Condition Controlled Loops
Introduction to Programming - Python
Decision Structures Review
Programming Challenge: Review

• Ask the user for a number from 1 to 7.

• Tell the user which day of the week was selected!

• Assume the week starts on a Sunday.

• If the user enters a number that is not valid … be sure to tell the user that negative numbers, zero, and numbers over 7 do not represent a day of the week!
Trace the Output

```python
a = 5
b = 10

if a < b:
    print("one")
if a > b:
    print("two")
if a*2 == b:
    print("three")
if b < a:
    print("four")
```
Trace the Output

```
a = 5
b = 10

if a < b:
    print ("one")
if a > b:
    print ("two")
if a*2 == b:
    print ("three")
if b < a:
    print ("four")
else:
    print ("five")
```
Trace the Output

```python
a = 5
b = 10

if a < b:
    print("one")
elif a > b:
    print("two")
elif a*2 == b:
    print("three")
elif b < a:
    print("four")
else:
    print("five")
```
Generating Random Numbers
Random Integer Example

# ask Python to import the random module
import random

# generate a random number
num = random.randint(1,5)

print("your lucky number is", num)
Programming Challenge

• Ask the user to guess a number between 1 and 10. Assume they will enter an Integer.

• Have the program randomly pick a number between 1 and 10 that is your “secret” number (for example, 5)

• If the user types in your secret number, tell them that they win!

• If the user types in a number less than or greater than your secret number, tell them that they’re either above or below the number and to try once again.

• Every time you play, the secret number should change.
Programming Challenge: Rock, Paper, Scissors

• Write a program to ask the user to select one of three options - Rock (r), Paper (p) or Scissors (s)

• Use the random.randint() function to select an option for the computer

• Determine the winner and print the result.
  • Rock beats Scissor
  • Scissor beats Paper
  • Paper beats Rock
Random Numbers in the Wild

Digital Monkeys with Typewriters recreate Shakespeare
Random Painting
Random Numbers in the Wild

http://alteredqualia.com/visualization/evolve/
Repetition Structures
Repetition Structures

• Programmers commonly find that they need to write code that performs the same task over and over again
# set up our graphical canvas
# width = 500, height = 500
turtle.setup(500, 500)
wn = turtle.Screen()

# hide the turtle
turtle.hideturtle()

# set the speed to the absolute fastest speed possible
turtle.speed(0)

# set up our background
wn.bgcolor("red")

# pick up the pen and move to a random position on the screen
turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

# draw a square
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)

# pick up the pen and move to a random position on the screen
turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

# draw a square
turtle.forward(100)
turtle.right(90)
turtle.forward(100)

turtle.pendown()

# draw a square
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)

turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

# draw a square
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)

turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

# draw a square
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)

turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()
Repetition Structures

• In the previous example our code ended up being one long sequence structure which contained a lot of duplicate code

• There are several disadvantages to this approach
  • Your programs will tend to get very large
  • Writing this kind of program can be extremely time consuming
  • If part of the duplicated code needs to be corrected then the correction must be implemented many times
Repetition Structures

- One solution to this kind of problem is to use a repetition structure, which involves the following:
  - Write the code for the operation one time
  - Place the code into a special structure that causes Python to repeat it as many times as necessary

- We call this a “repetition structure” or, more commonly, a “loop”

- There are a variety of different repetition structures that can be used in Python
Condition Controlled Loops
Condition Controlled 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.
Condition Controlled Loops
The “While” Loop

• In Python we can implement a condition controlled loop by writing a “while” loop

• “while” loops work as follows:
  • Evaluate a Boolean expression.
  • If it is False, skip the block of statements associated with the while loop and condition the program as normal
  • If it is True
    • Execute a series of statements.
    • At the end of the statement block re-evaluate the condition
    • If it is True, repeat the block of statements
    • If it is False, skip the block of statements associated with the while loop and continue the program as normal
The “While” Loop

while condition:
  statement
  statement
  statement
  statement

{  the statements that will be repeated

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

standard Boolean condition that evaluates to True or False
# set up our graphical canvas
# width = 500, height = 500
turtle.setup(500, 500)
wn = turtle.Screen()
# hide the turtle
turtle.hideturtle()

# set the speed to the absolute fastest speed possible
turtle.speed(0)

# set up our background
wn.bgcolor("red")

# pick up the pen and move to a random position on the screen
turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

# draw a square
x=1
while x<5:
    turtle.forward(100)
    turtle.right(90)
    x+=1

# pick up the pen and move to a random position on the screen
turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

# draw a square
while x<5:
    turtle.forward(100)
    turtle.right(90)
    x+=1

# pick up the pen and move to a random position on the screen
Some notes of “while” loops

- We refer to the process of going through a loop as an “iteration”

- If a loop cycles through 5 times then we say we have “iterated” through it 5 times

- The “while” loop is considered a “pre-test” loop, meaning that it only iterates upon the successful evaluation of a condition

- This means that you always need to “set up” your loop prior to Python being able to work with it (i.e. setting up a control variable)
Warning!

- When working with a “while” loop there is nothing to prevent you from writing a Boolean condition that will never evaluate to False

- If this happens your loop will continue executing forever, or until you send an “interrupt” to IDLE using the CTRL-C key combination

- We call this an “infinite loop” since it never stops executing

- With the exception of a few special cases you want to try and avoid writing infinite loops
Trace the Output

```python
a = 5
while a < 10:
    print("A is less than 10!")
```
Programming Challenge: Guess the Number

- Rewrite the "guess the number" game we wrote back in the selection statement unit to use a "while" loop

- Allow the user to continually guess a number until they eventually guess the correct number
Accumulator Variables and Augmented Assignment Operators
Accumulator Variables

• Many programming tasks require you to calculate the total of a series of numbers or the number of times you iterate through a loop (kind of like your homework!)

• We can utilize an “accumulator” variable to do this.
Using Accumulator Variables

- Set up your accumulator variables outside of your loops. I generally initialize my accumulator variables right before I enter a repetition structure.

- Decide on a value you want to start your accumulator values at. 0 or 0.0 is generally a good starting point depending on whether you are counting whole numbers or numbers with fractional values.

- Use a self-referential assignment statement when incrementing an accumulator variable. Example:
  
  - counter = counter + 1
Self-referential assignment statements

```plaintext
# default x to 5
x = 5

x = x + 1
5 + 1 = 6
```
# set up our graphical canvas
# width = 500, height = 500
turtle.setup(500, 500)
wn = turtle.Screen()
# hide the turtle
turtle.hideturtle()

# set the speed to the absolute fastest speed possible
turtle.speed(0)

# set up our background
wn.bgcolor("red")

# pick up the pen and move to a random position on the screen
turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

draw a square
x=1
while x<5:
    turtle.forward(100)
turtle.right(90)
x+=1
Augmented Assignment Operators

- The self-referential assignment statement that we just used is extremely useful, and can be extended to use any of the other math operations we have covered in class so far.
  - $a = a + 1$
  - $b = b * 2$
  - $c = c / 3$
  - $d = d - 4$
Augmented Assignment Operators

• However, Python (and most other programming languages) contains a series of “shortcuts” that can be used to cut down the amount of typing when working with self-referential assignment statements.

• We call these shortcuts the “augmented assignment operators”
# Augmented Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Usage</th>
<th>Equal to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>c += 5</td>
<td>c = c + 5</td>
</tr>
<tr>
<td>-=</td>
<td>c -= 2</td>
<td>c = c – 2</td>
</tr>
<tr>
<td>*=</td>
<td>c *= 3</td>
<td>c = c * 2</td>
</tr>
<tr>
<td>/=</td>
<td>c /= 3</td>
<td>c = c / 3</td>
</tr>
<tr>
<td>%=</td>
<td>c %= 3</td>
<td>c = c % 3</td>
</tr>
</tbody>
</table>
Programming Challenge: AI “Guess a Number” game

• Write a program that asks the user to supply a secret number between 1 and 1,000,000

• Then have the computer continually guess until they find the secret number

• Keep track of the number of attempts

• Extension: How can this be optimized?
Programming Challenge: ELIZA-Line of Inquiry

• Take your Eliza code from last week—save it with a new version number

• add a keyword look out re.search() for the word “game”

• reply with “I love games—Rock, Paper, Scissors—and shoot”

• wait for user input

• say which one ELIZA pulled

• End the game when either the player or the computer earns 3 points

see youShoot.py
Sentinels
Sentinels

• Imagine that you want to ask your users to enter in a large number of items that need to be calculated in a certain way.

• You don’t know how many values the user will be entering.

• Given our current toolset we really only have ways to handle this kind of scenario:
  • Ask the user at the end of each iteration if they want to continue. This can be annoying and make your program cumbersome if you will be entering in hundreds or thousands of values.
  • Ask the user ahead of time how many items they will be entering. This can be difficult since the user may not know at the beginning of the loop how many items they will be working with.
Sentinels

- A sentinel value is a pre-defined value that the user can type in to indicate that they are finished entering data.

Example:
- >> 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): -1
- >> Your test average is: 90%

In the example above the value -1 is considered a sentinel -- it indicates to the program that the user is finished entering data.

Sentinels must be distinctive enough that they will not be mistaken for regular data (in the previous example the value -1 was used – there is no way that a “real” test value could be -1).
Programming Challenge: Adding Machine

• Write a program that asks the user for an integer

• Add the supplied integer to a total variable

• Draw that amount of squares

• Ask the user for another integer

• Draw that amount of squares

• When the user enters a 0 value end the program and display the sum (and all the squares) for the user
# make the turtle graphics module available
import turtle

# also make the random module available
import random
z=float(input("How many squares you want?"))

# set up our graphical canvas
# width = 500, height = 500
turtle.setup(500, 500)
wn = turtle.Screen()
# hide the turtle
turtle.hideturtle()

# set the speed to the absolute fastest speed possible
turtle.speed(0)

# set up our background
wnbgcolor("red")
y=1
while y<z:
    # pick up the pen and move to a random position on the screen
turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

    # draw a square
x=1
while x<5:
    turtle.forward(100)
turtle.right(90)
x+=1
y+=1
Repetition Flow Control
The “break” command

- The “break” command is a special Python command that can be used to immediately end a loop.

- It will not, however, end your program – it simply ends the current repetition structure and allows the program to pick up from the line directly after the end of your loop.

- Note that when the break command runs it will immediately terminate the current loop, which prevents any commands in the loop after the break command from running.
Trace the Output

```python
x = 0

while x < 10:
    if x >= 3:
        break

print(x)

x += 1
```
Prime Number Tester

• Write a program that asks the user for an integer

• Test to see if the number is prime. A prime number is any number that is evenly divisible by 1 and itself.

• If it is draw a square on that location and up until 500,500
Simple Data Validation
Simple Data Validation

• Often we need to ask the user to supply a value in our programs

• But as you know you can't always trust the user to supply you with usable data!

• One strategy you can use to ensure that you get "good" data is to "validate" the user's input. This involves asking the user for a value – if it meets our criteria we can continue. If not we will need to ask the user to re-supply the value.
Programming Challenge

- With our ELIZA Program, which game they would like to play (pretend ELIZA can play three games of your choice)

- If the user supplies an invalid game you should re-prompt them

- See HAL for reference: https://www.youtube.com/watch?v=ARJ8cAGm6JE&spfreload=10

- Once you have a game you can play start the game sequence.
Infinite Loops
Infinite Loops

• What happens when the condition is never false?

• The loop will run forever! (Or at least … as long as the computer is powered up … )

• In general, we want to be avoid infinite loops.
Infinite Loops: An Artistic View

• M.C. Escher (1898-1972) was a Dutch graphic artist who used mathematical concepts in some of his drawings. (http://en.wikipedia.org/wiki/M._C._Escher)

• Here is one vision of an infinite loop:
# make the turtle graphics module available
import turtle

# also make the random module available
import random

# set up our graphical canvas
# width = 500, height = 500
turtle.setup(500, 500)
wn = turtle.Screen()
# hide the turtle
turtle.hideturtle()

# set the speed to the absolute fastest speed possible
turtle.speed(0)

# set up our background
wn.bgcolor("red")
z=1
y=1
while y<=z:
    # pick up the pen and move to a random position on the screen
    turtle.penup()
turtle.goto(random.randint(-250, 250), random.randint(-250, 250))
turtle.pendown()

    # draw a square
    x=1
    while x<=5:
        turtle.forward(100)
turtle.right(90)
x+=1
    y+=1
z+=1