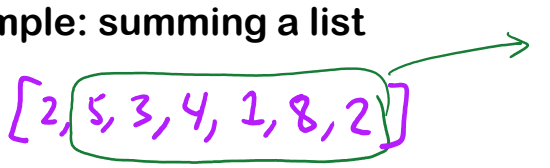
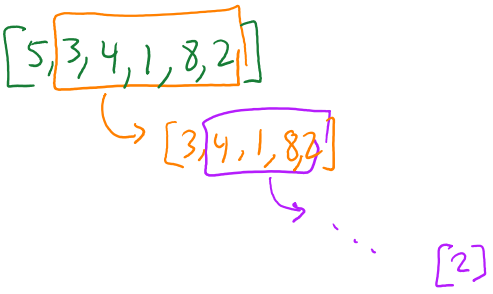
Computer Science 125
8 October 2020

1. Recursive function: a function that calls itself

Recursion is useful when you can take a big problem and break it into a smaller problem that is just like the big problem, but smaller.

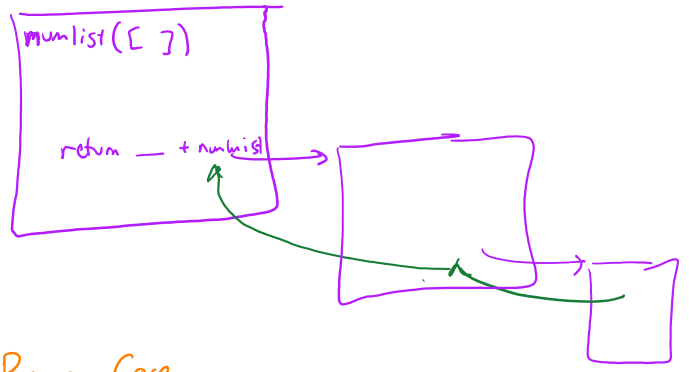
2. Example: summing a list





iterative:

```
def listsum(numList):
    theSum = 0
    for i in numList:
        theSum = theSum + i
    return theSum
```



recursive:

```
def listsum(numList):
    if len(numList) == 1:
        return numList[0]
    else:
        return numList[0] + listsum(numList[1:])
```

Base Case

first number + sum of the rest of the list

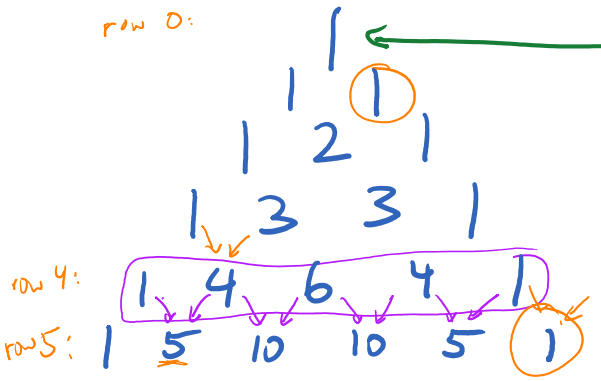
3. Three laws of recursion

1. A recursive algorithm must have a base case.

stops the recursion

2. A recursive algorithm must move toward the base case.
   big problem → smaller problem
3. A recursive algorithm must call itself, recursively.

4. Write a recursive function that prints the nth row of Pascal's triangle.



Base Case: row 0 is [1]

Move toward base case:

to compute row n:

first, compute row n-1 (recursive function call)

```
prevRow = row n-1
currRow = [1]
for i in range (n-1)
  currRow.append (prevRow[i] + prevRow[i+1])
```

```
# function to return row n
def pascal(n):
    # base case
    if n == 0:
        return [1]
    # move toward base case
    else:
        prevRow = pascal(n-1)
        print(prevRow)
        currRow = [1]
        for i in range(n-1):
            currRow.append(prevRow[i] + prevRow[i+1])
        currRow.append(1)
        return currRow

print(pascal(5))
```

PRACTICE WITH RECURSION – SOLUTIONS

Working with a partner/group, use the following steps to solve each of the following problems.

(a) Plan your code on the white board (either on the classroom wall or on Zoom). Write out your entire program. Think about what errors might occur and how to fix them.

(b) Plan multiple test cases. What input will you send to your function? For each input, what value should be returned?

(c) *Only after you have completed steps (a) and (b) should you type your code in Python.*

(d) After you have typed your code, run your test cases. Does your code work? If not, how can you fix it?

1. **Factorials**: Recall that the factorial of an integer n, denoted $n!$, is the product of all positive integers less than or equal to n. For example, the factorial of 5 is

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

Write a recursive function to compute the factorial of a number.

First, think about how the problem of computing the factorial of n can be reduced to the problem of computing the factorial of a smaller number — this gives you a recursive strategy. Then, identify the base case that will cause the recursion to stop.

```
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)
```

2. **First uppercase letter**: Write a recursive function that finds the first uppercase letter in a string.

First, think about how the problem of finding the first uppercase letter in a string can be reduced to a smaller problem. Then identify the base case that will cause the recursion to stop.

```
import string

def firstUppercase(text):
   if len(text) == 0:
         return None
   if text[0] in string.ascii_uppercase:
         return text[0]
   return firstUppercase(text[1:])
```

3. **Recamán's sequence**: Recamán's sequence is a sequence of integers $a_0, a_1, a_2, \ldots$ defined recursively as follows. First, $a_0 = 0$. Then, for $n > 0$:

$$a_n = \begin{cases} a_{n-1} - n & \text{if } a_{n-1} - n \text{ is positive and not already in the sequence,} \\ a_{n-1} + n & \text{otherwise.} \end{cases}$$

The first few terms of the sequence are 0, 1, 3, 6, 2, 7, 13, 20, ….

Write a recursive function `recaman(n)` that computes the first n terms of Recamán's sequence.

```
def recaman(n):
    if n == 0:
        return [0]

    # else: get terms a_0, ..., a_(n-1)
    seq = recaman(n-1)

    # now compute term a_n
    t = seq[-1] - n
    if t > 0 and t not in seq:
        seq.append(t)
    else:
        seq.append(seq[-1] + n)

    # done!
    return seq
```

4. **Bonus – Tower of Hanoi**: Write a recursive function that prints the moves necessary to solve the Tower of Hanoi puzzle for n disks. See en.wikipedia.org/wiki/Tower_of_Hanoi for an explanation of the puzzle.

```
#  Let disk 1 be the smallest, disk 2 be the second-smallest, etc.
#  Let the posts be labeled 1, 2, 3.
#  Disks start on post 1, and end on post 3.

#recursive function to PRINT the steps required to move the n-smallest disks
#  from the "start" post to the "end" post
def towerOfHanoi(n, start, end):
    if n == 1:
        print("Move disk 1 from post", start, "to post", end)
    else:
        temp = 6 - start - end
        towerOfHanoi(n-1, start, temp)
        print("Move disk", n, "from post", start, "to post", end)
        towerOfHanoi(n-1, temp, end)


#now start the program
num = int(input("How many disks? "))
towerOfHanoi(num, 1, 3)
print("DONE")
```