



ТЕХНОТРЕК

Лекция 2

ИНТЕНСИВ ПО PYTHON

Елизарова Алена

flashback to lesson 1



Команды linux

`pwd, ls, cd, mkdir, rm, cp, mv, tree, man`

`virtualenv venv` (venv - название окружения)

`deactivate` - деактивировать окружение

Альтернатива запуска скрипта под виртуальным окр.

`/Users/a.elizarova/workspace/track2sem/track2semvenv/bin/python file_name.py`

Редакторы кода

hard mode: vim, emacs

модульные: Sublime, Visual Studio Code, Atom

полноценные IDE: PyCharm

Python



Интерпретируемый язык с динамической типизацией и автоматической сборкой мусора.

Python - название спецификации языка

Реализации:

- CPython
- IronPython (DotNet)
- PyPy

<https://github.com/python/cpython>

<https://www.python.org/doc/>



Гвидо Ван Россум
(создатель языка Python)

Начинаем программировать >>>



```
python
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017,
20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
on darwin
Type "help", "copyright", "credits" or
"license" for more information.
>>> 2 * 3
6
>>>help(print)
>>>exit()
```

```
#Запуск скрипта
python file_name.py
```

Переменные



```
num = 1 # операция присваивания переменной  
        # num значения 1
```

```
# допустимые символы: буквы, цифры, _  
# НО: начинается с буквы или _
```

```
# valid:  
num = 1  
_num = 1  
__num = 1  
num35num35 = 1  
first_num = 1
```

```
# invalid:  
1num = 1
```

Базовые типы и конструкции



Целые числа (int)

```
num = 42  
num = 42_000_000 # начиная с python 3.6
```

```
>>>print(type(num))  
<class 'int'>
```

Вещественные числа (float)

```
float_num = 3.14  
float_num = 3.14e2 # 3.14 умножить на 10 в степени 2
```

Конвертация типов

```
float_num = float(num)
```

Базовые типы и конструкции



Комплексные числа (complex)

```
num = 14 + 1j # num.real, num.imag - для доступа  
              # до реальной и мнимой частей
```

Основные операции с числами

```
    + - * / ** % //  
>>> 6 * 6.0  
36.0
```

```
>>> 36 / 6  
6.0 # результат деления всегда вещественный
```

```
>>> 8 / 0 # ZeroDivisionError
```

```
# % - остаток от деления  
# ** - возведение в степень  
# // - целочисленное деление
```

Базовые типы и конструкции



Меняем местами переменные

```
>>>a, b = b, a # swap
```

Логический тип (bool)

```
True, False # подтип целого числа, т.е. 1 и 0
```

```
>>>13 == 13 # True
```

```
>>>13 != 13 # False
```

```
> < <= >=
```

```
1 < 2 < 3 # множественное сравнение
```


Базовые типы и конструкции



Логические выражения

and, or, not # порядок выполнения - по
приоритету или с помощью ()

Логические выражения ленивы

```
>>>x = 13
>>>y = False
>>>print(x or y)
13
```

Базовые типы и конструкции



Строки

```
some_str = 'Valid'
some_str2 = "Valid too"
some_str3 = """Valid as well"""
some_raw_str = r'Valid raw str'
```

Операции со строками

```
'Hello' + ' world'
'hello ' * 3
'hello'[1:4] # 'ell'
'hello'[::-1] # 'olleh'
len('hello') # 5
```

Базовые типы и конструкции



Методы строк

```
some_str = 'hello world'
some_str.count('l') # 3
some_str.capitalize() # Hello world
some_str.isdigit() # False
```

<https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html>

Оператор **in** проверяет на вхождение элемента в последовательность

```
'hello' in 'hello world' # True
```

Оператор **for ... in** итерируется по каждому элементу

```
for i in 'hello':
    print(i) # выведет каждую букву hello поочередно
```



Форматирование строк

```
>>>'%s do not like this method'%('I')  
'I do not like this method'
```

```
>>>'{} method is much {}'.format('I', 'better')  
'This method is much better'
```

```
>>>'{name} likes {what} most of all'.format(  
    name='Alena',  
    what='this method'  
)
```

```
#f-строки >= python 3.6  
attribute = 'new'  
>>> f'This method is {attribute} in Python 3.6'
```

Конструкции управления потоком



If ... elif ... else

```
welcome_str = 'Hello, System, this is Nick'
if 'Nick' in welcome_str:
    print('This is Nick')
elif 'Mary' in welcome_str:
    print('This is Mary')
else:
    print('Unknown person detected!')
```

```
# аналог тернарного оператора
person_name = None # объект типа NoneType
test_text = person_name if person_name else 'Name'
```

Конструкции управления потоком



while

```
i = 0
while i < 100:
    i += 1
print(i) # 100
```

range

```
for i in range(3):
    print(i) # 0 1 2
```

pass # определяет пустой блок

break # выход из цикла досрочно

continue # перейти к следующей итерации цикла



Списки

```
empty_list = []  
empty_list = list()
```

```
none_list = [None] * 10
```

```
user_list = [['Alena', 5.5], ['Sergey', 6.9]]  
len(user_list)  
user_list[0]  
user_list[0:]  
new_user_list = user_list[:] # скопирует список в  
новый  
user_list.append(['Oleg', 3.2])  
user_list.extend([[Lera, 5.7]])  
del user_list(2)
```



Сортировка списка

```
my_list = [7, 4, 6, 3]
```

```
# возвращает новый список, не изменяя старый  
my_sorted_list = sorted(my_list)
```

```
# сортирует исходный список  
my_sorted_list.sort()
```




Кортежи - неизменяемые списки

```
empty_tuple = ()  
empty_tuply = tuple()
```

```
immutables = ('int', 'str', 'tuple')
```

Кортежи хэшируемы и могут использоваться в качестве ключей словаря



Словари

```
empty_dict = ()  
empty_dict = dict()
```

```
teachers = {  
    'Math': 'Kate Johnson',  
    'English': 'David Lewis'  
}
```

```
teachers['Math'] # Kate Johnson  
teachers.get('Math', 'Unknown')  
teachers['Chemistry'] = 'Julia White'  
teachers.update({'Chemistry': 'Nolan Black'})  
del teachers['Math']
```

```
'Chemistry' in teachers # True
```

```
for i in teachers:  
    print(i) # итерируемся по ключам словаря
```



Множества

```
empty_set = set()  
empty_set = {1, 2, 3, 4}
```

```
empty_set.add(1)  
empty_set.remove(1)
```

```
# множества поддерживают мат. операции над множествами  
# объединение |  
# пересечение &  
# разность -  
# симметрическая разность ^
```

frozenset - неизменяемые множества

Функции



```
def multiply(a, b):  
    '''Multiply a by b'''  
    return a*b
```

`multiply.__doc__` - docstring для функции

Классы



```
isinstance(3, int) # проверка на принадлежность классу
```

```
# объявляем пустой класс
class Human(object):
    pass
```

```
print(dir(Human)) # методы
```

```
class Planet(object):
    count = 0 # экземпляр класса

    def __init__(self, name):
        self.name = name # атрибут экземпляра

    def __str__(self):
        return self.name
```

```
planet = Planet('Earth') # создаем экземпляр класса
```

Модули и пакеты



`file.py` - модуль

`--app/` - пакет
 `|--__init__.py`
 `|--hello.py`

Валидные импорты

```
import sys
from sys import path
from sys import *
```

```
import sys as s
from sys import path as sys_path
```

Тестирование. unittest



```
import unittest

def multiply(a, b):
    return a*b

class TestMultiplyItems(unittest.TestCase):
    def test_multiply(self):
        self.assertEqual(multiply(2, 3), 6)

if __name__ == '__main__':
    unittest.main()

# запуск
python -m unittest my_module
```

PEP8. Линтеры.



<https://www.python.org/dev/peps/pep-0008/>

<https://www.pylint.org/>

<http://flake8.pycqa.org/>



1.

```
a = b = 0
```

```
a += 1
```

```
print(a, b)
```

2.

```
x = y = []
```

```
x.append(1)
```

```
print(x, y)
```



3.

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

```
a.append(a)
```

```
print(a[4][4][4])
```

4.

```
12 * 5
```

```
print(_)
```



5.

```
Some = type("Some", (object,), {"x": "hello"})
```

6.1.

```
a,b,*c = (1,2,3,4,5,6,7,8)
```

6.2.

```
a,*b,c = (1,2,3,4,5,6,7,8)
```



7.

```
l = ["spam", "ham", "eggs"]  
list(enumerate(l))
```

8.

```
x = [1,2,3]  
y = x[:]
```

9.

```
import __hello__
```

Домашнее задание №2



- Реализовать класс DoubleLinkedList на Python - 5
- Написать тесты (unittest) для класса - 4
- Проверить граничные условия в тестах - 1
- Проверить корректность и стиль кода с помощью pylint или flake8 - 2

Срок сдачи

8 октября 2018



ТЕХНОТРЕК

**Спасибо за
внимание!**