**Name:_____ Partner: _____**

# Python Activity 40: Recursion

**Learning Objectives**
Students will be able to:
*Content:*
- Define **recursion** and explain why it is useful.
- List the **three steps** for building recursive solutions.

*Process:*
- Predict the output of recursive programs.
- Write code that uses recursion.

**Prior Knowledge**
- Activities 1-28, expressions, data types, functions, conditionals

*If you encounter any issues/typos, let Iris know! Questions? Ask Iris or the POGILing forum*

**Concept Model:**
*Consider the following story:*
There once was a dragon and a sorcerer's apprentice. The apprentice was tasked with determining which numbers in a list were odd (in contrast to even, not as in unusual), and he needed the dragon's help. The apprentice went down to the dungeons and asked, "Dragon, I need to know if any of the numbers in this list are odd: [3142, 5798, 6550, 8914]"

The dragon, being surly and rather dissatisfied with his dungeon accomodations replied:
**"Sorry, I can only tell you if the *first* number of the list is odd."**
The apprentice pleaded, "But I need to know if *any* number in the list is odd, not just the first!"
Unmoved, the dragon retorted,
"Well, I'll only look at the first number, but I'll look at as many lists as you like."

1. What should the sorcerer's apprentice do to solve his problem?

_____

_____

And so, the sorceror's apprentice presented the dragon with the original list.
Apprentice: `[3142, 5798, 6550, 8914]`
Dragon: The first number is **not odd**.
And then, the sorcerer's apprentice presented the dragon with a modified version of the list:
Apprentice: `[~~3142,~~ 5798, 6550, 8914]`
Dragon: The first number is **not odd**.
And then, the sorcerer's apprentice presented the dragon with a modified version of the list:

Apprentice: **[~~3142, 5798,~~ 6550, 8914]**
Dragon: The first number is **not odd**.

Once more, the sorcerer's apprentice presented the dragon with a modified version of the list:

Apprentice: **[~~3142, 5798, 6550,~~ 8914]**
Dragon: The first number is **not odd**.

Finally, the sorcerer's apprentice presented the dragon with a modified version of the list:

Apprentice: **[~~3142, 5798, 6550, 8914~~]**
Dragon: That's an **empty list**, it can't be odd!

Quite satisfied with the dragon's input, the apprentice smiled and remarked, "Ah, so none of the numbers are odd, thank you!" The dragon responded, "But how can you know that, I only told you if the first number was odd!"

2. How does the apprentice know that all the numbers are not odd?

_____

_____

The apprentice replied, "I gave you the following sub-lists of my original list, and you gave me an answer for the first item in each:"

```
[3142, 5798, 6550, 8914]
[      5798, 6550, 8914]
[            6550, 8914]
[                  8914]
                      []
```

The dragon simply grumbled, "It looks like you've discovered recursion."

3. How does the dragon know when to stop checking values?

_____

_____

4. What is the process the dragon repeatedly executes?

_____

_____

5. How does the apprentice get the dragon to proceed to the next number?

_____

_____

6. Based on this story, how might you define recursion?

_____

_____

**FYI:** ***Recursion*** is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem. There are three steps to consider when building a recursive problem solution:

1. What is the stopping condition / base case?
2. What is the small, repeated step?
3. How do we break the journey down into a smaller piece?

## Critical Thinking Questions:

1. Examine the sample code below from interactive python which represents the problem & solution from the Concept Model story above:

**Interactive Python**

```
0 >>> def printFirstOdd(lst):
1 ...       if lst:
2 ...             if lst[0]%2 != 0:
3 ...                   print('Odd!:', lst[0])
4 ...             printFirstOdd(lst[1:])
5 >>> mylist = [3142, 5798, 6550, 8914]
6 >>> printFirstOdd(mylist)
```

a. What does each line of code do?

0 _____

1 _____

2 _____

3 _____

4 _____

5 _____

6 _____

b. On which line is the stopping condition? _____

c. On which lines are the small repeated steps? _____

d. On which line is the journey broken down into smaller pieces? _____

e. Which lines might be said to be the Dragon's actions? _____

f. Which lines might be said to be the Apprentice's actions? _____

g. When `myList = [3142, 5798, 6550, 8914]`, what is the argument passed to `printFirstOdd(..)` each time it's called on line 4?

**On line 6:**_____ **Third time (4):**_____

**First time (4):**_____ **Fourth time (4):**_____

**Second time (4):**_____

h. If `myList = [0, 7, 9, 4, 2]`, how many times will `printFirstOdd(..)` be called? What will the arguments passed to the recursive call each time?

_____

_____

_____

_____

_____

2. Examine the sample code below from interactive python which is similar to the previous example:

**Interactive Python**

```
0  >>> def anyOdd(lst):
1  ...     if lst:
2  ...             if lst[0]%2!=0:
3  ...                     return True
4  ...             else:
5  ...                     return False or anyOdd(lst[1:])
6  ...     return False
7  >>> anyOdd([2,4,6,8])
8  False
9  >>> anyOdd([2,4,6,8,11])
10 True
11 >>> anyOdd([2,4,7,8,6])
12 True
13 >>> anyOdd([])
14 False
```

a. What does each line of code do?

0 _____

1 _____

2 _____

3 _____

4 _____

5 _____

6 _____

b. On which line is the stopping condition? _____

c. On which lines are the small repeated steps? _____

d. On which line is the journey broken down into smaller pieces? _____

e. When the initial argument passed to `anyOdd` is `[2,4,6,8]`, as on line 7, what is passed to `anyOdd(..)` as an argument each time it's called on line 5?

**On line 7:**_____ **Third time (5):**_____

**First time (5):**_____ **Fourth time (5):**_____

**Second time (5):**_____

f. When the initial argument passed to `anyOdd` is `[2,4,6,8]`, as on line 7, what is returned by `anyOdd(..)` each time it's called on line 5?

**1st time (5):**_____ `or anyOdd(`_____**)**

**2nd time (5):**_____ `or` _____ `or anyOdd(`_____**)**

**3rd time (5):**_____ `or` _____ `or` _____ `or anyOdd(`_____**)**

**4th time (5):**_____ `or` _____ `or` _____ `or` _____ `or anyOdd(`___**)**

**5th time (5):**_____ `or` _____ `or` _____ `or` _____ `or` _____

g. When the initial argument passed to `anyOdd` is `[2,4,7,8,6]`, as on line 11, what is returned by `anyOdd(..)` each time it's called on line 5?

**1st time (5):**_____ `or anyOdd(`_____**)**

**2nd time (5):**_____ `or` _____ `or anyOdd(`_____**)**

**3rd time (5):**_____ `or` _____ `or` _____ `or anyOdd(`_____**)**

h. Why does line 7 have 5 calls to `anyOdd(..)` and line 11's input only have 3 calls?

_____

i. How many calls of `anyOdd(..)` does line 13 entail?

_____

3. Examine the sample code below from interactive python:

**mystery1.py**

```
0 def mystery1(anyString):
1    if not anyString:
2         return 0
3    else:
4         return 1 + mystery1(anyString[1:])

5 if __name__ == '__main__':
6     print(mystery1('try1'))
```

a. What does each line of code do?

0 _____

1 _____

2 _____

3 _____

4 _____

5 _____

6 _____

b. On which line is the stopping condition? _____

c. On which lines are the small repeated steps? _____

d. On which line is the journey broken down into smaller pieces? _____

e. When the initial argument passed to `mystery1` is `'try1'`, as on line 6, what is passed to `mystery1(..)` as an argument each time it's called on line 4?

**On line 6:**_____ **Third time (4):**_____

**First time (4):**_____ **Fourth time (4):**_____

**Second time (4):**_____

f. When the initial argument passed to `mystery1` is `'try1'`, as on line 6, what is returned by `mystery1(..)` each time it's called on line 4?

**1st time (4):**_____ + `mystery1`**(**_____**)**

**2nd time (4):** _____ +_____ + `mystery1`**(**_____**)**

**3rd time (4):** _____ + _____ +_____ + `mystery1`**(**_____**)**

**4th time (4):** _____ +_____ + _____ +_____ + `mystery1`**(**_____**)**

**5th time (4):** _____ +_____ +_____ + _____ +_____

g. What does `mystery1('try1')` return? _____

h. If we add a line of code to the bottom, `print(mystery1('try')`, what will be returned by `mystery1(..)` each time it's called on line 4?

**1st time (4):**_____ + `mystery1`**(**_____**)**

**2nd time (4):** _____ +_____ + `mystery1`**(**_____**)**

**3rd time (4):** _____ + _____ +_____ + `mystery1`**(**_____**)**

**4th time (4):** _____ +_____ +_____ + _____

i. What does `mystery1('try')` return? _____

j. What might `mystery1('hello')` return? _____

k. What does the `mystery1(anyString)` function do?

_____

**Application Questions: Use the Python Interpreter to check your work**

1. Write a recursive function, `recursiveListLength(anyList)`, that will return the length of a given list, `anyList`, using recursive means:

2. What does the following program do? (Step-through with examples for `fi, inSequence`).

```
def mystery2(fi, inSequence):
    if not inSequence:
        return False
    elif fi == inSequence[0]:
        return True
    else:
       return False or mystery2(fi, inSequence[1:])
```

3. Write a recursive function, `countChar(ch, anyString)`, that will count the number of occurrences of a given character, `ch`, in a given string, `anyString`, using recursive means:

4. Write a recursive function, `getItem(index, start, anyList)`, that will return the element located at `index`, in a given list, `anyList`, using recursive means. It should start looking at the `index` provided in start and should return `None` if that index is not found: