第7章 用户输入和while 循环

大多数程序旨在解决最终用户的问题，为此通常需要从用户那里获取一些信息。 例如，假设有航空公司购票系统。 要编写回答这个问题的程序，就需要知道用户的信息。 因此，这种程序需要让用户输入个人信息，以判断用户是否可以购票，从而给出订单。

在本章中，你将学习如何接受用户输入，以便程序进行处理。

程序需要旅客输入信息，为此，将使用input()函数 。 还将学习如何让程序不断地运行，以便用户根据需要输入信息，并在程序中使用这些信息。 为此，将使用while循环让程序不断运行，直到指定的条件不满足为止（机票已买光）。 通过获取用户输入并学会控制程序的运行时间，就能编写出交互式程序。

7.1 函数input() 的工作原理

函数input()让程序暂停运行，等待用户输入一些文本。 获取用户输入后，Python将其赋给一个变量，以方便你使用。 例如，下面的程序让用户输入一些文本，再将这些文本呈现给用户：

In [ ]:
```
message = input("Tell me something, and I will repeat it back to you: ")
print(message)
```

函数input() 接受一个参数——要向用户显示的提示 （prompt）或说明，让用户知道该如何做。 在本例中，Python运行第一行代码时，用户将看到提示Tell me something, and I will repeatit back to you: 。 程序等待用户输入，并在用户按回车键后继续运行。 输入被赋给变量message，接下来的print(message)将输入呈现给用户.

注意Sublime Text等众多编辑器不能运行提示用户输入的程序。 你可以使用Sublime Text来编写提示用户输入的程序，但必须从终端运行它们。

7.1.1 编写清晰的程序

每当使用函数input()时，都应指定清晰易懂的提示，准确地指出希望用户提供什么样的信息——指出用户应该输入何种信息的任何提示都行，如下所示：

In [ ]:
```
name = input("Please enter your name: ")
print(f"\nHello, {name}!")
```

有时候，提示可能超过一行。 例如，你可能需要指出获取特定输入的原因。 在这种情况下，可将提示赋给一个变量，再将该变量传递给函数input()。 这样，即便提示超过一行，input()语句也会非常清晰。

In [ ]:
```
prompt = "If you tell us who you are, we can personalize the messages you see."
prompt += "\nWhat is your first name? "
name = input(prompt)
print(f"\nHello, {name}!")
```

7.1.2 使用int() 来获取数值输入

使用函数input() 时，Python将用户输入解读为字符串。 请看下面让用户输入年龄的解释器会话：

In [ ]:
```
age = input("How old are you? ")
```

```
print(type(age))
```

In [ ]:
```
age = input("How old are you? ")

if age >= 18:
    print("you can buy alcohol")
```

试图将输入用于数值比较时，Python会引发错误，因为它无法将字符串和整数进行比较： 不能将赋给age 的字符串'21' 与数值18进行比较。

为解决这个问题，可使用函数int()，它让Python将输入视为数值。 函数int() 将数的字符串表示转换为数值表示，如下所示：

In [ ]:
```
height = input("How tall are you, in cm? ")
height = float(height)
if height >= 120:
    print("\nYou're tall enough to ride!")
else:
    print("\nYou'll be able to ride when you're a little taller.")
```

7.2 while 循环简介

for 循环用于针对集合中的每个元素都执行一个代码块，而while循环则不断运行，直到指定的条件不满足为止。

7.2.1 使用while 循环

可使用while 循环来数数。例如，下面的while 循环从1数到5：

In [ ]:
```
current_number = 1
while current_number <= 5:
    if current_number == 3:
        continue
        print(current_number)
        current_number += 1
```

在第一行，将1 赋给变量current_number，从而指定从1开始数。 将接下来的while 循环设置成：只要current_number小于或等于5，就接着运行这个循环。 循环中的代码打印current_number的值，再使用代码current_number += 1（代码current_number = current_number + 1的简写）将其值加1。 只要满足条件current_number <= 5，Python就接着运行这个循环。 因为1小于5，所以Python打印1并将current_number加1，使其为2； 因为2小于5，所以Python打印2并将current_number加1，使其为3； 依此类推。一旦current_number 大于5，循环就将停止。

7.2.2 让用户选择何时退出

可以使用while循环让程序在用户愿意时不断运行，如下面的程序parrot.py所示。 我们在其中定义了一个退出值，只要用户输入的不是这个值，程序就将接着运行：

In [ ]:
```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program."

message = ""
while message != 'quit':
    message = input(prompt)
```

```
    message = message.lower()
    print(message)
```

7.2.3 使用标志

在前一个示例中，我们让程序在满足指定条件时执行特定的任务。 但在更复杂的程序中，很多不同的事件会导致程序停止运行。 在这种情况下，该怎么办呢？

In [ ]:
```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "

active = True

while active:
    message = input(prompt)
    if message == 'quit':
        active = False
    else:
        print(message)
```

7.2.4 使用break退出循环

要立即退出while 循环，不再运行循环中余下的代码，也不管条件测试的结果如何，可使用break语句。 break语句用于控制程序流程，可用来控制哪些代码行将执行、哪些代码行不执行，从而让程序按你的要求执行你要执行的代码。 例如，来看一个让用户指出他到过哪些地方的程序。在这个程序中，可在用户输入'quit'后使用break语句立即退出while循环：

In [ ]:
```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "

while True:
    city = input(prompt)
    if city == 'quit':
        break
    else:
        print(f"I'd love to go to {city.title()}!")
```

注意在任何Python循环中都可使用break语句。 例如，可使用break语句来退出遍历列表或字典的for循环。

7.2.6 避免无限循环

每个while 循环都必须有停止运行的途径，这样才不会没完没了地执行下去。 例如，下面的循环从1数到5：

In [ ]:
```
x = 1
while x <= 5:
    print(x)
```

In [ ]:
但如果像下面这样不小心遗漏了代码行`x += 1` ，这个循环将没完
没了地运行：

In [ ]:
```
# 这个循环将没完没了地运行!
x = 1
while x <= 5:
    print(x)
```

在这里，x的初始值为1 ，但根本不会变。 因此条件测试x <= 5 始终为True，导致while循环没完没了地打印1。

每个程序员都会偶尔因不小心而编写出无限循环，在循环的退出条件比较微妙时尤其如此。 如果程序陷入无限循环，可按Ctrl + C，也可关闭显示程序输出的终端窗口。

要避免编写无限循环，务必对每个while 循环进行测试，确保其按预期那样结束。 如果你希望程序在用户输入特定值时结束，可运行程序并输入这样的值。 如果在这种情况下程序没有结束，请检查程序处理这个值的方式，确认程序至少有一个这样的地方能让循环条件为False，或者让break语句得以执行。 注意Sublime Text等一些编辑器内嵌了输出窗口，这可能导致难以结束无限循环，不得不通过关闭编辑器来结束。 在这种情况下，可在输出窗口中单击鼠标，再按Ctrl + C，这样应该能够结束无限循环。

7.3 使用while 循环处理列表和字典

到目前为止，我们每次都只处理了一项用户信息： 获取用户的输入，再将输入打印出来或做出应答； 循环再次运行时，获悉另一个输入值并做出响应。 然而，要记录大量的用户和信息，需要在while循环中使用列表和字典。 for循环是一种遍历列表的有效方式，但不应在for循环中修改列表，否则将导致Python难以跟踪其中的元素。 要在遍历列表的同时对其进行修改，可使用while循环。 通过将while 循环同列表和字典结合起来使用，可收集、存储并组织大量输入，供以后查看和显示。

7.3.1 在列表之间移动元素

假设有一个列表包含新注册但还未验证的网站用户。 验证这些用户后，如何将他们移到另一个已验证用户列表中呢？ 一种办法是使用一个while 循环，在验证用户的同时将其从未验证用户列表中提取出来，再将其加入另一个已验证用户列表中。 代码可能类似于下面这样：

In [1]:
```
# 首先，创建一个待验证用户列表
# 和一个用于存储已验证用户的空列表。
unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []
# 验证每个用户，直到没有未验证用户为止。
# 将每个经过验证的用户都移到已验证用户列表中。
while unconfirmed_users:
    current_user = unconfirmed_users.pop()
    print(f"Verifying user: {current_user.title()}")
    confirmed_users.append(current_user)

    # 显示所有已验证的用户。
print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
    print(confirmed_user.title())
```

```
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice
```

7.3.2 删除为特定值的所有列表元素

在第3章中，我们使用函数remove()来删除列表中的特定值。 这之所以可行，是因为要删除的值只在列表中出现一次。 如果要删除列表中所有为特定值的元素，该怎么办呢？ 假设你有一个宠物列表，其中包含多个值

为'cat'的元素。 要删除 所有这些元素，可不断运行一个while 循环，直到列表中不再包含值'cat'，如下所示：

In [2]:
```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'cat','cat','cat','rabbit', 'cat']
print(pets)
while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'cat', 'cat', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

去掉所有重复值

In [3]:
```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'cat','cat','cat','rabbit', 'cat', 'gold
pets_set = set(pets)
#pets_set.remove('cat')
print(pets_set)
```

```
{'goldfish', 'dog', 'rabbit', 'cat'}
```

7.4 小结

在本章中，学习了：

- 如何在程序中使用input()来让用户提供信息；
- 如何处理文本和数的输入，以及如何使用while循环让程序按用户的要求不断运行；

多种控制while循环流程的方式： 设置活动标志、使用break语句以及使用continue语句； 如何使用while循环在列表之间移动元素，以及如何从列表中删除所有包含特定值的元素； 如何结合使用while循环和字典。

在第8章中，将学习函数。 函数让你能够将程序分成多个很小的部分，每部分都负责完成一项具体任务。 你可以根据需要调用同一个函数任意次，还可将函数存储在独立的文件中。 使用函数可让你编写的代码效率更高、更容易维护和排除故障，还可在众多不同的 程序中重用。

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

In [ ]:
```
git
```