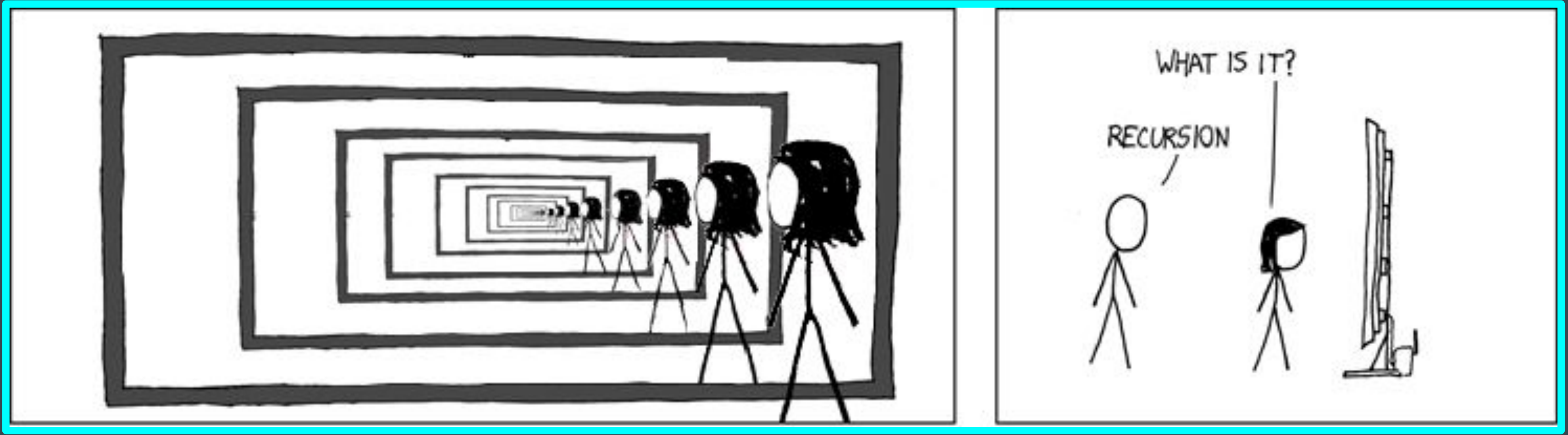
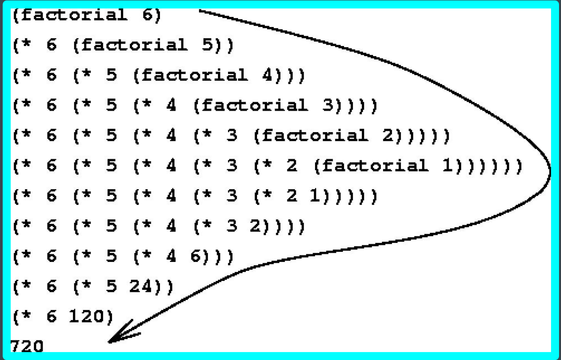
# Recursion



A tutorial by Joshua Rovner

# What is Recursion?

- ❑ Recursion occurs when a method calls itself.
- ❑ Typically, a certain condition must be met for a recursive call to be made.
- ❑ A recursive method runs until it reaches the base case, at which point a specific value is returned, and the recursive calls are traced backwards with the determined value.
- ❑ The example below shows the trace for a recursive method which finds the factorial of a number by multiplying the passed parameter by the method with parameter - 1.



```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

# Types of Recursion

- ❑ **Linear recursion** - when a method only makes one call to itself each time it runs (e.g. factorial).
  - ❑ **Tail recursion** - a type of linear recursion, where the recursive call is the last thing the method does - can often be implemented iteratively (e.g. finding the greatest common denominator of two numbers).
- ❑ **Mutual recursion** - when several (typically 2) functions recursively call each other (e.g. finding whether a number is even or odd)
- ❑ **Binary recursion** - when a method makes two calls to itself each time it runs (e.g. fibonacci sequence).
- ❑ **Exponential recursion** -  makes an exponential number of calls relative to the data set being worked with.
- ❑ **Nested recursion** - a recursive call within another recursive call.

## Pros, Cons, and When to Use

- Unfortunately, recursion takes up much space as each call must create more storage space on the **stack** (collection of elements) for parameters and variables. Also, lots of time must be allocated to running through and performing a method’s tasks every time it is called.
- However, recursive solutions are typically shorter and easier to understand than ones with several for-loops. In addition, they are very efficient when processing tree structures.
- Recursion should be used when a similar action can be repeated in order to reach a smaller, identifiable target point. Often, recursive methods can be used instead of loops to solve a problem.