了解 python

1. 了解 Python
Python 是一种解释型（这意味着开发过程中没有了编译这个环节）、面向对象（支持面向对象的风格或代码封装在对象的编程技术）、动态数据类型的交互式（可在命令行中通过 Python 提示符及直接代码执行程序）高级程序设计语言。

2. Python 标识符
标识符由字母、数字、下划线组成，但不能以数字开头，且区分大小写。

以下划线开头的标识符是有特殊意义的。以单下划线开头 _foo 的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用 from xxximport * 而导入；

以双下划线开头的 __foo 代表类的私有成员 ;以双下划线开头和结尾的 __foo__ 代表 Python 里特殊方法专用的标识，如__init__() 代表类的构造函数。

Python 可以同一行显示多条语句，方法是用分号 ; 分开。

3. Python 中的保留字符
这些保留字不能用作常数或变数，或任何其他标识符名称。所有 Python 的关键字只包含小写字母。

and

exec

not

assert

finally

or

break

for

pass

class

from

print

continue

global

raise

def

if

return

del

import

try

elif

in

while

else

is

with

except

lambda

yield

4. Python 的缩进与多行语句写法

4.1 Python 中的缩进要求非常严格，必须严格对齐。因为 Python 的代码块不由{}控制，而是由缩进就可以控制。

4.2 使用斜杠（ \）将一行的语句分为多行显示，如下所示：当然，使用各类括号括起语句

之后不需要斜杠换行。

```
total = item_one + \
        item_two + \
    item_three
```

英文三个双引号用在等式中也可以写多行文本，直接使用和三个单引号作用一样，可以做多行注释。

5. Python 中的不换行与换行输出

```
# 换行输出
print(x)
# 不换行输出
print(x,end='')
```

Python 变量类型

创建变量就意味着会在内存中开辟空间，同时变量必须赋值才会被创建。

1. Python 中的变量赋值

单变量赋值：

```
counter = 100#赋值整型变量
miles = 1000.0#浮点型
name = "John"#字符串
```

多变量赋值：

```
a = b = c= 1 #三个变量的值都为 1
a, b, c =1, 2, "john"  #分别为三个变量赋值
```

2. Python 中的数据类型

Python 有五个标准的数据类型：

1. Numbers（数字）
1）不可改变的数据类型：
当其类型被改变时，将会赋值给一个新的对象。当对变量赋予了数值后，这个对象就会被创建，可通过 del 语句删除对这些对象的引用。

2）Python 支持的数字类型：
int（有符号整型，如 0x69，10）；long（长整型[也可以代表八进制和十六进制]，如-4721885298529L，Python 用数字后面的 L 表示长整型）；float（浮点型，如 70.2E-12）；complex（复数，如 4.53e-7j）。

3）Python 数据类型转换：
int(x [,base]) 将 x 转换为一个整数
long(x [,base]) 将 x 转换为一个长整数
float(x) 将 x 转换到一个浮点数
complex(real [,imag]) 创建一个复数
str(x) 将对象 x 转换为字符串
repr(x) 将对象 x 转换为表达式字符串
eval(str) 用来计算在字符串中的有效 Python 表达式,并返回一个对象
tuple(s) 将序列 s 转换为一个元组
list(s) 将序列 s 转换为一个列表
chr(x) 将一个整数转换为一个字符
unichr(x) 将一个整数转换为 Unicode 字符
ord(x) 将一个字符转换为它的整数值
hex(x) 将一个整数转换为一个十六进制字符串
oct(x) 将一个整数转换为一个八进制字符串
4）Python 数学函数
函数

返回值（ 描述 ）

abs(x)

返回数字的绝对值，如 abs(-10) 返回 10

ceil(x)

返回数字的向上取整值，如 math.ceil(4.1) 返回 5

cmp(x, y)

比较函数，如果 x < y 返回 -1，如果 x == y 返回 0，如果 x > y 返回 1

exp(x)

返回 e 的 x 次幂(ex),如 math.exp(1) 返回 2.718281828459045

fabs(x)

返回数字的绝对值，如 math.fabs(-10) 返回 10.0

floor(x)

返回向下取整值，如 math.floor(4.9)返回 4

log(x)

如 math.log(math.e)返回 1.0,math.log(100,10)返回 2.0

log10(x)

返回以 10 为基数的 x 的对数，如 math.log10(100)返回 2.0

max(x1, x2,...)

返回给定参数的最大值，参数可以为序列。

min(x1, x2,...)

返回给定参数的最小值，参数可以为序列。

modf(x)

返回 x 的整数部分与小数部分，两部分的数值符号与 x 相同，整数部分以浮点型表示。

pow(x, y)

x**y 运算后的值。

round(x [,n])

返回浮点数 x 的四舍五入值，如给出 n 值，则代表舍入到小数点后的位数。

sqrt(x)

返回数字 x 的平方根

5）Python 随机数函数

常用于游戏、数学、安全等领域。

函数

描述

choice(seq)

从序列的元素中随机挑选一个元素，比如 random.choice(range(10))，从 0 到 9 中随机挑选一个整数。

randrange ([start,] stop [,step])

从指定范围内，按指定基数递增的集合中获取一个随机数，基数缺省值为 1

random()

随机生成下一个实数，它在[0,1)范围内。

seed([x])

改变随机数生成器的种子 seed。如果你不了解其原理，你不必特别去设定 seed，Python 会帮你选择 seed。

shuffle(lst)

将序列的所有元素随机排序

uniform(x, y)

随机生成下一个实数，它在[x,y]范围内。

6） Python 三角函数

函数

描述

acos(x)

返回 x 的反余弦弧度值。

asin(x)

返回 x 的反正弦弧度值。

atan(x)

返回 x 的反正切弧度值。

atan2(y, x)

返回给定的 X 及 Y 坐标值的反正切值。

cos(x)

返回 x 的弧度的余弦值。

hypot(x, y)

返回欧几里德范数 sqrt(x*x + y*y)。

sin(x)

返回的 x 弧度的正弦值。

tan(x)

返回 x 弧度的正切值。

degrees(x)

将弧度转换为角度,如 degrees(math.pi/2) ， 返回 90.0

radians(x)

将角度转换为弧度

7) Python 数学常量

常量	描述
pi	数学常量 pi（圆周率，一般以 π 来表示）
e	

数学常量 e，e 即自然常数（自然常数）。

2. String（字符串）
由数字、字母、下划线组成。

1） 字符串截取
Python 字符串从左至右截取：索引范围（0，长度-1），从右至左截取（-1，字符串开头）。

2） Python 中不存在单字符
Python 中即使有单字符，也会被当作字符串处理。

3） Python 转义字符
转义字符

描述

\

出现在行尾时表现为续行符，出现在行中时，用于“翻译”特殊字符表示特殊含义，如下面选项所示

\\

反斜杠符号

\'

单引号

\"

双引号

\a

响铃

\b

退格(Backspace)

\e

转义

\000

空

\n

换行

\v

纵向制表符

\t

横向制表符

\r

回车

\f

换页

\oyy

八进制数，yy 代表的字符，例如：\o12 代表换行

\xyy

十六进制数，yy 代表的字符，例如：\x0a 代表换行

\other

其它的字符以普通格式输出

4） Python 字符串运算
下表实例变量 a 值为字符串 "Hello"，b 变量值为 "Python"：

操作符

描述

+

字符串连接。

*

重复输出字符串。

[]

通过索引获取字符串中字符

[:]

截取字符串中的一部分

in

成员运算符 - 如果字符串中包含给定的字符返回 True

not in

成员运算符 - 如果字符串中不包含给定的字符返回 True

r/R

原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母"r"（可以大小写）以外，与普通字符串有着几乎完全相同的语法。

a='hello'

```
b='world'
print(a+b)                #helloworld，+号连接字符串
print(a*2)                #hellohello，*号重复字符串
print(a[1])               #e，[]索引字符
print(a[1:4])             #ell，[:]截取字符串
print("h" in a)          #True，in 是否包含
print("M" not in a) #True，not in 是否不包含
print(r'\n')              #\n，r 原始字符串（不解析转义字符）
```

5） Python 字符串格式化（方便 print 时定义类型，如 C 语言中 printf 字符串时在引号内写%s 一样）

```
print("My name is %s and weight is %d kg!" % ('Zara', 21) )
```

输出：

```
My name is Zara and weight is 21 kg!
```

符 号	描述
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	

格式化无符号十六进制数（大写）

%f

格式化浮点数字，可指定小数点后的精度

%e

用科学计数法格式化浮点数

%E

作用同%e，用科学计数法格式化浮点数

%g

%f 和%e 的简写

%G

%f 和 %E 的简写

%p

用十六进制数格式化变量的地址

6）使用三引号输出一大串带特殊字符的字符串
当使用三引号将字符串框起来时，就不需要再通过转义字符打很多换行符等符号了，可以直接打换行。对比如下：

>>> hi = '''hi

there'''

>>> hi = 'hi\nthere'

这两种输出的结果都是换行的字符串，但是使用单引号时，当转义字符很多时，会很痛苦。

9） String 可使用的内建函数
方法

描述

string.capitalize()

把字符串的第一个字符大写

string.center(width)

返回一个原字符串居中,并使用空格填充至长度 width 的新字符串

string.count(str, beg=0, end=len(string))

返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数

string.decode(encoding='UTF-8', errors='strict')

以 encoding 指定的编码格式解码 string，如果出错默认报一个 ValueError 的异常，除非 errors 指定的是 'ignore' 或者'replace'

string.encode(encoding='UTF-8', errors='strict')

以 encoding 指定的编码格式编码 string，如果出错默认报一个 ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'

string.endswith(obj, beg=0, end=len(string))

检查字符串是否以 obj 结束，如果 beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.

string.expandtabs(tabsize=8)

把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8。

string.find(str, beg=0, end=len(string))

检测 str 是否包含在 string 中，如果 beg 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1

string.format()

格式化字符串

string.index(str, beg=0, end=len(string))

跟 find()方法一样，只不过如果 str 不在 string 中会报一个异常.

string.isalnum()

如果 string 至少有一个字符并且所有字符都是字母或数字则返

回 True,否则返回 False

string.isalpha()

如果 string 至少有一个字符并且所有字符都是字母则返回 True,

否则返回 False

string.isdecimal()

如果 string 只包含十进制数字则返回 True 否则返回 False.

string.isdigit()

如果 string 只包含数字则返回 True 否则返回 False.

string.islower()

如果 string 中包含至少一个区分大小写的字符, 并且所有这些(区分大小写的)字符都是小写, 则返回 True，否则返回 False

string.isnumeric()

如果 string 中只包含数字字符，则返回 True，否则返回 False

string.isspace()

如果 string 中只包含空格，则返回 True，否则返回 False.

string.istitle()

如果 string 是标题化的(见 title())则返回 True，否则返回 False

string.isupper()

如果 string 中包含至少一个区分大小写的字符, 并且所有这些(区分大小写的)字符都是大写, 则返回 True，否则返回 False

string.join(seq)

以 string 作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串

string.ljust(width)

返回一个原字符串左对齐,并使用空格填充至长度 width 的新字符串

string.lower()

转换 string 中所有大写字符为小写.

string.lstrip()

截掉 string 左边的空格

string.maketrans(intab, outtab])

maketrans() 方法用于创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。

max(str)

返回字符串 str 中最大的字母。

min(str)

返回字符串 str 中最小的字母。

string.partition(str)

有点像 find()和 split()的结合体,从 str 出现的第一个位置起,把字符串 string 分成一个 3 元素的元组 (string_pre_str,str,string_post_str),如果 string 中不包含 str 则 string_pre_str == string.

string.replace(str1, str2, num=string.count(str1))

把 string 中的 str1 替换成 str2,如果 num 指定，则替换不超过 num 次.

string.rfind(str, beg=0,end=len(string))

类似于 find()函数，不过是从右边开始查找.

string.rindex(str, beg=0,end=len(string))

类似于 index()，不过是从右边开始.

string.rjust(width)

返回一个原字符串右对齐,并使用空格填充至长度 width 的新字符串

string.rpartition(str)

类似于 partition()函数,不过是从右边开始查找.

string.rstrip()

删除 string 字符串末尾的空格.

string.split(str="", num=string.count(str))

以 str 为分隔符切片 string，如果 num 有指定值，则仅分隔 num 个子字符串

string.splitlines([keepends])

按照行('\r', '\r\n', \n')分隔, 返回一个包含各行作为元素的列表, 如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。

string.startswith(obj, beg=0,end=len(string))

检查字符串是否是以 obj 开头，是则返回 True，否则返回 False。如果 beg 和 end 指定值，则在指定范围内检查.

string.strip([obj])

在 string 上执行 lstrip()和 rstrip()

string.swapcase()

翻转 string 中的大小写

string.title()

返回"标题化"的 string,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())

string.translate(str, del="")

根据 str 给出的表(包含 256 个字符)转换 string 的字符,

要过滤掉的字符放到 del 参数中

string.upper()

转换 string 中的小写字母为大写

string.zfill(width)

返回长度为 width 的字符串，原字符串 string 右对齐，前面填充 0

string.isdecimal()

isdecimal()方法检查字符串是否只包含十进制字符。这种方法只存在于 unicode 对象。

3. List（列表）
使用非常频繁，支持数字、字符、字符串甚至列表的集合结构。

1） 增加或删除列表元素
直接重新赋值给根据索引值取出的值，或通过 append（）函数来添加。

通过 del 语句删除列表项，如：dellist1[2]

2） 列表的脚本操作符
和对字符串的操作类似。

Python 表达式

结果

描述

len([1, 2, 3])

3

长度

[1, 2, 3] + [4, 5, 6]

[1, 2, 3, 4, 5, 6]

组合

['Hi!'] * 4

['Hi!', 'Hi!', 'Hi!', 'Hi!']

重复

3 in [1, 2, 3]

True

判断元素是否存在于列表中

for x in [1, 2, 3]: print x,

1 2 3

迭代

3） 列表的截取

Python 表达式

结果

描述

L[2]

'Taobao'

读取列表中第三个元素

L[-2]

'Runoob'

读取列表中倒数第二个元素

L[1:]

['Runoob', 'Taobao']

从第二个元素开始截取列表

4） Python 中列表的函数及方法

Python 包含以下函数:

函数

描述

cmp(list1, list2)
比较两个列表的元素

len(list)

列表元素个数

max(list)

返回列表元素最大值

min(list)

返回列表元素最小值

list(seq)

将元组转换为列表

Python 包含以下方法:

函数

描述

list.append(obj)

在列表末尾添加新的对象

list.count(obj)

统计某个元素在列表中出现的次数

list.extend(seq)

在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）

list.index(obj)

从列表中找出某个值第一个匹配项的索引位置

list.insert(index, obj)

将对象插入列表

list.pop(obj=list[-1])

移除列表中的一个元素（默认最后一个元素），并且返回该元素的值

list.remove(obj)

移除列表中某个值的第一个匹配项

list.reverse()

反向列表中元素

list.sort([func])

对原列表进行排序

4. Tuple（元组）
1）与列表的区别
类似列表，但列表用[ ]标识，元组用()标识，并且列表元素可二次赋值，但元组元素不能。

2）元组的创建
创建空元组：tuple（） 。

创建只有一个元素的元组：tuple（a, ），必须要在元素后加逗号。

3）元素的访问
虽然创建时用（）包含，但是在访问单个元素时，与列表一样，通过[索引号]来访问。

4）删除元组
元组中的单个元素不能被删除，但是元组可以通过 del 语句整个删除。

5）元组运算符（同列表）

6）任意无符号的对象，以逗号隔开，默认为元组（无关闭分隔符）
7）元组内置函数
函数

描述

cmp(tuple1, tuple2)
比较两个元组元素。

len(tuple)
计算元组元素个数。

max(tuple)
返回元组中元素最大值。

min(tuple)
返回元组中元素最小值。

tuple(seq)
将列表转换为元组。

5. Dictionary（字典）

1) 与列表的差别

列表是有序的对象集合，字典是无序的对象结合。字典中的元素通过 Key 来获取，而列表中的元素通过位移来获取。

2) 字典的定义

下面是两种定义字典的方法，两种方法都与列表的定义方法类似。

```
dict = {}

dict['one'] ="This is one"

dict[2] ="This is two"



tinydict = {'name':'john','code':6734,'dept':'sales'}
```

3) 数据类型的转换

函数	描述
int(x [,base])	将 x 转换为一个整数
long(x [,base])	

将 x 转换为一个长整数

float(x)

将 x 转换到一个浮点数

complex(real [,imag])

创建一个复数

str(x)

将对象 x 转换为字符串

repr(x)

将对象 x 转换为表达式字符串

eval(str)

用来计算在字符串中的有效 Python 表达式,并返回一个对象

tuple(s)

将序列 s 转换为一个元组

list(s)

将序列 s 转换为一个列表

set(s)

转换为可变集合

dict(d)

创建一个字典。d 必须是一个序列 (key,value)元组。

frozenset(s)

转换为不可变集合

chr(x)

将一个整数转换为一个字符

unichr(x)

将一个整数转换为 Unicode 字符

ord(x)

将一个字符转换为它的整数值

hex(x)

将一个整数转换为一个十六进制字符串

oct(x)

将一个整数转换为一个八进制字符串

Python 运算符
Python 支持以下八种运算符：

1. 算数运算符
假设 a=10，b=20

运算符

描述

实例

+

加 - 两个对象相加

a + b 输出结果 30

-

减 - 得到负数或是一个数减去另一个数

a - b 输出结果 -10

*

乘 - 两个数相乘或是返回一个被重复若干次的字符串

a * b 输出结果 200

/

除 - x 除以 y

b / a 输出结果 2（整数除整数，只能得整数，要求小数需要将其中一个改为浮点数）

%

取模 - 返回除法的余数

b % a 输出结果 0

**

幂 - 返回 x 的 y 次幂

a**b 为 10 的 20 次方， 输出结果 100000000000000000000

//

取整除 - 返回商的整数部分

9//2 输出结果 4 , 9.0//2.0 输出结果 4.0

2. 比较运算符
运算符

描述

实例

==

等于 - 比较对象是否相等

(a == b) 返回 False。

!=

不等于 - 比较两个对象是否不相等

(a != b) 返回 true.

<>

不等于 - 比较两个对象是否不相等

(a <> b) 返回 true。这个运算符类似 != 。

>

大于 - 返回 x 是否大于 y

(a > b) 返回 False。

<

小于 - 返回 x 是否小于 y。所有比较运算符返回 1 表示真，返回 0 表示假。这分别与特殊的变量 True 和 False 等价。注意，这些变量名的大写。

(a < b) 返回 true。

>=

大于等于 - 返回 x 是否大于等于 y。

(a >= b) 返回 False。

<=

小于等于 - 返回 x 是否小于等于 y。

(a <= b) 返回 true。

3. Python 赋值运算符
运算符

描述

实例

=

简单的赋值运算符

c = a + b 将 a + b 的运算结果赋值为 c

+=

加法赋值运算符

c += a 等效于 c = c + a

-=

减法赋值运算符

c -= a 等效于 c = c - a

*=

乘法赋值运算符

c *= a 等效于 c = c * a

/=

除法赋值运算符

c /= a 等效于 c = c / a

%=

取模赋值运算符

c %= a 等效于 c = c % a

**=

幂赋值运算符

c **= a 等效于 c = c ** a

//=

取整除赋值运算符

c //= a 等效于 c = c // a

4. Python 位运算符
按位运算符是把数字看作二进制来进行计算的。Python 中的按位运算法则如下：

运算符

描述

实例

&

按位与运算符：参与运算的两个值,如果两个相应位都为 1,则该位的结果为 1,否则为 0

(a & b) 输出结果 12，二进制解释： 0000 1100

|

按位或运算符：只要对应的二个二进位有一个为 1 时，结果位就为 1。

(a | b) 输出结果 61，二进制解释： 0011 1101

^

按位异或运算符：当两对应的二进位相异时，结果为 1

(a ^ b) 输出结果 49，二进制解释： 0011 0001

~

按位取反运算符：对数据的每个二进制位取反,即把 1 变为 0,把 0 变为 1 。~x 类似于 -x-1

(~a) 输出结果 -61，二进制解释： 1100 0011，在一个有符号二进制数的补码形式。

<<

左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补 0。

a << 2 输出结果 240，二进制解释： 1111 0000

>>

右移动运算符 :把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数

a >> 2 输出结果 15，二进制解释： 0000 1111

5. Python 逻辑运算符
运算符

逻辑表达式

描述

实例

and

x and y

布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。

(a and b) 返回 20。

or

x or y

布尔"或" - 如果 x 是非 0，它返回 x 的值，否则它返回 y 的计算值。

(a or b) 返回 10。

not

not x

布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。

not(a and b) 返回 False

6. Python 成员运算符
运算符

描述

实例

in

如果在指定的序列中找到值返回 True，否则返回 False。

x 在 y 序列中 ，如果 x 在 y 序列中返回 True。

not in

如果在指定的序列中没有找到值返回 True，否则返回 False。

x 不在 y 序列中 ，如果 x 不在 y 序列中返回 True。

7. Python 身份运算符（判断引用的对象）
1） is 与==的区别
==判断两者是否完全相等，而 is 判断两个对象引用的对象是否是同一个。

运算符

描述

实例

is

is 是判断两个标识符是不是引用自一个对象

x is y,类似 id(x) == id(y)，如果引用的是同一个对象则返回 True，否则返回 False

is not

is not 是判断两个标识符是不是引用自不同对象

x is not y， 类似 id(a) != id(b)。如果引用的不是同一个对象则返回结果 True，否则返回 False。

8. Python 运算符优先级
运算符

描述

**

指数 (最高优先级)

~ + -

按位翻转，一元加号和减号（最后两个的方法名为 +@ 和 -@）

* / % //

乘，除，取模和取整除

+ -

加法减法

>> <<

右移，左移运算符

&

位 'AND'

^ |

位运算符

<= < > >=

比较运算符

<> == !=

等于运算符

= %= /= //= -= += *= **=

赋值运算符

is is not

身份运算符

in not in

成员运算符

not or and

逻辑运算符

Python 语句

1. 条件语句

Python 不支持 switch 语句，因此判断结果对应多种执行方式时，只能用 elif 来做。

```
num = 5
if num ==3:             #判断 num 的值
    print('boss')
elif num ==2:
    print('user')
elif num ==1:
    print('worker')
```

2. 循环语句

Python 中没有 do while 循环。

循环类型

描述

while 循环

在给定的判断条件为 true 时执行循环体，否则退出循环体。

for 循环

重复执行语句

嵌套循环

你可以在 while 循环体中嵌套 for 循环（for 中也可以嵌套 for 吧）

循环控制语句：

控制语句

描述

break 语句

在语句块执行过程中终止循环，并且跳出整个循环

continue 语句

在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。

pass 语句

pass 是空语句，是为了保持程序结构的完整性。

1）pass 语句在函数中的作用
当你在编写一个程序时，执行语句部分思路还没有完成，这时你可以用 pass 语句来占位，也可以当做是一个标记，是要过后来完成的代码。比如下面这样：

```
def iplaypython():
    pass
```

定义一个函数 iplaypython，但函数体部分暂时还没有完成，又不能空着不写内容，因此可以用 pass 来替代占个位置。
2）pass 语句在循环中的作用
pass 也常用于为复合语句编写一个空的主体，比如说你想一个 while 语句的无限循环，每次迭代时不需要任何操作，你可以这样写：

```
while True:
    pass
```

以上只是举个例子，现实中最好不要写这样的代码，因为执行代码块为 pass 也就是空什么也不做，这时 python 会进入死循环。
3）pass 语句用法总结
1、空语句，什么也不做
2、在特别的时候用来保证格式或是语义的完整性

4）While 循环（可在循环中使用 else 语句）

```
# continue  和  break 用法

i=1
while i<10:
    i+= 1
    if i%2>0:   #  非双数时跳过输出
        continue
    print(i)   #  输出双数 2、4、6、8、10

i = 1
while 1:   #  循环条件为 1 必定成立
```

```
    print(i)   # 输出 1~10
    i+= 1
    if i>10:   # 当 i 大于 10 时跳出循环
        break
```

在循环中使用 else 语句，即当条件不满足之后，结束循环，执行 else 语句

```
count = 0
while count <5:
    print(count," is less than 5")
    count = count +1
else:
    print(count," is not less than 5")
```

5）for 循环（可在循环中使用 else 语句）

可以通过直接取值迭代，也可以通过序列索引迭代

取值迭代：

```
for letter in 'Python':   # 逐个输出字符串中的字符
    print('当前字母 :', letter)

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:   # 逐个输出列表中的元素
    print('当前水果 :', fruit)
print("Good bye!")
```

索引迭代：

```
fruits = ['banana','apple','mango']
#通过 len（）函数获得列表的长度，通过 range（）函数获得了一个值不超过长度-1 的索引序列
for index in range(len(fruits)):
    print('当前水果 :',fruits[index])
    print("Good bye!")
```

Python 日期和时间

Python 是以时间戳来记录时间的，也就是当前时间距离 1970 年 1 月 1 日过去了多少秒，因此获取时间的方法一般是先获取时间戳，再将时间戳转换为时间元组，再将时间元组转换为不同格式的时间数据。

1.获取时间戳

```
import time; #引入 time 模块

ticks = time.time()
```

2.获取时间
什么是时间元组？

很多 Python 函数用一个元组装起来的 9 组数字处理时间，也就是 struct_time 元组：

属性

字段

值

tm_year
4 位数年

2008

tm_mon
月

1 到 12

tm_mday
日

1 到 31

tm_hour
小时

0 到 23

tm_min
分钟

0 到 59

tm_sec
秒

0 到 61 (60 或 61 是闰秒)

tm_wday
一周的第几日

0 到 6 (0 是周一)

tm_yday
一年的第几日

1 到 366 (儒略历)

tm_isdst
夏令时

-1, 0, 1, -1 是决定是否为夏令时的旗帜

代码接上，将获取的时间戳转换为时间元组：

localtime = time.localtime(time.time())

print("本地时间为 :", localtime)

3.获取格式化的时间
localtime = time.asctime(time.localtime(time.time()))

print("本地时间为 :", localtime)

4.获取更多格式的格式化时间
time.strftime(format[, t]) #总的代码形式

格式化成 2016-03-20 11:45:39 形式

print(time.strftime("%Y-%m-%d%H:%M:%S", time.localtime()))

python 中时间日期格式化符号：

- %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

5.获取某个月的日历

```
import calendar

cal = calendar.month(2016,1)
```

6.Time 和 Calendar 模块的函数及属性

函数

描述

time.altzone
返回格林威治西部的夏令时地区的偏移秒数。如果该地区在格林威治东部会返回负值（如西欧，包括英国）。对夏令时启用地区才能使用。

time.asctime([tupletime])
接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2008"（2008 年 12 月 11 日 周二 18 时 07 分 14 秒）的 24 个字符的字符串。

time.clock( )
用以浮点数计算的秒数返回当前的 CPU 时间。用来衡量不同程序的耗时，比 time.time()更有用。

time.ctime([secs])
作用相当于 asctime(localtime(secs))，未给参数相当于 asctime()

time.gmtime([secs])
接收时间辍（1970 纪元后经过的浮点秒数）并返回格林威治天文时间下的时间元组 t。注：t.tm_isdst 始终为 0

time.localtime([secs])
接收时间辍（1970 纪元后经过的浮点秒数）并返回当地时间下的时间元组 t（t.tm_isdst 可取 0 或 1，取决于当地当时是不是夏令时）。

time.mktime(tupletime)
接受时间元组并返回时间辍（1970 纪元后经过的浮点秒数）。

time.sleep(secs)
推迟调用线程的运行，secs 指秒数。

time.strftime(fmt[,tupletime])
接收以时间元组，并返回以可读字符串表示的当地时间，格式由 fmt 决定。

time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')
根据 fmt 的格式把一个时间字符串解析为时间元组。

time.time( )
返回当前时间的时间戳（1970 纪元后经过的浮点秒数）。

time.tzset()
根据环境变量 TZ 重新初始化时间相关设置

Time 模块包含了以下 2 个非常重要的属性：

属性

描述

time.timezone
属性 time.timezone 是当地时区（未启动夏令时）距离格林威治的偏移秒数（>0，美洲;<=0 大部分欧洲，亚洲，非洲）。

time.tzname
属性 time.tzname 包含一对根据情况的不同而不同的字符串，分别是带夏令时的本地时区名称，和不带的。

函数

描述

calendar.calendar(year,w=2,l=1,c=6)
返回一个多行字符串格式的 year 年年历，3 个月一行，间隔距离为 c。 每日宽度间隔为 w 字符。每行长度为 21* W+18+2* C。l 是每星期行数。

calendar.firstweekday( )
返回当前每周起始日期的设置。默认情况下，首次载入 caendar 模块时返回 0，即星期一。

calendar.isleap(year)
是闰年返回 True，否则为 false。

calendar.leapdays(y1,y2)
返回在 Y1，Y2 两年之间的闰年总数。

calendar.month(year,month,w=2,l=1)
返回一个多行字符串格式的 year 年 month 月日历，两行标题，一周一行。每日宽度间隔为 w 字符。每行的长度为 7* w+6。l 是每星期的行数。

calendar.monthcalendar(year,month)
返回一个整数的单层嵌套列表。每个子列表装载代表一个星期的整数。Year 年 month 月外的日期都设为 0;范围内的日子都由该月第几日表示，从 1 开始。

calendar.monthrange(year,month)
返回两个整数。第一个是该月的星期几的日期码，第二个是该月的日期码。日从 0（星期一）到 6（星期日）;月从 1 到 12。

calendar.prcal(year,w=2,l=1,c=6)
相当于 print calendar.calendar(year,w,l,c).

calendar.prmonth(year,month,w=2,l=1)
相当于 print calendar.calendar（year，w，l，c)。

calendar.setfirstweekday(weekday)
设置每周的起始日期码。0（星期一）到 6（星期日)。

calendar.timegm(tupletime)
和 time.gmtime 相反：接受一个时间元组形式，返回该时刻的时间辍（1970 纪元后经过的浮点秒数)。

calendar.weekday(year,month,day)
返回给定日期的日期码。0（星期一）到 6（星期日)。月份为 1（一月） 到 12（12 月)。

7.其他可以处理时间的模块
· datetime 模块

· pytz 模块

· dateutil 模块

Python 函数

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。

1.Python 函数定义
def functionname(parameters):

"函数_文档字符串"

function_suite

return [expression]

2.对象创建
在 python 中，类型属于对象，变量是没有类型的：

a=[1,2,3] #赋值后这个对象就已经创建好了

a="Runoob"

以上代码中，[1,2,3] 是 List 类型，"Runoob" 是 String 类型，而变量 a 是没有类型，她仅仅是一个对象的引用（一个指针），可以是 List 类型对象，也可以指向 String 类型对象。

3.可更改对象和不可更改对象
在 python 中，strings,tuples, 和 numbers 是不可更改（重新赋值后，原值不再存在）的对象，而 list,dict 等则是可以修改（重新赋值后，原来的值依旧存在，依旧可以获取到）的对象。

· 不可变类型：变量赋值 a=5 后再赋值 a=10，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变 a 的值，相当于新生成了 a。

· 可变类型：变量赋值 la=[1,2,3,4] 后再赋值 la[2]=5 则是将 list la 的第三个元素值更改，本身 la 没有动，只是其内部的一部分值被修改了。

4.可更改对象和不可更改对象的参数传递
python 函数的参数传递：

不可变类型：类似 c++的值传递，如 整数、字符串、元组。如 fun（a），传递的只是 a 的值，没有影响 a 对象本身。比如在 fun（a）内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。

```
def ChangeInt(a):
a = 10

b = 2
ChangeInt(b)
print(b)  #结果是 2
```

int2 对象指向变量 b，而调用 changeInt 函数时，变量 a 就是变量 b，此时的变量 a 和变量 b 都对应 int 2 对象，但是在之后 a 被重新赋值为 10，此时变量 a 指向对象 int 10，产生了一个新的 int 型对象，而变量 b 所指向的对象不变。

可变类型：类似 c++的引用传递，如 列表，字典。如 fun（la），则是将 la 真正的传过去，修改后 fun 外部的 la 也会受影响

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说传不可变对象和传可变对象。

```
def changeme( mylist):
mylist.append([1,2,3,4]);  #"修改传入的列表"
    print("函数内取值: ", mylist)
    return

#  调用 changeme 函数
mylist = [10,20,30];
changeme(mylist);
print("函数外取值: ", mylist)
```

因为上面的代码传递的是可变对象，因此在函数内取值和函数外取值都是同样的结果。

5.调用函数时的参数使用
1）必备参数
必须与函数声明时一致的顺序来传递参数。

2）关键字参数
传参数时可与函数声明的顺序不一样，因为 Python 解释器可以用参数名来匹配参数值。

3）缺省参数
传入参数时未给参数赋值，则保持默认值。

```
#可写函数说明



def printinfo( name, age=35):
print("Name: ", name);  #"打印任何传入的字符串"
    print("Age ", age);
    return;

#调用 printinfo 函数
printinfo(age=50,name="miki");
printinfo(name="miki");
```

这里第二句的 age 没有输入值，则输出默认值 35。

4）不定长参数（也就是包含非必备参数的参数定义）
当不确定会传入参数的个数时，可以对可以不输入的参数名前面加 “*” 号，按顺序输入时进行对应即可。

```
def printinfo( arg1,*vartuple):
print("输出: ",arg1)  #"打印任何传入的参数"
    for var in vartuple:
      print var
    return;

#  调用 printinfo 函数
printinfo(10);
printinfo(70,60,50);
```

5.匿名函数（使用 lambda 创建）
python 使用 lambda 来创建匿名函数。

lambda 只是一个表达式，函数体比 def 简单很多。

lambda 的主体是一个表达式，而不是一个代码块。仅仅能在 lambda 表达式中封装有限的逻辑进去。

lambda 函数拥有自己的命名空间，且不能访问自有参数列表之外或全局命名空间里的参数（只能访问自己的命名空间里的参数）。

虽然 lambda 函数看起来只能写一行，却不等同于 C 或 C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

```
#  用一个语句来定义一个 Sum 函数


sum = lambda arg1, arg2: arg1+ arg2;

#  调用 sum 函数
print("相加后的值为  : ", sum(10,20))
print("相加后的值为  : ", sum(20,20))
```

6.变量作用域
1）全局变量
定义在所有函数外则为全局变量，可在所有代码中使用。

当需要在函数内定义全局变量时，必须用 global 语句。

2）局部变量
定义在函数内则为局部变量，只能在相应的代码块内使用。

```
total = 0; #  这是一个全局变量
#  可写函数说明
def sum( arg1, arg2):    #返回 2 个参数的和."
    total = arg1 + arg2; # total 在这里是局部变量.
    print("函数内是局部变量 : ", total)
    return(total);

#调用 sum 函数
sum( 10, 20);
print("函数外是全局变量 : ", total)
```

在上面的例子中，在函数内部，total 是局部变量，而在外部 total 是全局变量，局部变量的改变不会改变全局变量的值，因此第一个打印结果是 30，而第二个是 0。

Python 模块

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和 Python 语句。

模块让你能够有逻辑地组织你的 Python 代码段。

把相关的代码分配到一个模块里能让你的代码更好用，更易懂。

1.导入模块的三种方法的区别

1）import 导入

```
import support    # 导入模块，并不导入单个函数



#  现在可以调用模块里包含的函数了 ，但是必须通过模块名.函数名的方式调用

support.print_func("Runoob")
```

2）From … import 导入

下面的语句可以将模块中的某一部分导入，它只会将里这个部分单个引入到执行这个声明的模块的全局符号表。这个部分可以是某个函数块，也可以是函数块的子函数块。

```
from modname import name1[, name2[, ... nameN]]
```

3）From … import * 导入

前两种语句都是导入模块的部分，只是部分导入的位置不同，而使用 From…import*则是导入模块的全部项目，尽量少用，太耗内存。

2.命名空间和作用域

变量是拥有匹配对象的名字（标识符）。

命名空间是一个包含了变量名称们（键）和它们各自相应的对象们（值）的字典。每个函数都会有自己的命名空间，当变量出现在函数内部时，Python 默认其为局部变量，若存在一个与局部变量重名的全局变量，则由局部变量覆盖全局变量。

Python 的命名空间分局部命名空间和全局命名空间。

3.找出模块中所有的模块名、函数名、变量名（dir()函数）

```
# 导入内置 math 模块
import math
content = dir(math)
print content;
```

以上实例输出结果：

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

在这里，特殊字符串变量__name__指向模块的名字，__file__指向该模块的导入文件名。

4.查看全局变量和局部变量命名空间

根据调用地方的不同，globals()和 locals()函数可被用来返回全局和局部命名空间里的名字。

如果在函数内部调用 locals()，返回的是所有能在该函数里访问的命名。

如果在函数内部调用 globals()，返回的是所有在该函数里能访问的全局名字。

两个函数的返回类型都是字典。所以名字们能用 keys() 函数摘取。

5. reload() 函数

当一个模块被导入到一个脚本，模块顶层部分的代码只会被执行一次。

因此，如果你想重新执行模块里顶层部分的代码，可以用 reload() 函数。该函数会重新导入之前导入过的模块。语法如下：在这里，module_name 要直接放模块的名字，而不是一个字符串形式。

```
reload(module_name)
```

Python I/O 函数（不只是文件读写）

1.读取键盘输入

读取输入的行：

```
str = input("请输入：");
print("你输入的内容是: ", str)
```

input() 函数可以接受 Python 表达式的输入，并输出表达式的计算结果

```
请输入：[x*5 for x in range(2,10,2)]
你输入的内容是:  [10, 20, 30, 40]
```

2.打开、关闭读写文件

可以用 file 对象对大多数文件进行操作。

1）打开文件

你必须先用 Python 内置的 open()函数打开一个文件，创建一个 file 对象，相关的方法才可以调用它进行读写。

```
file object = open(file_name [, access_mode][, buffering])
```

· file_name：file_name 变量是一个包含了你要访问的文件名称的字符串值。

· access_mode：access_mode 决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

· buffering:如果 buffering 的值被设为 0，就不会有寄存。如果 buffering 的值取 1，访问文件时会寄存行。如果将 buffering 的值设为大于 1 的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

· 当文件不存在但通过 open 函数打开时，如果是写入则会自动创建文件，如果是读取内容则会报错。

打开文件的模式

描述

r

以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。

rb

以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。

r+

打开一个文件用于读写。文件指针将会放在文件的开头。

rb+

以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。

w

打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

wb

以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

w+

打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

wb+

以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

a

打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

ab

以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

a+

打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾（不然怎么追加呢）。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。

ab+

以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

2）File 对象的属性
文件被打开后就会创建一个 File 对象。

对象属性

描述

file.closed

返回 true 如果文件已被关闭，否则返回 false。

file.mode

返回被打开文件的访问模式。

file.name

返回文件的名称。

file.softspace

如果用 print 输出后，必须跟一个空格符，则返回 false。否则返回 true。

3）关闭文件
当一个文件对象的引用被重新指定给另一个文件时，Python 会关闭之前的文件（Python 会自动地关闭之前的文件对象，这样不那么耗费内存）。用 close（）方法关闭文件是一个很好的习惯。

4）读写文件
无论读、写、还是关闭文件，都需要先用 open 打开文件。

```
# 写一个文件
fo = open("foo.txt", "wb")
fo.write( "www.runoob.com!\nVery good site!\n");

# 关闭打开的文件
fo.close()


# 读一个文件
fo = open("foo.txt", "r+")
str = fo.read(10);    #这里的参数 10 表示的是被读取的字节数
print("读取的字符串是 : ", str)
# 关闭打开的文件
fo.close()
```

5）文件内定位
tell()方法告诉你文件内的当前位置（那应该就是告诉你文件指针在哪个位置）；换句话说，

下一次的读写会发生在文件开头这么多字节之后。

seek(offset [,from])方法改变当前文件的位置。Offset 变量表示要移动的字节数。From 变量指定开始移动字节的参考位置。如果 from 被设为 0，这意味着将文件的开头作为移动字节的参考位置。如果设为 1，则使用当前的位置作为参考位置。如果它被设为 2，那么该文件的末尾将作为参考位置。

```
# 打开一个文件
fo = open("foo.txt", "r+")
str = fo.read(10);
print("读取的字符串是 : ", str)

# 查找当前位置
position = fo.tell();
print("当前文件位置 : ", position)

# 把指针再次重新定位到文件开头
position = fo.seek(0, 0);
str = fo.read(10);
print("重新读取字符串 : ", str)
# 关闭打开的文件
fo.close()
```

3.文件 File 对象的方法

方法

描述

file.close()
关闭文件。关闭后文件不能再进行读写操作。

file.flush()
刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件，而不是被动的等待输出缓冲区写入。

file.fileno()
返回一个整型的文件描述符(file descriptor FD 整型)，可以用在如 os 模块的 read 方法等一些底层操作上。

file.isatty()
如果文件连接到一个终端设备返回 True，否则返回 False。

file.next()
返回文件下一行。

file.read([size])
从文件读取指定的字节数，如果未给定或为负则读取所有。

file.readline([size])
读取整行，包括 "\n" 字符。

file.readlines([sizehint])
读取所有行并返回列表，若给定 sizeint>0，返回总和大约为 sizeint 字节的行，实际读取值可能比 sizhint 较大，因为需要填充缓冲区。

file.seek(offset[, whence])
设置文件当前位置

file.tell()
返回文件当前位置。

file.truncate([size])
截取文件，截取的字节通过 size 指定，默认为当前文件位置。

file.write(str)
将字符串写入文件，没有返回值。

file.writelines(sequence)
向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。

4.文件 OS（Python 中处理文件和目录的模块）
方法

描述

os.access(path, mode)
检验权限模式

os.chdir(path)
改变当前工作目录

os.chflags(path, flags)
设置路径的标记为数字标记。

os.chmod(path, mode)
更改权限

os.chown(path, uid, gid)
更改文件所有者

os.chroot(path)
改变当前进程的根目录

os.close(fd)
关闭文件描述符 fd

os.closerange(fd_low, fd_high)
关闭所有文件描述符，从 fd_low (包含) 到 fd_high (不包含)，错误会忽略

os.dup(fd)
复制文件描述符 fd

os.dup2(fd, fd2)
将一个文件描述符 fd 复制到另一个 fd2

os.fchdir(fd)
通过文件描述符改变当前工作目录

os.fchmod(fd, mode)
改变一个文件的访问权限，该文件由参数 fd 指定，参数 mode 是 Unix 下的文件访问权限。

os.fchown(fd, uid, gid)
修改一个文件的所有权，这个函数修改一个文件的用户 ID 和用户组 ID，该文件由文件描述符 fd 指定。

os.fdatasync(fd)
强制将文件写入磁盘，该文件由文件描述符 fd 指定，但是不强制更新文件的状态信息。

os.fdopen(fd[, mode[, bufsize]])
通过文件描述符 fd 创建一个文件对象，并返回这个文件对象

os.fpathconf(fd, name)
返回一个打开的文件的系统配置信息。name 为检索的系统配置的值，它也许是一个定义系统值的字符串，这些名字在很多标准中指定（POSIX.1, Unix 95, Unix 98，和其它）。

os.fstat(fd)
返回文件描述符 fd 的状态，像 stat()。

os.fstatvfs(fd)
返回包含文件描述符 fd 的文件的文件系统的信息，像 statvfs()

os.fsync(fd)
强制将文件描述符为 fd 的文件写入硬盘。

os.ftruncate(fd, length)
裁剪文件描述符 fd 对应的文件，所以它最大不能超过文件大小。

os.getcwd()
返回当前工作目录

os.getcwdu()
返回一个当前工作目录的 Unicode 对象

os.isatty(fd)
如果文件描述符 fd 是打开的，同时与 tty(-like)设备相连，则返回 true，否则 False。

os.lchflags(path, flags)
设置路径的标记为数字标记，类似 chflags()，但是没有软链接

os.lchmod(path, mode)
修改连接文件权限

os.lchown(path, uid, gid)
更改文件所有者，类似 chown，但是不追踪链接。

os.link(src, dst)
创建硬链接，名为参数 dst，指向参数 src

os.listdir(path)
返回 path 指定的文件夹包含的文件或文件夹的名字的列表。

os.lseek(fd, pos, how)
设置文件描述符 fd 当前位置为 pos, how 方式修改: SEEK_SET 或者 0 设置从文件开始的计算的 pos; SEEK_CUR 或者 1 则从当前位置计算; os.SEEK_END 或者 2 则从文件尾部开始. 在 unix，Windows 中有效

os.lstat(path)
像 stat(),但是没有软链接

os.major(device)
从原始的设备号中提取设备 major 号码 (使用 stat 中的 st_dev 或者 st_rdev field)。

os.makedev(major, minor)
以 major 和 minor 设备号组成一个原始设备号

os.makedirs(path[, mode])
递归文件夹创建函数。像 mkdir()，但创建的所有 intermediate-level 文件夹需要包含子文件

夹。

os.minor(device)
从原始的设备号中提取设备 minor 号码 (使用 stat 中的 st_dev 或者 st_rdev field)。

os.mkdir(path[, mode])
以数字 mode 的 mode 创建一个名为 path 的文件夹.默认的 mode 是 0777 (八进制)。

os.mkfifo(path[, mode])
创建命名管道，mode 为数字，默认为 0666 (八进制)

os.mknod(filename[, mode=0600, device])
创建一个名为 filename 文件系统节点（文件，设备特别文件或者命名 pipe）。

os.open(file, flags[, mode])
打开一个文件，并且设置需要的打开选项，mode 参数是可选的

os.openpty()
打开一个新的伪终端对。返回 pty 和 tty 的文件描述符。

os.pathconf(path, name)
返回相关文件的系统配置信息。

os.pipe()
创建一个管道. 返回一对文件描述符(r, w) 分别为读和写

os.popen(command[, mode[, bufsize]])
从一个 command 打开一个管道

os.read(fd, n)
从文件描述符 fd 中读取最多 n 个字节，返回包含读取字节的字符串，文件描述符 fd 对应文件已达到结尾，返回一个空字符串。

os.readlink(path)
返回软链接所指向的文件

os.remove(path)
删除路径为 path 的文件。如果 path 是一个文件夹，将抛出 OSError; 查看下面的 rmdir()删除一个 directory。

os.removedirs(path)
递归删除目录。

os.rename(src, dst)

重命名文件或目录，从 src 到 dst

os.renames(old, new)
递归地对目录进行更名，也可以对文件进行更名。

os.rmdir(path)
删除 path 指定的空目录，如果目录非空，则抛出一个 OSError 异常。

os.stat(path)
获取 path 指定的路径的信息，功能等同于 C API 中的 stat()系统调用。

os.stat_float_times([newvalue])
决定 stat_result 是否以 float 对象显示时间戳

os.statvfs(path)
获取指定路径的文件系统统计信息

os.symlink(src, dst)
创建一个软链接

os.tcgetpgrp(fd)
返回与终端 fd（一个由 os.open()返回的打开的文件描述符）关联的进程组

os.tcsetpgrp(fd, pg)
设置与终端 fd（一个由 os.open()返回的打开的文件描述符）关联的进程组为 pg。

os.tempnam([dir[, prefix]])
返回唯一的路径名用于创建临时文件。

os.tmpfile()
返回一个打开的模式为(w+b)的文件对象 .这文件对象没有文件夹入口，没有文件描述符，将会自动删除。

os.tmpnam()
为创建一个临时文件返回一个唯一的路径

os.ttyname(fd)
返回一个字符串，它表示与文件描述符 fd 关联的终端设备。如果 fd 没有与终端设备关联，则引发一个异常。

os.unlink(path)
删除文件路径

os.utime(path, times)

返回指定的 path 文件的访问和修改的时间。

os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])
输出在文件夹中的文件名通过在树中游走，向上或者向下。

os.write(fd, str)
写入字符串到文件描述符 fd 中. 返回实际写入的字符串长度

Python 中的异常处理
1.异常类型
异常名称

描述

BaseException

所有异常的基类

SystemExit

解释器请求退出

KeyboardInterrupt

用户中断执行(通常是输入^C)

Exception

常规错误的基类

StopIteration

迭代器没有更多的值

GeneratorExit

生成器(generator)发生异常来通知退出

StandardError

所有的内建标准异常的基类

ArithmeticError

所有数值计算错误的基类

FloatingPointError

浮点计算错误

OverflowError

数值运算超出最大限制

ZeroDivisionError

除(或取模)零 (所有数据类型)

AssertionError

断言语句失败

AttributeError

对象没有这个属性

EOFError

没有内建输入,到达 EOF 标记

EnvironmentError

操作系统错误的基类

IOError

输入/输出操作失败

OSError

操作系统错误

WindowsError

系统调用失败

ImportError

导入模块/对象失败

LookupError

无效数据查询的基类

IndexError

序列中没有此索引(index)

KeyError

映射中没有这个键

MemoryError

内存溢出错误(对于 Python 解释器不是致命的)

NameError

未声明/初始化对象 (没有属性)

UnboundLocalError

访问未初始化的本地变量

ReferenceError

弱引用(Weak reference)试图访问已经垃圾回收了的对象

RuntimeError

一般的运行时错误

NotImplementedError

尚未实现的方法

SyntaxError

Python 语法错误

IndentationError

缩进错误

TabError

Tab 和空格混用

SystemError

一般的解释器系统错误

TypeError

对类型无效的操作

ValueError

传入无效的参数

UnicodeError

Unicode 相关的错误

UnicodeDecodeError

Unicode 解码时的错误

UnicodeEncodeError

Unicode 编码时错误

UnicodeTranslateError

Unicode 转换时错误

Warning

警告的基类

DeprecationWarning

关于被弃用的特征的警告

FutureWarning

关于构造将来语义会有改变的警告

OverflowWarning

旧的关于自动提升为长整型(long)的警告

PendingDeprecationWarning

关于特性将会被废弃的警告

RuntimeWarning

可疑的运行时行为(runtime behavior)的警告

SyntaxWarning

可疑的语法的警告

UserWarning

用户代码生成的警告

2.异常处理
1）try/except 语句
捕捉异常通常用 try（捕捉错误）/except（处理错误）语句。如果你不想在异常发生时结束你的程序，只需在 try 里捕获它。异常可带参数，用于说明异常原因

```
try:
    fh = open("testfile", "w")
    fh.write("这是一个测试文件，用于测试异常!!")
except (IOError,RuntimeError):  # 当出现这两种 error 中的一种时，执行 except 后面的操作
    print("Error: 没有找到文件或读取文件失败")
except Exception as e:  # 使用异常基类捕获
    print("Error: 出错")
else:
    print("内容写入文件成功")
    fh.close()
```

2）try/finally 语句
try-finally 语句无论是否发生异常都将执行最后的代码。

```
try:
    fh = open("testfile", "w")
```

```
    fh.write("这是一个测试文件，用于测试异常!!")
finally:
    print("Error: 没有找到文件或读取文件失败")
```

3.自己设置异常（用于 bug 修改、错误检查）
我们可以使用 raise 语句自己触发异常

raise 语法格式如下：

```
raise [Exception [, args [, traceback]]]
```

语句中 Exception 是异常的类型（例如，NameError）参数是一个异常参数值。该参数是可选的，如果不提供，异常的参数是"None"。

最后一个参数是可选的（在实践中很少使用），如果存在，是跟踪异常对象。例子如下：

```
def functionName( level ):
    if level < 1:
        raise Exception("Invalid level!", level)
        # 触发异常后，后面的代码就不会再执行
```

对于自定义的异常，在使用 except 语句处理异常时，一定要将 Exception 中的异常名作为 except 语句的一个参数。例子如下：

```
try:
    #正常逻辑
except "Invalid level!":
    #触发自定义异常
else:
    #其余代码
```

4.自己创建异常类型，用于常见异常复用
一个异常可以是一个字符串，类或对象。

```
class Networkerror(RuntimeError):  # 基于 RuntimeError 类创建一个新的类
    def __init__(self, arg):  # 定义一个函数
        self.args = arg
```

```
try:
    raise Networkerror("Bad hostname")  #抛出一个 Networkerror 异常
except Networkerror as e:  #匹配异常的类型，当出现了 Networkerror 型异常时，执行该 except 语句，并传递变量 e，变量 e 是用于创建 Networkerror 类的实例，这里的 e 应该就是这个异常对象？
    print(e.args)
```

Python 内置函数
内置函数

abs()

divmod()

input()

open()

staticmethod()

all()

enumerate()

int()

ord()

str()

any()

eval()

isinstance()

pow()

sum()

basestring()

execfile()

issubclass()

print()

super()

bin()

file()

iter()

property()

tuple()

bool()

filter()

len()

range()

type()

bytearray()

float()

list()

raw_input()

unichr()

callable()

format()

locals()

reduce()

unicode()

chr()

frozenset()

long()

reload()

vars()

classmethod()

getattr()

map()

repr()

xrange()

cmp()

globals()

max()

reversed()

zip()

compile()

hasattr()

memoryview()

round()

__import__()

complex()

hash()

min()

set()

delattr()

help()

next()

setattr()

dict()

hex()

object()

slice()

dir()

id()

oct()

sorted()

exec 内置表达式

Python 面向对象

1.面向对象技术简介

· 类(Class): 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例（对象是类实例化之后的结果）。

· 类变量 :类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。

· 数据成员：类变量或者实例变量用于处理类及其实例对象的相关的数据。

· 方法重写：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。

· 实例变量：定义在方法中的变量，只作用于当前实例的类。

· 继承：即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允

许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个 Dog 类型的对象派生自 Animal 类，这是模拟"是一个（is-a）"关系（例图，Dog 是一个 Animal）。

· 实例化：创建一个类的实例，类的具体对象。

· 方法：类中定义的函数。

· 对象：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

2.创建类

```
'所有员工的基类'
class Employee:
    empCount = 0
    def __init__(self, name, salary): #构造函数
        self.name = name                    #  添加实例属性
        self.salary = salary              #  添加实例属性
        Employee.empCount += 1          #  修改类属性
    def displayCount(self):               #  添加实例方法
        print("TotalEmployee %d" % Employee.empCount) #  读取类属性

    def displayEmployee(self):          #  添加实例方法
        print("Name:", self.name, ", Salary:", self.salary) #  读取实例属性
```

· empCount 变量是一个类变量，它的值将在这个类的所有实例之间共享。你可以在内部类或外部类使用 Employee.empCount 访问。

· 第一种方法__init__()方法是一种特殊的方法，被称为类的构造函数或初始化方法，当创建了这个类的实例时就会调用该方法

· self 代表类的实例，self 在定义类的方法时是必须有的，虽然在调用时不必传入相应的参数。Self 代表了一件事情，那就是：类的方法与普通的函数只有一个特别的区别——它们必须有一个额外的第一个参数名称,按照惯例它的名称是 self。

3.创建实例对象
实例化类其他编程语言中一般用关键字 new，但是在 Python 中并没有这个关键字，类的实例化类似函数调用方式。

以下使用类的名称 Employee 来实例化，并通过 __init__ 方法接受参数。

```
"创建 Employee 类的第一个对象"
emp1 = Employee("Zara",2000)
```

```
"创建 Employee 类的第二个对象"
emp2 = Employee("Manni",5000)
```

4.操作对象属性
下面是读取对象属性的实例：

```
'所有员工的基类'
class Employee:
    empCount = 0
    def __init__(self, name, salary): #构造函数
        self.name = name                    # 添加实例属性
        self.salary = salary              # 添加实例属性
        Employee.empCount += 1          # 修改类属性
    def displayCount(self):               # 添加实例方法
        print("TotalEmployee %d" % Employee.empCount) # 读取类属性

    def displayEmployee(self):          # 添加实例方法
        print("Name:", self.name, ", Salary:", self.salary) # 读取实例属性


# "创建 Employee 类的第一个对象"
emp1 =Employee("Zara",2000)
# "创建 Employee 类的第二个对象"
emp2 =Employee("Manni",5000)
emp1.displayEmployee()
emp2.displayEmployee()
print("TotalEmployee %d" %Employee.empCount)
```

以下函数还可以对属性进行读取之外的操作：

· getattr(obj,name[, default]) :访问对象的属性。

· hasattr(obj,name):检查是否存在一个属性。

· setattr(obj,name,value):设置一个属性。如果属性不存在，会创建一个新属性。

· delattr(obj,name) :删除属性。

5.Python 内置的类属性
· __dict__ :类的属性（包含一个字典，由类的数据属性组成）

· __doc__ :类的文档字符串

· __name__:类名

· __module__:类定义所在的模块（类的全名是'__main__.className'，如果类位于一个导入模块 mymod 中，那么 className.__module__ 等于 mymod）

· __bases__ :类的所有父类构成元素（包含了一个由所有父类组成的元组）

6.类的继承
面向对象的编程带来的主要好处之一是代码的重用，实现这种重用的方法之一是通过继承机制。继承完全可以理解成类之间的类型和子类型关系。

在 python 中继承中的一些特点：

· 1：在继承中基类的构造（__init__()方法）不会被自动调用，它需要在其派生类的构造中亲自专门调用。

· 2：在调用基类的方法时，需要加上基类的类名前缀，且需要带上 self 参数变量。区别于在类中调用普通函数时并不需要带上 self 参数

· 3：Python 总是首先查找对应类型的方法，如果它不能在派生类中找到对应的方法，它才开始到基类中逐个查找。（先在本类中查找调用的方法，找不到才去基类中找）。

如果在继承元组中列了一个以上的类，那么它就被称作"多重继承" 。

```
class Parent:
    # 定义父类
    parentAttr =100
    def __init__(self):
        print("调用父类构造函数")
    def parentMethod(self):
        print('调用父类方法')
    def setAttr(self,attr):
        Parent.parentAttr =attr
    def getAttr(self):
        print("父类属性 :",Parent.parentAttr)

class Child(Parent):
    # 定义子类
    def __init__(self):
        print("调用子类构造方法")  #无论子类还是父类，都要单独写一次_init_
    def childMethod(self):
        print('调用子类方法')
    def getAttr (self):
        print('重写父类方法，因为父类方法不能满足需求')
```

```
c = Child()  #实例化子类
c.childMethod()#调用子类的方法
c.parentMethod()#调用父类方法
c.setAttr(200)  #再次调用父类的方法 -设置属性值
c.getAttr()#再次调用父类的方法 -获取属性值
```

你可以使用 issubclass()或者 isinstance()方法来检测，一个类或对象是否为其他类或对象的子类。

· issubclass() - 布尔函数判断一个类是另一个类的子类或者子孙类，语法：issubclass(sub,sup)

· isinstance(obj, Class)布尔函数如果 obj 是 Class 类的实例对象或者是一个 Class 子类的实例对象则返回 true。

7.运算符重载
Python 同样支持运算符重载，实例如下：

```
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def __str__(self):
        return 'Vector(%d, %d)' % (self.a, self.b)
    def __add__(self, other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2, 10)
v2 = Vector(5, -2)
print(v1 + v2)
```

以上代码执行结果如下所示:

```
Vector(7, 8)
```

8.类的私有属性及方法
1）类的私有属性
__private_attrs：两个下划线开头，声明该属性为私有，不能在类的外部被使用或直接访问。在类内部的方法中使用是 self.__private_attrs。

2）类的私有方法

__private_method：两个下划线开头，声明该方法为私有方法，不能在类地外部调用。在类的内部调用 self.__private_methods

3）实例

```
class JustCounter:
    __secretCount = 0
    # 私有变量
    publicCount = 0
    # 公开变量
    def count(self):
        self.__secretCount += 1
        self.publicCount += 1
        print(self.__secretCount)  # 在内部使用私有化属性，不会产生错误

counter = JustCounter()
counter.count()
counter.count()
print(counter.publicCount)
print(counter.__secretCount)
# 报错，实例不能访问私有变量
```

9.单下划线、双下划线、头尾双下划线说明：

· __foo__:定义的是特列方法，类似 __init__() 之类的。

· _foo:以单下划线开头的表示的是 protected 类型的变量，即保护类型只能允许其本身与子类进行访问，不能用于 from module import *

· __foo:双下划线的表示的是私有类型(private)的变量,只能是允许这个类本身进行访问了。

作者：数据架构师
来源：CSDN
原文：https://blog.csdn.net/luanpeng825485697/article/details/78387573