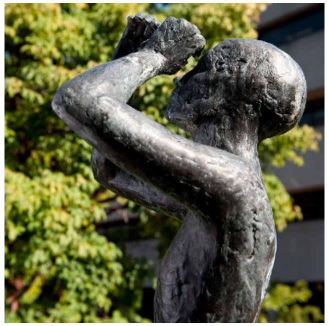


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

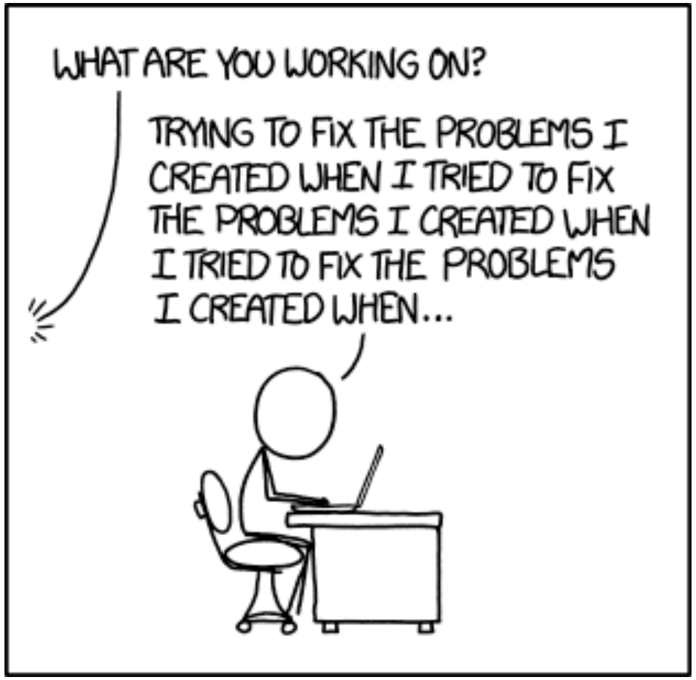
A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland

Coláiste na hOllscoile Corcaigh

Problems, problems, problems…



University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

- **Recursion**: The definition of an operation in terms of itself.
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem.

- **Recursive Programming**:
  - Writing functions that **call themselves** to solve problems.
  - We already know that in Python functions can call other functions.
  - Now it's also possible for a function to call/invoke itself?

Recursion

- Many functional programming languages (e.g. Haskell) use recursion only (no loops).

- A different way of thinking about problems

- + Leads to elegant, simplistic and short code (when used well)
- + Can solve certain types of problems better than iteration
- - Can be slower, performance isn't always as efficient as an iterative solution.

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

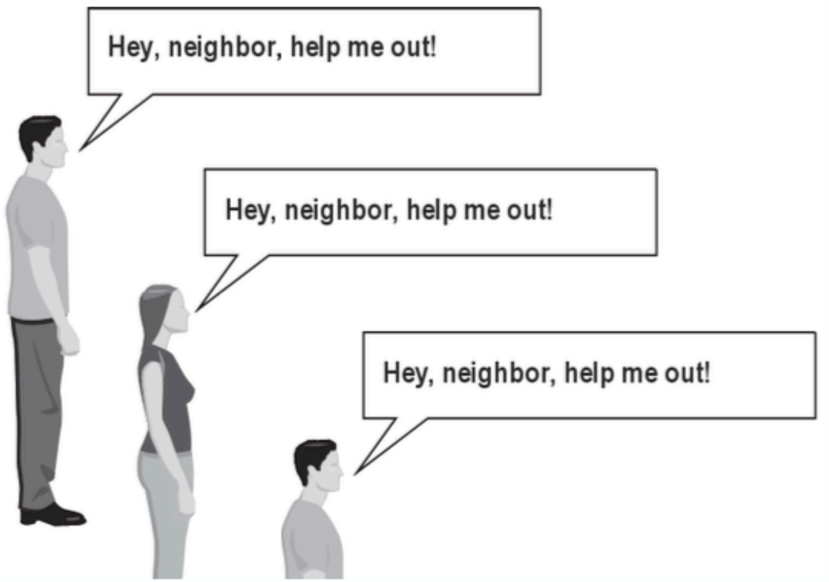
- **Question: How many students in total are behind you in your "column" of the classroom?**

- Assume you have poor eyesight and can't simply turn around and count them all.
- You can however see the person behind you.
- How do we solve the problem recursively?

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

- **Recursion breaks a bigger problem into smaller occurrences of the same problem**
- Each person solves a small part of the problem
- What's the smallest version of the problem that's easy to answer?
- What info rom a neighbour would help me?

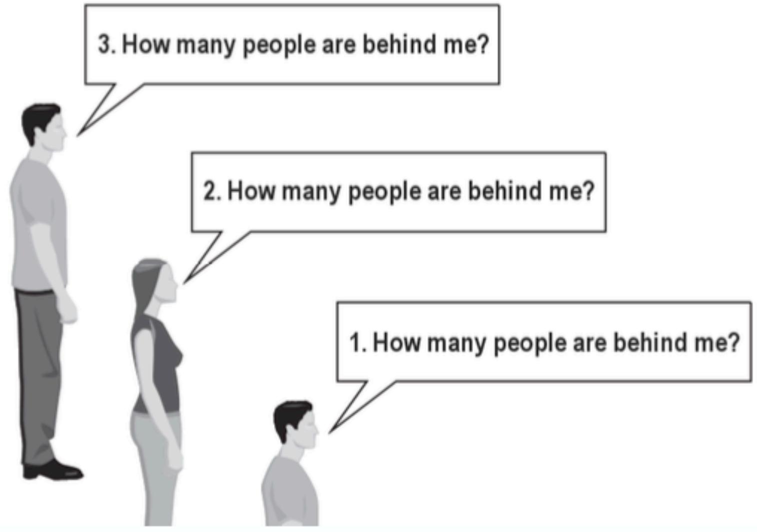


Recursion

- If there is someone behind me, ask him/her how many people behind him/her
  - When they respond with a value **N**, I will answer the person in front of me with **N + 1**

  -> Recursive Case

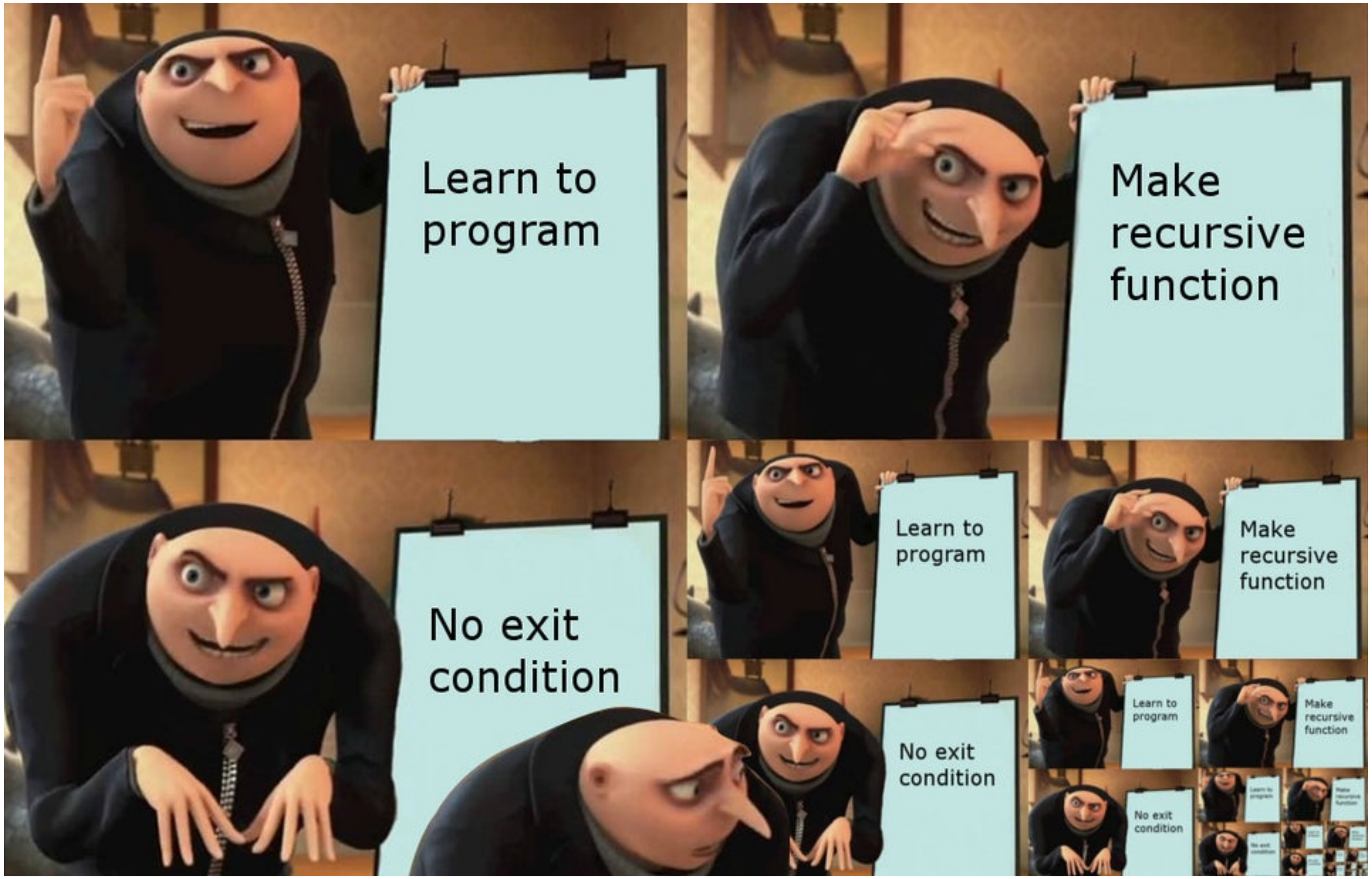
- If there's nobody behind me I will answer 0. -> Base Case



Recursion

- Every recursive algorithm involves at least 2 cases:
  - **Base Case**: A simple occurrence that can be answered directly.
  - **Recursive Case**: A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem

Recursion



University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

- **To solve a problem recursively:**
  1. Break into smaller problems
  2. Solve smaller sub-problems recursively ⟵ **Divide and Conquer**
  3. Assemble the sub-solutions

```
recursive-algorithm(input) {
    //base-case
    if  (isSmallEnough(input))
        compute the solution and return it
    else
        //recursive case
        break input into simpler instances input1, input 2,...
        solution1 = recursive-algorithm(input1)
        solution2 = recursive-algorithm(input2)
        ...
        figure out solution to this problem from solution1, solution2,...
        return solution
}
```

Recursion

- **To solve a problem recursively:**
  1. Break into smaller problems
  2. Solve smaller sub-problems recursively ⟵ **Divide and Conquer**
  3. Assemble the sub-solutions

```
recursive-algorithm(input) {
    //base-case
    if  (isSmallEnough(input))
        compute the solution and return it
    else
        //recursive case
        break input into simpler instances input1, input 2,...
        solution1 = recursive-algorithm(input1)
        solution2 = recursive-algorithm(input2)
        ...
        figure out solution to this problem from solution1, solution2,...
        return solution
}
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

$\sum 5 = 5 + 4 + 3 + 2 + 1$ **OR** $\sum 5 = 5 + \sum 4$

$\sum 8 = 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$ **OR** $\sum 8 = 8 + \sum 7$

What's the formal definition?

Recursion

```
def summationI (n):
    sum = 0
    for count in range (1, n, 1):
        sum = sum + count
    return sum
```

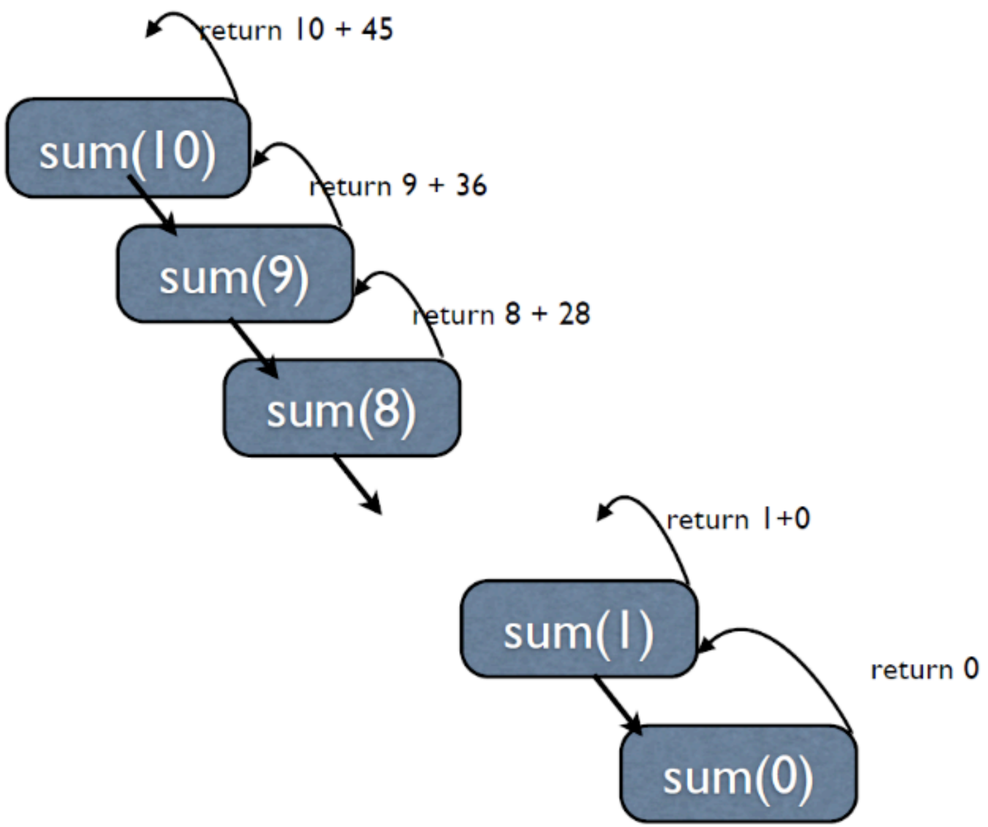
n+1 not n

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

```
def summationR (n):
    if n == 0:
        return 0
    else:
        sum = n + summationR (n-1)
    return sum
```

Recursion



The system keeps track of the sequence of method calls that have been started but not finished yet (active calls)

Recursion

```
5! = 5 * 4 * 3 * 2 * 1        OR       5! = 5 * 4!
8! = 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1    OR   8! = 8 * 7!
```

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

Base Line

Recursive Case

Recursion

```
def factorialI (n):
      fact = 1
      for count in range (2, n+1, 1):
            fact = fact * count
      return fact
```

Recursion

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

def factorialI (n):
    fact = 1
    for count in range (2, n+1, 1):
        fact = fact * count
    return fact
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recursion

Let's look at some code…

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh