ТЕХНОАТОМ

Интенсив по Python

Лекция 2

Илья Кириллов

Python

ТЕХНОАТОМ

Интерпретируемый язык с динамической типизацией и автоматической сборкой мусора.

Реализации:

- CPython
- IronPython (DotNet)
- PyPy
- Jython

```
class list(object):

    def append(self, *args, **kwargs): # real signature unknown
        """ Append object to the end of the list. """
        pass

    def clear(self, *args, **kwargs): # real signature unknown
        """ Remove all items from list. """
        pass

    ...
```

Гвидо Ван Россум
(создатель языка Python)

Python

ТЕХНОАТОМ

Интерпретируемый язык с динамической типизацией и автоматической сборкой мусора.

```
var = 1
var.append(2)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'append'
```

```
var = 1
var += '2'

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +=: 'int' and 'str'
```

Python

ТЕХНОАТОМ

```
> '5' - 3
< 2          // слабая типизация + приведение типов = головная боль
> '5' + 3
< '53'       // потому что все мы любим постоянство
> '5' - '4'
< 1          // строка - строка = число. Что?
> '5' + + '5'
< '55'       // окей, допустим
> 'foo' + + 'foo'
< 'fooNaN'   // чудесно!
> '5' + - '2'
< '5-2'      // хорошо
> '5' + - + - - + - - + + - + - + - + - - - '-2'
< '52'       // будем считать, что так и должно быть
> var x = 3;
> '5' + x - x
< 50         // логично
> '5' - x + x
< 5          // к чёрту математику
```

Python

ТЕХНОАТОМ

Интерпретируемый язык с динамической типизацией и автоматической сборкой мусора.

```
def function():
    var = 1
    return var

func_result = function()
```

```
import  gc

gc.disable()  # disable additional garbage collectors
gc.collect()
```

Переменные

ТЕХНОАТОМ

```
num = 1

# valid:
num = 1
num35num35 = 1
first_num = 1


# invalid:
1num = 1
```

```
# pythonic way - snake case
my_super_var = 1


# Non pythonic way - camel case
MySuperVar = 1
```

Переменные

ТЕХНОАТОМ

```
# protected
# ignored in "from module import *"
_num = 1



# private
# ignored in "from module import *"
__num = 1
```

Переменные: walrus оператор

ТЕХНОАТОМ

```
# Without Python 3.8 Walrus Operator 1
a = [1, 2, 3, 4]
length = len(a)
if length > 3:
    print("List is too long")
    print(length)


# Without Python 3.8 Walrus Operator 2
a = [1, 2, 3, 4]
if len(a) > 3:
    print("List is too long")
    print(len(a))


# With Python 3.8 Walrus Operator
a = [1, 2, 3, 4]
if (length := len(a)) > 3:
    print("List is too long")
    print(length)
```

Базовые типы и конструкции: numbers

ТЕХНОАТОМ

```
# Целые числа (int)
num = 42
num = 42_000_000  # начиная с python 3.6

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

```
# Вещественные числа (float)
float_num = 3.14
float_num = 3.14e2  # 3.14 умножить на 10 в степени 2

>>> type(float_num)
<class ‘float’>

>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.9999999999999999

>>> 0.1 -> 0.100000000001
```

```
# Конвертация типов
>>> float_num = float(num)
>>> num = int(float_num)
```

Базовые типы и конструкции: numbers

ТЕХНОАТОМ

```
# Комплексные числа (complex)
num = 14 + 1j

>>> num.real
14.0
>>> num.imag
1.0
```

```
# Основные операции с числами
+ - * / ** % //

>>> 6 * 6.0
36.0

>>> 36 / 6
6.0  # результат деления всегда вещественный начиная с python 3

>>> 8 / 0
ZeroDivisionError: division by zero

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

Базовые типы и конструкции

ТЕХНОАТОМ

```
# Меняем местами переменные
>>> a = 1
>>> b = 2
>>> a, b = b, a  # swap
>>> print(a, b)
2 1
```

```
# Логический тип (bool)
>>> 13 == 13
True
>>> 13 != 13
False

# Приведение типов
>>> bool(0)
False

# Операторы сравнения > < <= >= == !=

# Множественное сравнение
>>> 100 < 1000 < 10000  # 100 < 1000 and 1000 < 10000
True
```

Базовые типы и конструкции

ТЕХНОАТОМ

```
# Логические выражения

and, or, not

>>> True and not False or False
True
```

```
# Логические выражения ленивы

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

Базовые типы и конструкции: строки

ТЕХНОАТОМ

```
# Строки

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

```
# Операции со строками

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

Базовые типы и конструкции: строки

ТЕХНОАТОМ

```
# Методы строк

>>> some_str = 'hello world'
>>> some_str.count('l')
3
>>> some_str.capitalize()
'Hello world'
>>> some_str.isdigit()
False
>>> some_str.split(' ')
['hello', 'world']
```

```
# Оператор in проверяет на вхождение элемента в последовательность

>>> 'hello' in 'hello world'
True

# Оператор for … in  итерируется по каждому индексу

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

Базовые типы и конструкции: строки

ТЕХНОАТОМ

```
# Форматирование строк

>>> '%s do not like this method' % 'I'
'I do not like this method'

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

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


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

```
# Приведение строк

>>> bool('')
False
```

Базовые типы и конструкции: списки

ТЕХНОАТОМ

```
# Списки

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

```
# Операции со списками

>>> none_list = [None] * 10
[None, None, None, None, None]
>>> some_list = [['A', 5.5], ['B', 6.9]]
>>> len(some_list)
2
>>> some_list[0]
['A', 5.5]
>>> some_list.append(['C', 3.2])
>>> some_list.extend([['D', 5.7], ['E', 8.1]])
>>> some_list
[['A', 5.5], ['B', 6.9], ['C', 3.2], ['D', 5.7], ['E', 8.1]]
>>> some_list[1:4]
[['B', 6.9], ['C', 3.2]]
>>> new_some_list = some_list[:] # скопирует список в новый
>>> del some_list[2]
```

Базовые типы и конструкции: списки

ТЕХНОАТОМ

```
# Сортировка списка

my_list = [7, 4, 6, 3]

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

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

```
# Приведение списков

>>> bool([])
False
```

Базовые типы и конструкции: списки

ТЕХНОАТОМ

```
# Разворот списка

my_list = [1, 2, 3, 4]

# возвращает новый список, не изменяя старый
my_reversed_list = list(reversed(my_list))


# возвращает новый список, не изменяя старый
my_reversed_list = my_list[::-1]

# сортирует исходный список
my_list.reverse()
```

```
# Разворот списка на собеседовании
my_list = [1, 2, 3, 4, ..., n]
for i in range(len(my_list) // 2):   # сложность O(n/2)
    my_list[i], my_list[-i - 1] = my_list[-i - 1], my_list[i]
```

Базовые типы и конструкции: кортежи

ТЕХНОАТОМ

```
# Кортежи - неизменяемые списки

empty_tuple = ()
empty_tuple = tuple([1, 2, 3])

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

```
# Приведение кортежей

>>> bool(())
False
```

Базовые типы и конструкции: словари

ТЕХНОАТОМ

```
# Словари
empty_dict = {}
empty_dict = dict()
```

```
# Работа со словарями
>>> teachers = {'Math':    'Kate Johnson',
                'English': 'David Lewis'
               }

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

>>> 'Chemistry' in teachers
True
```

Базовые типы и конструкции: словари

ТЕХНОАТОМ

```
# Работа со словарями

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

for k, v in teachers.items():
    print(k, v) # итерируемся по кортежам (k, v)

for v in teachers.values():
    print(v) # итерируемся по значениям словаря
```

```
>>> d = {2: 0, 3: 0, 1: 0, 5: 0, 4: 0}
>>> d
{2: 0, 3: 0, 1: 0, 5: 0, 4: 0}

for k in sorted(d):
    print(d[k])

for k, v in sorted(d.items()):
    print(k, v)

# from collections import OrderedDict
```

Базовые типы и конструкции: коллекции

ТЕХНОАТОМ

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

```
# Действия со множествами

not_empty_set.add(1) # ничего не сделает, 1 уже во множествее
not_empty_set.remove(1) # удалит 1 из мноожества

>>> len(not_empty_set)
3

>>> 2 in not_empty_set
True

# множества поддерживают мат. операции над множествами
    # объединение {1, 2} | {3, 4} -> {1, 2, 3, 4}
    # пересечение {1, 2} & {2, 3} -> {2}
    # разность {1, 2} - {1, 3} -> {3}
    # симметрическая разность {1, 2, 3} ^ {2, 3, 4} -> {1, 4}

# frozenset - неизменяемые множества
```

Базовые типы и конструкции: распаковка

ТЕХНОАТОМ

```
# Распаковка
* **
>>> numbers = [2, 1, 3, 4, 7]

# эквивалентно
>>> more_numbers = numbers + [11, 18]
>>> more_numbers = [*numbers, 11, 18]

>>> print(', '.join(mode_numbers))
>>> print(*more_numbers, sep=', ')
2, 1, 3, 4, 7, 11, 18
```

```
>>> date_info = {'year': "2020", 'month': "01", 'day': "01"}

# эквивалентно
>>> filename = "{year}-{month}-{day}.txt".format(year=date_info['year'],
                                                 month=date_info['month'],
                                                 day=date_info['day'])

>>> filename = "{year}-{month}-{day}.txt".format(**date_info)

>>> filename
'2020-01-01.txt'
```

Базовые типы и конструкции: распаковка

ТЕХНОАТОМ

```
>>> my_tuple = (1, 2)
>>> a, b = my_typle
>>> a
1
>>> b
2
```

```
>>> my_tuple = (1, 2, 3, 4, 5)
>>> a, _, _, _, b = my_tuple
>>> a
1
>>> _ # не будет использоваться, но данные запомнили
4
>>> b
5

>>> a, b = my_tuple
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

Эквивалентность и равность

ТЕХНОАТОМ

```
# == или is?

>>> 1 == 1
True
>>> 1 is 1
True

>>> [1, 2, 3] == [1, 2, 3]
True
>>> [1, 2, 3] is [1, 2 ,3]
False

>>> lst = [1, 2, 3]
>>> new_lst = lst
>>> lst is new_lst
True
```

```
>>> 1 is True
<stdin>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
False
```

Конструкции управления потоком

ТЕХНОАТОМ

```
# if ... elif ... else

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

```
person_name = None
...
# классический вариант
if person_name is None:
    test_text = 'Name'
else:
    test_text = person_name

# аналог тернарного оператора
test_text = person_name if person_name is not None else 'Name'
```

Конструкции управления потоком

ТЕХНОАТОМ

```
# while

>>> i = 0
>>> while i < 100:
>>>     i += 1

>>>print(i)
100
```

```
import time

timeout = 60
interval = 1

start_time = time.time()
while time.time() - started <= timeout:
    print('Waiting...')
    do_something()
    time.sleep(interval)
```

Конструкции управления потоком

ТЕХНОАТОМ

```
# pass, continue, break

for i in range(100):
    if i <= 10:
        pass  # определяет пустой блок

    elif i <= 50:
        continue  # перейти к следующей итерации цикла

    else:
        break # выход из цикла досрочно

    print(i)
```

```
for i in range(0):
    print(i)

else:
    print('Empty')
```

Исключения

ТЕХНОАТОМ

```
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
a = int(input())
b = int(input())

if b == 0:
    raise ZeroDivisionError('Can not divide by zero!')

print(a // b)


>>> 1
>>> 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: Can not divide by zero!
```

Исключения

ТЕХНОАТОМ

```
try:
   a = int(input())
   b = int(input())
   print(a // b)

except TypeError:
   print(f'Some inputs are not integers: {a} {b}')
   exit(0)

except ZeroDivisionError:
   print('Not a good idea to divide by zero...')

except (ValueError, OSError):
   print('ValueError or OSError occured!')

except Exception as e:
   print(f'Something went wrong: {e}!')
   raise e

finally:
   print('I just tries all i could...')
```

Функции

ТЕХНОАТОМ

```
def add(x, y):
    return x + y

>>> add(1, 10)
11
>>> add('abc', 'def')
'abcdef'
```

```
def newfunc(n):

    def myfunc(x):
        return x + n

    return myfunc   # возвращаем объект функции, не вызываем ее

>>> new = newfunc(100)
>>> type(new)
<class 'function'>
>>> new(200)
300
```

Функции

ТЕХНОАТОМ

```
def func():
     pass

>>> print(func())
None
```

```
def func(a, b, c=2):
     return a + b + c

>>> func(1, 2)
5
>>> func(1, 2, 3)
6
>>> func(1, b=2, c=3)
6
>>> func(a=3, c=6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: func() missing 1 required positional argument: 'b'

>>> func(1, 2, b=6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: func() got multiple values for argument 'b'
```

Функции

ТЕХНОАТОМ

```
def func(*args):
    return args

>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
>>> func(1)
(1,)
```

```
def func(**kwargs):
    return kwargs

>>> func(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
>>> func()
{}
>>> func(a='python')
{'a': 'python'}
```

Функции

ТЕХНОАТОМ

```
def func(*args, **kwargs):
    return args, kwargs

>>> func(1, 2, 3, 'abc', a=4, b=5, c=6)
((1, 2, 3, 'abc'), {'a': 4, 'b': 5, 'c': 6})

>>> pos_args, named_args = func(1, 2, 3, 'abc', a=4, b=5, c=6)
>>> pos_args
(1, 2, 3, 'abc')

>>> named_args
{'a': 4, 'b': 5, 'c': 6}
```

Функции: анонимные функции

ТЕХНОАТОМ

```
# анонимные функции

>>> func = lambda x, y: x + y
>>> func(1, 2)
3
>>> func('a', 'b')
'ab'
>>> (lambda x, y: x + y)(1, 2)
3
>>> (lambda x, y: x + y)('a', 'b')
'ab'
```

```
>>> func = lambda *args: args
>>> func(1, 2, 3, 4)
(1, 2, 3, 4)
```

Функции: области видимости

ТЕХНОАТОМ

```
a = 0
def func(a):
    a += 1
    print(a)

>>> func(1)
2
>>> func(2)
3
>>> print(a)
0
```

```
a = 0
def func(a=a):
    a += 1
    print(a)

>>> func()
1
>>> func(2)
3
>>> print(a)
0
```

Функции: области видимости

ТЕХНОАТОМ

```
def func(a=[]):
    a.append(1)
    print(a)

>>> func()
[1]
>>> func()
[1, 1]
>>> func([])
[1]
```

```
b = 1
def func(a):
    print(a, b)
    print(c)

>>> func(0)
0, 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in func
NameError: name 'c' is not defined
```

Функции высших порядков

ТЕХНОАТОМ

```
# map()

>>> list1 = [7, 2, 3, 10, 12]
>>> list2 = [-1, 1, -5, 4, 6]
>>> map(lambda x, y: x * y, list1, list2)
[-7, 2, -15, 40, 72]
```

```
# filter()

>>> numbers = [10, 4, 2, -1, 6]
>>> filter(lambda x: x < 5, numbers)
[4, 2, -1]
```

```
# reduce()

from functools import reduce
>>> numbers = [2, 3, 4, 5, 6]
>>> reduce(lambda res, x: res + x, numbers)
20
```

Рекурсия

ТЕХНОАТОМ

```
def fibonacci(n):
    if n in (1, 2):
        return 1

    return fibonacci(n - 1) + fibonacci(n - 2)


>>> print(fibonacci(10))
55
```

```
def fibonacci(n):
    return fibonacci(n - 1) + fibonacci(n - 2)


>>> print(fibonacci(10))
Traceback (most recent call last):
 return fibonacci(n - 1) + fibonacci(n - 2)
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

Декораторы

ТЕХНОАТОМ

```
counter = 0

def func():
    counter += 1
    pass

>>> func()
>>> func()
>>> func()
>>> print(counter)
3
```

```
counter2 = 0

def func2():
    counter += 1
    pass

>>> func2()
>>> func2()
>>> print(counter2)
2
```

Декораторы

ТЕХНОАТОМ

```
counter = {}

def count_decorator(f):
    def wrapper():
        if (k := f.__name__) in counter:
            counter[k] += 1
        else:
            counter[k] = 1
        return f()
     return wrapper


def func():
    pass

>>> decorated = count_decorator(func)
>>> type(decorated)
<class 'function'>

>>> decorated()
>>> decorated()
>>> print(counter)
{'func': 2}
```

Декораторы

ТЕХНОАТОМ

```
counter = {}

def count_decorator(f):
    def wrapper():
        if (k := f.__name__) in counter:
            counter[k] += 1
        else:
            counter[k] = 1
        return f()
     return wrapper


@count_decorator
def func():
    pass

>>> func()
>>> func()
>>> print(counter)
{'func': 2}
```

Декораторы

ТЕХНОАТОМ

```
counter = {}

def count_decorator(f):
    def wrapper(*args, **kwargs):
        if (k := f.__name__) in counter:
            counter[k] += 1
        else:
            counter[k] = 1
        return f(*args, **kwargs)
     return wrapper


@count_decorator
def func(a, b):
    print(a, b)

>>> func(1, b=2)
1 2
```

Декораторы

ТЕХНОАТОМ

```
counter = {}

def count_decorator(f):
    def wrapper(*args, **kwargs):
        if (k := f.__name__) in counter:
            counter[k] += 1
        else:
            counter[k] = 1
        return f(*args, **kwargs)
     return wrapper


@count_decorator
def test():
    assert 1 == 1

>>> print(test.__name__)
wrapper
```

Декораторы

ТЕХНОАТОМ

```
from functools import wraps


counter = {}

def count_decorator(f):
    @wraps(f)
    def wrapper():
        if (k := f.__name__) in counter:
            counter[k] += 1
        else:
            counter[k] = 1
        return f()
     return wrapper

@count_decorator
def test():
    assert 1 == 1

>>> print(test.__name__)
test
```

Классы

ТЕХНОАТОМ

```
# проверка на принадлежность классу
isinstance(3, int) -> True
isinstance('123', int) -> False
```

```
>>> dir('123')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

Классы

ТЕХНОАТОМ

```
# объявляем класс
class Human:
    age = None
    sex = None
    weight = None
```

```
>>> human = Human()  # создаем экземпляр класса
>>> human
<__main__.Human object at 0x10dfd2880>

>>> human2 = Human()  # создаем второй экземпляр класса
>>> human2
<__main__.Human object at 0x10dfd2f70>

>>> human2.age = 20
>>> human2.sex = 'M'
>>> human2.weight = 80

>>> human.age
None
>>> human2.__dict__
{'age': 20, 'sex': 'M', 'weight': 80}
```

Классы

ТЕХНОАТОМ

```
class Human:
    __age = None

    def set_age(self, age)
        self.__age = age

    def get_age(self):
        return self.__age
```

```
class Man(Human):
    pass

class Woman(Human):

    def get_age(self):
        return self.__age - 10
```

```
>>> women = Woman()
>>> women.set_age(30)
>>> print(women.get_age())
20
>>> women.__age -> 30  # нарушаем соглашение о приватности
>>> issubclass(Woman, Human) -> True
>>> isinstance(women, Human) -> True
```

Классы

ТЕХНОАТОМ

```
class Planet:
    count = 0  # атрибут класса

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

```
planet_1 = Planet('Earth')  # создаем экземпляр класса
planet_1.count += 1  # инкрементируем атрибут экземпляра

Planet.count += 2    # глобально инкрементируем статический атрибут класса

planet_2 = Planet('Mars')  # создаем экземпляр класса
planet_2.count += 1  # инкрементируем атрибут экземпляра

>>> planet_1.count
1
>>> planet_2.count
3
>>> planet_2.name
'Mars'
```

Классы

ТЕХНОАТОМ

```
class Planet:
    count = 0

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f'Name: "{self.name}"\nCount: {self.count}'

    def __int__(self):
        return self.count
```

```
>>> planet = Planet('Earth')
>>> print(planet)
Name: "Earth"
Count: 0

>>> print(int(planet))
0
```

Контекстные менеджеры

ТЕХНОАТОМ

```
f = open('my_file', 'r')
print(f.name)

print(f.readline())
print(f.readline())

print(f.tell())
f.close()
```

Контекстные менеджеры

ТЕХНОАТОМ

```
class FileOpener(object):

    def __init__(self, file_path):
        self.file_path = file_path

    def __enter__(self):
        print(self.file_path)
        self.f = open(self.file_path, 'r')
        return self.f

    def seek_to_start(self):
        self.f.seek(0)

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(self.f.tell())
        self.f.close()


with FileOpener('my_file') as f:
    print(f.readline())
    f.seek_to_start()
    print(f.readline())
```

Контекстные менеджеры

ТЕХНОАТОМ

```
from contextlib import contextmanager


@contextmanager
def file_opener(file_path):
    f = open(file_path, 'r')
    print(f.name)

    yield f

    print(f.tell())
    f.close()


with file_manager('my_file') as f:
    print(f.readline())
    print(f.readline())
```

Контекстные менеджеры

ТЕХНОАТОМ

```
import pytest

def test():
   with pytest.raises(ZeroDivisionError):
       assert 1 / 0
```

```
import pytest


class MyTestException(Exception):
    pass


def check_divide(a, b):
    try:
        assert a // b
    except ZeroDivisionError:
        raise MyTestException('Divided by zero!')

def test():
   check_divide(1, 1)
   with pytest.raises(MyTestException):
       check_divide(1, 0)
```

Контекстные менеджеры

ТЕХНОАТОМ

```
from contextlib import contextmanager


class DidNotRaisedException(Exception):
    pass

class RaisedAnotherException(Exception):
    pass


@contextmanager
def raises(exception_type):
    try:
        yield
        raise DidNotRaisedException(f'Did not raised exception {exception_type}'

    except exception_type:
        print('Exception raised, correct!')

    except Exception as e:
        raise RaisedAnotherException(f'Raised exception {e} but {exception_type} expected')
```

Итераторы

ТЕХНОАТОМ

```
num_list = [1, 2, 3, 4, 5]
for i in num_list:
    print(i)
```

```
>>> itr = iter(num_list)
>>> print(next(itr))
1
>>> print(next(itr))
2
>>> print(next(itr))
3
>>> print(next(itr))
4
>>> print(next(itr))
5
>>> print(next(itr))
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
   print(next(itr))
StopIteration
```

Итераторы и генераторы

ТЕХНОАТОМ

```
class SimpleIterator:

    def __init__(self, limit):
        self.limit = limit
        self.counter = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return 1
        else:
            raise StopIteration

s_iter2 = SimpleIterator(5)

for i in s_iter2:
    print(i)
```

Генераторы

ТЕХНОАТОМ

```
def simple_generator(val):
   while val > 0:
       val -= 1
       yield 1

>>> gen_iter = simple_generator(5)
>>> print(next(gen_iter))
1
>>> print(next(gen_iter))
1
>>> print(next(gen_iter))
1
>>> print(next(gen_iter))
1
>>> print(next(gen_iter))
1
>>> print(next(gen_iter))
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
   print(next(itr))
StopIteration
```

Генераторы

ТЕХНОАТОМ

```
# /tmp/my_file  100Gb

>>> f = open('/tmp/my_file', 'r')
>>> print(f.read())
Killed
```

```
import os


def generator_read(file_path):
    f = open(file_path, 'r')
    while f.tell() < os.path.getsize(file_path):
        yield f.readline()

    f.close()


for line in generator_read('/tmp/my_file'):
    print(line)
```

Генераторы

ТЕХНОАТОМ

```
import os


def generator_read(file_path):
    with open(file_path, 'r') as f:
        while f.tell() < os.path.getsize(file_path):
            yield f.readline()


for line in generator_read('/tmp/my_file'):
    print(line)
```

```


with open('/tmp/my_file', 'r') as f:
    for line in f:
        print(line)
```

Генераторные выражения

ТЕХНОАТОМ

```
my_list = [1, 2, 3, 4, 5]
gen = (x for x in my_list)

>>> type(gen)
<class 'generator'>


condition_gen = (x for x in my_list if x % 2 == 0)
changing_gen = (x ** x for x in my_list)
```

```
for i in condition_gen:
    assert i % 2 == 0
```

```
>>> r = range(10)
>>> type(r)
<class 'range'>
>>> list(r)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Генераторы списков / словарей

ТЕХНОАТОМ

```
>>> my_list = [1, 2, 3, 4, 5]
>>> gen = [x for x in my_list]

>>> type(gen)
<class 'list'>


>>> [x for x in my_list if x % 2 == 0]
[2, 4]
>>> [x ** x for x in my_list]
[1, 4, 27, 256, 3125]
```

```
>>> my_dict = {1: 1, 2: 2, 3: 3, 4: 4, 5: 5}

>>> {k: v for k, v in my_dict.items() if k % 2 == 0}
{2: 2, 4: 4}
```

Модули и пакеты

ТЕХНОАТОМ

```
file.py - модуль

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

Валидные импорты

```
import app
from app import hello
from app import *

import app as a
from app import hello as new_hello
```

Интерактив

ТЕХНОАТОМ

```
1.
a = b = 0
a += 1
print(a, b)


2.
x = y = []
x.append(1)
print(x[0], y[0])
```

Интерактив

ТЕХНОАТОМ

```
3.
a = [1, 2, 3, 4]
a.append(a)
print(a[4][4][4])
```

```
4.
12 * 5
print(_)
```

Интерактив

ТЕХНОАТОМ

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

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

7.
```
tpl = (1, [2, 3], 5)
tpl[1].append(4)
tpl
```

Интерактив

ТЕХНОАТОМ

8.
```
l = ["spam", "ham", "eggs"]
list(enumerate(l))
```

9.
```
x = [1,2,3]
del x[::2]
```

10.
True == False in [False]

Интерактив

ТЕХНОАТОМ

11.
```
import __hello__
```

ТЕХНОАТОМ

Спасибо за внимание!