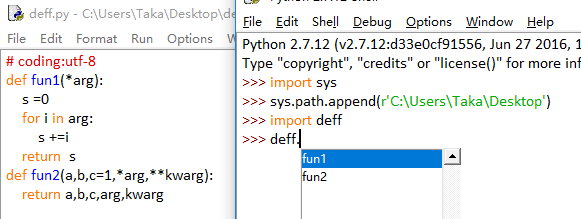
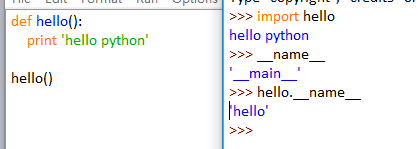
模块(module)

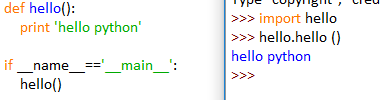
1. 模块是包括python定义和声明的文件，文件名就是模块名加上.py后缀。模块中的模块名（作为字符串）可以由全局变量\_\_name\_\_得到。
2. 程序保存的位置很重要，可以通过sys模块来告诉解释器在哪里寻找模块。



1. 在导入模块的时候，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。
2. 导入模块多次和导入一次的效果是一样的。
3. 定义的所有类和函数已经赋值后的变量都成了模块的特性。



1. 如例子中，在模块中加了函数的调用那么在导入模块的时候这调用语句也被执行了，而这个调用只是为了测试代码是否正常运行的，不需要导入到我们程序中。那么我们只会使用\_\_name\_\_变量。
2. 从例子中可以看到，当前函数的\_\_name\_的值是’\_\_main\_\_’，而导入的模块中的\_\_name\_\_的值是这个模块的名字。所以有了一下的方法：



1. 所以在我们的模块当前模块中将测试语句放在if \_\_name\_\_==’\_\_main\_\_’:语句后面，那么在导入模块时，这段语句将不会运行。
2. 模块的引用方式

import module #导入module模块

import module as new\_name #将module模块导入并重命名为new\_name

from module import \* #导入module模块中的所有组件类和常量

form module import objectname #来自module模块导入objectname组件

from module import objectname as newname #来自module模块导入objectnamel组件并重命

Os模块 (os连接的是电脑)

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) 删除单层空的目录，如改目录非空则抛出异常

os.removedirs(path) 递归删除空de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

>>> import os #导入os模块

>>> os.getcwd() #返回当前路径

'D:\\Program Files\\python3.6.0'

>>> os.chdir(r'C:\Users\Win8\Desktop') #改变路径

>>> os.getcwd() #返回当前路径

'C:\\Users\\Win8\\Desktop'

>>> os.mkdir('ff') #创建ff文件夹

>>> os.makedirs(r'ss\ff') #创建多层文件夹ss\ff

>>> os.rmdir('ff') #删除ff文件夹

>>> os.removedirs(r'ss\ff') #删除多层文件夹ss\ff

>>> os.remove('m.py') #删除m.py文件

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’) 相当于剪切

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

>>> import shutil

>>> shutil.copyfile('m.py',r'ff\m.py') #把m.py文件复制到ff

'ff\\m.py'

>>> shutil.copy('xx.py','ff') #把m.py文件复制到ff

'ff\\xx.py'

>>> shutil.move('类2.py','ff') #把类2.py文件移动到ff文件里

'ff\\类2.py'

包

1. 什么是包管理？把处理同一个事件的多个模块放到同一个目录下。
2. 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。
3. 使用dir函数，它会将对象的所有特性（以及模块的所有函数，类，变量等）列出来。

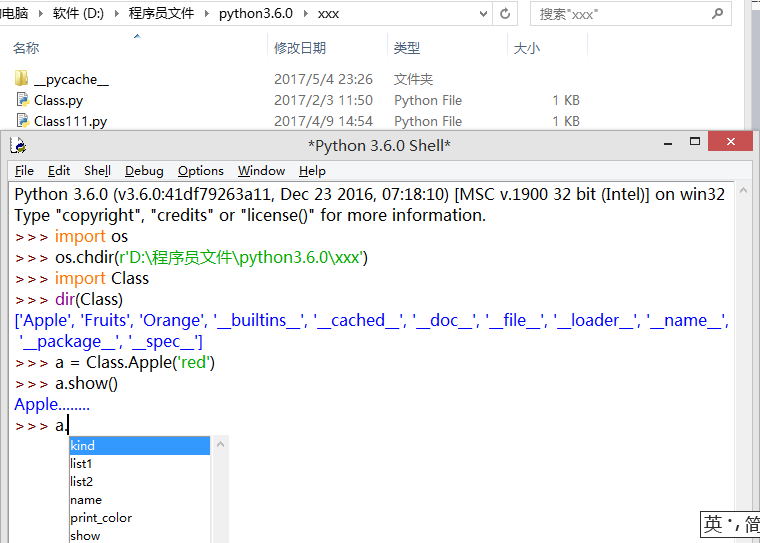
步骤：

先建立一个文件夹，把需要用到的模块放进这个文件夹

Import os 模块

更改到这个包的路径

从这个包导入模块



sys模块 (sys连接的python)

sys.argv 命令行参数的列表，列表的第一个元素就是正在运行的文件名

sys.exit() 退出当前程序

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。

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

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。

>>> import sys

>>> sys.path

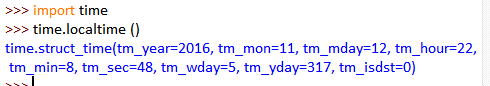
['', 'D:\\Program Files\\python3.6.0\\Lib\\idlelib', 'D:\\Program Files\\python3.6.0\\python36.zip', 'D:\\Program Files\\python3.6.0\\DLLs', 'D:\\Program Files\\python3.6.0\\lib', 'D:\\Program Files\\python3.6.0', 'D:\\Program Files\\python3.6.0\\lib\\site-packages']

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

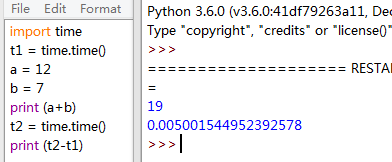
>>> time.clock()

4.665319322446344e-07

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