# 2018.8.11

### 转义字符”\”

列：

Print( ‘Let\’s go’ )

解：

斜杠后转换成字符串

### 换行显示：

在print中”\n”可以换行显示

### 取消斜杠转义功能

Print r”c:/a/b”

### 保留换行显示，可以注释

“””

Aaaaa

bbbbb

“””

# 2018.8.12

### 加括号会优先执行

(20 > 4) and 5 < 2

### 数字类型

#### 关于平方数

1.5e4 = 1.5E4 = 15000.0

如果使用e记法, ”整数类型” 会变成 ”浮点数类型“。

#### 关于布尔类型的小细节

布尔类型可以计算

True + False = 1

因为，True = 1 ，False = 0

#### 数据类型判断

type()可以显示数据类型,不过不推荐使用

isinstance() 返回布尔值

a = 5

isinstance(a,int) #如果a符号要求的数据类型,则返回"True"否则返回"False"

# 2018.8.13

### 计算符

A = 2

A = /= 3 相当于 A = 2 / 3

加减乘除类似

#### % ，\*\* ，//

5 % 2 结果取余数

5 \*\* 2 表示5的二次方

5 // 2 结果保留整数

### 优先级



### 3元操作符

x = 1  
y = 2  
a = x if x > y else y #如果条件: 为"真"则返回x否则返回y  
print (a)

### 自爆代码

assert 3>5 #如果条件为假则,程序主动崩溃

# 2018.8.15

### 计算参数的长度

each = '123456'

print (each , len(each)) #len显示参数长度

### range（）生成数字序列

常用于for中

For i in range(1 , 5): #生成从1开始5结束( 不包括5 )

List( range(1 , 10 , 2) ) #生成列表，每隔2个单位生成一个数字

### 关于列表

#### 增加元素

test = [ 'A','B',12,21.2,'c' ]

test.append('D') #增加单个元素

test.extend(['d','e']) #增加多个元素

test.insert(0,'S') #自定义位置注入元素

#### 删除元素

test = ['A', 'B', 12, 21.2, 'c']

test.remove('c') # 必须知列表元素值才能删除

del test[0] # 删除元素

cut = test.pop(1) # 取出元素

#### 复制列表方法

test = ['A', 'B', 12, 21.2, 'c']

test2 = test[ : ] # 全复制

test3 = test[ 0:3 ] # 切片复制

#### 查看列表元素方法

list = [1,2,3,4,5,6,1,1,3,3,44,5]

list.count(1) # 查看列表中相同元素数量

list.index(44) # 查看元素在列表中的序列

#### 列表排序方法

list.reverse() # 列表序列元素翻转

list.sort() # 从小到大排序

list.sort(reverse=True) # 先排序后翻转( sort可以添加一些参数来进行排序 )

#### 列表的正确复制方法

List = [ 1,2,3,4 ]

List2 = list[ : ]

不能通过赋值的方式复制列表, 那只是增加一个指引而已, 并没有真正的复制。

# 208.8.19

错误：count() 计算列表函数 ， sort()是从小到大排序函数



### 请用一句话描述什么是列表？再用一句话描述什么是元组？

列表可包含各种类型数据并允许进行修改，元组被创建后则相当于无法修改的列表

元组可以使用切片功能 如: temp1 = (1,2,3,4) temp2 =temp1[:2]

#### 元组创建单个元素

Temp = （1，）关键点在逗号

### 字符串的小秘密

# 索引字符串数据

A = 'abcdefg'

print ( A[0], A[3] ) # 通过这种方法直接可以索引出字符串数据

print ( type( A[0]))

字符串可以使用切片功能

A = 'abcdefg'

B = A[ :2 ]

# 2018.8.20

## 字符串方法大全

#### # 字符串方法,位置更替format()

a = "{0} love {1}".format("I" , "you")

b = "{a} love {b}".format(a="I" , b = "you")

print (a,b)

# # 字符串格式修改函数  
  
 # 字符串开头字母大写  
 a1 = 'zhao teng'.capitalize()  
 print (a1)  
  
 # 字符串全部小写  
 a2 = 'Zhao'.casefold()  
 print (a2)  
  
 # 字符串剧中  
 a3 = 'a'.center(15)  
 print (a3)  
  
 # 设置tab空格大小  
 a6 = 's\tpace'.expandtabs(tabsize=4)  
 print(a6)  
  
 # 以字符串作为分隔符，插入到111中所有的字符之间  
 a19 = 'oo7'.join('111')  
 print (a19)  
  
 # 大写转小写字母  
 a20 = 'oosdASD'.lower()  
 print (a20)  
  
 # 去掉字符串左边空格  
 a21 = ' oo77 '.lstrip()  
 print (a21)  
  
 # 替换(原内容,新内容,替换次数)  
 a22 = 'abcdeasdfasdcas'.replace('a','1',3)  
 print (a22)  
  
 # 去掉末尾空格  
 a24 = 'oo7a '.rstrip()  
 print (a24)  
  
 # 查看是否以指定字符开头(可自定义开始位置)  
 a25 = 'zhaosdsd'.startswith('zhao',0,4)  
 print (a25)  
  
 # 删除前后空格  
 a26 = ' asdasdaadss '.strip()  
 print (a26)  
  
 # 大小写翻转  
 a27 = 'asdsASD'.swapcase()  
 print (a27)  
  
 # 强制开头大写其他小写  
 a28 = 'zhao'.title()  
 print (a28)  
  
 # 所有小写转化为大写  
 a29 = 'asdasd'.upper()  
 print (a29)  
  
  
# # 部分字符串内容判断函数  
  
 # 统计指定字符串数量(可自定义位置),然后返回数量  
 a4 = 'jkhaksdnbfkbnavnboabdofe'.count('a',0,4)  
 print (a4)  
  
 # 判断是否以指定字符串结尾(可自定义位置),是则返回True否则返回False  
 a5 = 'asdasoo7'  
 test = a5.endswith('oo7')  
 print (test)  
  
 # 查看字符串中是否有指定字符串,有则返回索引值,否则返回-1  
 a7 = 'asdasdasdzhaoasfloo7asds'.find('oo7')  
 print (a7)  
  
 # 与find功能一样,从右开始查找  
 a23 = 'zhaoteng'.rfind('en')  
 print(a23)  
  
# # 整体字符串类型判断函数  
  
 # 如果字符串只有字母数字则返回True  
 a8 = 'asdasdasd'.isalnum()  
 a9 = '123123123'.isalnum()  
 a10 = 'asdas/23123asd'.isalnum()  
 print (a8,a9,a10)  
  
 # 如果字符串全部是字母则返回True  
 a11 = 'aaa22'.isalpha()  
 print (a11)  
  
 # 如果字符串全为10进制则返回True  
 a12 = '123123'.isdecimal()  
 print (a12)  
  
 # 如果字符串全为数字则返回True  
 a13 = '12312a333'.isdigit()  
 print (a13)  
  
 # 如果字符串全为小写字母则返回True  
 a14 = 'asds'.islower()  
 print (a14)  
  
 # 如果字符串全为数字则返回True  
 a15 = '12312a333'.isnumeric()  
 print (a15)  
  
 # 如果字符串全为空格则返回True  
 a16 = ' '.isspace()  
 print (a16)  
  
 # 如果字符串开头大写,其余小写则为True  
 a17 = 'Aasd'.istitle()  
 print (a17)  
  
 # 如果字符串字母全部为大写则返回True  
 a18 = 'ASDDa'.isupper()  
 print (a18)

### 格式化方法

其实就是给数据划一个“标准的规范”用各种函数制定规范

# 转化为Ascall码  
 a = '%c %c' % (97,98)  
 print (a)  
  
 # 格式化字符串类型  
 b = 90  
 a1 = '%s' % b  
 print (a1)  
 print (type(b))  
 print (type(a1))  
  
 # 格式化整数类型  
 a2 = '%d' % 20.56  
 print (a2)  
  
 # 格式化8进制  
 a3 = '%o' % 123  
 print (a3)  
  
 # 格式化为16进制  
 a4 = '%x' % 1232323395946  
 print (a4)  
  
 # 格式化为16进制大写  
 a5 = '%X' % 123213123123547788769  
 print (a5)  
  
 # 格式化为浮点数  
 a6 = '{0:.1f}'.format(23.68)  
 a7 = '%f' % 123  
 print (a6)  
 print (a7)  
  
 # 格式化e单位  
 print (1e3)  
 a8 = '%e' % 1213123124324.21313  
 print (a8)  
  
 # 合理化格式化单位选择 %f or %e  
 a9 = '%g' % 12123123.123  
 print (a9)  
  
# # 格式化辅助命令  
  
 # m( 占用格式大小 ).n( 保留几位小数 )  
 a10 = '%10.1f' % 273.233  
 print (a10)  
  
 # -左对齐  
 a11 = '%-10.1f' % 123.2  
 print (a11)  
  
 # 显示正号  
 a12 = '%+d' % 12  
 print (a12)  
  
 # 加入#可以识别0o为8进制, 0x为16进制  
 a13 = '%#o' % 1231412  
 a14 = '%#x' % 123120987  
 print (a13 + '\n'+ a14)  
  
 # 0表示空格替换成0  
 a15 = '%010d' % 123  
 print(a15)

# 2018.8.22

### 序列

# 关于list()与len()  
a = 'asdasd'  
b = list(a) # 关于list()将,字符串转化成列表的元素  
c = len(b) # 关于len(),返回元素个数  
print (b)  
print (c)  
  
# 关于max(), 返回最大值.min()相反;在比较字母时实际比较是ascll码,但是字母和数字不可以混合比较  
e = (1,23,4,3425,32)  
e1 = [123,123,2134,4,523,21]  
e = max(e)  
e1 = max(e1)  
print (e,e1)  
  
# 关于sum() , 元素相加返回和  
f = [1,23,4,5,3]  
f1 = (12,23,2,43)  
print (sum(f) + sum(f1))  
  
# 关于sorted()与reversed()与sort()和reverse()没有什么区别  
  
# 关于enumerate(), 将引导值与元素值组成元组  
g = [23,34,5,6,23,2]  
g1 = list(enumerate(g))  
print (g1)  
  
# 关于zip() ,二个列表合并对应元素位置,组合成元组  
h = [1,2,3,4,5,6,7]  
h1 = [7,6,54,3,21,]  
print(list(zip(h,h1)))

# 十进制转换为二进制

Bin()

# 2018.8.29

### 关于函数参数问题

# 给于参数默认值,可以保证在没有参数情况下不出错  
def test (name = 0 , score = 0):  
 print (name,':',score)  
  
# 防止参数传递顺序错误的问题  
def test2 (name = 0 , score = 0):  
 print(name, ':', score)  
  
# 收集参数,可以传入多个参数( 保存为元组 )  
def test3 (\*name , score = 0): # "\*name"可以收集多个参数,并保存为元祖.  
 print(name[:], ':', score)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test()  
 test2(score=88 , name ='赵飞')  
 test3('赵',1,2,3,4,'a',score='5000000000')

# 关于全局变量与局部变量

函数内是可以访问全局变量。

如果全局变量名称与局部变量名称相同时，则互不影响。

## 然后在函数内部修改全局变量

def test():

global x # 加入global , 就相当于全局变量的x , 如果没有它就只是局部变量.

x = 5

return x

## 非全局外部变量引用

def test2():

s = 20

def test3():

nonlocal s # 在使用" 非全局外部变量 "时加入nonlocal可防止错误 , 可以有效的将变量引用.

s \*= 2

return s

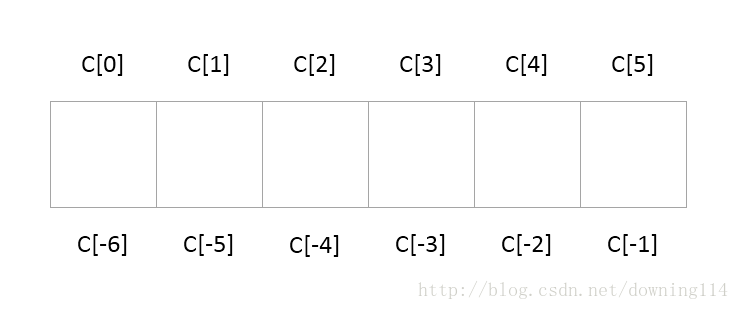
return test3()

## 关于end=‘’

print默认是打印一行，结尾加换行。[end](https://www.baidu.com/s?wd=end&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "_blank)=' '意思是末尾不换行

## 关于切片

切片a[ start , end , step ] start开始，end结束(不包含结束值位置)，step跳跃步数



此时，如果我们将步长设为-1，那么就可以很方便的得到一个反序的序列了：

>>> C[::-1]

['F', 'E', 'D', 'C', 'B', 'A']

# 正则表达式提前小知识

## 生成和提取

Re.compile()用法

pattern = re.compile('[a-zA-Z]') #生成字母

findall()用法

print( pattern.findall( 'asdasd234234asd123213' ) ) #进符合条件的提取出来返回成列表

pattern2 = re.compile(' ')

print( pattern2.findall(x) ) # findall()只接受pattern对象,并以列表的形式返回相等的元素

# 2018.9.6

### Lambda的用法

一些简洁的def语句可以用lambda代替更加方便

g = lambda x : x\*\*2

print(g(5))

add1 = lambda x,y : x+y

print(add1(5,6))

### Filter()函数过滤器的用法及与lambda写法结合

Filter()其实是过滤器,只返回“真”，可以传入2个参数,一个函数一个加工对象

加工结果为真则返回出

print(list(filter(None,[ 1,0,True,False ]))) **# 返回真**

test = filter( lambda x : x % 2 , range(9) ) **# 返回偶数**

print(list(test))

### Map()用法

Map()可以传入2个参数，一个函数，一个加工对象，并且返回一个可迭代结果( 可以以列表的形式返回结果 )

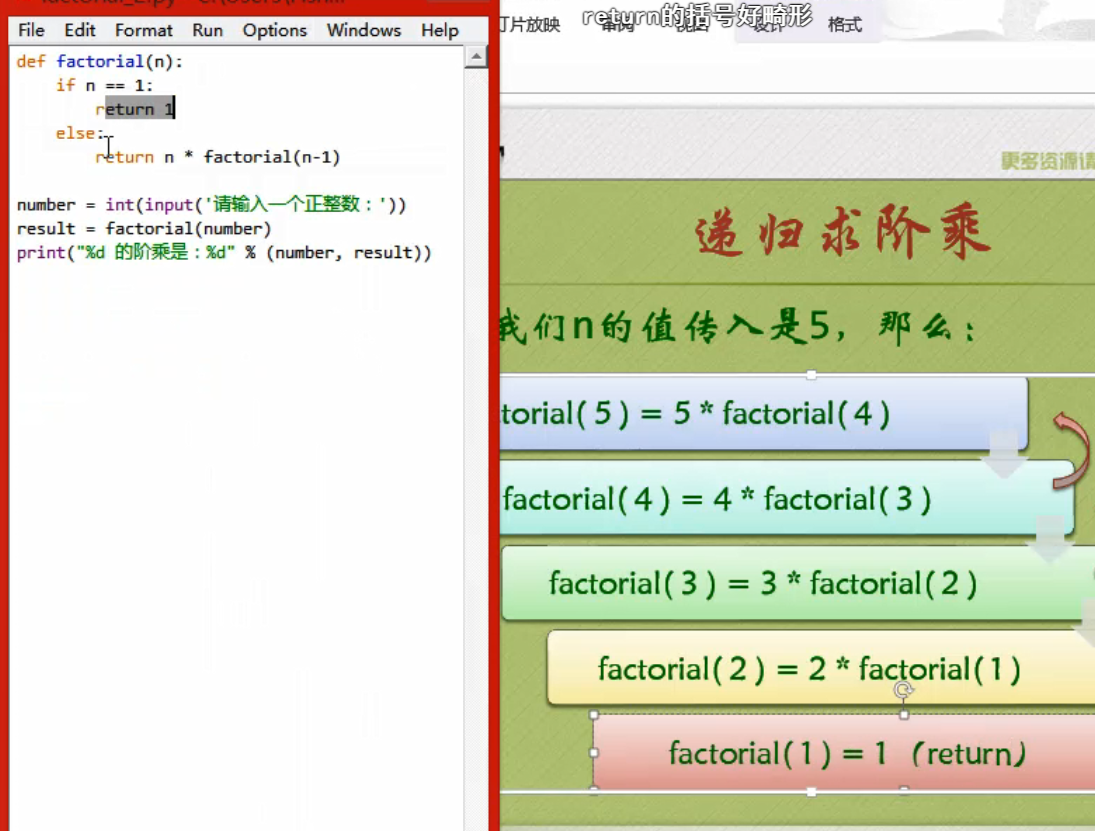
test2 = map( lambda x : x\*\*2 , range(9) ) **# 便捷式加工**

print( list(test2) )

## 递规（用到恰到好处才能体现水平）

什么才是递规?

1. 调用自身
2. 不是死循环



# 2018.9.10

## 字典创建与调用

### # # 创建字典

#### # 标准创建字典方法

dict1 = { 1 : ' one ', 2 : 'two' }

#### # 利用dict()方法创建字典

dict2 = dict(三='three', 四='four' ) # 此方法"键值"有问题!

dict3 = dict((( 5,' five ' ), (6,' six ')))

#### # 利用fromkeys()创建字典( 主要用来创建键值 )

dict4 = {}

dict5 = {}

dict4 = dict4.fromkeys((range(5)),'seven')

dict5 = dict5.fromkeys((1,2,3),'eight')

print(dict1,dict2,dict3,dict4,dict5)

### # # 字典的修改/创建元素

a={1:'what' , 2:'fuck'}

print(a)

a[1] = 'hello' # 通过键值修改对应元素

a[3] = '!' # 添加元素

print(a)

### 字典值迭代需要用的方法keys(),values(),items()

# keys()引用字典的键值

# values()引用字典的内容

# items()以元组的形式索引出字典元素

For eachKey in dict1.keys():

Print(eachKey)

### 用正确的姿势引导字典元素

#### Get()索引字典元素用法

因为使用

Dict1 = {}

Dict1 = dict1.fromkeys( (range(31)),’hello’ )

Dict1[32] # 因为没有32元素所以会报错

Dict1.get(32 , ’ 没有 ’) # 则不会报错,并提示“没有”( 默认返回None )

#### 利用关系符判断是否存在字典

继上方dict1字典

判断有无元素

32 in dict1 # 有则返回True否则返回False

#### 清空字典

# # 清空字典

c = {}

c = c.fromkeys((range(10)),'hello') # 生成字典

print(c)

c.clear() # 执行清空函数

print(c)

#### 拷贝字典

D = {}

D = d.fromkeys(fromkeys(),’hello’)

E = d.copy() # 复制字典

d.clear() # 删除被复制的字典

print(e) # e字典依然有自己的元素

#### 取出字典元素

# # 取出字典元素

# pop() 取出字典单个元素

# popitem() 随机返回并删除字典中的一对键和值(一般删除末尾对)

f = {}

f = f.fromkeys((range(5)),'hello')

f.pop(1) # 取出对应字典元素

f.popitem() # 取出最后一位字典元素

print(f)

#### 添加字典元素

添加字典元素有很多方法

方法1：

a[3] = '!' # 添加元素

方法2：

Setdefault()方法添加元素

# # 通过setdefault() 添加字典元素

g = {}

g = g.fromkeys((range(5)),'wokao!')

print(g)

g.setdefault(5,'2333')

print(g)

# 通过便捷方法添加字典元素

print( 6 in g ) # 用关系符检查一下字典中是否有无键值

g[6] = '55555' # 添加元素

print(g)

#### 更新字典方法

说白喽其实将一个字典元素添加到另外一个字典

# # 更新字典

h = {}

h = h.fromkeys((range(5)),'++++')

j = {666:'---'}

print(h,j)

h.update(j) # 更新 ( 将j字典中的元素加入到h字典中 )

print(h)

### 列表转化为字典

Test = dict([('two', 2), ('one', 1), ('three', 3)])

可以将其转化为字典

### 恢复记忆字符分割split(',')

split(',')以逗号为分割点转化为列表

data = "1000,小甲鱼,男"  
MyDict = {}  
**# 还记得字符串的分割方法吧，别学过就忘啦^\_^**  
(MyDict['id'], MyDict['name'], MyDict['sex']) = **data.split(',')**  
  
print("ID: " + MyDict['id'])  
print("Name: " + MyDict['name'])  
print("Sex " + MyDict['sex'])

### 集合set()

Set1 = {1,2,3}

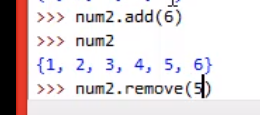
list = [1,2,3]

set1 = set(list) 转化成集合

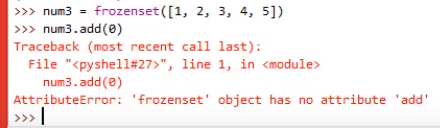
1.集合中没有重复的元素，并且不可索引

2.可以用in等关系符判断是否存在元素

#### 修改集合



不可变集合



# 2018.9.13

关于字典底层原理通过哈希方法存储数据请参考：

<https://fishc.com.cn/thread-45016-1-1.html>

### 集合set()

1.集合不支持索引

2.元素不重复

3.也可以通过关系符判断是否有对应元素

#### 集合常用的内置函数

set1 = {1,2,3,4}

print(set1)

set1.add(233) # 添加元素

set1.remove(2) # 移除元素

print(set1)

# 创建不可变合集

set2 = frozenset([1,2,3,4])

# 常用内置函数

set1.pop() # 移除并返回一个任意元素

set3 = {5,6}

set1.update(set3) # 更新函数

print(set1)

# 2018.9.17

### 文件操作

#/usr/bin/python3

#-\*- coding:utf-8 -\*-

if \_\_name\_\_ == "\_\_main\_\_":

# # 文件的读取 , 查看 , 写入

# # 读取

# 读写模式很重要

# 'a' : 追加模式

# 'w' : 覆盖写入模式

# 't' : 文本模式查看( 默认 )

# 'r' : 只读打开方式( 默认 )

# 'b' : 以二进制模式打开

# '+' 可读写模式（可添加到其他模式中使用）

# 'x' 这里要注意的是'x'和'w'均是以“可写入”的模式打开文件，但以'x'模式打开的时候，如果路径下已经存在相同的文件名，会抛出异常，而'w'模式的话会直接覆盖同名文件。

# 因此，'w'模式打开文件会比较危险，容易导致此前的内容遗失，因此使用'w'模式打开文件前先检查该文件名是否已经存在显得非常重要！

filetest = open('G:test.txt','at+',encoding='utf-8') # 使用open函数( 路径，读写模式，文件编码 )

# # 查看

f.read() # 查看全部 , 如果有值则查看一部分

f.readline() # 查看一行, 如果有值则查看一部分

f.tell() # 查看当前标签位置

# # 写入

# 二个文件内容复制时 f2.write(f1.read())

f.write( 'str' ) # 写入字符串

f.writelines(列表) # 直接写入一个列表或者序列内容

f.seek(x , y) # y设置起点(0代表起点,1代表当前位置,2代表尾部),x设置标签前进字节大小

f.tell() # 返回当前标签位置

#### 关于换行问题直接在即将写入的字符串中加入”\n”

# open()更多信息请查看: <https://fishc.com.cn/forum.php?mod=viewthread&tid=45279&extra=page%3D1%26filter%3Dtypeid%26typeid%3D403>

### 迭代器

import sys

list = [1,2,3,4]

it = iter(list) # iter()创建一个迭代器

while True:

try:

print(next(it),end="") # next(it)输出迭代器下一个元素

except StopIteration: # StopIteration防止迭代器出现死循环

sys.exit()

#### 关于iter（）函数多行输入小技巧

stopword = ':w'

str = ''

for line in iter(input, stopword): # 此时iter有二个参数，而stopword是设置停止条件

str += line + '\n' # 在每一行尾加入换行符号

### 关于try提前的探索

如果try发生异常，则执行except语句

try:

f = open('D:test3.txt','rt')

except:

f = 0

print(f)

# 2018.9.23

### os模块更多信息：(https://fishc.com.cn/forum.php?mod=viewthread&tid=45512&extra=page%3D1%26filter%3Dtypeid%26typeid%3D403 )

#### Os.path分支



#### 对于路径以及文件的判断,采用函数

import os

# # 判断路径

os.path.exists('E://test.txt') # 判断文件/目录是否存在

os.path.isabs() # 判断是否是绝对路径

os.path.isdir() # 判断是否存在目录

os.path.isfile() # 判断文件是否存在

os.path.islink() # 判断是否是一个符号链接

os.path.ismount() # 判断是否是挂载点( 硬盘分区 )

os.paht.samefile('路径1','路径2') # 判断2个路径是否指定的是一个文件

# 2018.9.30

### Pickle存储模块

具体使用方法

# 存储

test = [1,2,34,5,123,12,324,4,234,324,234,234]

pickle\_file = open('test.pkl','wb') # 后缀名随便写,不过最好要有提示性( 注意读取模式使用二进制写入 )

pickle.dump( test , pickle\_file ) # test列表保存到test.pkl文件

pickle\_file.close()

# 读取

pickle\_file = open('test.pkl','rb') # 注意读取模式使用二进制读取

test = pickle.load( pickle\_file ) # 读取pickle文件

print(test)

#### 关于os,walk下的使用详解

#!/usr/bin/python3

import os

road = input('输入: ')

# # 关于os.walk

# 接受:

# 接受3个数据参数,开始路径road,onerror错误报告默认为None可不写,反向目录获取topdown=False

# 返回:

# 一个元祖( road(所有文件夹路径 - 字符串类型) , dirs(所有文件夹名称 - 列表类型) , file(所有文件名称 - 列表类型) )

for road, dirs, files in os.walk(road, topdown=False):

# 显示所有文件( 带路径位置 )

for i in files:

print( road+os.sep+i ) # os.sep是本系统路径分隔符

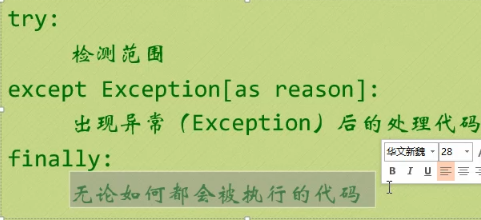
# 2018.10.9

### 异常处理

Test = [1,2,3]

Assert len(test) > 0 # 断言验证,为假则抛出异常

#### 关于try….except异常处理



关于TRY有二种使用方法，1，具体报错处理，2，全异常处理

错误类型的更多信息在: <https://fishc.com.cn/forum.php?mod=viewthread&tid=45814&extra=page%3D1%26filter%3Dtypeid%26typeid%3D403>

方法1: 推荐使用具体错误信息处理：

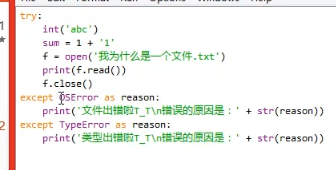
关于Except OSError(指定检测错误的类型) as reason(具体报错信息变量) :

try:

int('abc')

except ValueError as x:

print('错误ValueError: '+str(x))





方法2: 全异常处理



#### Raise故意引出异常

故意引出数字报错异常



s = 0

try:

int('12')

raise ZeroDivisionError # 抛出异常

except (ValueError, ZeroDivisionError) as x:

print('错误: '+str(x))

### 关于else用法

if else , while else , try except else，for else

#### While…..else….

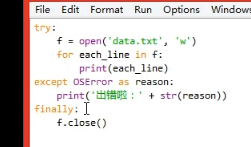
While else如果while被完整的执行,中间没有break那么会执行else的语句,否则不会执行else语句

#### Try……except……else

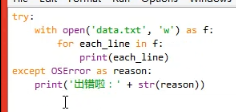
如果try出现异常则不会执行else，相反如果try没有异常则执行else

### With自动关闭文件用法

普通使用try方法：



使用with方法：



try:

with open('e://test.txt','wt+') as file:

for i in file:

print(i)

except NameError as x:

print('NameError'+str(x))

else:

print('成功读取')

#### with打开多文件方法

1. with A() as a, B() as b:
2. suite

with open(file1) as f1 , open(file2) as f2:

print(‘OK’)

### 导入模块的3种方法

第一种: import easygui

第二种: from eaysui import \* ( 这种方法，在打包文件时减少大小有非常重要的作用 )

第三种: import easygui as g ( 匿名模块,写起来轻松 )



### Os.system下多条语句执行方法

os.system('第一条命令 && 第二条命令')多条命令执行方式

b=os.popen('ls') # 功能与system相同，不过可以读取执行后的所显示的内容

b.read()

### 使用subprocess调用cmd命令，并且等待子程序完成才执行其它命令

cmd = "%s && %s" % ( setpath,ffmpegset ) # 调用多条语句

a = subprocess.Popen(cmd,shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT) # 调用cmd命令

a.wait() # 等待子程序( cmd命令 )运行结束,才开始执行新的命令