第4章 操作列表

在第3章，学习了如何创建简单的列表，还学习了如何操作列表元素。 在本章中，将学习如何遍历整个列表，这只需要几行代码，无论列表有多长。 循环能够对列表的每个元素都采取一个或一系列相同的措施，从而高效地处理任何长度的列表，包括包含数千乃至数百万个元素的列表。

4.1 遍历整个列表

你经常需要遍历列表的所有元素，对每个元素执行相同的操作。 例如，在游戏中，可能需要将每个界面元素平移相同的距离； 对于包含数字的列表，可能需要对每个元素执行相同的统计运算；

在网站中，可能需要显示文章列表中的每个标题。 需要对列表中的每个元素都执行相同的操作时，可使用Python中的for循环。

In [1]:
```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

```
alice
david
carolina
```

4.1.1 深入研究循环

循环这种概念很重要，因为它是让计算机自动完成重复工作的常见方式之一。 例如，在前面使用的简单循环里，Python将首先读取其中的第一行代码：

```
for magician in magicians:
```

这行代码让Python获取列表magicians 中的第一个值'alice' ，并将其与变量magician 相关联。 接下来，Python读取下一行代码：

```
    print(magician)
```

Python再次打印变量magician的值，当前为'david' 。 接下来，Python再次执行整个循环，对列表中的最后一个值'carolina'进行处理。 至此，列表中没有其他的值了，因此Python接着执行程序的下一行代码。

4.1.2 在for 循环中执行更多操作

In [2]:
```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
```

```
Alice, that was a great trick!
David, that was a great trick!
Carolina, that was a great trick!
```

4.1.3 在for 循环结束后执行一些操作

In [3]:
```
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
```

```
    print(f"I can't wait to see your next trick, {magician.title()}.\n")
print("Thank you, everyone. That was a great magic show!")
```

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!
```

4.2 避免缩进错误

Python根据缩进来判断代码行与前一个代码行的关系。 在前面的示例中，向各位魔术师显示消息的代码行是for循环的一部分，因为它们缩进了。 Python通过使用缩进让代码更易读。简单地说，它要求你使用缩进让代码整洁而结构清晰。 在较长的Python程序中，你将看到缩进程度各不相同的代码块，从而对程序的组织结构有大致的认识。

4.2.1 忘记缩进

对于位于for 语句后面且属于循环组成部分的代码行，一定要缩进。 如果忘记缩进，Python会提醒你：

In [4]:
```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
print(magician)
```

```
  Input In [4]
    print(magician)
    ^
IndentationError: expected an indented block
```

4.2.2 忘记缩进额外的代码行

有时候，循环能够运行且不会报告错误，但结果可能出人意料。 试图在循环中执行多项任务，却忘记缩进其中的一些代码行时，就会出现这种情况。

In [26]:
```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
print(f"I can't wait to see your next trick, {magician.title()}.\n")
```

```
Alice, that was a great trick!
David, that was a great trick!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

4.2.3 不必要的缩进

如果你不小心缩进了无须缩进的代码行，Python将指出这一点：

In [6]:
```
message = "Hello Python world!"
    print(message)
```

```
  Input In [6]
```

```
    print(message)
    ^
IndentationError: unexpected indent
```

4.2.4 循环后不必要的缩进

如果不小心缩进了应在循环结束后执行的代码，这些代码将针对每个列表元素重复执行。 在有些情况下，这可能导致Python报告语法错误，但在大多数情况下，这只会导致逻辑错误。

In [7]:
```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")
    print("Thank you everyone, that was a great magic show!")
```

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

Thank you everyone, that was a great magic show!
David, that was a great trick!
I can't wait to see your next trick, David.

Thank you everyone, that was a great magic show!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you everyone, that was a great magic show!
```

4.3 创建数值列表

需要存储一组数的原因有很多。例如，在游戏中，需要跟踪每个角色的位置，还可能需要跟踪玩家的几个最高得分； 在数据可视化中，处理的几乎都是由数（如温度、距离、人口数量、经度和纬度等）组成的集合。列表非常适合用于存储数字集合，而Python提供了很多工具，可帮助你高效地处理数字列表。 明白如何有效地使用这些工具后，即便列表包含数百万个元素，你编写的代码也能运行得很好。

4.3.1 使用函数range()

Python函数range() 让你能够轻松地生成一系列数。例如，可以像下面这样使用函数range() 来打印一系列数：

4.3.2 使用range() 创建数字列表

要创建数字列表，可使用函数list() 将range() 的结果直接转换为列表。 如果将range() 作为list() 的参数，输出将是一个数字列表。

在前一节的示例中，只是将一系列数打印出来。要将这组数转换为列表，可使用list() ：

In [30]:
```
for i in range(0,6):
    print(i)
```

```
0
1
2
3
4
5
```

In [9]:
```
numbers = list(range(1, 6))
print(numbers)
```

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

使用函数range() 时，还可指定步长。为此，可给这个函数指定第三个参数，Python将根据这个步长来生成数。 例如，下面的代码打印1～10的偶数：

In [10]:
```
even_numbers = list(range(2, 11, 2))
print(even_numbers)
```

```
[2, 4, 6, 8, 10]
```

使用函数range() 几乎能够创建任何需要的数集。例如，如何创建一个列表，其中包含前10个整数（1～10）的平方呢？

In [11]:
```
squares = []
for i in range(1, 11):
    squares.append(i**2)

print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

4.3.3 对数字列表执行简单的统计计算

In [12]:
```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
print(min(digits))
print(max(digits))
print(sum(digits))
```

```
0
9
45
```

4.3.4 列表解析

前面介绍的生成列表squares 的方式包含三四行代码，而列表解析让你只需编写一行代码就能生成这样的列表。 列表解析将for循环和创建新元素的代码合并成一行，并自动附加新元素。 面向初学者并非都会介绍列表解析，这里之所以介绍列表解析，是因为等你开始阅读他人编写的代码时，很可能会遇到它。

In [13]:
```
squares = [value**2 for value in range(1, 11)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [36]:
```
squares = [value**2 for value in range(1, 11) if value%2 == 0]
print(squares)
```

```
[4, 16, 36, 64, 100]
```

In [37]:
```
squares = []
for value in range(1,11):
    if value%2 == 0:
        squares.append(value**2)

print(squares)
```

```
[4, 16, 36, 64, 100]
```

4.4 使用列表的一部分

在第3章中，学习了如何访问单个列表元素。在本章中，一直在学习如何处理列表的所有元素。其实还可以处理列表的部分元素，Python称之为切片。

4.4.1 切片

要创建切片，可指定要使用的第一个元素和最后一个元素的索引。与函数range()一样，Python在到达第二个索引之前的元素后停止。 要输出列表中的前三个元素，需要指定索引0和3，这将返回索引为0、1和2的元素。 下面的示例处理的是一个运动队成员列表：

In [2]:
```
players = ['王楠', '张怡宁', '刘诗雯', '丁宁', '伊藤美诚','福原爱']
print(players[0:3])
```

```
['王楠', '张怡宁', '刘诗雯']
```

可以生成列表的任意子集。例如，如果要提取列表的第二、第三和第四个元素，可将起始索引指定为1 ，并将终止索引指定为4

In [16]:
```
players = ['王楠', '张怡宁', '刘诗雯', '丁宁', '伊藤美诚','福原爱']
print(players[1:4])
```

```
['张怡宁', '刘诗雯', '丁宁']
```

In [5]:
```
#如果没有指定第一个索引，Python将自动从列表开头开始：
print(players[:])


print(players[3:])
print(players[0:7:2])
print(players[-1::-1])
```

```
['王楠', '张怡宁', '刘诗雯', '丁宁', '伊藤美诚', '福原爱']
['丁宁', '伊藤美诚', '福原爱']
['王楠', '刘诗雯', '伊藤美诚']
['福原爱', '伊藤美诚', '丁宁', '刘诗雯', '张怡宁', '王楠']
```

4.4.2 遍历切片

如果要遍历列表的部分元素，可在for循环中使用切片。 下面的示例遍历前三名队员，并打印他们的名字：

In [18]:
```
players = ['王楠', '张怡宁', '刘诗雯', '丁宁', '伊藤美诚','福原爱']
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.title())
```

```
Here are the first three players on my team:
王楠
张怡宁
刘诗雯
```

在很多情况下，切片都很有用。 例如，编写游戏时，你可以在玩家退出游戏时将其最终得分加入一个列表中，然后将该列表按降序排列以获取三个最高得分，再创建一个只包含前三个得分的切片； 处理数据时，可使用切片来进行批量处理；编写Web应用程序时，可使用切片来分页显示信息，并在每页显示数量合适的信息。

4.4.3 复制列表

我们经常需要根据既有列表创建全新的列表。 下面来介绍复制列表的工作原理，以及复制列表可提供极大帮助的一种情形。 要复制列表，可创建一个包含整个列表的切片，方法是同时省略起始索引和终止索引（[:]

）。 这让Python创建一个始于第一个元素、终止于最后一个元素的切片，即整个列表的副本。 例如，假设有一个列表包含你最喜欢的四种食品，而你想再创建一个列表，并在其中包含一位朋友喜欢的所有食品。 不过，你喜欢的食品，这位朋友也都喜欢，因此可通过复制来创建这个列表：

In [19]:
```
my_foods = ['pizza', 'pasta', 'carrot cake']
friend_foods = my_foods
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'pasta', 'carrot cake']

My friend's favorite foods are:
['pizza', 'pasta', 'carrot cake']
```

In [20]:
```
my_foods = ['pizza', 'pasta', 'carrot cake']
friend_foods = my_foods

my_foods.append('beef combo')
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'pasta', 'carrot cake', 'beef combo']

My friend's favorite foods are:
['pizza', 'pasta', 'carrot cake', 'beef combo']
```

In [21]:
```
my_foods = ['pizza', 'pasta', 'carrot cake']
friend_foods = my_foods[:]
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'pasta', 'carrot cake']

My friend's favorite foods are:
['pizza', 'pasta', 'carrot cake']
```

4.5 元组(Tuple)

列表非常适合用于存储在程序运行期间可能变化的数据集。 列表是可以修改的，这对处理网站的用户列表或游戏中的角色列表至关重要。 然而，有时候你需要创建一系列不可修改的元素，元组可以满足这种需求。 Python将不能修改的值称为不可变的 ，而不可变的列表被称为元组 。

4.5.1 定义元组

元组看起来很像列表，但使用圆括号而非中括号来标识。 定义元组后，就可使用索引来访问其元素，就像访问列表元素一样。

In [6]:
```
dimensions = (200, 50)
dimensions[0] = 250
```

```
---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)
Input In [6], in <cell line: 2>()
      1 dimensions = (200, 50)
----> 2 dimensions[0] = 250

TypeError: 'tuple' object does not support item assignment
```

4.5.2 遍历元组中的所有值

In [23]:
```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

```
200
50
```

元组 vs. 列表

- 列表是动态的，属于可变序列，它的元素可以随时增加、修改或者删除，而元组是静态的，属于不可变序列，无法增加、删除、修改元素，除非整体替换。
- 列表可以使用append()、extend()、insert()、remove()和pop()等方法实现添加和修改列表元素，而元组则没有这几个方法，因为不能向元组中添加和修改元素。同样，也不能删除元素，可以整体替换。
- 列表可以使用切片访问和修改列表中的元素。元组也支持切片，但是它只支持通过切片访问元组中的元素，不支持修改。
- 元组比列表的访问和处理速度快。所以如果只需要对其中的元素进行访问，而不进行任何修改，建议使用元组而不使用列表。
- 因为列表可以修改，元组不可以修改，因此元组比列表具有更高的安全性。
- 列表不能作为字典的键，而元组可以。
- 存储方式不同：空列表比空元组多占用16个字节。

In [24]:
```
import sys
l = []
print(sys.getsizeof(l))
t = ()
print(sys.getsizeof(t))
```

```
56
40
```

https://blog.csdn.net/qq_43797487/article/details/123879963

4.7 小结

在本章中，学习了： 如何高效地处理列表中的元素； 如何使用for循环遍历列表，Python如何根据缩进来确定程序的结构，以 及如何避免一些常见的缩进错误； 如何创建简单的数字列表，以及可对数字列表执行的一些操作； 如何通过切片来使用列表的一部分和复制列表。

还学习了元组（它对不应变化的值提供了一定程度的保护），以及在代码变得越来越复杂时如何设置格式，使其易于阅读。

在第5章中，将学习如何使用if语句在不同的条件下采取不同的措施； 如何将一组较复杂的条件测试组合起来，并在满足特定条件时采取相应的措施。 还将学习如何在遍历列表时，通过使用if语句对特定元素采取特定的措施。

In [25]:
```
# The end
```