

## 文件操作与模块

### 复习

#### 飞机大战游戏-功能实现

搭建界面和键盘监测

添加背景音乐

控制飞机玩具

显示敌机

敌机发射子弹

#### 飞机大战游戏-优化

运行性能优化

代码结构优化

### 导入

文件操作是很常见的功能，我们 `python` 当然也支持，`python` 自带文件的基本操作方法，但是借助 `os` 等模块更加方便快捷。在学习过程中，我们时不时会用到模块，这里也将详细讲解模块、常用模块、以及怎样制作、发布和安装模块。

### 目录

- 1、文件打开关闭
- 2、文件读写
- 3、应用：文件备份脚本
- 4、应用：linux 密码暴力破解机
- 5、文件定位
- 6、模块介绍
- 7、`os` 模块操作文件

8、time、datetime 模块

9、模块的制作、发布、安装

## 目标

- 1、通过使用文件的打开关闭、读写等，完成文件备份脚本和 linux 密码暴力破解程序
- 2、通过文件定位的相关方法，完成精确控制文件读写
- 3、使用 os 模块操作文件，提高效率
- 4、使用 time、datetime 模块完成时间相关操作
- 5、通过模块的制作、发布、安装，进一步明白模块的意义

## 一、文件打开关闭

### 1、文件操作一般步骤

- 打开文件
- 读/写文件
- 保存文件
- 关闭文件

在 python 中操作文件也是遵循这几个步骤的。

### 2、打开文件

在 Python 中打开文件使用 open 函数，可以打开一个已经存在的文件，或者创建一个新文件

语法格式：

`open('文件名称','打开模式')`

示例：

```
open('test.txt','w')
```

## 文件打开模式

打开模式	模式说明
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
w	打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
w+	打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会追加模式。如果该文件不存在，创建新文件用于读写。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

## 3、关闭文件

语法格式：

`close()` 方法关闭文件

示例：

```
f = open('text.txt','w')
f.close()
```

注意： 打开一个文件之后，一定要关闭，否则后面无法继续操作这个文件。

## 4、with 上下文管理

`with` 语句，不管在处理文件过程中是否发生异常，都能保证 `with` 语句执行完毕后已经关闭打开的文件句柄。

示例：

```
def main():
    with open('setup.py','w') as f:
```

```
        content = f.read()
        print(content)
    main()
```

## 二、文件读写

### 1、写文件

写入文件 `write()` 方法，参数就是需要写入的内容。

```
# 写模式打开一个 test.txt 文件
f = open('test.txt', 'w')
f.write('我爱中国') # write 方法写入内容
f.close() # 关闭文件
```

`writelines()` 可传一个可迭代对象

```
# 写模式打开一个 test.txt 文件
f = open('test.txt', 'w')
f.writelines(['我', '爱', '我的', '国家']) # writelines 方法将可迭代对象，迭代写入文件
f.close() # 关闭文件
```

### 2、读文件

读取文件 `read()` ,将文件的内容全部读取出来。

```
# 写模式打开一个 test.txt 文件
f = open('test.txt', 'r')
f.read() # 一次性将文件内容全部取出
f.close() # 关闭文件
```

读取指定字符个数 `read(num)`传入一个数字做参数，表示读取指定字符个数。

```
# 写模式打开一个 test.txt 文件
f = open('test.txt', 'r')
content = f.read(2) # 读取两个字符
print(content)
content = f.read()
print(content) # 第二次读取将从第一次读取的位置继续读取
f.close() # 关闭文件
```

`readlines()` 按行读取，一次性读取所有内容，返回一个列表，每一行内容作为一个元素。

```
# 写模式打开一个 test.txt 文件
f = open('test.txt', 'w')
# 写入多行 hello world
f.write('hello world\nhello world\nhello world\nhello
world\nhello world\n')
f.close()

#打开文件 test.txt
f = open('test.txt', 'r')
content = f.readlines() # 一次性读取所有内容，返回一个列表，列表元素为每一行内容
print(content)
f.close() # 关闭文件
```

`readline()` 按行读取，但是一次只读取一行。

```
# 写模式打开一个 test.txt 文件
f = open('test.txt', 'w')
# 写入多行 hello world
f.write('hello world\nhello world\nhello world\nhello world\nhello
world\nhello world\n')
f.close()

#打开文件 test.txt
f = open('test.txt', 'r')
content = f.readline() # 按行读取，一次读取一行
print(content)
f.close() # 关闭文件
```

## 三、应用：文件备份脚本

### 1、需求 1

利用脚本完成自动备份，要求用户输入文件名称，完成自动备份

```
def copyfile():
    # 接收用户输入的文件名
    old_file = input('请输入要备份的文件名: ')
    file_list = old_file.split(".")
    # 构造新的文件名，加上备份后缀
    new_file = file_list[0] + '_备份.' + file_list[1]
    # 打开需要备份的文件
    old_f = open(old_file, 'r')
    # 以写的模式打开新文件, 不存在则创建
```

```
new_f = open(new_file, 'w')
# 键文件内容读取出来
content = old_f.read()
# 将读取的内容写入备份文件
new_f.write(content)
# 将打开的文件关闭
old_f.close()
new_f.close()
```

copyfile()

## 2、需求 2

如果处理超大文件，一次将全部内容读取出来显然是不合适的，在需求 1 的基础上改进下代码，让它备份大文件也不会导致内存被占满。

```
def copyfile():
    # 接收用户输入的文件名
    old_file = input('请输入要备份的文件名: ')
    # 如果没有输入文件名则打印提示
    if not old_file:
        print('[ERROR]: 请输入正确的文件路径')
        return
    file_list = old_file.split(".")
    # 构造新的文件名，加上备份后缀
    if len(file_list) < 2:
        new_file = file_list[0] + '_备份' # 文集名没有后缀的情况
    else:
        new_file = file_list[0] + '_备份.' + file_list[1]
    try:
        # 同时打开需要备份的文件，新文件
        with open(old_file, 'r') as old_f, open(new_file, 'a') as new_f:
            while True:
                # 一次读取 1024 字符
                content = old_f.read(1024)
                new_f.write(content)
                # 当读取的内容字符长度小于 1024 说明已经读取完毕
                if len(content) < 1024:
                    break
    except Exception as e:
        print(e)
```

copyfile()

## 四、应用：linux 密码暴力破解机

### 1、shadow 文件

linux 密码保存在

/etc/shadow

shadow 文件的保存格式

```
python:$6$mWSyC6Pv$hpMreQT77R9ML/Xx1QnRAow1tUTDjIowaTssV7bZw9S44FXyd93kfrFQ8Y2vpj/bQfrub/Q.Z6XkYDt4gAUBE0:17514:0:99999:7:::
```

加密格式：

{用户名}：{加密后的口令密码}：{口令最后修改时间距原点(1970-1-1)的天数}：  
{口令最小修改间隔(防止修改口令，如果时限未到，将恢复至旧口令)}：{口令最大修改间隔}：{口令失效前的警告天数}：{账户不活动天数}：{账号失效天数}：{保留}

加密后的口令密码是由三部分组成的，即：*idsalt\$encrypted*。

id 为 1 时，采用 md5 进行加密；

id 为 5 时，采用 SHA256 进行加密；

id 为 6 时，采用 SHA512 进行加密。

shadow 只有超级用户才有权限查看，所以拷贝出来的 shadow 文件需要修改权限

```
chmod 755 shadow
```

### 2、crypt 模块

在 python 中有一个 crypt 模块可以进行密码计算,而且不用我们考虑加密方式是什么。

crypt 方法接收两个参数，第一个是需要加密的明文，第二个参数是 salt.

linux salt 为 6mWSyC6Pv\$ 形式 其中 6 表示加密方式 sha512

```
>>> import crypt
>>> crypt.crypt('sdssdd', '$6$mWSyC6Pv$')
'$6$mWSyC6Pv$8AhSQcqAsoGsMIFHnLhvCRf4cg5gWvPHhWtliw39yIGUYwK5uOD06bzL7GUxD.X.U
5N14m3MA0Pikf5fsy00/'
>>>
```

### 3、功能实现

```
import crypt

def testPass(cryptPass):
    salt = cryptPass[0:12] # 获取加密方式以及密码盐
    try:
        dictFile = open('dictionary', 'r') # 打开密码字典
        try:
            for word in dictFile.readlines():
                word = word.strip('\n')
                cryptWord = crypt.crypt(word, salt) # 将字典中读取的明文密码
                通过加密与 shadow 密码进行判断
                if cryptWord == cryptPass:
                    print('[+] Found Password: ' + word + "\n")
                    return
            print('[-] Password not Found.\n')
            return
        except:
            dictFile.close()
    except:
        print('[-] dictionary 文件无法打开')
        exit(0)

def main():
    try:
        passFile = open('shadow') # 打开 shadow 文件
        try:
            for line in passFile.readlines(): # 一行一行读取
                if ":" in line:
                    user = line.split(':')[0] # 利用 : 分隔 获取用户名
                    cryptPass = line.split(':')[1].strip(' ') # 去掉前后空格
                    print('[*] Cracking Password For : ' + user)
                    testPass(cryptPass)
        except:
            passFile.close()
```



```
except:
    print('[ - ] shadow 文件无法打开')

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

## 五、文件定位

### 1、tell()

文件定位，指的是当前文件指针读取到的位置，光标位置。在读写文件的过程中，如果想知道当前的位置，可以使用 `tell()` 来获取

```
# 以读模式打开 test.txt 文件
f = open('test.txt', 'r')
content = f.read(3) # 读取三个字符
# 查看当前游标所在位置
cur = f.tell()
print(cur)

content = f.read(3) # 读取三个字符
# 查看当前游标所在位置
cur = f.tell()
print(cur)
```

### 2、seek()

如果在操作文件的过程，需要定位到其他位置进行操作，用 `seek()`。

`seek(offset, from)` 有 2 个参数, `offset`，偏移量单位字节，负数是往回偏移，正数是往前偏移，`from` 位置：0 表示文件开头，1 表示当前位置，2 表示文件末尾

注意：Python3 中取消了当前位置与文末偏移。只能将光标定位到文末，不可往回偏移。`seek` 方法在 Python3 中只能做文件开头往前偏移，或者将光标定位到文末。

示例（Python2 版本）：

```
# 以读模式打开 test.txt 文件
f = open('test.txt', 'r')
content = f.read(3) # 读取三个字符
print(content)
f.seek(-2, 1) # 在当前位置往回偏移两个字节
```

```
content = f.read(3) # 读取三个字符
print(content)

f.seek(-5,2) # 定位到文章末尾，往回偏移 5 个字节
content = f.read(3) # 读取三个字符
print(content)

f.seek(5,0) # 定位到文章末尾，往回偏移 5 个字节
content = f.read(3) # 读取三个字符
print(content)

# 偏移量为负数，是往回偏移，正数是往前偏移
```

## 六、模块介绍

我们经常说 Python 有强大的第三方库，有很多常用功能 Python 提供自带的内置模块。简单的讲模块就是封装了一堆函数的 py 文件，就像是一个工具包，要使用里面的工具，得先将工具包那过来。模块也是一样，在程序中需要用到模块得先将模块导入。

### 1、import 导入模块

在 Python 中导入模块使用 import 关键字。比如我们导入一个时间模块 time，获取当前时间。模块导入一般放在文件最前面。

```
>>> import time
>>> time.ctime()
'Wed Apr 11 15:03:34 2018'
>>>
```

调用模块的方法，格式：模块名.函数名，这样调用可以防止不同模块中有同名方法导致错误。

### 2、搜索路径

当解释器遇到 import 关键字，如果模块在当前的搜索路径就会被导入。查看 import 导入的搜索路径，导入模块时会优先搜索当前目录下是否有这个名字的模块，所以在模块命名的时候不要与系统中的模块有重名。

(1)、当前目录

(2)、如果当前目录没有，到环境变量中搜索，可以用 `sys` 模块中的 `path` 变量查看所有路径。

(3)、如果都找不到，搜索默认路径，linux 系统，默认路径一般为 `/usr/local/lib/python/`

第三方模块安装位置，在 `python` 安装目录下的 `lib/site-packages/` 目录下。

```
>>> import sys
>>> sys.path
['', '/usr/lib/python3.5.zip', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/home/python/.local/lib/python3.5/site-packages', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages']
>>>
>>>
>>>
```

表示当前目录

第三方模块目录

### 3、from ... import 导入模块的方法

一个模块可能会存在很多函数，如果只想导入其中几个函数，可以使用 `from xx import xx` 方式导入

示例：只想导入 `time` 模块中的 `ctime`，`time` 两个方法，可以这样导入。

```
>>> from time import ctime,time
>>> ctime()
'Wed Apr 11 15:09:07 2018'
>>> time()
1523430550.5778434
>>>
```

使用 `from` 导入，如果函数名相同，后面导入的会覆盖前面导入的。

把模块中的所有函数一次性全部导入

语法格式：`from xxx import *`

示例：

```
from time import *
```

### 4、as 给模块取别名

有时候导入的模块名称很长，调用的时候很不方便，这个使用就可以用 `as` 给这个模块取别名。

```
In [16]: import time as mytime
In [17]: time.time() 取别名后不能用原来模块名字调用
AttributeError                                Traceback (most recent call last)
<ipython-input-17-632c90c82b25> in <module>()
----> 1 time.time()

AttributeError: 'builtin_function_or_method' object has no attribute 'time'

In [18]: mytime.time() 调用模块的时候需要用别名
Out[18]: 1512705440.852654

In [19]: _
```

## 七、os 模块操作文件

### 1、概述

对文件进行重命名、删除等一些操作，在 python 中可以利用 os 模块。

os 模块提供一些系统级别的操作命令。

方法	解释
os.getcwd()	获取当前工作目录，即当前 python 脚本工作的目录路径
os.chdir("dirname")	改变当前脚本工作目录；相当于 shell 下 cd
os.curdir	返回当前目录: ('.')
os.pardir	获取当前目录的父目录字符串名: ('..')
os.makedirs('dir1/dir2')	可生成多层递归目录
os.removedirs('dirname1')	若目录为空，则删除，并递归到上一级目录，如若也为空，则删除，依此类推
os.mkdir('dirname')	生成单级目录；相当于 shell 中 mkdir dirname
os.rmdir('dirname')	删除单级空目录，若目录不为空则无法删除，报错
os.listdir('dirname')	列出指定目录下的所有文件和子目录，包括隐藏文件，并以列表方式打印
os.remove()	删除一个文件
os.rename("oldname","new")	重命名文件/目录
os.stat('path/filename')	获取文件/目录信息
os.sep	操作系统特定的路径分隔符，win 下为"\\",Linux 下为"/"
os.linesep	当前平台使用的行终止符，win 下为"\t\n",Linux 下为"\n"

<code>os.pathsep</code>	用于分割文件路径的字符串
<code>os.name</code>	字符串指示当前使用平台。win->'nt'; Linux->'posix'
<code>os.system("bash command")</code>	运行 shell 命令，直接显示
<code>os.environ</code>	获取系统环境变量
<code>os.path.abspath(path)</code>	返回 path 规范化的绝对路径
<code>os.path.split(path)</code>	将 path 分割成目录和文件名二元组返回
<code>os.path.dirname(path)</code>	返回 path 的目录。其实就是 <code>os.path.split(path)</code> 的第一个元素
<code>os.path.basename(path)</code>	返回 path 最后的文件名。如何 path 以 / 或 \ 结尾，那么就会返回空值。即 <code>os.path.split(path)</code> 的第二个元素
<code>os.path.exists(path)</code>	如果 path 存在，返回 True；如果 path 不存在，返回 False
<code>os.path.isabs(path)</code>	如果 path 是绝对路径，返回 True
<code>os.path.isfile(path)</code>	如果 path 是一个存在的文件，返回 True。否则返回 False
<code>os.path.isdir(path)</code>	如果 path 是一个存在的目录，则返回 True。否则返回 False
<code>os.path.join(path1[, path2[, ...]])</code>	将多个路径组合后返回，第一个绝对路径之前的参数将被忽略
<code>os.path.getatime(path)</code>	返回 path 所指向的文件或者目录的最后存取时间
<code>os.path.getmtime(path)</code>	返回 path 所指向的文件或者目录的最后修改时间

## 2、常用方法演示

修改文件名：

`rename(需要修改的文件名, 新的文件名)`

```
import os
os.rename('1.py', '666.py')
```

删除文件：

`remove(待删除的文件名)`

```
import os
os.remove('3.py')
```

创建文件夹: `mkdir(文件夹名称)`

```
import os
os.mkdir('gl')
```

删除文件夹: `rmdir(文件夹名称)`

```
import os
os.rmdir('gl')
```

获取当前目录: `getcwd()`

```
import os
os.getcwd()
```

切换目录 `chdir(路径)`

```
import os
os.chdir('../')
```

路径拼接 `os.path.join(path1[, path2[, ...]])` 将多个路径组合后返回

```
import os
path = os.path.join(os.getcwd(), 'gl')
```

## 八、time、datetime 模块

### 1、time 模块的使用

`time.sleep(num)`，让程序执行暂停，num 单位是秒

```
import time
time.sleep(5) # 程序暂停 5s
```

`time.time()`，返回时间戳时间戳

```
import time
print(time.time()) # 返回时间戳
print(time.ctime()) # 字符串格式
print(time.asctime()) # 返回时间格式"Fri Dec 8 14:31:15 2017",
ret= time.gmtime() # 返回utc时间的struct时间对象格式
print("%s-%s-%s"%(ret.tm_year,ret.tm_mon,ret.tm_mday))
print(time.localtime()) # 本地时间
print(time.strftime("%Y-%m-%d %H:%M:%S",time.localtime())) # 格式化输出时间
```

时间格式化输出:

```
In [47]: import time

In [48]: print(time.localtime()) # 本地时间
time.struct_time(tm_year=2017, tm_mon=12, tm_mday=8, tm_hour=14, tm_min=45, tm_sec=41, tm_wday
=4, tm_yday=342, tm_isdst=0)

In [49]: print(time.strftime("%Y-%m-%d %H:%M:%S",time.localtime())) #格式化输出时间
2017-12-08 14:45:47

In [50]:
```

## 时间格式化符号

- %y 两位数的年份表示（00-99）
- %Y 四位数的年份表示（000-9999）
- %m 月份（01-12）
- %d 月内中的一天（0-31）
- %H 24 小时制小时数（0-23）
- %I 12 小时制小时数（01-12）
- %M 分钟数（00=59）
- %S 秒（00-59）
- %a 本地简化星期名称
- %A 本地完整星期名称
- %b 本地简化的月份名称
- %B 本地完整的月份名称
- %c 本地相应的日期表示和时间表示
- %j 年内的一天（001-366）
- %p 本地 A.M.或 P.M.的等价符
- %U 一年中的星期数（00-53）星期天为星期的开始
- %w 星期（0-6），星期天为星期的开始
- %W 一年中的星期数（00-53）星期一为星期的开始
- %x 本地相应的日期表示

- %X 本地相应的时间表示
- %Z 当前时区的名称
- %% %号本身

## 2、datetime 模块

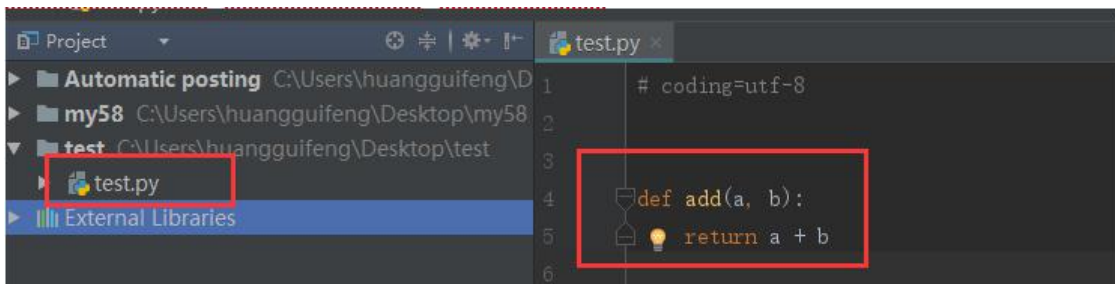
datetime 模块主要用于时间计算。

```
import datetime
print(datetime.datetime.now()) #返回 2017-12-08 15:00:18.312624
print(datetime.date.fromtimestamp(time.time()))#时间戳直接 (1512715165.0285344)
转成日期格式 2017-12-08
print(datetime.datetime.now() )
print(datetime.datetime.now() + datetime.timedelta(3)) #当前时间+3 天
print(datetime.datetime.now() + datetime.timedelta(-3)) #当前时间-3 天
print(datetime.datetime.now() + datetime.timedelta(hours=3)) #当前时间+3 小时
print(datetime.datetime.now() + datetime.timedelta(minutes=30)) #当前时间+30 分
print(datetime.datetime.now() + datetime.timedelta(hours=3, minutes=30)) #当前
时间 1 小时 30 分.
```

## 九、模块的制作、发布、安装

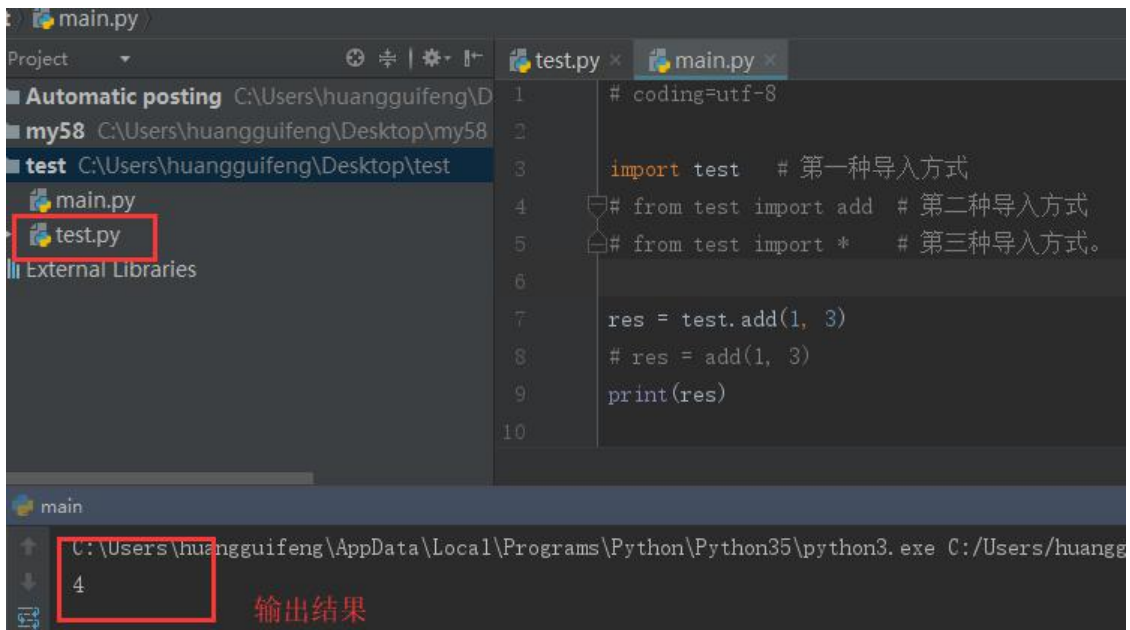
### 1、模块的制作

(1)、Python 文件都可以作为一个模块，模块的名字就是文件的名字。比如创建一个 test.py 文件，文件中创建一个 add 函数。test.py 就是一个模块。

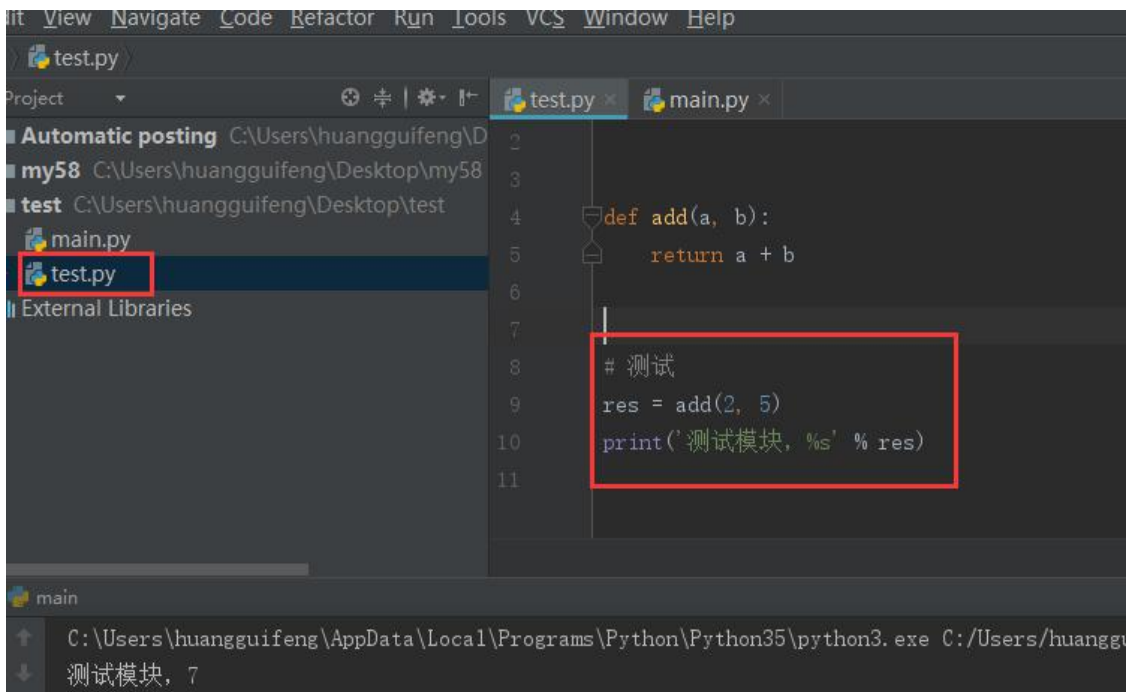


(2)、调用 test.py 模块





(3)、模块测试一般写完模块之后都会进行测试，下面来看下这个例子 写好模块之后，在模块中写了一段测试的代码。



在 main.py 导入 test 模块，执行时模块中的测试代码也执行了。

```
main.py
1 # coding=utf-8
2
3 import test
4
5 res = test.add(1, 3) # 调用模块的add方法
6 print(res)
7
8
```

main

C:\Users\huangguifeng\AppData\Local\Programs\Python\Python35\python3.exe C:/Users/huangguifeng/Desktop/test.py

测试模块, 7

导入模块的时候模块里的测试代码执行了

(4)、为了避免这种情况使用到一个\_\_name\_\_的变量。

```
test.py
2
3
4 def add(a, b):
5     return a + b
6
7
8 # 测试
9 # res = add(2, 5)
10 # print('测试模块, %s' % res)
11 print('模块__name__变量 = %s' % __name__)
```

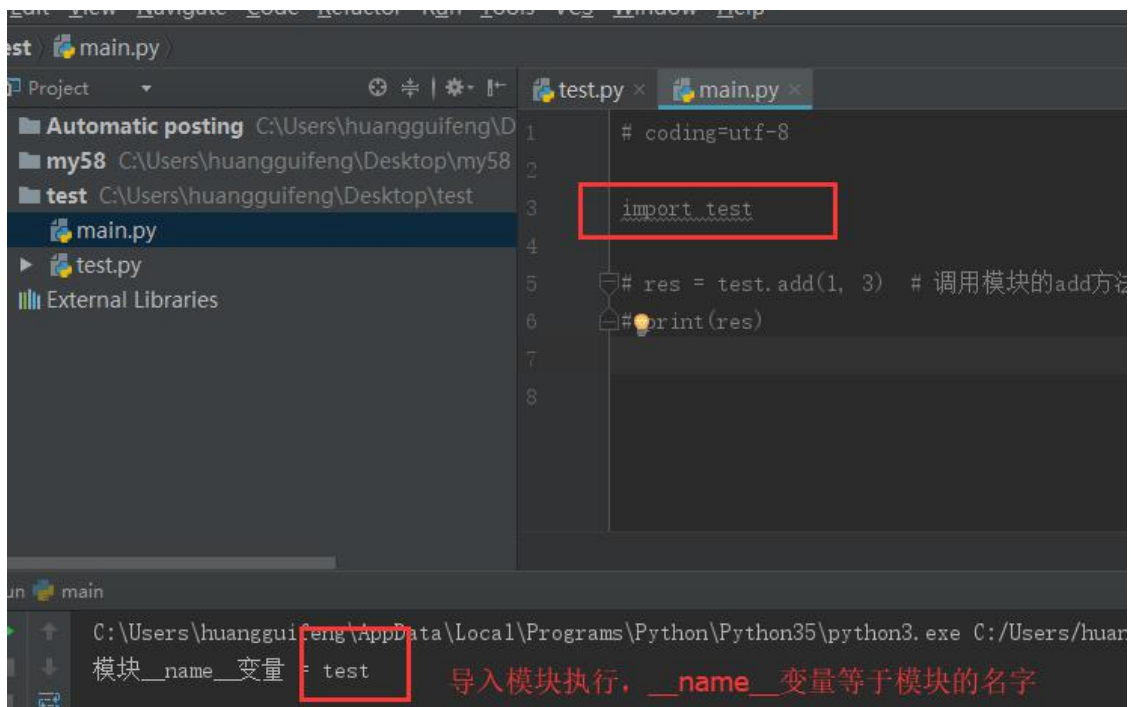
test

C:\Users\huangguifeng\AppData\Local\Programs\Python\Python35\python3.exe C:/Users/huangguifeng/Desktop/test.py

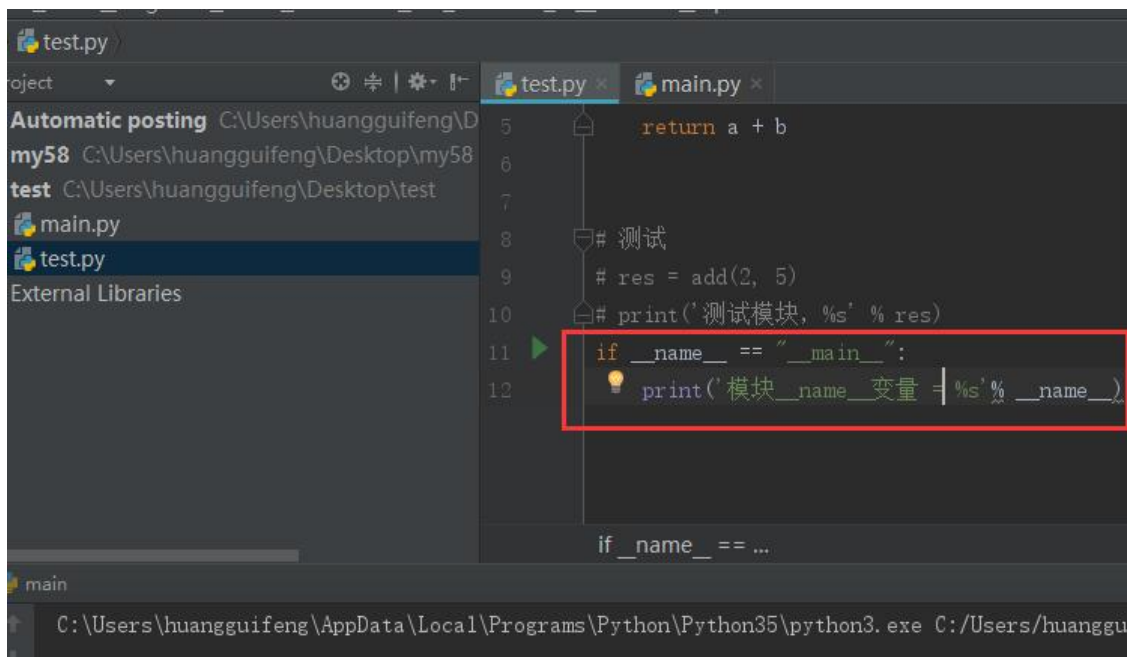
模块\_\_name\_\_变量 = \_\_main\_\_

直接运行test.py, \_\_name\_\_变量等于\_\_main\_\_

在 main.py 中导入执行



知道\_\_name\_\_变量的原理之后, 就可以很好的处理测试代码了。将测试的代码放到 if \_\_name\_\_ == '\_\_main\_\_':



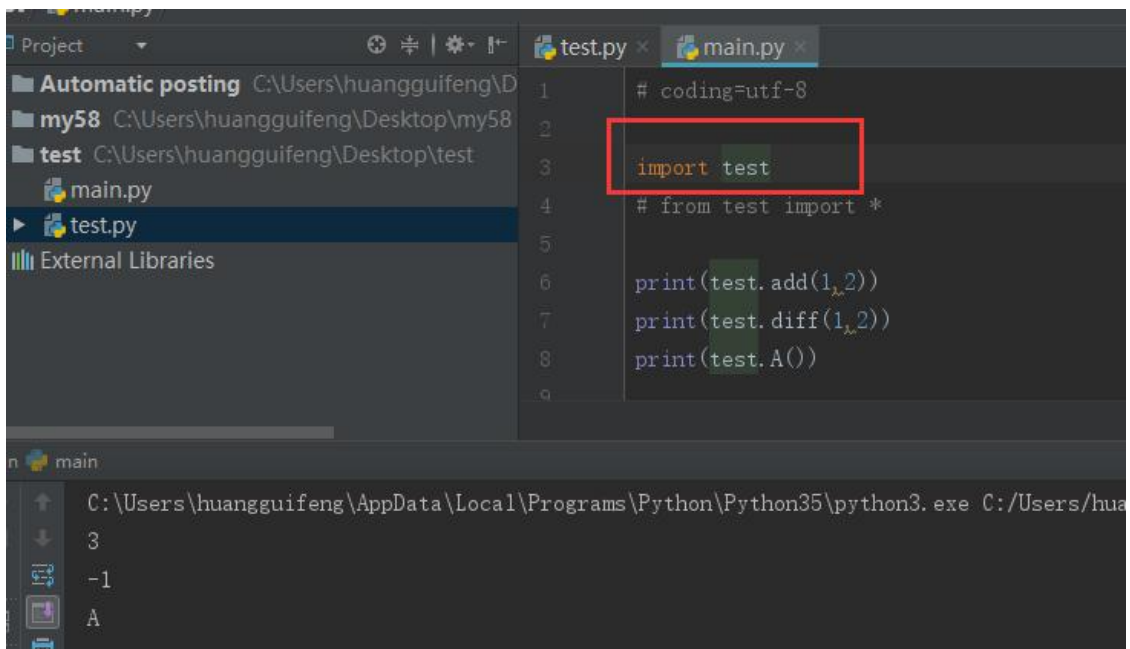
(5)、\_\_all\_\_ 的作用, 如果一个文件中有\_\_all\_\_变量, 那么也就意味着这个变量中的元素, 不会被 from xxx import \* 时导入



The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'Automatic posting' with subfolders 'my58' and 'test'. The 'test' folder contains 'main.py' and 'test.py'. The code editor shows 'test.py' with the following code:

```
1 # coding=utf-8
2
3 __all__ = ['add', 'diff']
4
5 def add(a, b):
6     return a + b
7
8 def diff(a, b):
9     return a - b
10
11 def A():
12     return 'A'
```

有 all 变量 import 方式导入，可以无异常，可以正常使用。



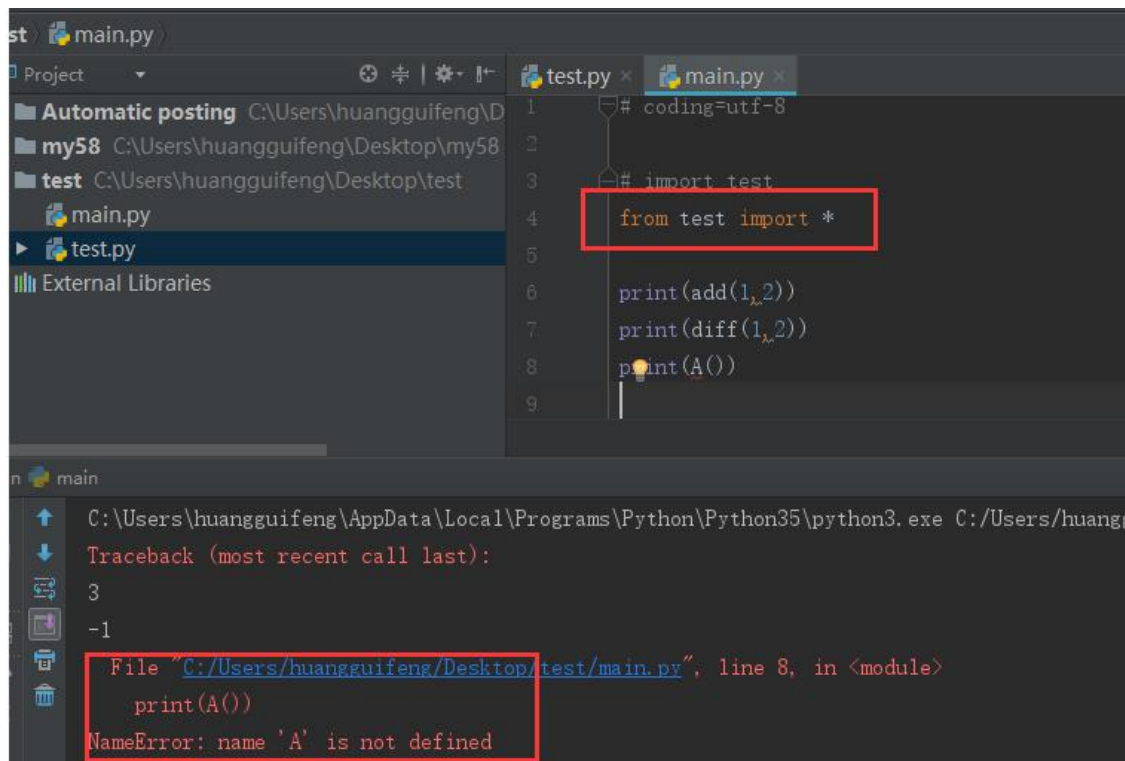
The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'Automatic posting' with subfolders 'my58' and 'test'. The 'test' folder contains 'main.py' and 'test.py'. The code editor shows 'main.py' with the following code:

```
1 # coding=utf-8
2
3 import test
4 # from test import *
5
6 print(test.add(1,2))
7 print(test.diff(1,2))
8 print(test.A())
9
```

Below the code editor, there is a terminal window showing the command prompt output:

```
C:\Users\huangguifeng\AppData\Local\Programs\Python\Python35\python3.exe C:/Users/hua
3
-1
A
```

from test import \* 方式导入

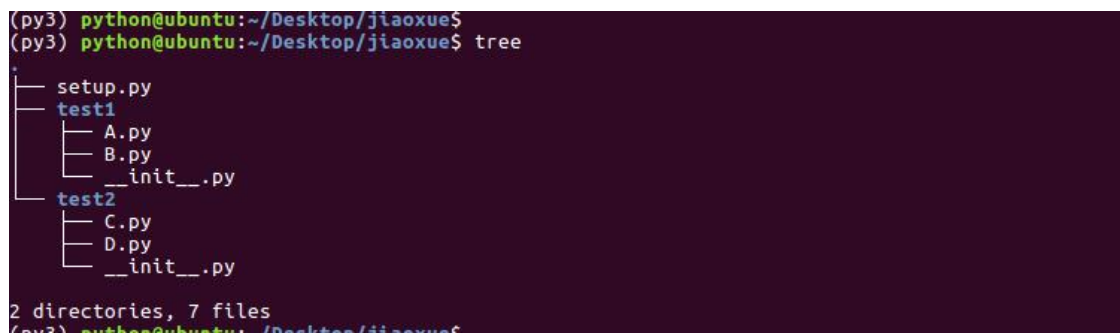


上述例子可以看出：使用 `from xxx import` 导入方式，不在 `_all_` 变量中是无法导入的，其他导入方式不影响。

## 2、模块的发布

平时使用第三方模块是其他开发者发布出来，需要安装后调用。下面我们来学习怎么去发布一个模块。

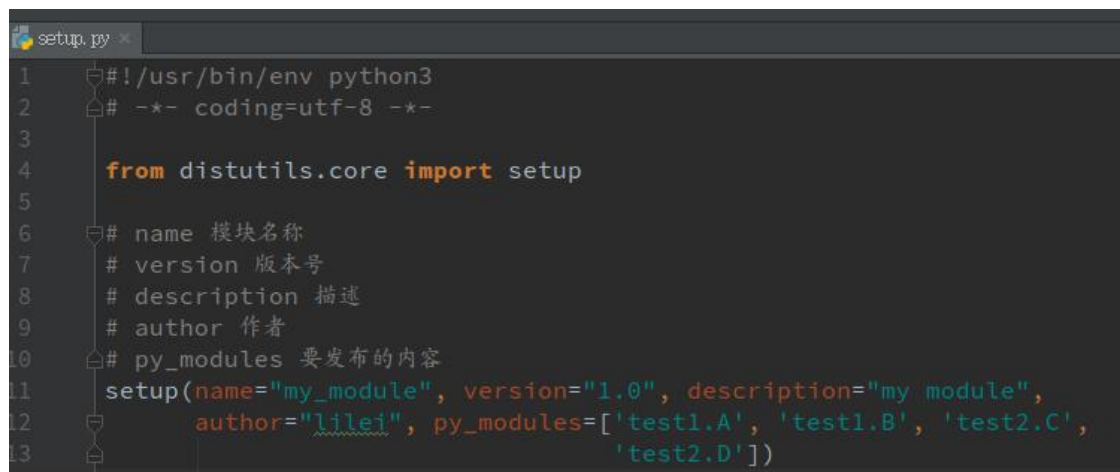
(1)、将写好的包放到一个 `jiaoxue/` 目录下



(2)、在 `jiaoxue/` 目录下创建一个文件 `setup.py` 文件

文件里写入下面代码

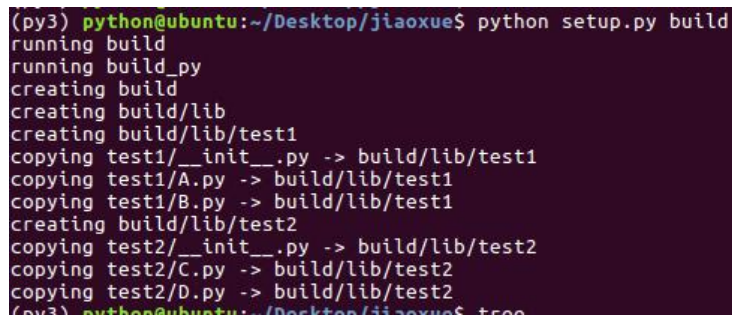
```
from distutils.core import setup
# name 模块名称
# version 版本号
# description 描述
# author 作者
# py_modules 要发布的内容
setup(name="my_module", version="1.0", description="my module",
      author="lilei", py_modules=['test1.A', 'test1.B', 'test2.C',
                                  'test2.D'])
```



```
1  #!/usr/bin/env python3
2  # -*- coding=utf-8 -*-
3
4  from distutils.core import setup
5
6  # name 模块名称
7  # version 版本号
8  # description 描述
9  # author 作者
10 # py_modules 要发布的内容
11 setup(name="my_module", version="1.0", description="my module",
12       author="lilei", py_modules=['test1.A', 'test1.B', 'test2.C',
13                                   'test2.D'])
```

### (3)、创建模块

python setup.py build



```
(py3) python@ubuntu:~/Desktop/jiaoxue$ python setup.py build
running build
running build_py
creating build
creating build/lib
creating build/lib/test1
copying test1/__init__.py -> build/lib/test1
copying test1/A.py -> build/lib/test1
copying test1/B.py -> build/lib/test1
creating build/lib/test2
copying test2/__init__.py -> build/lib/test2
copying test2/C.py -> build/lib/test2
copying test2/D.py -> build/lib/test2
(py3) python@ubuntu:~/Desktop/jiaoxue$ tree
```

### (4)、生成压缩包

python setup.py sdist



```
(py3) python@ubuntu:~/Desktop/jiaoxue$ python setup.py sdist
running sdist
running check
warning: check: missing required meta-data: url
warning: check: missing meta-data: if 'author' supplied, 'author_email' must be supplied too
warning: sdist: manifest template 'MANIFEST.in' does not exist (using default file list)
warning: sdist: standard file not found: should have one of README, README.txt

writing manifest file 'MANIFEST'
creating my_module-1.0
creating my_module-1.0/test1
creating my_module-1.0/test2
making hard links in my_module-1.0...
hard linking setup.py -> my_module-1.0
hard linking test1/A.py -> my_module-1.0/test1
hard linking test1/B.py -> my_module-1.0/test1
hard linking test1/__init__.py -> my_module-1.0/test1
hard linking test2/C.py -> my_module-1.0/test2
hard linking test2/D.py -> my_module-1.0/test2
hard linking test2/__init__.py -> my_module-1.0/test2
creating dist
Creating tar archive
removing 'my_module-1.0' (and everything under it)
```

生成压缩包

(5)、tree 看下 jiaoxue 目录下的结构

```
(py3) python@ubuntu:~/Desktop/jiaoxue$ tree
.
├── build
│   └── lib
│       ├── test1
│       │   ├── A.py
│       │   ├── B.py
│       │   └── __init__.py
│       └── test2
│           ├── C.py
│           ├── D.py
│           └── __init__.py
├── dist
│   └── my_module-1.0.tar.gz
├── MANIFEST
├── setup.py
├── test1
│   ├── A.py
│   ├── B.py
│   └── __init__.py
└── test2
    ├── C.py
    ├── D.py
    └── __init__.py
```

这个压缩包就可以发布给别人安装了

7 directories, 15 files

### 3、模块的安装

(1)、将上一节生成的压缩包复制到桌面解压

```
tar -zxvf my_module-1.0.tar.gz
```

解压后在桌面会生成一个文件夹 my\_module-1.0

```
(py3) python@ubuntu:~/Desktop$ tar -zxvf my_module-1.0.tar.gz
my_module-1.0/
my_module-1.0/PKG-INFO
my_module-1.0/test2/
my_module-1.0/test2/C.py
my_module-1.0/test2/D.py
my_module-1.0/test2/__init__.py
my_module-1.0/setup.py
my_module-1.0/test1/
my_module-1.0/test1/B.py
my_module-1.0/test1/__init__.py
my_module-1.0/test1/A.py
(py3) python@ubuntu:~/Desktop$ ls
11.txt  a.txt          b.txt  dodi    d.txt  jiaoxue  my_module-1.0.tar.gz
       Automatic posting  c.txt  dodi_spider  hello.py  my_module-1.0  myproject
(py3) python@ubuntu:~/Desktop$
```

(2)、进入 my\_module-1.0 文件夹

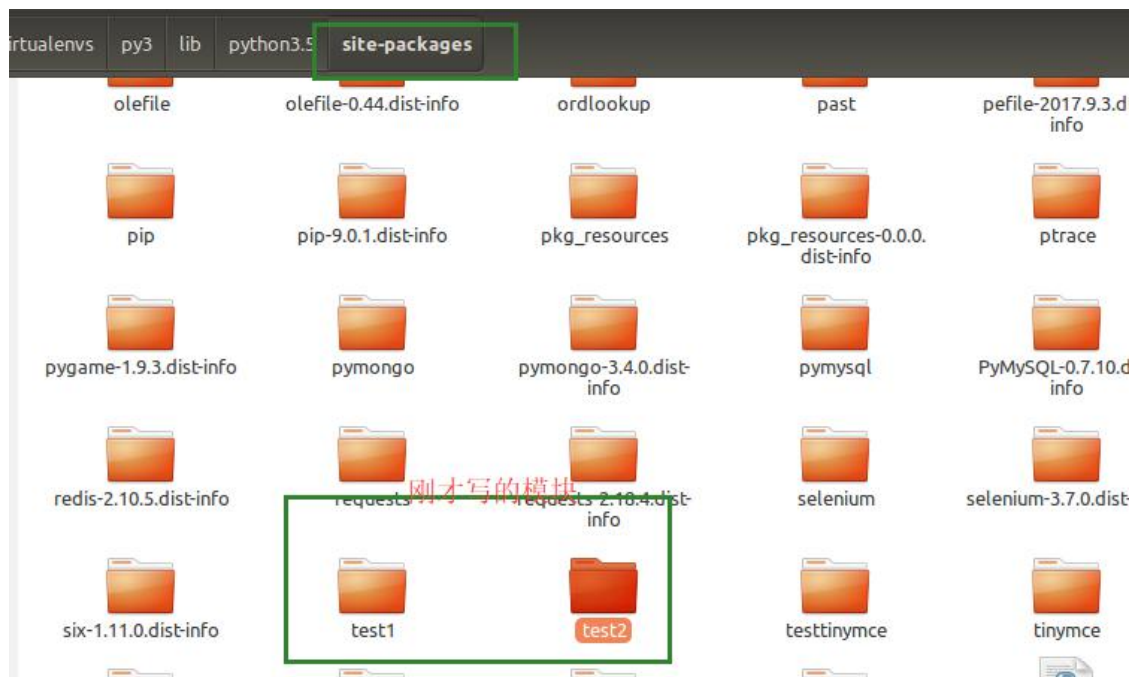
cd my\_module-1.0

(3)、执行命令安装

python setup.py install

(4)、查看是否安装成功

在 python 的安装目录下的 site-packages 目录下



模块引入，可以导入说明安装成功



```
>>> from test1 import A
>>> A.A()
'我来自test1,我是A函数'
>>> from test2 import C
>>> C.C()
'我来自test2,我是C函数'
>>>
```

## 小结

文件打开关闭

文件操作一般步骤、文件打开、文件关闭、with 上下文管理

文件读写

写文件、读文件

应用：文件备份脚本

应用：linux 密码暴力破解机

shadow 文件、crpy 模块、功能实现

文件定位

tell()、seek()

模块介绍

import 导入模块、搜索路径、from...import 导入模块的方法、as 给模块取别名

os 模块操作文件

概述、常用方法

time、datetime 模块

time 模块的使用、datetime 模块

模块的制作、发布、安装

## 课后作业

### 课后问答题

1、文件打开方式有哪些？往文件里追加内容用的是那个模式。

2、怎么查看 python 环境变量。

## 课后实操题

1、现在有一个 5G 的大文件需要做备份，请写一个脚本备份这个文件。

2、有一批文件需要在文件名加上日期，比如文件“亚瑟.py”，要重命名成“2017-12-9\_亚瑟.py 请编写程序完成重命名，（提示：这个脚本会将当前文件夹下的所有文件重命名，最好新建一个测试文件夹完成）

3、time，os 模块使用(练习模块使用方法，熟悉模块各个方法)。

4、编写一个自己的模块，发布并安装。

5、题目：有两个磁盘文件 A 和 B,各存放一行字母,要求把这两个文件中的信息合并(按字母顺序排列),输出到一个新文件 C 中。

6、日志统计

要求：

（1）、有一个日志文件 log.txt，内容如下，一条日志一行。

（2）、将日志的内容按照 ID 进行分组显示，ID 为 262207(2836-G-34) 形式。

（3）、将整理好的文件保存到一个新 new\_log.txt 文件中。

原日志内容样式：

```
[2016-05-23 15:48:02]:262207(2836-G-34):总空间:1629,已用:1058,剩余:571,
预计剩余存储7天,ok.
[2016-05-23 15:49:54]:262207(2836-G-34):总空间:1629,已用:1058,剩余:571,
预计剩余存储7天,ok.
[2016-05-23 15:50:54]:262147(2903-H-4):总空间:1942,已用:1229,剩余:713,预
计剩余存储7天,ok.
[2016-05-23 15:50:54]:262149(2904-C-5):总空间:2802,已用:0,剩余:2802,预计
剩余存储17天,ok.
[2016-05-23 15:50:55]:262151(2901-H-6):总空间:1737,已用:1195,剩余:542,预
计剩余存储6天,ok.
[2016-05-23 15:50:55]:262153(2902-S-7):总空间:1737,已用:1195,剩余:542,预
计剩余存储6天,ok.
[2016-05-23 15:50:55]:262155(2801-H-8):总空间:1737,已用:1218,剩余:519,预
计剩余存储6天,ok.
[2016-05-23 15:50:55]:262157(2907-C-9):总空间:2802,已用:310,剩余:2492,预
计剩余存储16天,ok.
[2016-05-23 15:51:05]:262231(2832-G-46):总空间:1629,已用:1057,剩余:572,
预计剩余存储7天,ok.
[2016-05-23 15:53:57]:262231(2832-G-46):总空间:1629,已用:1057,剩余:572,
预计剩余存储7天,ok.
```

.....

整理后的内容样式:

```
***** 262231 (2832-G-46)
*****
[2016-05-23 15:51:05]:262231 (2832-G-46):总空间:1629, 已用: 1057, 剩余: 572,
预计剩余存储 7 天, ok.
[2016-05-23 15:53:57]:262231 (2832-G-46):总空间:1629, 已用: 1057, 剩余: 572,
预计剩余存储 7 天, ok.
***** 262207 (2836-G-34)
*****
[2016-05-23 15:48:02]:262207 (2836-G-34):总空间:1629, 已用: 1058, 剩余: 571,
预计剩余存储 7 天, ok.
[2016-05-23 15:49:54]:262207 (2836-G-34):总空间:1629, 已用: 1058, 剩余: 571,
预计剩余存储 7 天, ok.
```

## 7、Python 简陋版工资管理系统

要求:

彦 100000 盖伦 80000 莫甘娜 50000 赵信 30000

-----以上是 info.txt 文件----- 实现效果:

从 info.txt 文件中读取员工及其工资信息, 最后将修改或增加的员工工资信息也写入原 info.txt 文件。

需要以下四个功能:

1. 查询员工工资
2. 修改员工工资
3. 增加新员工记录
4. 删除员工信息
5. 退出