# 1.第一个大坑

Python2中接收输入用x=raw\_input(“please input name”)

Python3中接收输入用x=input(“please input name”)

# 2.Python3中有六个标准的数据类型：

字符串（String）str=’abc’或者str= “abc”  
数字（Digit）num=256

(int)(long) python3.x之后已经不区分int和long,统一用int

(float)  
列表（List） ListA=[‘a’,’b’] (可重复，有序，可以改，类型可不同)  
元组（Tuple）TupleA=(‘a’,’b’) (可重复，有序，不能改，)  
集合（Sets）SetsA={‘a’,’b’} (不重复，无序，可以改，可用来去重)  
字典（Dictionary）DictionaryA={1:"nihao",2:"hello"}  
日期（date）

布尔(bool) Type False

# print()返回值为None

# for 循环后面的变量是全局变量

**for** takes **in** range(1,10):  
 print(**'takes'**,takes)  
print(**'globle takes='**,takes)

结果\*\*\*\*\*\*\*\*\*\*\*\*\*\*

takes 7

takes 8

takes 9

globle takes= 9

spam = ['cat', 'bat', 'rat', 'elephant']

# 4.列表加、乘字符串

>>> spam = 'Hello'

>>> spam += ' world!'

>>> spam

'Hello world!'

列表加一个字符串，相当于往后边添加一个列表元素

>>> bacon = ['Zophie']

>>> bacon \*= 3

>>> bacon

['Zophie', 'Zophie', 'Zophie']

列表乘以数字相当于复制

# 列表的insert方法

insert() 方法 的 第一个 参数 是 新 值 的 下标， 第二个 参数 是要 插入 的 新 值。

>>> spam = ['cat', 'dog', 'bat']

>>> spam. insert( 1, 'chicken')

>>> spam

['cat', 'chicken', 'dog', 'bat']

# 列表可以用的方法

Spam=[‘ants’,’cats’,’dogs’]

Spam.append(‘badgers’)

Spam.insert(1,’chicken’)

Spam.remove(‘cat’)

Spam.sort() #按照ASCII

Spam.sort(key=str.lower) #统一按照小写字母排序

# 一行代码太长需要换行，在行末加一个\,

# 下一行缩不缩进都行，都会被看成是上一行的延续

print(' Four score and seven ' + \

'years ago...')

# 概念知识点，其实字符串也可以看成是单个文本的列表，当然也可以切片与及用于in与not in的操作，但字符串不能改变，例如不能给name[2]=’the’会报错

>>> name = 'Zophie'

>>> name[ 0]

'Z'

>>> name[- 2]

'i'

>>> name[ 0: 4]

'Zoph'

>>> 'Zo' in name

True

>>> 'z' in name

False

>>> 'p' not in name

False

>>> for i in name: print('\* \* \* ' + i + ' \* \* \*')

\* \* \* Z \* \* \*

\* \* \* o \* \* \*

\* \* \* p \* \* \*

\* \* \* h \* \* \*

\* \* \* i \* \* \*

\* \* \* e \* \* \*

# 元组Tuple的值不能被修改，添加或者删

注意：当元组只有一个值里，需要加**逗号**\*\*\*

>>> type((' hello',))

< class 'tuple'

> >>> type((' hello'))

< class 'str'>

# 类型转换

>>> tuple([' cat', 'dog', 5])

('cat', 'dog', 5)

>>> list((' cat', 'dog', 5))

['cat', 'dog', 5]

>>> list(' hello')

['h', 'e', 'l', 'l', 'o']

# 11.列表变量附值，把一个列表变量的值给另一个列表变量，只是把列表变量对应ID给了另一个列表变量，改变一个列表变量里面的值，两个列表变量都会变

列表、字典（值可变）python存的是引用（类似id），

字符串、 整型 或 元 组，（值不可变） Python 变量 就 保存 值 本身。

# 如何复制列表跟字典呢？

（import copy 然后用cheese=copy.copy(spam)）

>>> import copy

>>> spam = ['A', 'B', 'C', 'D']

>>> cheese = copy. copy( spam)

>>> cheese[ 1] = 42

>>> spam ['A', 'B', 'C', 'D']

>>> cheese ['A', 42, 'C', 'D']

# 字典，概念知识点（1.无序2.整形数值可为键）

>>> spam = ['cats', 'dogs', 'moose']

>>> bacon = ['dogs', 'moose', 'cats']

>>> spam == bacon False

>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}

>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}

>>> eggs == ham True

# 字典的方法keys(),values()和items()--(返回的是元组)

数据 类型（ 分别 是 dict\_ keys、 dict\_ values 和 dict\_ items） 可以 用于 for 循环。

>>> spam = {'color': 'red', 'age': 42}

>>> for v in spam. values(): print( v)

red

42

这里， for 循环 迭代 了 spam 字典 中的 每个 值。 for 循环 也可以 迭代 每个 键， 或者 键- 值 对：

>>> for k in spam. keys(): print( k)

Color

age

>>> for i in spam. items(): print( i)

('color', 'red')

('age', 42)

## 遍历字典也可以用多重赋值的技巧

>>> spam = {'color': 'red', 'age': 42}

>>> for k, v in spam. items():

print(' Key: ' + k + ' Value: ' + str( v))

Key: age Value: 42

Key: color Value: red

## 技巧1：因为字典中如果没有对应的键值会报错，所以可以用get()方法，并给定默认值的方法

>>> picnicItems = {'apples': 5, 'cups': 2}

>>> 'I am bringing ' + str( picnicItems. get(' cups', 0)) + ' cups.'

'I am bringing 2 cups.'

>>> 'I am bringing ' + str( picnicItems. get(' eggs', 0)) + ' eggs.'

'I am bringing 0 eggs.'

## 技巧2：试探一个字典中有没有某一个键值，如果有则不做任何操作，如果没有，就给这个键附一个值，用一个方法

## spam. setdefault(' color', 'white')

>>> spam = {'name': 'Pooka', 'age': 5}

>>> spam. setdefault(' color', 'black')

'black'

>>> spam

{'color': 'black', 'age': 5, 'name': 'Pooka'}

>>> spam. setdefault(' color', 'white')

'black'

>>> spam

{'color': 'black', 'age': 5, 'name': 'Pooka'}

## 可以用到循环中，给变量附初始值，因为没有初始化，所以只执行一次

for character in message:

count. setdefault( character, 0)

count[ character] = count[ character] + 1

# 字符串转义字符\

>>> spam = 'Say hi to Bob\' s mother.'

也可以用双引号里边直接输入单引号

>>> spam = "That is Alice' s cat."

其它转义字符：

转义 字符 打印 为

\' 单 引号

\" 双 引号

\t 制表符

\n 换行符

\\ 倒 斜杠

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*重要\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

可以 在 字符串 开始 的 引号 之前 加上 r， 使它 成为 原始 字符串。“ 原始 字符串” 完全 忽略 所有 的 转义 字符，

>>> print( r' That is Carol\' s cat.') That is Carol\' s cat.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*重要\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

多行文本用三个单引号或三个双引号，多行注释也是这个。

# 字符串切片

' H e l l o w o r l d ! '

0 1 2 3 4 5 6 7 8 9 10 11

>>> a=’abcde’

>>>a[1:4:-1]

dcb

第三个参数是步长

>>>a[::2]

ace

当为-1时从反方向取步长

# 注意：

字符串 方法 upper()、 lower()、

方法 没有 改变 字符串 本身， 而是 返回 一个 新 字符串。

spam = spam. upper()， 才能 改变 spam 中的 字符串的值

其它方法：

isalpha() 返回 True， 如果 字符串 只 包含 字母， 并且 非 空；

isalnum() 返回 True， 如果 字符串 只 包含 字母 和 数字， 并且 非 空；

isdecimal() 返回 True， 如果 字符串 只 包含 数字 字符， 并且 非 空；

isspace() 返回 True， 如果 字符串 只 包含 空格、 制表符 和 换行， 并且 非 空； .

istitle() 返回 True， 如果 字符串 仅 包含 以 大写字母 开头、 后面 都是 小写字 母的 单词。

在 交互式 环境 中 输入 以下 代码：

>>> 'hello'. isalpha()

True

>>> 'hello123'. isalpha()

False

>>> 'hello123'. isalnum()

True >>> 'hello'. isalnum()

True >>> '123'. isdecimal()

True

>>> ' '.isspace()

True

>>> 'This Is Title Case'. istitle()

True

>>> 'This Is Title Case 123'. istitle()

True

>>> 'This Is not Title Case'. istitle()

False

>>> 'This Is NOT Title Case Either'. istitle()

False

# 字符串以XX开始与结束

startswith() 和 endswith()

# 字符串以X合并，与以X分隔

方法 join() 和 split()

Join参数是一个列表

>>> ', '.join([' cats', 'rats', 'bats'])

'cats, rats, bats'

Split()

默认 情况下， 按照 各种 空白 字符 分割， 诸如 空格、 制表符 或 换行符。 这些 空白 字符 不 包含 在 返回 列表 的 字符串 中。

>>> 'My name is Simon'. split()

['My', 'name', 'is', 'Simon']

# 20.字符串对齐输出函数

用 rjust()、 ljust() 和 center() 方法 对齐 文本，缺少第二个填充的参数，**默认为空格**

>>> 'Hello'. rjust( 10)

' Hello'

>>> 'Hello'. rjust( 20, '\*')

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hello'

>>> 'Hello'. ljust( 20, '-')

'Hello---------------'

# 删除空白字符

用 strip()、 --左右都删

rstrip() --只删右边

lstrip() --只删左边

删除 空白 字符

# 将字符串内容输出到系统粘贴板

pyperclip 模块 有 copy() 和 paste() 函数， 可以向 计算机 的 剪贴板 发送 文本， 或 从 它 接收 文本， 将 程序 的 输出 发送 到 剪贴板。

>>> import pyperclip

>>> pyperclip. copy(' Hello world!')

>>> pyperclip. paste() 'Hello world!'

# 正则表达式匹配

用括号分组

（字符串转义回顾：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*重要\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

可以 在 字符串 开始 的 引号 之前 加上 r， 使它 成为 原始 字符串。“ 原始 字符串” 完全 忽略 所有 的 转义 字符

）

用括号给表达式分级，方便用group()取值

注意：group(0)取的是整个匹配的值，group(1)是取的第一个匹配的值

>>> phoneNumRegex = re. compile( r'(\ d\ d\ d)-(\ d\ d\ d-\ d\ d\ d\ d)')

>>> mo = phoneNumRegex. search(' My number is 415- 555- 4242.')

>>> mo. group( 1)

'415'

>>> mo. group( 2)

'555- 4242'

>>> mo. group( 0)

'415- 555- 4242'

>>> mo. group()

'415- 555- 4242'

用groups()获取所有匹配的项目

注意：返回的数据类型是一个不能改变值的元组（）括起来的。回顾：只有一个元素时，用逗号，例如yuanzu=（‘a’,）

>>> mo. groups()

('415', '555- 4242')

注意：python允许同时给多个变量附值，注意返回值的顺序要与附值参数的顺序要一致

>>> areaCode, mainNumber = mo. groups()

>>> print( areaCode)

415

>>> print( mainNumber)

555- 4242

字符| 称为“ 管道”。 匹配 许多 表达式 中的 一个时用，只会匹配第一个

>>> heroRegex = re. compile (r' Batman| Tina Fey')

>>> mo1 = heroRegex. search(' Batman and Tina Fey.')

>>> mo1. group()

'Batman'

>>> mo2 = heroRegex. search(' Tina Fey and Batman.')

>>> mo2. group()

'Tina Fey'

注意：

group()返回正则表达式匹配的整个文本，group(1)匹配的是分组括号里面的

>>> batRegex = re. compile( r' Bat( man| mobile| copter| bat)')

>>> mo = batRegex. search(' Batmobile lost a wheel')

>>> mo. group()

'Batmobile'

>>> mo. group( 1)

'mobile'

# 正则表达式匹配次数

？ 匹配0次或1次

\* 匹配0次或N次

+ 匹配1次或N次

{3} 匹配3次

{3,5} 匹配3次到5次

{,5} 匹配0次到5次

# 贪心与非贪心

在{3,5}后面不加问号，是贪心模式，如果三次，四次，五次都匹配，贪心模式取最多次数，取5次

在{3,5}后面加问号，是非贪心模式，如果三次，四次，五次都匹配，取最少的匹配次数，取3次

# findall()与search()类似

Search()返回的Match对象只包含第一次匹配的文本，即使有两个匹配项，也只返回第一个

>>> phoneNumRegex = re. compile( r'\ d\ d\ d-\ d\ d\ d-\ d\ d\ d\ d')

>>> mo = phoneNumRegex. search(' Cell: 415- 555- 9999 Work: 212- 555- 0000')

>>> mo. group()

'415- 555- 9999'

Findall()返回所有的匹配项，返回一个列表[]，**注意：返回的数据类型是列表，回顾：**groups()返回的是一个元组

>>> phoneNumRegex = re. compile( r'\ d\ d\ d-\ d\ d\ d-\ d\ d\ d\ d') # has no groups

>>> phoneNumRegex. findall(' Cell: 415- 555- 9999 Work: 212- 555- 0000')

['415- 555- 9999', '212- 555- 0000']

**注意：如果正则表达式有（）分组，那么列表的每一项是一个元组**

>>> phoneNumRegex = re. compile( r'(\ d\ d\ d)-(\ d\ d\ d)-(\ d\ d\ d\ d)') # has groups

>>> phoneNumRegex. findall(' Cell: 415- 555- 9999 Work: 212- 555- 0000')

[('415', '555', '1122'), ('212', '555', '0000')]

注意：如果正则表达式没有分组，返回的是一个字符串的列表

['415- 555- 9999', '212- 555- 0000']

如果正则表达式有分组，返回的是一个元组的列表

[('415', '555', '1122'), ('212', '555', '0000')]

# 正则表达式中的替换字符

常用 字符 分类 的 缩写 代码

缩写字符 分类 表示

\d 0 到 9 的 任何 数字

\D 除 0 到 9 的 数字 以外 的 任何 字符

\w 任何 字母、 数字 或 下划线 字符（ 可以 认为是 匹配“ 单词” 字符）

\W 除 字母、 数字 和 下划线 以外 的 任何 字符

\s 空格、 制表符 或 换行符（ 可以 认为是 匹配“ 空白” 字符）

\S 除 空格、 制表符 和 换行符 以外 的 任何 字符

例子：

注意：字符 分类[ 0- 5] 只 匹配 数字 0 到 5， 这 比 输入( 0| 1| 2| 3| 4| 5) 要 短 很多。

注意： 在 方 括号 内， 普通 的 正 则 表达式 符号 不会 被 解释。 这 意味着， 你 不需要 前面 加上 倒 斜杠 转义.、\*、? 或() 字符。 例如， 字符 分类 将 匹配 数字 0 到 5 和 一个 句点。 你 不需要 将它 写成[ 0- 5.]。

注意：在 字符 分类 的 左方 括号 后加 上一个 插入 字符（^）， 就可以 得到“ 非 字符 类”。 非 字符 类 将 匹配 不在 这个 字符 类 中的 所有 字符。

# 正则匹配的开始与结束 ^表示开始 $表示结束

表达式 r'^ Hello' 匹配 以' Hello' 开始 的 字符串。

表达式 r'\ d$' 匹配 以 数字 0 到 9 结束 的 字符串。

表达式 r'^\ d+$' 匹配 从 开始 到 结束 都是 数字 的 字符串。

# 通配符.\*(点号星号)--.\*贪婪--.\*?非贪婪

.（ 句点） 字符 称为“ 通配符”。 它 匹配 除了 换行 之外 的 所有 字符。

记住， 句点 字符 只 匹配 一个 字符， 可 以用 点- 星（.\*） 表示“ 任意 文本”。

同样的，用加问号的方式匹配贪心与非贪心模式

>>> nongreedyRegex = re. compile( r'<.\*?>')

>>> mo = nongreedyRegex. search('< To serve man> for dinner.>')

>>> mo. group()

'< To serve man>'

>>> greedyRegex = re. compile( r'<.\*>')

>>> mo = greedyRegex. search('< To serve man> for dinner.>')

>>> mo. group()

'< To serve man> for dinner.>'

# 通配符.\*(点号星号)匹配

点- 星 将 匹配 除 换行 外 的 所有 字符。 通过 传入 re. DOTALL 作为 re. compile() 的 第二个 参数， 可以 让 句点 字符 匹配 所有 字符， 包括 换行 字符。

.\*不匹配换行符（\n）

>>> noNewlineRegex = re. compile('.\*')

>>> noNewlineRegex. search(' Serve the public trust.\ nProtect the innocent. \nUphold the law.'). group()

'Serve the public trust.'

.\*跨行匹配

>>> newlineRegex = re. compile('.\*', re. DOTALL)

>>> newlineRegex. search(' Serve the public trust.\ nProtect the innocent. \nUphold the law.'). group()

'Serve the public trust.\ nProtect the innocent.\ nUphold the law.'

# 正则表达式匹配次数

?匹配 零 次 或 一次 前面 的 分组。

\*匹配 零 次 或 多次 前面 的 分组。

+ 匹配 一次 或 多次 前面 的 分组。

{n} 匹配 n 次 前面 的 分组。

{n,} 匹配 n 次 或 更多 前面 的 分组。

{,m} 匹配 零 次 到 m 次 前面 的 分组。

{n, m} 匹配 至少 n 次、 至多 m 次 前面 的 分组。

{n, m}? 或\*? 或+? 对 前面 的 分组 进行 非 贪心 匹配。

^ spam 意味着 字符串 必须 以 spam 开始。

spam$ 意味着 字符串 必须 以 spam 结束。

.匹配 所有 字符， 换行符 除外。

\d、\ w 和\ s 分别 匹配 数字、 单词 和 空格。

\D、\ W 和\ S 分别 匹配 出 数字、 单词 和 空 格外 的 所有 字符。

[abc] 匹配 方 括号 内 的 任意 字符（ 诸如 a、 b 或 c）。

[^abc] 匹配 不在 方 括号 内 的 任意 字符。

# 正则匹配替换--sub()方法

>>> namesRegex = re. compile( r' Agent \w+')

>>> namesRegex. sub(' CENSORED', 'Agent Alice gave the secret documents to Agent Bob.')

'CENSORED gave the secret documents to CENSORED.'

# python去掉字符串前后的空格

1.strip()：把头和尾的空格去掉

2.lstrip()：把左边的空格去掉

3.rstrip()：把右边的空格去掉

4.replace('c1','c2')：把字符串里的c1替换成c2。故可以用replace(' ','')来去掉字符串里的所有空格

5.split()：通过指定分隔符对字符串进行切片，如果参数num 有指定值，则仅分隔 num 个子字符串

>>> a=' ddd dfe dfd efre ddd '

>>> a

' ddd dfe dfd efre ddd '

>>> a.strip()

'ddd dfe dfd efre ddd'

>>> a.lstrip()

'ddd dfe dfd efre ddd '

>>> a.rstrip()

' ddd dfe dfd efre ddd'

>>> a.replace(' ','')

'ddddfedfdefreddd'

>>> a.split()

['ddd', 'dfe', 'dfd', 'efre', 'ddd']

>>> a = 'dfdfd\*dfjdf\*\*fdjfd\*22\*'

>>> a

'dfdfd\*dfjdf\*\*fdjfd\*22\*'

>>> a.split('\*')

['dfdfd', 'dfjdf', '', 'fdjfd', '22', '']

>>> a.split('\*',2)

['dfdfd', 'dfjdf', '\*fdjfd\*22\*']

 6.使用正则表达式

>>> re.split(r'\s+', 'a b c')

['a', 'b', 'c']

# 34.使用匹配的文本本身，作为替换的一部分

\w代表匹配一个 大小写字母或数字或下划线，只匹配这三种

需要 使用 匹配 的 文本 本身， 作为 替换 的 一部分。 在 sub() 的 第一个 参数 中， 可以 输入\ 1、\ 2、\ 3……。 表示“ 在 替换 中 输入 分组 1、 2、 3…… 的 文本”。

例如， 假定 想要 隐去 密探 的 姓名， 只 显示 他们 姓名 的 第一个 字母。 要做 到这 一点， 可以 使用 正 则 表达式 Agent (\w)\ w， 传入 r'\ 1\*' 作为 sub() 的 第一个 参数。 字符串 中的\ 1 将由 分组 1 匹配 的 文本 所 替代， 也就是 正 则 表达式 的(\ w) 分组。

>>> agentNamesRegex = re. compile( r' Agent (\w)\ w\*')

>>> agentNamesRegex. sub( r'\ 1\*\*\*\*', 'Agent Alice told Agent Carol that Agent Eve knew Agent Bob was a double agent.')

A\*\*\*\* told C\*\*\*\* that E\*\*\*\* knew B\*\*\*\* was a double agent.'

# 35.复杂的正则表达式可以拆分多行

管理 复杂 的 正 则 表达式 如果 要 匹配 的 文本 模式 很 简单， 正 则 表达式 就很 好。 但 匹配 复杂 的 文本 模式， 可能 需要 长的、 费解 的 正 则 表达式。 你 可以 告诉 re. compile()， 忽略 正 则 表达式 字符串 中的 空白 符 和 注释， 从而 缓解 这一点。 要 实现 这种 详细 模式， 可以向 re. compile() 传入 变量 re. VERBOSE， 作为 第二个 参数。 现在， 不必 使用 这样 难以 阅读 的 正 则 表达式：

phoneRegex = re. compile( r'((\ d{ 3}|\(\ d{ 3}\))?(\ s|-|\.)?\ d{ 3}(\ s|-|\.)\ d{ 4} (\s\*( ext| x| ext.)\ s\*\ d{ 2, 5})?)')

你 可以 将 正 则 表达式 放在 多 行中， 并 加上 注释， 像这样：

phoneRegex = re. compile( r'''(

(\d{ 3}|\(\ d{ 3}\))? # area code

(\s|-|\.)? # separator \d{ 3} # first 3 digits

(\s|-|\.) # separator

\d{ 4} # last 4 digits

(\s\*( ext| x| ext.)\ s\*\ d{ 2, 5})? # extension

)''', re. VERBOSE)

# 多种模式匹配

组合 使用 re. IGNOREC ASE（忽略大小写）、 re. DOTALL（.号匹配换行） 和 re. VERBOSE （多行正则表达式）

如果 希望 正 则 表达式 不 区分 大 小写， 并且 句点 字符 匹配 换行， 就可以 这样 构造 re. compile() 调用：

>>> someRegexValue = re. compile(' foo', re. IGNORECASE | re. DOTALL)

# 获取当前路径与改变路径

>>> import os >>> os. getcwd()

'C:\\ Python34'

>>> os. chdir(' C:\\ Windows\\ System32')

>>> os. getcwd()

'C:\\ Windows\\ System32'

创建路径

>>> import os

>>> os. makedirs(' C:\\ delicious\\ walnut\\ waffles')

# 处理绝对路径与相对路径

### 调用 os. path. abspath( path) 将 返回 参数 的 绝对 路径 的 字符串。 这是 将相 对路 径 转换 为 绝对 路径 的 简便 方法。

### 调用 os. path. isabs( path)， 如果 参数 是一 个 绝对 路径， 就 返回 True， 如果 参数 是一 个 相对 路径， 就 返回 False。

### 调用 os. path. relpath( path, start) 将 返回 从 start 路径 到 path 的 相对 路径 的 字符串。 如果 没有 提供 start， 就 使用 当前工作 目录 作为 开始 路径。

>>> os. path. abspath('.')

'C:\\ Python34'

>>> os. path. abspath('.\\ Scripts')

'C:\\ Python34\\ Scripts'

>>> os. path. isabs('.')

False

>>> os. path. isabs( os. path. abspath('.'))

True

>>> os. path. relpath(' C:\\ Windows', 'C:\\')

'Windows'

>>> os. path. relpath(' C:\\ Windows', 'C:\\ spam\\ eggs')

'..\\..\\Windows'

>>> os. getcwd()

'C:\\ Python34'

### 调用 os. path. dirname( path) 将 返回 一个 字符串， 它 包含 path 参数 中最 后 一个 斜杠 之前 的 所有 内容。

### 调用 os. path. basename( path) 将 返回 一个 字符串， 它 包含 path 参数 中最 后 一个 斜杠 之后 的 所有 内容。

>>> path = 'C:\\ Windows\\ System32\\ calc. exe'

>>> os. path. basename( path)

'calc. exe'

>>> os. path. dirname( path)

'C:\\ Windows\\ System32'

### 调用 os. path. split()， 获得 这 两个 字符串 的 元 组

>>> calcFilePath = 'C:\\ Windows\\ System32\\ calc. exe'

>>> os. path. split( calcFilePath)

('C:\\ Windows\\ System32', 'calc. exe')

### 系统分隔符为os.path.sep返回当前系统的分隔符

>>> calcFilePath. split( os. path. sep)

['C:', 'Windows', 'System32', 'calc. exe']

>>> '/usr/ bin'. split( os. path. sep)

['', 'usr', 'bin']

# 检查文件与路径是否存在

>>> os. path. exists(' C:\\ Windows')

True

>>> os. path. exists(' C:\\ some\_ made\_ up\_ folder')

False

>>> os. path. isdir(' C:\\ Windows\\ System32')

True

>>> os. path. isfile(' C:\\ Windows\\ System32')

False

>>> os. path. isdir(' C:\\ Windows\\ System32\\ calc. exe')

False

>>> os. path. isfile(' C:\\ Windows\\ System32\\ calc. exe')

True

# 打开一个文件

(注意：读写文件的一般流程1.调用open()函数返回file对象2.调用File对象的read()或者write()方法读或写文件 3.调用File对象的close()方法关闭该文件)

Open()函数第二个参数为打开模式分别为（r-读模式-默认,w-写模式-覆盖,a-追加模式-不会覆盖在原文件末尾添加内容）缺省第二个参数，默认为r-读模式不能写

>>> helloFile = open(' C:\\ Users\\\_ your\_ home\_ folder\_\\ hello. txt')

如果 使用 OS X， 在 交互式 环境 中 输入 以下 代码：

>>> helloFile = open('/ Users/\_ your\_ home\_ folder\_/ hello. txt')

>>> helloContent = helloFile. read()

>>> helloContent

'Hello world!'

# 41.读文本文件

### 把文件所有内容讲到一个字符串变量中（文件对象的.read()方法）

如果 使用 OS X， 在 交互式 环境 中 输入 以下 代码：

>>> helloFile = open('/ Users/\_ your\_ home\_ folder\_/ hello. txt')

>>> helloContent = helloFile. read()

>>> helloContent

'Hello world!'

# 只读取一行（当文件比较大，内存不足时才用，因为读取效率比较慢）

>>> helloContent = helloFile.readline()

### 把文件读到一个字符串列表中（注意每一行存入一个元素，以\n换行符结尾-文件对象的. readlines()方法--处理时最好用这个）

文本内容：

When, in disgrace with fortune and men' s eyes,

I all alone beweep my outcast state,

And trouble deaf heaven with my bootless cries,

And look upon myself and curse my fate,

代码：

>>> sonnetFile = open(' sonnet29. txt')

>>> sonnetFile. readlines()

# 写文本文件（注意：打开方式要指定第二个参数为w覆盖写方式或者a追加写方式）

>>> baconFile = open(' bacon. txt', 'w')

>>> baconFile. write(' Hello world!\ n')

13

>>> baconFile. close()

>>> baconFile = open(' bacon. txt', 'a')

>>> baconFile. write(' Bacon is not a vegetable.')

25

>>> baconFile. close()

# 保存python变量到文件，服务器重启后仍然可读取变量（要先导入shelve包，再用shelve.open(‘路径’)获取shelve文件对象）

>>> import shelve

>>> shelfFile = shelve. open(' mydata')

>>> type( shelfFile)

< class 'shelve. DbfilenameShelf'>

>>> shelfFile[' cats']

['Zophie', 'Pooka', 'Simon']

>>> shelfFile. close()

>>> shelfFile = shelve. open(' mydata')

>>> list( shelfFile. keys())

['cats']

>>> list( shelfFile. values())

[['Zophie', 'Pooka', 'Simon']]

>>> shelfFile. close()

# 用 pprint. pformat() 函数 保存 变量,并且将变量保存到py文件中，其它程序导入该py文件，并读取变量

>>> import pprint

>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]

>>> pprint. pformat( cats) "[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"

>>> fileObj = open(' myCats. py', 'w')

>>> fileObj. write(' cats = ' + pprint. pformat( cats) + '\n')

83

>>> fileObj. close()

## 另一个文件导入myCats

>>> import myCats

>>> myCats. cats [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]

>>> myCats. cats[ 0] {'name': 'Zophie', 'desc': 'chubby'}

>>> myCats. cats[ 0][' name']

'Zophie'

# 总结：该方式适合保存基本数据类型，可以用文本编辑器查看保存的内容，其它类型例如File文件只能保存到shevle，基本数据类型也可以保存到shevle.

# 文件操作

shutil 模块用于复制文件或者文件夹

## 单个文件复制

### shutil. copy(' source', 'destination') 如果目标路径不是目录，是一个文件，则相当于复制并重命名，返回值为拷贝文件的新路径

>>> import shutil, os

>>> os. chdir(' C:\\')

>>> shutil. copy(' C:\\ spam. txt', 'C:\\ delicious')

' C:\\ delicious\\ spam. txt'

>>> shutil. copy(' eggs. txt', 'C:\\ delicious\\ eggs2. txt')

' C:\\ delicious\\ eggs2. txt'

# 复制目录

### Shutil.copytree(source,destination)将 路径 source 处 的 文件夹， 包括 它的 所有 文件 和 子 文件夹， 复制 到 路径 destination 处 的 文件夹。 source 和 destination 参数 都是 字符串。 返回新文件夹路径

>>> import shutil, os

>>> os. chdir(' C:\\')

>>> shutil. copytree(' C:\\ bacon', 'C:\\ bacon\_ backup')

'C:\\ bacon\_ backup'

# 文件或目录移动

### shutil. move( source, destination)，source可以是文件或者是目录，destination如果是一个新的文件地址，如果与原名不一致则相当于改名，如果现在文件夹中有同名文件会覆盖

>>> import shutil

>>> shutil. move(' C:\\ bacon. txt', 'C:\\ eggs')

'C:\\ eggs\\ bacon. txt'

# 永久删除文件或者目录

用 os. unlink( path) 将 删除 path 处 的 文件。

调用 os. rmdir( path) 将 删除 path 处 的 文件夹。 该 文件夹 必须 为 空， 其中 没有 任何 文件 和 文件夹。

调用 shutil. rmtree( path) 将 删除 path 处 的 文件夹， 它 包含 的 所有 文件 和 文件夹 都会 被 删除。

# 删除文件到回收站

>>> import send2trash

>>> baconFile = open(' bacon. txt', 'a') # creates the file

>>> baconFile. write(' Bacon is not a vegetable.')

25

>>> baconFile. close()

>>> send2trash. send2trash(' bacon. txt')

# 遍历目录os.walk(path)

for folderName, subfolders, filenames in os. walk(' C:\\ delicious'):

print(' The current folder is ' + folderName)

for subfolder in subfolders:

print(' SUBFOLDER OF ' + folderName + ': ' + subfolder)

for filename in filenames:

print(' FILE INSIDE ' + folderName + ': '+ filename)

print('')

# 开始调试

抛出异常

>>>raise Exception(' This is the error message.')

捕获异常并输出错误信息：

**try**:  
 boxPrint(sym,w,h)  
**except** Exception **as** err:  
 print(**'A exception happpended:'**+str(err))

输出结果：

A exception happpended:Width must be greater than 2.

A exception happpended:Symbol must be a single character string.

# 51.把错误信息保存到文件

**先导入traceback包，再用**traceback.format\_exc()方法获取追踪异常的信息

**import** traceback

**def** saveTheExceptionInfoIntoAFile():  
 **try**:  
 **raise** Exception(**' This is the error message.'**)  
 **except**:  
 errorFile = open(**' errorInfo. txt'**, **'w'**)  
 errorFile. write( traceback.format\_exc())  
 errorFile. close()  
 print(**' The traceback info was written to errorInfo. txt.'**)

# 断言

代码格式：

assert 关键字； 条件（ 即 求值 为 True 或 False 的 表达式）； 逗号； 当 条件 为 False 时 显示 的 字符串。

例如：

assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'

代码：

>>> podBayDoorStatus = 'open'

>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'

>>> podBayDoorStatus = 'I\' m sorry, Dave. I\' m afraid I can' t do that.''

>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'

控制台：

C:\python\python.exe E:/tensor/pytest/pytest/dataStructure/10exception.py

Traceback (most recent call last):

File "E:/tensor/pytest/pytest/dataStructure/10exception.py", line 62, in <module>

assertForTest()

File "E:/tensor/pytest/pytest/dataStructure/10exception.py", line 61, in assertForTest

assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'

AssertionError: The pod bay doors need to be "open".

Process finished with exit code 1

# 禁用断言

Pass

待完善

# 日志

先要导包：

**import** logging  
logging. basicConfig( level= logging.DEBUG, format=**' %(asctime) s - %(levelname) s - %(message) s'**)

**def** testLogging():  
 logging.debug(**'Some debugging details.'**)  
 logging.info(**'The logging module is working.'**)  
 logging.warning(**'An error message is about to be logged.'**)  
 logging.error(**'An error has occurred.'**)  
 logging.critical(**'The program is unable to recover!'**)  
*testLogging()*

控制台：

C:\python\python.exe E:/tensor/pytest/pytest/dataStructure/10exception.py

2017-11-08 15:17:39,789 - DEBUG - Some debugging details.

2017-11-08 15:17:39,789 - INFO - The logging module is working.

2017-11-08 15:17:39,789 - WARNING - An error message is about to be logged.

2017-11-08 15:17:39,789 - ERROR - An error has occurred.

2017-11-08 15:17:39,789 - CRITICAL - The program is unable to recover!

Process finished with exit code 0

# 禁止输出日志信息

只 要向 logging. disable() 传入 一个 日志 级别， 它 就会 禁止 该 级别 和 更低 级别 的 所有 日志 消息。 所以， 如果 想要 禁用 所有 日志， 只要 在 程序 中 添加 logging. disable（ logging.CRITICAL）。

级别顺序：

CRITICAL（最高）->ERROR->WARNING->INFO->DEBUG(最低)

>>> import logging

>>> logging. basicConfig( level= logging. INFO, format=' %(asctime) s - %(levelname) s - %(message) s')

>>> logging. critical(' Critical error! Critical error!')

2015- 05- 22 11: 10: 48, 054 - CRITICAL - Critical error! Critical error!

>>> logging. disable( logging. CRITICAL)

>>> logging. critical(' Critical error! Critical error!')

>>> logging. error(' Error! Error!')

注意：logging. disable( logging. CRITICAL) 放到import logging下面，所有日志都不会打印

# 把日志输出到文件（指定filename）

import logging

logging. basicConfig( filename=' myProgramLog. txt', level= logging. DEBUG, format=' %(asctime) s - %(levelname) s - %(message) s')

# 从web爬取信息

用系统默认浏览器打开指定网址：（webbrowser.open(**'https://www.baidu.com'**)）

代码：

**import** webbrowser,sys,pyperclip  
*#testkey，need copy into the clipboard:cjk--yourname***def** openWebbrowserWithUrlInTheClipBoard():  
 word=pyperclip.paste()  
 **if** len(word) > 1:  
 webbrowser.open(**'https://www.baidu.com/s?word='**+word)  
*openWebbrowserWithUrlInTheClipBoard()*

# 通过requests库获取指定文本

**先导入requests库再用**requests.get(**'http://www.gutenberg.org/cache/epub/1112/pg1112.txt'**)获取Response

<class 'requests.models.Response'>  
代码示例：

**import** requests  
**def** requestsATXTFile():  
 res=requests.get(**'http://www.gutenberg.org/cache/epub/1112/pg1112.txt'**)  
 print(type(res))  
 print(len(res.text))  
 print(res.text[:250])  
 res=requests.get(**'http://www.baidu.com'**)  
 print(res.text[:100])  
*#requestsATXTFile()*

控制台：

C:\python\python.exe E:/tensor/pytest/pytest/dataStructure/11crawlFromWeb.py

<class 'requests.models.Response'>

178981

﻿The Project Gutenberg EBook of Romeo and Juliet, by William Shakespeare

This eBook is for the use of anyone anywhere at no cost and with

almost no restrictions whatsoever. You may copy it, give it away or

re-use it under the terms of the Proje

<!DOCTYPE html>

<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charse

Process finished with exit code 0

# 防止下载失败的方法（或者返回错误状态码）抛出异常

**使用**requests.models.Response对象的res.raise\_for\_status()方法 ，当没有正常访问地址时抛出异常

**def** catchRequestsException():  
 res=requests.get(**'http://inventwithpython.com/page\_that\_does\_not\_exist'**)  
 **try**:  
 res.raise\_for\_status()  
 **except** Exception **as** exc:  
 print(**'There are a problem:%s'** % (exc))  
*#catchRequestsException()*

# 访问指定地址并保存到指定文件

res.iter\_content(100000)获得指定的Unicode文本

**def** saveResponseTextIntoAFile():  
 res=requests.get(**'http://www.gutenberg.org/cache/epub/1112/pg1112.txt'**)  
 res.raise\_for\_status()  
 playFile=open(**'RomeoAndJuliet.txt'**,**'wb'**)  
 **for** chunk **in** res.iter\_content(100000):  
 playFile.write(chunk)  
saveResponseTextIntoAFile()

说明：

iter\_ content() 方法 在 循环 的 每次 迭代 中， 返回 一段 内容。 每一 段 都是 bytes 数据 类型， 你 需要 指定 一段 包含 多少 字节。 10 万字 节 通常 是 不错 的 选择， 所以 将 100000 作为 参数 传递 给 iter\_ content()。

write() 方法 返回 一个 数字， 表示 写入 文件 的 字节数。 在前

回顾 一下， 下载 并 保存 到 文件 的 完整 过程 如下：

1. 调用 requests. get() 下载 该 文件。
2. 用' wb' 调用 open()， 以 写 二进制 的 方式 打开 一个 新 文件。
3. 利用 Respose 对象 的 iter\_ content() 方法 做 循环。
4. 在 每次 迭代 中 调用 write()， 将 内容 写入 该 文件。
5. 调用 close() 关闭 该 文件。

# 用Bs4模块解析html文件

先导入库

**import** bs4

从网络获取html文本：bs4.BeautifulSoup(res.text,**"html.parser"**)

res=requests.get(**'https://www.baidu.com'**)  
res.raise\_for\_status()  
nosStarchSoup=bs4.BeautifulSoup(res.text,**"html.parser"**)  
print(type(nosStarchSoup))

从html文件获取：

exampleFile=open(**'example.html'**)  
exampleSoup=bs4.BeautifulSoup(exampleFile,**"html.parser"**)  
print(type(exampleSoup))

# 常用select()查找页面元素的方法

传递 给 select() 方法 的 选择器 将 匹配…

soup. select(' div') 所有 名为< div> 的 元素

soup. select('# author') 带有 id 属性 为 author 的 元素

soup. select('. notice') 所有 使用 CSS class 属性 名为 notice 的 元素

soup. select(' div span') 所有 在< div> 元素 之内 的< span> 元素

soup. select(' div > span') 所有 直接 在< div> 元素 之内 的< span> 元素， 中间 没有 其他 元素

soup. select(' input[ name]') 所有 名为< input>， 并有 一个 name 属性， 其 值 无所谓 的 元素

soup. select(' input[ type=" button"]') 所有 名为< input>， 并有 一个 type 属性， 其 值 为 button 的 元素

# 解析html，根据id或标签查找元素

与js类似，获取指定id的元素的方法为：

exampleSoup=bs4.BeautifulSoup(exampleFile,**"html.parser"**)  
 elems=exampleSoup.select(**'#author'**)  
元素的类型为：<class 'bs4.element.Tag'>

获取元素的文本：elems[0].get\_text()

获取元素的属性（封装为字典）：print(str(elems[0].attrs))  
{'id': 'author'}

获取指定元素：

代码：

**def** findTheElementFromHtmlByBs4WithSelectMethod():  
 createAHTMLFile() *# create html file* exampleFile=open(**'example.html'**)  
 exampleSoup=bs4.BeautifulSoup(exampleFile,**"html.parser"**)  
 elems=exampleSoup.select(**'#author'**)  
 print(**'1--'**,type(elems))  
 print(**'2--'**,len(elems))  
 print(**'3--'**,type(elems[0]))  
 print(**'4--'**,elems[0].get\_text())  
 print(**'5--'**,str(elems[0]))  
 print(**'6--'**,str(elems[0].attrs))  
 pElems=exampleSoup.select(**'p'**)  
 print(**'7--'**,str(pElems[0]))  
 print(**'8--'**,pElems[0].getText())  
 print(**'9--'**, pElems[0].get\_text())  
findTheElementFromHtmlByBs4WithSelectMethod()

控制台：

C:\python\python.exe E:/tensor/pytest/pytest/dataStructure/11crawlFromWeb.py

1-- <class 'list'>

2-- 1

3-- <class 'bs4.element.Tag'>

4-- Al Sweigart

5-- <span id="author"> Al Sweigart</span>

6-- {'id': 'author'}

7-- <p> Download my <strong> Python</strong> book from <a href="http://inventwithpython.com"> my website</a>.</p>

8-- Download my Python book from my website.

9-- Download my Python book from my website.

Process finished with exit code 0

# 用selenium模块控制浏览器

方 法名 返回 的 WebElement 对象/ 列表

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

browser. find\_ element\_ by\_ class\_ name( name)

browser.find\_ elements\_ by\_ class\_ name( name) 使用 CSS 类 name 的 元素

browser. find\_ element\_ by\_ css\_ selector( selector)

browser.find\_ elements\_ by\_ css\_ selector( selector) 匹配 CSS selector 的 元素

browser. find\_ element\_ by\_ id( id)

browser.find\_ elements\_ by\_ id( id) 匹配 id 属性 值 的 元素

browser. find\_ element\_ by\_ link\_ text( text)

browser. find\_ elements\_ by\_ link\_ text( text) 完全 匹配 提供 的 text 的< a> 元素

browser. find\_ element\_ by\_ partial\_ link\_ text( text)

browser. find\_ elements\_ by\_ partial\_ link\_ text( text) 包含 提供 的 text 的< a> 元素

browser. find\_ element\_ by\_ name( name)

browser. find\_ elements\_ by\_ name( name) 匹配 name 属性 值 的 元素

browser. find\_ element\_ by\_ tag\_ name( name)

browser. find\_ elements\_ by\_ tag\_ name( name) 匹配 标签 name 的 元素 (大小 写 无关，< a> 元素 匹配' a' 和' A')

**def** openUrlBySeleniumWithFireFox():  
 browser=webdriver.Firefox()  
 type(browser)  
 browser.get(**'http://www.baidu.com'**)  
openUrlBySeleniumWithFireFox()

# WebElement方法

属性 或 方法 描述

tag\_ name 标签 名， 例如 'a' 表示< a> 元素

get\_ attribute( name) 该 元素 name 属 性的 值 text 该 元素 内 的 文本， 例如< span> hello< /span> 中的' hello'

clear() 对于 文本 字段 或 文本 区域 元素， 清除 其中 输入 的 文本

is\_ displayed() 如果 该 元素 可见， 返回 True， 否则 返回 False

is\_ enabled() 对于 输入 元素， 如果 该 元素 启用， 返回 True， 否则 返回 False

is\_ selected() 对于 复选框 或 单选 框 元素， 如果 该 元素 被选 中， 选择 True， 否则 返回 False

location 一个 字典， 包含 键' x' 和' y'， 表示 该 元素 在 页 面上 的 位置

获取一个元素：

**def** submitFormWithInputSend\_keys():  
 browser=webdriver.Firefox()  
 browser.get(**'http://www.baidu.com'**)  
 keyWordElem=browser.find\_element\_by\_id(**'kw'**)  
 keyWordElem.send\_keys(**'Trump'**)  
 buttonElem=browser.find\_element\_by\_id(**'su'**)  
 buttonElem.click()  
*#submitFormWithInputSend\_keys()*

### 操作浏览器web元素发送特殊键（键盘上不显示字符的键，如回车，退格，方向键等）

属性 含义

Keys. DOWN, Keys. UP, Keys. LEFT, Keys. RIGHT 键盘 箭头 键

Keys. ENTER, Keys. RETURN 回车 和 换行 键

Keys. HOME, Keys. END, Keys. PAGE\_ DOWN, Keys. PAGE\_ UP Home 键、 End 键、 PageUp 键 和 Page Down 键

Keys. ESCAPE, Keys. BACK\_ SPACE, Keys. DELETE Esc、 Backspace 和 字母 键

Keys. F1, Keys. F2, . . . , Keys. F12 键盘 顶部 的 F1 到 F12 键

Keys. TAB Tab 键

**def** sendKeysWithHomeAndEnd():  
 browser=webdriver.Firefox()  
 browser.get(**'http://www.qq.com'**)  
 htmlElem=browser.find\_element\_by\_tag\_name(**'html'**)  
 htmlElem.send\_keys(Keys.END)  
 htmlElem.send\_keys(Keys.HOME)  
 browser.back()#退一页  
 browser.forward()#前进一页  
 browser.refresh()#刷新  
 browser.quit()#浏览器退出  
sendKeysWithHomeAndEnd()