ITP 115 – Programming in Python

Strings as Sequences

While Loops

Outline

- `while` loops – THIS WEEK
- `for` loops – NEXT WEEK

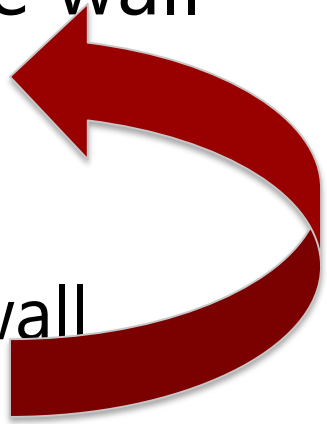
- `range()` function
- strings as sequences
- sequence functions and operators

Consider…

- Why might we want code to repeat?

Loop Motivation #1: We want to do things multiple times

- 99 bottles of cold brew coffee on the wall
  - Take one down
  - Pass it around
  - 98 bottles of cold brew coffee on the wall



Do again until you have 0 bottles on the wall

Pseudocode

```
bottles = 99

while
```

Pseudocode

```
bottles = 99

while bottles is greater than 0
        sing bottles on wall
        take one down (bottles - 1)
        pass it around
        repeat again
```

Loop Motivation #2: Users make mistakes

- It is easy for users to make mistakes when they have to type "answers"
- Ideally, we should ask them to try again

```
Do you want cream (y/n)? Q
Oops! Invalid choice

<program ends>
```

vs

```
Do you want cream (y/n)? Q
Oops! Invalid choice

Do you want cream (y/n)? Y
```

How would you describe the fix?

`while`

Pseudocode

```
while
```

user has not entered `"y"` AND user has not entered `"n"`

Ask the user for a value of `"y"` or `"n"`

`while` Loops

- Loops let us repeat something
- **`while`** loop is similar to the **`if`** structure
- As long as the **condition** is **`true`**, the block (*loop body*) is executed
- When the condition is **`false`**, the loop exits and the program continues

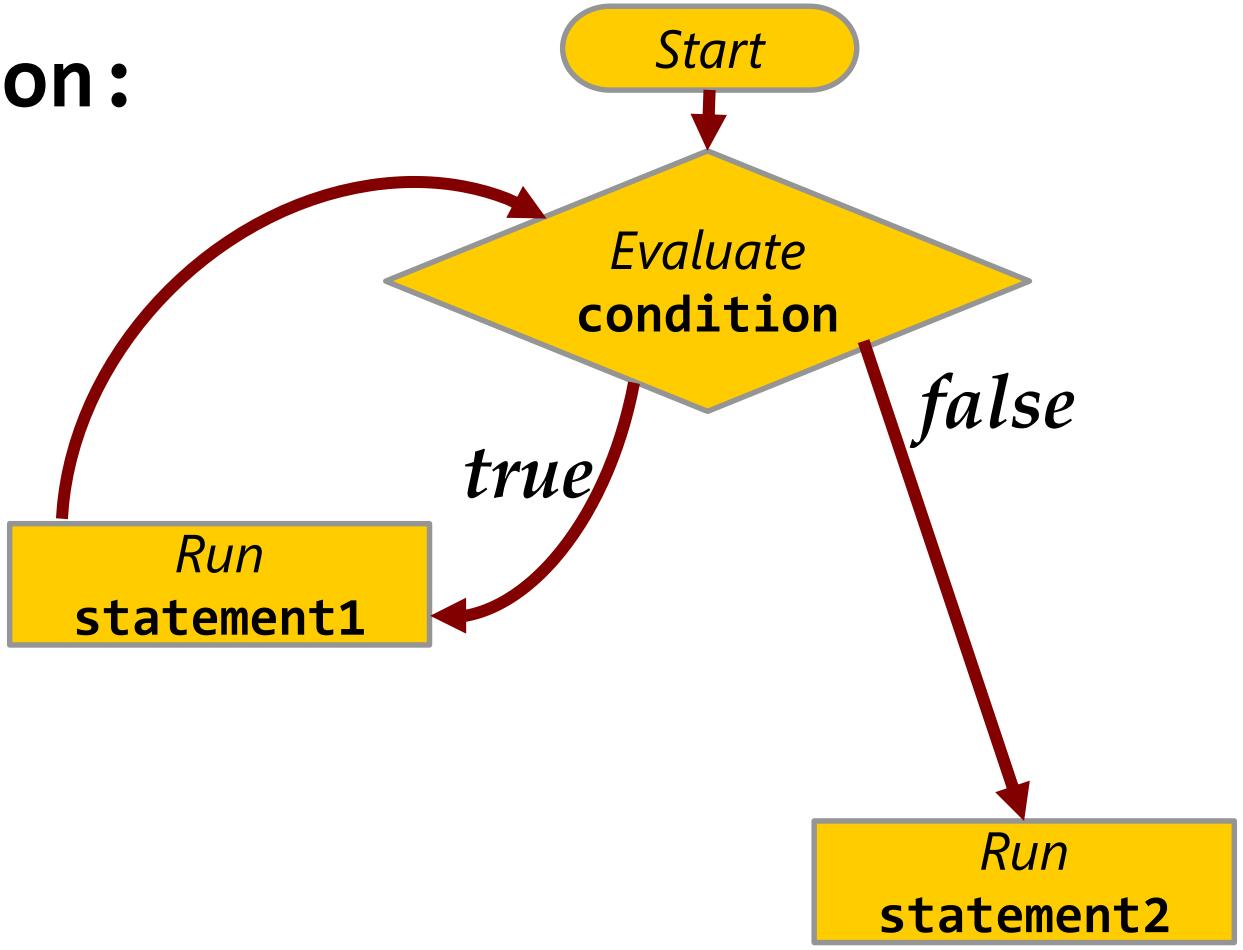
The `while` Statement

- Syntax

```
while condition:
        statement1
        statement2
```

Semantics of `while`

```
while condition:
    statement1

statement2
```



Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Check this condition at the start of each loop iteration

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Do this if the condition is **`true`** at the beginning of the block

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

When you reach the end of the **while** block, go to the top again

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

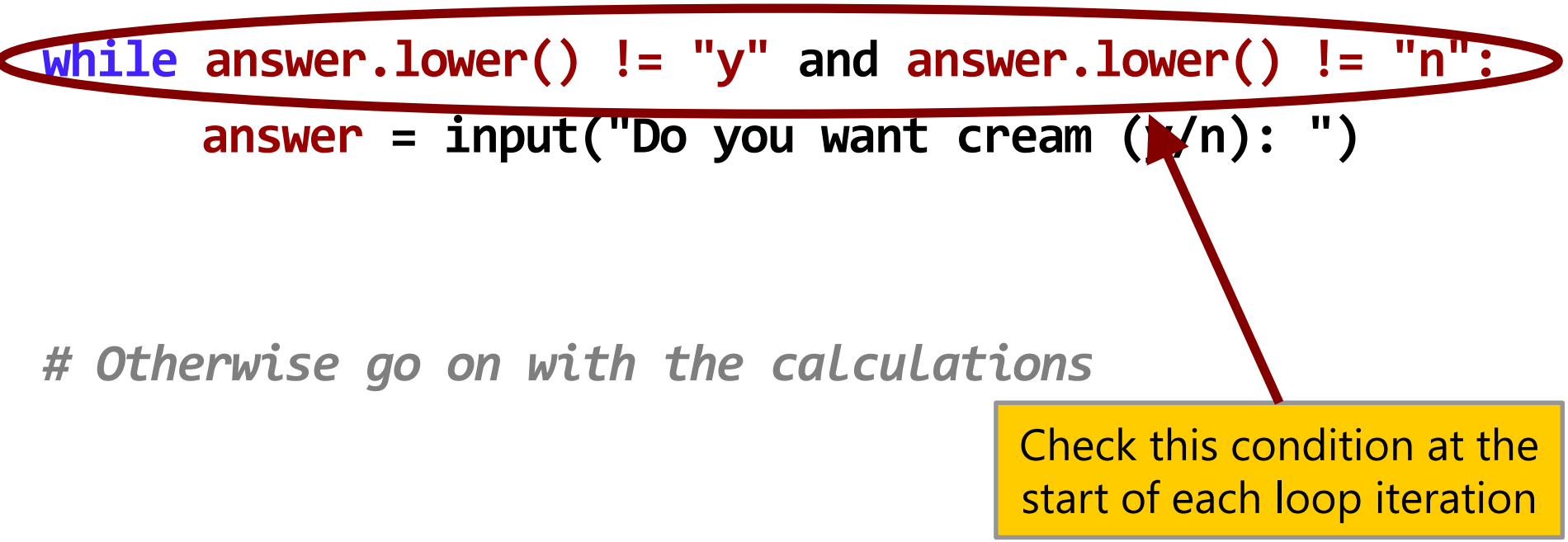
Check this condition at the again

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
      answer = input("Do you want cream (y/n): ")


# Otherwise go on with the calculations
```

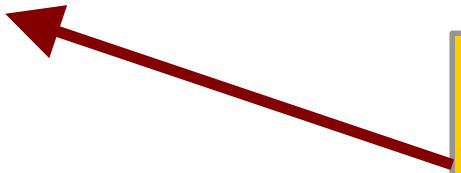
Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
    answer = input("Do you want cream (y/n): ")

# Otherwise go on with the calculations
```



Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
      answer = input("Do you want cream (y/n): ")



# Otherwise go on with the calculations
```



Do this if the condition is **true** at the beginning of the block

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
      answer = input("Do you want cream (y/n): ")

  ----------------------------------

# Otherwise go on with the calculations
```

When you reach the end of the **while** block, go to the top again

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
    answer = input("Do you want cream (y/n): ")

# Otherwise go on with the calculations
```

Check this condition at the start of next iteration (loop)

Consider…

```
counter = 10
while counter > 0:
    print(counter)
    counter = counter - 1
print("Blast OFF!")
```

Consider…

```
counter = 10
while counter > 0:
    print(counter)
    counter = counter + 1
print("Never!")
```

Infinite Loops

- A loop that always repeats without ending is an **infinite loop**
- This happens when the boolean condition never becomes false
- There should always be a way to exit a loop

Infinite Loops Examples

```
counter = 10
while counter > 0:
      print(counter)
      counter = counter + 1
```

```
word = input("Enter a word: ")
while word != "exit":
      print(word)
```

```
while True:
        print("Wheee!")
```

Tips for Designing a Loop Body

- Write out the algorithm in pseudocode
- Look for a repeated pattern—this is **loop body**
  - May not be first action
- Look for variables to initialize **BEFORE** loop
- Look for things to do **AFTER** the loop ends

Tips: Initializing Statements

- Some values are set **before** the loop begins
  - Often these variables get a value of **zero** or **one**
  - Other times it's based on 1 iteration through the loop

- Other variables get values only when the loop is executing

Tips: Updating Variables

- A while loop is controlled by a Boolean condition (often related to a variable)
  - "As long as" that condition is True, the loop continues

- Inside the loop body, you must do something that gets you closer to ending the loop
  - Otherwise we have an infinite loop

- *End lecture*

Common Types of Loops

- Count-controlled
  - Define a counter (number) outside your loop
- Sentinel value
  - Loop stops due to a particular value
  - Similar to an exit command
- "Do while"
  - Define a Boolean, string, or number outside your loop to kickstart the first run
  - This type of while loop will always run at least once

Common Loops: Count-Controlled

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of coffee on the wall")
    numBottles = numBottles - 1
```

Initialization

Update

Common Loops: Sentinel Value

- Sentinel value can be used mark the end of a list
  - Should be different from all other input

- Common sentinel values

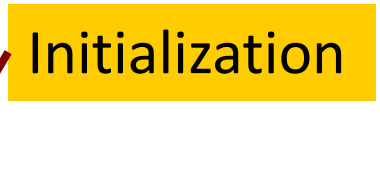
  `-1` after a long list of positive numbers

  `1 5 100 0 -1`

  `exit` after other commands

  `show get delete exit`

Common Loops: Sentinel Value




```
nextNum = int(input("Enter a num or -1 to quit"))
while nextNum != -1:
        print("num entered =", nextNum)
        nextNum = int(input("Enter a num or -1 to quit"))

print("You entered -1.")
```

Common Loops: Do While

- Many programming languages have a special loop called a **do while** loop

- Python doesn't have a **do while** loop, but we can mimic its behavior

- **while** and **do while** are equivalent so sometimes it is easier to write a loop with one or the other

Common Loops: Do While

- Define a variable
  - Commonly a string, integer, or Boolean expression
  - Create this variable outside the loop
- Write the loop condition, already satisfied by your variable
  - This ensures the loop will always run the first time
- Within the loop body there must be some action which affects that variable

Do While: String

- Using a string

```
name = “unknown"
while name != "Pikachu":
        print("I’m looking for Pikachu!")
        name = input("Which Pokémon are you?")
print("I found Pikachu!")
```

```
I’m looking for Pikachu!
Which Pokémon are you? Charmander
I’m looking for Pikachu!
Which Pokémon are you? Pikachu
I found Pikachu!
```

Do While: String

- Using an integer (similar to Sentinel)

```
keepGoing = 1
while keepGoing != 0:
        print("The loop keeps running!")
        keepGoing = int(input("Enter 0 to quit: "))
```

```
The loop keeps running!
Enter 0 to quit: 5
The loop keeps running!
Enter 0 to quit: 0
```

Do While: Boolean Value

Initialization

```
areMore = True
print("Enter positive numbers or -1 to quit")
while areMore == True:
    nextNum = int(input("Enter a number: "))
    if nextNum < 0:
        areMore = False
print("You exited the loop")
```

Update

```
Enter numbers or -1 to quit
1 2 3 -1
You exited the loop
```

Do While: Boolean Value

Initialization

```
sum = 0
areMore = True
print("Enter positive numbers or -1 to quit")
while areMore == True:
    nextNum = int(input("Enter a number: "))
    if nextNum < 0:
        areMore = False
    else:
        sum = sum + nextNum
print("The sum is " + sum)
```

Update

```
Enter numbers or -1 to quit
1 2 3 -1
The sum is 6
```

FOR REFERENCE ONLY

`break` and `continue` Statements

- In this class, do **not** use break and continue
  - Though sometimes useful, they can lead to poor understanding of loops

- The **`break`** statement means "*break out of the loop*"

- The **`continue`** statement means "*jump back to the top of the loop*"

Example

```
count = 0
while True:
    count += 1

    # end loop if count >= 10
    if count > 10:
        break

    # skip 5
    if count == 5:
        continue

    print(count)
```

```
1
2
3
4
6
7
8
9
10
```