# Углубленный Python Лекция 5 Стандартная библиотека

Кандауров Геннадий



#### Напоминание отметиться на портале

+ оставить отзыв



#### Квиз про прошлой лекции



### Содержание занятия

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

#### **ABC**

Добавляем абстракции

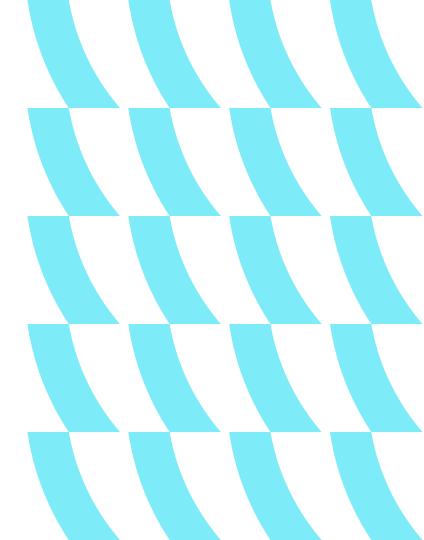
#### **ABC**

```
>>> 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
   aclassmethod
   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



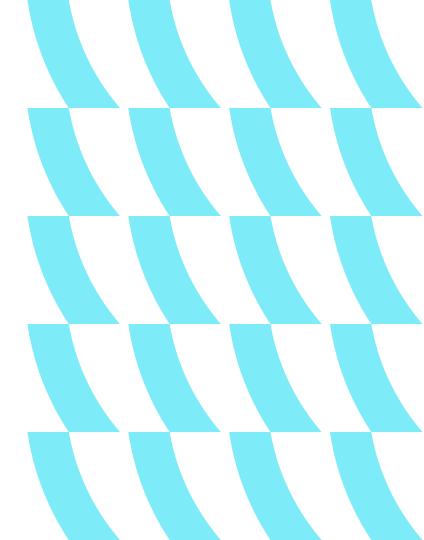
#### float

```
>>> 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('0.1428571428571428571428571429')
>>> from fractions import Fraction
>>> Fraction(1, 10)
Fraction(1, 10)
>>> Fraction(1, 10) + Fraction(2, 10) == Fraction(3, 10)
True
```

### Строки



```
isalpha()
    isascii()
    isidentifier()
    isalnum()
    isdecimal()
    isdigit()
    isnumeric()
>>> s = "1^{122344}"
>>> s.isalnum() # ???
>>> s.isdigit()
                 # ???
>>> s.isnumeric()
                   # ???
>>> s.isdecimal()
                   # ???
```

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

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

```
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"])
\Rightarrow 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.

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

```
>>> d = deque("qhi")
o append(x)
                                    >>> d.append("j")
appendleft(x)
                                    >>> d.appendleft("f")
                                    >>> d
o extend(iterable)
                                    deque(['f', 'g', 'h', 'i', 'j'])
o extendleft(iterable)
                                    >>> d.pop()
o insert(i, x)
                                    >>> d.popleft()
o pop()/popleft()
                                    ' f '
o remove(value)
                                    >>> d
                                    deque(['g', 'h', 'i'])
```

#### **functools**

Для функций высшего порядка

#### functools

```
o cache(user_function)
 cached property(func)
   lru cache(user function)
lru cache(maxsize=128, typed=False)
afunctools.cache
def factorial(n):
    if n <= 1:
       return 1
    return n * factorial(n - 1)
```

#### singledispatch, singledispatchmethod(func)

```
afunctools.singledispatch
def fun(arg, prefix="Hello"):
    print(f"{prefix} any type {arg}")
afun.register(str)
def (arg, prefix="Hi"):
    print(f"{prefix} str {arq}")
@fun.register
def (arg: int, prefix="Hello"):
    print(f"{prefix} int {arq}")
>>> fun(123) # Hello int 123
>>> fun.registry.keys()
# dict keys([<class 'object'>, <class 'str'>, <class 'int'>])
```

#### functools

```
o partial(func, /, *args, **keywords)

    partialmethod(func, /, *args, **keywords)

 total ordering
    reduce(function, iterable[, initializer])

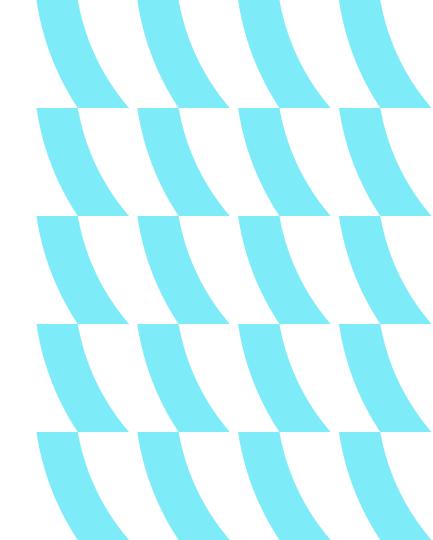
    wraps(wrapped, assigned=WRAPPER ASSIGNMENTS, updated=WRAPPER UPDATES)

    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 = \lceil 1, 2, 3 \rceil
                                       class SpecialIterator:
                                           def init (self, limit):
itr = iter(num list)
                                               self.limit = limit
                                               self.counter = 0
print(next(itr)) # 1
                                           def next (self):
print(next(itr)) # 2
                                               if self.counter < self.limit:</pre>
                                                   self.counter += 1
print(next(itr)) # 3
                                                   return self.counter
                                               else:
print(next(itr)) # StopIteration
                                                   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

```
o count(start=0, step=1)
```

- o cycle(iterable)
- o repeat(object[, times])

#### **Itertools**

accumulate(iterable[, func, \*, initial=None]) o chain(\*iterables) compress(data, selectors) dropwhile(predicate, iterable) takewhile(predicate, iterable) filterfalse(predicate, iterable) qroupby(iterable, key=None) islice(iterable, start, stop[, step]) o pairwise(iterable) starmap(function, iterable) o 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):
   0K = 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, [7])
adataclasses.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.qlob(pat)
p.open(), p.read text(), p.write text()
p.unlink()
```

#### heapq

```
heappush(heap, item)
heappop(heap)
heappushpop(heap, item)
heapify(x)
heapreplace(heap, item)
merge(*iterables, key=None, reverse=False)
nlargest(n, iterable, key=None)
nsmallest(n, iterable, key=None)
```

#### Домашнее задание #05

- LRU cache без OrderedDict
- Тесты
- flake8 + pylint перед сдачей

#### Hапоминание отметиться на портале Vol 2

+ оставить отзыв после лекции



## Спасибо за внимание

