Python基础语法三

原创 xyzso1z 最后发布于2019-01-06 19:36:15 阅读数 66 收藏 编辑 展开

数据结构

Python 中有四种内置的数据结构——列表（List）、元组（Tuple）、字典（Dictionary）和集 合（Set）。

列表

```python
 # This is my shopping list
shoplist=['apple','mango','carrot','banana']
print('I have',len(shoplist),'items to purchase')
print('These items are:',end='')
for item in shoplist:
    print(item+',',end='')

print('\nI also have to buy rice.')
shoplist.append('rice')
print('my shopping list is now',shoplist)

print('I will sort my list now')
shoplist.sort()
print('Sort shopping list is',shoplist)

print('The first item I will buy is ',shoplist[0])
olditem=shoplist[0]
del shoplist[0]
print('I bought the ',olditem)
print('My shopping list is now',shoplist)
```

输出：

```
I have 4 items to purchase
These items are:apple,mango,carrot,banana,
I also have to buy rice.
my shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sort shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is  apple
I bought the  apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

元组

```
 #推荐总是使用括号 来指明元组的开始与结束 尽管括号是一个可选选项，，明了胜过晦涩，显式胜过隐式
zoo=('python','elephant','penguin')
print('Number of animals in the zoo is',len(zoo))

new_zoo='monkey','camel',zoo
print('Number of cages in the new zoo is',len(new_zoo))
print('All animals in new zoo are',new_zoo)
print('Animals brought from old zoo are',new_zoo[2])
print('Last animal brought from old zoo is',new_zoo[2][2])
print('Number of animal in the new zoo is',len(new_zoo)-1+len(new_zoo[2]))
```

输出：

```
Number of animals in the zoo is 3
Number of cages in the new zoo is 3
All animals in new zoo are ('monkey', 'camel', ('python', 'elephant', 'penguin'))
Animals brought from old zoo are ('python', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
Number of animal in the new zoo is 5
```

包含 0 或1个项目的元组 一个空的元组由一对圆括号构成，就像 myempty = () 这样。然而，一个只拥有一个项 目的元组并不像这样简单。你必须在第一个（也是唯一一个）项目的后面加上一个逗号 来指定它，如此一来Python才可以识别出在这个表达式想表达的究竟是一个元组还是只 是一个被括号所环绕的对象，也就是说，如果你想指定一个包含项目2的元组，你必 须指定 singleton = (2,) 。

> 字典

```
 # 'ab'是地址（Adress)薄（Book）的缩写
ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotamil.com'
}
print('Swaroop\'s address is', ab["Swaroop"])

# 删除一对键值-值配对
del ab['Spammer']

print('\nThere are {} contacts in the address-book\n'.format(len(ab)))

for name, address in ab.items():
    print('Contact {} at {}'.format(name, address))

# 添加一对键值-值配对
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print('\nGuido\'s address is', ab['Guido'])
```

结果：

```
Swaroop's address is swaroop@swaroopch.com

There are 3 contacts in the address-book

Contact Swaroop at swaroop@swaroopch.com
Contact Larry at larry@wall.org
Contact Matsumoto at matz@ruby-lang.org

Guido's address is guido@python.org
```

> 序列

```python
shoplist = ['apple', 'mango', 'carrot', 'banana']
name = 'swaroop'

# Indexing or 'Subsciption' operation #
# 索引或'下标(Subcsciption)'操作符 #
print('Item 0 is', shoplist[0])
print('Item 1 is', shoplist[1])
print('Item 2 is', shoplist[2])
print('Item 3 is', shoplist[3])
print('Item -1 is', shoplist[-1])
print('Item -2 is', shoplist[-2])
print('Item 0 is', shoplist[0])
print()
# Slicing on a list
print(shoplist)
print('Item 1 to 3 is',shoplist[1:3])
print('Item 2 to end is',shoplist[2:])
print('Item 1 to -1 is',shoplist[1:-1])
print('Item start to end is',shoplist[:])
print()
#从某一行开始切片
print(name)
print('characters 1 to 3 is',name[1:3])
print('character 2 to end is',name[2:])
print('charecter 1 to -1 is',name[1:-1])
print('character start to end is',name[:])

```

输出：

```
Item 0 is apple
Item 1 is mango
Item 2 is carrot
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Item 0 is apple
```

```

['apple', 'mango', 'carrot', 'banana']
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot', 'banana']

swaroop
characters 1 to 3 is wa
character 2 to end is aroop
charecter 1 to -1 is waroo
character start to end is swaroop
```

集合

集合（Set）是简单对象的无序集合（Collection）。当集合中的项目存在与否比起次序或其出 现次数更加重要时，我们就会使用集合。
通过使用集合，你可以测试某些对象的资格或情况，检查它们是否是其它集合的子集，找到 两个集合的交集，等等

```
>>>     bri     =set(['brazil', 'russia',       'india'])
>>>'india' in bri
True
>>>'usa' in bri
False
>>>bric = bri.copy()
>>>bric.add('china')
>>>bric.issuperset(bri)
True
>>>bri.remove('russia')
>>>bri & bric   #       OR      bri.intersection(bric)
{'brazil',      'india'}

```

引用

当你创建了一个对象并将其分配给某个变量时，变量只会查阅（Refer）某个对象，并且它也 不会代表对象本身。也就是说，变量名只是指向你计算机内存中存储了相应对象的那一部 分。这叫作将名称绑定（Binding）给那一个对象。
一般来说，你不需要去关心这个，不过由于这一引用操作困难会产生某些微妙的效果，这是 需要你注意的：

```
print('Simple Assignment')
shoplist=['apple','mango','carrot','banana']
#mylist 只是指向同一对象的另一种名称
mylist=shoplist

#我购买了第一项项目，所以我将其从列表中删除
del shoplist[0]

print('shoplist is',shoplist)
print('mylist is',mylist)

```

```
#注意到shoplist 和mylist 二者都打印出了其中都没有Apple的同样的列表
#以此我们确认他们指向的是同一个对象

print('copy by making a full slice')
#通过生成一份完整的切片制作一份列表的副本
mylist=shoplist[:]
#删除第一个项目
del mylist[0]

print('shoplist is',shoplist)
print('mylist is',mylist)
```

输出：

```
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

面向对象编程

类

最简单的类（Class）可以通过下面的案例来展示（保存为 oop_simplestclass.py ）：

```
class Person:
    pass # 一个空的代码块

p =     Person()
print(p)
```

我们通过使用class语句与这个类的名称来创建一个新类。在它之后是一个缩进的语句块，代表这个类的主体。在本案例中，我们创建的是一个空代码块，使用pass语句予以标明。
然后，我们通过采用类的名称后跟一对括号的方法，给这个类创建一个对象（或是实例，我 们将在后面的章节中了解有关实例的更多内容）。为了验证我们的操作是否成功，我们通过 直接将它们打印出来来确认变量的类型。结果告诉我们我们在Person 类的 __main__模块 中拥有了一个实例。
要注意到在本例中还会打印出计算机内存中存储你的对象的地址。案例中给出的地址会与你 在你的电脑上所能看见的地址不相同，因为 Python 会在它找到的任何空间来存储对象。

方法

我们已经在前面讨论过类与对象一如函数那般都可以带有方法（Method），唯一的不同在于 我们还拥有一个额外的self变量。现在让我们来看看下面的例子（保存为 oop_method.py ）

```
class Person:
    def say_hi(self):
        print('Hello, how       are      you?')
p=Person()
p.say_hi()
#前面两行同样可以写作
#Person().say_hi()
```

输出：

```
$ python oop_method.py Hello,
how      are      you?
```

init 方法

在 Python 的类中，有不少方法的名称具有着特殊的意义。现在我们要了解的就是 **init** 方法的意义。
__init__方法会在类的对象被实例化（Instantiated）时立即运行。这一方法可以对任何你想 进行操作的目标对象进行初始化（Initialization）操作。这里你要注意在 init 前后加上的双下 划线。

```
clas Person:
    def __init__(self,  name):
        self.name = name
        def     say_hi(self):
            print('Hello, my name is', self.name)
p = Person('Swaroop')
p.say_hi()
#前面两行同时也能写作
#Person('Swaroop').say_hi()
```

输出：

```
$ python oop_init.py
Hello, my name is Swaroop
```

类变量与对象变量

我们已经讨论过了类与对象的功能部分（即方法），现在让我们来学习它们的数据部分。数据部分——也就是字段——只不过是绑定（Bound）到类与对象的命名空间（Namespace） 的普通变量。这就代表着这些名称仅在这些类与对象所存在的上下文中有效。这就是它们被 称作 “命名空间” 的原因。
字段（Field）有两种类型——类变量与对象变量，它们根据究竟是类还是对象拥有这些变量 来进行分类。
类变量（Class Variable）是共享的（Shared）——它们可以被属于该类的所有实例访问。 该类变量只拥有一个副本，当任何一个对象对类变量作出改变时，发生的变动将在其它所有 实例中都会得到体现。
对象变量（Object variable）由类的每一个独立的对象或实例所拥有。在这种情况下，每个 对象都拥有属于它自己的字段的副本，也就是说，它们不会被共享，也不会以任何方式与其 它不同实例中的相同名称的字段产生关联。下面一个例子可以帮助你理解（保存为 oop_objvar.py ）

```python
class Robot:
    """标识有一个带有名字的机器人。"""
    #一个类变量，用来计数机器人的数量
    population=0

    def __init__(self,name):
        """初始化数据。"""
        self.name=name
        print("Initialiing {}".format(self.name))

        #当有人被创建时，机器人数量将会增加
        Robot.population+=1

    def die(self):
        """我挂了"""
        print("{} is being destroyed !".format(self.name))

        Robot.population-+1
        if Robot.population==0:
            print("{} was the last one".format(self.name))
        else:
            print("there are still {:d} robot working.".format(Robot.population))

    def say_hi(self):
        """来自机器人的诚挚问候

        没问题，你做到了。"""
        print("Greetings, my masters call me {}.".format(self.name))

    @classmethod
    def how_many(cls):
        """打印出当前的人口数量。"""
        print("we hava {:d} robots.".format(cls.population))

droid1=Robot("R2-D2")
droid1.say_hi()
Robot.how_many()

droid2=Robot("C-3PO")
droid2.say_hi()
Robot.how_many()

print("\nRobots can do some work here.\n")
print("Robots hava finished their work. So let's destroy them.")
droid1.die()
droid2.die()

Robot.how_many()

```

输出：

```
Initialiing R2-D2
Greetings, my masters call me R2-D2.
```

```
we hava 1 robots.
Initialiing C-3PO
Greetings, my masters call me C-3PO.
we hava 2 robots.

Robots can do some work here.

Robots hava finished their work. So let's destroy them.
R2-D2 is being destroyed !
there are still 2 robot working.
C-3PO is being destroyed !
there are still 2 robot working.
we hava 2 robots.
```

继承

```
class SchoolMember:
    '''代表任何学校里的成员。'''
    def __init__(self,name,age):
        self.name=name
        self.age=age
        print("(Initialized SchoolMember:{})".format(self.name))
    def tell(self):
        """告诉我有关我的细节"""
        print('Name :{} Age:{}'.format(self.name,self.age),end="")

class Teacher(SchoolMember):
    '''代表一位老师'''
    def __init__(self,name,age,salary):
        SchoolMember.__init__(self,name,age)
        self.salary=salary
        print("(Initialized Teacher:{})".format(self.name))
    def tell(self):
        SchoolMember.tell(self)
        print('Salary:{:d}'.format(self.salary))

class Student(SchoolMember):
    '''代表一位学生'''
    def __init__(self,name,age,marks):
        SchoolMember.__init__(self,name,age)
        self.marks=marks
        print("(Initialized Student:{})".format(self.name))
    def tell(self):
        SchoolMember.tell(self)
        print("Maks:{:d}".format(self.marks))

t=Teacher("Mrs.Shrividya",40,30000)
s=Student("Swaroop",25,75)

#打印一行
print()
members=[s,t]
for member in members:
    member.tell()


```

输出：

```
(Initialized SchoolMember:Mrs.Shrividya)
(Initialized Teacher:Mrs.Shrividya)
(Initialized SchoolMember:Swaroop)
(Initialized Student:Swaroop)

Name :Swaroop Age:25Maks:75
Name :Mrs.Shrividya Age:40Salary:30000
```

输入与输出

用户输入内容

```
def reverse(text):
    return text[::-1]
def is_palindrome(text):
    return text==reverse(text)

something=input("Enter text:")
if is_palindrome(something):
    print("Yes,it is a palindrom")
else:
    print("No,it is not a palindrom")
```

文件

```
 poem='''\
Programming is fun
when the work is done
if you wanna make your work also fun:
use python
'''

#打开文件以编辑('w'riting)
f=open('C:/Users/shenyonghui/Desktop/poem.txt',"w")
#向文件中编写文本
f.write(poem)
#关闭文件
f.close()

#如果没有特别指定
#将假定启用默认的阅读（'r'ead)模式
f=open('poem.txt')

```

```
while True:
    line=f.readline()
    #零长度指示 EOF
    if len(line)==0:
        break
    #每行‘line'的结尾 都已经有了换行符 因为它是从一个文件中进行读取的
    print(line,end='')
#关闭文件
f.close()
```

Pickle

Python 提供了一个叫作 Pickle 的标准模块，通过它你可以将任何纯 Python 对象存储到一 个文件中，并在稍后将其取回。这叫作持久地（Persistently）存储对象。

```
import pickle

#我们储存相关对象的文件的名称
shoplistfile='shoplist.data'
#需要购买的物品清单
shoplist=['apple','mango','carrot','na']

#准备写入文件
f=open(shoplistfile,"wb")
#转存储对象指文件
pickle.dump(shoplist,f)
f.close()

#清除shoplist变量
del shoplist

#重新打开储存文件
f=open(shoplistfile,"rb")
#从文件中载入对象
storedlist=pickle.load(f)
f.close()
print(storedlist)
```

异常

处理异常

```
try:
    text= input('Enter something -->')
except EOFError:
    print('Why did you do an EOF on me?')
except KeyboardInterrupt:
    print('You cancelled the operation.')
else:
    print('You  entered {}'.format(text))

```

抛出异常

```
#encoding=UTF-8
class ShortInputException(Exception):
    '''由用户定义的异常类'''
    def __init__(self,length,atleast):
        Exception.__init__(self)
        self.length=length
        self.atleast=atleast

try:
    text=input('Enter something-->')
    if len(text)<3:
        raise ShortInputException(len(text),3)
    #其它工作能在此处继续正常进行
except EOFError:
    print("Why did you do an EOF on me?")
except ShortInputException as ex:
    print(('ShortInputException:The input was '+'{0} long ,expected at least {1}').format(ex.lengt
else:
    print('No exception was raised.')

```

Try ... Finally

```
import sys
import time

f=None
try:
    f=open("poem.txt")
    #我们常用的文件阅读风格
    while True:
        line=f.readline()
        if len(line)==0:
            break
        print(line,end='')
        sys.stdout.flush()
        print('press ctrl+c now')
        #为了确保它能进行一段时间
        time.sleep(2)
except IOError:
    print("Could not find file poem.txt")
except KeyboardInterrupt:
    print('!!You cancelled the reading from the file.')
finally:
    if f:
        f.close()
    print("(Cleaning up:Closed the file)")

```

输出:

```
Programming is fun
press ctrl+c now
when the work is done
press ctrl+c now
if you wanna make your work also fun:
press ctrl+c now
use python
press ctrl+c now
(Cleaning up:Closed the file)
```

> with 语句
> 在try块中获取资源，然后在finally块中释放资源是一种常见的模式。因此，还有一个with语句使得这一过程可以以一种干净的姿态得以完成。

```
with open("poem.txt") as f:
    print(f)
    for line in f:
        print(line,end='')
```