Recursion (Basic)

November 23, 2022

1 Recursion

Recursive function is a function which calls itself until some condition is not fulfilled with some input

The condition on basis of which function stops calling itself is called **Base Case**

Recursion is based on the mathematical concept known as **PMI**

1. *PMI* is method has 3 steps:
    1. prove for trivial input
    2. Assume for n
    3. prove for n+1
2. *Recusrion* is also has three:
    1. Trivial Case or Base Case
    2. Assume Answer Smaller Solution
    3. Calculate the Solution For Larger Input

1.1 Examples

1.1.1 1. Factorial

[1]:
```
def factorial(num):
    if num==0 : # Base Case
        return 1
    smaller_ans = factorial(num - 1) # Smaller Answer
    return num * smaller_ans # Calculation For The Larger Answer
```

[2]:
```
# n = int(input())
n = 4
print(factorial(4))
print(factorial(5))
print(factorial(6))
print(factorial(7))
```

```
24
120
720
5040
```

1.1.2 2. Sum of the n natural numbers

[3]:
```
def sum_of_n(num):
    if num==0:
        return 0
    smaller_ans = sum_of_n(num - 1)
    return num + smaller_ans
```

[4]:
```
n = 10  # 10+9+8 ... +2+1 = 55
print(sum_of_n(10))
print(sum_of_n(11))
print(sum_of_n(12))
print(sum_of_n(13))
print(sum_of_n(14))
```

```
55
66
78
91
105
```

1.1.3 3. Print 1 to n

[5]:
```
def print_linear(num):
    if num==0:
        return
    print_linear(num-1)
    print(num)
    return
```

[6]:
```
n = 15
print_linear(n)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

1.1.4 4. Print n to 1

[7]:
```
def print_rev_linear(num):
    if num ==0:
        return
    print(num)
    print_rev_linear(num - 1)
    return
```

[8]:
```
n = 10
print_rev_linear(n)
```

```
10
9
8
7
6
5
4
3
2
1
```

1.1.5 5. Fibbonacci Series

1,1,2,3,5,8,……..

[9]:
```
def fibo(num):
    if num==0 or num==1:
        return 1
    return fibo(num-1)+fibo(num-2)
```

[10]:
```
n = 5
print(fibo(n))
# 0 1 2 3 4 5
# 1 1 2 3 5 8
```

```
8
```

1.1.6 6. Is A List Sorted

[11]:
```
def is_sorted_list(nums,i=0):
    if i>=len(nums)-1:
        return True
    elif nums[i]>nums[i+1]:
        return False
    return is_sorted_list(nums,i+1)
```

[12]:
```
nums_list = [1,2,3,4,6]
nums_list2 = [1,2,6,5,4]
print(is_sorted_list(nums_list))
print(is_sorted_list(nums_list2))
```

```
True
False
```

1.1.7 7. Sum of Array

[13]:
```
def sum_of_array(nums,i=0):
    if i == len(nums):
        return 0
    return nums[i] + sum_of_array(nums,i+1)
```

[14]:
```
nums_list = [1,2,3,4,10,6,7,8]
print(sum_of_array(nums_list))
```

```
41
```

1.1.8 8. First In the Array

[15]:
```
# Basic function
def in_array(nums,target,i=0):
    if i == len(nums):
        return -1
    elif nums[i] == target:
        return i
    return in_array(nums,target,i+1)
```

[16]:
```
nums_list = [1,2,20,4,5,20]
print(in_array(nums_list,20))
print(in_array(nums_list,10))
```

```
2
-1
```

1.1.9 9.Last in the Array

[17]:
```
# Basic function
def last_array(nums,x,i=0):
    index = len(nums)-i-1
    if i == len(nums):
        return -1
    elif nums[index]==x:
        return index
    return last_array(nums,x,i+1)
```

[18]:
```
# from back
def last_index_better(nums,x,li):
    if li == 0 :
        return -1
    elif nums[li-1]==x:
        return li-1
    return last_index_better(nums,x,li-1)
```

[19]:
```
# form fornt
def last_index(nums,x,si=0):
    if si == len(nums):
        return -1
    smaller_ans = last_index(nums,x,si+1)
    if smaller_ans != -1:
        return smaller_ans
    elif nums[si] == x:
        return si
    return -1
```

[20]:
```
nums_list = [1,2,20,4,5,20]
print(last_array(nums_list,20))
print(last_index(nums_list,20))
print(last_index_better(nums_list,20,len(nums_list)))
print(last_array(nums_list,10))
print(last_index(nums_list,10))
print(last_index_better(nums_list,10,len(nums_list)))
```

```
5
5
5
-1
-1
-1
```