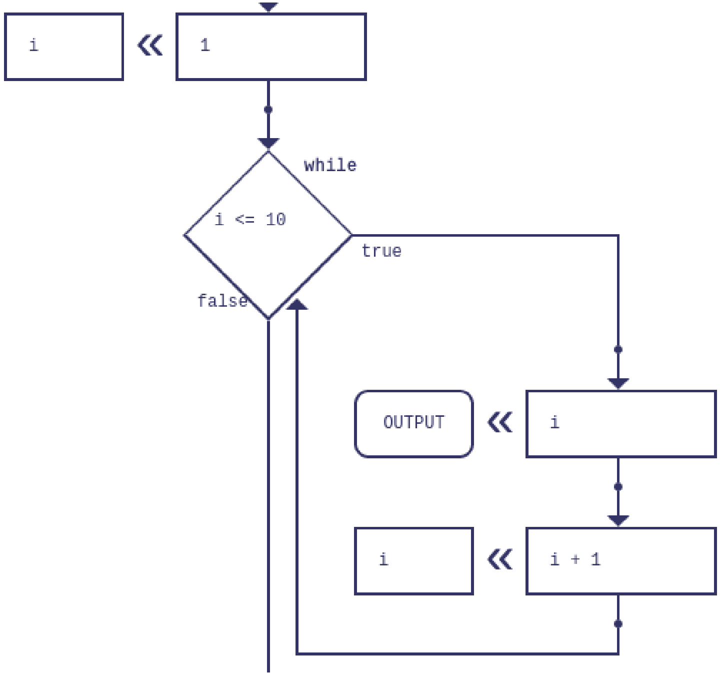
CS105 Problem Solving

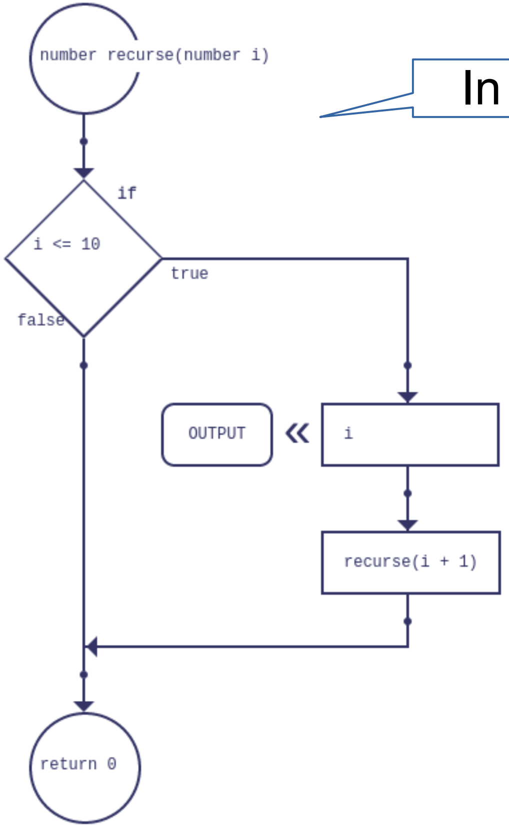
Recursion

Wholeness

- A loop executes the same instructions each time with slightly different data.
- When a function calls itself the same instructions are executed again, but with slightly different data.
- Recursion is when a function calls itself, creating the same effect as a loop.
- The stack then holds multiple copies of the function.

Normal loop & Recursive Version





In main: recurse(1)

Example / See Stack

- Let’s run the recursive function shown on the previous page
    - Pay close attention to what happens on the stack!

- Each call gets its own stack frame (its own values)
    - Just like a loop, same instructions, different values

Exercise

- Use recursion to output all the numbers from 1 to 100
    - Once you have it working change it to go up 2 at a time

Stack Limit

- With recursion every ‘loop’ iteration creates a new stack frame
  - A new copy of the function
  - With new local variables

- There is a limit to the amount of stack frames
  - Reaching the limit is the ‘maximum’ recursion depth

Example

- Show an infinite recursive loop and see the max recursion depth

- Any function that calls itself without an if statement will reach this

Main Point 1

- Recursion works by creating multiple copies of the same function on the stack. Each has its own local variables
- Just like the stack, all of life is found in layer

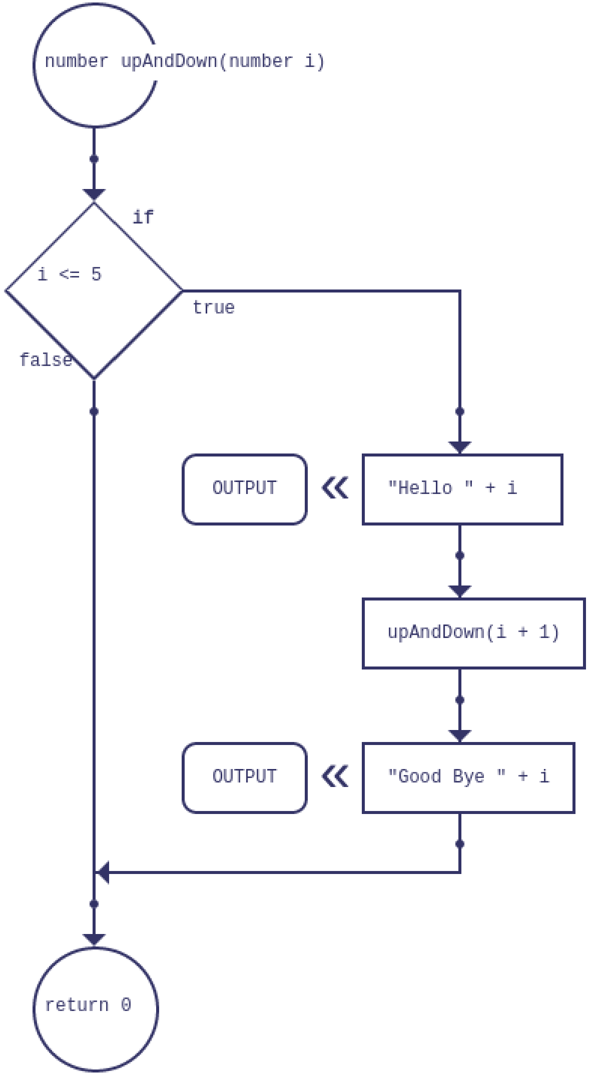
Base Case

- To stop recursion going into an infinite loop (reach the max recursion depth)
  - Ensure that each iteration moves us closer to a base case
  - Base case: returns (a value) instead of calling itself
- Recursion creates frames until the base case
  - Then it comes back down through the frames

Up and Down

- Prints Hello 5 times going up the stack
  - Then hits the base case
  - This is our first return

- Prints Good Bye 5 times going down the stack
  - Notice that it counts down when saying goodbye!



Exercise

- Make a recursive function called goingDown()
  - Main should call it as: goingDown(1)
  - It should use recursion to print the numbers 1 through 5 in reverse (5 printed first, then 4, then 3, 2, 1) by printing the number going “down” the stack.

Going Up, Going Down

- Instructions that are before the recursive call are executed while “going up” the stack
- Instructions after the recursive call are executed while “going down” the stack
  - Doing things going down makes it seem like it’s going in reverse

Main Point 2

- Each recursion step always has to move towards a base case, so that it doesn’t go on indefinitely
- Reaching the base case is not the end of execution, just the turning point.
- Change is the nature of the relative, you cannot keep doing the same thing indefinitely

Return Values

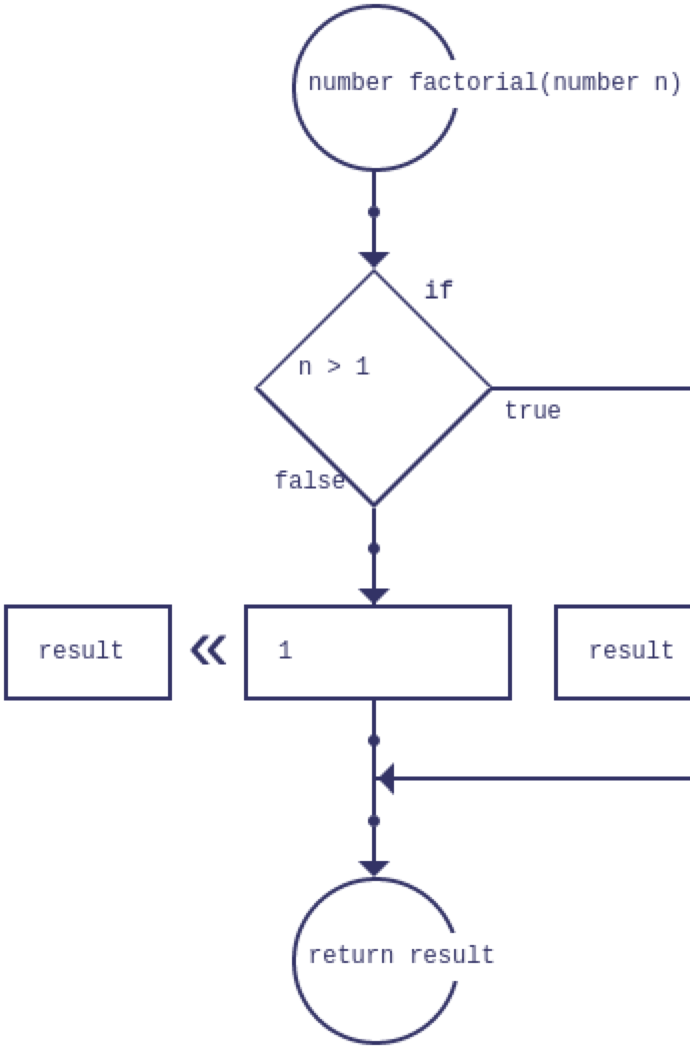
- So far non of our examples have returned a value
- The instructions going down the stack can receive / work with the return value from the calls that were above them
- This is very useful for certain tasks!

Factorial

- In Math the factorial of a positive interger n is defined as:
    - factorial(n) = n * factorial(n -1)
        - for all positive integers less than or equal to n

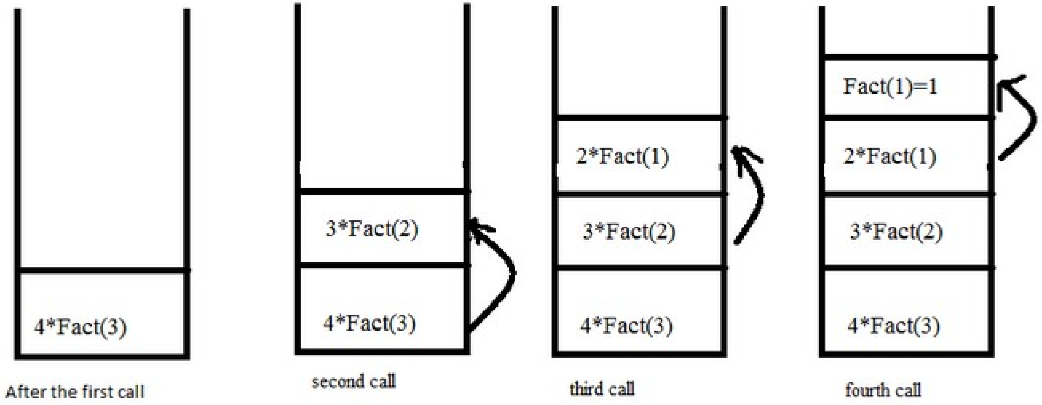
- This definition is recursive
    - We can implement it almost exactly as written

Demo Factorial

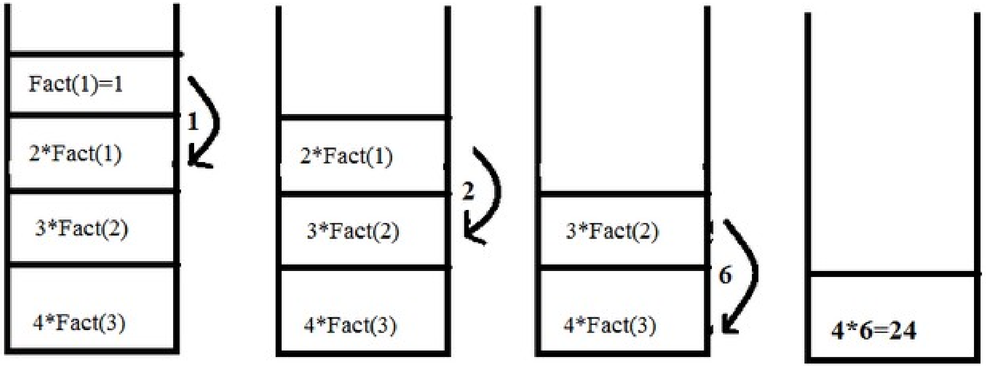


In the stack

When function call happens previous variables gets stored in stack



Returning values from base case to caller function



Exercise

- The sum of an array of integers can be seen as
  - The first number plus the sum of the rest of the array
  - The second number plus the sum of the rest of the array
  - The third number plus …
  - …
  - For the last number just return the last number

- Write recSum(array, position) that implements this
  - Each call passes the same array, but the next position

Main Point 3

- Because you can use the return value from the frame above you, recursion can express solutions to certain problems in a very clean and efficient way

Summary

- Recursion is an alternate way to create a loop
  - It works by creating multiple copies of a function on the stack
  - Recursion has to reach a base case to start going down the stack again
  - Using return values from previous calls allows you to write really clean and readable solutions to certain problems.
  - Understanding recursion is vital to becoming a good programmer

- Understanding deeper levels of nature gives greater power