

Лекция 2

Язык программирования Python.

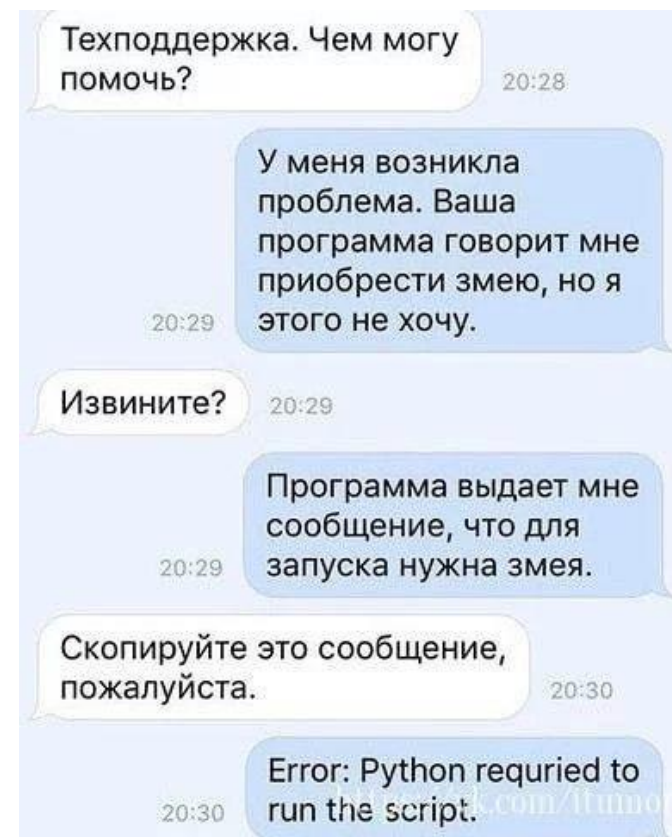
Домашнее задание

1. Установить Python, в командной строке “import this”, установить IDE
2. Разработайте приложение принимающее на вход два числа и выводящее сумму этих чисел
3. Реализовать программу, которая спрашивает у пользователя: имя, фамилию, год рождения. После ввода всех данных программа должна выводить строку следующего вида:

```
"Hello {Name} {Surname} your age is {year} year"
```

GitHub курса

<https://github.com/Daterdum/python-course>



План занятия

- Как написать и запустить программу
- Виртуальное окружение
- Алгоритмы
- Программные блоки
- Логические операторы
- Условные операторы
- Циклы
- ДЗ

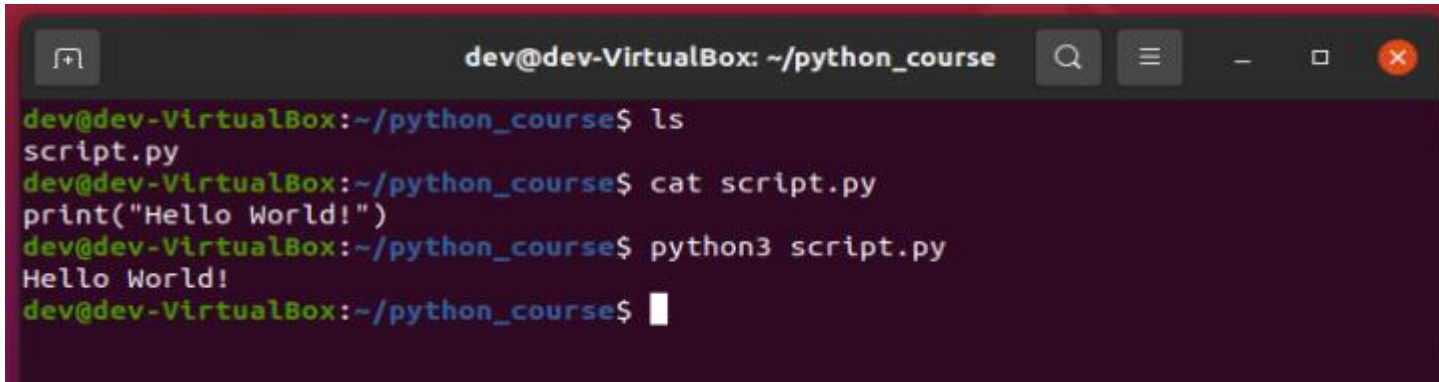
Что такое программа?

Комбинация компьютерных инструкций и данных, позволяющая аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления (стандарт ISO/IEC/IEEE 24765:2010).

Что такое программа (в терминах Python)

Набор исходного папок/файлов (пакетов/модулей) с исходным кодом. Каждый модуль содержит набор команд для интерпретатора выполнение которых начинается с точки входа, обычно мы явно указываем интерпретатору какой модуль(файл) является точкой входа для программы. Разберемся на примере...

Как написать и запустить программу

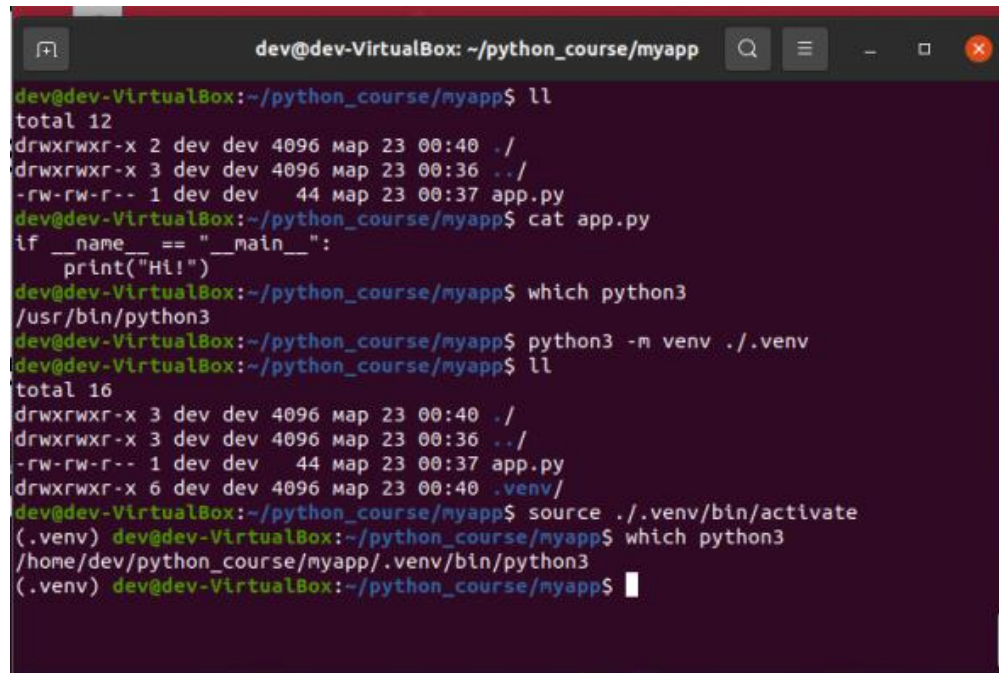
A terminal window with a dark background and light-colored text. The window title bar shows 'dev@dev-VirtualBox: ~/python_course' and standard window controls. The terminal content shows a series of commands and their outputs: listing files, viewing the content of 'script.py', and running it with Python 3, which prints 'Hello World!'.

```
dev@dev-VirtualBox: ~/python_course
dev@dev-VirtualBox:~/python_course$ ls
script.py
dev@dev-VirtualBox:~/python_course$ cat script.py
print("Hello World!")
dev@dev-VirtualBox:~/python_course$ python3 script.py
Hello World!
dev@dev-VirtualBox:~/python_course$
```

Виртуальное окружение

Изолированная среда для разработки приложения (Конкретная версия интерпретатора и отдельных библиотек)

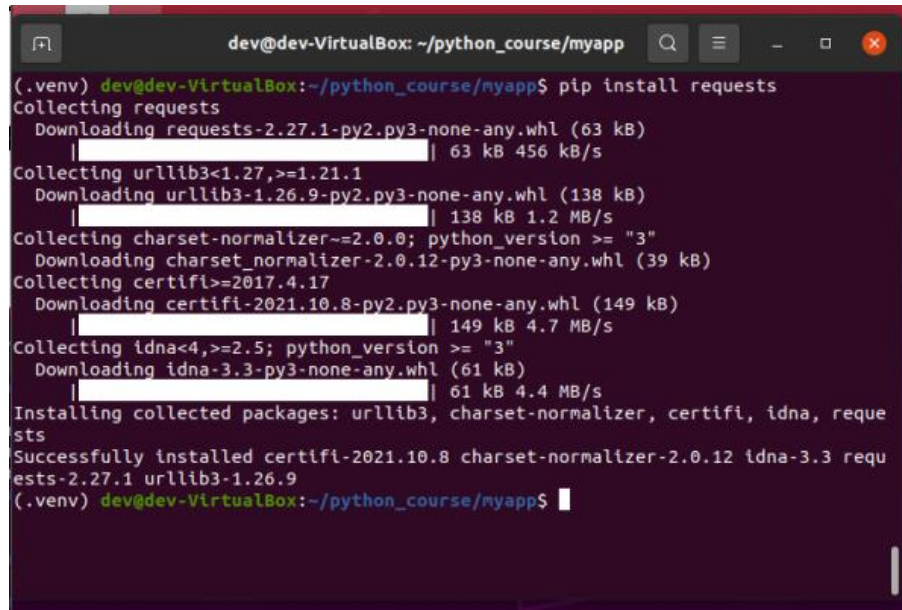
Создание и подключение виртуального окружения



```
dev@dev-VirtualBox: ~/python_course/myapp
dev@dev-VirtualBox:~/python_course/myapp$ ll
total 12
drwxrwxr-x 2 dev dev 4096 map 23 00:40 ./
drwxrwxr-x 3 dev dev 4096 map 23 00:36 ../
-rw-rw-r-- 1 dev dev  44 map 23 00:37 app.py
dev@dev-VirtualBox:~/python_course/myapp$ cat app.py
if __name__ == "__main__":
    print("Hi!")
dev@dev-VirtualBox:~/python_course/myapp$ which python3
/usr/bin/python3
dev@dev-VirtualBox:~/python_course/myapp$ python3 -m venv ./venv
dev@dev-VirtualBox:~/python_course/myapp$ ll
total 16
drwxrwxr-x 3 dev dev 4096 map 23 00:40 ./
drwxrwxr-x 3 dev dev 4096 map 23 00:36 ../
-rw-rw-r-- 1 dev dev  44 map 23 00:37 app.py
drwxrwxr-x 6 dev dev 4096 map 23 00:40 .venv/
dev@dev-VirtualBox:~/python_course/myapp$ source ./venv/bin/activate
(.venv) dev@dev-VirtualBox:~/python_course/myapp$ which python3
/home/dev/python_course/myapp/.venv/bin/python3
(.venv) dev@dev-VirtualBox:~/python_course/myapp$
```


Менеджер пакетов pip

- Установить пакет

A terminal window titled 'dev@dev-VirtualBox: ~/python_course/myapp' showing the execution of 'pip install requests'. The output displays the collection and download of 'requests-2.27.1' and its dependencies: 'urllib3-1.26.9', 'charset-normalizer-2.0.12', 'certifi-2021.10.8', and 'idna-3.3'. The packages are successfully installed.

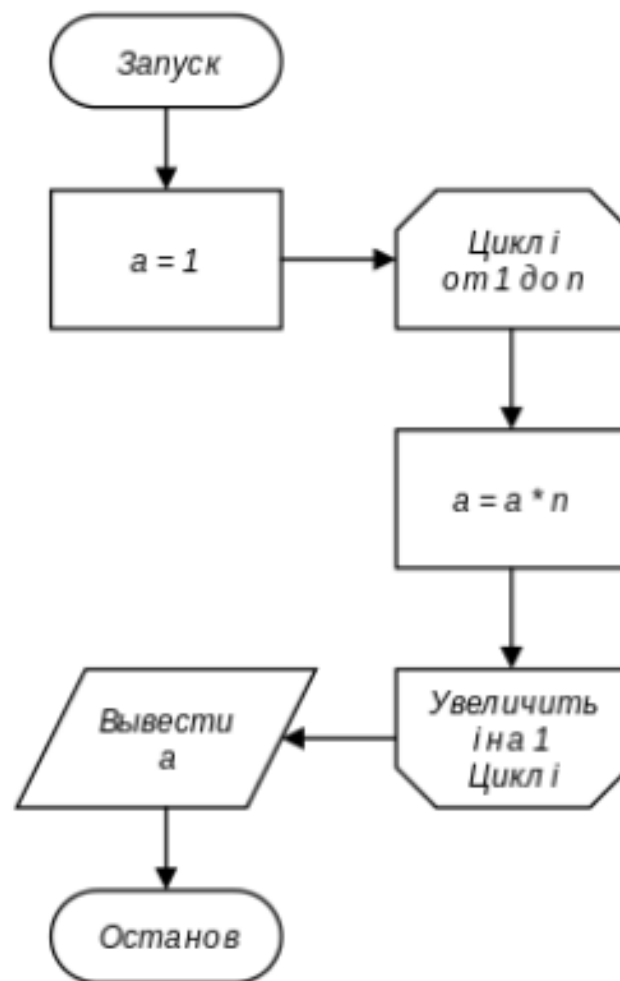
```
dev@dev-VirtualBox: ~/python_course/myapp
(.venv) dev@dev-VirtualBox:~/python_course/myapp$ pip install requests
Collecting requests
  Downloading requests-2.27.1-py2.py3-none-any.whl (63 kB)
    | 63 kB 456 kB/s
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.9-py2.py3-none-any.whl (138 kB)
    | 138 kB 1.2 MB/s
Collecting charset-normalizer~=2.0.0; python_version >= "3"
  Downloading charset_normalizer-2.0.12-py3-none-any.whl (39 kB)
Collecting certifi>=2017.4.17
  Downloading certifi-2021.10.8-py2.py3-none-any.whl (149 kB)
    | 149 kB 4.7 MB/s
Collecting idna<4,>=2.5; python_version >= "3"
  Downloading idna-3.3-py3-none-any.whl (61 kB)
    | 61 kB 4.4 MB/s
Installing collected packages: urllib3, charset-normalizer, certifi, idna, requests
Successfully installed certifi-2021.10.8 charset-normalizer-2.0.12 idna-3.3 requests-2.27.1 urllib3-1.26.9
(.venv) dev@dev-VirtualBox:~/python_course/myapp$
```

Алгоритмы

Алгоритм — набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата. Независимые инструкции могут выполняться в произвольном порядке, параллельно, если это позволяют используемые исполнители.

(Википедия)

Алгоритмы



Свойства

- Конечность описания
- Дискретность
- Направленность
- Массовость
- Детерминированность

Конечность описания

Конечность описания — любой алгоритм задается как набор инструкций конечных размеров, т. е. программа имеет конечную длину.

Дискретность

Дискретность — алгоритм выполняется по шагам, происходящим в дискретном времени. Шаги четко отделены друг от друга. В алгоритмах нельзя использовать аналоговые устройства и непрерывные методы.

Направленность

Направленность — у алгоритма есть входные и выходные данные. В алгоритме четко указывается, когда он останавливается, и что выдается на выходе после остановки.

Массовость

Массовость — алгоритм применим к некоторому достаточно большому классу однотипных задач, т. е. входные данные выбираются из некоторого, как правило, бесконечного множества.

Детерминированность

Детерминированность (или конечная недетерминированность) — вычисления продвигаются вперед детерминировано, т. е. вычислитель однозначно представляет, какие инструкции необходимо выполнить в текущий момент. Нельзя использовать случайные числа или методы. Конечная недетерминированность означает, что иногда в процессе работы алгоритма возникает несколько вариантов для дальнейшего хода вычислений, но таких вариантов лишь конечное.

Программные блоки

Куски программного кода, определяющие некоторую независимую часть логики. Обычно выделяются специализированными символами либо ключевыми словами

В python программные блоки выделяются 4мя пробелами или одной табуляцией - это позволяет визуально отделять блоки и дисциплинирует программиста писать более читаемый код.

```
def foo():  
    x = 1  
    x = x + 1  
    for i in range(10):  
        if i % 2 == 0:  
            print('I is even')  
        else:  
            print('I is odd')
```

```
def foo():  
    x = 1  
    x = x + 1|  
    for i in range(10):  
        if i % 2 == 0:  
            print('I is even')  
        else:  
            print('I is odd')
```

Операции сравнения

Операция сравнения может быть применена только к объектам, которые можно сравнивать между собой! Кроме того объекты должны поддерживать соответствующую операцию.

Вы не можете сравнивать, к примеру, число со строкой, **НО** вы можете проверять на равенство одного объекта с другим

Результатом сравнения всегда является значение типа **bool**, то есть **True** или **False**

Операции сравнения

| Operator | Description | Example |
|----------|---|---|
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

https://www.tutorialspoint.com/python/python_basic_operators.ht

Логические операторы

| Operator | Description | Example |
|-----------------------|--|------------------------|
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

Оператор is

Возвращает значение **True**, если переменные по обе стороны от оператора указывают на один и тот же объект, и **False** в противном случае. **ТО ЕСТЬ СРАВНЕНИЕ ИДЕТ НЕ ПО ЗНАЧЕНИЮ НА КОТОРЫЕ УКАЗЫВАЮТ ПЕРЕМЕННЫЕ, А ПО АДРЕСУ НА КОТОРЫЙ ССЫЛАЮТСЯ ПЕРЕМЕННЫЕ.**

```
In [24]: a = 1  
In [25]: b = 1  
In [26]: a is b  
Out[26]: True
```

Оператор in

Возвращает **True**, если находит переменную в указанной последовательности, и **False** в противном случае.

```
In [1]: l = [1,2,3]
```

```
In [2]: 1 in l  
Out[2]: True
```

```
In [3]: x = 1
```

```
In [4]: x in l  
Out[4]: True
```

```
In [5]: id(1)  
Out[5]: 9666944
```

```
In [6]: id(l[0])  
Out[6]: 9666944
```

```
In [7]: id(x)  
Out[7]: 9666944
```

Оператор and

expression_1 and expression_2 -> True если оба выражения истинны в других случаях False

| x | y | x and y |
|-------|-------|---------|
| True | True | True |
| False | False | False |
| False | True | False |
| True | False | False |

Оператор or

expression_1 or expression_2 -> True если **хотя бы одно** выражения истинно в других случаях False

| x | y | x and y |
|-------|-------|---------|
| True | True | True |
| False | False | False |
| False | True | True |
| True | False | True |

Оператор not

`not expression_1` -> True если выражение False и наоборот

| x | not x |
|-------|-------|
| True | False |
| False | True |

Tricks

Выражения можно комбинировать они могут быть очень сложными, **НО** будьте осторожны с этим слишком сложные выражения сложно отлаживать и поддерживать, так что лучше их разбивать на мелкие составные части.

```
In [9]: x = True
```

```
In [10]: y = False
```

```
In [11]: x is True and not y or y is False and x is False
```

```
Out[11]: True
```

Условные операторы if ... else ...

Синтаксис

```
if logic_expression:  
    do_some_work  
else: # если выражение не верно  
    do_other_work
```

```
19 i = 1  
20 if i > 10:  
21     print("i is greater than 10")  
22 else:  
23     print("i is less than 10")  
24  
25
```

Условные операторы (switch case - like)

Синтаксис

if logic_expression:

 do_some_work

elif other_logic_expression:

 do_other_work

else:

 do_smth_else

```
26 i = 20
27 if i > 30:
28     print("i is greater than 30")
29 elif i < 30 and i >= 20:
30     print("i is less than 30 but it greater or equal to 20")
31 else:
32     print("To small value for i")
```

Условные операторы (match case)

Синтаксис

match variable:

case logic_expression:

 do_some_work

case other_logic_expression:

 do_other_work

```
num = random.choice(range(5))

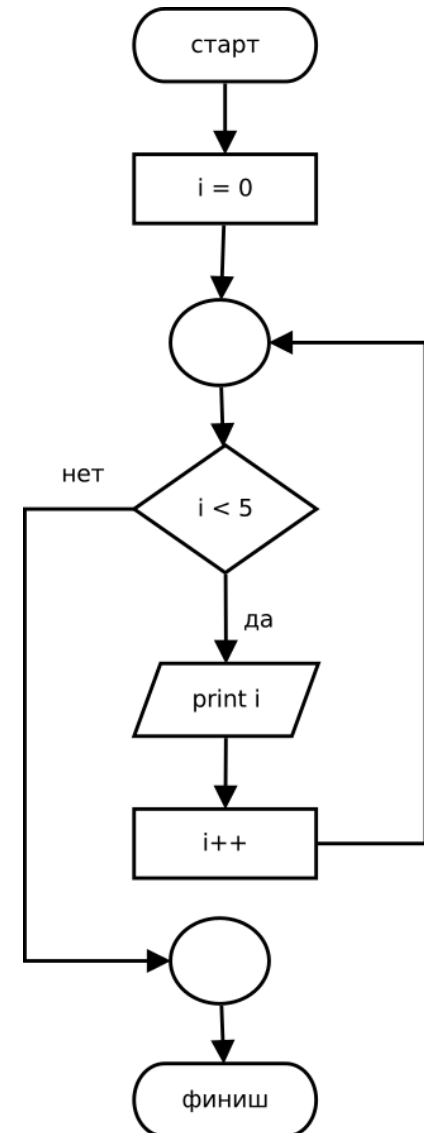
match num:
    case 0: # if num == 0
        print("Zero")

    case 1: # elif num == 1
        print("One")

    case _: # else
        print("Not zero or one")
```

Циклы

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.



while

Синтаксис

while <logic_expression>:
 do_smth

ATTENTION! Логика определяющая работу цикла определяется в новом программном блоке - еще называется **телом цикла**

```
1  i = 1
2  while i < 10:
3      print(i)
4      i += 1
5
```


for

Синтаксис

for <var> **in** <collection>:
 do_some_work

```
7  l = [1,2,3,4,5,6,7,8,9]
8  for i in l:
9      print(l)
10
11  for i in range(1, 10, 1):
12      print(l)
```

while/for ... else

Если дополнительно нужно обработать ситуацию запланированного завершения цикла, то можно использовать конструкцию.

```
In [18]: i = 1

In [19]: while i < 10:
...:     print(i)
...:     i += 1
...: else:
...:     print("End")
...:
```

```
1
2
3
4
5
6
7
8
9
End
```

```
In [20]: for i in range(5):
...:     print(i)
...: else:
...:     print('Finish')
...:
```

```
0
1
2
3
4
Finish
```

Ключевое слово **continue**

В ситуациях, когда вы предполагаете, что в каком-то месте тела цикла нужно прервать итерацию и начать следующую нужно использовать ключевое слово **continue**. Работает и для `for` и для `while`

```
In [21]: for i in range(10):  
...:     if i % 2 == 0:  
...:         continue  
...:     print(f"{i} is odd")  
...:  
1 is odd  
3 is odd  
5 is odd  
7 is odd  
9 is odd
```

Домашнее задание

1. Выведите последовательность чисел от 112..133 в обратном порядке различными типами цикла (`while` и `for`)
2. Выведите все простые числа для заданного интервала (`hw_2.py`)
3. На входе подается последовательность из десяти чисел, выведите максимально возможную сумму трех чисел из этой последовательности (`hw_3.py`)
- 4.