Python基础

1. 函数

学习目标

1. 学会定义一个函数；
2. 学会调用函数

1.1 函数的定义

例1.我们已经知道，为了输出一个乘法表，我们需要输入这样的命令：

```
>>> for i in range(1, 10):
...   print
...   for j in range(1, i+1):
...     print "%d*%d=%d" % (i, j, i*j),
...
```

其结果就是一个乘法表：

```
1*1=1
2*1=2 2*2=4
3*1=3 3*2=6 3*3=9
4*1=4 4*2=8 4*3=12 4*4=16
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

如果我们在多个地方都需要打印出这样一个表，那么我们每次都要调用上面的代码。消除这种麻烦的一个方法就是：我们定义一个函数，将以上代码封装起来，以后再次需要打印这个乘法表时，我们执行这个函数就可以了。

函数的含义

- 一个函数是将一些语句集合在一起的部件，使得它们能够不止一次地在程序中运行。
- 函数还能计算出一个返回值，并能够改变作为函数输入的参数，这些参数在函数输入时可以每次都不相同。
- 函数是一个通用的程序结构部件。

基本语法

```
def <函数名> (参数1, 参数2, ...):
    """
```

```
    关于该函数的文档.
    """
    代码块
```

为什么要使用函数？

- 函数可以封装一个功能，该功能可以被多次使用从而达到代码别重复使用。
- 函数可以最小化代码冗余。
- 使用函数可以将复杂的系统分解为简单的部件。（举例）

如何定义函数

对于上述问题，我们可以定义出打印乘法表的函数：

```
def multip_tab():
  for i in range(1, 10):
    print
    for j in range(1, i+1):
      print "%d*%d=%d" % (i, j, i*j),
```

调用该函数:

```
multip_tab()
```

函数定义的举例 (终端展示)

例2. 这是一个在交互式模式下输入的定义语句。它定义了一个名为times的函数，这函数将返回两个参数乘积。

```
>>> def times(x,y):
...     result = x + y
...     return result
...
```

注意：

- 语句def标志着函数定义的开始
- 语句def后接函数名。执行def语句后，Python将生成新的函数对象，封装代码，并将其赋值给函数名
- 括号中的x,y是函数的参数，要放在函数名后的括号内
- 冒号后一行，必须缩进
- 要结束像定义这样的多行语句，需要按两次Enter键
- return将计算机得到的值传回给调用者

例3.

```
>>> def my_print():
...     print "This is a function."
```

```
...
```

1.2 函数的调用

函数的调用需要用到圆括号以及 0 个或多个参数，并且可以将返回值赋给一个变量（让一个变量引用该函数返回的对象）。例如：

```
>>> result = f(x, y)
```

在终端调用：

在def运行以后，可以在程序中通过在函数名后增加括号来调用这个函数。 括号中可以包含一个或多个对象参数。执行语句

语法：

```
函数名(参数1,参数2,...)
```

调用times(x,y)：

可以直接把数值对象传给函数

```
>>> times(10,-8)
```

这表达式传递了两个参数给times函数。函数头部的x赋值为10,

或者先把数值对象赋值给变量，再把变量作为实际参d >>> b = 11 >>> times(a,b)

注意：

在终端调用函数时，函数名前可以省略“print”.函数默认会打印返回值。

调用定义在文件中的函数

为了能够永久地保存程序，需要将代码写入文件中。这样的文件通常称为模块。

模块是一个包含了Python语句的简单文本文件。

如函数是定义在模块中，必须先导入这个函数，再调用它。导入函数的语法为

```
from 模块名 import 函数名
```

例4. 函数times()定义在模块utils.py中，调用该函数。

步骤 1 ：在Pycharm新建文件utils.py,写入

```
#coding: utf-8
def times(x,y):
    result = x * y
    return result
```

步骤 2 ：又用Pycharm在同目录下新建脚本文件run.py，写入：

```
#coding:utf-8
from utils import times

a = 4
b = 9
s = times(a,b)

print "The multiplication of {0} and {1} is {2}.".format(a,b,s)
```

步骤 3 ：点击运行run(运行)图标，可以看到运行结果如下

```
/usr/bin/python2.7 /home/huang/.PyCharmCE2017.2/config/scratches/run.py
The sum of 4 and 9 is 13.
```

深入理解

例4-a.

```
#coding:utf-8
#引入函数
from utils import times
#调用函数
a = "itsource "
b = 5
s = times(a,b)
print "The multiplication of {0} and {1} is: \n{2}".format(a,b,s)
```

注意：此处函数作用和上一个例子完全不同。

- 字符串传给x,整数5传值给y
- 在Python的定义中，我们不需要对参数和变量的类型做声明
- 我们把times用作数字的乘法或序列的重复
- times的作用取决于传递给它的值　(掌握)

1.3 函数的参数

函数的参数是一些Python对象，要放在函数名后的() 内。 函数的参数分为形式参数(形参)和实际参数(实参)。

1.3.1 形式参数

形参写在定义中，位于函数名后的括号里。如函数的定义

```
>>> def add_numbers(x,y):
...     result = x + y
...     return result
...
```

其中， x,y 是形参。

使用形参的原因:

- 对大多数函数，它们要完成其功能需要未知数据参与
- 在定义函数时此未知数据尚未确定，只有在使用该功能时未知数据才能够确定

1.3.2 实际参数

调用函数时，需要把形式参数换成实际参数

```
>>> s = add_numbers(a,b)
```

此例中，a,b是实参。

注意：

1. 实际参数要与形参的顺序一致
2. 参数可以有一个或多个，也可以没有参数
3. 如果没有参数，()需要保留

1.4 函数的返回值

1.4.1 函数返回值的含义

函数的return语句返回的一个变量的值到调用该函数的地方。 该返回值的类型就是这个变量所指向的对象的类型。

1.4.2 为什么需要函数的返回值

函数的执行结果需要被使用。 举例如下：

例5.含有return语句的函数

```
>>> def add_numbers(x,y):
...     result = x + y
...     return result
...
>>> z = add_numbers(3,5)
>>> z
8
>>> type(z)
<type 'int'>
```

函数末尾也可以没有return语句。 如果没有return语句，则函数返回None类型对象。

例6. 定义函数

```
>>> def my_print():
...     print "This is a function."
...
```

调用my_print()

```
>>>my_print()
>>>a = my_print()
>>>a
>>>type(a)
```

运行结果如下：

```
>>> my_print()
This is a function!
>>> a = my_print()
This is a function!
>>> a
>>> type(a)
<type 'NoneType'>
```

注意：

1. 函数可以没有参数,但函数名后的()不能省略
2. 如果函数末尾没有return语句，则返回None类型对象。

1.5 位置参数和关键字参数

关键字参数可以给函数的使用带来很大的灵活性。 利用关键字参数，实参的位置不再需要和形参的位置一一对应。

例7.

```
>>> def substract_numbers(x=1,y=2):
...     result = x - y
...     return result
...
>>> substract_numbers(y=3,x=2)
-1
```

注意:

1. 定义函数时形参的值为关键字参数的默认值，执行函数时形参的值可以改变
2. 运行时，实参的位置（顺序）和形参的位置（顺序）不必一致

函数中可以既有位置参数，又有关键字参数。

例8. 定义函数

```
>>> def my_function(x, y, z=2.5):
...     if z < 2:
...         return x + y + z
...     else:
...         return z / (x + y)
...
```

注意：

1. 函数可以有位置参数和一些关键字参数；
2. 关键字参数必须位于位置参数之后

1.6 Python对象的方法

一般地，Python对象都有一些附属的函数，称为该对象的方法。方法可以访问该对象的内部数据。方法的调用方法如下：

```
object.some_method(x, y)
```

方法举例

例9.

```
>>> s = 'LOVE'
>>> result = s.lower()
>>> lis = [1, 2, 5, -3]
>>> lis.append(7)
>>> lis.reverse()
>>> lis.pop(2)
>>> lis.remove(-3)
```