# Интенсив Python

## Лекция 4
## Классы

Кухтичев Антон

образование

# Классы

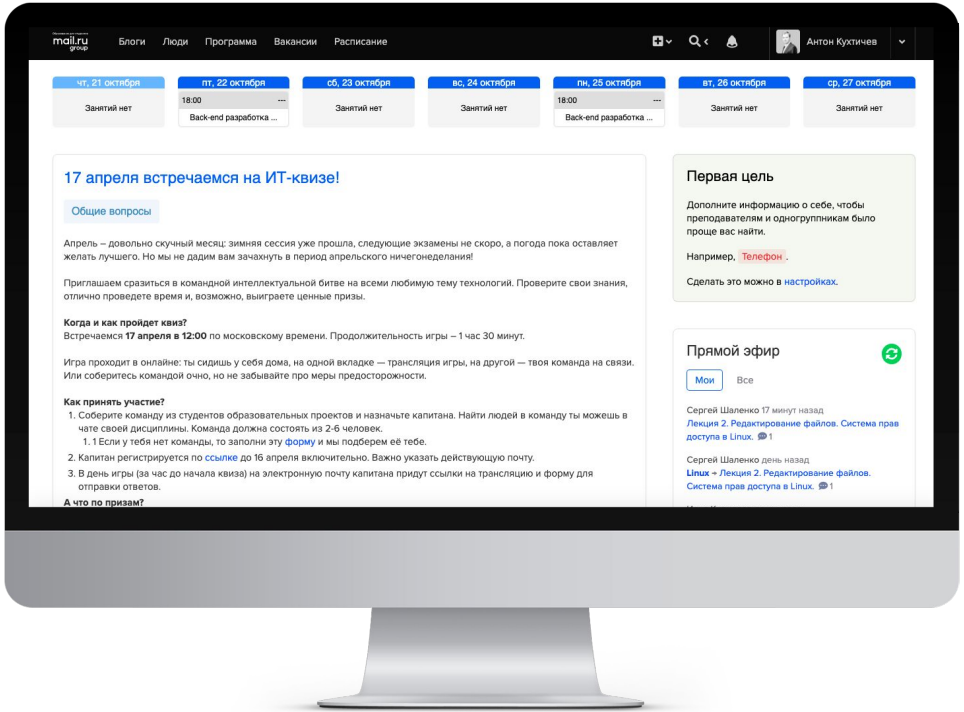
Антон Кухтичев

# Содержание занятия

- Встроенные функции
- Классы

# Напоминание отметиться на портале

и оставить отзыв после лекции



# Встроенные функции

# map

```
map(function, iterable, [iterable 2, iterable 3, ...])
```

```
def func(el1, el2):
    return '%s|%s' % (el1, el2)
list(map(func, [1, 2], [3, 4, 5]))  # ['1|3', '2|4']
```

Применяет указанную функцию к каждому элементу указанной последовательности/последовательностей.

# reduce

```
from functools import reduce

reduce(function, iterable[, initializer])

items = [1,2,3,4,5]

sum_all = reduce(lambda x,y: x + y, items)
```

Применяет указанную функцию к элементам последовательности, сводя её к единственному значению.

# filter

```
filter(function, iterable)
def is_even(x):
    return x % 2 == 0:
>>> print(list(filter(is_even, [1, 3, 2, 5, 20, 21])))
[2, 20]
```

Функция filter предлагает элегантный вариант фильтрации элементов последовательности. Принимает в качестве аргументов функцию и последовательность, которую необходимо отфильтровать.

# zip

```
>>> a = [1,2,3]
>>> b = "xyz"
>>> c = (None, True)
>>> print(list(zip(a, b, c)))
[(1, 'x', None), (2, 'y', True)]
```

Функция zip объединяет в кортежи элементы из последовательностей переданных в качестве аргументов.

# compile

```
compile(source, filename, mode, flag, dont_inherit, optimize)

# выполнение в exec
>>> x = compile('x = 1\nz = x + 5\nprint(z)', 'test', 'exec')
>>> exec(x)
# 6


# выполнение в eval
>>> y = compile("print('4 + 5 =', 4+5)", 'test', 'eval')
>>> eval(y)
# 4 + 5 = 9
```

# exec

```
exec(obj[, globals[, locals]]) -> None
```

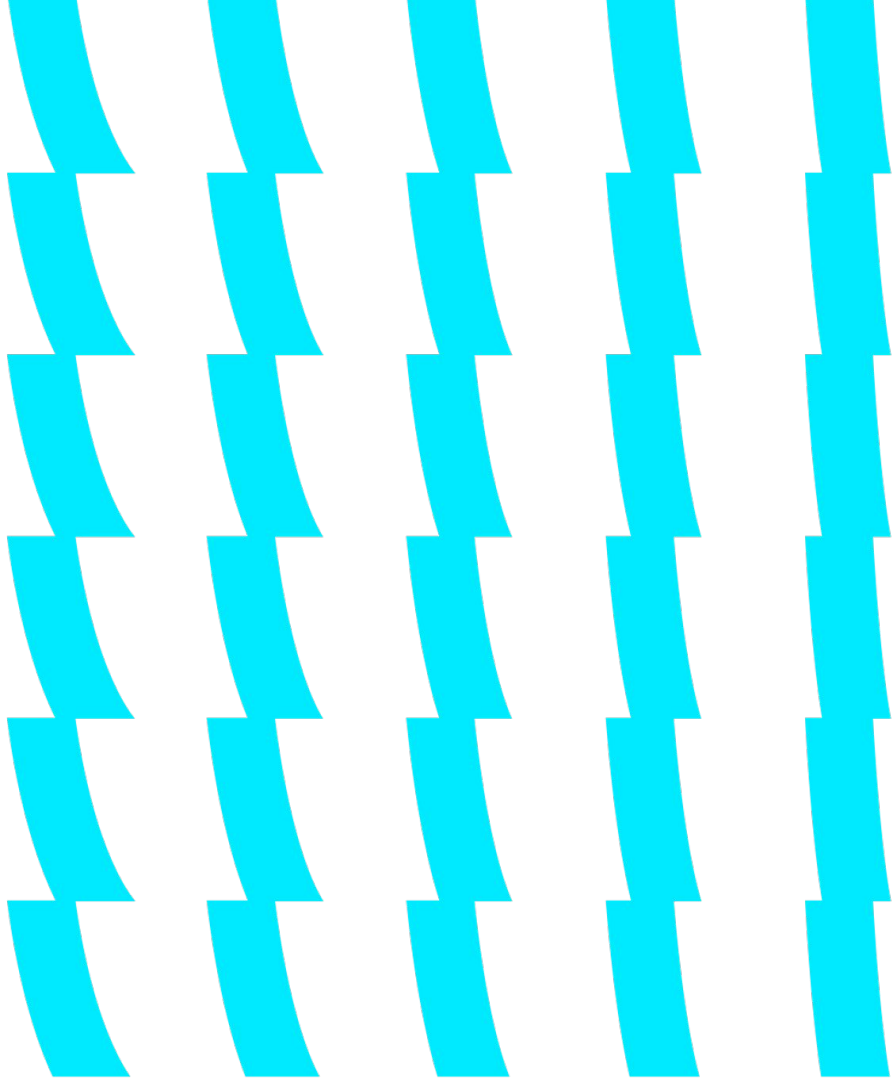
Динамически исполняет указанный код.

# eval

```
eval(expression[, globals[, locals]])
```

- в eval() запрещены операции присваивания;
- SyntaxError также вызывается в случаях, когда eval()не удается распарсить выражение из-за ошибки в записи;
- Аргумент globals опционален. Он содержит словарь, обеспечивающий доступ eval() к глобальному пространству имен;
- В locals-словарь содержит переменные, которые eval() использует в качестве локальных имен при оценке выражения.

# Классы



*Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects.*

docs.python.org

# Классы: атрибуты

```
class A:
    name = "cls_name"
    __cls_private = "cls_private"

    def __init__(self, val):
        self.val = val
        self._protected = "protected"
        self.__private = "private"

    def print(self):
        print(
            f"{self.val=}, {self._protected=}, {self.__private=}, "
            f"{self.name=}, {self.__cls_private=}"
        )
```

# Классы: свойства

```
# классический подход

class Author:

    def __init__(self, name):
        self.__name = ""
        self.set_name(name)

    def get_name(self):
        return self.__name

    def set_name(self, val):
        self.__name = val
```

```
# pythonic

class Author:

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

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, val):
        self.__name = val
```

# Классы: свойства

```
class Author:
    def __init__(self, name):
        self.name = name

    @property
    def name(self):
        """name doc"""
        return self.__name

    @name.setter
    def name(self, val):
        self.__name = val

    @name.deleter
    def name(self, val):
        self.__name = val
```

```
class Author:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.__name

    def set_name(self, val):
        self.__name = val

    def del_name(self):
        del self.__name

    name = property(
        get_name,
        set_name,
        del_name,
        "name doc",
    )
```

# Классы: свойства read/write only

```
class Author:
    def __init__(self, name, password):
        self.__name = name
        self.password_hash = None
        self.password = password

    @property
    def name(self):
        """name is read-only"""
        return self.__name

    @property
    def password(self):
        raise AttributeError("Password is write-only")

    @password.setter
    def password(self, plaintext):
        self.password_hash = make_hash_from_password(plaintext)
```

# Классы: методы

```
class A:
    @staticmethod
    def print_static():
        print("static")

    @classmethod
    def print_cls(cls):
        print(f"class_method for {cls.__name__}")

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

    def print_offset(self, offset=10):
        print(self.val + offset)

    def __str__(self):
        return f"{self.__class__.__name__}:val={self.val}"
```

# Классы: доступ к атрибутам

Чтобы найти атрибут объекта `obj`, python обыскивает:

1. Сам объект (`obj.__dict__` и его системные атрибуты)
2. Класс объекта (`obj.__class__.__dict__`).
3. Классы, от которых унаследован класс объекта (`obj.__class__.__mro__`)

# Классы: магические атрибуты

**Классы**

`__name__` — имя класса

`__module__` — модуль, в котором объявлен класс

`__qualname__` — fully qualified имя

`__doc__` — докстринг

`__annotations__` — аннотации статических полей класса

`__dict__` — namespace класса

**Методы**

`__self__` — объект класса

`__func__` — сама функция, которую мы в классе объявили

# Классы: магические атрибуты

**Поля, относящиеся к наследованию**

`__bases__` — базовые классы

`__base__` — базовый класс, который указан первым по порядку

`__mro__` — список классов, упорядоченный по вызову функции `super`

```
class B(A): pass
>>> B.__bases__
(__main__.A,)
>>> B.__base__
__main__.A
>>> B.__mro__
(__main__.B, __main__.A, object)
```

# Классы: MRO

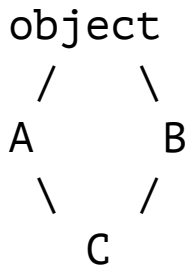
**Порядок разрешения методов** (method resolution order) позволяет python выяснить, из какого класса-предка нужно вызывать метод, если он не обнаружен непосредственно в классе-потомке.

```
cls.__mro__
cls.mro()
```

```
>>> B.mro()
[__main__.B, __main__.A, object]
```

# Классы: локальный порядок старшинства

```
>>> class A:
...     pass
...
>>> class B:
...     pass
...
>>> class C(A, B):
...     pass
...
>>> C.mro()
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]
>>>
>>> class C(B, A):
...     pass
...
>>> C.mro()
[<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
```



# Классы: магические методы

**`object.__new__(cls[, ...])`**

Статический метод, создает новый экземпляр класса.

После создание экземпляра вызывается (уже у экземпляра) метод `__init__`.

`__init__` ничего не должен возвращать (кроме None), иначе - `TypeError`

```
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls, *args, **kwargs)
        return cls._instance
```

# Классы: магические методы

**Доступ к атрибутам**

- `__getattribute__(self, name)`
- `__getattr__(self, name)`
- `__setattr__(self, name, val)`
- `__delattr__(self, name)`
- `__dir__(self)`

# Классы: магические методы

```
object.__call__(self[, args...])


class Adder:
    def __init__(self, val):
        self.val = val

    def __call__(self, value):
        return self.val + value


a = Adder(10)
a(5)  # 15
```

# Классы: магические методы

**To string**

`__repr__` — представление объекта. Если возможно, должно быть валидное python выражение для создание такого же объекта

`__str__` — вызывается функциями `str, format, print`

`__format__` — вызывается при форматировании строки

# Классы: магические методы

**Сравнение**

```
object.__lt__(self, other)
object.__le__(self, other)
object.__eq__(self, other)
object.__ne__(self, other)
object.__gt__(self, other)
object.__ge__(self, other)
```

```
x < y == x.__lt__(y)  # <=, ==, !=, >, >=
```

# Классы: магические методы

**Эмуляция чисел**

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__matmul__(self, other) (@)
object.__truediv__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)
object.__xor__(self, other)
object.__or__(self, other)
```

# Классы: магические методы

**Эмуляция чисел**

Методы вызываются, когда выполняются операции (`+`, `-`, `*`, `@`, `/`, `//`, `%`, `divmod()`, `pow()`, `**`, `<<`, `>>`, `&`, `^`, `|`) над объектами

```
x + y == x.__add__(y)
```

Есть все такие же с префиксом `r` и `i`:

`__radd__` - вызывается, если левый операнд не поддерживает `__add__`

`__iadd__` - вызывается, когда `x += y`

# Классы: магические методы

**Эмуляция контейнеров**

```
object.__len__(self)
object.__length_hint__(self)
object.__getitem__(self, key)
object.__setitem__(self, key, value)
object.__delitem__(self, key)
object.__missing__(self, key)
object.__iter__(self)
object.__next__(self)
object.__reversed__(self)
object.__contains__(self, item)
```

# Классы: магические методы

## __hash__

Вызывается функцией `hash()` и коллекциями, которые построены на основе hash-таблиц. Нужно, чтобы у равных объектов был одинаковый hash.

Если определен метод `__eq__` и не определен `__hash__`, то объект не может быть ключом в hashable коллекции.

```
>>> key1 = (1, 2, 3)
>>> key2 = (1, 2, 3, [4, 5])
>>> s = set()
>>> s.add(key1)  # ???
>>> s.add(key2)  # ???
```

# Классы: магические методы

## `__slots__`

Позволяет явно указать поля, которые будут в классе.

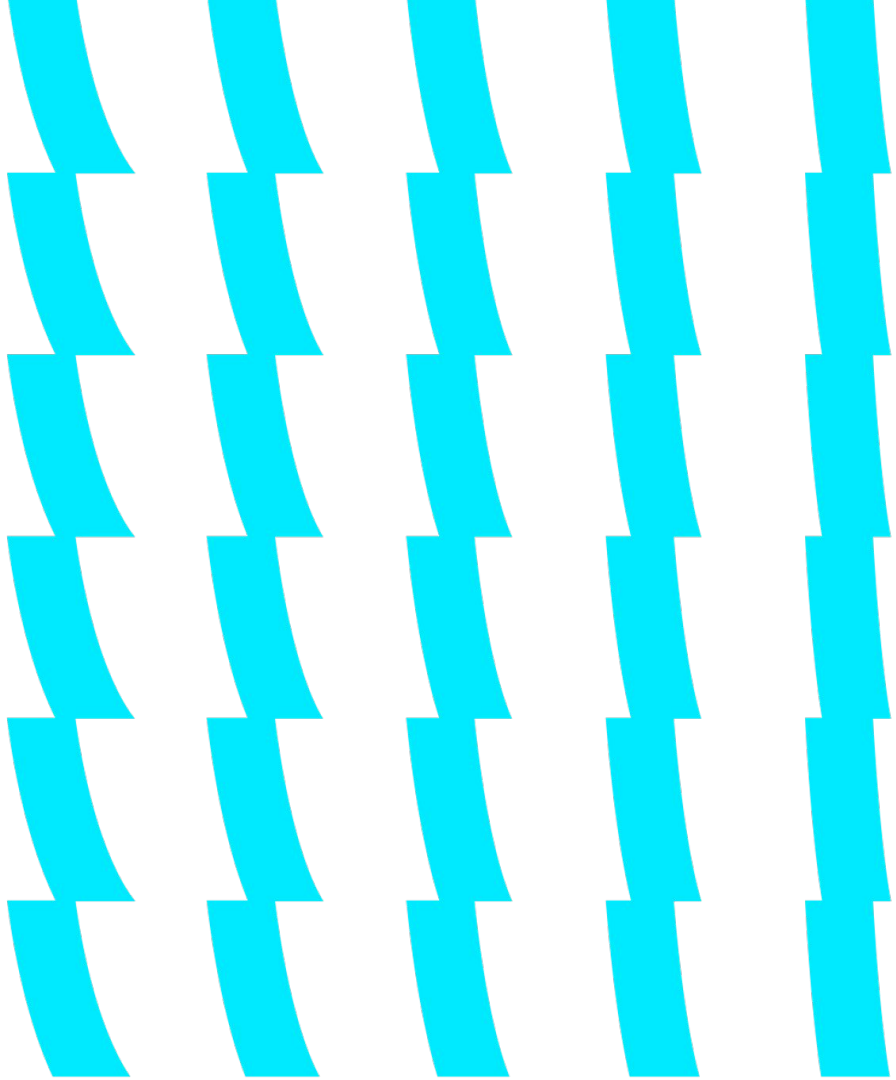
В случае указания `__slots__` пропадают поля `__dict__` и `__weakref__`.

Используя `__slots__` можно экономить на памяти и времени доступа к атрибутам объекта.

```
class Point:
    __slots__ = ('x', 'y')
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

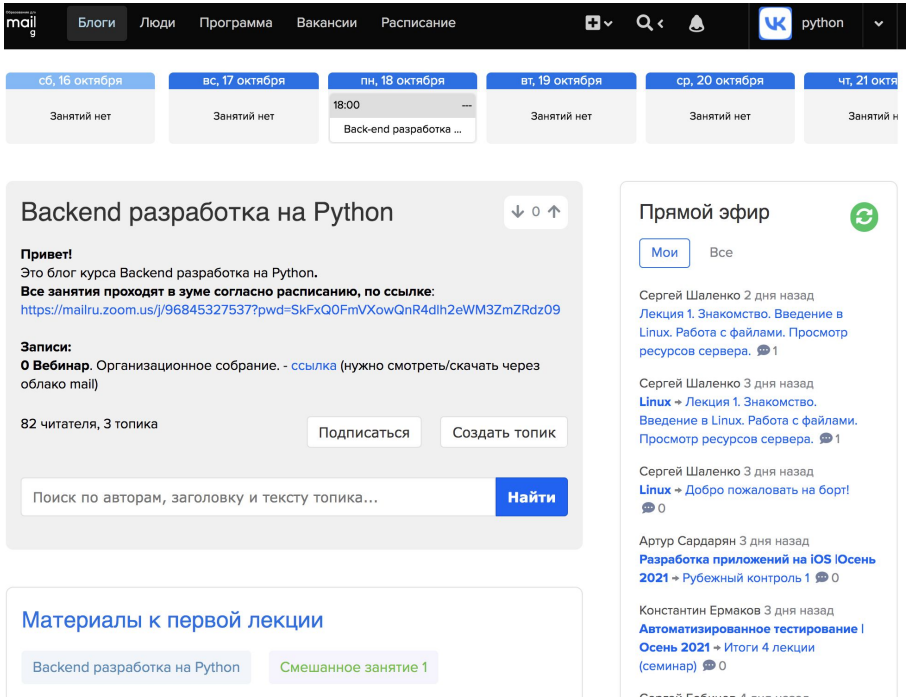
# Домашнее задание #4

- Реализовать кастомный список, унаследованный от `list`
- +тесты
- flake8 + pylint перед сдачей



# Напоминание отметиться на портале Vol 2

+ оставить отзыв



# Спасибо за внимание

образование