Loops and Conditionals

can code communities

Loops Review

- What is the difference between a For and While Loop?
- How do I write a For Loop?
- How do I write a While Loop?

Break

Break is a keyword that when placed in a loop will leave the loop without executing any more code when the keyword is reached.

You can **break** if a condition is met in a while loop, and it will end the loop instantly, instead of waiting until the beginning of the loop again to check the condition.

```
userin = ""
while userin != "stop":
    userin = input("Enter something, or \"stop\" to stop")
    print(userin)
```

```
userin = ""
while True:
    userin = input("Enter something, or \"stop\" to stop")
    if userin == "stop":
        break
    print(userin)
```

What is the difference between these two while loops?

Break in Nested Loops

If you use break and you have nested loops, it will only break from the inner-most loop, and then continue executing the outer loop.

This means you could enter the inner loop again on the next iteration of the outer loop.

```
userin = ""
# Take strings from the user until "stop" is entered
while userin != "stop":
    userin = input("Enter a word, \"stop\" to stop: ")
    # Print all the letters in the word up to the first
    #  character that isn't a letter
    for l in userin:
        if l.isalpha():
            print(l)
        else:
            break
    print() # Print a blank line
```

After breaking, this is the next line that is run. We exited the for loop, but we're still in the outer while loop.

Exercise

Write some code that takes in numbers from a user one at a time. This should keep going until the user enters 0. Then print the sum of all the numbers.
If the user inputs something that isn't a number, an error message should appear and the program should stop. (Hint: use **break**)

Example (no error):

```
5
12
0
Sum: 17
```

Example (error):

```
5
7
c
Error: Not a number
```

Continue

Continue is a keyword that when placed in a loop skips over the remaining code and brings back to the top when the keyword is reached.

Instead of exiting the loop entirely, it just skips **this iteration**, and goes back to the beginning of the loop for the next iteration.

You can use **continue** to skip certain iterations based on a condition.

Example

Use the **continue** keyword to loop through a string and only print the vowels.

Exercise: Sum of Even Digits

Take a string as user input, and verify that it's a number.

Loop through each digit of the number, and only add it to the sum if it's even.

Print the sum of all the even digits at the end.

Make sure to use the **continue** keyword.

Break vs. Continue

```
word = "hello"
vowels = ['a','e','i','o','u']
for letter in word:
    if letter in vowels:
        break
    print(letter)
```

```
word = "hello"
vowels = ['a','e','i','o','u']
for letter in word:
    if letter in vowels:
        continue
    print(letter)
```

What is the output of each of these code snippets? How does it differ?

Discussion: When to use break, and when to use continue?

Pass

Pass is a keyword that essentially does nothing when run, however is useful when fleshing out parts of code. You can use it as a placeholder.

```
word = "hello"
vowels = ['a','e','i','o','u']
for letter in word:
    if letter in vowels:

    print(letter)
```

Without pass, notice that there is an error.

```
word = "hello"
vowels = ['a','e','i','o','u']
for letter in word:
    if letter in vowels:
        pass
    print(letter)
```

With pass, notice there is no error. This allows us to have empty if statements and loops, basically anything that requires a line of code to be valid or an indent following it.

Exercise

Write some code that takes in strings from a user one at a time.

After each string is taken in evaluate if the string is empty, a number, a set of letters, or contains symbols.

- If the string is empty, stop the loop.
- If the string is a number, convert it to a float and add it to a total.
- If the string is a set of letters, concatenate to the other letter strings passed in.
- If it contains a symbol, or is none of the above, do nothing and repeat the loop.

Make sure to use break and/or continue.

While-Else and For-Else

You can add an else: block to a for loop or while loop. It executes once the condition of the loop is no longer true.

If you break from the loop, the else block doesn't execute.

```
i = 3
while i > 0:
    print(i)
    i -= 1
    if i == 4:
        break
else:
    print("done")
```

Discussion:

What is the output of this code?

What would the output be if i is initialized to 6?