

Знакомство с Языком Программирования Python

Курс



Оглавление

Введение	2
Термины, используемые в лекции	3
Начало работы с Python	3
Операторы ввода и вывода данных	11
Логические операции	14
Управляющие конструкции: if, if-else	15
Управляющие конструкции: while и вариация while-else	19
Цикл for, range	21
Срезы	24

Введение

Доброго времени суток, уважаемые студенты! Перед тем как начать изучать Python, я бы хотел с Вами окунуться в историю создания данного языка. Оглянемся назад, а именно в декабрь 1989 года голландец Гвидо (Guido van Rossum) — будущий создатель одного из самых популярных языков программирования — искал хобби-проект, которому можно было бы посвятить рождественские каникулы... Сам Гвидо вспоминает это время так: ***“Каждое приложение, которое мы должны были писать для Атоева, представляло собой либо shell-скрипт, либо программу на С. И я обнаружил, что у обоих вариантов были недостатки. Мне захотелось, чтобы существовал третий язык, который был бы посередине: ощущался как настоящий язык программирования (возможно, интерпретируемый), был проще в использовании, кратким и выразительным как shell-скрипты, но чтобы с читаемостью всё было не так ужасно, как у этих скриптов.”***



В качестве названия **Guido van Rossum** выбрал **Python** в честь комедийных серий ВВС "Летающий цирк **Монти-Пайтона**", а вовсе не по названию змеи.

Требовался второй язык программирования на С или С++ для решения задач, для которых написание программы на С было просто неэффективным.

Первая «официальная» версия языка увидела свет в 1994 году, когда Гвидо еще работал в CWI. Среди прочего в первой версии появились инструменты функционального программирования и поддержка комплексных чисел. Но самое главное, что следующий шаг сделал не только проект, но и его сообщество.

В том же 1994 году состоялась первая рабочая встреча пользователей Python. Встреча прошла в государственном бюро стандартов США (NBS, сегодня это государственный институт стандартов и технологий — NIST).

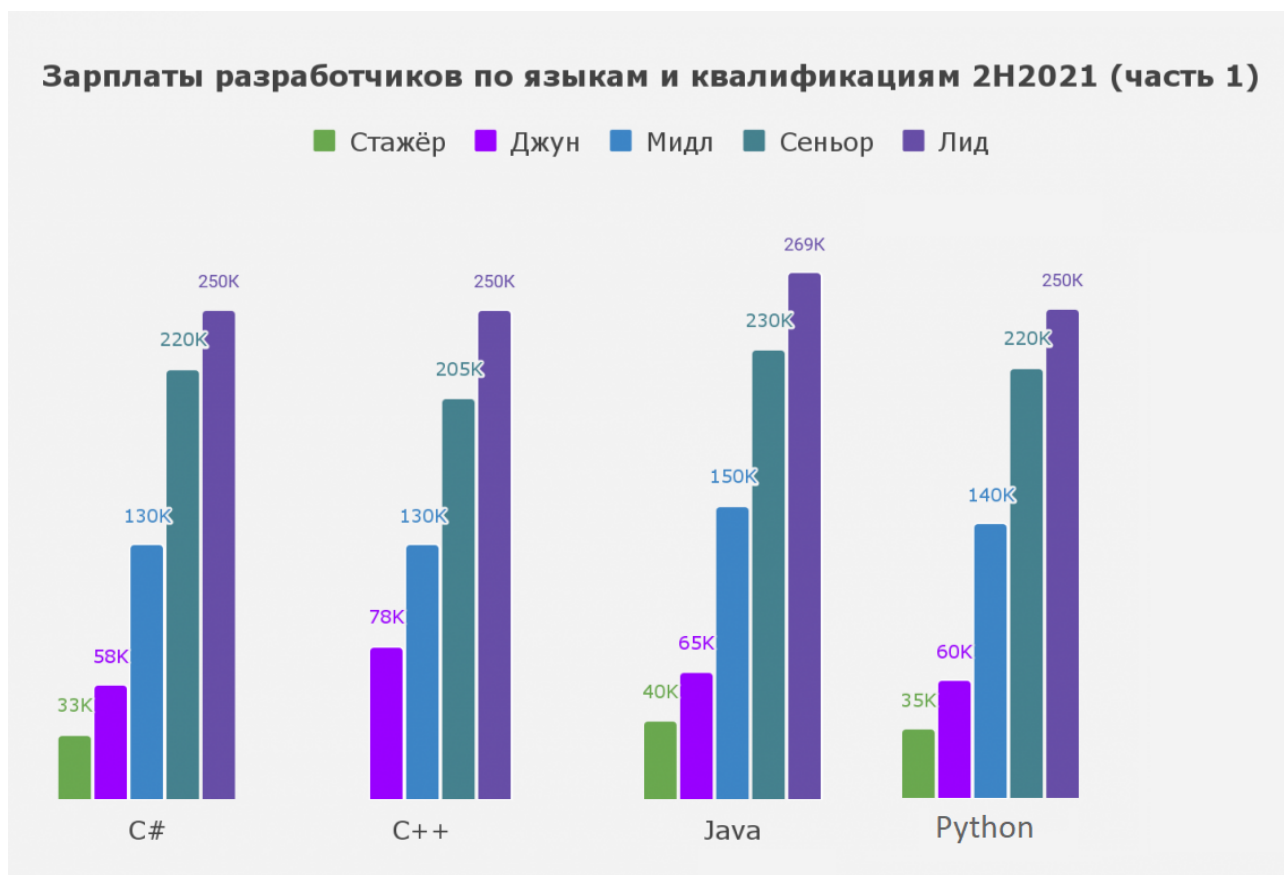
Начиная с 1994 года по настоящее время мы можем пользоваться таким удобным и простым языком Python. Почему люди после появления Python все равно пользуются такими языками как: С++, Java, С, ведь на Python можно создавать графические интерфейсы, игры, веб-приложения. Давайте разберемся.

Начнем мы с того что, Python - это скриптовый язык программирования с динамической типизацией данных. Это означает, что при создании переменных не нужно будет указывать ее тип. Python самостоятельно определяет ее тип согласно присвоенному значению. А вот Java и С++ — статические типизированные языки. Все типы переменных здесь должны быть объявлены. Если допустить ошибку, то программа работать не будет или будет, но с проблемами.

У статически типизированных языков есть недостатки, часть которых была описана выше. Но у них есть и достоинства, которых тоже немало. Так, например, Java обеспечивает безопасность типов, которая улавливает все потенциальные ошибки во время компиляции, а не в процессе выполнения, как Python. Таким образом, вероятность появления ошибок уменьшается. В конечном итоге все это упрощает управление большими приложениями. Ошибки во время выполнения (которые появляются при разработке веб-приложений, например, на Python) сложнее идентифицировать и исправлять, чем ошибки во время компиляции. Кроме того, анализировать Java-код гораздо легче, чем код Python, что полезно в ситуациях, когда над одним проектом работает команда программистов. Java-программисты быстро поймут код друг друга, поскольку все объявлено явно, а вот Python-программисты могут столкнуться с несколькими проблемами при чтении кода веб-приложения. Дело в том, что все определяется или отображается в ходе выполнения приложения, когда становятся известны переменные или сигнатуры. Собственно, ни Java, ни Python не являются лучшим вариантом для создания высоконагруженных приложений, но у первого языка есть солидные преимущества по сравнению со вторым. Все это благодаря JIT (Just-in-Time Compiler), преобразующему обычный код в машинный язык. В итоге производительность Java-приложений примерно равна производительности того, что написано на С/С++.

После слов, сказанных ранее, не нужно думать, что Python плохой язык программирования и совершенно не востребованный, ниже привожу его статистику.

Уровень профессиональных качеств разработчика является ключевым коэффициентом формирования его заработной платы. В IT сформировались 4 основные градации специалистов: Junior, Middle, Senior и Lead. Узнаем, сколько зарабатывают Python-разработчики в разрезе этих характеристик на графике:



Как мы видим Python-разработчики получают почти наравне с C++ и Java разработчиками. Но на деле зарплата C++ и Java разработчиков больше в среднем на 15-20%.

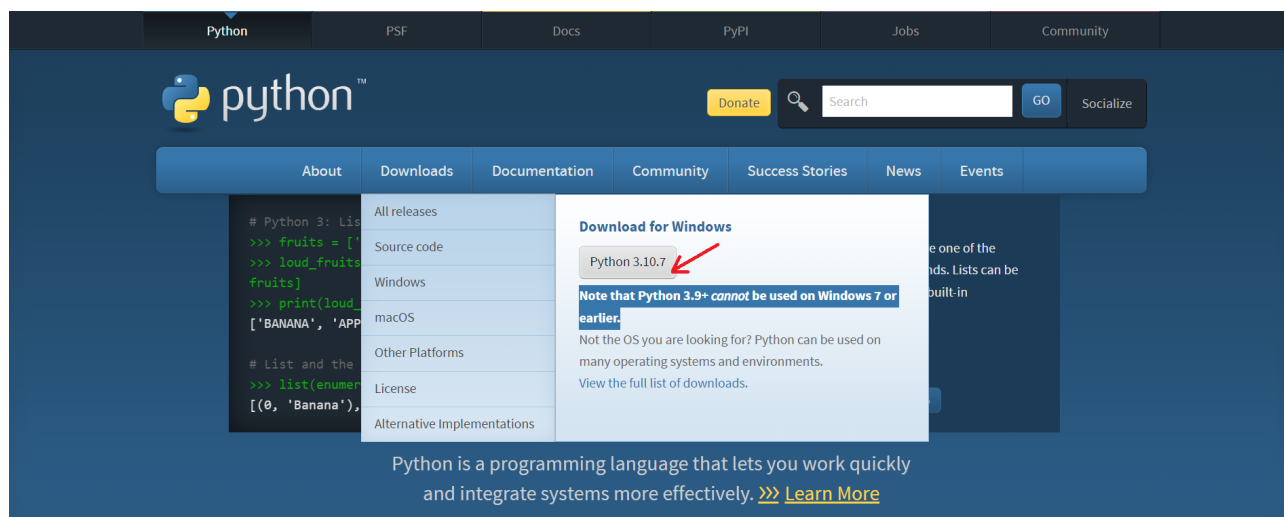
Давайте теперь перейдем к изучению одного из самых популярных языков программирования, Python.

Начало работы с Python

Перед тем как начать учить синтаксис Python, необходимо установить программу(интерпретатор), чтобы мы смогли запустить программный код. Интерпретация - это построчный анализ, обработка и выполнение исходного кода программы или запроса, в отличие от компиляции, где весь текст программы, перед

запуском анализируется и транслируется в машинный или байт-код без её выполнения.

1. Необходимо установить интерпретатор: [по ссылке](#)
2. С Windows, начиная с версии 8 и выше, можно смело устанавливать последнюю версию интерпретатора, но с Windows 7 или ниже необходимо установить версию 3.8 или ниже



3. После установки откройте командную строку(cmd), введите слово “python”, если все успешно установилось, у Вас выведется сообщение: “Python 3.10.7...”, какая именно версия была установлена

```
Командная строка - python
Microsoft Windows [Version 10.0.19044.2006]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\79190>python
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Интерпретатор успешно установлен!

Мы будем работать в **Visual Studio Code**

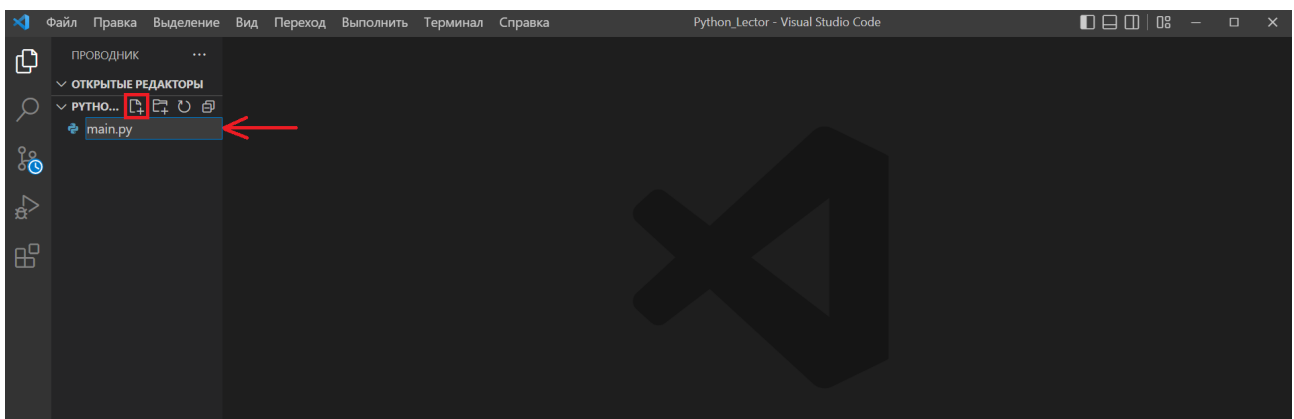
Если Вас не устраивает данная среда, в дополнительных материалах будет ссылка и текст для установки среды разработки от компании JetBrains(PyCharm)

💡 Чтобы работать было удобнее, установите расширение, которое будет подсвечивать синтаксис Python.

- Расширения(Extension) → введите в поиске «python» → Установить(install).



- В рабочем пространстве (Explorer) создайте папку с файлом для будущей программы с расширением .py (Указываем, что это файл Python).



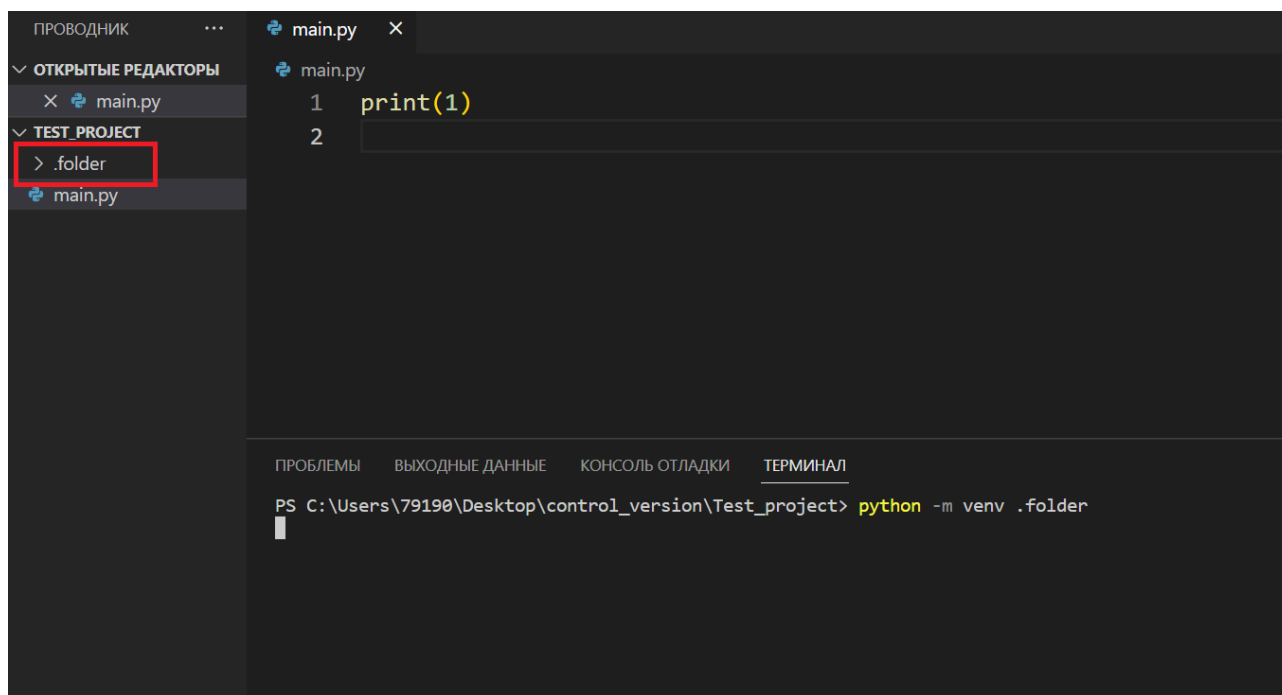
Виртуальное окружение

Мы с Вами установили интерпретатор, который позволит нам запускать пусть наши скрипты на Python. Представьте себе такую ситуацию: допустим у нас есть два проекта: **"Project A"** и **"Project B"**. Оба проекта зависят от библиотеки **Simplejson**. Проблема возникает, когда для **"Project A"** нужна версия **Simplejson 3.0.0**, а для проекта **"Project B"** — **3.17.0**. Python не может различить версии в глобальном каталоге **site-packages** — в нем останется только та версия пакета, которая была установлена последней.

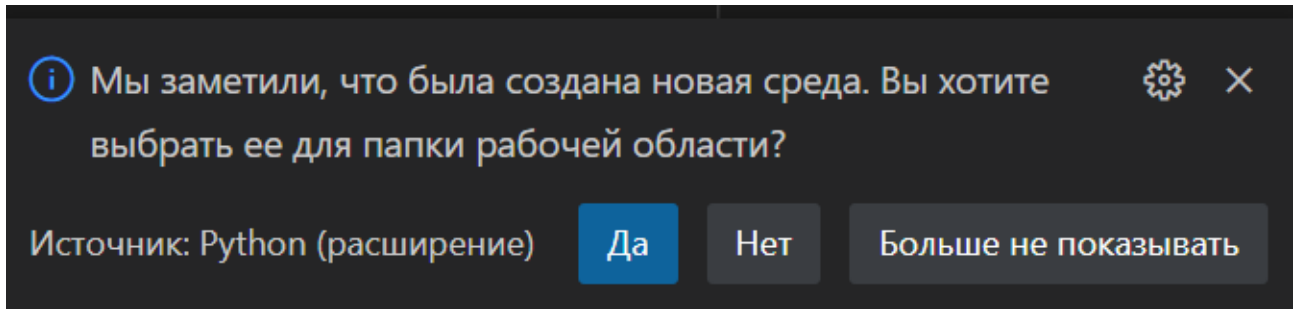
Решение данной проблемы — создание виртуального окружения (**virtual environment**).

Как добавить виртуальное окружение в свое приложение:

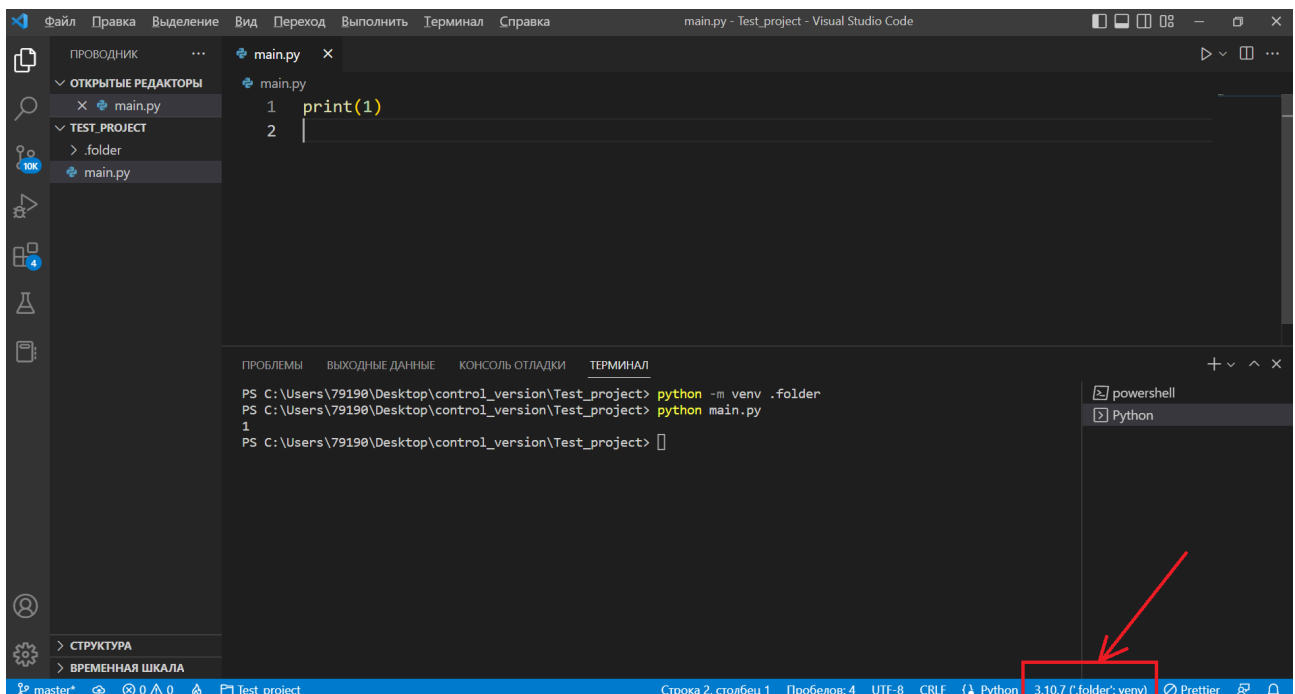
1. Создать новый проект в пустой папке (рабочего пространства).
2. В терминале прописать **python3 -m venv .folder** (.folder — название папки, в которой будет работа).



3. Выбираем “Yes” для работы с новым виртуальным окружением.



4. Справа внизу у Вас должно появиться виртуальное окружение, которое мы установили



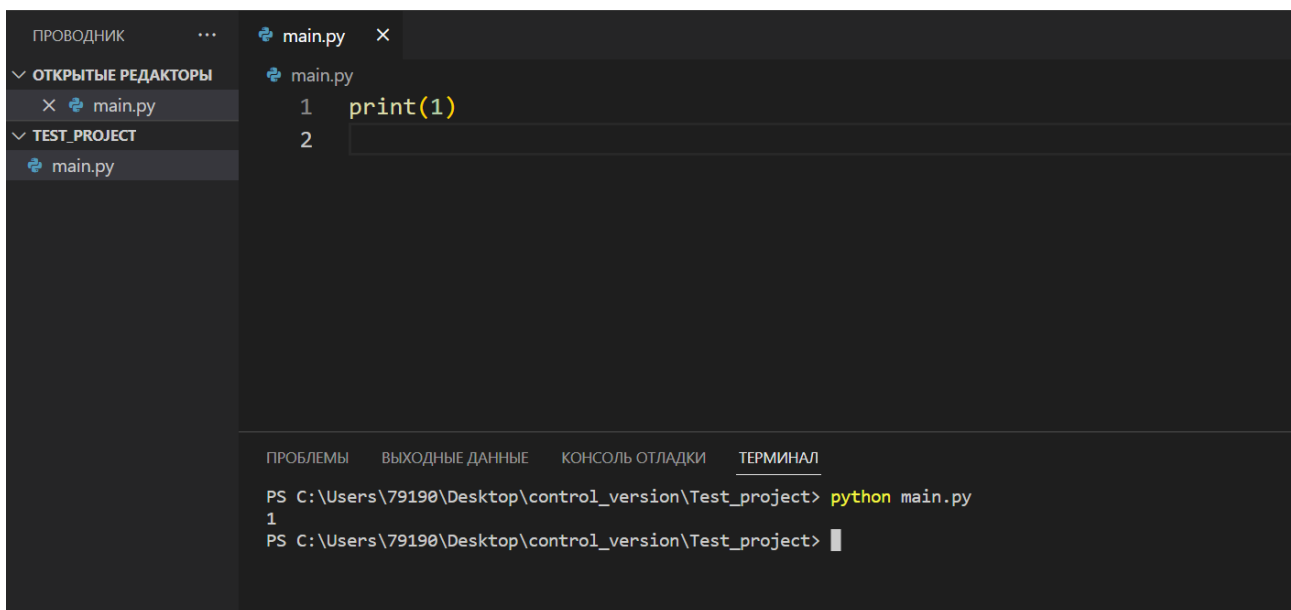
Как запустить скрипт:

💡 Если после выполнения команды в терминале остаются стрелочки >>>, нажмите ctrl + Z, чтобы выйти непосредственно в терминал.

Чтобы запустить программный код, используйте следующую команду в терминале

```
python name_file.py
```

Где name_file - имя вашего файла



```
main.py
1 print(1)
2

ПРОБЛЕМЫ Выходные данные Консоль отладки ТЕРМИНАЛ
PS C:\Users\79190\Desktop\control_version\Test_project> python main.py
1
PS C:\Users\79190\Desktop\control_version\Test_project> 
```

Здесь мы написали программный код для проверки правильности установки интерпретатора Python.

Оператор вывода данных

print(var1, var2, var3) - функция, которая выводит данных на экран, где var1, var2, var3 - переменные или значения.

Синтаксис Python

Синтаксис Python очень простой и примитивный, приведу пример с языком программирования C#, который Вы проходили ранее.

Console.WriteLine(1);	print(1)
int n = 1;	n = 1
int n = Convert.ToInt32(Console.ReadLine());	n = int(input())

Как Вы видите синтаксис Python очень простой. Давайте познакомимся с базовыми типами данных.

Базовые типы данных Python:

int	Целые числа
float	Дробные числа
bool	Логический тип данных (True/False)
str	Строка

Объявление переменной

- название переменной = значение переменной (один знак равенства обозначает присвоение значения к переменной)

Программный код:

```
a = 123
b = 1.23
print(a)
print(b)
```

Вывод:

```
123
1.23
```

💡 Нельзя указать переменную, не присвоив ей какое-либо значение. Но можно присвоить значение None и использовать переменную дальше по коду.

Программный код:

```
value = None
a = 123
b = 1.23
print(a)
print(b)
value = 1234
print(value)
```

Вывод:

```
123
1.23
1234
```

Как узнать, какой тип данных содержится в переменной value?

Возникают такие ситуации, когда мы хотим узнать тип данных у переменной, для того, чтобы это выполнить необходимо применить функции `type(varName)`.

```
print(type(name)) # функция, которая указывает на тип данных
```

Как объявить строку?

Чтобы создать строку и сохранить ее в переменную необходимо написать следующим образом:

```
s = 'hello,' # создание 1-ой строки
s = "world" # создание 2-ой строки
print(s, w)
```

Как мы видим, строку можно создавать как одинарными кавычками, так и двойными.

Как сделать комментарий?

Если Вы хотите закомментировать 1 строку достаточно применить специальный символ “#”, если Вам нужно закомментировать сразу несколько строк выделите их и нажмите **ctrl + /** или же используйте тройные кавычки `'''`

```
# print(1)
# -----
'''print(1)
print(1)
print(1)
print(1)
print(1)'''
```

Использование одинарных или двойных кавычек внутри строки

Можно ли писать кавычки в виде текста внутри строки? Пример: **my mom shouted: “good luck!”**. Но для того чтобы создать строку мы должны использовать еще одни кавычки, как это сделать?

Используйте разные кавычки для объявления переменной и внутри строки:

```
s = 'hello "world"'\ns = "hello 'world'" \ns = 'hello \'world'
```

Иногда возникают такие ситуации, когда нужно вывести в одном предложении и числа и текст, но как это сделать более рационально и красиво, обратимся к такому понятию, как **интерполяция**



Интерполяция — способ получить сложную строку из нескольких простых с использованием специальных шаблонов.

```
a = 3\nb = 11\ns = 2022\nprint(a, b, s)\nprint(a, '-b, '-s)\nprint('{} - {} - {}'.format(a,b,s))\nprint(f'first - {a}  second - {b}  third  - {s}')
```

Логическая переменная

Ранее мы с Вами проговорили основные типы данных, такие как `int`, `str`, `float` и **`bool`**.

Bool - это логический тип данных, которые используются для представления двух значений истинности логики и булевой алгебры. Он назван в честь Джорджа Буля, который впервые определил алгебраическую систему логики в середине 19 века.

Но где именно можно и нужно применять этот тип данных? На самом деле мы с Вами поговорим об этом чуть-чуть позже, когда поговорим о циклах.

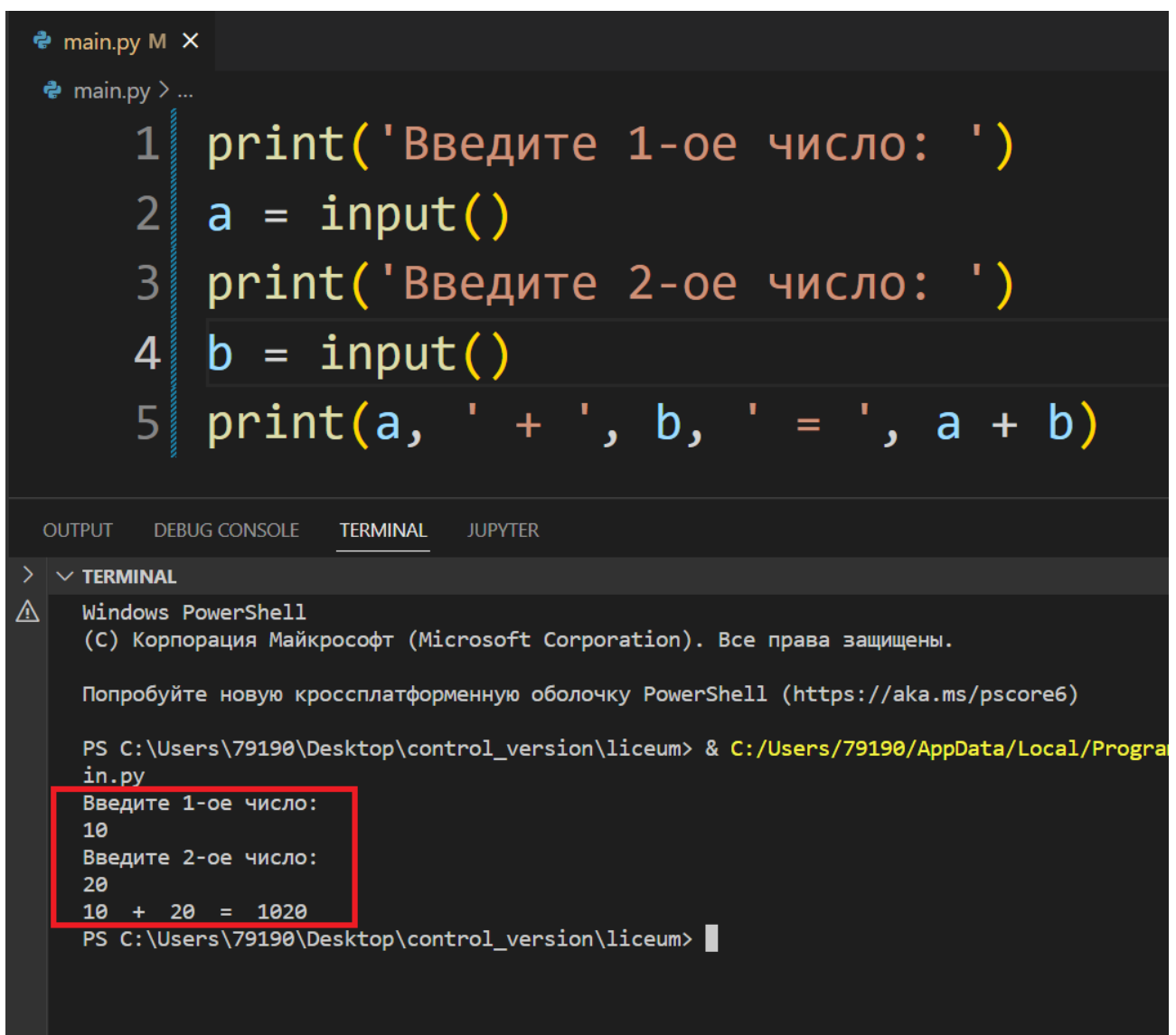
Оператор ввода данных

Как и в любом языке программирования, у Python есть операторы ввода данных. Не все так просто, как может показаться :)

- `input()` — ввод данных(строка)

Функция **input()** вводит строку, но тогда как ввести число? Давайте разбираться.

Как показать сумму двух чисел?



The screenshot shows a Jupyter Notebook interface. The top pane displays a Python script in `main.py` with five lines of code:

```
1 print('Введите 1-ое число: ')\n2 a = input()\n3 print('Введите 2-ое число: ')\n4 b = input()\n5 print(a, ' + ', b, ' = ', a + b)
```

The bottom pane shows the `TERMINAL` output. It displays the Windows PowerShell prompt and the execution of the script. The output shows the prompts and user input:

```
Windows PowerShell\n(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.\n\nПопробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)\n\nPS C:\\Users\\79190\\Desktop\\control_version\\liceum> & C:/Users/79190/AppData/Local/Programs/Python/Python39-64/Python.exe main.py\nВведите 1-ое число:\n10\nВведите 2-ое число:\n20\n10 + 20 = 1020\n\nPS C:\\Users\\79190\\Desktop\\control_version\\liceum>
```

A red rectangle highlights the input sequence: the prompt "Введите 1-ое число:", the input "10", the prompt "Введите 2-ое число:", the input "20", and the resulting output line "10 + 20 = 1020".

Когда мы ввели 2 числа(a, b). В переменных находились на самом деле не числа, а строки: `a = '10'` `b = '20'`

Поэтому при сложении получился такой результат: строки соединились.

Так как по умолчанию с помощью функции `input()` вводится строка, необходимо воспользоваться вспомогательными функциями, которые позволят вводить числа и работать с ними.

Встроенные типы

- `int()` - функция, которая позволяет перевести из любого типа данных в число(если это возможно)

Программный код:

```
n = 1.345
print(int(n)) # Отбрасывается дробная часть вне зависимости больше 0.5 или
меньше

m = '345'
print(m * 2) # При умножении строки на число, она повторяется столько раз на
какое была умножена
print(int(m) * 2)
```

Вывод:

```
1
345345
690
```

- `str()` - функция, которая позволяет перевести из любого типа данных в строку(если это возможно)

Программный код:

```
n = 1.345
print(str(n) * 2)
```

Вывод:

```
1.3451.345
```

- `float()` - функция, которая позволяет перевести из любого типа данных в вещественный(если это возможно)

Программный код:

```
n = '1.345'
print(float(n) * 2)

m = 2
print(float(m))
```

Вывод:

```
2.69
2.0
```

Примечание:

Иногда все-таки нельзя перевести один тип данных в другой.

Программный код:

```
n = '123Hello'
print(int(n))
print(float(n))
```

Вывод:

```
ValueError: invalid literal for int() with base 10: '123Hello'
```

Это ошибка типа данных. Невозможно сделать число из строки.

Познакомившись, с функциями **`int()`**, **`float()`**, **`str()`**, пора бы поговорить нам о том, как все-таки ввести число в Python?

```
n = int(input()) # 5
print(n * 2) # 10
```

Арифметические операции

Давайте посмотрим какой синтаксис в Python у базовых арифметических операций



Без них Вы не напишете ни одной программы.

Знак операции	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление (по умолчанию в вещественных числах)
%	Остаток от деления
//	Целочисленное деление
**	Возведение в степень

Приоритет арифметических операций

1. Возведение в степень (**)
2. Умножение (*)
3. Деление (/)
4. Целочисленное деление (//)
5. Остаток от деления (%)
6. Сложение (+)
7. Вычитание (-)



В Python нет лимита по хранению данных (нет ограничения по битам для хранения числа из-за динамической типизации данных)

Округление числа

Можно указать количество знаков после запятой

```
a = 1.43425
b = 2.2983
c = round(a * b, 5) # 3,29633
```

Сокращенные операции присваивания

Помните в C# внутри цикла for мы писали `i++`. Это было сокращение от `i = i + 1`.
Посмотри как можно сокращать операторы присваивания в Python

```
iter = 2
iter += 3 # iter = iter + 3
iter -= 4 # iter = iter - 4
iter *= 5 # iter = iter * 5
iter /= 5 # iter = iter / 5
iter //= 5 # iter = iter // 5
iter %= 5 # iter = iter % 5
iter **= 5 # iter = iter ** 5
```

Логические операции

Знак операции	Операция
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно (проверяет, равны ли числа)
!=	Не равно (проверяет, не равны ли значения)
not	Не (отрицание)
and	И (конъюнкция)
or	Или (дизъюнкция)

Кое-что ещё: is, is not, in, not in

В Python мы можем выполнять следующие сравнения. Результатом чего будет либо True, либо False

```
a = 1 > 4
print(a) # False
```

```
a = 1 < 4 and 5 > 2
print(a) # True
```

```
a = 1 == 2
print(a) # False
```

```
a = 1 != 2  
  
print(a) # True
```

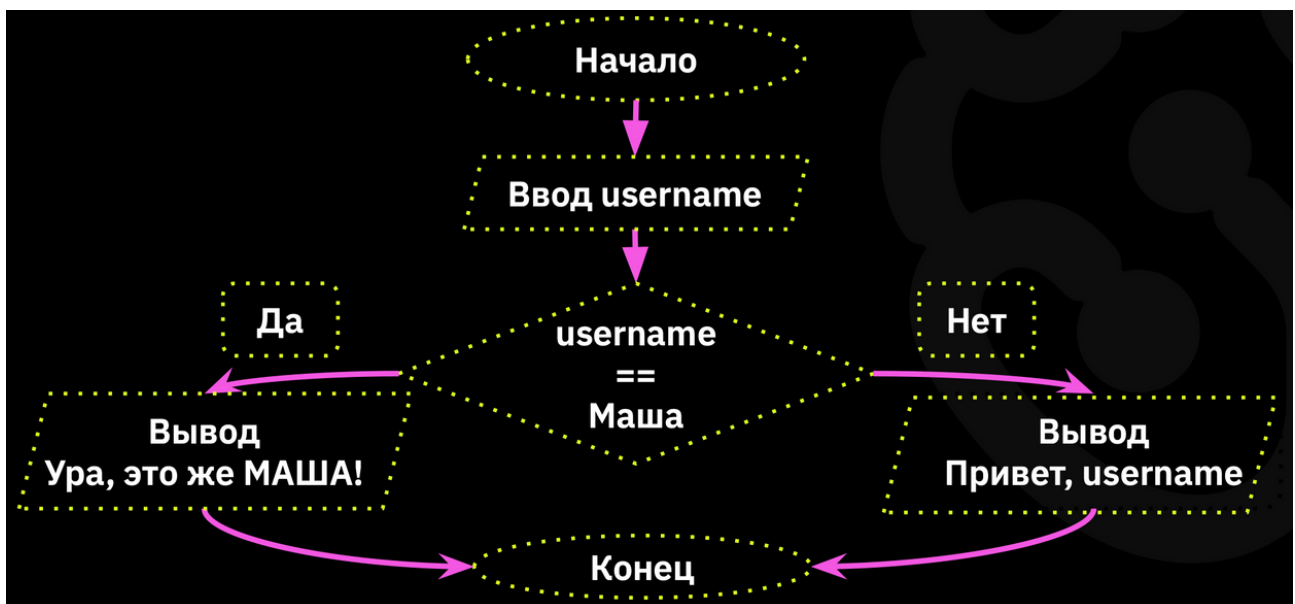
Можно сравнивать не только числовые значения, но и строки:

```
a = 'qwe'  
b = 'qwe'  
  
print(a == b) # True
```

В Python можно использовать тройные и даже четверные неравенства:

```
a = 1 < 3 < 5 < 10  
  
print (a) # True
```

Управляющие конструкции: if, if-else



Отступы в Python

Отступы в Python играют огромную роль, стоит Вам поставить на 1 пробел меньше, чем нужно, Ваша программа будет не рабочая.

Отступом отделяется блок кода, который находится внутри операторов ветвления, циклов, функций и тд. Обычно внутри VSC отступы ставятся автоматически, но Вы должны знать чему равны отступы:

- Кнопка TAB
- или
- 4 пробела

Но необязательно это кнопка **TAB** или **4** пробела, можно настроить чему равен отступ, как Вам больше нравится, мы же будем использовать вариант, который описан Выше.

Пример оформления программного кода с операторами ветвления:

```
if condition:
    # operator 1
    # operator 2
    # ...
    # operator n
else:
    # operator n + 1
    # operator n + 2
    # ...
    # operator n + m
```

```
a = int(input("a = "))
b = int(input("b = "))
if a > b:
    print(a)
else:
    print(b)
```

Ещё один вариант использования операторов else-if → в связке с elif (else if)

Проверяем первое условие, если оно не выполняется, проверяем второе и так далее. Как только будет найдено верное условие, все остальные будут игнорироваться.

```
if condition1:
    # operator
elif condition2:
    # operator
elif condition3:
    # operator
else:
    # operator
```

Пример программного кода:

```
username = input('Введите имя: ')
if username == 'Маша':
    print('Ура, это же МАША!')
elif username == 'Марина':
    print('Я так ждала Вас, Марина!')
elif username == 'Ильнар':
    print('Ильнар - топ')
else:
    print('Привет, ', username)
```

Сложные условия

Сложные условия создаются с помощью логических операторов, таких как: **and**, **or**, **not**

```
if condition1 and condition2: # выполнится, когда оба условия окажутся верными
    # operator

if condition3 or condition4: # выполнится, когда хотя бы одно из условий
    окажется верным
    # operator
```

```
n = int(input())

if n % 2 == 0 and n % 3 == 0:
    print('Число кратно 6')

if n % 5 == 0 and n % 3 == 0:
    print('Число кратно 15')
```

Управляющие конструкции: while и вариация while-else



Цикл позволяет выполнить блок кода, пока условие является верным.

Пример программного кода:

```
while condition:
    # operator 1
    # operator 2
    # ...
    # operator n
```

```
n = 423
summa = 0
while summa > 0:
    x = n % 10
    summa = summa + x
    n = n // 10
print(summa) # 9
```

Управляющие конструкции: while-else

```
while condition:
    # operator 1
    # operator 2
    # ...
    # operator n
else:
    # operator n + 1
    # operator n + 2
    # ...
    # operator n + m
```

Блок `else` выполняется, когда основное тело цикла перестает работать **самостоятельно**. А разве кто-то может прекратить работу цикла? Если мы вспомним С#, то да и это конструкция `break`. Внутри Python она также существует и используется точно также. Пример:

```
i = 0
while i < 5:
    if i == 3:
        break
    i = i + 1
else:
    print('Пожалуй')
    print('хватит ')
print(i)
```

После выполнения данного кода в консоль выведется только цифра 3, то что находится внутри `else` будет игнорироваться, так как цикл завершился не самостоятельно.

Пример программного кода без использования break:

```
n = 423
summa = 0
while summa > 0:
    x = n % 10
    summa = summa + x
    n = n // 10
else:
    print('Пожалуй')
    print('хватит ')
print(summa)
# Пожалуй
# хватит )
# 9
```

После того, как мы с Вами обговорили оператор **break** и цикл **while**, стоит рассказать почему не стоит использовать **break** и как в этом случае нам поможет Булевский тип данных? Давайте разбираться. **break** отличная конструкция, которую нельзя **не** использовать в некоторых алгоритмах, но break не функциональный стиль программирования. В ООП нет ничего, что предполагает **break** внутри метода-плохая идея, так как может произойти путаница. На замену **break** отлично подходит метод флажка.

Задача: Пользователь вводит число, необходимо найти минимальный делитель данного числа

```
n = int(input())

flag = True

i = 2
while flag:

    if n % i == 0: # если остаток при делении числа n на i равен 0

        flag = False

        print(i)

    elif i > n // 2: # делить числа не может превышать введенное число, деленное
на 2

        print(n)

        flag = False

    i += 1
```

Данный алгоритм будет работать до тех пор, пока не найдется минимальный делитель введенного числа. Когда будет найден первый делитель цикл остановит свою работу, так как условие, которое находится внутри станет ложным(False)

Цикл for, range

В Python цикл for в основном используется для перебора значений

Пример использования цикла for:

```
for i in enumeration:
    # operator 1
    # operator 2
    # ...
    # operator n

for i in 1, -2, 3, 14, 5:
    print(i)
# 1 -2 3 15 5
```

Range

- Range выдает значения из диапазона с шагом 1.
- Если указано только одно число — от 0 до заданного числа.
- Если нужен другой шаг, третьим аргументов можно задать приращение.

```
r = range(5) # 0 1 2 3 4
r = range(2, 5) # 2 3 4
r = range(-5, 0) # ----
r = range(1, 10, 2) # 1 3 5 7
r = range(100, 0, -20) # 100 80 60 40 20
r = range(100, 0, -20) # range(100, 0, -20)

for i in r:
    print(i)
# 100 80 60 40 20
```

```
for i in range(5):  
  
    print(i)  
  
# 0 1 2 3 4
```

Можно использовать цикл **for()** и со строками, так как у строк есть нумерация, такая же как и у массивов, начинается с 0:

```
for i in 'qwerty':  
  
    print(i)  
  
# q  
  
# w  
  
# e  
  
# r  
  
# t  
  
# y
```

Можно использовать вложенные циклы:

```
line = ""  
  
for i in range(5):  
  
    line = ""  
  
    for j in range(5):  
  
        line += "*"   
  
    print(line)
```

Программный код выведет 5 строк "*****". Сначала запускается внешний цикл с i(счетчик цикла). После этого запускается внутренний цикл с j(счетчик цикла). После того как внутренний цикл завершил свою работу, переменная line = "*****" и выводится на экран, далее опять повторяется внешний цикл и так 5 раз.

Немного о строках

Возникают ситуации, когда в некоторых задачах необходимо поработать со строкой, которую ввел пользователь. Например: необходимо сделать все буквы маленькими, или поменять все буквы “ё” на “е”.

Команды для работы со строками:

```
text = 'СъЕШЬ ещё этих МяГкИх французских булок'
```

1. Определить количество символов в строке:

```
print(len(text)) # 39
```

2. Проверить наличие символа в строке (in):

```
print('ещё' in text) # True
```

3. Функция, которая делает все буквы строки маленькими:

```
print(text.lower()) # съешь ещё этих мягких французских булок
```

4. Функция, которая делает все буквы строки большими:

```
print(text.upper()) # СЪЕШЬ ЕЩЁ ЭТИХ МЯГКИХ ФРАНЦУЗСКИХ БУЛОК
```

5. Заменить слово на другое :

```
print(text.replace('ещё', 'ЕЩЁ')) # СЪЕШЬ ЕЩЁ этих МяГкИх французских булок
```



`help(название функции)` — встроенная справка о функции.

Срезы

- Мы представляем строку в виде массива символов. Значит мы можем обращаться к элементам по индексам.
- Отрицательное число в индексе — счёт с конца строки

```
text = 'съешь ещё этих мягких французских булок'

print(text[0])           # с
print(text[1])           # ъ
print(text[len(text)-1]) # к
print(text[-5])          # б
print(text[:])           # съешь ещё этих мягких французских булок
print(text[:2])          # съ
print(text[len(text)-2:]) # ок
print(text[2:9])         # ешь ещё
print(text[6:-18])       # ещё этих мягких
print(text[0:len(text):6]) # сеикакл
print(text[::6])         # сеикакл
text = text[2:9] + text[-5] + text[:2] # ...
```

Итоги:

1. Изучили историю создания языка программирования Python и проанализировали востребованность на рынке
2. Познакомились с функциями ввода и вывод данных
3. Изучили операторы ветвления
4. Изучили циклы внутри Python и проговорили отличия с цикла на C#

На следующей лекции мы пройдем коллекции данных в Python, а также пройдем очень важную тему “Профилирование и отладка”, именно эта тема позволить Вам анализировать программный код.

Дополнительный материалы:

1. Как установить Pycharm? - <https://it-robionek.ru/pt/pycharm-ustanovka.html>
2. Грокаем Алгоритмы - https://en.ecomed.dgmu.ru/wp-content/uploads/2020/01/Грокаем_алгоритмы.pdf