collections

2019年10月27日 23:34

collection模块

1) defaultdict有默认值的字典

```
from collections import defaultdict

none_dict = defaultdict(lambda: None)
none_dict[10]  # add key=10, value=None
print(none_dict)
```

2) OrderDict有序字典

```
from collections import OrderedDict

normal_dict = {'a': 2, 'b': 1}
order_dict = OrderedDict(sorted(normal_dict.items(), key=lambda pair:
pair[-1]))
print(order_dict)
```

3) deque双向队列

```
from collections import deque

deque_list = deque()
```

等价于list，只是对于delete和append的效率更高

```
# deque_list = deque(maxlen=10)
```

还可以设置最大长度，一旦添加元素导致超出，另一端就会pop

4) namedtuple有字段名的tuple

```
from collections import namedtuple

Point = namedtuple('Point', 'x y')
point1 = Point(1, 2)
```

5) Counter词频统计

```
import re
from collections import Counter

terms = re.split(r'\s+', 'this is a python script')
print(terms)  # ['this', 'is', 'a', 'python', 'script']
counter = Counter(terms)
print(counter)  # Counter({'this': 1, 'is': 1, 'a': 1, 'python': 1, 'script': 1})
```

*ChainMap非常鸡肋，可以不管

operator

2020年5月29日 14:08

operator库包含了所有的运算

重要的2个函数。attrgetter和itemgetter的速度略快于lambda函数

- attrgetter，类似于getattr函数

```
class Teacher():
   def __init__(self, name, salary, age):
      self.name = name
      self.age  = age
      self.salary = salary
   def __repr__(self):
      return  repr((self.name,self.age,self.salary))

teachers = [
Teacher("A",1200,30),
Teacher("B",1200,31),
Teacher("C",1300,30)
]

from operator import attrgetter
print(sorted(teachers,key=attrgetter("age")))  # 根据age排序
print(sorted(teachers,key=attrgetter("salary","age")))  # 根据salary和age排序
```

结果：

```
[('A', 30, 1200), ('C', 30, 1300), ('B', 31, 1200)]
[('A', 30, 1200), ('B', 31, 1200), ('C', 30, 1300)]
```

- itemgetter，类似于__getitem__函数

```
>>> itemgetter(1)('ABCDEFG')
'B'
>>> itemgetter(1,3,5)('ABCDEFG')
('B', 'D', 'F')
>>> itemgetter(slice(2,None))('ABCDEFG')
'CDEFG'
>>> soldier = dict(rank='captain', name='dotterbart')
>>> itemgetter('rank')(soldier)
'captain'

>>> inventory = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]
>>> getcount = itemgetter(1)
>>> list(map(getcount, inventory))
[3, 2, 5, 1]
>>> sorted(inventory, key=getcount)
[('orange', 1), ('banana', 2), ('apple', 3), ('pear', 5)]
```

Pathlib

2020年5月29日 14:37

```
    p = Path()
    p.resolve()  # current path
    p = Path('../')
    p / Path('_itertools.py')  # concate file path
    p.resolve()  # current path with path, '/**/
_itertools.py'
    p.name  # file name without path, '_itertools.py'
    p.stem  # file name without suffix, '_itertools'
    p.suffix  # file suffix, '.py'
    p.parent  # os.dirname
    p.parents  # all parent directions
    p.parts  # split with '/'
    p.exists()  # = os.path.exist()
    p.is_file()  # = os.path.isfile()
    p.is_dir()  # = os.path.isdir()
    p.iterdir()  # = os.listdir
    p.stat()  # file info
    p.stat().st_size  # file size
    p.glob(pattern='*')  # = os.listdir, like glob.glob
    p.rglob(pattern='*.py')  # all files with recursion
    p.mkdir(exist_ok=True, parents=True)  # =
os.makedirs()
    Path('~/').expanduser()  # = os.path.expanduser()
```

itertools

2020年5月29日 16:07

itertools模块-迭代器模块

迭代器是实时产生的，不占用存储空间，耗时。但是安全

1）合并2个序列

```
import itertools

list1 = [0, 1, 2, 3]
list2 = [0, 1, ]
combination = itertools.chain(list1, list2)
print(list(combination))  # [0, 1, 2, 3, 0, 1]
```

2）创造多个序列的迭代器

```
a = itertools.chain.from_iterable([list1, list2])
print(a.__next__())
```

3）迭代器，用于for/while循环，只能用break跳出循环

```
itertools.count(start=1, step=2)
```

4）筛选条件为false的数据

```
print(list(itertools.filterfalse(None, [1, 0])))  # [0]
print(list(itertools.filterfalse(lambda x: x < 2, range(5))))  # [2, 3, 4]
```

5）筛选条件为True的数据

```
print(list(itertools.compress(['a', 'b', 'c'], [True, False, False])))  # ['a']
```

6）itertools.cycle(range(10))重复迭代器

7）映射函数，类似于map

```
print(list(itertools.starmap(max, [(1, 2), (2, 3)])))  # [2, 3]
```

8）重复函数

```
itertools.repeat(object, times)
```

9）笛卡尔积组合。将不同迭代器的**按照次序**组合而成，repeat是重复的次数，默认为1

```
print(list(itertools.product('ab', '12', repeat=1)))
# [('a', '1'), ('a', '2'), ('b', '1'), ('b', '2')]
print(list(itertools.product('ab', '12', repeat=2)))
# ('a', '1', 'a', '1'), ('a', '1', 'a', '2'), ('a', '1', 'b', '1'), ('a', '1', 'b', '2'), \
# ('a', '2', 'a', '1'), ('a', '2', 'a', '2'), ('a', '2', 'b', '1'), ('a', '2', 'b', '2'), \
# ('b', '1', 'a', '1'), ('b', '1', 'a', '2'), ('b', '1', 'b', '1'), ('b', '1', 'b', '2'), \
# ('b', '2', 'a', '1'), ('b', '2', 'a', '2'), ('b', '2', 'b', '1'), ('b', '2', 'b', '2')]
```

10）随机组合，随机选择迭代器中的若干元素，组合（给出所有组合方式）。

```
print(list(itertools.permutations(range(3), 2)))
# [(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]
```

11）一个迭代器的子序列

```
print(list(itertools.combinations(range(3), 2)))  # [(0, 2), (0, 2), (1, 2)]
```

Dict
in > __contains__

functools

2020年5月29日 16:19

functools模块

1) partial函数，用于生成新的函数

```
from functools import partial

def sum(a, b, c):
    return a + b + c

add = partial(sum, 1)  # set a=1, and make a new function
print(add(2, 3))  # set b=2, c=3
```

2) reduce累计运算，定义运算方式

```
from functools import reduce

numbers = list(range(1, 10))
print(numbers)  # [1, 2, 3, 4, 5, 6, 7, 8, 9]
result = reduce(lambda x, y: x + y, numbers)
print(result)  # 45
```

3) total_ordering，定义比较函数。必须定义2个比较函数

1. __eq__
2. __lt__, __le__, __gt__, __ge__中的一个

```
from functools import total_ordering


@total_ordering
class Person:
    def __eq__(self, other):
        return ((self.lastname.lower(), self.firstname.lower()) ==
                (other.lastname.lower(), other.firstname.lower()))

    def __lt__(self, other):
        return ((self.lastname.lower(), self.firstname.lower()) <
                (other.lastname.lower(), other.firstname.lower()))


p1 = Person()
p2 = Person()

p1.lastname = "123"
p1.firstname = "000"

p2.lastname = "1231"
```

```
p2.firstname = "000"

print(p1 < p2)
print(p1 <= p2)
print(p1 == p2)
print(p1 > p2)
print(p1 >= p2)
```

4) lru_cache给函数的输入/输出，建立一个缓冲池，加速处理

```
@lru_cache(maxsize=None)
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)
```