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]:
```
10**100//2
```

Out[4]:
```
500000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

- Исключение: возведение в степень выполняется справа налево:

`2**3**2` – это 2^{3^2}.

In [5]:
```
2**3**2
```

Out[5]:
```
512
```

In [6]:
```
(2**3)**2
```

Out[6]:
```
64
```

Операции сравнения

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

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

In [10]:
```
5 == 8
```

Out[10]:
```
False
```

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 [11]:
```
float(100)
```

Out[11]:
```
100.0
```

In [12]:
```
int(3.9)
```

Out[12]:
```
3
```

In [14]:
```
# Что получится?
bool(0)
```

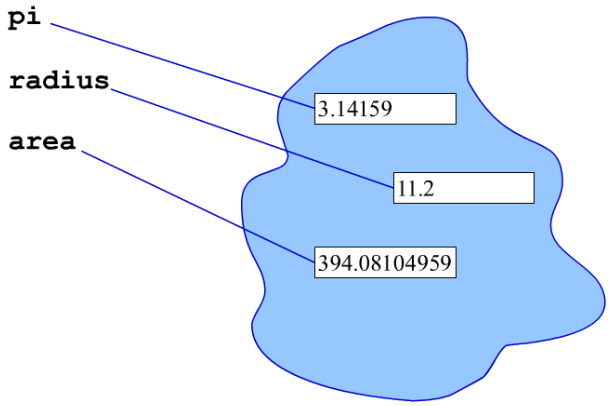
Out[14]:
```
False
```

Переменные и значения

In [15]:
```
pi = 3.14159
radius = 11.2
area = pi*(radius**2)
area
```

Out[15]:
```
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.

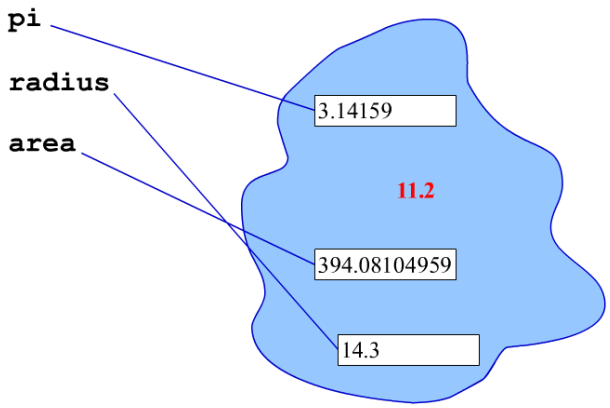
Важно: имя, или переменная, получает тип в момент присваивания выражению в правой части. Это — **динамическая типизация**.

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 [16]:
```
radius = 14.3
area
```

Out[16]:
```
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 [17]:
```
'abc'
"abc"
'''Привет!
В этой строке -
сразу три строчки'''
'123'
```

Out[17]: '123'

Некоторые операции

(будем подробно изучать позже)

In [18]:
```
type('123')
```

Out[18]: str

In [19]:
```
'мех' + 'мат'
```

Out[19]: 'мехмат'

In [20]:
```
'A'*3
```

Out[20]: 'AAA'

In [21]:
```
str(100)
```

Out[21]: '100'

In [22]:
```
int('100')
```

Out[22]: 100

Оператор вывода

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

В виде исключения в дальнейшем программой часто будем называть и код, содержащийся в одной ячейке среды Jupyter Notebook. Обычно такой код должен удовлетворять дополнительным правилам: в нем должно быть несколько действий, в том числе получение каких-то данных от пользователя и вывод результатов расчетов.

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

In [23]:
```
# 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 [24]:
```
name = input('Как Вас зовут?  ')
age = int(input('Сколько Вам лет? '))
weight = float(input('Каков Ваш вес в кг? '))
```

```
Как Вас зовут?  М. И.
Сколько Вам лет? 61
Каков Ваш вес в кг? 91.5
```

In [25]:
```
print(age)
```

```
61
```

Перед тем, как начать программировать...

In [26]:
```
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!
```

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 (https://pep8.ru/doc/pep8/)](https://pep8.ru/doc/pep8/)

PEP 8 -- Style Guide for Python Code! (https://www.python.org/dev/peps/pep-0008/)

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»),
`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 [27]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'brea
k', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finall
y', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonloc
al', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yiel
d']
```

In [ ]: