Python中不支持类似C++的函数重载：

例：程序function\_test/function\_test1.py

def greet\_user():  
 *"""显示简单的问候语"""* print('Hello!')  
  
  
def greet\_user(username):  
 *"""显示简单的问候语"""* print('Hello! ' + str(username).title())  
  
  
greet\_user()  
greet\_user('Jame')

Error:

Traceback (most recent call last):

File "C:/study/PythonStudy/function\_test/function\_test1.py", line 17, in <module>

greet\_user()

TypeError: greet\_user() missing 1 required positional argument: 'username'

但可以像如下方式来处理：

def greet\_user():  
 *"""显示简单的问候语"""* print('Hello!')  
  
  
greet\_user()  
  
  
def greet\_user(username):  
 *"""显示简单的问候语"""* print('Hello! ' + str(username).title())  
  
  
greet\_user('Jame')

输出为：

Hello!

Hello! Jame

在定义了def greet\_user(username):  
后，之前定义的def greet\_user():就失效了。

函数中参数的传递

例：function\_test/function\_test4.py

def modify(para):  
 if isinstance(para, int):  
 print("para is int.")  
 para = 5  
 elif isinstance(para, str):  
 print("para is string.")  
 para = 'Jam'  
 elif isinstance(para, list):  
 print("para is list.")  
 # para = ['hamster']  
 para.append('cat')  
 print("In modify() para is modified: " + str(para))  
  
  
parameter = 1  
print("before modify(): " + str(parameter))  
modify(parameter)  
print("after modify(): " + str(parameter))  
print()  
  
parameter = 'Tom'  
print("before modify(): " + str(parameter))  
modify(parameter)  
print("after modify(): " + str(parameter))  
print()  
  
parameter = ['dog']  
print("before modify(): " + str(parameter))  
modify(parameter)  
print("after modify(): " + str(parameter))

输出为：

before modify(): 1

para is int.

In modify() para is modified: 5

after modify(): 1

before modify(): Tom

para is string.

In modify() para is modified: Jam

after modify(): Tom

before modify(): ['dog']

para is list.

In modify() para is modified: ['dog', 'cat']

after modify(): ['dog', 'cat']

从输出可以看出，传递给modify()数字和字符串时，函数内部修改，不会影响到外部实参，传递列表给函数时，函数内部修改了列表的元素，会影响外部的实参。

参考：Python学习3-对象与类型.docx：

传递实参给函数的形参时，实际是将两个变量引用到同一个对象，Python中的引用类似C语言中的指针，对于不可变对象，函数内部的修改相当于改变了其引用的对象，类似C语言修改了指针的指向，所以不会影响外部的实参；对于可变对象，如果在函数内部修改了其指向的元素，就会影响到外部的实参，如果直接改变其引用的对象，也不会影响外部的实参。当传递列表给modify()函数，如果直接使用语句：

para = ['hamster']

就不会影响外部的实参parameter。

根据以上情况，在使用默认参数时，尽量不要使用默认的可变对象，这有可能会产生预料之外的结果。

例：function\_test/function\_test3.py