# Loops – While and For

- Repetition Statements
  - While
  - For
- Reading for this Lecture:
  - Dawson, Chapter 3
  - Dawson, Chapter 4 (until p. 93)
  - http://introcs.cs.princeton.edu/python/13flow

# Repetition Statements

- *Repetition statements* – better known as ***loops*** – allow us to execute code multiple times
- The repetition is controlled by boolean expressions
- Python has two kinds of loops:
  - `while`
  - `for`
- The programmer should choose the right kind of loop for the situation

# The while Loop

- A *while loop* has the following syntax:

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

- If **condition** is True, **statement**s are executed
- Then **condition** is evaluated again, and if it is still True, **statement** is executed again
- **statement**s are executed repeatedly until **condition** becomes False

# The while Loop

- An example of a while loop:

```
done = False
while not done:
    body of loop statements
    if some condition:
        done = True
```

- If the condition of a `while` loop is False to begin with, the statements are **never** executed
- Therefore, the body of a `while` loop will execute **0+ times**

# The while Loop

- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum* (for example, a dice game)
- You can have a flag or signal (called a *sentinel value*) that represents the end of input (not data!) and stops the loop
- A loop can also be used for *input validation*, making a program more *robust*

**See sum_loop.py**

**See sum_loop_alt.py**

**See loop_validate.py and exclusive_network.py**

# Infinite Loops

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

- Executing ***statement***s must eventually make ***condition*** False
- If not, you have an *infinite loop*, which will run until the user interrupts the program
- This is a common **logical** error
- You should always double check the logic of your program to ensure that your loops will eventually **terminate**

**See infinite_loop.py**

# Infinite Loops

- An example of an infinite loop:

```
done = False
while not done:
   print ("Whiling away the time ...")
   # Note: no update for the value of done!!
```

- This loop will go on forever (*in theory, at least!*) until the user externally interrupts the program

# Nested Loops

- As with `if` statements, you can have loops inside of loops!
- For each iteration of the outer loop, the inner loop runs through completely
- How many times will the string `"Here"` be printed?

**10 * 20 = 200**

```
count1 = 1
while count1 <= 10:

    count2 = 1
    while count2 <= 20:
        print ("Here")
        count2 += 1

    count1 += 1
```

**See nested_loops.py**

# Indeterminate vs Determinate Loops

- A `while` loop will continue to run until its continuation condition becomes False.
- ***In theory***, what stops the loop is a result of what happens during loop execution, so we may not yet know how many times the loop code should execute, so the while loop is **indeterminate**
- Other times, however, we will be able to determine this in advance – which means we can use a **determinate** loop

# The for Loop

- A *for loop* has the following syntax:

The ***variable*** refers to the current item being processed

The ***collection*** is the series of objects being processed

```
for variable in collection:
    statement
    statement
    statement
    statement
```

The ***statement***s are executed for the current item

# The for Loop

- A `for` loop – a **determinate** loop – is functionally equivalent to the following `while` loop structure:

```
size = len(collection)
counter = 0
while counter < size:
    variable = collection[counter]
    statement
    statement
    statement
    counter += 1
```

# The for Loop

- An example of a `for` loop:

```
for count in range(5):
    print (count)
```

- The **variable** section can be used to declare a variable for counting
- Like a `while` loop, the execution is dependent on a **condition** (here, implicit)
- Therefore, the body of a `for` loop will execute **0+ times**

# The for Loop

- You can even count by multiples:

```
# Prints by fives

for i in range(0, 50, 5):
    print(i, end=" ")
```

- A `for` loop is well suited for executing the body a specific number of times that can be calculated or determined in advance
- See counter.py
- See loopy_string.py