Name:______________________________ **Partner:** ________________________

Python Activity 14: Looping Structures -- Nested Loops

To look through a sequence of sequences, we need a loop of loops!

Learning Objectives
Students will be able to:
Content:
- Trace through the output of nested for.. loops with lists and strings
- Identify inner and outer loops

Process:
- Write code that uses a **nested for.. loop** with accumulator variables

Prior Knowledge
- for-each loops, lists, strings, range

FYI: **Stepping** or **tracing through code** by hand is an important skill for debugging logic errors related to *computational thinking*. Keep track of the relevant variables' values and how they change line-by-line.

1. Observe the following code snippet:

Python Program

```python
def mystery_print():
    for letter in ['b', 'd', 'r']:
        for suffix in ["ad", "ib", "ump"]:
            print(letter + suffix)

mystery_print()
```

a. Examine the code above. What is the output of this program? Trace through the values as they change:

	letter	suffix	printed
Before the outerloop:			
Outer Iteration 0:			
Inner Iteration 0:			
Inner Iteration 1:			
Inner Iteration 2:			
Outer Iteration 1:			
Inner Iteration 0:			
Inner Iteration 1:			
Inner Iteration 2:			
Outer Iteration 2:			
Inner Iteration 0:			
Inner Iteration 1:			
Inner Iteration 2:			

FYI: A loop within another loop is known as a **nested loop.** Proper indentation is essential for the loops to work correctly.

b. How many for-each loops are in this code? ________ Is one loop completely executed before the next loop begins? ________ What do you call this type of loop?
____

c. Label the **inner loop** and the **outer loop**.

d. What does the **inner loop** do? ____
How does the **inner loop** know when to stop? ______________________________________

e. What does the **outer loop** do? ____
How does the **outer loop** know when to stop? ______________________________________

f. How many times is the following line of code executed in the program? ______________
`print(letter + suffix)`

```
bad
bib
bump
dad
dib
dump
rad
rib
rump
```

g. The following is the code's output, how does it differ from what you expected?
____
____

2. Observe the following code snippet:

Python Program

```
def mystery_return(char, list_of_str):
    locations = []
    for word in list_of_str:
        found = False
        for i in range(len(word)):
            if not found and word[i] == char:
                locations = locations + [i]
                found = True
    return locations

print(mystery_return('e', ["eat", "more", "cheese"]))
```

a. Examine the code above. What is the output of this program? Trace through the values as they change:
`char` → __________
`list_of_str` → ____________________

	locations	word	found	range(len(word))	i
Before the outerloop:					
Outer Iteration 0:		________			
Inner Iteration 0:	____________	________	________	_________________________	____
Inner Iteration 1:	____________	________	________	_________________________	____
Inner Iteration 2:	____________	________	________	_________________________	____

Outer Iteration 1:		________			
Inner Iteration 0:	____________	________	________	_________________________	____
Inner Iteration 1:	____________	________	________	_________________________	____
Inner Iteration 2:	____________	________	________	_________________________	____
Inner Iteration 3:	____________	________	________	_________________________	____
Outer Iteration 2:		________			
Inner Iteration 0:	____________	________	________	_________________________	____
Inner Iteration 1:	____________	________	________	_________________________	____
Inner Iteration 2:	____________	________	________	_________________________	____
Inner Iteration 3:	____________	________	________	_________________________	____
Inner Iteration 4:	____________	________	________	_________________________	____
Inner Iteration 5:	____________	________	________	_________________________	____

b. Label the **inner loop** and the **outer loop**.

c. What does the **inner loop** do? ____
How does the **inner loop** know when to stop? ________________________________

d. What does the **outer loop** do? ____
How does the **outer loop** know when to stop? ________________________________

e. How many times is the following line of code executed in the program?

```
locations = locations + [i]          ____________
if not found and word[i] == char:    ____________
```

Why might the number of times executed be different for these two pieces of code?

____

f. What does the `found` variable do in this code?

____

____

____

FYI: We can use optional **flag variables** with loops to identify when to begin or stop certain code – often used in conjunction with accumulator variables.

g. The following is the code's output, how does it differ from what you expected?

```
[0, 3, 2]
```

____

____

____

3. Observe the following python program:

```
b_str = ''
for i in range(1, 5):
    for j in range(1, 4):
        b_str = b_str + str(i * j) + "\t"
    b_str = b_str + "\n"
```

a. Examine the code above. What is the output of this program? Trace through the values as they change:

`i → range(1,5): [____,____,____,____] j → range(1,4): [____,____,____]`

	i	j	b_str
Before the outer loop:			______________________
Iteration 1:	________	________	______________________
Iteration 2:	________	________	______________________
Iteration 3:	________	________	______________________
Iteration 4:	________	________	______________________
Iteration 5:	________	________	______________________
Iteration 6:	________	________	______________________
Iteration 7:	________	________	______________________
Iteration 8:	________	________	______________________
Iteration 9:	________	________	______________________
Iteration 10:	________	________	______________________
Iteration 11:	________	________	______________________
Iteration 12:	________	________	______________________
Final value			______________________

Application Questions: Use the Python Interpreter to check your work

1. If you were asked to create a Python function that *returned* the adjacent rectangle, you could easily do it with a series of concatenation statements. You can also create it with a for-each loop and accumulator variable with far fewer lines of code. This exercise will go through the steps to create a function that will *return* and *then* print similar output but allows the user to determine the length and width of the figure when they execute the program.

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

a. Create a function, `make_rectangle`, that takes a **string** parameter, `width`, representing the width of the rectangle in characters (i.e., if `width` is "wwww" the function should return `"****"`). Use a for-each loop to *accumulate* the string of asterisks of the correct width. Return this string.

b. You want the function to create several lines of asterisks. Extend the code in (a) to take a second parameter, `height`, that is a **string** representing the height of the rectangle in characters (i.e., if `height` is "`hhh`" the function should return a string with 3 rows of asterisks). Use an "outer" loop to print that many lines of asterisks. Write the revised code below (*Hint: "\n" is the character for newline)*:

```
def make_rectangle(                              )
```

c. Write a main block of code that prompts the user for strings representing the desired height and width of the rectangle, using characters (i.e., "`www`" and "`hh`" will produce a rectangle 3 asterisks wide and two rows tall). Print the rectangle of asterisks.

```
def main():
```

d. Where might you modify your code to test that the width of the rectangle will be less than 10, and display an error message if not? Write the code below:

2. Use two for..loops with range() to *print* the following output:

```
$
$$
$$$
$$$$
*
**
***
****
```

3. Use a **nested** for..loop to *print* the following output, using range():

```
$
*
$$
**
**
$$$
***
***
***
$$$$
****
****
****
****
```