Computing Laboratory

Basics of Python - V

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

August, 2023

Basics of recursion

A recursive function is a function that calls itself.

- The task should be decomposable into sub-tasks that are smaller, but otherwise identical in structure to the original problem.
- The simplest sub-tasks (called the base case) should be (easily) solvable directly, i.e., without decomposing it into similar sub-problems.

Recursive versus iterative implementations

```
def factorial(n): # Iterative version
    prod = 1
    if n < 0:
        return -1 # Error condition
    while n:
        prod *= n
        n = n - 1
    return prod
```

Recursive versus iterative implementations

Let's execute the following complete program:

```
def factorial(n): # Iterative version
    prod = 1
    if n < 0:
        return -1 # Error condition
    while n:
        print('Value of n ( @', hex(id(n)), ') = ', n)
        prod *= n
        n = n - 1
    return prod
factorial(5)
```

Recursive versus iterative implementations

Output:

```
Value of n ( @ 0x7fb0a6c10170 ) =  5
Value of n ( @ 0x7fb0a6c10150 ) =  4
Value of n ( @ 0x7fb0a6c10130 ) =  3
Value of n ( @ 0x7fb0a6c10110 ) =  2
Value of n ( @ 0x7fb0a6c100f0 ) =  1
```

Recursive versus iterative implementations

```
def factorial(n):                          # Recursive version
    if n < 0:
        return -1                          # Error condition
    if n == 0:
        return 1                           # Base case
    else:
        return n * factorial(n-1) # Recursive call
```

Recursive versus iterative implementations

Let's execute the following complete program:

```
def factorial(n):                          # Recursive version
    if n < 0:
        return -1                          # Error condition
    print('Value of n ( @', hex(id(n)), ') = ', n)
    if n == 0 or n == 1:
        return 1                           # Base case
    else:
        print ('Entering into factorial(', n-1, ')');
        return n * factorial(n-1) # Recursive call
factorial(5)
```

Recursive versus iterative implementations

Output:

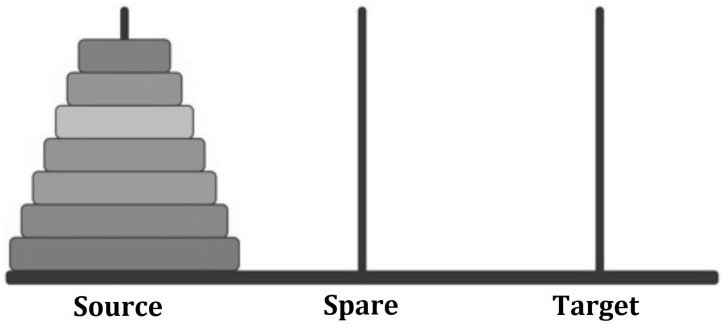
```
Value of n ( @ 0x7f0f35500170 ) =  5
Entering into factorial( 4 )
Value of n ( @ 0x7f0f35500150 ) =  4
Entering into factorial( 3 )
Value of n ( @ 0x7f0f35500130 ) =  3
Entering into factorial( 2 )
Value of n ( @ 0x7f0f35500110 ) =  2
Entering into factorial( 1 )
Value of n ( @ 0x7f0f355000f0 ) =  1
```

Tower of Hanoi

Solve the Tower of Hanoi problem (as shown in the picture below) where the objective is to move an entire stack of disks from the source (leftmost) tower to the target (rightmost) tower using a spare (middle one) tower, abiding by the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one tower and placing it on top of another tower or on an empty tower.
- No disk may be placed on top of a disk that is smaller than it.

Given the number of disks n as user input, output the total count of moves and print the individual moves.



Recursive structure of Tower of Hanoi

1. Move $n-1$ disks from the source to the spare tower, by the same general solving procedure. Rules are not violated, by assumption. This leaves the disk n as a top disk on the source tower.
2. Move the disk n from the source to the target tower.
3. Move the $n-1$ disks that we have just placed on the spare tower, from the spare to the target tower by the same general solving procedure, so they are placed on top of the disk n without violating the rules.
4. The base case is to move no disks (in steps 1 and 3).

Tower of Hanoi – Writing the code

```
def ToI(n, source, target, spare):
    if n == 1:
        print('Move disk 1 from', source, 'to', target)
        return
    ToI(n-1, source, spare, target)
    print('Move disk', n, 'from', source, 'to', target)
    ToI(n-1, spare, target, source)
n = int(input('Enter the number of disks: '))
ToI(n, 'Source', 'Spare', 'Target')
```

Command line arguments

```
import sys as s
print('argc = ', len(s.argv)) # Multiple arguments viable
for i, a in enumerate(s.argv):
    print(str(i) + '-th argument:', a)
```

Output:

```
argc =  1
0-th argument: filename.py
...
```

Problems – Day 6

1. Count the total number of 9's appearing in all the non-negative integers no more than an integer given as user input.
2. Consider a 2-D matrix of size $2^m \times 2^m$. The entries of the matrix are, in row-major order, $1, 2, 3, \ldots, 2^{2m}$. Print the entries of the matrix in Z-curve order (as shown in the picture below).