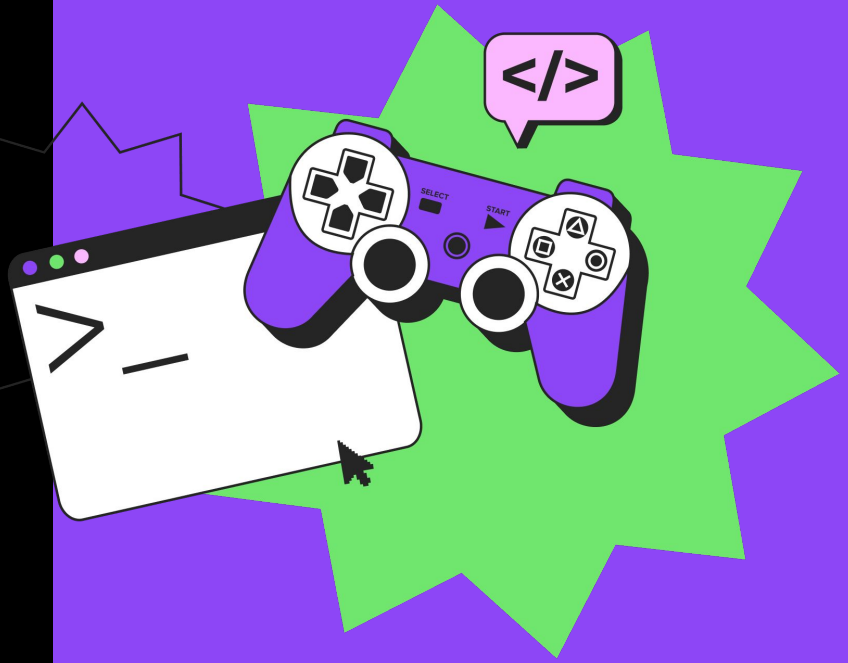
GeekBrains

Знакомство с Языком Программирования Python

Лекция 1



План курса

1. Знакомство с Python
2. Коллекции данных. Профилирование и отладка
3. Функции, рекурсия, алгоритмы
4. Функции высшего порядка. Работа с файлами
5. Google Colab. Знакомство с аналитикой

Что будет на лекции сегодня

- Немного истории языка
- Настройка Python. Начало работы
- Операторы ввода и вывода данных
- Арифметические и логические операции
- Управляющие конструкции: if, if-else
- Управляющие конструкции: while, while-else
- Цикл for, функция range()
- Срезы

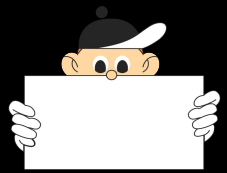
Немного истории

История языка Python

В декабре 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 Amoeba, представляло собой либо shell-скрипт, либо программу на C. 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 попроще в использовании, кратким и выразительным как shell-скрипты, но чтобы с читаемостью всё было не так ужасно, как у этих скриптов.”*

Guido van Rossum



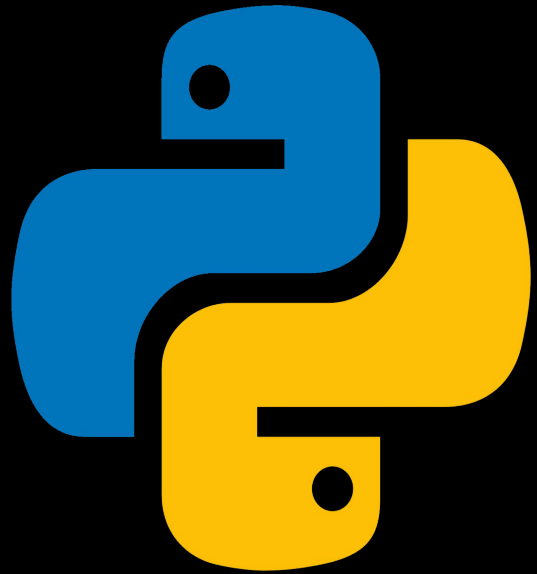
История языка Python

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

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

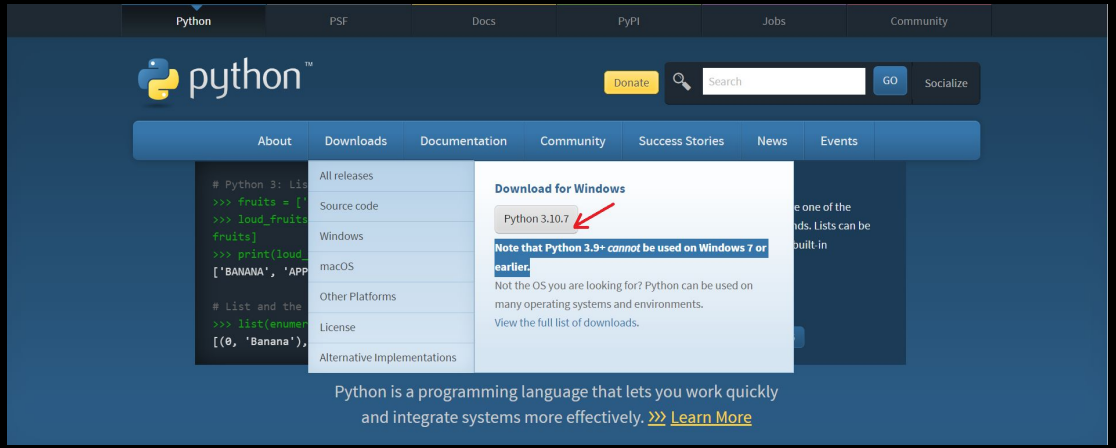
Первая «официальная» версия языка увидела свет в 1994 году, когда Гвидо еще работал в CWI. Среди прочего в первой версии появились инструменты функционального программирования и поддержка комплексных чисел. Но самое главное, что следующий шаг сделал не только проект, но и его сообщество.

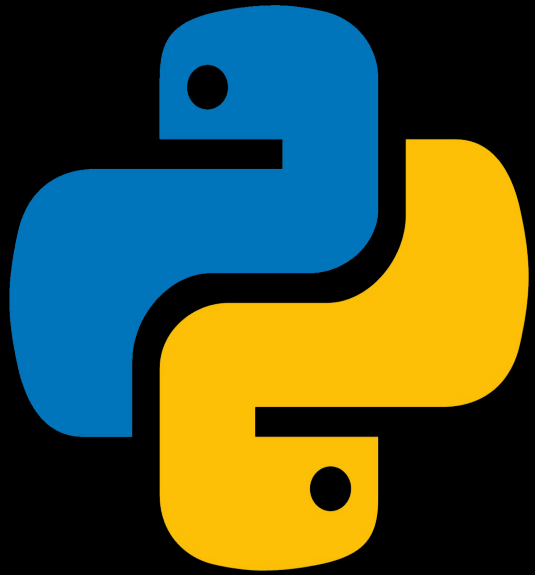
В том же 1994 году состоялась первая рабочая встреча пользователей Python. Встреча прошла в государственном бюро стандартов США (NBS, сегодня это государственный институт стандартов и технологий — NIST).



Начало работы с Python

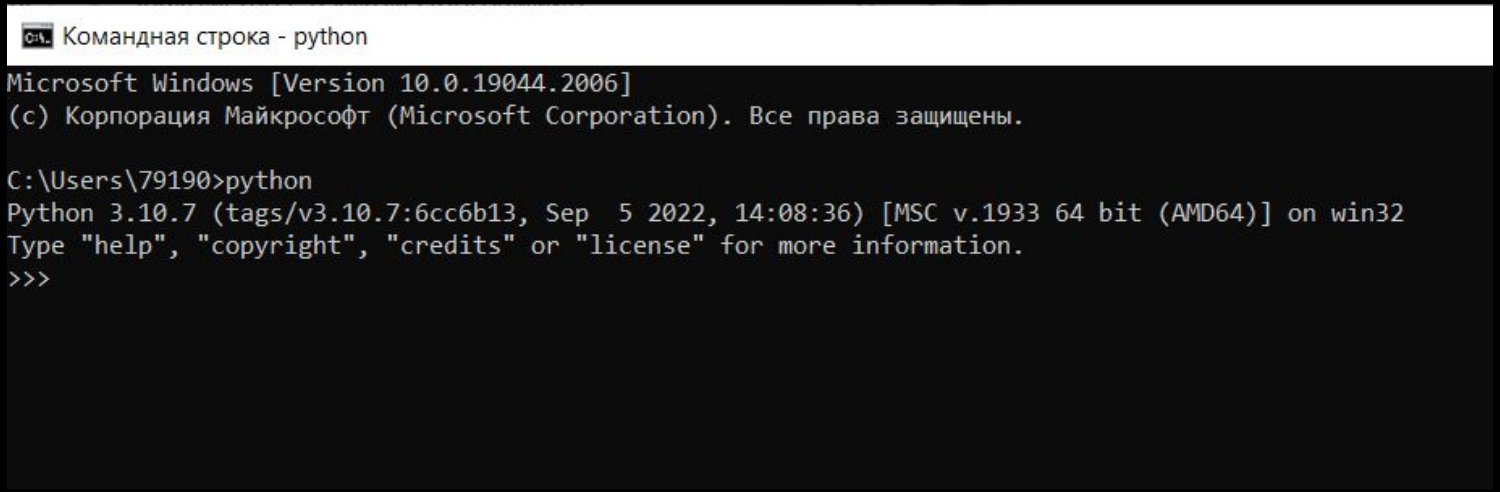
1. Python является интерпретируемым ЯП, поэтому перед тем, как начать изучать синтаксис необходимо установить интерпретатор: по ссылке
2. На macOS и Linux версию необходимо обновить, так как уже Python предустановлен
3. С Windows, начиная с версии 8 и выше, можно смело устанавливать последнюю версию интерпретатора, но с Windows 7 или ниже необходимо установить версию 3.8 или ниже





Начало работы с Python

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

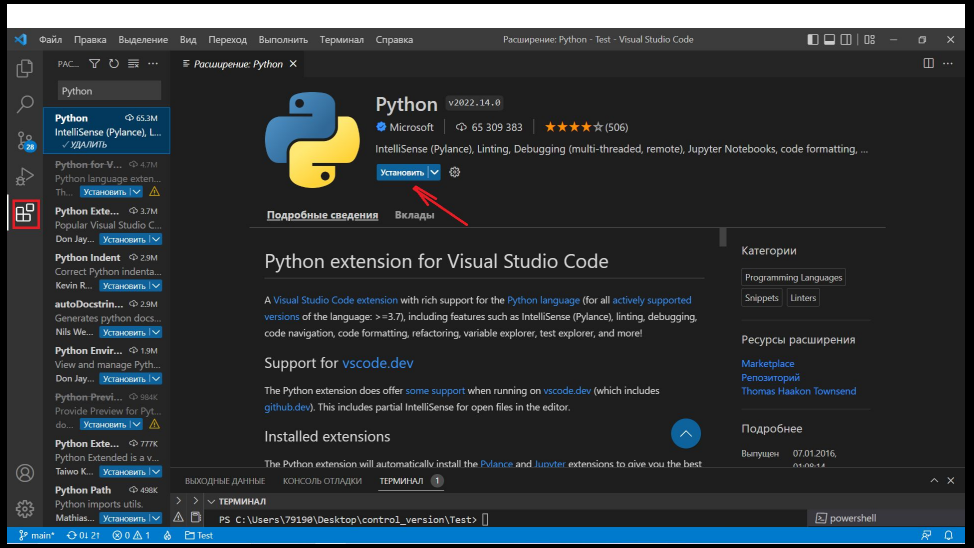


Интерпретатор успешно установлен!

Среда разработки

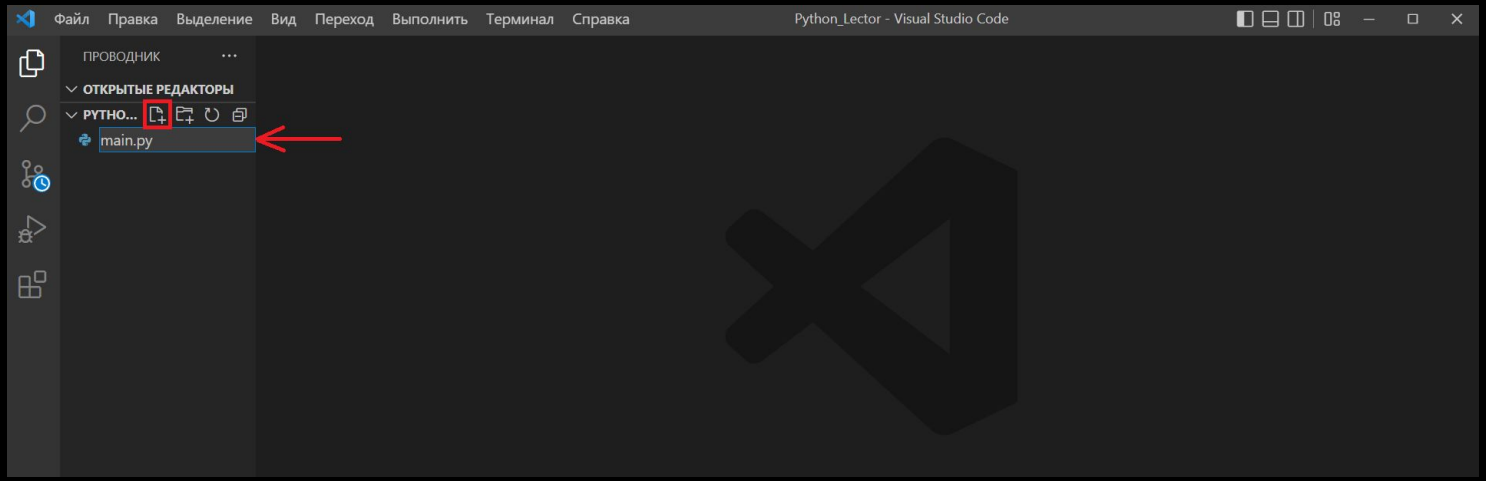
Мы будем работать в Visual Studio Code

- Чтобы работать было удобнее, установите расширение, которое будет подсвечивать синтаксис Python.
- Расширения(Extension) → введите в поиске «python» → Установить(install).



Среда разработки

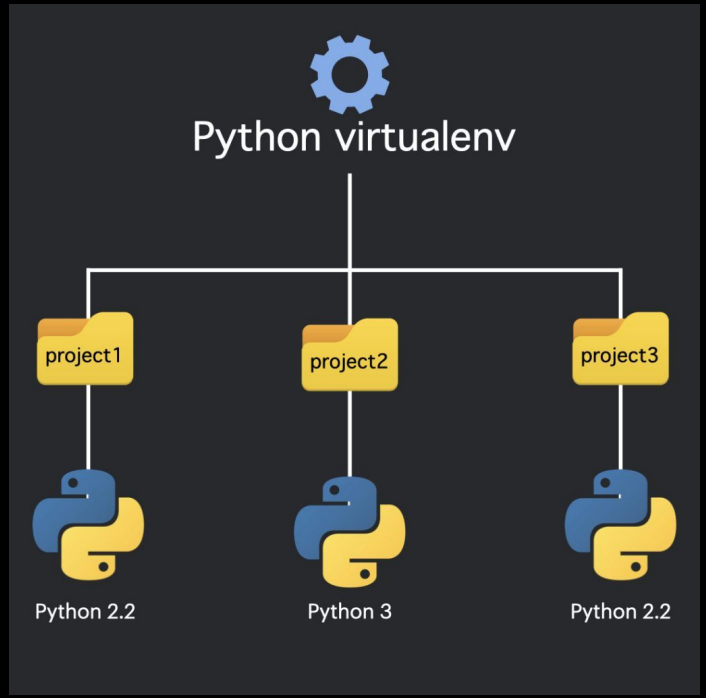
В рабочем пространстве (Explorer) создайте папку с файлом для будущей программы с расширением .py (Указываем, что это файл Python).



Пишем внутри файла следующий код:
print('Hello, world!')

Виртуальное окружение

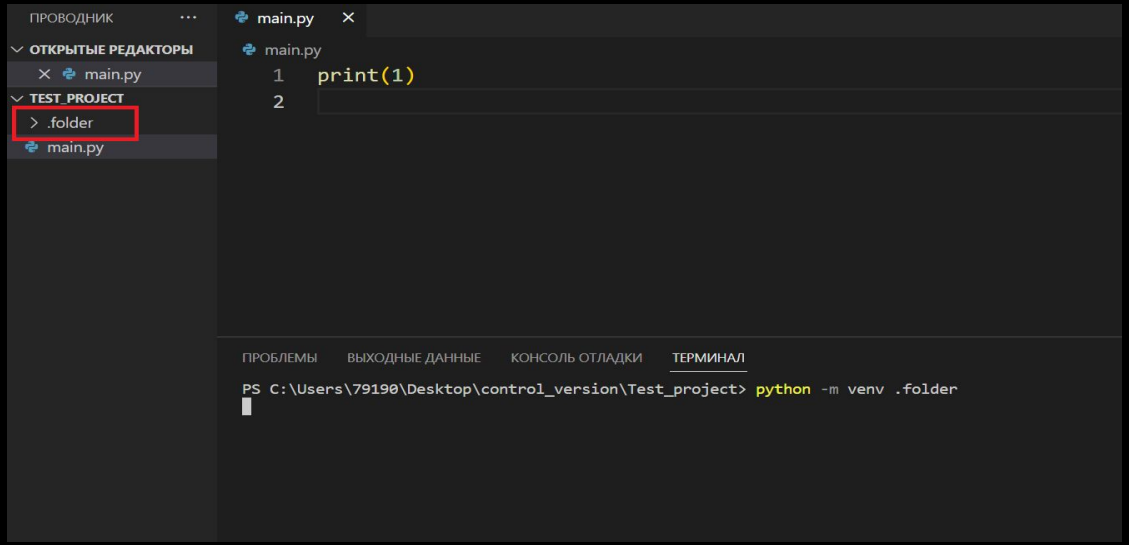
Мы с Вами установили интерпретатор, который позволит нам запускать пусть наши скрипты на Python. Представьте себе такую ситуацию: допустим у нас есть два проекта: "Project A" и "Project B". Оба проекта зависят от библиотеки Simplejson. Проблема возникает, когда для "Project A" нужна версия Simplejson 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).



Виртуальное окружение

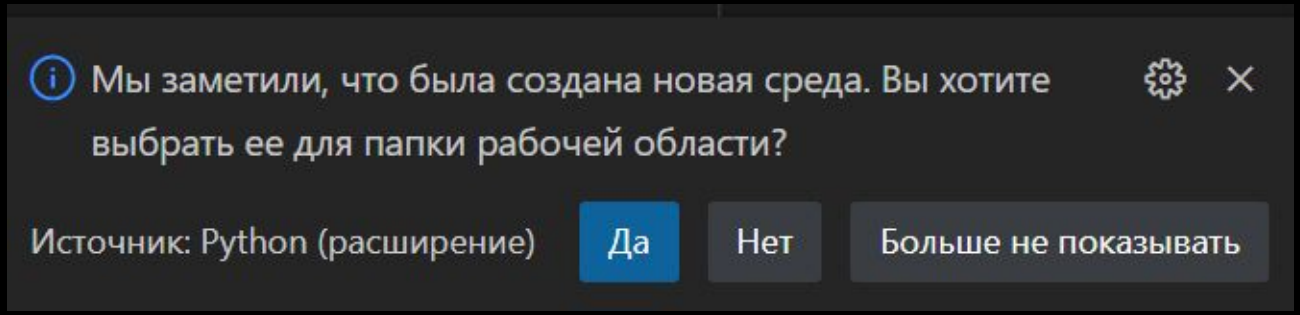
Как добавить виртуальное окружение в свое приложение:

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



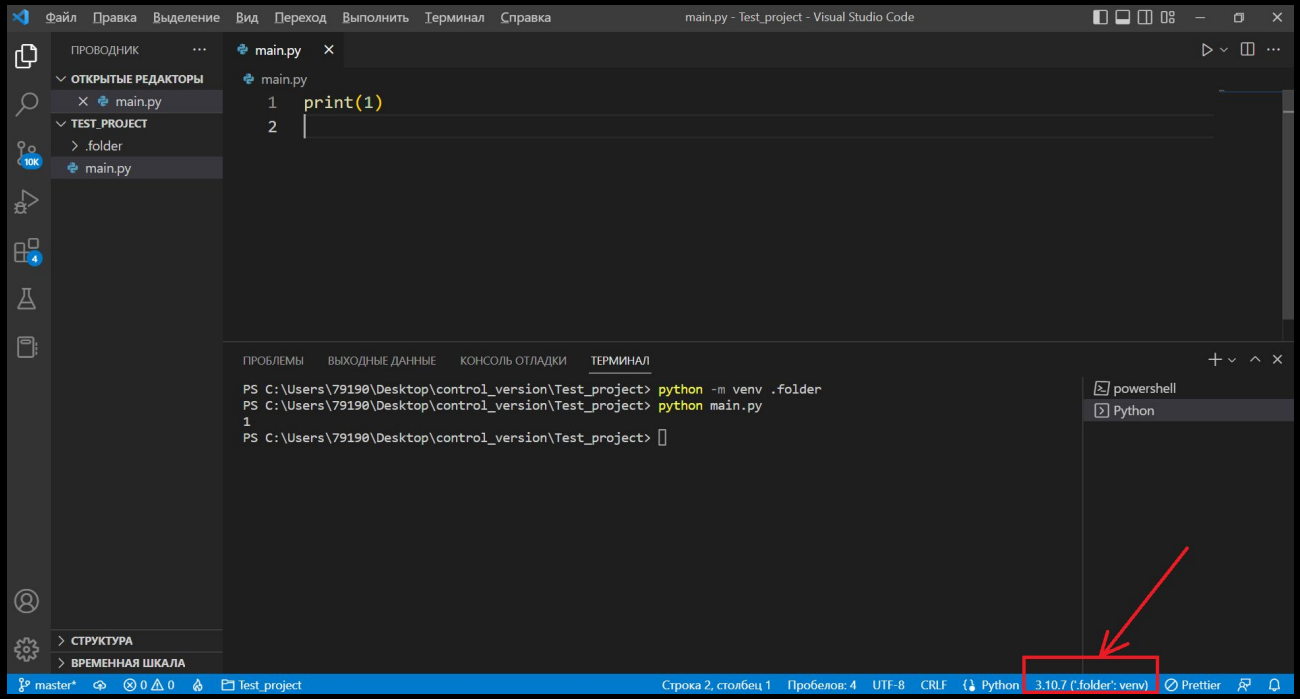
Виртуальное окружение

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



Виртуальное окружение

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



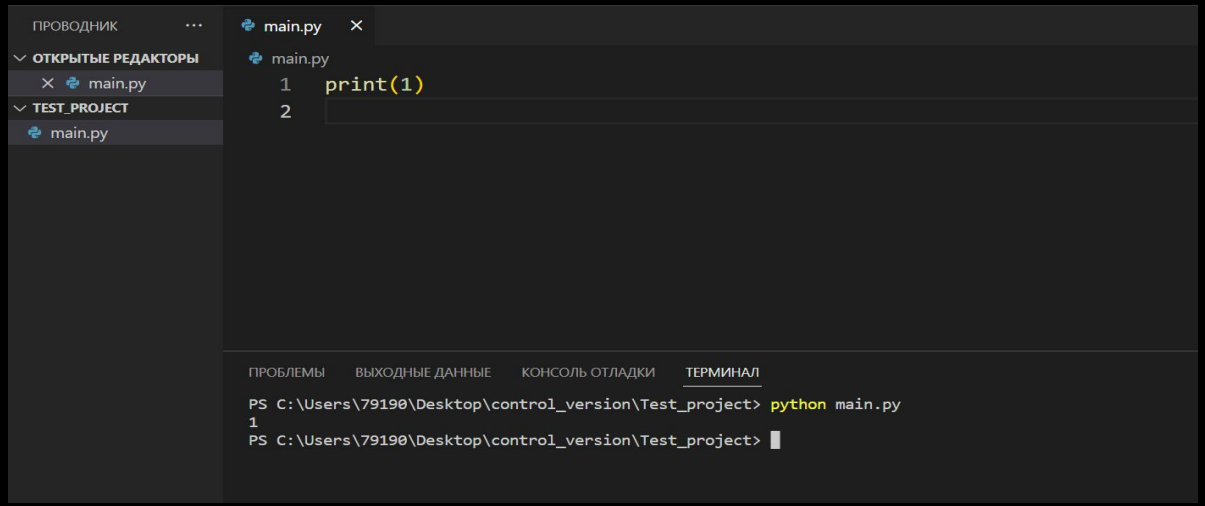
Как запустить скрипт?

Если после выполнения команды в терминале остаются стрелки >>>, то 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



Оператор вывода и ввода данных

print(var1, var2, var3) - функция, которая выводит данных на экран, где var1, var2, va3 - 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
```

Базовые типы данных Python

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 в переменной?

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

Интерполяция

Иногда возникают такие ситуации, когда нужно вывести в одном предложение 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
b = 11
s = 2022
print(a, b, s)
print(a,'-'b,'-'s)
print('{} - {} - {}'.format(a,b,s))
print(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 используется 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 :)

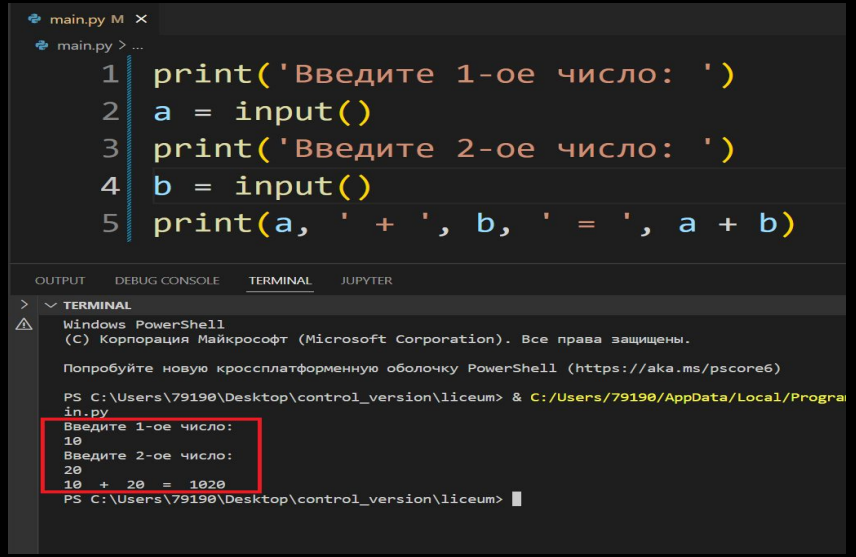
- input() — ввод данных(строка)

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

Как показать сумму двух чисел?

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 данных

- 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
```

Встроенные типы данных

- 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
```

Встроенные типы данных

- 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
```

Встроенные типы данных

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 арифметический 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)

Сравнение в Python

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
```

Сравнение в Python

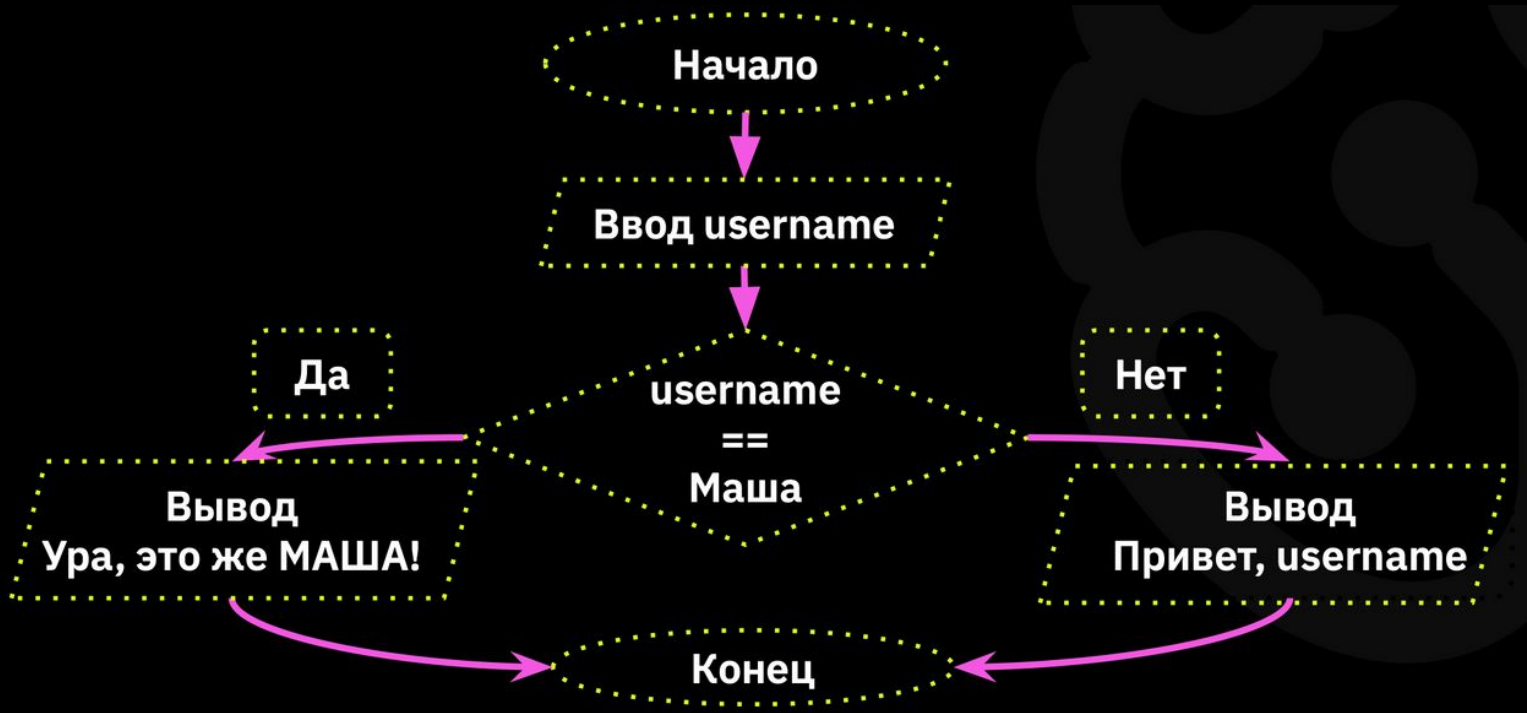
Можно сравнивать не только числовые значения, но и строки:

```
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 на следующем слайде.

Условия в Python

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
```

Условия в Python

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
```

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
```

Цикл While

Цикл позволяет выполнить блок кода, пока условие является верным.

```
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 C#, то да и это конструкция break.

Управляющие конструкции: while-else

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.

Управляющие конструкции: while-else

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
```

Управляющие конструкции: while-else

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

Решение:

```
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
```

Задача

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, функция 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
```

Цикл for, функция 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
```

Цикл for, функция range()

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
```

Цикл for, функция range()

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”.

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

print(len(text)) # 39

print('ещё' in text) # True

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

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

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

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
```

Срезы

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

```
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

- *Изучили историю создания языка программирования Python и проанализировали востребованность на рынке*
- *Познакомились с функциями ввода и вывод данных*
- *Изучили операторы ветвления*
- *Изучили циклы внутри Python и проговорили отличия с цикла на C#*

Спасибо за внимание!