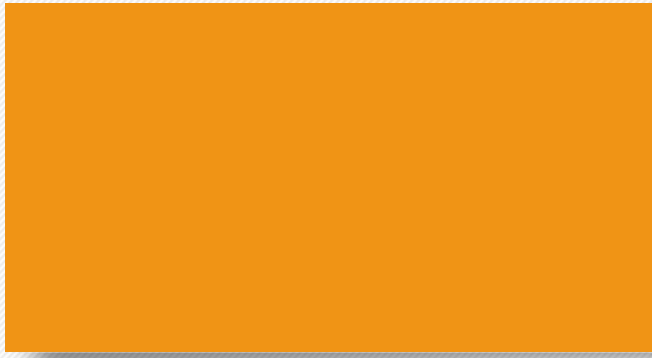
Введение в язык Python



Возможности языка Python

Гвидо ван Россум

1991 год
нидерландский
программист

- создавать web-приложения (Django, Flask),
- разрабатывать игры (Pygame),
- заниматься математическими вычислениями и анализом данных (NumPy, Pandas, Matplotlib),
- работать с текстовыми файлами, изображениями, аудио и видео файлами (PyMedia),
- реализовывать графический интерфейс пользователя (PyQT, PyGObject),
- работа с изображениями (OpenCV, Pillow)
- применение технологий ИИ (TensorFlow, Keras)

Среды разработки

- VS Code
- PyCharm
- Anaconda
- Google Colab
- Python IDLE

Синтаксис

Инструкция

```
>>> print("Python is awesome!")
Python is awesome!
>>> var = "first string"
>>> print(var)
first string
>>> var = 2 + 9
>>> print(var)
11
```

Кавычки

```
>>> print("String")
String
>>> print('String')
String
>>> print('''Str
... ing''')
Str
ing
>>> print('Str"ing')
Str"ing
```

Комментарии

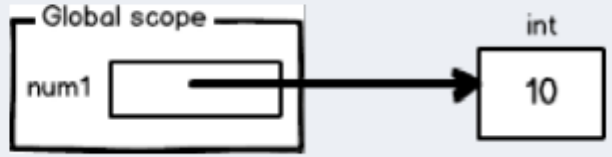
```
>>> print("комментарий справа") # Функция print() позволяет выводить результат на экран
комментарий справа
```

Отступ - 4 пробела

Типы

Типы данных

```
num1 = 10
```



Неявная динамическая типизация

Присвоение значения - процесс связывания ссылки с объектом

Типы:

- None - неопределенное значение переменной

```
null_variable = None
```

- Логический тип данных (bool)

  - **True** - логическая переменная, истина
  - **False** - логическая переменная, ложь

```
a = 0
b = 0
print(a < b)
print(a > b)
print(a == b)
```

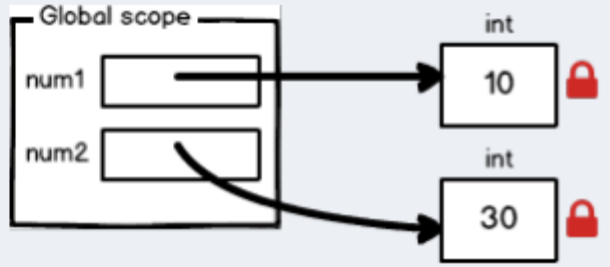
Строгая типизация ‘10’+20

Числовые типы

- **int** - целое число
- **float** - число с плавающей точкой
- **complex** - комплексное число

неизменяемые типы данных

```
num1 = 10
num2 = num1
num2 += 20
```



Строки

- **str** - строка

Целые числа

```
>>> 255 + 34
289
>>> 5 * 2
10
>>> 20 / 3
6.666666666666667
>>> 20 // 3
6
>>> 20 % 3
2
>>> 3 ** 4
81
>>> pow(3, 4)
81
```

```
>>> bin(3)
'0b11'
>>> hex(123)
'0x7b'
>>> oct(15)
'0o17'
```

Вещественные числа

```
>>> round(16.76)
17
>>> int(123.823)
123
```

Комплексные числа

```
>>> z = -14.3 + 7.083j
>>> z.real
-14.3
>>> z.imag
7.083
```

Типы коллекций

- **list** - список
- **tuple -** кортеж
- **range** - диапазон, неизменяемая последовательность целых чисел.
- **set** - множество
- **frozenset** - неизменяемое множество
- **dict** - словарь
- **bytes** - байты
- **bytearray** - массивы байт

Типы операторов

Арифметические операторы

```
>>> print(5 + 8)
13
```

```
>>> print(31 - 2)
29
```

```
>>> print(12 * 9)
108
```

```
>>> print(6 / 4)
1.5
```

```
>>> print(6 % 4)
2
```

```
>>> print(9 ** 2)
81
```

```
>>> print(6 // 4)
1
```

Операторы сравнения

```
>>> print(5 == 5)

True
>>> print(6 == 44)

False
```

```
>>> print(12 != 12)

False
>>> print(1231 != 0.4)

True
```

```
>>> print(53 > 23)

True
>>> print(432 >500)

False
```

```
>>> print(5 < 51)

True
>>> print(6 < 4)

False
```

```
>>> print(5 >= 5)

True
>>> print(6 >=44)

False
```

```
>>> print(32 <= 232)

True
>>> print(65 <= 9)

False
```

Операторы присваивания

```
>>> var = 5
>>> print(var)
5
```

```
>>> var = 5
>>> var += 4
>>> print(var)
9
```

```
>>> var = 5
>>> var -= 2
>>> print(var)
3
```

```
>>> var = 5
>>> var *= 10
>>> print(var)
50
```

```
>>> var = 5
>>> var /= 4
>>> print(var)
1.25
```

```
>>> var = 5
>>> var %= 10
>>>print(var)
5
```

```
>>> var = 5
>>> var **= 8
>>>print(var)
390625
```

Логические операторы

- and, or, not

Операторы членства

```
>>> print('he' in 'hello')

True
```

```
>>> print(5 in [1, 2, 3, 4, 5])

True
```

```
>>> print(12 in [1, 2, 4, 56])

False
```

Ввод - вывод

```
a = float(input("Enter number 1: "))
b = float(input("Enter number 2: "))
print("The sum of {} and {} is {}".format(a, b, a + b))
```

Фигурные скобки могут содержать дополнительные спецификаторы формата, например:

- {: 10} - дополнить выводимое значение до 10 символов
- {:< 8} - дополнить до 8 символов, выровнять по левому краю (по правому краю: >, по центру: ^)
- {: .4f} - вывести как вещественное число с 4 знаками после запятой

str.format:

```
print("The sum of {} and {} is {:.2f}".format(a, b, a + b))
```

f-string:

```
print(f"The sum of {a} and {b} is {c:.2f}")
```

Разветвляющиеся алгоритмы

Простое условие

```
# объявляем переменную
var = 5
# выполняем проверку условия
if var < 10:
# если условие выполняется, то
    print("var less than 10")
```

```
# объявляем переменные
var_1 = 10
var_2 = 9
# выполняем проверку условия
if var_1 == var_2:
# если True
    print("var_1 equal var_2")
# иначе
else:
# выполняется другая вложенная инструкция
    print("var_1 not equal var_2")
```

Вложенное условие

```
# объявляем переменные
var_1 = 10
var_2 = -10
# выполняем проверку условия
if var_1 == var_2:
# если True
    print("var_1 equal var_2")
# иначе если выполняется другое условие
elif var_1 < var_2:
# если True для elif
    print("var_1 less than var_2")
# если False для всех
else:
    print("var_1 more than var_2")
```

```
if (условие):
    (выполнение условия)
elif (другое условие):
    (выполнение другого условия)
elif (третье условие):
    (выполнение третьего условия)
elif (четвертое условие):
...
...
...
else:
    (выполнение при всех других не рассмотренных ранее случаях)
```

Независимые условия

```
var = 10
if var == 10:
    print("var equal 10")
if var < 10:
    print("var less than 10")
else:
    print("var more than 10")
```

```
var = 10
if var == 10:
    print("var equal 10")
elif var < 10:
    print("var less than 10")
else:
    print("var more than 10")
```

Сложные вложения

```
if (условие):
    if (дополнительное условие):
        (выполнение дополнительного условия)
    elif (другое дополнительное условие):
        (выполнение другого дополнительного условия)
    elif ...
        ...
    ...
    else:
        ...
elif (другое условие):
    (выполнение другого условия)
elif ...
    ...
else:
    ...
```

Условные выражения

```
<expression 1> if <condition> else <expression 2>
```

Без условного выражения:

```
if n >= 0:
    print("Absolute value =", n)
else:
    print("Absolute value =", -n)
```

С условным выражением:

```
print("Absolute value =", n if n >= 0 else -n)
```

Циклические алгоритмы

Цикл while (пока)

```
# создаем переменную, равную 1
var = 1
# прописываем цикл с условием - выполнять до тех пор, пока переменная
# меньше или равна 13
while var <= 13:
# выводим значение переменной
    print(var)
# увеличиваем переменную на 1
    var += 1
```

Действие будет повторяться до тех пор, пока не выполнится условие.

Бесконечный цикл

Цикл for (для)

- Способен проходить по любому итерируемому объекту, будь то списки, словари, кортежи, строки и не требует ручного увеличения счетчика итераций

```
for i in [1, 2, 3, 4, 5 ,6 ,7 ,8 ,9 , 10, 11, 12, 13]:
    print(i)
```

```
for i in range(1,14):
    print(i)
```

```
for i in range(1,14):
    print(i, end=' ')
1 2 3 4 5 6 7 8 9 10 11 12 13
```

Оператор continue, break, else

```
for var in 'Python':
    if var == 'h':
        continue
    print(var)
```

```
for var in 'Python':
    if var == 'h':
        break
    print(var)
```

```
for var in 'Python':
    if var == 'a':
        break
else:
    print('Символа a нет в слове Python')
```

Цикл с постусловием

```
while True:
    n = int(input("Enter a positive number: "))
    if n > 0:
        break
    print("Incorrect value. Try again!")
```

Бесконечный цикл

```
i = 1
while i <= n:
    print(i)
```

```
i = 1
while i <= n:
    print(i)
i += 1 # за пределами цикла
```

Вложенные циклы

```
# n = 6
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

```python
n = int(input("Enter n: "))
i = 0
while i < n:
    j = 0
    while j < i + 1:
        print(j + 1, end=' ')  # Нет переноса строки
        j += 1
    print()
    i += 1
```

Списки

Понятие списка

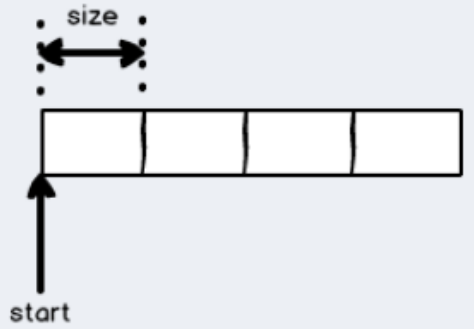
- Хранит набор элементов (возможно относящихся к разным типам данных) в виде непрерывного блока в памяти, в котором элементы следуют строго друг за другом
- Доступ к отдельному элементу предоставляется по индексу (начинается с 0)
- Поддерживает динамическое изменение размера

```
mixed_list = [10, "20", 30.0, True]
```

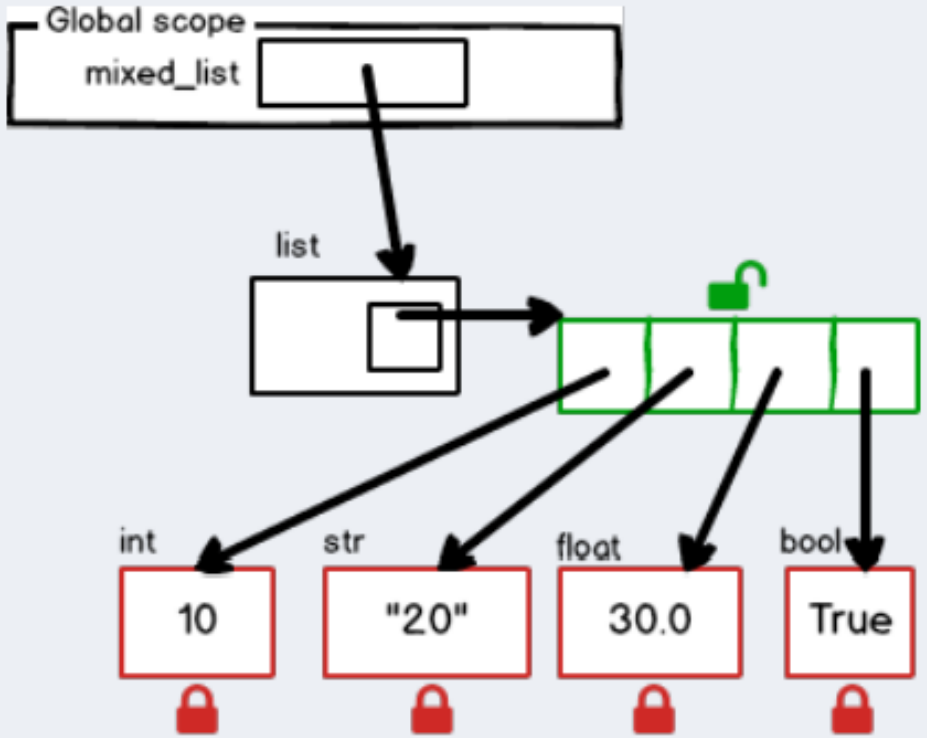
Модель памяти

Тогда чтобы получить доступ к i-му элементу, достаточно выполнить следующее простое вычисление:

$$address(i) = start + i * size$$




```
mixed_list = [10, "20", 30.0, True]
```



Понятие списка

```
>>> list('Python')
['P','y','t','h','o','n']
```

```
# Пустой список
>>> s = []
# список с данными разных типов
>>> l = ['s', 'p', ['isok'], 2]
>>> s
[]
>>> l
['s', 'p', ['isok'], 2]
```

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> b = [i * 3 for i in a]
>>> b
['PPP', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
```

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> len(a)
6
```

```
>>> s = ['P', 'y', 't', 'h', 'o', 'n']
>>> print(s[2], s[3])
t h
```

Методы для работы со списками

— **append(a)** - добавляет элемент a в конец списка

```
>>> var = ['l', 'i', 's', 't']
>>> var.append('a')
>>> print(var)
['l', 'i', 's', 't', 'a']
```

— **extend(L)** - расширяет список, добавляя к концу все элементы списка L

```
>>> var = ['l', 'i', 's', 't']
>>> var.extend(['l', 'i', 's', 't'])
>>> print(var)
['l', 'i', 's', 't', 'l', 'i', 's', 't']
```

Методы для работы со списками

— **insert(i, a)** - вставляет на **i** позицию элемент **a**

```
>>> var = ['l', 'i', 's', 't']
>>> var.insert(2,'a')
>>> print(var)
['l', 'i', 'a', 's', 't']
```

— **remove(a)** - удаляет первое найденное значение элемента в списке со значением **a**, возвращает ошибку, если такого элемента не существует

```
>>> var = ['l', 'i', 's', 't', 't']
>>> var.remove('t')
>>> print(var)
['l', 'i', 's', 't']
```

Методы для работы со списками

— **pop(i)** - удаляет **i**-ый элемент и возвращает его, если индекс не указан, удаляет последний элемент

```
>>> var = ['l', 'i', 's', 't']
>>> var.pop(0)
'l'
>>> print(var)
['i', 's', 't']
```

— **index(a)** - возвращает индекс элемента **a** (индексация начинается с 0)

```
>>> var = ['l', 'i', 's', 't']
>>> var.index('t')
3
```

Методы для работы со списками

— **count(a)** - возвращает количество элементов со значением **a**

```
>>> var = ['l', 'i', 's', 't']
>>> var.count('t')
1
```

— **sort([key = функция])** - сортирует список на основе функции, можно не прописывать функцию, тогда сортировка будет происходить по встроенному алгоритму

```
>>> var = ['l', 'i', 's', 't']
>>> var.sort()
>>> print(var)
['i', 'l', 's', 't']
```

Методы для работы со списками

— **reverse()** - разворачивает список

```
>>> var = ['l', 'i', 's', 't']
>>> var.reverse()
>>> print(var)
['t', 's', 'i', 'l']
```

— **clear()** - очищает список

```
>>> var = ['l', 'i', 's', 't']
>>> var.clear()
>>> print(var)
[]
```

— **copy()** - поверхностная копия списка, при присвоении переменной копии списка, значение данного списка не изменяется в случае изменения первого. Если переменной присвоить список через "**=**", тогда значение этой переменной будет меняться при изменении оригинала

```
>>> var = ['l', 'i', 's', 't']
>>> asd = var.copy()
>>> print(asd)
['l', 'i', 's', 't']
```

Вложенные списки

```
matrix = [
    [20, 40, 80],
    [-70, 100, 60]
]

# выводит 60
print(matrix[1][2])
```

20	40	80
-70	100	60

Словари

Словари - это неупорядоченные коллекции пар "ключ-значение".

В качестве ключей могут использоваться ссылки на хешируемые объекты, а в качестве значений - ссылки на объекты любого типа.

Т.к. словари являются неупорядоченными коллекциями, то к ним не применяется понятие индекса элемента и не применяется операция извлечения среза.

Чтобы создать словарь можно использовать метод **dict()**:

```
>>> d = dict(short='dict', long='dictionary')
>>> print(d)
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> print(d)
{1: 1, 2: 4}
```

```
>>> d = {}
>>> print(d)
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> print(d)
{'dict': 1, 'dictionary': 2}
```

Еще один способ - использовать метод **fromkeys()**:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> print(d)
{'a': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> print(d)
{'a': 100, 'b': 100}
```

```
>>> d = {a: a ** 2 for a in range(7)}
>>> print(d)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Также можно использовать генератор словарей:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> print(d[1])
2
>>> d[4] = 4 ** 2
>>> print(d)
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d['1']
Traceback (most recent call last):
File "", line 1, in
    d['1']
KeyError: '1'
```

Методы для работы со словарями

clear() - словарь очищается;

```
>>> d = {'a': 1, 'b': 2}
>>> d.clear()
>>> print(d)
{}
```

copy() - копия словаря возвращается;

```
>>> d = {'a': 1, 'b': 2}
>>> b = d.copy()
>>> print(b)
{'a': 1, 'b': 2}
```

fromkeys(seq[,value]) - словарь создается с ключами из **seq** и значением **value;**

```
>>> d.fromkeys(['a', 'b'], 10)
{'a' : 10, 'b' : 10}
```

get(key[, default]) - значение ключа возвращается, или если его нет, то возвращается **default;**

```
>>> d = {'a': 1, 'b': 2}
>>> d.get('a')
1
```

items() - пары (ключ, значение) возвращаются;

```
>>> d = {'a': 1, 'b': 2}
>>> d.items()
dict_items([('a', 1), ('b', 2)])
```

keys() - ключи в словаре возвращаются;

```
>>> d = {'a': 1, 'b': 2}
>>> print(d.keys())
dict_keys(['a', 'b'])
```

pop(key[, default]) - ключ удаляется и значение возвращается, или если ключа нет, то возвращается **default;**

```
>>> d = {'a': 1, 'b': 2}
>>> d.pop('a')
1
>>> print(d)
{'b': 2}
```

popitem() - с конца удаляется и возвращается пара (ключ, значение);

```
>>> d = {'a': 1, 'b': 2}
>>> d.popitem()
('b', 2)
>>> print(d)
{'a': 1}
```

setdefault(key[, default]) - значение ключа возвращается, или если его нет, то создается ключ со значением **default;**

```
>>> d = {'a': 1, 'b': 2}
>>> d.setdefault('e', 6)
6
>>> d.setdefault('f')
>>> print(d)
{'a': 1, 'b': 2, 'e': 6, 'f': None}
```

update([other]) - добавляются пары (ключ, значение) из **other**, обновляя словарь, при это перезаписываются существующие ключи;

```
>>> d = {'a': 1, 'b': 2}
>>> d.update({'d':5})
>>> print(d)
{'a': 1, 'b': 2, 'd': 5}
```

values() - в словаре возвращаются значения.

```
>>> d = {'a': 1, 'b': 2}
>>> d.values()
dict_values([1, 2])
```

Функции

Функция в Python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции **def**.

```
def add(x, y):
    return x + y
>>> add(1, 10)
11
>>> add('abc', 'def')
'abcdef'
```

Параметр — это имя в списке параметров в первой строке определения функции. Он получает свое значение при вызове.

Аргумент — это реальное значение или ссылка на него, переданное функции при вызове.

Функции с произвольным числом элементов

```
>>> def func(a, b, c=2): # c - необязательный аргумент
...     return a + b + c
...
>>> func(1, 2)  # a = 1, b = 2, c = 2 (по умолчанию)
5
>>> func(1, 2, 3)  # a = 1, b = 2, c = 3
6
>>> func(a=1, b=3)  # a = 1, b = 3, c = 2
6
>>> func(a=3, c=6)  # a = 3, c = 6, b не определен
Traceback (most recent call last):
  File "", line 1, in
    func(a=3, c=6)
TypeError: func() takes at least 2 arguments (2 given)
```

Пример

```
def inplace(x, mutable=[]):
    mutable.append(x)
    return mutable
res = inplace(1)
res = inplace(2)
print(inplace(3))
[1, 2, 3]
```

```
def prime_numbers(x:int) -> (int, list):
  l=[]
  for i in range(x+1):
    if checkPrime(i):
      l.append(i)
  return len(l), l
```

```
def prime_numbers(x):
  l=[]
  for i in range(x+1):
    if checkPrime(i):
      l.append(i)
  return len(l), l

no_of_primes, primes_list = prime_numbers(100)
```