User Input and While Loops

University of Wyoming COSC 1010

Adapted from: *Python Crash Course 3rd Ed* By Eric Matthes

User Input and While Loops

- Most programs are written to solve and end user's problems
- To do so you typically need to get some information from the user
- For example, maybe someone wants to see if they are old enough to vote
- To write a program to solve this you need to know the user's age

User Input and While Loops

- The program would need to ask the user to enter, or input their age
- Once the program has the information they can make a decision regarding if the user is old enough to vote
- We will talk about accepting input from a user
- As well as keeping programs running indefinitely

How the `input()` function works

- The `input()` function pauses your program and waits for the user to enter some text
- When the input is received it can be assigned to a variable
- This makes it very convenient to work with
- The `input()` function takes one argument, the *prompt* that displays to the user

In [ ]:
```
echo = input("Tell me something and I will say it back")
print(echo)
```

```
HIClass
```

How the `input()` function works

- In this example python runs the first line

    - Showing the suer the prompt
- Then the program waits until the user inputs something by typing out thew message and hitting enter
- The response is then assigned to the variable `message` and displayed back

Writing Clear Prompts

- Each time `input()` is used, it should include a clear message telling the user exactly what it wants
- Add a space at the end of the prompts to separate the prompt from the user input
    - `input("Please enter your name: )`
- Sometimes you will need a long prompt, one longer than a line
- To do so you can build your message in a variable and pass that variable to `input()`

In [ ]:
```
message = "Good Morning Class,\n"
message += "I hope you are doing well today.\n"
message += "And enjoying COSC1010\n"
message += "Are you enjoying class? yes or no"

answer = input(message)

if answer.lower() == "yes":
    print("Oh good!")
else:
    print("womp womp")
```

```
Oh good!
```

Using `int()` to Accept Numerical Input

- When using the `input()` function Python interprets everything as a string
- As it is text entered through the input
- But, what if you are trying to judge a user's age?
- You cannot compare string adn integer values

In [ ]:
```
age = input("how old are you?")
print(type(age))
print(age)
```

```
<class 'str'>
21
```

In [ ]:
```
age = input("how old are you?")
if age >= 18:
    print("You can vote")
else:
    print("You can't vote")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_12737/2494762085.py in <module>
      1 age = input("how old are you?")
----> 2 if age >= 18:
      3     print("You can vote")
      4 else:
      5     print("You can't vote")

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

Using `int()` to Accept Numerical Input

- When the entered age is compared to the integer age an error is returned
  - `TypeError: '>=' not supported between instances of 'str' and 'int'`
- The string representation of the numerical value is entered
- Again, Python doesn't know how to compare integers and strings

Using `int()` to Accept Numerical Input

- This issue can be resolved by using the `int()` function
- This converts the input string to a numerical
- Once the conversion occurs the comparison can happen

In [ ]:
```
age = input("how old are you?")
age = int(age)
print(type(age))
if age >= 18:
    print("You can vote")
else:
    print("You can't vote")
```

```
<class 'int'>
You can vote
```

Using `int()` to Accept Numerical Input

- Now when you enter a number it is converted to an integer value

- So, the comparison is able to occur
- Now, when you use numerical input to do calculations and comparisons you can convert the string to a number

The Modulo Operator

- A useful tool for working with numerical information is teh *modulo operator* %
- this divides one number by another and returns the remainder
- It doesn't tell you how many times one number fits in another, just what the remainder is
- When one number is divisible by another the remainder is 0

In [ ]:
```
print(4%3)
print(5%3)
print(33%3)
print(7%3)
```

```
1
2
0
1
```

Introducing `while` Loops

- The `for` loop takes a collection of items
- It executes a block of code once for each item in the collection
- The `while` loop runs as long as a condition is met
- Or, *while* a condition is met

The `while` Loop in Action

- For a basic example `while` loops can be used to count through a series of numbers

In [ ]:
```
num = 0
while num < 5:
    print(num)
    num += 1
```

```
0
1
2
3
4
```

The `while` Loop in Action

- The first line we assign 0 to the `num` variable
- The `while` loop is then set to run while `num` is `< 5`
- The code inside prints the value of `num` and increments it by 1
- Python repeats the loop until `num >= 5`

The `while` Loop in Action

- Often programs contain `while` loops
- While loops can be used to keep a game running as long as it is played
- While loops can be used to keep programs running until told to stop

Letting the User Choose When to Quit

- You can make a program run as long as needed by embedding code in a `while`
- Then you can define an *exit* value
- This value will be used to exit a while loop
- And as a result stop the execution of a program

In [ ]:
```
prompt = "Give me something to echo, 'exit' to quit"

message = ""
while message != "exit":
    message = input(prompt)
    if message != "exit":
        print(message)
```

```
test
hello
```

Letting the User Choose When to Quit

- The prompt is declared, telling the user how to exit
- The `message` variable is set to be the empty string
- The `while` loop is set to run *while* `message != "exit"`
  - Inside the loop users are prompted for input
  - There is an `if` before the message is printed, don't want to print `exit`

Using a Flag

- In the previous loop the program performed tasks while a condition was true
- But as programs become more complicated, many event could cause a program to start
- For a program that should run only as long as many conditions are `True` , a variable can be used
- These kinds are variables are known as *flags*, acting as a signal to the program

Using a Flag

- Programs can be written so they set the flag to `True`
- This can happen when any number of events set the flag to `False`
- Then all other tests can be neatly placed in the rest of the program

In [ ]:
```
prompt = "Give me something to echo, 'exit' to quit"

active = True
while active:
    message = input(prompt)
    if message != "exit":
        print(message)
    else:
        active = False
```

```
hi
class
```

Using a Flag

- Rather than checking message to control the loop, the `active` variable is set as `True`
- Doing so makes the `while` statement simpler, as it doesn't include a comparison
- In the `if` statement inside an `else` has been added to set the flag to `False`
- The program functions in the same fashion as previously

Using `break` to Exit a Loop

- To exit a while loop immediately without running any remaining code a `break` can be used
- This works regardless of the result of any condition test
- Tt directs the flow of a program, and can be used to control which lines of code are executed

- And which aren't

In [ ]:
```
prompt = "Give me something to echo, 'exit' to quit"


while True:
    message = input(prompt)
    if message == "exit":
        break
    else:
        print(message)
```

```
hi
there
```

Using `break` to Exit a Loop

- The loop starts with `while True:` while would typically cause the loop to run forever
- The loop then continues asking for user input until 'quit' is entered
- At which point the `break` will trigger in the `if`, exiting the loop

Using `continue` in a Loop

- Rather than breaking out of a loop without executing the code, `continue` can return it to the top of the loop
- This is done based on a conditional test within the loop

In [ ]:
```
num = 0
while num < 10:
    num += 1
    if num % 2 == 0:
        continue
    print(num)
```

```
1
3
5
7
9
```

Using `continue` in a Loop

- That loop starts counting at 0
- It increments the number right away
- If the number is even the loop jumps back to the increment, thanks to the continue

    - This skips the print statement
- If the number is odd, the continue is not executed so the print happens

Avoiding Infinite Loops

- Every `while` loop needs a way tos top running, so it doesn't run forever
- For example consider a loop to count 1-5

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

Avoiding Infinite Loops

- If the `x += 1` were omitted the loop would run indefinitely, print 1 each time

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

Avoiding Infinite Loops

- Every programmer writes an infinite `while` loop at some point
    - Especially if the loop has subtle exit conditions
- If your program gets stuck in an infinite loop `ctrl-c` typically stops it
    - or close the terminal window

Avoiding Infinite Loops

- To avoid infinite loops test every `while` loop
- If you want the program to end when a user enters a specific value test for that
    - Run the program and test the input
- If the program doesn't end as expected go through your code

Using a `while` loop with Lists and Dictionaries

- Thus far only one piece of user has been worked with at a time

- The user was able to input something and then a response was printed
- The program only held one message at a time, not maintaining a record of which user submitted it
- To keep track of many users and pieces of information lists adn dictionaries are needed

Using a `while` loop with Lists and Dictionaries

- `for` loops work well for looping through a list
- But no modification should be made while looping though the list as Python will have issues tracking elements
- To modify a list as you work through it a `while` loop should instead be used
- Using `while` loops with lists and dictionaries allows you to collect, store and organize many inputs

Moving Items from One List to another

- Suppose you need to move elements from on list to another
  - Perhaps you have a website that users can sign up for
  - And you have lists of verified and unverified users
  - Unverified users would need to be moved to the verified list upon verification

In [ ]:
```
unverified_users = ["cowboy joe","pistol pete","steamboat"]
verified_users =[]

while unverified_users:
    verified_users.append(unverified_users.pop())
print(verified_users)
```

```
['steamboat', 'pistol pete', 'cowboy joe']
```

Moving Items from One List to another

- The `unverified_users` list began empty
- The `verified_users` list had three elements initially
- At the end of execution `unverified_users` is empty
- `verified_users` now has the three items

Removing All Instances of Specific Value in a List

- Recall we talked about the `.remove()`

- It only removed the first occurrence of an element
- We can now use `while` loops to remove all occurrences without having to know their index position

In [ ]:
```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
print(pets)

while "cat" in pets:
    pets.remove("cat")
print(pets)
```

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

Removing All Instances of Specific Value in a List

- Using the while loop allows one to remove all occurrences from a list
- It will go through however many times is needed, removing the first occurrence each time
- Until none are left

Filling a Dictionary with User Input

- Within a while loop there is no limit to how much prompting is done
- Once the prompting has been done you can store it in whatever fashion you wish

In [ ]:
```
responses = {}
polling = True

while polling:
    name = input("What is your name? ")
    response = input('What is your favorite color')
    responses[name] = response

    ask_again = input("Should another be asked yes/no")
    if ask_again == "no":
        polling = False
print("Printing responses")
for name in responses:
    print(f"{name}'s favorite color is {responses[name]}")
```

```
Printing responses
pistol pete's favorite color is gold
cowboy joe's favorite color is brown
```

Filling a Dictionary with User Input

- The program defines an empty list
- It then goes through a loop prompting users several time
- As many users can respond to the prompt as needed