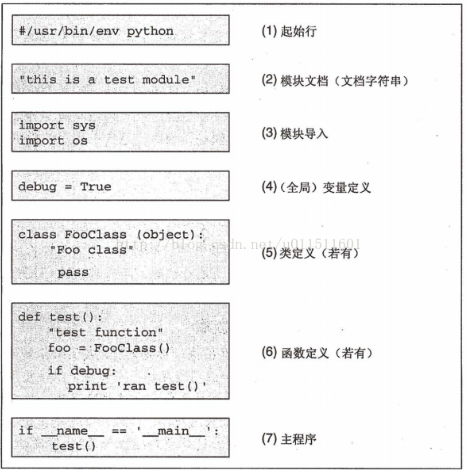
# **\_\_name\_\_变量**:

# 1）当文件是被调用时，\_\_name\_\_的值为模块名；

# 2）当文件被执行时，\_\_name\_\_的值为 ‘\_\_main\_\_’。



# **装饰器@decorator**

Python的 decorator 本质上就是一个高阶函数，它接收一个函数作为参数，然后，返回一个新函数。 使用 decorator 用Python提供的 @ 语法，这样可以避免手动编写 f = decorate(f) 这样的代码。

1)

def test(f):

print(“before...”)

f()

print(“after...””)

@ test

def func():

print(“func was called.”)

输出:

before...

func was called.

after...

2)

def test(func):

func()

print(“call test”)

def test1(f):

f()

print(“call test1”)

def main():

@test

def fun():

print(“call fun”)

@test1

def fun1():

print(“call fun1”)

main()

输出：

call fun

call fun1

call test1

call test

# **Python map() 函数**

map()会根据提供的函数对指定序列做映射。

第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

defsquare(x): # 计算平方数

returnx\*\* 2

map(square, [1,2,3,4,5])# 计算列表各个元素的平方

[1, 4, 9, 16, 25]

map(lambdax: x\*\* 2, [1, 2, 3, 4, 5])# 使用 lambda 匿名函数

[1, 4, 9, 16, 25]

**from . import XXX :**

在当前程序所在文件夹里\_\_init\_\_.py程序中导入XXX,.代表当前目录，..代表上一层目录，...代表上上层目录。

# [**Python 头部 #!/usr/bin/python 和 #!/usr/bin/env 的区别**](http://www.cnblogs.com/scofi/p/4867851.html)

# 大部分python文件的头部都会写上 #!/usr/bin/python 或者 #!/usr/bin/env ，这个语句主要和运行模式有关，如果我们用普通运行模式例如(linux) ： python \*.py 那么这个语句在此运行模式下无效。如果想让python程序像普通程序一样运行，例如：./\*.py (文件要有可执行权限chmod a+x \*.py)，这个语句就起作用了，他用来为脚本语言指定解释器，通常认为用 #!/usr/bin/env python 要比 #!/usr/bin/python 更好，因为 python 解释器有时并不安装在默认路径，例如在虚拟环境中。

# 解释：

#!/usr/bin/env python

这种写法在你机器上安装了多个版本的python的时候有意义，这样声明的时候，会去取你机器的 PATH 中指定的第一个 python 来执行你的脚本。如果这时候你又配置了虚拟环境的话，那么这样写可以保证脚本会使用你虚拟环境中的 python 来执行。

#!/usr/bin/python

表示写死了就是要 /usr/bin/python 这个目录下 python 来执行你的脚本。这样写程序的可移植性就差了，如果此路径下python命令不存在就会报错。所以一般情况还是用第一种写法。

**属性函数(property):**

1)将类方法转换为属性或者只读属性

2)重新实现一个属性的setter和getter方法

模块标准模版：

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

' a test module '

\_\_author\_\_ = 'Michael Liao'

import sys

第1行和第2行是标准注释，第1行注释可以让这个hello.py文件直接在Unix/Linux/Mac上运行，第2行注释表示.py文件本身使用标准UTF-8编码；第**4**行是一个字符串，表示模块的文档注释，任何模块代码的第一个字符串都被视为模块的文档注释；第**6**行使用**\_\_author\_\_**变量把作者写进去，这样当你公开源代码后别人就可以瞻仰你的大名；

### **作用域：**

在一个模块中，我们可能会定义很多函数和变量，但有的函数和变量我们希望给别人使用，有的函数和变量我们希望仅仅在模块内部使用。在**Python**中，是通过**\_**前缀来实现的。正常的函数和变量名是公开的（**public**），可以被直接引用，比如：**abc**，**x123**，**PI**等；类似**\_\_xxx\_\_**这样的变量是特殊变量，可以被直接引用，但是有特殊用途，比如上面的**\_\_author\_\_**，**\_\_name\_\_**就是特殊变量，**hello**模块定义的文档注释也可以用特殊变量**\_\_doc\_\_**访问，我们自己的变量一般不要用这种变量名；类似**\_xxx**和**\_\_xxx**这样的函数或变量就是非公开的（**private**），不应该被直接引用，比如**\_abc**，**\_\_abc**等；之所以我们说，**private**函数和变量“不应该”被直接引用，而不是“不能”被直接引用，是因为**Python**并没有一种方法可以完全限制访问**private**函数或变量，但是，从编程习惯上不应该引用**private**函数或变量。

在**Python**中，变量名类似**\_\_xxx\_\_**的，也就是以双下划线开头，并且以双下划线结尾的，是特殊变量，特殊变量是可以直接访问的，不是**private**变量，所以，不能用**\_\_name\_\_**、**\_\_score\_\_**这样的变量名。以一个下划线开头的实例变量名，比如**\_name**，这样的实例变量外部是可以访问的，但是，按照约定俗成的规定，当你看到这样的变量时，意思就是，“虽然我可以被访问，但是，请把我视为私有变量，不要随意访问”。

### 模块搜索路径：

默认情况下，**Python**解释器会搜索当前目录、所有已安装的内置模块和第三方模块，搜索路径存放在**sys**模块的**path**变量中：如果我们要添加自己的搜索目录，有两种方法：一是直接修改**sys.path**，添加要搜索的目录；第二种方法是设置环境变量**PYTHONPATH**，该环境变量的内容会被自动添加到模块搜索路径中。

类：

和静态语言不同，**Python**允许对实例变量绑定任何数据，也就是说，对于两个实例变量，虽然它们都是同一个类的不同实例，但拥有的变量名称都可能不同：

### 静态语言 vs 动态语言动态语言的“鸭子类型”，它并不要求严格的继承体系，一个对象只要“看起来像鸭子，走起路来像鸭子”，那它就可以被看做是鸭子。

多重继承：

在设计类的继承关系时，通常，主线都是单一继承下来的，但是，如果需要“混入”额外的功能，通过多重继承就可以实现，这种设计通常称之为**MixIn**。**MixIn**的目的就是给一个类增加多个功能，这样，在设计类的时候，我们优先考虑通过多重继承来组合多个**MixIn**的功能，而不是设计多层次的复杂的继承关系。

对象调用：

通过callable()函数，我们就可以判断一个对象是否是“可调用”对象。