【Python】No.4 与三剑客无关的Python基础内容

丁诗琦 NAU Analysts 2020-03-01 16:25

上一篇介绍了Pandas——一种包含金融时间序列的金融数据分析工具，这次介绍与三剑客无关的python基础内容，以供新手们更多的了解python。

四大基本容器

序列是Python中最基本的数据结构。序列中的每个元素都分配一个数字——它的位置，或索引，第一个索引是0，第二个索引是1，依此类推。Python有6个序列的内置类型，但最常见的是列表和元组。首先，什么叫做容器呢？容器顾名思义就是里面可以容纳很多东西，可以理解成已经被包装好的一个盒子。Python中的container是一种内置的封装好的数据结构，是可以直接拿来使用，不需要自己构造。当然是容器也是可以通过自己构造的。

一、列表（list）

列表作为最常用的数据结构，它不仅可以包纳各种类型的数据，而且在list当中，数据是有序的，与字符串的索引一样，列表索引从0开始。列表可以进行截取、组合等。

1. 索引，切片（步长）

索引：。

与字符串的索引相同，列表索引从0开始。

切片：

list[开始索引值:结束索引值:步长]

注意：“顾头不顾尾”的特性。方向取值步长应为负值且不可省略，返回的数据也是逆向的。

```
a[0:4:2]
[1,3]
```

2.常用的方法：

方法	用法	实例
Append	在列表尾部 增加新的元素	a.append(6) a=[1,2,3,4,5] #输出结果[1,2,3,4,5,6]#
Index	获取某个元素的 索引	a.index（1） #输出结果 0 #
Clear	删除列表所有 数目	a.clear（） #输出结果 a #
Insert	在特定位置插入 数据	a.insert（） #输出结果 [1,2,5,3,4,5] #
Len	获取列表数据 数目	len(a) #输出结果 5 #

二、集合（set）

集合当中的数据是无序的，而且是不可以重复的，用{}表示。

常用的方法：

方法	用法	实例
Add	添加元素	c={1,5,7,8,3,0,2} c.add("python") #输出结果{0,1,2,3,5,7,8,'python'}#
Remove	移除指定元素	c={1,5,7,8,3,0,2} c.remove（3）

		3 #输出结果{0,1,2,5,7,8}#
Intersection	交集	1 c={1,5,7,8,3,0,2} 2 d={5,7,9} 3 c.intersection(d) 4 #输出结果{5,7}#
difference	差集	1 c={1,5,7,8,3,0,2} 2 d={5,7,9} 3 c.difference(d) 4 #输出结果{0,1,2,3,8}#
union	并集	1 c={1,5,7,8,3,0,2} 2 d={5,7,9} 3 c.union(d) 4 #输出结果{1,2,3,5,7,8,9}#

三、元组（uple）

Python元组与列表类似，不同之处在于元组的元素不能修改。

```
x=(7,23,46,24,145,764)
#输出结果(7,23,46,24,145,764)#
```

常用方法：

1. count # 统计元素出现的次数

```
x.count(24)
#输出结果  1  #
```

2. index # 查找元素在列表中的位置，注意：如果元素不存在，则显示异常，如果该元素存在多个，则返回第一个。

```
x.index(145)
#输出结果  4  #
```

3.元组当中的元素是不可以单个删除的，但是我们可以用del语句来删除整个元组。

```
tup = ('Google', 'Runoob', 1997, 2000)
print (tup)
del tup print ("删除后的元组 tup : ")
print (tup)l
```

```
删除后的元组 tup :
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-279adc8d5f55> in <module>
      4 del tup
      5 print ("删除后的元组 tup : ")
----> 6 print (tup)

NameError: name 'tup' is not defined
```

注意：在Python中，如果只有一个元素，并且该元素是一个数字，则需要加上一个逗号来表示，该变量是元组而不是数字。

```
z=(27,)
#输出结果  (27,)  #
```

四、字典（dict）

需要注意的点：

- 使用键-值 （key-value）存储，具有极快的查找速度。
- 键和值键必须是唯一的，但值则不必。
- 值可以取任何数据类型，如字符串，数字或元组。
- dict内部存放的顺序和key放入的顺序是没有关系的。

```
i={"姓名":"吴亦凡","年龄":"20","性别":"男"}
i
#输出结果  {"姓名":"吴亦凡","年龄":"20","性别":"男"}  #
```

常见方法:

- copy（复制、拷贝对象）

```
i={"姓名":"吴亦凡","年龄":"20","性别":"男"}
k=i.copy()
k
#输出结果  {"姓名":"吴亦凡","年龄":"20","性别":"男"}  #
```

注意：Python字典提供了内置函数copy方法，实现对字典的复制操作。字典复制分为浅复制和深复制两种方式。浅复制只是把当前字典内的子对象的引用复制给新字典，当被复制字典的子对象内容改变时，新字典的子对象内容也会改变；深复制把当前字典的全部内容，也包括子对象的内容都完全复制给新字典，被复制字典和新字典是两个独立的数据对象，被复制字典内容的改变不会影响新字典的内容。浅复制用copy方法，深复制用deepcopy方法。

- popitem（随机删除一个键值对）

```
i={"姓名":"吴亦凡","年龄":"20","性别":"男"}
i.popitem()
#输出结果  {"姓名":"吴亦凡","年龄":"20"}  #
```

- values（以列表返回一个字典所有的值）

```
m={"姓名":"吴亦凡","年龄":"20"}
m.values()
#输出结果  dict_valuse(['吴亦凡','25'])#
```

- items（以列表返回可遍历的元组数组）

```
m={"姓名":"吴亦凡","年龄":"20"}
m.items()
#输出结果
dict_items([('姓名', '吴亦凡'),('年龄', '25')
```

循环判断

一、if判断

计算机之所以能做很多自动化的任务，因为它可以自己做条件判断,结构为：

```
if 判断条件 #进行判断条件满足之后执行下方语句
  执行语句
elif 判断条件 #在不满足上面所有条件基础上进行条件筛选匹配之后执行下方语句
  执行语句
else #再不满足上面所有的添加下执行下方语句
  执行语句
```

举例：flag在不满足第一个if判断条件之后，也不满足elif判断条件,最终在else中执行print为flag已倒。

```
flag = true
if flag:
        print('flag还在')
elif 1 == 2:
        print('考研失败')
else:
        print('flag已倒')
#输出结果 flag已倒 #
```

二、for循环

需要注意的点：

- for语句是python中的循环控制语句，是有限循环。
- for还附带else块，主要是用来处理for语句中的break语句。
- 在for语句块中间，可以使用break和continue来控制循环，break会跳出整个循环，continue则是结束一次循环，进行下一次循环。for还附带else块，主要是用来处理for语句中的break语句。

举例：for循环编写登陆程序：编写一个登陆程序，允许用户登陆三次，如果三次输入错误，则退出程序。

```
_use = "CFA"
_passwd = "NAU"
for i in range(3):
        username = input("Username: ")
        password = input("Password: ")
        if username == _use and password == _passwd:
                print("Welcome %s login..." % username)
                break
        else:
                print("Username or Password is Falst")
else:
        print("test login too many!!!")
```

执行之后会出现登陆情况：

```
Username: CFA
Password: NAU
Welcome CFA login...
```

此外，for循环最常用来循环遍历集合中的内容,我们借用len函数举一个例子：

```
cities = ["shanghai","beijing","nanjing","shenzhen"]
for i in range(len(cities)):
        print(i+1,cities[i])
```

可以看到输出的结果为：

```
1 shanghai
2 beijing
3 nanjing
4 shenzhen
```

三、while循环

1. While循环结构：

只要条件符合，就可以一直执行动作。这种循环叫做 死循环，一经触发，只要条件符合，就无限循环。此条件实际上就是布尔值-->True、False。若想进行某判断不定期结束循环，可设定变量为布尔值（True），达到目的想要结束时，重新设定布尔值（False）以结束循环。

```
while 条件:
动作
```

循环在日常的使用中，还需要配合几个表达方法，分别是：break、continue、end、else。

2.break的用法：

终止循环，当循环碰到break就会立即终止。

举例：

print1到10的整数，当num==4的时候，终止循环。

```
num = 1
while num <= 10:
        print(num)
        num = num + 1
        if num ==4:
                break
```

结果是：

```
1
2
3
```

3.continue的用法：

跳出次循环，之后的循环继续，不受到影响。

举例：

```
num = 1
while num <= 10:
        num = num + 1
        if num == 4:
                continue
        print(num)
```

结果是：

```
2
3
5
6
7
8
9
10
11
```

此循环首先是加一次1再进行判断打印，所以第一次打印出来的是'2'，当'num=4'时，continue跳过或者说是结束了这次循环，不执行continue的结果，所以不打印'4'，当最后一次循环的时候'num = 10'，和第一次一样，是先加1，再进行判断打印，所以会出现'11'。

4.else的用法：

当循环正常结束时，最后再执行else后边的动作。正常结束指循环不报错或不碰到break。

```
while...else...结构:
while  条件:
            动作
else:
            动作
```

举例：

```
1   num  =  1
2   while  num  <=  10:
3           num  =  num  +  1
4           if  num  ==  4:
5                   continue
6           print(num)
7   else:
8           print("This  is  else  statement")
```

结果是：

```
2
3
5
6
7
8
9
10
11
This is else statement
```

异常机制

一、什么是异常

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。

一般情况下，在Python无法正常处理程序时就会发生一个异常。异常是Python对象，表示一个错误。当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

二、Python中的异常类型

当我们在敲完代码，运行之后可能会出现一些errors，这些就是异常机制在起作用向我们报错。

1、NameError：尝试访问一个未申明的变量

```
NameError: name 'v' is not defined
```

2、ZeroDivisionError：除数为0

```
ZeroDivisionError: int division or modulo by zero
```

3、SyntaxError：语法错误

```
SyntaxError: invalid syntax (<pyshell#14>, line 1)
```

4、IndexError：索引超出范围

```
IndexError: list index out of range
```

5、KeyError：字典关键字不存在

6、IOError：输入输出错误

```
IOError: [Errno 2] No such file or directory: 'abc'
```

7、AttributeError：访问未知对象属性

举例：

```
>>> class Worker:
def Work():
print("I am working")
>>> w = Worker()
>>> w.a
Traceback (most recent call last):
File "<pyshell#51>", line 1, in <module>
w.a
AttributeError: 'Worker' object has no attribute 'a'
```

8、ValueError：数值错误

举例：

```
>>> int('d')
Traceback (most recent call last):
File "<pyshell#54>", line 1, in <module>
int('d')
ValueError: invalid literal for int() with base 10: 'd'
```

9、TypeError：类型错误

举例：

```
>>> iStr = '22'
>>> iVal = 22
```

```
>>> obj = iStr + iVal;
Traceback (most recent call last):
File "<pyshell#68>", line 1, in <module>
obj = iStr + iVal;
TypeError: Can't convert 'int' object to str implicitly
```

10、AssertionError：断言错误

三、异常解决机制：

1.默认的异常处理器：

如果我们没有对异常进行任何预防，那么在程序执行的过程中发生异常，就会中断程序，调用python默认的异常处理器，并在终端输出异常信息。

举例：

```
1  <span style="font-size:18px;">s = 'hello girl!'
2  print s[100]
3  print 'continue</span>'
```

这样的情况之下，第三条代码并不会执行。

2.try…except

举例：

```
1  def main():
2          a=3/0
3          print(a)
4  try:
5          main()
6  except ZeroDivisionError:
7          print("error")
```

try/except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理，如果没有try/except，程序发生错误会终止运行，且抛出不友好的traceback，有了该语句，try语句发生错误，只会打印出“error”提示。当程序执行到第二句的时候，会发现进入了try语句，但是执行try语句当中出现了异常，回到try语句层，寻找在此之后有没有except语句。找到except语句以后，调用系统自定义的异常处理器，之后会继续执行程序，此时的两个print语句都会执行下去。except后面也可以为空，表示捕获任何类型的异常。所以本例最后结果就是error。

3.try…finally

举例：

```
1  try:
2          fh = open("testfile", "w")
3          try:
4                  fh.write("这是一个测试文件，用于测试异常!!")
5          finally:
6                  print ("关闭文件")
7                  fh.close()
8  except IOError:
9          print ("Error: 没有找到文件或读取文件失败")
```

最后的结果就是：关闭文件

finally语句表示，无论是否会发生异常，finally里面语句都必须执行。但是，由于没有except，finally执行完后程序便中断。

4. Assert

断言，就是判断这个expression这个表达式语句是否正确。

先判断assert后面紧跟的语句是True还是False，如果是True则继续执行print，如果是False则中断程序，调用默认的异常处理器，同时输出assert语句逗号后面的提示信息。提示error，后面的print不执行。

举例：

```
1  def testAssert(x):
2          assert x < 1, "Invalid value"
3          print("程序无异常才会执行到这里")
4
5  testAssert(2)
```

结果就是：AssertionError

5. with…as

在使用类似文件的流对象时，使用完毕后要调用close方法关闭,这里with...as语句提供了一个非常方便的替代方法，open打开文件后将返回的文件流对象赋值给f，然后在with语句块中使用，之后会自动关闭文件

如果with语句发生异常，会调用默认的异常处理器，文件也会正常关闭，但后面的print语句不执行

举例：

```
1  with open('a.txt', 'w') as f:
2          f.write('Hello!')
```

类和函数

一、总览

- Python的函数是什么？

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。我们已经知道Python提供了许多内建函数，比如print()。但也可以自己创建函数，这被叫做用户自定义函数。

- Python的类是什么？

类 ：是一个独立存放变量（属性/方法）的空间，使用运算符’ . ’来调用类的属性和方法。

二、如何定义一个函数？

- 函数代码块以def关键词开头，后接函数标识符名称和圆括号()。
- 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
- 函数内容以冒号起始，并且缩进。

- return［表达式］ 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

介绍一个较为简单的函数：

```
def sun(x,y)
return x+y
```

输出结果为：

```
a = sum(1,2)
3
```

在上期中有介绍lambda函数，在这里就不再赘述了。

三、类：

类的定义和函数类似，需要使用一个关键字“class”来定义。

```
class  className:
       [statement]
```

当我们调用类时，需要进行类的实例化，即创建一个类的实例。Python实例化后，会自动调用__init__函数：

__init__函数（方法）

1.首先说一下，带有两个下划线开头的函数是声明该属性为私有,不能在外部被使用或直接访问。

2.init函数（方法）支持带参数的类的初始化 ，也可为声明该类的属性

3.init函数（方法）的第一个参数必须是 self（self为习惯用法，也可以用别的名字），后续参数则可以自由指定，和定义函数没有任何区别。

举例：

```
class Myclass:
      def __init__(self):
            print('init')
print(1,Myclass)#不会调用
print(2,Myclass())#调用_init_
a = Myclass()#调用_init_
```

输出的结果为：

```
1 <class '__main__.Myclass'>
init
2 <__main__.Myclass object at 0x000001829ECA6E80>
init
```

需要注意的是类的私有属性，类中所有以“ _ ”开头的方法、参数，均不可在类的外部直接调用，所有以“ ”开头的方法、参数均不可使用“from module import * ”来直接调用。

本期内容到这就结束了，希望这期文章能够帮助到学习python或者是对于python感兴趣的小伙伴，从小白晋升到大佬！拭目以待！

本期撰写人：丁诗琦
南京审计大学2018级CFA3班
邮箱：2205363925@qq.com

南京审计大学CFA

推进CFA学术交流，集聚南审CFA校友资源