小强期中考试（考察1-9章）

142 可变类型和不可变类型分别列举 3 个

可变类型：mutable type，常见的有 ： list, dict, set, deque 等

不可变类型：immutable type,，常见的有：int, float, str, tuple, frozenset 等

只有不可变类型才能作为字典等的键。

143 容量为 100 的样本，怎样使用 list 实现随机抽样 10 个

使用 random 模块中 randint 和 sample 方法，使用列表生成式：

复制
```
from random import randint,sample
lst = [randint(0,50) for _ in range(100)]
lst_sample = sample(lst,10)
print(lst_sample)
```

144 实现文件按行读取和操作数据分离功能

使用 `yield` 解耦按行读取和操作数据的两步操作

复制
```
def read_line(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f:
            yield line

def process_line(line:str):
    pass

for line in read_line('.'):
    process_line(line)
```

145 找出列表中的所有重复元素

遍历列表，如果出现次数大于1，且不在返回列表 `ret` 中，则添加到 `ret` 中。

满足 `x not in ret` ，则表明 `x` 不在列表中。

复制
```
def find_duplicate(lst):
    ret = []
    for x in lst:
        if lst.count(x) > 1 and x not in ret:
            ret.append(x)
    return ret
```

调用 `find_duplicate`：

复制
```
r = find_duplicate([1, 2, 3, 4, 3, 2])
print(r)
```

结果：[2, 3]

146 斐波那契数列

斐波那契数列第一、二个元素都为 1，第三个元素等于前两个元素和，依次类推。下面是普通实现版本：

复制
```
def fibonacci(n):
    if n <= 1:
        return [1]
    fib = [1, 1]
    while len(fib) < n:
        fib.append(fib[len(fib) - 1] + fib[len(fib) - 2])
    return fib
```

调用 `fibonacci`：

复制
```
r = fibonacci(5)
print(r)
```

结果：[1, 1, 2, 3, 5]，这不是高效的实现，使用生成器更节省内存。

147 出镜最多的元素

`max` 函数是 Python 的内置函数，所以使用它无需 `import`

`max` 有一个 `key` 参数，指定如何进行值得比较。

下面案例，求出现频次最多的元素，当出镜最多的元素有多个时，按照下面方法，默认只返回一个。

复制
```
def mode(lst):
    if lst is None or len(lst)==0:
        return None
    return max(lst, key=lambda v: lst.count(v))
```

调用 `mode`：

复制
```
lst = [1, 3, 3, 2, 1, 1, 2]
r = mode(lst)
print(f'{lst}中出现次数最多的元素为:{r}')
```

结果：

[1, 3, 3, 2, 1, 1, 2]中出现次数最多的元素为: 1

148 更长列表

带有一个 `*` 的参数为可变的位置参数，意味着能传入任意多个位置参数。

`key` 函数定义怎么比较大小；

`lambda` 的参数 `v` 是 `lists` 中的一个元素。

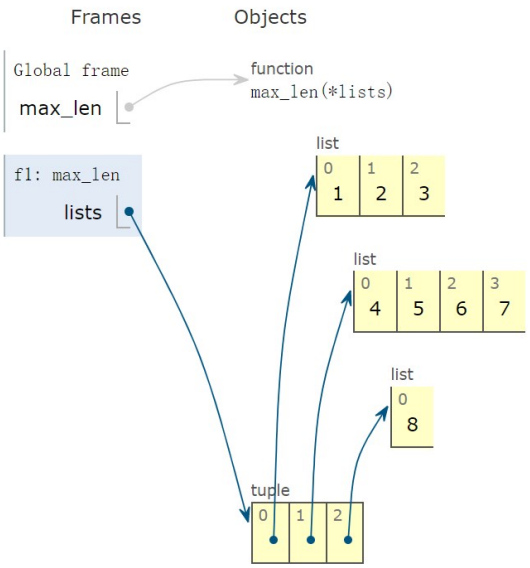
```
In [15]: def max_len(*lists):
    ...:     return max(*lists, key=lambda v: len(v))
```

调用 `max_len` ，传入三个列表，正是 `v` 可能的三个取值。

```
In [17]: r = max_len([1, 2, 3], [4, 5, 6, 7], [8])
    ...: print(f'更长的列表是{r}')
更长的列表是[4, 5, 6, 7]
```

关于 lambda 函数，在此做图形演示。

`max_len` 函数被传入三个实参，类型为 `list` ，如下图所示，`lists` 变量 指向最下面的 tuple 实例。

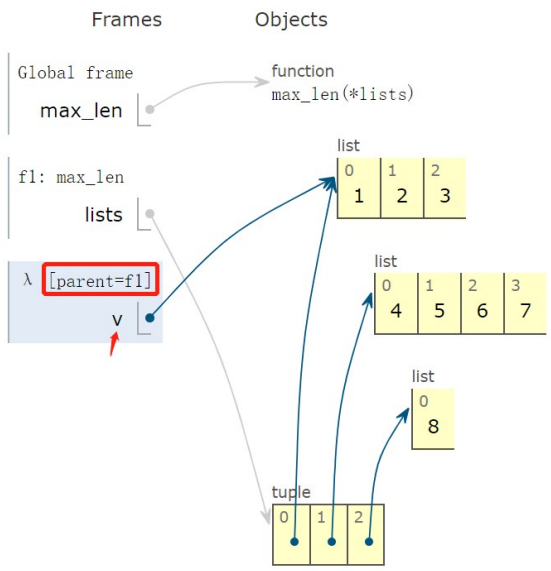


程序运行到下一帧，会出现 lambda 函数，它的父函数为 f1, 也就是 max_len 函数。

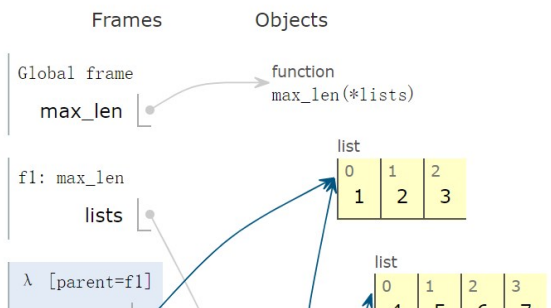
有些读者可能不理解两点，这种用法中：

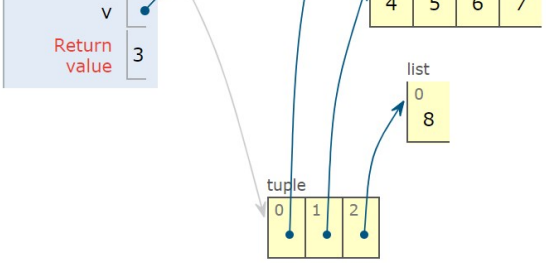
- 参数 `v` 取值到底是多少？
- lambda 函数有返回值吗？如果有，返回值是多少？

通过下面图形，非常容易看出，`v` 指向 tuple 实例的第一个元素，指向的线和箭头能非常直观的反映出来。

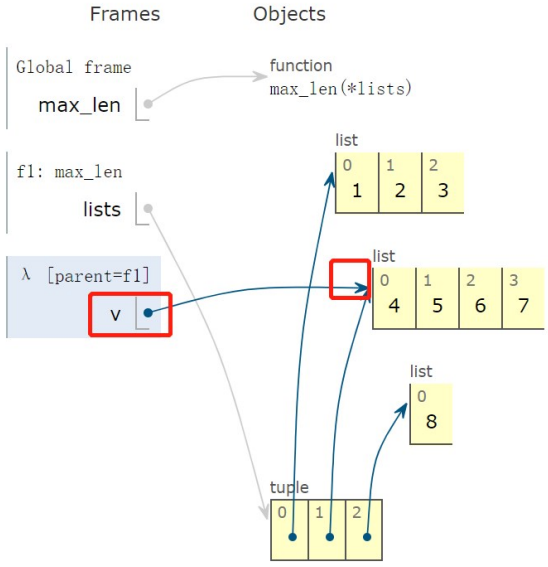


下面示意图中，看到返回值为 3，也就是 len(v) 的返回值，其中 v = [1,2,3]。

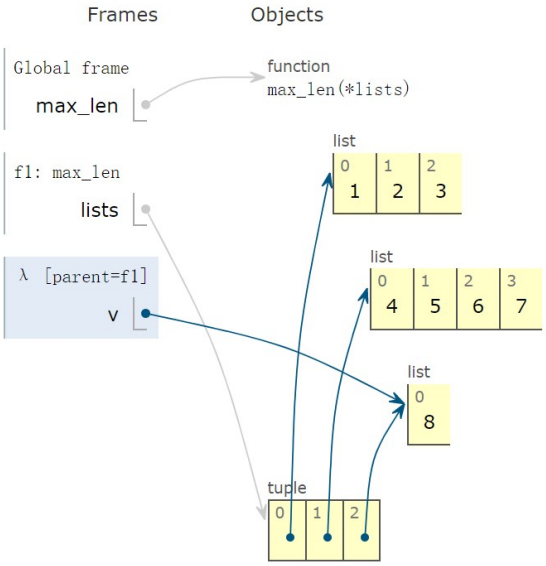




然后，v 指向 tuple 中的下一个元素，返回值为 4.

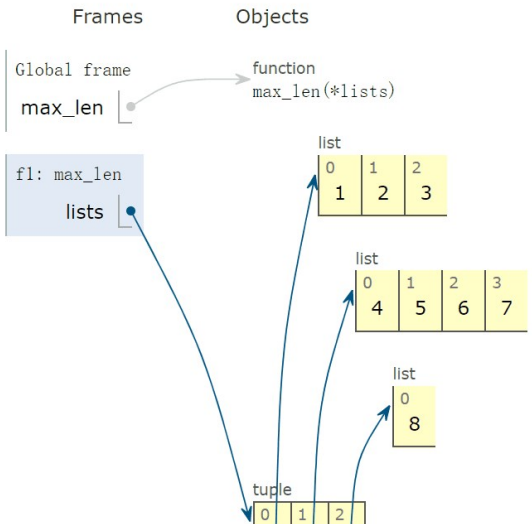


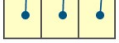
然后，v 指向 tuple 的最后一个元素 [8]，返回值为 1.



根据 key 确定的比较标准，max 函数的返回值为红色字体指向的元素，也就是返回 [4,5,6,7].

完整动画演示：





149 重洗数据集

内置 `random` 中的 `shuffle` 函数，能实现对数据的重洗。

值得注意，`shuffle` 是对输入列表就地（`in place`）洗牌，节省存储空间。

复制

```
In [34]: from random import shuffle
    ...: lst = [randint(0,50) for _ in range(100)]
    ...: shuffle(lst)
    ...: print(lst[:5])
[22, 49, 34, 9, 38]
```

150 生成满足均匀分布的坐标点

random模块，`uniform(a,b)` 生成 `[a,b)` 内的一个随机数。

如下，借助列表生成式，生成 100 个均匀分布的坐标点。

复制

```
from random import uniform
 x, y = [i for i in range(100)], [
        round(uniform(0, 10), 2) for _ in range(100)]
print(y)

[3.09, 9.02, 1.87, 1.43, 4.25, 9.66, 9.11, 0.12, 3.3, 2.35, 0.15, 0.34, 6
.47, 9.47, 8.63, 8.41, 6.02, 2.87, 5.93, 2.29, 8.61, 4.71, 6.87, 7.42, 0.
53, 3.04, 6.02, 7.51, 0.3, 0.27, 6.6, 7.48, 9.96, 8.05, 0.09, 1.07, 6.77,
6.98, 5.96, 2.1, 0.32, 9.12, 2.11, 0.45, 9.74, 3.33, 9.72, 8.4, 0.77, 9.75
, 4.61, 5.37, 1.59, 7.8, 0.88, 1.2, 8.21, 8.06, 8.97, 2.02, 8.32, 8.56, 5
.74, 5.03, 0.65, 5.84, 3.67, 5.38, 8.13, 2.0, 0.75, 1.92, 5.27, 0.65, 8.29
, 9.28, 3.28, 3.6, 1.76, 4.44, 1.15, 9.89, 1.65, 6.75, 0.62, 4.97, 7.03,
6.48, 8.14, 1.8, 7.41, 7.06, 3.73, 5.37, 7.17, 6.94, 0.29, 4.36, 6.
17]
```

使用 `pyecharts` 绘图，版本1.6.2.

注意，运行以下代码至少保证版本要在 1.0 以上：

复制

```
from pyecharts.charts import Scatter
import pyecharts.options as opts
from random import uniform

def draw_uniform_points():
    x, y = [i for i in range(100)], [
        round(uniform(0, 10), 2) for _ in range(100)]
    print(y)
    c = (
        Scatter()
        .add_xaxis(x)
        .add_yaxis('y', y)
    )
    c.render()


draw_uniform_points()
```

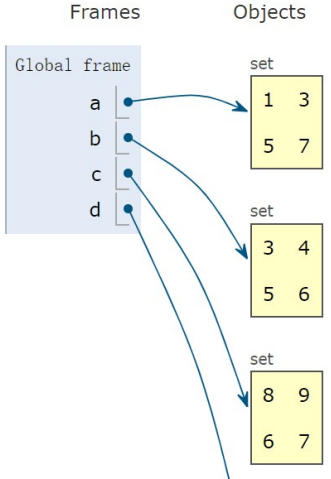
得到结果如下，变量 `y` 取值满足均匀分布。

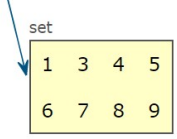
151 如何求两个集合的并集？

求并集：

复制

```
a = {1,3,5,7}
b, c = {3,4,5,6}, {6,7,8,9}
d = a.union(b,c) # {1, 3, 4, 5, 6, 7, 8, 9}
```



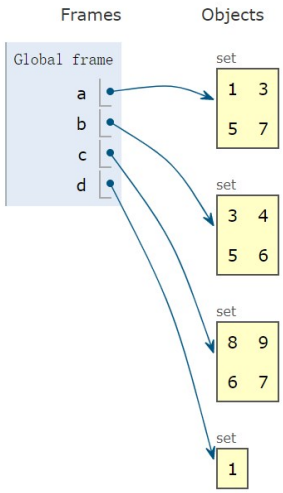


152 求两个集合的差集

求差集：

复制

```
a = {1,3,5,7}
b, c = {3,4,5,6}, {6,7,8,9}
d = a.difference(b,c) # {1}
```

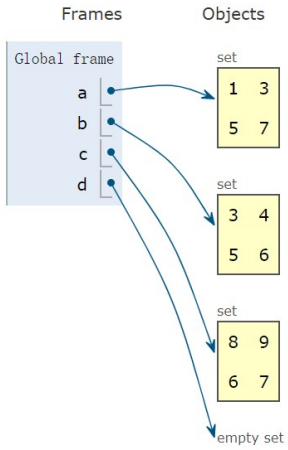


153 求两个集合的交集

求交集：

复制

```
a = {1,3,5,7}
b, c = {3,4,5,6}, {6,7,8,9}
d = a.intersection(b,c) # {}
```

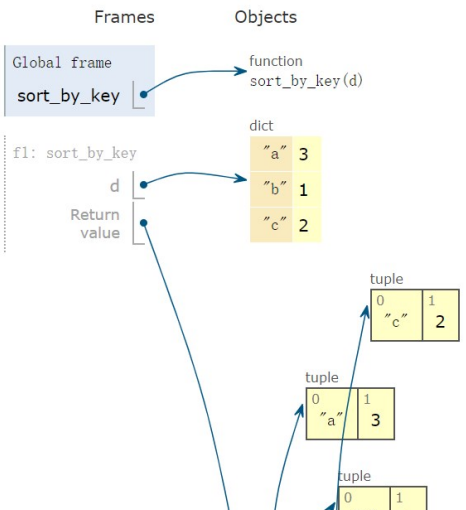


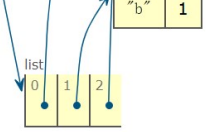
154 字典如何按键排序

复制

```
In [54]: def sort_by_key(d):
    ...:     return sorted(d.items(),key=lambda x: x[0])
    ...:

In [55]: sort_by_key({'a':3,'b':1,'c':2})
Out[55]: [('a', 3), ('b', 1), ('c', 2)]
```

`sorted` 函数返回列表，元素为 `tuple`：





155 字典如何按值排序

与按照键排序原理相同，按照值排序时，`key` 函数定义为按值(`x[1]`)比较。

为照顾小白，解释为什么是 `x[1]` ，`d.items()` 返回元素为 `(key, value)` 的可迭代类型(`Iterable`)，`key` 函数的参数 `x` ，便是元素 `(key, value)` ，所以 `x[1]` 取到字典的值。

复制

```
In [59]: def sort_by_value(d):
    ...:     return sorted(d.items(),key=lambda x: x[1])
    ...:

In [60]: sort_by_value({'a':3,'b':1,'c':2})
Out[60]: [('b', 1), ('c', 2), ('a', 3)]
```

156 如何获取字典的最大键

通过 `keys` 拿到所有键，获取最大键，返回 `(最大键, 值)` 的元组

复制

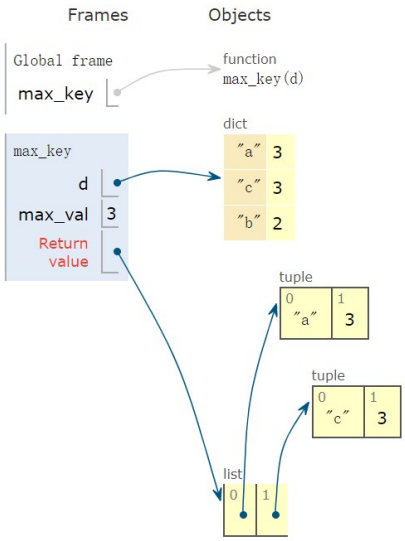
```
In [68]: def max_key(d):
    ...:     if len(d)==0:
    ...:         return []
    ...:     max_key = max(d.keys())
    ...:     return (max_key,d[max_key])

In [69]: max_key({'a':3,'c':3,'b':2})
Out[69]: ('c', 3)
```

157 最大字典值

最大值的字典，可能有多对：

复制

```
In [70]: def max_key(d):
    ...:     if len(d)==0:
    ...:         return []
    ...:     max_val = max(d.values())
    ...:     return [(key,max_val) for key in d if d[key]==max_val]
    ...:

In [71]: max_key({'a':3,'c':3,'b':2})
Out[71]: [('a', 3), ('c', 3)]
```



158 集合最值

找出集合中的最大、最小值，并装到元组中返回：

复制

```
In [76]: def max_min(s):
    ...:     return (max(s),min(s))

In [77]: max_min({1,3,5,7})
Out[77]: (7, 1)
```

`

159 是否为单字符串判断

若组成字符串的所有字符仅出现一次，则被称为单字符串。

复制

```
In [73]: def single(string):
    ...:     return len(set(string)) == len(string)

In [74]: single('love_python') # False
Out[74]: False

In [75]: single('python') # True
Out[75]: True
```

160 更长集合

`key` 函数定义为按照元素长度比较大小，找到更长的集合：

复制

```
In [78]: def longer(s1,s2):
```

```
    ...:     return max(s1,s2, key=lambda x: len(x))
    ...:

In [79]: longer({1,3,5,7},{1,5,7}) # {1,3,5,7}
Out[79]: {1, 3, 5, 7}
```

161 在两个列表中，找出重叠次数最多的元素

默认只返回一个

解决思路：

- 1 求两个列表的交集
- 2 遍历交集列表中的每一个元素，`min(元素在列表1次数, 元素在列表2次数)` ，就是此元素的重叠次数
- 3 求出最大的重叠次数

复制

```
In [80]: def max_overlap(lst1,lst2):
    ...:     overlap = set(lst1).intersection(lst2)
    ...:     ox = [(x,min(lst1.count(x), lst2.count(x)))  for x in overla
p]
    ...:     return max(ox, key=lambda x: x[1])

In [81]: max_overlap([1,2,2,2,3,3],[2,2,3,2,2,3])
Out[81]: (2, 3)
```

`

162 topn 键

找出字典前 n 个最大值，对应的键。

导入 Python 内置模块 `heapq` 中的 `nlargest` 函数，获取字典中的前 n 个最大值。

`key` 函数定义按值比较大小

复制

```
In [82]: from heapq import nlargest

In [83]: def topn_dict(d, n):
    ...:     return nlargest(n, d, key=lambda k: d[k])

In [84]: topn_dict({'a': 10, 'b': 8, 'c': 9, 'd': 10}, 3)
Out[84]: ['a', 'd', 'c']
```

163 一键对多值的字典

一键对多个值的实现方法1，按照常规思路，循序渐进：

复制

```
In [85]: d = {}
    ...: lst = [(1,'apple'),(2,'orange'),(1,'compute')]
    ...: for k,v in lst:
    ...:     if k not in d:
    ...:         d[k]=[]
    ...:     d[k].append(v)

In [86]: d
Out[86]: {1: ['apple', 'compute'], 2: ['orange']}
```

以上方法，有一处 `if` 判断 ，确认 `k` 是不是已经在返回结果字典 `d` 中。

不是很优雅！

可以使用 collections 模块中的 `defaultdict` ，它能创建属于某个类型的自带初始值的字典。

使用起来更加方便：

复制

```
In [87]: from collections import defaultdict
    ...:
    ...: d = defaultdict(list)
    ...: for k,v in lst:
    ...:     d[k].append(v)

In [88]: d
Out[88]: defaultdict(list, {1: ['apple', 'compute'], 2: ['orange']})
```

164 ChainMap 如何实现字典的逻辑合并

案例 3 中合并字典的方法：

复制

```
In [94]: dic1 = {'x': 1, 'y': 2 }
In [95]: dic2 = {'y': 3, 'z': 4 }
In [96]: merged = {**dic1, **dic2}

In [97]: merged
Out[97]: {'x': 1, 'y': 3, 'z': 4}
```

修改 `merged['x']=10` ，dic1 中的 x 值不变， `merged` 是重新生成的一个`新字典`。

但是，`collections` 模块中的 `ChainMap` 函数却不同。

它在内部创建了一个容纳这些字典的列表。

使用 `ChainMap` 合并字典，修改 `merged['x']=10` 后，dic1 中的 x 值改变。

如下所示：

复制

```
In [98]: from collections import ChainMap

In [94]: dic1 = {'x': 1, 'y': 2 }
In [95]: dic2 = {'y': 3, 'z': 4 }

In [99]: merged = ChainMap(dic1,dic2)

In [100]: merged
Out[100]: ChainMap({'x': 1, 'y': 2}, {'y': 3, 'z': 4})

In [101]: merged['x'] = 10
```

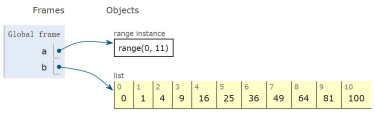
```
In [102]: dic1 # 改变, 共用内存的结果
Out[102]: {'x': 10, 'y': 2}
```

165 数据再运算

如下，实现对每个元素的乘方操作后，利用列表生成式返回一个新的列表。

复制

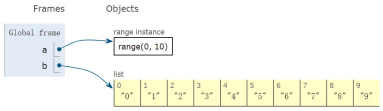
```
In [1]: a = range(0,11)

In [2]: b = [x**2 for x in a] # 利用列表生成式创建列表

In [3]: b
Out[3]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```



数值型的元素列表，转换为字符串类型的列表：

复制

```
In [1]: a = range(0,10)

In [2]: b = [str(i) for i in a]

In [3]: b
Out[3]: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

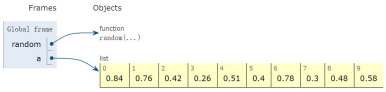


166 Python 生成一串随机数

生成 10 个 0 到 1 的随机浮点数，保留小数点后两位：

复制

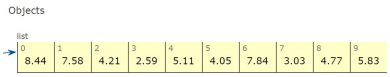
```
In [6]: from random import random

In [7]: a = [round(random(),2) for _ in range(10)]
```



生成 10 个 0 到 10 的满足均匀分布的浮点数，保留小数点后两位：

复制

```
In [16]: from random import uniform

In [10]: a = [round(uniform(0,10),2) for _ in range(10)]
```

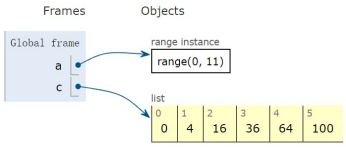


167 if 和嵌套 for

对一个列表里面的数据筛选，只计算 [0,11) 中偶数的平方：

复制

```
In [10]: a = range(11)

In [11]: c = [x**2 for x in a if x%2==0]

In [12]: c
Out[12]: [0, 4, 16, 36, 64, 100]
```



列表生成式中嵌套 for

如下使用嵌套的列表，一行代码生成 99 乘法表的所有 45 个元素：

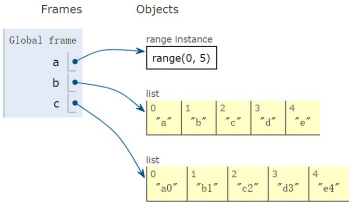
复制

```
In [9]: a = [i*j for i in range(10) for j in range(1,i+1)]
```

168 zip 和列表

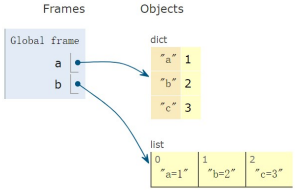
复制

```
In [13]: a = range(5)

In [14]: b = ['a','b','c','d','e']
In [20]: c = [str(y) + str(x) for x, y in zip(a,b)]
In [21]: c
Out[21]: ['a0', 'b1', 'c2', 'd3', 'e4']
```



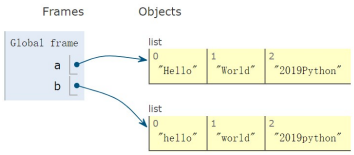
169 打印键值对

```
In [22]: a = {'a':1,'b':2,'c':3}
In [24]: b = [k+ '=' + str(v) for k, v in a.items()]
In [25]: b
Out[25]: ['a=1', 'b=2', 'c=3']
```



170 转为小写

```
In [34]: a = ['Hello', 'World', '2019Python']

In [35]: b = [w.lower() for w in a]
```



以上写法可能会有问题，Python 列表内元素类型可能不同，如果列表 a：

```
In [25]: a = ['Hello', 'World',2020,'Python']

In [26]: [w.lower() for w in a]
---------------------------------------------------------------------------
--
AttributeError: 'int' object has no attribute 'lower'
```

出现 int 对象没有 `lower` 方法的问题。

需要先转化元素为 str

```
In [27]: [str(w).lower() for w in a]
Out[27]: ['hello', 'world', '2020', 'python']
```

更友好的做法，使用 `isinstance`，判断元素是否为 `str` 类型。

如果是，再调用 `lower` 做转化：

```
In [34]: [w.lower() for w in a if isinstance(w,str) ]
Out[34]: ['hello', 'world', 'python']
```

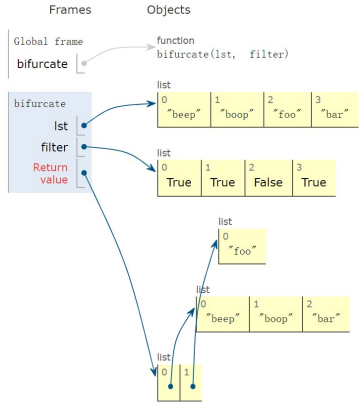
171 保留唯一值

```
In [64]: def filter_non_unique(lst):
    ...:   return [item for item in lst if lst.count(item) == 1]

In [65]: filter_non_unique([1, 2, 2, 3, 4, 4, 5])
Out[65]: [1, 3, 5]
```

有了上面这些基础后，再来看几个难度大点的案例。

172 筛选分组

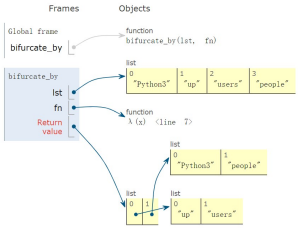
```
In [36]: def bifurcate(lst, filter):
    ...:   return [
    ...:     [x for i,x in enumerate(lst) if filter[i] == True],
    ...:     [x for i,x in enumerate(lst) if filter[i] == False]
    ...:   ]
    ...:
In [37]: bifurcate(['beep', 'boop', 'foo', 'bar'], [True, True, False, Tr
ue])

Out[37]: [['beep', 'boop', 'bar'], ['foo']]
```



173 函数分组

```
In [38]: def bifurcate_by(lst, fn):
    ...:   return [
    ...:     [x for x in lst if fn(x)],
    ...:     [x for x in lst if not fn(x)]
    ...:   ]
```

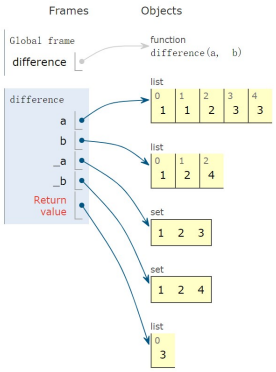
```
In [39]: bifurcate_by(['Python3', 'up', 'users', 'people'], lambda x: x[0
] == 'u')
Out[39]: [['up', 'users'], ['Python3','people']]
```



174 差集

复制

```
In [53]: def difference(a, b):
    ...:     _a, _b =set(a),set(b)
    ...:     return [item for item in _a if item not in _b]

In [54]: difference([1,1,2,3,3], [1, 2, 4])
Out[54]: [3]
```



175 函数差集

列表 `a`、`b` 中元素经过 `fn` 映射后，返回在 `a` 不在 `b` 中的元素。

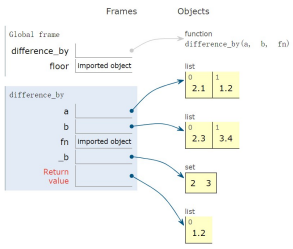
复制

```
In [14]: def difference_by(a, b, fn):
    ...:     _b = set(map(fn, b))
    ...:     return [item for item in a if fn(item) not in _b]
```

列表元素为单个元素：

复制

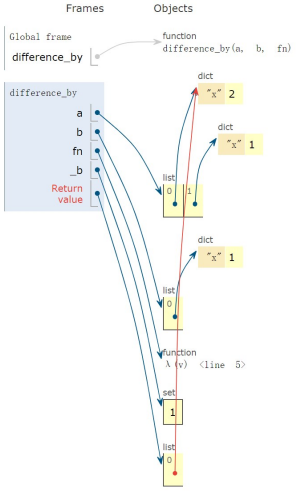
```
In [15]: from math import floor

In [16]: difference_by([2.1, 1.2], [2.3, 3.4],floor)
Out[16]: [1.2]
```



列表元素为字典：

复制

```
In [63]: difference_by([{ 'x': 2 }, { 'x': 1 }], [{ 'x': 1 }], lambda v :
 v['x'])
Out[63]: [{'x': 2}]
```



176 打乱一个列表

使用 `random` 模块， `shuffle` 函数打乱原来列表，值得注意是 in-place 打乱。

```
import random
a = range(10)
random.shuffle(a)
print(a) # [4, 6, 1, 2, 5, 3, 9, 0, 8, 7]
```

177 字典按 value 排序并返回新字典?

原字典：

d= {'a':12,'b':50,'c':1,'d':20}

使用 Python 内置函数 sorted 排序

```
In [10]: d = dict(sorted(d.items(),key=lambda item: item[1]))

In [11]: d
Out[11]: {'c': 1, 'a': 12, 'd': 20, 'b': 50}
```

178 如何删除 list 里重复元素，并保证元素顺序不变

给定列表：a = [3,2,2,2,1,3]

如果只是删除重复元素，直接使用内置 `set` 函数，去重，但是不能保证原来元素顺序。

不要这么做，列表删除某个元素后，后面的元素整体会向前移动。

```
def del_duplicated(a):
    ac = a.copy()
    b = []
    for index, i in enumerate(ac):
        if i in b:
            del ac[index]
        else:
            b.append(i)
    return ac

In [18]: r = del_duplicated(a)

In [19]: r
Out[19]: [3, 2, 2, 1] # wrong
```

正确做法：

```
def del_duplicated(a):
    b = []
    for i in a:
        if i not in b:
            b.append(i)
    return b
```

179 怎么找出两个列表的相同元素和不同元素？

给定列表 a = [3,2,2,2,1,3]，列表 b = [1,4,3,4,5]

使用集合，找出相同元素：

```
def ana(a,b):
    aset, bset = set(a), set(b)
    same = aset.intersection(bset)
    differ = aset.difference(bset).union(bset.difference(aset))
    return same, differ

In [28]: ana(a,b)
Out[28]: ({1, 3}, {2, 4, 5})
```

180 统计一个文本中单词频次最高的 10 个单词

使用 yield 解耦数据读取 `python_read` 和数据处理 `process`

`python_read`：逐行读入

`process`： 正则替换掉空字符，并使用空格，分隔字符串，保存到 `defaultdict` 对象中。

```
from collections import Counter, defaultdict
import re

def python_read(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f:
            yield line

d = defaultdict(int)

def process(line):
    for word in re.sub('\W+', " ", line).split():
        d[word] += 1
```

使用两个函数

最后，使用 `Counter` 类统计出频次最高的 10 个单词

```
for line in python_read('write_file.py'):
    process(line)

most10 = Counter(d).most_common(10)
print(most10)
```

181 反转一个整数，例如-12345 --> -54321

如果 x 位于 （ -10，10 ）间，直接返回；

然后，将 x 转换为字符串对象 sx ；

如果 x 是负数，截取 sx[1:]，并反转字符串；

如果 x 是正数，直接反转字符串 ；

最后使用内置函数 int() 转化为整数

```
def reverse_int(x: int):
    if -10 < x < 10:
        return x
    sx = str(x)

    def reverse_str(sx):
        return sx[::-1]

    if sx[0] == "-":
        sx = reverse_str(sx[1:])
        x = int(sx)
        return -x
    sx = reverse_str(sx)
    return int(sx)
```

182 以下代码输出结果

此题需要注意，内嵌函数 `foo` 使用的两个变量 `i` 和 `x` ，其中 `x` 为其形参，`i` 为 enclosing 域内定义的变量。

`rtn` 添加三个函数 `foo`，但是并未发生调用。

```
def f():
    i = 0
    def foo(x):
        return i*x
    rtn = []
    while i < 3:
        rtn.append(foo)
        i += 1
    return rtn

# 调用函数 f
for fs in f():
    print(fs(10))
```

直到 执行 `fs(10)` 时，内嵌函数 `foo` 才被调用，但是此时的 enclosing 变量 i 取值为 3

```
for fs in f():
    print(fs(10))
```

所以输出结果为：

```
30
30
30
```

183 如下函数 `foo` 的调用哪些是正确的

```
def foo(filename,a=0,b=1,c=2):
    print('filename: %s \n c: %d'%(filename,c))
```

已知 filename 为 '.'，c 为 10，正确为 `foo` 函数传参的方法，以下哪些是对的，哪些是错误的？

```
A foo('.', 10)

B foo('.', 0,1,10)

C foo('.',0,1,c=10)

D foo('.',a=0,1,10)

E foo(filename='.', c=10)

F foo('.', c=10)
```

分析：

```
A 错误, a 被赋值为 10
B 正确, c 是位置参数
C 正确, c 是关键字参数
D 错误, 位置参数不能位于关键字参数后面
E 正确, filename 和 c 都是关键字参数
F 正确, filename 位置参数, c 是关键字参数
```

验证测试：

```
In [58]: foo('.', 10)
filename: .
 c: 2

In [59]: foo('.', 0,1,10)
filename: .
 c: 10

 c: 10

In [61]: foo('.',a=0,1,10)
  File "<ipython-input-61-e3909182c523>", line 1
    foo('.',a=0,1,10)
            ^
SyntaxError: positional argument follows keyword argument


In [62]: foo(filename='.', c=10)
filename: .
 c: 10

In [63]: foo('.', c=10)
filename: .
 c: 10
```

184 单机 4 G 内存，处理 10 G文件的方法？

假定可以单独处理一行数据，行间数据相关性为零。

方法一：

仅使用 Python 内置模板，逐行读取到内存。

使用 yield，好处是**解耦读取操作和处理操作**：

```
def python_read(filename):
    with open(filename,'r',encoding='utf-8') as f:
        for line in f:
            yield line
```

以上每次读取一行，逐行迭代，逐行处理数据

```
if __name__ == '__main__':
    g = python_read('./data/movies.dat')
    for c in g:
        print(c)
        # process c
```

方法二：

方法一有缺点，逐行读入，频繁的 IO 操作拖累处理效率。是否有一次 IO ，读取多行的方法？

`pandas` 包 `read_csv` 函数，参数有 38 个之多，功能非常强大。

关于单机处理大文件，`read_csv` 的 `chunksize` 参数能做到，设置为 `5`，意味着一次读取 5 行。

```
def pandas_read(filename,sep=',',chunksize=5):
    reader = pd.read_csv(filename,sep=sep,chunksize=chunksize)
    while True:
        try:
            yield reader.get_chunk()
        except StopIteration:
            print('---Done---')
            break
```

使用如同方法一：

```
if __name__ == '__main__':
    g = pandas_read('./data/movies.dat',sep="::")
    for c in g:
        print(c)
        # process c
```

以上就是单机处理大文件的两个方法，推荐使用方法二，更加灵活。

185 使用 filter() 求出列表中大于 10 的元素

filter 函数使用 lambda 函数，找出满足大于 10 的元素。

```
a = [15,2,7,20,400,10,9,-15,107]
al = list(filter(lambda x: x > 10, a))
In [74]: al
Out[74]: [15, 20, 400, 107]
```

186 说说下面 map 函数的输出结果

map 函数当含有多个列表时，返回长度为最短列表的长度；

lambda 函数的形参个数等于后面列表的个数。

```
m = map(lambda x,y: min(x,y), [5, 1, 3, 4], [3,4,3,2,1])

print(list(m))
```

结果为：

```
[3, 1, 3, 2]
```

187 说说 reduce 函数的输出结果

reduce 实现对列表的归约化简，规则如下：

`f(x,y) = x*y + 1`

因此，下面归约的过程为：

f(1,2) = 3

f(3,3) = 3*3 + 1 = 10

f(10,4) = 10*4 + 1 = 41

f(41,5) = 41*5 + 1 = 206

```
from functools import reduce

reduce(lambda x,y: x*y+1,[1,2,3,4,5])
```

结果为：

```
206
```

188 x = (i for i in range(5))，x 是什么类型

x 是生成器类型

与 for 等迭代，输出迭代结果：

```
x = (i for i in range(5))
for i in x:
    print(i)
```

结果为：

```
0
1
2
3
4
```

189 写一个学生类 `Student`

添加一个属性 `id`，并实现若 `id` 相等，则认为是同一位同学的功能。

重写 `__eq__` 方法，若 `id` 相等，返回 `True`

```
class Student:
    def __init__(self,id,name):
        self.id = id
        self.name = name
    def __eq__(self,student):
        return self.id == student.id
```

判断两个 Student 对象，`==` 的取值：

```
s1 = Student(10,'xiaoming')
s2 = Student(20,'xiaohong')
s3 = Student(10,'xiaoming2')

In [85]: s1 == s2
Out[85]: False

In [86]: s1 == s3
Out[86]: True
```

下一章

互动评论

说点什么

评论

The Scrapper 1 个月前

习题也不错

鼓掌

Yooda 2 个月前

176 不能shuffle非列表

鼓掌