

函数

函数概念

什么是函数

我们经常需要在同一个程序里多次复用相同的代码，函数可以很好的帮助我们完成这一点。我们在函数里写我们要重复做的事，然后我们在任何需要的时候调用它。函数有参数和返回值，在函数内部对参数进行处理，并把处理结果返回给调用者。

内置函数

内置函数就是 Python 解释器中不用引入任何包，一直可以使用的函数。我们已经在前面的实验中用到了一些内置的函数，比如 `len()`，`type()`。下表列举了 Python 中全部的内置函数，共计 68 个。

Python 内置函数 (3.6.4)				
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	 实验楼 shiyancelou.com
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

关于以上 Python 3 的内置函数说明，可以参考：《Python 3 内置函数官方说明文档》(<https://docs.python.org/3/library/functions.html>)。

在项目开发中，最常使用的仍然是自己定义的函数。

知识点

- 函数的概念
- 函数的定义与调用
- 变量作用域

- 函数的五种参数
- 楼之Python实战第10期 (/courses/1190)
- 函数中修改参数值

定义和调用

我们使用关键字 `def` 来定义一个函数，定义函数后需要在函数名的括号中写上参数，最后加：`:`，再换行输入函数内部的代码，注意函数内部代码的缩进：

```
def functionname(params):  
    statement1  
    statement2
```

我们写一个函数接受一个字符串和一个字母作为参数，并将字符串中出现的该字母的数量作为返回值，回忆下先前的知识，我们提到过字符串是一个特殊的列表，列表中可以使用 `count()` 函数返回指定元素的数量。

```
>>> def char_count(str_, char):  
...     return str_.count(char)
```

第二行的 `return` 关键字，我们把 `str_` 中包含 `char` 的次数返回给调用者。

关于函数的返回值

如上面的代码所示，Python 函数由 `return` 语句来定义函数的返回值。每个函数都必须有返回值，如果不写 `return` 语句，则默认返回值为 `None`。或者在函数末行写

```
return
```

这样，`return` 后不写任何代码，则返回值也是 `None`。

如何使用函数呢，我们必须像下面这样调用这个函数：

```
>>> char_count('shianlou.com', 'o') # 运行函数，返回值为 2  
2  
>>> result = char_count('shianlou.com', 's') # 运行函数，返回值为 1 并赋值给变量 result  
>>> result  
1
```

其中，`result` 变量用来保存函数的返回值，传入的两个参数分别是用来检测的字符串和字母。

现在我们希望改变这个 `char_count()` 函数，接受一个参数，并将所有的字母及出现的频次打印出来，这个程序我们实现在一个 Python 脚本文件中。

首先使用 sublime 或 vim 等编辑器创建文件，在 Xfce 终端中输入下面的命令：

🔗 楼+之Python实战第10期 (/courses/1190)

```
$ cd /home/shiyanlou
$ vim count_str.py
```

依此输入下面的代码：

```
#!/usr/bin/env python3

def char_count(str_):
    char_list = set(str_)
    for char in char_list:
        print(char, str_.count(char))

if __name__ == '__main__':

    s = input("Enter a string: ")

    char_count(s)
```

输入后保存并执行程序，程序会要求你输入一个字符串，并将打印出当前字符串中所有字符出现的频次。

```
$ python3 count_str.py
Enter a string: shiyanlou.com
a 1
i 1
c 1
y 1
h 1
l 1
o 2
. 1
u 1
s 1
n 1
m 1
```

现在我们详细说下这个程序：

1. 第一行的内容是说明需要使用 Python 3 的解释器执行当前的脚本
2. 函数 char_count 没有返回值，就是没有 return 关键字，这是允许的，返回值和参数对于函数都是可选的
3. char_count 中首先使用集合获得字符串中所有不重复的字符集
4. 然后再使用 for 对集合进行遍历，每个字符都使用上一个例子中用到的 str.count() 计算频次
5. 最后函数中使用 print 打印字符和对应的频次
6. if __name__ == '__main__': 这一句相当于 C 语言的 main 函数，作为程序执行的入口，实际的作用是让这个程序 python3 count_str.py 这样执行时可以执行到 if __name__ == '__ma

❶ `in_`: 这个代码块中的内容, 当通过 `import count_str` 作为模块导入到其他代码文件时不会执行 `if __name__ == '__main__':` 中的内容。

另外, 需要注意的是这是一个效率低下的程序, 你发现了吗? 另外是否有改进的思路? 欢迎在 QQ 群中与团队和助教讨论。这里是参考答案链接, 供参考:

<https://www.shiyanlou.com/questions/45207>

(<https://www.shiyanlou.com/questions/45207>)。

练习题：定义函数

在 `/home/shiyanlou` 目录下创建代码文件 `dicttest.py` :

```
$ cd /home/shiyanlou/  
$ touch dicttest.py
```

在这个文件中, 我们需要实现以下需求:

1. 执行程序可以输入多个命令行参数
2. 每个命令行参数中间都有一个冒号, 需要使用字符串的 `split()` 进行切分并存储到字典中
3. 按照示例的格式要求输出重新处理后的数据

例如:

```
$ cd /home/shiyanlou  
$ python3 dicttest.py 100:shiyan 101:louplus 102:jack 103:lee  
ID:100 Name:shiyan  
ID:103 Name:lee  
ID:101 Name:louplus  
ID:102 Name:jack
```

注意字典是无序的, 所以可以不必严格按照参数输入的顺序输出。

程序的代码框架如下, 请在代码中完善函数 `handle_data()` 和 `print_data()` :

```
#!/usr/bin/env python3  
  
output_dict = {}  
  
if __name__ == '__main__':  
    for arg in sys.argv[1:]:  
        handle_data(arg)  
  
    for key in output_dict:  
        print_data(key, output_dict[key])
```

`handle_data()` 用来处理参数并将处理的结果存入到 `output_dict` , `print_data()` 用来按照格式打印输出。

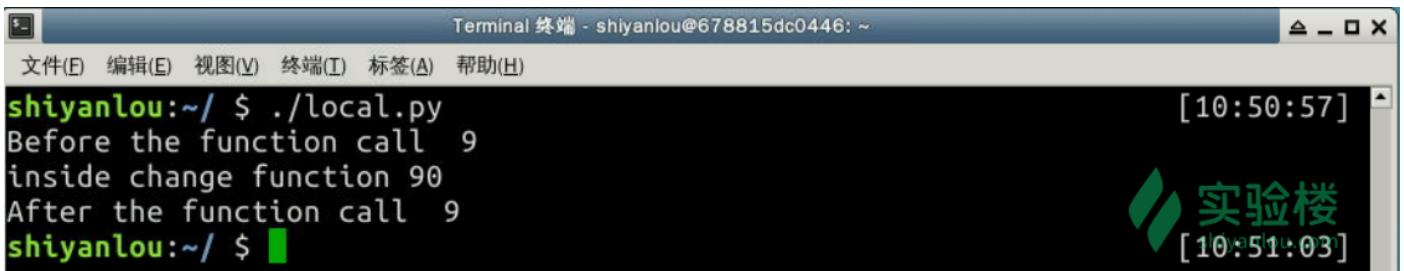
程序完成后, 点击 下一步 , 系统将自动检测完成结果。

变量作用域

我们通过几个例子来弄明白局域或全局变量, 首先我们在函数内部和函数调用的代码中都使用同一个变量 `a` , 将下方代码写入 `/home/shiyanlou/local.py` :

```
#!/usr/bin/env python3
def change():
    a = 90
    print(a)
a = 9
print("Before the function call ", a)
print("inside change function", end=' ')
change()
print("After the function call ", a)
```

运行程序：



首先我们对 `a` 赋值 9, 然后调用更改函数, 这个函数里我们对 `a` 赋值 90, 然后打印 `a` 的值。调用函数后我们再次打印 `a` 的值。

当我们在函数里写 `a = 90` 时, 它实际上创建了一个新的名为 `a` 的局部变量, 这个变量只在函数里可用, 并且会在函数完成时销毁。所以即使这两个变量的名字都相同, 但事实上他们并不是同一个变量。

那么如果我们先定义 `a` , 在函数中是否可以直接使用呢?

例如下面这段代码：

```
#!/usr/bin/env python3
a = 9
def change():
    print(a)
change()
```

这段代码是没有问题的, 可以直接打印输出 9。稍微改动一下：

```
#!/usr/bin/env python3
# 楼之Python实战第10期 (/courses/1190)
a = 9
def change():
    print(a)
    a = 100
change()
```

现在就会报错了：“UnboundLocalError: local variable 'a' referenced before assignment”，原因是当函数中只要用到了变量 `a`，并且 `a` 出现在表达式等于号的前面，就会被当作局部变量。当执行到 `print(a)` 的时候会报错，因为 `a` 作为函数局部变量是在 `print(a)` 之后才定义的。

现在我们使用 `global` 关键字，对函数中的 `a` 标志为全局变量，让函数内部使用全局变量的 `a`，那么整个程序中出现的 `a` 都将是这个：

```
#!/usr/bin/env python3
a = 9
def change():
    global a
    print(a)
    a = 100
print("Before the function call ", a)
print("inside change function", end=' ')
change()
print("After the function call ", a)
```

程序中的 `end=' '` 参数表示，`print` 打印后的结尾不用换行，而用空格。默认情况下 `print` 打印后会在结尾换行。

程序执行的结果，不会报错了，因为函数体内可以访问全局的变量 `a`：

```
Before the function call  9
inside change function 9
After the function call  100
```

在函数内使用 `global` 会有什么作用呢？尝试下面的代码：

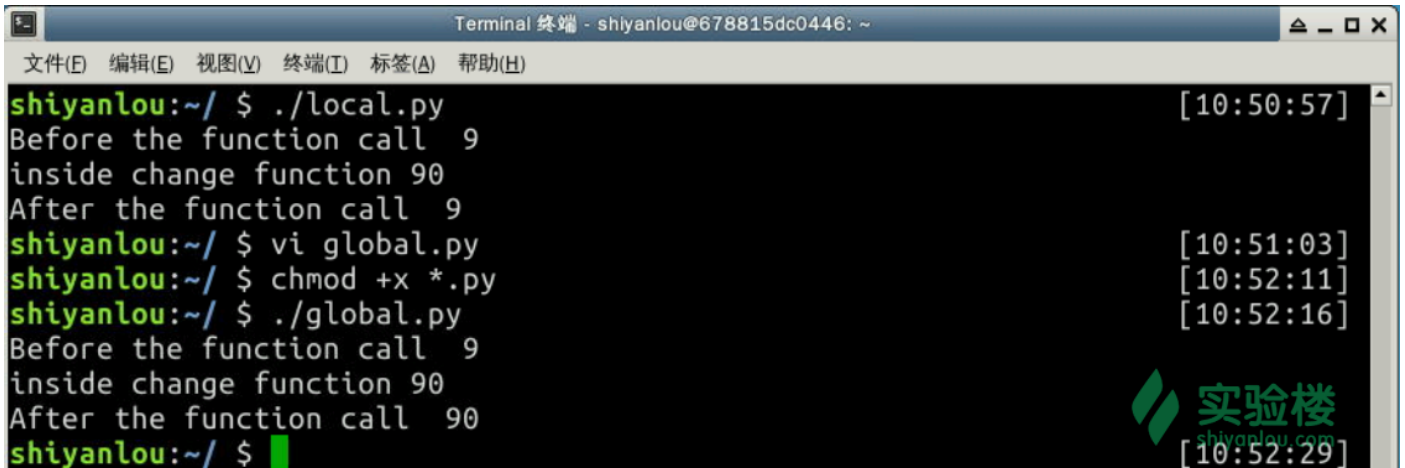
```
#!/usr/bin/env python3
def change():
    global a
    a = 90
    print(a)
a = 9
print("Before the function call ", a)
print("inside change function", end=' ')
change()
print("After the function call ", a)
```

程序执行的结果：

```
Before the function call 9
inside change function 90
After the function call 90
```

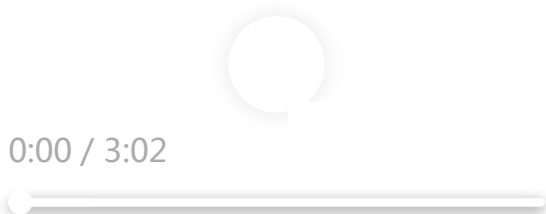
这里通过关键字 `global` 来告诉 `a` 的定义是全局的，因此在函数内部更改了 `a` 的值，函数外 `a` 的值也实际上更改了。

运行程序：



```
Terminal 终端 - shiyanlou@678815dc0446: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./local.py [10:50:57]
Before the function call 9
inside change function 90
After the function call 9
shiyanlou:~/ $ vi global.py [10:51:03]
shiyanlou:~/ $ chmod +x *.py [10:52:11]
shiyanlou:~/ $ ./global.py [10:52:16]
Before the function call 9
inside change function 90
After the function call 90
shiyanlou:~/ $
```

变量作用域讲解视频：



练习题：修复函数 BUG

在 `/home/shiyanlou` 目录下创建代码文件 `funcbugtest.py`：

```
$ cd /home/shiyanlou/
$ touch funcbugtest.py
```

`funcbugtest.py` 的作用很简单，就是计算从 1(start) 加到 100(end) 的和，并输出。

向 `funcbugtest.py` 文件写入以下代码内容：

🔗 楼+之Python实战第10期 (/courses/1190)

```
result = 0
start = 1
end = 100

def compute():
    while start < end:
        result += start
        start += 1
    print(result)

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

预期输出的是 5050，可是程序有 BUG，没有正确输出，请修复程序，让程序可以正常执行。

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

函数的参数

Python 常用参数有四种：必选参数、默认参数、可变参数和关键字参数。

以上四种参数可复合使用，切记：复合使用时各类参数的定义顺序须同上。

Python 函数中的参数传递有几个问题要特别小心，这些问题是很容易出现 BUG 的地方，现在我们依次说明。

必选参数

此类参数最为常见，使用函数时必选参数可以不写参数名，但必须对其赋值，参数赋值顺序是定义的时候指定的，所以必选参数又称为位置参数。如果不按照定义顺序传参，就要使用参数名进行传参。举例说明，在交互环境中，实现一个连接服务器的程序，需要给出服务器的 IP 地址和端口号作为参数：


```
>>> def connect(ipaddress, port):
...     print("IP: ", ipaddress)
...     print("Port: ", port)
...
>>> connect('192.168.1.1', 22)
IP: 192.168.1.1
Port: 22
>>> connect(22, '192.168.1.1')
IP: 22
Port: 192.168.1.1
>>> connect(port=22, ipaddress='192.168.1.1')
IP: 192.168.1.1
Port: 22
>>>
```

上面的例子中尝试了三次传参：第一次使用必选参数默认的顺序，结果正确；第二次使用错误的顺序，结果错误；第三次虽然顺序错误但使用了参数名传参，所以结果正确。

默认参数

函数的参数可以设定默认值，这种参数称为默认参数。调用函数时，如果我们对默认参数没有赋值，则会自动赋其默认值。改变上面的程序，将端口号设为默认参数，默认值为 22：

```
>>> def connect(ipaddress, port=22):
...     print("IP: ", ipaddress)
...     print("Port: ", port)
...
>>>
```

这表示如果调用者未给出 port 的值，那么 port 的值默认为 22。这是一个关于默认参数的非常简单的例子。

可以通过调用函数测试代码：

```
>>> connect('192.168.1.1', 2022)
IP: 192.168.1.1
Port: 2022
>>> connect('192.168.1.1')
IP: 192.168.1.1
Port: 22
>>>
```

有两点需要注意：一是前文提到的各类参数的顺序问题，默认参数后面不能再有必选参数，例如 `f(a,b=90,c)` 就是错误的；二是默认参数的默认值必须设为不可变的数据类型（如字符串、元组、数字、布尔值、None 等）。下面的代码是错误的示例，默认参数的默认值是空列表，因为列表是可变对象，调用函数后，默认参数的默认值会改变：

```
>>> def f(a, data=[]):
...     data.append(a)
...     return data
...
>>> print(f(1))
[1]
>>> print(f(2))
[1, 2]
>>> print(f(3))
[1, 2, 3]
```

要避免这个问题，可以像下面这样定义默认值：

```
>>> def f(a, data=None):
...     if data == None:
...         data = []
...     data.append(a)
...     return data
...
>>> print(f(1))
[1]
>>> print(f(2))
[2]
```

这里两个函数的区别体现在执行第二次的时候。上一个函数的默认参数 `data = []`，第二次调用该函数时 `data` 默认值会受第一次执行结果的影响而发生改变，所以造成每次默认值是不同的。而下一个函数就没有这个问题，因为默认参数的默认值是不可变对象，所以无论执行多少次，默认值都不会变。默认参数讲解视频：

0:00 / 3:43

可变参数

如果一个函数需要传入的参数数量不固定，可能是零个、一个，也可能是 N 个，如何处理？这种情况可以为函数设置一个可变参数。顾名思义，可变参数意味着参数的数量是可变的，调用函数时，在可变参数的位置直接传入一个元组或者任意数量的参数。可变参数的定义格式是在参数名前面加上 `*`，参数名可以自定义，通常写成 `*args`。注意：在函数体内部使用该参数时，前面不要加 `*`。举例说明，我们的 `connect` 函数要连接目标服务器的多个端口号：

```
> def porttest(ipaddress, ports):  
...     print("IP: ", ipaddress)  
...     for port in ports:  
...         print("Port: ", port)  
...
```

调用函数时可以传递零个或多个端口号，或一个端口号的元组做参数：

```
>>> connect('192.168.1.1')  
IP: 192.168.1.1  
>>>  
>>> connect('192.168.1.1', 22, 23, 24)  
IP: 192.168.1.1  
Port: 22  
Port: 23  
Port: 24  
>>> connect('192.168.1.1', 22)  
IP: 192.168.1.1  
Port: 22  
>>> params = (25, 26, 27)  
>>> connect('192.168.1.1', *params)  
IP: 192.168.1.1  
Port: 25  
Port: 26  
Port: 27
```

调用含可变参数的函数时，如果在可变参数的位置传入多个参数而不是元组，那么这些参数会自动生成一个元组传入函数。

可变参数讲解视频：



0:00 / 1:34

关键字参数

以上三种函数的参数（必选参数、默认参数、可变参数）在赋值时都可以不写参数名，而关键字参数允许传入零个或任意多个带参数名的参数，其中参数名可自定义，这些关键字参数会在函数内部自动生成一个字典，用来扩展函数的功能。关键字参数的定义格式是在参数名前面加上 `**`，参数名可以自定义，通常写成 `**kw`。注意：在函数体内部使用该参数时，前面不要加 `**`。示例如下：

```

def connect(ipaddress, ports, *params, **kw):
    print("IP: ", ipaddress)
    for port in ports:
        print("Port: ", port)
    for key, value in kw.items():
        print('{}: {}'.format(key, value))

```

和可变参数类似，对关键字参数传参时可以使用字典：

```

>>> ipaddress = '192.168.1.1'
>>> params = (25, 26, 27)
>>> prop = {'device': 'eth0', 'proto': 'static'}
>>> connect(ipaddress, *params, **prop)
IP: 192.168.1.1
Port: 25
Port: 26
Port: 27
device: eth0
proto: static

```

当然也可以直接传关键字参数，就是带参数名的参数，结果一样：

```

>>> connect(ipaddress, *params, device='eth0', proto='static')
IP: 192.168.1.1
Port: 25
Port: 26
Port: 27
device: eth0
proto: static

```

命名关键字参数

以上四种是 Python 函数中常见常用的参数类型，下面简单介绍一下第五种不常用但需要了解的参数类型：命名关键字参数。前文提到，必选参数、默认参数、可变参数在赋值时都可以不写参数名，而命名关键字参数恰好相反：赋值时必须写上参数名。此参数的特征就是前面有一个用逗号隔开的 *。例如我们定义函数 `def test(m, *, a, b)`，其中 `m` 为必选参数，`a` 和 `b` 就是命名关键字参数，用户调用函数时，每一个命名关键字参数都必须使用相应的参数名赋值，否则会抛出 `TypeError`：

```
楼+之Python实战第10期 (/courses/1190)
...     print("Hello", name)
...
>>> hello('shianlou')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: hello() takes 0 positional arguments but 1 was given
>>> hello(name='shianlou')
Hello shianlou
```

- 拓展阅读：深入 Python 函数定义
(<http://www.pythondoc.com/pythontutorial3/controlflow.html?highlight=python%20%E5%87%BD%E6%95%B0%E5%AE%9A%E4%B9%89#tut-defining>)

函数中修改参数值

在函数中是否可以改变传递的参数值？

在 C/C++ 语言中有传值和传引用（指针）的概念，直接影响到函数是否能够改变参数的值。

函数参数传值的意思是函数调用过程中，在函数内部使用到的参数只是一个局部变量，在函数执行结束后就销毁了。不影响调用该函数的外部参数变量的值。

函数参数传引用的意思是传递给函数的参数就是外部使用的参数，函数执行过程中对该参数进行的任何修改都会保留，当函数调用结束后，这个参数被其他代码使用中都是函数修改过后的数据。

但在 Python 中情况有些不同，Python 函数的参数是没有类型的，可以传递任意类型的对象作为参数。但不同类型的参数在函数中，有的可以修改（例如列表对象），有的不可以修改（例如字符串对象）。

举例说明：

🔗 楼+之Python实战第10期 (/courses/1190)

```
def connect(ipaddress, ports):  
    print("IP: ", ipaddress)  
    print("Ports: ", ports)  
    # 此处实际是创建了一个新的局部变量，并不是修改了 ipaddress 的值  
    ipaddress = '10.10.10.1'  
    # 修改 ports 列表的值，使用 append 向列表中增加一个元素，会影响到传入的列表 ports 的值  
    ports.append(8080)  
  
if __name__=="__main__":  
    ipaddress = '192.168.1.1'  
    ports = [22,23,24]  
    print("Before connect:")  
    print("IP: ", ipaddress)  
    print("Ports: ", ports)  
    print("In connect:")  
    connect(ipaddress, ports)  
    print("After connect:")  
    print("IP: ", ipaddress)  
    print("Ports: ", ports)
```

执行这个程序的输出结果：

```
Before connect:  
IP: 192.168.1.1  
Ports: [22, 23, 24]  
In connect:  
IP: 192.168.1.1  
Ports: [22, 23, 24]  
After connect:  
IP: 192.168.1.1  
Ports: [22, 23, 24, 8080]
```

可以发现在函数中改变了 ipaddress 的值没有起到效果，但 ports 列表的值却改变了。

Python 中的对象有不可变对象，指的是数值、字符串、元组等，和可变对象，指的是列表、字典等。如果是不可变对象作为参数，函数中对该参数的修改只能用等号赋值，实际上是创建了一个新的局部变量。如果是可变对象作为参数，函数中的修改会保留。

函数中修改参数值讲解视频：

0:00 / 2:27

练习题：修复列表参数的 BUG

楼+之Python实战第10期 (7/courses/1190)

在 /home/shiyanlou 目录下创建代码文件 listbugtest.py：

```
$ cd /home/shiyanlou/  
$ touch listbugtest.py
```

listbugtest.py 的作用很简单，就是一个列表（base）中所有数字和一个数字（value）的和，并输出。

向 listbugtest.py 文件写入以下代码内容：

```
#!/usr/bin/env python3  
  
def compute(base, value):  
    base.append(value)  
    result = sum(base)  
    print(result)  
  
if __name__ == '__main__':  
    testlist = [10, 20, 30]  
    compute(testlist, 15)  
    compute(testlist, 25)  
    compute(testlist, 35)
```

预期输出的是：

```
$ python3 /home/shiyanlou/listbugtest.py  
75  
85  
95
```

但现在的程序输出的是：

```
$ python3 /home/shiyanlou/listbugtest.py  
75  
100  
135
```

程序有 BUG，没有正确输出，请修复程序，让程序可以正常执行。

程序完成后，点击 下一步 ，系统将自动检测完成结果。

总结

本节实验没有需要提交到 Github 代码仓库中的代码，如果你觉得有哪些代码需要保存，可以自行提交。后续较大的示例代码、项目实验及挑战的代码我们都会要求你提交到自己的 Github 中保存。

经过本实验应当知道如何定义函数，局域变量和全局变量一定要弄清楚，参数默认值、传递参数的各种情况也需要掌握。

另外，其它高级语言常见的*函数重载*，Python 是没有的，这是因为 Python 有默认参数这个功能，*函数重载*的功能大都可以使用默认参数达到。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：列表、元组、集合与字典 (/courses/1190/labs/8520/document)

下一节：挑战：完善工资计算器 (/courses/1190/labs/8522/document)