Table of Contents

Введение в программирование на python

Автор: Калитвин В.А.

kalitvin@gmail.com

Python - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью.

https://www.python.org/

Автор - сотрудник голландского института CWI Гвидо ван Россум.

Год создания - 1991.

Особенности языка

- Язык высокого уровня
- Объектно-ориентированный
- Интерпретируемый
- Язык со строгой динамической типизацией
- Универсальный
- Свободно-распространяемый

Основные ветки

- Python 2.x
- Python 3.x

Парадигмы программирования

- Императивное (структурный, процедурный, модульный подходы)
- Объектно-ориентированное
- Функциональное

PEP8 - рекомендации по оформлению кода

- отступ - 4 пробела
- длина строки < 80 символов
- змеиный стиль именования переменных: my_variable
- константы: CONST_NAME

https://www.python.org/dev/peps/pep-0008/

Популярные среды разработки:

- PyCharm
- Eclispe + pydev
- Anaconda
- Thonny
- Vim
- Emacs

Проекты на python

- Dropbox
- Instagram
- Google
- Spotify
- Netflix
- Uber
- Pinterest
- Bitbucket

Дзен python

In [1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Перевод

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если они не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один и, желательно, только один очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить, то это плохая идея.
- Если реализацию легко объяснить, то идея, возможно хороша.
- Пространства имён - отличная штука! Будем делать их больше!

Простейшая программа на python

In [2]:
```
print("Hello, World!")
```

```
Hello, World!
```

лучше

In [3]:
```
def main():
    print("Hello, World!")


if __name__ == 'main':
    main()
```

Схема выполнения программы

Типы данных

- None
- Логический (bool)
- Числа (int, float, complex)
- Последовательности (str, list, tuple)
- Множества (set, frozenset)
- Словари (dict)
- Файлы (file)
- Исключения (exceptions)
- Другие (byte, ...)

Операторы

Арифметические операторы

+	сложение (конкатенация)
-	вычитание
*	умножение (повторение)
/	деление
//	деление с усечением дробной части
**	возведение в степень
%	остаток от деления

Операторы сравнения:

\<	a < b	a меньше b
>	a > b	a больше b
\<=	a <= b	a меньше или равно b
>=	a >= b	a больше или равно b
==	a == b	a равно b
!=	a != b	a не равно b

Логические операторы

and логическое И

or логическое ИЛИ

not логическое отрицание

Операторы проверки идентичности объектов: is; is not.

Операторы проверки вхождения в последовательность: in, not in.

Операторы присваивания

+=	a += b	a = a + b
-=	a -= b	a = a — b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
**=	a **= b	a = a ** b
%=	a %= b	a= a % b

Битовые операторы

Битовые операторы

Группировка в выражениях осуществляется с помощью круглых скобок.

Переменные

В python все является объектом (в том числе и переменные).

Переменные можно представить как ссылки на объекты, содержащие значения.

В python не требуется объявлять тип переменных.

Тип данных определяется автоматически по типу выражения, стоящего после занка =. При этом можно изменять значение переменной одного типа на значение другого типа.

Тип переменной можно узнать с помощью команды type.

У каждого объекта есть идентификатор. Узнать идентификатор можно с помощью метода id().

In [4]:
```
a = 1  # int
print(type(a))
print(id(a))
```

```
<class 'int'>
140008430502128
```

In [5]:
```
b = 1.0  # float
print(type(b))
print(id(a))
```

```
<class 'float'>
140008430502128
```

In [6]:
```
c = 1 + 1j  # complex
print(type(c))
print(id(c))
```

```
<class 'complex'>
140008275196752
```

Изменяемые и неизменяемые типы

К неизменяемым типам относятся int, float, str, tuple. К изменяемым типам относятся list, dict.

In [7]:
```
x = 1
print(id(x))
x = 2
print(id(x))
```

```
140008430502128
140008430502160
```

In [8]:
```
x = [1, 2, 3]
print(x)
print(id(x))
x[0] = 5
print(x)
print(id(x))
```

```
[1, 2, 3]
140008222078336
[5, 2, 3]
140008222078336
```

Имена переменных

- Имя переменной может содержать только буквенно-цифровые символы и подчеркивание (Az, 0-9 и _).
- Имя переменной должно начинаться с буквы или символа подчеркивания _.
- Имя переменной не может начинаться с цифры.
- Имена переменных чувствительны к регистру (abc, Abc и ABC — три разные переменные).
- Имя переменной не должно совпадать с зарезервированным ключевым словом.

Список ключевых слов

In [9]:
```
import keyword

print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'bre
ak', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'fina
lly', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'n
onlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with
', 'yield']
```

Комментарии

Однострочные комментарии начинаются с символа #. Многострочных комментариев нет, но можно заключать текст комментария в тройные кавычки.

```
"""
comment
comment
comment
"""
```

Условный оператор

Применяется для организации ветвлений в программе

```
if logic_expression1:
    code1
elif logic_expression2:
    code2
else:
    code3
```

In [10…

```
a = 8
b = 80
if a > b:
    print("a > b")
elif a < b:
    print("a < b")
else:
    print("a = b")
```

```
a < b
```

Сокращенный вариант

```
a = y if x else z
```

In [11…

```
number = 5
a = "чётное число" if number % 2 == 0 else "нечётное число"
print(number, a)
number = 6
a = "чётное число" if number % 2 == 0 else "нечётное число"
print(number, a)
```

```
5 нечётное число
6 чётное число
```

Цикл while

Применяется для выполнения повторяющихся действий

```
while logic_expression:
    code # тело цикла
```

Пример: вывести на экран целые числа от 1 до 10

In [12…

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Пример: найти сумму целых чисел от 1 до n

In [13…

```
n = 10
summa = 0

i = 1
while i <= n:
    summa += i
    i += 1

print(summa)
```

```
55
```

Прервать работу цикла можно с помощью инструкции break

In [14…

```
i = 1
while i <= 10:
    if i == 5:
        break
    print(i)
    i += 1
```

```
1
2
3
4
```

Полная форма записи цикла while

```
while logic_expression:
    code1 # тело цикла
else:
    code2 # выполняется, если не была выполнена команда break
```

In [15…

```
i = 1
while i <= 10:
    if i == 5:
        break
    print(i)
    i += 1
else:
    print("ok")
```

```
1
2
```

```
3
4
```

In [16…

```python
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print("ok")
```

```
1
2
3
4
5
6
7
8
9
10
ok
```

In [16…