Практикум по программированию на языке Python

Занятие 1: введение в язык программирования Python

Мурат Апишев (mel-lain@yandex.ru)

Москва, 2019

Зачем изучать Python?

Несколько смежных вопросов:

- зачем вы сюда находитесь?

Зачем изучать Python?

Несколько смежных вопросов:

- зачем вы сюда находитесь?
- зачем уметь программировать?

Зачем изучать Python?

Несколько смежных вопросов:

- зачем вы сюда находитесь?
- зачем уметь программировать?
- зачем уметь программировать хорошо?

Зачем изучать Python?

Несколько смежных вопросов:

- зачем вы сюда находитесь?
- зачем уметь программировать?
- зачем уметь программировать хорошо?
- зачем учиться программировать хорошо на Python?

Python - хороший язык, потому что он

- *читабельный и согласованный* - его легко понимать и сопровождать
- *высокоуровневый* - код лаконичнее, чем у большинства других языков
- *универсальный* - вести успешную разработку на Python можно почти в любой IT-области
- *мультипарадигмальный* - процедурный, объектно-ориентированный, функциональный
- *кроссплатформенный* - легко разрабатывать и запускать код под любой популярной ОС
- *легко интегрируемый* - активно используется в связке с другими языками
- *прост в изучении* - сложно найти язык с более низким порогом входа
- *богатый и активно развивающийся* - open-source + большое сообщество способны на многое

Чему мы посвятим курс

- Базовая часть курса будет состоять из 7 примерных тем (в идеале - занятий)

    - *Введение в язык программирования Python*
    - *Модель памяти, встроенные типы данных*
    - *Итераторы, встроенные функции. Пользовательские функции, генераторы*
    - *Основы ООП, особенности ООП в Python*
    - *Продвинутое использование ООП, проектирование кода*
    - *Методы эффективного представления, обработки и анализа данных, введение в визуализацию*
    - *Введение в Python для задач машинного обучения*

- Если у лектора будет время, а у студентов - желание, то есть ещё 3 более продвинутые темы:

    - *Профессиональная разработка на языке Python*
    - *Написание эффективного кода с помощью Python*
    - *Несколько полезных вещей напоследок*

Правила курса

- Курс будет состоять из 7-10 примерных тем (в идеале - занятий)
- Каждые 2 недели будет выдаваться домашнее задание (срок сдачи - до следующего задания), всего 4 или 5 заданий
- Списывать и пропускать дедлайны нельзя
- Зачёт выставляется по результатам выполнения заданий

Где писать код на Python

- в любом удобном текстовом редакторе
- в Jupyter Notebook
- в специализированной IDE (например, PyCharm)

Обычно файлы с кодом на Python имеют расширение `.py`

Где исполнять код на Python

- Python - скриптовый язык, исполняемый интерпретатором
- Стандартная версия интепретатора - CPython, будем пользоваться ей
- Можно установить интерпретатор + стандартные библиотеки языка (https://www.python.org (https://www.python.org))
- Удобнее поставить распространённый дистрибутив Anaconda (https://www.anaconda.com/download (https://www.anaconda.com/download))
- В комплекте идут дополнительно популярные библиотеки и Jupyter Notebook

При установке обратите внимание, что в курсе мы ориентируемся на Python версии 3.7

Как можно запускать код на Python

- Запускаем интерпретатор в терминале и напрямую пишем команды
- Пишем код в файле и запускаем командой `python <имя файла>`
- Создаём файл Jupyter Notebook и запускаем код в его ячейках

Классическая первая программа

In [4]:
```
print('Hello world!')
```

```
Hello world!
```

- не требуется специфических объявлений точки входа программы (main в C++) - код запускается с первой строки файла/ячейки
- `print` - стандартная функция языка, печатающая аргументы в стандартный поток вывода

Операторы присваивания

In [218]:
```
a = 5
print(a)
```

```
5
```

In [217]:
```
a, b, c = 5, 6, 7
print(a, b, c)
```

```
5 6 7
```

Ввод с клавиатуры

In [195]:
```
f_name = input('Enter your first name:')
l_name = input('Enter your last name:')
print(f_name, l_name)
```

```
Enter your first name:Murat
Enter your last name:Apishev
Murat Apishev
```

Объекты и переменные

In [219]:
```
4
```

Out[219]: 4

- создан объект типа `int`, который содержит значение 4
- поскольку на этот объект нет ссылок, он уничтожается после создания и печати

In [206]:
```
a = 5
```

- создан объект типа `int`, в котором содержится значение 5
- затем создана переменная с именем a, которая ссылается на этот объект

In [220]:
```
del a
```

- переменная a принудительно удалена
- объект, на который она ссылалась, тоже был удалён, поскольку больше на него нет ссылок

Объекты и переменные

In [207]:
```
a = 5
b = a
```

- создан объект типа `int`, в котором содержится значение 5
- затем создана переменная с именем a, которая ссылается на этот объект
- создана переменная b, которая указывает на тот же объект типа `int`

In [208]:
```
a = a + 1  # equals to a += 1
print(b)
```

```
5
```

- выражение `a = a + 1` создаёт новый объект типа `int`, равный 6
- переменная a начинает ссылаться на него и больше не связана с объектом, равным 5
- b продолжает ссылаться на него

Основные встроенные типы данных

- *числовые*, *логический*, *None*
- *строковые*
- *контейнерные*: *списки*, *словари*, *кортежи*, *множества*
- *файловый*
- *прочие*: *функции*, *классы*, *модули*, *сами типы*

Python -

- динамически типизируемый (сам следит за выводом типов)
- язык со строгой типизацией (объект заданного типа допускает только связанные с ним операции)

Тип объекта можно получить с помощью функции type

Целые числа

In [6]:
```
2 + 5
```

Out[6]: 7

In [7]:
```
4 // 3
```

Out[7]: 1

In [8]:
```
4 % 3
```

Out[8]: 1

In [16]:
```
2 ** 100
```

Out[16]: 1267650600228229401496703205376

In [5]:
```
type(1)
```

Out[5]: int

Числа с плавающей точкой

In [17]:
```
1.0 + 2
```

Out[17]:
```
3.0
```

In [18]:
```
4 / 3
```

Out[18]:
```
1.3333333333333333
```

In [19]:
```
3 ** 2.5
```

Out[19]:
```
15.588457268119896
```

In [22]:
```
type(1.0)
```

Out[22]:
```
float
```

In [33]:
```
type(float())
```

Out[33]:
```
float
```

Логический тип

In [25]:
```
True or False
```

Out[25]: True

In [26]:
```
True or destroy_the_world()
```

Out[26]: True

In [27]:
```
True and (False or True)
```

Out[27]: True

In [34]:
```
bool()
```

Out[34]: False

In [28]:
```
type(True)
```

Out[28]: bool

Тип None

In [35]:
```
None == None
```

Out[35]:
```
True
```

In [37]:
```
None or True
```

Out[37]:
```
True
```

In [39]:
```
type(None)
```

Out[39]:
```
NoneType
```

In [40]:
```
5 * None  # strong typing
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-8d0e337763fe> in <module>
----> 1 5 * None

TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'
```

Строки

In [43]:
```
s = 'qwerty'
type(s)
```

Out[43]:
```
str
```

In [45]:
```
len(s)  # built-in function to get sequence length
```

Out[45]:
```
6
```

In [46]:
```
s[0]  # make a copy of element
```

Out[46]:
```
'q'
```

In [50]:
```
s[-1]  # make a copy of element
```

Out[50]:
```
'y'
```

In [52]:
```
s[0: 3]  # make a copy of elements from 0 to 3 excluding
```

Out[52]:
```
'qwe'
```

In [47]:
```
s + '-add-on'
```

Out[47]:
```
'qwerty-add-on'
```

Строки

In [48]:
```
s + 5  # strong typing
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-bdd2275bd5ba> in <module>
----> 1 s + 5

TypeError: can only concatenate str (not "int") to str
```

In [49]:
```
s * 5  # strong typing
```

Out[49]:
```
'qwertyqwertyqwertyqwertyqwerty'
```

In [59]:
```
s[0] = 'a'  # str type is immutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-59-99e5ad368a97> in <module>
----> 1 s[0] = 'a'  # str type is immutable

TypeError: 'str' object does not support item assignment
```

Строка как набор байтов

In [65]:
```
b = bytearray(b'qwerty')
type(b)
```

Out[65]:
```
bytearray
```

In [73]:
```
b[1] = ord('a')
b
```

Out[73]:
```
bytearray(b'qaerty')
```

- `bytearray` - массив байтов, в который можно записывать номера 8-битных символов
- в отличие от `str`, объекты этого типа являются изменяемыми
- функция `ord` возвращает по символу его номер в таблице ASCII

Методы строк

In [83]:
```
s = 'Qwe{}'.format('rty')
s
```

Out[83]:
```
'Qwerty'
```

In [84]:
```
print(s.find('w'), s.find('9'))
```

```
1 -1
```

In [78]:
```
s.split('r')
```

Out[78]:
```
['Qwe', 'ty']
```

In [79]:
```
'-'.join(['New', 'York'])
```

Out[79]:
```
'New-York'
```

In [80]:
```
s.replace('ty', 'TY')
```

Out[80]:
```
'QwerTY'
```

In [82]:
```
s[0].isupper()
```

Out[82]:
```
True
```

Как ещё можно задавать строки

In [91]:
```
'qwe\'rty'
```

Out[91]:
```
"qwe'rty"
```

In [90]:
```
"qwe'rty"
```

Out[90]:
```
"qwe'rty"
```

In [89]:
```
'''qwe'rty'''
```

Out[89]:
```
"qwe'rty"
```

In [92]:
```
'qwe\0rty'  # print hex code in '\xNN' format for non-printable characters
```

Out[92]:
```
'qwe\x00rty'
```

In [97]:
```
'qwe\nrty'
```

Out[97]:
```
'qwe\nrty'
```

In [99]:
```
print('qwe\nrty')  # we'll discuss difference between __repr__ and __str__ further
```

```
qwe
rty
```

Контейнерные типы: списки

In [102]:
```
lst = [1, None, 'qwerty']
list([1, None, 'qwerty']) == lst  # list() gets any iterable object as input
```

Out[102]:
```
True
```

In [103]:
```
lst[0] = [10, 20]
lst
```

Out[103]:
```
[[10, 20], None, 'qwerty']
```

- список - более общий аналог массива (не путать со структурой данных "список")
- элементы могут быть произвольного типа
- сам список является изменяемым типом

Списки

In [104]:
```
len(lst)
```

Out[104]:
```
3
```

In [113]:
```
type(lst)
```

Out[113]:
```
list
```

In [107]:
```
lst[-1]
```

Out[107]:
```
'qwerty'
```

In [109]:
```
lst[: 100]  # slice bounds can be exceed real ones
```

Out[109]:
```
[[10, 20], None, 'qwerty']
```

In [125]:
```
lst[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-125-843c5cb805fc> in <module>
----> 1 lst[5]

IndexError: list index out of range
```

Методы списков

In [118]:
```
lst = [1, 2, 3]
lst.append(4)
lst
```

Out[118]: [1, 2, 3, 4]

In [119]:
```
print(lst.pop(), lst)
```

```
4 [1, 2, 3]
```

In [122]:
```
lst.sort(reverse=True)  # equals to lst.reverse()
lst
```

Out[122]: [3, 2, 1]

In [123]:
```
lst * 2
```

Out[123]: [3, 2, 1, 3, 2, 1]

In [124]:
```
lst + list('abc')
```

Out[124]: [3, 2, 1, 'a', 'b', 'c']

Списковые включения и генераторы списков

In [126]:
```
[x ** 2 for x in [1, 2, 3]]  # list comprehension
```

Out[126]:
```
[1, 4, 9]
```

In [138]:
```
lst = [x ** 2 for x in [1, 2, 3] if x > 1]
print(type(lst))

for _ in [1, 2]:  # _ means value we really don't need
    for e in lst:
        print(e)
```

```
<class 'list'>
4
9
4
9
```

In [136]:
```
lst = (x ** 2 for x in [1, 2, 3] if x > 1)  # list generator
print(type(lst))

for _ in [1, 2]:
    for e in lst:
        print(e)
```

```
<class 'generator'>
4
9
```

Снова про объекты и переменные

In [227]:
```
a = [2]
b = a
a += [1]
print(b)
```

```
[2, 1]
```

- список - изменяемый объект, создание новой ссылки не приводит к появлению нового объекта
- есть две переменные, указывающие на один и тот же список

In [228]:
```
del a
print(b)
```

```
[2, 1]
```

- удаление a не приводит у удалению списка, поскольку на него ещё ссылается b

Контейнерные типы: словари

- словарь (`dict`) - это ассоциативный массив (отображение), т.е. набор пар "ключ-значение"
- словарь поддерживает возможность быстро искать значение по ключу
- в Python словари реализованы на основе хэш-таблиц (сложность поиска O(1) в среднем)
- словари также являются изменяемым типом данных
- значения могут иметь произвольный тип
- ключи - произвольный неизменяемый тип

Словари

In [152]:
```
type(dict()), type({})
```

Out[152]:
```
(dict, dict)
```

In [158]:
```
# simple creation of nested structures!
d = {'1': 1, 's': 2, None: [1, {None: True}]}
d
```

Out[158]:
```
{'1': 1, 's': 2, None: [1, {None: True}]}
```

In [154]:
```
d['s'] = 'two'
d
```

Out[154]:
```
{'1': 1, 's': 'two', None: 5}
```

In [155]:
```
del d['1']  # remove object
print(d)
```

```
{'s': 'two', None: 5}
```

In [156]:
```
d2 = {('s', 3), ('a', 'b')}
```

In [157]:
```
d.update(d2)
d
```

Out[157]:
```
{'s': 3, None: 5, 'a': 'b'}
```

Контейнерные типы: кортежи

- очень похожи на списки, но являются неизменяемыми

In [159]:
```
type(()), type(tuple())
```

Out[159]:
```
(tuple, tuple)
```

In [160]:
```
(1, 2, 'str', 4.0, None)
```

Out[160]:
```
(1, 2, 'str', 4.0, None)
```

In [161]:
```
(1, 2) + (3.0, 6, None)
```

Out[161]:
```
(1, 2, 3.0, 6, None)
```

In [163]:
```
tpl = tuple([1, 2, 3])
tpl[2]
```

Out[163]:
```
3
```

Контейнерные типы: множества

- Как и словари, основаны на хэш-таблицах
- Элементы должны быть неизменяемыми

In [165]:
```
type(set()), type({1}), type({})  # Empty '{}' is a dict, not a set!
```

Out[165]:
```
(set, set, dict)
```

In [166]:
```
{1, 2, 3, 3, 2, 1, 'set', 'a'}
```

Out[166]:
```
{1, 2, 3, 'a', 'set'}
```

In [168]:
```
s = {1, 2, 3}
s.add(4)
s
```

Out[168]:
```
{1, 2, 3, 4}
```

In [170]:
```
s1, s2 = {1, 2, 3}, set([3, 4, 5])

print('{}\t{}\t{}\t{}'.format(s1 - s2, s1 | s2, s1 & s2, s1 < s2))
```

```
{1, 2}  {1, 2, 3, 4, 5} {3}     False
```

Файловый тип

In [191]:
```
fin = open('some_file.txt', 'r')  # 'r' can be avoided
print(fin.read())
fin.close()
fin.closed
```

```
hello
world!
```

Out[191]: True

In [192]:
```
fin = open('some_file.txt', 'r')  # 'r' can be avoided

# strip removes blank and special characters from edges
lines = [line.strip() for line in fin.readlines()]

print(' '.join(lines))
fin.close()
fin.closed
```

```
hello world!
```

Out[192]: True

Файловый менеджер контекста

In [177]:
```
fin = open('some_file.txt', 'r')  # 'r' can be avoided
print(fin.read(), 0/0)
fin.close()
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-177-4fbd946c8ecb> in <module>
      1 fin = open('some_file.txt', 'r')  # 'r' can be avoided
----> 2 print(fin.read(), 0/0)
      3 fin.close()

ZeroDivisionError: division by zero
```

In [178]:
```
fin.closed
```

Out[178]:
```
False
```

Файловый менеджер контекста

In [183]:
```
with open('some_file.txt', 'r') as fin:
    print(fin.read(), 0/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-183-0f73f85c4f29> in <module>
      1 with open('some_file.txt', 'r') as fin:
----> 2     print(fin.read(), 0/0)

ZeroDivisionError: division by zero
```

In [184]:
```
fin.closed
```

Out[184]:
```
True
```

Запись в файл

- отступы используются для обозначения операторных блоков (вместо фигурных скобок в C++)

In [186]:
```
with open('some_file.txt', 'a') as fout:
    fout.write('additional string')
```

In [187]:
```
with open('some_file.txt', 'r') as fin:
    print(fin.read())
```

```
hello
world!
additional string
```

In [188]:
```
with open('some_file.txt', 'w') as fout:
    fout.write('new\ncontent')
```

In [189]:
```
with open('some_file.txt', 'r') as fin:
    print(fin.read())
```

```
new
content
```

Условный оператор if

In [229]:
```
if 3 % 3 == 0:
    print('success')
```

```
success
```

In [230]:
```
a = 5
if a == 2:
    print(2)
elif a == 4:
    print(4)
else:
    print(a)
```

```
5
```

In [231]:
```
print(a if a > 5 else 2.5)  # ternary operator
```

```
2.5
```

Оператор цикла while

In [239]:
```
a = 5
while a > 4:
    print('Current a: {}'.format(a))
    a -= 1
```

```
Current a: 5
```

In [ ]:
```
while True:  # infinity loop
    pass  # in Python 3 the equal statement is '...'
```

In [241]:
```
a = 6
while a > 2:
    a -= 1
    if a == 5: continue
    if a == 3: break
    print(a)
else:
    print('else')
```

```
4
```

- `pass` - оператор пустого действия
- `continue` - досрочный переход на следующую итерацию цикла
- `break` - досрочный выход из цикла
- `else` - обозначает секцию, которая выполняется, если цикл был завершён не по `break`

Оператор цикла for

- в отличие от C/C++, в Python чаще всего используются range-based for-циклы
- классические циклы по индексам так же доступны, но при прочих равных менее предпочтительны

In [243]:
```
lst = ['a', 'b', 'c']
for element in lst:  # range-based for
    print(element)
```

```
a
b
c
```

In [244]:
```
for index in range(len(lst)):  # index-based for
    print(lst[index])
```

```
a
b
c
```

- функция `range` создаёт генератор, позволяющий итерироваться по последовательности чисел с заданными границами и шагом

Оператор цикла for

In [246]:
```
for x, y in [(1, 'a'), [2, 'b']]:
    print(x, y)
else:
    print('For-loop ended without break!')
```

```
1 a
2 b
For-loop ended without break!
```

In [254]:
```
for x, y, z in [[1, 2, 3], ['a', 'b', 'c'], [True, False, None]]:
    if isinstance(x, int):
        continue
    print(x, y, z)
```

```
a b c
```

- `isinstance` - функция языка, позволяющая проверить принадлженость объекта типу (учитывает иерархию наследования типов)

Как правильно называть переменные

- `regular_variable`
- `ClassName`
- `SOME_CONSTANT_10`
- `_private_class_field_or_method`
- `__very_private_class_field_or_method`
- `__magic_method__` or `__special_field__`

Не используйте переменные с unicode-символами:

- `плохое_имя_для_переменной`

Импорт модулей

- любой файл на Python представляет собой модуль
- импорт модуля вызовет выполнение содержащегося в нём кода

In [268]:
```
with open('test_module.py', 'w') as fout:
    fout.write('print(10)\n')
    fout.write('def new_print(a):\n\tprint(a)\n')
```

- создали текстовый файл и записали в него оператор печати и определение функции
- ключевое слово `def` используется для определения функций и классов (не единственный способ)

In [274]:
```
import test_module
```

```
10
```

- импортируем модуль, операторы выполнятся

Пространства имён модулей

In [270]:
```
new_print(5)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-270-2209d66c60e3> in <module>
----> 1 new_print(5)

NameError: name 'new_print' is not defined
```

In [277]:
```
test_module.new_print(5)
```

```
5
```

In [279]:
```
from test_module import new_print
new_print(5)
```

```
5
```

Повторная загрузка модуля

- если модуль был один раз загружен (исполнен), второй импорт ничего не произведёт
- это верно даже в том случае, если модуль был изменён после импорта

In [280]:
```
import test_module  # no any output!
```

- но модуль можно перезагрузить принудительно

In [282]:
```
from importlib import reload
reload(test_module)
```

```
10
```

Out[282]:
```
<module 'test_module' from '/home/mel-lain/mipt-python/tmp/test_module.py'>
```

Выполнение кода на Python в виде строки

In [283]:
```
eval('123 + 45')
```

Out[283]:
```
168
```

In [293]:
```
exec(
'''
a = 12
b = 3
print(a ** b)
''')
print(a, b)
```

```
1728
12 3
```

- `eval` позволяет вычислить выражение и возвращает его результат (чистая функция)
- `exec` позволяет запустить любой фрагмент кода на python (пример: ячейки Jupyter Notebook)

Спасибо за внимание!