本节主要内容:

time模块

datetime模块

random模块

序列化-pickle模块

序列化-json模块

hashlib模块

shutil模块

logging模块

异常处理

traceback

zipfile模块

正则表达式

re模块

os模块

sys模块

模块和包

第三方模块安装和使用

模块其实就是我们把一写公用的功能代码写入在一个py文件中. 在其他模块中可以随意的使用该文件中的代码. 那么导入一个模块的语法

1. import 模块

2. from 模块 import 功能

本节我们主要是学习和了解一些python的内置模块的使用

**1. time模块**

我们一般会使用time模块让程序休息n秒. 或者计算简单的时间差

import time

print(1)

time.sleep(3) # 让程序睡3秒

print(2)

# 简单的计算时间差

a = time.time() # 当前系统时间, 从1970年1月1日0点0分0秒开始到现在经过了多少秒

time.sleep(2)

b = time.time() # 当前系统时间, 从1970年1月1日0点0分0秒开始到现在经过了多少秒

print(b - a)

**2. datetime**

在后期我们处理时间的时候会经常使用datetime来处理.

from datetime import datetime

# # 当前系统时间

# d = datetime.now()

# print(d)

#

# # 格林尼治时间, 和我们相差8小时

# d1 = datetime.utcnow()

# print(d1)

#

# # 用指定的时间创建datetime

# d2 = datetime(2015, 6, 5, 12, 11, 8, 3553)

# print(d2)

# 时间格式化

# d = datetime.now()

# s = d.strftime("%Y-%m-%d %H:%M:%S")

# print(s)

# 字符串转化成时间

str\_date = input("请输入一个时间(格式 yyyy-mm-dd HH:MM:SS):")

d = datetime.strptime(str\_date, "%Y-%m-%d %H:%M:%S")

print(d)

print(type(d))

# 计算时间差

d1 = datetime(2019, 1, 2, 11, 0, 0)

d2 = datetime(2019, 1, 1, 12, 3, 0)

diff = d2 - d1

print(diff.seconds) # 单纯从时分秒来计算

print(diff.total\_seconds()) # 包括年月日计算

日期格式化的标准:

%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

**3. random**

python中专门用来产生随机数的一个模块

import random

# 随机数(0-1)

a = random.random()

print(a)

# 随机小数(10,20)

b = random.uniform(10, 20)

print(b)

# 随机整数[50, 60]

c = random.randint(50, 60)

print(c)

lst = ["胡辣汤", "疙瘩汤", "菠菜汤", "海带汤", "大酱汤"]

# 从列表中随机选1个

one = random.choice(lst)

print(one)

# 从列表中随机选n个

some = random.sample(lst, 3)

print(some)

# 打乱一个列表

random.shuffle(lst)

print(lst)

**4. pickle模块**

pickle用起来很简单. 说白了. 就是把我们的python对象写入到文件中的一种解决方案. 但是写入到文件的是bytes. 所以这东西不是给人看的. 是给机器看的.

把python对象转化成字节的过程被称为序列化

import pickle

lst = ["周润发", "李嘉诚"]

# 把一个对象(数据)转化成字节

bs = pickle.dumps(lst)

print(bs)

# 把一个数据转化成字节, 写入到文件

pickle.dump(lst, open("lst.dat", mode="wb"))

我们还可以从文件中读取二进制字节, 转化回我们原来的数据, 该过程被称为反序列化

# 把一个文件中的二进制字节转化回我们的数据

lst = pickle.load(open("lst.dat", mode="rb"))

print(lst)

bs = b'\x80\x03]q\x00(X\t\x00\x00\x00\xe5\x91\xa8\xe6\xb6\xa6\xe5\x8f\x91q\x01X\t\x00\x00\x00\xe6\x9d\x8e\xe5\x98\x89\xe8\xaf\x9aq\x02e.'

lst = pickle.loads(bs) # 把一个文件中的二进制字节转化回我们的数据

print(lst)

**5. json模块(重要)**

终于到json了. json是我们前后端交互的枢纽. 相当于编程界的普通话. 大家沟通都用json. 为什么这样呢? 因为json的语法格式可以完美的表示出一个对象. 那什么是json: json全称javascript object notation. 翻译过来叫js对象简谱.  很复杂是吧? 来上一段我们认识的代码:

wf = {

    "name":"汪峰",

    "age":18,

    "hobby":"上头条",

    "wife":{

        "name":'子怡',

        "age":19,

        "hobby":["唱歌", "跳舞", "演戏"]

    }

}

这个不是字典么? 对的. 在python里这玩意叫字典. 但是在javascript里这东西叫json. 一模一样的.  我们发现用这样的数据结构可以完美的表示出任何对象. 并且可以完整的把对象表示出来. 只要代码格式比较好. 那可读性也是很强的. 所以大家公认用这样一种数据结构作为数据交互的格式. 那在这个鬼东西之前是什么呢? XML.....来看一段代码

<?xml version="1.0" encoding="utf-8" ?>

<wf>

    <name>汪峰</name>

    <age>18</age>

    <hobby>上头条</hobby>

    <wife>

        <name>子怡</name>

        <age>18</age>

        <hobbies>

            <hobby>唱歌</hobby>

            <hobby>跳舞</hobby>

            <hobby>演戏</hobby>

        </hobbies>

    </wife>

</wf>

古人(老程序员)都是用这样的数据进行传输的.  先不管这个东西好不好看. 这玩意想要解析.. 那简直了. 想死的心都有. 所以老版本的xml在维护和处理上是非常复杂和繁琐的.  多说一嘴, 就是因为这个鬼东西太难解析. 以前的项目几乎没有用ajax的.

OK. 那json既然这么牛B好用. 怎么用呢? 注意. 这里又出来一个新问题. 我们的程序是在python里写的. 但是前端是在JS那边来解析json的. 所以. 我们需要把我们程序产生的字典转化成json格式的json串(字符串). 然后网络传输. 那边接收到了之后. 它爱怎么处理是它的事情. 那, 如何把字典转化成我们的json格式的字符串呢?很简单, 上代码.

import json

dic = {"a": "女王", "b": "萝莉", "c": "小清新"}

s = json.dumps(dic) # 把字典转化成json字符串

print(s)  # {"a": "\u5973\u738b", "b": "\u841d\u8389", "c": "\u5c0f\u6e05\u65b0"}

结果很不友好啊. 那如何处理中文呢? 在dumps的时候给出另一个参数ensure\_ascii=False就可以了

import json

dic = {"a": "女王", "b": "萝莉", "c": "小清新"}

s = json.dumps(dic, ensure\_ascii=False) # 把字典转化成json字符串

print(s)  # {"a": "女王", "b": "萝莉", "c": "小清新"}

搞定了. 接下来. 前端给你传递信息了. 你要把前端传递过来的json字符串转化成字典.

import json

s = '{"a": "女王", "b": "萝莉", "c": "小清新"}'

dic = json.loads(s)

print(type(dic), dic)

json也可以像pickle一样把序列化的结果写入到文件中.

dic = {"a": "女王", "b": "萝莉", "c": "小清新"}

f = open("test.json", mode="w", encoding="utf-8")

json.dump(dic, f, ensure\_ascii=False) # 把对象打散成json写入到文件中

f.close()

同样也可以从文件中读取一个json

f = open("test.json", mode="r", encoding="utf-8")

dic = json.load(f)

f.close()

print(dic)

**6. hashlib模块**

MD5是一种不可逆的加密算法.  它是可靠的. 并且安全的. 在python中我们不需要手写这一套算法. 只需要引入一个叫hashlib的模块就能搞定MD5的加密工作

import hashlib

obj = hashlib.md5()

obj.update("jolin".encode("utf-8"))  # 加密的必须是字节

miwen = obj.hexdigest()

print(miwen)    # 13308dd423cebfea676b4f4aa5dc9d77

        那这样的密文安全么?  其实是不安全的. 当我们用这样的密文去找一个所谓的MD5解密工具. 是有可能解密成功的.



        这就尴尬了. MD5不是不可逆么? 注意. 这里有一个叫撞库的问题. 就是. 由于MD5的原始算法已经存在很久了. 那就有一些人用一些简单的排列组合来计算MD5. 然后当出现相同的MD5密文的时候就很容易反推出原来的数据是什么. 所以并不是MD5可逆, 而是有些别有用心的人把MD5的常见数据已经算完并保留起来了.

        那如何应对呢?  加盐就行了. 在使用MD5的时候. 给函数的参数传递一个byte即可.

import hashlib

obj = hashlib.md5(b"fjlksajflkjasfsalwer123dfskjf") # 加盐

obj.update("jolin".encode("utf-8")) # 加密的必须是字节

miwen = obj.hexdigest()

print(miwen) # 8a6564795f20e0e446049b0fba6222de

此时你再去任何网站去试. 累死他也解不开密.

那MD5如何应用呢?

import hashlib

def my\_md5(s):

    obj = hashlib.md5(b"fjlksajflkjasfsalwer123dfskjf")

    obj.update(s.encode("utf-8"))  # 加密的必须是字节

    miwen = obj.hexdigest()

    return miwen

# jolin: 8a6564795f20e0e446049b0fba6222de

username = input("请输入用户名:")

password = input("请输入密码:")

# 数据存储的时候.

# username: my\_md5(password)

# 假设现在的用户名和密码分别是

# admin: 8a6564795f20e0e446049b0fba6222de  ==> admin:jolin

# 用户登录

if username == "admin" and my\_md5(password) == "8a6564795f20e0e446049b0fba6222de":

    print("成功")

else:

    print("失败")

所以. 以后存密码就不要存明文了. 要存密文. 安全, 并且. 这里加的盐不能改来改去的.  否则, 整套密码就都乱了.

**7. shutil**

shutil主要封装了文件和文件夹的相关操作. 比如文件复制粘贴, 文件移动, 文件夹的复制, 移动等等

注意: 在mac或者linux中可以使用ls -all查看文件夹内所有文件的权限和修改时间. 在windows中使用dir命令来查看相关的内容

import shutil

fsrc = open("dir/a.txt", mode="rb")

fdst = open("dir/c.txt", mode="wb")

shutil.copyfileobj(fsrc, fdst, 1024\*10) # 拷贝文件对象

shutil.copyfile("dir/a.txt", "dir/d.txt") # 拷贝文件, 直接给出路径就可以了

shutil.copymode("dir/a.txt", "dir/e.txt")

shutil.copymode(".", "dir/e.txt") # 把访问权限拷贝

shutil.copystat(".", "dir/d.txt") # 把权限, 访问时间, 修改时间等信息拷贝

shutil.copy("dir/a.txt", "dir/f.txt") # copyfile() + copymode()

shutil.copy2("dir/a.txt", "dir/f.txt") # copyfile() + copystat()

shutil.copytree("./dir", "./baby") # 拷贝文件夹

shutil.rmtree("dir") # 删除文件夹

shutil.move(src, dst) # 移动文件, 如果dst文件目录不存在. 则重命名src文件

**8. logging**

首先, logging模块的代码不用你记. 你只需要记住怎么用就可以了(日志等级). 那如何在python中创建这个日志系统呢? 很简单.

1. 导入logging模块.

2. 简单配置一下logging

3. 出现异常的时候(except). 向日志里写错误信息.

# filename: 文件名

# format: 数据的格式化输出. 最终在日志文件中的样子

#     时间-名称-级别-模块: 错误信息

# datefmt: 时间的格式

# level: 错误的级别权重, 当错误的级别权重大于等于leval的时候才会写入文件

logging.basicConfig(filename='x1.txt',

                    format='%(asctime)s - %(name)s - %(levelname)s -%(module)s:  %(message)s',

                    datefmt='%Y-%m-%d %H:%M:%S',

                    level=0) # 当前配置表示 10以上的分数会被写入文件

logging.critical("我是critical")  # 50分. 最贵的

logging.error("我是error")    # 40分

logging.warning("我是警告")    # 警告 30

logging.info("我是基本信息")      # 20

logging.debug("我是调试")   # 10

logging.log(2, "我是自定义")     # 自定义. 看着给分

最后, 如果你系统中想要把日志文件分开. 比如. 一个大项目, 有两个子系统, 那两个子系统要分开记录日志. 方便调试. 那怎么办呢? 注意. 用上面的basicConfig是搞不定的. 我们要借助文件助手(FileHandler), 来帮我们完成日志的分开记录

import logging

# 创建一个操作日志的对象logger（依赖FileHandler）

file\_handler = logging.FileHandler('l1.log', 'a', encoding='utf-8')

file\_handler.setFormatter(logging.Formatter(fmt="%(asctime)s - %(name)s - %(levelname)s -%(module)s:  %(message)s"))

logger1 = logging.Logger('s1', level=logging.ERROR)

logger1.addHandler(file\_handler)

logger1.error('我是A系统')

# 再创建一个操作日志的对象logger（依赖FileHandler）

file\_handler2 = logging.FileHandler('l2.log', 'a', encoding='utf-8')

file\_handler2.setFormatter(logging.Formatter(fmt="%(asctime)s - %(name)s - %(levelname)s -%(module)s:  %(message)s"))

logger2 = logging.Logger('s2', level=logging.ERROR)

logger2.addHandler(file\_handler2)

logger2.error('我是B系统')

**9. 异常处理**

首先, 什么是异常? 异常就是程序运行中产生的错误. 比如以下代码就会出错

print(1/0)

Traceback (most recent call last):

File "/Users/sylar/PycharmProjects/python课程/test.py", line 1, in <module>

print(1/0)

ZeroDivisionError: division by zero

这个错误应该上过小学都能明白, 0不能作为除数. 那程序出现了这样的错误. 直接把报错信息显示出来显然不够好. 那如何处理呢?

python提供try..except语句来处理异常

语法:

try:

# 尝试执行一些代码

except 错误类型 as 变量:

# 出现该错误类型的时候, 执行这一段代码

finally:

# 不论出错还是不出错, 都会执行这一段代码

这里面的finally是可以省略的.

说一下具体的执行流程:

1. 先执行try中的代码, 如果出现了错误. 就自动执行except中的代码. 最后执行finally中的语句

2. 如果try中没有错误. 就直接执行finally中的代码

这样, 我们遇见错误的时候, 就可以用try...except来处理了.

try:

print("我是其他代码")

print(1/0)

print("我也是其他代码")

except Exception as e:

print("出错了. 请联系管理员")

finally:

print("出不出错, 你都能看见我")

注意, except可以出现多次. 表示, 出现不同的错误时, 有不同的解决方案

try:

print("我是其他代码")

print(1/0) # ZeroDivisionError

print(open('fajslfkjasjflsa.txt', mode="r").read()) # FileNotFoundError

print("我也是其他代码")

except ZeroDivisionError as e1:

print("除数不能是0")

except FileNotFoundError as e2:

print("文件不存在啊")

except Exception as e3:

print("出错了. 请联系管理员")

finally:

print("出不出错, 你都能看见我")

以上是处理异常, 但是有些时候呢, 我们没办法直接处理异常. 需要把异常上报出去, 这个动作叫抛异常. 什么关系呢? 打个比方, 处理异常就相当于 你领导让你买盒烟, 你去买, 发现领导给的钱不够. 你就自己掏了. 自己处理了. 抛异常呢, 就是你领导让你买原子弹. 你去买了. 买不到. 挨揍了. 你得回来把这个事儿上报给领导. 这叫抛异常. 抛出异常用raise

def func(a, b):

return a + b

这个函数能正常执行必须保证a和b是可以进行+操作的. 那万一是一个不能加的呢? 程序就很容易出问题. 此时我们在写函数的时候就可以进行判断, 如果a或者b不是能执行+操作的类型. 那么就抛出一个异常. 告诉调用方, 我这里是有要求的. 不是什么数据都要的.

def func(a, b): # 暂时只考虑整数

if type(a) == int and type(b) == int:

return a + b

else:

raise Exception("我要的是int") # 把错误抛出. 谁访问这个函数, 就把错误抛给谁.

func("heh", "haha")

Traceback (most recent call last):

File "/Users/sylar/PycharmProjects/python课程/test.py", line 10, in <module>

func("heh", "haha")

File "/Users/sylar/PycharmProjects/python课程/test.py", line 7, in func

raise Exception("我要的是int") # 把错误抛出. 谁访问这个函数, 就把错误抛给谁.

Exception: 我要的是int

总结: 处理异常用try...except, 抛异常用raise

**10. traceback**

在我们通过try... except捕获到异常之后. 我们发现一个比较严重的问题, 我们确实可以处理这个异常了. 但是, 这个异常, 错误信息是哪里发生的, 不知道. 如果想知道报错的位置和信息, 就需要用到traceback这个模块

import traceback

try:

print(1/0)

except:

print("出错了")

print(traceback.format\_exc()) # 错误信息就能看到了

结果:

出错了

Traceback (most recent call last):

File "/Users/sylar/PycharmProjects/python课程/test.py", line 3, in <module>

print(1/0)

ZeroDivisionError: division by zero

完整的异常处理流程:

import traceback

import logging

# 准备好记录日志的logging

logging.basicConfig(filename='x1.txt',

format='%(asctime)s - %(name)s - %(levelname)s -%(module)s: %(message)s',

datefmt='%Y-%m-%d %H:%M:%S',

level=0) # 当前配置表示 10以上的分数会被写入文件

try:

print(1/0)

except:

# 把错误信息, 错误位置, 时间 记录在日志文件中

logging.error(traceback.format\_exc())

**11. zipfile**

zipfile主要封装了和zip压缩包相关的功能. 我们就简单的讨论俩事儿. 一个事儿是如何让程序帮我们打包一个压缩包. 另一个事儿是如何用程序解压缩. 其他的功能你要是有兴趣也可以自己研究一下.

import zipfile

# 创建zip压缩文件

# f = zipfile.ZipFile("zip\_dir/abc.zip", mode="w")

# f.write("x1.txt")

# f.write("x2.txt")

# f.close()

# 解压缩

# 全部解压

# f = zipfile.ZipFile("zip\_dir/abc.zip", mode="r")

# f.extractall("zip\_dir/abc") # 全部解压缩到文件夹

# # 挨个解压

# f = zipfile.ZipFile("zip\_dir/abc.zip", mode="r")

# for name in f.namelist():

# f.extract(name, "zip\_dir/abc")

**12. 正则表达式**

首先, 我们在网页上进行注册或者登陆的时候经常能看到一些格式上的错误提示. 比如: 你在注册百度账号的时候<https://passport.baidu.com/v2/?reg&regType=1&tpl=wk> 输入用户名随意的输入系统会提示你. 你的账号过长或者不允许使用中文等等操作. 那这种操作如果使用我们现有的知识点是可以完成的. 但是完成的效果并不好. 写起来也不容易. 尤其是对邮箱的匹配. 电话号码的匹配. 那正则表达式就是专门来处理类似问题的一种表达式. 英文全称: Regular Expression. 简称 regex或者re. 但你要知道我们在使用python的re模块之前. 我们首先要对正则有一定的了解和认识. 就像我们使用time模块之前. 我们已经对时间有了一定的认识.

正则表达式是对字符串操作的一种逻辑公式. 我们一般使用正则表达式对字符串进行匹配和过滤. 使用正则的优缺点:

优点:  灵活, 功能性强, 逻辑性强.

缺点:  上手难. 一旦上手, 会爱上这个东西

工具: 各大文本编辑器一般都有正则匹配功能. 我们也可以去<https://tool.oschina.net/regex/>进行在线测试.

正则表达式由普通字符和元字符组成. 普通字符包含大小写字母, 数字. 在匹配普通字符的时候我们直接写就可以了. 比如"abc" 匹配的就是"abc". 我们如果用python也可以实现相同的效果. 所以普通字符没什么好说的. 重点在元字符上.

元字符: 元字符才是正则表达式的灵魂. 元字符中的内容太多了, 在这里我们只介绍一些常用的.  可以匹配单一文字符号的

1. 字符组

字符组很简单用[]括起来. 在[]中出现的内容会被匹配. 例如:[abc] 匹配a或b或c

如果字符组中的内容过多还可以使用- ,  例如: [a-z] 匹配a到z之间的所有字母 [0-9]匹配所有阿拉伯数字

思考: [a-zA-Z0-9]匹配的是什么?

2. 基本元字符

基本的元字符. 这个东西网上一搜一大堆. 但是常用的就那么几个:

.     匹配除换行符以外的任意字符

\w    匹配字母或数字或下划线

\s    匹配任意的空白符

\d    匹配数字

\n    匹配一个换行符

\t    匹配一个制表符

^     匹配字符串的开始

$     匹配字符串的结尾

\W    匹配非字母或数字或下划线

\D    匹配非数字

\S    匹配非空白符

a|b   匹配字符a或字符b

()    匹配括号内的表达式，也表示一个组

[...]    匹配字符组中的字符

[^...]    匹配除了字符组中字符的所有字符

3. 量词

我们到目前匹配的所有内容都是单一文字符号. 那如何一次性匹配很多个字符呢, 我们要用到量词

注意: 量词控制前面的元字符出现的次数

\*    重复零次或更多次

+    重复一次或更多次

?    重复零次或一次

{n}    重复n次

{n,}    重复n次或更多次

{n,m}    重复n到m次

4. 惰性匹配和贪婪匹配

在量词中的?,\*, +,{} 都属于贪婪匹配. 就是尽可能多的匹配到结果.

str: 玩儿吃鸡游戏, 晚上一起上游戏, 干嘛呢? 打游戏啊

reg: 玩儿.\*游戏

此时匹配的是: 玩儿吃鸡游戏, 晚上一起上游戏, 干嘛呢? 打游戏

在使用.\*后面如果加了? 则是尽可能的少匹配. 表示惰性匹配

str: 玩儿吃鸡游戏, 晚上一起上游戏, 干嘛呢? 打游戏啊

reg: 玩儿.\*?游戏

此时匹配的是: 玩儿吃鸡游戏

str: <div>胡辣汤</div>

reg: <.\*>

结果: <div>胡辣汤</div>

str: <div>胡辣汤</div>

reg: <.\*?>

结果:

    <div>

    </div>

str: <div>胡辣汤</div><span>饭团</span>

reg: <div>.\*?</div>

结果:

    <div>胡辣汤</div>

.\*?x的含义  找到最近的x.

str: abcdefgxhijklmn

reg: .\*?x

结果:abcdefgx

5. 分组

在正则中使用()进行分组. 比如. 我们要匹配一个相对复杂的身份证号. 身份证号分成两种. 老的身份证号有15位. 新的身份证号有18位. 并且新的身份证号结尾有可能是x.

身份证号 15位, 18位

^[1-9]\d{13,16}[0-9x]$

^[1-9]\d{14}(\d{2}[0-9x])?$

^([1-9]\d{16}[0-9x]|[1-9]\d{14})$

6. 转义

在正则表达式中, 有很多有特殊意义的是元字符, 比如\n和\s等,如果要在正则中匹配正常的"\n"而不是"换行符"就需要对"\"进行转义, 变成'\\'.在python中, 无论是正则表达式, 还是待匹配的内容, 都是以字符串的形式出现的, 在字符串中\也有特殊的含义, 本身还需要转义. 所以如果匹配一次"\n", 字符串中要写成'\\n', 那么正则里就要写成"\\\\n",这样就太麻烦了. 这个时候我们就用到了r' '这个概念, 此时的正则是r'\\n'就可以了. 记住, 在python中写正则的时候前面带着r

**13. re模块**

re模块是python提供的一套关于处理正则表达式的模块. 核心功能有四个:

1. findall 查找所有. 返回list

lst = re.findall("m", "mai le fo len, mai ni mei!")

print(lst)     # ['m', 'm', 'm']

lst = re.findall(r"\d+", "5点之前. 你要给我5000万")

print(lst)     # ['5', '5000']

2. search 会进行匹配. 但是如果匹配到了第一个结果. 就会返回这个结果. 如果匹配不上search返回的则是None

ret = re.search(r'\d', '5点之前. 你要给我5000万').group()

print(ret) # 5

3. match  只能从字符串的开头进行匹配

ret = re.match('a', 'abc').group()

print(ret)     # a

4. finditer 和findall差不多. 只不过这时返回的是迭代器

it = re.finditer("m", "mai le fo len, mai ni mei!")

for el in it:

    print(el.group()) # 依然需要分组

5. 其他操作

ret = re.split('[ab]', 'qwerafjbcd')  # 先按'a'分割得到'qwer'和'fjbcd',在对'qwer'和'fjbcd'分别按'b'分割

print(ret)  # ['qwer', 'fj', 'cd']

ret = re.sub(r"\d+", "\_sb\_", "jolin250tory250tony250kevin38") # 把字符串中的数字换成\_\_sb\_\_

print(ret) # jolin\_sb\_tory\_sb\_tony\_sb\_kevin\_sb\_

ret = re.subn(r"\d+", "\_sb\_", "jolin250tory250tony250kevin38") # 将数字替换成'\_\_sb\_\_'，返回元组(替换的结果,替换了多少次)

print(ret) # ('jolin\_sb\_tory\_sb\_tony\_sb\_kevin\_sb\_', 4)

obj = re.compile(r'\d{3}')  # 将正则表达式编译成为一个 正则表达式对象, 规则要匹配的是3个数字

ret = obj.search('abc123eeee') # 正则表达式对象调用search, 参数为待匹配的字符串

print(ret.group())  # 结果: 123

爬虫重点:

obj = [re.compile](http://re.compile)(r'(?P<id>\d+)(?P<name>e+)')  # 从正则表达式匹配的内容每个组起名字

ret = [obj.search](http://re.compile)('abc123eeee') # 搜索

print([ret.group(](http://re.compile)))  # 结果: 123eeee

print([ret.group(](http://re.compile)"id"))  # 结果: 123 # 获取id组的内容

print([ret.group(](http://re.compile)"name")) # 结果: eeee # 获取name组的内容

正则在爬虫中的使用(简单案例):

import re

from urllib.request import Request, urlopen

import ssl

# 干掉数字签名证书

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

def getPage(url):

r = Request(url, headers={"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_14\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"})

response = urlopen(r)

return response.read().decode('utf-8')

def parsePage(s):

    ret = re.findall(

        '<div class="item">.\*?<div class="pic">.\*?<em .\*?>(?P<id>\d+).\*?<span class="title">(?P<title>.\*?)</span>'

       '.\*?<span class="rating\_num" .\*?>(?P<rating\_num>.\*?)</span>.\*?<span>(?P<comment\_num>.\*?)评价</span>', s, re.S)

    return ret

def main(num):

    url = 'https://movie.douban.com/top250?start=%s&filter=' % num

    response\_html = getPage(url)

    ret = parsePage(response\_html)

    print(ret)

count = 0

for i in range(10):   # 10页

    main(count)

    count += 25

此时利用的就是分组之后. 匹配成功后获取到的是分组后的结果. (?P<id>\d+) 此时当前组所匹配的数据就会被分组到id组内. 此时程序可以改写成:

import ssl

import re

from urllib.request import Request, urlopen

# 干掉数字签名证书

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

def getPage(url):

r = Request(url, headers={"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_14\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"})

response = urlopen(r)

return response.read().decode('utf-8')

def parsePage(s):

com = re.compile(

ret = com.finditer(s)

for i in ret:

yield {

"id": i.group("id"),

"title": i.group("title"),

"rating\_num": i.group("rating\_num"),

"comment\_num": i.group("comment\_num"),

}

def main(num):

url = 'https://movie.douban.com/top250?start=%s&filter=' % num

response\_html = getPage(url)

ret = parsePage(response\_html)

# print(ret)

f = open("move\_info7", "a", encoding="utf8")

for obj in ret:

print(obj)

data = str(obj)

f.write(data + "\n")

count = 0

for i in range(10):

main(count)

count += 25

    正则表达式和re模块就说这么多. 如果要把正则所有的内容全部讲清楚讲明白, 至少要一周以上的时间. 对于我们日常使用而言. 上述知识点已经够用了. 如果碰到一些极端情况建议想办法分部处理. 先对字符串进行拆分. 然后再考虑用正则.

**14. os模块**

os模块主要封装了关于操作系统文件系统的相关操作. 比如. 创建文件夹, 删除文件夹等

所有和操作系统相关的内容都在os模块

os.makedirs('dirname1/dirname2')    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；相当于shell中rmdir dirname

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","newname")  重命名文件/目录

os.system("bash command")  运行shell命令，直接显示

os.popen("bash command").read()  运行shell命令，获取执行结果

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.path

os.path.abspath(path) 返回path规范化的绝对路径

os.path.split(path) 将path分割成目录和文件名二元组返回

os.path.dirname(path) 返回path的目录。其实就是os.path.split(path)的第一个元素

os.path.basename(path) 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os.path.split(path)的第二个元素

os.path.exists(path)  如果path存在，返回True；如果path不存在，返回False

os.path.isabs(path)  如果path是绝对路径，返回True

os.path.isfile(path)  如果path是一个存在的文件，返回True。否则返回False

os.path.isdir(path)  如果path是一个存在的目录，则返回True。否则返回False

os.path.join(path1[, path2[, ...]])  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

os.path.getatime(path)  返回path所指向的文件或者目录的最后访问时间

os.path.getmtime(path)  返回path所指向的文件或者目录的最后修改时间

os.path.getsize(path) 返回path的大小

# 特殊属性:

os.sep    输出操作系统特定的路径分隔符，win下为"\\",Linux下为"/"

os.linesep    输出当前平台使用的行终止符，win下为"\r\n",Linux下为"\n"

os.pathsep    输出用于分割文件路径的字符串 win下为;,Linux下为:

os.name    输出字符串指示当前使用平台。win->'nt'; Linux->'posix'

用os.walk遍历文件夹

import os

for root, dirs, fils in os.walk("."):

for dir in dirs:

print(os.path.join(root, dir))

for f in fils:

print(os.path.join(root, f))

用递归遍历文件夹

def func(path):

lst = os.listdir(path)

for f\_name in lst:

new\_path = os.path.join(path, f\_name)

if os.path.isdir(new\_path):

func(new\_path)

else:

print(new\_path)

func("./")

**15. sys模块**

    所有和python解释器相关的都在sys模块.

sys.argv           命令行参数List，第一个元素是程序本身路径

sys.exit(n)        退出程序，正常退出时exit(0),错误退出sys.exit(1)

sys.version        获取Python解释程序的版本信息

sys.path           返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值

sys.platform       返回操作系统平台名称

**16. 模块和包**

本质上模块和包一样. 都是把一些特定功能的代码封装在一起. 方便我们访问和使用.

语法:

import 模块

from xxx import xxx

导入模块的时候都做了些什么?

首先. 在导入模块的一瞬间. python解释器会先通过sys.modules来判断该模块是否已经导入了该模块. 如果已经导入了则不再导入. 如果该模块还未导入过. 则系统会做三件事.

1. 为导入的模块创立新的名称空间(独立的内存)

2. 在新创建的名称空间中运行该模块中的代码

3. 创建模块的名字. 并使用该名称作为该模块在当前模块中引用的名字.

a模块:

print("我爱你")

def func():

print("我是func")

name = "jolin"

b模块

import a

print(a.name)

a.func()

此时, 我们运行b模块. 会发现a模块中的打印语句率先出来. 所以. 它是先把被导入的模块执行了. 然后再执行自己模块的内容

由于模块在导入的时候会创建其自己的名称空间. 所以. 我们在使用模块中的变量的时候一般是不会产生冲突的.

特别特别要注意.  如果我们在不同的模块中引入了同一个模块. 并且在某一个模块中改变了被引入模块中的全局变量. 则其他模块看到的值也跟着变. 原因是python的模块只会引入一次. 大家共享同一个名称空间

a模块

name = "join"

b模块

import a

a.name = "kevin"

c模块

import b

import a

print(a.name)

此时运行c模块. 看到的是kevin.

关于\_\_main\_\_

我们刚才讲了. 如果一个模块被其他模块导入了. 首先会把该模块执行了. 但是, 我们在编写一些测试性的代码的时候, 经常的会在自己模块里运行一次看看效果. 比如:

def eat():

print("我特别喜欢吃骨头")

# 测试一下, 看看好用不

eat()

但是上述代码如果被其他模块使用, 导入的时候. 会自动的把这个模块执行了.

import a # 这句话会自动执行a模块. 但是a模块里有些代码是测试用的.

a.eat() # 这才是我要执行的代码

怎么办. 这时, 我们一个模块就会有两种执行方式, 一种是被别人导入了. 会执行, 还有一种是右键run执行. 那如何区分呢? 在每个模块里. 都有一个默认的属性叫\_\_name\_\_. 这个\_\_name\_\_比较特殊, 如果该模块是被别人导入的. \_\_name\_\_就是这个模块名, 而如果该模块是被run执行的就是\_\_main\_\_. 所以, 我们可以通过\_\_name\_\_ == "\_\_main\_\_"来区分该模块是被导入的还是被run执行的. 像我们写的那种测试性的代码. 肯定是想在run的时候执行. 所以. 我们通常会把测试性的代码写在\_\_main\_\_里, 更改a模块如下:

def eat():

print("我特别喜欢吃骨头")

# 此时, 只有该模块被右键run执行的时候. 才会执行下面的代码

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

eat()

关于from xxx import xxx 它的加载过程和import是一样的. 区别就是他会把模块中的某些个内容单独导入到当前模块. 和import的区别是: import导入的是一个整体. from 导入的是一部分.

import json # 导入整个json

from json import dumps, loads # 从json中导入dumps和loads

正确的导入模块的顺序:

1. 所有的模块导入都要写在最上面. 这是最基本的

2. 先引入内置模块

3. 再引入扩展模块

4. 最后引入你自己定义的模块

然后我们来看看包

包的含义其实很简单, 就是文件夹. 我们一个py文件可能写不了所有的功能模块. 怎么办. 多写几个py文件. 然后封装在一个文件夹里. 这个文件夹就是包. 使用起来呢, 和我们用模块差不多. 不过有一些小小的坑. 在写代码的时候尽量避免就好了

首先, 我们创建包:

import os

os.makedirs('glance/api')

os.makedirs('glance/cmd')

os.makedirs('glance/db')

l = []

l.append(open('glance/\_\_init\_\_.py','w'))

l.append(open('glance/api/\_\_init\_\_.py','w'))

l.append(open('glance/api/policy.py','w'))

l.append(open('glance/api/versions.py','w'))

l.append(open('glance/cmd/\_\_init\_\_.py','w'))

l.append(open('glance/cmd/manage.py','w'))

l.append(open('glance/db/\_\_init\_\_.py','w'))

l.append(open('glance/db/models.py','w'))

map(lambda f:f.close() ,l)

各个包下的代码:

#policy.py

def get():

    print('from policy.py')

#versions.py

def create\_resource(conf):

    print('from version.py: ',conf)

#manage.py

def main():

    print('from manage.py')

#models.py

def register\_models(engine):

    print('from models.py: ',engine)

接下来. 我们在test中使用包中的内容. 并且, 我们导入包的时候可以使用import或者from xxx import xxx这种形式.

首先, 我们看import

import glance.db.models

glance.db.models.register\_models('mysql')

没问题,  很简单, 我们还可以使用from xxx import xxx 来导入包内的模块

from glance.api.policy import get

get()

也很简单, 但是, 要注意. from xxx import xxx这种形式, import后面不可以出现"点" 也就是说from a.b import c是ok的. 但是 from a import b.c 是错误的

好了, 到目前为止, 简单的包已经可以使用了. 那包里的\_\_init\_\_.py是什么鬼? 其实. 不论我们使用哪种方式导入一个包, 只要是第一次导入包或者是包的任何其他部分, 都会先执行\_\_init\_\_.py文件. 这个文件可以是空的. 但也可以存放一些初始化的代码. (随意在glance中的\_\_init\_\_.py都可以进行测试)

接下来, 我们来看一下绝对导入和相对导入, 我们的最顶级包glance是写给别人用的. 然后再glance包内部也会有彼此之间互相导入的需求, 这时候就又绝对导入和相对导入两种方式了.

        1. 绝对导入: 以glance作为起始

        2. 相对导入: 用. 或者..作为起始

例如, 我们在glance/api/version.py中使用glance/cmd/manage.py

# 在glance/api/version.py

#绝对导入

from glance.cmd import manage

manage.main()

#相对导入

# 这种情形不可以在versions中启动程序.

#  attempted relative import beyond top-level package

from ..cmd import manage

manage.main()

测试的时候要注意. python包路径跟运行脚本所在的目录有关系. 说白了. 就是你运行的py文件所在的目录.  在python中不允许你运行的程序导包的时候超过当前包的范围(相对导入). 如果使用绝对导入. 没有这个问题. 换个说法. 如果你在包内使用了相对导入. 那在使用该包内信息的时候. 只能在包外面导入.

接下来. 我们来看一个大坑. 比如. 我们想在policy中使用verson中的内容.

# 在policy.py

import versions

如果我们程序的入口是policy.py 那此时程序是没有任何问题的. 但是如果我们在glance外面import了glance中的policy就会报错 原因是如果在外面访问policy的时候. sys.path中的路径就是外面. 所以根本就不能直接找到versions模块. 所以一定会报错:

ModuleNotFoundError: No module named 'versions'

在导包出错的时候. 一定要先看sys.path  看一下是否真的能获取到包的信息.

最后, 我们看一下如何单独导入一个包.

# 在test.py中

import glance

此时导入的glance什么都做不了. 因为在glance中的\_\_init\_\_.py中并没有关于子包的加载. 此时我们需要在\_\_init\_\_.py中分别取引入子包中的内容.

我们自己创建的py文件的名字不要和系统内置的模块重名

**17. 第三方模块安装**

pip install 模块 -i 国内源

注意, 如果pip在控制台输入没有反应, 很可能是环境变量没有配置成功. 需要把python中的Script文件夹也配置到path环境变量中才可以.

有些情况我们安装某些模块的时候可能需要对pip进行升级

pip install -U pip # 更新pip

pip uninstall 模块 # 卸载模块

pip show 模块 # 显示xxx模块

pip list # 显示出已安装的模块

清华大学help网站--> 不会了点这里 : <https://mirrors.tuna.tsinghua.edu.cn/help/pypi/>

清华大学镜像源: pip install -i https://pypi.tuna.tsinghua.edu.cn/simple some-package

**练习:**

1、计算两个格式化时间之间差了多少年月日时分秒

2、生成一个6位随机验证码(包含数字和大小写字母)

3、编写红包, 制定金额和个数随机分配红包金额

4、分别列出给定目录下所有的文件和文件夹

5、获取当前文件所在目录 os.path.dirname()

6、在当前目录下创建一个文件夹、在这个文件夹下创建一个文件

7、计算某路径下所有文件和文件夹的总大小

8、写正则匹配一下内容:

1. 匹配邮箱

2. 匹配手机号

3. 匹配生日. 日期格式(yyyy-MM-dd)

**作业:**

1. 编写爬虫, 抓取dytt(<https://www.dytt8.net/>)中的指定位置电影信息, 并将抓取的电影名称, 电影下载链接存储在movie.json文件中.



2. 安装第三方模块requests+bs4. 并尝试抓取<优美图库>里面的小姐姐的漂亮照片.

import requests # 自己安装这个模块

# 这个网址直接丢在浏览器里可能不好使. 请在'www.umei.cc'首页找'美女写真'栏目

resp = requests.get("http://www.umei.cc/meinvtupian/meinvxiezhen/")

resp.encoding = 'utf-8'

print(resp.text)

3. 尝试使用第三方模块requests+bs4. 抓取<北京新发地>菜价.

import requests

resp = requests.get("http://www.xinfadi.com.cn/marketanalysis/0/list/1.shtml")

resp.encoding = 'utf-8'

print(resp.text)