一、函数基础

想一想，下面这段代码打印一个美女。如果在程序中不同地方打印美女呢？那是不是没一个地方都要写一段？要解决这种代码重复的问题，函数上场了！

```
print("                      .::::.")
print("                    .::::::::.")
print("                   ::::::::::::")
print("               ..::::::::::::'")
print("             '::::::::::::'")
print("               .::::::::::")
print("           '::::::::::::::..")
print("                ..::::::::::::.")
print("              ``::::::::::::::::")
print("               ::::``:::::::::'        .:::.")
print("              ::::'   ':::::'       .::::::::.")
print("            .::::'      ::::     .::::::::':::::.")
print("           .:::'       :::::  .::::::::' ':::::.")
print("          .::'        :::::.:::::::::'      ':::::.")
print("         .::'         ::::::::::::::'         ``::::.")
print("     ...:::           ::::::::::::'              ``::.")
print("     ''''':.          '::::::::::'                  ::::..")
print("                       '.::::::'                    ':'''''..")
```

1、函数的定义:

def 关键字开头，后接函数标识符名称和圆括号():

```
# 函数定义，def 关键字定义一个函数
def 函数名():
    代码块
```

2、函数的调用

```
# 函数的调用，直接函数名加()调用
#
# 函数名()

# 定义一个叫function 的函数
def function():
    '''打印一句话'''
    print('wo shi ligang')

# 调用函数function, 调用函数会执行函数内部代码
function()

# 输出 'wo shi ligang'
```

3、函数的说明文档

在函数下面写的一段字符串形式的函数功能说明。可以用 help(函数名)查看

```
def function():
    """打印一hello world"""    #函数的文档说明，用来说明这个函数的功能
    print('hello world')

help(function)    # 查看函数的文档说明
```

4、总结

在编写程序的过程，有某一功能代码块出现多次， 但是为了提高编写的效率以及代码的重用，所以把具有独立功能的代码块组织为一个小模块，这就是函数。

二、函数的参数

1、想一想，定义一个函数计算两个数的和。下面这段代码有什么缺陷？

```
def addnum():
    '''定义一个函数计算两个数的和'''
    a = 1
    b = 2
    c = a + b
    print(c)

# 调用函数
addnum()
```

2、为了让这个函数更加通用，这里引入函数的参数。即想让它计算哪两个数的和，就让它计算哪两个数的和，而不是一个函数只能固定计算两个数，在定义函数的时候让这个函数接收调用者传过来的数据，这样这个函数就可以计算任意两个数的和了，这就是函数的参数。

```
def addnum(a,b):  # 括号内的a,b 是形式参数
    c = a + b
    print(c)

addnum(1,2)  # 调用函数时传的是，实际参数
# 输出 3
addnum(3,6)
# 输出 9
```

3、函数传参是的顺序，默认是按位置将实参传给形参。

```
def addnum(a,b):
    print('参数 a=%s'%a)
    print('参数 b=%s'%b)
```

```
addnum(1,2)
print('-'*20)
addnum(2,1)
#输出结果

参数 a=1
参数 b=2
--------------------
参数 a=2
参数 b=1
```

4、指定参数传参，

```
def addnum(a,b):
    print('参数 a=%s'%a)
    print('参数 b=%s'%b)

addnum(1,b=2)

print('-'*20)  # 打印分割线

addnum(b=1,a=2)

#输出结果
参数 a=1
参数 b=2
--------------------
参数 a=2
参数 b=1
```

注意指定形参的方式传参数只能放在后面。如下图：

1、缺省参数

缺省参数，在调用函数时如果没有传参数，那么会使用定义函数时给的缺省值。缺省参数必须在参数列表的最后面，否则会报错。

```
def function(a,b=1):
    print('参数 a=%s'%a)
    print('参数 b=%s'%b)

function(5)
#输出结果
#参数 a=5
#参数 b=1
```

2、不定长参数

一个函数有时候会处理比当初声明的参数要多，这就是不定长参数，定义函数时不用声明参数名。

```
def function(a,b,*args,**keargs):
    pass
```

加了星号（*）的变量 args 会存放所有未命名的变量参数，args 为元组；而加**的变量 kwargs 会存放命名参数，即形如 key=value 的参数， kwargs 为字典。

```
def function(a,b,*args,**kwargs):
    print('a =',a)
    print('b =',b)
    print('args =',args)
    print('kwargs =',kwargs)
function(1,2,3,4,5,name='xiaoming',age=18)
```

执行结果：

```
a = 1
b = 2
args = (3, 4, 5)
kwargs = {'age': 18, 'name': 'xiaoming'}
```

不定长参数也可以直接传元祖，或者字典。输入结果跟上面是一样的。

```
def function(a,b,*args,**kwargs):
    print('a =',a)
    print('b =',b)
    print('args =',args)
    print('kwargs =',kwargs)
c=(3,4,5)
d={'name':'xiaoming','age':18}
function(1,2,3,4,5,*c,**d)
```

如果不加*号，他会将整个元祖或字典当成一个参数传过去。不会将里面的元素当成参数。

3、引用传参

Python 中函数参数是引用传递（注意不是值传递）。对于不可变类型，因变量不能修改，所以运算不会影响到变量自身；而对于可变类型来说，函数体中的运算有可能会更改传入的参数变量。

```
>>> a = 1
>>> b = [1,2,3]
>>> def function(a,b):
...     a+=1
...     b.append(4)
...
>>> function(a,b)
>>> a
1
>>> b
[1, 2, 3, 4]
>>>
```

通过以上代码，我们可以发现 a 不可变类型值不会因为传参后计算改变，b 可变类型值会改变。

三、函数返回值

1、定义

所谓“返回值”，就是程序中函数完成一件事情后，最后给调用者的结果

举个生活中例子： 你想要组装一台电脑，你将你的电脑配置参数传给电脑店老板，电脑店老板最后得返回一台组装好的电脑给你。你调用电脑店老板这个方法去帮你组装电脑，他总得给你返回一台组装好的电脑吧，这台电脑就是返回值。

程序中也需要返回值，比如用调用函数去计算两个数的和，最后得返回两个数相加的结果给调用者。

2、return

函数中使用 return 返回函数的计算结果。

```
def addnum(a,b):
    """计算两个数的和"""
    c = a+b
    return c    # 使用return 返回计算结果
```

3、接收返回值

```
def addnum(a,b):
    """计算两个数的和"""
    c = a+b
    return c    # 使用return 返回计算结果
result = addnum(1,2)
print(result)
```

4、多个返回值

如果函数需要返回多个结果呢？将要返回的值用逗号隔开。最终会返回一个包含所有返回值的元祖

```
def addnum(a,b):
    """计算两个数的和"""
    c = a + b
    return c,a,b  # 同时返回多个值
result = addnum(2,3)
print(result)     # 结果会返回一个元祖
```

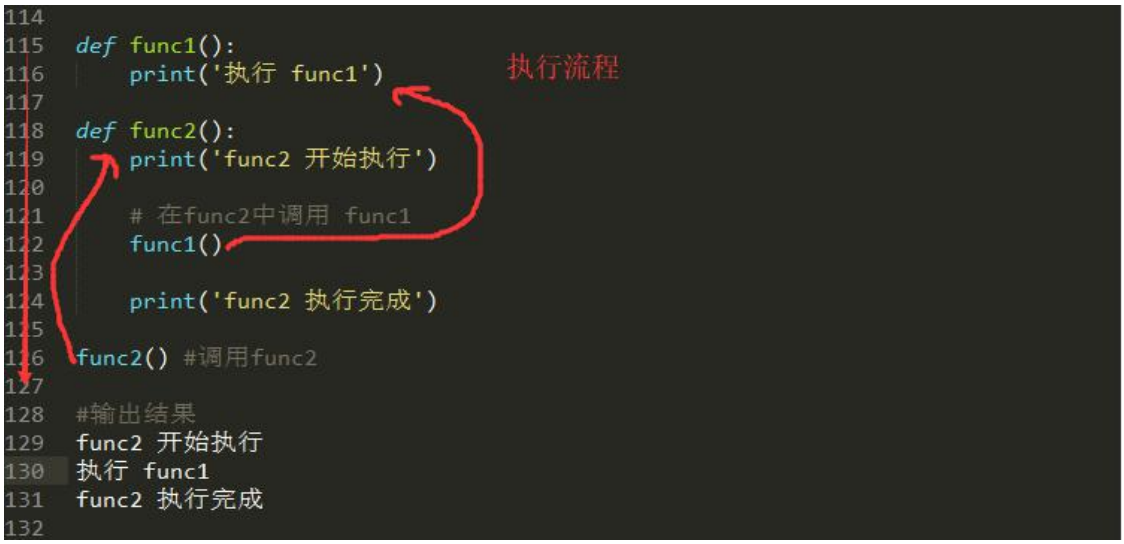
四、函数嵌套调用

1、嵌套调用

函数也是可以嵌套调用的，即在一个函数内部调用另外一个函数。

```
115  def func1():
116      print('执行 func1')
117
118  def func2():
119      print('func2 开始执行')
120
121      # 在func2中调用 func1
122      func1()
123
124      print('func2 执行完成')
125
126  func2() #调用func2
127
128  #输出结果
129  func2 开始执行
130  执行 func1
131  func2 执行完成
```

func2 中嵌套了 func1,调用 func2 的时候代码从上到下执行，遇到 func2 内部的函数 func1 时会回到 func1 函数，执行 func1 函数。func1 执行完后再回到 func1 继续执行后面的代码。



2、函数的应用

```
def card(name,position,company):

    print("-"*20)
    print('姓名：%s'%name)
    print('职位：%s'%position)
    print("公司：%s"%company)
    print("-"*20)

card('小明','python 开发工程师','美国中情局')
card('李小龙','python 开发工程师','美国中情局')
Copy
```

输出结果：

```
--------------------
姓名：小明
职位：python 开发工程师
公司：美国中情局
--------------------
--------------------
姓名：李小龙
职位：python 开发工程师
公司：美国中情局
--------------------
Copy
```

3、函数嵌套应用

```
#定义个函数计算三个数的和
def addnum(a,b,c):
```

```
    return a + b + c

# 定义一个函数求三个数的平均数
# 平均数计算 (a+b+c)/3
def average(a,b,c):
    # 先计算三个数的和在除以3，前面我们已经写了一个函数计算和这里可以直接调用

    he = addnum(a,b,c)
    avg = he/3

    return avg  #返回平均值
avg = average(1,3,5)
print(avg)
#输出结果
#3.0
```