

Name: \_\_\_\_\_

Partner: \_\_\_\_\_

### Python Activity 13: 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 of lists, strings

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

1. Observe the following code:

**Python Program**

```
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
<i>Before the outerloop:</i>			
<b>Outer Iteration 1:</b>	_____		
Inner Iteration 1:	_____	_____	_____
Inner Iteration 2:	_____	_____	_____
Inner Iteration 3:	_____	_____	_____
<b>Outer Iteration 2:</b>	_____		
Inner Iteration 1:	_____	_____	_____
Inner Iteration 2:	_____	_____	_____
Inner Iteration 3:	_____	_____	_____
<b>Outer Iteration 3:</b>	_____		
Inner Iteration 1:	_____	_____	_____
Inner Iteration 2:	_____	_____	_____
Inner Iteration 3:	_____	_____	_____

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)`

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

bad  
bib  
bump  
dad  
dib  
dump  
rad  
rib  
rump

2. Observe the following code:

```

Python Program

def question2():
    lst_lsts = [[1,2,3],[4,5,6],[7,8,9]]
    new_lstlsts = []
    for row in lst_lsts:
        new_row = []
        for item in row:
            new_row = new_row + [item*item]
        new_lstlsts = new_lstlsts + [new_row]
    return new_lstlsts

```

Can be helpful to think of it like this:  
`lst_lsts = [[1,2,3],`  
`[4,5,6],`  
`[7,8,9]]`

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

	new_lstlsts	row	new_row	item
<i>Before the outerloop:</i>	_____			
<b>Outer Iteration 1:</b>	_____	_____	_____	
Inner Iteration 1:	_____	_____	_____	_____
Inner Iteration 2:	_____	_____	_____	_____
Inner Iteration 3:	_____	_____	_____	_____
<b>Outer Iteration 2:</b>	_____	_____	_____	
Inner Iteration 1:	_____	_____	_____	_____
Inner Iteration 2:	_____	_____	_____	_____

Inner Iteration 3: \_\_\_\_\_

**Outer Iteration 3:** \_\_\_\_\_

Inner Iteration 1: \_\_\_\_\_

Inner Iteration 2: \_\_\_\_\_

Inner Iteration 3: \_\_\_\_\_


*Final:* \_\_\_\_\_

b. What does this code do?

\_\_\_\_\_


\_\_\_\_\_

c. There are two accumulator variables in this code, what are their names?  
 \_\_\_\_\_ and \_\_\_\_\_

 Why do we need two accumulator variables?

\_\_\_\_\_

\_\_\_\_\_

 d. What might happen if `[item * item]` did not have square brackets around it in the code above? \_\_\_\_\_

\_\_\_\_\_

What might happen if `[new_row]` did not have square brackets around it in the code above? \_\_\_\_\_

\_\_\_\_\_

**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, `makeRectangle`, that takes a **string** parameter, `width`, representing the width of the rectangle in characters (i.e., if `width` is "www" 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 makeRectangle (
```

---

---

---

---

---

---

---

---

---

---

- 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:

---

---

---

---

---

---