Link for Code visualization: https://pythontutor.com/render.html#mode=display

Let's take a simple example like calculating the factorial of a number. The factorial of a number is the product of all positive integers up to that number.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)

factorial(5)
```

Here's how it works: If you ask for `factorial(5)`, it goes like this:

1. `factorial(5)` calls `factorial(4)`
2. `factorial(4)` calls `factorial(3)`
3. `factorial(3)` calls `factorial(2)`
4. `factorial(2)` calls `factorial(1)`
5. `factorial(1)` returns 1 (base case)
6. `factorial(2)` multiplies 2 with the result of `factorial(1)` and returns 2
7. `factorial(3)` multiplies 3 with the result of `factorial(2)` and returns 6
8. `factorial(4)` multiplies 4 with the result of `factorial(3)` and returns 24
9. `factorial(5)` multiplies 5 with the result of `factorial(4)` and returns 120

And there you have it! Recursion is cool, right?

Now, about stack overflow errors. Imagine you have a stack of plates, and you keep adding more and more without taking any away. Eventually, it's gonna topple over, right? That's kind of like what happens in programming when you have too many function calls.

In our example, if you ask for `factorial(10000)`, you'll run into a stack overflow error because the computer can't handle so many nested function calls. To avoid this, we can use something called tail recursion or iteration. But for now, just keep in mind that too much recursion without a proper base case can lead to a stack overflow error.