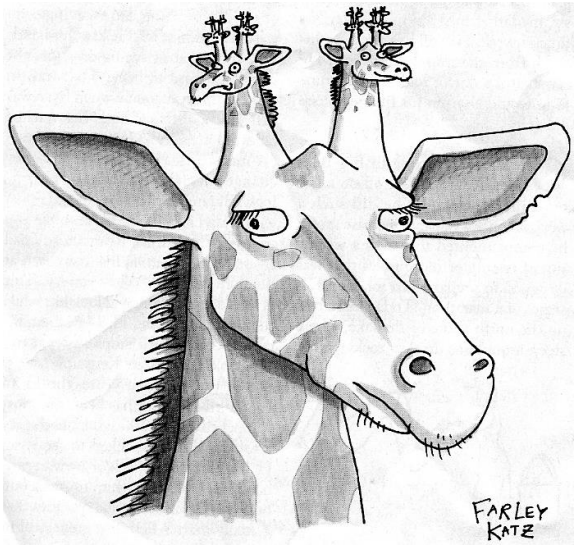
CSCA48 Winter 2018 Week 8:Recursion

Marzieh Ahmadzadeh, Nick Cheng

University of Toronto Scarborough

Recursion

- You can create a chain of function call
- For each call, all the states of the current function is pushed into a **stack**.
- When return statement is reached, all the states of this function are popped from the stack.



New Yorker Magazine, August 11, 2008

- Recursion: When a function calls itself.

Importance of learning Recursion

- It’s a new way of algorithm design
- It’s a powerful programming paradigm that can be used for many computations that are naturally recursive
  - Most of sorting algorithm
  - Tree traversal
  - Some mathematical computations (Factorial, GCD, etc.)

Recursion: Design

- Problems that are solved by recursion, can be solved by iterative methods too.
  - The iterative solution will use loop statements one way or another.

- Our focus in next two weeks is on:
  1. Design a recursive algorithm for a given problem
  2. Trace a given recursive algorithm and show the flow of execution and memory status.

Recursion: Design

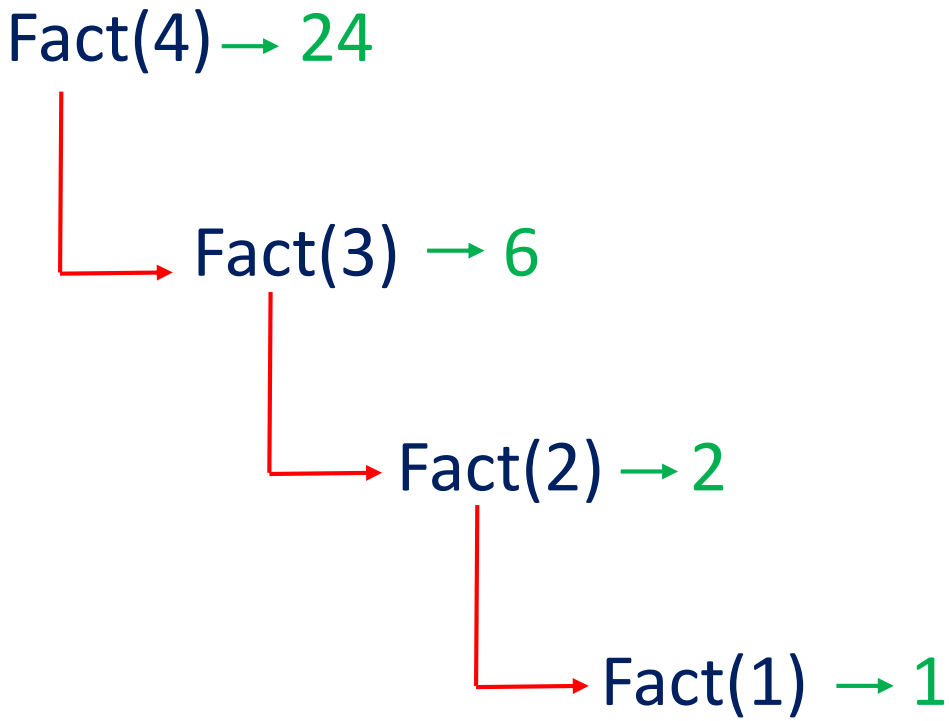
1. **Base Case:** What is the result of executing the function when the input
   - either has the smallest possible value
   - Or is in its simplest form

2. **Recursive Case**:
   - Either solve the problem for input `n`, using the solution for input `n-1`.
   - Or use divide and conquer strategy

- Every recursive function must have a base case, otherwise you create an infinite loop.
- A recursive case must get to the base case so that the function call ultimately stops at some point.
- It is possible to have more than one base case.

How to trace a recursive algorithm?

- We have agreed on a convention for this course that you should follow.
- Start with the original function call.
- Use an arrow, pointing to right, when you get to a recursive call
- Indentation is required for inner recursive call.
- When something is returned, write the return value to the right of the function call.



More design issues

- Sometimes, it is not an easy job, if possible at all, to solve a problem using one single recursive function. In this case,
  - either use a helper function
  - or change the number of parameters to help you solve the problem
    - This is not the case for the exercises, assignments or exams where the function signature is given

- Sometimes, you need to have more than one base cases.

Fibonacci Numbers

- In 1225, Fibonacci solved the following problem and on that time Fibonachi numbers were born.
  - Start with a single pair of rabbits.
  - Every month each productive pair bears a new pair
  - This new pair becomes productive when they are 1 month old.
  - Rabbits never die.
  - How many rabbits will be there after n months?

Fibonacci Analysis

- Let n_k denotes the number of calls for fib(k)

```
n0=1
n1=1
n2=n1+n0+1=1+1+1=3
n3=n2+n1+1=3+1+1=5
n4=n3+n2+1=5+3+1=9
n5=n4+n3+1=9+5+1=15
n6=n5+n4+1=15+9+1=25
n7=n6+n5+1=25+15 +1=41
n8= n7+n6+1=41+25+1=67
```

```
def fib(n):
    if (n == 0):
        result = 0
    elif (n == 1):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
    return result
```

- Look at every other value of n_k. It at least gets doubled! This means $n_k > 2^{k/2}$. Extremely inefficient!
  - executing fib(50), you should expect at least 33,554,431 function call!

Recursion Types

- **Binary Recursion**: When there are two recursive calls in recursive case of the function.
- **Linear Recursion**: When there are one recursive call in recursive case of the function.

- If a problem can be solved with a `linear` solution, you should avoid a `binary` solution for efficiency purposes.