

ChatGPT:

ваш наставник по Python

Овладение основами за 100 запросов

Серхио Рохас-Галеано

ChatGPT: ваш наставник по Python

Овладение основами за 100 запросов

Серхио Рохас-Галеано, доктор философии

Богота, Колумбия

Об авторе

Серхио Рохас-Галеано — увлеченный ученый-компьютерщик, страстно любящий смежные области между искусственным интеллектом, машинным обучением, эволюционными вычислениями, обработкой естественного языка, разработкой программного обеспечения и научной публицистикой. Имея профессиональный опыт работы как в академических кругах, так и в промышленности в Колумбии и Великобритании, он глубоко привержен соблюдению этических стандартов и развитию области компьютерных наук. В настоящее время он работает профессором в Университете округа Франсиско Хосе де Калдас (Колумбия), где более двух десятилетий преподает программирование и вычислительный интеллект. Имея докторскую степень в области компьютерных наук Университетского колледжа Лондона (UCL) в Великобритании (2009 г.), он также работал в UCL (2022 г.) и Университете Малаги в Испании (2023 г.). Он является автором или соавтором более 70 научных работ и ранее опубликовал книгу «*Модели обучения и оптимизации для специалистов по данным*» (LeanPub, 2019).

email: sargaleano@gmail.com

Лицензия

Этот документ лицензируется как:



Разрешение на использование этого документа зависит от Лицензии; любое другое использование строго запрещено. Содержание этого документа предоставляется на условиях «КАК ЕСТЬ» без каких-либо гарантий, явных или подразумеваемых. Автор не несет никакой ответственности за любые последствия, возникающие в результате использования этого документа, включая, помимо прочего, ошибки, упущения или неточности. Читатель берет на себя полную ответственность за применение концепций и методов, описанных в этой книге.

© 2023, Серхио Рохас-Галеано.

Выходные данные

Этот документ был набран с использованием \LaTeX и класса kaobook. Автор создал контент с помощью системы генерации контента OpenAI ChatGPT, основываясь на собственных идеях, целях и стиле. Дизайн обложки был создан автором с использованием шаблонов и рисунков, доступных на платформе графического дизайна Canva. Все названия продуктов, логотипы, бренды, товарные знаки и зарегистрированные товарные знаки являются собственностью соответствующих владельцев. Все названия компаний, продуктов и услуг, а также названия знаменитостей, книг, персонажей, групп, команд или фильмов, упомянутые в этой книге, используются исключительно в целях идентификации и в образовательных целях. Использование этих названий, товарных знаков и брендов не подразумевает получение положительного заключения.

Первое издание

июнь 2023 г.

Богота, Колумбия

Код подобен юмору. Если его приходится объяснять, это плохо.

- Кори Хаус

Предисловие

Добро пожаловать в *ChatGPT: ваш наставник по Python. Владение основами за 100 запросов!* Если вы читаете это, значит, вы собираетесь отправиться в захватывающее путешествие в мир программирования. Но почему Python, спросите вы? Ну, Python — один из наиболее широко используемых и универсальных языков программирования, с приложениями, варьирующимися от веб-разработки и анализа данных до машинного обучения и искусственного интеллекта (ИИ).

Фактически, в последние годы наблюдается растущий интерес к ИИ и его приложениям, включая разработку языковых моделей, таких как ChatGPT. Хотя существуют обоснованные опасения относительно этических последствий, предвзятости данных и возможности создания поддельного контента, стоит отметить, что в сфере программирования эти опасения можно несколько смягчить, поскольку контент, генерируемый ChatGPT, в основном состоит из кода, который можно легко оценить на предмет достоверности и уместности, просто запустив его. Это дает преимущество в минимизации рисков, связанных с галлюцинациями или неточной информацией, поскольку результат можно проверить на практике. Таким образом, языковые модели, такие как ChatGPT, стали ценными инструментами для задач программирования, предлагая разработчикам расширенную помощь в их усилиях по кодированию.

Это руководство было составлено с особым тщанием, чтобы помочь вам с легкостью разобраться в основных концепциях программирования на Python, а ChatGPT при этом станет вашим надежным наставником. ChatGPT — это мощная языковая модель, разработанная, чтобы помочь вам с любыми вопросами или сомнениями, которые могут у вас возникнуть при изучении Python. Она предоставит вам примеры, объяснения и отзывы, что сделает ваш процесс обучения интерактивным и увлекательным.

В этом руководстве вы изучите основы программирования Python, включая переменные, типы данных, операторы, структуры управления потоком, такие как условные операторы и циклы, контейнеры, такие как списки и словари, а также концепции модульности и повторного использования. Вы также сможете поработать над веселыми и увлекательными задачами, которые проверят ваши новообретенные навыки.

Но это руководство — больше, чем просто набор упражнений и примеров. Это приглашение изучить возможности программирования и его потенциальное влияние на нашу жизнь, а также увидеть, как ИИ и языковые модели, такие как ChatGPT, будут использовать способности и производительность программистов для создания мощных приложений для многообещающего будущего. Погружаясь глубже в мир Python, вы увидите, как программирование можно использовать для решения реальных проблем, автоматизации утомительных задач и инновационных

продуктов и услуг.

Большая часть содержания данной книги, включая это предисловие, изначально создавалась с помощью ChatGPT. Однако важно отметить, что черновой материал послужил основой, которую я затем тщательно переработал, скорректировал и обогатил. Этот процесс был обусловлен моим образовательным опытом с целью создания дидактического и информативного контента для читателей. Хотя ChatGPT сыграл роль в создании начального материала, окончательный контент был тщательно отобран и адаптирован для обеспечения его образовательной ценности и корректности.

Чтобы способствовать прозрачности этого инновационного подхода к написанию книги с помощью ИИ, я намеренно включил полный список из 100 запросов, которые я придумал и использовал для подачи ChatGPT, что обеспечивает полный охват предполагаемой тематики данного руководства. Этот список будет помещен в конце книги для удобства использования. Кроме того, этот *список запросов* может дать читателям возможность воспроизвести аналогичную, но персонализированную версию книги, соответствующую их индивидуальным предпочтениям и требованиям.

Кроме того, я буду использовать примечания для ссылки на каждый запрос с соответствующей позицией в списке всякий раз, когда он используется в тексте. Например, первый запрос, использованный для создания первоначального варианта этого предисловия, упоминается как следующее примечание¹.

Наконец, я создал сопутствующий репозиторий, в котором хранится коллекция Jupyter Notebooks, содержащих примеры кода Python из книги, продуманно организованные по главам для удобного просмотра и исследования:

<https://github.com/Sargaleano/chatgpt-python-coach>

Кроме того, начиная с Главы 4, я добавил интересную функцию. Для более привлекательных примеров кода я включил ссылки на веб-сайт [PythonTutor.com](https://python.tutor.com). Эти ссылки позволяют визуализировать пошаговое выполнение кода, динамически наблюдая за состоянием памяти программы. Ищите эти ссылки в нижней части списков кодов и полей вывода. Это фантастический способ улучшить ваше понимание и оживить код!

Итак, если вы хотите научиться программировать и начать создавать программы с помощью Python и ChatGPT, я надеюсь, что вы найдете это руководство полезным и приятным для чтения.

Серхио Рохас-Галеано
Богота, 23 июня 2023 г

Содержание

Предисловие	v
Содержание	vii
Введение	3
Это подходящая книга для вас?.....	3
О Python.....	3
Цикл программирования.....	4
Инструментарий.....	5
Использование Colab	6
Использование PythonTutor	8
ChatGPT как наставник по программированию	10
Использование ChatGPT	11
Шаблон запроса ChatGPT	14
1 Знакомство с основами	15
1.1 Hello, World!	15
1.2 Строки и форматирование строк.....	17
1.3 Время поработать руками!	22
2 Строительные блоки данных	27
2.1 Типы данных	27
2.2 Базовый ввод/вывод.....	30
2.3 Товарищ по кодированию.....	32
2.4 Время поработать руками!	36
3 Аппаратная	39
3.1 Арифметические операторы	39
3.2 Операторы сравнения	41
3.3 Логические операторы.....	43
3.4 Побитовые операторы	44
3.5 Операторы присваивания	48
3.6 Операторы идентификации.....	49
3.7 Операторы принадлежности.....	51
3.8 Оператор возведения в степень	52
3.9 Расширенные математические операторы	53
3.10 Время поработать руками!	55
4 Башня управления	59
4.1 Операторы If/Else.....	59
4.2 Операторы соответствия.....	64
4.3 Операторы цикла for.....	66
4.4 Операторы цикла while.....	69
4.5 Вложенные управляющие структуры	74

4.6	Время поработать руками!	81
5	Зона контейнеров	91
5.1	Списки	91
5.2	Кортежи	99
5.3	Наборы	103
5.4	Строки	107
5.5	Словари	113
5.6	Время поработать руками!	126
6	Модульные отношения	133
6.1	Модульность и возможность повторного использования	133
6.2	Функции	136
6.3	Классы и объекты.....	143
6.4	Время поработать руками!	153
7	Финал	161
7.1	Копаем дальше	161
7.2	Предлагаемое для дополнительного чтения.....	167
	Список из 100 запросов	169

Список рисунков

1	Цикл разработки программы.....	4
2	Главный экран Colab	7
3	Блокнот без названия	7
4	Блокнот с текстом и ячейками кода.....	8
5	Веб-сайт PythonTutor	9
6	Экран визуализации PythonTutor	9
7	Страница приветствия ChatGPT	11
8	Страница входа в ChatGPT	12
9	Домашняя страница ChatGPT	12
10	Страница чата ChatGPT	13
7.1	Рекомендуемый курс обучения веб-разработке на Python.....	161
7.2	Предлагаемый курс обучения Data Science на Python.....	162
7.3	Предлагаемый курс обучения Python для автоматизации и создания сценариев.....	162
7.4	Рекомендуемый курс обучения Python для разработки игр.....	163
7.5	Предлагаемый курс обучения Python для DevOps.....	163
7.6	Предлагаемый курс обучения по расширенным концепциям Python.....	164
7.7	Предлагаемый курс обучения для тестирования и отладки на Python	164
7.8	Предлагаемый курс обучения по дополнительным ресурсам Python	165

Введение

Это подходящая для вас книга?

Эта книга предназначена для того, чтобы помочь новичкам, таким как вы, с легкостью освоить основы программирования, особенно на языке Python, одном из самых универсальных и широко используемых языков.

Но эта книга — больше, чем просто типичное руководство по программированию. Это как будто рядом с вами находится преданный наставник, который направляет вас в процессе обучения примерами, объяснениями и отзывами. И этим наставником является языковая модель ИИ — ChatGPT. Используя ChatGPT, вы получите доступ к мощному инструменту, который поможет вам понять основные концепции программирования. Вы будете получать в режиме реального времени обратную связь, которая поможет вам учиться быстрее и эффективнее, что в конечном итоге поможет вам стать опытным программистом².

Эта книга служит вводным руководством по фундаментальным концепциям Python, включая алгоритмы, структуры управления потоком, контейнеры и модульное проектирование. Овладение этими базовыми знаниями даст вам прочную основу для изучения более сложных тем в других книгах. Итак, примите эти фундаментальные концепции и приготовьтесь к захватывающим приключениям, которые ждут вас впереди. А пока в этой главе представлены инструменты, которые будут использоваться на протяжении всей книги. Если вы уже знакомы с этими инструментами и хотите сразу погрузиться в программирование, можете пропустить эту главу и перейти к Главе 1.

О Python

Python — это язык программирования, который так же прост, как пирог, или, скорее, так же прост, как написать рецепт вашего любимого десерта! Он был создан в конце 1980-х годов голландским программистом по имени Гвидо ван Россум, который назвал его в честь комедийной труппы Monty Python³.

Сегодня Python используется всеми, от НАСА до Instagram, для создания всего, от космических марсоходов до фотофильтров.

Содержание главы	
Эта книга для вас?.....	3
О Python.....	3
Цикл программирования.....	4
Инструментарий.....	5
Использование Colab.....	6
Использование PythonTutor.....	8
ChatGPT как наставник.....	10
Использование ChatGPT.....	11
Шаблоны запросов.....	13

Monty Python
Monty Python — британская комедийная группа, известная своим уникальным и абсурдистским юмором. Их телешоу «Летающий цирк Монти Пайтона» транслировалось с 1969 по 1974 год и стало культовым. Они также создали несколько популярных фильмов, в том числе «Монти Пайтон и Святой Грааль» и «Жизнь Брайана». Их влияние на комедию и поп-культуру было значительным, и поклонники по всему миру продолжают это отмечать ⁴ .

Версии Python

Вот некоторые ключевые отличия последних 5 версий.

- Python 2.x имеет ограниченную поддержку неанглийских символов, тогда как Python 3.6+ использует кодировку Unicode для других языков.

- Python 3.6+ позволяет расширенное форматирование текста для вывода на печать по сравнению с предыдущими версиями.

- Python 2.x возвращает целые числа для целочисленных операндов, тогда как Python 3.6+ предоставляет как десятичное деление по умолчанию, так и отдельную команду целочисленного деления.

- В Python 3.5 и более поздних версиях добавлена возможность одновременного управления несколькими задачами.

- Python 3.10 обрабатывает несколько условий с помощью одного теста соответствия.

- Python 3.11 — последняя версия (апрель 2023 г.).

- Для получения обновленной информации о выпусках Python см.: <https://devguide.python.org/versions/>.

Он известен своей простотой, удобочитаемостью и универсальностью. Фактически, это один из самых популярных языков программирования в мире, и на то есть веские причины. Это язык, который легко выучить, но при этом он обладает мощными возможностями, что делает его идеальным как для начинающих, так и для экспертов. Будущее Python весьма многообещающее, так как ожидается, что в ближайшие годы он станет еще более популярным из-за его растущего использования в таких областях, как машинное обучение, искусственный интеллект и наука о данных.

В Интернете доступно множество ресурсов Python. Официальный веб-сайт Python (<https://www.python.org/>) содержит огромное количество информации, включая учебные пособия, документацию и файлы для загрузки. Другие полезные веб-сайты включают Real Python (<https://realpython.com/>), Python для начинающих (<https://www.pythonforbeginners.com/>) и Codecademy (<https://www.codecademy.com/learn/learn-python>).

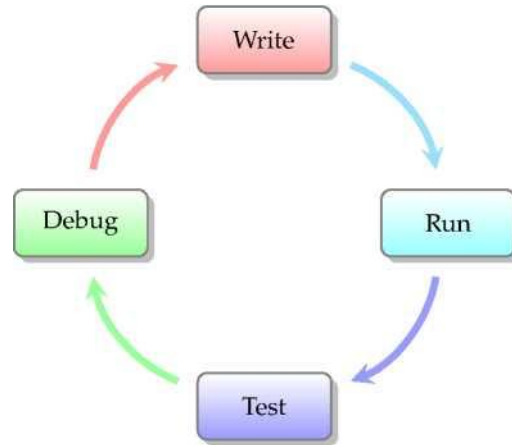
Цикл программирования

Программирование — это творческий процесс, требующий тщательного планирования, сосредоточенности и внимания к деталям. Независимо от того, пишете ли вы простой сценарий или сложное приложение, существует стандартный цикл разработки программы, которому вы должны следовать, чтобы ваш код работал должным образом. Четыре ключевых шага этого цикла показаны на Рис. 1⁶.

Давайте подробно рассмотрим каждый шаг⁷:

- **Написание (кодирование):** Первым шагом является написание кода. Здесь вы превращаете свои идеи в настоящие инструкции, понятные компьютеру. Вы используете язык программирования, такой как Python, для написания своего кода, и вам нужно уделять пристальное внимание синтаксису и структуре вашего кода.
- **Запуск:** После того, как вы написали свой код, пришло время его запустить. Вы увидите свой код в действии и проверите, работает ли он так, как задумано. Для запуска вашего кода вы можете использовать такую программу, как IDLE, либо вы можете использовать интерфейс командной строки.

Рис. 1: Типовой цикл разработки программы



- ▶ **Тестирование:** Тестирование является неотъемлемой частью процесса разработки программы. Так вы проверяете, правильно ли работает ваш код, и выявляете любые ошибки или сбои. Вам нужно будет запускать различные тестовые сценарии, чтобы убедиться, что ваш код надежен и работает в различных ситуациях.
- ▶ **Исправление (отладка):** Наконец, если вы обнаружили какие-либо ошибки или сбои во время тестирования, пришло время их исправить. Этот процесс называется отладкой программы. Вам нужно будет вернуться к своему коду и определить причину ошибки. Это может включать пошаговое выполнение вашего кода построчно или использование инструментов отладки для отслеживания потока вашего кода. После того, как вы определили проблему, вы можете внести необходимые изменения в свой код, чтобы устранить ее.

Помните, что программирование – это путь непрерывного обучения и совершенствования. Требуется время и усилия, чтобы развить хорошие навыки программирования. Примите вызовы, которые встречаются на вашем пути, и не бойтесь просить о помощи, когда это необходимо. Придерживаясь этих принципов и посвятив себя своему делу, вы будете на пути к созданию программного обеспечения, которое не только отвечает потребностям ваших пользователей, но и может оказать положительное влияние на мир.

Инструментарий

Python – это интерпретируемый компьютерный язык, что означает, что код выполняется непосредственно построчно, минуя этап компиляции. Это отличает его от компилируемых языков, таких как C++, Java или Go, где исходный код компилируется в машинный код, который затем может выполняться компьютером⁸.

Одним из преимуществ интерпретируемого языка, такого как Python, является то, что он обеспечивает более быструю разработку и прототипирование. Поскольку код может выполняться немедленно, разработчики могут быстро тестировать и отлаживать свои программы. Тем не менее, интерпретируемые языки могут работать медленнее, чем

компилируемые языки, поскольку код должен интерпретироваться заново при каждом запуске программы. Это может быть проблемой для программ, требующих высокой производительности или работающих с большими массивами данных.

Существует множество интегрированных сред разработки Python (IDE), доступных для выполнения цикла разработки, показанного на Рис. 1. Это такие приложения, как PyCharm, Spyder и Visual Studio Code, которые предлагают расширенные функции, помогающие опытным разработчикам ускорить процесс разработки и повысить производительность труда.

Однако в этой книге мы хотим сосредоточиться на концепциях и алгоритмах программирования без неудобств, связанных с установкой программного обеспечения.

Поэтому мы будем использовать в качестве основной платформы для написания, запуска и тестирования нашего кода Python готовые онлайн-инструменты. Эти инструменты подробно описаны ниже:

- ▶ **Jupyter Notebooks:** Интерактивная вычислительная среда в сети Интернет, позволяющая создавать и совместно использовать документы, содержащие живой код, уравнения, визуализации и описательный текст. Сайт: <https://jupyter.org/>. Для ее использования не требуется учетная запись.
- ▶ **Google Colaboratory (Colab):** Бесплатная облачная среда Jupyter Notebook, предоставляемая Google. Это позволяет вам запускать код Python в вашем браузере без необходимости какой-либо настройки. Чтобы запустить Colaboratory, вам просто нужно войти в свою учетную запись Google, перейти на веб-сайт (<https://colab.research.google.com/>) и нажать "New Notebook", чтобы создать новый блокнот. Там вы можете начать писать код Python и запускать его в режиме реального времени. Ваша работа будет сохранена в вашей учетной записи Google Диска.
- ▶ **PythonTutor:** Веб-визуализатор кода Python, который помогает учащимся понять, как код выполняется интерпретатором Python. Чтобы использовать PythonTutor, вам нужно зайти на сайт (<http://pythontutor.com/>); отсюда вы можете либо вставить свой код Python, либо написать его прямо на сайте и нажать "Visualize Execution". Для его использования не требуется учетная запись.
- ▶ **ChatGPT:** Языковая модель, которая может помочь в обучении, кодировании, объяснении, отладке, документировании и предоставлении отзывов о программах Python. Сайт: <https://chat.openai.com>. Требуется учетная запись OpenAI.

Использование Colab

Давайте посмотрим, как создать новый блокнот Jupyter в Google Colab, где мы можем выполнять базовое программирование на Python. Во-первых, войдите в свою личную учетную запись Google (если вам это нужно, перейдите по этой ссылке, чтобы создать ее: <https://support.google.com/accounts/answer/27441?hl=en>). Затем выполните следующие действия⁹:

7 INTRODUCTION

1. Перейдите на <https://colab.research.google.com/> в вашем веб-браузере, чтобы открыть Colab. Откроется окно, показанное на Рис. 2.
2. Выберите *File* → *New Notebook* в строке меню. Это создаст новый блокнот без названия. Вы можете присвоить ему название, щелкнув текст "Untitled0.ipynb" в верхней части страницы и введя новое имя, как показано на Рис. 3.

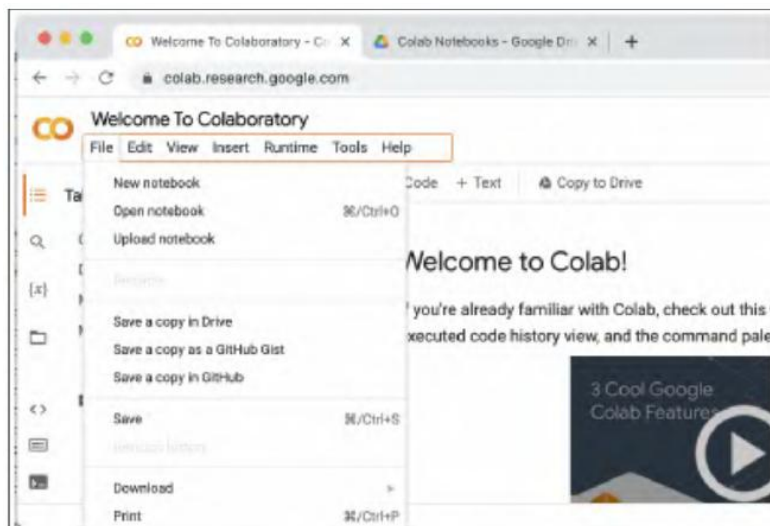


Рис. 2: Главный экран Colab.

3. Теперь можно приступать к записи содержимого в ячейки блокнота. В Jupyter Notebook¹⁰ существует два типа ячеек:

- ▶ **Ячейки кода:** Эти ячейки позволяют пользователю записывать и выполнять код в блокноте. Ячейки кода обозначаются ключом `in []` в левой части ячейки. Когда вы выполняете код в ячейке, выходные данные отображаются в текстовом поле под ячейкой, а в квадратных скобках слева от ячейки появляется число, указывающее, каким по порядку он был выполнен. Colab поддерживает различные языки программирования, но по умолчанию используется Python.
- ▶ **Ячейки разметки:** Эти ячейки позволяют пользователю добавлять в Блокнот текст, изображения и другие мультимедийные элементы. Ячейки разметки используют язык Markdown, который представляет собой упрощенный язык разметки, позволяющий легко форматировать текст с помощью простого синтаксиса. Когда вы запускаете ячейку разметки, текст отображается как форматированный текст или мультимедиа контент.

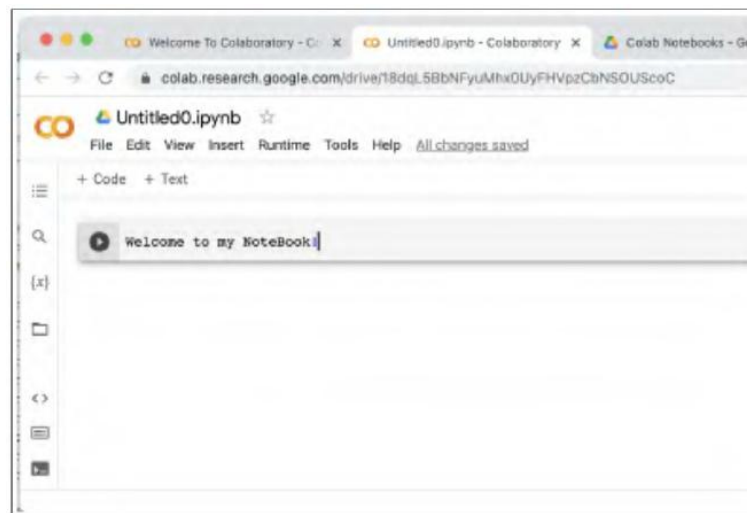


Рис. 3: Блокнот без названия.

4. Вы можете запустить код в ячейке, нажав кнопку `>` в левой части ячейки или используя сочетание клавиш "Shift+Enter".
5. Если вам нужно добавить больше ячеек, вы можете нажать на кнопку `+` в левом верхнем углу страницы или перейти в меню *Insert* `-> Code Cell` or *Insert* `-> Text Cell`.
6. Чтобы сохранить блокнот, вы можете либо щелкнуть *Tile* `-> Save`, либо использовать сочетание клавиш "Ctrl+S".
7. Colab автоматически сохраняет ваш блокнот на вашем Google

Диске, поэтому вы можете получить к нему доступ с любого устройства, если вы вошли в ту же учетную запись Google.

На Рис. 4 показан пример блокнота, который включает ячейку разметки с вступительным текстом и ячейку кода с текстом программы «Hello, World!». После запуска кода также можно увидеть выведенные результаты.

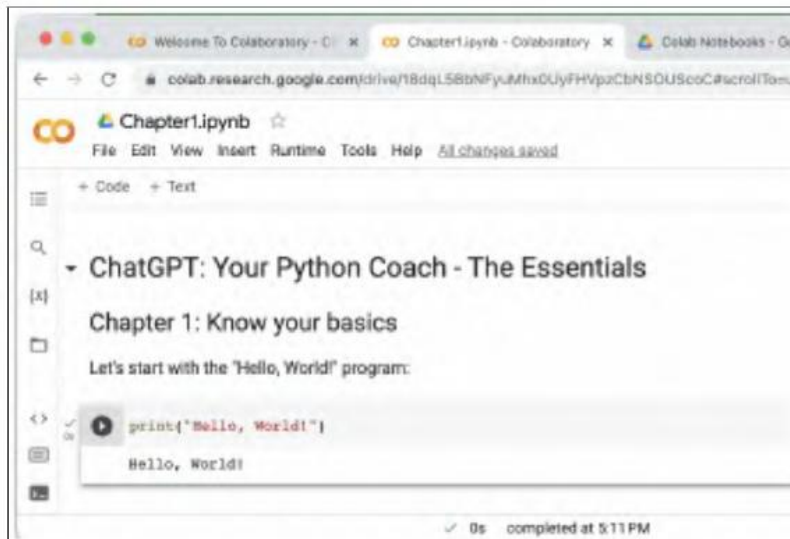


Рис. 4: Блокнот с текстом и кодовыми ячейками.

Использование PythonTutor

PythonTutor — это онлайн-инструмент, который позволит вам шаг за шагом визуализировать выполнение кода Python. Он может помочь вам понять, как работает код Python, показывая вам, как код выполняется построчно и как значения переменных меняются во времени¹¹.

Чтобы использовать PythonTutor, вам необходимо выполнить следующие шаги:

1. Перейдите на веб-сайт PythonTutor по адресу www.pythontutor.com. Откроется экран, показанный на Рис. 5.
2. Выберите язык Python, нажав соответствующую кнопку. Выберите версию Python 3.6.
3. Введите код Python в предложенное текстовое поле.

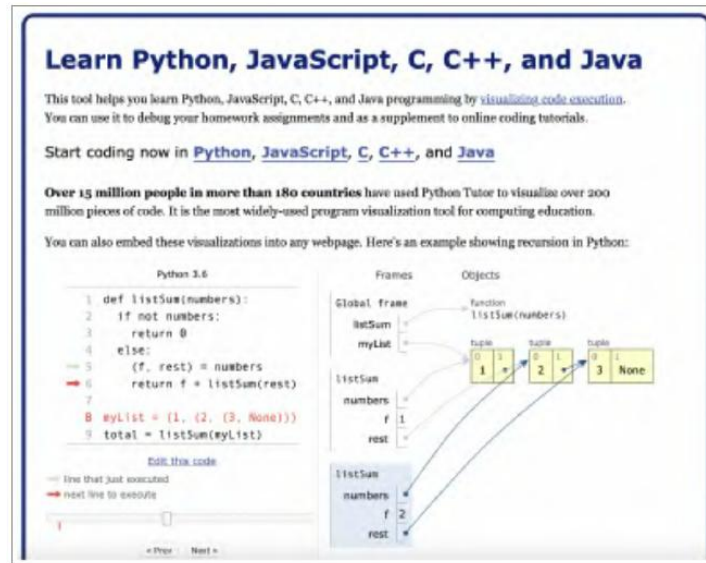


Рис. 5: Веб-сайт Python Tutor, где вы можете визуализировать и пошагово выполнять код

4. Выберите параметры визуализации, установив флажки и выбрав настройки из раскрывающихся меню. Вы можете просмотреть результат вывода кода, значения переменных и стек вызовов, а также другие параметры.
5. Нажмите кнопку "Visualize Execution", чтобы начать визуализацию.
6. После запуска визуализации вы увидите временную шкалу в левой части экрана, которая показывает каждый шаг выполнения вашего кода. Вы можете выполнять код по одной строке за раз, нажимая кнопку "Next" или "Prev", либо использовать ползунок для прокрутки его выполнения. Пример визуализации программы можно увидеть на Рис. 6.

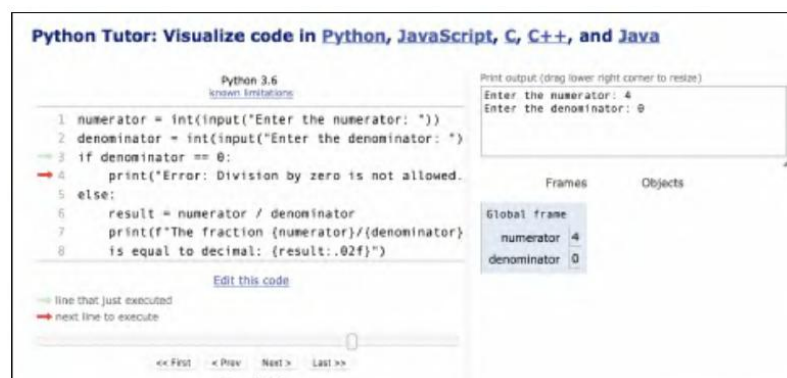


Рис. 6: Экран визуализации Python Tutor.

7. По мере выполнения кода в правой части экрана будет отображаться текущее состояние программы, включая значения переменных и любые выходные данные, сгенерированные кодом.
8. Вы также можете взаимодействовать с визуализацией, щелкнув код или переменные, чтобы просмотреть более подробную информацию.

Пояснение
Эта книга была написана при содействии ChatGPT. На протяжении всей книги фрагменты автоматически сгенерированного контента служили черновиками, которые я затем просматривал, проверял, корректировал, микшировал и дополнял, пока они не достигли своей окончательной формы. Ради прозрачности я также включил запросы, которые я использовал для создания черновиков, которые я пометил надстрочными номерами, непосредственно следующим за окончательным фрагментом текста, где они использовались. Эти числа указывают их положение в <i>Перечне запросов</i> , приведенном в конце книги.

ChatGPT как наставник по программированию

ChatGPT — это языковая модель, созданная OpenAI и предназначенная для общения с людьми на естественном языке. Это тип искусственного интеллекта, который может понимать и реагировать на ввод текста в разговорной манере. Он был обучен на огромном количестве данных, чтобы иметь возможность понимать и генерировать естественный язык, включая структурированные компьютерные языки высокого уровня, что делает его отличным инструментом, помогающим студентам изучать программирование в разговорной развлекательной форме¹².

Как студенту, изучающему программирование, ChatGPT может послужить вам полезным наставником. Вы можете задавать ему вопросы о концепциях программирования, запрашивать примеры кода или даже получать отзывы о своем собственном коде. Он может давать объяснения и разъяснения по темам программирования и служить инструментом для отработки навыков кодирования.

Конкретно, вот несколько способов, которыми ChatGPT может помочь вам в изучении программирования:

- ▶ Он может предоставить объяснения и примеры концепций программирования и синтаксиса, помогая вам лучше понять их.
- ▶ Он может отвечать на вопросы по кодированию, например, зачем нужно конкретное синтаксическое правило или как выполнить конкретную задачу.
- ▶ Он может помочь вам отладить ваш код, предлагая возможные решения и объясняя, почему возникают определенные ошибки.
- ▶ Он может помочь вам научиться писать эффективный и оптимизированный код, предоставляя отзывы о производительности вашего кода.
- ▶ Он может предложить ресурсы и учебные материалы, которые помогут вам лучше понять программирование.
- ▶ Он может помочь вам документировать ваш код, чтобы его было легче понять другим программистам или чтобы вы могли его запомнить в будущем.
- ▶ Он может переводить код между разными языками программирования для создания новых версий ваших программ.

- Он может содержать контекстуальные ссылки на значение, историю или анекдоты, связанные с развитием программирования и компьютерных наук в целом.

Хотя ChatGPT может быть ценным инструментом для изучения программирования, не следует полагаться на него как на единственный источник информации или обратной связи. Важно использовать его ответственно, сверять ответы с другими источниками и помнить о потенциальных неточностях или предвзятости в данных, используемых для обучения этой языковой модели ИИ.

Говоря об этических проблемах, связанных с использованием ChatGPT, важно отметить, что они не могут заменить ключевые навыки, которые вам необходимо приобрести как талантливому программисту, особенно навыки решения проблем и критического мышления. Хотя ChatGPT может стать отличным компаньоном в вашей учебе или даже партнером по кодированию, когда вы приступаете к программированию своих проектов, он не может заменить (по крайней мере, на момент написания этой книги) преимущества, которые вы можете получить благодаря человеческим изобретательности, взаимодействию и руководству.

Более того, вы скоро поймете, насколько важно совершенствовать свое умение задавать правильно сформулированные запросы, чтобы максимально использовать ChatGPT в этом опыте обучения. Теперь это называется быстрой разработкой, которая включает в себя четкое указание модели того, с чем вам нужна помощь, таким образом, чтобы уменьшить неточности или ложные результаты.

В целом, ChatGPT — невероятный ресурс для улучшения ваших навыков программирования, но используйте его ответственно. Обладая проницательным мышлением, вы можете использовать его потенциал, чтобы стать опытным программистом и начать многообещающую карьеру в этой области.

Использование ChatGPT

Вот краткий обзор того, как использовать ChatGPT:

1. Откройте свой веб-браузер и перейдите на сайт chat.openai.com. После загрузки страницы вы должны увидеть окно, показанное на Рис. 7:

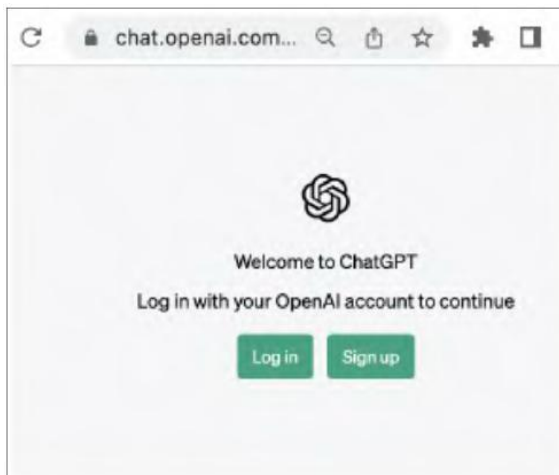


Рис. 7: Страница приветствия ChatGPT.

2. Войдите в свою учетную запись OpenAI, используя окно, показанный на Рис. 8. Если у вас нет учетной записи, выберите вариант "Sign up" и следуйте инструкциям.

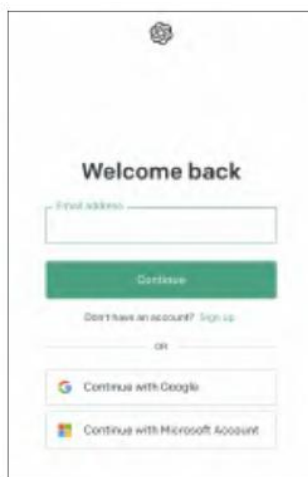


Рис. 8: Страница входа в ChatGPT.

3. После входа в систему вы попадете на главную страницу ChatGPT, показанную на Рис. 9. В текстовом поле "Send a message" введите вопрос (то есть запрос), который вы хотите отправить в ChatGPT, а затем нажмите клавишу "Enter" или кнопку со стрелкой.

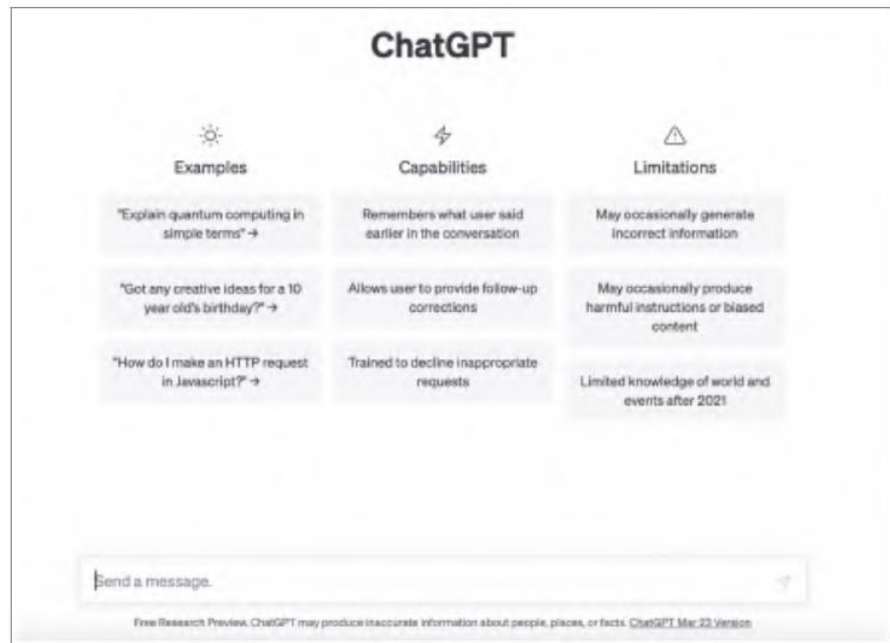


Рис. 9: Главная страница ChatGPT.

4. Подождите, пока ChatGPT сгенерирует ответ. Ответ появится в окне чата под вашим сообщением. Обратите внимание, что ответ может включать пояснительный текст, а также программный код. Вы можете нажать кнопку "Copy code", как показано на скриншоте на Рис. 10, чтобы скопировать код и вставить его в платформу, такую как Colab или PythonTutor, для выполнения программы.
5. Если ответ не полностью отвечает на ваш вопрос или вам нужна дополнительная информация, введите другой запрос, отправьте его в ChatGPT и продолжите беседу. История прошлых диалогов сохраняется в левой части экрана, поэтому, если хотите, вы можете вернуться и возобновить любой из них.

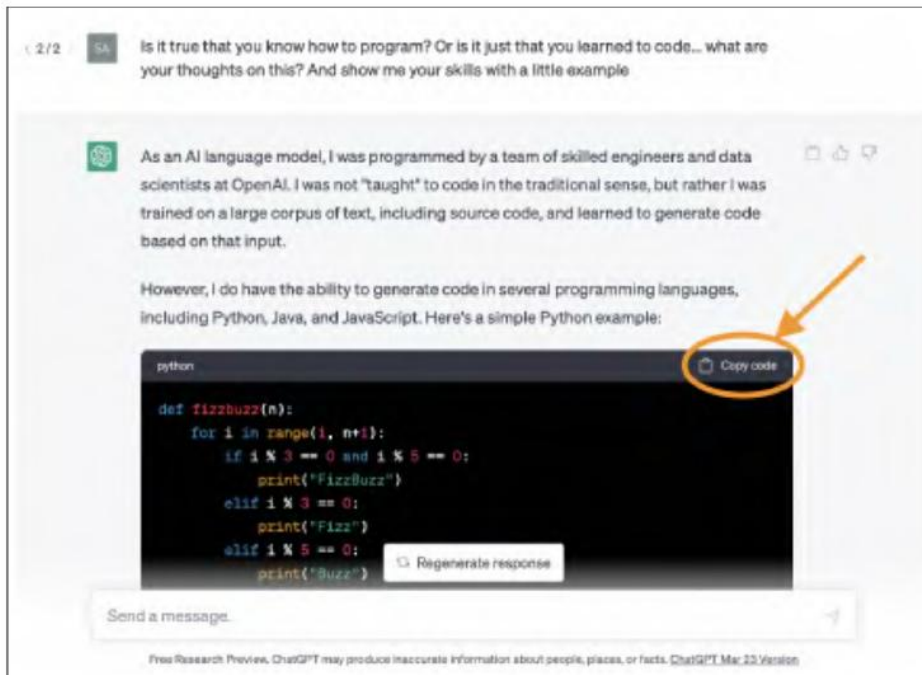


Рис. 10: Страница чата ChatGPT.

В этой книге я буду использовать ChatGPT, как я писал выше. Чтобы сделать материал более читабельным, я отформатирую запросы, пояснительный текст, код и вывод ответа, используя текстовые поля разного цвета. Для иллюстрации я воспроизвел приведенный выше скриншот в вышеупомянутом отформатированном стиле:

Is it true that you know how to program? Or is it just that you learned to code... what are your thoughts on this? And show me your skills with a little example.

As an AI language model, I was programmed by a team of skilled engineers and data scientists at OpenAI. I was not "taught" to code in the traditional sense, but rather I was trained on a large corpus of text, including source code, and learned to generate code based on that input. Here's an example with its output:

```

1 def fizzbuzz(n):
2     for i in range(1, n+1):
3         if i % 3 == 0 and i % 5 == 0:
4             print("FizzBuzz")
5     ...

```

```

1
2
Fizz
4
Buzz
...

```


Шаблоны запросов наставника ChatGPT

Как упоминалось ранее, создание эффективных запросов при обращении к ChatGPT за помощью в программировании — это навык, который будет развиваться по мере взаимодействия с инструментом и достижения промежуточных целей. В следующих главах мы будем использовать следующие шаблоны запросов для освещения различных тем программирования¹³:

1. Введение и объяснение концепций программирования:
 - ▶ "Объясни [концепцию программирования] простым способом."
 - ▶ "Что стоит за [концепцией программирования]."
 - ▶ "Приведи аналог [концепции программирования] из реальной жизни."
2. Написание примеров кода, связанных с концепциями программирования:
 - ▶ "Напиши пример кода для [концепции программирования]."
 - ▶ "Каково практическое применение [концепции программирования]?"
 - ▶ "Как [концепция программирования] используется для решения [какой-либо проблемы]?"
 - ▶ "Проиллюстрируйте использование [несколько концепций программирования] в одном примере кода, который [использует/вычисляет/сравнивает] [данные/информацию/статистику] по [теме из реальной жизни]. Включите пояснительные комментарии в каждый блок кода."
3. Описание ключевых строк кода:
 - ▶ "Что означает эта строка кода: [фрагмент кода]?"
 - ▶ "Как [конкретная строка кода] влияет на функциональность программы?"
 - ▶ "Что произойдет, если я заменю [конкретную строку кода] на [другой код]?"
4. Документирование и комментирование кода:
 - ▶ "Напиши строки документации для моего кода Python"
 - ▶ "Добавь встроенные комментарии для описания функциональности блоков кода моей программы"
5. Создание новых задач:
 - ▶ "Предложи упражнения, похожие на [проблема]"
 - ▶ "Какие варианты [проблемы] я могу попытаться решить?"
 - ▶ "Как я могу практиковать [концепцию программирования] в различных сценариях?"
6. Включение полезных ссылок на ресурсы:
 - ▶ "Порекомендуй хорошие ресурсы для обучения [концепции программирования]?"
 - ▶ "Где я могу найти больше информации о [концепции программирования]?"
 - ▶ "Какие существуют полезные онлайн-руководства по [концепции программирования]?"
7. Делаем тему более интересной:
 - ▶ "Расскажи мне анекдот о программировании, связанный с [концепцией программирования]?"
 - ▶ "Перечисли несколько забавных фактов о [концепции программирования]"

Итак, давайте без лишних раздумий приступим к нашему захватывающему путешествию по изучению Python с помощью ChatGPT. Я надеюсь, что вы найдете этот опыт веселым и полезным.

Узнай свою базу

1.1 Привет, мир!

Объясни концепцию программы «Hello, World!» в простой, но красочной форме, без использования кода. Кроме того, добавь соответствующий анекдот, чтобы обеспечить новичкам в программировании комфортное введение в тему¹⁴.

Ах, легендарная программа "Hello, World!"! Это идеальный способ начать свое путешествие в программирование, и это опыт, который вы всегда будете помнить. Представьте, что вы сидите за компьютером и чувствуете смесь возбуждения и нервозности. Вы собираетесь написать свою первую программу на Python!

Программа «Hello, World!» – простейшая программа, которую вы можете написать, но это важное событие. Это эквивалент первого крика новорожденного или первой ноты музыканта на новом инструменте. Эта программа является основой всех языков программирования и идеальной отправной точкой для всех, кто хочет научиться программировать.

Теперь вы можете задаться вопросом: «С какой стати мне писать программу, которая просто говорит «Hello, World!»?» Ну тут дело не в самом сообщении, а скорее в процессе написания программы. Эта простая программа – отличный способ познакомиться с новым языком, преодолеть страх начинать что-то новое и обрести уверенность в себе, чтобы продолжить изучение и исследование.

Но почему сообщение «Hello World»? Кто это придумал? В чем смысл?¹⁵

Происхождение программы «Hello, World!» несколько туманно, но считается, что она возникла в Bell Labs в начале 1970-х годов. Первый случай использования фразы в контексте программирования был зафиксирован в книге 1974 года Брайана Кернигана и Денниса Ричи о языке программирования C.

Что касается смысла сообщения, это просто способ для программистов проверить, правильно ли работает их программа и способна ли она отображать выходные данные. Это традиция и обряд посвящения новичков в мир программирования, сигнализирующий о начале их пути к тому, чтобы стать опытными программистами.

Ну хватит слов. Давайте углубимся в код и начнем обучение с ChatGPT.

На старт, внимание, пиши код!

Напиши классическую программу «Hello, World» на Python и объясните ее. Также предоставь результат выполнения программы. Я буду использовать Colab.¹⁶

Вот классическая программа "Hello, World!" на Python:

```
1 print("Hello, world!")
```

Функция `print()` — это встроенная в Python функция, позволяющая выводить на консоль текст или другие данные, в данном случае текст (также известный как *строка*) «Hello, World!». Результат выполнения программы должен выглядеть так:

```
Hello, world!
```

Чтобы запустить эту программу в Google Colab, выполните следующие действия:

1. Нажмите на кнопку "Copy code" в ChatGPT.
2. Вставьте его в первую ячейку блокнота в Colab.
3. Нажмите кнопку "play" рядом с ячейкой или нажмите "Shift + Enter"

Объясни, что такое синтаксическая ошибка, и продемонстрируй 3 возможные синтаксические ошибки в программе «Hello, World!» с соответствующими результатами выполнения программы.¹⁷

При написании программы необходимо соблюдать корректный синтаксис языка программирования. Синтаксическая ошибка возникает, когда вы делаете ошибку в синтаксисе кода, например пропускаете знак препинания, используете неправильный регистр для ключевого слова или вводите неверный оператор. Эти ошибки препятствуют правильной интерпретации кода. Вот три возможные синтаксические ошибки, которые могли возникнуть в программе «Hello, World!»:

1. Отсутствует скобка:

```
print("Hello, World!"
      File "<stdin>", line 1
print("Hello, World!"
      ^
SyntaxError: unexpected EOF while parsing
```

Сообщение об ошибке говорит нам, что при анализе кода обнаружен неожиданный конец файла (EOF).

2. Ключевое слово с ошибкой:

```

prnt("Hello, World!")
File "<stdin>", line 1
  prnt("Hello, World!")
    ^
NameError: name 'prnt' is not defined

```

Здесь сообщение об ошибке указывает, что команда `prnt()` не существует.

3. Отсутствующие кавычки:

```

print(Hello, World!)
File "<stdin>", line 1
print(Hello, World!)
                ^
SyntaxError: invalid syntax

```

Здесь ошибка указывает на то, что строковое сообщение должно быть заключено в кавычки.

Почему это называется `print()`? Нужен ли принтер, чтобы использовать его?¹⁸

Функция `print()` в Python используется для отображения или печати заданного сообщения или вывода на экран консоли. Она называется «print», потому что первоначально она относилась к распечатке информации на физическом принтере. Это было тогда, когда у компьютеров не было экранов. Однако в наше время это просто означает вывод информации на экран компьютера, и нет необходимости в принтере для ее использования.

1.2 Строки и форматирование строк

Объясни простым языком, что такое строка, и правила синтаксиса для определения строк в Python.¹⁹

В Python строка — это последовательность символов, заключенная в кавычки (одинарные или двойные кавычки). Она может содержать буквы, цифры, символы и пробелы. Строки обычно используются в программировании для хранения и обработки текстовых данных.

Чтобы определить строку в Python, вы просто заключаете нужную последовательность символов в кавычки. Например:

```
1 my_string = "Hello, World!"
2 print(my_string)
```

```
Hello, World!
```

В Python вы можете использовать как одинарные, так и двойные кавычки для определения строки. Важно, чтобы ваш выбор кавычек был последовательным во всем коде.

Вот несколько примеров определения строк в Python. Каждая строка присваивается, то есть сохраняется в памяти в переменной, с помощью оператора присваивания '='. Обратите внимание, что вы можете напечатать несколько строк, используя список переменных, разделенных запятыми. Используйте модификатор `sep='\n/` для печати в разных строках вывода:

```
1 string1 = "This is a string using double quotes"
2 string2 = 'This is a string using single quotes'
3 string3 = "This string contains numbers and symbols: $5.99"
4 print(string1, string2, string3, sep='\n')
```

```
This is a string using double quotes
This is a string using single quotes
This string contains numbers and symbols: $5.99
```

Вы также можете использовать тройные кавычки (одинарные или двойные) для определения многострочной строки:

```
1 multi_line_string = """This is a
2 multi-line
3 string"""
4 print(multi_line_string)
```

```
This is a
multi-line
string
```

Кроме того, вы можете использовать escape-символы (например, «\n» для новой строки, «\t» для табуляции), чтобы включать в строку специальные символы. Например:

```

1 escaped_string = "This string contains a new line.\nHere's the next line.\n\tAnd here's
   another line indented with a tab to the right"
2 print(escaped_string)

```

```

This string contains a new line.
Here's the next line.
    And here's another line indented with a tab to the right

```

Объясни форматирование строк простыми словами. Напиши забавные примеры кода для каждого из вариантов форматирования.²⁰

Форматирование строки — это процесс создания новой строки путем встраивания значений переменных или выражений в заполнители внутри строки. Это позволяет вам создавать динамические строки, которые могут отображать разные значения в зависимости от изменения данных на входе.

В Python существует несколько способов форматирования строк:

- ▶ **f-строки:** Это новейший и самый простой способ форматирования строк в Python. Вы просто включаете переменную или выражение в фигурные скобки внутри строки, перед которой ставится буква "f".
- ▶ **str.format():** Это немного более старый метод форматирования строк, который до сих пор широко используется. Он включает в себя вызов метода `format()` для строки и передачу значений, которые вы хотите вставить в заполнители, используя позиционные или ключевые аргументы.
- ▶ **% оператор:** Это старый метод форматирования строк, который в наши дни используется реже. Он включает использование оператора `%` для замены значений переменных или выражений заполнителями в строке.
- ▶ **конкатенация:** В этом подходе оператор `+` используется для объединения одной строки.

Вот несколько примеров каждого типа форматирования:


```

1 # f-strings
2 name = "Alice"
3 age = 30
4 print(f"{name} is {age} years old and loves Python!")
5
6 # str.format()
7 food = "pizza"
8 price = 10.99
9 print("I love {0}, it costs ${1:.2f} per slice.".format(food, price))
10
11 # % operator
12 city = "Paris"
13 country = "France"
14 print("I'd love to visit %s in %s someday." % (city, country))
15
16 # concatenation
17 print('Hello, ' + 'World' + '!')

```

```

Alice is 30 years old and loves Python!
I love pizza, it costs $10.99 per slice.
I'd love to visit Paris in France someday.
Hello, World!

```

Почему в коде есть зеленые строки, начинающиеся с символа решетки?²¹

Зеленые строки в коде, начинающиеся с символа решетки, называются комментариями. Комментарии используются для пояснений или напоминаний для человека, читающего код. Интерпретатор Python игнорирует комментарии, поэтому они не влияют на выполнение программы. Комментарии — это полезный способ для программистов общаться друг с другом, также комментарии могут помочь упростить понимание и сопровождение кода.

Хорошо. Nozu предоставляет пример программы, использующей только команды print() с реальной аналогией.²²

Вот пример программы, которая выводит рецепт на экран:

```

1 # Example program: "Cooking Pancakes"
2 # This program demonstrates the steps for cooking pancakes
3 print("-"*30 + " Cooking Pancakes " + "-"*30)
4 print("Step 1: Mix the pancake batter.")
5 print("Step 2: Heat the griddle or frying pan on medium heat.")
6 print("Step 3: Pour batter onto the griddle or frying pan, using a 1/4 cup measuring cup.
7     ")
8 print("Step 4: Cook for 1-2 minutes, until the edges start to dry and the surface is
9     bubbly.")
10 print("Step 5: Flip the pancake with a spatula, and cook for an additional 1-2 minutes on
11     the other side.")
12 print("Step 6: Remove the pancake from the griddle or frying pan and place on a plate.")
13 print("Step 7: Repeat steps 3-6 for the remaining batter.")
14 print("Step 8: Serve with your favorite toppings, such as butter and syrup.")

```

Результат показан ниже:

```

----- Cooking Pancakes -----
Step 1: Mix the pancake batter.
Step 2: Heat the griddle or frying pan on medium heat.
Step 3: Pour batter onto the griddle or frying pan, using a 1/4 cup measuring cup.
Step 4: Cook for 1-2 minutes, until the edges start to dry and the surface is bubbly.
Step 5: Flip the pancake with a spatula, and cook for an additional 1-2 minutes on
        the other side.
Step 6: Remove the pancake from the griddle or frying pan and place on a plate.
Step 7: Repeat steps 3-6 for the remaining batter.
Step 8: Serve with your favorite toppings, such as butter and syrup.

```

Могут ли я использовать параметры форматирования, такие как ширина и выравнивание, для построения строк? Покажи мне интересные примеры.²³

Вы можете использовать параметры форматирования ширины и выравнивания для построения строк в Python. Ширина определяет общую ширину строки, а выравнивание указывает, как выравнивать строку в пределах этой ширины. Вот некоторые примеры:

1. **Текст по центру:** Вы можете центрировать строку, используя шаблон `{:"width}`, чтобы указать выравнивание по центру и указать ширину. Обратите внимание, как выражение `'-'*8` используется для многократного вывода символа.

```

1 text = "Hello, World!"
2 centered_text = f"|{text:^40}|"
3 print('-'*8, centered_text, '-'*8)
4

```

```

----- |                Hello, World!                | -----

```

2. **Текст с выравниванием по левому краю:** Вы можете выровнять строку по левому краю, используя шаблон `{:<width}`, чтобы указать выравнивание по левому краю и указать ширину.

```

1 name = "John"
2 age = 30
3 job = "Engineer"
4 output = f"|Name: {name:<10}|Age: {age:<5}|Job: {job:<10}|"
5 print(output)
6

```

```

|Name: John      |Age: 30  |Job: Engineer  |

```

3. **Текст с выравниванием по правому краю:** Вы можете выровнять строку по правому краю, используя шаблон `{:>width}`, чтобы указать выравнивание по правому краю и указать ширину.

```

1 name = "Sarah"
2 age = 28
3 job = "Designer"
4 output = f"|Name: {name:>10}|Age: {age:>5}|Job: {job:>10}|"
5 print(output)
6

```

```

|Name:      Sarah|Age:   28|Job:   Designer|

```

4. **Усеченный текст:** Вы можете обрезать строку, указав максимальную ширину, используя шаблон `:.width` внутри заполнителя.

```

1 text = "This is a very long string that needs to be truncated"
2 truncated_text = f"{text:.20}"
3 print(truncated_text+"...")
4

```

```
This is a very long ...
```

5. **Числа, дополненные нулями:** Вы можете дополнить число нулями, используя шаблон `{:0width}`, чтобы указать заполнение нулями и ширину.

```

1 number = 42
2 padded_number = f"{number:05}"
3 print(padded_number)
4

```

```
00042
```

6. **Заполнение и десятичные разряды:** Чтобы форматировать число с плавающей запятой с определенным количеством десятичных разрядов и заполнением, используйте шаблон `{:width.precisionf}`.

```

1 num = 3.14159
2 formatted_num = "{:08.2f}".format(num)
3 print("The formatted number is:", formatted_num)
4

```

```
The formatted number is: 00003.14
```

Что вы подразумеваете под числом с плавающей запятой?²⁴

Число с плавающей запятой — это числовое значение, которое имеет десятичную точку. Оно часто используется в языках программирования для представления действительных чисел. Числа с плавающей запятой могут быть как положительными, так и отрицательными и могут иметь различную степень точности в зависимости от количества цифр, используемых для их представления. Они называются «с плавающей запятой», потому что десятичная точка может «плавать» или перемещаться в разные позиции в числе, в зависимости от величины числа. Они представлены в памяти компьютера в стандартном формате, обеспечивающем эффективное вычисление и хранение. В Python числа с плавающей запятой представлены с использованием типа данных `float`.

1.3 Время поработать руками!

Я хотел бы применить на практике то, что я только что узнал о `print()` и форматировании строк. Пожалуйста, предоставь четыре упражнения с кодом, в которых мне нужно исправить ошибки, чтобы они работали правильно.²⁵

Вот четыре упражнения, в которых вы должны исправить ошибки, чтобы программы могли выполняться правильно:

Упражнение 1:

```
1 print(Hello, World!)
```

Упражнение 2:

```
1 message = "I have {} apples"
2 print(message+format(5))
```

Упражнение 3:

```
1 name = input("What is your name? ")
2 age = input("How old are you? ")
3 print("Your name is {} and you are {} years old."format(name, age))
```

Упражнение 4:

```
1 number = 42
2 print("The answer is {:.10}".format(number))
```

Я хотел бы применить на практике то, что я только что узнал о `print()` и форматировании строк. На этот раз предложи мне несколько упражнений для написания кода, которые можно решить, используя только `print()` и форматирование строк.²⁶

Упражнение 1:

Выведите на печать квитанцию о покупке, например, показанную ниже.

```

-----
|           RECEIPT           |
-----
| Item       | Quantity | Price |
|-----|-----|-----|
| Apples    | 3        | $1.50 |
| Bananas   | 2        | $0.80 |
| Oranges   | 4        | $2.00 |
-----

```

Упражнение 2:

Создайте программу, которая печатает прямоугольный треугольник, используя символ *, см. ниже.

```

*
* *
* * *
* * * *
* * * * *

```

Упражнение 3:

Создайте программу, которая выводит на экран следующую сумму:

```

  1
 20
300
+ 4000
-----
4321

```

Одним из интересных аспектов ChatGPT является его способность пересматривать, исправлять и предоставлять отзывы о фрагменте кода. Эта функция может быть невероятно полезна для программистов, особенно для тех, кто плохо знаком с программированием или борется с определенной проблемой. Просто отправив фрагмент кода или проблему в ChatGPT, пользователи могут получить ценную обратную связь, предложения по улучшению и даже исправленный код.²⁷

Эта функция может сэкономить время и улучшить качество кода, предлагая свежий взгляд и обнаруживая ошибки, которые могли быть упущены. Кроме того, способность ChatGPT предоставлять объяснения своих изменений и отзывы может помочь

пользователям лучше понять концепции программирования и лучшие практики.

В целом, способность ChatGPT пересматривать, исправлять код и предоставлять отзывы о коде — это мощный инструмент, который может быть полезен как начинающим, так и опытным программистам. Используя знания и опыт этой языковой модели ИИ, пользователи могут улучшить свои навыки кодирования и создавать более качественный код.

Для этой цели может быть полезен следующий шаблон:²⁸

```
Below is the code I wrote to solve the exercise: [insert the exercise statement here].
```

```
Please revise/improve/correct my code and provide feedback on how to make it better, more efficient, or more readable. Also, demonstrate how you would have coded the solution differently, if applicable.
```

Code:

```
[Insert code here]
```

Для иллюстрации применим этот шаблон к Упражнению 2:

*Ниже приведен код, который я написал для решения упражнения: «Создайте программу, которая печатает прямоугольный треугольник с помощью символа *».*

Пожалуйста, пересмотри/улучши/исправь мой код и сообщи, как сделать его лучше, эффективнее или читабельнее. Кроме того, продемонстрируй, как бы ты закодировал решение по-другому, если это возможно.²⁹

Код:

```
1 print('* ')
2 print('* ' * 2)
3 print('* ' * 3)
4 print('* ' * 4)
5 print('* ' * 5)
```

Вот улучшенная версия вашего кода:

```
1 for i in range(1, 6):
2     print('* ' * i)
```

Этот код использует цикл for и функцию диапазона для печати каждой строки прямоугольного треугольника. Это более лаконично и избавляет от необходимости

писать каждую строку отдельно.

Обратите внимание, что существует много возможных способов решения этой проблемы, и лучшее решение зависит от требований и ограничений проблемы.

В любом случае, прилагайте большие усилия по написанию вашего исходного кода! Продолжайте практиковаться и учиться.

(NB: ChatGPT прав, предполагая, что использование цикла for было бы более элегантным и лаконичным подходом к решению проблемы. Подробнее о циклах будет рассказано в Главе 4.)

Строительные блоки данных

2.1 Типы данных

Что такое программа? Как это связано с данными?³⁰

Программа — это набор инструкций, которые сообщают компьютеру, что делать. Это последовательность операторов, которые выполняются компьютером для выполнения определенной задачи. Программы пишутся на языке программирования, который представляет собой набор синтаксических правил, определяющих, как должны быть написаны инструкции.

Данные — это любая информация, которая может быть сохранена и обработана компьютером. Это могут быть числа, текст, изображения, аудио или видео. Данные используются программами для выполнения своих задач. Например, программа, которая вычисляет среднее значение набора чисел, должна будет прочитать числа из файла данных или попросить пользователя ввести их.

Связь между программами и данными заключается в том, что программы могут использоваться для создания, хранения, извлечения данных и управления ими. Их также можно использовать для анализа данных и получения новой информации.

Вот несколько примеров того, как связаны программы и данные:

- ▶ Текстовый процессор использует данные (текст) для создания документов.
- ▶ Электронная таблица использует данные (числа) для расчета значений.
- ▶ База данных использует данные (записи) для хранения информации.
- ▶ Веб-браузер использует данные (веб-страницы) для отображения информации на экране.
- ▶ Игра использует данные (персонажи, уровни и т. д.) для создания интерактивного опыта.

Хорошо. Объясни с помощью примеров и выходных данных, как данные представлены в Python. Приведи примеры персонажей комиксов, чтобы лучше понять концепцию.³¹

В Python данные представлены путем присвоения значений переменным. Эти значения могут быть разных типов, таких как строки, целые числа и логические значения. Вот некоторые примеры:

```
1 # Assigning a string to a variable
2 hero = "Spider-Man"
3 print("Character:", hero)
4 # Assigning an integer to a variable
5 age = 23
6 print("Age:", age)
7 # Assigning a boolean to a variable
8 is_villain = False
9 print("Is villain?", is_villain)
```

```
Character: Spider-Man
Age: 23
Is villain? False
```

Переменным также могут быть присвоены значения разных типов:

```
1 # Assigning a string to a variable and then changing it to an integer
2 power = "Spider-sense"
3 print("Power:", power)
4
5 power = 10
6 print("Power:", power)
7
8 # Assigning a boolean to a variable and then changing it to a string
9 is_hero = True
10 print("Is hero?", is_hero)
11
12 is_hero = "yes"
13 print("Is hero?", is_hero) # Output: yes
```

```
Power: Spider-sense
Power: 10
Is hero? True
Is hero? yes
```

Python также допускает множественное присваивание в одной строке, например:

```

1 # Assigning multiple variables at once
2 name, team, universe = "Iron Man", "Avengers", "Marvel"
3 print("-"*10)
4 print("Character:", name)
5 print("Team:", team)
6 print("Universe:", universe)
7
8 # Update the values of a variable
9 name, team, universe = "Wonder Woman", "Justice League", "DC Comics"
10
11 print("-"*10)
12 print("Character:", name)
13 print("Team:", team)
14 print("Universe:", universe)

```

```

-----
Character: Iron Man
Team: Avengers
Universe: Marvel
-----
Character: Wonder Woman
Team: Justice League
Universe: DC Comics

```

В целом, вы можете думать о типах данных как о разных типах контейнеров, которые содержат разные вещи. Как и в реальной жизни, у нас есть разные контейнеры для разных типов вещей. Например, мы используем стакан для хранения жидкостей, коробку для хранения предметов и сумку для продуктов. Точно так же в Python мы используем разные типы данных для хранения разных типов данных.

Итак, как я могу узнать тип переменной в данной точке программы?³²

Вы можете использовать встроенную функцию `type()` для определения типа переменной в Python. Например:

```

1 name = "Peter Parker"
2 age = 25
3 height = 1.83
4 is_spiderman = True
5
6 print("Type of name:", type(name))
7 print("Type of age:", type(age))
8 print("Type of height:", type(height))
9 print("Type of is_spiderman:", type(is_spiderman))
10
11 # Change value (and type) of a variable
12 is_spiderman = "yes"
13 print("Type of is_spiderman:", type(is_spiderman))

```

2.2 Базовый ввод/вывод

Как программа может получить различные типы входных данных от пользователя? Используй примеры из комиксов.³³

Программа может получать различные типы входных данных от пользователя, используя различные встроенные функции в зависимости от типа входных данных. Вот несколько примеров использования персонажей из комиксов:

- 1. Ввод текста:** Функцию `input()` можно использовать для ввода текста пользователем. Например, в комиксе «Человек-паук» программа может попросить пользователя ввести свое имя:

```

1 name = input("What's your name? ")
2 print("Hello, " + name + "!")

```

```

What's your name? Peter
Hello, Peter!

```

- 2. Целочисленный ввод:** Функцию `int()` можно использовать для получения целочисленного ввода от пользователя. Например, в комиксе «Люди Икс» программа может попросить пользователя ввести свой возраст:

```

1 age = int(input("What's your age? "))
2 if age >= 18:
3     print("You're old enough to join the X-Men!")
4 else:
5     print("Sorry, you're not old enough to join the X-Men yet.")

```

```

What's your age? 20
You're old enough to join the X-Men!

```

- 3. Ввод с плавающей запятой:** Функция `float()` может использоваться для получения от пользователя ввода с плавающей запятой. Например, в комиксе «Железный человек» программа может попросить пользователя ввести вес его костюма:

```

1 weight = float(input("What's the weight of your Iron Man suit (in pounds)? "))
2 print("Your suit weighs " + str(weight) + " pounds.")

```

```

What's the weight of your Iron Man suit (in pounds)? 500.5
Your suit weighs 500.5 pounds.

```

- 4. Логический ввод:** Функцию `bool()` можно использовать для получения логического ввода от пользователя. Например, в комиксе «Мстители» программа может спросить пользователя, хочет ли он присоединиться к команде:

```

1 answer = bool(input("Do you want to join the Avengers? (True/False) "))
2 if answer:
3     print("Welcome to the team!")
4 else:
5     print("Maybe next time.")

```

```

Do you want to join the Avengers? (True/False) True
Welcome to the team!

```

Хочешь присоединиться к Мстителям? (**True** (Верно) / **False** (Неверно)) Верно? Добро пожаловать в команду!

Что это за команды `if/else`, которые мы использовали в некоторых из предыдущих примеров? Я могу себе представить, что они имеют в виду, но можешь ли объяснить это кратко и

ясно?³⁴

Конечно! Оператор `if/else` позволяет программе принимать решения в зависимости от того, является ли определенное условие `True` или `False`. Базовая структура оператора `if/else` выглядит следующим образом:

```

1 if condition:
2     # Code to be executed if the condition is true
3 else:
4     # Code to be executed if the condition is false

```

Обратите внимание, что два блока кода являются взаимоисключающими: если условие `True`, то выполняется только блок кода под оператором `if` в противном случае, если условие равно `False`, то блок под оператором `else` является единственным выполняемым кодом.

Например, предположим, что мы хотим написать программу, которая проверяет длину строки и выводит сообщение в зависимости от того, длиннее или короче строка определенного количества символов. Мы могли бы использовать для этого оператор `if/else`:

```

1 # Define a string
2 hero = "Mister Fantastic"
3
4 # Check the length of the string
5 if len(hero) > 10:
6     print("The hero's name is really long!")
7 else:
8     print("The hero's name is pretty short.")

```

В этом примере функция `len()` используется для получения количества символов (длины) строки, которая в данном случае равна 15, поэтому программа выводит следующее сообщение:

```
The hero's name is really long!
```

Подробнее об условных операторах будет рассказано в Главе 4.

2.3 Партнер по кодированию

ChatGPT может не только предлагать упражнения по кодированию, но и решать их

самостоятельно! Эта удивительная функция открывает для вас целый новый мир захватывающих возможностей обучения! Вы можете бросить себе вызов, попросив ChatGPT предложить задачу по кодированию. После того, как вы решите ее, используя собственные знания и навыки, вы можете сравнить свое решение с решением ChatGPT. Вот где начинается настоящее веселье! Вы можете получить отзыв от ChatGPT о различиях между вашим решением и его собственным. Вы можете узнать, какое решение является более эффективным, или узнать больше о новых методах и хитростях, о которых вы не знали раньше. Только представьте, как весело вы проведёте время с вашим новым партнером по программированию, ChatGPT!

Давайте посмотрим на примере, как это работает.

Привет, ChatGPT! Я хотел бы попрактиковаться в своих навыках программирования с помощью упражнения, которое включает ввод пользователем, преобразование типов данных и вывод на печать, возможно, даже включая меню параметров. Не мог бы ты предложить мне упражнение для тренировки! Когда я получу выражение, я также был бы признателен, если бы ты мог показать мне пример ожидаемого результата. Спасибо^{1,35}

Привет, начинающий программист! Давайте попрактикуемся в вводе пользователем, преобразовании типов данных и печати с помощью занимательного упражнения. Напишите программу, которая запрашивает у пользователя как минимум 5 личных данных, таких как имя, возраст, хобби и т. д. Затем предложите пользователю меню из двух вариантов: «1-краткая биография» и «2-длинная биография». В зависимости от выбора пользователя ваша программа должна печатать краткую или длинную биографию. Проявите творческий подход к выводу данных и получайте удовольствие! Вот пример вывода:

```
Welcome to the Personal Bio Program!
```

```
Please enter your name: John
```

```
Please enter your surname: Smith
```

```
Please enter your age: 25
```

```
Please enter your favorite hobby: books
```

```
Please enter your favorite food: pizza
```

```
Please enter your favorite color: red
```

```
Menu of options:
```

```
1. Short Bio
```

```
2. Long Bio
```

```
Enter your choice (1 or 2): 1
```

```
--- Short Bio ---
```

```
John, 25, is a programming student whose favourite hobby is reading books.
```

Хорошо, теперь покажи мне решение.³⁶

Конечно! Вот одно из возможных решений личной биопрограммы:


```

1 # Personal Bio Program
2 # Ask user for personal data
3 name = input("What's your name? ")
4 age = int(input("How old are you? "))
5 city = input("What city do you live in? ")
6 hobby = input("What's your favorite hobby? ")
7 pet_name = input("What's your pet's name? ")
8
9 # Print menu of options
10 print("Please choose an option:")
11 print("1. Short bio")
12 print("2. Long bio")
13
14 # Ask user for choice
15 choice = input("Enter your choice (1 or 2): ")
16
17 # Print bio based on choice
18 if choice == "1":
19     print(f"Hi, my name is {name}. I'm {age} years old and I live in {city}. My favorite
20         hobby is {hobby} and I have a pet named {pet_name}.")
21 elif choice == "2":
22     print(f"My name is {name} and I'm {age} years old. I currently live in {city} and in
23         my free time, I enjoy {hobby}. I have a pet named {pet_name} who keeps me company.
24         Thanks for reading!")
25 else:
26     print("Invalid choice. Please enter 1 or 2.")

```

Круто. Моя версия решения показана ниже. Что ты об этом думаешь? По сравнению с твоим?³⁷

```

name = input("What is your name? ")
age = int(input("How old are you? "))
city = input("What city do you live in? ")
hobby = input("What is your favorite hobby? ")
food = input("What is your favorite food? ")
color = input("What is your favorite color? ")
credit_card = input("What is your credit card number? ")

```

```

print('+'+'-'*20+'+')
print(f"|{'OPTIONS':^20}|")
print('+'+'-'*20+'+')
print(f"|{'1. Short Bio':^20}|")
print(f"|{'2. Long Bio':^20}|")
print('+'+'-'*20+'+')

choice = int(input("Enter choice number: "))

if choice == 1:
    print(f"\n----- Personal Bio for: {name} -----")
    print(f"{name} is {age} years old and lives in {city}.")
    print("-----")
else:
    print(f"\n----- Personal Bio for: {name} -----")
    print(f"{name} is {age} years old and lives in {city}.")
    print(f"In {name}'s free time, {name} enjoys {hobby} and eating {food}.")
    print(f"A fun fact about {name} is that their favourite color is {color}.")
    print(f"For your information, {name}'s credit card number
    is:'+'*len(credit_card) )
    print("-----")

```

Ваше решение выглядит хорошо и выполняет ту же задачу, что и решение 1. Одно отличие, которое я заметил, заключается в том, что вы добавили дополнительный уровень форматирования к пунктам меню, используя символы «+» и «-» для создания рамки вокруг меню. Это добавляет приятную визуальную привлекательность программе. Кроме того, ваша программа также использует f-строки для форматирования вывода, что является кратким и элегантным способом отображения информации. Единственное, что я бы порекомендовал, — это избегать в такой программе запроса конфиденциальной информации, такой как номера кредитных карт, особенно потому, что программа распечатывает ее частично замаскированной. Важно уделять первоочередное внимание безопасности и конфиденциальности пользователей во всех приложениях.

В общем, отличная работа!

Но я постарался скрыть номер кредитной карты. Я просто пошутил.³⁸

А, я вижу! Я извиняюсь за замечание. В таком случае ваш код выглядит великолепно и работает правильно. Всегда полезно быть осторожным с конфиденциальной информацией, даже в примерах кода. В целом, ваш вариант лаконичен и удобочитаем, а также эффективно использует

параметры форматирования строк, доступные в Python, чтобы выходные данные выглядели аккуратно и организованно.

Хорошая работа!

Кстати, мне нравится, как ты прокомментировал каждый раздел кода; это помогает лучше его понять. Пожалуйста, сделай то же самое с моим кодом.³⁹

Конечно, вот ваш код с комментариями:

```

1
2 # Ask for user input on personal data
3 name = input("What is your name? ")
4 age = int(input("How old are you? "))
5 city = input("What city do you live in? ")
6 hobby = input("What is your favorite hobby? ")
7 food = input("What is your favorite food? ")
8 color = input("What is your favorite color? ")
9 credit_card = input("What is your credit card number? ")
10
11
12 # Print the menu options
13 print('+ '+'-'*20+'+')
14 print(f"|{'OPTIONS':^20}|")
15 print('+ '+'-'*20+'+')
16 print(f"|{'1. Short Bio':^20}|")
17 print(f"|{'2. Long Bio':^20}|")
18 print('+ '+'-'*20+'+')
19
20
21 # Ask user to choose an option
22 choice = int(input("Enter choice number: "))
23
24
25 # Print the chosen personal bio
26 if choice == 1:
27     print(f"\n----- Personal Bio for: {name} -----")
28     print(f"{name} is {age} years old and lives in {city}.")
29     print("-----")
30 else:
31     print(f"\n----- Personal Bio for: {name} -----")
32     print(f"{name} is {age} years old and lives in {city}.")
33     print(f"In {name}'s free time, {name} enjoys {hobby} and eating {food}.")
34     print(f"A fun fact about {name} is that their favourite color is {color}.")
35     print(f"For your information, {name}'s credit card number is: '+'*len(credit_card) )
36     print("-----")

```

2.4 Время поработать руками!

Предоставь четыре упражнения с кодом, связанным с вводом и преобразованием типов

данных, в которых мне нужно исправить синтаксические или логические ошибки, чтобы они могли работать правильно.⁴⁰

Конечно, вот четыре упражнения с кодом, связанным с вводом и преобразованием типов данных, с синтаксическими и/или логическими ошибками, которые необходимо исправить:

Упражнение 1:

```
1 age = input("Please enter your age: ")
2 if age >= 18:
3     print("You are old enough to vote!")
4 else:
5     print("You are not old enough to vote yet.")
```

Упражнение 2:

```
1 number = input("Enter a decimal number: ")
2 number = bool(number)
3 print(f"The integer part of the {number} is {number}")
```

Упражнение 3:

```
1 password = input("Please enter your password: ")
2 if length(password) >= 8:
3     print("Your password is strong!")
4 else:
5     print("Your password is too short. Please choose a password that is at least 8
    characters long.")
```

Упражнение 4:

```
1 name = input("Please enter your name: ")
2 age = input("Please enter your age: ")
3 print("Hello, name! You are age years old.")
```

Предложи мне несколько упражнений для написания кода, который можно реализовать, используя только пользовательский ввод и операции преобразования типа данных.⁴¹

Упражнение 1:

Напишите программу, которая просит пользователя ввести год своего рождения и вычисляет его возраст. Пример распечатки сообщения показан ниже.

```

Please enter your name: Tony Stark
Please enter your birth year: 1970

Hi Tony Stark! As of 2023, you are 53 years old.

```

Упражнение 2:

Напишите программу, которая просит пользователя ввести температуру в градусах Цельсия и конвертирует ее в градусы Фаренгейта. Формула преобразования: $F = (C * 9/5) + 32$. Пример распечатки сообщения показан ниже.

```

Please enter the temperature in Celsius: 30
30 C degrees is equal to 86 F degrees.

```

(NB: Это упражнение требует использования арифметических операций, включая сложение, умножение и деление. Подробнее об этих и других математических операторах будет рассказано в Главе 3).

Упражнение 3:

Создайте программу-калькулятор, которая предложит пользователю ввести два числа и выбрать операцию для выполнения над ними. Программа должна отображать меню операций (сложение, вычитание, умножение и деление) и предлагать пользователю выбрать соответствующее число для желаемой операции. После того, как пользователь ввел свое значение, программа должна вывести результат выбранной операции. Пример распечатки сообщения показан ниже.

```

Welcome to the calculator app!
Please enter the first number: 10
Please enter the second number: 5
----- Menu of operations -----
  1. Addition
  2. Subtraction
  3. Multiplication
  4. Division
Enter operation number: 3
10 * 5 = 50

```

(NB: Опять же, больше об этих и других математических операторах будет рассказано в Главе 3).

Место операторов

Перечисли все операторы Python и дай краткое описание для каждого из них.⁴²

Вот список различных типов операторов Python и их определения:

1. **Арифметические операторы:** используются для выполнения математических операций, таких как сложение, вычитание, умножение, деление и т. д.
2. **Операторы сравнения:** используются для сравнения значений и возврата логического значения (Истина (True) или Ложь (False)), например, равно, не равно, больше, меньше и т. д.
3. **Логические операторы:** используются для объединения нескольких условий и возврата логического значения, такого как И, ИЛИ, НЕ.
4. **Побитовые операторы:** используются для выполнения операций над отдельными битами двоичных чисел, таких как И, ИЛИ, исключающее ИЛИ и т. д.
5. **Операторы присваивания:** используются для присвоения значений переменным, например =, +=, -=, *=, /= и т. д.
6. **Операторы идентичности:** используются для сравнения местоположения в памяти двух объектов, таких как есть, нет.
7. **Операторы принадлежности:** используются для проверки наличия значения в последовательности, например, в, не в.
8. **Оператор возведения в степень:** используется для возведения числа в степень, представленную **.

Итак, давайте подробнее рассмотрим каждый из них.

3.1 Арифметические операторы

Образно и подробно объясни арифметические операторы Python и включи один пример кода, иллюстрирующий использование всех операторов. В примере следует сравнить занимательную статистику о фильмах Marvel и Люди X и кассовые сборы.⁴³

В Python арифметические операторы используются для выполнения основных математических операций, таких как сложение, вычитание, умножение и деление. Эти операторы используются для управления числовыми типами данных, такими как целые числа и числа с плавающей запятой.

Вот ключевые арифметические операторы Python:

- ▶ **Сложение:** Знак плюс (+) используется для добавления. Он складывает два значения вместе. Например, $2 + 2$ равно 4.
- ▶ **Вычитание:** Знак минус (-) используется для вычитания. Он вычитает второе значение из первого значения. Например, $5 - 2$ равно 3.
- ▶ **Умножение:** Звездочка (*) используется для умножения. Она перемножает друг с другом два значения. Например, $3 * 4$ равно 12.
- ▶ **Деление:** Косая черта (/) используется для операции деления. Она делит первое значение на второе. Например, $8 / 2$ равно 4.
- ▶ **Целочисленное деление:** Двойная косая черта (//) используется для операции целочисленного деления. Она делит первое значение на второе, но округляет результат до ближайшего целого числа. Например, $8 // 3$ равно 2.
- ▶ **Модуль:** Знак процента (%) используется для вычисления модуля. Он возвращает остаток от деления первого значения на второе значение. Например, $7 \% 3$ равно 1.

Давайте поэкспериментируем с этими операторами. Вот пример, который сравнивает некоторые статистические данные о фильмах Marvel и Люди X и кассовые сборы:

```

1 # Marvel and X-Men movie statistics
2 marvel_movies = 28
3 xmen_movies = 13
4 marvel_box_office = 22.55e9 # in US dollars
5 xmen_box_office = 6.01e9 # in US dollars
6 single_ticket = 15 # in US dollars
7
8
9 # Addition
10 total_movies = marvel_movies + xmen_movies
11 print("Total number of movies from both franchises:", total_movies)
12
13 # Subtraction
14 difference_movies = marvel_movies - xmen_movies
15 print("Marvel has", difference_movies, "more movies than X-Men")
16
17 # Multiplication
18 money_needed = single_ticket * total_movies
19 print("Total cost of watching all movies in the theatre: US$", money_needed)
20
21 # Division
22 total_box_office = marvel_box_office + xmen_box_office
23 average_box_office = total_box_office / total_movies
24 print(f"Average box office revenue per movie: US$ {average_box_office:,.2f}")
25
26 # Integer Division
27 years_needed = total_movies // 12 # assuming watching 1 movie per month
28
29 # Modulo
30 plus_months_needed = total_movies % 12 # assuming watching 1 movie per month
31
32 print(f"Time required to watch all movies (one per month): {years_needed} years plus {
    plus_months_needed} months.")

```

Код не требует пояснений и выдает следующие результаты:

```
Total number of movies from both franchises: 41
Marvel has 15 more movies than X-Men
Total cost of watching all movies in the theatre: US$ 615
Average box office revenue per movie: US$ 696,585,365.85
Time required to watch all movies (one per month): 3 years and 5 months.
```

Что означают для переменных `box_office` эти числа `22.55e9` и `6.01e9`?⁴⁴

Числа `22.55e9` и `6.01e9` представлены в экспоненциальной форме записи, которая представляет собой способ выражения очень больших или очень малых чисел с использованием степени 10. В данном случае «e9» означает «умножить на 10 в 9-й степени», или один миллиард. Следовательно, `22.55e9` означает 22,55 миллиарда, а `6.01e9` означает 6,01 миллиарда. Эти цифры, вероятно, представляют собой кассовые сборы в долларах США для фильмов Marvel и Люди X соответственно.

3.2 Операторы сравнения

Объясни подробно и образно операторы сравнения Python и включи один пример кода, чтобы проиллюстрировать использование всех их, возможно, используя забавные факты о футбольных командах.⁴⁵

Операторы сравнения, также называемые реляционными операторами, используются в Python для сравнения значений. Результат сравнения всегда является логическим значением: либо **True** (Истина), либо **False** (Ложь).

Вот список операторов сравнения Python:

- ▶ **== (равно)**: Проверяет, равны ли значения двух операндов.
- ▶ **!= (не равно)**: Проверяет, не равны ли значения двух операндов.
- ▶ **< (меньше чем)**: Проверяет, меньше ли значение левого операнда значения правого операнда.
- ▶ **<= (меньше или равно)**: Проверяет, меньше ли значение левого операнда или равно значению правого операнда.
- ▶ **> (больше чем)**: Проверяет, больше ли значение левого операнда значения правого операнда.
- ▶ **>= (больше или равно)**: Проверяет, больше ли или равно значение левого операнда значению правого операнда.

Давайте проиллюстрируем использование всех этих операторов на забавном примере с футбольными командами! Представьте, что у нас есть две команды, «Реал Мадрид» и

«Барселона», и мы хотим сравнить их статистику:

```

1 # Real Madrid statistics
2 real_points = 70
3 real_goals_scored = 76
4 real_goals_conceded = 39
5 real_goal_difference = real_goals_scored - real_goals_conceded
6
7 # Barcelona statistics
8 barca_points = 65
9 barca_goals_scored = 85
10 barca_goals_conceded = 38
11 barca_goal_difference = barca_goals_scored - barca_goals_conceded
12
13 # Comparison operators
14 print(f"Real Madrid ({real_points}) and Barcelona ({barca_points}) have equal points:",
15       real_points == barca_points)
16 print(f"Real Madrid ({real_goals_scored}) and Barcelona ({barca_goals_scored}) have
17       different goals scored:",
18       real_goals_scored != barca_goals_scored)
19 print(f"Real Madrid ({real_goals_conceded}) has conceded fewer goals than Barcelona ({
20       barca_goals_conceded}):",
21       real_goals_conceded < barca_goals_conceded)
22 print(f"Real Madrid's goal difference ({real_goal_difference}) is less than or equal to
23       Barcelona's goal difference ({barca_goal_difference}):",
24       real_goal_difference <= barca_goal_difference)
25 print(f"Real Madrid ({real_points}) has more points than Barcelona ({barca_points}):",
26       real_points > barca_points)
27 print(f"Barcelona ({barca_goals_conceded}) has conceded more goals than Real Madrid ({
28       real_goals_conceded}):",
29       barca_goals_conceded >= real_goals_conceded)

```

```

Real Madrid (70) and Barcelona (65) have equal points: False
Real Madrid (76) and Barcelona (85) have different goals scored: True
Real Madrid (39) has conceded fewer goals than Barcelona (38): False
Real Madrid's goal difference (37) is less than or equal to Barcelona's goal
    difference (47): True
Real Madrid (70) has more points than Barcelona (65): True
Barcelona (38) has conceded more goals than Real Madrid (39): False

```

Как видно из этого примера, данные операторы сравнения очень полезны в программировании на Python, поскольку они позволяют нам проверять отношения между

различными значениями и принимать решения на основе этих отношений.

3.3 Логические операторы

Объясни концепцию логических операторов Python в непринужденной форме. После определения каждого оператора включи один пример кода, чтобы проиллюстрировать их использование. Используй для этой цели меню блюд быстрого питания с указанием того, насколько они подходят для вегетарианцев и веганов.⁴⁶

Логические операторы Python — это инструменты, которые позволяют нам комбинировать и манипулировать логическими условиями, представляющими истинные или ложные утверждения. Эти операторы в основном используются с логическими переменными, которые могут быть только **True** (Истина) или **False** (Ложь).

Наиболее часто используемые логические операторы Python — это **and**, **or** и **not**. Оператор **and** возвращает значение **True**, только если оба операнда имеют значение **True**, в то время как оператор **or** возвращает значение **True**, если хотя бы один из операндов имеет значение **True**. Оператор **not** используется для обращения (смены на противоположное) логического значения операнда.

Рассмотрим следующую таблицу блюд быстрого питания с указанием, насколько они подходят для вегетарианцев и веганов:

Блюдо	Вегетарианец?	Веган?
Чизбургер	Нет	Нет
Пицца Маргарита	Да	Нет
Овощной салат	Да	Да
Мороженое	Да	Нет

Мы можем использовать логические операторы, чтобы ответить на такие вопросы, как «Какие блюда подходят как для вегетарианцев, так и для веганов?» или «Какие блюда не подходят ни вегетарианцам, ни веганам?». Вот пример кода, демонстрирующий использование логических операторов:

```
1 # Define the boolean values
2 # (cb: Cheeseburger, mp: Margherita Pizza, gs: Garden Salad, ic: Ice Cream)
3 cb_veg = False
4 cb_vegan = False
5 mp_veg = True
6 mp_vegan = False
7 gs_veg = True
8 gs_vegan = True
9 ic_veg = True
10 ic_vegan = False
11
12 # Print the results of logical operators
13 print("Is Cheeseburger both vegetarian and vegan? ", cb_veg and cb_vegan)
14 print("Is Margherita Pizza both vegetarian and vegan? ", mp_veg and mp_vegan)
15 print("Is Garden Salad both vegetarian and vegan? ", gs_veg and gs_vegan)
16 print("Is Ice Cream both vegetarian and vegan? ", ic_veg and ic_vegan)
17 print("Is Cheeseburger either vegetarian or vegan? ", cb_veg or cb_vegan)
18 print("Is Margherita Pizza either vegetarian or vegan? ", mp_veg or mp_vegan)
19 print("Is Garden Salad either vegetarian or vegan? ", gs_veg or gs_vegan)
20 print("Is Ice Cream either vegetarian or vegan? ", ic_veg or ic_vegan)
21 print("Is Cheeseburger not vegetarian? ", not cb_veg)
22 print("Is Cheeseburger not vegan? ", not cb_vegan)
```

Результат показан ниже:

```
Is Cheeseburger both vegetarian and vegan? False
Is Margherita Pizza both vegetarian and vegan? False
Is Garden Salad both vegetarian and vegan? True
Is Ice Cream both vegetarian and vegan? False
Is Cheeseburger either vegetarian or vegan? False
Is Margherita Pizza either vegetarian or vegan? True
Is Garden Salad either vegetarian or vegan? True
Is Ice Cream either vegetarian or vegan? True
Is Cheeseburger not vegetarian? True
Is Cheeseburger not vegan? True
```

3.4 Побитовые операторы

Объясни концепцию побитовых операторов Python дружелюбным и профессиональным тоном. Включи пример кода после определения каждого оператора, чтобы продемонстрировать, как их использовать. Покажи результаты операций, включая битовые последовательности и целые числа операндов.⁴⁷

Побитовые операторы Python используются для управления битами или двоичными цифрами целого числа. Они применяются независимо к каждой битовой позиции целого числа, отсюда и название «побитовый».

В Python есть шесть побитовых операторов: побитовое И (AND или &), ИЛИ (OR или |), исключающее ИЛИ (XOR или ^), НЕ (NOT или ~), сдвиг влево («) и сдвиг вправо (»).

- Побитовый оператор AND возвращает новое целое число, в котором каждый бит установлен в 1, только в том случае, если оба бита из исходных целых чисел равны 1. В противном случае бит устанавливается в 0.

Вот пример:

```

1 a = 12 # 1100 in binary
2 b = 25 # 11001 in binary
3 c = a & b # 1000 in binary
4 print(f" {a:08b} (decimal: {a})\n & {b:08b} (decimal: {b})\n  -----\n  {c:08
   b} (decimal: {c})")
5
```

Обратите внимание, что я использовал модификатор формата :08b, чтобы печатать целые числа в виде 8-битных двоичных строк для большей ясности. Результатом будет:

```

00001100 (decimal: 12)
& 00011001 (decimal: 25)
-----
00001000 (decimal: 8)
```

- Оператор побитового ИЛИ возвращает новое целое число, в котором каждому биту присваивается значение 1, если хотя бы один бит исходных целых чисел равен 1. В противном случае биту присваивается значение 0.

Пример:

```

1 a = 12 # 1100 in binary
2 b = 25 # 11001 in binary
3 c = a | b # 11001 in binary
4 print(f" {a:08b} (decimal: {a})\n | {b:08b} (decimal: {b})\n  -----\n  {c:08
   b} (decimal: {c})")
5

```

Результатом будет:

```

 00001100 (decimal: 12)
 | 00011001 (decimal: 25)
 -----
 00011101 (decimal: 29)

```

- Побитовый оператор XOR возвращает новое целое число, в котором каждому биту присваивается значение 1, только в том случае, если один из битов исходного целого числа равен 1. В противном случае бит присваивается 0.

Пример:

```

1 a = 12 # 1100 in binary
2 b = 25 # 11001 in binary
3 c = a ^ b # 10001 in binary
4 print(f" {a:08b} (decimal: {a})\n ^ {b:08b} (decimal: {b})\n  -----\n  {c:08
   b} (decimal: {c})")
5

```

Результатом будет:

```

 00001100 (decimal: 12)
 ^ 00011001 (decimal: 25)
 -----
 00010101 (decimal: 21)

```

- Побитовый оператор НЕ возвращает целое число, то есть изменяет каждый бит с 0 на 1 и наоборот. Обратите внимание, что он возвращает результат в форме дополнения до двух, которая представляет собой представление целых чисел со знаком.

Пример:


```

1 a = 12 # 1100 in binary
2 c = ~a # -13 in decimal (complement of 1100 in binary)
3 print(f" ~ {a:08b} (decimal: {a})\n -----\n {c:08b} (decimal: {c})")
4

```

Результатом будет:

```

~ 00001100 (decimal: 12)
-----
-0001101 (decimal: -13)

```

- Оператор сдвига влево перемещает все биты целого числа влево на заданное количество позиций и заполняет крайние правые биты нулями. Это эквивалентно умножению целого числа на 2, возведенному в степень количества позиций.

Пример:

```

1 a = 12 # 1100 in binary
2 c = a << 3 # 110000 in binary, which is 96 in decimal
3 print(f" {a:08b} (decimal: {a})\n<< 3 \n -----\n {c:08b} (decimal: {c})")
4

```

Результатом будет:

```

 00001100 (decimal: 12)
<< 3
-----
01100000 (decimal: 96)

```

- Оператор сдвига вправо перемещает все биты целого числа вправо на заданное количество позиций и заполняет крайние левые биты нулями (для неотрицательных целых чисел) или единицами (для отрицательных целых чисел). Это эквивалентно делению целого числа на 2, возведенному в степень числа позиций.

Пример:

```

1 a = 12 # 1100 in binary
2 c = a >> 3 # 11 in binary, which is 1 in decimal
3 print(f" {a:08b} (decimal: {a})\n>> 3 \n -----\n {c:08b} (decimal: {c})")
4

```

Результатом будет:

```

00001100 (decimal: 12)
>> 3
-----
00000001 (decimal: 1)

```

Побитовые операторы в основном используются при низкоуровневом программировании, таком как встроенные системы или драйверы устройств, но также могут использоваться в криптографии, графике и при сжатии данных.

Хорошо, но не мог бы ты кратко объяснить, что ты подразумеваешь под «двоичными цифрами целого числа».

Приведи мне несколько забавных аналогий из реальной жизни. ⁴⁸

В компьютерных системах вся информация в конечном счете представляется в виде двоичных цифр или, для краткости, битов. Это нули и единицы, которые вы могли видеть в компьютерном коде или цифровой электронике. Двоичная цифра — это бит, представляющий собой наименьшую единицу данных, которую компьютер может хранить или обрабатывать. Бит может иметь только два возможных значения: 0 или 1.

Забавной аналогией для понимания битов и байтов может служить куча выключателей света. Каждый переключатель может быть либо «включен», либо «выключен», точно так же, как один бит может принимать значение либо 0, либо 1. Располагая несколько переключателей рядом, мы можем формировать большие группы переключателей и представлять большие значения.

Например, 4 переключателя могут представлять 16 различных комбинаций или значений от 0 до 15. Точно так же 8 переключателей могут представлять 256 различных комбинаций или значений от 0 до 255. Например, число 6 может быть представлено в 4-битной двоичной последовательности как 0110, что означает $0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$ или $0 + 4 + 2 + 0$. Аналогично, число 15 будет выглядеть как 1111, что означает $1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0$, или $8 + 4 + 2 + 1$. Поэтому, когда мы говорим о двоичных разрядах целого числа, мы имеем в виду представление этого целого числа в двоичной системе счисления.

Побитовые операторы работают, манипулируя этими отдельными переключателями или битами, а также их группами. Это позволяет нам выполнять множество полезных операций над базовыми данными, например сдвигать их влево или вправо, зеркально изменять отдельные биты, либо комбинировать несколько значений различными способами.

Хорошо, еще одно: пожалуйста, кратко и понятно объясни, что ты подразумеваешь под «представлением в форме дополнения до двух». ⁴⁹

В компьютерных системах нам часто нужно представлять как положительные, так и отрицательные числа. Одним из способов представления отрицательных чисел является

использование представления «знаковая величина», где крайний левый бит отображает знак числа (0 для положительного, 1 для отрицательного), а остальные биты отображают значение. Однако у этого представления есть некоторые недостатки, такие как необходимость в особых случаях для обработки нуля и тот факт, что у нас есть два представления нуля (положительное и отрицательное).

Представление с дополнением до двух является более экономичной и широко используемой альтернативой. Оно представляет отрицательные числа как дополнение до двух их абсолютного значения. Это означает, что для получения дополнения числа до двух мы сначала берем дополнение (зеркально меняем все биты) двоичного представления его абсолютного значения, а затем добавляем к результату 1. См. пример:

1. Находим двоичное представление числа 5: 00000101.
2. Инвертируем все биты: 11111010.
3. Добавляем 1: 11111011.

Таким образом, форма дополнения до двух для 5 в 8-битном двоичном коде равна 11111011.

Отрицательные числа в дополнении до двух полезны в арифметических операциях, потому что они позволяют компьютеру выполнять сложение и вычитание, используя те же схемы и алгоритмы, что и для целых чисел без знака. На самом деле добавление отрицательного дополнительного числа до двух эквивалентно вычитанию его абсолютного значения из целого числа без знака. Это упрощает выполнение арифметических операций и уменьшает количество необходимого аппаратного обеспечения.

Как следствие, представление с дополнением до двух обеспечивает простой и эффективный способ представления отрицательных чисел в компьютерных системах, что необходимо для многих приложений, таких как цифровая обработка сигналов, искусственный интеллект и криптография.

3.5 Операторы присваивания

Объясни концепцию арифметических операторов присваивания Python в игровой форме.⁵⁰

Арифметические операторы присваивания Python — это просто забавный способ сочетания математики и ссылок. Вы же знаете, как иногда хочется обновить значение, добавляя или вычитая другое значение? Что ж, арифметические операторы присваивания упрощают кодирование!

Приведу несколько примеров из жизни. Представьте, что у вас есть банка, наполненная конфетами. Вы хотите отслеживать, сколько конфет вы едите каждый день. Вы начинаете с 10 конфет и в первый день съедаете 3 конфеты. Вместо того, чтобы обновлять значение вручную, вы можете использовать арифметический оператор присваивания, дабы вычесть 3 из текущего значения конфет в банке. Итак, вместо того, чтобы писать `jar = 7`, вы можете написать `jar -= 3`.

Точно так же предположим, что у вас есть копилка, полная монет. Предположим, что эта копилка — это современное волшебное устройство, которое позволяет вам зарабатывать

10% процентов на деньгах, которые вы откладываете каждый месяц. Таким образом, вы начинаете с 10 долларов и в конце месяца получаете 11 долларов. Вместо того, чтобы вручную назначать конечное значение состояния копилки, используйте арифметический оператор присваивания "=", дабы добавить 1 к текущему балансу. Вместо того, чтобы писать `piggy_bank = 11`, вы можете написать `piggy_bank *= 1.1`:

```

1 piggy_bank = 10
2 print("Money now in the piggy_bank:\t$ ", piggy_bank)
3 piggy_bank *= 1.1
4 print("Money one month later:\t\t$ ", piggy_bank)

```

```

Money now in the piggy_bank: $ 10
Money one month later:      $ 11.0

```

Вот еще несколько примеров арифметических операторов присваивания Python:

```

1 x = 10
2 x += 5 # equivalent to x = x + 5
3 print(x) # Output: 15
4
5 y = 20
6 y -= 3 # equivalent to y = y - 3
7 print(y) # Output: 17
8
9 z = 7
10 z *= 2 # equivalent to z = z * 2
11 print(z) # Output: 14
12
13 a = 16
14 a /= 4 # equivalent to a = a / 4
15 print(a) # Output: 4.0
16
17 b = 5
18 b %= 2 # equivalent to b = b % 2
19 print(b) # Output: 1

```

Как видите, арифметические операторы присваивания — интересный и эффективный способ обновления значений переменных в Python.

3.6 Операторы идентификации

Объясните концепцию операторов идентичности Python в игровой форме.⁵¹

По сути, операторы идентификации используются для сравнения ячеек памяти двух переменных Python. Они возвращают логическое значение **True** (Истина) или **False** (Ложь), указывающее, являются ли переменные одинаковыми или разными с точки зрения их расположения в памяти.

Чтобы лучше понять это, представьте, что у вас есть две одинаковые на вид чашки. На первый взгляд они могут показаться совершенно одинаковыми, но если вы присмотритесь, то заметите, что на них разные отпечатки пальцев. В Python операторы идентификации похожи на проверку того, имеют ли два объекта одинаковые отпечатки пальцев или нет.

Вот два оператора идентификации Python:

- ▶ **is**: возвращает **True**, если две переменные ссылаются на один и тот же объект в памяти, и **False** в противном случае.
- ▶ **is not**: возвращает **True**, если две переменные не ссылаются на один и тот же объект в памяти, и **False** в противном случае.

Давайте рассмотрим пример. В этом коде мы определяем переменные с именами персонажей из фильма *Cars*. Затем мы используем операторы тождества, чтобы сравнить их:

```

1 # Define the cars
2 mcqueen = "Lightning McQueen"
3 doc = "Doc Hudson"
4 mater = "Tow Mater"
5
6 # Use identity operators to compare the cars
7 print(f"{mcqueen} is {doc}: ", mcqueen is doc)
8 print(f"{mcqueen} is not {mater}: ", mcqueen is not mater)
9
10 # Assign one variable to the other
11 mcqueen = doc
12 print(f"After assigning, {mcqueen} is {doc}: ", mcqueen is doc)
13
14 # Check if a car is None
15 sally = None
16 print(f"{sally} is None: ", sally is None)
17
18 sally = "Sally Carrera"
19 print(f"After assigning, {sally} is None: ", sally is None)

```

```

Lightning McQueen is Doc Hudson: False
Lightning McQueen is not Tow Mater: True
After assigning, Doc Hudson is Doc Hudson: True
None is None: True
After assigning, Sally Carrera is None: False

```

В последнем блоке мы используем оператор `is`, чтобы проверить, является ли объект, на который ссылается переменная `sally`, специальным объектом Python `None`. Первоначально результатом будет `True`, поскольку переменной не было присвоено никакого имени, но после присвоения имени результат будет `False`.

Таким образом, операторы идентификации Python позволяют сравнивать два объекта и выяснять, одинаковы они или нет, фактически не глядя на их значения.

Хорошо. Два вопроса. Во-первых, что именно ты подразумеваешь под «расположением в памяти»? Я не вижу в коде ячеек памяти; пожалуйста, уточни в дружественной и живой манере. Во-вторых, что такое специальная переменная Python «None»? Просьба уточнить.⁵²

Конечно, области памяти в программировании подобны единицам хранения, где в памяти компьютера данные хранятся. Думайте об этом как о большой сети почтовых ящиков, где каждый ящик представляет собой пространство памяти, расположенное в данной строке и столбце сети, и где каждый ящик может хранить часть информации.

Итак, когда мы создаем переменную, мы, как правило, резервируем одно из этих подпространств памяти для хранения некоторых данных, таких как число, строка или даже другой объект. Когда мы присваиваем значение этой переменной, мы сохраняем это значение в пространстве памяти, закрепленном за этой переменной.

Что касается вашего второго вопроса, в Python `None` — это специальное значение, которое означает отсутствие значения. Оно используется, чтобы указать, что переменная не имеет значения или что функция ничего не возвращает. Думайте об этом как о пустой коробке в вашем хранилище — это все еще коробка, но внутри нее ничего нет. Когда вы присваиваете переменной значение `None`, вы, по сути, резервируете место в памяти для этой переменной, но не присваиваете ей никакого фактического значения.

3.7 Операторы принадлежности

Объясни концепцию операторов принадлежности Python в игровой форме.⁵³

Операторы принадлежности Python похожи на секретных детективов, которые помогают нам выяснить, принадлежит ли конкретное значение к определенной группе или нет.

Первым оператором принадлежности является `in`, который проверяет, присутствует ли заданное значение в большей строке или нет. Это как спросить у вышибалы в танцевальном клубе, есть твое имя в списке гостей или нет.

Например:

```
1 name = "Alice"
2 another_name = "Bob"
3 letter = 'A'
4 print(f"{name} starts with {letter}: ", letter in name)
5 print(f"{another_name} starts with {letter}: ", letter in another_name)
```

```
Alice starts with A: True
Bob starts with A: False
```

В этом примере код проверяет, является ли буква «А» первой буквой любого из имен (при условии, что первая буква имени является буквой верхнего регистра).

Второй оператор принадлежности — это оператор `not in`, который проверяет, отсутствует ли заданное значение в большей строке или нет. Значение может быть буквой или последовательностью букв.

Например:


```

1 sentence1 = "Java is fun!"
2 sentence2 = "Java is fun, but Python is really fantastic!"
3 language = "Python"
4
5 print(f'"{language}" is not in "{sentence1}': ", language not in sentence1)
6 print(f'"{language}" is not in "{sentence2}': ", language not in sentence2)

```

```

'Python' is not in 'Java is fun!': True
'Python' is not in 'Java is fun, but Python is really fantastic!': False

```

Эти операторы принадлежности особенно полезны при работе со строками и другими типами данных (такими как списки, наборы или словари, которые будут рассмотрены в последующих главах).

3.8 Оператор возведения в степень

Объясни концепцию оператора возведения в степень Python в игровой форме.⁵⁴

Конечно, давайте поговорим об операторе возведения в степень! Это как иметь волшебную палочку, которая может заставить числа расти очень быстро. Представьте, что вы волшебник и у вас есть палочка, которая может умножать любое число само на себя столько раз, сколько вы хотите. Например, если вы взмахнете палочкой и скажете «2 в степени 3», вы получите $2 \times 2 \times 2 = 8$. Если вы взмахнете еще раз и скажете «2 в степени 4», вы получите $2 \times 2 \times 2 \times 2 = 16$. Оператор возведения в степень Python работает точно так же — он берет число и возводит его в степень (также известную как показатель степени). Поэтому, если вы хотите с помощью Python возвести 2 в степень 3, вы можете просто написать `2**3`, что даст вам результат 8. Вы можете использовать любое число в качестве основания и любое целое число в качестве показателя степени, и Python рассчитает результат за вас. Это похоже на волшебную силу, позволяющую увеличивать числа!

Вот пример использования оператора возведения в степень:

```

1 # Calculating the area of a square with side length 5
2 area = 5 ** 2
3 print(area) # Output: 25

```

В этом примере мы используем оператор возведения в степень (`**`), чтобы возвести длину стороны квадрата в степень 2 для вычисления его площади. Затем результат сохраняется в области переменных, который мы и выводим на консоль.

Что происходит, когда выражение имеет несколько операндов? В каком порядке они оцениваются?⁵⁵

Когда выражение имеет несколько операндов, они выполняются на основе правил приоритета операторов и ассоциативности. Операторы с более высоким приоритетом выполняются первыми, за ними следуют операторы с более низким приоритетом.

Если есть операторы с одинаковым приоритетом, порядок их выполнения зависит от их ассоциативности. Например, арифметические операторы (+, -, *, /) имеют ассоциативность слева направо, что означает, что операнды вычисляются слева направо.

Вот пример, иллюстрирующий это: $x = 2 + 3 * 4 / 2$. В этом выражении операторы умножения и деления имеют более высокий приоритет, чем оператор сложения. Итак, сначала вычисляется $3 * 4$, в результате чего получается 12. Затем вычисляется $12 / 2$, в результате чего получается 6. Наконец, вычисляется $2 + 6$, в результате чего получается 8. Следовательно, значение x будет равно 8.

При написании сложных выражений на Python важно помнить об иерархии приоритетов операций и ассоциативности, чтобы гарантировать, что код ведет себя так, как задумано. Эта иерархия определяет порядок, в котором операторы исполняются в выражении с несколькими операторами.

Чтобы помочь нам запомнить порядок старшинства, мы можем использовать аббревиатуру КЭУДСВ, которая означает круглые скобки, экспоненты, умножение и деление, сложение и вычитание. Это означает, что сначала вычисляются операторы в круглых скобках, затем осуществляется возведение в степень, затем умножение и деление (которые имеют одинаковый приоритет и выполняются слева направо) и, наконец, сложение и вычитание (которые также имеют одинаковый приоритет и выполняются слева направо).

Однако важно отметить, что мы можем изменить порядок выполнения, используя круглые скобки для группировки определенных операций. Например, в выражении $(3 + 4) * 5$ сначала выполняется сложение в скобках, в результате чего получается значение 7, которое затем умножается на 5, чтобы получить окончательный результат 35. Напротив, выражение $3 + 4 * 5$ дает 23.

3.9 Расширенные математические операторы

Как я могу запрограммировать более сложные математические операции, такие как извлечение квадратного корня, вычисление логарифма и т. д.?⁵⁶

Python предоставляет встроенный модуль `math`, который предлагает множество функций для выполнения сложных математических операций. Вот несколько примеров:

```

1
2 # Import the built-in math module for various mathematical operations
3 import math
4
5 # Calculate the square root of a number
6 num = 25
7 sqrt_num = math.sqrt(num) # Use the sqrt() function to calculate the square root
8 print(f"The square root of {num} is {sqrt_num}")
9
10 # Calculate the logarithm of a number with base 10
11 num2 = 1000
12 log_num2 = math.log10(num2) # Use the log10() function to calculate the base 10
    logarithm
13 print(f"The base 10 logarithm of {num2} is {log_num2}")
14
15 # Get the value of pi, a mathematical constant
16 pi = math.pi
17 print(f"The value of pi is {pi}")
18
19 # Convert degrees to radians
20 deg = 45
21 rad = math.radians(deg) # Use the radians() function to convert degrees to radians
22 print(f"{deg} degrees is equal to {rad} radians")
23
24 # Calculate the sine of an angle in radians
25 angle = math.pi / 4
26 sin_angle = math.sin(angle) # Use the sin() function to calculate the sine of the angle
27 print(f"The sine of {angle} radians is {sin_angle}")

```

Визуализируйте пошаговое выполнение этого кода в Python Tutor: <https://tinyurl.com/3r4vb9y5>

```

The square root of 25 is 5.0
The base 10 logarithm of 1000 is 3.0
The value of pi is 3.141592653589793
45 degrees is equal to 0.7853981633974483 radians
The sine of 0.7853981633974483 radians is 0.7071067811865475

```

В этом примере мы импортировали модуль `math` и использовали его встроенные функции для выполнения сложных математических операций. Мы вычислили квадратный корень числа с помощью функции `sqrt()`, логарифм числа по основанию 10 с помощью функции `log10()` и синус угла с помощью функции `sin()`. Мы также использовали встроенную константу `pi` для вычисления значения `pi` и конвертировали градусы в радианы с помощью

функции `radians()`.

Используя модуль `math`, вы можете легко выполнять сложные математические операции в своих программах на Python.

3.10 Время поработать руками!

Предложи несколько забавных и информативных упражнений по кодированию для практики в арифметике, сравнение, логических, побитовых операциях, принадлежности, возведении в степень и математических операторах.⁵⁷

Вот несколько упражнений по кодированию для обработки различных операторов Python:

1. Арифметические операторы:

- a) Напишите программу, которая вычисляет площадь круга заданного радиуса, используя оператор `*` для умножения и оператор `**` для возведения в степень.
- b) Напишите программу, которая переводит заданное количество минут в часы и минуты с помощью операторов `/` и `%`.
- c) Напишите программу, которая переводит время из военного формата в гражданский.
- d) Напишите программу, которая вычисляет окончательную цену, включая налог, с учетом цены товара и процента НДС.

2. Операторы сравнения:

- a) Напишите программу, которая проверяет, является ли заданное число положительным, отрицательным или нулем, используя операторы `>`, `<` и `==`.
- b) Напишите программу, которая проверяет, является ли заданное число четным или нечетным, используя оператор `%` для модуля и оператор `==`.
- c) Напишите программу, которая проверяет, является ли данный год високосным (т. е. делится ли он на 4, но не делится на 100, за исключением лет, кратных 400), используя операторы `%` и `AND`.

3. Логические операторы:

- a) Напишите программу, которая по показаниям температуры проверяет, попадает ли она в нормальный диапазон температур человеческого тела.
- b) Напишите программу, которая проверяет, содержит ли данное предложение слова «dog» и «cat», используя оператор `AND` и оператор `IN`.

4. Побитовые операторы:

- a) Напишите программу, которая подсчитывает количество битов, равных 1, в двоичном представлении заданного целого числа в диапазоне `[0,15]` (т. е. максимум четыре бита), используя операторы `«»`, `&` и `+=`.
- b) Напишите программу, которая зеркально отражает биты двоичного

представления заданного целого числа в диапазоне [0,15] (чтобы двоичное представление оставалось в пределах 4 бит), используя побитовый оператор XOR.

- c) Напишите программу, которая по заданному целому числу выводит удвоенную сумму без использования арифметических операторов.
- d) Напишите программу, которая по заданному четному целому числу выводит половину суммы без использования арифметических операторов.

5. Операторы принадлежности:

- a) Напишите программу, которая проверяет, начинается ли данное имя с согласной (не с гласной), используя операторы NOT, ИЛИ и IN. Предположим, что имя начинается с заглавной буквы.
- b) Напишите программу, которая проверяет, соответствует ли адрес электронной почты, введенный пользователем, минимальным допустимым критериям, включая наличие символа «at» (@) и суффикса «.com».

6. Оператор возведения в степень:

- a) Напишите программу, которая вычисляет куб заданного числа с помощью оператора **.
- b) Напишите программу, которая вычисляет квадратный корень заданного числа, используя оператор ** и показатель степени.

7. Математические операторы:

- a) Напишите программу, которая находит гипотенузу прямоугольного треугольника по длинам двух его катетов.
- b) Напишите программу, которая вычисляет расстояние между двумя точками (x1, y1) и (x2, y2), используя оператор - для вычитания, оператор ** для возведения в степень и математическую функцию sqrt().
- c) Напишите программу, которая преобразует градусы в радианы и наоборот, используя математические функции radians () и degrees().
- d) Напишите программу, которая вычисляет высоту объекта, используя длину его тени и угол подъема солнца, используя тригонометрическую функцию tan().

Предложи красивую и увлекательную задачу по кодированию, чтобы попрактиковаться в двоичном представлении целых чисел. Предположим, у нас есть таблица блюд со столбцами, представляющими ингредиенты, использованные для их приготовления, и последний столбец, отображающий целочисленный код блюда в результате интерпретации столбцов ингредиентов как битов двоичного числа (покажи таблицу с четырьмя блюдами и пятью ингредиентами). Программа должна запрашивать название блюда и ряд вопросов типа «да/нет» для определения целочисленного кода блюда, а затем распечатать строку таблицы с этой информацией (показать пример вывода программы).⁵⁸

Проблема: декодер кода блюда

У вас есть таблица блюд с колонками, представляющими ингредиенты, использованные для их приготовления. Каждое блюдо уникально идентифицируется целочисленным кодом, основанным на его ингредиентах. Вам нужно создать программу, которая запрашивает у пользователя название блюда и задает серию вопросов типа «да/нет», чтобы определить целочисленный код блюда на основе ингредиентов.

Вот пример таблицы с четырьмя блюдами и пятью ингредиентами:

Блюдо	Сыр	Латук	Томат	Лук	Мясо	Код
Чизбургер	Да	Да	Да	Нет	Да	29
Пицца Маргарита	Да	Нет	Да	Нет	Нет	20
Салат Цезарь	Да	Да	Нет	Да	Да	27
Мороженое	Нет	Нет	Нет	Нет	Нет	0

Ваша программа должна предложить пользователю ввести название блюда, а затем задать ряд вопросов, чтобы определить наличие каждого ингредиента. На основе ответов пользователя программа должна вычислить и отобразить соответствующую строку таблицы, включая вычисленный код.

Пример вывода:

```

Welcome to the Dish Code Decoder!

Let's find out the dish code based on the ingredients.

Please enter the name of the dish: Pizza

Does the dish contain Cheese? (Y/N): Y
Does the dish contain Lettuce? (Y/N): N
Does the dish contain Tomato? (Y/N): Y
Does the dish contain Onion? (Y/N): Y
Does the dish contain Meat? (Y/N): Y

The code for the dish is: 11

Here is the dish you selected:
|   Dish   | Cheese | Lettuce | Tomato | Onion | Meat || Code |
|-----|-----|-----|-----|-----|-----||-----|
| Cheeseburger | Yes   | Yes    | Yes   | No   | Yes  || 29   |

Thank you for using the Dish Code Decoder! Bon appetit!

```

Вы можете попробовать добавить в таблицу новые блюда, например пасту или омлет, и посмотреть, как программа точно определяет их целочисленные коды на основе ингредиентов. Это отличная возможность укрепить свои навыки обработки пользовательского ввода и работы с двоичными представлениями.

Наслаждайтесь созданием декодера Dish Code!

(NB: Вам может понадобиться использовать оператор `if/else`, упомянутый в Главе 2, для обработки пользовательского ввода и вычисления целочисленного кода. Подробнее об операторах `if/else` будет рассказано в Главе 4.)

Башня управления

Перечисли все управляющие структуры в программировании и дай краткое описание каждой из них.⁵⁹

В программировании существует несколько управляющих структур, которые позволят вам управлять потоком выполнения вашего кода. Эти управляющие структуры можно разделить на три основных типа: ветвящиеся структуры, повторяющиеся структуры и вложенные структуры.

1. Ветвящиеся структуры (условные операторы):

- ▶ **Оператор If-Else:** Исполняет блок кода, если выполняется определенное условие. Он также может включать необязательный оператор `else`, чтобы обеспечить выполнение альтернативного блока кода, если условие ложно.
- ▶ **Оператор Match-case:** Оценивает выражение и сопоставляет его с несколькими вариантами, выполняя блок кода, связанный с соответствующим случаем.

2. Повторяющиеся структуры (циклы):

- ▶ **Цикл For:** Исполняет блок кода для определенного количества итераций. Часто используется, когда количество итераций известно заранее.
- ▶ **Цикл While:** Повторно исполняет блок кода до тех пор, пока выполняется указанное условие. Полезно, когда количество итераций заранее неизвестно.
- ▶ **Цикл Do-While:** Аналогичен циклу `while`, но гарантирует, что блок кода будет выполнен хотя бы один раз перед проверкой условия.

3. Вложенные структуры:

- ▶ **Вложенный условный оператор:** Позволяет использовать оператор `if-else` внутри другого оператора `if-else`. Он обеспечивает более сложное ветвление на основе множества условий.
- ▶ **Вложенный цикл:** Предполагает размещение одного цикла внутри другого цикла. Это позволяет более сложно контролировать повторяющиеся задачи.

Эти управляющие структуры дадут вам возможность принимать решения, повторять действия и разрабатывать более сложную программную логику. Эффективно комбинируя эти структуры, вы можете создавать мощные и гибкие программы, отвечающие различным требованиям.

Итак, давайте подробнее рассмотрим каждую из них.

4.1 Операторы If/Else

Объясни все, что мне нужно знать об операторах if/else языка Python, в увлекательной и творческой форме с помощью забавных примеров.⁶⁰

Конечно! Давайте погрузимся в захватывающий мир условных операторов Python.

Представьте, что вы детектив, расследующий дела в охваченном преступностью городе. Ваша задача — принимать решения, основываясь на уликах, которые вы собираете. Это именно то, для чего нужны операторы if/else — принятие решений в вашем коде!

Вот забавный пример: допустим, вы детектив, расследующий загадочное преступление. У вас есть подозреваемый, и вам нужно определить, виновен он или невиновен. Вы можете использовать оператор if/else, чтобы помочь себе. Общая структура этого условного оператора такова:

```
1 if condition:
2     # Code to execute if the condition is true
3 else:
4     # Code to execute if the condition is false
```

Итак, давайте закодируем предложенный пример:

```
1 # Ask the user to enter their name
2 suspect = input("Enter your name: ")
3
4 # Check if the entered name matches "Jack the Ripper"
5 if suspect == "Jack the Ripper":
6     # If the name matches, print an arrest message
7     print(f"You're under arrest, {suspect}! You're guilty!")
8 else:
9     # If the name doesn't match, print a message indicating innocence
10    print(f"You're free to go {suspect}! You're innocent!")
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/ykdth9bd>

Здесь у вас есть два разных варианта запуска этой программы:

```
Enter your name: Sherlock Holmes
You're free to go Sherlock Holmes! You're innocent!
```

```
Enter your name: Jack the Ripper
You're under arrest, Jack the Ripper! You're guilty!
```

Кроме того, вы также можете использовать операторы `if/elif/else` для обработки более двух разных сценариев. Такие сценарии можно создать, объединив несколько условий с помощью логических операторов. Допустим, вы хотите определить, хороший ли сегодня день для посещения пляжа, исходя из погоды. Вот пример:

```

1 # Set the initial conditions
2 sunny = True
3 temperature = 22
4
5 # If it's sunny and the temperature is above 25 degrees
6 if sunny and temperature > 25:
7     print("It's a perfect day for the beach!")
8 # If it's sunny but the temperature is 25 degrees or lower
9 elif sunny and temperature <= 25:
10    print("It's a bit chilly, but you can still enjoy the beach!")
11 # If it's not sunny
12 else:
13    print("Maybe it's better to stay home today.")

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/362yh7w6>

В этом коде у нас есть две переменные: `sunny` и `temperature`. Давайте пройдемся по коду и поймем, как исполняются условные операторы, а также роль каждой переменной:

1. Установлены следующие исходные условия `sunny = True` и `temperature = 22`.
2. Первое условное выражение `if sunny and temperature > 25`. Это условие проверяет, солнечно ли сейчас (`sunny == True`) и превышает ли температура 25 degrees (`temperature > 25`). Если оба условия верны, код внутри этого блока будет выполнен, а на экран будет выведено сообщение "It's a perfect day for the beach!" (Это идеальный день для пляжа!).
3. Если первое условие не выполняется, программа переходит к следующему условному оператору, `elif sunny and temperature <= 25`. Это условие проверяет, солнечно ли сейчас (`sunny == True`) и не находится ли температура ниже 25 градусов (`temperature <= 25`). Если оба условия истинны, код внутри этого блока будет выполнен, а на экран будет выведено сообщение "It's a bit chilly, but you can still enjoy the beach!" (Немного прохладно, но вы все равно можете наслаждаться пляжем!).
4. Если ни первое, ни второе условие не выполняются, программа переходит к блоку `else`. Этот блок будет выполняться только тогда, когда не будет солнечно (`sunny == False`). В этом случае на экран будет выведено сообщение "Может быть, сегодня лучше остаться дома." ().

It's a bit chilly, but you can still enjoy the beach!

Порядок условий важен, потому что алгоритм выполняет их одно за другим сверху вниз.

Как только условие истинно, выполняется соответствующий блок кода, и программа выходит из структуры `if/elif/else`. Это означает, что будет выполняться только один блок кода, в зависимости от того, какое условие выполняется первым. В данном случае переменная `temperature` не влияет на выполнение блока `else`, потому что поток алгоритма достигает этой точки только тогда, когда предыдущие условия `if` и `elif`, включающие переменную `sunny` не выполняются, то есть когда `sunny == False`. Следовательно, блок `else` будет выполняться, когда не будет солнечно, вне зависимости от значения переменной `temperature`.

Помните, что операторы `if/else` могут обрабатывать различные условия и позволяют вашей программе принимать различные решения на основе этих условий. Они похожи на выбор, который вы делаете в игре и который определяет результат.

Итак, разгадываете ли вы загадки, планируете пляжные поездки или создаете захватывающие программы, операторы `if/else` в Python — ваши надежные инструменты для принятия решений в ваших алгоритмах.

Есть ли что-нибудь еще, что я должен знать об условных выражениях в Python?⁶¹

Да, есть еще кое-что, что нужно знать об условных выражениях в Python:

- **Вложенные условные операторы:** Вы можете использовать условные операторы внутри условных операторов для реализации более сложных сценариев. Они известны как вложенные условные операторы. Внутри блока `if` или `elif` вы можете поместить еще один оператор `if` для дальнейшего уточнения условий с дополнительными критериями.

(NB: Подробнее о вложенных структурах мы поговорим в конце главы.)

- **Цепочка условных операторов:** В некоторых случаях у вас может быть несколько перекрывающихся условий, то есть алгоритм может выполнять несколько блоков `if` в зависимости от значения переменной. Синтаксис для цепочек условных операторов следующий:

```

1 if condition1:
2     # code block
3 if condition2:
4     # code block
5 if condition3:
6     # code block

```

В этом случае каждый оператор `if` будет выполняться независимо, и их соответствующие блоки кода будут выполняться, когда их условия будут истинными `True`.

Вот пример цепочки условных выражений, в которых условия могут перекрываться, что приводит к выполнению нескольких блоков `if`:

```

1 # Prompt the user for their age
2 age = int(input("How old are you? "))
3
4 # Check the age range using chained conditionals
5 if age > 13:
6     print("Congratulations, you've survived your childhood!")
7 if age > 18:
8     print("You've made it through the rollercoaster of teenage years!")
9 if age > 35:
10    print("You're now a fully-fledged adult, and bills are your new best friends!")
11 if age >= 65:
12    print("Welcome to the exclusive club of senior citizens! Discounts await!")

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/2s42xwnc>

В этом коде условия не являются исключительными, и каждый блок будет выполняться, если будет выполнено соответствующее возрастное условие. Например, если пользователь введет возраст 15 лет, программа напечатает "Congratulations, you've survived your childhood !" (Поздравляем, вы пережили свое детство!). Если пользователь введет возраст 25, программа напечатает оба "Congratulations, you've survived your childhood!" (Поздравляем, вы пережили свое детство!) и "You've made it through the rollercoaster of teenage years !" (Вы преодолели американские горки подросткового возраста!). Давайте посмотрим, что произойдет, когда пользователь введет возраст 40 лет:

```

How old are you? 40
Congratulations, you've survived your childhood!
You've made it through the rollercoaster of teenage years!
You're now a fully-fledged adult, and bills are your new best friends!

```

Как видите, эти распечатки отражают этапы, пройденные пользователем, в зависимости от его возраста, что позволяет одновременно выполнять несколько условий.

- ▶ **Упрощенное вычисление:** Python использует упрощенное вычисление в условных операторах. Это означает, что если результат условия может быть определен на основе первой части условия, вторая часть выполняться не будет. Например, в условии `and`, если первый операнд имеет значение `False`, второй операнд не будет выполняться, поскольку общий результат всегда будет `False`.
- ▶ **Тернарный оператор:** Python также предоставляет компактный способ написания простых условных выражений, называемый тернарным оператором. Это позволяет вам оценить условие и вернуть одно значение, если условие истинно, и другое значение, если условие ложно. Синтаксис следующий: `value = true_value if condition else`

`false_value`. Пример:

```

1 # Prompt the user for the speed of their vehicle
2 speed = int(input("Officer: What was the speed of your vehicle in mph? "))
3
4 # Use the ternary operator to determine the response in the officer's message
5 response = "speeding ticket, sorry" if speed > 70 else "high, five!"
6 print(f"Officer: Alright, I may have to issue you a...{response}. Drive safe!")

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/e6b9v554>

В этом коде оператор печати включает в ответ офицера результат тернарного оператора, что формирует сообщение с оттенком юмора, выписывая «speeding ticket», если скорость превышает 70 миль в час, и игривое «high five», если скорость в пределах нормы. Сообщение также призывает к безопасному вождению:

```

Officer: What was the speed of your vehicle in mph? 45
Officer: Alright, I may have to issue you a... high, five!. Drive safe!

```

```

Officer: What was the speed of your vehicle in mph? 85
Officer: Alright, I may have to issue you a... speeding ticket, sorry. Drive
safe!

```

Таким образом, вышеупомянутые аспекты являются дополнительными соображениями, которые следует учитывать при работе с условными выражениями в Python. Четкое понимание этих концепций позволит вам писать более гибкий и эффективный код при работе со сценариями, предполагающими принятие сложных решений.

4.2 Операторы `match/case`

Теперь давай поговорим об операторах `match/case` в Python. Объясни все, что мне нужно знать о них, в веселой и творческой форме, используя игровые примеры.⁶²

Начиная с Python 3.10, оператор `match/case` представлен как новая функция, называемая «сопоставлением структурного шаблона». Он предоставляет мощный и краткий способ сопоставления, извлечения значений и решения случаев или ответвлений на основе шаблонов.

Основной синтаксис оператора `match/case` в Python выглядит следующим образом:

```

1 match expression:
2 case value1:
3     # Code to execute if expression matches value1
4 case value2:
5     # Code to execute if expression matches value2
6 ...
7 case valueN:
8     # Code to execute if expression matches valueN
9 case _:
10    # Code to execute if no cases match

```

Вот расшифровка синтаксиса:

- ▶ **expression:** Выражение, значение которого будет сравниваться с блоками `case`.
- ▶ **value1, value2, valueN:** Возможные значения, которым может соответствовать выражение. Это могут быть константы, переменные или даже шаблоны.
- ▶ **_:** Случай подстановочного знака, который соответствует любому значению, которое не было явно указано в предыдущих блоках `case`.

Внутри каждого блока `case` код выполняется, когда выражение соответствует соответствующему значению. Если совпадают несколько блоков `case`, выполняется только код из первого совпадающего блока `case`.

Оператор `match/case` обеспечивает краткий способ обработки нескольких случаев на основе значения выражения, делая код более читабельным и устраняя необходимость в длинных цепочках операторов `if/elif/else`.

Помните, что оператор `match/case` был представлен в Python 3.10, поэтому убедитесь, что у вас установлена соответствующая версия Python для использования этой функции.

Давайте рассмотрим оператор `match/case` на вымышленном примере заказа пиццы. Представьте, что вы любитель пиццы и хотите заказать пиццу с разными начинками. Итак, давайте посмотрим, как мы можем использовать оператор `match/case` для обработки этой ситуации:

```

1 pizza = input("What pizza would you like today? (veggie/meat/hawaiian)")
2
3 match pizza.lower():
4     case "veggie":
5         toppings = "Mushrooms, olives, and peppers"
6     case "meat":
7         toppings = "Ground beef, sausage, and pepperoni"
8     case "hawaiian":
9         toppings = "Ham, pineapple, and extra cheese"
10    case _:
11        toppings = "Tomato and cheese only"
12
13 print(f"You ordered a {pizza} pizza with toppings: {toppings}")

```

В этом примере мы спрашиваем пользователя, какую пиццу он хочет заказать в первую очередь. Затем с помощью оператора `match/case` определяются соответствующие начинки. Каждый `case` блоке `match` проверяет, соответствует ли наименование пиццы одному из указанных значений. Если совпадение найдено, соответствующие ингредиенты присваиваются переменной `toppings`. Если ни один из случаев не соответствует пицце, срабатывает случай подстановки, указывающий на базовую начинку для пиццы:

```

What pizza would you like today? (veggie/meat/hawaiian) Hawaiian
You ordered a Hawaiian pizza with toppings: Ham, pineapple, and extra cheese

```

```

What pizza would you like today? (veggie/meat/hawaiian) surprise me!
You ordered a surprise me! pizza with toppings: Tomato and cheese only

```

С оператором `match/case` вы можете элегантно обрабатывать различные случаи в зависимости от совпадающего значения. Это улучшает читаемость и снижает потребность в сложных структурах `if/elif/else`.

Хорошо. В операторе `match` я вижу, что вы использовали команду «`lower()`». Нужно ли вставлять его каждый раз, когда я использую в коде оператор `match/case`?⁶³

Нет, использование команды `lower()` в операторе `match` было характерно для примера с начинкой для пиццы. Он использовался для преобразования пользовательского ввода в нижний регистр, чтобы сопоставление не учитывало регистр. Это означает, что независимо от того, вводит ли пользователь «`veggie`», «`VEGGIE`» или «`VeGGie`», это будет считаться совпадением.

В общем, вам не нужно использовать `lower()` в каждом операторе `match/case`. Это необходимо только в том случае, если вы хотите выполнить сопоставление без учета

регистра. Если регистр входных значений не имеет значения или если вы хотите, чтобы принимался во внимание регистр, вы можете опустить команду `lower()`.

4.3 Операторы цикла For

Пожалуйста, дай краткое техническое определение цикла for.⁶⁴

Конечно! Цикл `for` — это программная конструкция, которая позволяет многократно выполнять блок кода определенное количество раз или для каждого элемента в коллекции. Она следует общему синтаксису:

```
1 for variable in iterable:
2     # Code block to be executed
```

В этом синтаксисе:

1. **variable:** Это переменная, которая принимает значение каждого элемента в итерируемом объекте во время каждой итерации (повторении) цикла.
2. **iterable:** Представляет последовательность или набор элементов, например диапазон, список или строку.
3. **code block:** Является кодом, следующим с отступом под оператором `for`, состоящим из набора инструкций, которые должны выполняться многократно.

Во время каждой итерации цикла переменная принимает значение следующего элемента в итерации, и выполняется блок кода. Этот процесс продолжается до тех пор, пока все элементы в итерируемом объекте не будут обработаны.

Цикл `for` особенно полезен, когда мы заранее знаем количество итераций, которые мы хотим, чтобы блок кода повторялся, или когда мы хотим перебрать набор элементов. Это упрощает повторяющиеся задачи и позволяет нам эффективно выполнять операции без написания избыточного кода.

Теперь давай посмотрим на циклы for в действии в веселой и творческой форме, используя забавные примеры.⁶⁵

Конечно! Давайте изучим мир операторов цикла `for` языка Python на нескольких забавных примерах.

4. *Подсчет с легкостью.* Цикл `for` позволяет без труда подсчитывать числа в указанном диапазоне. Например, чтобы посчитать от 1 до 10, можно использовать функцию `range()`:


```

1 # range(1, 11) generates a sequence from 1 to 11-1, i.e from 1 to 10
2 for number in range(1, 11):
3     print(number)

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/pcvbud98>

Этот код будет итеративно печатать числа от 1 до 10, что упрощает подсчет.

```

1
2
3
4
5
6
7
8
9
10

```

5. *Шаг к успеху.* Вы также можете контролировать размер шага при переборе диапазона. Предположим, вы хотите сосчитать от 1 до 10, но включать только четные числа. Благодаря параметру шага вы можете добиться этого:

```

1 # the last parameter in range() is the count step
2 for number in range(2, 11, 2):
3     print(number)

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/ycysvpds>

В этом случае цикл начинается с 2 и увеличивается на 2, пока не достигнет 10, печатая при этом только четные числа.

```

2
4
6
8
10

```

6. *Побуквенный вывод Волшебного Слова.* Цикл `for` не ограничивается числами; он также может перебирать символы строки. Допустим, у нас есть волшебное слово «abracadabra» («абракадабра»), и мы хотим отобразить его буквы одну за другой:

```

1 magic_word = "abracadabra"
2 print(f"Let's unveil the magical letters of '{magic_word}': ")
3 for letter in magic_word:
4     print("+---+")
5     print(f"| {letter} |")

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/2p8vdpnm>

Запуск этого кода отобразит каждую букву волшебного слова в волшебных квадратиках на отдельных строках, раскрывая его скрытую силу.

```

Let's unveil the magical letters of 'abracadabra':
+---+
| a |
+---+
| b |
+---+
| r |
+---+
| a |
+---+
| c |
+---+
| a |
+---+
| d |
+---+
| a |
+---+
| b |
+---+
| r |
+---+
| a |

```

7. *Заклинание умножения.* Цикл **for** можно использовать для создания таблиц умножения без особых усилий. Предположим, вы хотите распечатать таблицу умножения для заданного числа от 1 до 10:

```

1 number = int(input("Enter a number to cast a multiplication spell: "))
2 for i in range(1, 11):
3     result = number * i
4     print(f"{number} x {i} = {result}")

```

Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/3fmxfdap>

Запуск этого кода отобразит таблицу умножения для числа, введенного пользователем, показывая каждую операцию умножения вместе с соответствующим результатом.

```

Enter a number to cast a multiplication spell: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

```

5. *Преобразование строк.* Цикл `for` также можно использовать для манипулирования строками, например для преобразования регистра каждого символа. Допустим, у нас есть строка «Hello, World!», и мы хотим преобразовать ее в символы верхнего регистра:

```

1 text = "Hello, World!"
2 converted_text = ""
3 for char in text:
4     converted_text += char.upper()
5 print("Original text : ", text)
6 print("Converted text: ", converted_text)

```

Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/2p8kw9ee>

Запуск этого кода напечатает «HELLO, WORLD!», преобразуя все символы в строке в верхний регистр.

```

Original text : Hello, World!
Converted text: HELLO, WORLD!

```

Как видите, цикл `for` — мощная конструкция, упрощающая итерацию и позволяющая выполнять широкий круг задач, от создания математических шаблонов до манипулирования строками. Его универсальность и простота использования делают его ценным инструментом в вашем путешествии по программированию.

4.4 Операторы цикла `While`

*Пожалуйста, дай краткое техническое определение цикла `while`.*⁶⁶

Цикл `while` — это оператор управления потоком в программировании, который позволяет многократно выполнять блок кода до тех пор, пока заданное условие выполняется. Он предоставляет способ создания циклов, которые повторяются динамически на основе оценки условия, а не заранее определенного количества итераций. Цикл `while` состоит из условия, которое проверяется перед каждой итерацией, и пока условие остается истинным, выполняется блок кода внутри цикла. Цикл продолжается до тех пор, пока условие не станет ложным `False`, после чего программа выходит из цикла и продолжает следующий набор инструкций.

Синтаксис цикла `while` следующий:

```
1 while condition:
2     # Code block to be executed
```

В этом синтаксисе условие `condition` представляет собой логическое выражение, которое вычисляется перед каждой итерацией. Пока `condition` является истинным, блок кода `code block` будет выполняться; если оно ложно, цикл завершается. Внутри `code block`, вы можете выполнять любые нужные операции или вычисления. После выполнения `code block` программа возвращается к началу цикла и повторно оценивает условие, чтобы определить, продолжить выполнение цикла или выйти из него.

Чтобы избежать бесконечных циклов, которые продолжают выполняться бесконечно без остановки, важно убедиться, что `condition` в конечном итоге становится ложным. Если `condition` никогда не становится ложным, цикл будет повторяться бесконечно, что приведет к зависанию или сбою программы. Чтобы предотвратить это, очень важно спроектировать `condition` таким образом, чтобы цикл можно было завершить. Этого можно достичь, включив в цикл переменные или операторы, которые обновляют или изменяют `condition`, гарантируя, что оно в конечном итоге будет оценено как ложное и покинет цикл.

*Теперь давайте рассмотрим циклы `while` в действии весело и творчески, используя забавные примеры.*⁶⁷

Конечно! Давайте рассмотрим циклы `while` на некоторых забавных примерах:

1. *Считаем овец.* Представьте, что вы пытаетесь заснуть, но не можете. Давайте напишем программу, которая считает овец, пока вы окончательно не заснете. Вот ее код:

```

1 # Initialize the variable sheep to 0
2 sheep = 0
3
4 # Prompt the user for their sleep status
5 asleep = input("Are you asleep yet (yes/no)? ")
6
7 # Continue the loop as long as the user is not asleep
8 while asleep.lower() != "yes":
9     sheep += 1
10    print(f"Counting sheep #{sheep}: Baaa...\n")
11    asleep = input("Are you asleep yet (yes/no)? ")
12
13 # Once the loop is terminated, acknowledge that the user is asleep
14 print("You've fallen asleep! Sweet dreams!")

```



Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/2p8kw9ee>

В этом коде мы имитируем подсчет овец, чтобы помочь пользователю заснуть. Программа запрашивает у пользователя статус сна с помощью функции `input()`. Цикл `while` продолжается до тех пор, пока пользователь не ответит "yes" («да»).

Внутри цикла мы увеличиваем количество овец `sheep` на 1 и печатаем сообщение, чтобы помочь пользователю визуализировать овцу, перепрыгивающую через ограду. Затем мы снова запрашиваем у пользователя статус сна. Как только пользователь отвечает "yes", условие цикла `while` становится `False`, и программа выходит из цикла. Наконец, мы печатаем сообщение о том, что пользователь заснул.

Важно отметить, что в этом примере мы заранее не знаем, сколько раз будет повторяться цикл. Цикл продолжается до тех пор, пока пользователь не ответит "yes", и количество повторений цикла зависит от ввода пользователя.

```

Are you asleep yet (yes/no)? no
Counting sheep #1: Baaa...

Are you asleep yet (yes/no)? no
Counting sheep #2: Baaa...

Are you asleep yet (yes/no)? no
Counting sheep #3: Baaa...

Are you asleep yet (yes/no)? yes
You've fallen asleep! Sweet dreams!

```

2. *Разблокировка сундука с сокровищами.* Представьте себя охотником за сокровищами, пытающимся открыть таинственный сундук. К сожалению, сундук заперт, и вам нужно продолжать угадывать секретный код до тех пор, пока не найдется код, открывающий его. Давайте напишем программу, имитирующую эту охоту за сокровищами:

```

1 # Define a secret word and generate the secret code
2 secret_word = "OpenSesame"
3 secret_code = secret_word.swapcase()
4
5 # Prompt the user to enter their guess
6 guess = input("Enter the secret code: ")
7
8 # Continue the loop as long as the guess is incorrect
9 while guess != secret_code:
10     print("Wrong code! Try again.\n")
11     guess = input("Enter the secret code: ")
12
13 # Print a message indicating the chest open
14 print("Congratulations! You've unlocked the treasure chest!")

```



Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/2p8kw9ee>

В этом коде мы моделируем секретный код, который нужно правильно угадать, чтобы открыть сундук с сокровищами. `secret_word` определяется как "OpenSesame". Мы генерируем `secret_code`, меняя регистр символов в `secret-word` с помощью метода `swapcase()`.

Программа предлагает пользователю ввести свое предположение с помощью функции `input()`. Цикл `while` продолжается до тех пор, пока догадка не равна `secret_code`. Внутри цикла мы печатаем сообщение о том, что `guess` неверно, и снова

предлагаем пользователю ввести свою догадку. Как только пользователь вводит правильный `secret_code`, условие цикла становится `False`, и программа выходит из цикла. Наконец, мы печатаем сообщение, поздравляющее пользователя с открытием сундука с сокровищами.

```
Enter the secret code: OpenSesame
Wrong code! Try again.

Enter the secret code: opensesame
Wrong code! Try again.

Enter the secret code: oPENSesame
Congratulations! You've unlocked the treasure chest!
```

Эти забавные примеры демонстрируют, как можно использовать циклы `while` для создания интерактивных и увлекательных программ. Используя творческие сценарии, мы можем сделать программирование более увлекательным и приятным.

В примере «Считаем овец» вы спрашиваете пользователя дважды, спит ли он. Разве это не лишнее! Есть ли способ написать цикл, не задавая один и тот же вопрос дважды!⁶⁸

Вы правы, спрашивать пользователя дважды в примере «Считаем овец», спит ли он, действительно излишне. Мы можем упростить цикл и избежать повторения одного и того же вопроса дважды, используя цикл `while True` с условным оператором `break`. Вот обновленная версия кода, которая решает эту проблему:

```
1 sheep = 0
2
3 while True:
4     asleep = input("Are you asleep yet (yes/no)? ")
5     if asleep == "yes":
6         break
7     sheep += 1
8     print(f"Counting sheep #{sheep}: Baaa...")
9
10 print("You've fallen asleep! Sweet dreams!")
```



Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/2p8kw9ee>

В этом исправленном коде цикл `while True` будет продолжаться бесконечно, пока не встретится оператор `break`. Внутри цикла мы запрашиваем у пользователя статус сна с помощью функции `input()`. Если пользователь отвечает "yes", мы выходим из цикла и переходим к следующему оператору после цикла. В противном случае, если пользователь

отвечает "no", мы увеличиваем число `sheep` на 1 и печатаем сообщение, чтобы визуализировать овцу, перепрыгивающую через изгородь.

Используя этот подход, мы устраняем избыточный вопрос и добиваемся желаемого поведения, спрашивая пользователя, спит ли он, только один раз во всем коде.

Есть ли что-нибудь еще, что я должен знать о циклах `for` или `while` в Python?⁶⁹

Вот несколько дополнительных моментов, которые следует помнить о циклах `for` и `while` в Python:

► циклы `for`:

- Функция `range()` обычно используется в циклах `for` для создания последовательности чисел, которую можно повторять.
- Оператор `break` можно использовать для преждевременного выхода из цикла `for` до завершения всех итераций.
- Оператор `continue` может использоваться для пропуска текущей итерации и перехода к следующей итерации цикла.
- Вы можете организовывать циклы для различных структур данных Python, включая строки, списки, кортежи и словари.
(NB: Подробнее об этих структурах в следующей главе)

► циклы `while`:

- Условие в цикле `while` оценивается перед каждой итерацией, и цикл продолжается до тех пор, пока условие остается истинным.
- Важно убедиться, что условие в конечном итоге становится ложным, чтобы избежать бесконечных циклов, которые могут привести к бесконечному запуску программы.
- Оператор `break` может использоваться для преждевременного выхода из цикла `while` на основе определенного условия.
- Оператор `continue` может использоваться для пропуска текущей итерации и продолжения следующей итерации цикла.
- Циклы `while` полезны, когда вы не знаете заранее, сколько раз цикл должен повториться, поскольку количество итераций зависит от оцениваемого условия.

Понимание того, как эффективно использовать циклы, может значительно повысить вашу способность перебирать данные, выполнять повторяющиеся задачи и решать сложные проблемы в Python. Рекомендуется всегда тестировать и отлаживать циклы, чтобы убедиться, что они ведут себя так, как ожидается, и правильно обрабатывают различные сценарии.

4.5 Вложенные управляющие структуры

Что такое вложенная структура управления и почему она полезна?⁷⁰

Вложенная управляющая структура — это управляющая структура (например, оператор `if`, цикл `for`, либо цикл `while loop`), которая размещается внутри другой управляющей структуры. Это позволяет реализовать в программе несколько уровней ветвления или заикливания.

Вложенные управляющие структуры — это мощные инструменты программирования, которые позволяют обрабатывать сложную логику и выполнять сложные задачи. Они позволяют создавать сложные процессы принятия решений и повторяющиеся операции, обеспечивая гибкость и адаптируемость программ. Благодаря вложенным управляющим структурам вы можете разрабатывать код, реагирующий на множество условий, выполняющий повторяющиеся задачи различной сложности и реализующий сложные алгоритмы. Этот динамический подход делает программы более эффективными и способными решать сложные задачи, поскольку они могут адаптироваться к изменяющимся условиям, принимать решения на основе множества факторов и выполнять итерационные операции на разных уровнях глубины.

Однако важно разумно использовать вложенные управляющие структуры и поддерживать удобочитаемость кода. По мере увеличения глубины вложенности код становится все труднее понимать и отлаживать. Поэтому крайне важно найти баланс между использованием вложенных управляющих структур для решения сложных сценариев и сохранением понятности кода в сопровождении.

Давай теперь погрузимся во вложенные структуры управления Python. Покажи мне, как использовать вложенные условные операторы, допускающие несколько уровней ветвления, вложенные циклы, включая вложенные двойные циклы `for`, двойные циклы `while` и смешанные циклы `while/for` или `for/while`, а также комбинации циклов внутри условных операторов и условных циклов внутри. Используй примеры из супергероев и комиксов, чтобы твои объяснения были творческими и образными.⁷¹

Конечно! Давайте погрузимся в захватывающий мир супергероев, чтобы изучить концепцию вложенных управляющих структур Python.

1. *Супергерой против суперзлодея.* Вот пример программы, которая просит пользователя выбрать супергероя и суперзлодея, а затем определяет, кто кого побеждает, используя двухуровневые вложенные условные операторы.

В этом примере вложенные условные операторы позволяют обрабатывать различные комбинации выбора супергероев и суперзлодеев. Каждый вложенный блок проверяет выбранного супергероя и суперзлодея и на основе комбинации печатает соответствующее сообщение "who beats who" («кто кого побеждает»). Если выбранная комбинация не соответствует ни одному из вложенных условий, отображается соответствующее сообщение об ошибке.

Давайте рассмотрим код:

```

1 # Get inputs
2 superhero = input("Choose a superhero: ")
3 supervillain = input("Choose a supervillain: ")
4
5 # Check superhero/supervillain combination
6 if superhero == "Superman":
7     if supervillain == "Lex Luthor":
8         print("Superman beats Lex Luthor!")
9     elif supervillain == "Doomsday":
10        print("Doomsday beats Superman!")
11    else:
12        print("Invalid supervillain choice!")
13 elif superhero == "Batman":
14    if supervillain == "Joker":
15        print("Batman beats Joker!")
16    elif supervillain == "Bane":
17        print("Bane beats Batman!")
18    else:
19        print("Invalid supervillain choice!")
20 elif superhero == "Spider-Man":
21    if supervillain == "Green Goblin":
22        print("Spider-Man beats Green Goblin!")
23    elif supervillain == "Venom":
24        print("Venom beats Spider-Man!")
25    else:
26        print("Invalid supervillain choice!")
27 else:
28    print("Invalid superhero choice!")

```

Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/yesc8st4>

Вот несколько примеров результата выполнения этой программы:

```

Choose a superhero: Batman
Choose a supervillain: Joker
Batman beats Joker!

```

```

Choose a superhero: Spider-Man
Choose a supervillain: Venom
Venom beats Spider-Man!

```

2. *Тренировки супергероев.* Теперь давайте рассмотрим вложенные циклы `for`, используя

сценарий тренировок супергероев:

```

1 import time
2 num_sessions = int(input("Enter the number of training sessions: "))
3 session_duration = int(input("Enter the duration of each training session (in
   seconds): "))
4
5 for session in range(1, num_sessions + 1):
6     print(f"\nTraining Session {session}:")
7     for second in range(1, session_duration + 1):
8         print(f"Seconds elapsed: {second}. Training in progress...")
9         time.sleep(1) # Pause for one second
10    print("Session complete. Rest 5 seconds and prepare for the next session...")
11    time.sleep(5) # Pause for five seconds
12
13 print("\nWell done! See you soon for more training!")

```

Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinypurl.com/5yhkw5wd>

В этом примере у нас есть вложенные циклы для моделирования нескольких тренировок с определенной продолжительностью. Внешний цикл повторяется в течение программы тренировок, а внутренний цикл повторяется в течение каждой тренировки. Он выводит номер тренировки и отображает прогресс за каждую секунду. После завершения тренировки он выводит сообщение о завершении тренировки, а затем процесс повторяет для следующей тренировки.

Обратите внимание, что мы импортируем модуль `time`, который предоставляет функцию `sleep()`, используемую для приостановки выполнения программы на указанное количество секунд. Мы используем его во внутреннем цикле, чтобы создать односекундную паузу между каждой минутой тренировки. Вот пример вывода этой программы:

```

Enter the number of training sessions: 2
Enter the duration of each training session (in seconds): 2

Training Session 1:
Seconds elapsed: 1. Training in progress...
Seconds elapsed: 2. Training in progress...
Session complete. Rest 5 seconds and prepare for the next session...

Training Session 2:
Seconds elapsed: 1. Training in progress...
Seconds elapsed: 2. Training in progress...
Session complete. Rest 5 seconds and prepare for the next session...

Well done! See you soon for more training!

```

3. *Приветствуем супергероев:* Вы можете использовать двойные циклы `while` для повторения последовательности действий для последовательности супергероев. Давайте рассмотрим пример, где мы говорим «ура» разным супергероям:

```

1 import random
2 # Outer loop runs until "exit" is entered
3 while True:
4     superhero = input("\nEnter the name of a superhero (or 'exit' to quit): ")
5     if superhero == "exit":
6         break;
7     # Generate a random number of hoorays using randint()
8     hoorays = random.randint(1, 10)
9     # Inner loop runs as long as there are hoorays left
10    while hoorays > 0:
11        print(f"Hip hip hooray, {superhero}!") # Print the cheer message
12        hoorays -= 1 # Decrement the hoorays count
13 print("\nBye!")

```

Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/yc6прых>

В этом коде функция `random.randint()` используется для генерации случайного количества ура для каждого супергероя. Внешний цикл `while` позволяет пользователю вводить имена супергероев до тех пор, пока не будет введено "exit" («выход»), в то время как внутренний цикл `while` контролирует количество ура, которое должно быть напечатано для каждого супергероя. Запустив эту программу, вы можете повеселиться, боля за разных супергероев со случайным количеством ура каждый раз. Обратите внимание, что мы не знаем заранее количество повторений в обоих циклах. Вот пример результата выполнения этой

программы:

```
Enter the name of a superhero (or 'exit' to quit): Wonder Woman
Hip hip hooray, Wonder Woman!
Hip hip hooray, Wonder Woman!
Hip hip hooray, Wonder Woman!
Hip hip hooray, Wonder Woman!
Hip hip hooray, Wonder Woman!
Hip hip hooray, Wonder Woman!
Hip hip hooray, Wonder Woman!


Enter the name of a superhero (or 'exit' to quit): Superman
Hip hip hooray, Superman!
Hip hip hooray, Superman!

Enter the name of a superhero (or 'exit' to quit): exit

Bye!
```

4. *Королевская битва*. Сочетание цикла и условных структур позволяет программировать более сложную логику, например королевскую битву супергероев. В программе *Battle Royale* пользователь вводит двух героев, выбранных из Железного человека, Капитана Америки или Черной Вдовы. Программа присваивает каждому герою случайные уровни силы от 1 до 100 и имитирует битву, в которой герои по очереди бьются друг с другом, уменьшая уровень силы с каждым ударом на 10. Битва продолжается до тех пор, пока уровень силы одного из героев не достигнет или не упадет ниже 0. Программа позволяет пользователю повторить битву, если он хочет продолжить:

```
1 import random
2
3 heroes = "iron-man/captain america/black widow"
4 while True:
5     hero1 = input("\nEnter the name of the first hero (or 'exit' to quit): ")
6     if hero1 == "exit":
7         break
8
9     hero2 = input("Enter the name of the second hero (or 'exit' to quit): ")
10    if hero2 == "exit":
11        break
12
13    if hero1.lower() not in heroes or hero2.lower() not in heroes:
14        print('Invalid hero names. Only "Iron-man", "Captain America" or "Black
15        Widow" are allowed. Please try again.')
16        continue
17
18    power1 = random.randint(1, 100)
19    power2 = random.randint(1, 100)
20
21    print(f"\nLet the battle begin between {hero1} (power level: {power1}) and {
22    hero2} (power level: {power2})!")
23
24    while power1 > 0 and power2 > 0:
25        hit1 = random.randint(1, 10)
26        hit2 = random.randint(1, 10)
27        power1 -= hit2
28        power2 -= hit1
29
30        print(f"\n{hero1} hits {hero2} with power {hit1}! {hero2}'s power level: {
31        power2}")
32        print(f"{hero2} counters and hits {hero1} with power {hit2}! {hero1}'s
33        power level: {power1}")
34
35    if power1 <= 0 and power2 <= 0:
36        print("\nIt's a draw! Both heroes are knocked out.")
37    elif power1 <= 0:
38        print(f"\n{hero2} is the winner!")
39    elif power2 <= 0:
40        print(f"\n{hero1} is the winner!")
41
42    print("\nThank you for playing Superhero Battle Royale. See you next time!")
```



Визуализируйте пошаговое выполнение этого кода в PythonTutor: <https://tinyurl.com/2p8z9уар>

Обратите внимание на то, как в этом коде вложенные циклы используются для управления ходом битвы и повторениями между битвами. Внешний цикл позволяет пользователю продолжить выполнение программы, либо выйти из нее, в то время как внутренний цикл имитирует саму битву, в которой герои сменяют друг друга и снижают уровни силы друг друга. Вложенные условные операторы используются для проверки ввода пользователя, определения победителя/проигравшего в битве и управления завершением программы. Они обеспечивают завершение программы, когда пользователь вводит "exit" («выход»), и определяют логику вычисления исхода битвы. Вот пример результата выполнения этой программы:

```
Enter the name of the first hero (or 'exit' to quit): Black Widow
Enter the name of the second hero (or 'exit' to quit): Captain America

Let the battle begin between Black Widow (power level: 95) and Captain America
(power level: 72)!

Black Widow hits Captain America with power 9! Captain America's power level:
63
Captain America counters and hits Black Widow with power 8! Black Widow's
power level: 87

Black Widow hits Captain America with power 5! Captain America's power level:
58
Captain America counters and hits Black Widow with power 8! Black Widow's
power level: 79

Black Widow hits Captain America with power 8! Captain America's power level:
50
Captain America counters and hits Black Widow with power 7! Black Widow's
power level: 72

Black Widow hits Captain America with power 10! Captain America's power level:
```

40

Captain America counters and hits Black Widow with power 7! Black Widow's power level: 65

Black Widow hits Captain America with power 6! Captain America's power level: 34

Captain America counters and hits Black Widow with power 5! Black Widow's power level: 60

Black Widow hits Captain America with power 4! Captain America's power level: 30

Captain America counters and hits Black Widow with power 5! Black Widow's power level: 55

Black Widow hits Captain America with power 3! Captain America's power level: 27

Captain America counters and hits Black Widow with power 9! Black Widow's power level: 46

Black Widow hits Captain America with power 9! Captain America's power level: 18

Captain America counters and hits Black Widow with power 3! Black Widow's power level: 43

Black Widow hits Captain America with power 8! Captain America's power level: 10

Captain America counters and hits Black Widow with power 5! Black Widow's power level: 38

Black Widow hits Captain America with power 7! Captain America's power level: 3

Captain America counters and hits Black Widow with power 10! Black Widow's power level: 28

Black Widow hits Captain America with power 3! Captain America's power level: 0

Captain America counters and hits Black Widow with power 10! Black Widow's power level: 18

Black Widow is the winner!

Enter the name of the first hero (or 'exit' to quit): exit

Thank you for playing Superhero Battle Royale. See you next time!

4.6 Время поработать руками!

Предложи забавные и увлекательные упражнения по программированию для отработки операторов `if/else`, `if/elif/else` и операторов `match/case`, а также тернарного оператора, включая описания проблем и примеры ожидаемого результата.⁷²

1. **Переводчик смайликов:** Напишите программу, которая переводит заданный символ смайлика в соответствующее ему значение. Используйте оператор `match/case`, чтобы определять значения.

Пример:

```
Input: ":)"
Output: "Smile"
```

2. **Калькулятор оценок:** Создайте программу, которая вычисляет и отображает буквенную оценку на основе заданной числовой оценки. Используйте операторы `if/elif/else`, чтобы присвоить соответствующую буквенную оценку.

Пример:

```
Input: 85
Output: B
```

3. **Заказ пиццы:** Напишите программу, которая принимает пользовательский ввод для выбора размера пиццы и начинки и вычисляет ее общую стоимость. Используйте операторы `if/elif/else` и оператор `in` для определения стоимости в зависимости от размера и начинки.

Пример:

```
Input: Size - Medium, Toppings - Pepperoni, Mushrooms
Output: Total cost: $12.50
```

4. **Игра «Камень, ножницы, бумага»:** Реализуйте простую текстовую игру «камень, ножницы, бумага». Используйте операторы `match/case`, чтобы определить победителя на основе выбора игрока.

Пример:

```
Input: Player 1 - Rock, Player 2 - Scissors
Output: Player 1 wins!
```

5. **Консультант по одежде в зависимости от погоды:** Создайте программу, которая советует пользователю, какую одежду надеть в зависимости от погодных условий.

Используйте операторы `match/case`, чтобы предоставить соответствующие рекомендации по одежде.

Пример:

```
Input: Weather - Sunny, Temperature - 28 C
Output: It's sunny! Wear a T-shirt and shorts.
```

6. **Рекомендация по музыке для возрастных групп:** Напишите программу, которая классифицирует людей по разным возрастным группам в зависимости от их возраста и рекомендует подходящие музыкальные группы. Используйте операторы `if/elif/else` для категоризации возрастной группы.

Пример:

```
Input: Age - 45
Output: Listen to music from: Nirvana, R.E.M
```

7. **Сравнение возрастов:** Создайте программу, которая сравнивает три возраста, чтобы определить, кто самый старший, а кто самый младший. Используйте операторы `if/elif/else` для сравнения.

Пример:

```
Input: Ages - 10, 5, 8
Output: Oldest: 10, Youngest: 5
```

В этих упражнениях представлены практические сценарии, в которых могут применяться операторы `if/else`, `if/elif/else` и `match/case`, а также тернарный оператор. Они помогают укрепить понимание в программировании условных операторов и принятия решений.

Предложи забавные и увлекательные упражнения по программированию для отработки циклов `for` и `while`, включая описания проблем и примеры ожидаемого результата.⁷³

1. **Таймер обратного отсчета:** Создайте программу, которая принимает введенное число и ведет обратный отсчет от этого числа до единицы, печатая каждое число в новой строке с паузой в одну секунду между каждым числом.

Пример:

```

Input: 5
Output:
5 (wait a second...)
4 (wait a second...)
3 (wait a second...)
2 (wait a second...)
1 (wait a second...)
Done!

```

2. **Таблица повторений:** Создайте программу, которая генерирует таблицу повторений для заданного числа. Программа должна отображать таблицу от 1 до 10.

Пример:

```

Input: 7
Output:
7 x 1 = 7
7 x 2 = 77
7 x 3 = 777
...
7 x 10 = 77777777

```

3. **Калькулятор факториала:** Напишите программу, вычисляющую факториал заданного числа с помощью цикла `while`. Факториал числа — это произведение всех положительных целых чисел, меньших или равных этому числу. Программа должна отображать всю последовательность умножения.

Пример:

```

Input: 5
Output: 5! = 5 x 4 x 3 x 2 x 1 = 120

```

4. **Игра «Угадай число»:** Создайте программу, генерирующую случайное число от 1 до 100. Пользователь должен угадать число, а программа должна обеспечивать обратную связь (больше или меньше) до тех пор, пока не будет угадан правильный номер. Используйте вложенный оператор `while/if` и отобразите сообщение с количеством использованных попыток.

Пример:

```

Output:
Guess a number between 1 and 100: 50
Too high!
Guess a number between 1 and 100: 25
Too low!
Guess a number between 1 and 100: 40
Congratulations! You guessed the number in 3 attempts.

```

5. **Сумма цифр:** Напишите программу, которая вычисляет сумму цифр заданного числа, используя цикл `while`.

Пример:

```

Input: 1234
Output: 10 (since 1 + 2 + 3 + 4 = 10)

```

6. **Камень, ножницы, бумага против компьютера (2 из 3):** Реализуйте 2 из 3 текстовых игр «камень, ножницы, бумага» (точнее, сначала выиграть два броска, так как ничьих может быть неограниченное количество). Используйте вложенные операторы `match/case` в цикле с оператором `break`, чтобы определить победителя на основе выбора игрока в общем пуле.

Пример:

```

Output:
Make your choice: Rock.
Computer plays: Scissors
Result: Player 1 wins!

Make your choice: Scissors.
Computer plays: Scissors
Result: It's a tie!

Make your choice: Paper.
Computer plays: Rock
Result: Player 1 wins!

Winner: Player 1 is the overall winner!

```

7. **Угадай возраст пользователя:** Создайте программу, которая угадывает возраст пользователя, задавая серию вопросов «да/нет». Программа должна сужать возможный возрастной диапазон с каждым вопросом, пока не будет достигнут

правильный возраст. Программа должна делать обоснованные предположения на основе ответов пользователя на вопросы.

Пример:

```
Output:
Are you older than 50 (yes/no): no
Are you older than 25 (yes/no): yes
Are you older than 37 (yes/no): no
Are you older than 31 (yes/no): yes
Are you older than 34 (yes/no): no
Are you older than 32 (yes/no): yes
Are you older than 33 (yes/no): yes

I know! You are 34 years old!
```

Предложи несколько забавных и увлекательных упражнений по кодированию, чтобы попрактиковаться в структурах вложенных циклов, включая описания проблем и примеры ожидаемых результатов.⁷⁴

1. **Простые числа:** Напишите программу, которая генерирует простые числа до заданного предела, используя вложенный цикл `while/for`. Простое число — это число, которое делится только на 1 и само на себя.

Пример:

```
Input: 20
Output: 2, 3, 5, 7, 11, 13, 17, 19
```

2. **Печать шаблонов:** Создайте программу, которая печатает различные шаблоны, используя вложенные циклы. Например, вы можете напечатать треугольник, квадрат или даже ромб. Используйте вложенные циклы для управления количеством строк и столбцов.

Пример:

Output:

Triangle Pattern:

```
*
* *
* * *
* * * *
* * * * *
```

Square Pattern:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Diamond Pattern:

```
  *
 * *
* * * *
* * * * *
* * * * * *
* * * * *
 * * * *
  * *
   *
```

3. **Матрица таблицы умножения:** Напишите программу, которая генерирует матрицу таблицы умножения от 1 до 10. Используйте вложенные циклы для перебора строк и столбцов и вычисления произведения.

Пример:

Output:

Times Table Matrix:

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
...
10 20 30 40 50 60 70 80 90 100
```

4. **Калькулятор оценок:** Создайте программу, которая вычисляет среднюю оценку класса. Она должна запрашивать у пользователя количество учеников и количество предметов. Используйте вложенные циклы для ввода и вычисления оценок, а затем

отображайте среднюю оценку.

Пример:

```
Output:
Enter the number of students: 3
Enter the number of subjects: 4

Enter grades for student 1:
Subject 1: 85
Subject 2: 90
Subject 3: 80
Subject 4: 95

Enter grades for student 2:
Subject 1: 75
Subject 2: 70
Subject 3: 85
Subject 4: 80

Enter grades for student 3:
Subject 1: 95
Subject 2: 90
Subject 3: 92
Subject 4: 88

Average Grade: 85.83
```

5. **Гонка букв:** Создайте программу, которая имитирует гонку между двумя заданными пользователем буквами в течение 5 последовательных ходов. Каждый участник может сместить свою позицию вправо не более чем на 5 случайных шагов за каждый ход. Программа должна отображать позиции букв после каждого хода и определять победителя в конце последнего хода с помощью тернарного оператора. Вы можете использовать вложенные циклы для управления ходами гонки и шагами участников соответственно.

Пример:

```

Input: A, Z
Output:
[Move 1]
..A
....Z

[Move 2]
.....A
.....Z

[Move 3]
.....A
.....Z

[Move 4]
.....A
.....Z

[Move 5]
.....A
.....Z

It's a tie!

```

6. **Игра FizzBuzz:** Напишите программу, которая печатает числа от 1 до 100. Если число кратно 3, вместо числа выводите слово «Fizz». Если число кратно 5, выводите слово «Buzz». Для чисел, кратных как 3, так и 5, выводите «FizzBuzz». Используйте вложенные структуры `for/if/else` для проверки условий.

Пример:

Output :

```

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz

```

Предложи задачу написания консольной программы-калькулятора, которая берет два числа и выбранную операцию из меню (обозначенного рамкой), вычисляет и отображает результат, а также спрашивает пользователя, хочет ли он продолжить. Включи пример ожидаемого результата.⁷⁵

Консольный научный калькулятор:

Напишите консольную программу-калькулятор, позволяющую пользователю выполнять арифметические операции. Калькулятор должен предоставлять меню с выбором следующих операций:

1. Addition (+): Сложение двух чисел.
2. Subtraction (-): Вычитание второго числа из первого.
3. Multiplication (*): Перемножение двух чисел.
4. Division (/): Деление первого числа на второе.
5. Exponentiation (~): Возведение первого числа в степень второго числа.
6. Square Root (sqrt): Вычисление квадратного корня числа.
7. Logarithm (log): Вычисление логарифма числа по основанию 10.

Пример вывода:

```

*****
*           CALCULATOR MENU           *
*****
* 1. Addition                          *
* 2. Subtraction                       *
* 3. Multiplication                    *
* 4. Division                          *
* 5. Exponentiation                   *
* 6. Square Root                      *
* 7. Logarithm                        *
* 8. Exit                              *
*****

Select an operation (1-8): 3

Enter the first number: 5
Enter the second number: 7

Result: 5 * 7 = 35

*****
*           CALCULATOR MENU           *
*****
* 1. Addition                          *
* 2. Subtraction                       *
* 3. Multiplication                    *
* 4. Division                          *
* 5. Exponentiation                   *
* 6. Square Root                      *
* 7. Logarithm                        *
* 8. Exit                              *
*****

Select an operation (1-8): 8

Bye!

```

Реализация этой задачи позволит вам применить свои навыки обработки пользовательского ввода, реализации систем на основе меню, использования математических функций из модуля `math` и реализации вложенных управляющих структур. Получайте удовольствие от создания своего научного калькулятора!

Зона контейнеров

Объясни, что такое структура контейнера, затем перечисли все структуры контейнеров, доступные в Python, и дай краткое описание каждой из них, используя аналогии из реальной жизни и дружелюбный тон.⁷⁶

Структура контейнера в Python — это структура данных, способная хранить несколько значений или объектов. Это позволяет структурировать и хранить данные. Каждая контейнерная конструкция имеет свои уникальные характеристики и подходит для разных целей.

1. **Списки:** Представьте себе сумку для покупок, куда вы можете положить различные предметы. Эти предметы представляют собой упорядоченный набор элементов, которые могут быть разных типов.
2. **Кортежи:** Представьте себе запечатанный конверт, содержащий несколько частей информации. Кортеж похож на список, но он неизменяем, то есть его нельзя изменить после создания.
3. **Наборы:** Представьте себе ящик, наполненный уникальными предметами. Набор — это неупорядоченный набор уникальных элементов.
4. **Строки:** Представьте себе строку как предложение или последовательность символов. Это неизменяемый контейнер, содержащий набор символов.
5. **Словари:** Представьте себе реальный словарь, в котором хранятся слова и их определения. Словарь представляет собой структуру данных типа пары ключ-значение.

Эти контейнерные структуры обеспечивают гибкость в организации данных и управлении ими в соответствии с вашими конкретными потребностями. Выбрав правильную структуру контейнера, вы сможете эффективно хранить и обрабатывать данные в своих программах, написанных на Python.

Итак, давайте подробнее рассмотрим каждый из них.

5.1 Списки

Объясни все, что мне нужно знать о списках в Python, весело и творчески, используя забавные примеры.⁷⁷

Конечно! Давайте отправимся в веселое приключение в мир списков Python! Представьте, что вы волшебник с волшебным рюкзаком, в котором можно хранить всевозможные

фантастические предметы. В Python списки похожи на ваш волшебный рюкзак, где вы можете хранить, добавлять, удалять и извлекать элементы по своему усмотрению!

1. **Создание списков:** Чтобы создать список, представьте его как свой волшебный рюкзак. Вы можете поместить внутрь все, что захотите, например, блестящие драгоценные камни, волшебные зелья или заколдованные свитки. Просто используйте квадратные скобки [] и разделяйте элементы запятыми.
2. **Индексация:** У каждого предмета в вашем волшебном рюкзаке есть специальный номер, называемый индексом. Это как ярлык, который помогает вам быстро находить нужные вещи. Помните, волшебники начинают отсчет с 0! Итак, первый элемент имеет индекс 0, второй элемент — индекс 1 и так далее. Вы можете получить доступ к элементам в своем списке, используя эти индексы.
3. **Обновление и добавление предметов:** Ваш волшебный рюкзак универсален! Вы можете добавлять новые элементы или изменять существующие в любое время. Только представьте, что во время приключений вы находите редкий предмет и заменяете его еще более редким. Вы можете использовать индекс для обновления элементов в вашем списке.
4. **Получение срезов:** Иногда хочется похвастаться своей коллекцией волшебных предметов перед другими волшебниками. Получение срезов позволяет вам взять подмножество вашего списка и с гордостью показать его. Это аналогично тому, как показать им ваш необычный инвентарь.
5. **Длина и принадлежность:** Ваш волшебный рюкзак может больше! Он может сказать вам, сколько предметов у вас есть, используя функцию `len()`. Вы также можете спросить, есть ли конкретный элемент в вашем списке, используя ключевое слово `in`. Это похоже на проверку того, есть ли у вас конкретный магический артефакт.
6. **Операции со списками:** Вы можете выполнять магические операции со своим списком! Хотите объединить два списка? Просто используйте оператор `+`, чтобы объединить их в мощный суперсписок. Вы даже можете умножить свой список, чтобы создать несколько копий его содержимого. Это похоже на умножение ваших магических предметов, чтобы создать армию артефактов!
7. **Методы списков:** В вашем волшебном рюкзаке есть удобные заклинания, известные как методы. Эти методы могут помочь вам отсортировать элементы, добавить новые, удалить ненужные или даже изменить порядок списка. Они похожи на особые полномочия, которые облегчают управление вашим списком.

Итак, возьмите свой волшебный рюкзак (то есть список Python) и начните наполнять его всевозможными чудесами! Помните, чем больше творчества вы проявите, тем больше удовольствия получите!

Хорошо. Покажи мне пример кода, иллюстрирующий каждый из пунктов, перечисленных выше. Включи комментарии в каждый блок кода, чтобы объяснить его назначение. Ты можешь использовать тему супергероя и злодея.⁷⁸

Конечно! Вот примеры кода, иллюстрирующие каждый из перечисленных выше элементов, с использованием темы супергероя и злодея. Каждый блок кода сопровождается комментариями, поясняющими соответствующий элемент, а также выводом на консоли.

```
1 # Creating Lists
2 superheroes = [] # Start with an empty list
3 print("Your magical backpack of superheroes is ready!")
4 print(f"List of superheroes: {superheroes}")
```

```
Your magical backpack of superheroes is ready!
List of superheroes: []
```

```
1 # Indexing
2 print("\nIndexing Example:")
3 print(f"The first superhero is: {superheroes[0]}")
4 print(f"The second superhero is: {superheroes[1]}")
5 print(f"The last superhero is: {superheroes[-1]}")
```

```
Indexing Example:
The first superhero is: Spider-Man
The second superhero is: Iron Man
The last superhero is: Captain America
```

```
1 # Updating and Adding Items
2 print("\nUpdating and Adding Items Example:")
3 superheroes[1] = "Black Widow" # Update an item at index 1
4 print("Updated list of superheroes:", superheroes)
5
6 superheroes.append("Hulk") # Add a new item to the end of the list
7 print("Extended list of superheroes:", superheroes)
```

```
Updating and Adding Items Example:
Updated list of superheroes: ['Spider-Man', 'Black Widow', 'Captain America']
Extended list of superheroes: ['Spider-Man', 'Black Widow', 'Captain America',
'Hulk']
```

```
1 # Slicing
2 print("\nSlicing Example:")
3 villains = ["Green Goblin", "Loki", "Red Skull", "Thanos", "Venom"]
4 print("List of villains: ", villains)
5 print("The first three villains are:", villains[:3])
6 print("The last two villains are:", villains[-2:])
```

Slicing Example:

List of villains: ['Green Goblin', 'Loki', 'Red Skull', 'Thanos', 'Venom']

The first three villains are: ['Green Goblin', 'Loki', 'Red Skull']

The last two villains are: ['Thanos', 'Venom']

```

1 # Length and Membership
2 print("\nLength and Membership Example:")
3 print("Number of superheroes:", len(superheroes))
4 print("Number of villains:", len(villains))
5
6 hero = input("Enter a superhero name to check if it's in the list: ")
7
8 if hero in superheroes:
9     print(f"Yes, {hero} is in your magical backpack!")
10 else:
11     print(f"Sorry, {hero} is not in your magical backpack.")

```

Length and Membership Example:

Number of superheroes: 4

Number of villains: 5

Enter a superhero name to check if it's in the list: Superman

Sorry, Superman is not in your magical backpack.

```

1 # List Operations
2 print("\nList Operations Example:")
3 all_characters = superheroes + villains # Merge two lists
4 print("All characters in the superhero universe:", all_characters)
5
6 duplicate_villains = villains * 2 # Create multiple copies of the list
7 print("Duplicate villains:", duplicate_villains)

```

List Operations Example:

All characters in the superhero universe: ['Spider-Man', 'Black Widow', 'Captain America', 'Hulk', 'Green Goblin', 'Loki', 'Red Skull', 'Thanos', 'Venom']

Duplicate villains: ['Green Goblin', 'Loki', 'Red Skull', 'Thanos', 'Venom', 'Green Goblin', 'Loki', 'Red Skull', 'Thanos', 'Venom']

```

1 # List Methods
2 print("\nList Methods Example:")
3 print("Original list of superheroes:", superheroes)
4 superheroes.sort() # Sort the list in alphabetical order
5 print("Sorted list of superheroes:", superheroes)
6
7 superheroes.remove("Captain America") # Remove a specific item from the list
8 print("Updated list of superheroes:", superheroes)
9 superheroes.reverse() # Reverse the order of the list
10 print("Reversed list of superheroes:", superheroes)

```

List Methods Example:

Original list of superheroes: ['Spider-Man', 'Black Widow', 'Captain America', 'Hulk']

Sorted list of superheroes: ['Black Widow', 'Captain America', 'Hulk', 'Spider-Man']

Updated list of superheroes: ['Black Widow', 'Hulk', 'Spider-Man']

Reversed list of superheroes: ['Spider-Man', 'Hulk', 'Black Widow']

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/2s4bz4sy>

Интересно. Теперь продемонстрирую, как можно использовать циклы и условные операторы в списках. На этот раз используй творческие примеры, связанные с едой, используя возможности рандомности в Python.⁷⁹

Конечно! Вот пример того, как вы можете использовать циклы и условные операторы со списками, используя забавную тему, связанную с едой, которая включает в себя функции рандомности в Python. Код демонстрирует различные способы перебора списка продуктов питания и применения условных операторов для различных сценариев, связанных с продуктами питания. Наслаждайтесь вкусными примерами!

```

1 import random
2 foods = ["pizza", "burger", "sushi", "taco", "pasta"]
3 # Looping over a list
4 print("=== Menu ===")
5 for food in foods:
6     print(food)

```

=== Menu ===

pizza

burger

sushi

taco

pasta

```

1 # Looping with conditionals
2 print("\n=== Vegetarian Options ===")
3 for food in foods:
4     if food != "burger":
5         print(food)

```

```

=== Vegetarian Options ===
pizza
sushi
taco
pasta

```

```

1 # Randomly selecting a food
2 print("\n=== Random Food ===")
3 random_food = random.choice(foods)
4 print("Enjoy your", random_food)

```

```

=== Random Food ===
Enjoy your sushi

```

```

1 # Looping with a random twist
2 print("\n=== Food Roulette ===")
3 for i in range(3):
4     random_food = random.choice(foods)
5     if random_food == "sushi":
6         print(f"Round {i+1}: It's your lucky day! You got sushi!")
7     else:
8         print(f"Round {i+1}: Better luck next time. You got {random_food}")

```

```

=== Food Roulette ===
Round 1: Better luck next time. You got taco
Round 2: It's your lucky day! You got sushi!
Round 3: Better luck next time. You got pasta

```

```

1 # Random prices with list comprehensions
2 prices = [random.randint(2, 20) for _ in range(len(food))]
3 print("\n=== Price list ===")
4 for food, price in zip(foods, prices):
5     print(f"{food}\t${price}")

```



```

=== Price list ===
pizza $7
burger $15
sushi $5
taco $10
pasta $14

```

```

1 # Nested Lists and Nested Loops
2 deals = [{"sandwich", "crisps", "juice"}, {"burger", "fries", "soda"}, {"taco", "fruit",
3         "beer"}]
4 print("\n=== Meal deals ===")
5 for deal in deals:
6     print(f"\nItems in the {deal[0].upper()} meal deal:")
7     for i, item in enumerate(deal):
8         print(f"Item {i+1}: {item}")

```

```

=== Meal deals ===

Items in the SANDWICH meal deal:
Item 1: sandwich
Item 2: crisps
Item 3: juice

Items in the BURGER meal deal:
Item 1: burger
Item 2: fries
Item 3: soda

Items in the TACO meal deal:
Item 1: taco
Item 2: fruit
Item 3: beer

```

```

1 # List Methods and Operations
2 beverages = ["soda", "juice", "beer"]
3
4 print("\n=== Beverages ===")
5 print("Original list of beverages:", beverages)
6 beverages.insert(1, "water")
7 print("Updated list of beverages:", beverages)

```

```

8
9 all_together = foods + beverages
10 print("\n=== All together ===")
11 print("Combined list of foods and beverages:", all_together)

```

```

=== Beverages ===
Original list of beverages: ['soda', 'juice', 'beer']
Updated list of beverages: ['soda', 'water', 'juice', 'beer']

=== All together ===
Combined list of foods and beverages: ['pizza', 'burger', 'sushi', 'taco', 'pasta',
'soda', 'water', 'juice', 'beer']

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/mumh24r8>

Подожди минуту. В приведенных выше примерах ты используешь некоторые изящные приемы, о которых я не знаю, такие как: `random.choice()`, `zip()`, `enumerate()` и `insert()`. Можешь ли ты объяснить мне это?⁸⁰

Конечно! Давайте рассмотрим каждый из этих изящных приемов, использованных в примерах кода.

► `random.choice()`:

- Это функция из модуля `random` стандартной библиотеки Python.
- Она позволит вам случайным образом выбирать элемент из заданного списка.
- В примере `random.choice(foods)` выбирает один случайный продукт из списка `foods`.

► `zip()`:

- `zip()` — еще одна встроенная в Python функция, которая объединяет несколько итерируемых объектов (списки, кортежи и т. д.) в один итерируемый объект.
- Она возвращает пары соответствующих элементов из входных итерационных данных.
- В примере `for food, price in zip(foods, prices)` функция `zip()` используется для одновременного перебора двух списков, `foods` и `prices`, а переменные `food` и `price` содержат соответствующие элементы из каждого списка.

► `enumerate()`:

- `enumerate()` — это встроенная в Python функция, которая позволяет выполнять итерацию по списку (или любому итерируемому объекту), а также отслеживать индекс каждого элемента.
- Возвращает пары кортежей индекс-элемент.
- В примере `for i, item in enumerate(deal)` функция `enumerate()` используется для перебора каждого из подсписков расклада, а переменная `i` содержит индекс каждого элемента в раскладе.

► `insert()`:

- Метод `insert()` — это встроенная операция со списком в Python, которая позволяет вам вставлять элемент по определенному индексу в списке.
- Он принимает два аргумента: индекс, по которому следует вставить элемент, и сам элемент.
- Существующий элемент с указанным индексом и после него сдвигаются вправо. Метод `insert()` изменяет исходный список и не создает новый список.

Эти мощные приемы широко используются в Python для улучшения функциональности и повышения читабельности кода. Они могут быть очень удобны в различных сценариях, таких как создание случайных выборок, перебор списков с индексами или одновременный перебор нескольких списков.

5.2 Кортежи

Объясни все, что мне нужно знать о кортежах в Python, весело и творчески, используя забавные примеры.⁸¹

Конечно! Давайте погрузимся в увлекательный мир кортежей Python. Представьте, что вы устраиваете грандиозный званый обед для своих друзей и хотите следить за рассадкой. Кортежи могут помочь вам в этом! Представьте себе кортеж как особую сервировку стола, где каждому гостю отводится определенное место. Вот что вам нужно знать о кортежах:

1. **Неизменяемый состав:** После того, как вы установили расположение мест с помощью кортежа, оно остается фиксированным. Вы не можете произвести замену или поменять местами без создания нового кортежа. Как и на изысканном ужине, когда гости рассядутся, вы не сожете их перемещать.
2. **Упорядоченное размещение:** Кортежи сохраняют порядок элементов. Точно так же, как вы тщательно планируете размещение своих гостей, кортежи обеспечивают неизменность порядка элементов. Первый элемент всегда будет на первом месте, второй элемент на втором месте и так далее.
3. **Смешивание и соответствие:** Кортежи позволяют вам реализовать разнообразную рассадку. Каждое место может содержать разные типы данных, такие как строки, числа или даже другие кортежи. Вы можете смешивать и устанавливать соответствие, чтобы создавать сложные схемы рассадки. Это как устроить вечеринку, на которой собираются люди с разным опытом и интересами.
4. **Эффективность и легкость:** Кортежи являются легкими и эффективными. Они занимают меньше памяти по сравнению с другими структурами данных. Это как элегантные раскладочные карточки, которые не занимают много места на столе.
5. **Трюк с распаковкой:** Одним из полезных аспектов кортежей является распаковка кортежей. Вы можете легко присвоить элементы кортежа отдельным переменным в

одной строке кода. Это похоже на обнаружение рассадки путем раздачи карточек каждому гостю.

Помните, что кортежи лучше всего подходят для ситуаций, когда вам нужна упорядоченная коллекция элементов, которая не должна изменяться. Будь то рассадка, координаты на карте или любой другой сценарий, где важны неизменность и порядок, кортежи помогут вам!

Хорошо. Давай рассмотрим некоторые примеры кода, чтобы увидеть кортежи в действии. На этот раз, почему бы не использовать рок-группы для иллюстрации элементов, которые мы перечислили ранее?⁸²

Конечно! Давайте рассмотрим несколько примеров кода с использованием рок-групп, чтобы проиллюстрировать различные аспекты кортежей. Мы изучим каждый элемент, упомянутый ранее, в увлекательной и творческой форме.

1. **Неизменяемый состав:** Скажем, вы организуете рок-концерт, вы можете использовать кортеж, чтобы определить состав групп. После того, как состав определен, он остается неизменным на протяжении всего концерта. Давайте рассмотрим пример:

```
1 lineup = ("The Beatles", "Led Zeppelin", "AC/DC", "Guns N' Roses", "Queen")
2 print(lineup)
3
4 # Uncommenting the line below will raise an error
5 # lineup[0] = "Coldplay"
```

После того, как с помощью кортежа было установлено расположение, его нельзя изменить. Попытка изменить элемент кортежа приведет к ошибке, поскольку кортежи неизменяемы, как и состав группы после начала концерта.

```
=== Immutable Lineup ===
('The Beatles', 'Led Zeppelin', 'AC/DC', 'Guns N' Roses', 'Queen')
```

2. **Порядок размещения:** При планировании концерта вы хотите, чтобы группы выступали в определенном порядке. Кортежи сохраняют порядок элементов. Посмотрим, как это выглядит:

```
1 lineup = ("The Beatles", "Led Zeppelin", "AC/DC", "Guns N' Roses", "Queen")
2
3 for band in lineup:
4     print("Next band:", band)
```

В этом примере группы выступают в порядке, определенном кортежем. Цикл

повторяется по последовательно, и каждая группа печатается в корректном порядке.

```
=== Ordered Placement ===  
Next band: The Beatles  
Next band: Led Zeppelin  
Next band: AC/DC  
Next band: Guns N' Roses  
Next band: Queen
```

3. **Перемешивание и установление соответствия:** Рок-группы могут иметь разные составы, участники которых играют разные роли. Кортежи позволяют создавать сложные структуры, комбинируя различные типы данных. Давайте рассмотрим это на примере с Битлз:

```
1 john = ("John Lennon", 1940, False)  
2 paul = ("Paul McCartney", 1942, True)  
3 george = ("George Harrison", 1943, False)  
4 ringo = ("Ringo Starr", 1940, True)  
5  
6 beatles = (john, paul, george, ringo)  
7  
8 for member in beatles:  
9     print("\nBeatles member:", member[0])  
10    print("Year of birth:", member[1])  
11    print("Is alive?", member[2])
```

В этом примере каждый член группы «Битлз» представлен кортежем, содержащим имя, год рождения и логическое значение, указывающее, живы они или нет. Кортеж `beatles` объединяет кортежи отдельных участников, чтобы сформировать полную группу.

```

=== Mix and Match ===

Beatles member: John Lennon
Year of birth: 1940
Is alive? True

Beatles member: Paul McCartney
Year of birth: 1942
Is alive? True

Beatles member: George Harrison
Year of birth: 1943
Is alive? False

Beatles member: Ringo Starr
Year of birth: 1940
Is alive? True

```

4. **Эффективность и легкость:** Кортежи являюмя легкими и потребляют меньше памяти по сравнению с другими структурами данных, такими как списки. Давайте рассмотрим пример:

```

1 beatles_list = ["John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr"]
2 beatles_tuple = ("John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr")
3
4 print("Size of the list:", beatles_list.__sizeof__())
5 print("Size of the tuple:", beatles_tuple.__sizeof__())

```

В этом примере мы используем функцию `sizeof__()` для определения использования памяти списком `beaties_iist` и кортежем `beaties_tupte`. Вы заметите, что размер кортежа меньше по сравнению со списком.

```

=== Efficient and Lightweight ===

Size of the list: 72
Size of the tuple: 56

```

5. **Трюк с распаковкой:** Распаковка кортежа позволяет вам назначать элементы кортежа отдельным переменным. Давайте рассмотрим это на примере Битлз:

```

1 band = ("The Beatles", "John Lennon", "Paul McCartney", "George Harrison", "Ringo
   Starr")
2
3 band_name, *members = band
4
5 print("Band name:", band_name)
6 print("Band members:", members)

```

В этом примере мы распаковываем кортеж `band` в отдельные переменные. Первая переменная `band_name` содержит название группы, а переменной `members` назначается список, содержащий отдельных участников группы.

```

=== Unpacking Trick ===
Band name: The Beatles
Band members: ['John Lennon', 'Paul McCartney', 'George Harrison', 'Ringo
               Starr']

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/2mprsujd>

Эти примеры кода демонстрируют различные аспекты кортежей, используя забавные аналогии с рок-группами. Кортежи универсальны и могут использоваться в различных сценариях, помимо рок-групп.

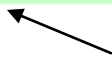
Постой! В примере с трюком с распаковкой перед переменной «members» стоит странная штука. Что это такое?»⁸³

Звездочка (*) перед переменной при распаковке кортежа используется для сбора всех оставшихся элементов кортежа в список. В Python он известен как синтаксис «расширенной итерируемой распаковки». В примере с Beatles звездочка используется для присвоения оставшихся членов кортежа группы переменной `members`. Это обеспечивает гибкость в случае, если количество участников в группе заранее неизвестно.

Однако, если вы знаете количество оставшихся элементов в кортеже, вы можете присвоить их отдельным переменным, как это показано в следующем примере:

```
1 band = ("The Beatles", "John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr")
2
3 band_name, vocalist, guitarist, bassist, drummer = band
4
5 print("Band name:", band_name)
6 print("Vocalist:", vocalist)
7 print("Guitarist:", guitarist)
8 print("Bassist:", bassist)
9 print("Drummer:", drummer)
```

```
Band name: The Beatles
Vocalist: John Lennon
Guitarist: Paul McCartney
Bassist: George Harrison
Drummer: Ringo Starr
```



5.3 Наборы

Объясни все, что мне нужно знать о наборах в Python, весело и творчески, используя забавные примеры.⁸⁴

Конечно! Давайте погрузимся в волшебный мир наборов в Python. Представьте, что у вас есть коллекция уникальных волшебных существ. Каждое существо обладает различными способностями и характеристиками.

- 1. Уникальная коллекция:** Набор похож на особый сундук с сокровищами, в котором можно хранить коллекцию уникальных предметов. Это похоже на волшебный зоопарк, где нет двух одинаковых существ. Каждое существо уникально и отличается от других.
- 2. Отсутствие дубликатов:** Наборы гарантируют, что не будет разрешено дублирование существ. Если вы попытаетесь добавить одно и то же существо дважды, набор волшебным образом сохранит только один его экземпляр. Словно набор обладает магической силой сохранять свою уникальность.
- 3. Быстрое тестирование на членство:** Наборы лучше всего отвечают на такие вопросы, как «Это существо уже есть в коллекции?» Они могут выполнять молниеносные тесты на членство. Это похоже на наличие магического сканера, который быстро определяет, является ли существо частью набора или нет.
- 4. Магические операции:** Наборы также предлагают мощные магические операции, которые вы можете выполнять с ними. Вы можете комбинировать наборы для создания новых наборов, находить общих для наборов существ, либо даже находить уникальных существ, которые существуют только в одном наборе и отсутствуют в другом.
- 5. Изменяемые и неупорядоченные:** Наборы изменяемы, то есть вы можете добавлять в них или удалять из них существ. Однако особого порядка у них нет. Это похоже на волшебную коллекцию, где существа могут свободно перемещаться без какого-либо фиксированного порядка.
- 6. Волшебное удаление:** Вы можете волшебным образом удалять существ из набора в зависимости от их уникальных свойств. Просто произнесите имя существа, которое хотите удалить, и набор сотворит свое волшебство и удалит это существо.
- 7. Объединение усилий:** Наборы позволяют объединять силы, выполняя операции объединения. Вы можете комбинировать несколько наборов вместе, и это создаст грандиозную коллекцию всех уникальных существ из всех наборов. Это подобно слиянию волшебных племен в одно большое сообщество.

Хорошо. Теперь давайте рассмотрим некоторые примеры кода, чтобы увидеть наборы в действии. На этот раз давайте воспользуемся забавными фактами о планетах, чтобы проиллюстрировать пункты, которые вы перечислили ранее.⁸⁵

Конечно! Вот несколько примеров кода, демонстрирующих множество фактов с использованием знания о планетах:

```

1 # Unique Collection
2 print("\n=== Unique Collections ===")
3 planets = set(["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "
    Neptune"])
4 moons = set([0, 0, 1, 2, 79, 82, 27, 14])
5 locations = set(["inner", "inner", "inner", "inner", "outer", "outer", "outer", "outer"])
6 galaxies = set(["Milky Way", "Milky Way", "Milky Way", "Milky Way", "Milky Way", "Milky
    Way", "Milky Way", "Milky Way"])
7
8 print(f"Planets (unique values: {len(planets)}):", planets)
9 print(f"Moons (unique values: {len(moons)}):", moons)
10 print(f"Locations (unique values: {len(locations)}):", locations)
11 print(f"Galaxies (unique values: {len(galaxies)}):", galaxies)

```

В этом примере мы создаем наборы из списка планет и списков, содержащих соответствующее количество лун, местоположений в Солнечной системе и галактик, к которым они принадлежат.

Чтобы создать набор в Python, вы можете использовать фигурные скобки ({}), или функцию `set()`. Просто заключите элементы, которые вы хотите включить в набор, в фигурные скобки или передайте их в качестве аргументов функции `set()`. Наборы — это неупорядоченные наборы уникальных элементов, что означает, что повторяющиеся значения автоматически удаляются. Наборы полезны, когда вы хотите хранить коллекцию элементов без какого-либо определенного порядка и при этом обеспечить уникальность этих элементов.

```

=== Unique Collections ===
Planets (unique values: 8): {'Saturn', 'Mars', 'Mercury', 'Venus', 'Neptune',
    'Jupiter', 'Uranus', 'Earth'}
Moons (unique values: 7): {0, 1, 2, 14, 79, 82, 27}
Locations (unique values: 2): {'inner', 'outer'}
Galaxies (unique values: 1): {'Milky Way'}

```

```

1 # No Duplicates
2 print("\n=== No Duplicates ===")
3 planets = {"Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"}
4 print('Before adding "Earth":', planets)
5 planets.add("Earth") # Adding a duplicate planet will have no effect
6 print('After adding "Earth":', planets)

```

```

=== No Duplicates ===
Before adding "Earth": {'Saturn', 'Mars', 'Mercury', 'Venus', 'Neptune', 'Jupiter',
    'Uranus', 'Earth'}

After adding "Earth": {'Saturn', 'Mars', 'Mercury', 'Venus', 'Neptune', 'Jupiter',
    'Uranus', 'Earth'}

```

```

1 # Fast Membership Testing
2 print("\n=== Fast Membership ===")
3 print("Is Earth in planets? ", "Earth" in planets)
4 print("Is Pluto in planets? ", "Pluto" in planets)

```

```

=== Fast Membership ===
Is Earth in planets? True
Is Pluto in planets? False

```

```

1 # Magical Operations
2 print("\n=== Magical Operations ===")
3 gas_giants = {"Jupiter", "Saturn"}
4 rocky_planets = {"Mercury", "Venus", "Earth", "Mars"}
5 ring_systems = {"Saturn", "Uranus", "Neptune"}
6 common_gas_rocky = gas_giants.intersection(rocky_planets) # Outputs an empty set
7 common_ring_gas = ring_systems.intersection(gas_giants)
8
9 print("Planets that are gas giants: ", gas_giants)
10 print("Planets with rocky surfaces: ", rocky_planets)

```

```

11 print("Planets having ring systems: ", ring_systems)
12 print("Planets with gas and rock in common: ", common_gas_rocky)
13 print("Planets with rings and gas in common: ", common_ring_gas)

```

```

=== Magical Operations ===
Planets that are gas giants: {'Jupiter', 'Saturn'}
Planets with rocky surfaces: {'Earth', 'Mars', 'Mercury', 'Venus'}
Planets having ring systems: {'Neptune', 'Saturn', 'Uranus'}
Planets with gas and rock in common: set()
Planets with rings and gas in common: {'Saturn'}

```

```

1 # Mutable and Unordered
2 print("\n=== Mutable and Unordered ===")
3 print("Before removing 'Neptune':", planets)
4 planets.remove("Neptune")
5 print("After removing 'Neptune':", planets)
6
7 # Magical Removal
8 print("\n=== Magical Removal ===")
9 print("Before discarding 'Pluto':", planets)
10 planets.discard("Pluto") # Removing a planet that doesn't exist has no effect
11 print("After discarding 'Pluto':", planets)

```

Основное различие между методами `remove()` и `discard()` в наборах заключается в том, как они обрабатывают удаление элемента, которого нет в наборе:

- ▶ `remove()` выдает ошибку, если элемент не найден в наборе, и останавливает выполнение программы.
- ▶ `discard()` не вызывает ошибку, если элемент не найден; вместо этого он молча продолжает выполнение программы без внесения каких-либо изменений в набор; таким образом, это позволяет избежать прерывания программы. Рекомендуется использовать `discard()`, если вы не уверены, что элемент находится в наборе.

```

=== Mutable and Unordered ===
Before removing 'Neptune': {'Saturn', 'Mars', 'Mercury', 'Venus', 'Neptune',
    'Jupiter', 'Uranus', 'Earth'}
After removing 'Neptune': {'Saturn', 'Mars', 'Mercury', 'Venus', 'Jupiter',
    'Uranus', 'Earth'}
=== Magical Removal ===
Before discarding 'Pluto': {'Saturn', 'Mars', 'Mercury', 'Venus', 'Jupiter',
    'Uranus', 'Earth'}
After discarding 'Pluto': {'Saturn', 'Mars', 'Mercury', 'Venus', 'Jupiter',
    'Uranus', 'Earth'}

```

```

1 # Joining Forces
2 print("\n=== Joining Forces ===")
3 dwarf_planets = {"Pluto", "Eris", "Haumea"}
4 all_planets = planets.union(dwarf_planets)
5 print("Original planets: ", planets)
6 print("Dwarf planets: ", dwarf_planets)
7 print("All planets: ", all_planets)

```

```

=== Joining Forces ===
Original planets: {'Saturn', 'Mars', 'Mercury', 'Venus', 'Jupiter', 'Uranus',
                  'Earth'}
Dwarf planets: {'Haumea', 'Pluto', 'Eris'}
All planets: {'Saturn', 'Eris', 'Pluto', 'Mars', 'Haumea', 'Mercury', 'Venus',
              'Jupiter', 'Uranus', 'Earth'}

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: https://tinyurl.com/sf_haje

С помощью наборов вы можете создавать уникальные коллекции планет, выполнять такие операции, как проверка принадлежности, поиск общих планет и добавление или удаление планет из набора. Наборы предлагают удобный и эффективный способ работы с коллекциями уникальных элементов, так же как и изучение увлекательного мира планет с их уникальными характеристиками.

5.4 Строки

Объясни все, что мне нужно знать о строках в Python, весело и творчески, используя забавные примеры.⁸⁶

Конечно! Давайте отправимся в увлекательное путешествие по миру строк Python!

Представьте себе строки как волшебную коллекцию букв, символов и слов, которыми можно манипулировать и которые можно преобразовывать для создания всевозможных захватывающих вещей. Они похожи на красочные строительные блоки текста в Python.

- 1. Перемешивание и установление связей:** Строки обладают способностью супер-смешивания! Их можно комбинировать с помощью оператора «+», точно так же, как смешивание зелий для создания новых заклинаний. Например, представьте, что вы объединили строки «Hello» и «World», чтобы создать мощное заклинание «HelloWorld». Ух ты!
- 2. Мощная репликация:** Цепочки также можно умножать, например, копировать заклинание, чтобы сделать его более мощным. Только представьте, что вы умножаете строку "Abracadabra" на 3, и вот, вы получаете «AbracadabraAbracadabraAbracadabra»! Это как произнести заклинание несколько

раз!

3. **Секреты регистра:** Раскройте скрытый потенциал, изменив регистр строк, например, превратив «Hello» в «hELLO» или «hello» в «HELLO». Это похоже на использование секретных кодов для разблокировки различных возможностей.
4. **Поиск персонажа:** «Строки похожи на сундуки с сокровищами, наполненные символами. Другими словами, строка — это последовательность символов, включающая буквы (например, «ABC...Zabc...z»), символы («! @:~/>+...»), или цифры ("0123...9"), заключенные в одинарные или двойные кавычки ('или"). Вы можете получить доступ к отдельным символам в строке с помощью индексации, аналогично обнаружению спрятанных сокровищ на карте. Например, обратившись к третьему символу строки «Python», вы обнаружите букву «t». Это как найти спрятанный драгоценный камень в текстовом сундуке!»
5. **Магия трансформации:** Знаете что? Строки не просто статичны. Их можно преобразовать с помощью методов. Вы можете разбить строку на список слов, заменить определенные слова или символы или даже объединить несколько строк вместе. Это похоже на волшебный набор инструментов для формирования текстовой информации.

Итак, дайте волю своему творчеству! Используйте строки, чтобы создавать причудливые истории, забавные шутки или даже свой собственный секретный язык. Строки в Python — ваши верные спутники и мощные инструменты для работы с текстом!

Хорошо. Теперь давай рассмотрим несколько примеров кода, чтобы увидеть строки в действии! Рассмотрим возможность использования классических детских книг для иллюстрации свойств, которые вы перечислили ранее.⁸⁷

Конечно! Давайте погрузимся в чарующий мир классических детских книг и познаем силу строк. Вот несколько восхитительных примеров кода:

```

1 # Mix and Match (single strings)
2 print("\n=== Mix and Match (single strings) ===")
3 book1 = "Alice in Wonderland"
4 book2 = "Peter Pan"
5 books = book1 + " and " + book2
6 print(f"Book 1: {book1}")
7 print(f"Book 2: {book2}")
8 print(f"{books} together make a terrific reading pack for children!")
9
10 # Mix and Match (list of strings)
11 print("\n=== Mix and Match (list of strings) ===")
12 titles = ["The Fellowship of the Ring", "The Two Towers", "The Return of the King"]
13
14 # Use str.join() instead of "+" to concatenate multiple strings
15 collection = " and ".join(titles)
16 print(f"{collection} are the beloved books of the Lord of the Rings trilogy.")

```

```

=== Mix and Match (single strings) ===
Book 1: Alice in Wonderland
Book 2: Peter Pan
Alice in Wonderland and Peter Pan together make a terrific reading pack for children!

=== Mix and Match (list of strings) ===
The Fellowship of the Ring and The Two Towers and The Return of the King are the
beloved books of the Lord of the Rings trilogy.

```

```

1 # Mighty Replication
2 print("\n=== Mighty Replication ===")
3 phrase = "\tI'll huff and I'll puff and I'll blow your house down\n"
4 wolf_talk = phrase * 3
5 dialogue = "And then,\n" + wolf_talk + "the wolf exclaimed."
6 print(dialogue)

```

```

=== Mighty Replication ===
And then,
    I'll huff and I'll puff and I'll blow your house down
    I'll huff and I'll puff and I'll blow your house down
    I'll huff and I'll puff and I'll blow your house down
the wolf exclaimed.

```

```

1 # Case Secrets
2 print("\n=== Case Secrets ===")
3 book = "Harry Potter and the sorcerer's stone"
4 title_case = book.title()
5 upper_case = book.upper()
6 lower_case = book.lower()
7 swap_case = book.swapcase()
8 print("Original: ", book)
9 print("Title case: ", title_case)
10 print("Upper case: ", upper_case)
11 print("Lower case: ", lower_case)
12 print("Swap case: ", swap_case)

```

```
=== Case Secrets ===
```

```
Original:   Harry Potter and the sorcerer's stone
Title case: Harry Potter And The Sorcerer'S Stone
Upper case: HARRY POTTER AND THE SORCERER'S STONE
```

```
Lower case: harry potter and the sorcerer's stone
Swap case:  hARRY pOTTER AND THE SORCERER'S STONE
```

```

1 # Character Quest
2 print("\n=== Character Quest ===")
3 story = "Little Red Riding Hood"
4 print(f"Original text:\t\t {story}")
5 print(f"First character:\t {story[0]}") # Accessing first character with index
6 print(f"First 6 characters:\t {story[:6]}") # First 6 characters using slicing
7 print(f"Every other character:\t {story[::2]}") # Traverse every other character
8 vowels = "aeiou"
9 print(f"Consonants only:\t {''.join([c for c in story if c not in vowels])}")
10 print(f"Vowels found:\t\t {set([c for c in story if c in vowels])}")

```

```
=== Character Quest ===
```

```
Original text:   Little Red Riding Hood
First character: L
First 6 characters: Little
Every other character: Ltl e iigHo
Consonants only: Lttl Rd Rdng Hd
Vowels found:   {'o', 'e', 'i'}
```



```

1 # Transforming Magic
2 print("\n=== Transforming Magic ===")
3 quote = "Oh, mirror on the wall, who's the fairest one of all?"
4 split_words = quote.split()
5 replaced_word = quote.replace("all", "[any]")
6 blanks_removed = quote.replace(" ", "")
7 reversed_string = quote[::-1]
8 reversed_words = split_words[::-1]
9 print("Original quote:\t\t", quote)
10 print("Character length:\t", len(quote))
11 print("Split words:\t\t", split_words)
12 print("Words length:\t\t", len(split_words))
13 print("Replaced word:\t\t", replaced_word)
14 print("Blanks removed:\t\t", blanks_removed)
15 print("Reversed string:\t", reversed_string)
16 print("Reversed words:\t\t", reversed_words)
17 print("Sentence completion:")
18 for i in range(len(split_words)):
19     print("\t" + " ".join(split_words[:i + 1]))

```

```

=== Transforming Magic ===
Original quote:      Oh, mirror on the wall, who's the fairest one of all?
Character length:   53
Split words:        ['Oh,', 'mirror', 'on', 'the', 'wall,', "who's", 'the',
                    'fairest', 'one', 'of', 'all?']
Words length:       11
Replaced word:      Oh, mirror on the w[any], who's the fairest one of [any]?
Blanks removed:     Oh,mirroronthewall,who'sthefairestoneofall?
Reversed string:    ?lla fo eno tseriaf eht s'ohw ,llaw eht no rorrim ,h0
Reversed words:     ['all?', 'of', 'one', 'fairest', 'the', "who's", 'wall,',
                    'the', 'on', 'mirror', 'Oh,']
Sentence completion:
    Oh,
    Oh, mirror
    Oh, mirror on
    Oh, mirror on the
    Oh, mirror on the wall,
    Oh, mirror on the wall, who's
    Oh, mirror on the wall, who's the
    Oh, mirror on the wall, who's the fairest
    Oh, mirror on the wall, who's the fairest one
    Oh, mirror on the wall, who's the fairest one of
    Oh, mirror on the wall, who's the fairest one of all?

```

```
1 # More Tricks
2 print("\n=== More Tricks ===")
3 books = ["Beauty and the Beast", "The Little Prince", "The Wizard of Oz", "1001
4     Nights", "The Jungle Book"]
5 for book in books:
6     # Find
7     print("\nBook title:\t\t", book)
8     print("'Oz' found at position:", book.find("Oz"))
9     # StartsWith and EndsWith
10    print("Starts with 'The':\t", book.startswith("The"))
11    print("Ends with 'Book':\t", book.endswith("Book"))
12    # IsAlpha and IsDigit
13    print("Begins with a word:\t", book[1:4].isalpha())
14    print("Begins with a number:\t", book[1:4].isdigit())
15    # Strip
16    print('Stripped of double quotes:', book.strip('"'))
```

```
=== More Tricks ===
Book title:      "Beauty and the Beast"
'Oz' found at position: -1
Starts with 'The':    False
Ends with 'Book':    False
Begins with a word:   True
Begins with a number: False
Stripped of double quotes: Beauty and the Beast

Book title:      "The Little Prince"
'Oz' found at position: -1
Starts with 'The':   True
Ends with 'Book':    False
Begins with a word:   True
Begins with a number: False
Stripped of double quotes: The Little Prince

Book title:      "The Wizard of Oz"
'Oz' found at position: 15
Starts with 'The':   True
Ends with 'Book':    False
Begins with a word:   True
Begins with a number: False
Stripped of double quotes: The Wizard of Oz

Book title:      "1001 Nights"
'Oz' found at position: -1
Starts with 'The':   False
Ends with 'Book':    False
Begins with a word:   False
Begins with a number: True
Stripped of double quotes: 1001 Nights

Book title:      "The Jungle Book"
'Oz' found at position: -1
Starts with 'The':   True
Ends with 'Book':    True
Begins with a word:   True
Begins with a number: False
Stripped of double quotes: The Jungle Book
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/y55mdy6n>

Приведенный выше код выполняет несколько строковых операций со списком названий книг:

1. Он использует метод `find ()` для определения позиции подстроки `Oz` в названии каждой книги.
2. Метод `startswith ()` проверяет, начинается ли название книги с подстроки `"The`.
3. Метод `endswith ()` проверяет, заканчивается ли название книги подстрокой `Book"`.
4. Нарезая часть названия книги, он определяет, состоит ли она из буквенных символов, используя `isalpha()`, или цифр, используя `isdigit()`.
5. Метод `strip ()` убирает двойные кавычки из названия книги.

Эти операции позволяют получить представление о названиях книг, например найти определенные подстроки, проверить префиксы и суффиксы, изучить состав символов частей и удалить ненужные символы. Итак, вперед, исследуйте другие примеры и наслаждайтесь открытием волшебного и яркого мира строк, используя все их удивительные инструменты!

5.5 Словари

Объясни все, что мне нужно знать о словарях в Python, весело и творчески, используя забавные примеры.⁸⁸

Конечно! Давайте отправимся в увлекательное путешествие в мир словарей Python. Представьте, что у вас есть вселенная супергероев, где каждый герой обладает уникальными способностями, союзниками и заклятыми врагами. Вы хотите создать энциклопедию такой сверхвселенной, запечатлев эту информацию каждого героя. У каждого героя может быть своя страница в этой энциклопедии, уникальная биография, наполненная их характеристиками и связями. Вот тут-то и пригодятся словари!

1. **Пары ключ-значение:** Словари состоят из пар ключ-значение, точно так же, как секретные коды и соответствующие им значения. Личность каждого героя служит ключом, а их свойства, такие как способности, союзники и враги, являются значениями, связанными с этим ключом. Это похоже на мощную систему индексации, которая связывает героев с их собственными атрибутами.
2. **Сверхбыстрый поиск:** Словари превосходно отвечают на такие вопросы, как «Каковы возможности этого героя?» Они могут выполнять молниеносный поиск на основе личности героя. Это похоже на сверхмощную базу данных, которая быстро извлекает информацию о конкретном герое.
3. **Отсутствие дубликатов:** Благодаря словарям каждый герой уникален. Как и в

сверхвселенной, в словаре не может быть двух одинаковых героев. Если вы попытаетесь добавить уже существующего героя, словарь сохранит только один его экземпляр, гарантируя сохранность уникальности каждого героя.

4. **Динамичность и универсальность:** Словари динамичны и универсальны. Вы можете модифицировать их на лету, обновляя силы или союзников героев, добавляя новых героев или удаляя существующих. Словари обеспечивают гибкую структуру для размещения постоянно развивающегося мира супергероев.
5. **Перекрестные ссылки:** Словари позволяют разгадать запутанные отношения героев и злодеев и связать героев с их союзниками. Вы можете искать злодеев на основе героев или наоборот, обнаруживая эпические сражения и соперничество во вселенной героев. Словари также позволяют исследовать запутанную сеть отношений между героями, создавая взаимосвязанное сообщество супергероев.

Помните, что словари — это ваш портал к созданию динамических контейнеров, заполненных уникальными элементами и связанными с ними атрибутами. Они дают вам возможность создавать, манипулировать и исследовать их содержимое гибким и универсальным образом. Итак, я призываю вас раскрыть секреты словарей в Python и использовать их возможности, чтобы раскрыть свой творческий потенциал!

Великолепно. Давай рассмотрим некоторые примеры кода, чтобы увидеть словари в действии. Давай придерживаться темы супергероев, чтобы проиллюстрировать свойства, которые мы перечислили ранее.⁸⁹

Конечно! Вот несколько примеров кода, демонстрирующих силу словарей в мире супергероев:

```
1 # Create a superhero dictionary
2 spiderman = {
3     "powers": ["superhuman strength", "wall-crawling", "web-slinging"],
4     "allies": ["Iron Man", "Captain America", "Black Widow"],
5     "enemies": ["Green Goblin", "Venom", "Doc Octopus"]
6 }
7 # Key-value Pairs
8 print("\n=== Key-Value Pairs ===")
9 print("\nKeys in Spiderman dictionary:")
10 for key in spiderman.keys():
11     print(f"\t{key}")
12
13 print("\nValues in Spiderman dictionary:")
14 for value in spiderman.values():
15     print(f"\t{value}")
16
17 print("\nKey-Value pairs in Spiderman dictionary:")
18 for key, value in spiderman.items():
19     print(f"{key} -> {value}")
```

```
=== Key-Value Pairs ===
```

```
Keys in Spiderman dictionary:
```

```
powers
allies
enemies
```

```
Values in Spiderman dictionary:
```

```
['superhuman strength', 'wall-crawling', 'web-slinging']
['Iron Man', 'Captain America', 'Black Widow']
['Green Goblin', 'Venom', 'Doc Octopus']
```

```
Key-Value pairs in Spiderman dictionary:
```

```
powers -> ['superhuman strength', 'wall-crawling', 'web-slinging']
allies -> ['Iron Man', 'Captain America', 'Black Widow']
enemies -> ['Green Goblin', 'Venom', 'Doc Octopus']
```

```
1 # Superfast Lookups
2 print("\n=== Superfast Lookups ===")
3
4 # Attributes are indexed by key
5 print("Spider-Man's powers:", spiderman["powers"])
6 print("Spider-Man's allies:", spiderman["allies"])
7 print("Spider-Man's enemies:", spiderman["enemies"])
```

```
=== Superfast Lookups ===
```

```
Spider-Man's powers: ['superhuman strength', 'wall-crawling', 'web-slinging']
Spider-Man's allies: ['Iron Man', 'Captain America', 'Black Widow']
Spider-Man's enemies: ['Green Goblin', 'Venom', 'Doc Octopus']
```

```
1 # No Duplicates
2 print("\n=== No Duplicates ===")
3 print("Spider-Man's powers:\t", spiderman["powers"])
4
5 # Duplicate key, overwrite
6 spiderman.update({"powers" : ["stamina", "fancy custome"]})
7 print("Spider-Man's updated powers:", spiderman["powers"])
```

```
=== No Duplicates ===
```

```
Spider-Man's powers: ['superhuman strength', 'wall-crawling', 'web-slinging']
Spider-Man's updated powers: ['stamina', 'fancy custome']
```

```

1 # Dynamic and Versatile
2 print("\n=== Dynamic and Versatile ===")
3 print("Spider-Man's enemies:\t", spiderman["enemies"])
4
5 spiderman["enemies"].remove("Green Goblin")
6 spiderman["enemies"].append("Carnage")
7 print("Spider-Man's updated enemies:", spiderman["enemies"])

```

```

=== Dynamic and Versatile ===
Spider-Man's enemies:  ['Green Goblin', 'Venom', 'Doc Octopus']
Spider-Man's updated enemies: ['Venom', 'Doc Octopus', 'Carnage']

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/epvbefcm>

Как насчет того, чтобы составить энциклопедию супергероев? Можно ли реализовать ее как словарь словарей? Какой-то вложенный словарь? Используйте пример кода для иллюстрации.⁹⁰

Конечно! Вы можете создать энциклопедию супергероев, используя структуру вложенного словаря. Каждый супергерой может быть ключом во внешнем словаре, а соответствующее значение — внутренним словарем, в котором хранятся сведения о супергерое. Вот пример фрагмента кода вместе с соответствующими результатами выполнения программы:

```

1 # Creating the superhero encyclopedia
2 superheroes = {
3     "Wonder Woman": {
4         "powers": ["Superhuman strength", "Lasso of Truth", "Flight"],
5         "allies": ["Superman", "Batman"],
6         "enemies": ["Cheetah", "Ares"],
7         "debut": 1941
8     },
9     "Black Widow": {
10        "powers": ["Master spy", "Hand-to-hand combat", "Espionage"],
11        "allies": ["Captain America", "Hawkeye"],
12        "enemies": ["Taskmaster", "Red Guardian"],
13        "debut": 1964
14    },
15    "Captain Marvel": {
16        "powers": ["Superhuman strength", "Energy projection", "Flight"],
17        "allies": ["Iron Man", "Thor"],
18        "enemies": ["Thanos", "Yon-Rogg"],
19        "debut": 1968
20    },

```



```

21 "Storm": {
22     "powers": ["Weather manipulation", "Flight", "Energy absorption"],
23     "allies": ["Wolverine", "Cyclops"],
24     "enemies": ["Magneto", "Apocalypse"],
25     "debut": 1975
26 },
27 "Superman": {
28     "powers": ["Superhuman strength", "Flight", "Heat vision"],
29     "allies": ["Batman", "Wonder Woman"],
30     "enemies": ["Lex Luthor", "Doomsday"],
31     "debut": 1938
32 },
33 "Spider-Man": {
34     "powers": ["Wall-crawling", "Spider-sense", "Web-slinging"],
35     "allies": ["Iron Man", "Captain America"],
36     "enemies": ["Green Goblin", "Venom", "Doc Octopus"],
37     "debut": 1962
38 },
39 "Hulk": {
40     "powers": ["Superhuman strength", "Invulnerability", "Regeneration"],
41     "allies": ["Thor", "Black Widow"],
42     "enemies": ["Abomination", "Red Hulk"],
43     "debut": 1962
44 },
45 "Batman": {
46     "powers": ["Intelligence", "Riches", "Martial arts"],
47     "allies": ["Superman", "Wonder Woman"],
48     "enemies": ["Joker", "Bane"],
49     "debut": 1939
50 },
51 "Wolverine": {
52     "powers": ["regeneration", "adamantium claws", "heightened senses"],
53     "allies": ["Professor X", "Cyclops", "Storm"],
54     "enemies": ["Sabretooth", "Magneto", "Mystique"],
55     "debut": 1974
56 }
57 }
58 print(superheroes.keys())

```

```

dict_keys(['Wonder Woman', 'Black Widow', 'Captain Marvel', 'Storm', 'Superman',
'Spider-Man', 'Hulk', 'Batman', 'Wolverine'])

```

```

1 # Printing details of a superhero given a name in the encyclopedia
2 name = input("\nEnter a name to search for in the superhero encyclopedia: ")
3 name = name.title() # Capitalize first letter of each word in name
4
5 if name in superheroes:
6     superhero = superheroes[name]
7     print("-----")
8     print(f"{name:^20}")
9     print("-----")
10    print(f"Powers: {' , '.join(superhero['powers'])}")
11    print(f"Allies: {' , '.join(superhero['allies'])}")
12    print(f"Enemies: {' , '.join(superhero['enemies'])}")
13    print(f"Debut Year: {superhero['debut']}")
14 else:
15    print("Sorry, superhero not found in the encyclopedia.")

```

Enter a name to search for in the superhero encyclopedia: Thor
 Sorry, superhero not found in the encyclopedia.

Enter a name to search for in the superhero encyclopedia: black widow

 Black Widow

 Powers: Master spy, Hand-to-hand combat, Espionage
 Allies: Captain America, Hawkeye
 Enemies: Taskmaster, Red Guardian
 Debut Year: 1964

```

1 # Adding new superheroes
2 superheroes["Captain America"] = {
3     "powers": ["Superhuman strength", "Shield throwing", "Leadership"],
4     "allies": ["Iron Man", "Thor"],
5     "enemies": ["Red Skull", "Winter Soldier"],
6     "debut": 1941
7 }
8 # Removing a superhero
9 del superheroes["Hulk"]
10 # Updating superheroes details
11 superheroes["Wolverine"]["allies"].append("Jean Grey")
12 superheroes["Spider-Man"]["enemies"].remove("Doc Octopus")

```

```

13 # Printing the updated superhero encyclopedia
14 value = input("\nWhich attribute (powers/allies/enemies/debut) would you like to see? ")
15 value = value.lower() # Convert attribute to lowercase
16 print("-----")
17 print(f"{'Superhero Encyclopedia':^50}")
18 print("-----")
19
20 for key in superheroes.keys():
21     if value in superheroes[key]:
22         print(f"{key:^20} -> {superheroes[key][value]}")
23     else:
24         print(f"Sorry, no {value} found for superhero {key}")

```

Which attribute (powers/allies/enemies/debut) would you like to see? allies

```

-----
                Superhero Encyclopedia
-----

Wonder Woman    -> ['Superman', 'Batman']
Black Widow    -> ['Captain America', 'Hawkeye']
Captain Marvel  -> ['Iron Man', 'Thor']
    Storm        -> ['Wolverine', 'Cyclops']
    Superman     -> ['Batman', 'Wonder Woman']
    Spider-Man   -> ['Iron Man', 'Captain America']
    Batman       -> ['Superman', 'Wonder Woman']
    Wolverine    -> ['Professor X', 'Cyclops', 'Storm', 'Jean Grey']
    Captain America -> ['Iron Man', 'Thor']

```

Which attribute (powers/allies/enemies/debut) would you like to see? age

```

-----
                Superhero Encyclopedia
-----

Sorry, no age found for superhero Wonder Woman
Sorry, no age found for superhero Black Widow
Sorry, no age found for superhero Captain Marvel
Sorry, no age found for superhero Storm
Sorry, no age found for superhero Superman
Sorry, no age found for superhero Spider-Man
Sorry, no age found for superhero Batman
Sorry, no age found for superhero Wolverine

```

```
Which attribute (powers/allies/enemies/debut) would you like to see? debut
```

```
-----  
Superhero Encyclopedia  
-----
```

```
Wonder Woman -> 1941  
Black Widow -> 1964  
Captain Marvel -> 1968  
Storm -> 1975  
Superman -> 1938  
Spider-Man -> 1962  
Batman -> 1939  
Wolverine -> 1974  
Captain America -> 1941
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinypu.com/5feekke>

Прекрасно! Ты также упомянул, что словари позволяют нам исследовать сложные отношения и связи между элементами в словаре. Что это значит? Могут ли словари помочь мне определить такие отношения, как «враги моих союзников – мои враги», «союзники моих союзников – мои друзья», «герои с такой же суперсилой, как у меня, – мои приятели» или «отобразить таблицу супергероев», отсортированных по году дебюта? Пожалуйста, приведи несколько примеров.⁹¹

Конечно! Вот четыре примера перекрестных ссылок на элементы данных с использованием словарей.

- Пример 1: Враги моих союзников - мои враги.

```

1 # Constructing a dictionary to store extended enemies
2 allies_enemies = {}
3 for hero, details in superheroes.items():
4     allies = details["allies"]
5     enemies = details["enemies"]
6     for ally in [hero] + allies:
7         # Check if the hero is not already present in the allies_enemies dictionary
8         if hero not in allies_enemies:
9             allies_enemies[hero] = []
10        # Check if the ally exists in the superheroes dictionary
11        if ally in superheroes:
12            # Extend the list of enemies for the hero with the enemies of the ally
13            allies_enemies[hero].extend(superheroes[ally]["enemies"])
14
15 # Printing the original enemies
16 print('-'*80)
17 print(f'Original Enemies:^80')
18
19 print('-'*80)
20 for hero, details in superheroes.items():
21     print(f'{hero:^10} -> {details["enemies"]}')
22
23 # Printing the extended enemies
24 print('-'*80)
25 print(f'The Enemies of My Allies are My Enemies:^80')
26 print('-'*80)
27 for ally, enemies in allies_enemies.items():
28     print(f'{ally:^10} -> {enemies}')

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/yckcbr4k>

Код направлен на создание словаря `allies_enemies` для хранения расширенных врагов героев на основе их союзников. Алгоритм перебирает каждого героя и его детали в словаре `superheroes`. Он извлекает списки союзников и врагов для каждого героя. Для каждого героя и его союзников проверяется, отсутствует ли герой в словаре `allies_enemies`. Если нет, он инициализирует пустой список для героя. Затем он проверяет, существует ли союзник в словаре `superheroes`. Если союзник существует, он расширяет список врагов героя врагами союзника с помощью функции `extend()`. Наконец, код выводит на печать исходных и дополнительных врагов.

Original Enemies

Wonder Woman -> ['Cheetah', 'Ares']
Black Widow -> ['Taskmaster', 'Red Guardian']
Captain Marvel -> ['Thanos', 'Yon-Rogg']
 Storm -> ['Magneto', 'Apocalypse']
 Superman -> ['Lex Luthor', 'Doomsday']
Spider-Man -> ['Green Goblin', 'Venom']
 Batman -> ['Joker', 'Bane']
Wolverine -> ['Sabretooth', 'Magneto', 'Mystique']
Captain America -> ['Red Skull', 'Winter Soldier']

The Enemies of My Allies are My Enemies

Wonder Woman -> ['Cheetah', 'Ares', 'Lex Luthor', 'Doomsday', 'Joker', 'Bane']
Black Widow -> ['Taskmaster', 'Red Guardian', 'Red Skull', 'Winter Soldier']
Captain Marvel -> ['Thanos', 'Yon-Rogg']
 Storm -> ['Magneto', 'Apocalypse', 'Sabretooth', 'Magneto', 'Mystique']
 Superman -> ['Lex Luthor', 'Doomsday', 'Joker', 'Bane', 'Cheetah', 'Ares']
Spider-Man -> ['Green Goblin', 'Venom', 'Red Skull', 'Winter Soldier']
 Batman -> ['Joker', 'Bane', 'Lex Luthor', 'Doomsday', 'Cheetah', 'Ares']
Wolverine -> ['Sabretooth', 'Magneto', 'Mystique', 'Magneto', 'Apocalypse']
Captain America -> ['Red Skull', 'Winter Soldier']

► Пример 2: Союзники моих союзников - мои друзья.

```
1 # Constructing a dictionary to store extended friends
2 allies_friends = {}
3 for hero, details in superheroes.items():
4     allies = details["allies"]
5     for ally in [hero] + allies:
6         if ally in superheroes:
7             # Finding friends of the current hero's ally, excluding the hero itself
8             friends = set(superheroes[ally]["allies"]) - set([hero])
9             allies_friends.setdefault(hero, []).extend(friends)
10
11 # Printing the original allies
12 print('-'*80)
13 print(f"{'Original Allies':^80}")
14 print('-'*80)
15 for hero, details in superheroes.items():
16     # Printing the allies of each hero
17     print(f"{hero:^10} -> {details['allies']}")
18
19 # Printing the extended allies (friends)
20 print('-'*80)
21 print(f"{'The Allies of My Allies are My Friends':^80}")
22 print('-'*80)
23 for hero, friends in allies_friends.items():
24     # Printing the extended friends list, excluding duplicates
25     print(f"{hero:^10} -> {set(friends)}")
26
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/4wsy98ra>

Предоставленный код предназначен для создания словаря `allies_friends`, в котором хранятся расширенные друзья каждого героя на основе их союзников. Алгоритм перебирает каждого героя и его свойства в словаре `superheroes`. Для каждого героя он получает список союзников. Затем он проверяет, существует ли каждый союзник в качестве ключа в словаре `superheroes`, и, если да, находит их друзей, вычитая самого героя из их союзников с помощью операций над множествами.

Функция `setdefault()` используется для расширения списка друзей текущего героя в словаре `allies_friends`. Это гарантирует, что если герой еще не является ключом в словаре `allies_friends`, в качестве значения по умолчанию создается новый пустой список. Это позволяет легко расширять список друзей, не перезаписывая существующих друзей. Наконец, код выводит исходных союзников для каждого

героя и дополнительных союзников (с друзьями), перебирая словарь `allies_friends` и исключая дубликаты с помощью наборов.

```
-----  
Original Allies  
-----  
Wonder Woman -> ['Superman', 'Batman']  
Black Widow -> ['Captain America', 'Hawkeye']  
Captain Marvel -> ['Iron Man', 'Thor']  
    Storm    -> ['Wolverine', 'Cyclops']  
    Superman -> ['Batman', 'Wonder Woman']  
Spider-Man -> ['Iron Man', 'Captain America']  
    Batman   -> ['Superman', 'Wonder Woman']  
Wolverine  -> ['Professor X', 'Cyclops', 'Storm', 'Jean Grey']  
Captain America -> ['Iron Man', 'Thor']  
-----  
The Allies of My Allies are My Friends  
-----  
Wonder Woman -> {'Superman', 'Batman'}  
Black Widow -> {'Iron Man', 'Captain America', 'Hawkeye', 'Thor'}  
Captain Marvel -> {'Iron Man', 'Thor'}  
    Storm    -> {'Cyclops', 'Professor X', 'Wolverine', 'Jean Grey'}  
    Superman -> {'Batman', 'Wonder Woman'}  
Spider-Man -> {'Captain America', 'Iron Man', 'Thor'}  
    Batman   -> {'Superman', 'Wonder Woman'}  
Wolverine  -> {'Storm', 'Professor X', 'Cyclops', 'Jean Grey'}  
Captain America -> {'Iron Man', 'Thor'}
```

► Пример 3: Герои с одинаковыми суперспособностями — друзья.

```
1 # Constructing a dictionary to store superpower buddies  
2 superpower_buddies = {}  
3 for hero, details in superheroes.items():  
4     powers = details["powers"]  
5     for power in powers:  
6         superpower_buddies.setdefault(power, []).append(hero)
```



```
7 print('-'*80)
8 print(f"{'Heroes with same superpowers are buddies':^80}")
9 print('-'*80)
10 for power, buddies in superpower_buddies.items():
11     print(f"|{power.title():^20}| {'|', '.join(buddies):<56}|")
12     print('-'*80)
13
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/4mvsck3s>

Код направлен на создание словаря `superpower-buddies` для хранения супергероев, обладающих одинаковыми сверхспособностями. Алгоритм перебирает каждого героя и его детали в словаре `superheroes`. Для каждого героя он получает список их способностей. Затем он перебирает каждую способность и использует функцию `append()`, чтобы добавить героя в список приятелей, связанных с этой способностью в словаре `superpower-buddies`.

Функция `setdefault()` гарантирует, что если способность еще не является ключом в словаре `superpower-buddies`, в качестве значения по умолчанию создается новый пустой список. Наконец, код выводит словарь `superpower-buddies`, отображая каждую способность и соответствующих друзей. Он форматирует вывод в табличном виде, используя `|` символ для разделения столбцов способностей и друзей.

Heroes with same superpowers are buddies

| Superhuman Strength | Wonder Woman, Captain Marvel, Superman, Captain America |

| Lasso Of Truth | Wonder Woman |

| Flight | Wonder Woman, Captain Marvel, Storm, Superman |

| Master Spy | Black Widow |

| Hand-To-Hand Combat | Black Widow |

| Espionage | Black Widow |

| Energy Projection | Captain Marvel |

| Weather Manipulation | Storm |

| Energy Absorption | Storm |

| Heat Vision | Superman |

```

-----
| Wall-Crawling | Spider-Man |
-----
| Spider-Sense | Spider-Man |
-----
| Web-Slinging | Spider-Man |
-----
| Intelligence | Batman |
-----
| Riches | Batman |
-----
| Martial Arts | Batman |
-----
| Regeneration | Wolverine |
-----
| Adamantium Claws | Wolverine |
-----
| Heightened Senses | Wolverine |
-----
| Shield Throwing | Captain America |
-----
| Leadership | Captain America |
-----

```

► Пример 4: Сортировка супергероев по году дебюта.

```

1 # Creating a sorted list of keys based on debut year
2 debut_dict = {key: superheroes[key]["debut"] for key in superheroes.keys()}
3 sorted_keys = sorted(debut_dict.keys(), key=debut_dict.get)
4 # Printing the table of superheroes sorted by debut year
5 print('- '*30)
6 print(f"{'Heroes sorted by debut year':^30}")
7 print('- '*30)
8 print(f"{'Superhero':^15} | {'Debut Year':^15}")
9 print('- '*30)
10 for key in sorted_keys:
11     print(f"{key:<15} | {superheroes[key]['debut']:^15}")
12 print('- '*30)
13

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/dshc9uf4>

Этот код сначала создает словарь `debut_dict` для хранения года дебюта супергероя. Охват словаря `{key: superheroes[key]["debut"] for key in superheroes.keys()}` перебирает ключи словаря `superheroes` и присваивает соответствующий год дебюта каждому ключу в `debut_dict`. Этот шаг эффективно извлекает годы дебюта для каждого супергероя, аналогично тому, как мы обсуждали охват списка ранее.

Вторая строка кода сортирует ключи в `debut_dict` на основе их соответствующих значений (года дебюта) с помощью функции `sorted()`. Параметр `key` имеет значение `debut_dict.get`, которое извлекает значение (год дебюта) для каждого ключа. Этот шаг сортировки гарантирует, что супергерои расположены в порядке возрастания в зависимости от года их дебюта. Последний блок кода печатает таблицу, перебирая отсортированный список ключей и применяя некоторые улучшения форматирования.

```
-----
Heroes sorted by debut year
-----
 Superhero   |   Debut Year
-----
Superman     |     1938
Batman       |     1939
Wonder Woman |     1941
Captain America |     1941
Spider-Man   |     1962
Black Widow |     1964
Captain Marvel |     1968
Wolverine    |     1974
Storm        |     1975
-----
```

Ну давай те же! Дайте волю своему воображению, изучите новые примеры и проведите фантастическое время, исследуя захватывающий мир словарей со всеми их невероятными функциями и возможностями! Приготовьтесь испытать магию организации и поиска данных!

5.6 Время поработать руками!

Предложи несколько забавных и увлекательных упражнений по программированию для отработки списков, кортежей и наборов. Каждое упражнение должно сопровождаться постановкой задачи и примерами ожидаемых результатов.⁹²

1. **Работа со списками:** Для заданного списка чисел напишите программу,

выполняющую следующие операции:

- ▶ Удаление всех четных чисел из списка.
- ▶ Возведение в квадрат каждое оставшееся в списке число.
- ▶ Выведение обновленного списка.

Пример:

```
Input: [1, 2, 3, 4, 5, 6, 7, 8]
Output: [1, 9, 25, 49]
```

2. **Конкатенация кортежей:** Имея два кортежа, напишите программу для их конкатенации и вывода нового кортежа. Пример:

```
Input:
Tuple1: (1, 2, 3)
Tuple2: (4, 5, 6)
Output: (1, 2, 3, 4, 5, 6)
```

3. **Операции с наборами строк:** Принимая во внимание два набора строк, выполните следующие операции и выведите результаты:

- ▶ Объединение: Найдите объединение двух множеств.
- ▶ Пересечение: найдите строки, общие для двух наборов.
- ▶ Различия: найдите строки, которые есть в первом наборе, но которых нет во втором.

Пример:

```
Input:
Set1: {"apple", "banana", "orange"}
Set2: {"orange", "grape", "kiwi"}
Output:
Union: {"apple", "banana", "orange", "grape", "kiwi"}
Intersection: {"orange"}
Difference: {"apple", "banana"}
```

4. **Фильтрация списка:** Для заданного списка чисел напишите программу, которая отфильтровывает все простые числа, которые больше 10, и выведет новый список. Пример:

```
Input: [5, 12, 8, 17, 3, 20, 41]
Output: [5, 12, 8, 3, 20]
```

5. **Уникальные перестановки:** Имея список чисел, напишите программу для генерации всех уникальных перестановок чисел. Пример:

```
Input: [1, 2, 3]
Output: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

1. **Сортировка кортежей:** Имея список кортежей, содержащих имена и возраст, напишите программу для сортировки кортежей по возрасту в порядке убывания. Пример:

```
Input: [("John", 25), ("Alice", 30), ("Bob", 22)]
Output: [("Alice", 30), ("John", 25), ("Bob", 22)]
```

2. **Операции с несколькими наборами:** Имея список наборов, напишите программу для выполнения следующих операций и вывода результатов:

- ▶ Объединение: Произведите объединение всех наборов.
- ▶ Пересечение: Найдите общие элементы во всех множествах.
- ▶ Отличия: найдите элементы, которые присутствуют в первом наборе, но отсутствуют в других наборах.

Пример:

```
Input:
Sets: [{1, 2, 3}, {2, 3, 4}, {3, 4, 5}]
Output:
Union: {1, 2, 3, 4, 5}
Intersection: {3}
Difference: {1}
```

3. **Перестановка списка:** Имея список строк, напишите программу для перестановки строк таким образом, чтобы каждая строка начиналась с последнего символа предыдущей строки. Если такой строки нет, продолжите с любой неиспользуемой строки из списка. Пример:

```
Input: ["apple", "orange", "banana", "rhino", "elephant", "tiger"]
Output: ["apple", "elephant", "tiger", "rhino", "orange", "banana"]
```

4. **Уникальные комбинации:** Для заданного списка чисел напишите программу, генерирующую для списка все уникальные комбинации определенной длины k . Пример:

```
Input: [1, 2, 3], k = 2
Output: [[1, 2], [1, 3], [2, 3]]
```

5. **Работа с кортежами:** Имея список кортежей, содержащих имена и баллы, напишите программу, которая находит средний балл для каждого имени и возвращает список кортежей в формате `(name, average-score)`, отсортированных по среднему баллу в порядке убывания. Пример:

```
Input: [("John", 80), ("Alice", 90), ("Bob", 75), ("Alice", 95)]
Output: [("Alice", 92.5), ("John", 80), ("Bob", 75)]
```

6. **Разбиение списка:** Для заданного списка чисел напишите программу, которая разбивает список на подспски так, чтобы сумма каждого подсписка была меньше или равна заданному максимальному значению `k`. Подспски должны быть максимально сбалансированы. Пример:

```
Input: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 15
Output: [[1, 2, 3, 4, 5], [6, 7], [8, 9], [10]]
```

7. **Операции с матрицами:** Имея две матрицы, представленные в виде списков списков, напишите программу для выполнения таких операций с матрицами, как сложение, вычитание и умножение. Пример:

```
Input:
Matrix A: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Matrix B: [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
Output:
Addition: [[10, 10, 10], [10, 10, 10], [10, 10, 10]]
Subtraction: [[-8, -6, -4], [-2, 0, 2], [4, 6, 8]]
Multiplication: [[30, 24, 18], [84, 69, 54], [138, 114, 90]]
```

Предложи забавные и увлекательные упражнения по кодированию для практики работы со строками и словарями, включая вложенные словари. Каждое упражнение должно сопровождаться постановкой задачи и примерами ожидаемых результатов.⁹³

1. **Счетчик слов:** Напишите программу, которая принимает строку в качестве входных данных и подсчитывает частоту появления каждого слова в строке. Пример:

```
Input: "I love to code, code, and code!"
Output: {'I': 1, 'love': 1, 'to': 1, 'code': 3, 'and': 1}
```

2. **Объединение словарей:** Напишите программу, которая получает в качестве входных данных два словаря и объединяет их в один словарь. Если ключ присутствует в обоих словарях, значения должны быть объединены в список. Пример:

```
Input:
dict1 = {'A': 1, 'B': 2, 'C': 3}
dict2 = {'B': 4, 'D': 5}
Output: {'A': 1, 'B': [2, 4], 'C': 3, 'D': 5}
```

3. **Доступ к вложенному словарю:** Напишите программу, которая получает доступ к значениям вложенного словаря, используя список ключей. Пример:

```
Input:
data = {'person': {'name': 'John', 'age': 25, 'address': {'city': 'New York',
    'country': 'USA'}}}
keys = ['person', 'address', 'city']
Output: 'New York'
```

4. **Счетчик частоты символов:** Напишите программу, которая принимает строку в качестве входных данных и подсчитывает частоту появления каждого символа в строке. Пример:

```
Input: "Hello, World!"
Output: {'H': 1, 'e': 1, 'l': 3, 'o': 2, ',': 1, ' ': 1, 'W': 1, 'r': 1, 'd':
    1, '!': 1}
```

5. **Работа со словарем:** Напишите программу, выполняющую различные действия со словарем. Программа должна отображать меню опций для выбора пользователем. Пример:

```
Input:
my_dict = {'A': 1, 'B': 2}
Output:
{'A': 1, 'B': 2, 'C': 3} # After adding {'C': 3}
{'A': 1, 'B': 5} # After updating 'B' to 5
{'A': 1} # After removing 'B'
Key 'A' exists in the dictionary: True
Key 'B' exists in the dictionary: False
```

6. **Средство проверки анаграмм:** Напишите программу, которая принимает на вход две строки и определяет, являются ли они анаграммами друг друга. Анаграмма — это слово или фраза, образованная перестановкой букв другого слова или фразы. Пример:


```
Input: "listen", "silent"
Output: True
```

7. **Замена символов:** Напишите программу, которая берет строку и заменяет все экземпляры указанного символа другим символом. Пример:

```
Input: "Hello, World!", replace 'l' with 'x'
Output: "Hexxo, Worxd!"
```

8. **Инверсия словаря:** Напишите программу, которая принимает словарь в качестве входных данных и возвращает новый словарь, в котором ключи и значения меняются местами. Пример:

```
Input: {'A': 1, 'B': 2, 'C': 3}
Output: {1: 'A', 2: 'B', 3: 'C'}
```

9. **Счетчик гласных:** Напишите программу, которая принимает строку в качестве входных данных и подсчитывает количество гласных (a, e, i, o, u) в строке. Пример:

```
Input: "Hello, World!"
Output: 3
```

10. **Обновление вложенного словаря:** Напишите программу, которая обновляет значения вложенного словаря на основе заданного условия. Пример:

```
Input:
data = {'A': {'score': 80}, 'B': {'score': 90}, 'C': {'score': 70}}
threshold = 75
Output: {'A': {'score': 'Pass'}, 'B': {'score': 'Pass'}, 'C': {'score':
'Fail'}}
```

11. **Поиск по вложенному словарю:** Напишите программу, которая ищет конкретное значение во вложенном словаре и возвращает соответствующий ключ(и), где это значение найдено. Пример:

```
Input:
data = {'A': {'score': 80}, 'B': {'score': 90}, 'C': {'score': 80}, 'D':
{'score': 70}}
target = 80
Output: ['A', 'C']
```

12. **Режим удаления слов:** Напишите программу, которая принимает строку в качестве входных данных и удаляет слово (слова) с наибольшей частотой встречаемости. Если

несколько слов имеют одинаковую максимальную частотность, удалите их все.
Пример:

```
Input: "The quick brown fox jumps over the lazy dog. The dog barks loudly."  
Output: "quick brown fox jumps over the lazy. barks loudly."
```

Модульные дела

6.1 Модульность и возможность повторного использования

Кратко объясни концепцию модульности и возможности повторного использования, а также чем они полезны в проектах программирования. Затем, используя аналогии из реального мира, объясни различные подходы к реализации этих понятий в дружелюбной, но аккуратной манере.⁹⁴

Многочисленное использование и модульность — это своего рода секретные приемы в разработке программного обеспечения, особенно в случае больших и сложных проектов. Они имеют ряд преимуществ, облегчающих жизнь программистам.

Когда вы пишете повторно используемый код, вам не нужно каждый раз начинать с нуля. Вы можете сэкономить много времени и усилий, используя существующие решения вместо того, чтобы изобретать велосипед. А с модульностью вы разбиваете свой код на более мелкие, управляемые части. Это как кусочки головоломки, которые идеально сочетаются друг с другом. Это упрощает понимание, обслуживание и исправление любых возникающих ошибок. Кроме того, вы можете повторно использовать эти модульные элементы в разных частях вашего проекта.

Таким образом, когда дело доходит до крупных проектов, ключевое значение имеет повторное использование и модульность. Это поможет вам работать продуктивнее, эффективно сотрудничать и поддерживать самобытность вашего кода в долгосрочной перспективе. В Python у вас есть куча крутых приемов и практик для воплощения этих идей в жизнь:

- 1. Функции:** Функции похожи на специализированные инструменты на кухне. Они инкапсулируют конкретные приправы и могут повторно использоваться в различных рецептах. Точно так же в программировании функции инкапсулируют определенные операции, способствуя повторному использованию кода. Используя функции, программисты могут писать модульный код, который можно легко использовать повторно, что упрощает работу с более крупными и сложными проектами.
- 2. Классы и объектно-ориентированное программирование (ООП):** Классы в ООП подобны специалистам на стройке. Каждый работник имеет определенную профессию и опыт. Точно так же классы инкапсулируют данные и действия в объекты, способствуя повторному использованию кода. Используя классы и

объекты, программисты могут создавать модульный и масштабируемый код, что упрощает управление сложностью в крупных проектах.

3. **Модули и пакеты:** Модули и пакеты похожи на наборы инструментов с помеченными для разных инструментов ячейками. Они предоставляют способ организации и упаковки связанных компонентов кода, способствуя повторному использованию кода. Используя модули и пакеты, программисты могут легко находить и повторно использовать функциональные возможности кода, что позволяет эффективно разрабатывать и поддерживать более крупные проекты.
4. **Библиотеки и фреймворки:** Библиотеки и фреймворки похожи на книги в библиотеке или книжном магазине. Вместо того, чтобы писать все с нуля, программисты могут использовать существующие библиотеки и фреймворки, которые предлагают готовый код и функции для конкретных задач. Используя эти ресурсы, программисты могут сэкономить время и усилия, ускоряя процесс разработки более крупных проектов.
5. **Шаблоны проектирования:** Шаблоны проектирования похожи на архитектурные чертежи для строительства зданий. Они обеспечивают проверенные решения общих проблем в разработке программного обеспечения. Следуя установленным шаблонам проектирования, программисты могут писать модульный и масштабируемый код, отвечающий сложным требованиям. Шаблоны проектирования способствуют повторному использованию кода и удобству сопровождения, что делает их необходимыми для более крупных и сложных проектов.
6. **Разделение задач:** Разделение задач похоже на разделение проекта между экспертами в определенных областях знаний. Разбивая программу на более мелкие модули, каждый из которых отвечает за определенную задачу или функциональность, программисты могут добиться модульности кода и возможности его повторного использования. Этот подход обеспечивает лучшую организацию, совместную работу и обслуживание кода в более крупных проектах.

Применяя эти методы и приемы, разработчики Python могут создавать повторно используемый модульный код, который легче понять, поддерживать и расширять. Эти концепции также способствуют общему качеству кода, совместной работе и масштабируемости.

Вау! Это впечатляющий список мощных идей. Однако, поскольку я все еще изучаю основы, пожалуйста, порекомендуй один или два метода из списка, с которых я должен начать как новичок, а затем опиши ключевые компоненты каждого метода в расслабляющей и занимательной манере.⁹⁵

Безусловно! Как новичку, я рекомендую вам начать с функций и классов. Эти

концепции являются фундаментальными строительными блоками в программировании и обеспечат вам прочную основу для понимания более сложных тем.

1. **Функции:** Функции — это как изысканные рецепты в мире программирования. Подобно тому, как рецепт помогает вам приготовить вкусное блюдо, функции помогают компьютеру выполнять определенные задачи. Думайте о функциях как о мини-программах внутри вашей программы. У них есть наименование и набор инструкций, которые выполняют определенное действие. Функции могут принимать входные данные (ингредиенты) и производить выходные данные (конечное блюдо). Лучшая часть? Однажды создав функцию, вы можете использовать ее снова и снова, экономя ваше время и усилия. Вот еще некоторые технические подробности о функциях Python:

- ▶ **Специализированные средства решения задач:** Функции похожи на специализированные средства решения задач. Они принимают входные данные (параметры) и выполняют над ними определенные операции для получения выходных данных (возвращаемых значений). Функции помогают разбить сложные проблемы на решаемые части. Вы можете думать о них как о мини-программах в вашем коде, которые можно вызывать и повторно использовать всякий раз, когда они вам нужны. Функции определяются с помощью оператора `def`.
- ▶ **Возвращение значений:** Функции могут возвращать значения с помощью оператора `return`. Это позволяет им выдавать результат или выходные данные, которые можно использовать в других частях кода. Это как функция, возвращающая вам подарок после выполнения своей задачи.
- ▶ **Параметры:** Функции могут принимать параметры, которые при вызове функции являются входными данными. Параметры позволяют передавать значения в функцию, предоставляя необходимые данные для работы функции. Это похоже на предоставление инструкций или информации функции, чтобы управлять ее действиями.
- ▶ **Лямбда-функции:** Лямбда-функции, также известные как анонимные функции, представляют собой компактные встроенные функции, не требующие формального оператора `def`. Они определяются с помощью ключевого слова `lambda` и полезны для простых однострочных операций. Лямбда-функции похожи на небольших, гибких помощников, которые могут быстро помочь вам без особых церемоний.

2. **Классы:** Представьте себе мир, в котором вы можете создавать своих собственных супергероев с уникальной силой и способностью. Это именно то, что классы позволяют делать в программировании. Класс похож на план, который определяет свойства (атрибуты) и методы (суперспособности) ваших созданных на заказ супергероев (объектов). Он инкапсулирует связанные данные и действие в единую сущность. С помощью классов вы можете создавать несколько экземпляров

(отдельных копий супергероев), которые имеют одни и те же атрибуты и способности, определенные классом. Вот еще некоторые технические подробности о классах Python:

- ▶ **Мощные шаблоны:** В ООП классы действуют как шаблоны для создания объектов. Они определяют атрибуты (переменные), описывающие состояние объекта, и методы (функции), определяющие его поведение. Это похоже на проектирование здания перед его строительством. Классы определяются с помощью оператора `class`.
- ▶ **Конструкторы:** Конструкторы – это специальные методы, используемые для инициализации объектов. Они вызываются при создании объекта и позволяют вам установить начальное состояние или значения атрибутов объекта. Конструкторы гарантируют, что объекты правильно инициализированы и готовы к использованию. Это похоже на создание чертежа для создания объектов с заранее заданными параметрами.
- ▶ **Атрибуты:** В ООП объекты имеют атрибуты, в которых хранятся данные. Атрибуты представляют характеристики или свойства объекта. Они могут быть переменными, связанными с объектом и содержащими информацию. Это похоже на объект, имеющий разные характеристики, такие как цвет, размер или тип.
- ▶ **Методы:** Методы – это функции, определенные внутри класса и используемые для выполнения операций или действий, связанных с объектом. Они могут манипулировать данными объекта, взаимодействовать с другими объектами или предоставлять функциональные возможности, характерные для цели объекта. Это похоже на объект, обладающий способностями или поведением, которое он может осуществлять.

Начав с функций и классов, вы получите практический опыт написания повторно используемого и модульного кода. Это позволит вам разбивать сложные задачи на более мелкие, управляемые части и обеспечит прочную основу для дальнейшего изучения захватывающего мира программирования. Так что надевайте передник и окунитесь в прелести гастрономических функций и приключения героических классов!

6.2 Функции

Хорошо. Итак, давайте начнем с нескольких примеров кода, чтобы увидеть функции в действии. Рассмотрим возможность использования идей, связанных с социальными сетями, для иллюстрации элементов, которые вы перечислили ранее. Не стесняйся использовать развлекательную тематику о Гарри Поттере и случайные функции.⁹⁶

Конечно! См. примеры ниже.

► **Специализированные средства решений:** Вот несколько примеров, иллюстрирующих эту концепцию.

```
1 # Example 1: Tagging Friends
2 import random
3
4 # Define a function to randomly tag a friend using the 'def' statement
5 def tag_friend():
6     friends = ["Alice", "Bob", "Charlie", "David", "Eve"]
7     tagged_friend = random.choice(friends)
8     # Display the result
9     print("You tagged a friend:", tagged_friend)
10
11 # Usage
12 print("\n=== Tagging a friend ===")
13 for _ in range(5):
14     tag_friend()
```

В этом примере функция `tag_friend ()` определяется с помощью оператора `def` без входных параметров, обозначенных пустыми скобками `()`. Эта функция служит специализированным средством решения задачи случайного выбора и пометки друга из сети пользователя. Для выполнения этой задачи используется функция `random.choice ()` из библиотеки `random`.

Вызывая функцию `tag_friend ()` несколько раз в цикле, вы можете многократно отмечать разных друзей, эффективно повторно используя код и экономя место в коде. Этот подход инкапсулирует логику выбора друзей и тегов в повторно используемую функцию, обеспечивая модульность кода и удобство сопровождения. Инкапсулируя эту функциональность в функцию, вы можете легко вызывать ее, когда это необходимо, избегая необходимости дублировать код в вашей программе.

```
You tagged a friend: Charlie
You tagged a friend: Eve
You tagged a friend: Charlie
You tagged a friend: Bob
You tagged a friend: Alice
```

```
1 # Example 2: Receiving Notifications
2 import datetime, time
3 # Define a function to get the notification datetime
4 def get_notification_datetime():
5     current_datetime = datetime.datetime.now()
6     notification_datetime = current_datetime.strftime("%d/%b/%Y %H:%M:%S")
7     # Display the result
8     print("New notification received at:", notification_datetime)
9     time.sleep(random.randint(0, 10)) # Wait some seconds before the next call
10
11 # Usage
12 print("\n=== Get notification ===")
13 for _ in range(3):
14     get_notification_datetime()
```

В этом примере функция `get_notification_datetime()` извлекает текущую дату и время и форматирует их как `'dd/mmm/yyyy hh:mm:ss'`. Эта функция служит специализированным средством решения проблемы извлечения метки времени полученного уведомления. Для выполнения этой задачи используется функция `datetime.datetime.now()` из библиотеки `datetime` для получения текущего значения даты и времени и метод `strftime()` для его форматирования в соответствии с указанным форматом. Полученное отформатированное уведомление с датой и временем затем отображается с использованием обновленной распечатки.

Вызывая функцию `get_notification_datetime()` несколько раз в цикле, вы можете имитировать получение уведомлений в разное время, эффективно повторно используя код без необходимости повторения одних и тех же строк кода. Этот подход способствует повторному использованию кода, устраняя избыточность и обеспечивая более эффективный и лаконичный код.

```
=== Get notification ===
New notification received at: 09/Jun/2023 02:49:37
New notification received at: 09/Jun/2023 02:49:45
New notification received at: 09/Jun/2023 02:49:51
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/ye29dv4m>

Возвраты: Вот несколько примеров, иллюстрирующих эту концепцию.


```

1 # Example 1: Get User Information
2 def get_user_info():
3     # Simulate retrieving some user information from a social media platform
4     name = "Harry Potter"
5     age = 17
6     return name, age
7
8 # Example 2: Get Profile Picture
9 def get_profile_picture():
10    # Simulate retrieving the user's profile picture
11    picture_url = "https://twitter.com/Official_HPWiki/photo"
12    return picture_url
13
14 # Example 3: Get Friends Count
15 def get_friends_count():
16    # Simulate retrieving the number of friends in the user network
17    friends = ["Hermione", "Ron", "Neville", "Luna"]
18    return len(friends)
19
20 # Main function
21 def main():
22    # Get user profile data
23    name, age = get_user_info()
24    picture_url = get_profile_picture()
25    friends_count = get_friends_count()
26
27    # Display user profile
28    print("==== User Information ====")
29    print("Name:", name)
30    print("Age:", age)
31    print("Picture:", picture_url)
32    print("Friends Count:", friends_count)
33
34 # Call the main function
35 main()

```

В Example 1 функция `get_user_info()` имитирует получение информации о пользователе из социальной сети. В этом примере мы предполагаем, что получаем `name` и `age` пользователя. Используя оператор `return`, функция может одновременно возвращать несколько значений, в данном случае строку и число.

В Example 2 функция `get_profile_picture()` определена для моделирования получения изображения профиля пользователя из платформы социальных сетей; здесь мы предполагаем, что он возвращает URL-адрес изображения, то есть строку `picture_url`.

В Example 3 функция `get_friends_count()` определена для имитации получения количества друзей, которые есть у пользователя в социальной сети. Оператор `return` в этом случае возвращает количество друзей, которое соответствует длине списка `friends`, полученного с помощью функции `len()`.

Наконец, функция `main()` служит точкой входа в программу. В Python обычной практикой является наличие функции `main()`, которая действует как центральная контрольная точка для выполнения основной логики программы.

В предоставленном фрагменте кода функция `main()` отвечает за координацию выполнения других функций. Она вызывает функции `get_user_info()`, `get_profile_picture()`, и `get_friends_count()` для получения информации о пользователе, URL-адреса изображения профиля и количества друзей. Мы присваиваем возвращенные значения переменным (`name`, `age`, `picture_url`, `friends_count`) и отображаем профиль пользователя с помощью операторов `print()`.

Наличие функции `main()` способствует организации кода и его удобочитаемости. Она отделяет основную логику управления программой от определений функций, что упрощает понимание и сопровождение кода. Кроме того, она позволяет улучшить структуру кода и его модульность, продвигая передовые методы программирования.

```
==== User Information ====
Name: Harry Potter
Age: 17
Picture: https://twitter.com/Official_HPWiki/photo
Friends Count: 4
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/2p8bj2k9>

► **Параметры:** Вот несколько примеров, иллюстрирующих эту концепцию.

```
1 import random
2
3 # Pre-built lists of users, posts, IDs, and comments
4 users = ["Harry Potter", "Hermione Granger", "Ron Weasley", "Luna Lovegood"]
5 posts = {
6     123: "I solemnly swear that I am up to no good.",
7     456: "Happiness can be found, even in the darkest of times, if one only
8         remembers to turn on the light.",
9     789: "It is our choices, Harry, that show what we truly are, far more than our
10        abilities.",
11     101112: "We've all got both light and dark inside us. What matters is the part
12            we choose to act on.",
13 }
14 comments = [
15     ("Hermione Granger", "That's a brilliant quote!"),
16     ("Ron Weasley", "Couldn't agree more, mate!"),
17     ("Luna Lovegood", "Such wisdom in those words."),
18     ("Harry Potter", "I'm glad these quotes resonate with everyone."),
19 ]
```

```

17
18 # Function to send friend request
19 def send_friend_request(sender, receiver):
20     print(f"[{sender}] has sent a friend request to [{receiver}]")
21
22 # Function to print a post
23 def print_post(post_id):
24     # print(f"'Post ID:' % post_id: {'-' }{'^'}{80}")
25     label = ' Post ID: %s ' % (post_id)
26     print(f"{label: {'-' }{'^'}{80}")
27     print(f"{posts[post_id]}")
28     print('-'*80)
29
30 # Function to like a post
31 def like_post(user, post_id):
32     print(f"{user} liked post with ID {post_id}")
33
34 # Function to comment on a post
35 def comment_on_post(user, post_id, comment):
36     print(f"{user} commented on post with ID {post_id}: {comment}")
37
38 # Main function
39 def main():
40     # Repeat actions multiple times
41     for _ in range(5):
42         sender = random.choice(users)
43         receiver = random.choice(users)
44         post_id = random.choice([key for key in posts.keys()])
45         commenter, comment = random.choice(comments)
46
47         print_post(post_id)
48         like_post(sender, post_id)
49         comment_on_post(commenter, post_id, comment)
50         send_friend_request(sender, receiver)
51         print()
52
53 # Call the main function
54 main()

```

В первом примере функция `send_friend_request()` принимает два параметра: отправитель и получатель. Эти параметры представляют персонажей Гарри Поттера, участвующих в запросе на добавление в друзья. Передавая соответствующие значения в функцию, вы можете имитировать отправку запроса на добавление в друзья от одного персонажа

другому.

Второй пример, `like_post()`, демонстрирует функцию, которая принимает параметры `user` и `post_id`. Параметр `user` представляет собой персонаж, которому нравится пост, а параметр `post_id` указывает идентификатор поста. Указав необходимые значения, вы можете имитировать персонажа, которому нравится конкретный пост.

Функция `comment_on_post()` принимает параметры `user`, `post_id`, и `comment`. Эти параметры позволяют указать пользователя, который комментирует, идентификатор записи, к которой сделан комментарий, и содержание комментария. Передавая соответствующие значения в функцию, вы можете имитировать персонажа, комментирующего конкретный пост конкретным комментарием.

Наконец, функция `main()` использует цикл `for` для многократного повторения действий. В каждой итерации функция случайным образом выбирает отправителя и получателя из списка пользователей, сообщение из списка сообщений, идентификатор из списка идентификаторов и комментарий из списка комментариев. Затем она вызывает соответствующие функции (`send_friend_request()`, `like_post()` и `comment_on_post()`) со случайно выбранными значениями.

Повторяя этот процесс, код имитирует отправку запросов на добавление в друзья, лайки сообщений и комментирование сообщений случайными пользователями, сообщениями, идентификаторами и комментариями. Это демонстрирует, как параметры позволяют вам предоставлять различные значения функциям, делая их адаптируемыми и повторно используемыми в различных сценариях.

----- Post ID: 789 -----
It is our choices, Harry, that show what we truly are, far more than our abilities.

Ron Weasley liked post with ID 789
Luna Lovegood commented on post with ID 789: Such wisdom in those words.
[Ron Weasley] has sent a friend request to [Ron Weasley]

----- Post ID: 123 -----
I solemnly swear that I am up to no good.

Luna Lovegood liked post with ID 123
Luna Lovegood commented on post with ID 123: Such wisdom in those words.
[Luna Lovegood] has sent a friend request to [Hermione Granger]

----- Post ID: 456 -----
Happiness can be found, even in the darkest of times, if one only remembers to turn on the light.

Hermione Granger liked post with ID 456
Luna Lovegood commented on post with ID 456: Such wisdom in those words.
[Hermione Granger] has sent a friend request to [Ron Weasley]

----- Post ID: 101112 -----
We've all got both light and dark inside us. What matters is the part we choose to act on.

Luna Lovegood liked post with ID 101112
Harry Potter commented on post with ID 101112: I'm glad these quotes resonate with everyone.
[Luna Lovegood] has sent a friend request to [Hermione Granger]

----- Post ID: 123 -----
I solemnly swear that I am up to no good.

Luna Lovegood liked post with ID 123
Hermione Granger commented on post with ID 123: That's a brilliant quote!
[Luna Lovegood] has sent a friend request to [Harry Potter]

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/mtu2uha9>

Лямбда-функции: Вот несколько примеров, иллюстрирующих эту концепцию.

```

1 # Example 1: Sorting friends by name length
2 friends = ["Harry Potter", "Hermione Granger", "Ron Weasley", "Draco Malfoy"]
3
4 # Sort the friends list based on the length of their names using a lambda function
5 sorted_friends = sorted(friends, key=lambda x: len(x))
6
7 # Print the sorted friends list
8 print("\n=== Sorted Friends by Name Length ===")
9 for friend in sorted_friends:
10     print(f"{friend} ({len(friend)})")
11
12 # Example 2: Filtering friends with names starting with 'H'
13 friends = ["Harry Potter", "Hermione Granger", "Ron Weasley", "Draco Malfoy"]
14
15 # Filter the friends list to names starting with 'H' using a lambda function
16 filtered_friends = list(filter(lambda x: x.startswith('H'), friends))
17
18 # Print the filtered friends list
19 print("\n=== Friends with Names Starting with 'H' ===")
20 for friend in filtered_friends:
21     print(friend)

```

Example 1 демонстрирует использование лямбда-функции в качестве ключевого параметра в функции `sorted()`. Лямбда-функция `lambda x: len(x)` принимает имя друга в качестве входных данных `x` и возвращает длину имени, используя `len(x)`. Эта лямбда-функция служит основой для сортировки списка друзей по длине их имен.

Example 2 демонстрирует использование лямбда-функции в качестве условия фильтрации в функции `filter()`. Лямбда-функция `lambda x: x.startswith('H')` берет имя друга в качестве входных данных `x` и возвращает логическое значение, указывающее, начинается ли имя с 'H', используя `startswith('H')`. Эта лямбда-функция используется для фильтрации списка друзей и включения только тех друзей, имена которых начинаются с 'H'.

```

=== Sorted Friends by Name Length ===
Ron Weasley (11)
Harry Potter (12)
Draco Malfoy (12)
Hermione Granger (16)

=== Friends with Names Starting with 'H' ===
Harry Potter
Hermione Granger

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/3jkk5meu>

Лямбда-функции — это компактные встроенные функции, которые полезны для простых однострочных операций. Они обеспечивают удобный способ определения функций на лету без необходимости использования формального оператора `def`.

6.3 Классы и объекты

Хорошо. Итак, давайте продолжим примеры кода, чтобы увидеть классы в действии! Рассмотрим возможность использования классических фильмов и информации о персонажах, чтобы проиллюстрировать элементы, которые мы перечислили ранее, включая случайные функции.⁹⁷

Конечно! Ниже приведены несколько примеров кода, демонстрирующих использование классов в контексте викторины по фильмам.

- **Могучие шаблоны:** Классы подобны мощным шаблонам для создания объектов с предопределенными структурами. Они определяют общую структуру и поведение объектов. Как только вы определили класс, вы можете использовать его для создания столько экземпляров или объектов из этого класса, сколько вам нужно. Каждый объект будет иметь свой собственный набор данных и будет вести себя в соответствии с шаблоном, определенным классом.

Чтобы создать класс в Python, используйте ключевое слово `class`, за которым следует имя класса и двоеточие. Чтобы определить методы внутри класса, используйте ключевое слово `def`, за которым следует имя метода, круглые скобки и любые необходимые параметры. Первым параметром каждого метода должен быть `self`, в котором хранится ссылка на фактический экземпляр, выполняющий метод. Отступ указывает на то, что методы являются частью определения класса.

Итак, без лишних слов, давайте приступим к созданию мощного шаблона для кинофраншизы!


```

1 class MovieFranchise:
2     # Setter method
3     def set_franchise(self, name, movies):
4         self.name = name
5         if movies > 0:
6             self.movies = movies
7         else:
8             print(f"Warning: Number of movies for '{name}' franchise must be
9 greater than 0.")
10            self.movies = 1
11
12     # Getter method
13     def get_franchise(self):
14         return self.name, self.movies
15
16 def main():
17     # Create movie franchises
18     harry_potter = MovieFranchise()
19     harry_potter.set_franchise("Harry Potter", 8)
20
21     star_wars = MovieFranchise()
22     star_wars.set_franchise("Star Wars", 9)
23
24     avatar = MovieFranchise()
25     avatar.set_franchise("Avatar", -2)
26
27     # Display movie franchises
28     print("\n----- Movie Franchises -----")
29     print(f"{'Name':^14}\t|{'Movies':^10}")
30     print("-----")
31     franchises = [harry_potter, star_wars, avatar]
32     for franchise in franchises:
33         name, movies = franchise.get_franchise()
34         print(f"{name:14}\t|{movies:^10}")
35
36 # Call the main function
37 main()

```

В этом примере мы определяем класс `MovieFranchise` с методами для отправки и получения имени и количества фильмов во франшизе. Для этой цели определены методы *сеттеров* и *геттеров*. Они обеспечивают контролируемый доступ к атрибутам объекта, инкапсулируя их внутри класса.

Сеттеры (или методы сеттеров) используются для установки значения атрибута. Они позволяют проверить ввод или выполнить дополнительные операции перед присвоением значения атрибуту. В коде `set_franchise()` — это метод передачи, который

устанавливает значения атрибутов наименования и фильмов класса `MovieFranchise` соответственно.

Геттеры (или методы геттеров) используются для извлечения значения атрибута. Они обеспечивают доступ к значению атрибута извне класса. В коде `get_franchise()` — это методы-получатели, которые возвращают значения атрибутов наименований и фильмов класса `MovieFranchise` соответственно.

Функция `main ()` создает три объекта франшизы и присваивает данные каждому из них. Если во франшизе отрицательное количество фильмов, оно автоматически исправляется. После этого она отображает франшизы на консоли в виде таблицы с разделителями. Таблица отформатирована с использованием пробелов для правильного выравнивания столбцов.

Обратите внимание на то, что в цикле `for` приведенного выше кода оператор `franchise.get_franchise()` извлекает и отображает соответствующие данные для каждой франшизы. Именно в этом смысле каждый объект ведет себя по шаблону, но каждый по-своему.

```
Warning: Number of movies for 'Avatar' franchise must be greater than 0.

----- Movie Franchises -----
      Name      | Movies
-----
Harry Potter   |      8
Star Wars      |      9
Avatar         |      1
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/4ab5dd89>

- **Конструкторы:** Конструкторы — это специальные методы, которые автоматически вызываются при создании объекта. Они инициализируют атрибуты объекта или выполняют любую необходимую настройку. Давайте создадим класс для персонажей с именем `Character` с конструктором для инициализации имени и фильма, в котором они снимались.

```

1 import random
2
3 class Character:
4     # The constructor method
5     def __init__(self, name, movie):
6         self.name = name
7         self.movie = movie
8
9     # A method to introduce the Character to the user
10    def introduce(self):
11        greetings = ["Hi, there!", "What's up?", "Hey!", "Yo!", "Howdy!", "Hello!"]
12        greeting = random.choice(greetings)
13        print(f"{greeting} I'm {self.name} from the movie {self.movie}!")
14
15 def main():
16     # Creating many instances of Character
17     dorothy = Character("Dorothy", "The Wizard of Oz")
18     james = Character("James Bond", "007 Series")
19     sarah = Character("Sarah Connor", "Terminator")
20     indy = Character("Indiana Jones", "Indiana Jones")
21     leia = Character("Princess Leia", "Star Wars")
22
23     # Show how all instances can introduce themselves
24     characters = [dorothy, james, sarah, indy, leia]
25     for character in characters:
26         character.introduce()
27
28     # Call the main function
29     main()
30

```

Метод `__init__` используется для определения конструктора, который получают в качестве параметров `self`, `name` и `movie`. В конструкторе атрибутам `name` и `movie` присваиваются значения, переданные в качестве параметров, с использованием зарезервированного ключевого слова `self`, которое ссылается на создаваемый объект. Это гарантирует, что каждый экземпляр класса `Character` будет иметь собственное уникальное имя и `name` и фильм `movie`. Общепринятой практикой является использование одних и тех же имен для параметров и атрибутов, хотя они могут называться по-разному.

Кроме того, класс реализует метод `introduce()`, который случайным образом выбирает другое приветствие из списка `greetings` и использует его для представления персонажа. Это динамичное поведение вносит элемент неожиданности и живости во взаимодействие персонажа с пользователем.

Кроме того, создание объектов персонажей выполняется внутри функции `main ()`.

Каждая переменная объекта определена, а атрибуты передаются в качестве параметров при вызове конструктора. Впоследствии создается список персонажей, а цикл используется для их представления:

```
Howdy! I'm Dorothy from the movie The Wizard of Oz!  
Hi, there! I'm James Bond from the movie 007 Series!  
What's up? I'm Sarah Connor from the movie Terminator!  
Hey! I'm Indiana Jones from the movie Indiana Jones!  
Hello! I'm Princess Leia from the movie Star Wars!
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/mtSf brzw>

► **Атрибуты:** Атрибуты — это характеристики или свойства объекта. Они содержат данные, определяющие состояние объекта в определенный момент выполнения программы. С вычислительной точки зрения атрибуты можно рассматривать как переменные, связанные с объектом. Например, давайте определим атрибуты для класса Actor:

```
1 class Actor:  
2     def __init__(self, name, country, oscar):  
3         # Non-mutable attributes  
4         self.name = name  
5         self.country = country  
6         # Mutable attribute  
7         self.oscar = oscar  
8  
9     def win_oscar(self):  
10        if not self.oscar:  
11            self.oscar = True  
12            print(f"{self.name} has won an Oscar!")  
13        else:  
14            print(f"{self.name} already has an Oscar.")  
15  
16    def __str__(self):  
17        return f"{self.name:20}\t{self.country:10}\t{'Yes' if self.oscar else 'No'}  
18        "
```

В этом коде атрибуты `name` и `country` считаются неизменяемыми, потому что нет смысла менять имя или страну актера после того, как они были установлены в конструкторе. Эти атрибуты задаются один раз при создании объекта и остаются постоянными в течение всего времени существования объекта.

С другой стороны, атрибут `oscar` является изменяемым, поскольку он представляет статус премии Оскар актера, который можно изменить с `False` на `True`, когда актер получает премию Оскар. Метод `win_oscar()` позволяет изменить этот атрибут, но его нельзя изменить обратно на `False`, поскольку актер не может быть лишен своей награды после того, как она была ему вручена.

Наконец, обратите внимание, что этот класс перезаписывает встроенный метод `__str__()`, чтобы предоставить строковое представление объекта `Actor`, которое подходит для вывода в табличном формате. Метод `__str__()` вызывается, когда объект необходимо преобразовать в строку, например, при печати или использовании форматирования строки. Определив этот метод, мы можем настроить выходной формат объекта `Actor`, в данном случае для отображения имени, страны и статуса премии Оскар в отформатированной табличной структуре.

Итак, теперь мы можем написать функцию `main()` для создания списка актеров, отображения их информации в таблице, обновления статуса премии Оскар для некоторых актеров и вывода обновленной таблицы:

```
1 def main():
2     actors = [
3         Actor("Meryl Streep", "United States", True),
4         Actor("Daniel Day-Lewis", "United Kingdom", True),
5         Actor("Cate Blanchett", "Australia", True),
6         Actor("Leonardo DiCaprio", "United States", False),
7         Actor("Anthony Hopkins", "United Kingdom", True),
8         Actor("Kate Winslet", "United Kingdom", True),
9         Actor("Denzel Washington", "United States", True),
10        Actor("Julianne Moore", "United States", True),
11        Actor("Tom Hanks", "United States", False),
12        Actor("Natalie Portman", "Israel", False),
13    ]
14
15    print("\n===== Actors' Information =====")
16    print(f"{'Name':20}\t{'Country':10}\t{'Oscar?'}")
17    print("="*45)
18    for actor in actors:
19        print(actor)
20
21    print("\n===== Updating Oscar Status =====")
22    actors[3].win_oscar()
23    actors[4].win_oscar()
24    actors[8].win_oscar()
25
26
```

```

27 print("\n===== Updated Actors' Information =====")
28 print("Name\t\tCountry\t\tOscar?")
29 print("="*45)
30 for actor in actors:
31     print(actor)
32

```

```

===== Actors' Information =====
Name          Country    Oscar?
=====
Meryl Streep  United States  Yes
Daniel Day-Lewis  United Kingdom Yes
Cate Blanchett  Australia    Yes
Leonardo DiCaprio  United States  No
Anthony Hopkins  United Kingdom Yes
Kate Winslet     United Kingdom Yes
Denzel Washington  United States  Yes
Julianne Moore   United States  Yes
Tom Hanks        United States  No
Natalie Portman  Israel        No

===== Updating Oscar Status =====
Leonardo DiCaprio has won an Oscar!
Anthony Hopkins already has an Oscar.
Tom Hanks has won an Oscar!

===== Updated Actors' Information =====
Name          Country    Oscar?
=====
Meryl Streep  United States  Yes
Daniel Day-Lewis  United Kingdom Yes
Cate Blanchett  Australia    Yes
Leonardo DiCaprio  United States  Yes
Anthony Hopkins  United Kingdom Yes
Kate Winslet     United Kingdom Yes
Denzel Washington  United States  Yes
Julianne Moore   United States  Yes
Tom Hanks        United States  Yes
Natalie Portman  Israel        No

```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/j4fnxc5m>

- **Методы:** Методы — это функции, определенные в классе, которые выполняют определенные действия или операции с объектами, используя свои собственные атрибуты или внешние данные, предоставленные другими объектами или функциями через входные параметры. Они также могут возвращать данные в результате своего выполнения. Говоря языком программистов, методы объектно-ориентированного программирования инкапсулируют алгоритмическую логику, которая определяет поведение объекта.

Чтобы проиллюстрировать эту концепцию, давайте создадим класс с именем **Movie**, который включает методы для обработки его атрибутов и предоставления полезной информации пользователю. Эти методы выполняют операции над атрибутами объекта **Movie** для извлечения соответствующей информации. Например, мы можем определить его продолжительность, получить список премий Оскар, присужденных фильму, и определить век, в котором он был снят.

```
1 class Movie:
2     # Movie constructor requires name, year, length, and optional list of oscars
3     def __init__(self, name, year, length, oscars=None):
4         self.name = name
5         self.year = year
6         self.length = length
7         self.oscars = oscars if oscars else [] # Set empty list if oscars==None
8
9     # Add an Oscar award to the movie. Non-return method with confirmation message
10    def add_oscar(self, award):
11        self.oscars.append(award)
12        print(f"Added {award} award to {self.name}")
13
14    # Return the list of Oscars as a comma-separated string
15    def get_oscar_list(self):
16        return ", ".join(self.oscars)
17
18    # Return the duration category: "Normal", "Long", or "Really long"
19    def get_duration(self):
20        if self.length < 90:
21            return "Normal" # < 90 minutes
22        elif self.length < 180:
23            return "Long" # 90-180 minutes
24        else:
25            return "Really long" # > 180 minutes
26
27    # Calculate the century based on the year and return as formatted string
28    def get_century(self):
29        century = (self.year // 100) + 1
30        return f"{century}th century"
```

```

31
32 # Return formatted string representation of the movie
33 def __str__(self):
34     return f"{self.name:<20}\t{self.get_century():<10}\t{self.get_duration()
35     :<15}\t{self.get_oscar_list()}"

```

Конструктор `__init__` принимает такие параметры, как имя `name`, год `year`, длительность `length` (в минутах) и список премий Оскар `oscars`. Параметр `oscars` по умолчанию принимает значение `None`, что позволяет опустить его при создании объекта. Если для `oscars` не указано значение, он принимает значение в виде пустого списка, что гарантирует правильную инициализацию объекта с соответствующими значениями, даже если список премий Оскар не указан во время создания объекта.

С другой стороны, метод `add_oscar` добавляет к фильму премию «Оскар», внося ее в атрибут списка `oscars`. Он также выводит подтверждающее сообщение, чтобы показать, что награда была добавлена.

Кроме того, класс предоставляет несколько методов получения, которые работают с атрибутами объекта `Movie` и возвращают новую информацию. Метод `get_oscar_list` извлекает список премий Оскар в виде строки, разделенной запятыми, путем слияния элементов атрибутов списка `oscars`. Это позволяет легко получить доступ к наградам, которые получил фильм.

Метод `get_duration` определяет категорию продолжительности фильма на основе его атрибута длины. Он классифицирует фильмы как "Normal" («Обычные»), если их продолжительность составляет менее 90 минут, "Long" («Длинные»), если они делятся от 90 до 180 минут, и "Really long" («Очень длинные»), если они превышают 180 минут. Это дает краткое описание продолжительности фильма.

Метод `get_century` вычисляет век, в котором был снят фильм, на основе его атрибута `year`. Он делит значение `year` на 100, добавляет к результату 1 и возвращает век в виде форматированной строки. Это позволяет пользователям быстро определить период времени, к которому относится фильм.

Наконец, метод `__str__` предоставляет форматированное строковое представление объекта `Movie`. Он возвращает строку, содержащую название фильма, год, категорию продолжительности и список премий Оскар. Это строковое представление позволяет легко вывести на печать или отобразить информацию о фильме.

Таким образом, класс `Movie` предоставляет удобные методы для взаимодействия с атрибутами объекта `Movie` и получения дополнительной информации из этих атрибутов. Теперь мы можем написать функцию `main()`, чтобы использовать этот класс для создания нескольких объектов `Movie` и отображения некоторой полезной информации о них.


```

1 def main():
2
3     # The lists of Oscars are just illustrative, not the real ones
4     movies = [
5         Movie("Titanic", 1997, 194, ["Best Picture", "Best Actor"]),
6         Movie("The Lord of the Rings", 2003, 201, ["Best Picture"]),
7         Movie("Ben-Hur", 1959, 212, ["Best Music", "Best Actress"]),
8         Movie("Gravity", 2013, 89, ["Best Actor", "Best Music"]),
9         Movie("Gone with the Wind", 1939, 238, ["Best Picture", "Best Music"]),
10        Movie("The Shape of Water", 2017, 123),
11    ]
12
13    print("Most Awarded Movie List:")
14    print("-"*80)
15    print("Name\t\t\tPeriod\t\tDuration\t\tOscars")
16    print("-"*80)
17    for movie in movies:
18        print(movie)
19
20    print("\nModifying Awards:")
21    movies[5].add_oscar("Best Actor")
22    movies[5].add_oscar("Best Actress")
23
24    print("\nUpdated Movie List:")
25    print("-"*80)
26    print("Name\t\t\tPeriod\t\tDuration\t\tOscars")
27    print("-"*80)
28    for movie in movies:
29        print(movie)
30
31    # Call the main function
32    main()

```

Функция `main ()` создает список объектов `Movie`, каждый из которых представляет отдельный фильм с такими атрибутами, как название, год, продолжительность (в минутах) и иллюстративный список полученных премий Оскар. В списке картин фильм "The Shape of Water" инициализируется без начального списка премий Оскар, поэтому по умолчанию он использует в конструкторе значение, которое является пустым списком.

Затем код выводит таблицу фильмов с наибольшим количеством наград. Он перебирает список фильмов и вызывает метод `__str__` каждого объекта `Movie`, чтобы получить строковое представление в формате столбца. Затем код изменяет награды за "The Shape of Water", вызывая метод `add_oscar()` для соответствующего

объекта **Movie**. Два новых «Оскара»: "Best Actor" («Лучшая мужская роль») и "Best Actress" («Лучшая женская роль») добавлены в список «Оскаров».

Наконец, распечатывается обновленный список фильмов, показывающий изменения, внесенные в перечень премий Оскар, врученных за "The Shape of Water", и остальные фильмы.

Ниже приведен результат выполнения этой программы:

```
Most Awarded Movie List:
-----
Name          Period      Duration    Oscars
-----
Titanic                20th century  Really long  Best Picture, Best Actor
The Lord of the Rings  21th century  Really long  Best Picture
Ben-Hur                20th century  Really long  Best Music, Best Actress
Gravity                21th century  Normal      Best Actor, Best Music
Gone with the Wind     20th century  Really long  Best Picture, Best Music
The Shape of Water     21th century  Long

Modifying Awards:
Added Best Actor award to The Shape of Water
Added Best Actress award to The Shape of Water

Updated Movie List:
-----
Name          Period      Duration    Oscars
-----
Titanic                20th century  Really long  Best Picture, Best Actor
The Lord of the Rings  21th century  Really long  Best Picture
Ben-Hur                20th century  Really long  Best Music, Best Actress
Gravity                21th century  Normal      Best Actor, Best Music
Gone with the Wind     20th century  Really long  Best Picture, Best Music
The Shape of Water     21th century  Long        Best Actor, Best Actress
```

Визуализируйте пошаговое выполнение этого кода на PythonTutor: <https://tinyurl.com/3pse4vrt>

Эти примеры демонстрируют использование классов в качестве шаблонов с атрибутами, методами и конструкторами в контексте классических фильмов. Используя мощь классов и объектно-ориентированного программирования, вы сможете раскрыть свой творческий потенциал и отправиться в захватывающее путешествие в мир, полный возможностей!

6.4 Время поработать руками!

Предложи забавные и увлекательные упражнения по программированию для отработки функций, возвращаемых значений, параметров и лямбда-выражений. Каждое упражнение должно сопровождаться постановкой задачи и примерами ожидаемых результатов. Ты можешь включить элементы математики или случайности, требуя импорта соответствующих библиотек для решения задач.⁹⁸

1. **Симулятор игральные костей:** Создайте функцию, имитирующую бросание игральные костей. Функция должна принимать количество бросков костей в качестве параметра и возвращать список результатов. Пример:

```
Input: 5  
Output: [3, 6, 1, 4, 2]
```

2. **Последовательность Фибоначчи:** Напишите функцию, которая генерирует последовательность Фибоначчи до заданного числа членов. Функция должна принимать число членов в качестве параметра и возвращать список последовательности. Пример:

```
Input: 8  
Output: [0, 1, 1, 2, 3, 5, 8, 13]
```

3. **Проверка простых чисел:** Реализуйте функцию, которая проверяет, является ли заданное число простым. Функция должна принимать число в качестве параметра и возвращать True, если оно простое, и False в противном случае. Пример:

```
Input: 17  
Output: True
```

4. **Проверка палиндрома:** Создайте функцию, которая определяет, является ли данное слово палиндромом. Функция должна принимать слово в качестве параметра и возвращать True, если это палиндром, и False в противном случае. Пример:

```
Input: "radar"  
Output: True
```

5. **Калькулятор площади круга:** Напишите функцию, которая вычисляет площадь круга для заданного радиуса. Функция должна принимать радиус в качестве параметра и возвращать рассчитанную площадь. Пример:

```
Input: 5
Output: 78.54
```

6. **Сумма квадратов:** Создайте лямбда-выражение, которое вычисляет сумму квадратов списка чисел. Лямбда должна принимать список чисел в качестве параметра и возвращать сумму их квадратов. Пример:

```
Input: [1, 2, 3, 4, 5]
Output: 55
```

7. **Счетчик частоты слов:** Напишите функцию, которая подсчитывает частоту появления каждого слова в заданном предложении независимо от регистра. Функция должна принимать предложение в качестве параметра и возвращать словарь, где ключами являются слова, а значениями являются частоты. Реализуйте функцию меню, которая предлагает пользователю ввести предложение и вызывает функцию счетчика частоты слов. Он должен отобразить результирующий словарь частот слов. Создайте функцию `main`, которая вызывает функцию меню в цикле, позволяя пользователю запускать счетчик частоты слов несколько раз, пока он не решит выйти из программы.

Пример вызова функции:

```
Input: "The cat in the hat"
Output: {'the': 2, 'cat': 1, 'in': 1, 'hat': 1}
```

8. **Обратная строка:** Реализуйте функцию, которая обращает заданную строку. Функция должна принимать строку в качестве параметра и возвращать обратную строку. Создайте функцию меню, которая предлагает пользователю ввести строку и вызывает функцию обращения строки. Она должна отображать обращенную строку. Напишите функцию `main`, которая вызывает функцию меню в цикле, позволяя пользователю переворачивать строки несколько раз, пока он не решит выйти из программы.

Пример вызова функции:

```
Input: "Hello, world!"
Output: "!dlrow ,olleH"
```

9. **Уникальные элементы:** Создайте функцию, которая принимает список чисел в качестве входных данных и возвращает новый список, содержащий только уникальные элементы исходного списка, следующие в том же порядке. Разработайте функцию меню, которая предлагает пользователю ввести список чисел и вызывает функцию уникальных элементов. Она должна отображать результирующий список уникальных элементов. Разработайте основную функцию, которая вызывает

функцию меню в цикле, позволяя пользователю находить уникальные элементы из разных списков чисел до тех пор, пока пользователь не решит выйти из программы.

Пример вызова функции:

```
Input: [1, 2, 3, 2, 4, 1, 5]
Output: [1, 2, 3, 4, 5]
```

10. **Средние оценки:** Напишите функцию, которая вычисляет среднюю оценку учащегося по списку оценок. Функция должна принимать список оценок в качестве параметра и возвращать среднее значение в виде числа с плавающей запятой. Создайте функцию меню, которая предлагает пользователю ввести список оценок и вызывает функцию средних оценок. Она должна отображать рассчитанную среднюю оценку. Разработайте функцию **main**, позволяющую пользователю несколько раз вычислять средние оценки, пока он не решит выйти из программы.

Пример вызова функции:

```
Input: [85, 90, 92, 88, 95]
Output: 90.0
```

11. **Шифр Цезаря:** Реализуйте функцию, которая шифрует заданное сообщение, используя Шифр Цезаря. Функция должна принимать сообщение и величину сдвига в качестве параметров и возвращать зашифрованное сообщение. Создайте функцию меню, которая предлагает пользователю ввести сообщение и величину сдвига, а затем вызывает функцию Шифра Цезаря.

Она должна отображать зашифрованное сообщение. Напишите функцию **main**, которая вызывает функцию меню в цикле, позволяя пользователю шифровать сообщения, используя шифр Цезаря несколько раз, пока пользователь не решит выйти из программы.

Пример вызова функции:

```
Input: ("Hello, world!", 3)
Output: "Khoor, zruog!"
```

12. **Решение линейных уравнений:** Создайте функцию, которая решает линейное уравнение вида $ax + b = 0$. Функция должна принимать значения a и b в качестве параметров и возвращать решение x . Реализуйте функцию меню, которая предлагает пользователю ввести значения a и b и вызывает функцию решения линейного уравнения. Она должна отображать решение x . Напишите функцию **main**, которая вызывает функцию меню в цикле, позволяя пользователю решать линейные уравнения несколько раз, пока он не решит выйти из программы.

Пример вызова функции:

```
Input: (2, 4)
```

```
Output: -2.0
```

Хорошо. Теперь предложи несколько забавных и увлекательных упражнений по программированию для практики ООП, включая классы, конструкторы, атрибуты и методы. Каждое упражнение должно сопровождаться постановкой задачи и примерами ожидаемых результатов. Ты можешь включить элементы математики или случайности, требуя импорта соответствующих библиотек для решения задач.⁹⁹

1. **Калькулятор оценок учащегося:** Создайте класс под названием `student`, который представляет информацию об учащемся, включая его имя и список его оценок. Реализуйте методы для вычисления средней оценки, определите, сдал ли учащийся экзамен (средняя оценка > 60), и отобразите информацию об учащемся. Кроме того, напишите функцию `main`, которая позволяет пользователю взаимодействовать с объектами ученика через меню. Пример:

► Входные данные:

```
1 student1 = Student("John Doe", [75, 80, 90, 65, 70])
2 student1.display_info() # Display student information
3 print("Average Grade:", student1.calculate_average_grade()) # Show avg. grade
4 print("Passed:", student1.has_passed()) # Check if student passed
5
```

► Результат на выходе:

```
Student Name: John Doe
Grades: [75, 80, 90, 65, 70]
Average Grade: 76.0
Passed: False
```

► Меню:

```
1. Add Student
2. Display Student Information
3. Calculate Average Grade
4. Check Pass Status
5. Exit
```

```
Enter your choice:
```

2. **Журнал банковских транзакций:** Создайте класс `BankAccount`, представляющий банковский счет. Класс должен иметь такие атрибуты, как номер счета, имя владельца счета и баланс. Реализовать методы ввода и вывода денег со счета, а также метод

отображения истории транзакций (журнала) счета. История транзакций должна отслеживать все депозиты и снятия средств со счета. Напишите функцию `main`, которая позволяет пользователю взаимодействовать с объектом `BankAccount` посредством меню. Пример:

► Входные данные:

```
1 account1 = BankAccount("123456789", "John Doe", 1000.0)
2 account1.deposit(500.0) # Deposit $500
3 account1.withdraw(200.0) # Withdraw $200
4 account1.deposit(300.0) # Deposit $300
5 account1.display_transaction_history() # Display transaction history
```

► Результат на выходе:

```
Transaction History:
-----
Deposit:    $500.0
Withdrawal: $200.0
Deposit:    $300.0
```

► Меню:

```
1. Deposit
2. Withdraw
3. Display Transaction History
4. Exit

Enter your choice:
```

3. Система управления запасами:

Создайте класс `Product`, который представляет продукт в системе управления запасами. Класс должен иметь такие атрибуты, как идентификатор продукта, имя, количество и цена. Реализуйте методы для увеличения или уменьшения количества продукта, расчета общей стоимости продукта (количество \times цена) и отображения информации о продукте. Кроме того, напишите функцию `main`, которая позволяет пользователю взаимодействовать с объектами `Product` посредством меню. Пример:

► Входные данные:

```

1 product1 = Product("P001", "iPhone", 10, 1000.0)
2 product1.increase_quantity(5) # Increase quantity by 5
3 product1.decrease_quantity(3) # Decrease quantity by 3
4 product1.display_info() # Display product information
5
6 print("Total Value:", product1.calculate_total_value()) # Calculate and print
   total value

```

► Результат на выходе:

```

Product ID: P001
Name: iPhone
Quantity: 12
Price: $1000.0
Total Value: $12000.0

```

► Меню:

```

1. Add Product
2. Increase Quantity
3. Decrease Quantity
4. Calculate Total Value
5. Display Product Information
6. Exit

Enter your choice:

```

4. Кино-викторина: Вам поручено создать игру Movie Trivia. Игра должна проверить знание игроком популярных фильмов, задав серию вопросов с несколькими вариантами ответов. Вам нужно закодировать необходимые классы и реализовать систему меню, чтобы позволить пользователю сыграть в игру. Вот требования:

- a) Создайте класс с именем **Question**, который представляет один простой вопрос. Каждый вопрос должен иметь следующие атрибуты:
 - **text**: Собственно вопрос, который нужно задать.
 - **choices**: Список вариантов множественного выбора.
 - **correct**: Индекс правильного ответа в списке вариантов.
- b) Создайте класс с именем **TriviaGame**, который управляет ходом игры. Он должен иметь следующие методы:
 - **__init__()**: Инициализирует игру, подсчитывает очки и устанавливает вопросы (создайте набор из не менее чем 10 вопросов).
 - **show_menu()**: Отображает для игрока опции игрового меню.
 - **start_game()**: Запускает викторину.
 - **ask_question(question)**: Задает один вопрос и проверяет ответ игрока. Вопросы

должны быть выбраны случайным образом и не должны повторяться.

▶ `show_score()`: Отображает текущий счет игрока.

▶ `finish_game ()`: Завершает игру и отображает окончательный счет.

с) Реализовать систему меню, используя функцию `main()`, чтобы позволить игроку взаимодействовать с игрой. В меню должны быть следующие опции:

▶ Start game: Запуск викторины.

▶ Show last score: Отображает счет самой последней игры, сыгранной во время выполнения программы, либо сообщение "Play first!", если никто еще не играл.

▶ Quit: Выход из игры.

Пример:

```
=====
Welcome to the Movie Trivia Game
=====
1. Start game
2. Show last score
3. Quit
Enter your choice: 1
=====

Question 1:
What movie won the Academy Award for Best Picture in 2020?

A. Parasite
B. Joker
C. 1917
D. The Irishman

Enter your answer: A

Correct!
```

Question 2:

Who played the character of Iron Man in the Marvel Cinematic Universe?

- A. Robert Downey Jr.
- B. Chris Evans
- C. Chris Hemsworth
- D. Mark Ruffalo

Enter your answer: B

Incorrect!

Question 3:

In the movie "The Shawshank Redemption," what is the name of the main character?

- A. Andy Dufresne
- B. Red Redding
- C. Ellis Boyd "Red" Redding
- D. Warden Samuel Norton

Enter your answer: C

Correct!

Your score: 2 out of 3

Thank you for playing the Movie Trivia Game!

=====

Welcome to the Movie Trivia Game

=====

- 1. Start game
- 2. Show last score
- 3. Quit

Enter your choice:

=====

7.1 Копаем дальше

До сих пор это было захватывающее путешествие. Но теперь, когда я усвоил основы программирования на Python (типы данных, переменные, консольный ввод/вывод данных, операторы, условные операторы, циклы, списки, кортежи, строки, словари, функции, классы и объекты), куда мне двигаться дальше? Не могли бы вы направить меня в правильном направлении? Предоставь несколько планов уроков для дальнейшего изучения различных направлений, чтобы я мог стать супер-специалистом по Python. (используй формат ментальной карты, такой как PlantUML).¹⁰⁰

Хорошо, мой вундеркинд от Python! Теперь, когда вы как босс освоили основы программирования на Python, пришло время подняться на ступеньку выше и перейти на новый уровень владения Python! Подготовьтесь к дальнейшему волнующему путешествию, так как я предлагаю вам умопомрачительные планы уроков, которые будут углубляться в конкретные направления обучения Python. Приготовьтесь погрузиться в мир веб-разработки, познать секреты науки о данных, раскрыть свои способности в области автоматизации и написания сценариев, погрузиться в мир разработки игр и в магию DevOps, разгадать тайны продвинутых концепций Python, освоить искусство тестирования и отладки, а также доступ к сокровищнице дополнительных ресурсов. С этими уроками за плечами вы станете виртуозом Python, готовым поразить мир своим волшебством программирования! Поехали, рок-звезда Python!

1. Веб-разработка на Python:

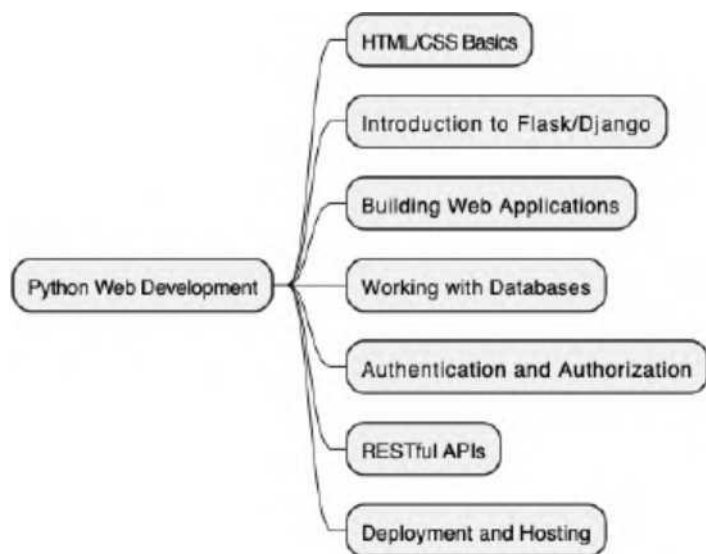


Рис. 7.1: Предлагаемый курс обучения веб-разработке на Python.

2. Наука данных на Python:

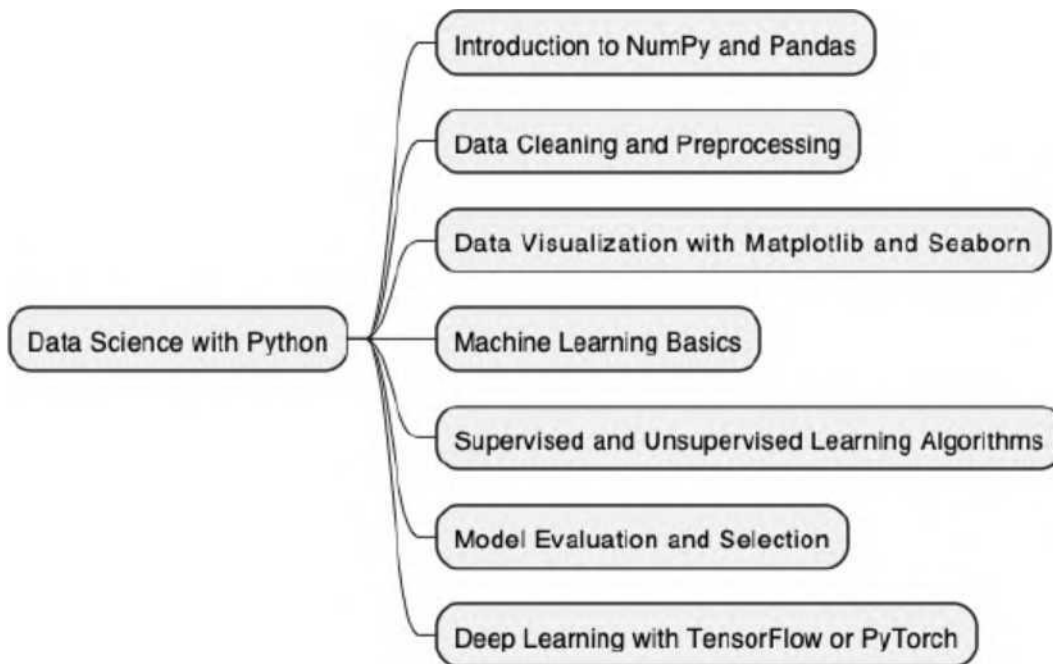


Рис. 7.2: Предлагаемый курс обучения Data Science на Python.

3. Python для автоматизации и сценариев:

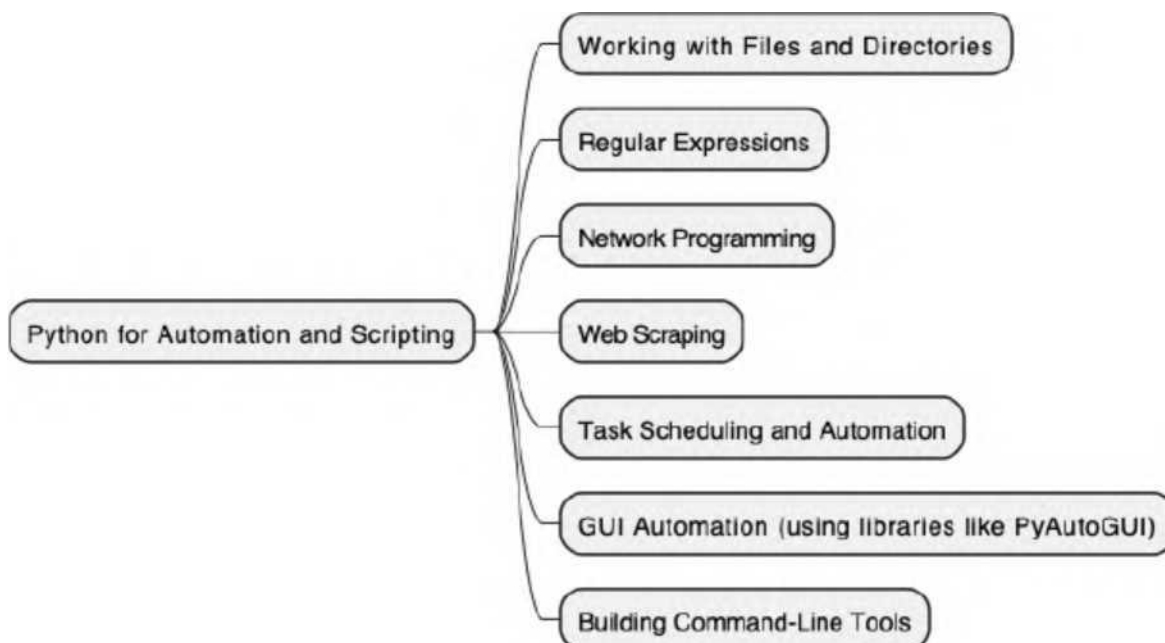


Рис. 7.3: Предлагаемый курс обучения Python по автоматизации и созданию сценариев.

4. Python для разработки игр:

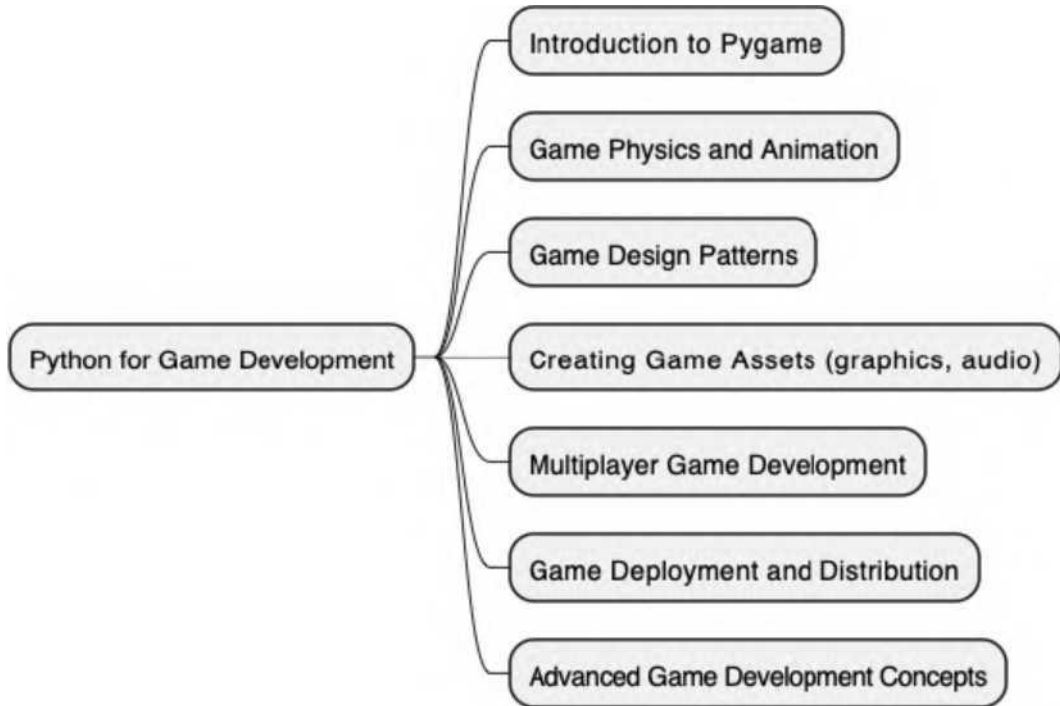


Рис. 7.4: Предлагаемый курс обучения Python для разработки игр.

5. Python для DevOps:

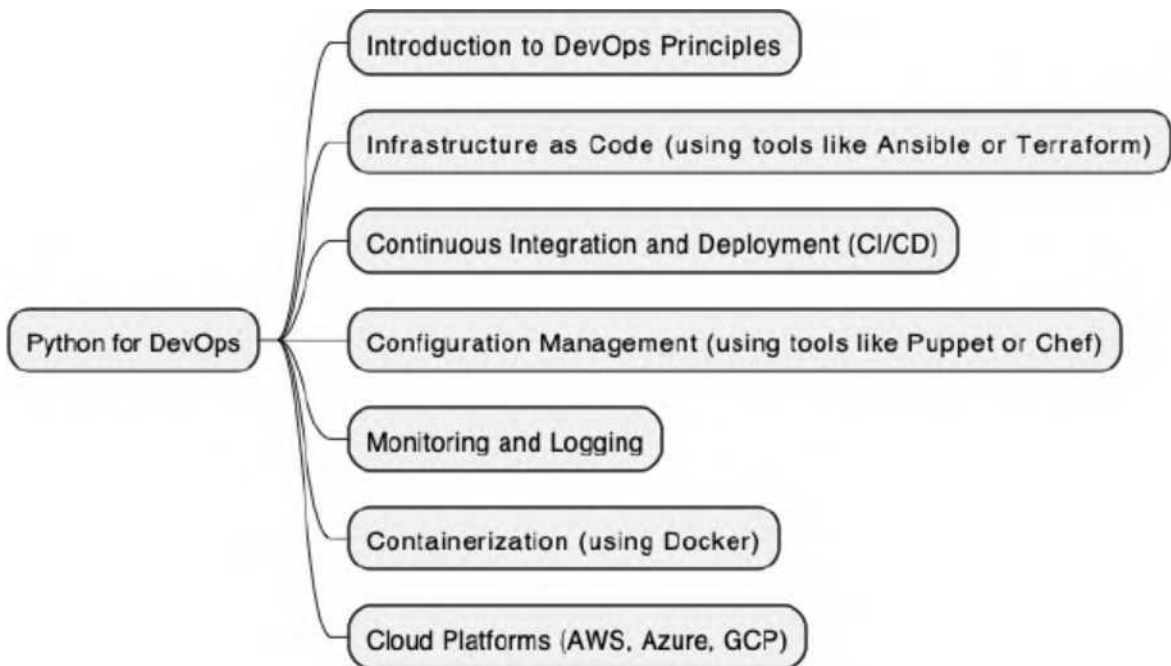


Рис. 7.5: Предлагаемый курс обучения Python для DevOps.

6. Расширенные концепции Python:

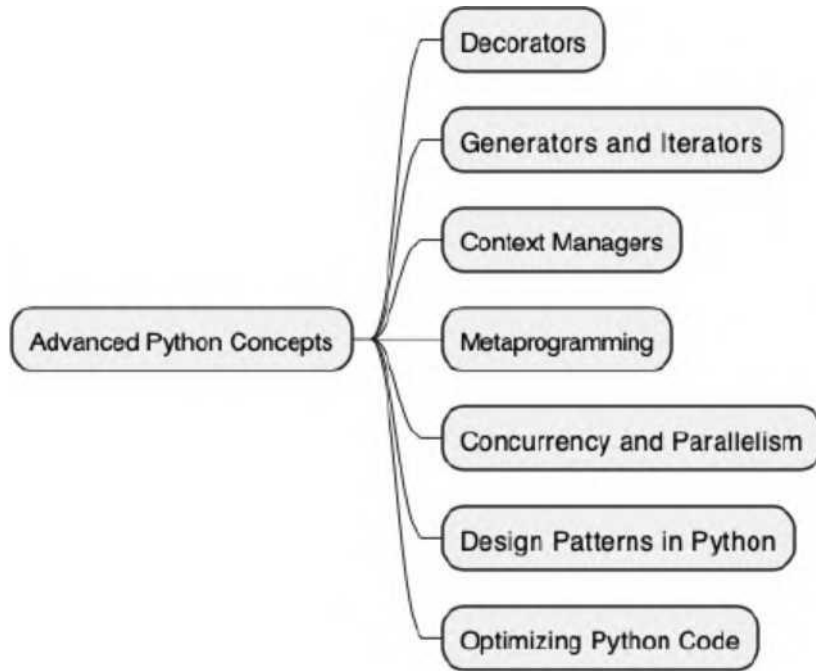


Рис. 7.6: Предлагаемый курс обучения по расширенным концепциям Python.

7. Тестирование и отладка на Python:

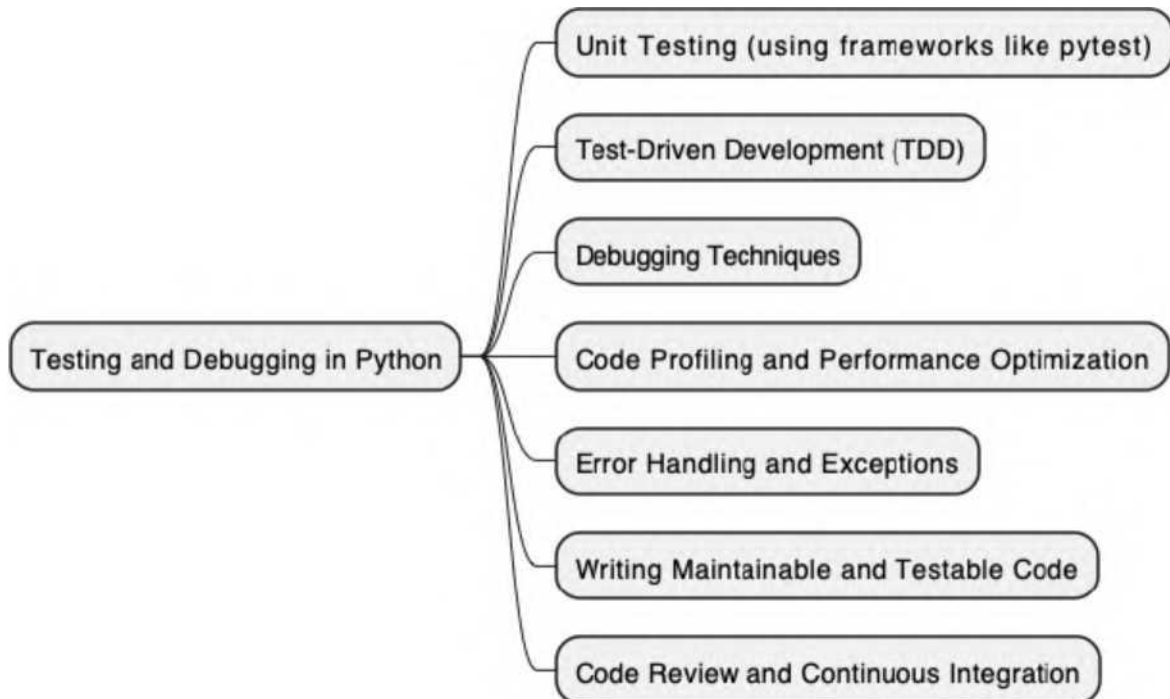


Рис. 7.7: Предлагаемый курс обучения по тестированию и отладке на Python.

8. Дополнительные ресурсы:

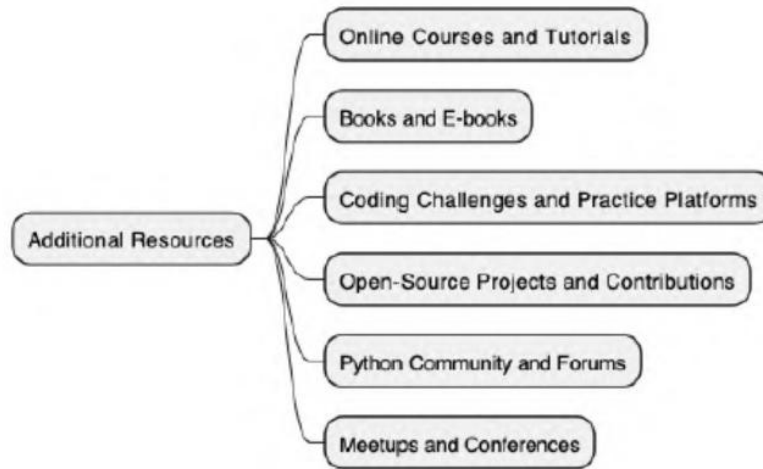


Рис. 7.8: Предлагаемый курс обучения по дополнительным ресурсам Python.

NB. К вашему сведению, вывод, сгенерированный ChatGPT на предыдущий запрос, показан ниже. Затем этот вывод был импортирован в draw.io в качестве данных ввода PlantUML для создания диаграмм ментальных карт.

@startmindmap

- 1 Python Learning Tracks
- 2 * Python Web Development
- 3 ** HTML/CSS Basics
- 4 ** Introduction to Flask/Django
- 5 ** Building Web Applications
- 6 ** Working with Databases
- 7 ** Authentication and Authorization
- 8 ** RESTful APIs
- 9 ** Deployment and Hosting

- 10 Data Science with Python
- 11 * Introduction to NumPy and Pandas
- 12 * Data Cleaning and Preprocessing
- 13 * Data Visualization with Matplotlib and Seaborn
- 14 * Machine Learning Basics
- 15 * Supervised and Unsupervised Learning Algorithms
- 16 * Model Evaluation and Selection
- 17 * Deep Learning with TensorFlow or PyTorch

- 18 Python for Automation and Scripting
- 19 * Working with Files and Directories
- 20 * Regular Expressions
- 21 * Network Programming
- 22 * Web Scraping
- 23 * Task Scheduling and Automation
- 24 * GUI Automation (using libraries like PyAutoGUI)
- 25 * Building Command-Line Tools

- 26 Python for Game Development
 - 27 * Introduction to Pygame
 - 28 * Game Physics and Animation
 - 29 * Game Design Patterns
 - 30 * Creating Game Assets (graphics, audio)
 - 31 * Multiplayer Game Development
 - 32 * Game Deployment and Distribution
 - 33 * Advanced Game Development Concepts

 - 34 Python for DevOps
 - 35 * Introduction to DevOps Principles
 - 36 * Infrastructure as Code (using tools like Ansible or Terraform)
 - 37 * Continuous Integration and Deployment (CI/CD)
 - 38 * Configuration Management (using tools like Puppet or Chef)
 - 39 * Monitoring and Logging
 - 40 * Containerization (using Docker)
 - 41 * Cloud Platforms (AWS, Azure, GCP)

 - 42 Advanced Python Concepts
 - 43 * Decorators
 - 44 * Generators and Iterators
 - 45 * Context Managers
 - 46 * Metaprogramming
 - 47 * Concurrency and Parallelism
 - 48 * Design Patterns in Python
 - 49 * Optimizing Python Code

 - 50 Testing and Debugging in Python
 - 51 * Unit Testing (using frameworks like pytest)
 - 52 * Test-Driven Development (TDD)
 - 53 * Debugging Techniques
 - 54 * Code Profiling and Performance Optimization
 - 55 * Error Handling and Exceptions
 - 56 * Writing Maintainable and Testable Code
 - 57 * Code Review and Continuous Integration

 - 58 Additional Resources
 - 59 * Online Courses and Tutorials
 - 60 * Books and E-books
 - 61 * Coding Challenges and Practice Platforms
 - 62 * Open-Source Projects and Contributions
 - 63 * Python Community and Forums
 - 64 * Meetups and Conferences
- @endmindmap

7.2 Предлагаемый перечень для дополнительного чтения

1. Python.org. (n.d.). Python documentation, <https://www.python.org/doc/>
2. Downey, A. (2015). Think Python: How to Think Like a Computer Scientist. O'Reilly Media, Inc. <https://greenteapress.com/wp/think-python-2e/>
3. Sweigart, A. (2019). Automate the Boring Stuff with Python: Practical Programming for Total Beginners. No Starch Press, <https://automatetheboringstuff.com/>
4. GeeksforGeeks. (2023). Python Tutorial, [https://www.geeksforgeeks.org/python-programming -](https://www.geeksforgeeks.org/python-programming/)
5. Real Python (n.d). <https://realpython.com/>
6. Parlante, N. (2022). Google's Python Class, <https://developers.google.com/edu/python/>

Конец

Список из 100 запросов

1. "ChatGPT: Your Python Coach. Mastering the Essentials in 100 Prompts" is the title of a book I'm writing to teach the fundamentals of Python. Provide a draft version of an engaging and thought-provoking Preface for it. Emphasise the impact AI, and language models in particular, are having in many areas, including programming, and how it is expected to leverage programmers' abilities and productivity to create powerful applications in the near future, mentioning in passing that the issues regarding fake content and hallucination that have been detected on these language models are alleviated in the area of programming because the generated code can be validated by simply running it. Comment also that the initial generated content has been carefully revised and enriched based on my educational expertise.
2. Explain with a colourful tone that the purpose of this book is to initiate a newcomer to the world of programming, in particular, to the Python language, and to help them learn the basic concepts with examples, explanations, and feedback, just like a teacher guides a student through their learning journey. Only that here, that teacher is ChatGPT. Emphasise the new possibilities and advantages of using modern developments in AI to assist the learning and development process of programmers.
3. Provide a witty description of the Python programming language that is suitable for a first-year college student. Make some notes on the language's history and current version, speculate on its future, and provide useful links to organisations or websites where the reader can obtain more technical information.
4. Write a short note on Monty Python.
5. Briefly describe the key differences between Python 2.x and Python 3.x versions. Highlight the latest version's advantages.
6. Create a scheme of the standard cycle of program development, using Latex smartdiagram package with the circular diagram type.
7. Explain to a first-year college reader the standard cycle of program development, including the steps of writing (coding), running, testing, and correcting (debugging) (do it with a bulleted list).
8. Clarify that Python is an interpreted computer language; explain how it differs from a compiled language (advantages and weaknesses). Mention that there are a variety of IDEs available, but in this book, rather than dealing with software installation hassles, I will rely on ready-to-use environments that meet the needs of this book, so we can focus on concepts and algorithms. Jupiter Notebooks, Colaboratory, PythonTutor, and ChatGPT will be used as tools. Make a bullet list of these tools, including a brief description, website, and the steps required to open an account if one is required.
9. Describe in detail how to create a new Jupiter Notebook in Google Colab.
10. Explain the two types of cells in Jupyter Notebook.
11. Explain in detail how to use PythonTutor.
12. Explain to a first-year college student what ChatGPT is, how it can help students learn programming, how to use it responsibly to achieve good learning goals, and whether there are any ethical concerns about using it or relying too heavily on it and how to address them.

170 LIST OF PROMPTS

13. Propose prompt templates for how a programming student can ask ChatGPT so they can get an accurate and engaging response to the following topics: 1) Introduce and explain programming concepts in an approachable manner. 2) Create code examples that are related to those concepts. 3) Describe the key lines of such code. 4) Write down documentation and comments in the code. 5) Based on the example, come up with new exercises for the student to practice in kind of variations of the solved problem. 6) Include useful links to resources that supplement the previous topics covered. 7) Make fun and witty comments to make the topic more enjoyable.
14. Explain the concept of the 'Hello, World!' program in a simple but colorful manner, without using any code. Additionally, include a relatable anecdote to provide beginners in programming with a warmful introduction to the topic.
15. But why the message "Hello World"? Who invented it? What's the meaning?
16. Write the classic "Hello, World!" program in Python and explain it. Provide the output as well. I will be using Colab.
17. Explain what a syntax error is and demonstrate 3 possible syntax errors in the 'Hello, World!' program with their respective outputs.
18. Why is it called print()? Do I need a printer to use it?
19. Explain in a simple way what a string is, and the syntax rules for defining strings in Python
20. Explain string formatting in simple terms. Write fun code examples for each of the formatting options.
21. Why are there some green lines in the code that begin with a hash symbol?
22. Provide an example of a program using only print() commands with a real-world analogy.
23. Can I use formatting options such as width and alignment to build strings? Show me engaging examples.
24. What do you mean by "floating-point number"?
25. I'd like to put what I've just learned about print() and string formatting into practice. Please provide four code exercises where I need to fix the errors to make them run correctly.
26. I'd like to put what I've just learned about print() and string formatting into practice. This time, suggest some exercises for me to write the code that can be solved using only print() and string formatting.
27. Mention in detail that one interesting feature of ChatGPT is that you can ask it to edit a piece of code, correct it, improve it, or provide feedback.
28. Create a simple prompt template for revise/improve/correct a given code.
29. Below is the code I wrote to solve the exercise: "Create a program that prints a right triangle using the '*' character.". Please revise/improve/correct my code and provide feedback on how to make it better, more efficient, or more readable. Also, demonstrate how you would have coded the solution differently, if applicable. Code: `print* ') print* ' * 2) print* ' * 3) print* ' * 4) print* ' * 5)`

30. What exactly is a program? How does it relate to data?
31. Explain with examples and outputs how data is represented in Python. Provide examples from comics characters to better understand the concept.
32. So, how can I know the type of a variable at a given point in the program?
33. How can a program get different types of input data from the user? Use examples from comic books.
34. What are those if/else statements that you used in some of the previous examples? I can understand the basic idea, but can you provide a clear and concise explanation?
35. Hi there, ChatGPT! I'd like to practice my coding skills with an exercise that involves user input, data type conversion, and printing output, maybe even including a menu of options. Could you please suggest an exercise for me to work on? Once I have the statement, I'd also appreciate it if you could show me an example of the expected output. Thanks!
36. Ok, now show me the solution.
37. Cool. My version of the solution is shown below. What are your thoughts on it? In comparison to yours?
38. But I took care to conceal the credit card number. I was just joking around.
39. By the way, I like how you commented each section of your code; it helps to clarify things. Please do the same with my code.
40. Provide four code exercises related to data type input and conversion that I need to correct for syntax or logic errors so that they can run properly.
41. Suggest some exercises for me to write code that can be solved using only user input and data type conversion operations.
42. List all of the Python operators and provide a brief description for each.
43. Explain the details of arithmetic operators in Python in a colourful tone and include a single code example to illustrate the use of all the operators. The example should compare entertaining statistics about Marvel vs X-men movies and box office.
44. What do those numbers 22.55e9 and 6.01e9 for the box_office variables mean?
45. Explain the details of comparison operators in Python in a colourful tone and include a single code example to illustrate the use of all of them, maybe using fun facts about football teams.
46. Explain the concept of logical operators in Python in a casual tone. After defining each operator, include a single code example to illustrate the usage of all of them. Use a table of fast food dishes indicating how suitable are for vegetarians and vegans, for this purpose.
47. Explain the concept of bitwise operators in Python in a friendly but professional tone. Include a code example after defining each operator to demonstrate how to use them all. Show the results of the operations, including bit sequences and integers of the operands.

172 LIST OF PROMPTS

48. Okay, but could you please explain briefly what you mean by "binary digits of an integer". Give me some amusing real-life analogies.
49. Okay, one more thing: please explain what you mean by "two's complement form representation" in a brief and friendly manner.
50. Explain the concept of arithmetic assignment operators in Python in a playful manner.
51. Explain the concept of identity operators in Python in a playful manner.
52. Ok. Two questions. First, what exactly do you mean by "memory locations"? I don't see any memory locations in the code; please elaborate in a friendly and colourful manner. Second, what does the Python special variable "None" mean? Please clarify.
53. Explain the concept of memberships operators in Python in a playful manner.
54. Explain the concept of exponentiation operator in Python in a playful manner.
55. What happens when an expression has multiple operands? In what order are they evaluated?
56. How can I program more complex math operations such as square roots, logarithms, and so on?
57. Suggest some fun and informative coding exercises for practising arithmetic, comparison, logical, bitwise, membership, exponentiation, and math operators.
58. Propose a nice and engaging coding problem to practice binary representation of integer numbers. Assume you have a table of food dishes, with columns representing the ingredients used to prepare them and a final column displaying the dish's integer code as a result of interpreting the ingredient columns as bits of a binary number (show a table with four dishes and five ingredients). The program should ask for the name of a dish and a series of yes/no questions to determine the dish's integer code, and then print out a row of the table with this information (show an example of the program's output).
59. List all of the control structures in programming and provide a brief description for each.
60. Explain everything I need to know about if/else statements in Python in a fun and creative way, with playful examples.
61. Is there anything else I should know about conditionals in Python?
62. Now let's talk about match/case statements in Python. Explain everything I need to know about them in a fun and creative way, using playful examples.
63. Ok. In the match statement, I see you used a "lower()" command. Do I have to put it every time I code a match/case statement?
64. Please provide a brief technical definition of the for loop.
65. Now let's see the for loops in action in a fun and creative way, using playful examples.
66. Please provide a brief technical definition of the while loop.

67. Now let's see the while loops in action in a fun and creative way, using playful examples.
68. In the "Counting Sheep" example, you ask the user if they are sleeping twice. Isn't that redundant? Is there a way to write the loop without asking the same question twice?
69. Is there anything else I should know about for or while loops in Python?
70. What is a nested control structure and why is it useful?
71. Let us now dive into nesting control structures in Python. Show me how to use nested conditionals allowing for multiple levels of branching, nested loops including nested double for loops, double while loops and mixed while/for or for/while loops, as well as combinations of loops within conditionals and conditional within loops. Use examples from superheroes and comics to be creative and playful in your explanation.
72. Suggest some fun and engaging coding exercises for practising if/ else, if/ elif/ else, and match/case statements, as well as the ternary operator, including problem descriptions and examples of expected output.
73. Suggest some fun and engaging coding exercises for practising for and while loops, including problem descriptions and examples of expected output.
74. Suggest some fun and engaging coding exercises to practice nested loop structures, including problem descriptions and examples of expected output.
75. Suggest the problem of writing a console-based calculator program that takes two numbers and a selected operation from a menu (decorated with a frame), computes and displays the result, and asks the user if they want to continue. Include an example of the expected output.
76. Explain what a container structure is, then list all of the container structures available in Python and provide a brief description for each, using real-life analogies and a friendly tone.
77. Explain everything I need to know about lists in Python in a fun and creative way, using playful examples.
78. Ok. Show me a code example that illustrates each of the items listed above. Include comments in each code block to explain its purpose. You are welcome to use the superhero vs. villain theme.
79. Interesting. Now demonstrate how I can use loops and conditionals on lists. This time, use creative food-related examples, taking advantage of Python's randomness features.
80. Wait a minute. In the examples above, you use some neat tricks that I'm not aware of, such as: `random.choice()`, `enumerate()`, and `zip()`. Can you pls explain these to me?
81. Explain everything I need to know about tuples in Python in a fun and creative way, using playful examples.
82. Ok. Now let's look at some code samples to see tuples in action! This time, why not use rock bands to illustrate the items you listed previously?
83. Wait! There is a weird '*' before the "members" variable in the unpacking trick example. What exactly is that?
84. Explain everything I need to know about sets in Python in a fun and creative way, using playful examples.
85. Good. Now let's look at some code samples to see sets in action! This time, let's use planets fun facts to

illustrate the items you listed previously.

86. Explain everything I need to know about strings in Python in a fun and creative way, using playful examples.
87. Good. Now let's look at some code examples to see strings in action! Consider using classic children's books to illustrate the items you listed previously.
88. Explain everything I need to know about dictionaries in Python in a fun and creative way, using playful examples.
89. Great. Now let's look at some code samples to see dictionaries in action! Let's stick to the superheroes theme to illustrate the items you listed previously.
90. What about putting together a superhero encyclopedia? Can I implement it as a dictionary of dictionaries? Some kind of nested dictionary? Use a code example to illustrate.
91. Nice! You also mentioned that dictionaries enable us to explore intricate relationships and connections between items within the dictionary. What does that mean? Can dictionaries help me determine relationships such as 'the enemies of my allies are my enemies,' 'the allies of my allies are my friends,' 'the heroes with the same superpower as mine are my buddies,' or 'display a table of superheroes sorted by debut year'? Please provide some examples.
92. Suggest some fun and engaging coding exercises for practising lists, tuples, and sets. Each exercise should be provided with a problem statement and examples of expected output.
93. Suggest some fun and engaging coding exercises for practising with strings and dictionaries, including nested dictionaries. Each exercise should be provided with a problem statement and examples of expected output.
94. Explain briefly the concept of modularisation and reusability, as well as why they are useful in programming projects. Then, using real-world analogies, explain the various approaches for implementing these notions in a friendly but neat manner.
95. Wow! That's a impressive list of powerful ideas. However, since I am still learning the ropes, please recommend one or two items from the list I should start with as a beginner, and then describe each item's key components in a relaxing and entertaining manner.
96. Ok. So let's start with some code examples to see functions in action! Consider using social networking-related ideas to illustrate the items you listed previously. Feel free to use entertaining Harry Potter trivia and random features.
97. Ok. So let's continue with some code examples to see classes in action! Consider using classic movies and characters trivia to illustrate the items you listed previously, including random features.
98. Suggest some fun and engaging coding exercises for practising functions, return values, parameters, and lambda expressions. Each exercise should be provided with a problem statement and examples of expected output. You can incorporate elements of math or randomness, requiring to import relevant libraries to solve the problems.
99. Ok. Now suggest some fun and engaging coding exercises for practising OOP, including classes, constructors,

attributes and methods. Each exercise should be provided with a problem statement and examples of expected output. You can incorporate elements of math or randomness, requiring to import relevant libraries to solve the problems.

100. It's been an exciting ride thus far! But, now that I've grasped the fundamentals of Python programming (data types, variables, console data input/output, operators, conditionals, loops, lists, tuples, strings, dictionaries, functions, classes and objects), where should I go next? Could you please guide me in the proper direction? Provide some lesson plans for digging further into various learning tracks so that I can become a Python superstar! (use a mindmap format such as PlantUML)