Углубленный Python

Лекция 5

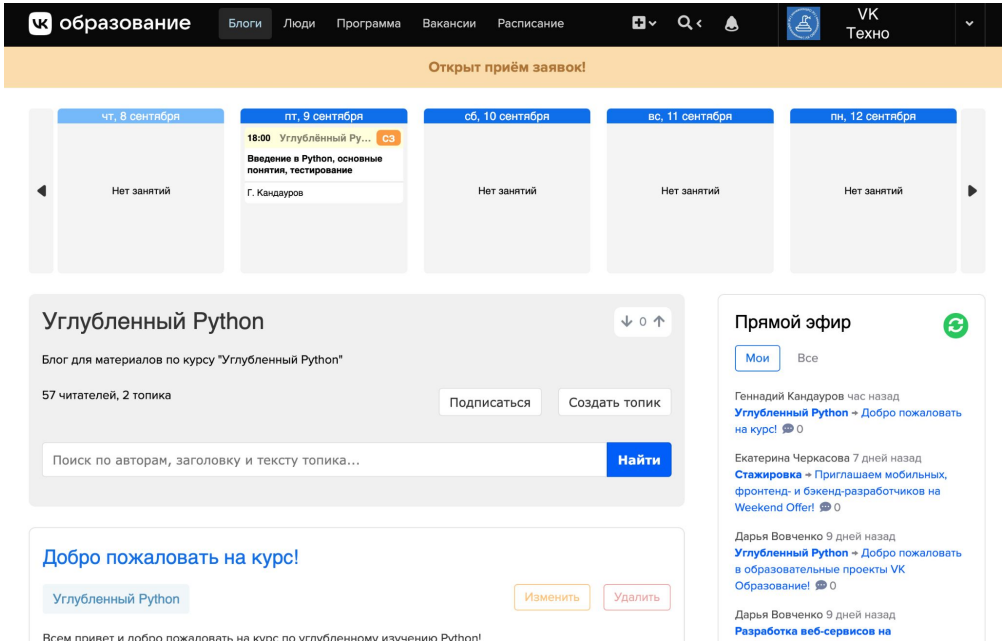
Метаклассы, ABC
Стандартная библиотека

Кандауров Геннадий

education

Напоминание отметиться на портале

+ оставить отзыв



Квиз про прошлой лекции

Содержание занятия

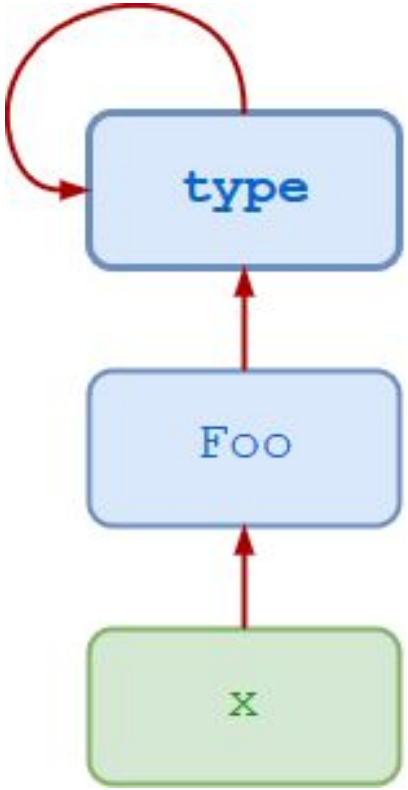
1. Метаклассы
2. ABC
3. Числа, строки
4. collections
5. functools
6. itertools
7. heapq
8. Файлы и каталоги

Метаклассы

Классы, экземпляры которых являются классами

Метаклассы: type

```
class Foo:
    pass

x = Foo()
```



Метаклассы: type

Новые классы создаются с помощью вызова

`type(<name>, <bases>, <classdict>)`

`name` – имя класса (`__name__`)

`bases` – базовые классы (`__bases__`)

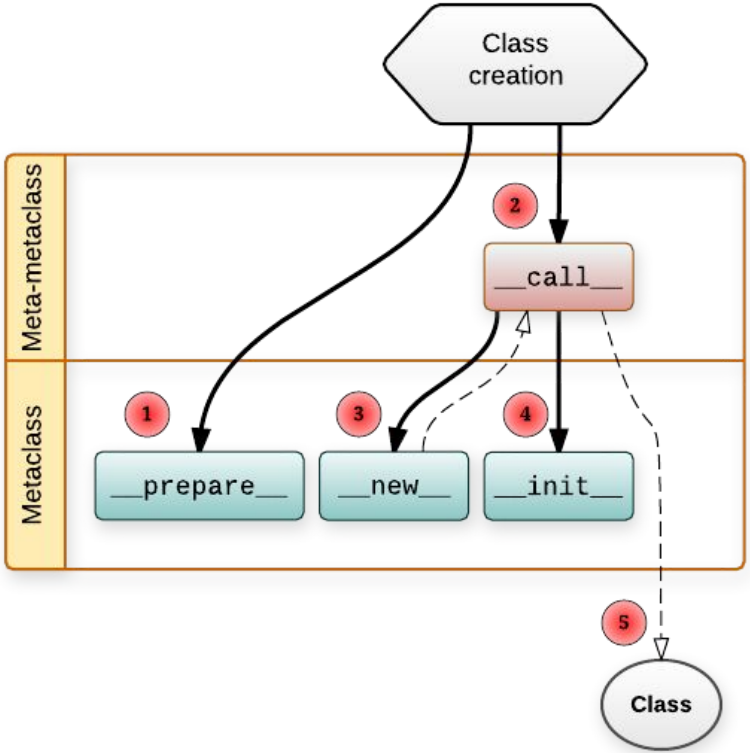
`classdict` – namespace класса (`__dict__`)

```
MyClass = type("MyClass", (), {})
```

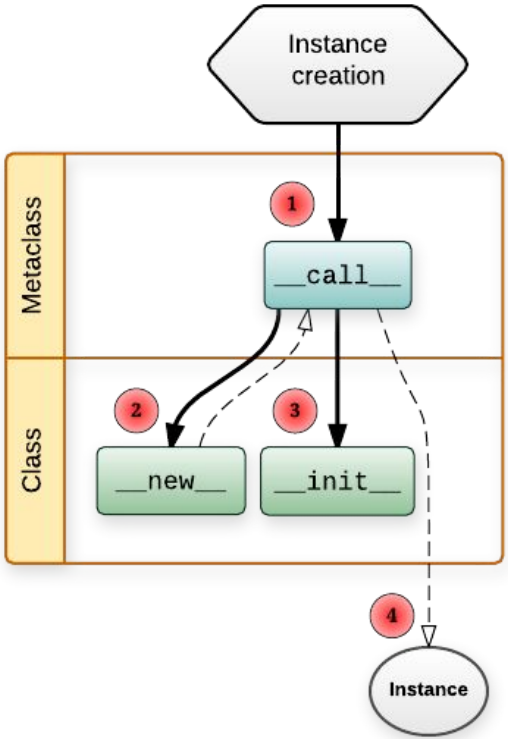
Метаклассы: type

```
>>> Bar = type('Bar', (Foo,), dict(attr=100))
>>> x = Bar()
>>> x.attr
100
>>> x.__class__
<class '__main__.Bar’>
>>> x.__class__.__bases__
(<class '__main__.Foo'>,)

>>> class Bar(Foo):
...     attr = 100
...
>>> x = Bar()
>>> x.attr
100
>>> x.__class__.__bases__
(<class '__main__.Foo'>,)
```

Метаклассы: создание класса



Метаклассы


Instance creation
Metaclass
1
__call__
Class
2
__new__
3
__init__
4
Instance

Метаклассы: создание класса

- определяются базовые классы
- определяется метакласс
- подготавливается namespace класса (`__prepare__`)
- выполняется тело класса
- создается класс (`__new__`, `__init__`)

Метаклассы

```
class AMeta(type):
    def __new__(mcs, name, bases, classdict, **kwargs):
        cls = super().__new__(mcs, name, bases, classdict)
        print('Meta __new__', cls)
        return cls

    def __init__(cls, name, bases, classdict, **kwargs):
        super().__init__(name, bases, classdict, **kwargs)

    def __call__(cls, *args, **kwargs):
        return super().__call__(*args, **kwargs)

    @classmethod
    def __prepare__(mcs, name, bases, **kwargs):
        print('Meta __prepare__', **kwargs)
        return {'b': 2, 'a': 2}
```

ABC

Добавляем абстракции

ABC

```
>>> from abc import ABCMeta, abstractmethod
>>> class C(metaclass=ABCMeta):
...     @abstractmethod
...     def abs_method(self):
...         pass
>>> c = C()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class C with abstract methods abs_method

>>> class B(C):
...     def abs_method(self):
...         print("Now a concrete method")
>>> b = B()
>>> b.abs_method()
Now a concrete method
```

ABC

```
class Hashable(metaclass=ABCMeta):
    __slots__ = ()
    @abstractmethod
    def __hash__(self):
        return 0

    @classmethod
    def __subclasshook__(cls, C):
        if cls is Hashable:
            return _check_methods(C, "__hash__")

        return NotImplemented
```

```
>>> from collections.abc import Hashable
>>> isinstance("123", Hashable)  # ???
>>> isinstance({}, Hashable)  # ???
```

Числа

float

```
>>> float("-inf"), float("inf"), float("nan")
(-inf, inf, nan)
>>> 0.1 + 0.2 == 0.3
False
>>> 0.1 + 0.2 <= 0.3
False
>>> 0.1 + 0.2
0.30000000000000004
>>> (0.1).as_integer_ratio()
(3602879701896397, 36028797018963968)
>>> format(0.1, ".25f")
'0.1000000000000000055511151'
>>> math.isclose(0.1 + 0.2, 0.3)
True
```

Decimal, Fraction

```
>>> from decimal import Decimal
>>> Decimal("0.1") + Decimal("0.2") == Decimal("0.3")
True
>>> Decimal(1) / Decimal(7)
Decimal('0.14285714285714285714285714285714')

>>> from fractions import Fraction
>>> Fraction(1, 10)
Fraction(1, 10)
>>> Fraction(1, 10) + Fraction(2, 10) == Fraction(3, 10)
True
```

Строки

str

- isalpha()
- isascii()
- isidentifier()
- isalnum()
- isdecimal()
- isdigit()
- isnumeric()

```
>>> s = "1¹²²³⁴⁴"
>>> s.isalnum()  # ???
>>> s.isdigit()  # ???
>>> s.isnumeric()  # ???
>>> s.isdecimal()  # ???
```

str

- `startswith(prefix[, start[, end]])`
- `endswith(suffix[, start[, end]])`
- `capitalize()`
- `upper()`
- `isupper()`
- `lower()`
- `islower()`
- `title()`
- `istitle()`
- `swapcase()`
- `isprintable()`
- `isspace()`

str

- count(sub[, start[, end]])
- find(sub[, start[, end]]), rfind(sub[, start[, end]])
- index(sub[, start[, end]]), rindex(sub[, start[, end]])
- replace(old, new[, count])
- center(width[, fillchar])
- encode(encoding='utf-8', errors='strict')
- expandtabs(tabsize=8)
- format(*args, **kwargs)
- ljust(width[, fillchar]) ,rjust(width[, fillchar])
- lstrip([chars]), strip([chars]), rstrip([chars])

str

- `split(sep=None, maxsplit=-1), rsplit(sep=None, maxsplit=-1)`
- `splitlines(keepends=False)`
- `partition(sep), rpartition(sep)`
- `join(iterable)`
- `zfill(width)`
- `removeprefix(prefix, /)`
- `removesuffix(suffix, /)`

collections

Специализированные контейнерные типы данных, предоставляющие альтернативы для встроенных `dict`, `list`, `set` и `tuple`

collections.namedtuple

namedtuple(typename, field_names, *, rename=False, defaults=None, module=None)

```
>>> Point = collections.namedtuple("Point", ["x", "y"])
>>> p = Point(11, y=22)  # p = (11, 22)
>>> p[0] + p[1]
33
>>> x, y = p
>>> x, y
(11, 22)
>>> p.x + p.y
33
>>> p._asdict()  # {'x': 1, 'y': 4}
```

collections.defaultdict

collections.defaultdict([default_factory[, ...]])

Словарь, у которого по умолчанию всегда вызывается функция **default_factory** .

```
>>> import collections
>>> defdict = collections.defaultdict(list)
>>> print(defdict)
defaultdict(<class 'list'>, {})
>>> for i in range(5):
...     defdict[i].append(i)
>>> print(defdict)
defaultdict(<class 'list'>, {0: [0], 1: [1], 2: [2], 3: [3], 4: [4]})
```

collections.OrderedDict

collections.OrderedDict([items])

Словарь, который помнит порядок, в котором ему были даны ключи.

```
>>> import collections
>>> d = collections.OrderedDict(
...     [("a", "A"), ("b", "B", ("c", "C")]
... )
>>> for k, v in d.items():
...     print(k, v)
'a', 'A'
'b', 'B'
'c', 'C'
>>> d.move_to_end("b")
```

collections.Counter

`collections.Counter([iterable-or-mapping])`

Словарь для подсчёта хешируемых объектов.

- `elements()`
- `most_common([n])`
- `subtract([iterable-or-mapping])`
- `update([iterable-or-mapping])`

```
>>> words = re.findall(r"\w+", open("hamlet.txt").read().lower())
>>> Counter(words).most_common(5)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669), ('i', 631)]
```

collections.deque

`collections.deque([iterable[, maxlen]])`

Двусторонняя очередь из итерируемого объекта с максимальной длиной `maxlen`.

Добавление и удаление элементов в начало или конец выполняется за константное время.

- `append(x)`
- `appendleft(x)`
- `extend(iterable)`
- `extendleft(iterable)`
- `insert(i, x)`
- `pop()/popleft()`
- `remove(value)`

```
>>> d = deque("ghi")
>>> d.append("j")
>>> d.appendleft("f")
>>> d
deque(['f', 'g', 'h', 'i', 'j'])
>>> d.pop()
'j'
>>> d.popleft()
'f'
>>> d
deque(['g', 'h', 'i'])
```

functools

Для функций высшего порядка

functools

- cache(user_function)
- cached_property(func)
- lru_cache(user_function)
- lru_cache(maxsize=128, typed=False)

```
@functools.cache
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)
```

singledispatch, singledispatchmethod(func)

```
@functools.singledispatch
def fun(arg, prefix="Hello"):
    print(f"{prefix} any type {arg}")

@fun.register(str)
def _(arg, prefix="Hi"):
    print(f"{prefix} str {arg}")

@fun.register
def _(arg: int, prefix="Hello"):
    print(f"{prefix} int {arg}")

>>> fun(123)  # Hello int 123
>>> fun.registry.keys()
# dict_keys([<class 'object'>, <class 'str'>, <class 'int'>])
```

functools

- partial(func, /, *args, **keywords)
- partialmethod(func, /, *args, **keywords)
- total_ordering
- reduce(function, iterable[, initializer])
- wraps(wrapped, assigned=WRAPPER_ASSIGNMENTS, updated=WRAPPER_UPDATES)
- update_wrapper(wrapper, wrapped, assigned=WA, updated=WU)

```
>>> basetwo = partial(int, base=2)
>>> basetwo("10010")
18
```

itertools

Можно бесконечно смотреть на бесконечный цикл

Итератор

Итератор представляет собой объект-перечислитель, который для данного объекта выдает следующий элемент, либо вызывает исключение, если элементов больше нет.

```
num_list = [1, 2, 3]
itr = iter(num_list)
print(next(itr))  # 1
print(next(itr))  # 2
print(next(itr))  # 3
print(next(itr))  # StopIteration
```

```
class SpecialIterator:
    def __init__(self, limit):
        self.limit = limit
        self.counter = 0
    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return self.counter
        else:
            raise StopIteration

iter_obj = SpecialIterator(3)
print(next(iter_obj))
```

Генератор

Генератор – функция, которая при вызове `next()` возвращает следующий объект по алгоритму ее работы. Вместо `return` в генераторе используем `yield` (или вместе).

```
def gen(count):
    while count > 0:
        yield count
        count -= 1
    return count  # будет аргументом StopIteration

for i in gen(5):
    print(i)  # 5, 4, 3, 2, 1
```

Itertools: бесконечные итераторы

- `count(start=0, step=1)`
- `cycle(iterable)`
- `repeat(object[, times])`

Itertools

- accumulate(iterable[, func, *, initial=None])
- chain(*iterables)
- compress(data, selectors)
- dropwhile(predicate, iterable)
- takewhile(predicate, iterable)
- filterfalse(predicate, iterable)
- groupby(iterable, key=None)
- islice(iterable, start, stop[, step])
- pairwise(iterable)
- starmap(function, iterable)
- tee(iterable, n=2)
- zip_longest(*iterables, fillvalue=None)

Itertools: комбинаторика

- `product(*iterables, repeat=1)`
- `permutations(iterable, r=None)`
- `combinations(iterable, r)`
- `combinations_with_replacement(iterable, r)`

Разное

Enum

```
from enum import Enum


class HttpStatus(enum.Enum):
    OK = 200
    NOT_FOUND = 404
    SERVER_ERROR = 500


print(HttpStatus.OK.value, HttpStatus.OK.name)  # (200, 'OK')


status = HttpStatus(200)

if status is HttpStatus.OK:
    print("OK")
```

Dataclasses

```
import dataclasses

@dataclasses.dataclass
class Point:
    x: int
    y: int
    vector: list[int] = dataclasses.field(default_factory=list)

p = Point(10, 20)
print(p.x, p.y, p.vector)  # (10, 20, [])

@dataclasses.dataclass(slots=True)
class PointSlots:
    x: int
    y: int

p = PointSlots(10, 20)
print(p)  # PointSlots(x=10, y=20)
```

Path

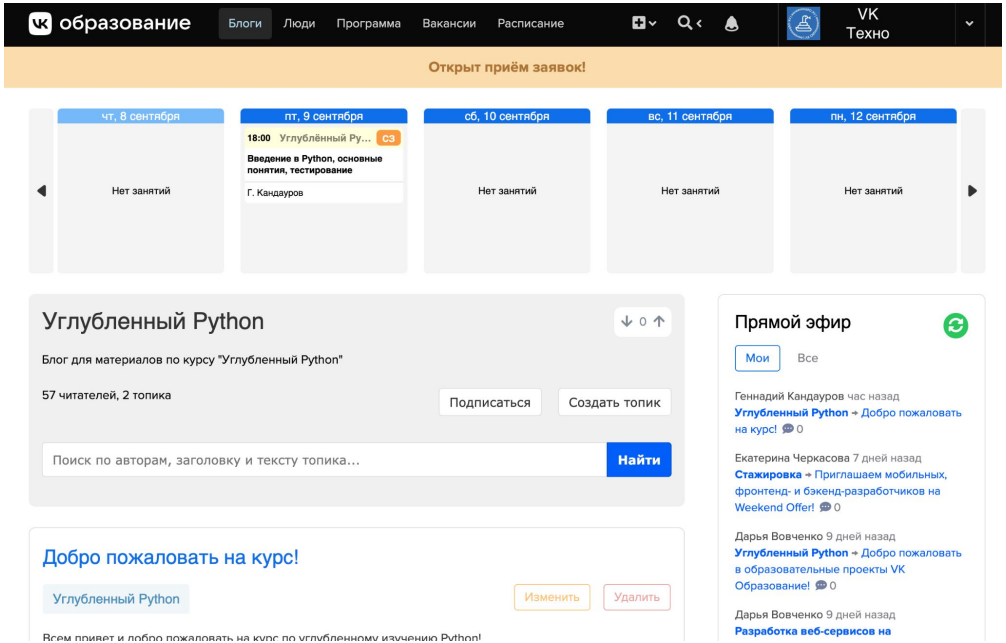
```
from pathlib import Path
p = Path(".")
pdf_path = p / "storage" / "slides.pdf"
abs_path = "/usr" / p / "storage" / "slides.pdf"

p.is_dir(), p.is_file()
p.is_absolute()
p.exists()
p.glob(pat)
p.open(), p.read_text(), p.write_text()
p.unlink()
```

Домашнее задание #05

- LRU cache без OrderedDict
- Тесты
- Зеленый пайплайн (тесты, coverage, линтеры)

Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции



Спасибо за внимание

education