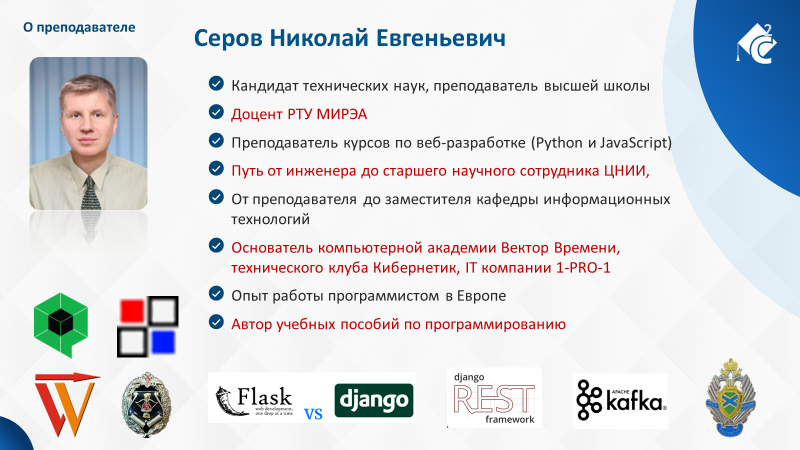
**Модуль 1. Установка Python, IDE, простые типы данных**



# Введение в курс

Для чего используется Питон?

Где вообще применяются языки программирования?

Веб-разработка

Системная разработка — разработка операционных систем, драйверов, встроенных систем (интернет-вещей) и низкоуровневых компонентов.

Десктоп-разработка.

Мобильная разработка — создание приложений для мобильных устройств: смартфонов и планшетов.

Геймдев — разработка компьютерных и мобильных игр.

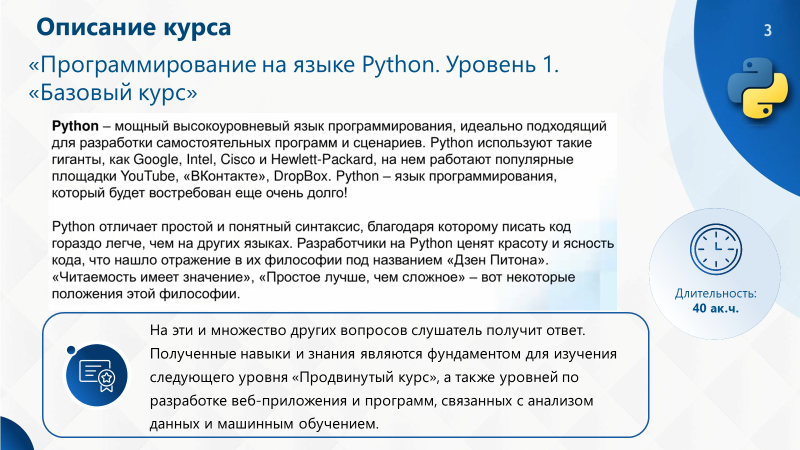
Data science

ML

ИИ

Блокчейн

Питон – в первую очередь инструмент.

****

Решение задач. Решение задач!

Теория – практика.

Но, все же, почему этот язык приобрел такую широкую популярность и завоевал любовь программистов по всему миру?

Считается, что все благодаря его простым, понятным, явным конструкциям, хорошо читаемому тексту программ, богатой библиотеке модулей для программирования самых разных задач и возможность использования языка практически на всех платформах: Windows, Linux, Mac OS, Android, iOS и др.

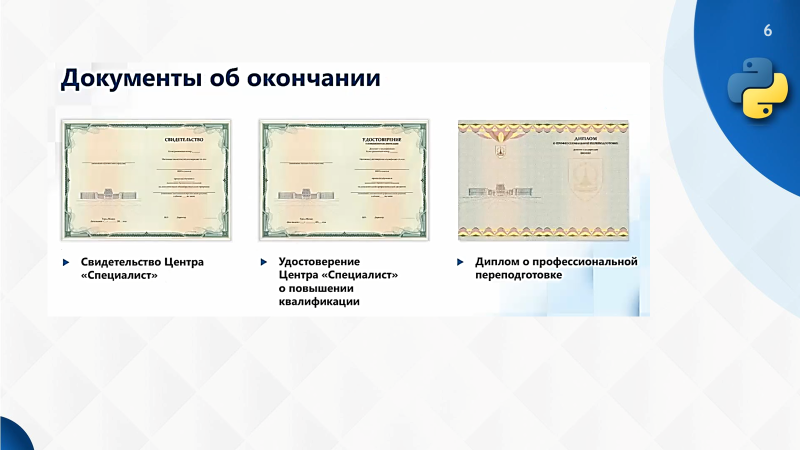


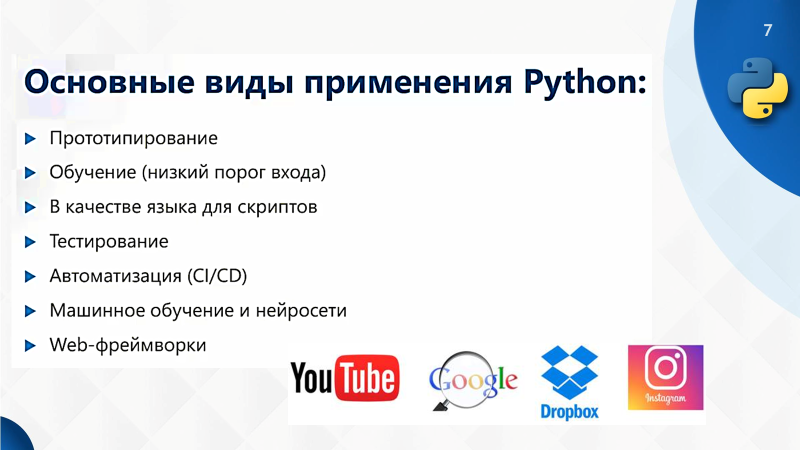
У каждого есть право – за свои священные деньги ничего не делать!

Курс гиперинтенсива. Но в конце будет зачет.

Решение задач на практике.







Python используется в многочисленных научных проектах, где требуются сложные математические вычисления или реализация алгоритмов обработки данных, в том числе и больших данных – Big Data.

В последнее время Python стали применять также и для создания игр, обычно мобильных. Этот язык ввели в ОГЭ и ЕГЭ по информатике, а также в олимпиадное программирование.

Язык оказался настолько удачным, что он стал широко применяться при разработке алгоритмов искусственного интеллекта, в частности, в нейронных сетях.

При разработке серверной части сайтов, используя известные фреймворки Django и Flask. Вперед рвется FastApi, которые может съесть Flask.

Например, на нем разработаны сайты Youtube, Instagram, поиск от Google, DropBox и многие другие.

На данный момент у Python пять кроссплатформенных инструментов, которые можно использовать написания серьезных приложений для Windows, Linux или Mac. Например — «Тикинтер», PyQt, PyGTK, WxPython.

Практическое использование показывает, что ни один из инструментов не может создавать на 100% кроссплатформенные приложения.

Поэтому создание коммерческого десктопа на этом языке программирования — весьма сомнительная затея. Маловероятно, что компании возьмутся за нее.

Можно ли с помощью Python создавать игры?

Существует много игр, которые созданы на «питоне». Есть распространенный миф, что он не годится, чтобы писать серьезные проекты. Но разработчики The Sims 4, World of Tanks, Цивилизация IV, Battlefield 2 точно с ним не согласятся.

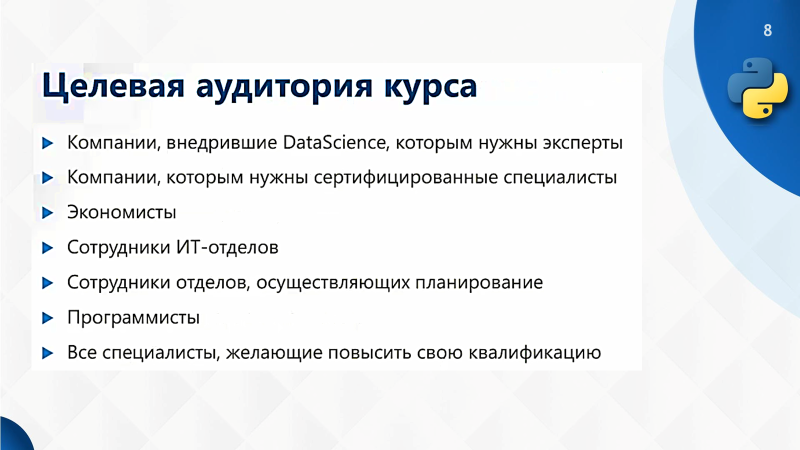
Обычно «питон» используется для создания сценариев. Например — он требуется для взаимодействия с персонажем, запуска сцен и обработки событий.

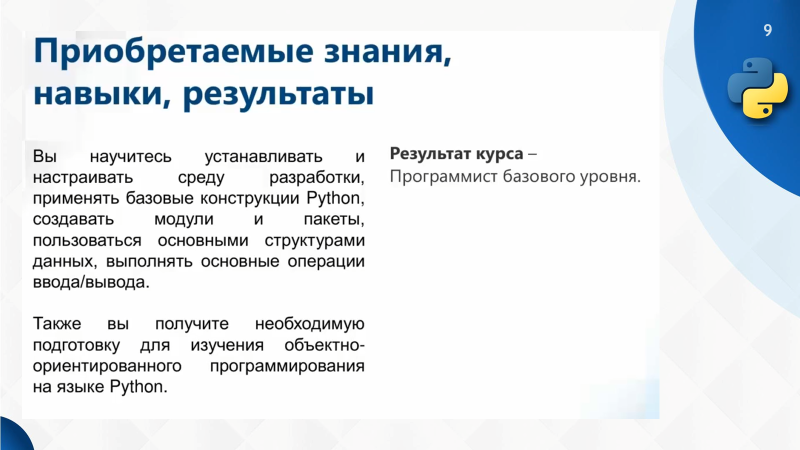
Python для веб-разработки

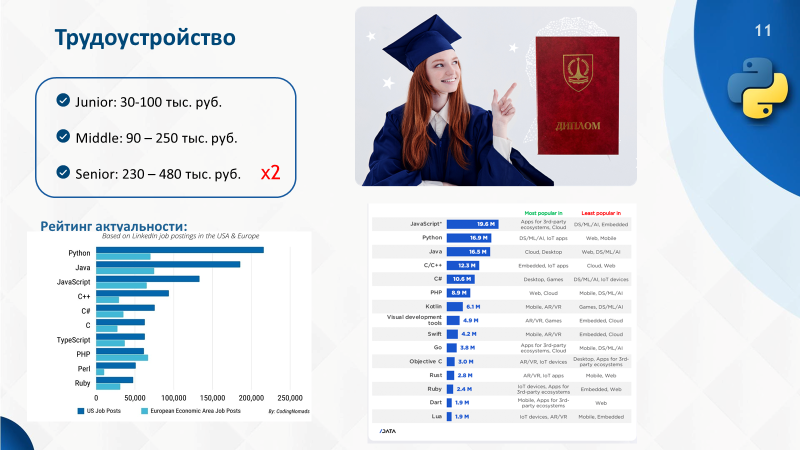
У питонистов есть как мощные фреймворки с богатым инструментарием, например — Django, так и более легковесные варианты для малых проектов в виде FastAPI или Flask. «Питон» позволяет полностью выстроить бизнес-логику и, используя готовые решения библиотек, быстро сделать готовый продукт.

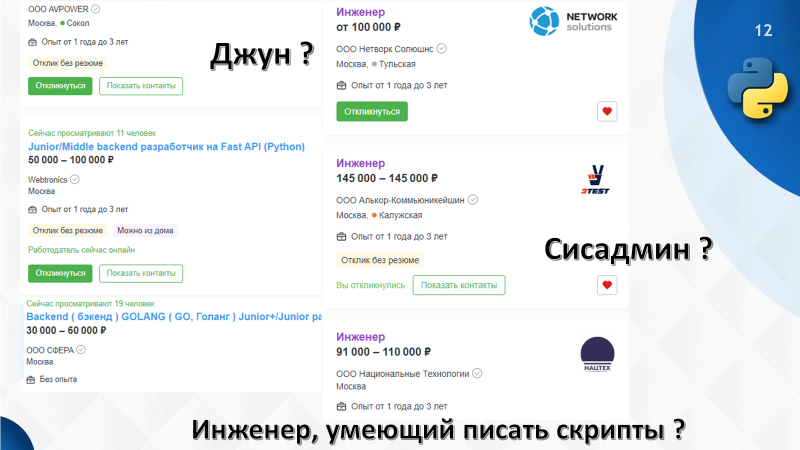
Для веб-разработки не нужно высшее образование. Для ML, ИИ нужен вышмат.

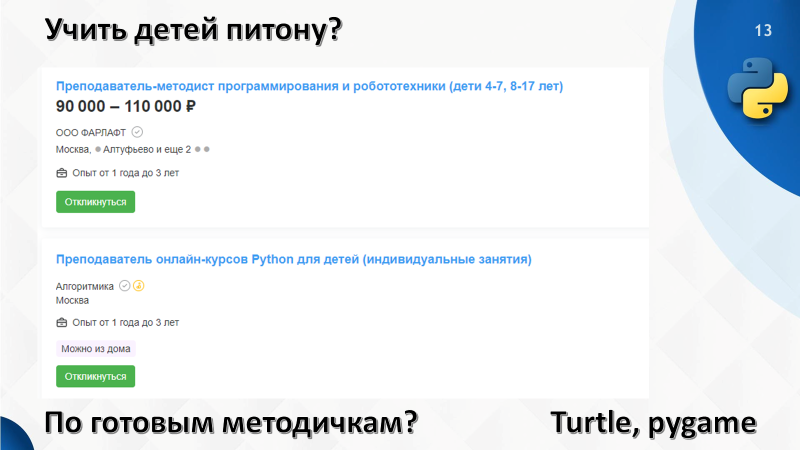
Глубоких знаний по математике или Computer Science не требуется. Код на Python понятнее и приятнее, чем любой другой код. А писать его легче.

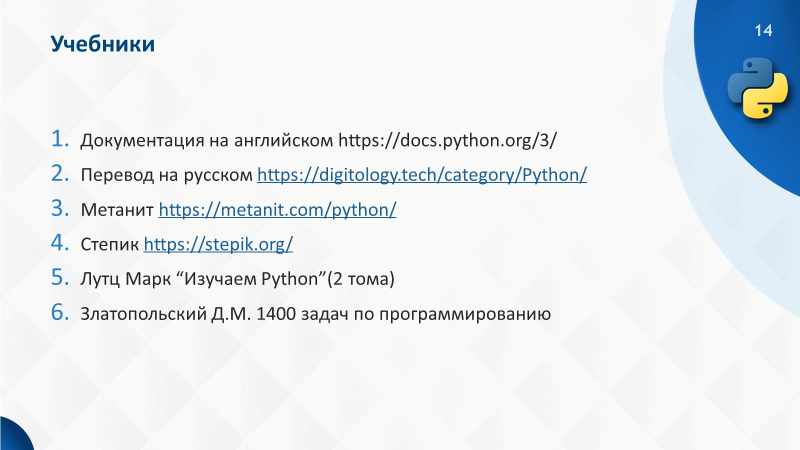




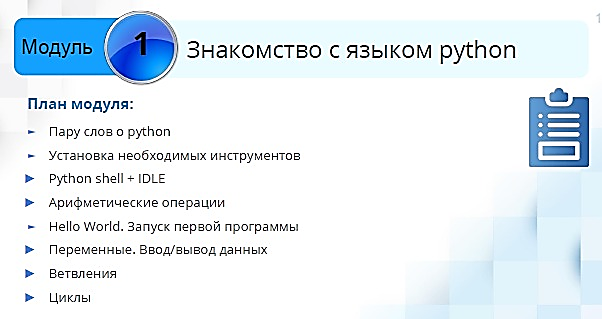




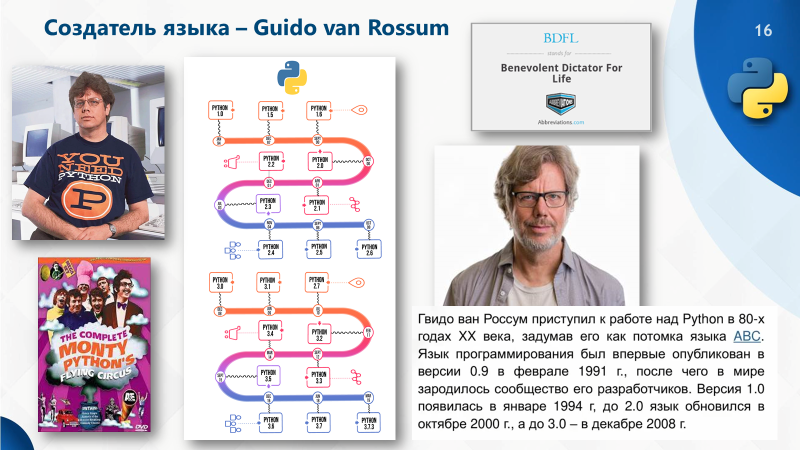








Почему именно Питон? Это нужно спросить у Гвидо ван Россума – главного идеолога и первоначального разработчика этого языка.



Гвидо ван Россум — создатель Python и, наверное, один из самых известных программистов в мире. Он — рок-н-ролльщик в мире айти, только вместо музыки Гвидо создал новый «жанр» языков высокого уровня, которые понятны и доступны каждому. За его плечами:

Годы работы в научно-исследовательском центре, где он изучал технологии и оттачивал навыки программирования.

Прототип Python — язык ABC, который должен был стать легендарным, но что-то не получилось.

Попытки написать скриптовый язык для новой операционной системы, который впоследствии станет первой версией Python.

Попытка выпустить третью версию Python — что получилось трагично и криво.

И красивый уход, сделав Python самым популярным языком среди программистов.

Задуман он был еще в 1980-x, а первая версия вышла в 1991 году. Конечно, с тех пор он претерпел множество улучшений, особенно в версии Python 3.0, выпущенной в декабре 2008 года.

Отличия оказались настолько значительными, что программы предыдущих версий 2.x далеко не всегда можно запустить в интерпретаторе Python 3.x.

Однако, сейчас это уже не проблема, так как с 2008 года прошло много времени и все важные программы были адаптированы к версии 3.

Именно эту последнюю, современную версию языка Python мы с вами и будем изучать.

Хотя, на самом деле понять, почему один проект «выстреливает», а большинство уходят в небытие, еще никому не удалось. Но Питон – это яркая история успеха, правда, не лишенный и своих недостатков.

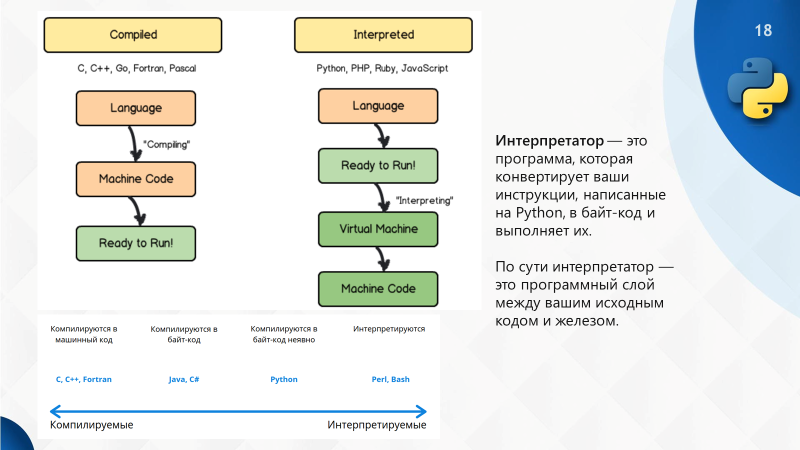
Главные из них – более медленная скорость работы программ и больший объем используемой памяти, по сравнению, например, с аналогичным кодом, написанном на языке С++.

Но, все же, Python позволяет заметно быстрее реализовывать сложные алгоритмы и этим качеством он затмевает большинство современных языков программирования.

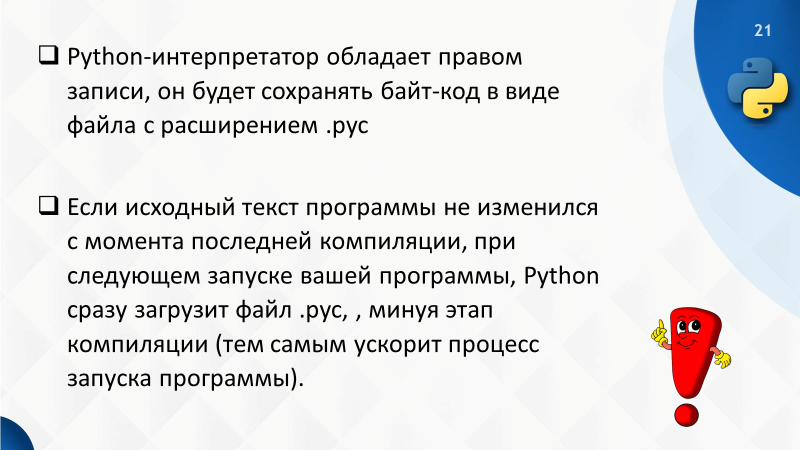
А скорость работы критического фрагмента кода можно увеличить, если реализовать его на том же С++, а затем, вызывать из Python-программы. Например, так делается при проектировании и обучении нейронных сетей. Благодаря этому получаем удобство программирования и высокую скорость исполнения.



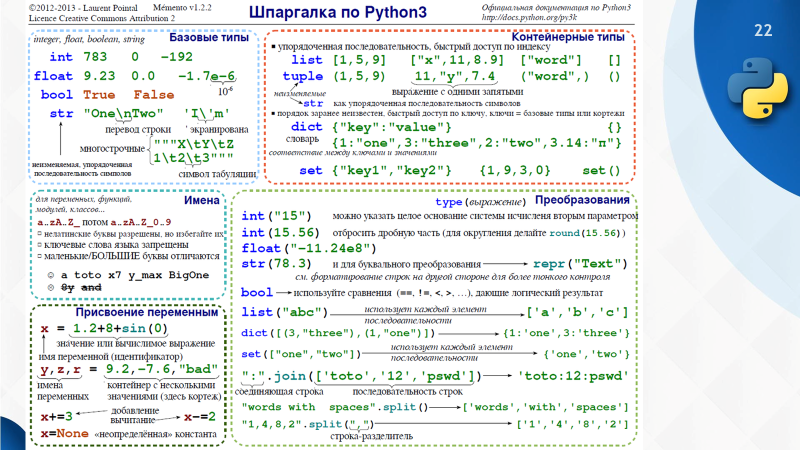
Как упрощенно работает компилятор? Еxe-файл – ни что иное как скомпилированный файл (для Windows). Больше компилятор не нужен. Скомпилировали под линукс, программа работает на линукс.



Интерпретатор же работает медленнее, построчно выполняя наш код! В реальном времени. Минус интерпретируемые языки – медленнее, плюс – они как правильно позволяют писать меньше кода, и они более понятны. Пробовали писать на С++.



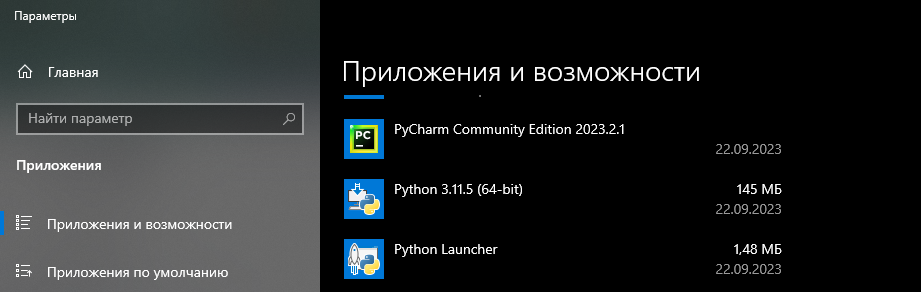
Попытки создать компилируемый пайтон продолжаются почти за всё время его существования. Jython, Cpyton и т.д. Но увы, воз и ныне там!





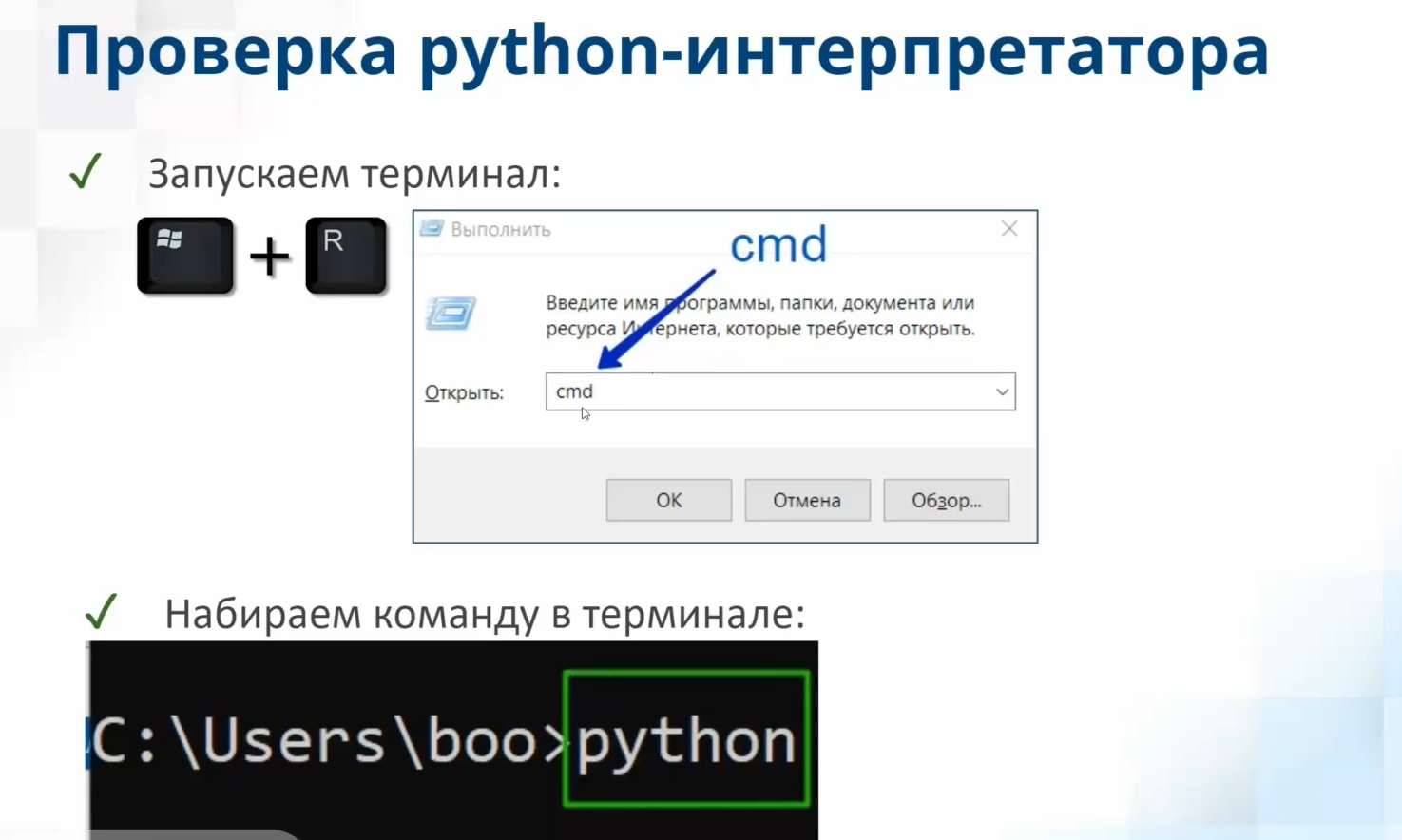
# Установка интерпретатора Python и IDE

Перед началом работы удалите интерпретатор и IDE из системы

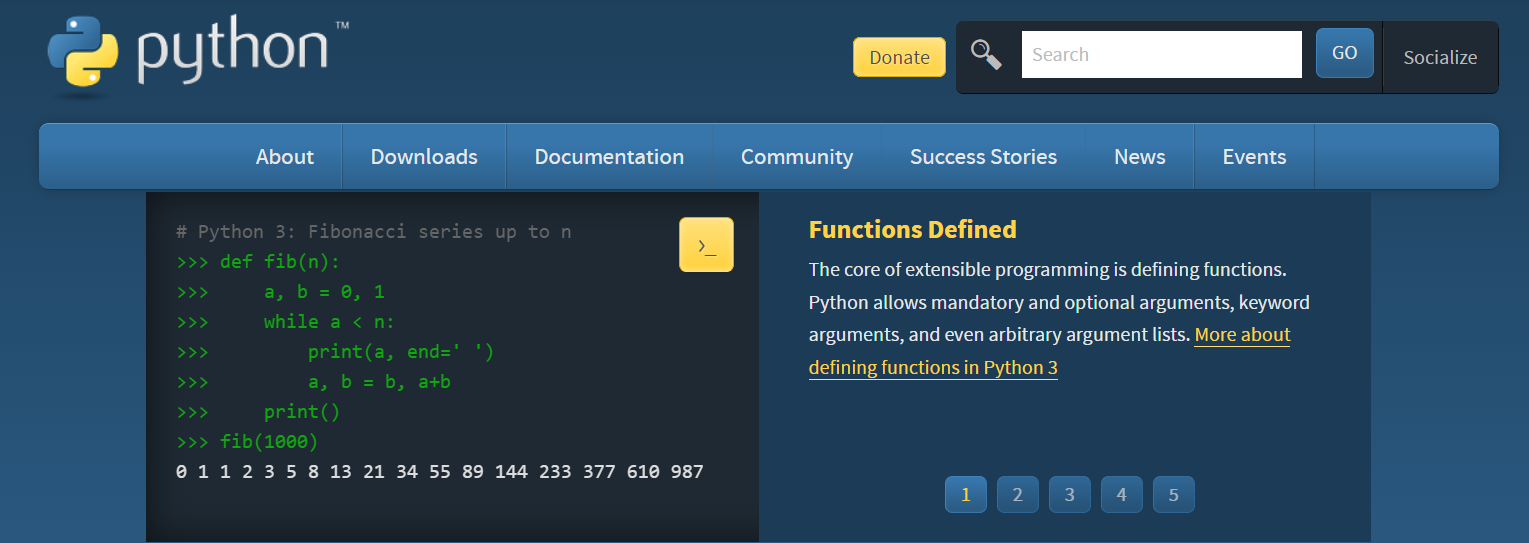


## Установка языка Python

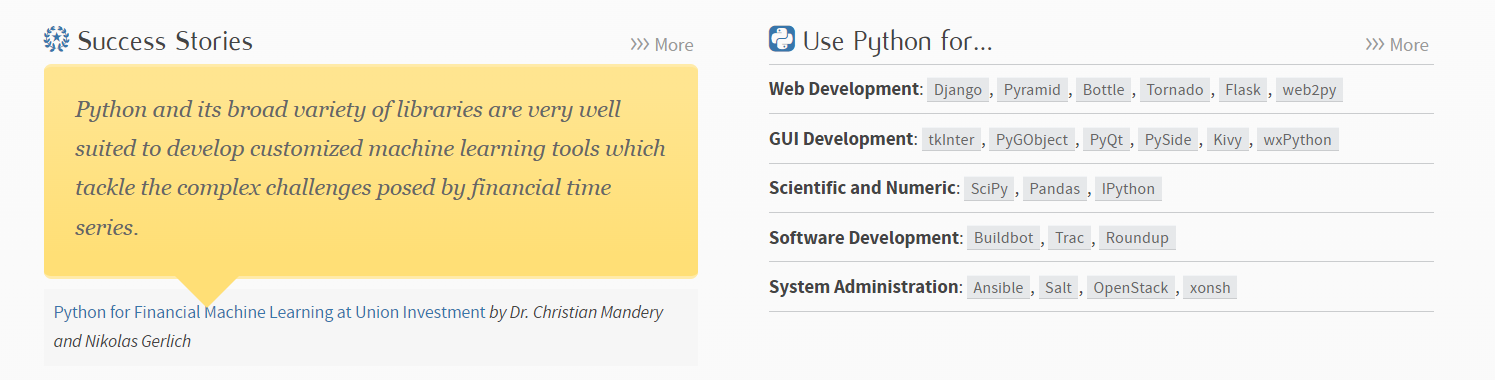
Проверка интепретатора в системе



Перейдем на сайт <https://www.python.org/>.



И прокрутим страницу до Use Python for



Python следует рассматривать не как язык, а как инфраструктуру, как платформу с различными расширениями, дополнениями, модулями, библиотеками.

Python – это ваш осознанный выбор, сделанные не на годы, а на десятилетия. Пусть самурая (но без харакири).

Адский количество зависимостей. Виртуальное окружение, девопсовкие вопросы.

Вопрос, что можно сделать на пайтоне.

Ответ – 99 % да. Не надо менять язык.

Оказывается IT это не только программисты. Индустрия очень стремительно разрастается и кроме разработчиков существуют project manager’ы, бизнес-аналитики, архитекторы баз данных, product owner’ы, scrum-мастера, тестировщики и DevOps’ы.

Последние двое тесно связаны с программистами и принимают непосредственное участие в разработке ПО.

Если в вашей работе нужны такие страшные вещи, как Docker, Ansible, Kubernetes или AWS, то вам определённо к этим ребятам-девопсам.

Давайте теперь выполним установку интерпретатора этого языка на свое устройство. У меня – это стационарный компьютер с ОС Windows 10.

Если я нажму клавиши **Win + Pause**, то появится окно с подробной информацией о системе. В частности, здесь написано название ОС и ее разрядность x64. Конечно, у вас может быть любое другое устройство и ОС – это не существенно, в любом случае, нам нужно перейти на официальный сайт языка Python:

[https://www.python.org](https://www.python.org/)

Затем, выбрать вкладку «Downloads» и здесь появится список различных ОС. Вам нужно найти свою ОС и перейти. Так как у меня Windows, то я выбираю этот пункт и появляется страница с выбором различных версий языка. Конечно же, следует взять последнюю стабильную для Python 3.

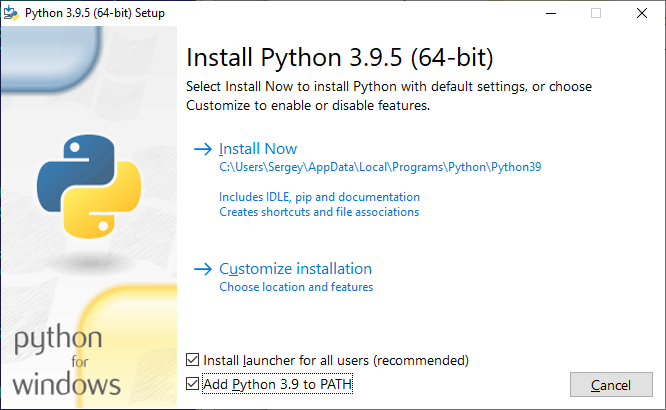
На момент записи видео – это версия ... Я скачаю 64-bit версию установщика:

Download Windows installer (64-bit)

так как у меня ОС Windows 64-разрядная (обычно, именно такая и установлена на домашнем компьютере).

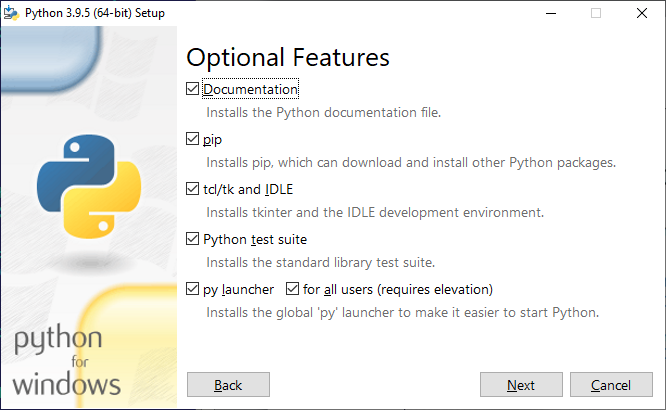
И запущу его. Появится окно, в котором обязательно следует отметить галочкой опцию «Add Python to PATH».

Она позволит в дальнейшем запускать интерпретатор языка без указания полного пути к нему:

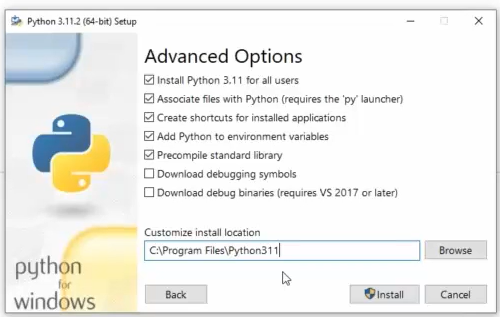


Затем, выбираем режим установки «Customize installation».

Здесь все опции должны быть отмечены галочками:



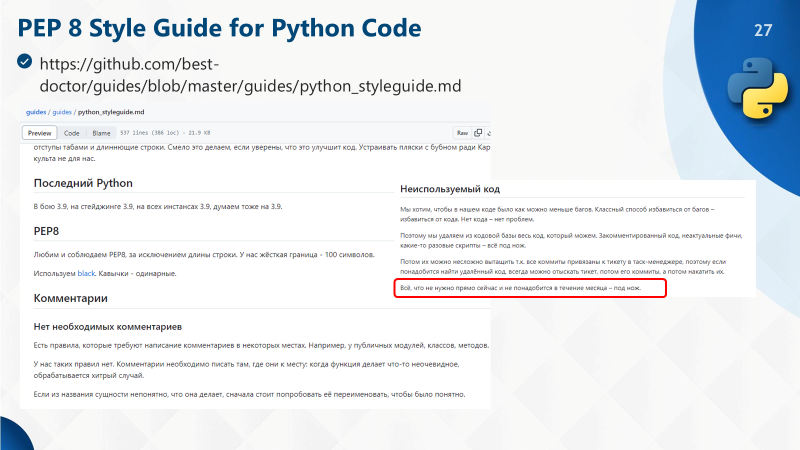
Нажимаем «Next» и в следующем окне я рекомендую путь, который прописан по умолчанию, изменить на другой. А именно, убрать все промежуточные подкаталоги и установить интерпретатор в корень выбранного диска:



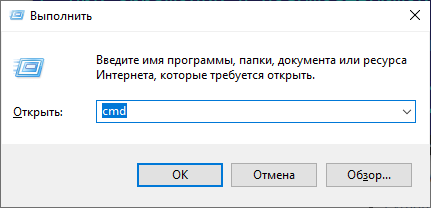
У меня получился путь «C:\Python39». Здесь дополнительно указана версия языка 3.9. Это также рекомендуется делать, так как разных версий на одном устройстве может быть несколько и чтобы переключаться между ними, они должны находиться в разных директориях.

После этого, нажимаем на кнопку «Install» и программа устанавливается в указанный каталог.

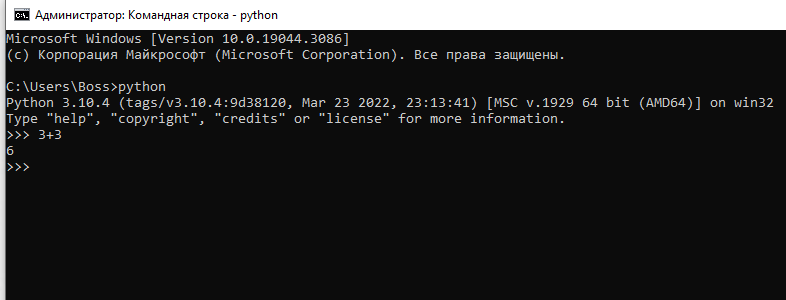
В процессе установки <https://github.com/best-doctor/guides/blob/master/guides/python_styleguide.md>



Осталось проверить работоспособность языка Python. Во-первых, мы можем открыть окно выполнения команд в ОС Windows (Win+R) и в появившемся окне набрать «cmd»:



Далее, в консольном командном окне набираем «python» и должны увидеть текущую версию интерпретатора этого языка:



Если вы работаете под Linux или Mac OS, то следует набрать команду «python3».

Но писать и исполнять команды самого языка лучше через среду IDLE, которая устанавливается вместе с интерпретатором.

Для ее запуска достаточно нажать на кнопку «Пуск» и выбрать оболочку IDLE. Появится окно, в котором можно в интерактивном режиме выполнять любые команды языка. Например, математические

2+4  
3\*5  
8/6

Мы о них еще подробно будем говорить. Здесь я просто привожу пример и показываю, как выполняются простейшие команды языка Python.

Преимущество такого подхода – мгновенное получение требуемого результата. Это бывает полезно при отладке программы, чтобы, например, быстренько посмотреть значения текущих данных и сделать для себя некоторые выводы.

Недостаток интерактивного режима – потеря ранее записанных команд. Например, если перезапустить IDLE, то все ранее написанное пропадает, не сохраняется. Поэтому писать полноценные программы в таком режиме не получится.

Для этого следует использовать другой, файловый режим исполнения команд. В оболочке IDLE это делается так. Выбираем меню File, затем, New File и у нас появляется второе окно. Здесь можно написать сразу несколько команд:

2+3  
8/3  
5\*6

Сохранить их в файл и, затем, выполнить программу (F5). Результат должен отображаться в командном окне IDLE. Но мы ничего не видим. Почему? Дело в том, при файловом режиме команды выполняются строчка за строчкой, то есть, последовательно, сверху-вниз, но результаты автоматически не отображаются, как это происходит в интерактивном режиме. Можно подумать, что это плохо. Возможно, мы хотели увидеть результат действия наших арифметических операций? Однако, это разумное поведение интерпретатора. Часто целью программ является не вывод всех промежуточных результатов, а только требуемого конечного значения. Но как его вывести? Для этого в языке Python существует специальная функция print(). Если все наши арифметические операции записать в виде:

**print**(2+3)

**print**(8/3)

**print**(5\*6)

то в консоли увидим три строчки, три вывода. В интерактивном режиме, кстати, она работает абсолютно также – выводит некоторое сообщение в консоль. На данный момент, просто запомните, то, что находится внутри круглых скобок функции print(), та информация и выводится. Причем, эту функцию можно записать и с пустыми скобками:

**print**()

Тогда просто отобразится пустая строка. Также можно просто указать число:

**print**(5)

**print**(5.6)

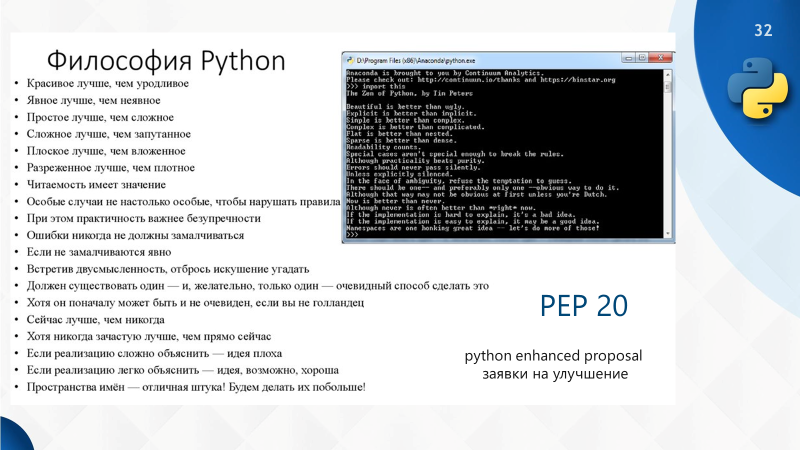
или строку:

**print**("hello")

Пока это просто демонстрация возможностей функции print(). Подробнее мы о ней еще поговорим. А до тех пор будем использовать в таком простом виде.

Python настолько крут, что у него есть своя философия или дзен. Хотите на него взглянуть? Конечно хотите! Для этого откройте IDLE и в интерактивном режиме наберите команду

import this



На русском это обозначает следующее:

*Дзен Python, Tim Peters*

*Красивое лучше, чем уродливое.  
Явное лучше, чем неявное.  
Простое лучше, чем сложное.  
Сложное лучше, чем запутанное.  
Плоское лучше, чем вложенное.  
Разреженное лучше, чем плотное.  
Читаемость имеет значение.  
Особые случаи не настолько особые, чтобы нарушать правила.  
При этом практичность важнее безупречности.  
Ошибки никогда не должны замалчиваться.  
Если они не замалчиваются явно.  
Встретив двусмысленность, отбрось искушение угадать.  
Должен существовать один и, желательно, только один очевидный способ сделать это.  
Хотя он поначалу может быть и не очевиден, если вы не голландец.  
Сейчас лучше, чем никогда.  
Хотя никогда зачастую лучше, чем прямо сейчас.  
Если реализацию сложно объяснить — идея плоха.  
Если реализацию легко объяснить — идея, возможно, хороша.  
Пространства имён — отличная штука! Будем делать их больше!*

## Основные арифметические операции

Пока такого понимания чисел будет вполне достаточно. Следующим шагом, нам с вами нужно научиться делать арифметические операции над ними. Что это за операции? Базовыми из них являются, следующие:

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Приоритет |
| + | сложение | 2 |
| - | вычитание | 2 |
| \* | умножение | 3 |
| *\*\** | *возведение в степень* | *4* |

Давайте, я поясню их работу на конкретных примерах.

Можно перейти в консоль языка Python, чтобы выполнять команды в интерактивном режиме.

ак будет удобнее для демонстрации возможностей вычислений. В самом простом варианте мы можем просто сложить два целых числа:

2+3

Получим результат 5.

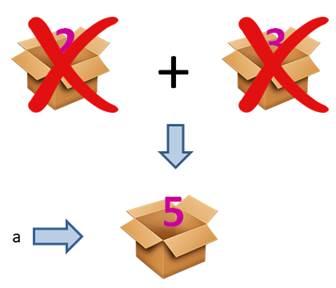
Но этот результат у нас нигде не сохраняется. Чтобы иметь возможность делать какие-либо действия с пятеркой, ее следует сохранить через переменную, например, вот так:

a = 2+3

Теперь a ссылается на объект с числом 5. Давайте разберемся, как работает эта строчка.

Сначала в Python создаются два объекта со значениями 2 и 3.

Оператор сложения берет эти значения, складывает их и формирует третий объект со значением 5. А, затем, через оператор присваивания, этот объект связывается с переменной a. В конце, если на объекты 2 и 3 не ссылаются никакие другие переменные, они автоматически удаляются из памяти сборщиком мусора.



Возможно, вас удивило, что при такой простой операции сложения двух чисел выполняется столько шагов. Но в Python реализовано все именно так. И это справедливо для всех арифметических операций. Мало того, раз операция сложения возвращает объект с результатом, то можно сделать и такое сложение из трех чисел:

b = 2+3+4

И так далее, можно записать сколько угодно операций сложения в цепочку.

Давайте теперь сложим целое число с вещественным:

c = 2 + 3.5

Очевидно, что результат получается тоже вещественным. Отсюда можно сделать вывод, что сложение целого числа с вещественным всегда дает вещественное значение.

А вот при делении двух любых чисел, мы всегда будем получать вещественное число (даже если числа можно разделить нацело):

d1 = 8 / 2

d2 = 3 / 6

Последняя арифметическая операция – это возведение в степень. Она работает просто:

2 \*\* 3  *# возведение в куб*

36 \*\* 0.5  *# 36 в степени 1/2 (корень квадратный)*

2 \*\* 3 \*\* 2 *# 2^3^2 = 512*

В последней строчке сначала 3 возводится в квадрат (получаем 9), а затем, 2 возводится в степень 9, получаем 512. То есть, оператор возведения в степень выполняется справа-налево. Тогда как все остальные арифметические операции – слева-направо.

## Практика 1. Работа с python-shell

Вычислить с помощью интерпретатора Python значение выражения

1. **Ответ для самопроверки:** -16

Вычислить с помощью интерпретатора Python значение выражения

1. **Ответ для самопроверки:** 40.7
2. Вычислить с помощью интерпретатора Python значение выражения

**Ответ для самопроверки:** 17.6

1. Вычислить с помощью интерпретатора Python значение выражения

**Ответ для самопроверки:** 3227.0

1. Используя python-shell выясните, что делает операция **//**
2. Используя python-shell выясните, что делает операция \*\*
3. Используя python-shell выясните ,что делает операция **%**

Как прошла самопроверка

Разберем оставшиеся операции

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Приоритет |
| // | деление | 3 |
| *%* | *остаток деления* | *3* |

Если же нам нужно выполнить деление с округлением к наименьшему целому, то это делается через оператор:

d3 = 7 // 2

На выходе получаем значение 3, так как оно является наименьшим целым по отношению к 3,5. Обратите внимание, что при делении отрицательных чисел:

d3 = -7 // 2

получим уже значение -4, так как оно наименьшее по отношению к -3,5. Вот этот момент следует иметь в виду, применяя данный оператор деления.

Следующий оператор умножения работает очевидным образом:

5 \* 6

2 \* 4.5

Обратите внимание, в последней операции получим вещественное значение 9.0, а не целое 9, так как при умножении целого на вещественное получается вещественное число.

Давайте теперь предположим, что мы хотим вычислить целый остаток от деления. Что это вообще такое? Например, если делить

10 : 3

то остаток будет равен 1. Почему так? Все просто, число 3 трижды входит в число 10 и остается значение 10 - 3∙3 = 1. Для вычисления этого значения в Python используется оператор:

10 % 3

Если взять:

10 % 4

то получим 2. Я думаю, общий принцип понятен. Здесь есть только один нюанс, при использовании отрицательных чисел. Давайте рассмотрим четыре возможные ситуации:

9 % 5   *# значение 4 (9 - 5)*

-9 % 5  *# значение 1 (*-9 – (-10) = 1)

9 % -5  *# значение -1 (*9 – 10 = -1)

-9 % -5 *# значение -4 (*-9 – (-5) = -4)

Почему получаются такие значения? Первое, я думаю, понятно. Здесь 5 один раз входит в 9 и остается еще 4.

При вычислении -9 % 5 по правилам математики следует взять наименьшее целое, делящееся на 5. Здесь – это значение -10. А, далее, как и прежде, вычисляем разность между наименьшим, кратным 5 и -9:

-9 – (-10) = 1

При вычислении 9 % -5, когда делитель отрицательное число, следует выбирать наибольшее целое, кратное 5. Это значение 10. А, далее, также вычисляется разность:

9 – 10 = -1

В последнем варианте -9 % -5 следует снова выбирать наибольшее целое (так как делитель отрицателен), получаем -5, а затем, вычислить разность:

-9 – (-5) = -4

Как видите, в целом, все просто, только нужно запомнить и знать эти правила. Кстати, они вам в дальнейшем пригодятся на курсе математики.

divmod(9,5) -

принимает два любых числа (за исключением комплексных чисел) и возвращает кортеж(!) состоящий из двух элементов.

## Приоритеты арифметических операций

Давайте теперь посмотрим, что будет, если выполнить команду:

27 \*\* 1/3

Получим значение 9. Почему так произошло? Ведь кубический корень из 27 – это 3, а не 9? Все дело в приоритете арифметических операций (проще говоря, в последовательности их выполнения). Приоритет у оператора возведения в степень \*\* - наибольший. Поэтому здесь сначала 27 возводится в степень 1, а затем, 27 делится на 3. Получаем искомое значение 9.

Если нам нужно изменить порядок вычисления, то есть, приоритеты, то следует использовать круглые скобки:

27 \*\* (1/3)

Теперь видим значение 3. То есть, по правилам математики, сначала производятся вычисления в круглых скобках, а затем, все остальное в порядке приоритетов.

Приведу еще один пример, чтобы все было понятно:

2 + 3 \* 5    *# 17*

(2 + 3) \* 5  *# 25*

То есть, приоритеты работают так, как нас учили на школьных уроках математики. Я думаю, здесь все должно быть понятно. Также не забывайте, что все арифметические операторы выполняются слева-направо (кроме оператора возведения в степень), поэтому в строчке:

32 / 4 \* 2

сначала будет выполнено деление на 4, а затем, результат умножается на 2.

## Интегрированная среда PyCharm

Далее я хочу познакомить вас с интегрированной средой программирования PyCharm, разработанной специальной для написания программ на Python. Она намного удобнее IDLE, с которой вы уже немного знакомы, поэтому все дальнейшие действия я буду выполнять в PyCharm.

Я рекомендую вам также ее установить, хотя это не обязательное действие, так как все программы курса можно будет выполнять и в IDLE. Но, как говорится: лучше один день потерять, чтобы потом за пять минут долететь. Поэтому, лучше затратить немного времени на установку PyCharm, чтобы потом стало проще и приятнее писать программы на Python.

Вначале нам нужно перейти на официальный сайт программы:

<https://www.jetbrains.com/pycharm/>

и скачать бесплатную версию «Community». Для решения большинства задач ее будет вполне достаточно. Затем, скачиваем приложение для установки PyCharm. Запускаем его, нажимаем «Next», выбираем каталог размещения программы, отмечаем, что хотим создать ярлык на рабочем столе и привязать расширение py к данной среде, нажимаем «Next» и, затем, «Install».

После установки программа предложит запустить PyCharm. Запускаем. Появляется окно для импортирования настроек из предыдущей версии (если она была). Я этот шаг пропущу «Do not import settings». В следующем окне мы выбираем тему оформления. Я выберу темную. Вы можете выбрать другую – это дело вкуса. Далее, нажимаем на кнопку «Skip Remaining and Set Defaults» пропустить все напоминания и сделать дальнейшие установки по умолчанию.

Здесь при первом запуске необходимо создать новый проект. Нажимаем «Create New Project». В поле «Location» указывается расположение проекта и его имя. Пусть проект называется «first01». Раскрываем вкладку «Project interpreter» интерпретатор проекта, здесь укажем существующий интерпретатор. Если его в списке нет, то нажмите вот на это троеточие и в новом окне выберите «System Interpreter». В этом списке отображаются все интерпретаторы, установленные на компьютере. Но я оставлю тот, что был найден по умолчанию. Нажимаем кнопку «Create» и создаем проект. Перед нами откроется окно PyCharm. Слева отображается структура проекта. Пока он не содержит ни одного файла с текстом программы. Создадим его. Нажимаем правую кнопку мыши, выбираем «New» -> «Python File». Вводим имя файла, например, ex1 и этот файл автоматически добавляется в наш проект. Здесь мы будем писать тексты программ, например, так:

**print**("Hello World!")

Далее, для запуска можно выбрать в меню «Run» -> «Run ex1», программа начнет выполняться и внизу появится результат ее работы. Однако, удобнее пользоваться «горячими клавишами» для выполнения типовых команд. В частности для запуска проекта можно нажать комбинацию клавиш:

* при первом запуске: Ctrl+Shift+F10
* при повторных запусках: Shift+F10

Пока этого функционала нам будет вполне достаточно. По мере работы с PyCharm, вы будете знакомиться с его возможностями и совсем скоро, сами не заметите, как станете с ним одним целым.

## Форматирование текста программ

Давайте вернемся к нашей программе. Как я уже говорил, конструкции выполняются последовательно сверху вниз. Причем, все они должны иметь единый нулевой начальный отступ от левого края. Если добавить перед print() хотя бы один пробел, то возникнет ошибка форматирования текста программы. Питон очень требователен к формату записи текста программы. И первое, что здесь нужно запомнить, - все начальные конструкции языка должны записываться без отступов слева и каждый print следует начинать с новой строчки. Это правила руководства PEP 8 по рекомендациям оформления текста программы на Python:

<https://www.python.org/dev/peps/pep-0008/>

Как видите, это достаточно объемный документ. Я буду стараться придерживаться его указаний и делать оформление программ в соответствии с ним, чтобы вы также приучались к правильному стилю оформления.

На данный момент достаточно запомнить, что все начальные конструкции языка записываются без отступов слева и каждая с новой строки. Конечно, для ясности и лучшей читаемости текста, можно добавлять пустые строки, например, так:

**print**(2+3)

**print**(8/3)

**print**(5\*6)

Давайте еще раз подведем итог:

* программа исполняется последовательно сверху-вниз;
* текст программы следует оформлять в соответствии с PEP 8;
* начальные отступы слева у команд должны отсутствовать, а каждая команда записана с новой строки.

Редакторы кода (IDE) для Python

# Настройка интерфейса Pycharm, горячие клавиши под Windows,

Табуляция

Некоторые конструкции, например такие как if, elif, else требуют обозначения вложенности (отступов), т.е. после условия нужно отступить от левого края нажав кнопку пробел на четыре раза. Такой ввод можно заменить, нажав один раз кнопку **Tab**. Обратная команда **Shit + Tab**, убирает отступ.

Закомментировать код

Очень полезное и всеми любимое сочетание клавиш быстрого закомментирования текста, выручает, когда закомментировать часть кода чтобы избежать его выполнения **Ctrl + /**.

Быстрое форматирование кода по стандарту PEP 8

Скопированный код с сайта чаще всего требует форматирования, особенно это касается отступов, вручную это делать долго и нудно, поэтому существует команда **Ctrl + Alt + L**, она в миг исправит все ошибки форматирования в коде и расставит все по своим местам.

Быстрое изменение регистра

Иногда полезно быстро изменить регистр слов с верхнего на нижний, или наоборот, в этом нам поможет сочетание клавиш **Ctrl + Shift + U**.

Быстро продублировать строку поможет сочетание клавиш **Ctrl + D**.

Посмотреть определение методов, функций, переменных и т.п.

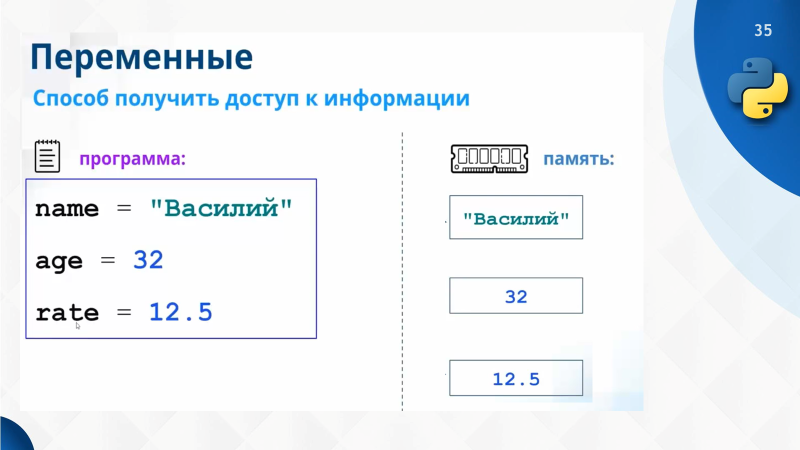
Если зажать **Ctrl** и кликнуть например на переменную, то мы попадем в блок кода где эта переменная определена. Или CTRL B

# Оператор присваивания. Типизация данных

На данный момент, мы с вами установили сам язык и научились запускать программы в средах IDLE и PyCharm. Пришла пора сделать первый шаг непосредственно в программирование. Вначале, давайте попробуем понять, что вообще могут делать компьютерные программы? В самой основе, совсем немного:

* хранить данные;
* выполнять арифметические операции;
* проверять логические условия (операторы ветвления);
* реализовывать циклы.

Фактически, на комбинации этих блоков и выстраивается логика всех программ. Мы с вами последовательно познакомимся с каждым из них. Начнем с самого первого – способа представления и хранения данных в Python.



Как же данные представляются в Python? Условно, это можно представить так. Есть некое хранилище (мы его будем называть объектом) и в нем могут располагаться или числа, или строки, или, какие-либо другие типы данных. Причем в объекте может быть что-то одно: или число, или строка.

Одновременно и число и строка находиться в одном хранилище не могут.



Таких хранилищ в программе может быть огромное количество. И как нам тогда обратиться к нужному и взять оттуда данные? Все просто. Для этого у нас должна быть ссылка на объект и мы обращаемся к хранилищу по имени этой ссылки. Такие ссылки называются **переменными**.



Как нам создать объект с некоторым содержимым и ссылкой на него? Тоже очень просто. Достаточно придумать имя переменной и через оператор присваивания связать ее с нужным объектом, например, так:

a = 7

В результате, интерпретатор языка Python создаст ссылку с именем a и объектом с целым числом 7. Мы в этом можем легко убедиться, если выведем содержимое объекта по этой ссылке (по переменной a):

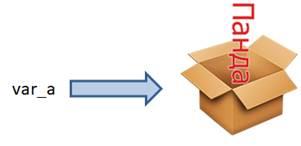
**print**(a)

Еще раз обратите внимание на оператор присваивания. В программировании он связывает операнд слева с операндом справа:

операнд слева = операнд справа

Например, строчка:

var\_a = "Панда"



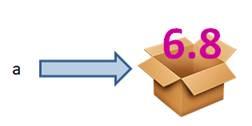
создает переменную с именем var\_a, которая ссылается на объект со строкой «Панда».

И, смотрите, здесь именно создается переменная var\_a, так как до этого она не существовала. К несуществующим переменным мы обращаться не можем. Например, строчка:

**print**(x)

приведет к ошибке, так как переменную с именем x мы нигде не создавали. Но если переменная уже была создана, и мы снова присваиваем ей какое-либо значение:

a = 6.8



то она второй раз уже не создается. Но как будет работать эта строчка? Здесь создается новый объект со значением 6.8 и на него инициализируется уже существующая переменная.

Если на прежний объект 7 нет других ссылок, то он автоматически удаляется из памяти устройства. При этом переменная a начинает ссылаться на другой тип данных – вещественное число (до этого было целочисленное значение).



То есть, тип переменной определяется в момент присваивания ей того или иного значения. В программировании это называется **динамической типизацией**. В противовес строгой типизации, когда тип переменной указывается в момент ее объявления.

Например, так делается в языках C++ или Java

Но, вернемся к оператору присваивания. Давайте посмотрим, что будет, если связать одну переменную с другой:

b = a



Теперь обе переменные будут ссылаться на один и тот же объект. То есть, копирования данных (создание нового объекта) не происходит, копируется лишь ссылка на объект.

И это важный момент, который хорошо следует запомнить и понимать! В языке Python переменные не хранят значения, а лишь ссылаются на них.

Как раз, благодаря этому можно одной и той же переменной присваивать самые разные типы данных:

b = "Hello"

b = 0

b = -8.4

Это очень удобно при программировании. Но при этом всегда следует помнить, что переменная – это всего лишь ссылка на данные, а не сами данные.

Иногда, в программах можно встретить вариант **каскадного присваивания**:

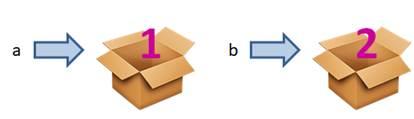
a = b = c = 0



В результате выполнения такой команды, все три переменные будут ссылаться на один и тот же объект со значением 0.

Если же мы хотим, чтобы каждая переменная ссылалась на свой отдельный объект, то можно воспользоваться **множественным присваиванием**:

a, b = 1, 2



Здесь переменная a будет ссылаться на 1, а b – на 2.

Используя такую команду, можно легко и просто выполнить операцию обмена значениями между двумя переменными:

a, b = b, a

Python — один из языков, который строго относится к типам данных. Поэтому на любую несовместимость типов он ответит ошибкой. Все дело в сильной типизации.

Нам известно про два разных типа данных: числа и строки. Например, мы могли складывать числа, потому что операция сложения — это операция для типа «числа». А что, если применить эту операцию не к двум числам, а к числу и строке?

print(1 + '7') # TypeError: unsupported operand type(s)...

Python не разрешит сложить число 1 и строку '7', потому что это значения разных типов. Нужно сначала либо сделать строку числом, либо число строкой. Как это сделать, мы поговорим позже.

Такое педантичное отношение к совместимости типов называется строгой типизацией или сильной типизацией. Python — язык со строгой типизацией.

Не все языки так делают. Например, PHP — это язык со слабой типизацией. Он знает о существовании разных типов, но относится к их использованию не очень строго. PHP пытается преобразовывать информацию, когда это кажется разумным. То же самое относится к JavaScript:

**Функция type()**

Так как в программе переменные могут иметь самые разные типы данных, то как можно узнать текущий тип, на который они ссылаются? Для этого в Python имеется специальная встроенная функция type(), возвращающая тип данных, связанный с указанной переменной:

**print**(type(a))

В консоли увидим целочисленный тип данных (int). Давайте объявим еще две переменные:

x = 5.8

s = "Hello"

И выведем для них типы:

**print**(type(x), type(s))

Увидим float и str.

Здесь у вас может возникнуть вопрос, а какие типы данных вообще существуют в Python? По мере прохождения курса, мы с вами познакомимся со всеми встроенными типами и постепенно у вас сложится общее представление. Но сначала рассмотрим как именовать переменные.

## Правильные имена переменных

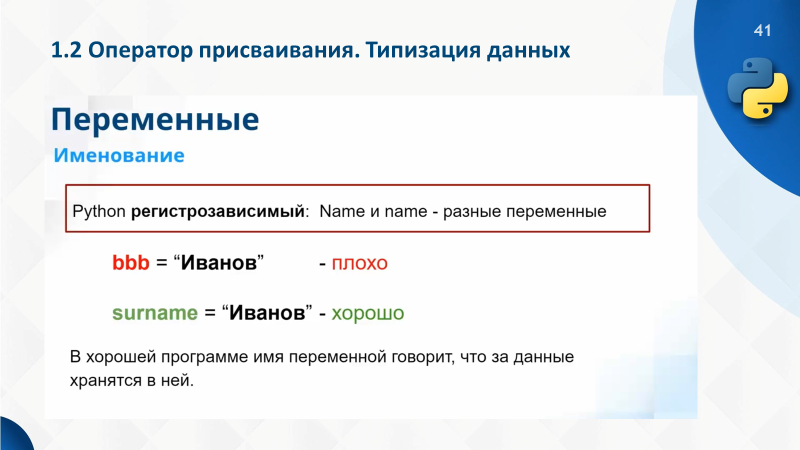
Есть несколько простых правил:



Например:

msg = "Сообщение"

count = 0



Причем, обратите внимание, переменные:

arg = 0

Arg = 0

Это две разные переменные, так как малая буква ‘a’ и большая ‘A’ – разные символы, а значит, имена тоже разные. Также нельзя использовать ключевые слова языка Python в качестве имен, например, писать:

True = 5

Полный их список можно посмотреть с помощью вызова в консоли функции:

help()

а, затем, набрать:

keywords

Также не следует использовать имена стандартных функций в качестве переменных. Например, если переопределить имя функции:

**print** = 5

то оно теперь будет ссылаться не на функцию для печати значений, а на числовой объект 5. И если, затем, попытаться вызвать функцию:

**print**(6)

то возникнет ошибка, так как мы, фактически, пытаемся выполнить объект, содержащий число 5. Но Python делать этого не умеет (по крайней мере, по умолчанию).



Поэтому имена функций стандартной библиотеки языка Python не следует использовать в качестве имен переменных.

Но как нам узнать, является ли какое-либо имя встроенной функцией? Очень просто.

Все зарезервированные имена, будь то функции или ключевые слова, автоматически подсвечиваются интегрированными средами и так подсказывают нам, что это имя уже имеет свою функциональность и его лучше не трогать.



В Python стандарт snake\_case, в JS – camelCase.

Со временем, вы запомните все наиболее употребительные имена языка Python и будете знать, какие имена лучше не использовать для собственных переменных.

Надеюсь, вы теперь понимаете, что из себя представляют переменные в языке Python, как они связываются с данными и как работает оператор присваивания.

## Типы данных для представления чисел

В Питоне имеются три базовых типа для представления чисел:

* int – для целочисленных значений;
* float – для вещественных;
* complex – для комплексных.

Мы затронем первые два: int и float. Первый целочисленный тип представляет собой, следующие числа:

0, 1, 2, 100, 6697959484, -1, -2, -7567658

Python поддерживает работу с очень большими числами, поэтому у вас, скорее всего не возникнет проблем с выходом за пределы диапазона.

Вещественные числа, то есть, дробные записываются через точку, например, так:

6.8, -5.567, 345.546, -65467.99

Также Python поддерживает экспоненциальную нотацию записи вещественного числа.

num = 13.4

print(num)

# 1.5 умножить на 10 в степени 2

num = 1.5e2

print(num)

## Сколько весят переменные

a = 1  
b = 1.0  
c = **'1'**d = 1e1  
  
print (a.\_\_sizeof\_\_()) #28  
print (b.\_\_sizeof\_\_()) # 24  
print (c.\_\_sizeof\_\_()) #50  
print (d.\_\_sizeof\_\_()) #24

Что в итоге?

|  |  |  |
| --- | --- | --- |
| Тип | Длина на 32 bit | Длина на 64 bit |
| Int | 12 | 24 |
| float | 16 | 24 |
| str | 21+длина | 37+длина (49\*) |
| tuple | 12+4\*длина | 24+8\*длина |
| list | 20+4\*длина | 40+8\*длина |
| Set/ frozenset | (<=5 элементов) 100 (>5 элементов) 100+8\*длина | (<=5 элементов) 200 (>5 элементов) 200+16\*длина |
| dict | (<=5 элементов) 124 (>5 элементов) 124+12\*длина | (<=5 элементов) 248 (>5 элементов) 248+24\*длина |

## Аннотация простых типов данных

Полезной особенностью языков со статической типизацией является то, что тип значения переменной всегда известен. Например, мы знаем, что строковые переменные могут быть только строками, int может быть только целым числом и т.д.

С динамически типизированными языками можно только догадываться, каким является или должно быть значение переменной.

В отличие от старых версий Python, аннотации типов пишутся не в комментариях или docstring, а непосредственно в коде. С одной стороны, это ломает обратную совместимость, с другой — явно означает что это часть кода и может обрабатываться соответственно.

Аннотации для переменных пишут через двоеточие после идентификатора. После этого может идти инициализация значения. Например,

price: int = 5  
title: str  
  
print (price,str, sep=**' '**)

Ошибка! Переменная в Python создается в момент в момент первого присваивания, а не “объявления”

## Ссылочная модель в Python, функция id(), оператор is.

Ранее мы ввели понятие «ссылки» и упомянули, что в Python все и всегда передается по ссылке. Но для начала нужно узнать о паре полезных для наших экспериментов инструментов — функции id и операторе is.

Функция id возвращает уникальный идентификатор объекта, который вы ей передаете в качестве аргумента и по ссылке.

**Идентификатор** — это обычное число. Но каждый отдельный объект имеет уникальный идентификатор, то есть любые два разных объекта всегда будут иметь отличающиеся идентификаторы. И пусть идентификаторы не сохраняются от одного запуска Python к другому, но в рамках одного запуска связь объекта и идентификатора нерушима.

Поэтому идентификаторы удобно использовать, чтобы отслеживать передачи ссылок на объект между разными участками кода — идентификатор объекта будет одним и тем же, по какой бы ссылке мы к объекту ни обращались:

a **=** "some string"

b **=** a

id(a) *# 139739990935280*

id(b) *# 139739990935280*

**print**(a **is** b) *# => True*

Когда мы присваиваем значение одной переменной другой, фактически создается новая именованная ссылка на исходное значение. Поэтому id(a) и id(b) возвращают одинаковый результат.

Оператор is проверяет равенство идентификаторов своих операндов. В этом примере обе переменные ссылаются на один объект, поэтому проверка a is b дает True.

# Введение в строки. Ввод-вывод данных. Преобразование строк в числа

Сегодня мы с вами познакомимся с еще одним типом данных – строками.

Но прежде обратим внимание на комментарии #

И многострочную строку ‘’’

Строки в Python задаются очень просто: или в двойных кавычках:

s1 = "Панда"

или в одинарных (апострофах):

s2 = 'Panda'

Всегда, когда задаются строки, не забывайте про кавычки, если их не поставить:

s2 = Panda

то Panda будет восприниматься как переменная и возникнет ошибка.

В Python есть еще один способ определения многострочных строк. Для этого используются тройные кавычки (одинарные или двойные, неважно) и в них прописывается текст, например, так:

text = '''Я Python бы выучил только за то,

что есть популярные курсы.

Много хороших курсов!'''

Если отобразить содержимое этой строки в консоли Питона, то увидим специальный символ ‘\n’:

'Я Python бы выучил только за то,**\n**что есть популярные курсы.**\n**Много хороших курсов!'

Это один символ, отвечающий в тексте за перенос на новую строку и когда функция print() встречает его, то осуществляет такой переход:

**print**(text)

То есть, просто запомните, что для перехода на новую строку используется спецсимвол, который записывается в виде ‘\n’. Если записать просто ‘n’ – это будет символ латинской буквы n, а при добавлении слеша – превращается в символ переноса строки.

Далее, строка может вообще не содержать ни одного символа:

a = ""

Получаем пустую строку. Но если добавить хотя бы один символ, даже если это будет пробел:

a = " "

то имеем уже не пустую строку, в данном случае содержащей символ пробела.

## Базовые операции над строками

Давайте посмотрим, какие базовые операции можно выполнять со строками в Python. Например, мы хотим соединить две строки между собой:

s1 = "Я люблю"

s2 = "язык Python"

Это можно сделать с помощью оператора +, который в случае со строками выполняет их объединение (конкатенацию):

s3 = s1 + s2

**print**(s3)

Но мы бы хотели добавить пробел между словами. Сделаем это так:

s3 = s1 + " " + s2

С помощью первого оператора + добавляем пробел к первой строке, а затем, вторым оператором + добавляем вторую строку s2.

Но при использовании оператора конкатенации следует быть осторожным – он объединяет строки между собой. Например, команда:

s3 = s1 + 5

приведет к ошибке, так как операнд справа является числом, а не строкой. Если нам все же необходимо соединить строку с числом, то предварительно число нужно преобразовать в строку. Сделать это можно с помощью специальной функции str():

s3 = s1 + str(5)

Функция str() выполняет преобразование в строки разные типы данных, не только числа, например, можно указать булевое значение:

str(True)

а также другие типы данных, о которых мы еще с вами будем говорить.

Следующий оператор \*, применительно к строкам, выполняет их дублирование, указанное число раз:

"ха " \* 5

Причем, здесь мы должны указывать именно целое число, для вещественных получим ошибку:

"ха " \* 5.5

И это понятно, так как продублировать строку 5,5 раз нельзя.

Следующая функция len() возвращает длину строки (число символов в строке):

a = "hello"

len(a)

Для пустой строки получим значение 0:

len("")

И, как видите, этой функции можно передавать или переменную на строку, или непосредственно записывать строки:

len("Python")

Следующий оператор in позволяет проверять наличие подстроки в строке, например:

'ab' **in** "abracadabra"

'abc' **in** "abracadabra"

Следующая важная группа операторов – сравнения строк. В самом простом случае, строки можно сравнивать на равенство:

a == "hello"

Но сравнение:

a == "Hello"

вернет False, так как большая буква H и малая h – это два разных символа. Для сравнения на неравенство используем оператор не равно:

a != "hello"

a != "hello "

Также смотрите строка "hello" (без пробела) и строка "hello " (с пробелом) – это две разные строки и они не равны между собой.

Наконец, строки можно сравнивать на больше и меньше, например, кот больше, чем кит с точки зрения строк:

'кот' > 'кит'

Почему так? Все просто. Здесь используется лексикографический порядок сравнения. Сначала берутся первые символы (они равны), затем переходим ко вторым символам. По алфавиту сначала идет символ ‘и’, а потом – символ ‘о’, поэтому ‘о’ больше, чем ‘и’. Как только встретились не совпадающие символы, сравнение завершается и последующие символы строк игнорируются.

Если взять равные строки:

'кот' > 'кот'

то получим False, так как ни один символ не больше соответствующего другого из второй строки. Но, добавив пробел в первую строку:

'кот ' > 'кот'

получим значение True, так как при всех прочих равных условиях больше считается более длинная строка. Наконец, если у первой строки первую букву сделать заглавной:

'Кот ' > 'кот'

то получим False. Почему? Дело в том, что каждый символ в компьютере связан с определенным числом – кодом, в соответствии с кодовой таблицей. Например, в таблице ASCII мы видим, что сначала идут символы заглавных букв, а затем – прописных. Поэтому коды больших букв меньше соответствующих кодов малых букв.

Конечно, в Python используется немного другая кодировка UTF-8, но в ней этот принцип сохраняется. Мы можем легко посмотреть код любого символа с помощью функции ord():

ord('К')

ord('к')

И, как видите, для буквы ‘К’ код меньше, чем для ‘к’.

Итак, из этого занятия вам нужно запомнить, как задавать обычные и многострочные строки. Что из себя представляет символ переноса строки. Знать базовые операции со строками:

* + (конкатенация) – соединение строк;
* \* (дублирование) – размножение строкового фрагмента;
* str() – функция для преобразования аргумента в строковое представление;
* len() – вычисление длины строки;
* in – оператор для проверки вхождения подстроки в строку;
* операторы сравнения: ==   !=   >   <
* ord() – определение кода символа.

Для закрепления этого материала вас ждут практические задания, а сейчас перейдем к операторам ввода-вывода строк. Речь пойдет о двух распространенных функциях:

* print() – вывод данных в консоль;
* input() – ввод данных из стандартного входного потока (часто клавиатуры).

## Функция print()

О функции print() мы с вами уже немного говорили, и вы видели ее использование в самых простых ситуациях, например:

**print**(1)

или

**print**(3, 5, 7)

выведет указанные значения через пробел. Также можно выводить отдельные переменные:

a = -6.84

**print**(a)

результаты арифметических операций:

**print**(a \* 2 + 3)

или функций:

**print**(abs(a \* 2 + 3))

Но у этой функции имеется два необязательных именованных параметра, которые довольно часто используются в практике программирования:

* sep – разделитель между данными;
* end – завершающий символ или строка.

Давайте я покажу на примере работу этих параметров. Предположим, мы выводим три переменные:

a = -6.84

b = 7

c = 25.6

**print**(a, b, c)

Как видите, между ними автоматически добавляется пробел. Но этот символ можно поменять через параметр sep, например, так:

**print**(a, b, c, sep=" | ")

Соответственно, чтобы параметр sep «сработал», необходимо хотя бы два аргумента в функции print(). Если указать, например, один:

**print**(a, sep=" | ")

то здесь разделять нечего и он будет проигнорирован. А вот для двух переменных уже появится вертикальная черта:

**print**(a, b, sep=" | ")

Второй параметр end задает окончание строки вывода и по умолчанию:

end = '\n'

переводу на следующую строку. Это такой спецсимвол, о которых мы также еще будем говорить. Так вот, благодаря такому параметру end два последовательных вызова функции print() напечатают текст с новой строки:

**print**("Hello")

**print**("World!")

Но если в первой функции print() добавить этот параметр с пробелом:

**print**("Hello", end=' ')

**print**("World!")

то увидим в одной строчке оба слова. Разумеется, здесь параметр end с пробелом применяется только к первому print(). У второго он уже берется по умолчанию с переносом строки.

Последнее, что я хочу рассказать о функции print() – это способ вывода форматированной информации в консоль. Давайте предположим, что у нас имеются две переменные (координаты точки) x и y:

x = 5.76

y = -8

И мы хотим вывести их в формате:

«Координаты точки: x = 5.76; y = -8»

Сделать это можно несколькими способами. Первый, самый очевидный, записать все через запятую:

**print**("Координаты точки: x = ", x, "; y = ", y)

Но, начиная с версии Python 3.6, появилась возможность использовать специальные F-строки. О них мы также еще будем говорить, но здесь я приведу простой пример и вы уже сейчас сможете применять этот механизм в своих программах. Запишем функцию print(), следующим образом:

**print**(f"Координаты точки: x = {x}; y = {y}")

Смотрите, здесь перед строкой ставится специальный символ f, указывающий, что это будет F-строка. А, в самой строчке внутри фигурных скобок мы можем записывать любые конструкции языка Python. В данном случае, я просто указал переменные x и y. Видите, как это просто, наглядно и удобно. Сейчас практически всегда используются F-строки для форматированного вывода информации.

## Функция input()

Вторая функция input() служит для ввода информации, как правило, с клавиатуры. В самом простом варианте ее можно вызвать так:

a = input()

При этом переменная a будет ссылаться на строку:

**print**(type(a))

И это важный момент: функция input() всегда возвращает строку. На что это может повлиять? Например, мы хотим вычислить модуль введенного числа:

a = input()

b = abs(a)

При вызове функции abs() возникнет ошибка, так как в качестве ее аргумента должно быть число, а не строка. Как решить эту проблему? Очень просто. Если мы знаем, что пользователь должен ввести число, предположим, целое число, то можно воспользоваться функцией:

a = "54"

b = int(a)

Теперь b будет ссылаться на число 54, а не строку. И наша программа примет вид:

a = input()

a = int(a)

b = abs(a)

**print**(b)

Здесь первые две строчки можно объединить и записать их так:

a = int(input())

Но функция int() преобразовывает только целые числа. Если в строке будет хотя бы один не цифровой символ:

int("64.56")

возникнет ошибка. То есть, в нашей программе пользователь обязательно должен вводить целые числа. А как тогда преобразовывать вещественные значения? Для этого есть другая функция:

float("64.56")

Поэтому, когда на входе ожидаются вещественные данные, то следует использовать именно ее:

a = float(input())

b = abs(a)

**print**(b)

Давайте напишем программу для вычисления периметра прямоугольника. Пользователь будет вводить два числа (стороны прямоугольника), а мы, затем, вычислим периметр:

a = float(input())

b = float(input())

**print**("Периметр:", 2 \* (a + b))

Как видите, все достаточно просто. Мы вводим первое число, нажимаем Enter и вводим второе число. Вводить два числа через пробел здесь нельзя, иначе получится строка с двумя числами, разделенные пробелом и ее нельзя будет преобразовать функциями int() и float().

Также в этой программе пользователю совершенно непонятно, что нужно вводить. Давайте добавим подсказки. Для этого в функции input() первым аргументом передается строка:

a = float(input("Введите длину прямоугольника: "))

b = float(input("Введите ширину прямоугольника: "))

**print**("Периметр:", 2 \* (a + b))

Теперь стало гораздо понятнее, что нужно вводить.

Забегая вперед, отмечу, что в Python можно выстроить конструкцию для ввода значений через пробел. Это делается так:

a, b = map(float, input("Введите две стороны прямоугольника: ").split())

Здесь к каждому введенному значению применяется функция float() для преобразования в вещественные числа. Пока просто запомните эту конструкцию, в дальнейшем она станет понятной, когда мы изучим строки и функцию map().

Или, для ввода целых чисел, она будет принимать вид:

a, b, c = map(int, input("Введите три целых числа: ").split())

**print**("Периметр треугольника:", a + b + c)

На этом мы завершим изучение функций print и input. Я надеюсь, вы теперь хорошо понимаете, как использовать простейший ввод-вывод

## Практика 2

**Задание 1**. Даны длины сторон треугольника. Вычислите его периметр и площадь.

Формат входных данных

На вход программе даются три целых числа, длины сторон треугольника. Гарантируется, что треугольник с заданными сторонами существует.

Формат выходных данных

Требуется вывести два числа — периметр и площадь треугольника с заданными сторонами.

Проверочные данные

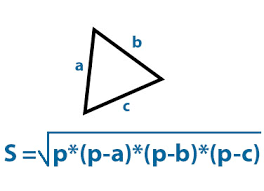
a b c S P

3 4 5 6 12

6 5 5 12 16

15 13 4 24 32

Подсказка-1. Для расчета площади треугольника воспользуйтесь "формулой Герона"



**Задача 2.**  Дано трехзначное число. Найдите его первую и последнюю цифру.

Примечание: решите задачу без преобразования числа к строке

Формат входных данных

Дано целое положительное трехзначное число.

Формат выходных данных

Требуется вывести первую и последнюю цифру числа.

Решение задачи

number = int(input("Введите трехзначное число: "))  
first = ...  
last = ...  
print("Первая цифра числа =", first)  
print("Последняя цифра числа =", last)

**Задача 3**. Гелевая ручка стоит x рублей. Сегодня магазин предоставляет скидку в 10% на каждую купленную ручку. Какое наибольшее количество таких ручек можно будет купить на 500 рублей? Результат выведите в консоль в виде целого числа.

x = int(input())

c = int(500 // (x \* 0.9))

print (c)

**Задача 4**. В летний лагерь нужно отвести n детей и m вожатых с помощью автобусов. Максимальная вместимость каждого автобуса 20 человек.

Допишите программу для вычисления минимального числа автобусов, необходимых для перевозки детей вместе с вожатыми. Результат выведите в консоль в виде целого числа.

Входные данные:

40 5

Выходные данные:

3

*import math*

*n, m = map(int, input().split())*

*a = math.ceil((n + m) / 20)*

*print (a)*

**Задача 5**. "Поменять значения местами"

Два числа хранятся в переменных. Напишите программу, меняющую значение переменных местами

Примечание: если знаете как решить задачу с использованием третьей переменной, попробуйте решить без нее.

Решение задачи

a = 5  
b = 7  
print("a =", a, "b =", b) *# Выводит a = 5 b = 7  
# Тут ваш код*print("a =", a, "b =", b) *# Должно вывести a = 7 b = 5  
# Примечание: код самих print'ов не менять!*

Подсказки

Сначала решите задачу, используя дополнительную переменную. Затем попробуйте решить задачу, без использование третьей переменной.

print(a,b)  
a = a + b *# 12*b = a - b *#12-7=5*a = a - b  
  
print(a,b)

**Задача 6. "**Бесконечно высокий дом"

Существует бесконечно высокий дом, в котором нумерация квартир начинается с единицы. Известен номер квартиры. Определите на каком этаже находится данная квартира, если всего на этаже располагается по 5 квартир.

Формат входных данных

Дано целое положительное число - номер квартиры

Формат выходных данных

Вывести целое число - номер этажа, на котором расположена заданная квартира

Примечание: постарайтесь решить задачу без использования оператора if

Решение задачи

a = 12  
  
b = (a - 1) // 5 + 1  
  
print(b)

Проверочные данные:

flat floor

1 1

2 1

5 1

7 2

9 2

10 2

12 3

45 9

46 10

**Задача 7.** Задание "Улитка"

Улитка ползет по вертикальному шесту высотой h метров, поднимаясь за день на a метров, а за ночь спускаясь на b метров. На какой день улитка доползет до вершины шеста?

Примечание: решите задачу без if'ов и циклов, выведя формулу.

Формат входных данных

Программа получает на вход три целых числа. Точно известно, что h != b.

Формат выходных данных

Программа должна вывести одно целое число - количество дней.

h a b days

10 3 2 8

20 7 3 5

10 10 4 1

h = int(input())  
a = int(input())  
b = int(input())  
print(int((h - a - 1) // (a - b) + 2))

## Форматирование строк: метод format и f-строки

На этом занятии мы подробнее поговорим о способах формирования строк на основе набора различных переменных. Мы уже знаем с вами как создавать строки и соединять их между собой. Предположим, у нас есть две переменные:

age=18; name="Александр"

и мы формируем такое сообщение:

"Меня зовут "+name+", мне "+str(age)+" и я люблю язык Python."

На выходе получим:

'Меня зовут Александр, мне 18 и я люблю язык Python.'

Но это не самый удобный и распространенный способ формирования строк по шаблону. Гораздо удобнее использовать специальный метод строки:

str.format(\*args)

который в данном конкретном случае можно использовать так:

"Меня зовут {0}, мне {1} и я люблю язык Python.".format(name, age)

Смотрите, здесь в фигурных скобках мы указываем индекс переменной, значение которой будет подставлено в это место строки. Индекс 0 – первая переменная, указанная в методе format, т.е. name, а индекс 1 – вторая переменная age. То есть, метод формат возвращает новую строку в заданном формате. И чтобы сохранить этот результат в какой-либо переменной, следует записать так:

msg = "Меня зовут {0}, мне {1} и я люблю язык Python.".format(name, age)

Теперь msg ссылается на созданную строку.

Также мы можем одну и ту же переменную указывать несколько раз, например, так:

"Меня зовут {0}, мне {1} и я люблю язык Python. {0}".format(name, age)

Как видите, это удобнее обычной конкатенации строк. Но эту запись можно сделать еще понятнее, используя именованные параметры. Для этого у каждой переменной пропишем ее имя (ключ). Оно придумывается программистом, например:

format(fio=name, old=age)

и, далее, в формате строки указываются уже эти имена:

"Меня зовут {fio}, мне {old} и я люблю язык Python. {fio}".format(fio=name, old=age)

Это уже гораздо нагляднее и понятнее. Глядя на эту строку, мы легко понимаем, что и где будет записано. Конечно, если указать неверное имя, например, так:

"Меня зовут {name}, мне {old} и я люблю язык Python. {fio}".format(fio=name, old=age)

То возникнет ошибка, говорящая, что ключа name не существует. То есть, в фигурных скобках указывается именно ключ, а не переменная. И, кроме того, благодаря использованию ключей, мы можем записывать переменные в методе format в любом порядке. Например, поменяем их местами:

"Меня зовут {fio}, мне {old} и я люблю язык Python. {fio}".format(old=age, fio=name)

На выходе получим то же самое.

Но, начиная с версии Python 3.6, появился новый, гораздо более удобный способ форматирования строк. Это, так называемые, F-строки. Здесь мы рассмотрим лишь основы использования этого подхода, для более глубокого изучения можно обратиться к стандарту PEP8:

<https://www.python.org/dev/peps/pep-0498/>

Итак, чтобы указать Python воспринимать строку, как F-строку, перед ее литералом ставится символ ‘f’ (именно малая буква). И, теперь, в фигурных скобках можно записывать любые конструкции языка Python, например, имена переменных:

f"Меня зовут {name}, мне {age} и я люблю язык Python."

Видите как это удобно?

Далее, внутри фигурных скобок F-строк можно записывать, например, и арифметические операции и методы строк:

f"Меня зовут {name.upper()}, мне {age\*2} и я люблю язык Python."

В результате, получим:

'Меня зовут АЛЕКСАНДР, мне 36 и я люблю язык Python.'

Или даже не прописывать переменные, а использовать обычные числовые операции и вызывать функции:

f"Меня зовут {len(name)}, мне {14\*2} и я люблю язык Python."

Как видите, все достаточно просто и наглядно. Теперь, в своих программах вы тоже сможете использовать F-строки, если ваш интерпретатор языка версии 3.6 и выше, либо метод format(), если F-строки не поддерживаются из-за низкой версии.

Можно встретить еще и такой способ форматирования строк

template = **"%s — главное достоинство программиста. (%s)"**print(template % (**"Лень"**, **"Larry Wall"**))

## Модификаторы форматирования

Рассмотрим на примерах

print(**f'{**10**:b}'**) *#двоичная СС*print(**f'{**10**:o}'**) *#8-ричная СС*print(**f'{**10**:x}'**) *#16-ричная СС*print(**f'{**10**:X}'**) *#16-ричная СС*print(**f'{**10**:e}'**) *# научная (экспоненциальный вид)*print(**f'{**10**:E}'**) *# научная (экспоненциальный вид)*print(**f'{**10.5**:f}'**) *# тип float (6-символов после запятой)*print(**f'{**100**:c}'**) *# символ юникода*

Модификаторы форматирования позволяют указывать, как именно будет выводиться в строке какая-либо переменная. Посмотрим на примере:

num = 8

f"Binary: {num:#b}" # число будет выводиться в двоичном представлении

'Binary: 0b1000'

num = 2 / 3

print(num)

print(f"{num:.3f}") # выводим число с точностью до 3-го знака

0.6666666666666666

0.667

Обычно, сейчас повсеместно применяют F-строки, так как они намного удобнее других способов форматирования строк.

# Математические операции

## Дополнительные арифметические операторы (syntax sugar)

В заключение этого занятия рассмотрим некоторые дополнения к арифметическим операторам. Предположим, что у нас имеются переменные:

i = 5

j = 3

И, далее, мы хотим переменную i увеличить на 1, а j – уменьшить на 2. Используя существующие знания, это можно сделать, следующим образом:

i = i + 1

j = j - 2

**print**(i, j)

Но можно проще, используя операторы:

i += 1

j -= 2

Результат будет прежним, но запись короче. Часто, в таких ситуациях на практике используют именно такие сокращенные операторы.

То же самое можно делать и с умножением, делением:

i \*= 3

j /= 4

**print**(i, j)

и другими арифметическими операторами.

+= Присвоение результата сложения

-= Присвоение результата вычитания

\*= Присвоение результата умножения

/= Присвоение результата от деления

//= Присвоение результата целочисленного деления

\*\*=Присвоение степени числа

%= Присвоение остатка от деления

Надеюсь, вам были понятны действия с арифметическими операторами. Как всегда, не забывайте проходить практику для закрепления материала.

## Математические функции

Первая встроенная функция abs() позволяет вычислять модуль чисел (из отрицательных делает положительные):

abs(-5)

То есть, для вызова функции нужно записать ее имя и в круглых скобках указать аргумент. Чтобы сохранить модуль того или иного числа, результат следует присвоить переменной:

a = abs(-5.6)

Если же передать положительное число, то оно просто возвращается данной функцией:

abs(1.5)

Следующая функция min() выбирает минимальное значение среди переданных ей чисел:

min(1, 2, 3, 0, -5, 10)

А, противоположная ей функция max() – ищет максимальное значение:

max(1, 2, 3, 0, -5, 10)

Число аргументов у этих функций может быть произвольным, но не менее одного. То есть, запись вида:

max()

приведет к ошибке, т.к. не указан ни один аргумент.

Следующая функция pow() возводит числа в указанную степень:

pow(6, 2)

Это аналог оператора:

6 \*\* 2

Или, с дробными значениями:

pow(27, 0.5)

pow(27, 1/3)

Последняя встроенная функция, которую мы рассмотрим – это round() для округления чисел:

round(0.5)

round(0.51)

У этой функции имеется второй необязательный параметр, указывающий точность округления. Если, например, записать:

round(7.8756, 2)

то число округляется с точностью до сотых (два знака после запятой). Если же указать отрицательное значение:

round(7.8756, -1)

то округление происходит до десятков. Или округление до сотен и тысяч:

round(78756, -2)

round(78756, -3)

Следующая возможность при работе с функциями – это вызов одной из другой. Например, запись вида:

max(1, 2, abs(-3), -10)

вернет значение 3, так как вначале вызываются функции в аргументах, а затем, сама функция max. Вложенность можно делать любой глубины, например:

max(1, 2, abs(min(10, 5, -3)), -10)

## Модуль math

Некоторые из вас могут заметить, что я привел достаточно ограниченный набор математических функций. Конечно, в Python есть и другие. Чтобы ими воспользоваться, нужно импортировать специальный модуль math:

**import** math

Пока просто запомните эту команду. Об импорте мы еще будем подробнее говорить. Итак, после ее выполнения у нас появляется дополнительный набор общеупотребительных функций, которые можно просмотреть так:

math.

Я отмечу наиболее используемые, а остальные применяются по аналогии.

Для округления до наибольшего целого:

math.ceil(5.2)

math.ceil(-5.2)

Для наименьшего целого:

math.floor(5.99)

math.floor(-3.3)

Факториал числа:

math.factorial(6)

Отбрасывание дробной части:

math.trunc(5.8)

Это аналог встроенной в Python функции:

int(5.8)

Далее, логарифм по основанию 2, 10 и произвольный:

math.log2(4)

math.log10(100)

math.log(2.7)

math.log(27, 3) *# по основанию 3*

Вычисление квадратного корня:

math.sqrt(49)

Также имеется полный набор тригонометрических функций, например:

math.sin(3.14/2)

math.cos(0)

Помимо этого есть, следующие константы:

math.pi

math.e

Этих представленных функций нам пока будет вполне достаточно. Для закрепления материала выполните практические задания и жду вас на следующем уроке.

## Побитовые операции

Python также поддерживает побитовые операции.

x = 4

y = 3

print("Побитовое или:", x | y)

print("Побитовое исключающее или:", x ^ y)

print("Побитовое и:", x & y)

print("Битовый сдвиг влево:", x << 3)

print("Битовый сдвиг вправо:", x >> 1)

print("Инверсия битов:", ~x)

Побитовое или: 7

Побитовое исключающее или: 7

Побитовое и: 0

Битовый сдвиг влево: 32

Битовый сдвиг вправо: 2

Инверсия битов: -5

Битовые операции, безусловно, важны для собеседований при приеме на работу в таких компаниях, как “большие” (Google, FB и т. Д.) и других местах, где ожидается и требуется глубокое знание основ.

Выполнение битовых операций архитектурно происходит намного быстрее, чем арифметических. Поэтому при обработки больших данных, оптимизации кода может сводиться к применению вместо деления, умножения - операций сдвига, побитовой инверсии и т.д.

# Логический тип bool. Операторы сравнения и операторы and, or, not

Рассмотрим, как в программах можно делать логические выводы на уровне:

True – истина;

False – ложь.

Предположим, мы бы хотели узнать:

4 > 2

Для этого достаточно записать такое сравнение и Python возвратит результат: либо True, либо False. В данном случае получаем True (истина), так как 4 действительно больше 2. Если же записать:

4 > 7

то получим False, так как 4 меньше 7. То же самое можно делать и с переменными:

a = 5

b = 7.8

a <= b

Мало того, полученный результат, при необходимости, можно сохранить в переменной, например, так:

res = a > b

получим значение False в переменной res.

Давайте теперь посмотрим на тип этой переменной:

type(res)

да, это новый тип bool (булевый тип), с которым мы только что познакомились, и он может принимать только два значения: True или False.

Мы можем присвоить эти значения напрямую переменной, например, так:

res = False

И здесь, естественно, возникает вопрос, какие операторы сравнения вообще существуют в Python? Они, следующие:

<

сравнение на меньше

>

сравнение на больше

<=

сравнение меньше или равно

>=

сравнение больше или равно

==

сравнение на равенство

!=

сравнение на неравенство

Приведу несколько примеров их использования:

5 == 7-2

Здесь справа записана арифметическая операция. Так тоже можно делать. Вообще, в качестве операндов могут быть любые конструкции языка Python: переменные, вычисления, функции и т.п.

Следующий пример, сравнения:

8 >= 8

8 > 8

В первом случае получаем True, а во втором – False, так как 8 не больше 8. Также имейте в виду, что оператор >= записывать наоборот => нельзя. Помимо этого знаки:

>=, !=, <=, ==

всегда записываются без пробелов. Если поставить пробел, будет ошибка.

Также частой, нет, очень частой ошибкой начинающих программистов – это запись вместо оператора сравнения == оператора присваивания =. Сравните две строчки:

x = 6

x == 6

В первом случае мы присваиваем переменной x число 6, а во втором случае сравниваем переменную x со значением 6. И эти операторы путать нельзя, хотя они и похожи друг на друга.

Еще несколько полезных примеров сравнения. Предположим, мы хотим проверить переменную x на четность. Я напомню, что четные числа – это, например, числа:

2, 4, 6, -8, -10

то есть, числа, кратные 2. Как выполнить такую проверку? Обычно, это делают, следующим образом:

x % 2 == 0

Помните оператор вычисления остатка от деления? Он здесь будет возвращать 0, если двойка укладывается в x без остатка. Это и есть проверка на четность.

Обратную проверку на нечетность, можно записать так:

x % 2 != 0

По аналогии мы можем проверить, кратна ли переменная какому-либо значению, например, трем:

a % 3 == 0

и так можно делать проверки с любыми числами.

# Операторы not, and, or и их приоритеты

Давайте теперь поставим более сложную задачу и определим, попадает ли число (значение переменной) в диапазон [-2; 5]? Какие логические умозаключения здесь нужно провести, чтобы ответить на этот вопрос? Для простоты представим, что у нас имеется переменная:

print (1 or 3) #?

print (1 and 3) #?

y = 1.85

Очевидно, чтобы она попадала в диапазон [-2; 5], нужно соблюдение двух условий:

y >= -2

и

y <= 5

В Python записать это можно, следующим образом:

y >= -2 and y <= 5

Здесь общее условие будет истинно, если истинно каждое из подусловий.

Противоположный пример, вывод о непопадании в этот же диапазон. Его можно сделать проверками, что:

y < -2

или

y > 5

В Python это записывается так:

y < -2 or y > 5

Оператор or выдает истину, если истинно, хотя бы одно из подусловий. Понимаете разницу между and и or? В and общее условие истинно, если истинны оба подусловия, а в or – хотя бы одно.

Также, если мы используем оператор and, то условие:

y >= -2 and y <= 5

можно записать в краткой форме:

-2 <= y <= 5

Это будет одно и то же. Какой вариант записи использовать, решать только вам, что удобнее, то и пишите в своих программах.

Приведу еще один пример. Допустим, мы хотим проверить x на кратность 2 или 3. Очевидно, это можно записать так:

x % 2 == 0 or x % 3 == 0

Также можно инвертировать это условие, то есть, определить, что x не кратна 2 и не кратна 3. Это можно сделать двумя способами:

x % 2 != 0 and x % 3 != 0

Или, используя оператор not (не):

not (x % 2 == 0 or x % 3 == 0)

Обратите внимание на круглые скобки. У оператора not самый высокий приоритет, то есть, он выполняется в первую очередь (как умножение и деление в математике). Поэтому, для инвертирования всего составного условия его необходимо поместить в круглые скобки. Если бы мы записали все в виде:

not x % 2 == 0 or x % 3 == 0

Это было бы эквивалентно условию:

x % 2 != 0 or x % 3 == 0

то есть, мы инвертировали бы только первое выражение. Поэтому, при определении составных условий очень важно знать приоритеты операторов:

or

1

and

2

not

3

Приведение к типу bool

В заключение я хочу отметить встроенную функцию

bool()

которая выполняет преобразование входного аргумента в логический тип: True или False.

Общее правило здесь такое. Пустое – это False (ложь), а не пустое – True (истина). Что значит пустое и не пустое? Покажу это на примерах. Любое не нулевое число – не пустое, то есть, истина:

bool(1)

bool(-10)

А вот 0 – пустое число, дающее False:

bool(0)

Похожим образом и со строками. Пустая строка:

bool("") # False

а любая не пустая:

bool("a") # True

bool(" ") # True

Аналогично дело обстоит и с другими типами данных, с которыми мы с вами еще только будем знакомиться.

Итак, из этого занятия вы должны знать операторы сравнения, работу операторов or, and и not, а также уметь пользоваться функцией bool().

## Практика-3

**Задание 1.** В программе определите две следующие строки:

s1 = "Hello"

s2 = "Pushkin"

С помощью двух функций print (каждая отображает соответствующую строку) отобразите строки в консоли так, чтобы они шли друг за другом через пробел.

Входные данные:

s1 = "Hello"

s2 = "Pushkin"

Выходные данные:

Hello Pushkin

**Задание 2**. Допишите программу, в которой вводятся два слова (в переменные s1 и s2) в одну строчку через пробел, и отображаются в консоли в формате:

Входные данные:

I love

Выходные данные:

Word 1: I | Word 2: love

**Задание 3.** Напишите программу ввода двух слов через пробел. Сформируйте новую строку, продублировав первое слово дважды, а второе - трижды (все слова в результирующей строке должны идти через пробел). Результат выведите на экран.

Программу следует реализовать без использования F-строк, а с применением оператора дублирования строк.

Входные данные

hello python

Выходные данные

hello hello python python python

**Задание 4.** Выполняется считывание двух целочисленных значений в переменные a и b (вводятся в одну строчку через пробел). Необходимо сформировать строку вида: "Переменная a = <значение>, переменная b = <значение>", используя оператор конкатенации (соединения) строк. Результат выведите на экран.

P. S. F-строки в программе не использовать.

Входные данные

2 -5

Выходные данные

Переменная a = 2, переменная b = -5

**Задание 5.** Написать программу ввода строки и формирования новой строчки вида: "Строка: <введенная строка>. Длина: <длина строки>". Результат сформированной строки вывести на экран.

P. S. В программе F-строки не использовать.

Входные данные

hello Pushkin

Выходные данные

Строка: hello Pushkin. Длина: 15

**Задание 6.** С клавиатуры вводятся две буквы (в одну строку через пробел). Вывести на экран следующую строку:

"Коды: <буква1> = <код буквы1>, <буква2> = <код буквы2>"

Входные данные

a z

Выходные данные

Коды: a = 97, z = 122

**Задание 7.** Вводится вещественное число. Нужно определить, что его целая часть кратна 3. На экран вывести True, если кратно и False - в противном случае. Задача делается без использования условного оператора.

Входные данные:

78.34

Выходные данные:

True

**Задача 8.** Вводится стоимость книги X рублей (например, X = 435.78) - положительное вещественное число с точностью до сотых (два знака после запятой). Требуется определить, является ли дробное значение (число после запятой) больше 50. На экран вывести True, если больше и False - в противном случае. Задача делается без использования условного оператора.

Входные данные:

456.56

Выходные данные:

True

**Задача 9**. Вводятся два целочисленных значения в одну строчку через пробел. Можно прочитать с помощью команды:

a, b = map(int, input().split())

Необходимо определить, можно ли первое число нацело разделить на второе. На экран вывести True, если делится и False - в противном случае. Задача делается без использования условного оператора.

Входные данные:

12 3

Выходные данные:

True

**Задача 10**. Вводятся три целых положительных числа. Прочитать в переменные их можно с помощью команды:

a, b, c = map(int, input().split())

Необходимо определить, можно ли их интерпретировать (воспринимать) как длины сторон треугольника. (Сумма длин двух произвольных сторон всегда должна быть больше третьей стороны). На экран вывести True, если треугольник формируется и False - в противном случае. Задача делается без использования условного оператора.

Входные данные:

8 11 12

Выходные данные:

True

a, b, c = map(int, input().split())

print (a < b + c and b < a+c and c < a+b)

**Задача 17.** Вводится вещественное число. Нужно проверить, что оно попадает или в диапазон [0; 2] или в диапазон [10; 20] (включительно). На экран вывести True, если попадает и False - в противном случае. Задача делается без использования условного оператора.

Входные данные:

1.2

Выходные данные:

True