## 模块及包

模块及包的导入

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。比如之前讲过了keywords模块，还有接下来要讲的os,sys,re等模块都是python的内置模块。

以tkinter模块为例，模块导入的方式有：

import tkinter #导入tkinter 模块

form tkinter import Label #导入模块中的方法，Label组件

from tkinter import \* #导入模块中的所有类和常量

import tkinter as tk #将tkinter模块导入进来并重命名为tk

from tkinter import Label as Lb #将Label类导入并重命名为Lb

模块是包括python定义和声明的文件，文件名就是模块名加上.py后缀。我们自己写的.py文件也是模块，如果是自定的模块要在程序中导入，模块保存的位置就很重要，可以通过sys模块来告诉解释器在哪里寻找模块。

自定义在导入模块的时候，会在当前的文件目录中自动生成\_\_pycache\_\_文件夹，文件夹里面会看到有新文件出现以.pyc扩展名的文件是（与平台无关的），经过处理（编译）的,已经转换成python能够更加有效处理的文件，如果稍后导入同一模块，pyhton会导入.pyc文件而不是.py文件，除非.py文件已改变，在这种情况下会生成新的.pyc文件。

例：

先写个ffr.py的程序，将程序保存在'E:\python课程\2017\例子'中。

程序如下：

def ss():

return 'hahahaha'

class Fruits:

def \_\_init\_\_(self,name,color,weight=90):

self.name = name

self.color = color

self.weight = weight

def show(self):

return('我是一个%s。' % self.name)

if \_\_name\_\_=='\_\_main\_\_':

fr1=Fruits('apple','red')

print(fr1.show())

print(ss())

运行程序后的结果为：

我是一个apple。

hahahaha

>>>

接下来把ffr.py作为模块，重新打开一个shell：

>>> import sys

>>> sys.path.append(r'E:\python课程\2017\例子')

>>> import ffr

>>> ffr

<module 'ffr' from 'E:\\python课程\\2017\\例子\\ffr.py'>

>>> ffr.Fruits

<class 'ffr.Fruits'>

>>> ffr.ss

<function ss at 0x039DE8E8>

>>>

在上面的例子中写了ffr.py程序，程序中定义了一个函数ss，一个类Fruits，然后把类的实例化及实例的方法调用和函数ss的调用语句都放在了if语句中，if语句的判断条件是\_\_name\_\_==’\_\_main\_\_\_’，所有的程序中自身的\_\_name\_\_属性都是’\_\_main\_\_’，而当程序作为模块导入到其他程序时，它的\_\_name\_\_属性就是模块名，所以这个条件里面的子语句只会在它自己运行时才会执行。

导入模块的时候要注意的就是文件保存的位置，这个例子中我们要导入ffr模块，所以在需要导入时先将文件路径添加到path列表中，这样才能找到我们的ffr模块。导入进来后，可以通过模块名来使用模块中创建的ss和Fruits对象，而我们在if语句中创建的fr1对象并不会导入进来。

包基本上就是另外一类模块。把用来处理一类事物的多个文件放在同一文件夹下组成的模块集。要让python 将其作为包对待，在python2中必须包含\_\_init\_\_.py的文件，但在python3中没有\_\_init\_\_.py文件也可以，使用包可以避免多个文件重名的情况，不同的包下的文件同名不影响。

常见的包结构是：

package

├── \_\_init\_\_.py

├── main.py

├── module1.py

└── module2.py

这里的package代表的就是包名，导入的时候可以是from package import module1或者是import package使用时就是package.module1的形式。

导入包和模块是一样的方法，包导入后会在目录中自动生成\_\_pycache\_\_文件夹存放生成的.pyc文件，要注意的是import后面的才是存在当前作用域中的对象名，from是告诉python从什么地方导入，使用sys.path添加文件的目录路径。

## Os模块

### 环境变量函数

1. os.name 如果是windows操作系统返回’nt’，如果是其他系统则返回 ‘posix’
2. os.environ 返回系统的环境变量，以dict形式显示

### 文件操作函数

Python os 模块包含普遍的操作系统功能。

常用方法

Os.getcwd() 返回当前工作目录

Os.chdir(path) 改变工作目录

Os.listdir(path=’.’) 列举指定目录中的文件名(‘.’表示当前目录，’..’代表上一级目录)

Os.mkdir(path) 创建单层目录，如该目录已存在抛出异常

Os.makedirs(path) 递归创建多层目录，如果该目录已经存在则抛出异常。

Os.remove(path) 删除文件

Os.rmdir(path) 删除单层目录，如改目录非空则抛出异常

Os.removedirs(path) 递归删除目录，从子目录到父目录逐层尝试删除，遇到目录非空则抛出异常。

Os.rename(old,new) 将文件old 重命名为new，文件和目录都使用这条命令

Os.path 模块中关于路径常用的函数使用方法：

Os.path.basename(path) 去掉目录路径，单独返回文件名

Os.path.dirname(path) 去掉文件名，单独返回目录路径

Os.path.join(path1,path2) 将path1,path2各部分组合成一个路径名

Os.path.split(path) 分割文件名和路径，返回（f\_path,f\_name）元组。如果完全使用目录，它也会将最后一个目录作为文件名分离，且不会判断文件或者目录是否存在。

Os.path.splitext(path)分离文件名与扩展名，返回（f\_name,f\_extension）元组

Os.path.getsize(file)返回指定文件的尺寸，单位是字节

Os.path.getatime(file) 返回指定文件最近的访问时间（浮点型秒数，可用time模块的gmtime()或localtime()函数换算）

Os.path.getctime(file) 返回指定文件创建时间（浮点型秒数，可用time模块的gmtime()或localtime()函数换算）

Os.path.getmtime(file) 返回指定文件最新的修改时间（浮点型秒数，可用time模块的gmtime()或localtime()函数换算）

返回True或False的函数

Os.path.exists(path) 判断指定路径（目录或文件）是否存在

Os.path.isabs(path) 判断指定路径是否为绝对路径

Os.path.isdir(path) 判断指定路径是否存在且是一个目录

Os.path.isfile(path) 判断指定路径是否存在且是一个文件

## shutil模块

复制文件

Shutil.copyfile(‘oldfile’,’newfile’) oldfile和newfile都只能是文件

Shutil.copy(‘oldfile’,’newfile’) oldfile只能是文件，newfile可以是文件也可以是目标目录

复制文件夹

Shutil.copytree(‘olddir’,’newdir’),olddir和newdir都只能是目录，且newdir必须不存在

移动文件（目录）

Shutil.move(‘oldname’,’newname’)

删除目录

Os.rmdir(‘dir’)只能删除空目录

Shutil.rmtree(‘dir’)空目录，有内容的目录都可以删

## Sys模块

Sys.argv

用来向python解释器传递参数，名曰“命令行参数”

Sys.exit()

退出当前程序

Sys.stdout

与Python中的函数功能对照，sys.stdin获得输入，等价于python2中的raw\_inpurt()，python3中的input(),sys.stdout负责输出

Sys.path

返回python目录下所有.pth路径文件下的内容集系统默认设置。

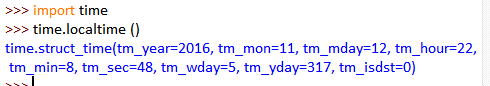
可以通过列表的操作对其进行修改，不过这种更改只对当前的程序起作用。

## Time模块

时间获取和转换

时间元组（time.struct\_time）

Gettime(),localtime()和striptime()以时间元组（struct\_time）的形式返回



返回的元组的内容

|  |  |  |
| --- | --- | --- |
| 索引值(index) | 属性(attribute) | 值(value) |
| 0 | tm\_year(年) | 例如：2016 |
| 1 | tm\_mon(月) | 1~12 |
| 2 | tm\_mday(日) | 1~31 |
| 3 | tm\_hour(时) | 0~23 |
| 4 | tm\_min(分) | 0~59 |
| 5 | tm\_sec(秒) | 0~61 |
| 6 | tm\_wday(星期几) | 0~6(0代表星期一) |
| 7 | tm\_yday(一年中的第几天) | 1~366 |
| 8 | tm\_isdst(是否为夏令时) | 0，1，-1(-1代表夏令时) |

time.asctime([t])

接收时间元组并返回一个可读的形式。



time.clock()

用以浮点数计算的秒数返回当前的cpu时间,用来衡量不同程序的耗时

time.time()

返回当前时间的时间戳，可以用来计算程序的耗时

时间戳(timestamp)表示的是从1970年1月1日00：00：00开始按秒计算的偏移量（time.gmtime(0)）

time.sleep(secs)

推迟调用线程的运行，secs的单位是秒