Python-00

Intro

Sergey M. Kabanov

National Research University Higher School of Economics

September 9, 2018

Course overview

- Module #1:
  - Object-Oriented Programming (OOP) in Python 3
  - Related problems with data analysis: primary data processing, web scraping
  - 10 seminars, 10 labs, 1 final test, 1 final project
- Module #2:
  - Intro to ML. MI algorithms. Text classification
  - 9 sems, 4 labs, 1 kaggle InClass competition
- Module #4:
  - Text classification and clustering
  - 11 sems, 4 labs, 1 kaggle InClass competition

Score system

- 10 labs, each 10 points. One time per week. Sometimes with bonus points. Lab is a mini-project or set of problems
- 1 final test (exam in class), 50 points
- 1 home project, 40 points
- Individual task (optional), 20 points
- Points-Score map (210+ points in total):

Excellent (10) – 150+ points, (9) – 140+ points, (8) – 130+ points
Good (7) – 110+ points, (6) – 100+ points
Fair (5) – 80+ points, (4) – 60+ points
Poor less than 60 points

Python overview

- Interpreted high-level programming language for general-purpose programming
- Dynamic typing (duck typing)
- Supports OO, imperative, functional, procedural paradigms
- CPython is portable reference implementation of Python (Python Software Foundation License)
- Has a large and comprehensive standard library
- Has his own styleguide (PEP8)
- Good documented language

Python story

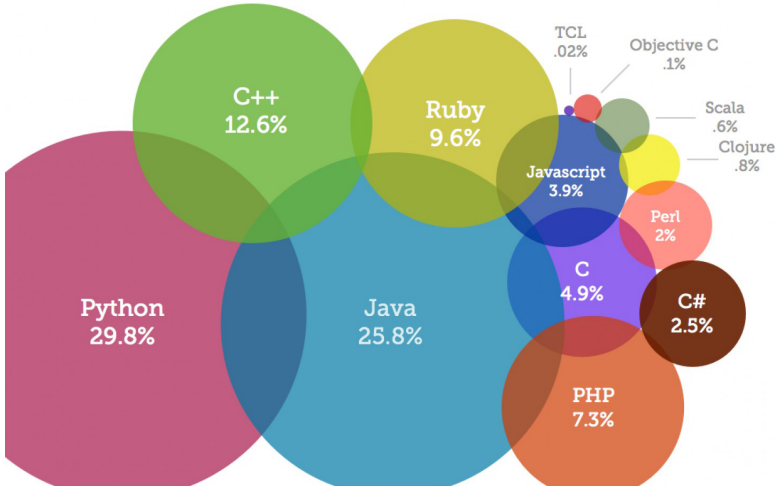
- Guido vam Rossum AKA Benevolent Dictator For Life:
  - 1994 – Python1.0
  - 2000 – Python2.0
  - 2008 – Python3.0
- Python2.x (supported until 2020)
  - 2.6
  - 2.7
- Python3.x (incompatible with 2.x)
  - 3.5
  - 3.6
  - 3.7 (latest stable release, June 2018)

Guido vam Rossum

Where and why Python?

- Scientific and Numeric Programming: Python vs. MATLAB, Python vs. R
- Web and Internet Development: Python vs. Perl, Python vs. JavaScript
- Education purposes: Python vs. C
- Python == bash
- ⊕ Fast development, easy debugging
- ⊕ Easy to start, well documented, big community
- ⊖ 5-100 times slower than compile languages
- ⊖ Exist two incompatible implementations

Job Rankings 2018



The Zen of Python

```
>>> 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.
```

Arithmetic

```
>>> 3 * 10 - 3 ** 2
21
>>> 100 ** 0.5
10.0
>>> 0.1 / 0.5
0.2
>>> 2e+01 - 1e+01
10.0
>>> 11 // 2
5
```

```
>>> from math import *
>>> cos(pi / 4)
0.7071067811865476
>>> floor(1.23)
1
>>> tan(pi / 4)
0.9999999999999999
>>> factorial(4)
24
>>> log10(1.e+20)
20.0
```

```
>>> e = 123
>>> e
123
>>> type(e)
<class 'int'>
>>> from math import *
>>> e
2.718281828459045
>>> type(e)
<class 'float'>
```

```
>>> e = 123
>>> e
123
>>> type(e)
<class 'int'>
>>> import math
>>> math.e
2.718281828459045
>>> type(math.e)
<class 'float'>
>>> e
123
```

Swap

```
>>> x = 123
>>> y = 'abc'
>>> print(x, y, sep=' ')
123 abc
>>> x, y = y, x
>>> print(x, y, sep=', ')
abc, 123
```

```
>>> z = 0.123
>>> type(z)
<class 'float'>
>>> x, y, z = z, x, y
>>> print(x, y, z, sep=' ')
0.123 abc 123
>>> z
123
>>> type(z)
<class 'int'>
```

Logic

```
>>> type(False)
<class 'bool'>
>>> 2 * 3 != 6
False
>>> 1 + 1 == 1 and 2 != 3
False
>>> 1 == 1 or 2 != 2
True
>>> x = True; y = False
>>> x or y and bool(11)
True
```

```
>>> x = 1
>>> 0 < x < 2
True
>>> 2 > x > 0
True
>>> y = 2
>>> 0 < x < y <= 2
True
>>> 0 < x < y < 2
False
```

Basic data types

```
>>> int_ = 1
>>> float_ = 2.123
>>> str_ = 'hello, world!'
>>> list_ = [1, 2, 3]
>>> tuple_ = (1, 2, 3)
>>> dict_ = {'a': 1, 'b': 2}
>>> set_ = {1, 2, 3, 3}
```

Output

```
>>> for i, j in [(1, 2), (3, 4)]:
...     print(i, j, sep=', ', end='\n')
...
1, 2
3, 4
>>> for i, j in [(1, 2), (3, 4)]:
...     print(i, j, sep=' ', end=' ')
...
1 2 3 4
```

Output

```
>>> lingvo = 'LingvoX\nElon Mask\n'
>>> with open('lingvo.txt', 'w') as f:
...     print(lingvo, file=f)
...
>>> letters = "a b c d"
>>> with open('letters.txt', 'w') as f:
...     for l in letters.split():
...         print(l, file=f)
...
```

Input

```
>>> with open('letters.txt', 'r') as f:
...     for i, line in enumerate(f.readlines()):
...         print(i, line)
...
0 a

1 b

2 c

3 d
```

If-statements

```
>>> x = 1
>>> if x in (3, 4):
...     print('if')
... elif x in (2, 5):
...     print('elif')
... else:
...     print('else')
...
else
```

```
>>> if x < 0:
...     sign = -1
... elif x == 0:
...     sign = 0
... else:
...     sign = 1
...
>>> sign
1
```

Cycles

```
>>> words = ['cat', 'dog', 'python']
>>> for word in words:
...     print(word, len(word))
...
cat 3
dog 3
python 6
```

Cycles

```
>>> x = 4
>>> while x > 0:
...     x -= 1
...     print("Current value:", x)
...
Current value: 3
Current value: 2
Current value: 1
Current value: 0
```

Cycles

```
>>> x = 4
>>> while True:
...     x -= 1
...     if x < 0:
...         break
...     print("Current value:", x)
...
Current value: 3
Current value: 2
Current value: 1
Current value: 0
```

Slicing

```
>>> x = [0, 1, 2, 3]
>>> x[0]
0
>>> x[2:]
[2, 3]
>>> x[1:4]
[1, 2, 3]
>>> x[-1]
3
```

```
>>> x[::-1]
[3, 2, 1, 0]
>>> x[0] = 'a'
>>> x
['a', 1, 2, 3]
>>> x[-2]
2
>>> x[::-2]
[3, 1]
```

Slicing

```
>>> y = '012345'
>>> y[0]
'0'
>>> y[2:]
'2345'
>>> y[1:4]
'123'
>>> y[-1]
'5'
```

```
>>> y[::-1]
'543210'
>>> y[::-2]
'531'
>>> y[1::-1]
'10'
>>> y[1::2]
'135'
```

Slicing

```
>>> y[0] = 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> y
'012345'
```

Functions

```
>>> def plus(x, y):
...     return x + y
...
>>> plus(100, 2)
102
```

```
>>> def square(x):
...     return x ** 2
...
>>> square(25)
625
```

```
>>> def printer(x):
...     print(x)
...
>>> print('LingvoX')
LingvoX
```

```
>>> def minus_one(x):
...     return x - 1
...
>>> minus_one(11)
10
```

Functions

```
>>> printer = lambda x: print(x)
>>> printer('abc')
abc
>>> add = lambda x, y: x + y
>>> add(13, 12)
25
>>> sign = lambda x: 1 if x >= 0 else -1
>>> sign(-123)
-1
>>> sign(555)
1
```

Built-in

```
>>> x = [1, 2, 3, 4]
>>> len(x)
4
>>> min(x)
1
>>> max(x)
4
>>> sum(x)
10
>>> abs(-3)
3
```

```
>>> ascii('ușor')
"'u\\u0219or'"
>>> chr(65)
'A'
>>> bin(7)
'0b111'
>>> bool(0)
False
>>> int('123')
123
>>> float('123')
123.0
```

Exceptions

```
>>> 'a' + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Exceptions

```
>>> raise ValueError('Error!')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Error!
>>> d = {'a': 1, 'b': 2}
>>> d['c']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'c'
```

Exceptions

```
>>> a = 33 + 1 +
  File "<stdin>", line 1
    a = 33 + 1 +
                 ^
SyntaxError: invalid syntax
>>> for i in 1:
...     pass
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Generators

```
>>> for i in range(2):
...     print(i, end=' ')
...
0 1
>>> x = ['a', 'b', 'c', 'd']
>>> for i, j in zip(range(3), x):
...     print(i, j)
...
0 a
1 b
2 c
```

Generators

```
>>> for i, value in enumerate(x):
...     print(i, value)
...
0 a
1 b
2 c
3 d
```

List comprehension

```
>>> squares = [i * i for i in range(7)]
>>> squares
[0, 1, 4, 9, 16, 25, 36]
>>> {k: v for k, v in enumerate(squares)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
>>> r = (i for i in range(7))
>>> r
<generator object <genexpr> at 0x7fdfe5922570>
>>> for i in r: print(i, end=' ')
...
0 1 2 3 4 5 6
```

List

```
>>> l = [0, 2, 1, 3, 4]
>>> l.append(5)
>>> l
[0, 2, 1, 3, 4, 5]
>>> sorted(l)
[0, 1, 2, 3, 4, 5]
>>> l
[0, 2, 1, 3, 4, 5]
>>> l.sort()
>>> l
[0, 1, 2, 3, 4, 5]
```

```
>>> del l[0]
>>> l
[1, 2, 3, 4, 5]
>>> l.reverse()
>>> l
[5, 4, 3, 2, 1]
>>> l.remove(5)
>>> l
[4, 3, 2, 1]
```

String

```
>>> module_id = 1
>>> 'Module #{}'.format(module_id)
'Module #1'
>>> digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> ''.join([str(i) for i in digits])
'0123456789'
>>> text = 'Love thy neighbor'
>>> text.split()
['Love', 'thy', 'neighbor']
>>> text.find('e')
3
```

Comments

```
>>> x = [1, 2, 3] # list of integers
>>> name = 'Sergey Kabanov' # my name
>>> # x = 11
>>> '''
... Docstring
... '''
'\nDocstring\n'
>>> def mul(x, y):
...     '''
...         Multiplication of x and y
...     '''
...     return x * y
...
```

Comments

```
>>> mul(11, 11)
121
>>> help(mul)
Help on function mul:

mul(x, y)
    Multiplication of x and y

>>> mul.__doc__
'\n        Multiplication of x and y\n    '
```

PEP8

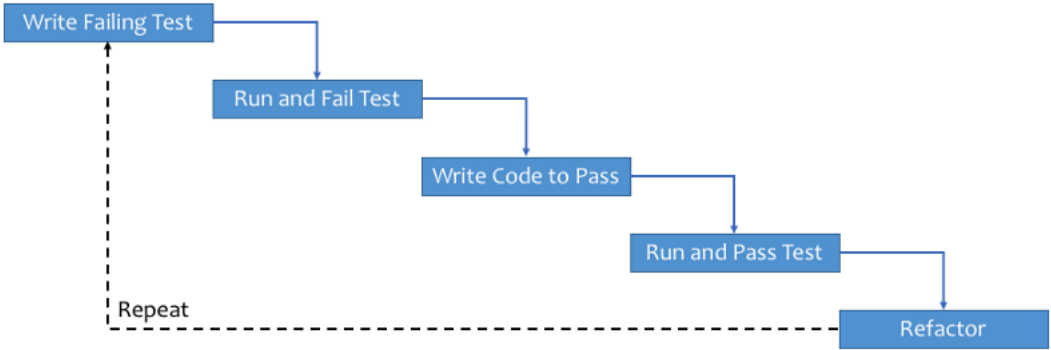
- PEP8 is the Style Guide for Python Code by Guido van Rossum, Barry Warsaw and Nick Coghlan.
  This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".
- PEP8 utility:

  ```
  $ pep8 --first <script_name>.py
  ```

Test Driven Development

- "I am a self-taught beginning developer who is able to write simple apps. But I have a confession to make. It's impossible to remember how everything is interconnected in my head. This situation is made worse if I come back to the code I've written after a few days. Turns out that this problem could be overcome by following a Test Driven Development (TDD) methodology." – Junior Pythonista
- Modules for testing: doctest, unittest, pytest

Test Driven Development



Doctest

File mul.py:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


def mul(x, y):
    '''
        Multiplication of x and y
        >>> mul(10, 2)
        20
    '''
    return x * y
```

Doctest

End of file *mul.py*:

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

```
$ python3 mul.py
$ python3 mul.py -v
```

```
Trying:
    mul(10, 2)
Expecting:
    20
ok
1 items passed all tests:
   1 tests in __main__.mul
1 tests in 1 items.
1 passed and 0 failed.
Test passed.
```

Learning resources

- Video lectures:
  Umnov, SHAD, Lecture 1, RU
  Umnov, SHAD, Lecture 2, RU
- PEP8:
  Official PEP8, ENG
  Brief overview of PEP8, RU
  pep8 utility, ENG
- Doctest:
  Official doc, ENG
  Ru doc, RU
- TDD:
  TDD and pytest, RU
  Set of articles, ENG