py2_day03

模块

文件是物理上组织代码的形式，模块就是逻辑上组织代码的形式。模块名是文件名去掉.py。

导入模块

python导入模块时，将会从以下两个位置搜索模块：

- sys.path定义的路径
- PYTHONPATH环境变量定义的路径

```
>>> import sys
>>> sys.path
['', '/usr/local/lib/python36.zip',
'/usr/local/lib/python3.6', '/usr/local/lib/python3.6/lib-
dynload', '/usr/local/lib/python3.6/site-packages']
```

导入模块的方法

```
# 常用的导入方法
>>> import os
>>> from time import strftime

# 不常用的导入方法
>>> import random, datetime
>>> import pickle as p
```

导入和加载

- import是导入
- load是加载。加载就是将模块中的代码运行一遍
- 不管导入多少次，只会加载一次。

hashlib模块

计算哈希值的模块。

- hash哈希是单向加密的算法。
- 通过原始数据可以生成固定长度的乱码

- 原始数据相同，乱码也一定相同
- 原始数据有微小的不同，乱码也一定完全不同
- 不能通过乱码反推回原始数据
- 常用的算法有：md5、sha
- 经常用于：存储加密密码、文件完整性校验

```
>>> import hashlib
>>> m = hashlib.md5(b'123456')
>>> m.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'

# 如果需要计算的数据量非常大，可以分批量进行更新
>>> m1 = hashlib.md5()
>>> m1.update(b'12')
>>> m1.update(b'34')
>>> m1.update(b'56')
>>> m1.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'
```

tarfile模块

实现归档压缩功能，支持gzip、bzip2、lzma格式的压缩。

```
>>> import tarfile
>>> tar = tarfile.open('/tmp/demo/myfile.tar.gz', 'w:gz')
>>> tar.add('/etc/security')
>>> tar.add('/etc/hosts')
>>> tar.close()

>>> tar = tarfile.open('/tmp/demo/myfile.tar.gz', 'r')
# 不指定解压目标，默认解压到当前目录
>>> tar.extractall('/tmp/demo')
>>> tar.close()
```

os.walk方法

```
>>> os.walk('/tmp/demo/security')
<generator object walk at 0x7f5a2cccfe60>
>>> list(os.walk('/tmp/demo/security'))
[('/tmp/demo/security', ['console.apps',
'console.perms.d', 'limits.d', 'namespace.d'],
['access.conf', 'chroot.conf', 'console.handlers',
'console.perms', 'group.conf', 'limits.conf',
'namespace.conf', 'namespace.init', 'opasswd',
'pam_env.conf', 'sepermit.conf', 'time.conf',
'pwquality.conf']), ('/tmp/demo/security/console.apps',
[], ['config-util', 'xserver', 'liveinst', 'setup']),
('/tmp/demo/security/console.perms.d', [], []),
('/tmp/demo/security/limits.d', [], ['20-nproc.conf']),
('/tmp/demo/security/namespace.d', [], [])]
>>> a = list(os.walk('/tmp/demo/security'))
>>> len(a)
5
>>> a[0]
('/tmp/demo/security', ['console.apps', 'console.perms.d',
'limits.d', 'namespace.d'], ['access.conf', 'chroot.conf',
'console.handlers', 'console.perms', 'group.conf',
'limits.conf', 'namespace.conf', 'namespace.init',
'opasswd', 'pam_env.conf', 'sepermit.conf', 'time.conf',
'pwquality.conf'])
>>> a[1]
('/tmp/demo/security/console.apps', [], ['config-util',
'xserver', 'liveinst', 'setup'])
```

os.walk()方法返回的数据由多个元组构成。每个元组又有三项，这三项分别是：路径字符串、该路径下的目录列表、该路径下的文件列表。

```
>>> list(os.walk('/tmp/demo/security'))
[
    ('父目录', ['父目录下的子目录列表'], ['父目录下的文件列
表']),
    ('子目录1', ['子目录1下的孙目录列表'], ['子目录下的文件列
表']),
    ('子目录2', ['子目录2下的孙目录列表'], ['子目录下的文件列
表']),
]
```

```
>>> for path, folders, files in
os.walk('/tmp/demo/security'):
...     for file in files:
...         os.path.join(path, file)
```

OOP：面向对象的编程

如果使用以前所学知识，可以定义字典保存人物属性，创建函数定义人物的行为：

```
lvbu = {'name': '吕布', 'weapon': '方天画戟', 'sex': '男'}
def walk():
    pass
def attack():
    pass
```

OOP的思想，是将现实世界的物体抽象成一个类class，这个类中有属性和行为，将数据和行为融合到一起。相当于创建了一个蓝图，然后再根据蓝图创建出具体的实例。

```
class GameCharacter:
    def __init__(self, name, weapon):
        self.name = name
        self.weapon = weapon
    def speak(self, word):
        print('我是%s, %s' % (self.name, word))

>>> lvbu = GameCharacter('吕布', '方天画戟')
>>> lvbu.name
'吕布'
>>> lvbu.weapon
'方天画戟'
>>> lvbu.speak('人在塔在')
我是吕布, 人在塔在
>>> guanyu = GameCharacter('关羽', '青龙偃月刀')
>>> guanyu.name
'关羽'
>>> guanyu.weapon
'青龙偃月刀'
>>> guanyu.speak('呵呵')
我是关羽, 呵呵
```

__init__方法一般用于为实例对象绑定属性。当创建实例的时候，__init__方法自动调用，实例（lvbu）会作为第一个参数传递。self不是关键字，java用this，但是都不是必须的名字，可以随意更换。一旦创建了实例，实例就会自动拥有类中定义的属性和方法（函数）。

self.属性 是绑定到某个实例上的属性，该属性在所有方法中均可见可用。没有绑定到对象上的，只是局部变量。

组合

当两个类完全不同，其中一个类是另一个类的组件时，使用组合。

继承

当两个类有很多相似之处，只有一部分不同，使用继承。

子类可以有多个父类。当多个类有同名方法的时候，查找的顺序是自下向上，自左向右。

正则表达式

例：为mac地址加冒号

1. 定位到mac地址
2. 每2个mac地址分一组
3. 在组之间加冒号

```
192.168.1.1     00525412A3B4
192.168.1.2     000C29123456

:%s/\(..\)\(..\)\(..\)\(..\)\(..\)\
(..\)$/\1:\2:\3:\4:\5:\6/
```