## day25-01(58)（常用模块）

## day25-02(59)(内容回顾)

## day25-03(60)(正则表达式的概念)

什么是正则表达式

一种匹配字符串的规则

正则表达式能做什么

可以制定一个规则，来确定某一个字符串是否符合规则

帮助学习的工具: http://tool.oschina.net/regex/

## day25-04(61)(正则表达式的基本语法)

### 元字符

#### 字符组

字符组[ ]在某一个字符的位置上能出现的内容

[abc]表示在某一个字符的位置上能出现的内容是a或b或c

匹配所有的数字【0-9】（注意匹配范围只能正着写，~~不能倒着写【9-4】~~）

匹配所有的小写字母【a-z】

匹配所有的大写字符【A-Z】

[0-9][A-Z][a-z]这是三个字符

既能匹配数字又能匹配字符[0-9a-zA-Z]

【0-9abc】这也是一个字符

匹配18位身份证号

【1-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9】【0-9Xx】

#### 非字符组

元字符:用一个字符串来匹配内容

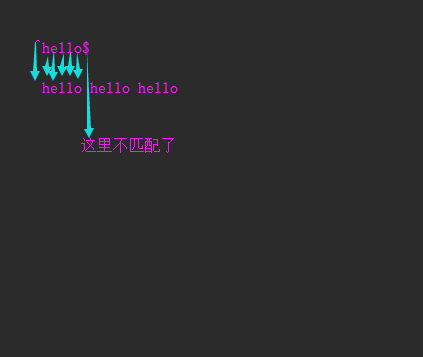
|  |  |  |
| --- | --- | --- |
| . | 匹配除换行符以外的任意字符 | 如果匹配2.3呢  将.转义\.  \d\.\d |
| \w=[0-9a-zA-Z\_] | 匹配字母或数字或下划线 | Word |
| \s==【\n \t】 回车空格制表符 | 匹配任意的空白符 | space |
| \d=[0-9] | 匹配数字 | digit |
| \n | 匹配一个换行符 |  |
| \t | 匹配一个制表符 |  |
| \b | 匹配一个单词的边界（边界部分前后） | 字符\b表示字符后面的是边界  \b字符 表示字符前面的是边界 |
| ^ | 匹配字符串的开始 | 只从开头匹配 |
| $ | 匹配字符串的结尾 | 只匹配结尾的 |
| \D | 匹配非数字 |  |
| \S | 匹配非空白符 |  |
| a|b | 匹配字符a或字符b | 注意如果a和b有重合的地方，把长的一个写下前面（原理是一旦前面的满足就不会匹配后面的）比如abc|ab |
| () | 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 |  |
| \W | 匹配非字母或数字或下划线 |  |
|  |  |  |
|  |  |  |

\n 匹配回车

\t 匹配制表符

【\d\D】一对大小写匹配所有的字符串

^hello$和hello hello hello的匹配结果（这样写的话，不仅要求内容一样还要求长度一样）



规则：从头开始去匹配一个字符串，能匹配到多个符合规则的就是多条结果，带着^号的只匹配到一个结果

### 量词

### 特殊的用法和现象

## day25-06(量词)

量词:约束前面的一个表达式

|  |  |
| --- | --- |
| 量词 | 用法说明 |
| \* | 重复零次或更多次 |
| + | 重复一次或更多次 |
| ？ | 重复零次或一次(匹配上0次也算死匹配上了) |
| {n} | 重复n次 |
| {n,} | 重复n次或更多次 |
| {n,m} | 重复n到m次(能尽量匹配m次就匹配m次) |
|  |  |



### 练习

|  |  |  |  |
| --- | --- | --- | --- |
| 正则 | 待匹配字符 | 匹配结果 | 说明 |
| 海. | 海燕海姣海东 | 海燕海姣海东 | 匹配所有海的字符 |
| ^海. | 海燕海姣海东 | 海燕 | 只从开头匹配海 |
| 海.$ | 海燕海姣海东 | 海东 | 只匹配结尾的海 |

|  |  |  |  |
| --- | --- | --- | --- |
| + \* ? {}（这4个量词都是贪婪匹配） | | | |
| 正则 | 待匹配字符 | 匹配结果 | 说明 |
| 李.? | 李杰和李莲英和李二棍子李 | 李杰  李莲  李二  李 |  |
| 李.\* | 李杰和李莲英和李二棍子李 | 李杰和李莲英和李二棍子李 | 这里注意值匹配一个 |
| 李.+ | 李杰和李莲英和李二棍子李 | 李杰和李莲英和李二棍子李 |  |
| 李.{1,3} | 李杰和李莲英和李二棍子李 | 李杰和李  李二棍子 |  |

### 特殊用法和现象

#### ？的使用

##### 在量词后面跟一个?取消贪婪匹配（就是尽可能少的匹配）

？？

\*？

+？

{n}?

|  |  |  |  |
| --- | --- | --- | --- |
| 李.{1,3}? | 李莲英和 | 李莲 |  |
| 李.{1,3}?和 | 李莲英和 | 李莲英和 | 以给的表达式为基础在匹配的上的情况下尽可能匹配少的次数（正则表达式中的.至少要出现两次才能匹配到结果（即在能匹配的上的情况上匹配两个）） |

##### 贪婪匹配



贪婪匹配算法：正则表达式的开头对应字符串的开头，正则表达式的结尾是从字符串后面往前面找（回溯算法）



|  |  |  |  |
| --- | --- | --- | --- |
| 正则 | 待匹配字符 | 匹配结果 | 说明 |
| 李[杰莲英二棍子]\* | 李杰和李莲英和李二棍子李 | 李杰  李莲英  李二棍子  李 | [杰莲英二棍子]这个字符组出现0或那次 |
| 李[^和]\* | 李杰和李莲英和李二棍子李 | 李杰  李莲英  李二棍子李 | 匹配一个不是和的字符任意次 |
|  |  |  |  |

### 组合

匹配18位身份证号

^(\d{6})(\d{4})(\d{2})(\d{2})(\d{3})([0-9]|X)$

匹配所有小数

\d\*(\.\d\*)? 将.和小树部分全部都扩起来作用是让它们一起出现

## day26-01(内容回顾)

正则表达式的应用：登陆表单验证，爬虫，自动化开发，日子分析

### 作业

#### 一

1. 匹配任意长度的正整数：[1-9]\d\*
2. 匹配小数 -?\d+\.\d+
3. 匹配整数或者小数 -?\d+(\.\d+)?（标准答案） \d+\.?d\*(有缺陷1. 2. 这样的内容都能被匹配上) \d+\.?\d+ （有缺陷 1，2 这样的一位数都匹配不上了）
4. 匹配负数 -0\.\d+|-[1-9]+(\.\d+)?
5. 匹配qq [1-9]\d{4,11}
6. 匹配长度为11位的电话号码 1[3-9]\d{9}
7. 长度为8-10位iede用户密码:包含数字字母下划线和？ 【\w@？】{8,10}
8. 匹配验证码：4位数字字母组成 \w{4}

#### 二

<a>wahaha<\a>

<b>banana<\b>

<h1>qqxing<\h1>

<script>salkdjgh</script>

匹配出wahaha,banana,qqxing内容（即匹配出标签内的内容） >\w+<

匹配出a,b,h1这样的内容（匹配出标签） <\w+>）

#### 三

(-40/5)(9-2\*5/3+7/3\*99/4\*298+10\*568/14)(-4\*3)(16-3\*2)匹配出最内层的小括号

正则表达式：\([\d\.+\-\*/]+\)

(-40/5)

(9-2\*5/3+7/3\*99/4\*298+10\*568/14)

(-4\*3)

(16-3\*2)

## day26-02

re模块

常用方法

Python中正则表达式的特点

代码的示例

## day26-03

### 在python中使用正则表达式

#### 转义符

##### 在python中使用转义符

“\n”\是转义符，赋予了n一个特殊的意义,表示一个换行符

如何打印\n+

1.print("\\n")

2.print(r"\n")

##### 实战

正则表达式匹配\n,所有的表达式在py中都是字符串的形式，因为“\n”表示一个换行符，所以要匹配\n就要写成[\\n](file:///\\n)，所以现在要做的就是写个正则表达式来匹配字符串“\\n”,

因为\是转义字符，所以正则表达式中我们要用\\来替换字符串中的\,所以在python中正则表达式

[\\\\n匹配字符串\\n](file:///\\\\n匹配字符串\\n),\\n是\n在python中的字符串表达形式

简单方法

###### Python中需要转义的

r’\\n’来匹配r’\n’

###### 正则表达式需要转义的

‘\(’表示匹配小括号

【（）+-\*/$】在字符组中一些特殊的字符会显出原形

所有的\w \d \s(\n,\t,) \W \D \S都表示它原本的意义

【-】只有写在字符组的首位的时候表示普通的减号，写在其他位置表示范围，如果你就是想匹配减号[1\-9]这样写

## day26-04(re模块的 基本用法)

#### Re模块(操作匹配字符串的方法)

###### 字符串匹配

findall

import re  
#第一个参数pattern表示正则表达式，string表示待匹配字符串  
ret=re.findall('\d',"111882211")  
print(ret)#参数 返回值类型:列表 返回值个数:1 返回值内容:所有匹配上的项

search

ret2=re.search("\d+","111efsa 22")  
print(ret2)#<re.Match object; span=(0, 3), match='111'>  
print(type(ret2))#返回值类型:正则匹配结果的对象 返回值个数:1 如果匹配上了就返回对象  
print(ret2.group())#返回的对象通过group来匹配到第一个结果（只能取第一个）  
ret3=re.search("\s","sdaf111")  
print(ret3)#没有匹配上就返回为none

match(和search的区别是：同一个正则表达式，只要在search中的增则表达式之前加^就和match的效果是一样的)

ret4=re.search("\d+","%1996wdd0102")#不需要从头开始  
print(ret4)  
ret5=re.match("\d+","1996wdd0102")#match是从头开始匹配只要在search中的正则表达式前面加一个^效果就和match一样  
print(ret5)  
print(ret5.group())

###### 替换

sub

ret6="1996wdd0206wdd1314wdd1314".replace("1314","520")  
print(ret6)#print(ret6)替换之前需要知道替换的内容  
ret7=re.sub("\d+","520","1996wdd0206wdd1314wdd1314")  
print(ret7)#520wdd520wdd520wdd520  
ret8="1996wdd0206wdd1314wdd1314".replace("1314","520",1)  
print(ret8)#替换之前需要知道替换的内容  
ret9=re.sub("\d+","520","1996wdd0206wdd1314wdd1314",2)  
print(ret9)# 520wdd520wdd1314wdd1314

subn(返回的元组 返回最终结果和替换个数)

ret6="1996wdd0206wdd1314wdd1314".replace("1314","520")  
print(ret6)#print(ret6)替换之前需要知道替换的内容  
ret7=re.subn("\d+","520","1996wdd0206wdd1314wdd1314")  
print(ret7)#('520wdd520wdd520wdd520', 4)  
ret8="1996wdd0206wdd1314wdd1314".replace("1314","520",1)  
print(ret8)#print(ret6)替换之前需要知道替换的内容  
ret9=re.subn("\d+","520","1996wdd0206wdd1314wdd1314",2)  
print(ret9)#('520wdd520wdd1314wdd1314', 2)

###### 切割

split

ret=re.split("\d+","alex80asdf50taibai90")  
print(ret)#['alex', 'asdf', 'taibai', '']

###### 进阶方法—爬虫

compile（将正则表达式提前编译好，编译好的结果直接调用匹配方法）

r=re.compile("\d+")  
rs=r.match("1980wdd06620")  
print(rs.group())#1980

finditer

r=re.finditer("\d+","da111af452faf 187af af1222")  
print(r)#<callable\_iterator object at 0x00000049A0476B00>  
for e in r:  
 print(e.group())

## day26-06（python中的正则的分组）

### python中的分组

#### findall的分组优先显示

正则表达式[www.(baidu|oldboy).com](http://www.(baidu|oldboy).com)匹配[www.baidu.com](http://www.baidu.com) www.oldboy.com

findall会优先显示分组中的内容(match和search不会出现这个问题)

ret2=re.findall("www.baidu.com|www.oldboy.com","www.baidu.com")  
print(ret2)#['www.baidu.com']  
ret3=re.findall("www.(baidu|oldboy).com","www.baidu.com")  
print(ret3)#baidu 这里匹配了[www.baidu.com](http://www.baidu.com)的内容，只不过只显示了baidu

分组优先的坑（取消分组优先）

ret3=re.findall("www.(?:baidu|oldboy).com","www.baidu.com")  
print(ret3)# ['www.baidu.com']

解释

ret=re.findall("-0\.\d+|[1-9]\d\*(\.\d+)?","-1adsfas-200")  
print(ret)#['', ''] 这里是（）里的优先匹配因为括号里的没有匹配的上所以是空但是照样匹配了两个（带了（）只显示括号里的）

只需要在（）里的开头加上？:

#### split遇到分组保留分组内被切掉的内容

ret4=re.split("\d+","wdd26wn22we23")  
print(ret4)#['wdd', 'wn', 'we', '']  
ret5=re.split("(\d+)","wdd26wn22we23")  
print(ret5)#['wdd', '26', 'wn', '22', 'we', '23', '']

#### 分组遇见search

ret=re.search("\d+(\.\d+)(\.\d+)(\.\d+)(\.\d+)?","1.2.3.4")  
print(ret.group())#1.2.3.4  
print(ret.group(0))#1.2.3.4  
print(ret.group(1))#.2  
print(ret.group(2))#.3  
print(ret.group(3))#.4  
print(ret.group(4))#None

总结:

findall为什么会有分组优先，search返回的是一个对象，这个对象有他的方法，使用.group（index）想拿到那个分组就拿到那个分组，findall返回的是一个列表，列表里面直接是值，这样我没办法选择拿不拿分组的分组的内容，这样的话就有一种机制，你选择是拿分组里的内容还是显示左右的字符串

## day26-07（分组练习）

import re  
#取出整数  
ret=re.findall("\d+","1-2\*(60+(-40.35/5)-(-4\*3))")  
print(ret)#['1', '2', '60', '40', '35', '5', '4', '3']  
ret1=re.findall("\d+(?:\.\d+)|(\d+)","1-2\*(60+(-40.35/5)-(-4\*3))")  
print(ret1)#['1', '2', '60', '', '5', '4', '3']

正则表达式是\d+(\.\d+)|\d+(即匹配所有的整数和小数)

findall在遇到分组时会优先显示，我们要取消优先显示\d+(?:\.\d+)|\d+

我们需要让整数优先显示处理啊就加（） 变成\d+(?:\.\d+)|(\d+)

匹配规则是，小数匹配到|左边就不会匹配|右边，只有整数才会匹配|右边

因为要优先显示，且优先显示只显示括号里的最后结果是所有数都匹配上了但是优先显示整数空字符串就是小数

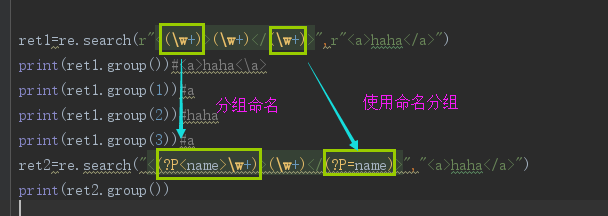
### 正则匹配标签

re1=re.findall(">(\w+)<","<a>haha</a>")  
print(re1)#['haha']  
re2=re.findall("<(\w+)>","<a>haha</a>")  
print(re2)#['a']  
  
ret1=re.search(r"<(\w+)>(\w+)</(\w+)>",r"<a>haha</a>")  
print(ret1.group())#<a>haha<\a>  
print(ret1.group(1))#a  
print(ret1.group(2))#haha  
print(ret1.group(3))#a

### 分组命名

（？P<name>正则表达式） 表示给分组起名字

（？P=name）表示使用这个分组，这里匹配到的内容应该和分组中的内容完全相同

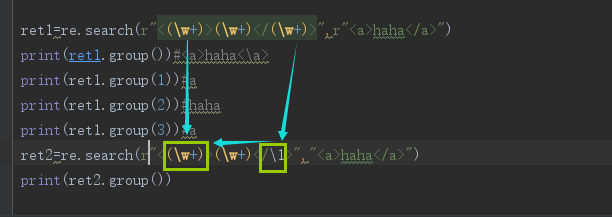


#### 好处（根据命名取分组）

ret2=re.search(r"<(?P<name>\w+)>(?P<body>\w+)</(?P=name)>","<a>haha</a>")  
print(ret2.group("name"))  
print(ret2.group("body"))

### 如果不使用分组命名，也可以用\序号来找对应的分组，表示要找的内容和前面的内容一致

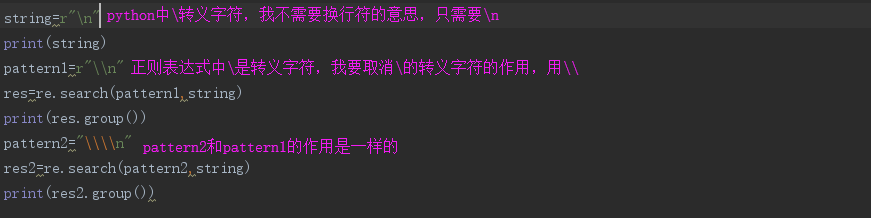
\1表示使用第一组，匹配到的内容必须和第一组中的内容完全相同（推荐使用分组命名法，要不然下次看代码的时候不知道是哪一个分组）



## day26-08(爬虫练习)（没怎么学）

## day26-09(作业)

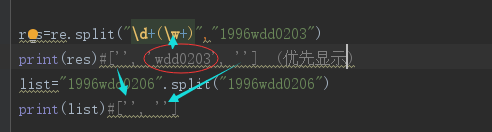
## day26-10(答疑)



r是取消字符中所有\的转义作用

### \的小技巧

在测试的工具中，如果带了\,你又担心它会出现转义的情况，不管三七二十一，在测试工具里测试好了的代码拿到python中，统一在字符串前加上r就行了



## day26-11(random)

### 随机小数

#0-1之内的随机小数  
print(random.random())  
#1-5之内的随机小数(给定任意范围)  
print(random.uniform(1,5))

### 随机整数

print(random.randint(1,2))#[1,2]  
print(random.randrange(1,2))#[1,2)  
print(random.randrange(1,10,2))#[1,10)之内随机取奇数

### 随机抽取(抽取后就没有了)

#随机抽取一个值  
l=[11,22,33,"aa",("aa","bb")]  
ret=random.choice(l)  
print(ret)  
  
#随机抽取多个值  
ret=random.sample(l,2)  
print(ret)

### 打乱顺序

#在原来列表的基础上做乱序  
random.shuffle(l)  
print(l)

### 抽奖|彩票|发红包|验证码|

## day26-12(关于模块和文件名之间的误会)

永远不要再一个文件下写你认识的模块名的名字（模块导入的话会取最近原则）

## day26-13(random模块的练习题)

随机6位字母数字（用chr(97)）

## day27-01(复习)

## day27-02(常用模块)

collections模块（不常用了）

时间模块

os模块

sys

序列化（3个模块）

## day27-03(时间模块)

### 时间戳时间（float数据类型）

英国伦敦的时间1970.1.1 0：0：0

北京时间 1970.1.1 8：0：0

1548761057.6669476

print(time.time())#1548760668.429915

### 结构化时间，时间对象

print(time.localtime())#time.struct\_time(tm\_year=2019, tm\_mon=1, tm\_mday=29, tm\_hour=19, tm\_min=20,   
# tm\_sec=49, tm\_wday=1, tm\_yday=29, tm\_isdst=0)  
time\_obj=time.localtime()#时间对象  
print(time\_obj.tm\_hour)

时间对象能够通过.属性名来获取对象中的值

## 格式化时间（str数据类型，字符串时间）

可以根据你需要的格式来显示时间

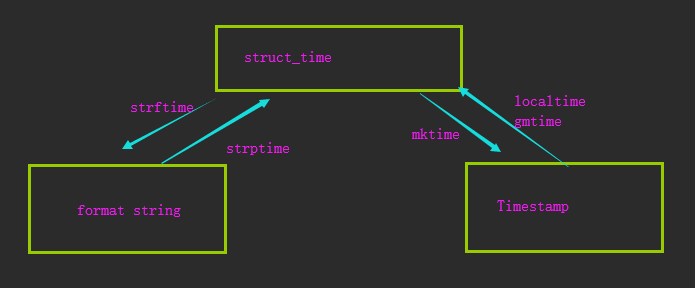
print(time.strftime('%Y-%m-%d'))#2019-01-29

print(time.strftime('%Y-%m-%d %H:%M:%S'))# 2019-01-29 19:39:29

格式化参数

|  |  |
| --- | --- |
| %Y | 四位数年份表示（00-99） |
| %y | 两位数年份表示（000-9999） |
| %m | 月份（01-12） |
| %d | 月内中的一天（0-31） |
| %H | 24小时小时制 |
| %I | 12小时小时制 |
| %M | 分钟数 |
| %S | 秒 |
| %a | 本地简化星期名称 |
| %A | 本地完整星期名称 |
| %b | 本地简化月份名称 |
| %B | 本地完整月份名称 |
| %c | 本地相应的日期表示和时间表示 |
| %j | 年内的一天 |
| %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 | 当地时区的名称 |
| %% | %号本身 |

三种格式转换（时间戳是给机器，格式化是给人看，结构化是从给机器看的转化为给人看的过程）



#时间戳时间  
#15000000000  
#结构化时间  
t1=time.localtime(1500000000)  
print(t1)  
#格式化时间  
t2=time.strftime("%Y-%m-%d %H %S %M %b %a",t1)  
print(t2)  
#结构化时间  
t3=time.strptime(t2,"%Y-%m-%d %H %S %M %b %a")  
print(t3)  
#时间戳时间  
t4=time.mktime(t3)  
print(t4)

## day27-04(时间模块练习题)

#计算本月一号的时间戳时间  
t1=time.localtime()  
t2=time.strftime("%Y-%m",t1)  
t2+="-01"  
t3=time.strptime(t2,"%Y-%m-%d")  
print(t3)  
t4=time.mktime(t3)  
print(t4)

作业计算两个时间差（两个格式化的时间差）

## day27-05(sys模块)

import sys #和python解释器打交道  
print(sys.path)#寻找模块的路径  
print(sys.modules)#存放了多少个模块路径  
print(sys.platform)#win32  
#sys.exit()



#第一元素 是执行这个文件的时候 写在python命令后的第一个值  
arg0=sys.argv[1]  
arg1=sys.argv[2]  
if arg0=="wdd" and arg1=="123":  
 print("success")  
else:  
 print("fail")  
 exit()

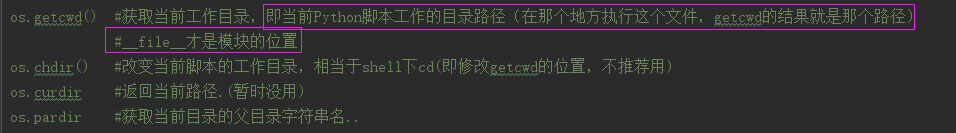
cmd 中执行

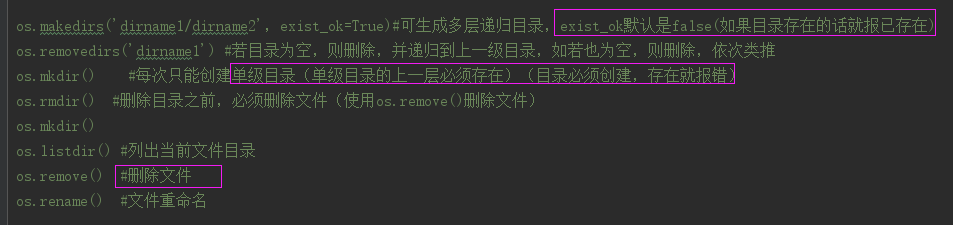
D:\wdd\pycharm\test\venv\Scripts>python D:/wdd/pycharm/python\_code/part2/34sys.py wdd 123

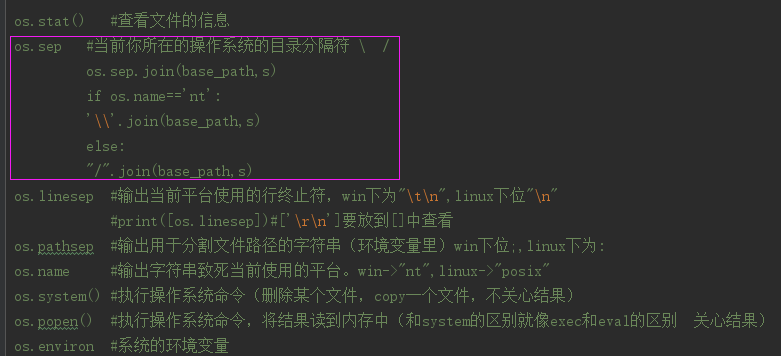
success

## day27-06(os模块1)

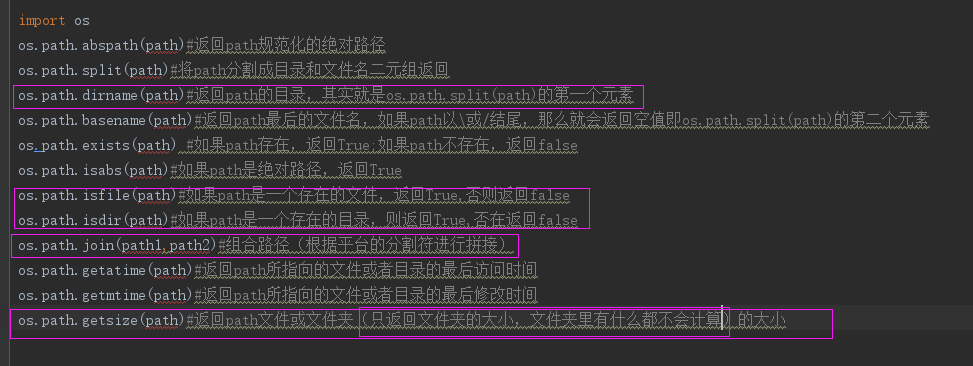








## day27-07(os模块2，os.path)



print(\_\_file\_\_)  
print(os.path.dirname(os.path.dirname(\_\_file\_\_)))

## day27-08(序列化模式)

归一化（多个程序导到一个接口中去）

实例化

初始化

序列化（将字典列表数字对象序列化成字符串）

为什么要序列化（要把内容写入文件，网线上传的就是电压，网络传输的时候要序列化）

### 序列化方法一

dic={"a":"c"}  
print(str(dic))#序列化成字符串  
print([str(dic)])  
print(eval(str(dic)))#反序列化

eval不能随便用（不推荐）

### 序列化的三种工具

#### json（dumps,dump,loads,load）(支持的数据类型少)



##### json的读和写(和文件有关)

with open('json',mode='w') as f:  
 #f.write(str\_dic)  
 json.dump(dic,f)#和f.write(str\_dic)等价  
  
with open('json',mode='r') as f:  
 print(json.load(f))

#### pickle(dumps,dump,loads,load不能跨语言，但是支持的类型比较多)

#### shelve

## day28-01,02

## day28-03(序列化)

### json格式的限制1（json格式的key必须是字符串数据类型，如果数字为key,那么dump之后会强制转成字符串数据类型）

dic={1:2,3:4}  
str\_dic=json.dumps(dic)  
print(str\_dic)#{"1": 2, "3": 4}  
new\_dic=json.loads(str\_dic)  
print(new\_dic)#{'1': 2, '3': 4}

### json格式的限制2（对元组做value的字典会把元组强制转换成列表）

dic={'abc':(1,2,3)}  
str\_dic=json.dumps(dic)  
print(str\_dic)#{"abc": [1, 2, 3]}  
new\_dic=json.loads(str\_dic)  
print(new\_dic)#{'abc': [1, 2, 3]}

### json的格式限制3（对元组做key会报错，因为元组序列化会变成list）

### json的规则4

json的规则（如果你自己写了个json数据所有的key都必须用双引号引起来）

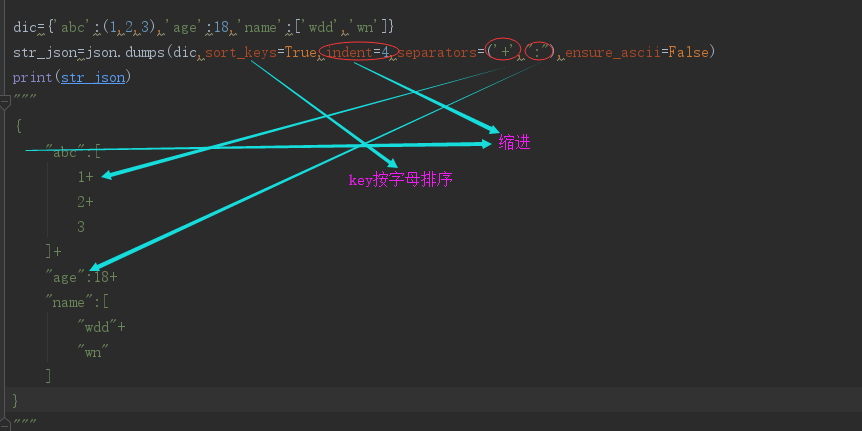
可以多次dump但是多次dump后不能load(因为多次dump后，数据类型的量变多了，也就是在文件了些了多个json数据，但是读数据的时候只能读取一个，如果非要这样做就要，只能这样)

dic={'abc':(1,2,3)}  
lst=['aaa','bbb','cccc','ddd']  
with open('json',mode="w",encoding="UTF-8") as f:  
 str\_dic=json.dumps(dic)  
 str\_lst=json.dumps(lst)  
 f.write(str\_dic+"\n")  
 f.write(str\_lst+"\n")  
  
with open('json',mode="r",encoding="UTF-8") as f:  
 for line in f:  
 ret=json.loads(line)  
 print(ret)

### 中文格式的ensure\_ascii=False

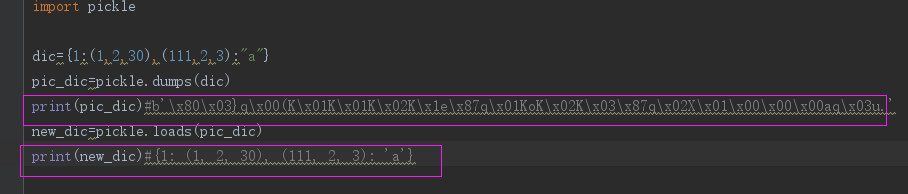
dic={'abc':(1,2,3),'country':'中国'}  
ret=json.dumps(dic,ensure\_ascii=False)  
print(ret)#{"abc": [1, 2, 3], "country": "中国"}  
ret2=json.dumps(dic)  
print(ret2)#{"abc": [1, 2, 3], "country": "\u4e2d\u56fd"}  
dic\_new=json.loads(ret)  
print(dic\_new)#{'abc': [1, 2, 3], 'country': '中国'}

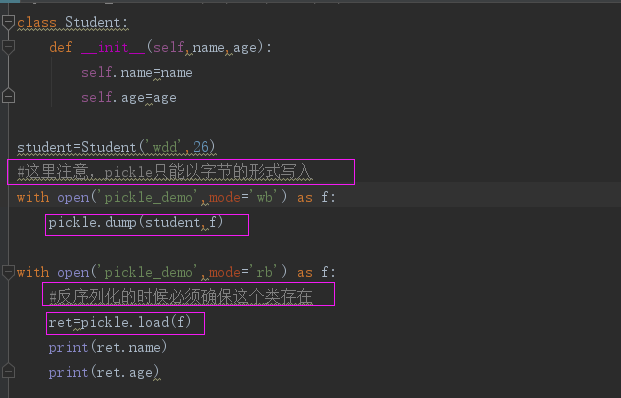
### json的其他参数（为了让程序员看起来方便，但是会浪费内存）



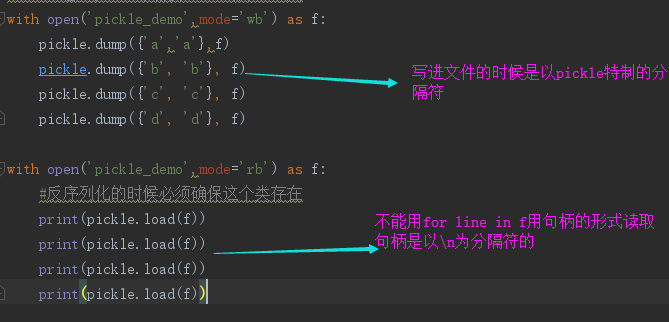
## day28-04(pickle上集)（解决json支持类型有限）

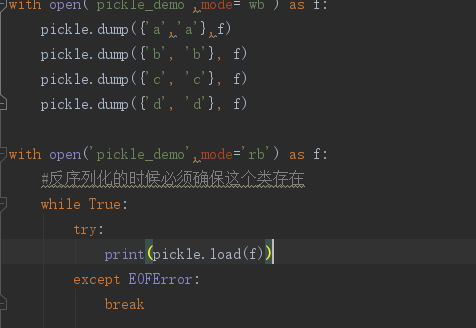
json不支持set数据类型，但pickle支持几乎所有的对象



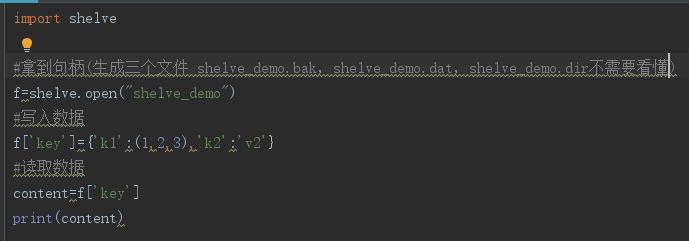


## day28-05(pickle下集)





## day28-06(shelve只有一个open方法 唯一的好处存进去的key可以直接取出来)



shelve如果你写定了一个文件

改动的比较少，读文件的操作比较多（频繁的取这个数据）

## day28-08(模块划分重点)

### hashlib模块（算法模块，摘要算法）

能够把一个字符串数据类型的变量转换成一个定长的密文的字符串，字符串里的每一个字符都是一个十六制数字（对数据库中的密码进行加密 字符串可以转成密文，但是密文不能转成字符串，这是一个不可逆的（所以对同一个字符串，用相同的算法，相同的手段进行摘要，获得的值总是相同的，只要不是相同的字符串，得到的结果一定不相同））

#### MD5(效率快，算法相对简单)

MD5是一个算法，32位的字符w 串，每个字符都是一个十六进制

import hashlib  
s="wdd"  
md5\_obj=hashlib.md5()#得到一个MD5对象  
md5\_obj.update(s.encode('utf-8'))#需要一个byte类型  
res=md5\_obj.hexdigest()#得到结果  
print(res,len(res),type(res))

#### sha1（效率慢，算法复杂）

sha1是一个算法，40位的字符串，每个字符都是一个十六进制

import hashlib  
s="wdd"  
md5\_obj=hashlib.sha1()#得到一个MD5对象  
md5\_obj.update(s.encode('utf-8'))#需要一个byte类型  
res=md5\_obj.hexdigest()#得到结果  
print(res,len(res),type(res))

#### 比较安全的形式

对数据加盐就是加任意字符串，只要代码不泄露就是安全）

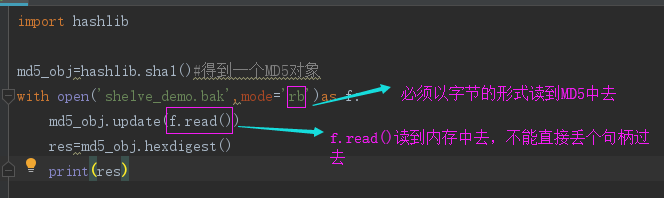
#### 最安全的形式

动态加盐（将用户名作为盐）

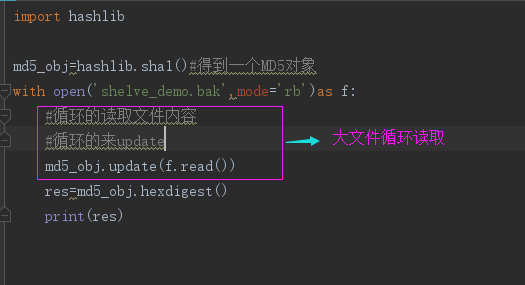
#动态加盐  
md5\_obj=hashlib.sha1(username)#得到一个MD5对象  
md5\_obj.update(s.encode('utf-8'))#需要一个byte类型  
res=md5\_obj.hexdigest()#得到结果  
print(res,len(res),type(res))

day28-09(hashlib下集)

#### 文件的一致性校验（MD5对象只能一次有效，每次用时都要实例化一个对象）

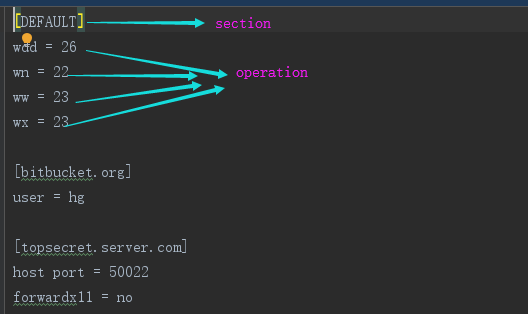


如果这个文件特别大，大到内存存不下

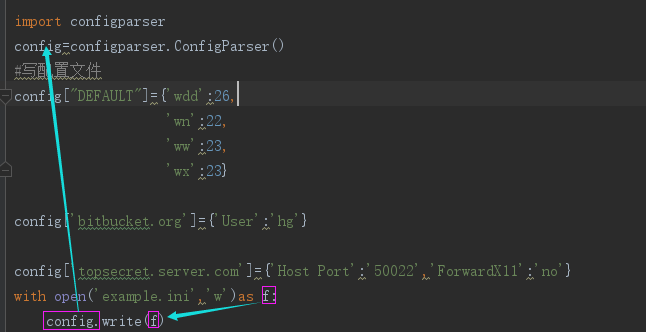


### configparse模块(配置文件的模块)

一种固定格式的配置文件，有一个固定的模块去帮你做这个文件的字符串处理

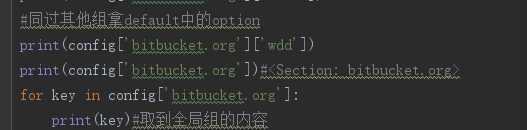


#### 写配置文件



#### 读配置文件





#### 增删改查

day28-11(logging)

### logging模块(一个指令控制所有的答应)

#### 功能

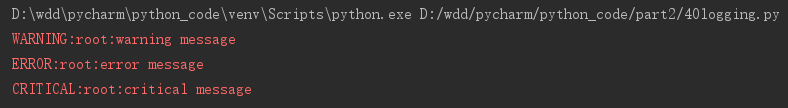
##### 日志格式的规范

##### 操作的简化

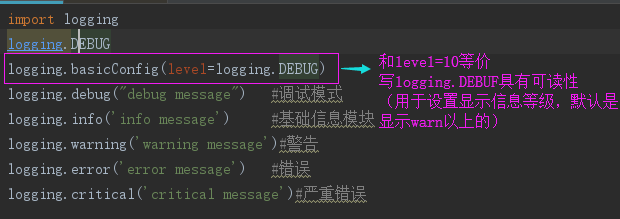
##### 日志的分级管理

import logging  
logging.debug("debug message") #调试模式  
logging.info('info message') #基础信息模块  
logging.warning('warning message')#警告  
logging.error('error message') #错误  
logging.critical('critical message')#严重错误

python自动输出警告以上的错误包括警告



怎么让他显示所有的警告

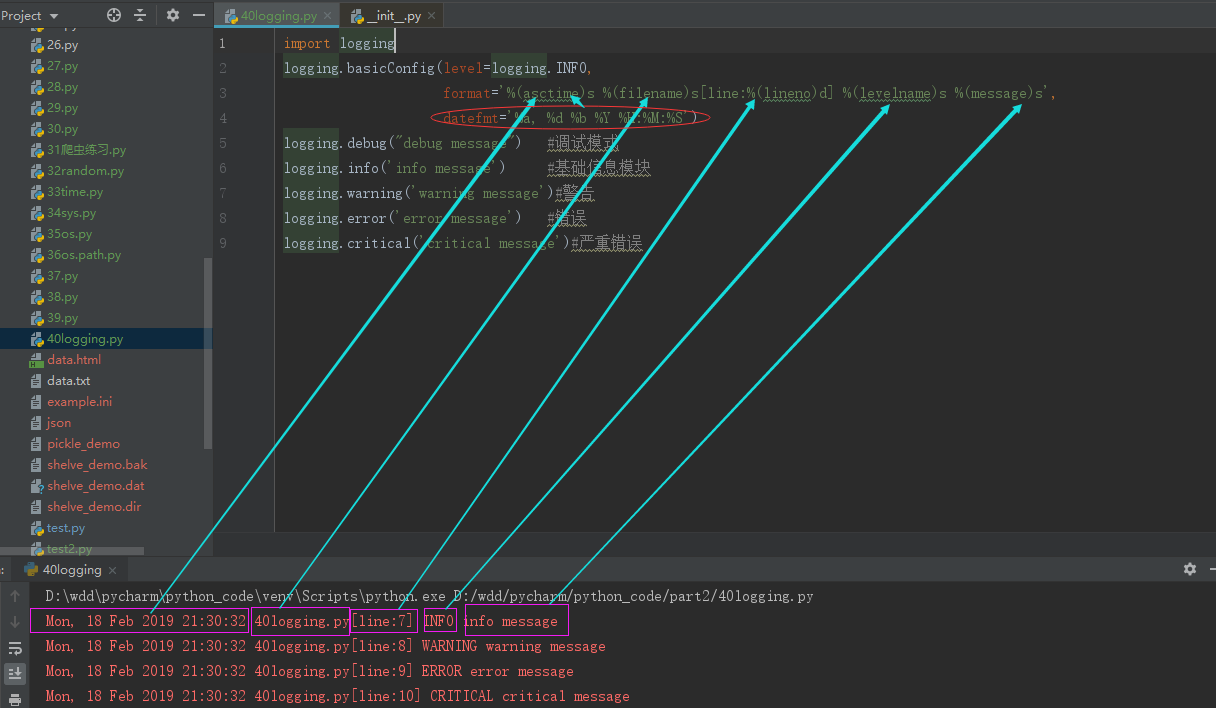


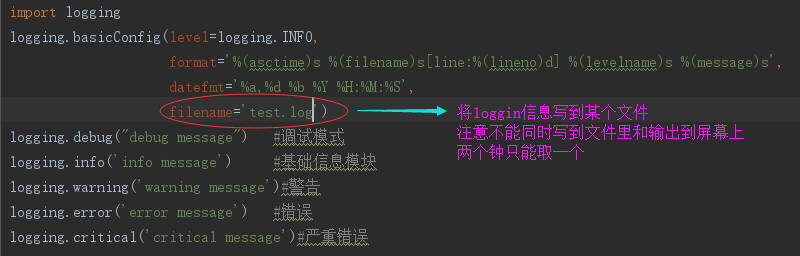
logging模块不能帮你做的事：

自动的生成你要打印的内容需要程序员在开发的时候定义好，要打印的内容是什么

### logging模块的使用：

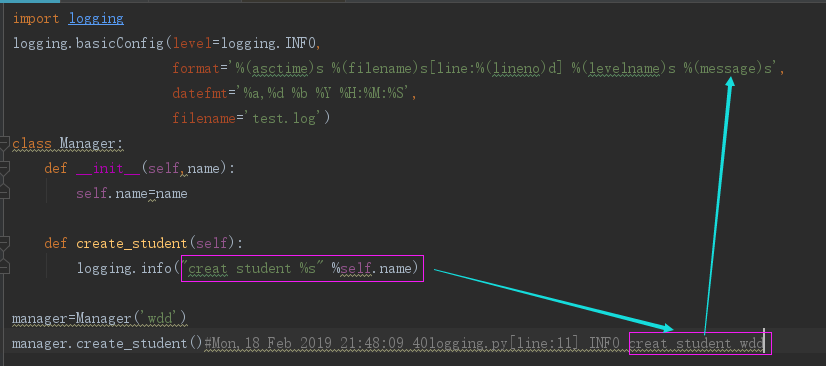
##### 普通配置型（简单的，可定制化差）





不能将一个log信息即输出到屏幕 又输出到文件

###### 实际应用

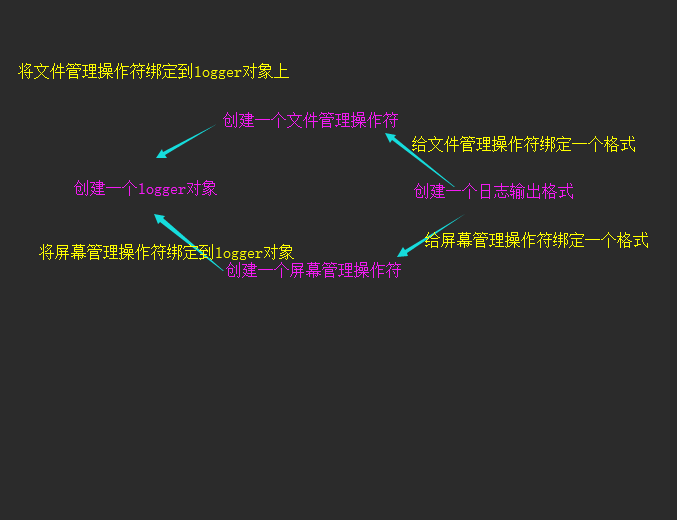


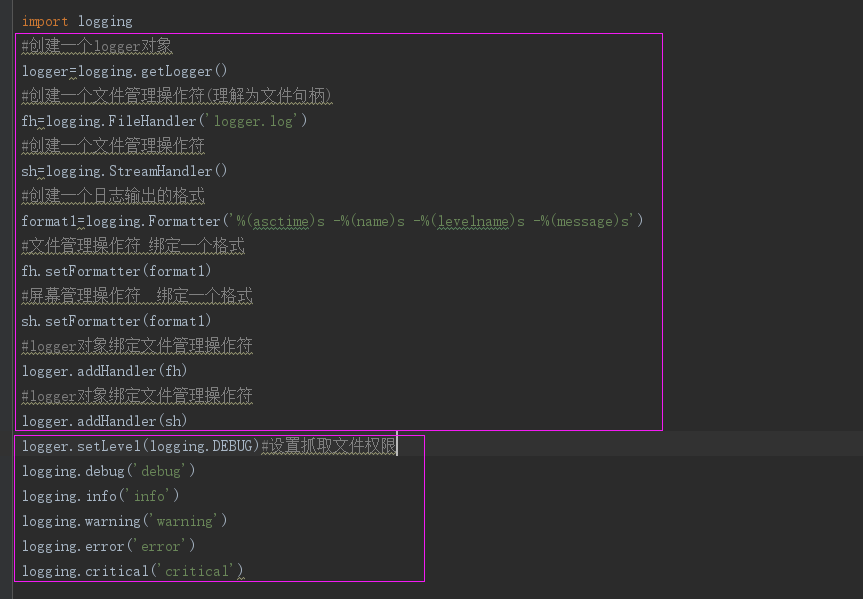
##### 对象配置型（复杂的，可定制化强）

logger对象的形式来操作日志文件（即往屏幕上输出又往文件里输出）

步骤：

1. 创建一个logger对象
2. 创建一个文件管理操作符
3. 创建一个屏幕管理操作符
4. 创建一个日志输出的格式（1，2，3，4这四个暂时没关系）
5. 文件管理操作符绑定一个格式
6. 屏幕管理操作符绑定一个格式
7. logger对象绑定文件管理操作符
8. logger对象绑定屏幕管理操作符





day27-12(内容概述)

day28-13(异常处理)

## 异常

### 什么是异常异常和错误的区别

Error错误(比较明显的错误，在编译代码阶段就能监测出来)

Iteration 异常（在执行代码的过程中产生的异常）

### 异常发生之后的效果

一旦在程序中发生异常，程序就不再继续执行了

### 如何看报错信息

### 最简单的异常处理

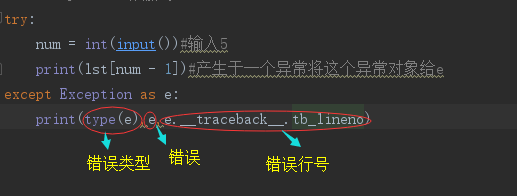
### 多分支异常处理（先处理一个异常在处理一个异常）

lst=['登陆','注册','取消']  
for i in enumerate(lst,1):  
 print(i[0],i[1])  
  
  
try:  
 num = int(input())  
 print(lst[num - 1])  
except ValueError:  
 print("请输入一个数字")  
except IndexError:  
 print("请输入1-3")

### 处理所有的异常

try:  
 pass  
except Exception:#处理所有的异常  
 pass

try:  
 num = int(input())#输入5  
 print(lst[num - 1])#产生于一个异常将这个异常对象给e  
except Exception as e:  
 print(type(e)，e)

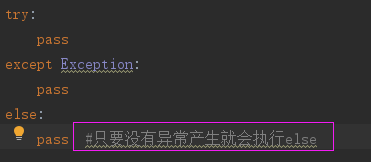


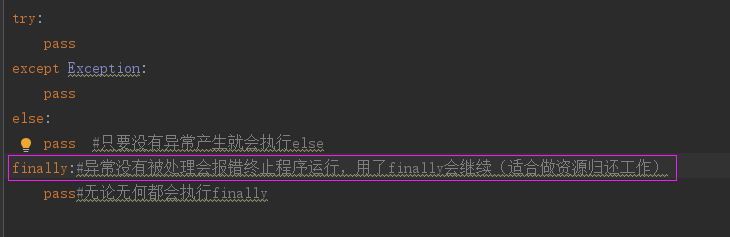
### 万能异常和分支异常的使用

万能异常放在所有异常的最相面，执行的顺序相当于条件判断，是互相排斥的

### 异常处理的其他机制

try:  
 num = int(input())  
 print(lst[num - 1])  
# except ValueError:print("请输入一个数字")  
# except IndexError:print("请输入1-3")  
except (IndexError,ValueError) as e:#用元组取代上面的分支语句  
 print(e)





总共四种情况

try/except

try/except/else

try/except/else/finally

try/except/finally

try/finally

### 主动抛异常

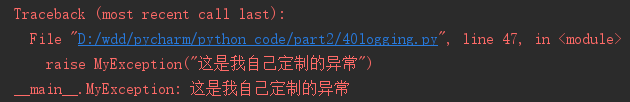
raise TimeoutError

#主动抛异常  
try:  
 num=int(input('>>'))  
except ValueError:  
 print("在出现了异常之后做点什么，再让它抛异常")  
 raise #原封不动的抛出try语句中出现的异常

### 自定义异常

BaseException是Exception的主宗（一般不用BaseException）

class MyException(Exception):  
 def \_\_init\_\_(self,msg):  
 self.msg=msg  
  
raise MyException("这是我自己定制的异常")



### 断言(和if 的判断效果一致，好处是不需要缩进)

assert True#是True就往下走  
assert False#是False就报异常

### 应用场景

断言和主动抛异常（自己写程序或者框架让别的程序员来调用）

使用-异常处理的注意事项

day28-14(双端队列)

## 双端队列

collection模块是数据类型的扩展模块

为什么直接跳过队列直接到双端队列

单队列

import queue  
#先进先出  
q=queue.Queue()  
q.put(1)  
q.put('a')  
q.put((123))  
q.put(({'k':'v'}))  
print(q)#<queue.Queue object at 0x000000F538B31E10>  
print(q.get())#1

print(q.qsize())

双端队列（两边都能放都能出，低层的数据结构是链表）

from collections import deque  
  
dq=deque()  
dq.append(2)  
dq.append(5)  
print(dq)#deque([2, 5])  
dq.appendleft('a')  
dq.append('A')  
print(dq)#deque(['a', 2, 5, 'A'])  
print(dq.pop())#A  
print(dq.popleft())#a  
print(dq.remove('a'))  
print(dq.insert(2,'123'))

在insert remove的时候deque的平均效率要高于列表

列表依据索引产看某个值得效率要高于deque

append和pop对于列表的效率是没有影响的