03

第 3 章

Python 函数、类与模块

简答 3-1：使用 Python 调用函数，在传递参数时，是传值还是传地址？
简答 3-2：更改全局变量的内容。
简答 3-3：设计一个含有多个回传值的函数时，请问这些回传值的数据形态如何？
简答 3-4：请用一句话解释什么样的程序语言可以使用装饰器。
简答 3-5：Python 模块（module）是什么？
简答 3-6：Python 的 __init__ 是什么？
简答 3-7：Python 的类内的关键词 self 是什么？
简答 3-8：请说明封装（encapsulation）。
简答 3-9：请说明 super( )。
简答 3-10：请说明多态（polymorphism）。
简答 3-11：Java 不支持多重继承，Python 是否支持多重继承？
简答 3-12：请说明 __iter__( ) 方法和 __next__( ) 方法。
面试实例 ch3_1.py：random 模块的 shuffle( ) 函数的应用。
面试实例 ch3_2.py：random 模块的 sample( ) 函数的应用。
面试实例 ch3_3.py：random 模块的 randint( ) 函数的应用。
面试实例 ch3_4.py：random 模块的 choice( ) 函数的应用。
面试实例 ch3_5.py：random 模块的 seed( ) 函数的应用。
面试实例 ch3_6.py：random 模块的 random( ) 函数的应用。
面试实例 ch3_7.py：产生 5 笔 0（含）～ 10.0（不含）的随机浮点数。

面试实例 ch3_8.py：Python 的 docstring 是什么？同时用程序说明。

面试实例 ch3_9.py：何谓全局变量（global variable）？何谓区域变量（local variable）？同时使用简单程序解说。

面试实例 ch3_10.py：lambda 是什么？请同时用程序实例解说。

面试实例 ch3_11.py：设计含 2 个参数的匿名函数应用，可以回传参数的积（相乘的结果）。

面试实例 ch3_12.py：使用 sorted 配合 lambda 将列表从小到大排序。

面试实例 ch3_13.py：使用 sorted 配合 lambda 将列表正值的元素从小到大排序，负值的元素从大到小排序。

面试实例 ch3_14.py：字典数据的姓名由小到大排序，年龄由大到小排序。

面试实例 ch3_15.py：元组数据的姓名由小到大排序，年龄由大到小排序。

面试实例 ch3_16.py：列表数据的姓名由小到大排序，年龄由大到小排序。

面试实例 ch3_17.py：将姓名由小到大排序，将年龄由小到大排序，当年龄相同时依照姓名由小到大排序。

面试实例 ch3_18.py：将字典转成列表，最后将列表转为字典，使用 zip( )。

面试实例 ch3_19.py：将字典转成列表，最后将列表转为字典，不使用 zip( )。

面试实例 ch3_20.py：为字符串 ['a', 'b', 'c'] 随机建立 1 ～ 10 的值，成为字典。

面试实例 ch3_21.py：在列表内依据字符串长度为元素排序。

面试实例 ch3_22.py：说明 map( ) 函数的用法，同时用程序解说。

面试实例 ch3_23.py：说明 filter( ) 函数的用法，同时用程序解说。

面试实例 ch3_24.py：说明 reduce( ) 函数的用法，同时用程序解说。

面试实例 ch3_25.py：何谓关键词参数？同时用程序解说。

面试实例 ch3_26.py：请说明如何设计可以传递任意数量参数的函数。

面试实例 ch3_27.py：请说明如何设计可以传递一般参数与任意数量参数的函数。

面试实例 ch3_28.py：请说明 *args 和 **kwargs，同时举程序实例说明。

面试实例 ch3_29.py：请说明函数是否可以当作一般函数的参数？

面试实例 ch3_30.py：请说明嵌套。

面试实例 ch3_31.py：请说明函数是否可以作为回传值，请举程序实例说明。

面试实例 ch3_32.py：请说明何谓闭包（closure）？请举程序实例说明。

面试实例 ch3_33.py：设计闭包 closure 的另一个应用。

面试实例 ch3_34.py：请说明 yield、next( ) 和 send( ) 的用法，请举程序例说明。

面试实例 ch3_35.py：重新设计 ch3_34.py，增加使用 send( ) 函数调用。

面试实例 ch3_36.py：设计自己的 range( ) 函数，此函数名称是 myRange( )。

面试实例 ch3_37.py：请说明装饰器（decorator）功能，请用实例解说。

面试实例 ch3_38.py：装饰器函数的基本操作。

面试实例 ch3_39.py：请说明 if __name__ == '__main__' 叙述的优点。

面试实例 ch3_40.py：导入 new_makefood，观察执行结果。

面试实例 ch3_41.py：如果有 A、B、C 三个变量供 test1.py、test2.py 和 test3.py 使用，应该如何设计程序？

面试实例 ch3_42.py：请说明 __init__ 和 __new__ 的区别，用文字和实例解说。

面试实例 ch3_43.py：请同时用文字和程序实例说明 isinstance( )。

面试实例 ch3_44.py 和 ch3_45.py：请同时用文字和程序实例说明 __str__( ) 方法。

面试实例 ch3_46.py：请同时用文字和程序实例说明 __repr__( ) 方法。

面试实例 ch3_47.py：斐波那契数列实践。

面试实例 ch3_48.py：不使用 sort( ) 方法，请设计列表排序。

简答 3-1：使用 Python 调用函数，在传递参数时，是传值（value）还是传地址（address）？

解答：

传地址。

简答 3-2：如果没有特别设定，可否在函数内更改全局变量的内容？如果要在函数内更改全局变量的值，须使用什么关键词设定此全局变量？

解答：

不可以。

在函数内经过 global 设定的全局变量可以在函数内更改其内容。

简答 3-3：设计一个含有多个回传值的函数时，请问这些回传值的数据形态如何？

解答：

元组（tuple）。

简答 3-4：请用一句话解释什么样的程序语言可以使用装饰器。

解答：

函数可以当作参数传递的程序语言。

简答 3-5：Python 模块（module）是什么？请列出 5 个 Python 内建的模块及其用法。

解答：

Python 模块是指包含 Python 程序代码的 .py 文件，文件内容可以是函数或类，或是一些常用变量的设定。

下列是 Python 内建的主要模块与功能简述，读者可以任选 5 个说明。

calendar	日历
csv	csv 文件
datetime	日期与时间
email	电子邮件
io	输入与输出
json	Json 文件
html	HTML
html.parser	解析 HTML
http	HTTP
math	常用数学
os	操作系统
os.path	路径管理
pickle	pickle 列表化数据
random	随机数
re	正则表达式

socket	网络应用
sqlite3	SQLite
statistics	统计
sys	系统
time	时间
urllib	URL
urllib.request	URL 的相关操作
xml.dom	XML DOM
zipfile	压缩与解压

简答 3-6：Python 的 __init__ 是什么？

解答：

建立类很重要的一个工作是**初始化整个类**，所谓的**初始化类**是在类内建立一个初始化**方法**（method），这是一个特殊**方法**，当在程序内宣告这个类的对象时将自动执行这个方法。初始化方法有一个固定名称是 __init__( )，写法是 init 左右皆有双下画线，init 其实是 initialization 的缩写。

简答 3-7：Python 的类内的关键词 self 是什么？

解答：

在 __init__( ) 初始化函数中，**self** 是必需的，它放在所有参数的**最前面**（相当于最左边），Python 在初始化时会自动传入这个参数 **self**，代表的是类本身的对象，未来在类内想要参照各**属性**与**函数**执行运算皆要使用 self。

简答 3-8：请说明封装（encapsulation）。

解答：

面向对象程序设计时，外部直接引用也代表可以直接修改类内的属性值，这将造成类数据不安全。

Python 提供了一个**私有属性**与**方法**的概念，这个概念的主要精神是类外无法直接更改类内的**私有属性**，类外也无法直接调用**私有方法**，这个概念又称**封装**（encapsulation）。

简答 3-9：请说明 super( )。

解答：

在面向对象的程序设计里，也可以使用 Python 的 super( ) 方法取得基类的属性。

简答 3-10：请说明多态（polymorphism）。

解答：

基类与衍生类有相同方法名称的实例，其实就是**多态**（polymorphism），但是在**多态**（polymorphism）的概念中是不局限在必须有父子关系的类。

简答 3-11：Java 不支持多重继承，Python 是否支持多重继承？

解答：

Python 支持多重继承，在 Python 面向对象的程序设计中，也常会发生一个类继承多个类的应用。

简答 3-12：请说明 __iter__( ) 方法和 __next__( ) 方法。

解答：

建立类的时候也可以将类定义成一个迭代对象，类似 list 或 tuple，供 for … in 循环内使用，这时类需设计 next( ) 方法，取得下一个值，直到达到结束条件，可以使用 raise StopIteration 终止继续。

面试实例 ch3_1.py：假设有 13 张扑克牌，定义如下：

```
porker = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
```

请用最简单的方式打散此牌，然后输出。

这一题主要是考 random 模块的 shuffle( ) 函数，这个函数可以将列表内容打散，主要内容如下：

```
random.shuffle(porker)
```

```
1  # ch3_1.py
2  import random
3  porker = ['A', '1', '2', '3', '4', '5', '6', '7',
4            '8', '9', '10', 'J', 'Q', 'K']
5  random.shuffle(porker)
6  print(porker)
```

执行结果

```
================ RESTART: D:/Python interveiw/ch3/ch3_1.py ================
['6', '7', '8', '5', 'A', '9', 'K', 'J', '3', '2', '4', 'Q', '10', '1']
```

面试实例 ch3_2.py：假设有 13 张扑克牌，定义如下：

```
porker = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
```

请随机每次发 5 张牌。

这一题主要是考 random 模块的 sample（列表，数量）函数，这个函数可以将列表内容打散，然后回传第 2 个参数数量的元素，主要内容如下：

```
random.sample(porker,5)
```

```
1  # ch3_2.py
2  import random
3  porker = ['A', '1', '2', '3', '4', '5', '6', '7',
4            '8', '9', '10', 'J', 'Q', 'K']
5  n = 5
6  print(random.sample(porker, n))
```

执行结果

```
================ RESTART: D:/Python interveiw/ch3/ch3_2.py ================
['2', '9', '8', 'Q', '3']
```

面试实例 ch3_3.py：请说明如何产生 5 笔 0 ～ 100 的随机数整数。

这一题主要是考 random 模块的 randint（min，max）函数，这个函数可以回传 min（含）和 max（含）之间的整数，主要内容如下：

```
random.randint(0,100)
```

```
1  # ch3_3.py
2  import random
3  n = 5
4  min = 0
5  max = 100
6  for i in range(n):
7      print(random.randint(min, max))
```

执行结果

```
================ RESTART: D:/Python interveiw/ch3/ch3_3.py ================
61
26
74
15
25
```

面试实例 ch3_4.py：请使用 choice( ) 方法产生 5 笔 0 ～ 100 的随机整数。

```
1  # ch3_4.py
2  import random
3
4  n = 5
5  for i in range(n):
6      print(random.choice(range(0, 101)))
```

执行结果

```
================ RESTART: D:/Python interveiw/ch3/ch3_4.py ================
18
8
57
72
20
```

面试实例 ch3_5.py：使用 random.randint( ) 方法每次产生的随机数皆不相同，例如：若是重复执行 ch3_3.py，可以看到每次皆是不一样的 5 个随机数。

```
================ RESTART: D:/Python interveiw/ch3/ch3_3.py ================
6
67
43
44
70
>>>
================ RESTART: D:/Python interveiw/ch3/ch3_3.py ================
39
29
71
15
56
>>>
================ RESTART: D:/Python interveiw/ch3/ch3_3.py ================
81
0
19
15
67
```

如果我们希望每次皆可以产生相同的随机数，应该如何处理？

其实可以使用 random 模块的 seed（x）方法，其中参数 x 是种子值，例如设定 x=5，当设定此种子值后，未来每次使用随机函数如 randint( )、random( ) 产生随机数时，都可以得到相同的随机数。

```
1  # ch3_5.py
2  import random
3  n = 5
4  min = 0
5  max = 100
6  random.seed(5)
7  for i in range(n):
8      print(random.randint(min, max))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_5.py =================
79
32
94
45
88
>>>
================= RESTART: D:/Python interveiw/ch3/ch3_5.py =================
79
32
94
45
88
>>>
================= RESTART: D:/Python interveiw/ch3/ch3_5.py =================
79
32
94
45
88
```

面试实例 ch3_6.py：请说明如何产生 5 笔 0（含）～ 1.0（不含）的随机浮点数。

这一题主要是考 random 模块的 random（）函数，这个函数可以回传 0.0（含）和 1.0（不含）之间的浮点数，主要内容如下：

```
random.random( )
```

```
1  # ch3_6.py
2  import random
3  n = 5
4  for i in range(n):
5      print(random.random())
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_6.py =================
0.31631674944917154
0.2259714972551281
0.624651917662772
0.6277901195045082
0.08075119571962819
```

面试实例 ch3_7.py：请说明如何产生 5 笔 0（含）～ 10.0（不含）的随机浮点数。

解答：

这一题主要是考 random 模块的 uniform（min，max）函数，这个函数可以回传 min（含）和 max（不含）之间的浮点数，主要内容如下：

```
random.randint(0,10.0)
```

```
1  # ch3_7.py
2  import random
3  n = 5
4  min = 0.0
5  max = 10.0
6  for i in range(n):
7      print(random.uniform(min, max))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_7.py =================
4.533288630187668
0.00014370976877065011
1.3817140454964927
0.30209024198817325
6.227072304364718
```

面试实例 ch3_8.py：Python 的 docstring 是什么？同时用程序说明。

解答：

所谓 docstring 全名是 documentation string，可以解释为文件字符串，是指在模块、函数、类方法中第一行使用 3 个单引号 ''' 或是双引号 """ 包含的内容，这个内容也常被用作注释。

```
1  # ch3_8.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi,", name, "Good Morning!")
5  greeting('Nelson')
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_8.py =================
Hi, Nelson Good Morning!
```

上述函数 greeting( ) 名称下方是 """**Python 函数需传递名字 name** """ 字符串，Python 语言将此**函数注释**称**文件字符串** docstring。一个公司设计大型程序时，常常将工作分成很多小程序，每个人的工作将用函数完成，为了要让其他团队成员可以了解你所设计的函数，需要用**文件字符串**注明此函数的功能与用法。

可以使用 help（函数名称）列出此函数的文件字符串，可以参考下列实例。假设程序已经执行了 ch3_8.py 程序，下列是列出此程序的 greeting( ) 函数的**文件字符串**。

```
>>> help(greeting)
Help on function greeting in module __main__:

greeting(name)
    Python函数需传递名字name
```

如果只想看函数注释，可以使用下列方法：

```
>>> print(greeting.__doc__)
Python函数需传递名称name
```

上述奇怪的 greeting.__doc__ 就是 greeting( ) 函数文件字符串的变量名称，__ 其实是双下画线，这是系统保留名称的方法。

面试实例 ch3_9.py：何谓全局变量（global variable）？何谓区域变量（local variable）？同时使用简单程序解说。

解答：

全局变量（global variable）：在函数或类外宣告的变量，这个变量可以在程序所有地方引用，直到整个程序执行结束。

区域变量（local variable）：在函数内宣告的变量，这个变量只能在所宣告的函数内使用，同时此函数执行结束后，区域变量所占用的内存空间将被收回，区域变量也将被销毁。

有一个程序如下，请列出全局变量（global variable）和区域变量（local variable）。

```
1  # ch3_9.py
2  def fun():
3      a = 5
4      c = a + b
5      print(c)
6
7  b = 3
8  fun()
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/ch3_9.py ==================
8
```

全局变量：b。

区域变量：a，c。

面试实例 ch3_10.py：lambda 是什么？请同时用程序实例解说。

解答：

匿名函数（anonymous function）是指一个没有名称的函数，适合使用在程序中只存在一小段时间的情况。Python 是使用 def 定义一般函数，匿名函数则是使用 **lambda** 来定义，有的人称之为 lambda 表达式，也可以将匿名函数称为 lambda 函数。

匿名函数另一个特色是可以有许多的参数，但是只能有一个程序码表达式，然后可以将执行结果回传。

```
lambda arg1[,arg2,… argn]:expression        # arg1 是参数，可以有多个参数
```

上述 expression 就是匿名函数 lambda 表达式的内容，下列是单一参数的匿名函数应用，可以回传立方值。

```
1  # ch3_10.py
2  # 定义lambda函数
3  cube = lambda x: x * x * x
4
5  # 输出立方值
6  print(cube(10))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_10.py =================
1000
```

面试实例 ch3_11.py：设计含 2 个参数的匿名函数应用，可以回传参数的积（相乘的结果）。

```
1  # ch3_11.py
2  # 定义lambda函数
3  product = lambda x, y: x * y
4
5  # 输出相乘结果
6  print(product(5, 10))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_11.py =================
50
```

面试实例 ch3_12.py：有一个列表内容如下：

```
lst = [-8,5,9,-2,-11,12,1,10,-5]
```

请使用 sorted 配合 lambda 将上述列表从小到大排序。

```
1  # ch3_12.py
2  lst = [-8, 5, 9, -2, -11, 12, 1, 10, -5]
3  sorted_lst = sorted(lst, key=lambda x:x)
4  print(sorted_lst)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_12.py =================
[-11, -8, -5, -2, 1, 5, 9, 10, 12]
```

面试实例 ch3_13.py：有一个列表内容如下：

```
lst = [-8,5,9,-2,-11,12,1,10,-5]
```

请使用 sorted 配合 lambda 将上述列表正值的元素从小到大排序，负值的元素从大到小排序。

```
1  # ch3_13.py
2  lst = [-8, 5, 9, -2, -11, 12, 1, 10, -5]
3  sorted_lst = sorted(lst, key=lambda x:(x<0, abs(x)))
4  print(sorted_lst)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_13.py =================
[1, 5, 9, 10, 12, -2, -5, -8, -11]
```

面试实例 ch3_14.py：有一个列表内部元素是字典，如下所示：

```
member = [{'name':'Peter',
           'age':25},
          {'name':'Mary',
           'age':22},
          {'name':'Kevin',
           'age':29},
          {'name':'Tom',
           'age':18}]
```

将姓名由小到大排序，将年龄由大到小排序。

```
# ch3_14.py
member = [{'name':'Peter',
           'age':25},
          {'name':'Mary',
           'age':22},
          {'name':'Kevin',
           'age':29},
          {'name':'Tom',
           'age':18}]
sorted_name = sorted(member, key=lambda x:x['name'])
print('依照姓名由小到大排序')
for n in sorted_name:
    print(n)
print('依照年龄由大到小排序')
sorted_age = sorted(member, key=lambda x:x['age'], reverse=True)
for a in sorted_age:
    print(a)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_14.py =================
依照姓名由小到大排序
{'name': 'Kevin', 'age': 29}
{'name': 'Mary', 'age': 22}
{'name': 'Peter', 'age': 25}
{'name': 'Tom', 'age': 18}
依照年龄由大到小排序
{'name': 'Kevin', 'age': 29}
{'name': 'Peter', 'age': 25}
{'name': 'Mary', 'age': 22}
{'name': 'Tom', 'age': 18}
```

面试实例 ch3_15.py：有一个列表内部元素是元组，如下所示：

```
member = [('Peter', 25),
          ('Mary', 22),
          ('Kevin', 29),
          ('Tom', 18)]
```

将姓名由小到大排序，将年龄由大到小排序。

```
# ch3_15.py
member = [('Peter', 25),
          ('Mary', 22),
          ('Kevin', 29),
          ('Tom', 18)]
sorted_name = sorted(member, key=lambda x:x[0])
print('依照姓名由小到大排序')
for n in sorted_name:
    print(n)
print('依照年龄由大到小排序')
sorted_age = sorted(member, key=lambda x:x[1], reverse=True)
for a in sorted_age:
    print(a)
```

执行结果

```
================ RESTART: D:\Python interveiw\ch3\ch3_15.py ================
依照姓名由小到大排序
('Kevin', 29)
('Mary', 22)
('Peter', 25)
('Tom', 18)
依照年龄由大到小排序
('Kevin', 29)
('Peter', 25)
('Mary', 22)
('Tom', 18)
```

面试实例 ch3_16.py：有一个列表内部元素是列表，如下所示：

```
member = [['Peter', 25],
          ['Mary', 22],
          ['Kevin', 29],
          ['Jessica', 22],
          ['Nancy', 22],
          ['Tom', 18]]
```

将姓名由小到大排序，将年龄由大到小排序。

```
# ch3_16.py
member = [['Peter', 25],
          ['Mary', 22],
          ['Kevin', 29],
          ['Jessica', 22],
          ['Nancy', 22],
          ['Tom', 18]]
sorted_name = sorted(member, key=lambda x:x[0])
print('依照姓名由小到大排序')
for n in sorted_name:
    print(n)
print('依照年龄由大到小排序')
sorted_age = sorted(member, key=lambda x:x[1], reverse=True)
for a in sorted_age:
    print(a)
```

执行结果

```
================== RESTART: D:\Python interveiw\ch3\ch3_16.py ==================
依照姓名由小到大排序
['Jessica', 22]
['Kevin', 29]
['Mary', 22]
['Nancy', 22]
['Peter', 25]
['Tom', 18]
依照年龄由大到小排序
['Kevin', 29]
['Peter', 25]
['Mary', 22]
['Jessica', 22]
['Nancy', 22]
['Tom', 18]
```

面试实例 ch3_17.py：有一个列表内部元素是列表，如下所示：

```
member = [['Peter', 25],
          ['Mary', 22],
          ['Kevin', 29],
          ['Jessica', 22],
          ['Nancy', 22],
          ['Tom', 18]]
```

将姓名由小到大排序，将年龄由小到大排序，当年龄相同时依照姓名由小到大排序。

```
1  # ch3_17.py
2  member = [['Peter', 25],
3            ['Mary', 22],
4            ['Kevin', 29],
5            ['Jessica', 22],
6            ['Nancy', 22],
7            ['Tom', 18]]
8  sorted_name = sorted(member, key=lambda x:x[0])
9  print('依照姓名由小到大排序')
10 for n in sorted_name:
11     print(n)
12 print('依照年龄由小到大排序，年龄相同则依照姓名由小到大')
13 sorted_age = sorted(member, key=lambda x:(x[1],x[0]))
14 for a in sorted_age:
15     print(a)
```

执行结果

```
================== RESTART: D:\Python interveiw\ch3\ch3_17.py ==================
依照姓名由小到大排序
['Jessica', 22]
['Kevin', 29]
['Mary', 22]
['Nancy', 22]
['Peter', 25]
['Tom', 18]
依照年龄由小到大排序，年龄相同则依照姓名由小到大
['Tom', 18]
['Jessica', 22]
['Mary', 22]
['Nancy', 22]
['Peter', 25]
['Kevin', 29]
```

面试实例 ch3_18.py：有一个字典内容如下：

```
member = {'Peter':25,
          'Kevin':29,
          'Tom':18}
```

将字典转成列表，列表元素是元组，然后将列表元组排序，最后将列表转为字典。

```
# ch3_18.py
member = {'Peter':25,
          'Kevin':29,
          'Tom':18}
mem_zip = zip(member.keys(), member.values())
mem = [i for i in mem_zip]
print('字典转成列表，元素是元组 :', mem)
sorted_mem = sorted(mem, key=lambda x:x[0])
print('依键值由小到大排序', sorted_mem)
dict_mem = {i[0]:i[1] for i in sorted_mem}
print('转为字典 :', dict_mem)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_18.py =================
字典转成列表，元素是元组 : [('Peter', 25), ('Kevin', 29), ('Tom', 18)]
依键值由小到大排序 [('Kevin', 29), ('Peter', 25), ('Tom', 18)]
转为字典 : {'Kevin': 29, 'Peter': 25, 'Tom': 18}
```

面试实例 ch3_19.py：有一个字典内容如下：

```
member = {'Peter':25,
          'Kevin':29,
          'Tom':18}
```

将字典转成列表，列表元素是元组，然后将列表元组排序，最后将列表转为字典，这一题要求不使用 zip( ) 。

```
# ch3_19.py
member = {'Peter':25,
          'Kevin':29,
          'Tom':18}
mem = member.items()
print('字典转成列表，元素是元组 :', mem)
sorted_mem = sorted(mem, key=lambda x:x[0])
print('依键值由小到大排序', sorted_mem)
dict_mem = {i[0]:i[1] for i in sorted_mem}
print('转为字典 :', dict_mem)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_19.py =================
字典转成列表，元素是元组 : dict_items([('Peter', 25), ('Kevin', 29), ('Tom', 18)
])
依键值由小到大排序 [('Kevin', 29), ('Peter', 25), ('Tom', 18)]
转为字典 : {'Kevin': 29, 'Peter': 25, 'Tom': 18}
```

面试实例 ch3_20.py：有一个列表如下：

```
['a','b','c']
```

将上述列表的元素当作字典的键，随机建立 1 ～ 10 的值。

```
# ch3_20.py
import random

my_dict = {i:random.randint(1, 10) for i in ['a', 'b', 'c']}
print(my_dict)
```

执行结果

```
================ RESTART: D:/Python interveiw/ch3/ch3_20.py ================
{'a': 3, 'b': 8, 'c': 7}
>>>
================ RESTART: D:/Python interveiw/ch3/ch3_20.py ================
{'a': 8, 'b': 2, 'c': 6}
```

面试实例 ch3_21.py：在列表内依据字符串长度为元素排序，先使用 sorted( ) 方法不更改原列表内容，再用 sort( ) 方法重新依据字符串长度为元素排序，此时将更改原列表内容。

```
# ch3_21.py
string = ['abc', 'ab', 'linda', 'a']
new_str = sorted(string, key = lambda x:len(x))
print('string  : ', string)
print('new_str : ', new_str)
string.sort(key=len)
print('string  : ', string)
```

执行结果

```
================ RESTART: D:/Python interveiw/ch3/ch3_21.py ================
string  :  ['abc', 'ab', 'linda', 'a']
new_str :  ['a', 'ab', 'abc', 'linda']
string  :  ['a', 'ab', 'abc', 'linda']
```

面试实例 ch3_22.py：说明 map( ) 函数的用法，同时用程序解说。

解答：

内建函数 map( )，它的语法格式如下：

```
map(func,iterable)
```

上述函数将依次把 iterable（可以重复执行的字符串 string、列表 list 或元组 tuple 等）的元素（item）放入 func( ) 内，然后将 func( ) 函数执行结果回传，上述参数 func 常常使用 lambda 表达式。

下列是使用匿名函数对列表元素执行平方运算。

```
# ch3_22.py
mylist = [5, 10, 15, 20, 25, 30]

squarelist = list(map(lambda x: x ** 2, mylist))

# 输出列表元素的平方值
print("列表的平方值: ",squarelist)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_22.py =================
列表的平方值:  [25, 100, 225, 400, 625, 900]
```

面试实例 ch3_23.py：说明 filter( ) 函数的用法，同时用程序解说。

解答：

内建函数 filter( ) 主要用于筛选序列，它的语法格式如下：

```
filter(func,iterable)
```

上述函数将依次把 iterable（可以重复执行的字符串 string、列表 list 或元组 tuple 等）的元素（item）放入 func( ) 内，然后将 func( ) 函数执行结果是 True 的元素组成新的筛选对象（filter object）回传，上述参数 func 常常使用 lambda 表达式。

下列是使用 lambda 配合 filter( ) 函数，将列表内容是奇数的元素筛选出来。

```
1  # ch3_23.py
2  mylist = [5, 10, 15, 20, 25, 30]
3
4  oddlist = list(filter(lambda x: (x % 2 == 1), mylist))
5
6  # 输出奇数列表
7  print("奇数列表: ",oddlist)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_23.py =================
奇数列表:  [5, 15, 25]
```

面试实例 ch3_24.py：说明 reduce( ) 函数的用法，同时用程序解说。

解答：

内建函数 reduce( ) 的语法格式如下：

```
reduce(func,iterable)                                    # func 必须有 2 个参数
```

它会先对可迭代对象的第 1 和第 2 个元素操作，得出结果再和第 3 个元素操作，直到最后一个元素。假设 iterable 有 4 个元素，可以用下列方式解说：

```
reduce(f,[a,b,c,d]) = f( f( f(a,b),c),d)      # f 其实就是 func 常用 lambda
```

早期 reduce( ) 是内建函数，现在被移至 funtools，所以使用前需在程序前方加上下列 import:

```
from functools import reduce          # 导入 reduce( )
```

下列是设计字符串转整数的函数，为了验证转整数是否正确，笔者将此字符串加 10，最后再输出。

```
1  # ch3_24.py
2  from functools import reduce
3  def strToInt(s):
4      def charToNum(s):
5          return {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9}[s]
6      return reduce(lambda x,y:10*x+y, map(charToNum,s))
7
8  string = '5487'
9  x = strToInt(string) + 10
10 print("x = ", x)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_24.py =================
x =  5497
```

面试实例 ch3_25.py：请说明在调用函数时，何谓关键词参数？同时用程序解说。

解答：

所谓的关键词参数（keyword arguments）是指调用函数时，参数是用“参数名称 = 值”的配对方式呈现，本质上关键词参数是字典形式（dict）。

下列程序传递参数时，直接使用关键词参数（keyword arguments），即参数名称 = 值的配对方式传送。

```
1  # ch3_25.py
2  def interest(interest_type, subject):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中，最喜欢的是 " + subject)
6      print( )
7
8  interest(interest_type = '旅游', subject = '敦煌')  # 位置正确
9  interest(subject = '敦煌', interest_type = '旅游')  # 位置更动
10 hobby = '旅游'
11 place = '敦煌'
12 interest(hobby, subject = place)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_25.py =================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌
```

面试实例 ch3_26.py：请说明如何设计可以传递任意数量参数的函数。

解答：

有时候可能会碰上不知道会有多少个参数会传递到这个函数，在设计这类函数时，可以在参数左边加上 *。例如：如果函数是 beef_noddle( )，参数是 toppings，当不知道有多少参数会传递时，可以使用下列方式设计此函数：

```
def beef_noodle(*toppings):
    ...
```

下列是建立一个牛肉面的配料程序，这个程序在调用牛肉面函数 beef_noodle( ) 时，可以传递 0 到多个配料，然后 beef_noodle( ) 函数会将配料结果的牛肉面列出来。

```
# ch3_26.py
def beef_noodle(*toppings):
    """ 列出制作牛肉面的配料 """
    print("这碗牛肉面所加配料如下")
    for topping in toppings:
        print("--- ", topping)

beef_noodle('牛肉')
beef_noodle('牛肉', '辣椒', '葱花')
```

执行结果

```
================ RESTART: D:\Python interveiw\ch3\ch3_26.py ================
这碗牛肉面所加配料如下
---  牛肉
这碗牛肉面所加配料如下
---  牛肉
---  辣椒
---  葱花
```

面试实例 ch3_27.py：请说明如何设计可以传递一般参数与任意数量参数的函数。

解答：

程序设计时会遇上需要传递一般参数与任意数量参数的情况，此时任意数量的参数必须放在最右边。

```
def make_noodles(noodle_type,*toppings):
            ...
```

下列程序在传递参数时第一个参数是面的种类，然后才是不同数量的面的配料。

```
# ch3_27.py
def make_noodle(noodle_type, *toppings):
    """ 列出制作各种面的配料 """
    print("这个 ", noodle_type, " 面所加配料如下")
    for topping in toppings:
        print("--- ", topping)

make_noodle('肉丝', '猪肉', '酸菜')
make_noodle('牛肉', '牛肉', '辣椒', '葱花')
```

执行结果

```
================ RESTART: D:\Python interveiw\ch3\ch3_27.py ================
这个  肉丝  面所加配料如下
---  猪肉
---  酸菜
这个  牛肉  面所加配料如下
---  牛肉
---  辣椒
---  葱花
```

面试实例 ch3_28.py：请说明 *args 和 **kwargs，同时举程序实例说明。

解答：

*args 是指函数可以有任意数量的参数，本质是元组（tuple）。

**kwargs 是指函数可以有任意数量的关键词参数，本质是字典（dict）。

在一个函数内同时有 *args 和 **kwargs 时，*args 必须在 **kwargs 前面，下列是程序实例解说。

```
# ch3_28.py
def fun(*args,**kwargs):
    print('args =', args)
    print('kwargs =',kwargs)

fun(1, 2, 3, 4, 5, A='I', B='like', C='Python')
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_28.py =================
args = (1, 2, 3, 4, 5)
kwargs = {'A': 'I', 'B': 'like', 'C': 'Python'}
```

面试实例 ch3_29.py：请说明函数是否可以当作一般函数的参数？如果可以，请举程序实例说明。

解答：

在 Python 中函数也可以当作参数被传递给其他函数，当函数当作参数传递时，可以不用加上 ( ) 符号，这样 Python 就可以将函数当作对象处理。如果加上括号，会被视为调用这个函数。

下列是函数当作是传递参数的基本应用。

```
# ch3_29.py
def add(x, y):
    return x+y

def mul(x, y):
    return x*y

def running(func, arg1, arg2):
    return func(arg1, arg2)

result1 = running(add, 5, 10)        # add函数当作参数
print(result1)
result2 = running(mul, 5, 10)        # mul函数当作参数
print(result2)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_29.py =================
15
50
```

面试实例 ch3_30.py：请说明嵌套函数，请举程序例说明。

解答：

所谓的嵌套函数是指函数内部也可以有函数，有时候可以利用这个特性执行复杂的运算。嵌套函数也具有可重复使用、封装、隐藏数据的效果。

下列是使用嵌套函数计算 2 个坐标点之距离，外层函数是第 2-7 行的 dist( )，此函数第 3-4 是内层 mySqrt( ) 函数。

```
# ch3_30.py
def dist(x1,y1,x2,y2):          # 计算2点之距离函数
    def mySqrt(z):              # 计算开根号值
        return z ** 0.5
    dx = (x1 - x2) ** 2
    dy = (y1 - y2) ** 2
    return mySqrt(dx+dy)

print(dist(0,0,1,1))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_30.py =================
1.4142135623730951
```

面试实例 ch3_31.py：请说明函数是否可以作为回传值，请举程序实例说明。

解答：

在嵌套函数的应用中，常常会将一个内层函数当作回传值，这时所回传的是内层函数的内存地址。

下列是计算 1 -（n-1）的总和，观察函数当作回传值的应用，这个程序的第 2 ～ 6 行是 outer( ) 函数，第 6 行的回传值是不含 ( ) 的 inner。

```
# ch3_31.py
def outer():
    def inner(n):
        print('inner running')
        return sum(range(n))
    return inner

f = outer()          # outer()回传inner地址
print(f)             # 打印inner内存
print(f(5))          # 实际执行的是inner()

y = outer()
print(y)
print(y(10))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_31.py =================
<function outer.<locals>.inner at 0x0275F150>
inner running
10
<function outer.<locals>.inner at 0x02B74738>
inner running
45
```

面试实例 ch3_32.py：请说明何谓闭包（closure）？请举程序实例说明。

解答：

内部函数是一个动态产生的程序，当它可以记住函数以外的程序所建立的环境变量值时，可以

称这个内部函数是**闭包**（closure）。

下列是一个线性函数 ax+b 的闭包说明。

```
1  # ch3_32.py
2  def outer():
3      b = 10                    # inner所使用的变量值
4      def inner(x):
5          return 5 * x + b      # 引用第3行的b
6      return inner
7
8  b = 2
9  f = outer()
10 print(f(b))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_32.py =================
20
```

上述第 3 行 b 是一个环境变量，这也是定义在 inner( ) 以外的变量，由于第 6 行使用 inner 当作回传值，inner( ) 内的 b 其实就是第 3 行所定义的 b，变量 b 和 inner( ) 就构成了一个 closure。 程序第 10 行 f（b）中的 b 将是 inner（x）的 x 参数，所以最后可以得到 5 * 2 + 10，结果是 20。

其实 __closure__ 内是一个元组，环境变量 b 就是存在 cell_contents 内。

```
>>> print(f)
<function outer.<locals>.inner at 0x0357F150>
>>> print(f.__closure__)
(<cell at 0x039D72D0: int object at 0x5B8EC910>,)
>>> print(f.__closure__[0].cell_contents)
10
```

面试实例 ch3_33.py：设计闭包 closure 的另一个应用，这也是线性函数 ax+b，不过环境变量是 outer( ) 的参数。

```
1  # ch3_33.py
2  def outer(a, b):
3      ''' a 和 b 将是inner()的环境变量 '''
4      def inner(x):
5          return a * x + b
6      return inner
7
8  f1 = outer(1, 2)
9  f2 = outer(3, 4)
10 print(f1(1), f2(3))
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_33.py =================
3 13
```

这个程序第 8 行建立了 x+2，第 9 行建立了 3x+4，相当于使用了 closure 将最终线性函数确定下来。第 10 行传递适当的值，就可以获得结果。在这里我们发现程序代码可以重复使用，此外如果没

有 closure，我们需要传递 a、b、x 参数，所以 closure 可以让程序设计更有效率，同时未来扩充时程序代码可以更容易移植。

面试实例 ch3_34.py：请说明 yield、next( ) 和 send( ) 的用法，请举程序实例说明。

解答：

可以将 yield 想成是 return，普通的 return 在函数中回传某个值后就不再往下执行，同时此函数所有区域数据将被删除。yield 其实是生成器（generator）的一部分，gen 函数内有 yield 时，若是执行到 yield 指令，只是将程序的主导权由 gen 函数交还原调用 gen 函数的位置，此 gen 函数数据依旧保留，下一次调用 gen 函数时，由上次暂停位置（yield）往下执行。

下列是简单的 yield 指令实例，由这个实例可以观察 yield 的操作。

```
1  # ch3_34.py
2  def gen():
3      ''' 测试yield '''
4      print('进入gen()')
5      while True:
6          rtn = yield 10
7          print('rtn :', rtn)
8
9  g = gen()                          # 建立生成器gen对象
10 print('主程序 :', next(g))
11 print('*'*50)
12 print('主程序 :', next(g))
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_34.py =================
进入gen( )
主程序 : 10
**************************************************
rtn : None
主程序 : 10
```

这个程序说明如下：

（1）执行第 9 行，这个指令调用 gen( )，由于 gen( ) 函数内有 yield 所以不会真正执行，而是建立生成器对象 g。

（2）执行第 10 行，这是 print( )，参数是 next（g），可以进入 gen( )，所以打印第 4 行的“进入 gen( )”，然后进入 while 循环，执行第 6 行的 rtn = yield 10，这时返回调用的主程序，读者需留意现在并没有设定赋值给 rtn。

```
rtn = yield 10          # 在现阶段只有执行 yield 10, 并没有执行赋值给 rtn
```

因为是回传 10，所以第 10 行打印“主程序：10”。

（3）执行第 11 行，这是 print( )，这会打印 *50 次。

（4）执行第 12 行，这时回到 gen( ) 函数内先前离开的 yield。

```
rtn = yield 10          # 在现阶段要执行赋值给 rtn
```

因为原先 yield 的 10 已经回传，此时没有值赋给 rtn，rtn 是 None，所以输出 rtn：None，再度进入 while 循环，当执行到第 6 行的 yield 10，这时返回调用的主程序，因为是回传 10，所以第 12

行打印“主程序：10”。

其实内含 yield 的函数式在 Python 中称生成器，这个生成器有一个函数 next( )，可以想成下一次生成哪一个数值，下一次 next( ) 执行的地方是从上次终止的地方开始执行，然后遇到 yield 后，可以将数值生成。

在这个生成器中有一个函数 send( ) 可以接着 yield 执行，然后将 send（x）函数的 x 参数给 yield 进行赋值操作。

面试实例 ch3_35.py：重新设计 ch3_34.py，增加使用 send( ) 函数调用。

```
1  # ch3_35.py
2  def gen():
3      ''' 测试yield '''
4      print('进入gen()')
5      while True:
6          rtn = yield 10
7          print('rtn :', rtn)
8
9  g = gen()                              # 建立生成器gen对象
10 print('主程序 :', next(g))
11 print('*'*50)
12 print('主程序 :', g.send(100))
```

执行结果

```
================ RESTART: D:\Python interveiw\ch3\ch3_35.py ================
进入gen()
主程序 : 10
**************************************************
rtn : 100
主程序 : 10
```

上述执行第 12 行，这时回到 gen( ) 函数内先前离开的 yield。

```
rtn = yield 10                    # 在现阶段要执行赋值给 rtn
```

这时第 12 行的 g.send（100）赋了 100 给 rtn，所以 rtn 是 100，所以输出 rtn：100。再度进入 while 循环，当执行到第 6 行的 yield 10，这时返回调用的主程序，因为是回传 10，所以第 10 行打印“主程序：10”。

当我们懂了上述概念后，就可以设计属于自己的 range( ) 函数了。

面试实例 ch3_36.py：设计自己的 range( ) 函数，此函数名称是 myRange( )。

```
1  # ch3_36.py
2  def myRange(start=0, stop=100, step=1):
3      n = start
4      while n < stop:
5          yield n
6          n += step
7
8  print(type(myRange))
9  for x in myRange(0,5):
10     print(x)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/ch3_36.py ==================
<class 'function'>
0
1
2
3
4
```

上述我们设计的 myRange( ) 函数，它的数据类型是 function，所执行的功能与 range( ) 类似，不过当我们调用此函数时，它的回传值不是使用 return，而是使用 yield，同时整个函数内部不是立即执行。第一次 for 循环执行时会执行到 yield 关键词，然后回传 n 值。下一次 for 循环迭代时会继续执行此函数的第 6 行 n += step，然后回到函数起点再执行到 yield，循环直到没有值可以回传。

我们又将此 range( ) 概念称为**生成器**（generator）。

面试实例 ch3_37.py：请说明装饰器（decorator）功能，请用实例解说。

解答：

在设计程序时，如果我们想在函数内增加一些功能，但是又不想更改原先的函数，这时可以使用 Python 所提供的**装饰器**（decorator）。装饰器其实也是一种函数，此函数会接收一个函数，然后回传另一个函数。下列是一个简单打印所传递的字符串然后输出的实例：

```
>>> def greeting(string):
        return string

>>> greeting('Hello! iPhone')
'Hello! iPhone'
```

假设我们不想更改 greeting( ) 函数的内容，但是希望可以将输出改成大写，此时就可以使用装饰器。

下列是装饰器函数的基本操作，这个程序将设计一个 upper( ) 装饰器，这个程序除了将所输入字符串改成大写，同时也列出所装饰的函数名称，以及函数所传递的参数。

```
1  # ch3_37.py
2  def upper(func):                  # 装饰器
3      def newFunc(args):
4          oldresult = func(args)
5          newresult = oldresult.upper()
6          print('函数名称 : ', func.__name__)
7          print('函数参数 : ', args)
8          return newresult
9      return newFunc
10
11 def greeting(string):              # 问候函数
12     return string
13
14 mygreeting = upper(greeting)      # 手动装饰器
15 print(mygreeting('Hello! iPhone'))
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_37.py =================
函数名称 :  greeting
函数参数 :  Hello! iPhone
HELLO! IPHONE
```

上述程序第 14 行是手动设定装饰器，第 15 行是调用装饰器和打印。

装饰器设计的原则是有一个函数当作参数，然后在装饰器内重新定义一个含有装饰功能的新函数，可参考第 3 ～ 8 行。第 4 行是获得原函数 greeting( ) 的结果，第 5 行是将 greeting( ) 的结果装饰成新的结果，也就是将字符串转成大写。第 6 行是打印原函数的名称，在这里我们使用了 func.__name__，这是函数名称变量。第 7 行是打印所传递参数内容，第 8 行是回传新的结果。

下面实例是用 @upper 直接定义装饰器名称。

面试实例 ch3_38.py：设计一个 upper( ) 装饰器，这个程序除了将所输入字符串改成大写，同时也列出所装饰的函数名称，以及函数所传递的参数。

```
1  # ch3_38.py
2  def upper(func):                  # 装饰器
3      def newFunc(args):
4          oldresult = func(args)
5          newresult = oldresult.upper()
6          print('函数名称 : ', func.__name__)
7          print('函数参数 : ', args)
8          return newresult
9      return newFunc
10 @upper                            # 设定装饰器
11 def greeting(string):             # 问候函数
12     return string
13
14 print(greeting('Hello! iPhone'))
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_38.py =================
函数名称 :  greeting
函数参数 :  Hello! iPhone
HELLO! IPHONE
```

面试实例 ch3_39.py：请说明下列程序的优点，并用实例解说。

```
if __name__ == '__main__'
dosomething( )
```

解答：

假设上述程序名称是 test.py，如果上述程序是自己执行，那么 __name__ 就一定是 __main__，会执行 dosomething( ) 函数。

如果上述程序 test.py 是被 import，则 __name__ 是程序名称 test。下列笔者将用多个程序解说。

所以如果所设计的程序是当作模块使用，则须加 if __name__ == '__main__'，下列是此要点的程序实例解说。

程序实例 old_makefood.py：设计不含 if __name__ == '__main__' 的传统 Python 程序。

```
# old_makefood.py
def make_icecream(*toppings):
    # 列出制作冰淇淋的配料
    print("这个冰淇淋所加配料如下")
    for topping in toppings:
        print("--- ", topping)

def make_drink(size, drink):
    # 输入饮料规格与种类,然后输出饮料
    print("所点饮料如下")
    print("--- ", size.title())
    print("--- ", drink.title())

make_icecream('草莓酱')
make_icecream('草莓酱', '葡萄干', '巧克力碎片')
make_drink('large', 'coke')
```

执行结果

```
============== RESTART: D:\Python interveiw\ch3\old_makefood.py ==============
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
```

上述是传统 Python 程序，接下来看含 if __name__ == '__main__' 的 Python 程序。

程序实例 new_makefood.py：设计含 if __name__ == '__main__' 的 Python 程序。

```
# new_makefood.py
def make_icecream(*toppings):
    # 列出制作冰淇淋的配料
    print("这个冰淇淋所加配料如下")
    for topping in toppings:
        print("--- ", topping)

def make_drink(size, drink):
    # 输入饮料规格与种类,然后输出饮料
    print("所点饮料如下")
    print("--- ", size.title())
    print("--- ", drink.title())

def main():
    make_icecream('草莓酱')
    make_icecream('草莓酱', '葡萄干', '巧克力碎片')
    make_drink('large', 'coke')

if __name__ == '__main__':
    main()
```

执行结果

```
============== RESTART: D:\Python interveiw\ch3\new_makefood.py ==============
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
```

表面上看，old_makefood.py 和 new_makefood.py 没什么不同，但是当这两个程序被当作模块 import 时，呈现的效果就会不同。

下列是 ch3_39.py 执行导入 old_makefood，观察执行结果。

```
1  # ch3_39.py
2  import old_makefood          # 导入模块old_makefood.py
3
4  old_makefood.make_icecream('草莓酱')
5  old_makefood.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  old_makefood.make_drink('large', 'coke')
```

执行结果

```
================= RESTART: D:\Python interveiw\ch3\ch3_39.py =================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
```

old_makefood.py的执行结果

ch3_39.py的执行结果

可以发现在导入 old_makefood.py 时，会执行第 14 ～ 16 行，所以会有上述输出，然后执行 ch3_39.py 时，会执行第 3 ～ 5 行，所以会再输出一次。

面试实例 ch3_40.py：导入 new_makefood，观察执行结果。

```
1  # ch3_40.py
2  import new_makefood          # 导入模块new_makefood.py
3
4  new_makefood.make_icecream('草莓酱')
5  new_makefood.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  new_makefood.make_drink('large', 'coke')
```

执行结果

```
================== RESTART: D:\Python interveiw\ch3\ch3_40.py ==================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
```

上述在导入 new_makefood.py 时，因为 __name__ 是 __new_makefood__，所以不会执行 main( )，此程序直接执行第 3 ～ 5 行，所以只输出一次。

面试实例 ch3_41.py：如果有 A、B、C 三个变量供 test1.py、test2.py 和 test3.py 使用，应该如何设计程序？

解答：

建立一个存储 A、B、C 变量的模块，然后在 test1.py、test2.py 和 test3.py 中分别导入此模块，则可以共享 A、B、C 变量。

程序实例 ch3_41.py：建立一个存储 A、B、C 变量的模块。

```
# ch3_41.py
A = 1
B = 2
C = 3
```

程序实例 test1.py：导入 ch3_41.py。

```
# test1.py
import ch3_41          # 导入模块ch3_41.py

print(ch3_41.A, ch3_41.B, ch3_41.C)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/test1.py ==================
1 2 3
```

程序实例 test2.py：导入 ch3_41.py。

```
# test2.py
import ch3_41          # 导入模块ch3_41.py

print(ch3_41.A, ch3_41.B, ch3_41.C)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/test2.py ==================
1 2 3
```

程序实例 test3.py：导入 ch3_41.py。

```
# test3.py
import ch3_41          # 导入模块ch3_41.py

ch3_41.A = ch3_41.B + ch3_41.C

print(ch3_41.A, ch3_41.B, ch3_41.C)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/test3.py ==================
5 2 3
```

面试实例 ch3_42.py：请说明 __init__ 和 __new__ 的区别，用文字和实例解说。

解答：

其实很多人都以为 __init__ 是建构方法（constructor），其实 __init__ 只是初始化对象时的初始化方法。真正的建构方法是 __new__，当宣告一个类对象时，最先被调用的方法是 __new__，这个方法可以建立一个类对象，当类对象被建立后，才开始调用 __init__ 进行对象初始化。

下列是用程序实例了解 __init__ 和 __new__ 的区别。

```
# ch3_42.py
class Person(object):
    def __new__(cls, name, age):
        print('__new__ is called, name={}, age={}'.format(name, age))
        instance = object.__new__(cls)
        return instance

    def __init__(self, name, age):
        print('__init__ is called, name={}, age={}'.format(name, age))

p = Person('Peter', 59)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/ch3_42.py ==================
__new__ is called, name=Peter, age=59
__init__ is called, name=Peter, age=59
```

面试实例 ch3_43.py：请同时用文字和程序实例说明 isinstance( )。

解答：

isinstance( ) 函数可以回传对象的类是否属于某一类，它包含 2 个参数，语法如下：

```
isinstance( 对象 , 类 )              # 可回传 True 或 False
```

如果对象的类是属于第 2 个参数类或属于第 2 个参数的子类，则回传 True，否则回传 False。

下列是一系列 isinstance( ) 函数的测试。

```python
# ch3_43.py
class Grandfather():
    """ 定义祖父类 """
    pass

class Father(Grandfather):
    """ 定义父亲类 """
    pass

class Ivan(Father):
    """ 定义Ivan类 """
    def fn(self):
        pass

grandfa = Grandfather()
father = Father()
ivan = Ivan()
print("ivan属于Ivan类: ", isinstance(ivan, Ivan))
print("ivan属于Father类: ", isinstance(ivan, Father))
print("ivan属于GrandFather类: ", isinstance(ivan, Grandfather))
print("father属于Ivan类: ", isinstance(father, Ivan))
print("father属于Father类: ", isinstance(father, Father))
print("father属于Grandfather类: ", isinstance(father, Grandfather))
print("grandfa属于Ivan类: ", isinstance(grandfa, Ivan))
print("grandfa属于Father类: ", isinstance(grandfa, Father))
print("grandfa属于Grandfather类: ", isinstance(grandfa, Grandfather))
```

执行结果

```
================ RESTART: D:\Python interveiw\ch3\ch3_43.py ================
ivan属于Ivan类:  True
ivan属于Father类:  True
ivan属于GrandFather类:  True
father属于Ivan类:  False
father属于Father类:  True
father属于Grandfather类:  True
grandfa属于Ivan类:  False
grandfa属于Father类:  False
grandfa属于Grandfather类:  True
```

面试实例 ch3_44.py 和 ch3_45.py：请同时用文字和程序实例说明 __str__( ) 方法。

解答：

这是类的特殊方法，一般在类中打印对象时，默认是打印对象的地址，可以协助返回易读取的字符串。加上 __str__( ) 方法，可以打印对象的属性信息。

在没有定义 __str__( ) 方法的情况下，列出类的对象。

```python
# ch3_44.py
class Name:
    def __init__(self, name):
        self.name = name

a = Name('Hung')
print(a)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_44.py =================
<__main__.Name object at 0x02ED7BB0>
```

在没有定义 __str__( ) 方法时，我们获得了一个不太容易阅读的结果。

面试实例 ch3_45.py：定义 __str__( ) 方法，重新设计上一个程序。

```
1  # ch3_45.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5      def __str__(self):
6          return '%s' % self.name
7
8  a = Name('Hung')
9  print(a)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch3/ch3_45.py =================
Hung
```

上述定义了 __str__( ) 方法后，就得到一个适合阅读的结果了。对于上述程序而言，如果我们在 Python Shell 窗口输入 a，将一样获得不容易阅读的结果。

```
================= RESTART: D:/Python interveiw/ch3/ch3_45.py =================
Hung
>>> a
<__main__.Name object at 0x02C27BF0>
```

原因是，如果只在 Python Shell 窗口输入类变量 a，系统是调用 __repr__( ) 方法做响应，为了获得容易阅读的结果，我们要定义此方法，请参考 __repr__( ) 的说明。

面试实例 ch3_46.py：请同时用文字和程序实例说明 __repr__( ) 方法。

解答：

定义 __repr__( ) 方法，此方法内容与 __str__( ) 相同，所以可以用等号取代。

```
1  # ch3_46.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5      def __str__(self):
6          return '%s' % self.name
7      __repr__ = __str__
8
9  a = Name('Hung')
10 print(a)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/ch3_46.py ==================
Hung
>>> a
Hung
```

面试实例 ch3_47.py：斐波那契数列的起源最早可以追溯到 1150 年印度数学家 Gopala，在西方最早研究这个数列的是意大利科学家列昂纳多·斐波那契（Leonardo Fibonacci），后来人们将此数列简称为费式数列。

请设计递归函数 fib（n），产生前 10 个费式数列数字，fib（n）的 n 主要是此数列的索引，费式数列的规则如下：

$F_0 = 0$ # 索引是 0

$F_1 = 1$ # 索引是 1

…

$F_n = F_{n-1} + F_{n-2}$ (n >= 2) # 索引是 n

最后值应该是 0，1，1，2，3，5，8，13，21，34，…

这个程序会产生小于 100 的数列值。

```
# ch3_47.py
class Fib():
    def __init__(self, max):
        self.max = max

    def __iter__(self):
        self.a = 0
        self.b = 1
        return self

    def __next__(self):
        fib = self.a
        if fib > self.max:
            raise StopIteration
        self.a, self.b = self.b, self.a + self.b
        return fib
for i in Fib(100):
    print(i)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/ch3_47.py ==================
0
1
1
2
3
5
8
13
21
34
55
89
```

面试实例 ch3_48.py：有一个列表如下：

```
data = [1,7,3,9,2]
```

不使用 sort( ) 方法，请设计一个函数可以将列表排序，下列是排序结果。

```
[1,2,3,7,9]
```

```
# ch3_48.py
def rtn_min(lst):
    min_ = min(lst)
    sort_lst.append(min_)
    lst.remove(min_)
    if lst:
        rtn_min(lst)
    return sort_lst

sort_lst = []
data = [1, 7, 3, 9, 2]
print(rtn_min(data))
```

执行结果

```
================== RESTART: D:/Python interveiw/ch3/ch3_48.py ==================
[1, 2, 3, 7, 9]
```