

DIRTY PYTHON 1.0

STONE 13TH PRESENTS

ЛЕКЦИЯ 1



DIRTY
PYTHON

ДАВАЙТЕ ЗНАКОМИТЬСЯ!



Автор, лектор и еще много кто этого курса
Стоун Семнадцатый,

он же Кирилл Панфилов

- финалист Comedy Баттл
- неоднократный победитель шоу НЕ СПАТЬ
- отец двоих детей
- панк и рокнрольщик :)

А также семинарист курсов «Введение в Python» и «Погружение в Python»

Так что у Стоуна можно не только ботам
поучиться – загляните на его Ютуб и ТГ
каналы

https://www.youtube.com/@dirty_python

https://t.me/dirty_python

<https://t.me/+knXQRpqb05o0ODUy>

https://t.me/STONE_Py



УСТАНОВКА РУТНОН И PYCHARM

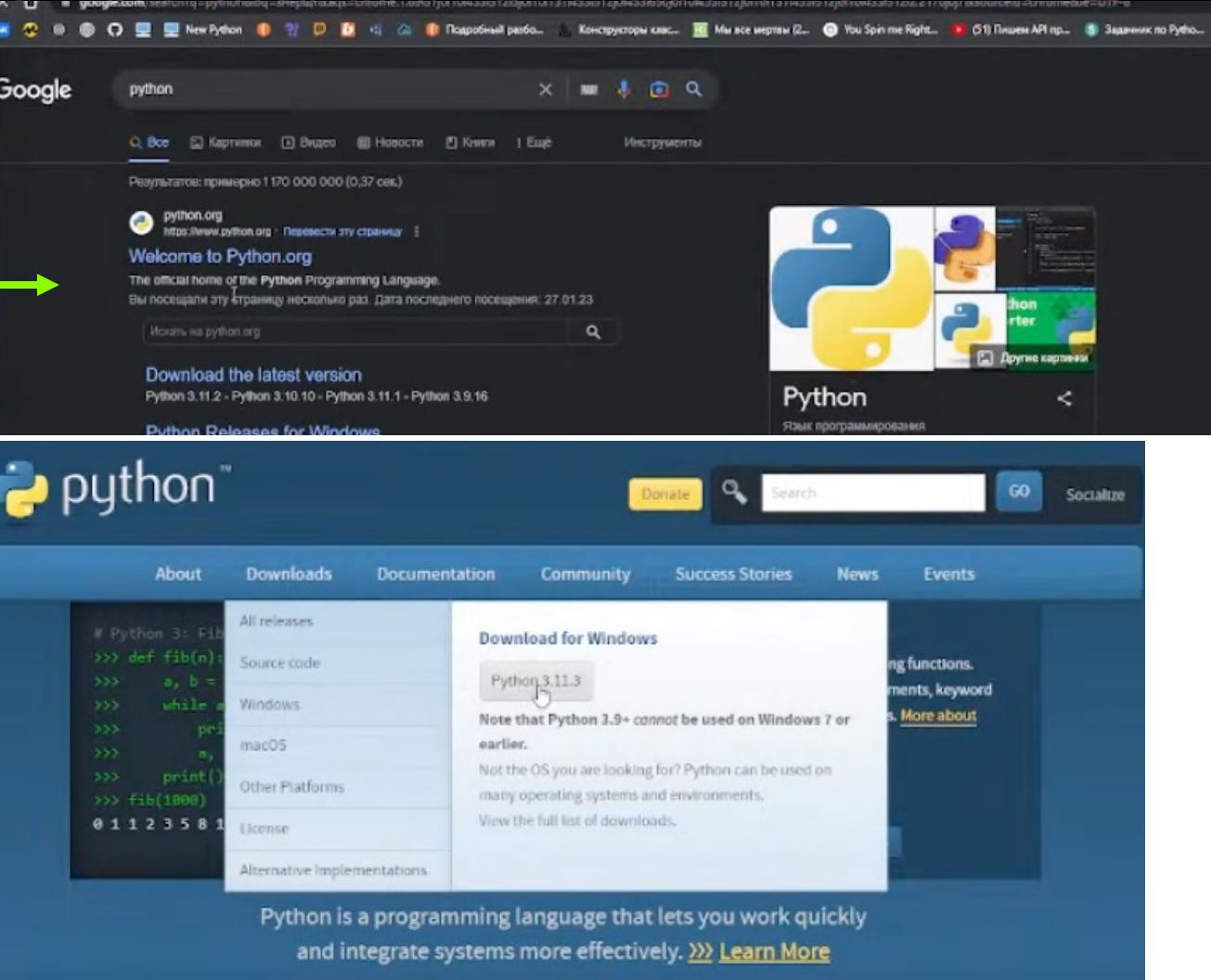
- Для того чтобы успешно освоить азы языка Python, нам, как это ни странно, потребуется сам Python
- Если у вас McBook или Linux, то он скорее всего у вас уже есть, вам придется его обновить до последней версии
- Если же у вас Windows то надо будет скачать, но при этом есть одна тонкость, о которой чуть позже
- Также нам понадобиться среда разработки. Мы будем использовать PyCharm, но если вам не нравится, вы можете использовать то, что вам удобно.



КАЧАЕМ PYTHON

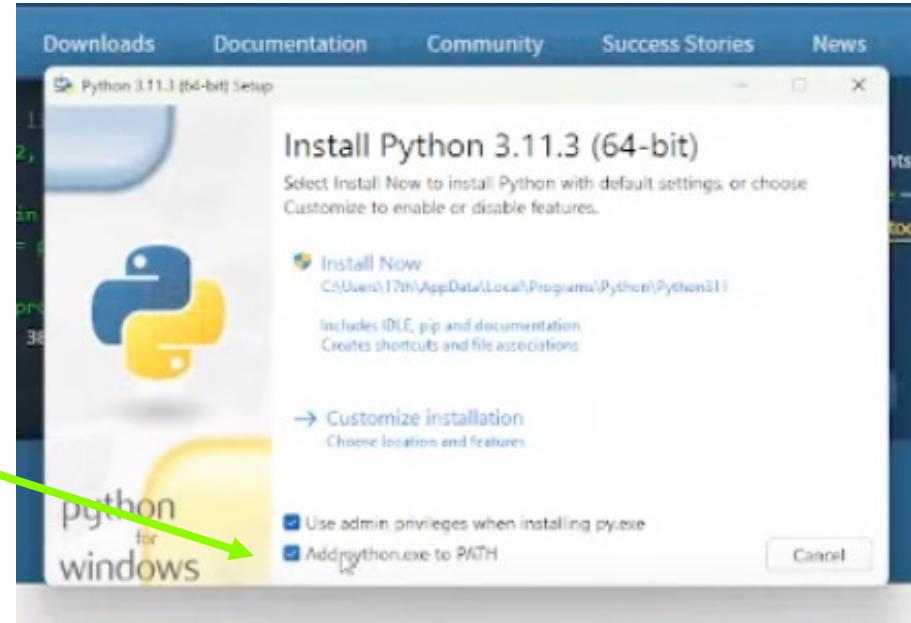


- Просто в поисковике вбиваем Python и переходим на python.org
- Заходим на вкладочку Download и скачиваем нужную версию
- Качается быстро)



СТАВИМ РУТИОН

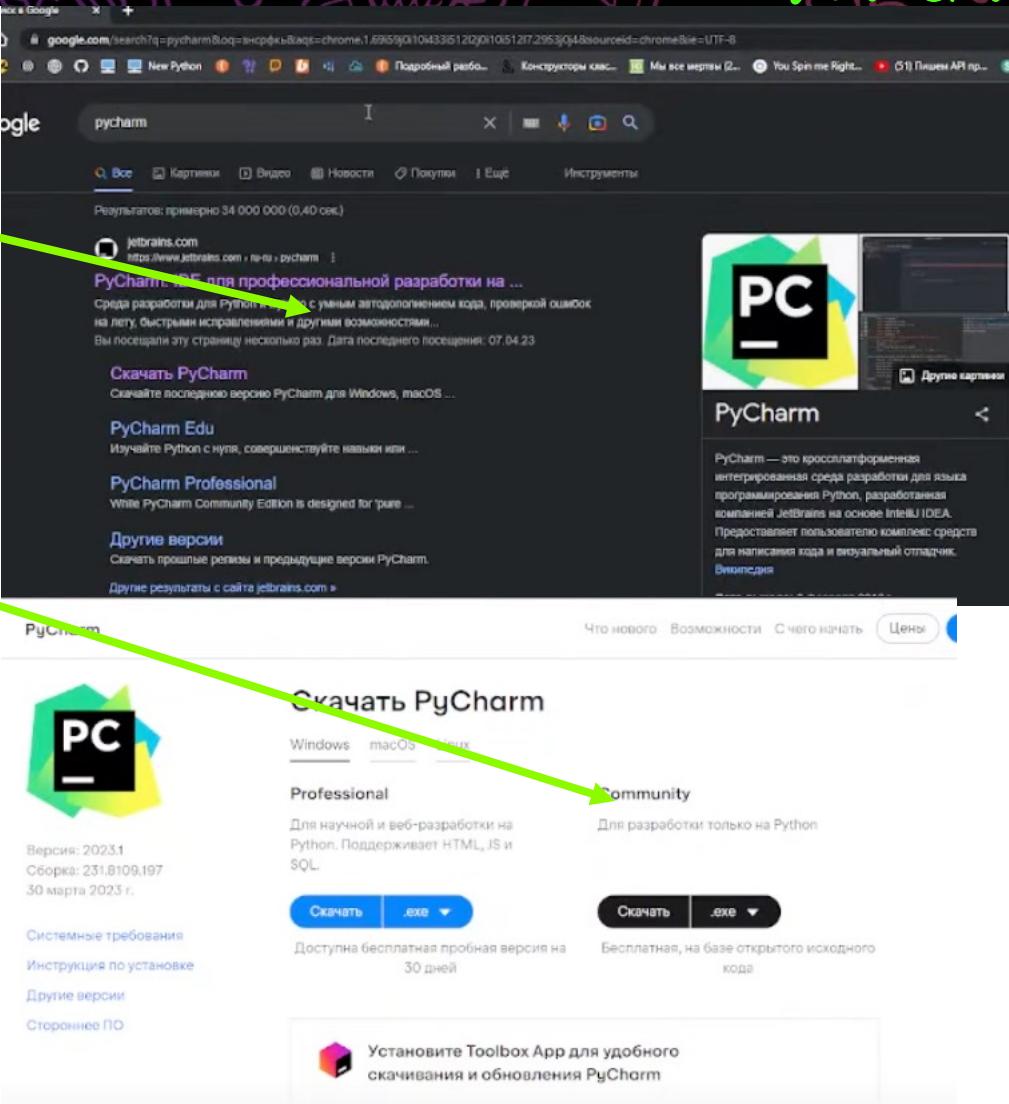
- Устанавливаем Python при помощи инсталлера
- Очень важный момент! **ОБЯЗАТЕЛЬНО СТАВИМ ВТОРУЮ ГАЛОЧКУ**
- Таким образом мы автоматически пропишем определение пути к Python и нам не придется делать это вручную
- В противном случае вы удивитесь, почему ваша среда разработки не видит, что Python есть
- Дальше устанавливаем нажав Install Now: дальше все стандартно





КАЧАЕМ PYCHARM

- Принцип тот же: в поисковике пишем pycharm, переходим по первой ссылке <https://www.jetbrains.com>, скачиваем версию Community - она бесплатная и нам хватит за глаза(тут придется подождать)
- И еще один момент! Не стремитесь устанавливать русификаторы. Если вы взялись за айтишечку, то английский вам придется освоить. И кстати, отличный способ погрузиться в среду, так что волей-неволей выучите



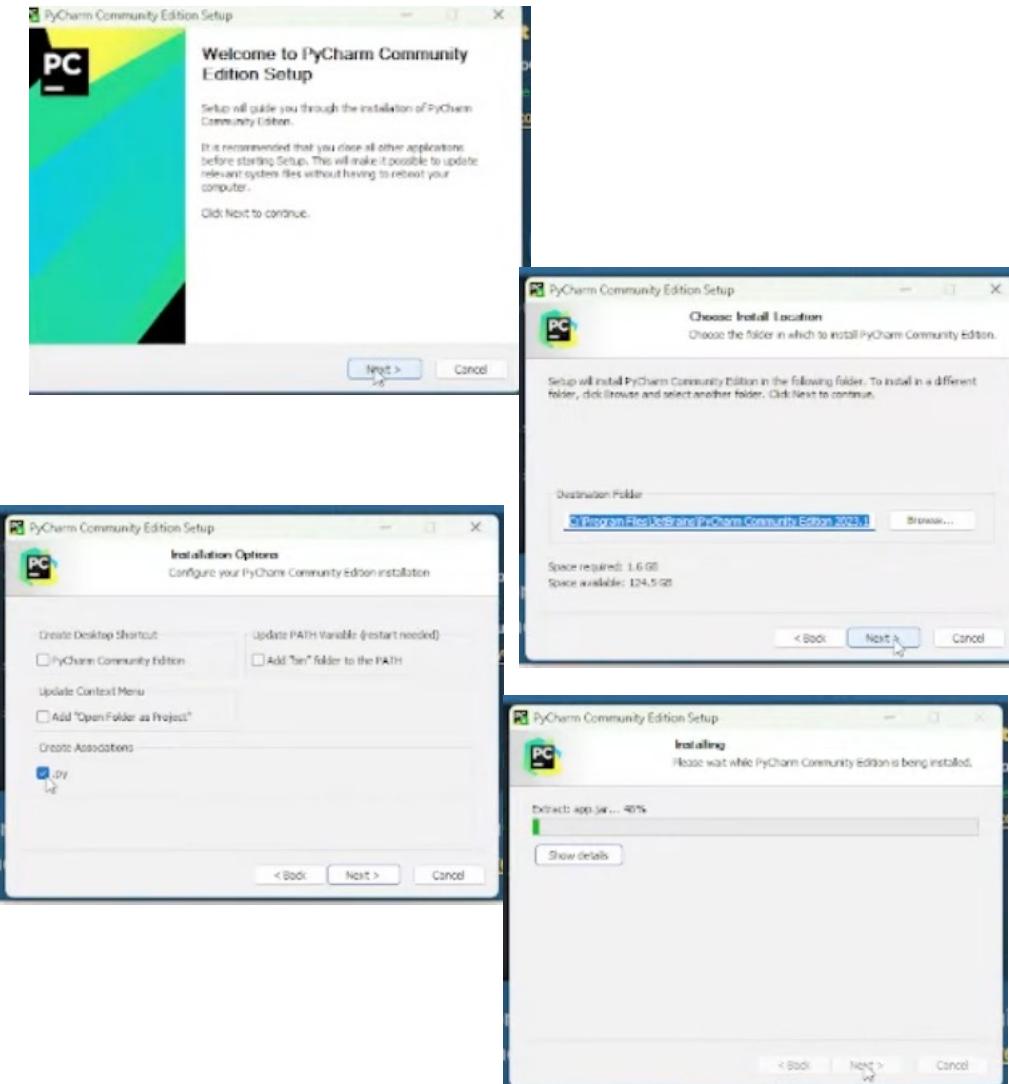
где живет английский язык?

тутъ

УСТАНАВЛИВАЕМ PYCHARM



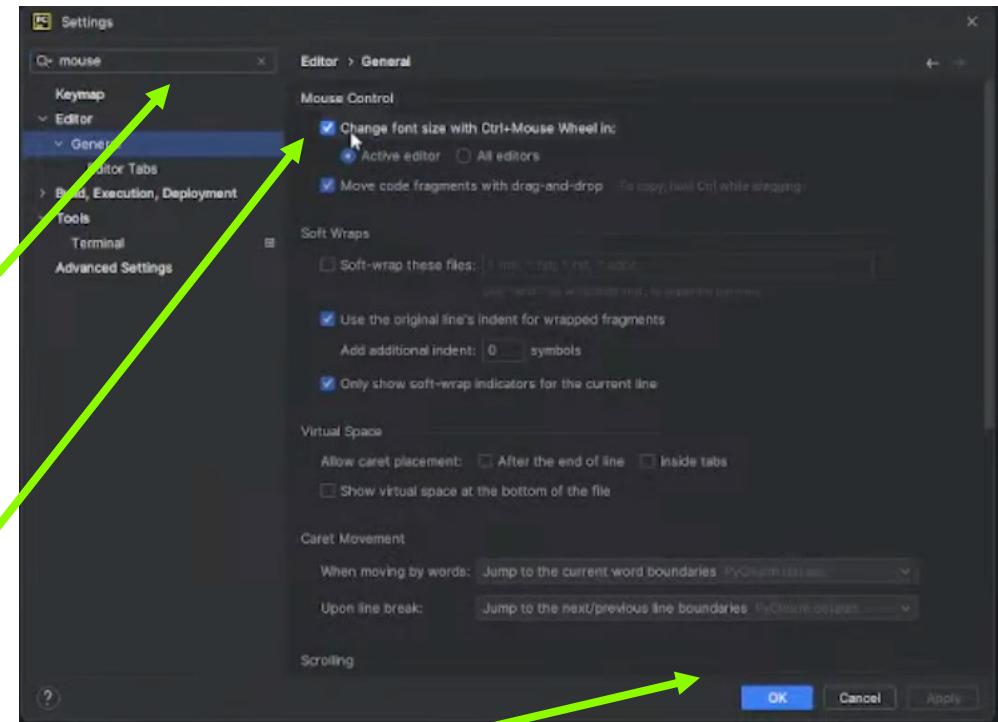
- После загрузки нажимаем на установщика – особых трудностей не возникнет
- Место установки можно поставить на свое усмотрение или оставить стандартный путь
- Можно поставить галочку, чтобы все файлы с расширением ru открывались Pycharm
- Ну и дальше просто ждем пока все установиться



НАСТРАИВАЕМ PYCHARM ДЛЯ РАБОТЫ



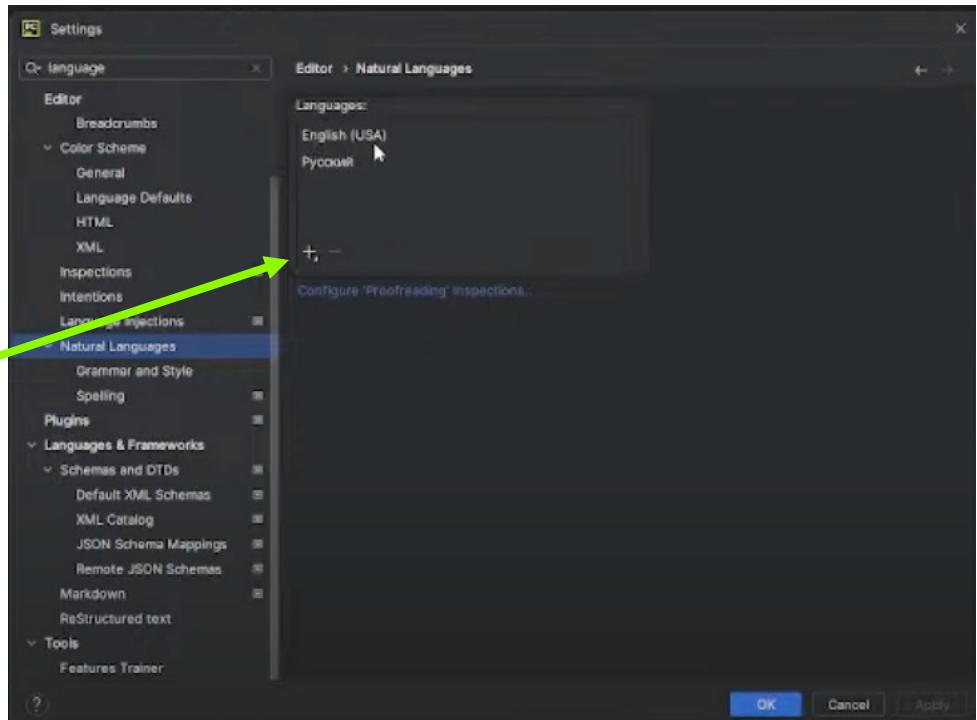
- В принципе на этапе знакомства и небольших проектиков виртуальное окружение не нужно.
- Но в Pycharm в отличие от VSC при создании проектов виртуальное окружение устанавливается сама. И чтобы это все работало нормально нам придется немного повозится в первый раз.
- Но начнем с настроек по проще)))
- Заходим в settings → в поисковике пишем mouse -> ставим галочку: это поможет нам масштабировать окно кода и терминала колесом мышки (что очень удобно при работе с другими людьми через зум) -> ставим ок



НАСТРАИВАЕМ PYCHARM ДЛЯ РАБОТЫ



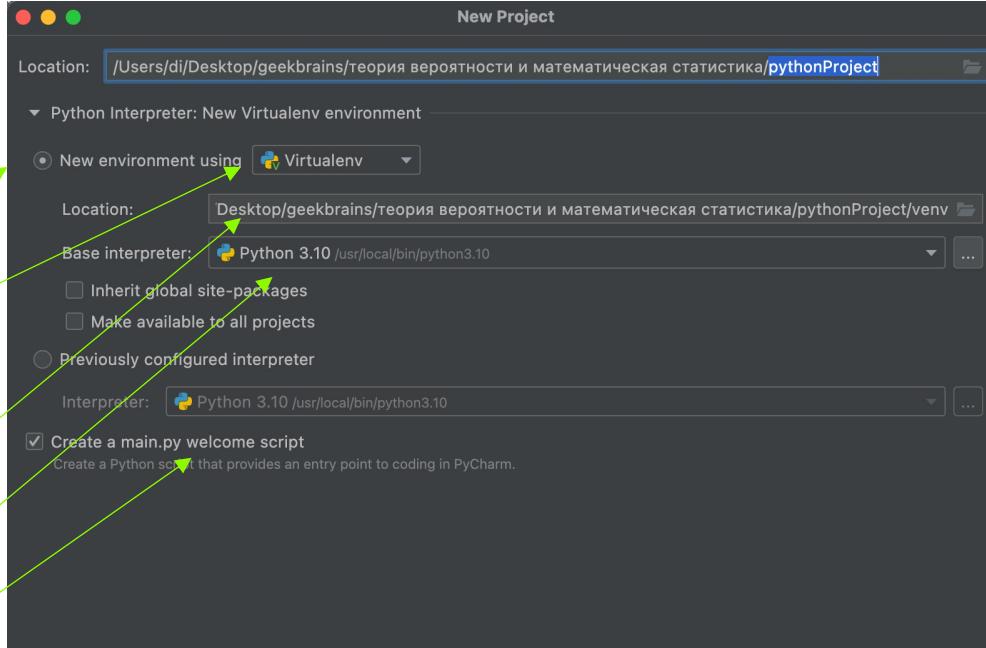
- Там же в настройках находим languages: мы ведь не хотим, чтобы наши комментарии на русском языке подчеркивало как ошибки
- Settings -> Languages -> Natural Languages -> + : выбираем русский язык -> ok
- В принципе этого уже достаточно для комфортной работы
- Про привязку к github сделаем отдельный ролик



НАСТРАИВАЕМ ВИРТУАЛЬНОЕ ОКРУЖЕНИЕ



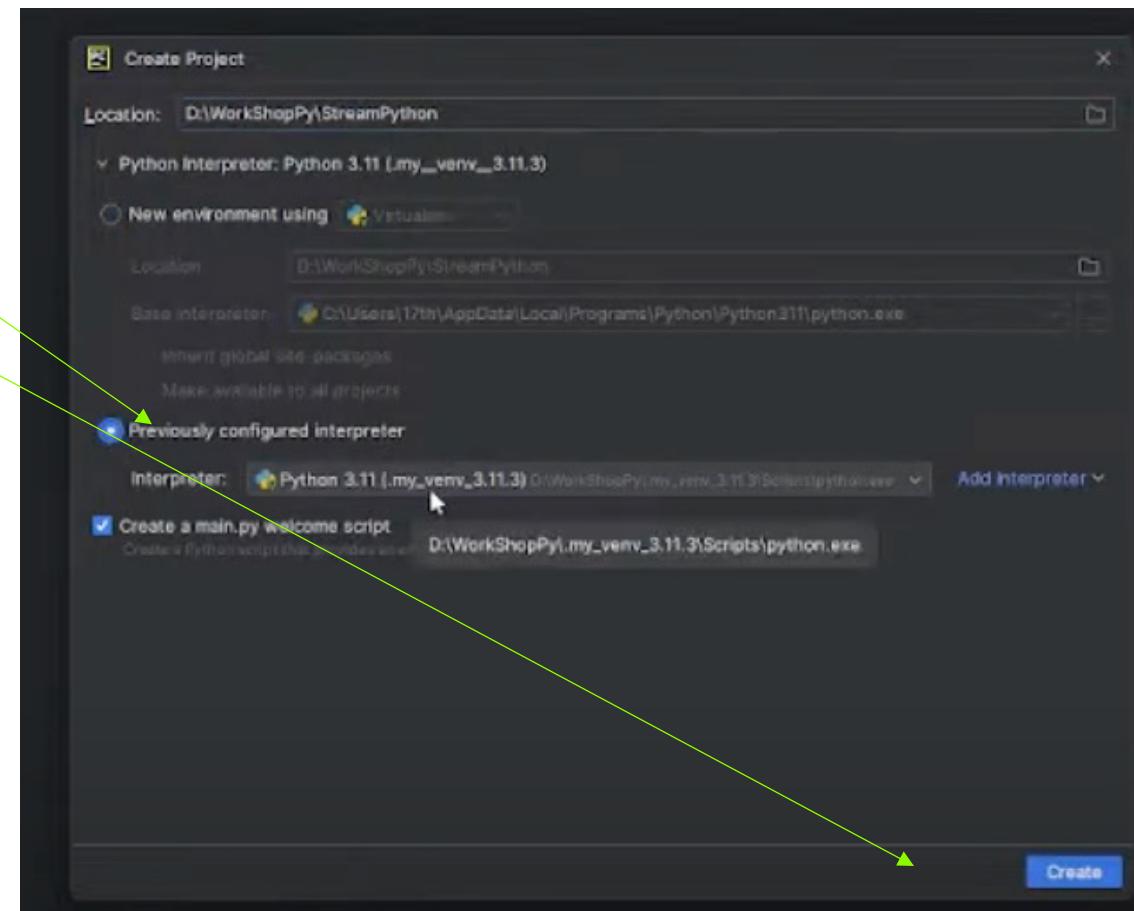
- У нас есть несколько вариантов:
- - создавать виртуальное окружение для каждого проекта отдельно
- - создать общее виртуальное окружение
- При создании нового проекта можно указать как именно вы хотите
- Итак, первый вариант:
- Ставим галочку New environment using
- Выбираем тип virtualenv
- Указываем место расположение(само)
- Базовый интерпретатор (по умолчанию или нужный)
- Если вы поставите галочку make available to all project, то оно будет доступно всем проектам.
- Создавать файл main.py



НАСТРАИВАЕМ ВИРТУАЛЬНОЕ ОКРУЖЕНИЕ



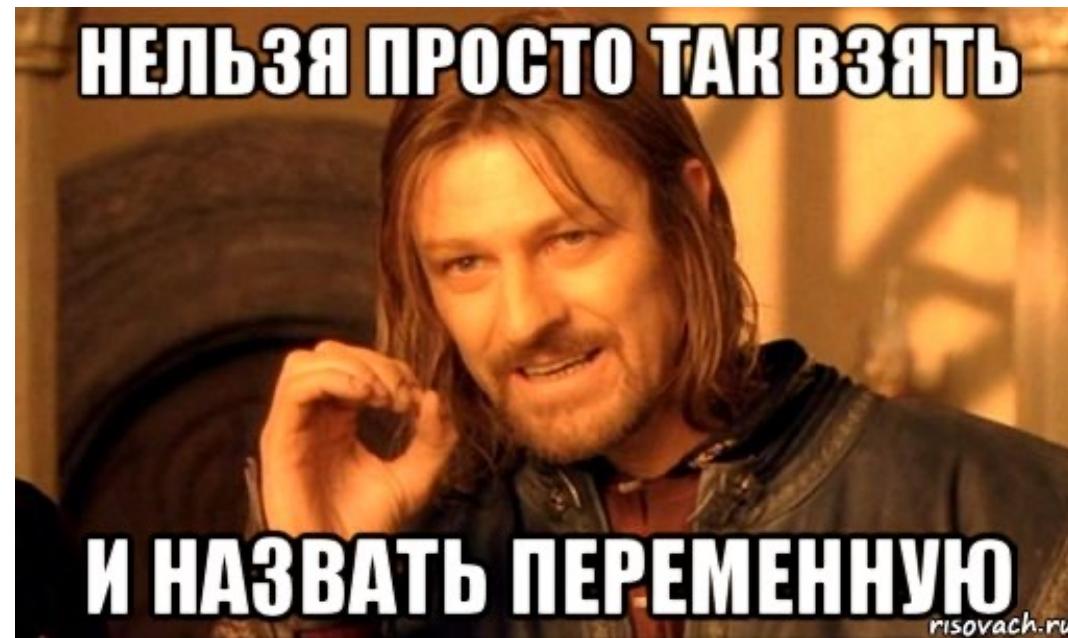
- Если вы поставили галочку, то теперь можете смело для нового проекта выбирать Previously configurated interpreter
- И все! Осталось нажать Create
- Так что такое виртуальное окружение?
- Это среда, в которой живет ваша программа, в которой вы устанавливаете все необходимое для вашего проекта, что обеспечивает для проекта:
 - - автономность
 - - версионность
 - - область видимости проекта (например спрятать токен от бота)



ПОЛЕЗНЫЕ ПЛАГИНЫ

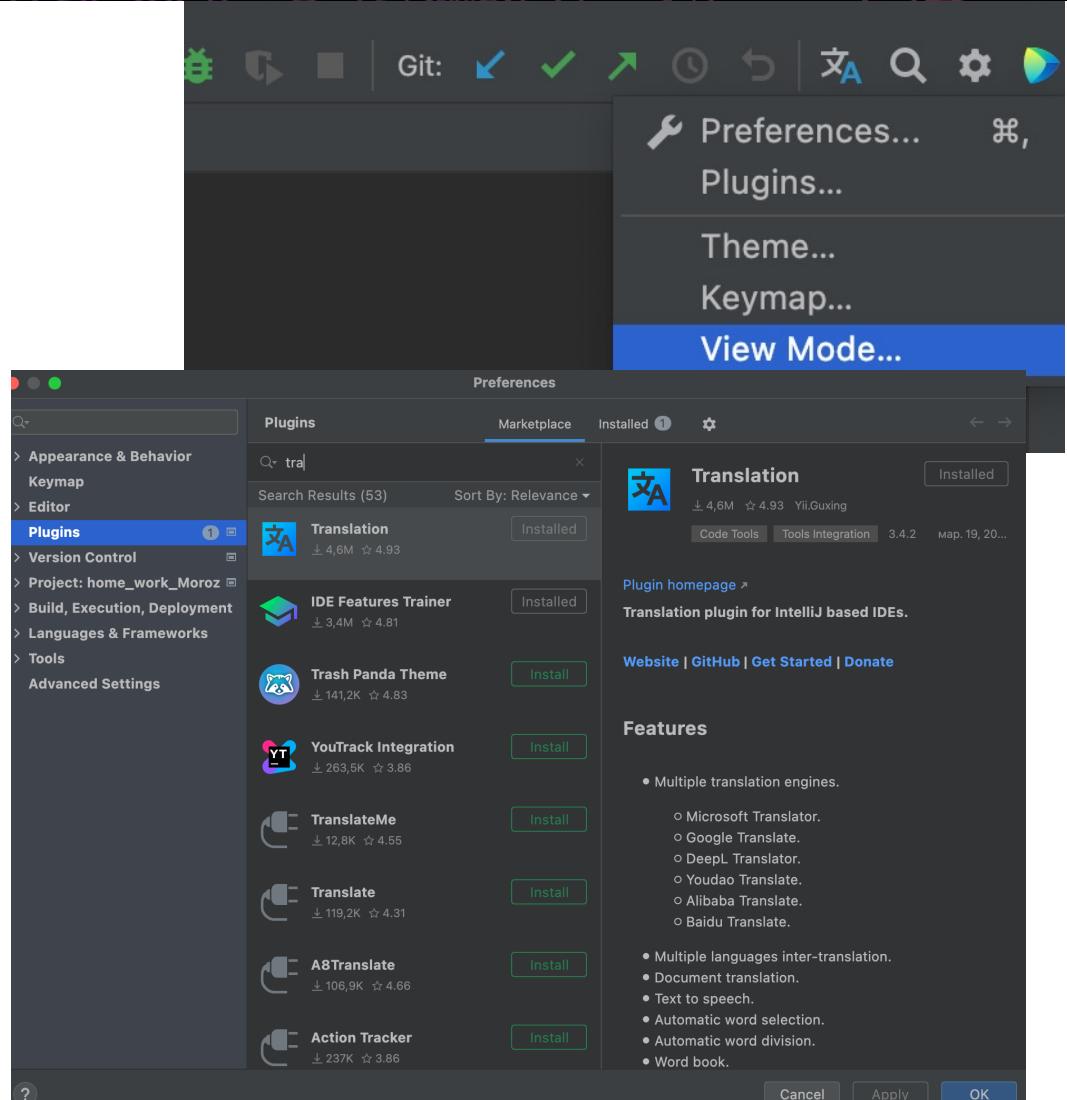


- Наименование переменных не такое простое дело, как может показаться. Не следует обзывать переменные просто буквами – вы не вспомните, что это значит (хотя для решения простых задачек это допустимо). И уж тем более не называйте переменные транслитерацией на латиницу русских слов. То есть переменная у нас не chislo, а number
- И если у вас проблемы с английским, чтоб всякий раз не лезть в переводчик можно установить плагин Translation



ПОЛЕЗНЫЕ ПЛАГИНЫ

- Идем в плагины. Это вот сюда: шестереночка в правом верхнем углу
- В поисковой строке плагина пишем translation и нажимаем инсталл. Возможно, попросит перезагрузить среду разработки
- И теперь немного донастроим: в верхнем правом углу у вас появится значок плагина. Нажимаем на него, выбираем настройки, устанавливаем галочку replace automatically if single result (см следующий слайд)





ПОЛЕЗНЫЕ ПЛАГИНЫ

The screenshot shows the PyCharm IDE interface with the 'Translation' plugin settings open. The main window displays code snippets in English, with a toolbar at the top. A yellow callout points to the 'Translate' icon in the toolbar, labeled 'Зачок плагина' (Plugin Snippet). Another yellow callout points to the 'Tools' section in the Settings sidebar, labeled 'Настройки плагина' (Plugin settings). A third yellow callout points to the 'Replace automatically if single result' checkbox in the 'Translate and Replace' section, labeled 'Галочка для автозамены' (Checkmark for automatic replacement).

home_work_Moroz - lesson8.py

task03

Git:

Зачок плагина

Translate

plt

Detect language ▾ English ▾

Настройки плагина

Tools > Translation

Restore Defaults

Text Selection

Keep content formatting

Take the nearest single word

Take word when translation dialog opens

Ignore regex: `[^/#$]` Check

The content matched by the expression will be removed before translation.

Translation Popup

Fold original text

Show word forms if available

Play Text to Speech automatically: Original

Translate and Replace

Add to context menu

Select target language before replacement

Replace automatically if single result

Separators: -

Word of the Day

Show words on startup

Show word explanation

Word Book

OK Cancel Apply

Settings

Notifications

Version Control

Project: StreamPython

Build, Execution, Deployment

Languages & Frameworks

Tools

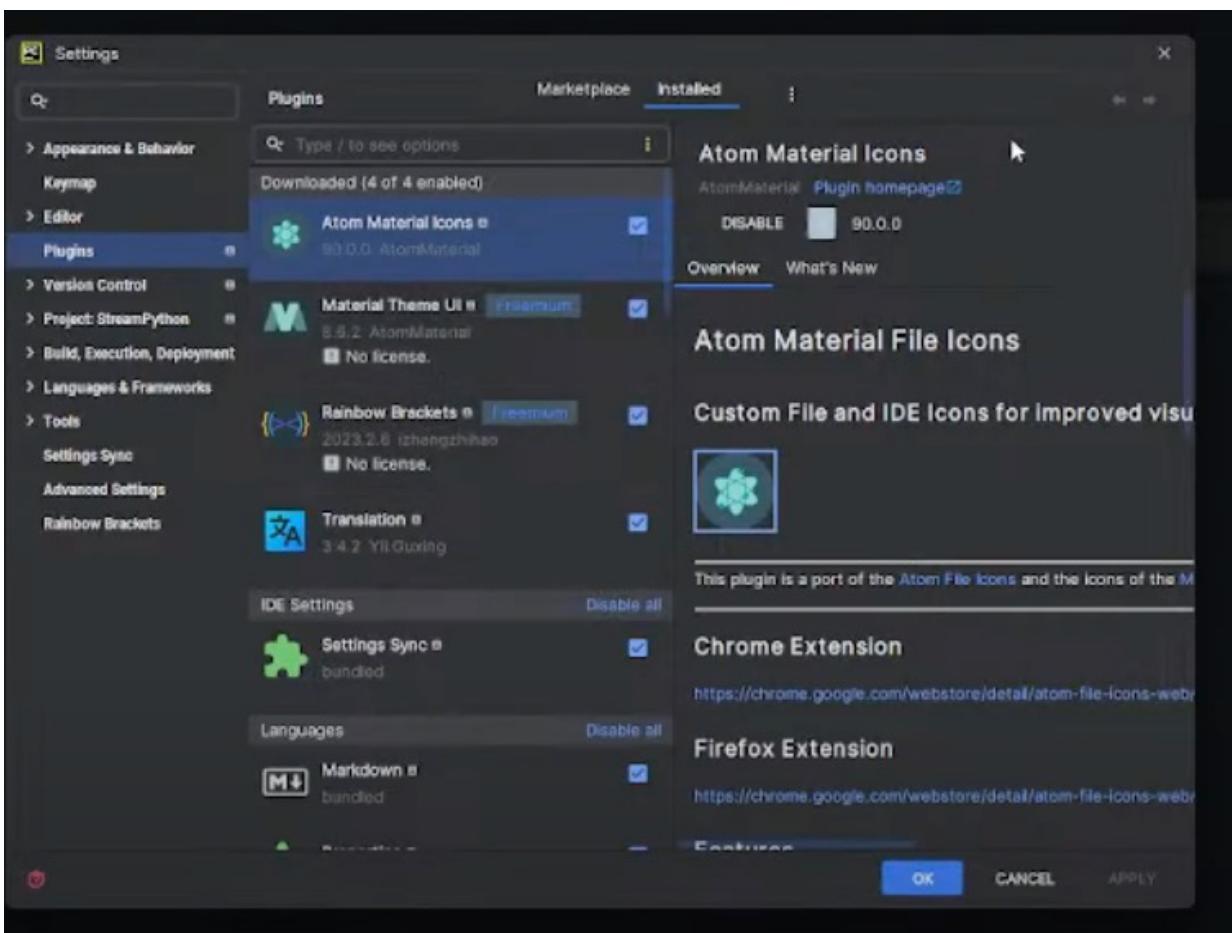
- Space
- Qodana
- Actions on Save
- Web Browsers and Preview
- External Tools
- Terminal
- Code With Me
- Diff & Merge
- External Documentation
- Features Suggester
- Features Trainer
- Python Integrated Tools
- Server Certificates
- Shared Indexes
- Startup Tasks
- Tasks
- Translation

Settings Sync

Advanced Settings

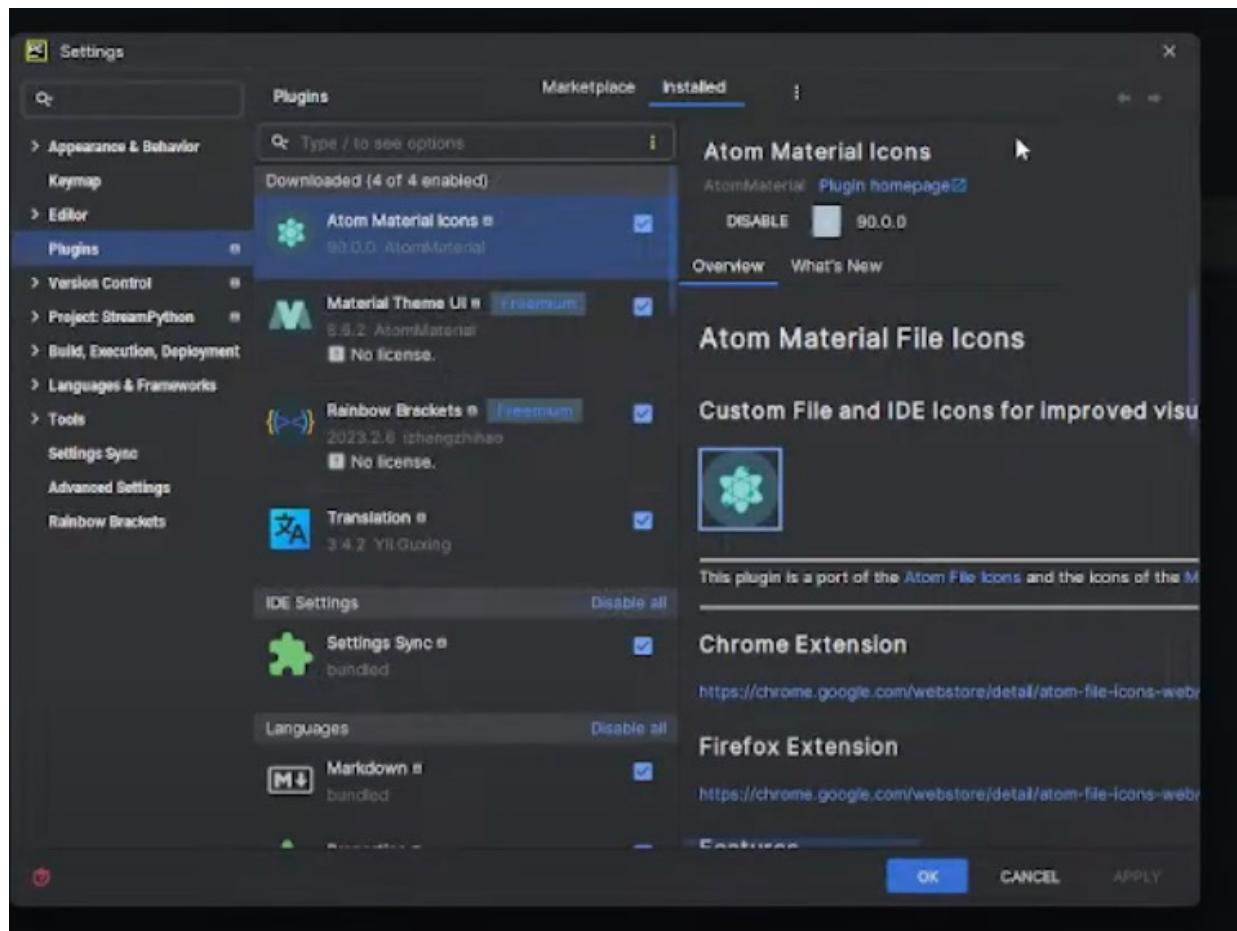
ПОЛЕЗНЫЕ ПЛАГИНЫ

- Еще один полезный плагин Rainbow brackets
- Позволяет подсвечивать скобочки, чтобы не перепутать какую вы открыли, какую закрыли
- Пишет, что платный, но вроде неплохо работает и так
- Принцип установки такой же



ПОЛЕЗНЫЕ ПЛАГИНЫ

- Еще один полезный плагин Rainbow brackets
- Позволяет подсвечивать скобочки, чтобы не перепутать какую вы открыли, какую закрыли
- Пишет, что платный, но вроде неплохо работает и так
- Принцип установки такой же
- Кроме того, вы можете настроить горячие клавиши: например **ctrl+D** – дублирование строк



СОЗДАЕМ ПРОЕКТЫ, ДОБАВЛЯЕМ ФАЙЛЫ



- При открытии PyCharm сам предложит вам создать проект - собственно так и делаем. Сразу в нем будет только папка с виртуальным окружением и файл main.py
- Чтобы добавить директорию (папку) пакет(специальная папка с файлом `__init__.py` – об этом позже), любой файл или файл ru – нажимаем мышью на саму папку проекта и выбираем, что хотим создать

```
Lecture > main.py
Project  Lecction ~/Desktop/DirtyPython/Lecture
          venv
          main.py
External Libraries
Scratches and Consoles

# This is a sample Python script.

# Press ^R to execute it or replace it with your code.

# Press Double e to search everywhere for classes, files, tool windows, actions, and settings.

def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.

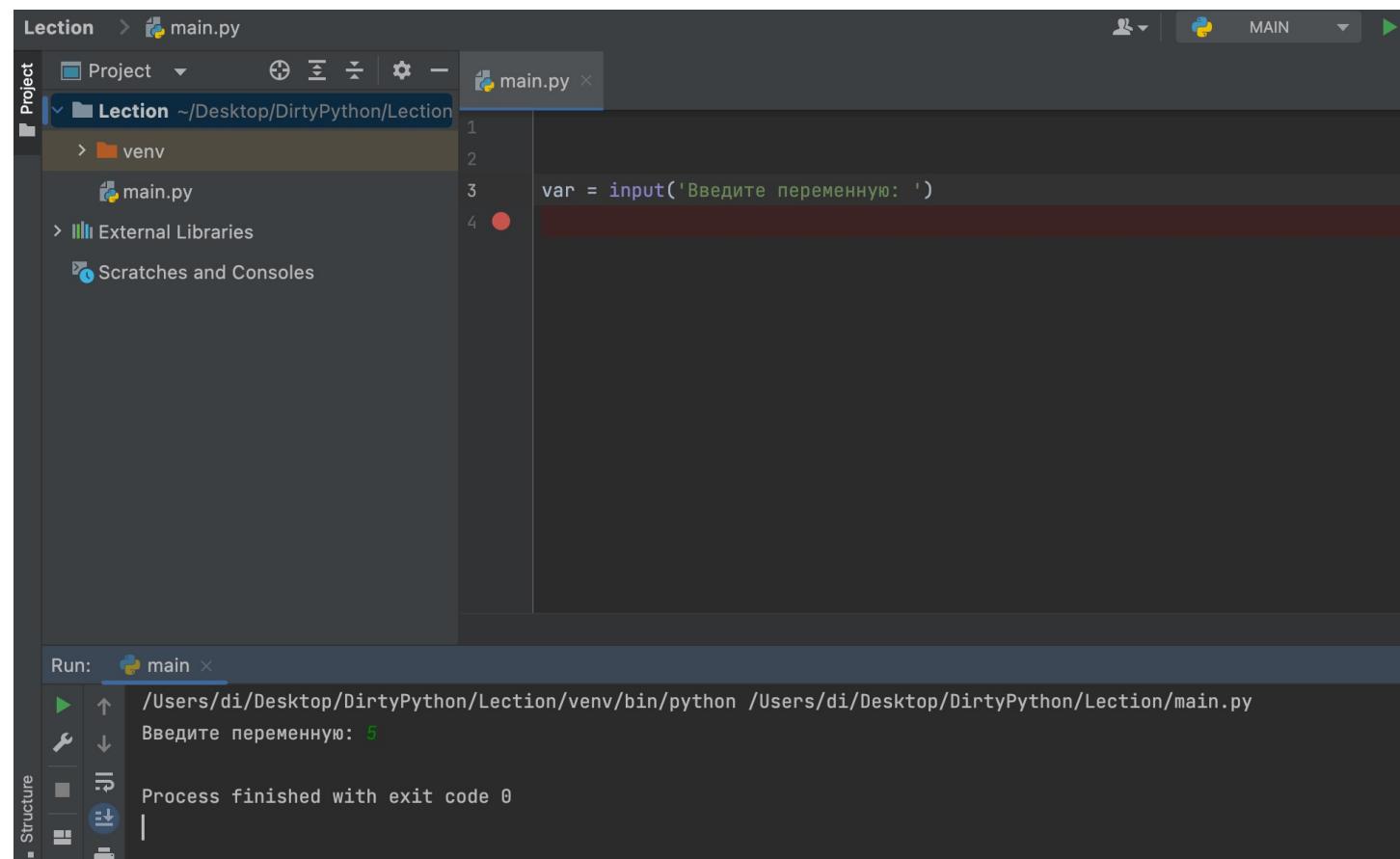
    print(f'Hi, {name}')  # Press wFR to toggle the breakpoint.

TrAnsL'etsA {dioT uJE
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project  StreamPython Version control
New
File
Scratch File
Directory
Python Package
Python File
HTML File
Resource Bundle
EditorConfig File
File >
Cut Ctr+X
Copy Ctr+C
Paste Ctr+V
Find Usages Alt+F7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Inspected Code...
Refactor
Clean Python Compiled Files
Bookmarks
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Open In
Local History
Repair IDE on File
Reload from Disk
Compare With...
Mark Directory as
Run
3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
5
5->5 5 следующий элемент 5
Process finished with exit code 0

1.1. CRLF UTF-8 StreamPython Dark Google 4 spaces Python 3.11 (my_venv_3.11.3)
```

INPUT()

- Для ввода данных используется функция `input()`
- При этом внутри скобок мы можем писать текст, который будет выводиться в терминал, что избавляет нас от лишних принтов.
- То, что будет введено в `input()` можно присвоить переменной, а затем уже делать с ней то, что нам нужно)



```
Project: Lection | File: main.py | Editor: Python | Run: main | Structure
```

Lection > main.py

Project: Lection ~/Desktop/DirtyPython/Lection

var = input('Введите переменную: ')

Run: main > /Users/di/Desktop/DirtyPython/Lection/venv/bin/python /Users/di/Desktop/DirtyPython/Lection/main.py

Введите переменную: 5

Process finished with exit code 0

PRINT()

- Для того, чтобы что-то вывести в терминал используется `print()`
- В скобках можно указать число, текст в кавычках, переменную и/или например функцию
- Если вы хотите, чтобы ваш `print()` заканчивался не переходом на новую строку, то вы можете указать через аргумент `end = '*' – например, в этом случае вместо перехода на новую строку будет напечатана звездочка`

Кстати, обратный слэш \ позволяет воспринимать знаки n и t как спецсимволы

```
4
5 print(var)      тут будет переход на новую строку
6 print(var, end="\n")  это то же самое)
7 print(var, end="->") между 3 и 4 будет стрелочка
8 print(var, end="      ") между 4 и 5 – куча пробелов
9 print(var, end=" следующий элемент ")
0 print(var, end="\t") табуляция
1

main.x
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
Введите переменную: 5
5
5
5->5      5 следующий элемент 5
Process finished with exit code 0
```

НЕИЗМЕНЯМЫЕ ТИПЫ ДАННЫХ



- В Pythonе есть следующие неизменяемые типы данных (их больше, но мы начнем с этих):
 - int – целые числа
 - float - числа с плавающей точкой
 - str – строка
 - bool – булевые (или Ложь/ Истина)
- Как проверить тип данных у переменных?
Очень просто – используя конструкцию
`print(type())`

И сейчас мы их по очереди разберем эти типы.

```
main.py
1
2
3
4
5
6
7
8
9
10
11

main.py:1: warning: "main" defined here but never used
var1 = 1
var2 = '1'
var3 = 1.1
var4 = True

print(type(var1))
print(type(var2))
print(type(var3))
print(type(var4))

Process finished with exit code 0
```



INT I.. 2.. 3456789999999999..

- Все целые числа относятся к типу int
- Python не делит их на большие и малые, а любит всех одинаково))) (посмотреть размер можно при помощи библиотеки sys sys.getsizeof(number))
- С цифрами мы можем делать все привычные нам математические операции и операции сравнения (проверяем на истинность)
- Объекты типа int мы можем использовать в любом месте, где могут быть целые числа – срезы, range и т.д. Что очень удобно, если у вас есть переменная n, значение которой может меняться

```
venv> python main.py
main.py:15
number_1 = 10
number_2 = 5
print(number_1 + number_2)      # сложение
print(number_1 - number_2)      # вычитание
print(number_1 / number_2)       # деление всегда дает float
print(number_1 * number_2)       # умножение
print(number_1 // number_2)      # целочисленное деление дает int
print(number_1 % number_2)       # остаток от деления
print(number_1 ** number_2)      # возведение в степень
print(number_1 == number_2)       # сравнение на равенство
print(number_1 != number_2)       # сравнение на неравенство.

main>
15
5
15
5
2.0
50
2
0
1000000
False
True
```

НЕМНОГО ПРО НЕИЗМЕНЯЕМОСТЬ



- Если мы задали переменную $a = 5$ – то это не столько коробочка, сколько стикер, потому как a хранит только ссылку на объект, в данном случае на пятерку
- Когда мы пишем $b = a$, мы только обозначаем, что и b теперь ссылается на этот объект в памяти
- Если мы после этого напишем $a = a + 1$ то мы не поменяем саму пятерку, а создадим новую связь. При этом в переменной b все еще хранится связь с пятеркой.
- Старый объект хранится в памяти, пока на него есть ссылки.
- **Меняя переменную мы не меняем сам объект!**

```
Lect... 1
> v... 2
m... 3
Exter... 4
Scr... 5
1
2
3
4
5
6
7
8
9
a = 5
b = 5
print(a)
print(b)
a = a + 1
print(a)
print(b)
```

main ×
/Users/di/Desktop/DirtyPython/Lection/venv/bin/python /U...
5
5
6
5

НЕМНОГО ПРО НЕИЗМЕНЯЕМОСТЬ



- Если мы задали переменную $a = 5$ – то это не столько коробочка, сколько стикер, потому как a хранит только ссылку на объект, в данном случае на пятерку
- Когда мы пишем $b = a$, мы только обозначаем, что и b теперь ссылается на этот объект в памяти
- Если мы после этого напишем $a = a + 1$ то мы не поменяем саму пятерку, а создадим новую связь. При этом в переменной b все еще хранится связь с пятеркой.
- Старый объект хранится в памяти, пока на него есть ссылки.
- **Меняя переменную мы не меняем сам объект!**

```
Lect... 1
> v... 2
m... 3
Exter... 4
Scr... 5
1
2
3
4
5
6
7
8
9
a = 5
b = 5
print(a)
print(b)
a = a + 1
print(a)
print(b)
```

main ×
/Users/di/Desktop/DirtyPython/Lection/venv/bin/python /U...
5
5
6
5



FLOAT

- Цифры с плавающей точкой – очень коварны!
- Когда компьютер пытается перевести в двоичную систему целые числа – проблем нет, но как только мы даем ему число с точкой – тут уже начинаются сложности, и такое число приобретает монстрообразный вид.
- Из-за этого могут возникнуть довольно странные вещи...
- История знает случаи, когда из-за этой плавающей точки взрывались ракеты и подрывались живые люди.
- Подробнее можно узнать здесь <https://www.youtube.com/watch?v=U0U8Ddx4TgE>

The screenshot shows a Python code editor with a dark theme. On the left, there's a sidebar with project navigation (Lect, venv, main, External, Scratch). The main area has a code editor with the following content:

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

```
print(0.1 + 0.1 + 0.1 - 0.3)
a = 0.1 + 0.1 + 0.1 - 0.3
b = 0
print(a==b)
c = 0.56
print(c)
print(c*100)
```

An orange arrow points from the text "из-за этой плавающей точки" in the slide content to the line of code `a = 0.1 + 0.1 + 0.1 - 0.3`. Below the code editor, the terminal window shows the output:

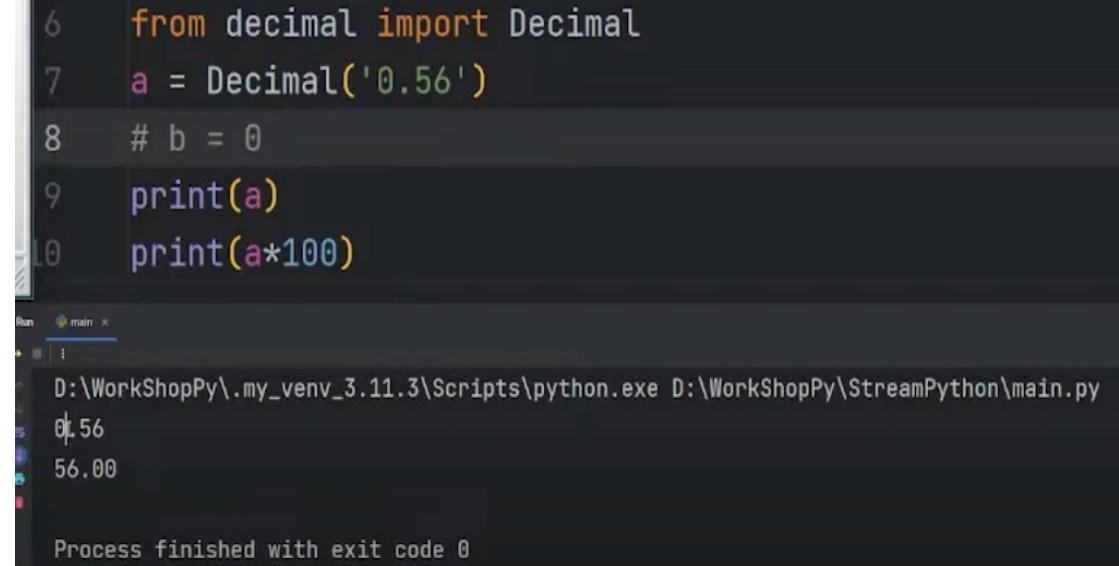
```
/Users/di/Desktop/DirtyPython/Lection/venv/bin/python main.py
5.551115123125783e-17
False
0.56
56.00000000000001
```

FLOAT

И что же делать?

К счастью есть специальная встроенная библиотека decimal, которая позволяет избежать всех этих неприятностей: все прекрасно работает.

И так мы с float можем проводить все обычные операции



```
from decimal import Decimal
a = Decimal('0.56')
# b = 0
print(a)
print(a*100)
```

D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py

0.56

56.00

Process finished with exit code 0

STRING

- Строку объявляем заключая какой-то текст в кавычки
- Кавычки могут быть двойными или одинарными, главное, открывать и закрывать одним типом.
- Это нужно, для использования кавычек внутри текста
- `text = 'Книга "Война и мир" Л.Н. Толстого'`
- `text = "Книга 'Война и мир' Л.Н. Толстого"`
- Кстати, три одинарные кавычки можно использовать для многострочных комментариев или переменных.

CONVERTING STRING TO INTEGER IN PYTHON



```
6  text = "2re'd'sgsd"
7
8  letters = '''wertgwrtwretg
9    egerwtewrtert
10   wertrwetertert
11   rwetertert'''
```

Run main

D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py

wertgwrtwretg
egerwtewrtert
wertrwetertert
rwetertert

STRING. ВСТРОЕННЫЕ МЕТОДЫ



- Встроенных методов у строк очень и очень много, ведь это основной способ хранения данных.
- Все их знать не обязательно, так их прям очень много: со временем из-за частого использования вы сами запомните нужное.
- Но возможности надо представлять: заменить букву, посчитать сколько раз она встречается, сделать все большими или маленькими, изменить регистр, написать с большой, разделить, проверить на то является цифрой, буквой и т.д.

```
5
6     text = "2re'd'sgsd"
7
8     text.
9
10    print(text)
11    print(text.isalnum())
12    print(text.count('e'))
```

STRING. ВСТРОЕННЫЕ МЕТОДЫ



- Встроенных методов у строк очень и очень много, ведь это основной способ хранения данных.
- Все их знать не обязательно, так их прям очень много: со временем из-за частого использования вы сами запомните нужное.
- Но возможности надо представлять: заменить букву, посчитать сколько раз она встречается, сделать все большими или маленькими, изменить регистр, написать с большой, разделить, проверить на то является цифрой, буквой и т.д.
- Все эти методы не изменяют саму переменную! Ее надо переназначить, если мы хотим сохранить изменения

```
5
6     text = "2re'd'sgsd"
7
8     text.
9
10    print(text)
11
12
Run main.py
D:\WorkShopPy\StreamPython\main.py
```

split(self, sep, maxsplit=..., str)
join(self, _iterable, str)
isdigit(self) str
count(self, x, _start=..., str)
__eq__(self, x) str
replace(self, __old, __new=..., str)
capitalize(self) str
casefold(self) str
center(self, __width=..., str)
encode(self, encoding='utf-8', errors='...', str)

SPLIT()



- `split()` разделяет строку на список (про списки позже)
- По умолчанию делит по пробелу, даже если их несколько. Но если вам нужны эти пробелы, то нужно указать вот так `text.split('')`
- В скобках можно указать по какому символу делить, например `text.split('r')`

```
# DOOL
*
6 text = "fg2sj    kkje235rw  lkf55o6wiyf uwk78jcb wkf4wf"
7 print(text)
8 print(text.split(' '))
# print(text.replace())
# print(text.isdigit())
# print(text.lower())

```

Run main x

```
D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
fg2sj    kkje235rw  lkf55o6wiyf uwk78jcb wkf4wf
['fg2sj', ' ', ' ', ' ', 'kkje235rw', ' ', ' ', 'lkf55o6wiyf', ' ', ' ', 'uwk78jcb', ' ', ' ', 'w
Process finished with exit code 0
```

```
# DOOL
*
6 text = "fg2sj kkje235rw lkf55o6wiyf uwk78jcb wkf4wf"
7 print(text)
8 print(text.split('r'))
# print(text.replace())
# print(text.isdigit())
# print(text.lower())

```

Run main x

```
D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
fg2sj kkje235rw lkf55o6wiyf uwk78jcb wkf4wf
['fg2sj kkje235', 'w lkf55o6wiyf uwk78jcb wkf4wf']

Process finished with exit code 0
```

REPLACE()



- При помощи replace мы можем заменить имеющийся символ или подстроку на другой символ или подстроку
- В скобках указываем сначала что будем заменять, а потом – на что будем заменять
- `text.replace('5', 'five')`

```
# DOOL
text = "fg2sj kkje235rw lkf55o6wiyf uwk78jcb wkf4wf"
print(text)
# print(text.split())
print(text.replace('_', '***'))
# print(text.isdigit())
# print(text.lower())

```

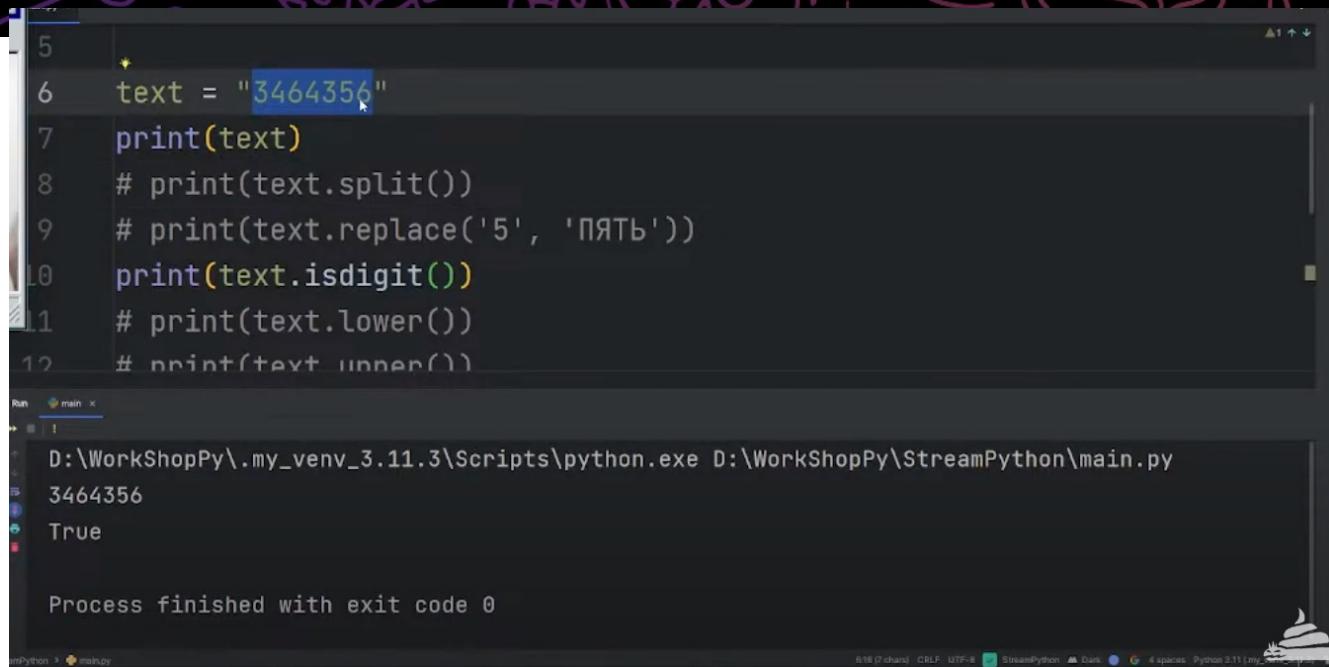
Run main.py

```
D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
fg2sj kkje235rw lkf55o6wiyf uwk78jcb wkf4wf
fg2sj***kkje235rw***lkf55o6wiyf***uwk78jcb***wkf4wf

Process finished with exit code 0
```

ISDIGIT()

- `isdigit()` проверяет состоит ли строка только из цифр и возвращает `True` или `False`
- `isalpha()` проверяет на буквенные символы
- `isnumeric()` возвращает `true`, если все символы в строке являются числовыми символами, и есть хотя бы один символ, иначе `false`. Числовые символы включают символы цифр и все символы, которые имеют свойство числового значения Unicode, например. U + 2155, VULGAR FRACTION ONE FIFTH. Формально числовыми являются символы со значением свойства `Numeric_Type = Digit`, `Numeric_Type = Decimal` или `Numeric_Type = Numeric`.



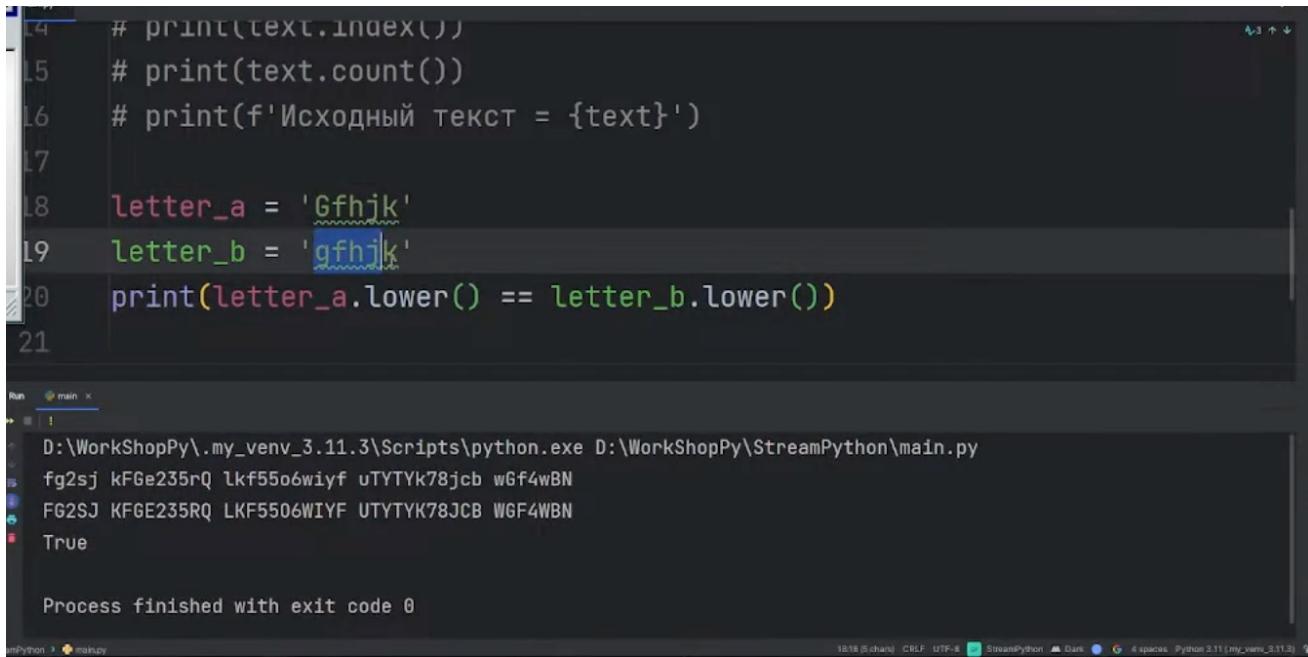
```
5
6 text = "3464356"
7 print(text)
8 # print(text.split())
9 # print(text.replace('5', 'ПЯТЬ'))
10 print(text.isdigit())
11 # print(text.lower())
12 # print(text.upper())

Run main.py
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
3464356
True

Process finished with exit code 0
```

LOWER() UPPER() TITLE()

- Много методов, которые работают с регистрами:
- lower() переводит все в нижний регистр
- upper() переводит все в верхний регистр
- capitalize() Переводит первый символ строки в верхний регистр, а все остальные в нижний
- title() первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
- startswith(str) начинается с подстроки str
- endswith(str) заканчивается на str
- И это не все!



```
# print(text.index())
# print(text.count())
# print(f'Исходный текст = {text}')

letter_a = 'Gfhjk'
letter_b = 'gfhjk'
print(letter_a.lower() == letter_b.lower())

Process finished with exit code 0
```



INDEX() RINDEX() FIND() RFIND()

- У каждого символа в строке есть свой индекс, и как часто бывает в программировании считать начинаем с нуля.
- Метод index() позволяет определить индекс первого вхождения символа. Первый аргумент – символ который ищем, второй – индекс, с которого начинаем поиск, по умолчанию это ноль.
- Метод rindex() – найдет последнее вхождение
- Но куда удобнее для этих целей использовать find() и rfind() – работают они также, но если вам вздумается поискать символ, которого в строке нет, то выдадут не ошибку, а значение -1
- Кстати, вместо цифры в качестве второго аргумента можно использовать переменную с типом int

```
# print(text.capitalize())
print(text.index('Y', 31))
# print(text.count())
# print(f'Исходный текст = {text}')

D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
fg2sj kFGe235rQ lkf55o6wiyf uT\TYk78jcb wGf4wBN
32

Process finished with exit code 0
```

МОРЖОВЫЙ ОПЕРАТОР :=



- И так, если мы хотим вывести индекс первого, а затем второго появления символа в строке, мы можем сделать так:
- Найти первый индекс и присвоить его переменной. А потом к этой переменной добавить 1 и использовать в качестве второго аргумента index()
- И фишка: моржовый оператор, который выглядит так := и позволяет присвоить значение переменной и вернуть это значение. Не обязательно использовать этот оператор сразу и везде. Не понимаете – просто оставьте на время.

```
# print(text.capitalize())
14 print(index := text.index('Y'))
15 print(text.index('Y', index+1))
16 # print(text.count())
17 # print(f'Исходный текст = {text}')
18
19
```

Run main.x

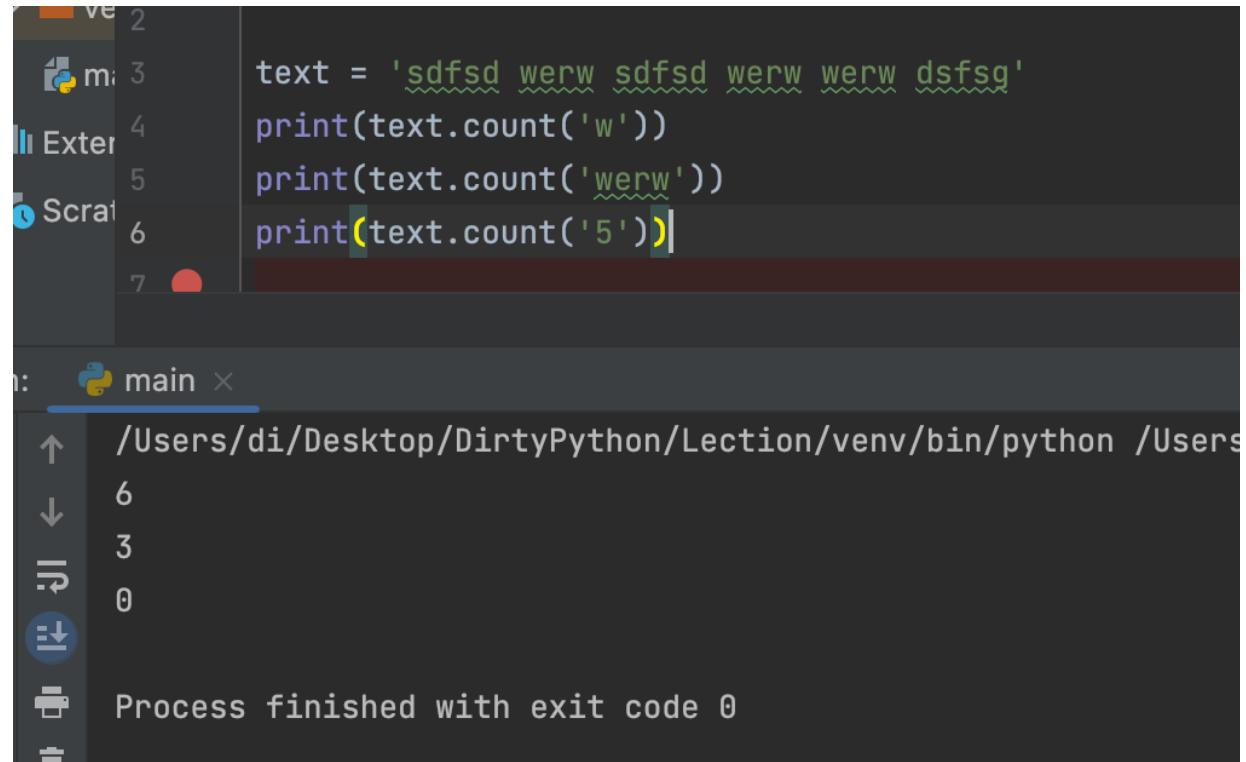
```
D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\StreamPython\main.py
fg2sj kFGe235rQ lkf55o6wiyf uTYTYk78jcb wGf4wBN
30
32

Process finished with exit code 0
```

Font size: 27pt Reset to 13pt Stream CRLF UTF-8

COUNT()

- Для того, чтобы посчитать сколько раз символ или подстрока встречаются в строке используем метод count().
- Помним, что даже если мы ищем цифру 5, то в count() передаем только в строчном виде '5', а то будет ошибка
- Если символа нет, то вернет 0



```
text = 'sdfsd werw sdfsd werw werw dsfsg'  
print(text.count('w'))  
print(text.count('werw'))  
print(text.count('5'))
```

main.py

/Users/di/Desktop/DirtyPython/Lecture/venv/bin/python /Users

6
3
0

Process finished with exit code 0

А ЧТО ЖЕ ПРОИСХОДИТ С САМОЙ СТРОКОЙ?



- А ничего. Какие бы методы не применяли, сама строка остается **неизменной**

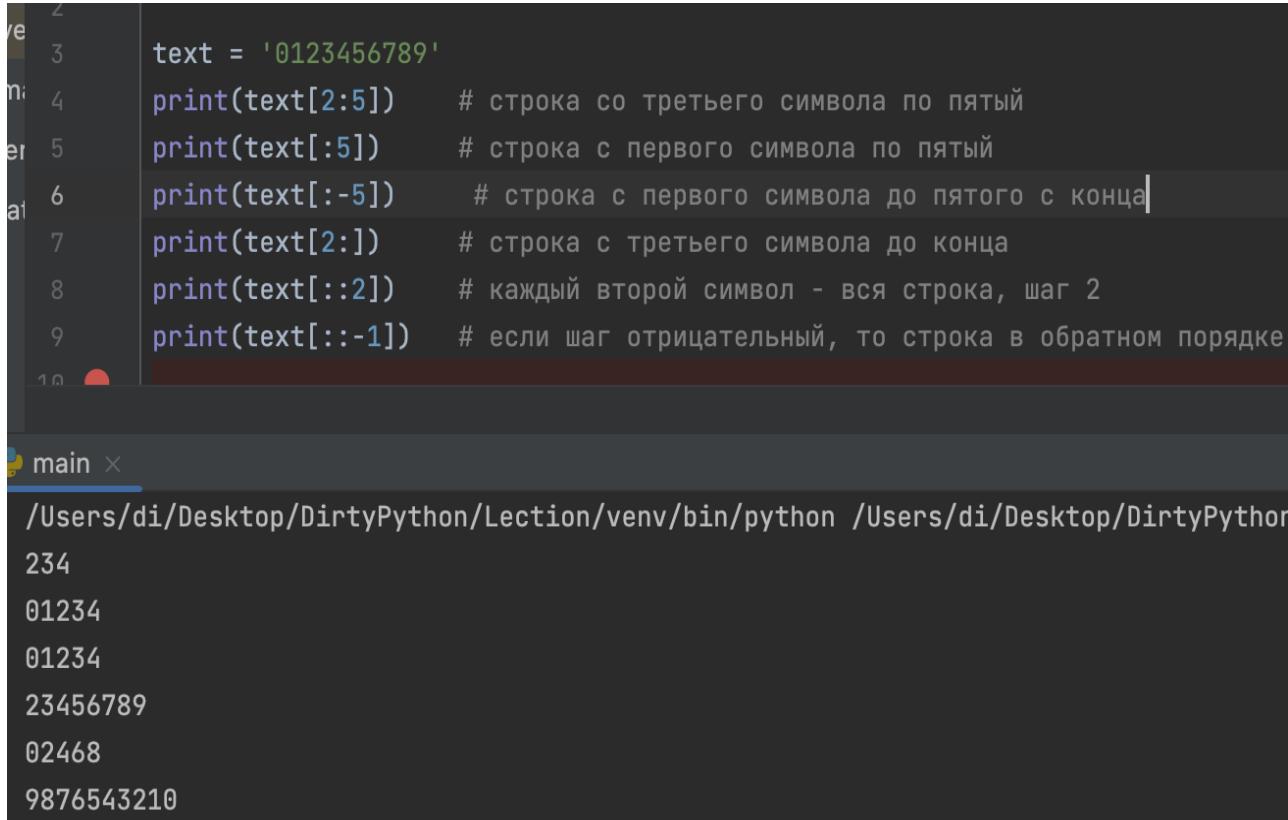
```
print(text.isdigit())
print(text.isalpha())
print(text.lower())
print(text.capitalize())
print(text.find('Y'))
```

```
U:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe U:\WorkShopPy\StreamPython\main.py
Дг2сј kFGe235rQ lkf55o6wiyf uTYTYk78jcb wGf4wBN
['Дг2сј', 'kFGe235rQ', 'lkf55o6wiyf', 'uTYTYk78jcb', 'wGf4wBN']
Дг2сј kFGe23ПЯTъrQ lkfПЯTъПЯTъo6wiyf uTYTYk78jcb wGf4wBN
False
False
дг2сј kfge235rq lkf55o6wiyf utytyk78jcb wgf4wbn
Дг2сј kfge235rq lkf55o6wiyf utytyk78jcb wgf4wbn
30
0
Исходный текст = Дг2сј kFGe235rQ lkf55o6wiyf uTYTYk78jcb wGf4wBN
```

Process finished with exit code 0

СРЕЗЫ

- Если нам нужен только кусок строки, например первые пять символов, или с десятого по двадцатый, то нам поможет срез
- Оформляется это так
- `Text[start:finish:step]`
- Где `start` – откуда начинаем (включительно)
- `finish` - до какого символа(не включительно)
- `step` – с каким шагом (по умолчанию единичка, можно не прописывать)
- И немногих примеров чтобы понять, как это работает



```
text = '0123456789'
print(text[2:5])      # строка со третьего символа по пятый
print(text[:5])       # строка с первого символа по пятый
print(text[:-5])     # строка с первого символа до пятого с конца
print(text[2:])       # строка с третьего символа до конца
print(text[::2])      # каждый второй символ - вся строка, шаг 2
print(text[::-1])     # если шаг отрицательный, то строка в обратном порядке
```

main

```
/Users/di/Desktop/DirtyPython/Lecture/venv/bin/python /Users/di/Desktop/DirtyPython
234
01234
01234
23456789
02468
9876543210
```

УМНОЖАЕМ И СКЛАДЫВАЕМ СТРОКИ



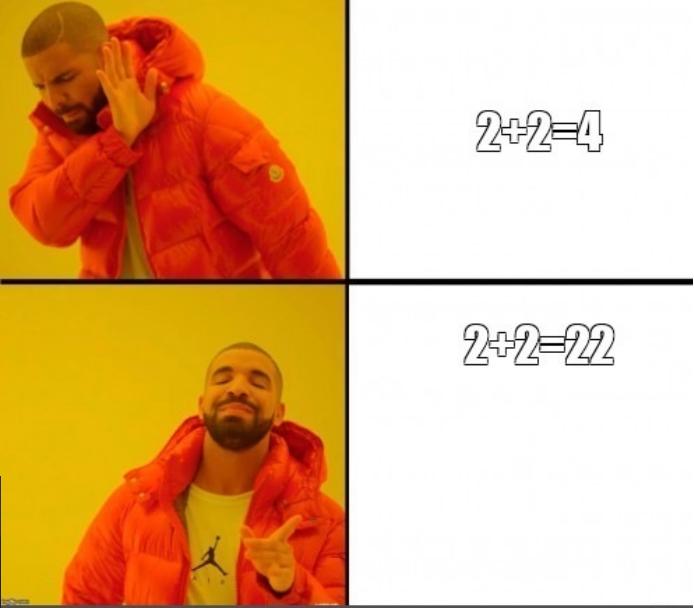
- А еще строки можно складывать. Правда, только со строками – конкатенация. И помним, что от перемены мест результат меняется
 - И умножать. Правда только на int

```
6 number_1 = '890'
7 number_2 = '10'
8
9 print(number_1 + number_2)
10
```

Run main x

D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.

89010



БУЛЕВЫЕ ЗНАЧЕНИЯ



- Принимают два вида True False
- Нужны нам для сравнения: написав 5>6 мы получим False
- and, or, not нам нужны для логический операций

```
5 a = True
6 b = False
7 print(a and a) # истина и истина -> истина
8 print(a and b) # истина и ложь -> ложь
9 print(b and b) # ложь и ложь -> ложь
10 print(a or a) # истина или истина -> истина
11 print(a or b) # истина или ложь -> истина
12 print(b or b) # ложь или ложь -> ложь
13 print(not a) # не истина -> ложь
14 print(not b) # не ложь -> истина
```

main ×

```
/Users/di/Desktop/DirtyPython/Lecture/venv/bin/python /Us
True
False
False
True
True
False
False
True
```

БУЛЕВЫЕ ЗНАЧЕНИЯ

- Кроме того, следует помнить что 0 - это False, а 1 – True
- Все пустые значения нам вернут False
- Все остальные случаи будут True

```
# print(a)
print(bool(None))
print(bool(0))
print(bool(''))
print(bool([]))
print(bool({}))
print(bool(()))
```

- Где используем булевые значения:
- Циклы, ветвления, флаги ...



И ЧТО ДАЛЬШЕ?

- А дальше еще много чего!

От Стоуна

- + максимально подробное и ЖИВОЕ объяснение, никаких предварительных записей трехлетней давности!
- + 6 лекций и 6 семинаров
- + разбор домашних заданий с примерами как надо и как не надо!
- + поддержка вне лекций и семинаров: вы всегда можете задать вопрос и получить на него ответ
- + невероятная харизма!

От команды:

- + конспекты
- + ответы на организационные вопросы и не только – по мере сил будем отвечать и по содержанию курса
- + доброжелательность и настрой

От вас:

- + желание и мотивация
- + 2500 руб



D1RTY
РУТНОН

ОСТАЛИСЬ ВОПРОСЫ?

- ПО ОРГАНИЗАЦИОННЫМ ВОПРОСАМ
СМЕЛО ПИШЕМ ДИАНЕ В ЛИЧКУ
- ЕСЛИ ОСТАЛИСЬ ВОПРОСЫ ПО ТЕМЕ:
 - 1) ПЕРЕСМОТРИТЕ ЕЩЕ РАЗ
 - 2) СПРОСИТЕ В ОБЩЕЙ ГРУППЕ (И ДА, НЕ СТЕСНЯЕМСЯ ПОМОГАТЬ ДРУГ ДРУГУ, ХОТЯ И МЫ ВАС НЕ БРОСИМ)
 - 3) ПОПРОБУЙТЕ ОБЪЯНТИТЬ УТОЧКЕ)
 - 4) ПОДУМАЙТЕ ТАК ЛИ ЭТО ВАЖНО, И ЕСЛИ ДА ПИШИТЕ СТОУНУ В ЛИЧКУ

https://t.me/et_oneya

