GoCode

We learn by doing, by falling down, and by picking ourselves back up

HTTP://GOCODENOW.COM

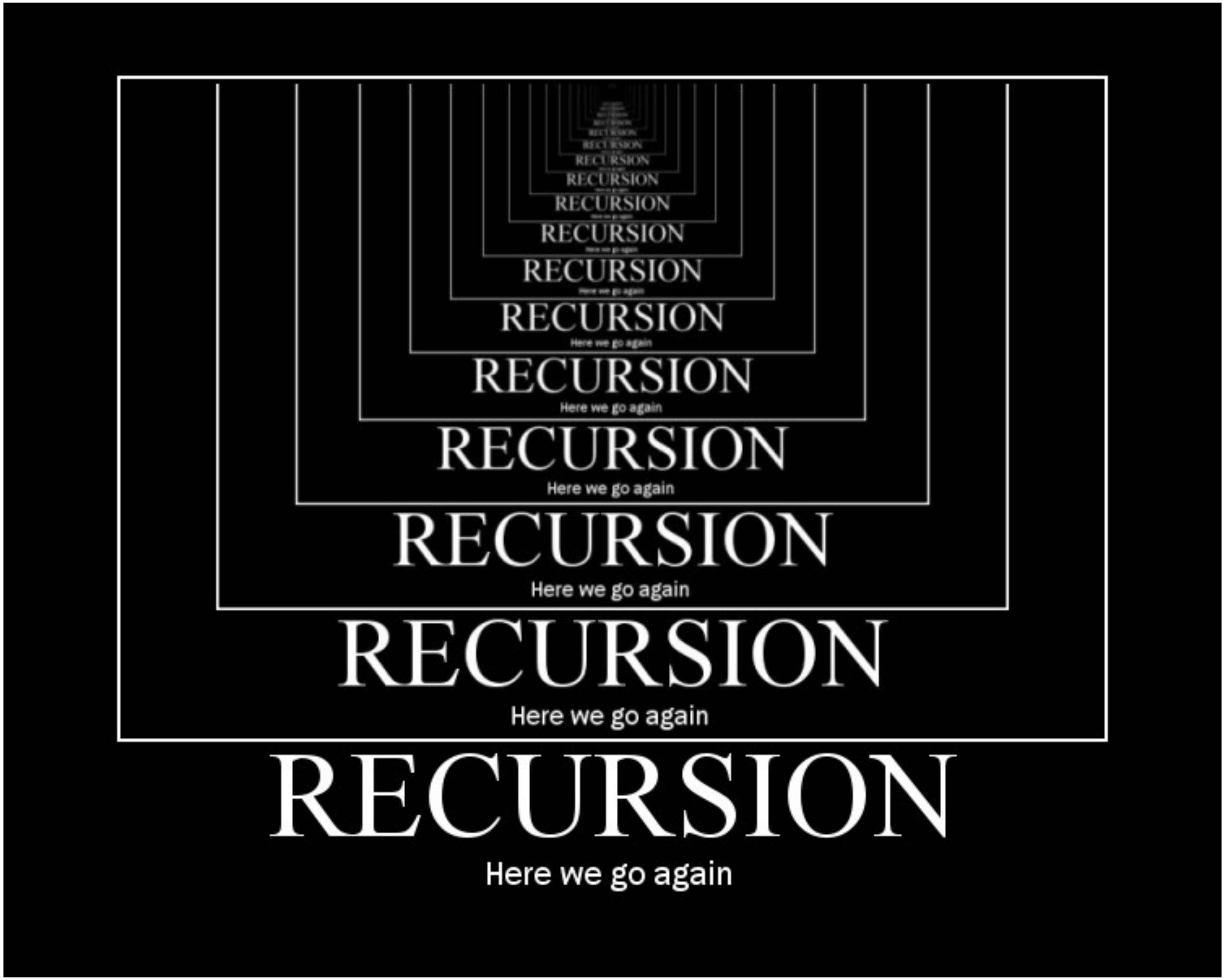
Recursion

1. What is it?

2. Five Common Problems

3. How to Approach

Recursion


RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
RECURSION
RECURSION
RECURSION
RECURSION

Five classes of recursion

1. **Linear Recursive (Factorial)**

- Functions with one recursive call (most common)

2. **Binary Recursive (Fibonacci)**

- Functions with two recursive calls

3. Tail Recursive (Greatest Common Denom.)

- Returns recursive call

4. Mutual Recursive (Is Odd/Is Even)

- Functions calling each other

5. Exponential Recursive (All permutations)

- If there were *n* elements, there would be *O*(a^n) calls

Factorial Iterative Approach

$$1! = 1 = 1$$

$$2! = 2 \times 1 = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

```
def iterative_factorial(n):
    result = 1
    for i in range(2,n+1):
        result *= i
    return result
```

Factorial Recursive Approach (Linear)

$$1! = 1 = 1$$

$$2! = 2 \times 1 = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

```
def factorial(n):
   if n == 1:
      return 1
   else:
      return n * factorial(n-1)
```

TRACING FACTORIAL(3)

Factorial(3)

↓

3* Factorial(2)

↓

2* Factorial(1)

↓

1* Factorial(0)

↓

1

∴ Factorial(3) = 3* Factorial(2)

= 3* (2* Factorial(1))

= 3* 2* (1* Factorial(0))

= 3* 2* 1*1

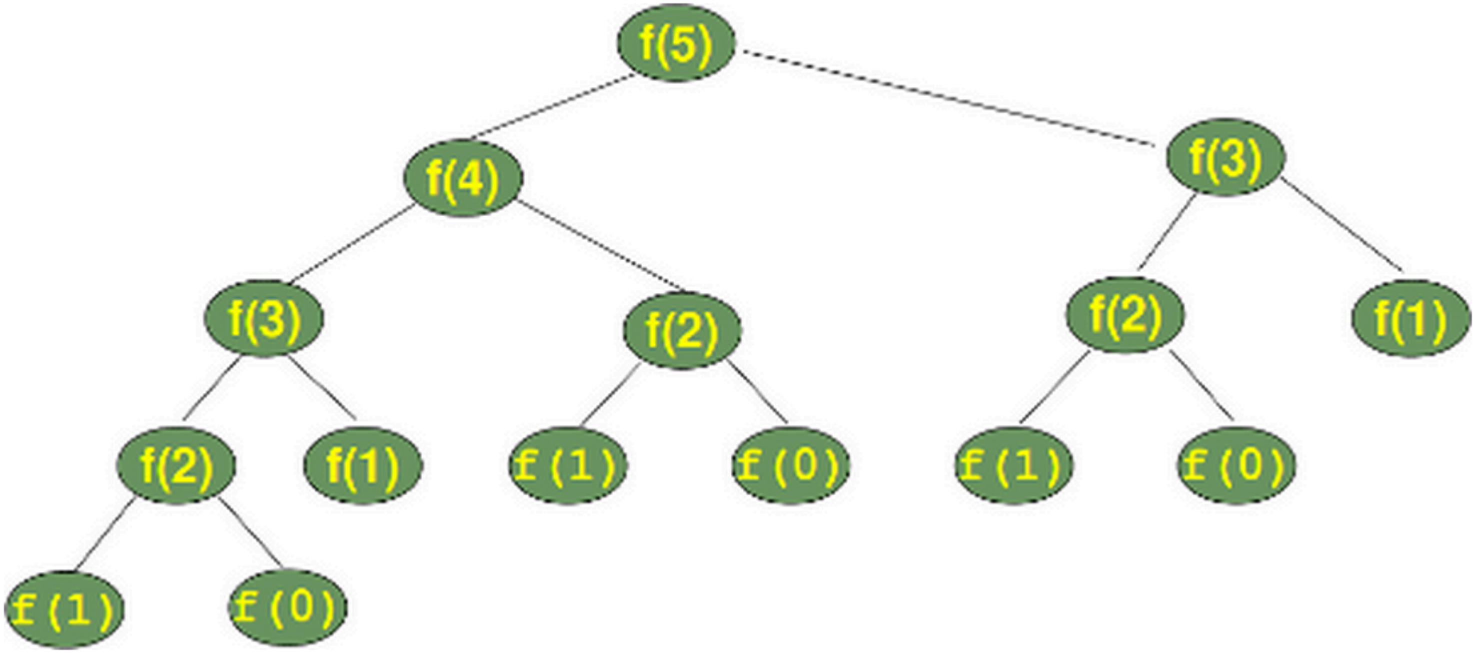
= 6

Fibonacci Recursive Approach (Binary)

```
def fibonacci(x):
        if x == 0 or x ==1:
                return 1
        else:
                return fibonacci(x-1) + fibonacci(x-2)

print fibonacci(10)
```

Recursion is slower than Iterative approach!


f(5)
f(4)
f(3)
f(3)
f(2)
f(2)
f(1)
f(2)
f(1)
f(1)
f(0)
f(1)
f(0)
f(1)
f(0)

Greatest Common Denom (Tail)

```
def gcd(a, b):
    if (0 == a % b):
        return b
    return gcd(b, a%b)
```

1) **Returns the function itself (no additions or other operations)**

2) **Can be further optimized for memory management**

3) **In functional programming languages replaces loops**

Odd or Even (Mutual)

```
def is_even(x):
    if x == 0:
        return True
    else:
        return is_odd(x - 1)

def is_odd(x):
    if x == 0:
        return False
    else:
        return is_even(x - 1)
```

How to Approach

1. Always start with thinking about base case (1)

2. Then start thinking about the repeated case (n) – what am I trying to repeat?

3. Then plan it out….

Tips for beginners

1) Draw diagrams (linear or tree)

2) Learn by example and pattern matching

3) Spot repeatable patterns that you can recurse into

4) You can use for/while loops to wrap recursion

By the way

- Recursion is not always efficient

- Recursion can be elegant and easy to read but harder to come up with solution

- Python has a limit (~1000 recursive calls, which can be changed)