User Inputs and Loops

BY: DR. SABEEHUDDIN HASAN

How the input() Function Works

- Most programs are written to solve an end user's problem
- To do so, you usually need to get some information from the user
- When your program needs a name, you'll be able to prompt the user for a name
- You need the input() function for this purpose

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

- The input() function takes one argument: the *prompt* that we want to display to the user, so they know what kind of information to enter

```
Tell me something, and I will repeat it back to you: Hello everyone!
Hello everyone!
```

How the input() Function Works

- ***Writing Clear Prompts***
  - Each time you use the input() function, you should include a clear, easy-to-follow prompt that tells the user exactly what kind of information you're looking for

    name = input("Please enter your name: ")

    print(f"\nHello, {name}!")

    Please enter your name: ***Eric***

    Hello, Eric!

How the input() Function Works

- ***Writing Clear Prompts***
  - Sometimes you'll want to write a prompt that's longer than one line

```
prompt = "If you share your name, we can personalize the messages you see."
prompt += "\nWhat is your first name? "
name = input(prompt)
print(f"\nHello, {name}!")
```

  - This example shows one way to build a multiline string

```
If you share your name, we can personalize the messages you see.
What is your first name? Eric
Hello, Eric!
```

How the input() Function Works

- ***Using int() to Accept Numerical Input***
  - When you use the input() function, Python interprets everything the user enters as a string

    >>> ***age = input("How old are you? ")***

    How old are you? ***21***

    >>> ***age***

    '21'
  - The user enters the number 21, but when we ask Python for the value of age, it returns '21'
  - Consider the following

    >>> ***age = input("How old are you? ")***

    How old are you? ***21***

    >>> ***age >= 18***

    Traceback (most recent call last):

    File "<stdin>", line 1, in <module>

    TypeError: '>=' not supported between instances of 'str' and 'int'

How the input() Function Works

- ***Using int() to Accept Numerical Input***
  - We can resolve this issue by using the int() function, which converts the input string to a numerical value

    >>> ***age = input("How old are you? ")***

    How old are you? ***21***

    >>> ***age = int(age)***

    >>> ***age >= 18***

    True

How the input() Function Works

- ***The Modulo Operator %***
  - It divides one number by another number and returns the remainder:

    >>> ***5 % 3***

    2

    >>> ***6 % 3***

    0

  - Consider the following program

```
number = input("Enter a number, and I'll tell you if it's even or odd: ")
number = int(number)
if number % 2 == 0:
        print(f"\nThe number {number} is even.")
else:
        print(f"\nThe number {number} is odd.")
```

Introducing while Loops

- The for loop takes a collection of items and executes a block of code once for each item in the collection
- In contrast, the while loop runs as long as, or *while*, a certain condition is true
- ***The while Loop in Action***
  - You can use a while loop to count up through a series of numbers
  - For example, the following while loop counts from 1 to 5

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

  - In the first line, we start counting from 1 by assigning current_number the value 1
  - The *while loop* is then set to keep running as long as the value of current_number is less than or equal to 5

Introducing while Loops

- ***Letting the User Choose When to Quit***
  - We'll define a *quit value* and then keep the program running as long as the user has not entered the quit value

```
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)
        print(message)
```

  - We first define a prompt that tells the user their two options: entering a message or entering the quit value (in this case, '*quit*').

Introducing while Loops

- ***Using a Flag***
  - In a game, several different events can end the game
  - When the player runs out of ships, their time runs out, or the cities they were supposed to protect are all destroyed, the game should end
  - For a program that should run only as long as many conditions are true, you can define one variable that determines whether or not the entire program is active
  - This variable, called a *flag*, acts as a signal to the program
  - We can write our programs so they run while the flag is set to True and stop running when any of several events sets the value of the flag to False

Introducing while Loops

- ***Using a Flag***
  - The following program uses a flag, which we'll call active
  - It will monitor whether or not the program should continue running:

```
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)
```

Output:
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. ***Hello everyone!***
Hello everyone!
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. ***Hello again.***
Hello again.
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. ***quit***

Introducing while Loops

- ***Using break to Exit a Loop***
  - To exit a while loop immediately without running any remaining code in the loop use the break statement.
  - The break statement directs the flow of your program
  - You can use it so the program only executes code that you want it to and when you want it to

```
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()}!")
```

Output:

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) ***New York***
I'd love to go to New York!
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) ***San Francisco***
I'd love to go to San Francisco!
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) ***quit***

Introducing while Loops

- ***Using continue in a Loop***
  - Rather than breaking out of a loop entirely without executing the rest of its code, you can use the continue statement to return to the beginning of the loop, based on the result of a conditional test

```
current_number = 0
while current_number < 10:
        current_number += 1
        if current_number % 2 == 0:
                continue
        print(current_number)
```

Output:
1
3
5
7
9

Introducing while Loops

- ***Avoiding Infinite Loops***
  - Every while loop needs a way to stop running so it won't continue to run forever.
  - For example, this counting loop should count from 1 to 5:

```
x = 1
while x <= 5:
        print(x)
        x += 1
```

Output:
1
2
3
4
5

  - However, if you accidentally omit the line x += 1, the loop will run forever
  - Now the value of x will start at 1 but never change
  - As a result, the conditional test x <= 5 will always evaluate to True and the while loop will run forever
  - If your program gets stuck in an infinite loop, press CTRL-C or just close the terminal window displaying your program's output

Looping through Lists

- **Using a for loop**
  - Print all items by referring to their index number:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
    print(thislist[i])
```

- **Using a while loop**
  - Print all items, using a while loop to go through all the index numbers

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
    print(thislist[i])
    i = i + 1
```

Looping through Lists

- **List Comprehension**
  - List Comprehension offers the shortest syntax for looping through lists:
  - A short hand for loop that will print all items in a list:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Using a while Loop with Lists and Dictionaries

- A for loop is effective for looping through a list, but you shouldn't modify a list inside a for loop because Python will have trouble keeping track of the items in the list
- To modify a list as you work through it, use a while loop
- Using while loops with lists and dictionaries allows you to collect, store, and organize lots of input to examine and report on later
- ***Moving Items from One List to Another***
  - Consider a list of newly registered but unverified users of a website
  - After we verify these users, how can we move them to a separate list of confirmed users?
  - One way would be to use a while loop to pull users from the list of unconfirmed users as we verify them and then add them to a separate list of confirmed users

Using a while Loop with Lists and Dictionaries

- *Moving Items from One List to Another*

```
unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []
# Verify each user until there are no more unconfirmed users.
# Move each verified user into the list of confirmed users.
while unconfirmed_users:
        current_user = unconfirmed_users.pop()
        print(f"Verifying user: {current_user.title()}")
        confirmed_users.append(current_user)
# Display all confirmed users.
print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
         print(confirmed_user.title())
```

Output:
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice

Using a while Loop with Lists and Dictionaries

- ***Removing All Instances of Specific Values from a List***
  - The remove() function worked because the value we were interested in appeared only once in the list
  - But what if you want to remove all instances of a value from a list?
  - Say you have a list of pets with the value 'cat' repeated several times.
  - To remove all instances of that value, you can run a while loop until 'cat' is no longer in the list

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

Output:
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']

Using a while Loop with Lists and Dictionaries

- ***Filling a Dictionary with User Input***
  - The following program passes through the loop and prompts for the participant's name and response
  - We'll store the data we gather in a dictionary, because we want to connect each response with a particular user
  - Following code stores the mountains against the respondent's name in a dictionary
  - Later prints the response against the name

```
responses = {}

# Set a flag to indicate that polling is active.

polling_active = True
```

Using a while Loop with Lists and Dictionaries

```
while polling_active:
        # Prompt for the person's name and response.
        name = input("\nWhat is your name? ")
        response = input("Which mountain would you like to climb someday? ")
        # Store the response in the dictionary.
        responses[name] = response
        # Find out if anyone else is going to take the poll.
        repeat = input("Would you like to let another person respond? (yes/ no) ")
        if repeat == 'no':
                polling_active = False
# Polling is complete. Show the results.
print("\n--- Poll Results ---")
for name, response in responses.items():
         print(f"{name} would like to climb {response}.")
```