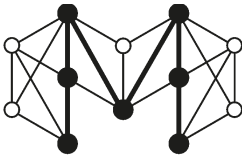
Программирование — 2026



Элементы программы на Python

Программа (или скрипт) представляет собой последовательность **определений** и **команд**

- Определения обрабатываются и команды выполняются в специальной **среде** (оболочка, *shell*)
- Они могут быть набраны прямо в оболочке, или сохранены в файле, который впоследствии может быть загружен в оболочку и выполнен

Команда (или **оператор**) – инструкция интерпретатору осуществить то или иное действие

Объекты

Суть работы программы – манипулирование с данными, обработка объектов, содержащих данные.

Каждый такой объект имеет важную характеристику – **тип**, определяющий, что за данные он содержит, и какие операции к нему применимы.

Основная классификация объектов:

- Скалярные (их нельзя разделить на составные части)
- Не скалярные (составные, имеющие внутреннюю структуру, к которой можно получить доступ)

Скалярные объекты

`int` – целые числа (5, -23, 1234567890)
`float` – действительные числа (3.14, 9.81, 1.2e-5)
`bool` – используется для представления двух логических значений: `True` (истина) и `False` (ложь)

Встроенная функция `type` служит для определения типа объекта

```
In [3]: type(3)
```

```
Out[3]: int
```

Выражения

Объекты и **знаки операций** объединяются в **выражения**, каждое из которых представляет собой объект определенного типа

Синтаксис простейшего выражения:
<**объект**> <**знак операции**> <**объект**>

Операции с числовыми объектами

`a + b` – сумма. Если оба числа – целые, то результат – объект `int`, в остальных случаях результат – `float`

`a - b` – разность

`a * b` – произведение

`a/b` – частное (в Python 3 всегда `float`)

`a//b` – результат деления нацело
`a % b` – остаток от деления

В двух предыдущих операциях, как правило, `a` и `b` – `int`, результат - `int`

`a**b` – возведение в степень, a^b.

Порядок действий:

- Сначала выражения в скобках
- Потом слева направо:

```
**
*, /
+, -
```

In [4]:
```
2**10//2
```

Out[4]:
```
512
```

Операции сравнения

```
a > b
    a >= b
    a < b
    a <= b
    a == b
    a != b
```

Результат сравнения - истина или ложь, `True` или `False`.

In [5]:
```
5 < 8
```

Out[5]:
```
True
```

Операции с объектами логического типа

```
a and b  (результат True, если оба аргумента истинны)
    a or b   (результат True, если хотя бы один аргумент истинен)
    not a    (противоположное значение)
```

Преобразование типов (type casting)

На основе объекта одного типа может быть создан объект другого типа. Для преобразования используется имя типа.

In [6]:
```
float(100)
```

Out[6]:
```
100.0
```

In [7]:
```
int(3.9)
```

Out[7]:
```
3
```

In [8]:
```
# Что получится?
bool(3.9)
```

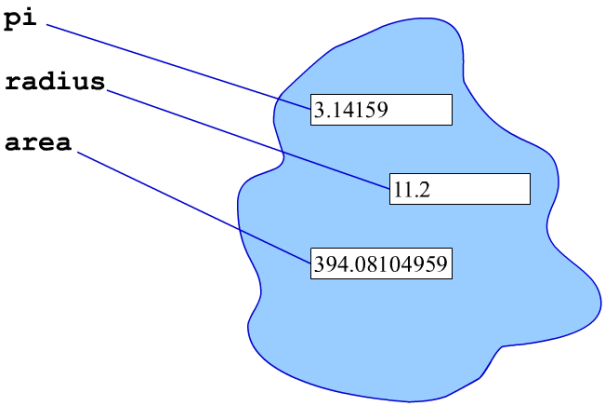
Out[8]:
```
True
```

Переменные и значения

In [9]:
```
pi = 3.14159
radius = 11.2
area = pi*(radius**2)
area
```

Out[9]:
```
394.0810495999999
```

Что происходит с памятью:



Сначала в памяти создается объект — результат вычислений, а потом он получает **имя**, позволяющее программе получать доступ к этому объекту.

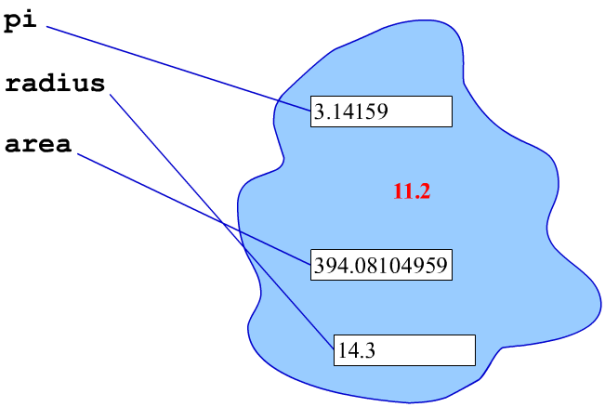
Важно: имя, или переменная, получает тип в момент присваивания выражению в правой части. Это — **динамическая типизация**.

Другой вариант (в других языках программирования): тип переменной определяется в программе в момент ее описания и не может быть изменен. Область памяти для хранения значения такой переменной тоже выделяется в момент ее описания (с некоторыми оговорками). Это — **статическая типизация**.

Изменение значения переменной

In [10]:
```
radius = 14.3
area
```

Out[10]:
```
394.0810495999999
```



Обратите внимание на оставшееся в памяти неиспользуемое значение 11.2. Оно "засоряет память", является "мусором". Его сборкой занимается специальная программа — сборщик мусора (garbage collector).

!!! Про допустимые и недопустимые имена переменных поговорим чуть позже

Пример нескалярного объекта: строки

In [11]:
```
'abc'
"abc"
'''Привет!
В этой строке -
сразу три строчки'''
'123'
```

Out[11]:
```
'123'
```

Некоторые операции

(будем подробно изучать позже)

In [12]:
```
type('123')
```

Out[12]:
```
str
```

In [13]:
```
'мех' + 'мат'
```

Out[13]:
```
'мехмат'
```

In [14]:
```
'A'*3
```

Out[14]:
```
'AAA'
```

In [15]:
```
str(100)
```

Out[15]:
```
'100'
```

In [16]:
```
int('100')
```

Out[16]:
```
100
```

Оператор вывода

Команды могут выполняться в shell, но **программой** будем называть текст на Питон, сохраненный в файле.

В случае выполнения команды в shell результат выполнения сразу выводится на экран. В случае программы требуется специальный оператор: оператор вывода

In [17]:
```
# Примеры
print('hello')
print(156*12)
print("I'll", "be", "back!")
print('''Привет!
В этой строке -
сразу три строчки''')
```

```
hello
1872
I'll be back!
Привет!
В этой строке -
сразу три строчки
```

Общий вид

```
print(a, b, ..., end='\n', sep=' ')
```

Ввод данных

- Простейший способ взаимодействия программы с пользователем: запрос ввода каких-то данных
- Такой ввод осуществляется в Python с помощью функции `input()`
- Ее результат имеет тип `str`, поэтому необходимо преобразование типа

In [ ]:
```
name = input('Как Вас зовут?  ')
age = int(input('Сколько Вам лет? '))
weight = float(input('Каков Ваш вес в кг? '))
```

In [ ]:
```
age
```

Перед тем, как начать программировать...

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

Дзен Питона

- 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 — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имён — отличная штука! Будем делать их больше!

Стиль кода:

PEP8 - стиль кода в языке Python

PEP 8 -- Style Guide for Python Code!

Правила именования переменных

- Имена переменных, функций, методов:

`lower_case_with_underscores`

- Константы:

`ALL_CAPS`

- Имена Классов:

`CapitalizeWords` (он же `StudlyCaps`)

- В крайнем случае (для совместимости с уже имеющимся кодом):

`mixedCase` (он же `camelCase`)

Имена, которых следует избегать

Никогда не используйте символы
`l` (маленькая латинская буква «эль»),
`O` (заглавная латинская буква «о») или
`I` (заглавная латинская буква «ай») как однобуквенные идентификаторы.

В некоторых шрифтах эти символы неотличимы от цифры один и нуля (и символа вертикальной палочки).

Если очень-очень нужно использовать имя переменной из одной буквы `l`, пишите вместо неё заглавную `L`.

Имена переменных ни в коем случае не должны совпадать с ключевыми словами языка

In [19]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'cla
ss', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from
', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pas
s', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

In [ ]:
```

```