第三章 第一节 Python基础篇--函数结构

Python 函数

函数是一种仅在调用时运行的代码块。
您可以将数据（称为参数）传递到函数中。
函数可以把数据作为结果返回。

创建函数

在Python中，使用 def 关键字定义函数:

实例

```
def my_function():
     print("Hello from a function")
```

调用函数

如需调用函数，请使用函数名称后跟括号:

实例

my_function()

```
Hello from a function
```

第三章 第一节 Python基础篇--函数结构

Python 函数

参数

信息可以作为参数传递给函数。

参数在函数名后的括号内指定。您可以根据需要添加任意数量的参数，只需用逗号分隔即可。

下面的例子有一个带参数（fname）的函数。当调用此函数时，我们传递一个名字，在函数内部使用它来打印全名：

实例

```
def my_function(fname):
     print(fname + " Gates")
my_function("Bill")
my_function("Steve")
my_function("Elon")
```

```
Rory John Gates
Jennifer Katharine Gates
Phoebe Adele Gates
```

第三章 第一节 Python基础篇--函数结构

Python 函数

默认参数值

下面的例子展示如何使用默认参数值。

如果我们调用了不带参数的函数，则使用默认值：

实例

```
def my_function(country = "China"):
     print("I am from " + country)
my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from China
I am from Brazil
```

第三章 第一节 Python基础篇--函数结构

Python 函数

以 List 传参

您发送到函数的参数可以是任何数据类型（字符串、数字、列表、字典等），并且在函数内其将被视为相同数据类型。

例如，如果您将 List 作为参数发送，它到达函数时仍将是 List（列表）：

实例

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

```
apple
banana
cherry
```

Python 函数

返回值

如需使函数返回值，请使用 return 语句：

实例

```
def my_function(x):
    return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

Python 函数

关键字参数

您还可以使用 key = value 语法发送参数。

参数的顺序无关紧要。

实例

```
def my_function(child3, child2, child1):
    print("The youngest child is " + child3)
    my_function(child1 = “Phoebe”, child2 = “Jennifer”, child3 = “Rory”)
```

```
The youngest child is Rory
```

在 Python 文档中，“关键字参数”一词通常简称为 kwargs。

第三章 第一节 Python基础篇--函数结构

Python 函数

任意参数

如果您不知道将传递给您的函数多少个参数，请在函数定义的参数名称前添加 *。

这样，函数将接收一个参数元组，并可以相应地访问各项：

实例

如果参数数目未知，请在参数名称前添加 *：

```
def my_function(*kids): print("The youngest child is " + kids[2])
my_function("Phoebe", "Jennifer", "Rory")
```

```
The youngest child is Rory
```

第三章 第一节 Python基础篇--函数结构

Python 函数

pass 语句

函数定义不能为空，但是如果您出于某种原因写了无内容的函数定义，请使用 pass 语句来避免错误。

实例

```
def myfunction:
    pass
```

第三章 第一节 Python基础篇--函数结构

Python 函数

递归

Python 也接受函数递归，这意味着定义的函数能够调用自身。

递归是一种常见的数学和编程概念。它意味着函数调用自身。这样做的好处是可以循环访问数据以达成结果。

开发人员应该非常小心递归，因为它可以很容易地编写一个永不终止的，或者使用过量内存或处理器能力的函数。但是，在被正确编写后，递归可能是一种非常有效且数学上优雅的编程方法。

在这个例子中，tri_recursion() 是我们定义为调用自身 ("recurse") 的函数。我们使用 k 变量作为数据，每次递归时递减（-1）。当条件不大于 0 时（比如当它为 0 时），递归结束。

对于新的开发人员来说，可能需要一些时间来搞清楚其工作原理，最好的方法是测试并修改它。

实例

递归的例子:

```
def tri_recursion(k):
     if(k>0):
       result = k+tri_recursion(k-1)
       print(result)
     else:
       result = 0
       return result
print("\n\nRecursion Example Results")
tri_recursion(6)
```

第三章 第一节 Python基础篇--函数结构

Python 函数

```
Recursion Example Results
1
3
6
10
15
21
```

第三章 第一节 Python基础篇--函数结构

Python 函数

Python Lambda

lambda 函数是一种小的匿名函数。

lambda 函数可接受任意数量的参数，但只能有一个表达式。

语法

```
lambda arguments : expression
```

执行表达式并返回结果：

实例

一个 lambda 函数，它把作为参数传入的数字加 10，然后打印结果：

```
x = lambda a : a + 10
print(x(7))
```

```
17
```

第三章 第一节 Python基础篇--函数结构

Python 函数

Python Lambda

lambda 函数可接受任意数量的参数:

实例

一个 lambda 函数，它把参数 a 与参数 b 相乘并打印结果:

```
x = lambda a, b : a * b
print(x(2, 5))
```

```
10
```

实例

一个 lambda 函数，它把参数 a、b 和 c 相加并打印结果:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

Python 函数

为何使用 Lambda 函数?

当您把 lambda 用作另一个函数内的匿名函数时，会更好地展现 lambda 的强大能力。

假设您有一个带一个参数的函数定义，并且该参数将乘以未知数字:

```
def myfunc(n):
  return lambda a : a * n
```

使用该函数定义来创建一个总是使所发送数字加倍的函数:

实例

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```

Python 函数

或者，使用相同的函数定义来创建一个总是使您发送的数字增加三倍的函数：

实例

```
def myfunc(n):
        return lambda a : a * n
mytripler = myfunc(3)
print(mytripler(11))
```

```
33
```

或者，在同一程序中使用相同的函数定义来生成两个函数：

实例

```
def myfunc(n): return lambda a : a * n
        mydoubler = myfunc(2)
        mytripler = myfunc(3)
        print(mydoubler(11))
        print(mytripler(11))
```

```
22
33
```

如果在短时间内需要匿名函数，请使用 lambda 函数。