

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

8.1. 高阶函数

8.1.1. map/reduce

8.1.2. filter

8.1.3. sorted

8.2. 返回函数

8.3. 匿名函数

8.4. 装饰器

8.5. 偏函数

9. 模块

9.1. 使用模块

9.2. 安装第三方模块

10. 面向对象编程

10.1. 类和实例

10.2. 访问限制

10.3. 继承和多态

10.4. 获取对象信息

10.5. 实例属性和类属性

11. 面向对象高级编程

11.1. 使用\_\_slots\_\_

11.2. 使用@property

11.3. 多重继承

11.4. 定制类

11.5. 使用枚举类

11.6. 使用元类

12. 错误、调试和测试

12.1. 错误处理

12.2. 调试

12.3. 单元测试

12.4. 文档测试

13. IO编程

14. 进程和线程

15. 正则表达式

16. 常用内建模块

17. 常用第三方模块

# 高阶函数



廖雪峰  
资深软件开发工程师，业余马拉松选手。

高阶函数英文叫Higher-order function。什么是高阶函数？我们以实际代码为例子，一步一步深入概念。

## 变量可以指向函数

以Python内置的求绝对值的函数 `abs()` 为例，调用该函数用以下代码：

```
>>> abs(-10)
10
```

但是，如果只写 `abs` 呢？

```
>>> abs
<built-in function abs>
```

可见，`abs(-10)` 是函数调用，而 `abs` 是函数本身。

要获得函数调用结果，我们可以把结果赋值给变量：

```
>>> x = abs(-10)
>>> x
10
```

但是，如果把函数本身赋值给变量呢？

```
>>> f = abs
>>> f
<built-in function abs>
```

结论：函数本身也可以赋值给变量，即：变量可以指向函数。

如果一个变量指向了一个函数，那么，可否通过该变量来调用这个函数？用代码验证一下：

```
>>> f = abs
>>> f(-10)
10
```

成功！说明变量 `f` 现在已经指向了 `abs` 函数本身。直接调用 `abs()` 函数和调用变量 `f()` 完全相同。

## 函数名也是变量

那么函数名是什么呢？函数名其实就是指向函数的变量！对于 `abs()` 这个函数，完全可以把函数名 `abs` 看成变量，它指向一个可以计算绝对值的函数！

如果把 `abs` 指向其他对象，会有什么情况发生？

```
>>> abs = 10
>>> abs(-10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```

把 `abs` 指向 `10` 后，就无法通过 `abs(-10)` 调用该函数了！因为 `abs` 这个变量已经不指向求绝对值函数而是指向一个整数 `10` ！

当然实际代码绝对不能这么写，这里是为了说明函数名也是变量。要恢复 `abs` 函数，请重启Python交互环境。

注：由于 `abs` 函数实际上是定义在 `import builtins` 模块中的，所以要让修改 `abs` 变量的指向在其它模块也生效，要用 `import builtins; builtins.abs = 10` 。

## 传入函数

## 5.7. 使用dict和set

## 6. 函数

### 6.1. 调用函数

### 6.2. 定义函数

### 6.3. 函数的参数

### 6.4. 递归函数

## 7. 高级特性

### 7.1. 切片

### 7.2. 迭代

### 7.3. 列表生成式

### 7.4. 生成器

### 7.5. 迭代器

## 8. 函数式编程

### 8.1. 高阶函数

#### 8.1.1. map/reduce

#### 8.1.2. filter

#### 8.1.3. sorted

### 8.2. 返回函数

### 8.3. 匿名函数

### 8.4. 装饰器

### 8.5. 偏函数

## 9. 模块

### 9.1. 使用模块

### 9.2. 安装第三方模块

## 10. 面向对象编程

### 10.1. 类和实例

### 10.2. 访问限制

### 10.3. 继承和多态

### 10.4. 获取对象信息

### 10.5. 实例属性和类属性

## 11. 面向对象高级编程

### 11.1. 使用\_\_slots\_\_

### 11.2. 使用@property

### 11.3. 多重继承

### 11.4. 定制类

### 11.5. 使用枚举类

### 11.6. 使用元类

## 12. 错误、调试和测试

### 12.1. 错误处理

### 12.2. 调试

### 12.3. 单元测试

### 12.4. 文档测试

## 13. IO编程

## 14. 进程和线程

## 15. 正则表达式

## 16. 常用内建模块

## 17. 常用第三方模块

既然变量可以指向函数，函数的参数能接收变量，那么一个函数就可以接收另一个函数作为参数，这种函数就称之为高阶函数。

一个最简单的高阶函数：

```
def add(x, y, f):  
    return f(x) + f(y)
```

当我们调用 `add(-5, 6, abs)` 时，参数 `x`，`y` 和 `f` 分别接收 `-5`，`6` 和 `abs`，根据函数定义，我们可以推导计算过程为：

```
x = -5  
y = 6  
f = abs  
f(x) + f(y) ==> abs(-5) + abs(6) ==> 11  
return 11
```

用代码验证一下：

```
def add(x, y, f):  
    return f(x) + f(y)  
  
print(add(-5, 6, abs))
```

编写高阶函数，就是让函数的参数能够接收别的函数。

## 参考源码

[do\\_f\\_add.py](#)

## 小结

把函数作为参数传入，这样的函数称为高阶函数，函数式编程就是指这种高度抽象的编程范式。

« 函数式编程

map/reduce »



## Comments

Loading comments...

©liaoxuefeng.com - 微博 - GitHub - License