

# Linear Recursion and multiple recursion

- A recursion is **linear** when each function call makes at most one recursive call.
- A recursion is **multiple** when each function call makes more than one recursive call.

```
def factorial(n):  
    if n == 0 or n == 1:    # Base case  
        return 1  
    return n * factorial(n - 1)  # One recursive call
```

```
def fibonacci(n):  
    if n == 0:  # Base case  
        return 0  
    elif n == 1:  
        return 1  
    return fibonacci(n - 1) + fibonacci(n - 2)  # Two recursive calls
```

# Sum of a list

```
def linearSum(L, n):  
    if n==0:  
        return 0  
    else:  
        return linearSum(L, n-1)+L[n-1]  
  
def main():  
    L = [1, 2, 3, 4, 5, 9, 100, 46, 7]  
    print('The sum is:', linearSum(L, len(L)))
```

The base case is the condition that stops the recursion. If there are 0 elements (empty list) to sum ( $n == 0$ ), the sum is 0 and the function returns immediately without making another recursive call.

The recursive case reduces the problem into a smaller instance of itself, eventually reaching the base case. The function calls itself with one fewer element ( $n - 1$ ) and adds the last element of the list ( $L[n - 1]$ ) to the sum returned by the smaller subproblem.

## Sum of a list

```
def linearSum(L, n):
    if n == 0:
        return 0
    else:
        return L[0] + linearSum(L[1:], n - 1)

def main():
    L = [1, 2, 3, 4, 5, 9, 100, 46, 7]
    print('The sum is:', linearSum(L, len(L)))

main()
```

# Binary sum

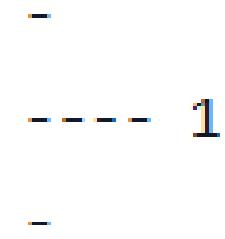
Calculate the sum of a list of numbers. Inside `binarySum()` two recursive calls should be made

```
def binarySum(L, start, stop):
    if start>=stop:
        return 0
    elif start==stop - 1:
        return L[start]
    else:
        mid = (start+stop)//2
        return binarySum(L, start, mid)+binarySum(L, mid, stop)

def main():
    L = [1, 2, 3, 4, 5, 6, 7]
    print(binarySum(L, 0, len(L)))
```

# Drawing an English ruler

If `draw_ruler(2, 4)` is called:

<pre>def draw_line(tickLen, tickLabel=''):    Draws <b>one tick</b> on the ruler     line = '-' * tickLen     if tickLabel:         line += ' ' + tickLabel     print(line)</pre>	
<pre>def draw_interval(centerLen):     if centerLen &gt; 0:         draw_interval(centerLen - 1)         draw_line(centerLen)         draw_interval(centerLen - 1)</pre>	<p>Draws the <b>minor ticks</b> between the major ticks using recursion. The <b>base case</b> is when <code>centerLen == 0</code> → Do nothing.</p> 
<pre>def draw_ruler(numInch, majorLen):     draw_line(majorLen, '0')      for j in range(1, 1+numInch):         draw_interval(majorLen - 1)         draw_line(majorLen, str(j))</pre>	<p>Draws the <b>whole ruler</b> given:</p> <ul style="list-style-type: none"><li>• <code>numInch</code>: Number of inches.</li><li>• <code>majorLen</code>: Length of the major tick marks.</li></ul> <p>Each iteration draws one major inch</p> 