GeekBrains

Знакомство с Python

Часть 1

Изучим основные конструкции и классы данных, используемые в Python

Илья Акчурин

Python-разработчик, преподаватель программирования в вузе

- Преподаю языки Python и C++
- Автор программ
- Общий стаж преподавания больше 7 лет
- В прошлом руководитель ИТ-подразделения

Подготовка

Необходимые компоненты

Сам Python

Без интерпретатора никуда.
Скачать можно здесь

IDE (среда разработки)

Используем PyCharm. Скачать можно здесь

Установка

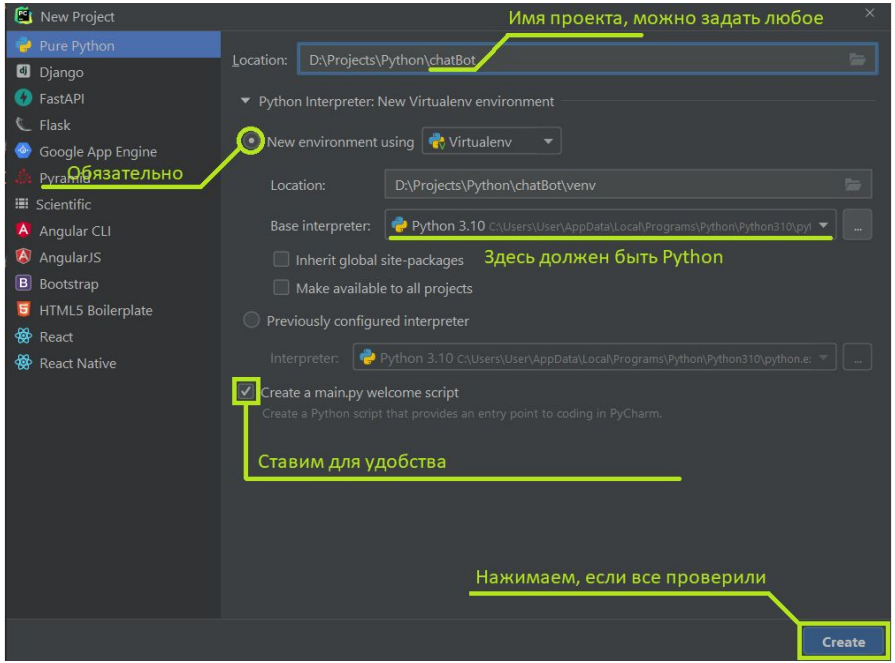
Стандартная,
без особенностей

Создание и запуск проекта

Обязательно создать проект, в котором будем работать дальше.

Для этого:

- Выбираем пункт **File → New Project**
- Убеждаемся, что всё соответствует изображению справа

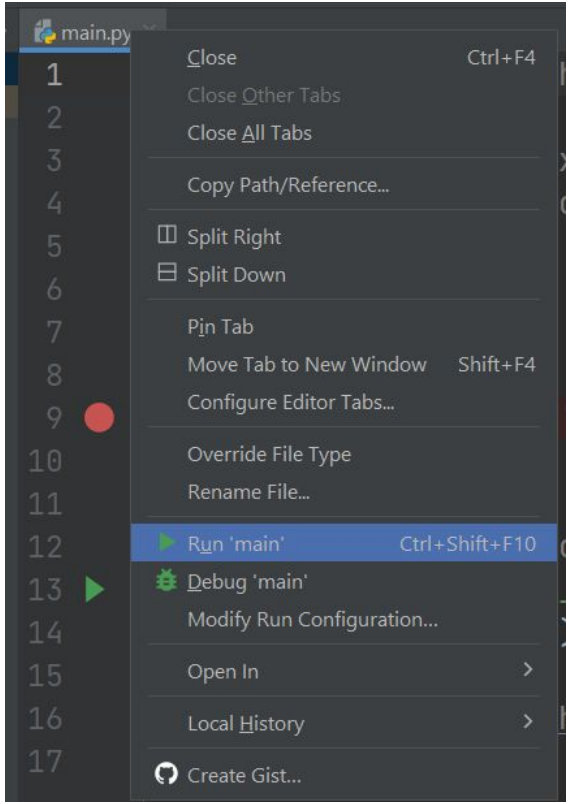


Создание и запуск проекта

Для запуска проекта можно кликнуть правой кнопкой мыши по закладке файла **main.py** и выбрать:

- **Run ‘main’** — если запускаем в обычном режиме (горячие клавиши *Ctrl+Shift+F10*)
- **Debug ‘main’** — если запускаем в режиме отладки.

Важно! Чтобы режим отладки заработал, необходимо сначала поставить Breakpoint на строчке, где необходимо начать отладку. На картинке справа Breakpoint стоит в девятой строке.



Классы и их объекты

Всё есть объект

Python — объектно-ориентированный язык, поэтому любой элемент, который вы создаёте (переменная, обычное число или что-то ещё), является объектом какого-либо класса. Чтобы узнать, к какому классу принадлежит объект, можно воспользоваться функцией **type()**. Напишем следующий код:

```
a = 10
print(type(a))
```

Увидим, что даже обычное число — это объект класса int:

```
<class 'int'>
```

Динамическая типизация

Python не является строго типизированным языком. А значит, при создании переменной мы можем не указывать класс (тип) данных, которые будут содержаться в ней. Python автоматически определит класс будущего объекта (в нашем случае — переменной) на основании внутреннего содержимого и создаст его.

Более того, мы можем «на ходу» переопределять класс (тип) хранимых в переменной данных. Такую типизацию называют **динамической**.

Справа показаны различия при создании переменных в языках C++ и Python.

C++ (статическая типизация)

```cpp
std::string price;      // Перменная типа string
price = "thousand";     // Записали значение строкове
price = 1000;           // Ошибка, изменить тип данных невозможно
```

Python (динамическая типизация)

```python
price = “thousand”  # У переменной есть тип
price = 1000        # Тип переменной динамически изменился
```

Основные классы

Основные классы (типы данных), используемые для решения любых задач, перечислены ниже.

```
a = 10          # <class 'int'>       целые числа
b = 5.2         # <class 'float'>     вещественные числа
c = 'Hello'     # <class 'str'>       строки
d = True        # <class 'bool'>      булев тип

```

Классы последовательностей

Помимо основных классов есть последовательности: классы, позволяющие хранить в себе несколько значений.

```
1 e = ['one', False, 3]   # <class 'list'>    списки
2 f = (1, 2, 3)           # <class 'tuple'>   кортежи
3 g = {'one': 1}          # <class 'dict'>    словари
4 h = {'a', 'd', 'f'}     # <class 'set'>     множества
```

Имена переменных (объектов)

snake_case!

first_name — верно

firstName — **неверно**

Имена со смыслом!

max_attempts — верно

a — **не очень удачно**

Никакого транслита!

name — хорошо

imya — **нельзя**

Нельзя ключевые слова

user_lst — можно

list, pass, def, len — **нельзя**

Операции с объектами

Арифметические операции

Python поддерживает все базовые арифметические операции и даже некоторые дополнительные. Результат вычисляется автоматически.

Операция	Название	Пример
+	Сложение	result = 340 + 52
-	Вычитание	result = 340 — 52
*	Умножение	result = 340 * 52
/	Деление	result = 340 / 52
//	Целочисленное деление	result = 340 // 52
%	Остаток от деления	result = 340 % 52
**	Возведение в степень	result = 3 ** 5

Базовые логические операции

Все операции сравнения, а также логические «И» и «ИЛИ». Результат — всегда объект класса bool.

Операция	Название	Пример
>, >=	Больше, Больше или равно	result = apple >= people
<, <=	Меньше, Меньше или равно	result = apple < people
==	Равно	result = apple == people
!=	Неравно	result = apple != people
and	логическое И	result = True and False
or	логическое ИЛИ	result = True or False or True

Приоритет операций

Как и в математике, все операции имеют приоритет, который можно изменять с помощью скобок.
Приоритеты выполнения операция от самого низкого к самому высокому:

=
if — else
or
and
not
in, not in, is, is not, <, <=, >, >=, !=, ==
+,-
*, /, //, %
+x, -x
**
x[index], x[index:index], x(arguments...), x.attribute

Ветвления

Варианты выполнения кода

Программы нелинейны и должны в разных условиях работать по-разному. Для этого существует конструкция if — elif — else.

Общая схема оператора выглядит следующим образом:

```
if <условие>:
    код, выполняется, если условие в if истинно
elif <условие>:
    код, выполняется, если условие в elif истинно
...
else:
    код, выполняется, если все условия выше были ложными
```

В <условии> всегда должно быть выражение, которое вернет **True** или **False** после своего выполнения. Это может быть как логическое выражение, так и функция, которая возвращает результат булевого типа.

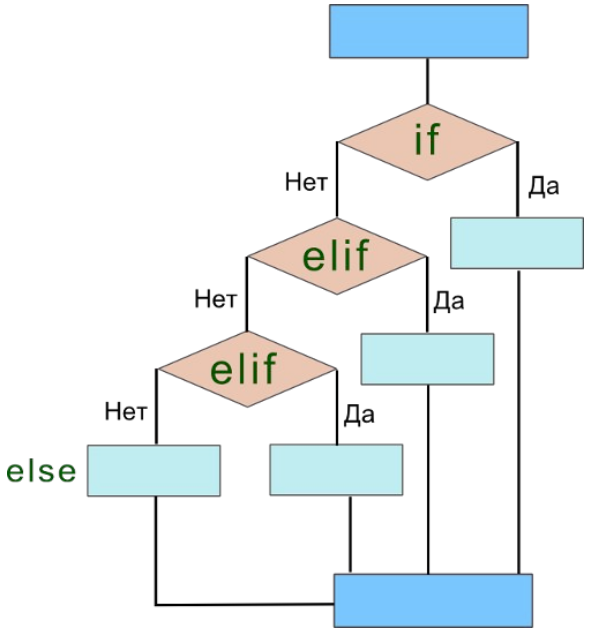
Количество блоков **elif** может быть сколь угодно большим.

Варианты выполнения кода

Пример использования конструкции:

```
age = 12
if age == 10:
    print("Что выйдет, если клюква наденет брюки?")
    print("Брюква!")
elif age == 11:
    print("Что сказала зеленая виноградина синей виноградине?")
    print("Дыши! Дыши!")
elif age == 12:
    print("Что сказал 0 числу 8?")
    print("Привет, ребята!")
elif age == 13:
    print("Что такое: на потолке сидит и хохочет?")
    print("Муха-хохотуха!")
else:
    print("Что-что?")
```

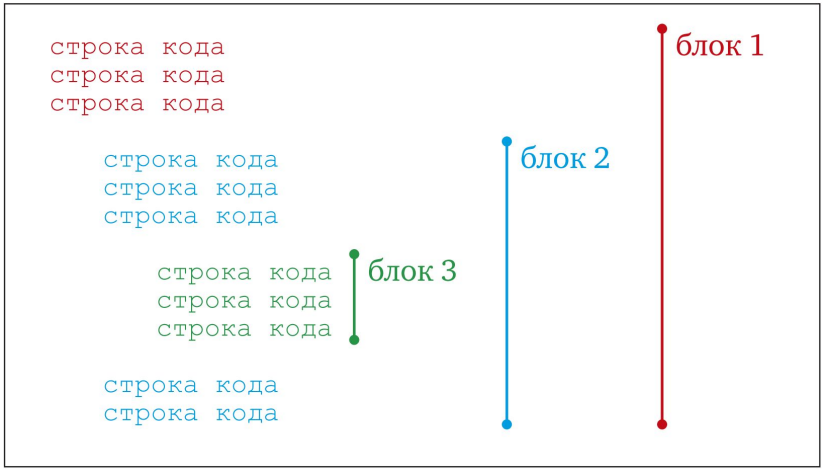
Схематично if-elif-else можно представить так:



Блоки

Обратите внимание: команды внутри if, elif, else написаны с отступом. Это не для красоты. Python жёстко требует выделять блоки кода (используя один Tab или 4 пробела), который будет выполняться при определённых условиях.

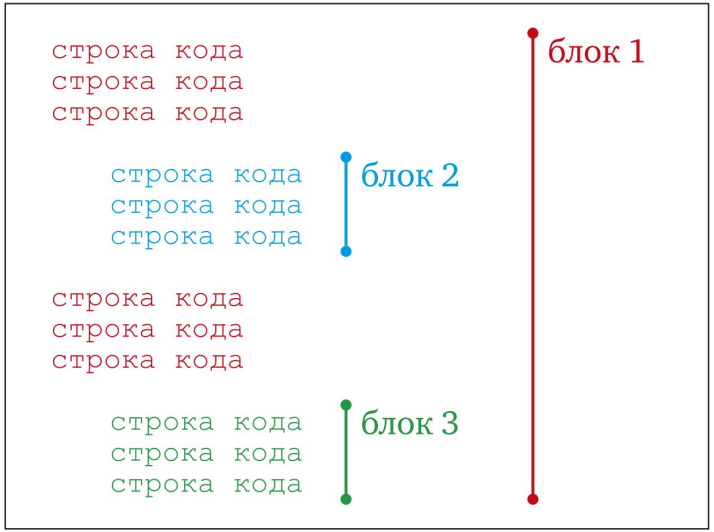
Также важно, чтобы команды внутри одного блока имели **одинаковые** отступы. Каждый раз, когда строка начинается с большего числа пробелов, чем предыдущая, создаётся новый блок, вложенный в предыдущий.



Блоки

Мы помещаем в блоки команды, которые логически связаны, то есть команды, которые нужно выполнять вместе.

Смена отступа — способ создания новых блоков. Вот пример трёх блоков, которые существуют только благодаря изменению величины отступов:



Практическое задание

Реализуйте проверку ввода на число.

Программа должна выводить “Correct”, если введено **целое** или **вещественное** число (в качестве разделителя должна использоваться одна точка).

Во всех остальных случаях должно выводиться “Wrong”.

5	3.4	3.4.1	1a	a3	-123	-5.321
Correct	Correct	Wrong	Wrong	Wrong	Correct	Correct

Для выполнения задания необходимо изучить методы строк

Вопросы?

Вопросы?

Вопросы?