Python С НУЛЯ

От новичка до собственных игр и программ!

Roman Gurbanov | Jean-Loup Chrétien

Jean-Loup Chrétien

Python с нуля: от новичка до собственных игр и программ

«Автор»
2023

Chrétien J.

Python с нуля: от новичка до собственных игр и программ /
J. Chrétien — «Автор», 2023

ISBN 979-8-22385-304-6

Окунитесь в мир программирования с книгой "Python с нуля: от новичка до собственных игр и программ"!Эта книга - ваш путеводитель по изучению Python, одного из самых популярных языков программирования в мире.Напишите свою первую строчку кода уже в первой главе и продолжайте осваивать ключевые принципы программирования, от переменных до циклов. От функций до классов и объектов!А затем, отточите свое мастерство с финальным проектом - игрой, симулирующей стыковку космического корабля с космической станцией! Этот симулятор поможет закрепить Ваши навыки объектно-ориентированного программирования.В завершении этой книги, четырнадцатая глава покажет Вам варианты Вашего дальнейшего развития, и предложит конкретные дальнейшие шаги.Эта книга не просто научит Вас программировать на Python, она поможет Вам думать как программист.Будущее начинается сегодня. Не упустите свой шанс обучиться Python и открыть для себя новые возможности. "Python с нуля" ждет вас.

ISBN 979-8-22385-304-6

Содержание

ПРЕДИСЛОВИЕ 7
ВВЕДЕНИЕ 8
1. Как получить максимум от этой книги? 8
2. Куда записывать код? 9
3. Как читать код в этой книге? 10
4. Что делать с тестами из книги? 11
5. Почему именно Питон? 12
ГЛАВА ПЕРВАЯ: НАЧИНАЕМ ПРОГРАММИРОВАТЬ НА PYTHON! 13
1. Ваша первая строка кода 13
2. Что такое программа? 14
3. Функция Print 15
4. Как Python читает код? 16
5. Программа подсчета 17
6. Самостоятельная работа 18
7. Итоги первой главы 19
8. Тест первой главы 20
ГЛАВА ВТОРАЯ: ПЕРЕМЕННЫЕ 21
1. Что такое переменная? 21
2. Как создать и вывести переменную? 22
3. Итоги второй главы 24
4. Тест второй главы 25
ГЛАВА ТРЕТЬЯ: ЧИСЛА 26
1. Целые и дробные числа 26
2. Математические Операторы в Python 27
3. Работаем с числами 28
4. Делим числа без остатка в Python 29
5. Порядок вычислений в Python 30
6. Числа и переменные в Python 31
7. Итоги третьей главы 32
8. Тест третьей главы 33
ГЛАВА ЧЕТВЕРТАЯ: СТРОКИ 34
1. Строки в Python 34
2. Строки и функция печати 35
3. Хранение строк в переменных 36
4. Конкатенация строк в Python 37
5. Конкатенация строк и переменные 38
6. Форматирование строк в Python 39
7. Итоги четвертой главы 41
8. Тест четвертой главы 42
ГЛАВА ПЯТАЯ: БУЛЕВА ЛОГИКА 43
1. Операторы сравнения 44
2. Булевы значения: правда или ложь 45
3. True и False в переменных 47
4. Сравнение переменных в Python 48
5. Программа для проверки пароля 49

6. Итоги пятой главы 50
7. Тест пятой главы 51
ГЛАВА ШЕСТАЯ: УСЛОВНЫЕ ЗАЯВЛЕНИЯ 52
1. Что такое условные операторы? 53
2. Условный оператор If 54
2.1. if и операторы сравнения 56
Конец ознакомительного фрагмента. 57

Jean-Loup Chrétien, Roman Gurbanov Python с нуля: от новичка до собственных игр и программ

ПРЕДИСЛОВИЕ

Сегодня в ваших телефонах больше вычислительной мощности, чем в компьютерах космических кораблей, на которых я совершал полеты в космос.

Если этих компьютеров и программ хватало для покорения космоса, только представьте, что можете сделать вы, написав свои программы сегодня.

Высокие технологии делают нашу жизнь лучше, интереснее и безопаснее. Каждый, кто желает заниматься чем-то интересным и полезным в современном мире, должен с ними дружить. И мне кажется, изучение программирования по книгам, таким как эта, – один из лучших способов это сделать.

Жан-Лу Кретьен

Первый европеец, вышедший в открытый космос, астронавт NASA, Герой Советского Союза.

ВВЕДЕНИЕ

1. Как получить максимум от этой книги?

В этой книге четырнадцать глав. Двенадцать из них посвящены основам программирования на Python. А тринадцатая содержит Ваш финальный проект (и о нем чуть позже)

Если Вы пройдете все тринадцать глав, то получите крепкие базовые навыки в программировании на Python. Научитесь писать не сложный код, и подготовитесь к финальному проекту, который ждет Вас в конце этой книги.

В этом проекте, Вы создадите собственную программу для бортового компьютера космического корабля. И эта программа отвечает за стыковку корабля с космической станцией.

Стало интересно? :) Тогда продолжим!

Главы этой книги расположены по мере роста сложности: от простого к более сложному.

Если Вы хотите пропустить какую-то главу, Вы, конечно можете это сделать. Но Вы должны знать, что каждая такая глава содержит информацию, необходимую для прохождения следующей главы.

Поэтому советую идти по порядку, и ничего не пропускать :)

Все четырнадцать глав можно пройти залпом за несколько часов. Но, очень рекомендую разделить обучение на ежедневные, небольшие уроки. Хотя бы по 15–30 минут в день.

Если Вы действительно хотите научиться программировать на Python, то лучше учиться этому ежедневно, и понемногу. Чем раз в неделю с утра и до вечера.

Есть еще кое-что. В каждой главе есть готовый код для примера. Я очень рекомендую экспериментировать и создавать свои версии кода, как только вы освоите эти примеры.

Просто меняйте код и наблюдайте за тем, как он влияет на результат работы программы. Так Вы быстрее научитесь программировать.

Примеры кода будут частично на русском. А позже, когда мы перейдем к изучению классов и объектов, уже полностью на английском.

Не бойтесь, знания английского для этого курса вам не нужны.

Но, для того чтобы быть полноценным программистом, в будущем Вам будет необходимо владеть английским, хотя бы базовым.

Вы ведь хотите работать в Google? Эммм? Не хотите? Но знать английский все равно придется.

2. Куда записывать код?

Работая по этой книге, Вам не нужно устанавливать и настраивать никаких редакторов кода.

Просто заходите на PythonOnline.kz. Там Вы можете писать, запускать, проверять и даже скачивать Ваш код на компьютер. Используя встроенный компилятор.

Компилятор PythonOnline.kz был создан для того, чтобы Вы не заморачивались на установке и настройке редакторов кода, а могли сразу приступить к программированию.

В этой книге я могу называть этот компилятор разными словами: редактор, консоль, компилятор. Это все одно и тоже.

3. Как читать код в этой книге?

Читая код в этой книге, пожалуйста обратите внимание на несколько нюансов:

Точки в начале некоторых строк кода указывают на отступы. Эти точки только в книге, и нужны они только для корректного отображения отступов. Вам не нужно ставить ни точки, ни отступы в своем коде, так как наш компилятор будет ставить отступы автоматически.

Все, что стоит в коде за знаком # – это комментарии к коду, для Вашего удобства.

Если Вы сейчас что-то не поняли, не волуйтесь. Время от времени, я буду напоминать Вам об этих нюансах, по мере прохождения данной книги.

4. Что делать с тестами из книги?

Очень рекомендую проходить тест в конце каждой главы.

Отвечайте на вопросы в уме. В случае, если в тесте нужно работать с кодом, записывайте, запускайте и проверяйте его на PythonOnline.kz

Если что-то не получается с тестами, Вы всегда можете подсмотреть готовые ответы. Они есть в приложении, в конце книги.

Только не бегите за подсказками сразу! Лучше пройдите тему еще раз. А затем вернитесь к тесту, и перепройдите его.

Здесь нет строгих учителей. И перепроходить тесты можно сколько угодно, пока Вы не останитесь довольны своим результатом.

5. Почему именно Питон?

Python – один из самых простых в освоении, но в то же время один из самых популярных и широко используемых языков программирования во всем мире.

Я бы порекомендовал Python как первый язык программирования всем, кто хочет научиться программировать.

Почему?

У Python чистый, минималистичный синтаксис. И это упрощает написание и чтение кода.

Например, для того, чтобы написать небольшую программу на Python, Вам может потребоваться всего несколько строчек кода.

А для того, чтобы написать такую же программу, скажем, на Java или C++, Вам придется писать куда больше кода.

Именно поэтому, на технических собеседованиях в Google или еще куда, Вам позволят решать задачи на Python. Даже если Вас рассматривают на позицию разработчика Java или C ++ и тд.

Python – это высокоуровневый язык программирования. Это означает то, что он автоматизирует многие процессы, такие, как управление памятью.

А это, в свою очередь, поможет Вам сосредоточиться на основных задачах, пока Вы пишите код.

Python чрезвычайно популярен в реальном мире. Возьмите для примера IT-гигантов, таких, как: Google, Apple, Netflix. Все они используют Python в повседневных задачах, связанных с обработкой данных, работой нейронных сетей, и других важными для этих компаний процессов.

Python, действительно, универсален. Он отлично работает не только для математических задач, связанных с данными. Но и для веб-приложений, видеоигр, и, вообще, чего угодно. И это благодаря огромному выбору расширений и библиотек, доступных для Python. Об этом Вы узнаете более подробно в четырнадцатой главе.

Наконец, у Python есть преданное и постоянно растущее сообщество разработчиков. Это означает, что количество сфер и задач, где применяется Python, будет только расти.

Ну, а про зарплаты программистов на Python, их карьерные возможности и спрос, я просто промолчу. Об этом уже итак с каждого угла крикнули :)

На этом все. Желаю Вам приятного обучения, и увидимся в следующей главе!

ГЛАВА ПЕРВАЯ: НАЧИНАЕМ ПРОГРАММИРОВАТЬ НА PYTHON!

1. Ваша первая строка кода

Любая, даже самая продвинутая программа на Python, начинается с первой строки кода.

Вот пример простой программы, которая состоит всего из одной строки. Все, что она делает, это выводит сообщение: “Привет! Это моя первая строка кода!”

Впишите эту строчку в компилятор, и запустите код:

print("Привет! Это моя первая строчка кода!")

Если Вы все сделали правильно, то компилятор ответит Вам вот таким сообщением:

“Привет! Это моя первая строка кода!”

Получилось? Поздравляю с первой строчкой кода!

И даже если Вы пока не понимаете что к чему, не волнуйтесь. Мы все разберем и узнаем по ходу прохождения книги. А пока продолжим!

2. Что такое программа?

Даже если Вы написали всего одну строчку кода, как в предыдущем разделе, Вы написали программу.

Такую же программу, как и те, что запускаются с компьютера или смартфона.

Но, что такое программа? Программа – это набор инструкций и правил для компьютера, написанный на языке программирования.

Если с этим все понятно, давайте продолжим и закрепим то, что мы пока узнали.

Вот код, который я перемешал:

"Илон Маск ест пингвинов!" () print

Впишите его в компилятор так, чтобы он заработал. Выполнив прошлую задачу, Вы сможете выполнить и эту.

Если Вы все сделали правильно, то вот, что вам ответит программа:

“Илон Маск ест пингвинов!”

Справились с задачей? Отлично!

В обеих программах, которые мы только что создали, мы использовали функцию print (печать).

В этой книге мы часто будем ее использовать. Но, для начала, давайте познакомимся с ней поближе.

3. Функция Print

Функция print делает именно то, чем она и называется. Она “печатает” текст на экране.

Программисты используют эту функцию для того, чтобы показывать сообщения пользователям программы.

Например, «Пожалуйста, войдите в систему, используя пароль» или «Ваш пароль слишком слабый, используйте более надежный пароль» и т. д.

Функция print выводит не только текст, но и результаты вычислений, представленные в виде цифр и чисел.

Об этом мы поговорим немного позже. А пока, давайте еще немного потренируемся с выводом текста.

Вот вам текст:

Эй! Я продолжаю кодить!

Впишите его в компилятор вместе с функцией print, так, чтобы программа вывела такое же сообщение на экран.

Вот, что у вас должно получиться:

“Эй! Я продолжаю кодить!”

Все получилось? Отлично! Теперь, когда Вы умеете выводить сообщения на экран, используя функцию print, Вы можете поэкспериментировать и повеселиться со своими сообщениями.

Придумайте что угодно, любые сообщения. Добавьте к сообщениям цифры и числа.

А еще, попробуйте добавить или удалить что-нибудь из своего кода. И посмотрите, что произойдет. Это лучший способ узнать, как работает Ваш код.

4. Как Python читает код?

Как только Вы запускаете код, компьютер начинает читать его построчно, сверху вниз. Точто также, как Вы сейчас читаете эту книгу.

Это может показаться не важным, но это следует учитывать при написании и организации нашего кода.

Вот почему некоторые элементы, такие как модули (мы изучим их позже), находятся в верхней части кода.

Мы всегда импортируем их сверху нашего основного кода. А затем вызываем эти модули, спускаясь ниже, строчка за строчкой.

5. Программа подсчета

Давайте создадим программу, которая считает от 1 до 3. Вот код, который нам для этого нужен:

```
print("1")
print("2")
print("3")
```

Впишите этот код в компилятор и запустите его.

Программа сосчитала до трех? Довольно просто, правда? Теперь расширьте код, чтобы программа смогла сосчитать до 10.

Получилось? Программа считает до десяти? Отличная работа!

6. Самостоятельная работа

Давайте отметим завершение первой главы небольшим испытанием.

Все, что Вам нужно сделать, это вывести это сообщение на экран:

Программировать на Python легко!

Ну как? Смогли? Я и не сомневался в Вас. Программировать на Python действительно не сложно. А теперь к итогам!

7. Итоги первой главы

В первой главе Вы выполнили следующее:

1. Узнали, что такое Python;
2. Узнали, что такое программа;
3. Поняли, как Python читает код;
4. Написали первую строчку кода;
5. Создали несколько простых программ;
6. Научились использовать функцию print.

Я Вас поздравляю! Теперь Вы готовы ко второй главе – Переменные. Давайте приступим к ней!

8. Тест первой главы

1. Компьютерная программа – это:

1. Набор инструкций и правил для компьютера, написанный на языке программирования.
2. Кусок кода, написанный на компьютере.
3. Загружаемая игра.

2. Как сделать так, чтобы компьютер вывел сообщение на экран?

1. Используя волшебное слово.
2. Используя функцию print().
3. Используя команду «Показать сообщение!»

3. В каком порядке компьютер обрабатывает (считывает) код?

1. Компьютер считывает код построчно. Сверху вниз.
2. Компьютер считывает код построчно. Снизу вверх.
3. Компьютер ничего не считывает; Он все помнит наизусть.

4. Расположите фрагменты кода так, чтобы программа отображала сообщение «Я люблю Python!»

1.)
2. (
3. "Я люблю Python!"
4. print

ГЛАВА ВТОРАЯ: ПЕРЕМЕННЫЕ

1. Что такое переменная?

Переменные в Python создаются просто. И в этой главе Вы с легкостью научитесь создавать и применять их в своем коде.

Итак, приступим.

Переменная – это простой тип данных, у которого есть имя и значение. Переменные нужны для того, чтобы хранить в них информацию.

Давайте объясню на примере.

Представьте себе машину. В нашем случае машина является переменной. Название переменной – машина.

У машины есть марка – Tesla. Это значение переменной.

Таким образом, переменная (машина), хранит информацию о марке машины – Tesla.

Вот, как эта переменная выглядит на языке Python:

```
машина = "Tesla"
```

Давайте разберем все по порядку:

Сначала мы дали нашей переменной имя – машина.

Затем выставили знак равенства =

И наконец, мы присвоили нашей переменной значение “Tesla”

Значение переменной всегда ставится в кавычки, как в нашем примере. Иначе переменная работать не будет.

Теперь, когда мы поняли, что такое переменная, и как ее прописать, давайте создадим нашу первую переменную, и выведем ее на экран!

2. Как создать и вывести переменную?

Для начала давайте запишем нашу переменную из примера выше в компилятор и запустим его:

```
машина = "Tesla"
```

Ничего не произошло? Все верно. Ведь переменная – это не сообщение, а всего лишь тип данных.

Для того, чтобы вывести переменную на экран, нам придется использовать функцию print, с которой Вы уже знакомы.

Вот, как мы это сделаем. Запишите следующий код в компилятор и запустите его:

```
машина = "Tesla"
print(машина)
```

Если Вы все сделали правильно, компилятор вернет вам значение переменной – Tesla.

А теперь давайте разберем код по порядку:

Сначала мы объявили переменную и дали ей имя – машина.

Затем мы присвоили переменной значение – “Tesla”.

Затем на второй строке мы прописали функцию print, и передали в эту функцию имя нашей переменной, поместив его в скобки функции.

Каждый раз, когда мы создаем переменную, и передаем ее имя функции print, эта функция будет выводить значение переменной на экран, как в нашем примере.

Кстати, если имя Вашей переменной состоит из более, чем одного слова, тогда Вам необходимо соединить эти слова нижним подчеркиванием. Например: моя_машина.

Однако, советую называть Ваши переменные только одним словом. Это обычная и корректная практика в работе с переменными.

Ну вот, теперь Вы знаете, как программисты создают и выводят переменные в Python на экран.

Совсем не сложно, правда?

А теперь давайте немного потренируемся. Ниже приведен код, в котором кое-чего не хватает. Вам надо это исправить так, чтобы программа могла создать переменную и отобразить ее значение.

```
= " "
```

()

К этому моменту у Вас должно быть достаточно знаний и навыков для выполнения этого легкого задания.

Как только Вы справитесь с этим заданием, потренируйтесь еще немного. Вы можете изменить имя и значение переменной. Добавить и вывести больше новых переменных на экран.

Чем больше повторений Вы выполните, тем лучше закрепите и отточите полученные навыки!

3. Итоги второй главы

Во второй главе Вы выполнили следующее:

1. Узнали, что такое переменные;
2. Научились создавать переменные в Python;
3. Научились выводить значения переменных на экран.

Отличная работа! Переходим к третьей главе – Числа. Обещаю, никакой скучной математики! Приступим.

4. Тест второй главы

1. Для чего нужны переменные?

1. Переменные нужны для хранения информации.
2. Переменные нужны для изменения информации.
3. Переменные нужны для извлечения или удаления информации.

2. Если имя переменной состоит из двух и более слов, Вы должны соединить их с помощью:

1. Нижнего подчеркивания.
2. Пунктирной линии.
3. Никак, это нормально – слепить все слова в одно.
4. Нужно прописать слова слитно, с большой буквы.

3. Мы можем вывести переменную на экран следующим образом:

1. Используя функцию print и поместив команду print в скобки.
2. Используя функцию print и поместив значение переменной в скобки.
3. Используя функцию print и поместив имя переменной в скобки.

4. Расположите фрагменты кода в правильной последовательности, чтобы получилась переменная, которая выводит "Илон" на экран.

1. (имя)
2. имя
3. =
4. print
5. "Илон"

ГЛАВА ТРЕТЬЯ: ЧИСЛА

1. Целые и дробные числа

Числа в Python, как и в обычной школьной математике бывают целые и дробные (их еще называют вещественными)

И если Вы не часто прогуливали уроки, то уверен, Вы знакомы с целыми числами, например, 5 или 10. А еще Вы знакомы с дробными числами, такими как 5.5 или 10.7, верно?

В программировании целые числа называют integer, а дробные – float.

Ну так вот, Python прекрасно работает и с целыми, и с дробными числами.

А еще Python отлично работает с математическими операторами. Давайте вместе на них посмотрим.

2. Математические Операторы в Python

Python прекрасно справляется с вычислениями. Для этого он применяет вот такие математические операторы:

Сложение: +
Вычитание: –
Умножение: *
Деление: /

Давайте теперь немного поработаем с этими операторами и числами.

3. Работаем с числами

Для начала давайте создадим переменную под названием результат, и присвоим ей значение 2+2.

Затем выведем результат на экран. Вот как это будет выглядеть:

```
результат = 2+2
print(результат)
```

Запишите этот код в компилятор, и запустите его.

Что получилось? Верно, получилось четыре.

А теперь, используя тот же код, вычтите 5 из 10, используя оператор вычитания.

Запускайте код.

Пятерка есть? Отлично.

Теперь давайте умножим 5 на 5, используя оператор умножения.

Двадцать пять получилось? Прекрасно.

Наконец, давайте разделим 10 на 2, используя оператор деления.

Что получилось? 5.0? Верно. Вы только что выполнили деление с остатком. Поэтому в результате у нас вышло дробное число.

А теперь давайте поговорим о делении без остатка.

4. Делим числа без остатка в Python

Итак, в прошлом примере у нас получилось дробное число (float).

Но, что если нам нужно получить целое число (integer)?

Это довольно просто. Для того, чтобы получить целое число при делении, все, что нам нужно сделать, это применить двойной оператор деления – //

Попробуйте сами, замените оператор деления на двойной оператор деления в нашем предыдущем примере, и запустите код:

```
результат = 10//2
print(результат)
```

Если Вы все сделали правильно, то увидите integer равный 5

5. Порядок вычислений в Python

Python делает вычисления точно в таком же порядке, какому Вас учили в школе.

Посчитайте в уме вот такой пример:

(5+5)*3

А затем впишите его в компилятор и запустите код:

```
результат = (5+5)*3
print(результат)
```

Вот как Python будет его решать:

Сначала Python вычислит все, что находится в скобках. Сделает он это в таком порядке: сначала умножение, затем деление, затем сложение, затем вычитание.

После этого, Python вычислит все, что находится за скобками. Сделает он это в том же порядке, что и выше (умножение, деление, сложение, вычитание).

Следовательно, Python сложит 5 и 5, что даст 10. И умножит 10 на 3, что даст 30.

Ну как, совпали Ваши результаты?

Хорошо! Тогда идем дальше.

6. Числа и переменные в Python

Как Вы уже заметили, работая с числами и математическими операторами в Python, мы также использовали переменные и функцию print.

Заметили, да?

Так вот, давайте теперь закрепим то, что мы сделали:

Во-первых, мы объявили переменную, дав ей имя «результат» и значение «(5+5)*3»;

Затем мы спустились на одну строку вниз, прописали функцию print, и передали ей имя нашей переменной;

Когда мы запустили код, Python вычислил (5+5)*3, получил 30, и присвоил это значение переменной «результат»;

Наконец, Python увидел функцию print с аргументом (результат), и понял, что надо вывести на экран значение переменной результат, которое как мы уже поняли равно 30.

Как видите, Python отлично комбинирует числа, математические операторы, переменные и функции.

А теперь давайте еще немного попрактикуемся и создадим свои примеры с числами и переменными, которые похожи на те, что мы только что использовали.

Вот несколько шаблонных примеров для Вас:

```
результат = 2+2
print(результат)

результат = 10–5
print(результат)

результат = 5*5
print(результат)

результат = 10/2
print(результат)
```

7. Итоги третьей главы

В третей главе Вы сделали следующее:

1. Узнали об integer и float – целых и дробный числах;
2. Применили математические операторы в вычислениях в Python;
3. Научились делить число без остатка;
4. Узнали порядок вычислений;
5. Научились комбинировать числа, переменные и функции в Python.

Молодцы! С числами мы разобрались. Настало время научиться создавать и применять строки в Python.

8. Тест третьей главы

1. Какие типы чисел есть в Python?

1. В Python есть три типа чисел: целые, почти целые, и дробные числа.
2. В Python есть два типа чисел: целые и полуцелые.
3. В Python есть два типа чисел: Целые и дробные числа.

2. Как Python делит 10 на 5? (множественный выбор)

1. 10:5
2. 10/5
3. 10–5
4. 10 * 5
5. 10//5

3. Расположите фрагменты кода так, чтобы получилось целое число 4.

1. //
2. 10*2
3. 5

Вопрос 4: Этот код выводит число 5 на экран. Но кое чего в нем не хватает. Определите, чего именно?

```
результат = 10/2
(результат)
```

ГЛАВА ЧЕТВЕРТАЯ: СТРОКИ

1. Строки в Python

Строки в Python – штука необходимая, и вот почему:

Помните сообщения, которые мы выводили в первой главе, например: “Привет! Это моя первая строчка кода!”

Так вот, это все строки.

В Python, строка – это тип данных, имеющий текстовый формат, или просто текст, заключенный в кавычки.

А значит, чтобы создать строку, мы должны ввести текст и заключить его в кавычки.

2. Строки и функция печати

Давайте напишем код, который бы отображал вот такую строку: “Теперь я знаю, что такое строка в Python.”

Вы уже знаете, как выводить строки на экран, в Python, верно?

Вот Вам код для начала. Введите этот код в компилятор, и запустите его:

```
print("Теперь я знаю, что такое строка в Python.")
```

А теперь, Ваша очередь. Придумайте какую-нибудь другую строку, перепишите и запустите приведенный выше код.

3. Хранение строк в переменных

Помните нашу первую переменную:

```
машина = "Tesla"
print(машина)
```

Вот теперь Вы знаете, что значение переменной – “Tesla” имело формат строки, потому что мы заключали это значение в кавычки.

Давайте теперь закрепим наши знания вот таким небольшим заданием:

Мы создадим сообщение в формате строки. Сохраним его в переменной и отобразим значение переменной на экране с помощью функции print.

Давайте я сделаю это первым. Впишите этот код в компилятор и запустите его:

```
сообщение = "Хьюстон, у нас проблема"
print(сообщение)
```

Получилось вывести сообщение? Отлично!

Теперь Ваша очередь. Глядя на пример выше, придумайте свой вариант сообщения, который можно сохранить в переменную. А затем вывести значение переменной на экран.

Придумайте и запустите столько примеров, сколько пожелаете. Чем больше, тем лучше. Так Вы укрепите полученный навык!

4. Конкатенация строк в Python

Мы можем объединять различные строки друг с другом. Этот прием называется “конкатенация”.

Все, что нам нужно сделать, чтобы объединить (конкатенировать) строки, это поставить между ними оператор сложения +.

Ничего сложного, правда? Давайте попрактикуемся с конкатенацией:

"Илон Маск отправил Теслу на" + "Марс"

А теперь выведем обе строки на экран. Впишите этот код в компилятор и запустите его:

print("Илон Маск отправил теслу на" + "Марс")

Заметили нечто странное? Кажется, наши строки слиплись.

Не вопрос! Мы можем это легко исправить. Есть несколько способов. Вот самый простой:

Все, что нам нужно сделать, это оставить пробел между первой кавычкой второй строки и словом, которое идет за этой кавычкой:

print("Илон Маск отправил теслу на" + " Марс")

Исправили? Запускайте код.

5. Конкатенация строк и переменные

Хочу обратить Ваше внимание на то, что мы можем конкатенировать строку только с другой строкой. Или с другим значением, имеющим формат строки.

Например, если мы создали переменную, и присвоили ей значение в формате строки, то мы можем объединить такую переменную с другой строкой.

В приведенном ниже примере я создал переменную (марка), и присвоил ей строковое значение “Tesla”.

Затем я вывел это значение в конкатенации с другой строкой “Машина называется”.

Вот, что получилось:

```
марка = "Tesla"
print("Машина называется " + марка)
```

Впишите этот код в компилятор и запустите его. Если Вы все сделали правильно, программа вернет сообщение “Машина называется Tesla”

А теперь потренируйтесь. Измените код по Вашему желанию. Вы даже можете объединить более двух строк!

Меняйте код и запускайте его. Наблюдайте за тем, как меняется результат.

6. Форматирование строк в Python

Мы уже научились объединять строки с помощью математического оператора +. Этот оператор может только конкатенировать строку с другой строкой.

Но что, если мы хотим конкатенировать строку с чем-то, что не имеет формата строки?

Для этого есть отличный способ! И он называется “Форматирование строки”. Программисты часто им пользуются.

Давайте объединим строку с переменной. Для этого мы переведем значение переменной в формат строки.

Для этого нам понадобятся две вещи:

Первое – это метод format() для форматирования не строкового значения и вложения его внутрь строки-заполнителя.

Второе – это сам заполнитель – {} для не строкового значения.

Давайте я покажу Вам, как это работает, на примере ниже:

```
print("Меня зовут Джо, и мне {} лет".format(20))
```

Введите этот код в компилятор и запустите его. Если Вы все сделали правильно, программа вернет Вам сообщение: Меня зовут Джо, и мне 20 лет.

Получилось? Отлично. А теперь давайте разберем все по порядку:

Мы вставили заполнитель в нашу строку. Вы можете распознать этот заполнитель по фигурным скобкам – {}.

Этот заполнитель нужен для того, чтобы хранить в себе место для возраста нашего Джо, который имеет числовое значение.

В конце строки мы помещаем метод format(), и передаем в его скобки сам возраст – 20, который имеет числовое значение.

В результате этих нехитрых действий Python взял наше числовое значение, отформатировал его в строку, и поместил в заполнитель.

И, наконец, превратив числовой формат в строку, мы вывели всю строку на экран при помощи функции print.

Вот и все. Ничего сложного, верно?

Очень рекомендую Вам поиграть с этим кодом. Поэкспериментировать с заполнителями и значениями, которые Вы передаете методу format().

Вот Вам более сложный пример, с двумя заполнителями:

print("Меня зовут {}, и мне {} лет". format("Джо",20))

Впишите этот пример в компилятор, и запустите его.

Потренируйтесь, объясните себе, как он работает. А затем придумайте свой вариант с двумя или более заполнителями.

7. Итоги четвертой главы

В четвертой главе Вы сделали следующее:

1. Узнали, что такое строки в Python;
2. Узнали, как хранить строки в переменной;
3. Научились конкатенировать строки в Python;
4. Научились конкатенировать строки со значениями переменных;
5. Узнали, что такое форматирование строки, и как переводить типы данных в формат строки.

Отличная работа! А впереди нас ждет очень интересная тема – Булева логика.

Приступим!

8. Тест четвертой главы

Вопрос 1: Что такое строка в Python?

1. Строка в Python – это линия, которая проходит через код.
2. Строка в Python – это простой текст, заключенный в кавычки.
3. Строка в Python – это значение с форматом целого числа.
4. Строка в Python – это значение с форматом вещественного числа.

Вопрос 2: Что нужно сделать, чтобы создать строку?

1. Нужно заключить текст в фигурные скобки.
2. Нужно заключить текст в круглые скобки.
3. Нужно заключить текст в кавычки.

Вопрос 3: Расставьте код так, чтобы получить переменную со значением в виде строки. А затем вывести значение переменной на экран.

1. message
2. =
3. "Привет Илон Маск!"
4. print
5. (message)

ГЛАВА ПЯТАЯ: БУЛЕВА ЛОГИКА

Булева логика в Python, как и в других языках программирования нужна нам для того, чтобы наш код мог сравнивать данные.

Например сравнивать пароль, который вводит пользователь, с паролем, который храниться в базе данных. Или сравнивать ранг игроков в видеоигре, чтобы подбирать соперников по уровню.

Таких примеров сравнений много, и здесь булева логика – крайне полезная штука.

Давайте узнаем из чего она состоит и какие примеры с ней можно создавать.

1. Операторы сравнения

Когда мы сравниваем числа друг с другом, мы обычно используем такие символы, как > (больше), < (меньше), = (равно) и так далее. Они работают и на Python. И вот как они выглядят там:

Больше >
Меньше <
Равно ==
Не равно !=
Больше или равно > =
Меньше или равно <=

Как Вы заметили, в Python вместо знака равенства = нам необходимо использовать двойной знак равенства ==.

Нам нужен двойной знак равенства для того, чтобы Python не подумал, что мы пытаемся создать переменную.

В Python эти символы называются операторами сравнения. И они нужны нам для того, чтобы наша программа могла сравнивать различные типы данных, например: числа, строки, переменные и так далее.

2. Булевы значения: правда или ложь

У Булевой логики есть логические значения: True и False:

True – когда условие истинно. И False – когда условие ложно.

Давайте теперь сыграем.

Я буду сравнивать числа, а Вы будете отвечать в уме – True, если это истина, или False, если это ложь.

Начнем?

```
2 > 4
10 < 20
3 == 3
5 != 7
```

А теперь давайте сделаем то же самое в Python. Для этого мы создадим переменную – результат, и присвоим ей значение 2 > 4

Затем выведем результат на экран с помощью функции print. Вот, как это будет выглядеть:

```
результат = 2 > 4
print(результат)
```

Впишите этот код в компилятор и запустите его.

Если значение переменной является истиной, программа вернет результат True. Если же значение окажется ложью, программа вернет False.

Ну как? Что возвращает программа?

А теперь подставьте остальные значения из списка, одно за другим, запускайте код и наблюдайте за результатом, который возвращает программа.

Вот так и работает булева логика в Python.

Но что, если мы попробуем сравнить строки? Попробуйте сравнить яблоки с апельсинами, используя следующее значение в переменной:

```
результат = "яблоки" == "апельсины"
print(результат)
```

Запишите этот код в компилятор и запустите его.

А теперь сравните яблоки с яблоками:

```
результат = "яблоки" == "яблоки"
print(результат)
```

Видите результат?

Теперь Вы знаете, что в программировании булева логика работает не только с числовыми, но и с другими форматами данных.

3. True и False в переменных

Как Вы уже заметили, мы можем хранить результаты сравнений, которые возвращают True или False в переменных.

Давайте рассмотрим эту тему подробнее вот с таким примером:

```
результат = "яблоки" == "бананы"
print(результат)
```

Давайте впишем этот код в компилятор и запустим его.

В этом коде мы создали переменную с именем “результат”, и присвоили ей логическое значение от сравнения двух строк «яблоки» и «бананы».

Затем мы спустились на одну строку вниз, и вывели значение переменной “результат” на экран, передав имя переменной в скобки функции print. Которую мы перед этим создали.

Правда, не сложно?

А теперь измените код, указав, что яблоки и бананы не равны. Вы уже знаете как это делать :)

Что теперь возвращает программа?

4. Сравнение переменных в Python

Мы можем сравнивать не только числа и строки, но и целые переменные!

Посмотрите, как это можно сделать:

```
игра = "Dota"
результат = игра == "FIFA"
print(результат)
```

Но, прежде чем записать это пример в компилятор и запустить его, подумайте и скажите, какой результат он вернет? True или False?

Решили? А теперь давайте посмотрим правильно Вы решили, или нет:

Как видите, сначала мы создали переменную под названием игра и присвоили ей значение сроки “Dota”.

Затем мы спустились на одну строку ниже, и создали вторую переменную по имени результат.

После этого мы присвоили переменной результат логическое значение от сравнения нашей первой переменной – игра, со строковым значением FIFA.

Затем мы спустились еще на одну строку ниже, и вывели значение переменной результат на экран, с помощью функции print.

А так как значение нашей переменной игра равно строке Dota, а не строке FIFA, то программа вернула False.

Ну как? Совпало Ваше решение с ответом программы?

Давайте теперь закрепим пройденный пример. Возьмите наш код, и напишите на его основе свою версию программы для создания и сравнения двух переменных.

Меняйте что угодно по вашему желанию: имена переменных, их значения. И конечно операторы сравнения.

5. Программа для проверки пароля

Ну и в завершение пройденной главы, давайте отработаем очень упрощенный пример из реальной жизни: Программа, которая проверяет правильность введенного пароля:

```
пароль = "рыба-меч"
приветствие = пароль == "рыба-меч"
print(приветствие)
```

Введите этот код в компилятор и запустите его.

Давайте рассмотрим все по порядку.

В первой строчке мы создали переменную – пароль, и присвоили ей значение – рыба-меч.

Затем на второй строчке мы создали новую переменную по имени приветствие. И присвоили ей логическое значение, которое сравнивает значение переменной пароль и строку “рыба-меч”.

И, наконец, в третьей строчке мы вывели на экран результат логического сравнения из второй строчки.

Поскольку в качестве оператора сравнения мы использовали == (Равно), а значение переменной пароль действительно равно строке “рыба-меч”, то, переменная приветствие вернула True.

Таким образом, можно представить, что:

“рыба-меч” из первой строчки – это пароль, который хранится в базе паролей.

“рыба-меч” из второй строчки – это пароль, который вводит пользователь, чтобы войти в личный кабинет.

приветствие = пароль == из второй строчки – это код, который сверяет пароль от пользователя с паролем из базы.

А print(приветствие) из третьей строчки – это всего лишь функция, которая выводит результат сверки на экран. True, если пароли совпадают, или False, если пароли не совпадают.

Теперь измените “рыба-меч” из первой или второй строчки на любое другое слово или словосочетание, и перезапустите программу. Программа должна вернуть False, так как пароли больше не совпадают.

Этот пример является упрощенным представлением того, как программисты используют булеву логику и операторы сравнения в Python для задач, связанных с авторизацией пользователей в приложениях.

6. Итоги пятой главы

В пятой главе Вы сделали следующее:

1. Узнали, что такое булева логика в Python;
2. Научились применять операторы сравнения;
3. Познакомились с логическими значениями True и False;
4. Научились использовать логические значения в переменных;
5. Узнали, как программисты используют булеву логику для авторизации пользователей.

Поздравляю! У Вас отличный прогресс! А теперь давайте применим булеву логику в более сложных примерах.

Это мы сделаем в следующей главе – Условные заявления в Python.

7. Тест пятой главы

Вопрос 1: Этот код хранит результаты сравнения двух строк, «яблоки» и «апельсины», в переменной по имени результат, а затем выводит значение переменной на экран. Но код перепутался. Расположите фрагменты кода в правильном порядке.

1. Результат
2. "апельсины"
3. (результат)
4. =
5. "яблоки"
6. print
7. !=

Вопрос 2: Этот код сравнивает две переменные. Какой результат вернет код, когда мы запустим программу?

```
машина = "Tesla"
результат = машина == "Toyota"
print (результат)
```

Вопрос 3: У булевой логики есть логические значения. Их всего два. Подставьте правильные определения для каждого из двух значений.

Когда условие оказывается правдой
Когда условие оказывается неправдой

True
False

ГЛАВА ШЕСТАЯ: УСЛОВНЫЕ ЗАЯВЛЕНИЯ

Условные заявления в Python, как и в других языках программирования позволяют нашей программе проверять определенные условия, и выполнять различные инструкции. В зависимости от того, сработало условие или нет.

Например можно создать такое условие и инструкцию: Если пароль верный, пустить пользователя на сайт. Если пароль не верный, выдать сообщение “Пароль не верный”

И так далее.

Условные заявления применяются в видеоиграх, на веб-сайтах, в мобильных приложениях и практически везде. И нам часто придется использовать их в своих программах.

А раз так, давайте научимся их применять!

1. Что такое условные операторы?

Мы проверяем условия, и даем себе инструкции каждый день:

Если сегодня солнечно, то надену очки.

Если в кафе есть WiFi, то спрошу у официанта пароль.

Как Вы заметили, оба примера сверху содержат условие, которое начинается с если (если сегодня холодно, если в кафе есть WiFi), и инструкции (надену очки, спрошу пароль).

Компьютерные программы тоже проверяют условия, чтобы следовать конкретным инструкциям.

Если условие срабатывает, программа возвращает True (помните True и False?), и выполняет одну инструкцию.

Если условие не срабатывает, программа возвращает False, и выполняет другую инструкцию.

Например, когда Вы хотите зайти на сайт, и вводите верный пароль, программа возвращает True, и выполняет инструкцию, которая говорит ей впустить вас.

Если Ваш пароль неверный, программа возвращает False, и следует другой инструкции, которая говорит ей не впускать вас.

Это и есть упрощенное представление того, как работают условные заявления в Python, и в других языках.

Программа проверяет условие и следует заявленной инструкции, в зависимости от того, сработало условие, или нет.

Так легче контролировать ход работы программы. Кстати, ход работы программы называется “Control flow”.

2. Условный оператор If

If – это условный оператор. Он переводится как “Если”.

Мы используем его, когда говорим программе, что нужно сделать, если условие сработало.

А теперь давайте посмотрим, как надо работать с этим условным оператором.

Вот Вам пример для иллюстрации.

Это небольшая программа, которая проверяет Ваш пароль и, если пароль верный, то программа приветствует Вас на сайте.

```
пароль = "секрет"
if пароль == "секрет":
….print("Добро пожаловать на сайт!")
```

Впишите этот код в компилятор и запустите его.

Давайте разберем его подробнее.

На первой строчке мы указали условие: пароль = “секрет”

На второй строчке мы прописали оператор if, а затем условие, которое нужно проверить. То есть проверить, что пароль действительно равен значению “секрет”. И поставили двоеточие после условия;

И, наконец, на третей строчке, мы прописали инструкцию вывести сообщение на экран на случай, если условие окажется верным, и программа вернет True.

При этом сама инструкция выводит сообщение “Добро пожаловать на сайт!” на экран.

Кстати, Вы заметили двоеточие в конце второй строчки?

Оно нужно для того, чтобы программа поняла, где заканчивается условие, которое мы задали, и где начинается инструкция.

Двоеточие всегда ставится после условия, и перед инструкцией.

И, наконец, Вы наверное обратили внимание на то, что в этом коде, инструкция начинается после отступа.

Отступ перед инструкцией нужен для того, чтобы программа поняла, что эта инструкция относится именно к этому условию.

Попробуйте убрать отступ, и запустите программу снова. Вы увидите, что программа вернула ошибку. И ругается на то, что нет отступа.

Не волнуйтесь, в нашем компиляторе отступы ставятся автоматически, после условий с двоеточиями.

2.1. if и операторы сравнения

Конец ознакомительного фрагмента.