Condition Controlled Loops
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!
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

```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")
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")
```
Trace the Output

```
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")
```
Sequence structures are sets of statements that execute in the order in which they appear.

A selection statement will perform an action only if a certain condition is exist.

A single alternate branch of execution does not allow for an alternate path given an input.

The IF-ELSE structure allows us to perform one set of statements if a condition is true, and another if it is false. It is like a dual alternate branch.
**Review**

**ELIF** is an optional structure that can be placed between your IF and ELSE statements.

It allows you to evaluate additional conditions at the same level as the original IF statement.

Conditions are tested in the order in which they are written. Once a condition evaluates to True all future conditions are skipped.
Python Modules

A Python module is a file containing a set of Python definitions and statements that have been grouped together due to some shared commonalities.
Modules are specialized function libraries that we can import into our programs.

Importing a module tells python that we will need to access that module and all of the functions that it might contain.
Python Modules

Want to learn about modules? Check out the python docs:

http://docs.python.org/3/

or

https://docs.python.org/3/py-modindex.html
Python Modules

or...

help('modules')

help('math')
Sometimes you need your program to generate information that isn’t available when you write your program.

One way to solve this problem is to ask your programming language to select a “random number” – from there you can use this number to construct a somewhat random set of running conditions.
You can generate a random number by using the `randint()` function. This function takes two parameters (a starting integer and an ending integer) and returns one value (a random integer in this range).

In order to use the `randint()` function you must first "import" the "random" module so that Python can access the necessary code library.
# ask Python to import the random module
import random

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

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

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 Scissors
- Scissors beats Paper
- Paper beats Rock
Random

Give 1 million monkeys 1 million typewriters and they'll eventually type the entire works of William Shakespeare.
Random
Repetition

“Repetition is the reality and seriousness of life” - Soren Kierkegaard
Repetition Structures

So far we have been writing code that will ask questions, get answers, possibly compute for a given single time run. What if we have multiple entries to enter?
Repetition Structures

Programmers often have to write code over and over again to perform a task.

Duplicating the code works but can be cumbersome when it is in need of being updated.

Duplicating code can also yield very large programs.

This can be time consuming and we have better things to do... like taking over the world right?
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
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
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 ‘Wild’ ahem excuse me the ‘While’ Loop

the “while’ keyword begins a repetition statement

while condition:
  statement
  statement
  statement

condition to be tested, evaluates to True or False

colon denotes end of condition

statements to be executed repeatedly

indentation indicates that the statements under the while should be repeated
Write a program that allows the user to calculate sales commission earned by each member of a sales team.

Input
- Gross sales (float)
- Commission Rate (float)

Process
- Commission = gross sales * commission rate

Output
- Commission earned
The ‘Wild’ ahem excuse me the ‘While’ Loop

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

```
a = 5

while a < 10:
    print("A is less than 10!")
```
Write a program that allows the user to convert a temperature in Fahrenheit into Celsius using the following formula

Celsius = (Fahrenheit – 32) * 5/9

After calculating the temperature ask the user if they wish to continue. If so, repeat the conversion with a new number. Otherwise end the program.
Write a program that lets a user test to see if a series of numbers are evenly divisible by another number (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)
Challenge

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
Challenge

Write a program that asks the user for three numbers.

Test those numbers against three “secret” numbers that represent the combination to a virtual padlock.

If the user gets the numbers right you should let them know that they have gained access to your program.

If not, allow them to continue to enter combinations until they guess correctly.
Write a program that asks the user to answer a simple math problem (5 + 6)

Continually prompt the user for the correct answer. If they answer correctly, congratulate them and end the program. If they answer incorrectly you should re-prompt them for the answer a second time.

Extension: Randomize the numbers used in the math problem

Extension: Randomize the type of problem the user is presented with (i.e. addition, subtraction)
Many programming tasks require you to calculate the total of a series of numbers or the number of times you iterate through a loop…

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

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

# default x to 5
x = 5

x = x + 1

5 + 1 = 6
Write a program that asks the user to continually enter in the following information for a group of employees:

- Sales
- Commission Rate

Calculate the commission earned by multiplying sales * commission rate

Keep track of the following information and print out a summary document at the end of your loop:

- Number of employees
- Total sales
- Average sales
- Total commission due
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.

\[
\begin{align*}
a &= a + 1 \\
b &= b * 2 \\
c &= c / 3 \\
d &= d - 4
\end{align*}
\]
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 * 3</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>
Challenge

Write a program that asks the user to enter in a series of price values

Calculate a running total of these values

Calculate sales tax (7%) on the total bill and display the result to the user at the end of the program
Challenge

Write a program that asks the user to enter in a test score along with the total points possible for the test

Allow the user to enter in as many scores as he or she wishes

When finished, calculate the user’s average score in the class
Write a program that simulates a coin flipping 1 million times

Count the # of heads and tails that result, and display the result to the user after you have finished running the simulation
Write a program that asks the user 5 simple math problems.

Each problem should utilize random numbers, but you can standardize on a single operation (i.e. subtraction).

Ask the user a question. If they answer correctly, they earn a point. If not, they do not earn a point.

At the end of the program present the user with their score.
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?
Challenge

Write a program that lets the user play a game of Rock, Paper, Scissors against the computer.

End the game when either the player or the computer earns 3 points.
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.
A sentinel value is a pre-defined value that the user can type in to indicate that they are finished entering data.
Sentinels

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)
Challenge

Write a program that asks the user to enter in a series of weight measurements taken over a period of days.

The user can enter as many or as few weight values as they would like. Entering the value “0” should indicate that the user has finished entering data.

Calculate the user’s average weight during this period.

Also calculate their weight change from the beginning of their weight loss program to the end of the program.
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 after the break command from running.
Trace the Output

```python
x = 0

while x < 10:
    if x >= 3:
        break
    print (x)

x += 1
```
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.
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
next steps:

begin “Self Paced Learning Module # 5”

work on Assignment #4: ‘Repetition Structures’