一、模块

模块名： 按照逻辑来组织Python代码的方法
模块文件： 物理层上组织模块的方法
名称空间：名称到对象的关系映射集合，每个模块都定义了它自己唯一的名称空间
加载模块时，有一个搜索路径的过程。

- python导入模块，在以下路径搜索：
  1、 sys.path定义的路径
  如果希望将自己写的程序放到sys.path中，可以使用site-packages这个目录

```
>>> 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']
```

2、PYTHONPATH环境变量定义的路径
定义PYTHONPATH：
export PYTHONPATH=模块所在路径

- 导入模块的方法：

序号	方法	例子
1	一行导入多个模块	import os, time, pickle #可读性不好，不推荐
2	从某个模块中导入功能	from random import randint, choice #常用
3	导入模块后设置别名	import pickle as p

- 加密的方式：

序号	加密类型	常用算法
1	对称加密	DES/3DES/AES
2	非对称加密	RSA/DSA
3	信息完整性校验	MD5/SHA

- 单向加密的作用：

序号	作用
1	存储加密的密码
2	校验文件检查文件的完整性

pycharm规范化代码的方法：
1、code -> reformat code

2、code -> optimize Imports

导入和加载
多次导入模块，只会加载一次，加载的时候顶层代码将会被执行

包：将目录当成一个特殊的模块

```
[root@room9pc01 project07]# mkdir mydemo
[root@room9pc01 project07]# cd mydemo
hi = 'hello world'
>>> import mydemo.foo
>>> mydemo.foo.hi
'hello world'
>>> exit()

[root@room9pc01 project07]# vim mydemo/__init__.py
[root@room9pc01 project07]# cat mydemo/__init__.py
star = '*' * 30
>>> import mydemo
>>> mydemo.star
'******************************'
>>> mydemo.foo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'mydemo' has no attribute 'foo'
>>> from mydemo import foo
>>> foo.hi
'hello world'
```

了解绝对导入和相对导入

hashlib模块
生成校验码：

```
>>> import hashlib
>>> m = hashlib.md5(b'123456')   #得到123456的md5对象，hashlib.md5(类型要是byte类型)
>>> m.hex()
>>> m.hexdigest()                #获取123456的md5值（16进制的数）
'e10adc3949ba59abbe56e057f20f883e'
```

文件太大的时候，可以分开逐个部分更新md5，最后生成一个总的md5值

```
>>> m = hashlib.md5()
>>> m.update(b'12')
>>> m.update(b'34')
>>> m.update(b'56')
>>> m.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'
```

校验文件时，可以将文件以b的方式读取出来，再校验

```
>>> with open('/tmp/passwd', 'rb') as fobj:
...     data = fobj.read()
...
>>> m = hashlib.md5(data)
>>> m.hexdigest()
'4efee2481b1327b5a909ecdf417ad332'
```

原文件的mdb如下：

```
[root@room9pc01 project07]# md5sum /tmp/passwd
4efee2481b1327b5a909ecdf417ad332  /tmp/passwd
```

案例：写python程序来校验一个大文件，生成md5校验码

```
import sys
import hashlib
def check_md5(fname):
    m = hashlib.md5()
    with open(fname, 'rb') as fobj:
        while True:
            data = fobj.read(4096)
            if not data:
                break
            m.update(data)

    return  m.hexdigest()

if __name__ == '__main__':
    print(check_md5(sys.argv[1]))
```

tarfile模块

压缩文件：

```
>>> import os
>>> import tarfile
>>> tar = tarfile.open('/tmp/security.tar.gz', 'w:gz')
>>> tar.add('/etc/hosts')
>>> os.chdir('/etc')
>>> tar.add('security')
>>> tar.close()
[root@room9pc01 project07]# file /tmp/security.tar.gz
/tmp/security.tar.gz: gzip compressed data, was "security.tar",
last modified: Sat Jan 19 14:06:44 2019, max compression
```

解压缩：

```
>>> os.mkdir('/tmp/demo')
>>> os.chdir('/tmp/demo')
>>> tar = tarfile.open('/tmp/security.tar.gz', 'r:gz')
>>> tar.extractall()
>>> tar.close()
```

作业：备份程序

要求如下：

支持完全备份和增量备份

周一执行全备

其它时间执行增量备份

备份文件以gzip格式压缩为一个tar包

1、判断文件是否需要增量备份

周一：｛文件1：对应的md5, 文件2：对应的md5｝

2、完全备份：把目录打包，计算每个文件的md5值

3、增量备份：判断有那些新增文件和改动的文件，打包这些文件，更新文件的md5的值

4、递归列出目录中的所有文件

```
= os.walk('/tmp/demo/security')
>>> a.__next__()
('/tmp/demo/security', ['namespace.d', 'console.apps', 'console.perms.d', 'limits.d'], ['pwquality.co
>>> a.__next__()
('/tmp/demo/security/namespace.d', [], [])
>>> a.__next__()
('/tmp/demo/security/console.apps', [], ['xserver', 'setup', 'liveinst', 'config-util'])
>>> a.__next__()
('/tmp/demo/security/console.perms.d', [], [])
>>> a.__next__()
('/tmp/demo/security/limits.d', [], ['20-nproc.conf'])
>>> a.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> for path, folders, files in os.walk('/tmp/demo/security'):
...     for file in files:
...             os.path.join(path, file)
...
结果：
'/tmp/demo/security/pwquality.conf'
'/tmp/demo/security/console.handlers'
'/tmp/demo/security/console.perms'
'/tmp/demo/security/sepermit.conf'
'/tmp/demo/security/namespace.conf'
'/tmp/demo/security/time.conf'
'/tmp/demo/security/opasswd'
'/tmp/demo/security/namespace.init'
'/tmp/demo/security/limits.conf'
'/tmp/demo/security/pam_env.conf'
'/tmp/demo/security/group.conf'
'/tmp/demo/security/chroot.conf'
'/tmp/demo/security/access.conf'
'/tmp/demo/security/console.apps/xserver'
'/tmp/demo/security/console.apps/setup'
'/tmp/demo/security/console.apps/liveinst'
'/tmp/demo/security/console.apps/config-util'
'/tmp/demo/security/limits.d/20-nproc.conf'
```

二、面向对象

OOP：面向对象的编程

类：用来描述具有相同的属性和方法的对象的集合。

对象是类的实例。

创建类：class语句

class classname:

classname 要用驼峰的方式来定义

```
class PigToy:    #定义玩具类
    pass
piggy = PigToy() #创建实例
```

实例化：创建一个类的实例，类的具体对象

方法：类中定义的函数

对象：通过类定义的数据结构实例。

1. 创建类，绑定方法的案例：

```
class PigToy:   #定义玩具类
    def init(self, name, color):
        self.name = name
        self.color = color

    def show_me(self):
        print('Hi, my name is %s , I am %s' %(self.name, self.color))

#第一个实例 piggy
piggy = PigToy()                    #创建实例
piggy.init('Piggy', 'pink')
#init(piggy, 'Piggy', 'pink')
#   piggy.name = 'Piggy'
#   piggy.color = 'pink'
piggy.show_me()

#第二个实例 george
george = PigToy()
george.init('George', 'red')   #调用实例中的方法
george.show_me()               #调用实例中的方法
```

构造器方法绑定实例（init->**init**），创建实例时，实例本身会作为第一个参数传递给self。

```
class PigToy:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def show_me(self):
        print('Hi, my name is %s , I am %s' %(self.name, self.color))


piggy = PigToy('Piggy', 'pink')
piggy.show_me()
george = PigToy('George', 'red')
george.show_me()
```

三、OOP进阶

组合和派生

组合：让不同的类混合，并加入其它类中来增强功能和代码的重用性。

组合实例：把Vender的类，混入到PigToy的类当中，增强PigToy类的功能。

```
class Vender:
    def __init__(self,company, phone):
        self.company = company
        self.phone = phone

    def call(self):
        print('Calling %s ......' % self.phone)
class PigToy:
    def __init__(self, name, color, company, phone):
        self.name = name
        self.color = color
        self.vender = Vender(company, phone)
```

```
    def show_me(self):
        print('Hi, my name is %s , I am %s' %(self.name, self.color))


piggy = PigToy('Piggy', 'pink', 'tedu', '400-800-1234')
#piggy.show_me()
print(piggy.vender.company)
piggy.vender.call()
```

派生：对一个已经定义好的类进行扩展或修改而不影响现存类及其代码片段。

应用场景：两个类很相似，只是小部份功能不一样时，派生是最好的选择。

派生的案例：

```
class PigToy:
    def __init__(self, name, color, company, phone):
        self.name = name
        self.color = color
        self.vender = Vender(company, phone)

    def show_me(self):
        print('Hi, my name is %s , I am %s' %(self.name, self.color))

class NewPigToy(PigToy):  #新类继承了它父类的所有属性
    def walk(self):
        print('walking')

a = NewPigToy('piggy', 'pink')
a.show_me()
a.walk()
```

子类、超类、基类（父类）

```
class PigToy:
    def __init__(self, name, color):
        self.name = name
        self.color = color


    def show_me(self):
        print('Hi, my name is %s , I am %s' %(self.name, self.color))

class NewPigToy(PigToy):  #新类继承了它父类的所有属性
    def __init__(self, name, color, size):
        #PigToy.__init__(self, name, color)
        super(NewPigToy, self).__init__(name, color)
        self.size = size

    def walk(self):
        print('walking')

a = NewPigToy('piggy', 'pink', 'Middle')
print(a.name, a.size)
```

多重继承：

子类可以有多个父类，继承所有父类的方法。
案例：C 是 A和B的子类，继承A和B类的所有方法。

```
class A:
```

```
    def foo(self):
        print('foo')

class B:
    def bar(self):
        print('bar')

class C(A, B):
    pass

c1 = C()
c1.foo()
c1.bar()
```

方法查找的顺序是自下向上，自左向右

```
class A:
    def foo(self):
        print('foo')
    def hi(self):
        print('hello')

class B:
    def bar(self):
        print('bar')
    def hi(self):
        print('你好')

class C(A, B):
    def hi(self):
        print('nihao')

c1 = C()
c1.foo()
c1.bar()
c1.hi()
```

3个需要掌握的特殊的方法：

序号	方法	调用场景
1	__init__	创建实例的时候调用
2	__str__	调用print()时调用
3	__call__	实例+（）时调用

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return '<%s>' %self.title

    def __call__(self):
        print("<%s> is writed by %s" % (self.title, self.author))

if __name__ == '__main__':
    core_py = Book('Core Python', 'Wesley')  #调用__init__返回Book实例的内存地址
```

```
    print(core_py)                          #打印的时候会自动调用__str__
    core_py()                               #调用实例的时候会自动调用__call__
```

classmethon类方法 与静态方法实例

```
class Date:
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

    @classmethod
    def create_date(cls, date_str):
        year, month, day = map(int, date_str.split('-'))
        return cls(year, month, day)

    @staticmethod
    def is_date_valid(date_str):
        year, month, day = map(int, date_str.split('-'))
        return year < 4000 and 1 <= month <= 12 and 1 <= day <= 31

if __name__ == '__main__':
    d1 = Date(2019, 1, 20)
    print(d1.month)
    d2 = Date.create_date('2019-1-21')
    print(d2.year)
    print(Date.is_date_valid('2019-13-21'))
```