саймон. Он ввел сообщение об ошибке в окне запроса ИИ:

Пожалуйста, объясните следующее сообщение Python об ошибке:

```

Traceback (most recent call last):
  File "/home/erik/mu_code/menu.py", line 30, in <module> print("Main
    product: ", drinks[drink])
TypeError: list indices must be integers or slices, not str

```

ИИ ответил:

*Это сообщение об ошибке означает, что вы пытаетесь использовать строку в качестве индекса для доступа к элементу в списке, что не разрешено в Python. В конкретном случае этого сообщения об ошибке кажется, что вы пытаетесь получить доступ к элементу в списке drinks, используя переменную drink, содержащую строку. Вот пример кода, который может вызвать эту ошибку:*

```

drinks = ["coffee", "tea", "soda"] drink
= "coffee"
print("Main product: ", drinks[drink])

```

*В этом коде напиток — это строковая переменная, содержащая значение "coffee". Когда мы пытаемся получить доступ к drinks[drink], Python выдает ошибку*

*TypeError*, поскольку индекс должен быть целым числом, а не строкой. Чтобы исправить эту ошибку, вам нужно использовать целочисленный индекс для доступа к элементу в списке. Например, если вы хотите получить доступ к первому элементу списка, вы должны использовать `drinks[0]`:

```
drinks = ["coffee", "tea", "soda"] drink
= 0
print("Main product: ", drinks[drink])
```

В результате будет выведено "Main product: coffee", который является первым элементом списка `drinks`.

«Он даже знает, о чем наше приложение!» — воскликнул Эрик.

«Да», — сказал Саймон. — «Мы предоставили ему всего одно сообщение об ошибке, и он догадался, что делает наша программа».

«Это связано с тем, как работает эта программа искусственного интеллекта», — добавил он.

«И как это работает? Вы можете объяснить?» — спросила Эмили.

"Хорошо..." — начал Саймон. — «Я не эксперт в области искусственного интеллекта. Я только что прочитал несколько статей. Попробую объяснить, как это понимаю я».

«Люди в компаниях, занимающихся искусственным интеллектом, таких как OpenAI, которая создала ChatGPT, знают, как *научить* компьютеры понимать человеческий язык и программы. Для этого они передают компьютерам *тонны* текстов и кода. Затем специальная программа, называемая *моделью* (кстати, обычно написанная на Python), анализирует все эти тексты и выясняет, какие слова чаще используются вместе с другими словами. Как и в нашем случае, модель ИИ выяснила, что часто, когда люди используют слово 'drinks' они также используют слова 'coffee', 'tea' и 'soda'. Вот почему он выдал нам пример, очень похожий на наш код».

«Кроме того», — продолжил Саймон, — «эти модели могут рассчитывать так называемое сходство текста и сравнивать разные слова и фразы. Затем они смогут использовать эти результаты, чтобы дать вам ответы на такие вопросы, как: «Что расположено в Испании так же, как Париж во Франции?» Модель рассчитает «расстояние» между «Парижем» и «Францией», а затем выполнит подобную математическая операция:

```
? = Spain + (Paris - France)
```

«Он получит ответ 'Madrid'».

«Могу ли я попробовать?» — спросила Эмили.

«Да, давай попробуем!» — сказал Саймон. — «Я читал это объяснение в нескольких статьях, но сам никогда не пробовал».

Эмили взяла клавиатуру и ввела в окно запрос:

Что связано с Испанией так же, как Париж связан с Францией?

ИИ ответил:

*Мадрид связан с Испанией так же, как Париж связан с Францией. И Мадрид, и Париж являются столицами своих стран, Испании и Франции.*

«Это действительно работает!» — сказал Саймон. — «Давай спросим, как он пришел к такому ответу».

Эмили напечатала:

Можете ли вы объяснить, как вы пришли к этому ответу?

ИИ сказал:

*Конечно! Я пришел к этому ответу, понимая отношения между Парижем и Францией. Париж – столица Франции. Итак, вопрос в том, что имеет схожие отношения с Испанией.*

*Зная, что Мадрид является столицей Испании, я определил, что он имеет с Испанией такие же отношения, как Париж с Францией. Таким образом, я предоставил Мадрид в качестве ответа на вопрос. Этот тип рассуждения называется рассуждением по аналогии, при котором я нахожу сходство между разными парами предметов на основе их отношений.*

Саймон добавил еще один запрос:

Использовали ли вы сходство текстов, чтобы найти эту связь?

ИИ ответил:

*В качестве языковой модели ИИ я не использовал явно сходство текстов, чтобы найти связь между Парижем и Францией, а затем Мадридом и Испанией. Вместо этого я полагался на свои уже существующие знания и понимание взаимоотношений между странами и их столицами, которые я усвоил во время обучения на большом наборе текстовых данных.*

*Однако методы сходства текста, такие как встраивание слов, могут помочь найти связи между словами на основе их контекста и использования. В данном случае я использовал свои знания и рассуждения по аналогии, чтобы установить взаимосвязь и дать ответ.*

"Как круто!" — сказал Саймон. — «Мы можем использовать ИИ, чтобы узнать об ИИ! Я знаю, что люди используют ИИ для многих других задач, помогая себе разрабатывать программы. Мы уже пробовали писать комментарии, проверять код и объяснять сообщения об ошибках. Интересно, чем еще ИИ может нам помочь? Давай спросим!» — Он ввел следующий запрос:

Мы попросили ChatGPT помочь нам с нашим проектом Python.

Мы уже пробовали писать комментарии, проверять код, объяснять сообщения об ошибках.

Чем еще нам может помочь ChatGPT?

ИИ ответил:



Помимо написания комментариев, проверки кода и объяснения сообщений об ошибках, ChatGPT также может помочь вам с различными другими аспектами вашего проекта на Python, такими как:

- *Создание примеров кода.* ChatGPT может предоставить вам примеры фрагментов кода, чтобы продемонстрировать, как использовать определенные функции, библиотеки или возможности языка.
- *Отладка.* Вы можете описать проблему, с которой столкнулись, а ChatGPT предложит потенциальные решения или способы определения основной причины проблемы.
- *Рекомендации библиотек и инструментов.* ChatGPT может рекомендовать подходящие библиотеки или инструменты для использования для конкретных задач, опираясь на свои знания о популярных и широко используемых ресурсах в экосистеме Python.
- *Лучшие практики и соглашения по кодированию.* ChatGPT может предоставить рекомендации по лучшим практикам, соглашениям по кодированию и рекомендациям по стилю, которые помогут вам писать чистый, удобный в сопровождении и эффективный код Python.
- *Проектирование алгоритмов.* ChatGPT может помочь вам понять или разработать алгоритмы, объясняя их логику и предоставляя примеры их реализации на Python.
- *Структура проекта.* ChatGPT может предоставить советы по организации вашего проекта Python, например, как структурировать ваши каталоги, файлы и модули, а также рекомендации по управлению пакетами и обработке зависимостей.

*Помните, что хотя ChatGPT может быть полезен, он не заменит тщательного исследования и понимания концепций. Кроме того, важно просмотреть сгенерированный код или выражения, чтобы убедиться в его правильности и пригодности для конкретных требований вашего проекта.*

«Все это отличные идеи!» — сказал Саймон. — «Я думаю, нам стоит попробовать их вместе. Завтра давайте обсудим все возможные способы улучшения нашего приложения и построим планы на будущее», — сказал Саймон. — «В ответе ИИ я думаю, последний абзац очень важен. Когда люди впервые видят код, написанный ИИ, они начинают думать, что ИИ может сделать за них все. Но я поигрался с ним и несколько раз замечал, что не весь сгенерированный код работает. Всегда нужно просматривать и тестировать код».

«Хорошо, вернемся к нашему приложению Coffee Shop», — продолжил он. — «Что еще нам с этим делать?»

«Ты сказал, что я могу открыть это приложение на своем iPad», — сказал Эрик.

«Да, это возможно, но нам придется внести пару незначительных изменений. Ты прав, это приложение не имеет особого смысла, если мы можем использовать его только на своем компьютере. Завтра я покажу вам, как открыть его на iPad или на телефоне».

«Мы поместим его в облако?» — спросила Эмили.

«Почти», — ответил Саймон. — «На данный момент оно будет в нашем собственном ‘облаке’ внутри нашей домашней сети. Но вам нужно будет разместить его в реальном облаке, чтобы к нему можно было получить доступ откуда угодно. Давай поговорим об этом завтра».

### **Новое, что вы узнали сегодня**

- *ИИ может помочь вам писать хорошие программы* — ИИ может помочь вам писать более качественный код, объяснять сообщения об ошибках, исправлять ошибки и комментировать ваш код. ИИ также может дать вам представление о структуре программы и полезных инструментах и библиотеках.
- *ИИ не заменяет программиста* — ИИ может делать много интересных вещей, но для создания работающих и полезных приложений ему по-прежнему нужна помощь человека. Не весь код, сгенерированный ИИ, работает без ошибок. Вы всегда должны просматривать и тестировать код.
- *ИИ использует специальные программы, называемые моделями* — ИИ использует языковые модели для генерации ответов. Модели *обучаются* на огромном объеме текста, включая Википедию, онлайн-статьи, программный код и многое другое. Модели могут рассчитывать связи между словами и предложениями и строить ответы на основе этих вычислений.

# 15

## *Следующие шаги: Планы на будущее*

---

### ***В этой главе***

- Эмили и Эрик создают домашнюю страницу для своего приложения
- Саймон демонстрирует, как получить доступ к приложению с других устройств
- Эрик и Эмили тестируют приложение на iPad и телефоне Android
- Саймон, Эмили и Эрик строят планы

«Осталось сделать одну незначительную вещь», — сказал Саймон, — «прежде чем мы сможем сказать, что закончили с этим приложением. Помните, что когда мы открываем наше приложение, мы видим веб-страницу ‘Hello from Mu!’. Я думаю, пришло время заменить ее нашей собственной домашней страницей».

«Конечно!» — сказала Эмили. — «Мы уже выбрали цвета и нашли хорошую картинку! Я думаю, нам следует просто разместить заголовок ‘Welcome to E&E Coffee Shop!’ и добавить изображение, которое мы использовали на странице заказа».

«Да, хорошая идея», — сказал Саймон. — «Но не забывайте об основной цели домашней страницы».

«Какова основная цель?» — спросил Эрик.

«На любом веб-сайте или в веб-приложении вы используете домашнюю страницу для *навигации*», — ответил Саймон. — «Когда люди открывают ваш сайт, они должны иметь возможность быстро понять, что он делает и как его использовать. Посмотрите на сайты своих любимых кофеен и расскажите мне, что вы там найдете. Ищите, что у них общего».

Эмили и Эрик искали знакомые компании и открывали их сайты на нескольких вкладках. Поизучав их некоторое время, Эрик сказал: «У всех есть страница под названием Menu (Меню). У них есть страница Locations (Местоположения) или Find a Store (Найти кофейню), где показано, как найти их кофейню».

«Вам тоже придется создать одну из них, когда вы откроете несколько кофеен», — прокомментировал Саймон.

«Да», — сказал Эрик. Он продолжил: «Кроме того, у них есть бонусные и подарочные карты. Я думаю, они нам тоже нужны. А внизу они обычно имеют множество более мелких ссылок. Там есть About Us (О нас), Contact Us (Свяжитесь с нами) и другие».

«Вы провели отличное исследование», — сказал Саймон. — «Теперь вы понимаете, что я имею в виду под навигацией. На данный момент у нас есть только одна страница, помимо главной, и это страница заказа. Я думаю, мы можем просто разместить на главной странице кнопку, которая приведет на страницу заказа».

«Как мы это сделали на странице ‘Here is your order’?» — Эмили спросила.

«Точно!» — сказал Саймон. — «Вы можете просто скопировать эти три строки со страницы `print.html`. Откройте файл `templates/index.html`, удалите все, что находится между второй строкой с содержимым блока и последней строкой с конечным блоком. Затем вставьте то, что мы только что обсудили: заголовок, кнопку и изображение».

«Да, теперь я знаю, что делать», — сказала Эмили. Они начали работать над главной страницей. Через несколько минут Эмили и Эрик показали Саймону свою домашнюю страницу:

#### Листинг 15.1 `templates/index.html`: Создание домашней страницы

```
{% extends "base.html" %}
{% block content %}

<h1>Welcome to E&E Coffee Shop</h1>

<form action="/order">
<input type="submit" value="New order" />
</form>


Image by <a href="https://www.freepik.com/free-vector/ take-away-coffee-coffee-
```

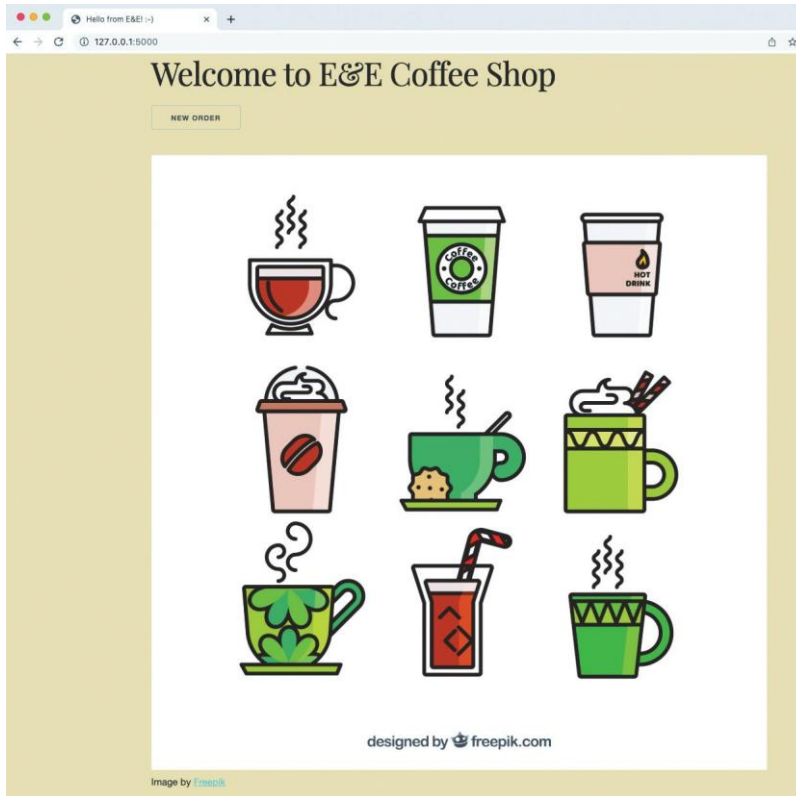
```
mugs_8507631.htm">Freepik</a>

{% endblock %}
```

### **Ваш ход! Отредактируйте свою домашнюю страницу**

Откройте файл `templates/index.html` и внесите изменения, как это только что сделали Эрик и Эмили. Не стесняйтесь менять картинку или название.

«Выглядит хорошо», — сказал он. — «Давай попробуем!»



«Хорошо. Теперь попробуйте нажать New Order (Новый заказ). Должна открыться страница заказа», — сказал Саймон Эрику.

Эрик так и сделал и увидел знакомую страницу заказа с меню.

"Все работает!" — сказал он. — «Теперь у нас есть полноценное веб-приложение!»

«Саймон», — начала свой вопрос Эмили, — «вчера ты сказал, что мы можем использовать это приложение и на наших устройствах. Можешь ли ты показать нам, как это сделать?»

«Да, я хочу попробовать это на своем iPad», — сказал Эрик.

«Отлично!» — ответил Саймон. — «Эрик, ты помнишь, что весь этот проект

начался, когда ты решил заказать с iPad? А я вам говорил, что можно создать веб-приложение, чтобы использовать iPad не только как ноутбук?»

«Да, я это помню», — сказал Эрики и улыбнулся. — «Это было всего пару недель назад».

«Хорошо, тогда давайте заставим это работать на вашем iPad», — сказал Саймон. — «Позвольте мне объяснить, что я собираюсь делать».

Он начал рисовать схему и объяснять.

«Когда мы запускаем наше приложение из редактора Му, по умолчанию оно доступно только с того компьютера, на котором мы его запускаем. Вот почему мы использовали `localhost` или адрес `127.0.0.1` при использовании нашего приложения. Flask и редактор Му позволяют нам работать над приложением, не показывая его другим компьютерам».

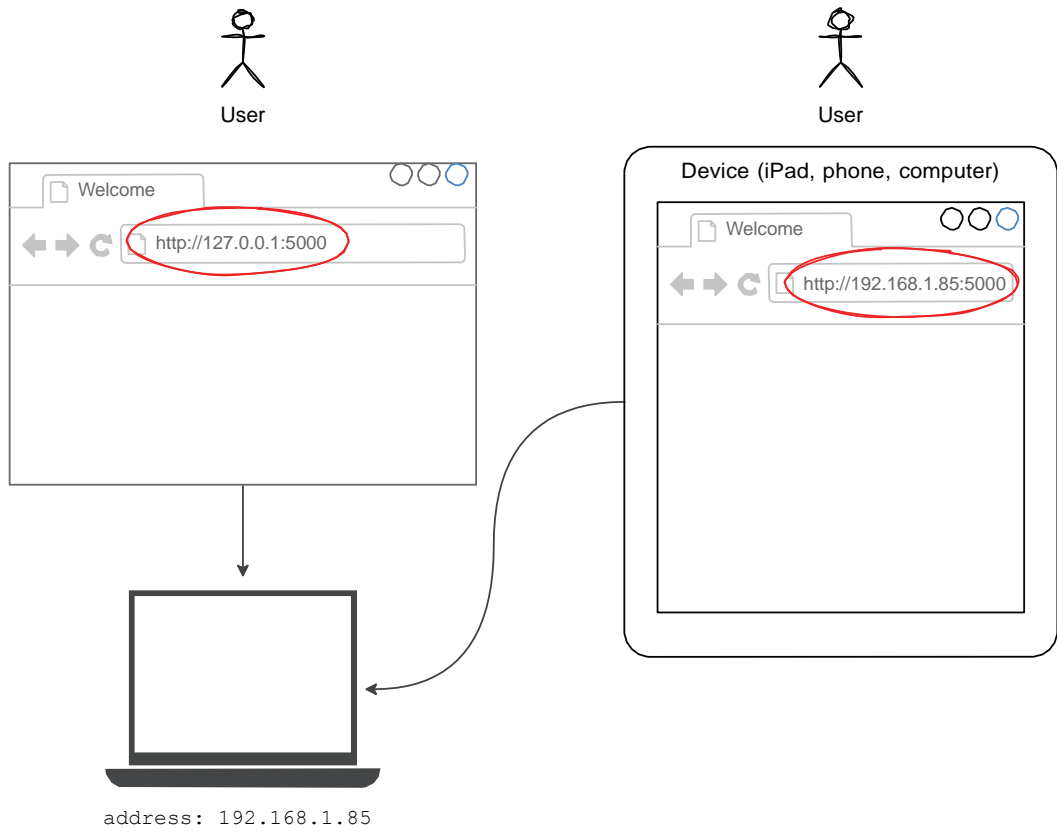
«Это имеет смысл», — сказал Эрик. — «До того, как мы добавили шрифты и изображения, все было не очень красивым».

«Верно», — сказал Саймон. — «Но теперь мы готовы использовать на других устройствах и хотим это изменить. К счастью, в конфигурации редактора Му это всего лишь одна строка. После того, как мы внесем это изменение, все устройства в нашей домашней сети смогут его использовать — если, конечно, на них установлен браузер».

«Могу ли я попробовать со своего телефона?» — спросила Эмили.

«Да, но для этого тебе следует подключиться к нашей домашней сети. Позволь мне ввести пароль нашего Wi-Fi», — и Саймон ввел пароль на телефоне Эмили

«Вот конфигурация по умолчанию находится слева», — продолжил он свое объяснение. — «Мы можем получить доступ к приложению только с компьютера, на котором оно запущено. А здесь справа мы можем получить доступ к приложению с других устройств».



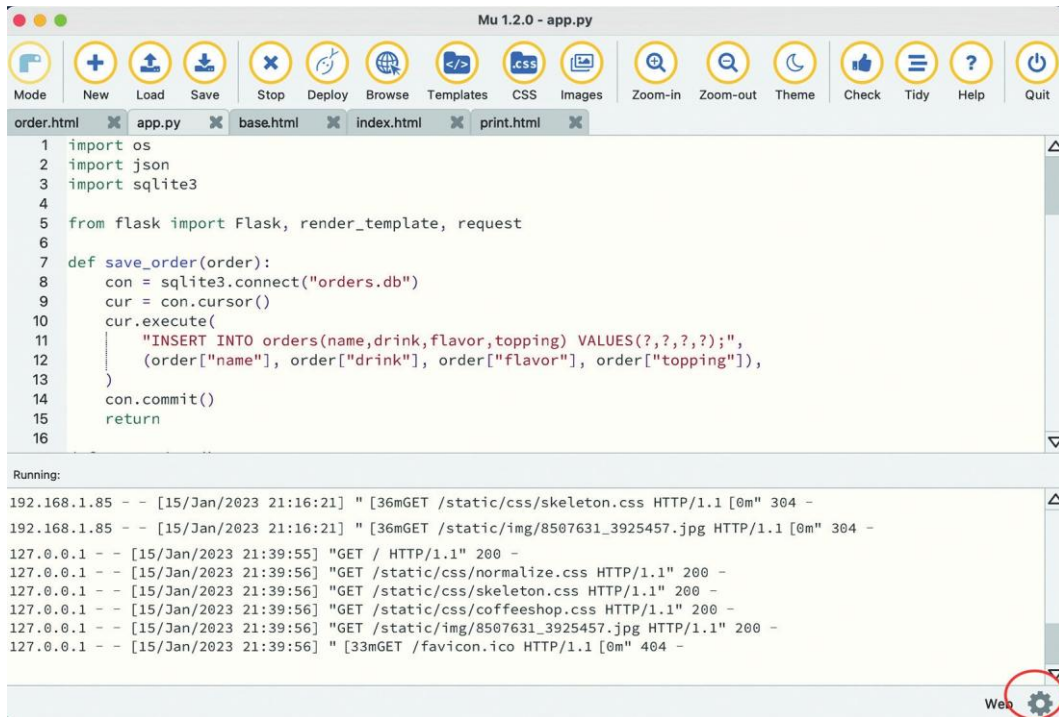
«Каждый компьютер имеет сетевой *адрес*. Твой ноутбук, Эрик, подключен к нашей домашней сети. Он имеет адрес 192.168.1.85. Любое другое устройство в нашей сети, желающее связаться с ним, должно использовать этот адрес. Если ваш компьютер хочет связаться *сам с собой*, он использует

127.0.0.1»

«Могу ли я попробовать сейчас?» — спросил Эрик.

«Подожди, мне нужно внести изменения в конфигурацию. Помни, что теперь мы показываем наше приложение всем устройствам в нашей домашней сети. Конечно, подключенных устройств не так много, но мы все равно хотим убедиться, что наше приложение полностью готово. В будущем мы загрузим его в облако, и тогда его увидят *все компьютеры мира*. Нам лучше заранее убедиться, что все работает правильно и выглядит красиво».

«Эрик, ты видишь эту маленькую шестеренку в правом нижнем углу редактора Му?» — продолжил Саймон. — «Обычно она означает *настройки* или *конфигурацию*. Нажми на нее, пожалуйста».

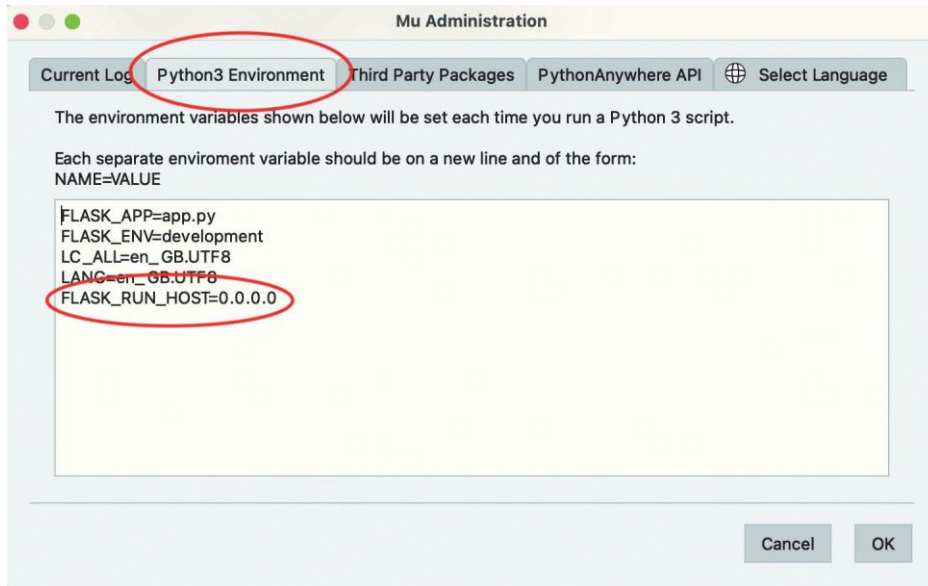


«Теперь выбери вкладку Python 3 Environment и добавь туда следующую строку в конце текста:»

```
FLASK_RUN_HOST=0.0.0.0
```

Эрик сделал это и увидел результат.





«Теперь нажми ОК и перезапусти приложение».

Эрик сохранил конфигурацию, нажав ОК. Он нажал в редакторе Stop, а затем снова Run.

Саймон сказал: «Теперь посмотрите на результат в нижнем окне. Видите эту строку Running on all addresses? Это именно то, что нам нужно. В двух строках под ней вы увидите адреса, которые мы можем использовать для доступа к приложению. Первая — 127.0.0.1, которую мы используем постоянно. Вторая — адрес вашего ноутбука в нашей домашней сети. Вам следует ввести его в браузере на iPad. А ты, Эмили, введи его на своем телефоне. В данном случае это `http://192.168.1.85:5000`.

«Имейте в виду, что этот адрес может измениться. Например, если вы придете домой к Эмили со своим ноутбуком и подключитесь к ее домашней сети, ваш ноутбук получит другой адрес. Вы увидите его в окне вывода, и в доме у Эмили вам придется использовать этот новый адрес».

```
Running:
[33m[WARNING: This is a development server. Do not use this in a
instead. [0m

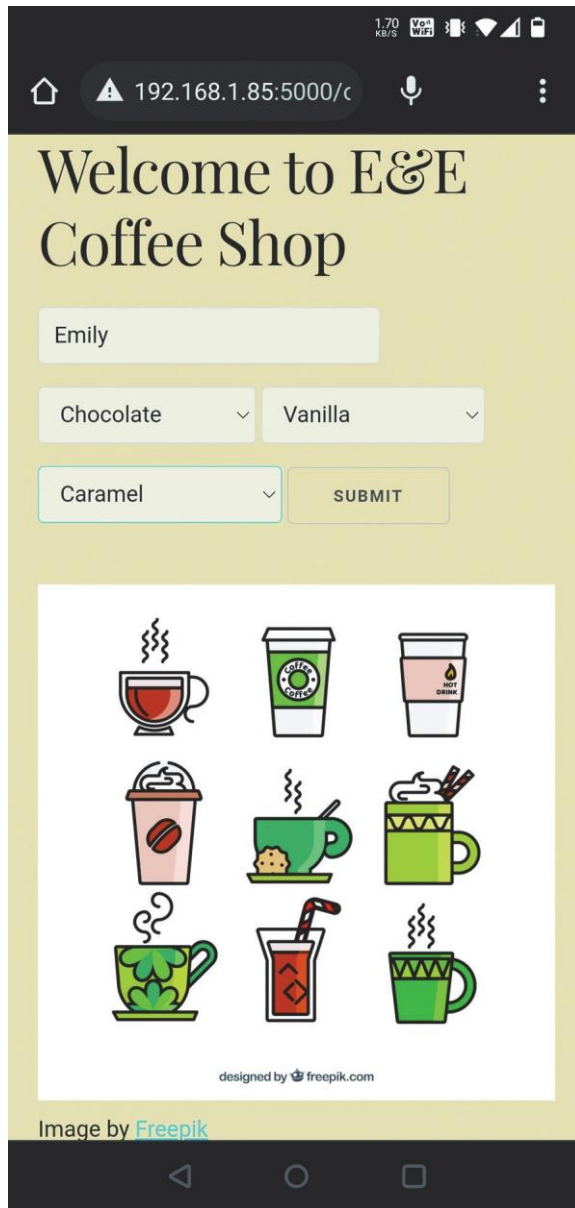
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.85:5000
[33mPress CTRL+C to quit [0m

* Restarting with stat
* Debugger is active!
* Debugger PIN: 675-232-744
```

«Хорошо, теперь откройте этот адрес на своих устройствах и покажите мне, что у вас появилось на странице заказа».

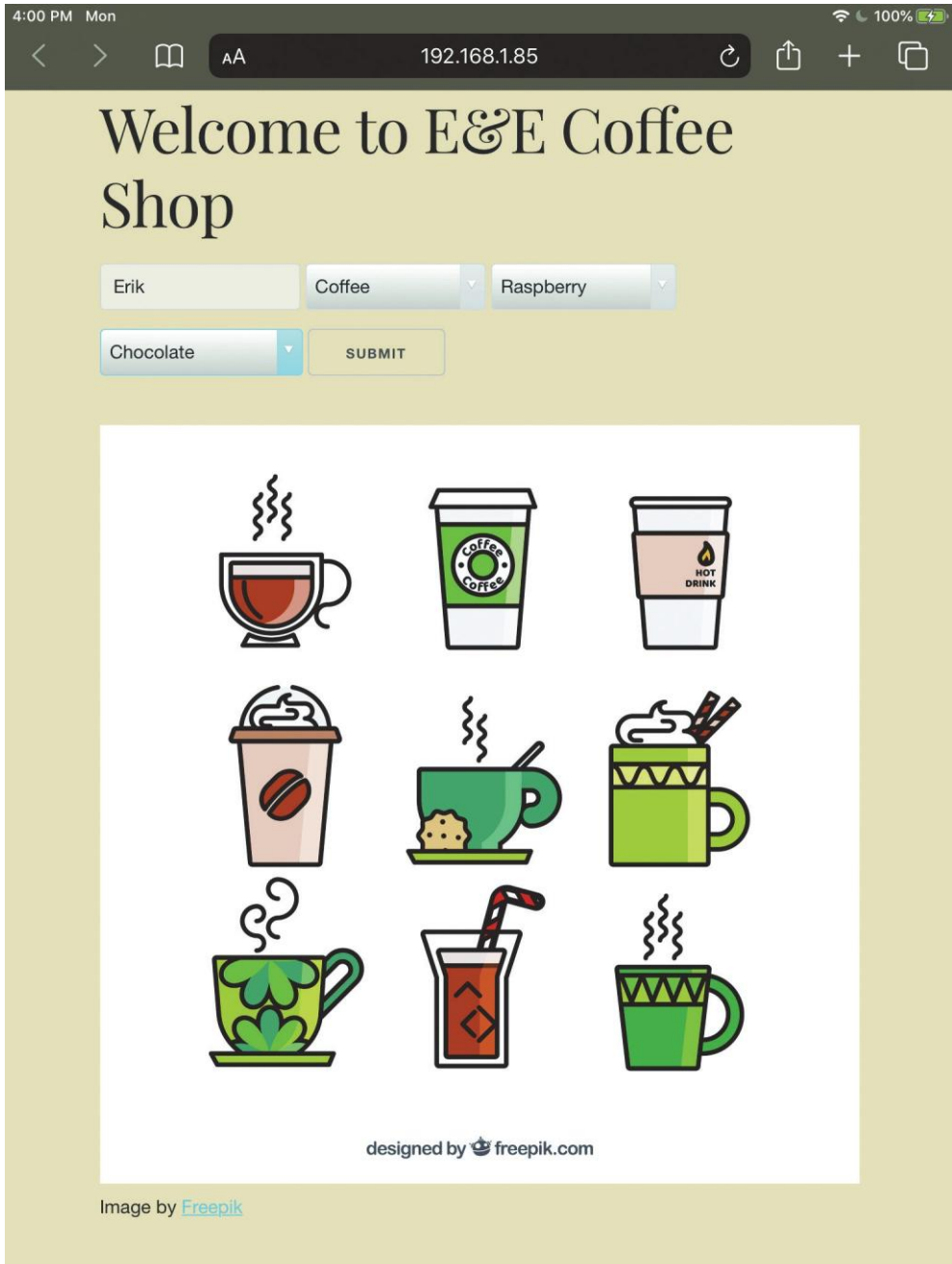
Эмили и Эрик открыли свои устройства и начали ввод.

«Есть!» — Эмили показала свой телефон.



«Выглядит отлично!» — сказал Саймон. — «Эрик, что у тебя на iPad?»

«Смотри!» — ответил Эрик.



«Приложение работает на обоих ваших устройствах и выглядит великолепно», — резюмировал Саймон. — «Помните, я говорил вам, что мы используем таблицы стилей, включенные в редактор Mu, и эти таблицы были разработаны для использования на разных устройствах. Здесь мы использовали ваше приложение на MacBook, iPad и Android-телефоне, и на всех из них оно выглядит великолепно. Более того, вы заметили, как наше меню было изменено в соответствии с размером страницы? В этом преимущество использования таблиц стилей `normalize.css` и `skeleton.css`. Мы только что добавили пару небольших изменений в нашу таблицу стилей `coffeeshop.css`».

«Замечательно, что клиенты в нашей кофейне могут заказывать напитки со своих телефонов», — замтила Эмили. — «Но для этого они должны находиться в кофейне и должны быть подключены к нашей сети, верно? Саймон, можем ли мы разместить наше приложение в реальном облаке, чтобы использовать его где угодно? Я бы хотела показать это своим друзьям в школе».

«Конечно, это возможно. Конечно, это несколько сложнее, чем просто добавить одну строку, как мы это только что сделали. Обычно облачные сервисы стоят денег, но для простых проектов вроде нашего можно найти бесплатный сервис. Один из них называется PythonAnywhere. Я знаю, что он может разместить ваше приложение Python, но сам я этого не пробовал. Дайте мне немного времени, я проверю и предоставлю вам инструкции».

**ПРИМЕЧАНИЕ** Инструкции по использованию PythonAnywhere вы найдете на сопутствующем веб-сайте книги <https://pythonicadventure.com>.

«Но прежде чем мы откроем приложение для всего мира, давайте поговорим о возможных улучшениях. Эрик, ты помнишь, что мы обсуждали при работе с базой данных?»

«Вы сказали, что было бы хорошо включить время заказа в таблицу базы данных», — напомнил Эрик.

«Да, верно! Давай начнем список дел, чтобы спланировать наши будущие улучшения». — он начал писать в текстовом редакторе:

```
* Timestamps in
  orders
```

«Мы можем запомнить каждого покупателя», — сказала Эмили, — «и показать ему, что он заказывал в прошлый раз».

«Хорошая идея», — сказал Саймон. — «Но в этом случае мы должны создать уникальное имя для каждого клиента. Мы не хотим спутать двух разных клиентов с именем Alex».

«А как насчет нашего меню?» — спросил Эрик. — «Когда мы запускаем приложение на моем ноутбуке, мы можем редактировать текстовые файлы. Но когда мы запускаем его в облаке, мы не можем редактировать файлы, не так ли?»

«Да, ты прав, Эрик», — сказал Саймон. — «Если мы переместимся в облако, все будет делаться посредством веб-браузера. Нам понадобится еще одна страница для редактирования меню. И не забывай, что этой страницей должны пользоваться только менеджеры кофейни».

«Я понимаю, что ты имеешь в виду», — сказала Эмили. — «Нам нужны пароли!»

«Верно», — подтвердил Саймон. — «Нам нужны разные учетные записи в нашем приложении. У нас будут учетные записи клиентов и учетные записи бариста, которые будут получать заказы».

«И у нас будут учетные записи менеджеров кофейни, где будут перечисляться заказы и где они будут редактировать меню и выполнять другие менеджерские функции».

«Нам нужно больше фотографий!» — сказала Эмили. — «Я имею в виду фотографии нашей продукции: всех наших вкусов и топингов».

«А как насчет рекламных акций?» — спросил Эрик. — «Вкус недели или что-то в этом роде. Саймон, ты сказал, что мы должны знать, сколько порций ароматизаторов и топингов мы использовали в течение дня или месяца. Можно ли отобразить это в Интернете?»

«Эй, эй, помедленнее, пожалуйста. Я не успеваю записывать все твои идеи», — сказал Саймон и улыбнулся. — «Серьезно, мне нравится, как мы проводим мозговой штурм. Один из основных принципов мозгового штурма — и мы часто используем его в нашем клубе робототехники — заключается в том, что критиковать идеи нельзя. Каждый может предложить самые безумные идеи. Мы не говорим: «О, это будет невозможно реализовать». Мы просто собираем *все* и обсуждаем позже».

«Тогда я хотел бы добавить голосовые команды для нашей кофейни», — сказал Эрик. — «И роботизированную руку, которая наливает кофе и добавляет вкусы и топинги».

«Принято к сведению», — сказал Саймон.

«Я думаю, что вместо паролей нам следует использовать распознавание лиц», — сказала Эмили. — «Пароли — это скучно. Представьте, что в кофейню заходит клиент, мы приветствуем его и говорим, ‘Эй, Макс, будешь, как обычно, с ванилью и карамелью?’».

«Отлично, тоже принял к сведению», — сказал Саймон. — «Вот что я собрал на данный момент».

Он показал свой список Эмили и Эрику:

- \* Метки времени в заказах
- \* Фиксация каждого клиента и того, что он заказал (создание учетных записей пользователей)
- \* Редактирование меню в браузере
- \* Учетные записи для менеджеров кофейни, бариста, кассиров, клиентов
- \* Список заказов за день, неделю, месяц; список всех заказов клиента
- \* Больше фотографий наших товаров
- \* Акции для товаров
- \* Отчеты: количество заказов с каждым вкусом и топингом в день, неделю, месяц
- \* Голосовые команды
- \* Роботы-манипуляторы для кофе, ароматизаторов и топингов
- \* Распознавание лиц для клиентов

«Также не забудь предложения, которые мы получили вчера от ИИ. Позвольте мне составить еще один список», — сказал Саймон и скопировал ответы из ChatGPT:

- \* Примеры кода: попросите ИИ просмотреть все наши функции и дать предложения по их улучшению
- \* Отладка: попросите ИИ проанализировать наши функции на наличие потенциальных ошибок
- \* Рекомендуемые библиотеки и инструменты: попросите ИИ дать нам рекомендации
- \* Разработка алгоритмов: попросите ИИ проанализировать наши алгоритмы (многие мы еще не использовали)
- \* Структура проекта: спросите у ИИ предложения по структуре проекта
- \* Создание тестов для наших функций (ИИ вчера нам не сказал, но я знаю, что люди используют его для тестирования)

«Работы на целый год хватит!» — сказал Саймон. — «Боюсь, в книге не хватит места, чтобы охватить все это».

«Что за книга?» — Эрик спросил.

«Эрик, ты не знаешь?» — спросил Саймон. — «Наш папа пишет книгу о нас — о тебе, Эмили и обо мне. Вся книга посвящена этому проекту кофейни и тому, как мы его разработали. Он сказал, что не может добавить в книгу больше глав, поэтому, боюсь, нам придется продолжить наш проект онлайн. Я предлагаю создать веб-сайт и обновлять его, отражая наши достижения и дополнительные функции, которые мы разрабатываем. Что вы об этом думаете?»

«Мне это нравится!» — сказала Эмили. — «Это будет что-то вроде блога, верно?»

«Да, типа блог! Возможно, нам также следует добавить репозиторий GitHub, где мы сможем хранить все наши программы и примеры».

«Великолепно!» — отозвался Эрик. — «Я слышал о GitHub — большинство разработчиков хранят там свой код».

«И когда мы это сделаем», — продолжил Саймон, — «другие люди смогут использовать наш код, учиться на его примере и, возможно,

добавлять свои собственные функции в наше приложение. Возможно, они поделятся своими приложениями, и мы им тоже поможем. Именно так сегодня разрабатывается большинство проектов».

"Отлично! Я в деле!" — сказала Эмили.

«Я тоже», — добавил Эрик.

«Отлично, тогда продолжим на нашем сайте. Я поговорю с папой, и мы вместе все устроим. Увидимся завтра!" — сказал Саймон. Ему понравилось работать над этим проектом с Эмили и Эриком. Благодаря проекту кофейни он сам узнал пару новых вещей.

«Пришло время узнать больше об учетных записях, роботах-манипуляторах и распознавании лиц, чтобы я мог объяснить это детям», — думал он.

#### **ПРИМЕЧАНИЕ**

Вот созданный ими сайт: <https://pythonicadventure.com/>. Посетите его и найдите новые идеи для своих проектов, советы по устранению ошибок и многое другое!

### **Новое, что вы узнали сегодня**

- *Домашняя страница* — первая страница, которую вы видите при посещении веб-сайта. Обычно он используется, чтобы сообщить посетителям, что они могут найти на сайте.
- *Сетевой адрес* — каждый компьютер, подключенный к сети, имеет адрес, по которому другие компьютеры могут найти его в сети. Если мы хотим получить доступ к веб-приложению на ноутбуке Эрика, нам нужно знать его сетевой адрес.

### **Код для этой главы**

Вы можете найти код для этой главы здесь:

<https://github.com/pavelanni/pythonicadventure-code/tree/main/ch15>.

# Приложение А

## Идеи для вашего первого приложения

---

Создание приложения для кофейни кажется вам не очень интересным? Создайте что-нибудь другое! Все идеи и методы программирования, которые я обсуждаю в этой книге, применимы ко многим другим проектам. Просто посмотрите вокруг, и вы получите идеи для других приложений.

Вот некоторые примеры:

### *Пиццерия*

Это должно быть очень похоже на приложение Coffee Shop. Вы можете спросить клиента о следующем:

- Какой у них любимый напиток?
- Какой вкус им нравится?
- Какой топинг им нравится?

Вы можете предоставить клиенту список вариантов по каждому из вопросов, а клиент выбирает нужный в меню.

Вы также можете спросить клиентов пиццерии следующее:

- Какое тесто вам нравится: тонкое или толстое?
- Какого диаметра — маленького, среднего или большого?
- Какой соус вам нравится — красный или белый?
- Какая пицца вам нравится: маргарита, пепперони или овощная? (Чтобы найти идеи, зайдите в свою любимую пиццерию и посмотрите, что там продается.)
- Какие дополнительные начинки вы хотите добавить?



## Кафе-мороженое

Зайдите в свое любимое кафе-мороженое и посмотрите, как готовят ваш заказ. Что они вас спрашивают? Какие варианты они вам предлагают? Они будут и в меню вашего приложения.

Скорее всего, вас спросят следующее:

- Какой рожок: сахарный или вафельный?
- Сколько шариков?
- Какие вкусы?
- Добавить какие-нибудь топинги?

Здесь есть небольшое отличие от кофейни. После того, как вы спросите: «Сколько шариков?» вам придется спросить вкус каждого шарика. Подумайте об этом: как бы вы реализовали это на Python?

Вот подсказка: в Python есть функция `range()`, которую можно использовать в цикле `for`. Мы использовали его в наших меню. Попробуйте использовать его, чтобы спросить о вкусе мороженого нужное количество раз.

## Минифигурки LEGO

Возможно, у вас есть хорошая коллекция минифигурок LEGO и их деталей, и вы хотите помочь своим друзьям построить что-то новое. Какие вопросы вы им зададите и какие варианты предложите? Ниже приведены некоторые примеры:

- *Выберите голову* — улыбающееся лицо, лицо в солнечных очках, лицо с бородой
- *Выберите головной убор* — темные волосы, светлые волосы, каска, полицейская фуражка
- *Выберите торс* — механик, полицейский, рубашка с галстуком, футболка
- *Выберите ноги* — синие джинсы, зеленые шорты, коричневые брюки-карго
- *Выберите аксессуар* — меч, радио, молоток, увеличительное стекло

Вы можете добавить в заявку особые условия. Например, если ваш друг выбрал полицейскую фуражку, он не сможет выбрать в качестве аксессуара бейсбольную битку. Подумайте о добавлении этого условия в ваше меню.

А как насчет случайного выбора деталей? Благодаря этому могут получиться забавные минифигурки. Как бы вы добавили случайный вариант в свое меню? Как бы вы это реализовали?

Вот подсказка: в Python есть модуль под названием `random`. Вам следует импортировать его с помощью оператора `import` в начале вашей программы и использовать функцию `choice()`. Вы можете задать для этой функции список вариантов, и она выберет один из них случайным образом. В следующий раз, когда вы ее вызовете, функция случайным образом выберет что-то другое (или, может быть, тот же элемент — это случайно!). Например, создайте следующую короткую программу и запустите ее. В этой программе мы пять раз просим Python случайным образом выбрать элемент из списка трех типов волос.

**Листинг A.1** `choice.py`

```
import random

for _ in range(5):
    print(random.choice(["dark hair", "blond hair", "red hair"]))
```

Запустите программу `choice.py`, и вы увидите что-то вроде этого:

```
blond hair
red hair
blond hair
blond hair
dark hair
```

Конечно, ваш список будет другим и, вероятно, он будет содержать еще 5 случайных вариантов в другом порядке.

### ***Идеи других проектов***

Есть ли у вас идеи других проектов? Пожалуйста, поделитесь ими на форуме liveBook: <https://livebook.manning.com/book/a-pythonic-adventure/discussion>.

# Приложение В

## Как установить редактор Mu и среду Python

---

В этом приложении я объясню, как установить Python на ваш компьютер. Самый простой способ — установить редактор программирования, содержащий Python. Я рекомендую вам установить редактор Mu. Я использую его в этой книге, поэтому вам будет легко следовать диалогам и инструкциям книги, если вы используете тот же редактор.

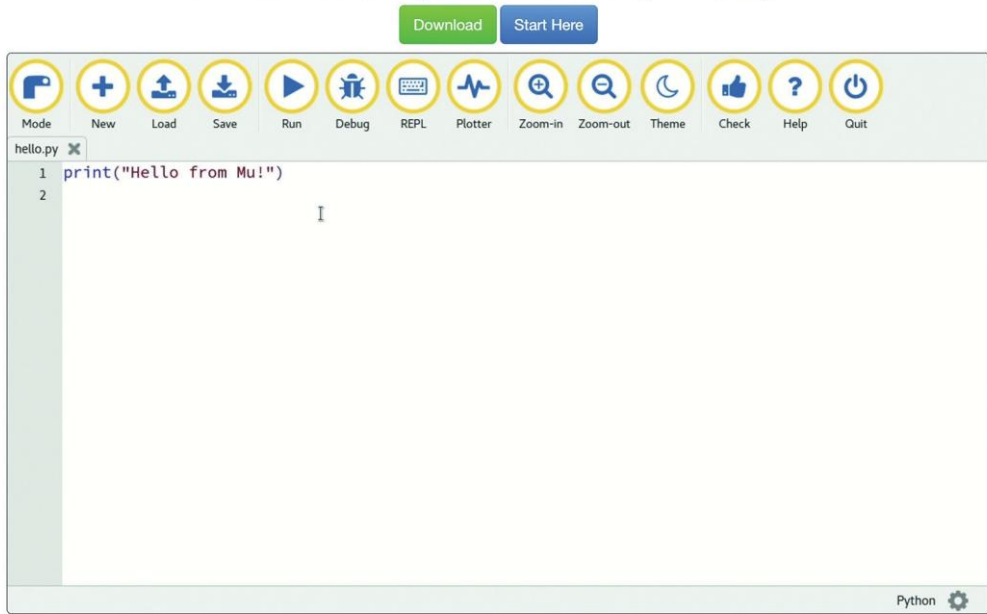
Я также дам вам ссылки на другие способы установки Python — попробуйте и их.

### **Mu**

Выполните следующие действия, чтобы установить Mu:

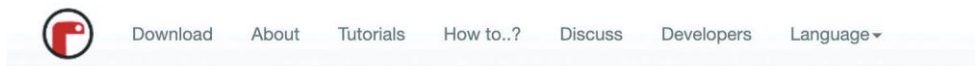
- 1 В веб-браузере откройте веб-страницу редактора Mu:  
<https://codewith.mu/>.

Code with Mu: a simple Python editor for beginner programmers.



© 2022 Nicholas H.Tollervey. Mu wouldn't be possible without [these people](#). ❤️ This site is licensed under the [Creative Commons by-nc-sa 4.0 International License](#).

2 Нажмите Download (зеленая кнопка). Вы увидите страницу загрузки Mu.



## Download Mu

The simplest and easiest way to get Mu is via the official installer for Windows or Mac OSX (we no longer support 32bit Windows). We also have an experimental Appliance for Linux users running on Intel based hardware.

The current recommended version is Mu 1.2.0. We advise people to update to this version via the links for each supported operating system. All previous beta versions of Mu [can be downloaded from here](#).



### Windows Installer

[Download](#)[Instructions](#)

### Mac OSX Installer

[Download](#)[Instructions](#)

### Linux Appliance Package (Experimental)

[Download](#)[Instructions](#)

- 3 Нажмите Download напротив вашей операционной системы. Ваш браузер загрузит для вашей операционной системы следующий установочный файл:
  - Для Windows это будет файл .msi
  - Для macOS это будет файл .dmg
  - Для Linux это будет файл .AppImage
- 4 Нажмите Instructions напротив вашей операционной системы и следуйте появившимся инструкциям.
- 5 Откройте Mu, как вы обычно открываете приложения в своей операционной системе. Теперь вы готовы работать над своим проектом!

Вы также можете использовать редактор Mu для программирования микроконтроллеров и сборки роботов, но это тема для другой книги.

## **Thonny**

Thonny — еще один замечательный редактор Python, созданный для новичков. Вы можете найти его здесь: <https://thonny.org/>.

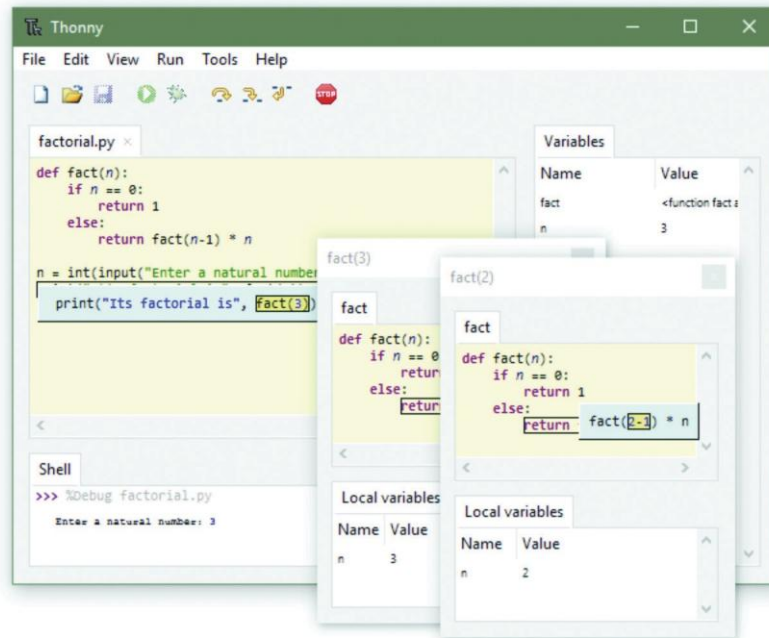
На первой странице вы найдете установщики для Windows, macOS и Linux. Для Windows загрузите файл .exe и запустите его. Для macOS загрузите файл .pkg и установите его. Для Linux выполните команду, указанную в инструкции.

# Thonny

Python IDE for beginners



Download version **4.0.1** for  
[Windows](#) • [Mac](#) • [Linux](#)



После того, как вы установили приложение, запустите его и изучите его настройки. Вы можете выбрать цветовую тему из десятка вариантов, шрифты редактора и терминала и множество других параметров.

У Thonny есть очень полезная функция под названием Assistant (Помощник). В меню Options вы можете настроить его запуск каждый раз, когда в вашем коде появляется предупреждение. Он также запускается, когда ваша программа показывает ошибку. Assistant даст вам несколько советов о том, что может быть не так с вашим кодом. Попробуйте допустить незначительную ошибку в своем коде (например, опечатку в имени переменной), а затем запустите программу. Вы увидите Assistant в действии.

## Python

Оба предыдущих редактора включают Python в свои установочные пакеты. Но по какой-то причине вы можете захотеть установить Python отдельно..

Если вы используете macOS или Linux, в вашей операционной системе уже установлен Python. Скорее всего, это не последняя версия Python, но это не проблема: все программы, которые мы разрабатываем в этой книге, будут работать с версиями Python, начиная с 3.5. В этих операционных системах не нужно ничего устанавливать — по крайней мере, для изучения этой книги.

Если вы используете Windows, вам придется зайти на официальный сайт Python и загрузить установщик оттуда: <https://www.python.org/downloads/windows/>.

Пожалуйста, внимательно прочитайте примечания и выберите правильную версию Python для вашей версии Windows.

## Python Releases for Windows

- [Latest Python 3 Release - Python 3.11.0](#)

### Stable Releases

- [Python 3.11.0 - Oct. 24, 2022](#)

**Note that Python 3.11.0 cannot be used on Windows 7 or earlier.**

- Download [Windows embeddable package \(32-bit\)](#)
- Download [Windows embeddable package \(64-bit\)](#)
- Download [Windows embeddable package \(ARM64\)](#)
- Download [Windows installer \(32-bit\)](#)
- Download [Windows installer \(64-bit\)](#)
- Download [Windows installer \(ARM64\)](#)

- [Python 3.9.15 - Oct. 11, 2022](#)

**Note that Python 3.9.15 cannot be used on Windows 7 or earlier.**

- No files for this release.

- [Python 3.8.15 - Oct. 11, 2022](#)

**Note that Python 3.8.15 cannot be used on Windows XP or earlier.**

- No files for this release.

- [Python 3.10.8 - Oct. 11, 2022](#)

**Note that Python 3.10.8 cannot be used on Windows 7 or earlier.**

Другой способ установить Python в Windows — открыть окно PowerShell и ввести `python`. Windows предложит правильную версию Python для установки. Вам просто нужно ее принять.