*

CSCI-UA-0002
Intro to Computer Programming (No Prior Experience)

Module 4: Condition-Controlled (While) Loops

Professor Emily Zhao

Section 008
T/R 12:30-1:45PM

Section 012
T/R 4:55-6:10PM

*

Agenda

— Review Ed Questions
— Module 4 Review
— Practice Problems

Module 4

— Repetition Structures + `while` loops
— Self-referential assignment statements
— Accumulator variables
— "While" loop control
  — Infinite loops
  — `break` and `continue`
  — Sentinels
— Data Validation
— Color in Turtle Graphics

Your Questions

→ How do you best count how many times a loop runs?

→ What does it mean to "iterate"?

→ Can you review `+=` and `-=`?

→ What is an accumulator variable?

→ Can you go over `continue`?

"While" loops

A **condition-controlled loop** is a programming structure that causes a statement or set of statements to repeat as long as a condition evaluates to true
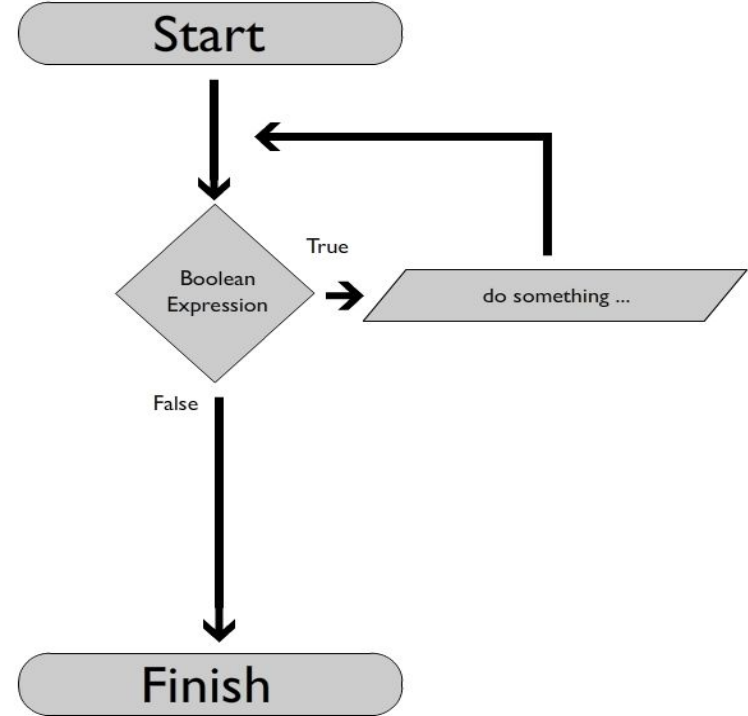
standard Boolean condition
that evaluates to True
or False

```
while condition:
    statement
    statement
    statement
    statement
```

the statements that
will be repeated

indentation indicates that
the statements under the while
loop should be repeated

Reasons to use a while loop

Repetition
Repeat a task an uncertain amount of times until a specific condition is met

Data Validation
Repeatedly prompt the user for input until they provide valid data

User Interaction
Require user input or interaction until the user chooses to exit

Reasons to use a while loop

Repetition
You want to keep rolling a dice until you get a 6

Data Validation
You want to ensure the user enters a positive number, and keep prompting them until they do

User Interaction
You are creating a simple menu-driven program that continues to display options to the user until they choose to exit

Repetition with Uncertain Iteration Count

Scenario: You want to keep rolling a dice until you get a 6.

```python
import random

roll = 0 # initialize variable

while roll != 6:
    roll = random.randint(1, 6) # repeat this while
    print(roll)                 # roll is not equal to 6
```

Data Validation

Scenario: You want to ensure the user enters a positive number, and keep prompting them until they do

```python
# ask for positive integer
num = int(input("Enter a positive number: ")) # initialize num as user input

while num <= 0:
    print("Invalid input. Please enter a positive number.")
    num = int(input("Enter a positive number: ")) # reask until num is positive

print("Thank you for entering a positive integer.")
```

User Interaction

Scenario: You are creating a simple menu-driven program that continues to display options to the user until they choose to exit

```python
choice = ""
while choice != "4":
    print("Menu:")
    print("1. View profile")
    print("2. Edit settings")
    print("3. Help")
    print("4. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        # View profile logic
    elif choice == "2":
        # Edit settings logic
    elif choice == "3":
        # Help logic
    elif choice == "4":
        print("Goodbye!")
    else:
        print("Invalid choice. Please try again.")
```

Let's practice counting

Self-referential assignment statements
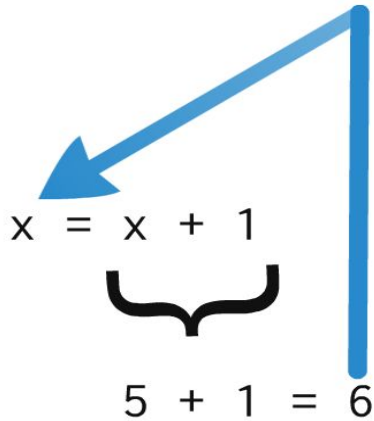

```
x = x + 1
```

```
x += 1
```

```
a = 0
b = 4

while a < b:
    a += 1   # Enters the loop: a = 0,
             # Run 1: a -> 1,
             # Run 2: a -> 2,
             # Run 3: a -> 3,
             # Run 4: a -> 4,
             # -> will not enter loop a 5th time
```

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Write code in `Python 3.6 [reliable stable version, select 3.11 for newest]` ▾

```
1 |
```

[Visualize Execution]   [**NEW:** Get AI Help]   If you use ChatGPT or other AI, [take this survey](#)

`hide exited frames [default]` ▾   `inline primitives, don't nest objects [default]` ▾   `draw pointers as arrows [default]` ▾

[Show code examples](#)

```python
# counter variable
count = 5

while count < 10:
    print(count)
    count -= 1
```

5, 4, 3, 2, 1 … Infinite loop
(control + c to end)

```python
# counter variable
count = 0

while count < 10:
    count += 1
    print(count)
```

> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

`break`

The break keyword is used to break out a loop.

`continue`

Used to end the current iteration in a loop
and continues to the next iteration.

```python
# counter variable
count = 0

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

> 1, 3, 5, 7, 9

```python
# counter variable
count = 0

while count < 10:
    count += 1
    if count % 2 == 0:
        break
    print(count)
```

> 1

Two ways to set up your while loops

while condition == True:

Keeps looping until the condition is no longer true (either arithmetic happens so that it is no longer true, or we set the condition to false)

while True:

Immediately enters and continues to loop indefinitely until the program encounters a `break`

```python
# set up control variable
keepAsking = True

while keepAsking == True: # enter the loop

    # will keep asking user to give them an integer
    num = int(input("Give me an integer: "))

    # until the condition is no longer true
    if num > 0:
        print("Thank you for inputting a valid integer")
        keepAsking = False
    else:
        print("Keep trying")
        # go back to the top of the loop and ask for input again
        # because keepAsking is still true
```

```python
while True: # enter loop immediately

    # will keep asking user to give them an integer
    num = int(input("Give me an integer: "))

    # until the program reaches a BREAK
    if num > 0:
        print("Thank you for inputting a valid integer")
        break
    else:
        print("Keep trying")
        # go back to the top of the loop and ask for input again
```

* Either way, make sure you create a condition to exit your loop, or face dealing with an INFINITE loop.

Programming Challenge: Divisibility Tester

Write a program that lets the user test to see if a series of numbers are evenly divisible by 3. If they are, print out a status message telling the user.

Extension: Start off by asking the user to enter in the number that should be used during the test (i.e. enter 5 if you want to test to see if a range of numbers is evenly divisible by 5)

```
>> Enter a number to check divisibility by 3: 9
>> Your number is divisible by 3.
>> Would you like to check another number's
divisibility? Type yes or no: yes
>> Enter a number to check divisibility by 3: 7
>> Your number is not divisible by 3.
>> Would you like to check another number's
divisibility? Type yes or no: no
>> Thanks for using our program.
```

Programming Challenge: Divisibility Tester

```python
# set up "control variable"
check_divis = "yes"

while check_divis == "yes":
    num = int(input("Enter a number to check divisibility by 3: "))

    if num % 3 == 0:
        print("Your number is divisible by 3.")
    else:
        print("Your number is not divisible by 3.")

    check_divis = input("Would you like to check another number's divisibility? Type yes or no: ")
else:
    print("Thanks for using our program.")
```

Programming Challenge: Divisibility Tester Extension

```python
1  # Divisibility Tester Extension
2
3  # set up control variable
4  check_divis = "yes"
5
6  # ask user for divisor input
7  divisor = input("Pick a divisor: ")
8
9  while check_divis == "yes":
10     num = int(input("Enter a number to check divisibility by " + divisor + ": "))
11
12     if num % int(divisor) == 0:
13         print("Your number is divisible by " + divisor + ": ")
14     else:
15         print("Your number is not divisible by " + divisor + ": ")
16
17     # update control variable
18     check_divis = input("Would you like to check another number's \
19 divisibility? Type yes or no: ")
20
21 print("Thanks for using our program.")
22
```

Sentinels

A sentinel value is a predefined value that the user can type in to indicate that they are finished entering data

Example:

```
>> Enter a number. Enter '0' to end: 5

>> Enter a number. Enter '0' to end: 8

>> Enter a number. Enter '0' to end: 0

>> Input finished.
```

Sentinels

```python
# Prompt the user to enter numbers and use '0' as the sentinel

while True:
    num = int(input("Enter numbers. Enter '0' to end: "))

    if num == 0:
        break  # Exit the loop when the sentinel value '0' is entered

# Display a message indicating the end of input
print("Input finished.")
```

Accumulator Variable

— a container that keeps track of a running total
— typically initialized to an initial value and then updated or modified
— used in loops to add up or collect information as the loop progresses
— a "virtual piggy bank"

Accumulator Variable

```python
 1  # Adding Machine
 2
 3  # initialize accumulator variable
 4  total = 0
 5
 6  while True:
 7      num = int(input("Input an integer to add: "))
 8
 9      # break condition with sentinel
10      if num == 0:
11          break
12      else:
13          total += num # add num to running total
14
15  # return total
16  print("Your total is:", total)
17
```

Programming Challenge: Calculate Running Average

Write a program to enter multiple test scores. When the user enters '-1', end the program and return the average score.

```
>> Enter a test score (type -1 to end): 100
>> Enter a test score (type -1 to end): 80
>> Enter a test score (type -1 to end): 50
>> Enter a test score (type -1 to end): -1
>> Your test average is:  76.67%
```

Programming Challenge: Calculate Running Average

```python
# initialize counter variable
count = 0

# initialize accumulator variable
total = 0

while True:
    score = int(input("Enter a test score (type -1 to end): "))

    # break condition with sentinel
    if score == -1:
        break

    count += 1 # increment count
    total += score # add score to running total

# return average
average = total/count
print("Your test score average is:", format(average/100, ".2%"))
```

Programming Challenge: Marbles

— Assume you have a jar that contains 5 marbles. The jar can hold 10 marbles total.
— Continually ask the user if they want to add or remove a marble
— If they add a marble you should increase the total # of marbles in the jar. If the jar is full tell the user and end the program.
— If they remove a marble you should decrease the # of marbles in the jar. If the jar is empty you should tell the user and end the program.

```python
marbles = 5

while True:
    response = input("Would you like to (a)dd or (r)emove a marble: ")

    if response == "a":
        marbles += 1
    elif response == "r":
        marbles -= 1
    else:
        print("Invalid input.")

    if marbles == 10:
        print("The jar is full.")
        break
    if marbles == 0:
        print("The jar is empty.")
        break

print("Thank you for using our program.")
```

Homework

— Assignment #3 (due Thurs @ 11:59PM)
— Self-Paced Learning Module #5 (due next Tues)
— Quiz #5 (due next Tues)