Folks, this is a brand new activity. If you encounter any issues/typos, please let Iris know!

Name:________________________________ **Partner:** ___________________________

Python Activity 33: Recursion versus Iteration

When should we choose a recursive approach versus an iterative approach?

Learning Objectives
Students will be able to:
Content:

- List **pros and cons** of using recursion to solve problems
- Diagram the function frame stack for recursive & iterative functions

Process:

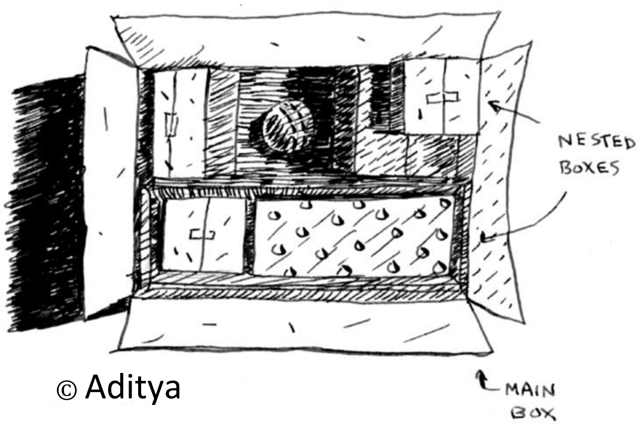
- Write iterative and recursive solutions to a given problem

Prior Knowledge

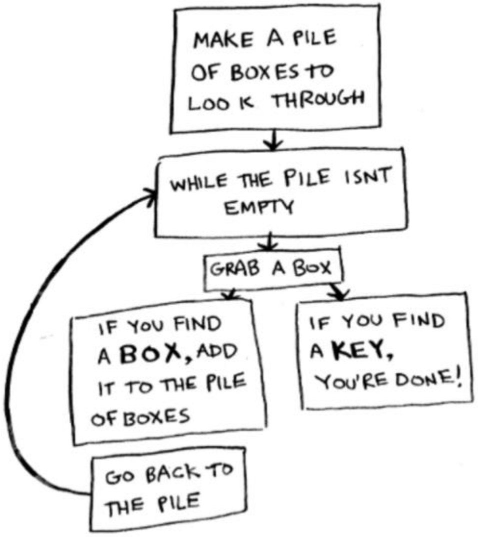
- Python concepts: recursion, loops, function frame model

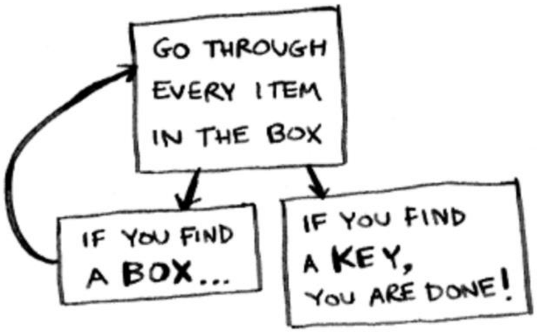
Concept Model:
Consider this real world task:
We are trying to find a key that is lost in a pile of boxes within a pile of boxes within a pile of boxes within…

In this case, we could describe the algorithm using an ***iterative*** approach. It would look something like the image on the left:



© Aditya


MAKE A PILE OF BOXES TO LOOK THROUGH
WHILE THE PILE ISNT EMPTY
GRAB A BOX
IF YOU FIND A BOX, ADD IT TO THE PILE OF BOXES
IF YOU FIND A KEY, YOU'RE DONE!
GO BACK TO THE PILE

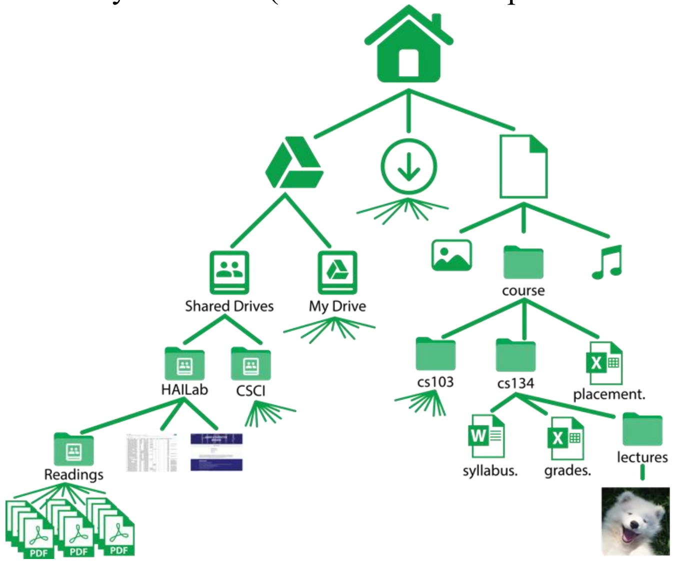

GO THROUGH EVERY ITEM IN THE BOX
IF YOU FIND A BOX...
IF YOU FIND A KEY, YOU ARE DONE!

Or, we could use a ***recursive*** approach which is described by the image on the right (above).

CM1. Which approach is simpler to describe? ______________________________

CM2. Which approach requires fewer keystrokes? ______________________________

While searching within boxes of boxes of boxes may seem like a stretched example, it is quite similar to finding a file of a puppy within a directory structure, and in fact, computer scientists do *typically* search file directory structures (and other tree-shaped structures) recursively!:



CM3. What other tasks fit this tree-shaped structure, and are therefore optimally solved with recursion?

Critical Thinking Questions:

1. Write a function to sum up a list of numbers, *iteratively*, such that calling
   `>>> sum_list_iterative([3, 4, 20, 12, 2, 20])` will return `61`:

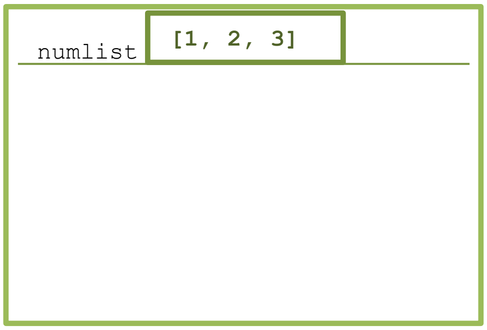
sum_list_iterative.py

```
def sum_list_iterative(numlist):
```

   a. Draw a function frame diagram for a call to this function, similar to what we did in the POGIL Activity on Function Frame Stack Model.
   (*Hint: The first function frame is begun for you below, do you need more?*)

```
>>> sum_list_iterative([1, 2, 3])
```

sum list iterative([1, 2, 3])



b. How many function frames are created? ______________

(*Hint: How many function calls to* `sum_list_iterative(..)` *does Python make?*)

2. Write a *recursive* version of the previous function to sum up a list of numbers, such that calling >>> `sum_list_iterative([3, 4, 20, 12, 2, 20])` will return `61`:

   a. What is our base case? ______________________________________

   b. What is our small step? ______________________________________

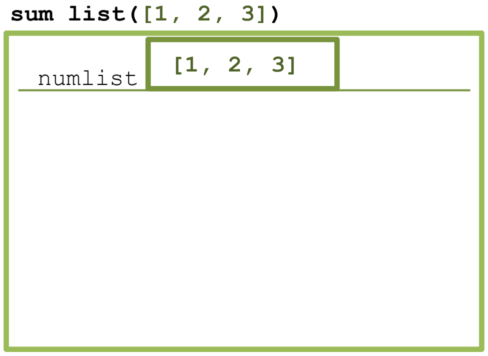
   c. How do we break the journey down? ______________________________

sumList.py

```
def sum_list(numlist):
```

d. Draw a function frame diagram for a call to this function, similar to what we did in the POGIL Activity on Function Frame Stack Model.
(*Hint: The first function frame is begun for you below, do you need more?*)

```
>>> sum_list([1, 2, 3])
```



e. How many function frames are created? ______________

(*Hint: How many function calls to* `sum_list (..)` *does Python make?*)

🔑 3. In the table below, specify if the statement on the left is a pro or con of iteration or recursion (or both):

Statement	Pro/ Con	Iteration/ Recursion
Can lead to syntactically simpler programs.		
Has a steeper learning curve.		
You will see code like this out in the real world.		
Is best for writing tree-type data structures.		
Creating new function frame stacks requires computational overhead.		
Is easier for novice computer scientists to understand.		
Is advanced computer science problem-solving approach.		

Application Questions: Use the Python Interpreter to check your work

1. a. Write a function, `file_found_iterative,` that takes a list of lists (`folder`) and a `target` item to look for in the list of lists. Use loops to find the target item. The function returns True if the item is found, False otherwise.

   b. Write a function similar to `file_found_iterative`, `file_found_recursive`, but instead uses a recursive approach.

   c. Which of these approaches may work for lists of lists of lists? Which may only work for a list of lists?

2. a. Implement two functions, `fibonacci_iterative(num)` and `fibonacci_recursive(num)`, one which finds the numth Fibonacci number using iteration (loops), and the other recursively. Recall that:

   $\text{Fibonacci}_{num} = \text{Fibonacci}_{num-1} + \text{Fibonacci}_{num-2}$

   $\text{Fibonacci}_0 = 0$ and $\text{Fibonacci}_1 = 1$

   Once you've done so, write code in `if __name__ == "__main__"` to time how long these two approaches take (you may need to use rather large values!). Recall that if we `from time import time`, we can use the `time()` function to retrieve number of milliseconds.

   b. Which one of these functions is faster?

   c. Write a third function, `fibonacci_recurs_fast(num)`, that uses your recursive approach, but stores (and retrieves) previously computed Fibonacci numbers (and their values) in a `dict`ionary. Then compare the runtimes of this function to the previous two.

   d. Which of your three functions is the fastest?