Python Programming

Recursive Functions 1

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Problem and subproblems

- Sometimes we can decompose a problem into a set of sub-problems
- E.g. Print all prime numbers that are palindrome and < 1000000
- We have 2 sub-problems
  - def is_prime(n):
  - def is_palindrome(n):
- Now we iterate from 1 to 1000000
  - If the number satisfies the 2 conditions: count it
- What if the sub-problem is the same type as the problem? Recursion!

Recall the factorial

- factorial(6) = **1 * 2 * 3 * 4 * 5** * 6
- factorial(5) = **1 * 2 * 3 * 4 * 5**
- factorial(4) = 1 * 2 * 3 * 4
- factorial(3) = 1 * 2 * 3
- factorial(2) = 1 * 2
- factorial(1) = 1
- Think for a few minutes:
  - What is relation between factorial(6) and factorial(5)?
  - Can you know factorial(6) if you know factorial(5)?

Factorial

```
3   def factorial(n):
4       res = 1
5
6       for i in range(2, n+1):
7           res *= i
8
9       return res
10
11
12  if __name__ == '__main__':
13      print(factorial(3))      # 1 * 2 * 3
14      print(factorial(4))      # 1 * 2 * 3 * 4
15
16      print(factorial(5))      # 1 * 2 * 3 * 4 * 5
17                               # factorial(4)  * 5 = 120
18
19      print(factorial(6))      # 1 * 2 * 3 * 4 * 5 * 6 = 720
20                               # factorial(5)      * 6 = 720
21                               # factorial(4)  * 5 * 6 = 720
22                               # factorial(3)*4* 5 * 6 = 720
23
```

Factorial: Problem and subproblem

- Let's say we want to solve factorial(6)
  - This is our problem
  - We can solve it directly with 1*2*3*4*5*6
- Another thinking is: can we think of it is
  - What is factorial(5)? A simpler subproblem
  - Would it help if u know its answer? Yes: 6 * factorial(5) = factorial (6)
  - Same logic for factorial(5). It is 5 * factorial(4).
- Going forever in smaller sub-problems? No
  - There must be a case where no more subproblems. We call it the **base case**
  - Factorial 1 = 1

Factorial: Problem and subproblem

```
2   def factorial1():
3       return 1    #    base case. No subproblems
4
5   def factorial2():
6       return factorial1() * 2
7
8
9   def factorial3():
10      return factorial2() * 3
11
12
13  def factorial4():
14      return factorial3() * 4
15
16
17  def factorial5():
18      return factorial4() * 5
19
20
21  def factorial6():
22      return factorial5() * 6
23
24
25  print(factorial6())
26
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."