Экспресс введение в Python 3.6

Муромцев Никита Андреевич
Выпускник ВМК МГУ,
аспирант геологического факультета МГУ
2017 год

Две версии Python 2 и 3

1. Изменили стандарт:
   - Что-то упростили
   - Сделали более логичным (к примеру, оператор *print* стал функцией *print()*)
   - Усилили форматы данных под 21 век (появился текст, другой подход к бинарным и т.п.)
   - Чуть удобнее читается, что хорошо в больших проектах
   - Добавили больше встроенных функций
2. Язык разделился на два лагеря
   - Много проектов на языке версии 2 потребовали переработки
   - Олдфаги
3. Активно поддерживаются оба стандарта

Возможности языка Python

Вот лишь некоторые из них:

- Работа с xml/html/txt/бинарными файлами
- Работа с http запросами
- GUI (графический интерфейс)
- Создание веб-сценариев
- Работа с изображениями, аудио и видео файлами
- Робототехника
- Программирование математических и научных вычислений

Другим языком:

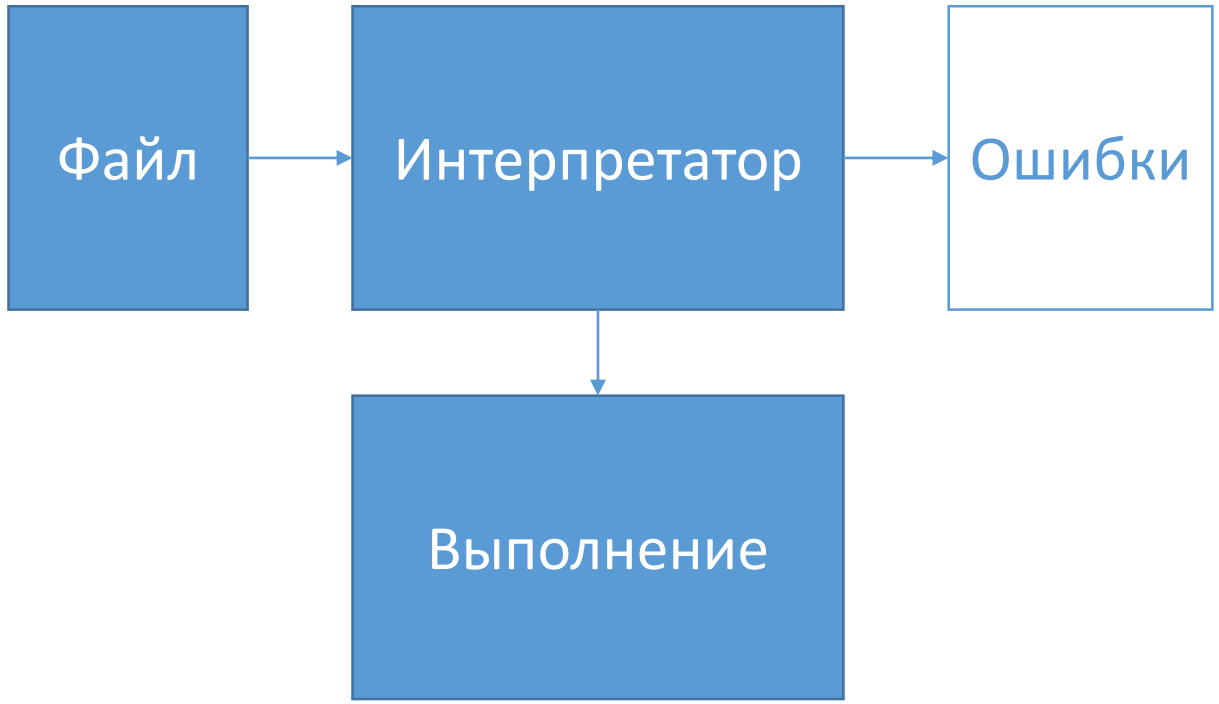
- Работа с файлами научных пакетов, файлами Excel
- Данные из интернета
- Приложения с кнопками/графикой
- Сайт / деталь сайта
- Сгенерировать график, анимацию
- Автоматизировать работу
- Автоматизировать свои вычисления объемные/рутиные задачи

Грубо говоря: ВСЁ

Интерпретатор Python

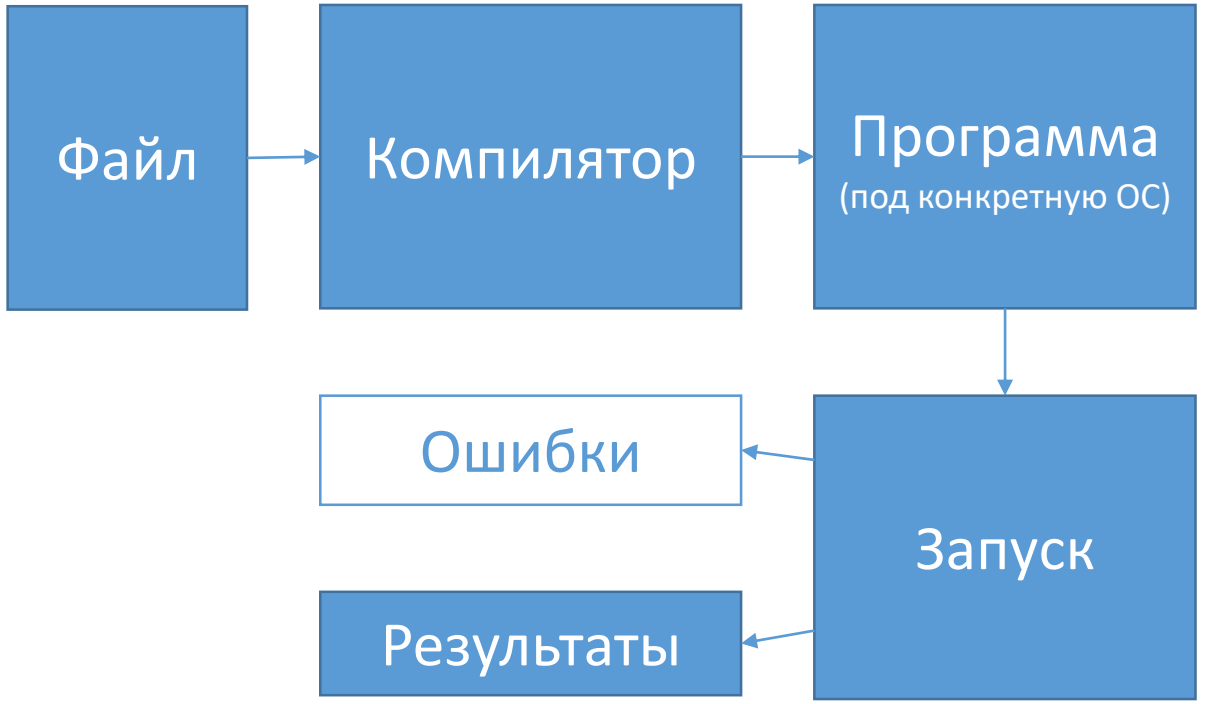
Python - интерпретируемый язык, но может быть и скомпилирован

Интерпретируемые языки



- Т.е. запускается везде, где есть интерпретатор
- Правится в текстовом редакторе
- После правки готово к повторному запуску

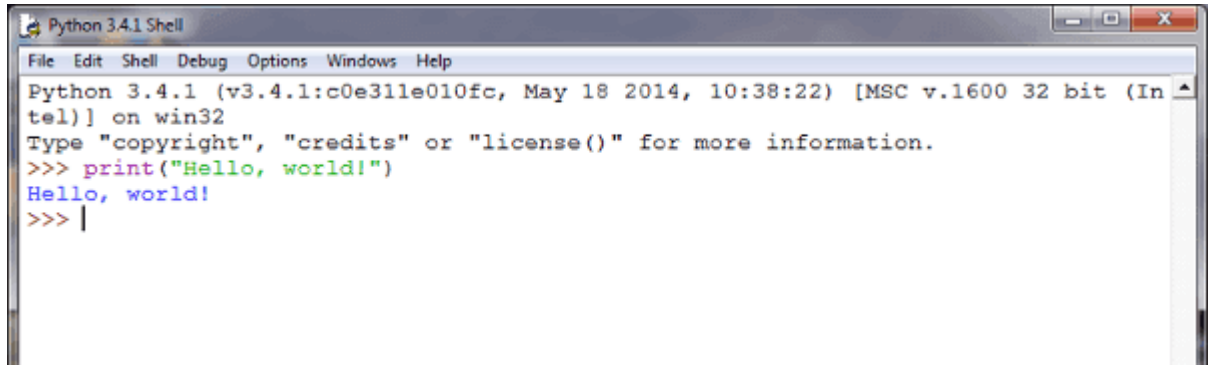
Компилируемые языки



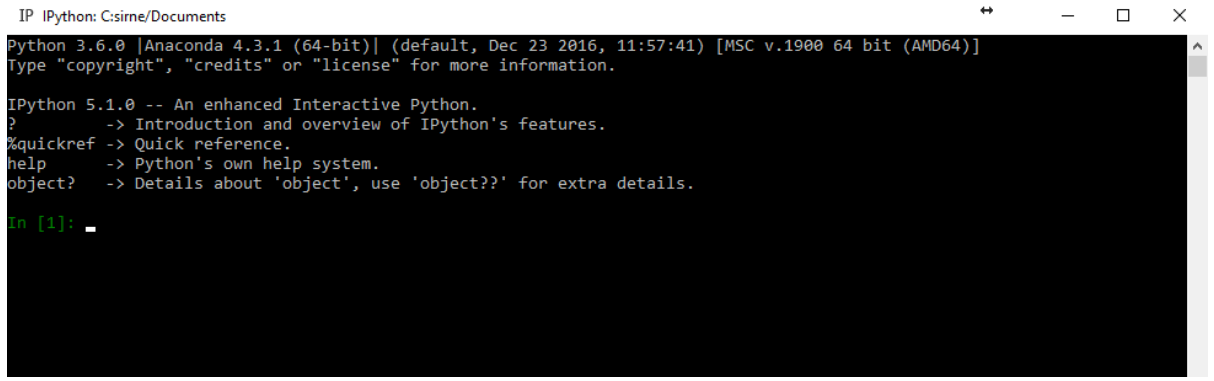
- Запускается на любой ОС, которую поддерживает
- Для изменений нужен исходный код
- Может не хватать библиотек и программа не соберётся

Консоль python

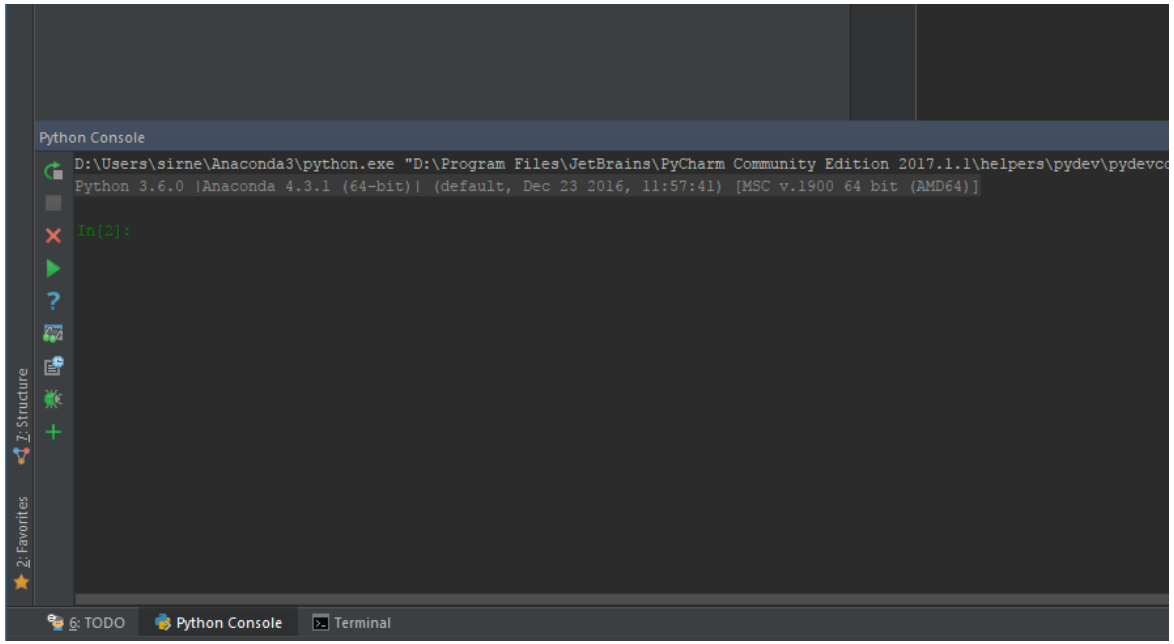
В любом интерпретируемом языке можно писать программу «налету», т.е. подавать команды и сразу получать результат.



Python консоль в стандартном пакете



Python консоль Anaconda



Python консоль в PyCharm

Базовые команды

Все команды ни раз описаны в интернете, поэтому здесь будет приведён список, чтобы вы знали что искать, если забудете

Простая арифметика (операторы)	+, -, *, /
Остаток от деления и целая часть от деления	% и //
Пара (x // y, x % y)	divmod(x,y)
Возведение в степень	x**y
Модуль числа и смена знака	abs(x), -x

Конечно, работают скобочки для придания приоритета

Пример:

```
>>> 1+1
2
```

Ещё можно применить операцию к самому себе:

```
x += 1  # увеличить на 1
```

Результат – целое число, если операнды (цифры, с которыми вы работаете) – целые числа

Битовые операции

Числа в компьютере представлены в двоичном виде. Так $5_{10} = 101_2$.

Побитовое «или», «исключающее или»	|, ^ (это на клавише 6)
Побитовое «и» и инверсия битов	& (клавиша 7) и ~x
Битовый сдвиг влево и вправо на n бит	x<<n, n>>y

Перевод в другие системы счисления числа n происходит с помощью функций:

- int(n, [x]) – перевод в x-ричную систему счисления; если x не указан, то 10-ричную
- bin(n) – в двоичную строку
- hex(n) – в шестнадцатеричную строку
- oct(n) – в восьмеричную строку

Типы данных

Некоторые операции работают только с определёнными(ым) типом данных

Целые числа (для перевода/создания - int(x))	1, 2345, 12334124124124153123123
Вещественные числа (для перевода/создания - float(x))	0.1, 234.1231234123
Рациональные (для перевода/создания - Fraction(x, y))	Fraction(2, 6) = Fraction(1, 3)
Вещественные с максимальной точностью (Decimal(x * y))	Decimal(7 * 4.9)
Комплексные (для перевода/создания - complex(x, y))	complex(1, 2)

```
int(0.11) = 0
int(1.1) = 1
float(1) = 1.0
```

```
float(«0.1») = 0.1
int(«0.1») = error
int(«1») = 1
```

```
x = complex(1, 2)
print(x)
>>> (1+2j)
y = complex(3, 4)
```

```
print(x + y)
>>> (4+6j)
print(x * y)
>>> (-5+10j)
```

Импорт библиотек

Для python 3 существует очень много сторонних библиотек (модулей) с функциями для большинства потребностей. Аналогично и для второй версии 2. При желании можно добиться совместимости библиотек 2 и 3 версий.

Импорт всей библиотеки	import <название>
Импорт библиотеки под другим именем	import <название> as <своё название>
Импортировать отдельную функцию	from <название библиотеки> import <название>
Импортировать под другим именем	from <библ.> import <название> as <название>

```
import math                 math.pi
import math as m            m.sqrt(85)
from math import sqrt       sqrt(85)
from math import sqrt as p  p
import random               random.random()
```

Ввод - вывод

Способов считывания и вывода информации программой большое количество

Вывод	print(<что выводить>)
Считывание	<куда считать> = import (<сообщение пользователю>)

```
print("Hello world!")
name = input("Как Вас зовут? ")
Как вас зовут _
```

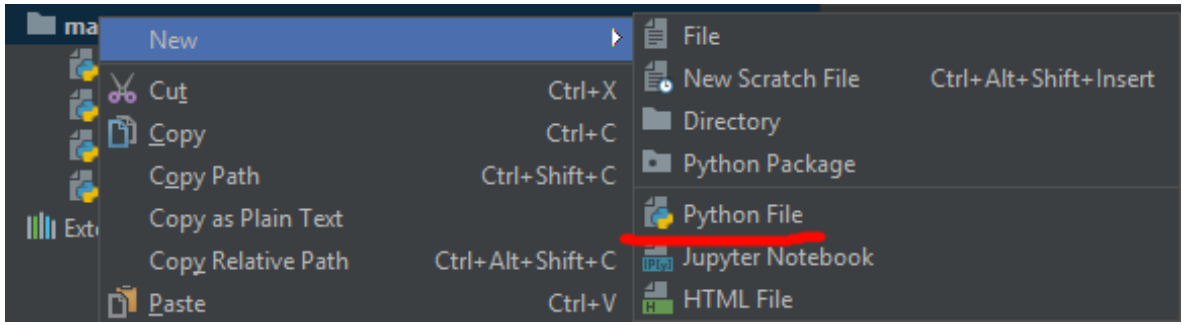
```
x = input()
z = x + 1
>>> error
```

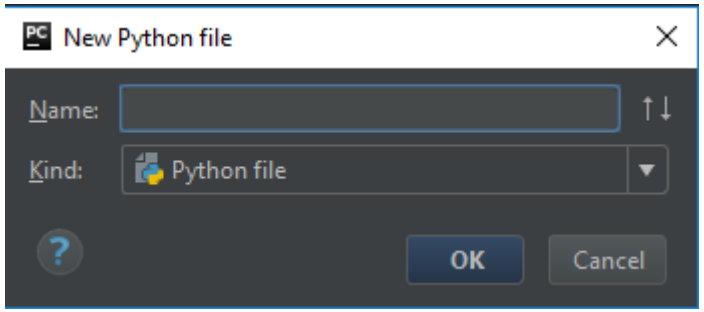
```
x = input()
z = int(x) + 1
>>> error
```

Создание python файла

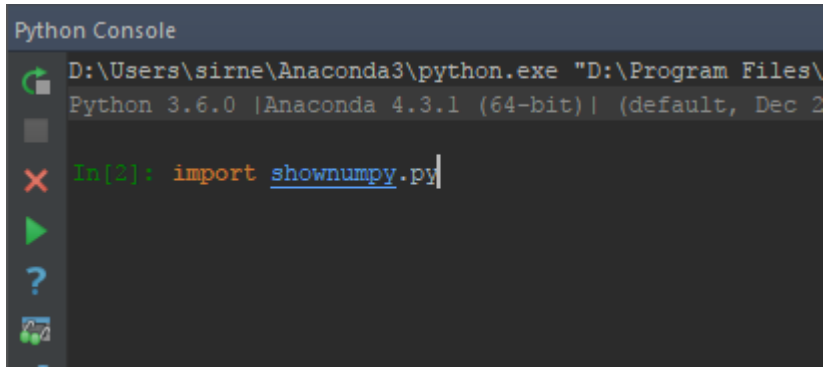
Для интерпретирования можно создать текстовый файл с любым названием и расширением .py

Как это делается в PyCharm





Как загрузить файл в консоль python



import <путь до файла>

После импорта будут доступны содержащиеся в нём функции, а также он запустится на исполнение, если какие-либо команды в нём не экранированы

Проверьте свои силы: http://informatics.mccme.ru/mod/statements/view.php?id=2296#1

Синтаксис

Синтаксис python достаточно прост, но есть несколько важных нюансов

Разделителем команд может служить либо переход на новую строку,
либо «;» (точка с запятой)

```
a = 1; b = 2; print(a, b)  # это корректно
```

Некоторые конструкции требую вложенности. В таком случае первая строка в конструкции является модулятором параметров или флагом запуска выполнения сложенных команд, а блок последующий команд являются инструкциями, которые выполняются в контексте запускающей строки.

```
if a > b:
    print("YEAH")
    print("21")
```

Здесь if a>b - условие запуска, а далее блок команд для выполнения

При этом верна команда if a > b: print("YEAH"), т.к. в блоке инструкций всего одна команда

Внутри файла каждый отступ должен состоять из одинаковых символов. Это может быть табуляция, либо 2,3,4 и более пробелов, но каждый раз одно и то же число.
Для каждого следующего уровня вложенности добавляем ещё один отступ.

Условия

Для использования условий необходимо знать логические операции. Не путайте с битовыми операциями, в данном случае результат либо «правда», либо «ложь»

И, или, отрицание(не)	X and Y, X or Y, not X
Больше, меньше, равно	X > Y, X < Y, X == Y (обратите внимание на два равно)
Не больше, не меньше, не равно	X <= Y, X >= Y (равно на втором месте), X != Y или X <> Y
Проверить есть ли элемент во множестве	X in Y аналогично «not in»
Сравнение операндов	x is y вернёт истину, если x и y ссылаются на один объект

Также бывают функции, которые выдают логический результат

```
from math import sqrt as s
from math import sqrt
print(s is sqrt)
>>> True
```

```
1 == 1
>>> True
1 <= 1
>>> True
1 < 1
>>> False
```

```
1 and 1
>>> 1
1 and 2
>>> 2
not 1
>>> False
```

```
False and True
>>> False
False or True
>>> False
not 1
>>> False
```

Условия

Варианты составления условий

Простое условие

```
if <условие>: <одна операция>
```

Условие с несколькими операциями

```
if <условие>:
    <операция 1>
    <операция 2>
```

Условие, в котором выполняется один из блоков операций

```
if <условие>:
    <операция 1>
    <операция 2>
else:
    <операция 1>
    <операция 2>
```

Серия условий, которые проверяются по очереди. Блок операций срабатывает у первого сработавшего. Если не будет else, то может не выполниться ничего

```
if <условие 1>:
    <операция 1>
    <операция 2>
elif <условие 2>:
    <операция 1>
    <операция 2>
elif <условие 3>:
    <операция 1>
    <операция 2>
else:
    <операция 1>
    <операция 2>
```

Трёхместное условие: A = Y if X else Z

<операция> if <условие> else <операция в противном случае>

Циклы

Цикл позволяет повторять блок операций по заданному условию

Цикл с предусловием

```
while <условие>:
        <операция 1>
        <операция 2>
```

Пока условие истино, повторять

- break досрочно прерывает цикл (в блоке операций)
- continue начинает следующий проход цикла, минуя оставшееся тело цикла (в блоке операций)
- else: операции в блоке else выполнятся, если цикл закончился без помощи break (на уровне с for/while)

Цикл с итератором по множеству

```
for <переменная> in  <множество>:
        <операция 1>
        <операция 2>
```

Операции повторятся столько раз, сколько элементов во множестве. В операциях можно использовать итератор.

Варианты множеств:

- Тестовая строка: «HELLO»
- Множество натуральных чисел:
  - range(start, stop)
  - range(count)
  - range(start, stop, step)

Функции

Чтобы блок операций можно было использовать многократно, для удобства, для некоторых стилей программирования и т.д. команды объединяют в функции

def <название> ([операнды, их может не быть]):

<операции>

return <результат, т.е. здесь писать переменную или сразу функцию>

return может не быть

```
a = 1
print("значение до", a)

def first(a):
    if a<10:
        a+=1
        print("сейчас a =", a)
        a = first(a)  # рекурсия
    return a

print("значение после", first(a))
```

Задачи для практики по занятию 1

Обязательно (1 балл, примерно на 1-2 часа):

- 2937 – ввод-вывод
- 2938 – яблоки
- 338 – цифры в обратном порядке
- 74 – a+b
- 596 – пробежка
- 595 – диета
- 2944 – сумма цифр
- 2949 – обмен значений
- 315 – сумма квадратов
- 120 – сумма и факториалы
- 334 – остаток
- 1476 – часовая стрелка
- 341 – делители числа
- 115 – количество нулей
- 3064 – длина последовательности

Дополнительно (1,5 балла):

- 597 – пробежка 2
- 2940 – мкад
- 2945 – четное число
- 2951 – пирожки
- 2955 – улитка
- 352 – степень
- 335 – полные квадраты
- 1433 – кролики

Особые (3,5 балла):

- 248 – шахматные слоны
- 2950 – расписание
- 2967 – найти значение
- 2957 – делимость
- 2958 - максимум
- 321 – сложная сумма

Дополнение

Если в одной строке при вводе будет несколько чисел, то нужно использовать функцию split()

Пример:

```
a, b, c = input().split()
```

a, b, c – получатся текстовыми

Чтобы a, b, c вышли числами необходимо применить следующую конструкцию^

```
a,b,c = map(int, input().split())
```