

COL1000: Introduction to Programming Functions

Subodh Sharma | Lec 23 | Oct 10



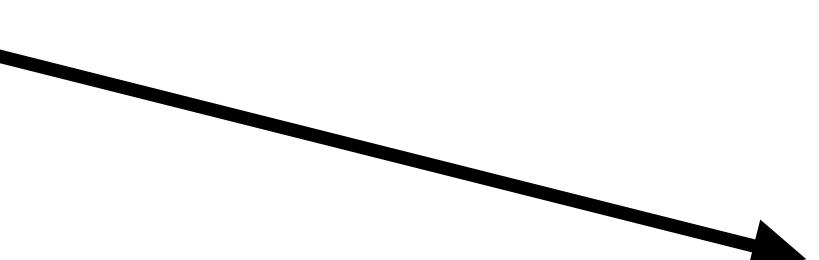
Non-linear Recursion: Tree Recursion

- Fibonacci
 - $\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$
 - **Recursive call stack grows in the form of a binary tree**
 - **Exponential Time** q
- Count the number of paths in a grid $m \times n$, moving right and down
 - $P(m,n) = 1$ if $m = 1$ or $n = 1$ else $P(m-1, n) + P(m, n-1)$

Tail Recursion

- Special form of recursion that is the **very last computation** performed in the function

- ```
def factorial(n:int) -> int:
 if n == 0: return 1
 elif n < 0:
 raise Error("Not defined on neg values")
 else: return n * factorial(n-1)
```



Not Tail Recursive

# Tail Recursion: Why it matters?

```
def fact(n:int) -> int:
 def fact_tr(n, res):
 if n == 0: return res
 return fact_TR(n-1, n*res)

 return fact(n, 1)
```

- Memory efficient
- Reuses the same stack frame
- Therefore no stack overflow

# Tail Recursion: Why it matters?

```
def fact(n:int) -> int:
 def fact_tr(n, res):
 if n == 0: return res
 return fact_TR(n-1, n*res)
 return fact_TR(n, 1)
```

factorial(5)  
→ factorial\_helper(5, 1)  
→ factorial\_helper(4, 5 \* 1 = 5)  
→ factorial\_helper(3, 4 \* 5 = 20)  
→ factorial\_helper(2, 3 \* 20 = 60)  
→ factorial\_helper(1, 2 \* 60 = 120)  
→ factorial\_helper(0, 1 \* 120 = 120)  
→ return 120

**Python is does not implement TAIL CALL Optimisation**

**Yet, you will see some benefits! (Example – Fibonacci)**

# (Tail) Recursion and Algorithmic Refinement

- Compute  $a^n$ , where  $a \in \mathbb{I}$  and  $b \geq 0$ 
  - **Give simple recursive function**
  - **Can you optimise it algorithmically?**
  - **Can you create tail-recursive version of algo-optimised version?**

```
def recPow(a,n):
 if n = 0:
 return 1
 else:
 return a*recPow(a,n-1)
```

# (Tail) Recursion and Algorithmic Refinement

- Compute  $a^n$ , where  $a \in \mathbb{I}$  and  $b \geq 0$

- Give simple recursive function

```
• def recPow(a, n):
 if n = 0:
 return 1
 else:
 return a*recPow(a, n-1)
```

Base case

Induction Step

# (Tail) Recursion and Algorithmic Refinement

- Compute  $a^n$ , where  $a \in \mathbb{I}$  and  $b \geq 0$ 
  - **Can you optimise it algorithmically?**
  - **Exploit the mathematical identity**
    - $a^{2k} = (a^k)^2$  and  $a^{2k+1} = a \cdot a^{2k}$
    - **This converges in  $O(\log n)$  steps**
      - As opposed to  $O(n)$  previously

```
def recFastPow(a, n):
 if n == 0:
 return 1
 if n % 2 == 0:
 h = recFastPow(a, n // 2)
 return h * h
 else:
 h = recFastPow(a, n // 2)
 return a * h * h
```

# (Tail) Recursion and Algorithmic Refinement

- Compute  $a^n$ , where  $a \in \mathbb{I}$  and  $b \geq 0$ 
  - **Can you create tail-recursive version of algo-optimised version? Exploit the mathematical identity**
  - **Use an accumulator variable to store intermediate results**

```
def recFastPow_TL(a, n, accum=1):
 if n == 0:
 return acc
 if n % 2 == 1:
 return recFastPow_TL(a * a, n // 2, acc * a)
 else:
 return fast_pow_tr(a * a, n // 2, acc)
```

# Functions: Exceptions

- Exceptions in functions **denote errors or unexpected conditions**
- There are two ways of handling exceptions in the code:
  - Explicitly **raising the errors with an appropriated exception message**
  - **Catching** them to handle them gracefully
  - Let us see examples of each!