Logistics

- Final Exam Re-Scheduled!

Tuesday, May 9th

2:00 – 3:50 pm

Room TBD
Count Controlled Loops
Python Loops

- A condition controlled loop iterates a variable number of times – we control the # of iterations through our Boolean condition

- A count controlled loop is a repetition structure that iterates a specific number of times
Count Controlled loops

- Python (and all other programming languages) have special structures which can be used to implement a count controlled loop without needing to use a condition controlled loop (though you could always use a condition controlled loop if you wanted to)
Count Controlled Loops

- You can write a count controlled loop using a while() loop. For example:

```python
counter = 0
while counter < 5:
    print("This will print 5 times")
    counter += 1
```
The “for” loop

- The “for” loop is Python’s native count controlled loop.
- Example:

```python
for num in [1,2,3,4,5]:
    print ("This will print 5 times")
```
The “for” loop

```
for variable in [value1, value2, etc]:
    statement
    statement
    statement
    {
        indentation
        statements to be executed
    }
```

- "for" keyword starts the loop
- "in" keyword
- list of items to iterate over
- target variable
The “for” loop

- The “for” loop will iterate once for each item defined in the list passed to it when the loop begins.

- Lists in Python are defined by the square bracket characters “[“ and “]”. Items in a list are separated by a comma.

- The first time a “for” loop iterates the target variable will assume the value of the first item in the list.

- The second time a “for” loop iterates the target variable will assume the value of the second item in the list.

- This continues until you reach the end of the list.
The “for” loop

for c in [1, 2, 3, 4]:
    print (c)

1
2
3
4
The “for” loop

- We will talk more about lists near the end of the semester. With that said, lists can contain collections of different kinds of data. For example:

```python
for name in ['Craig', 'John', 'Chris']:
    print("The current user is:", name)
```
Programming Mechanics Challenge

- Rewrite the following loop as a “for” loop:

```python
x = 0
while x < 5:
    print ("hi")
    x += 1
```
Programming Challenge: The Bug Collector

- A bug collector collects bugs every day for seven days.
- Write a program that keeps a running total of the number of bugs collected during the seven days. The loop you write should ask for the number of bugs collected for each day, and when it finishes it should display the total number of bugs collected.
Programming Challenge: Student Test Scores

- Write a program that iterates over the following student names:
  - John
  - Mary
  - Michael
  - Sophie

- Ask the user to input a test score for each student

- Calculate the average test score for the class
Programming Mechanics Challenge

- Rewrite the following loop as a “while” loop:

```python
for x in [10, 20, 30, 40]:
    print ("hi")
```
A Digression: Data Types
Data Types

- All data is represented internally as binary, 0's and 1's.

- In order to be useful, computer has to know how to interpret that data, so we have data types.

- A data type generally has a set of possible values, and a set of legal operations that can be performed on values of that type.
<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>1, 0, -23234</td>
<td>+,-,*,/,,,%</td>
</tr>
<tr>
<td>float</td>
<td>0.2, 3.54e+20</td>
<td>+,-,*,/</td>
</tr>
<tr>
<td>string</td>
<td>&quot;a&quot;,&quot;hello world&quot;</td>
<td>+,* ,len()</td>
</tr>
</tbody>
</table>
The range() function
Iterating over iterators

- For loops iterate over every item in an iterator object.

- An iterator object:
  - is a thing (object) that contains other values / members
  - is capable of returning each of its members one at a time

- Examples of iterator objects include:
  - Strings
  - Range objects (we'll learn these today)
  - Lists (for the future!)
The range() function

- So far we have been iterating over lists of pre-defined values in our for() loops

- Example:

  ```python
  for x in [1,2,3,4,5]:
      print ('hi')
  ```

- The range() function lets you dynamically generate iterators based on criteria that you define

- The range() function takes at least one argument. In its simplest form it takes a single integer.
The range() function

```python
for i in range(5):
    print ('iteration #', i)
```

```
iteration # 0
iteration # 1
iteration # 2
iteration # 3
iteration # 4
```
The range() function

- The range() function creates a value of type "range", which is another Python data type, and is iterable.

- A range is an arithmetic sequence of numbers

- When passed a single integer the range function will generate a range with values from 0 to the number specified minus one.

- Try this:

  ```python
  nums = range(5)
  print(nums)
  print(type(nums))
  ```
Range vs. List

- The range() function creates a value of type "range".

- When you print the result of the range() function, you see the string representation of the range, which is just the arguments you called range() with.

- To convert a range() to a list, use the list() function:
  ```python
  print(list(nums))
  ```

- For now, we'll just use this when we want to double-check the values in our range.
The range() function

- You can pass additional parameters to the range() function to cause it to behave differently
  
  \[ \text{range(start,stop,step)} \]

- Examples:

  \[
  \begin{align*}
  \text{range}(1,5) & \quad \# \text{ set a start and end value for the range} \\
  & \quad \# \, [1,2,3,4] \\
  \text{range}(5,10) & \quad \# \, [5,6,7,8,9] \\
  \text{range}(0,10,2) & \quad \# \text{ set a start, end and step (or increment) value} \\
  & \quad \# \, [0,2,4,6,8] \\
  \text{range}(1,10,2) & \quad \# \, [1,3,5,7,9]
  \end{align*}
  \]
The `range()` function

- You can pass additional parameters to the `range()` function to cause it to behave differently
  
  `range(start, stop, step)`

- With two arguments, `range()` returns a series of numbers starting with `start`, counting up to, but not including `stop`.
  
  `range(5) == range(0, 5)`

- With three arguments, `range()` returns a series of numbers starting with `start`, up to, but not including `stop`, counting by `step`.
  
  `range(5) == range(0, 5, 1)`
# The range() function

<table>
<thead>
<tr>
<th>range() function call</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>range(5)</td>
<td>[0, 1, 2, 3, 4]</td>
</tr>
<tr>
<td>range(10)</td>
<td>[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]</td>
</tr>
<tr>
<td>range(1, 5)</td>
<td>[1, 2, 3, 4]</td>
</tr>
<tr>
<td># set a start and end value for the range</td>
<td></td>
</tr>
<tr>
<td>range(5, 10)</td>
<td>[5, 6, 7, 8, 9]</td>
</tr>
<tr>
<td>range(0, 10, 2)</td>
<td>[0, 2, 4, 6, 8]</td>
</tr>
<tr>
<td># set a start, end and step (or increment) value</td>
<td></td>
</tr>
<tr>
<td>range(1, 10, 2)</td>
<td>[1, 3, 5, 7, 9]</td>
</tr>
</tbody>
</table>
Reverse ranges

- The step value passed to the range() function does not necessarily have to be positive.

- If you pass a negative step value to the range() function it will count backwards for you.
  - `range(8, 3, -2) → 8, 6, 4`
  - `range(100, 0, -25) → 100, 75, 50, 25`

- Start and stop can also be negative:
  - `range(-10, 10, 5) → -10, -5, 0, 5`
  - `range(100, -100, -25) → 100, 75, 50, 25, 0, -25, -50, -75`
Using loop targets
Loop targets

- In a for loop we generally use the target variable as a reference value for some kind of calculation.
- Remember that the value of the target variable changes with each iteration of the loop.
Programming Challenge: Squares

- Write a program that calculates the square of the numbers between 1 and 10

<table>
<thead>
<tr>
<th>Number</th>
<th>Square</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>
Programming Challenge: Stair Steps

- Write a program that prints out the following pattern of characters:

  **
  ****
  ******
  ********
  **********
  ***********
  *****************
Programming Challenge: Stair Steps

Write a program that prints out the following pattern of characters:

```
  **
 ****
******
*******
********
*********
**********
***********
```

Programming Challenge: Divisibility

- Write a program that asks the user to enter in an integer
- Then find all numbers between 1 and 10,000 that are evenly divisible by that number
Programming Challenge: Divisibility

Extensions:
- Extend your program to collect two integers when it starts up
- Find all numbers in the specified range that are divisible both of the two supplied numbers
- Extension: print your results such that you print 10 #'s per line
User controlled ranges
User controlled ranges

- In many cases a programmer knows how many iterations he or she needs in order to accomplish a desired task.

- However, sometimes we need to ask the user to control the number of iterations within a loop.

- You can easily do this by substituting a variable within the `range()` function to control the start, end and step values of the iterator that will be generated.
Programming Challenge: (imperfect) Lottery # Picker

- Write a program that generates random lottery numbers for the user

- Ask the user for the number of digits they need as well as the high and low value of each digit (i.e. 6 digit number with digits ranging from 1 to 60)

- Generate the desired lottery number
Programming Challenge: Blast Off

- Write a countdown program that prompts the user for a max value (i.e. 30)
- Print out a countdown from that number down to zero, then print “blast off!”
For vs. While
Use a for loop when...

- You know ahead of time how many iterations you'll have to go through
- You have an iterator structure that you have to traverse
  - a sequence of numbers
  - a list of items
Use a while loop when...

- You don't know how many iterations you'll have to go through!
- You must repeat something until some condition is met
- Generally not a great option for counting (need to keep track of counter separately)
Nested Loops
Nested Loops

- A nested loop can be described as a “loop inside of a loop”

- It’s the same idea as nested selection statements (“if” statements inside other “if” statements)
Write a program that prints out an Addition table for the number 5. For example:

- 5 plus 1 is 6
- 5 plus 2 is 7
- 5 plus 3 is 8
- ...
- 5 plus 10 is 15
Write a program that prints out an Addition table for the numbers 5 and 6. For example:

5 plus 1 is 6
5 plus 2 is 7
5 plus 3 is 8
...
5 plus 10 is 15

6 plus 1 is 7
6 plus 2 is 8
6 plus 3 is 9
...
6 plus 10 is 16
Addition Tables

- Write a program that prints out the Addition tables for the numbers 1 through 10.

- Extend your program to allow the user to type in a range of Addition tables they want printed. Example:

  Addition Table Generator 2000!
  Enter the first number in your range: 1
  Enter the last number for your table: 20
Programming Challenge: Multiplication Tables

- Write a program that prints out a multiplication table for the numbers 1 through 5:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

- Can you modify your program to print out a multiplication table for the numbers 1 through 10?
Programming Challenge: Clock Simulator

- Write a program that prints out every possible time value for a single day.

- Print out the hours, minutes and seconds starting at midnight and continue on to 11:59.59 PM.

- Output each value as follows:
  - 0:0:0
  - 0:0:1
  - 0:0:2
  - ...
  - 23:59:59
Some notes on Nested Loops

- Some notes on nested loops:
  - The innermost loop will iterate through all its iterations for every single iteration of an outer loop
  - Inner loops complete their iterations faster than outer loops
  - To get the total number of iterations of a nested loop, multiply the number of iterations of all the loops
Programming Challenge: Gradebook System

- Write a program that lets a teacher calculate graders for his or her class.
- Ask the teacher for the # of students in class as well as the # of assignments.
- Allow the teacher to input the desired values and calculate the average score for each student based on the information given.
Programming Challenge: Graphical Pattern

- Write a program that prints the pattern to the left using nested loops
Programming Challenge

- Reproduce the pattern to the right using nested loops

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
Programming Challenge

- Write a program to test to see if a given number is prime (this does not require a nested loop)
- Next, use this program to all prime numbers between 2 and 1,000
Nested Loops in the Wild!
Nested Loops in the Wild: Decorative Arts

- There are many examples of the Decorative Arts using nested loops to create wonderful and colorful patterns!

- The Grammar of Ornament (http://digicoll.library.wisc.edu/cgi-bin/DLDecArts/DLDecArts-idx?id=DLDecArts.GramOrnJones) is a Victorian Classic ... a collection of patterns used in the Decorative Arts Around the World.
+ Nested Loops in the Wild: Decorative Arts

- How many repeating and nested patterns can you isolate in the following Ancient Greek designs? (Source)