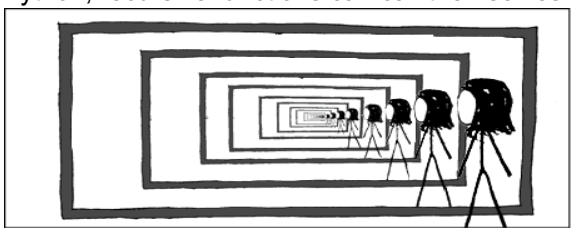
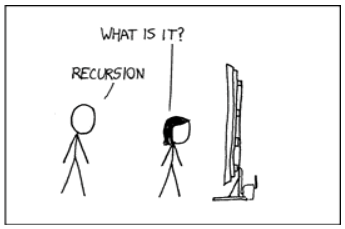
## Recursion 101

Recursion involves repetitions of a particular function or action within itself. In Python, recursive functions can call themselves in the same function.






Recursion resolves to the simplest case where no further calculations are needed. This is called the **base case**. The function will divide itself into smaller and smaller pieces until it converges on the base case.

### Recursion versus Iteration

Iteration and recursion are each based on control statements:

- Iteration uses `for` and `while`
- Recursion uses `if`, `elif`, and `else`

While both involve iteration as a concept, iteration needs a statement defining bounds or a range whereas recursion iterates via *n* number of function calls.

They also both use **termination tests**:

- Iteration – when the loop condition fails
- Recursion – when the base case is reached

Each approaches termination **gradually** or in a **controlled manner:**

- Iteration – modifying counter until loop fails
- Recursion – function calls until base case is reached

Both can run **infinitely:**

- Iteration – When the loop condition is never `False`
- Recursion – when the base case is never reached or never tested for

### Important concepts:

#### Stack Overflow

- When too many recursion calls occur than can be stored in the stack. This is often caused by i**nfinite recursion** in which there is a missing base case or incorrectly formatted recursion call where the base case is never reached.
- Also the name of the handy website.

#### Indirect recursion

- When a function calls a recursive function, it is still considered recursion because the first function hasn't completed execution.
- ```
  def A():
      B()
  def B():
      A()
  ```

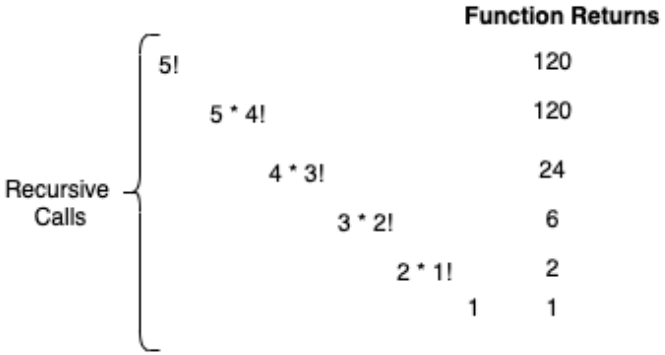
### Practical example of recursion in Python:

A common example of recursion is represented by factorials (*n!*).

```
def factorial(number):
    if number <= 1:
        return 1
    return number * factorial(number - 1)
```

In this example, the final `return` statement is the recursive call.

Under the hood, this is what is happening (provided `number = 5`)




### Drawbacks

High overheard cost in terms of processing time and memory. Each recursive call initiates the creation of a copy of the function and runs it, adding additional computations to the stack. Each value is also stored until the function call is terminated, requiring additional memory.

For example, if you were to create a function mimicking the Fibonacci sequence (hint hint) and calculate the Fibonacci value of 20, the function would require 21,891 total calls back to the function. To calculate the value of 30 the function would require 2,692,537 calls.

Sources:

Deitel, Paul, and Harvey Deitel. *Intro to Python for Computer Science and Data Science*. Pearson, 2020.
Image courtesy of xkcd, https://xkcd.com/
```