Python 3 Recursion

Eddie Guo

October 2019

1 Introduction to Recursion

1.1 Topics Covered

(i) What is recursion?

(ii) Conditions for termination

(iii) Stack frames

1.2 Recursion

- Recursion occurs when fn (or method) calls itself, either directly or indirectly
- If problem can be resolved by solving simple part of it & resolving rest of big problem in same way, can write a fn that solves simple part of problem then calls itself to resolve rest of problem
- For recursion to terminate, 2 conditions must be met:
  - Must be 1/more simple cases that do NOT make recursive calls (**base case**)
  - Recursive call must somehow be simpler than original call (change state to move towards base base)

```
def factorial(n):
    '''Return factorial of number.
    '''
    if (n == 0 or n == 1): # base case
        answer = 1
    else:
        answer = n * factorial(n-1)
    return answer
```

1.3 Fn Activations & Frames

- When fn invoked, frame/stack frame corresponding to that fn created & pushed onto the stack
- Frame stores all local vars assoc w/ that fn call
- Frame created when fn invoked & destroyed when fn finishes
- If fn invoked again, new frame is created for it w/ all its local vars

1.4 Multiple Activations of Fn

- When we invoke recursive fn, fn becomes active
- B4 it's finished, it makes recursive call to same fn
- This means that when recursion used, there's >1 copy of same fn active at once
- Each active fn has its own frame which contains indep copies of its local vars
- These frames stored on the call stack