使用模块

廖雪峰
资深软件开发工程师，业余马拉松选手。

Python本身就内置了很多非常有用的模块，只要安装完毕，这些模块就可以立刻使用。

我们以内建的 `sys` 模块为例，编写一个 `hello` 的模块：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

' a test module '

__author__ = 'Michael Liao'

import sys

def test():
    args = sys.argv
    if len(args)==1:
        print('Hello, world!')
    elif len(args)==2:
        print('Hello, %s!' % args[1])
    else:
        print('Too many arguments!')

if __name__=='__main__':
    test()
```

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；

以上就是Python模块的标准文件模板，当然也可以全部删掉不写，但是，按标准办事肯定没错。

后面开始就是真正的代码部分。

你可能注意到了，使用 `sys` 模块的第一步，就是导入该模块：

```
import sys
```

导入 `sys` 模块后，我们就有了变量 `sys` 指向该模块，利用 `sys` 这个变量，就可以访问 `sys` 模块的所有功能。

`sys` 模块有一个 `argv` 变量，用list存储了命令行的所有参数。 `argv` 至少有一个元素，因为第一个参数永远是该.py文件的名称，例如：

运行 `python3 hello.py` 获得的 `sys.argv` 就是 `['hello.py']` ；

运行 `python3 hello.py Michael` 获得的 `sys.argv` 就是 `['hello.py', 'Michael']` 。

最后，注意到这两行代码：

```
if __name__=='__main__':
    test()
```

当我们在命令行运行 `hello` 模块文件时，Python解释器把一个特殊变量 `__name__` 置为 `__main__` ，而如果在其他地方导入该 `hello` 模块时， `if` 判断将失败，因此，这种 `if` 测试可以让一个模块通过命令行运行时执行一些额外的代码，最常见的就是运行测试。

我们可以用命令行运行 `hello.py` 看看效果：

```
$ python3 hello.py
Hello, world!
```

```
$ python hello.py Michael
Hello, Michael!
```

如果启动Python交互环境，再导入 `hello` 模块：

```
$ python3
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import hello
>>>
```

导入时，没有打印 `Hello, word!` ，因为没有执行 `test()` 函数。

调用 `hello.test()` 时，才能打印出 `Hello, word!` ：

```
>>> hello.test()
Hello, world!
```

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。

private函数或变量不应该被别人引用，那它们有什么用呢？请看例子：

```
def _private_1(name):
    return 'Hello, %s' % name

def _private_2(name):
    return 'Hi, %s' % name

def greeting(name):
    if len(name) > 3:
        return _private_1(name)
    else:
        return _private_2(name)
```

我们在模块里公开 `greeting()` 函数，而把内部逻辑用private函数隐藏起来了，这样，调用 `greeting()` 函数不用关心内部的private函数细节，这也是一种非常有用的代码封装和抽象的方法，即：

外部不需要引用的函数全部定义成private，只有外部需要引用的函数才定义为public。

« 模块

安装第三方模块 »

Comments

杰 @ 2025/12/27 04:59:50

1、每个Python模块（.py文件）都有一个内置的变量 __name__ 。
2、当模块被直接运行时， __name__ 的值为 '__main__' 。
3、当模块被导入到其他模块中时， __name__ 的值就是模块的名字（即文件名，去掉.py后缀）。

因此，这个判断语句的作用是：
如果当前模块是作为主程序运行，那么 __name__ 的值就是 ' __main__' ，那么 if 下面的代码块会被执行
如果当前模块是被导入的，那么 __name__ 的值就是模块名，不等于 '__main__' ，那么 if 下面的代码块

这样设计的好处是：
我们可以将模块作为脚本直接运行，也可以被其他模块导入，而不会在导入时执行那些不应该执行的代码（通
它使得模块既可以独立运行，也可以作为库被其他模块使用。

Hypersomnia @ 2025/12/1 02:21:24

就是为了别的函数调用时不会先弹出一句Hello吧,只有在主函数使用时会先弹出来

多多多多 @ 2025/11/18 21:55:47

这是一个Python中常见的惯用法，用于判断当前模块是作为主程序运行还是被导入到其他模块中。

name 是Python的一个内置变量，用于表示当前模块的名字。

如果一个模块是被直接运行的，那么 **name** 的值就是 '**main**'。

如果一个模块是被导入的，那么 **name** 的值就是该模块的名字（即文件名，不包含.py后缀）。

因此，if **name** == '**main**': 这行代码就是判断当前模块是否被直接运行

iLeafy @ 2025/11/5 03:56:27

__name__是被调用模块的名字，'**main**'是你自己代码.py的名字，if **name**=='**main**'一般是在被打包好的代码也就是被调用的模块里面，一般用来测试代码功能是否正常啥的，被打包好的代码被调用在其他py文件是不会执行这下面的代码的，因为name是被打包好的代码，main是当前代码。

t夢jun @ 2024/11/19 04:21:09

在Python中，if **name** == '**main**': 这行代码有一个特定的用途。它用于判断当前运行的脚本是否是主程序，而不是被其他脚本作为模块导入时执行的代码。

当你直接运行一个Python文件时，比如通过命令行输入 python script.py，Python解释器会将特殊变量 **name** 设置为 '**main**'。但是，如果另一个脚本导入了这个文件（比如使用 import script），那么 **name** 变量将被设置为该脚本的模块名（在这个例子中是 'script'，但不包括 .py 扩展名）。

因此，if **name** == '**main**': 这行代码下面的代码块只有在该脚本被直接运行时才会执行。这允许一个Python文件既可以作为脚本直接运行，执行一些操作，也可以作为模块被其他脚本导入，提供函数、类和变量等，而不会在不希望的时候执行代码。

爱 贝里屋 @ 2025/10/10 04:29:32

没看懂

怪盗基德 @ 2025/11/3 09:27:16

感觉没啥实际用处

HellPlay @ 2021/1/4 20:48:29

首先新建一个 `hello.py` ，只写一句代码：

```
print(__name__)
```

1、在**命令行**直接调用运行：

`$ python hello.py`

`__main__`

2、在**Python交互环境**中加载模块：

`>>> import hello`

`hello`

The Chivalry @ 2022/7/19 05:33:52

牛逼！

九界唐家第一少 @ 2022/9/18 09:21:02

牛逼！！！！！！！！！！！！！

CDra @ 2022/11/4 23:24:08

什么意思呀，没怎么理解

长日将尽 @ 2023/4/20 05:38:13

回复楼上，就是把在python交互环境里import该文件的__name__和直接打开的__name__区别告诉你，也就是import后的__name__是文件名

爱 贝里屋 @ 2025/10/10 04:34:31

Python交互环境是什么?

清柠檬 @ 2025/7/28 01:05:26

核心解释： `if __name__ == '__main__':` 这行代码的作用是**判断当前模块是否是直接运行的。**

类比理解： 想象你有一个电灯（Python文件）：

- 当你直接按开关（直接运行这个文件）时， `__name__` 就是 `'__main__'` ，灯会亮（执行代码块里的内容）。
- 当这个电灯被接到其他电路里（被其他文件导入时）， `__name__` 就变成模块名，这时灯不会亮（不执行代码块里的内容）。

实际作用：

1. **直接运行时**：会执行这个代码块里的内容（就像程序的入口）
2. **被导入时**：不会执行这里的代码，只导入其他部分

示例：

Read More ▼

方羽 @ 2025/7/31 04:59:22

yeah

爱 贝里屋 @ 2025/10/10 04:25:03

直接运行是指在这个模块内直接运行吗?

tky @ 2025/8/10 05:42:03

•

name 在命令行运行hello模块文件时，Python解释器把一个特殊变量__name__置为__main__，而如果在其他地方导入该hello模块时，if判断将失败

•

作用域 正常的函数和变量名是公开的（public）,如：abc，x123；__xyz__这样的是特殊变量，一般不使用；形如_xyz或者__xyz的函数是模块中的不公开函数，正常不需要引用， 是一种非常有用的代码封装方法

爱 贝里屋 @ 2025/10/10 04:22:03

什么玩意啊

只是一个高性能萝卜子 @ 2025/7/22 09:24:37

`__name__` is a built-in variable that Python automatically sets for every module (Python file). It indicates how the current module is being executed.

How `__name__` is set:

1. When a file is run directly:

```
python myfile.py
```

Python sets `__name__ = '__main__'` for that file.

2. When a file is imported:

```
import myfile
```

Read More ▼

路 @ 2025/6/3 21:38:08

9.1. 使用模块 done

CQ44 @ 2025/5/25 09:22:59

```
我的理解很简单,不知对不对:
当在powershell运行.py文件时,__name__ = __main__
当在交互模式import时,__name__ = Hello(.py)
```

日向夏橘 @ 2025/5/9 07:20:47

打卡

Sky / @ 2025/4/16 02:27:51

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

'a test module'

__author__ = 'Emon'

import sys

def test():
    args = sys.argv
    if len(args) == 1:
        print('Hello, world!')
    elif len(args) == 2:
        print('Hello, %s!' % args[1])
    else:
```

Read More ▼

QK時間 @ 2025/4/10 05:58:35

封装private方法还是很重要的，只需要暴露应该public的方法就行了

介 @ 2024/10/30 21:10:18

如果导入好几个模块，每个模块中有相同的函数定义，调用的时候会出问题嘛

just like h @ 2024/11/13 03:05:40

建立目录存放就不会冲突了吧?

zy、 @ 2025/2/25 04:48:20

不会，有模块名在呢

廖雪峰 @ 2025/2/25 18:50:47

起个别名：

```
from x import fn as fn1
from y import fn as fn2
```

现在可以分别用fn1和fn2了

青铜神裔 @ 2019/3/16 08:44:50

转僵尸认识你 转载自https://www.cnblogs.com/1204guo/p/7966461.html

这几天开始学习Python，遇到一些问题，也解决了一些问题。

其中if **name** == '**main**'：这句估计很多和我一样的初学者都是不求甚解。 这里作一下解释：

1：name是一个变量。前后加了爽下划线是因为是因为这是系统定义的名字。普通变量不要使用此方式命名变量。 2：Python有很多模块，而这些模块是可以独立运行的！这点不像C++和C的头文件。 3：import的时候是要执行所import的模块的。 4：name就是标识模块的名字的一个系统变量。这里分两种情况：假如当前模块是主模块（也就是调用其他模块的模块），那么此模块名字就是main，通过if判断这样就可以执行"mian:"后面的主函数内容；假如此模块是被import的，则此模块名字为文件名字（不加后面的.py），通过if判断这样就会跳过"mian:"后面的内容。

通过上面方式，python就可以分清楚哪些是主函数，进入主函数执行；并且可以调用其他模块的各个函数等等。

刘烨超351929724 @ 2019/3/18 08:20:41

为什么我运行hello.test()结果是Too many arguments!

fbitct @ 2019/8/12 06:02:24

谢谢大佬，讲解的通俗易懂。语言组织能力很强。再次感谢大佬。

Nb-Boom @ 2019/8/15 20:45:31

赞!

红线凌空去 @ 2020/5/24 04:38:02

感谢讲解，通俗易懂!

1街头巷尾 @ 2021/2/22 01:37:17

爆赞!

陳大發的小红豆 @ 2021/3/30 04:38:41

感谢大佬总结，原来import x是需要执行x模块中非main函数的部分。

用户3229353851 @ 2021/10/6 00:47:33

陳大發的小红豆

楼上的一句话总结，又更进一步的总结了import x 和 main 函数的关系，真棒。

里昂tcxy @ 2021/12/10 03:20:05

通俗易懂，赞!

里昂tcxy @ 2021/12/10 03:53:55

因为__name__变量前面有__，表示私有函数，所以不能被其他人使用。import x 是执行公开函数，也就是没__前缀的函数，而不仅仅是不执行main函数

夜半窗留 @ 2021/12/21 03:44:19

点赞。我理解就是设置一个参数来决定要不要执行模块主体部分的问题

浮躁飘飘 @ 2022/1/13 00:59:13

说得真好

用户7507505625 @ 2022/1/15 04:00:01

感谢分享👍

白白茶与风 @ 2022/1/21 07:12:34

受教了，谢谢!

Simms_zh @ 2022/3/3 21:00:08

感谢大佬，学习了

longtometosee @ 2022/3/8 07:57:22

引用模块，执行的是非__main__部分，因为此时__name__ 指向原来 moudle name (不含.py),而__main__指向当前.py文件，此时if __name__==' __main__' : 判断为False ,不执行后续缩进代码块。

apprentice @ 2022/5/3 04:03:39

赞，救了我这个萌新

Mx @ 2022/5/9 05:18:26

学习了

Rrz @ 2024/10/25 03:19:32

赞赞

千夕夜 @ 2022/5/26 21:10:05

转僵尸认识你 转载自https://www.cnblogs.com/1204guo/p/7966461.html

下载PDF

这几天开始学习Python，遇到一些问题，也解决了一些问题。

其中if **name** == '**main**'：这句估计很多和我一样的初学者都是不求甚解。 这里作一下解释：

1：name是一个变量。前后加了爽下划线是因为是因为这是系统定义的名字。普通变量不要使用此方式命名变量。 2：Python有很多模块，而这些模块是可以独立运行的！这点不像C++和C的头文件。 3：import的时候是要执行所import的模块的。 4：name就是标识模块的名字的一个系统变量。这里分两种情况：假如当前模块是主模块（也就是调用其他模块的模块），那么此模块名字就是main，通过if判断这样就可以执行"mian:"后面的主函数内容；假如此模块是被import的，则此模块名字为文件名字（不加后面的.py），通过if判断这样就会跳过"mian:"后面的内容。

通过上面方式，python就可以分清楚哪些是主函数，进入主函数执行；并且可以调用其他模块的各个函数等等。

Rrz @ 2024/10/25 02:54:47

很棒！看完则条评论才真正懂了__name__的意义

纳里兜 @ 2023/12/10 01:50:38

__abc 或__abc__这些私有函数是非公开的，原则上是不被引用的但是不代表不能引用，强制掉用私有函数也是可以用的

憨巴鸭子 @ 2023/10/22 22:18:42

导入模块

import sys

使用模块中的函数

print(greeting())

常用的运行测试（用于写模块的时候）

内置函数_name_

if _name_ == '_main_'

小学生 @ 2023/6/4 11:41:45

我在Pycharm中使用jupyter notebook输入代码

```
import hello
hello.test()
```

小学生 @ 2023/6/4 11:43:39

我在Pycharm中使用jupyter notebook输入代码

```
import hello
hello.test()
```

得到结果是：

Too many arguments!

而在系统的python交互界面得到的是 Hello, World!

请问有人知道是什么原因吗?

sky @ 2023/6/7 20:56:38

你试一下打印args这个列表就知道了

```
sys.argv
```

这个函数是用来保存命令行参数的，把参数加入列表并返回，比如例子中在终端使用了：

```
python3 hello.py
```

这个命令来调用模块，此时args=sys.argv，就把'hello.py'存入列表args，长度为1，输出hello world；同理：

```
python3 hello.py Michael
```

这个命令行参数包括hello.py和Michael，所以列表长度为2，进行相应输出，那么现在我们来讨论你遇到的问题：

你直接在交互界面调用模块，这时命令是这样的：`python -u ".py文件的路径"`

这时args中就只存放一个路径，长度为1，而如果你在jupyter中调用模块并打印，就会发现列表长度特别长，因为jupyter在调用模块的时候会加入各种环境参数，

比如本机的ip地址之类的，这样就只能输出too many arguments了

Collapse ▲
