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], if None, it is bool function
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)
itertools.repeat(range(3), times=2) # [range(0, 3), range(0, 3)]
```

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（给出所有不重复的组合方式）。

```
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)]
```

12) 根据函数的输出分类

```
import itertools
for key, group in itertools.groupby('AaaBBbcCAAa', lambda c: c.upper()):
    print(list(group))

['A', 'a', 'a']
['B', 'B', 'b']
['c', 'C']
['A', 'A', 'a']
```

13) 累func（加，默认+法）求和

```
import itertools
from operator import add

out = itertools.accumulate(range(3), func=add)
print(list(out)) # [0, 1, 3]
```

14) 删除迭代器的内容

```
out = itertools.dropwhile(lambda x: x < 2, range(3))
print(list(out)) # [2]
```

15）深复制若干个迭代器

```
a,b = itertools.tee(range(3), 2)
```

16) zip迭代器，不足用default value代替

```
itertools.zip_longest(range(5), range(3), fillvalue=None)
```

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)
```

bytes

2020年6月7日 13:29

str-->encode()-->bytes-->decode()-->str
var = '123' # <class 'str'>, '123'
print(var)
bvar = var.encode() # <class 'bytes'>, b'123'
print(bvar)
bvar.decode() # btypes => string
bvar = b'123' # 定义二进制字符串，此种方式只允许是ASCII字符

bytes是不可变的，同str。bytearray是可变的，同list

typing

2020年6月7日 14:47

```
# Type aliases
# '_' at head denotes private, '_t' at the end denotes 'type'
_VECTOR_t = List[float]
array: _VECTOR_t = [1.0]
array: _VECTOR_t = ['sad']  # Expected type 'List[float]', got 'List[str]' instead

# NewType
_TENSOE_t = NewType('_TENSOE_t', List)
tensor: _TENSOE_t = _TENSOE_t('1')  # runnable, tensor is still str type, so we dont recommend

# Callable: similar to define a function
# e.g. Callable[[Arg1Type, Arg2Type],ReturnType]
max2 = lambda x, y: max(x, y)


def feed(func: Callable[[float, float], float], args: List):
    return func(*args)


feed(max2, [1.2, 2.2])

# Generics, e.g. Mapping=Dict, Sequence=[List, Tuple]
_Str2Int_DICT_t = Mapping[str, int]
dict: _Str2Int_DICT_t = {'1': 2}
dict: _Str2Int_DICT_t = {1: 2}  # Expected type 'Mapping[str, int]', got 'Dict[int, int]' instead

# TypeVar
T = TypeVar('T')  # Can be anything
A = TypeVar('A', str, bytes)  # Must be str or bytes
var: A = 1  # Expected type 'A', got 'int' instead
var: A = '123'


'''
在实际使用中， Any, Union, Tuple, List, Sequence, Mapping, Callable, TypeVar,Optional, Generic等的使用频率比较高，其中Union、Optional、Sequence、Mapping非常有用，注意掌握。
Union, 即并集，所以Union[X, Y] 意思是要么X类型、要么Y类型
Optional, Optional[X]与Union[X, None]，即它默认允许None类型
Sequence, 即序列，需要注意的是，List一般用来标注返回值；Sequence、Iterable用来标注参数类型
Mapping, 即字典，需要注意的是，Dict一般用来标注返回值；Mapping用来标注参数类型
Iterable,Iterator
'''
```

importlib

2020年6月7日 14:59

```
def dynamic_import_libs(module):
    return importlib.import_module(module)


module = '_python._typing'
_typing = dynamic_import_libs(module)
```

可以用_typing调用文件中的元素

struct

2020年6月7日 15:10

将Python数据写为C的二进制数据

格式符	C语言类型	Python类型	Standard size
x	pad byte(填充字节)	no value	
c	char	string of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I(大写的i)	unsigned int	integer	4
l(小写的L)	long	integer	4
L	unsigned long	long	4
q	long long	long	8
Q	unsigned long long	long	8
f	float	float	4
d	double	float	8
s	char[]	string	
p	char[]	string	
P	void *	long	

对齐

Character	Byte order	Size	Alignment
@(默认)	本机	本机	本机,凑够4字节
=	本机	标准	none,按原字节数
<	小端	标准	none,按原字节数
>	大端	标准	none,按原字节数
!	network(大端)	标准	none,按原字节数

```
import struct

'''
struct.pack(format, v1, v2, ...)
返回一个 bytes 对象，其中包含根据格式字符串 format 打包的值 v1, v2, ... 参数个数必须与格式字符串所要求的值完全匹配。

struct.unpack(format, buffer)
根据格式字符串 format 从缓冲区 buffer 解包（假定是由 pack(format, ...) 打包）。 结果为一个元组，即使其只包含一个条目。 缓冲区的字节大小必须匹配格式所要求的大小，如 calcsize() 所示。
'''

# this code fragment is tu parse 2 int variables into a buffer
var1, var2 = 10, 20
# 'ii' is for var1 and var2, 'i' stands for int type
buffer = struct.pack('ii', var1, var2)  # b'\n\x00\x00\x00\x14\x00\x00\x00'
```

```
print(struct.calcsize('ii'))  # occupy 8 bytes
struct.unpack('ii', buffer)  # (10, 20)
# print(struct.unpack('in', buffer))  # struct.error: unpack requires a buffer of 16 bytes

# 对齐
# 小端：较高的有效字节存放在较高的存储器地址，较低的有效字节存放在较低的存储器地址。
# 大端：较高的有效字节存放在较低的存储器地址，较低的有效字节存放在较高的存储器地址。
buffer = struct.pack('ii', var1, var2)  # b'\n\x00\x00\x00\x14\x00\x00\x00'
left_buffer = struct.pack('<ii', var1, var2)  # b'\n\x00\x00\x00\x14\x00\x00\x00'
print(struct.unpack('<ii', left_buffer))  # (10, 20)
print(struct.unpack('>ii', left_buffer))  # (167772160, 335544320)
right_buffer = struct.pack('>ii', var1, var2)  # b'\x00\x00\x00\n\x00\x00\x00\x14'
print(struct.unpack('<ii', right_buffer))  # (167772160, 335544320)
print(struct.unpack('>ii', right_buffer))  # (10, 20)

record = b'raymond   \x32\x12\x08\x01\x08'
print(struct.unpack('<10sHHb', record))  # left, 10s: len(string)=10, H: unsigned short, b: signed char

buffer = struct.pack('llh0l', 1, 2, 3)  # llh0l=l+l+h0l, h0l表示用0填充h至l长度，只能为0
print(buffer)  # b'\x01\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x03\x00\x00
\x00\x00\x00\x00\x00'
print(len(buffer))  # 24
```

另外2个pack_into和unpakc_into是对于buffer的操作，不能直接作用于struct.pack生成的buffer（readable-only），再次不做赘述