# 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

>>> 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')