**What is Recursion?**

- Calling the same method from within itself. Every recursive method must have a finishing condition that will stop the ever-expanding recursion.

**Why do we need Recursion?**

- Although solving a same problem seems easier with loops than recursion, recursion allows us to solve some certain problems that cannot be solved by a loop. For example, a singly linked list can not be reverse printed using a loop. But using recursion, it is a walk in the park [2.2].

Now let us compare and contrast between iterative and recursive approaches by solving same problems using both techniques. Suppose we have a singly linked list like this which will be used in problems 1, 2, 4 and 7:

## Iteration vs Recursion

### 1.1 Forward Printing a singly linkled list values using iteration

```
def iterativePrintList(head):
  while(head!=None):
    print(head.val)
    head=head.next

iterativePrintList(head)
```

```
5
10
15
-5
12
```

### 1.2 ForwardPrinting a singly linkled list values using recursion

```
[68]
def recursiveForwardPrintList(head):
  if head!=None:
    print(head.val)
    recursiveForwardPrintList(head.next)

recursiveForwardPrintList(head)
```

```
5
10
15
-5
12
```

### 2.1 Printing a singly linkled list values in reverse using iteration?

**Not Possible!**

### 2.2 Printing a singly linkled list values in reverse using recursion

```
[58]
def recursiveReversePrintList(head):
  if head!=None:
    recursiveReversePrintList(head.next)
    print(head.val)

recursiveReversePrintList(head)
```

```
12
-5
15
10
5
```

### 3.1 Calculation of exponenentiation

```
[61]
def exp(base,power):
  return base**power

print(exp(5,3))
```

```
125
```

### 3.2 Calculation of exponenentiation using iteration

```
[63]
def exp_iteration(base,power):
  num=base
  for i in range(power-1):
    num=num*base
  return num

print(exp_iteration(5,3))
```

```
125
```

### 3.3 Calculation of exponenentiation using recursion

```
[66]
def exp_recursion(base,power):
  if (power==0):
    return 1
  else:
    return base*exp_recursion(base,power-1)

print(exp_recursion(5,3))
```

```
125
```

### 4.1 Length of a linked list using iterations

```
[86]  1 def iterativeLinkedListLength(head):
      2   length=0
      3   while(head!=None):
      4     length+=1
      5     head=head.next
      6   return length
      7
      8 print(iterativeLinkedListLength(head))
```

```
5
```

### 4.2 Length of a linked list using recursion

```
[88]  1 def recursiveLinkedListLength(head):
      2   if(head==None):
      3     return 0
      4   else:
      5     return 1+recursiveLinkedListLength(head.next)
      6
      7 print(recursiveLinkedListLength(head))
```

```
5
```

### 5.1 Factorial Calculation using iteration

```
1 def iterative_factorial(num):
2   fac=1
3   for i in range(num,0,-1):
4     fac=fac*i
5   return fac
6
7 print(iterative_factorial(5))
```

```
120
```

### 5.2 Factorial Calculation using recursion

```
[73]  1 def recursive_factorial(num):
      2   if (num==0):
      3     return 1
      4   else:
      5     return num*recursive_factorial(num-1)
      6
      7 print(recursive_factorial(5))
```

```
120
```

### 6.1 First nth Fibonacci number printing using iteration

```
1 def iterative_fibonacci(num):
2   p= 0
3   q= 1
4   print(p)
5   print(q)
6   for i in range(num-2):
7     r=p+q
8     print(r)
9     p=q
10    q=r
11
12 iterative_fibonacci(8)
```

```
0
1
1
2
3
5
8
13
```

### 6.2 First nth Fibonacci number printing using recursion

```
[82]  1 def recursive_fibonacci(num):
      2   if (num==0 or num==1):
      3     return num
      4   else:
      5     return recursive_fibonacci(num-1)+recursive_fibonacci(num-2)
      6
      7 for i in range(8):
      8   print(recursive_fibonacci(i))
      9
     10 """This recursive_fibonacci method can find one fibonacci number
     11 at a time. That is why we are using a loop to call it repeatedly"""
```

```
0
1
1
2
3
5
8
13
```

### 7.1 Print the element of the last node using iteration [HW]

```
1 def last_element_iterative(head):
2   #Your Code Here
3
4
5
6
7
8
9
10 last_element_iterative(head)
```

```
12
```

### 7.2 Print the element of the last node using recursion [HW]

```
[91]  1 def last_element_recursive(head):
      2   #Your Code Here
      3
      4
      5
      6
      7
      8
      9
     10 last_element_recursive(head)
```

```
12
```