Recursion basics

In short, **recursion** in programming is when **a function calls itself**. It has a case where it terminates and a set of rules to reduce other cases to the first case. A function that can do it is called a **recursive function**.

Recursive matryoshka

Think of it like a Russian doll, matryoshka. It's a doll, or, more accurately, a set of dolls placed one inside another. You open the first doll, and there's the second, open this one and get the third, and so on until you get to this last doll, which won't open.

Designing a function

let's now create an algorithm to count the dolls. Each recursive function consists of the following steps:

- A **trivial base case** stops the recursion. This is the case we know the result for. For example, if we find a doll we can't open, we take it and proudly state: "it's our smallest doll!"
- A **reduction step** (one or more, imagine that our doll contains two dolls inside it!) gets us from the current problem to a simpler one. For example, if our doll can be opened, we open it and look at what is inside.

```
How to count X dolls:
   If X is 1, the result is 1.
   If X is not 1, see: "How to count X-1 dolls" + 1.
```

Advantages and disadvantages
Many recursive functions can be written another way: we could simply go through all numbers from 1 to n and compute the function for each number. For example, we can open the largest doll, say "One", throw the doll away, and repeat these steps until we found the last one. This way of computing is called **the loop**.

It depends on the programming language. As a rule, in Python and Java, loops are more efficient in terms of **time** and **memory**. Recursion is slower and "heavier" because each call of a function takes additional memory, and recursive functions usually get called many times.

Well, it has one certain advantage over loops: in some cases, it is intuitive.

So, recursion is usually slower and less memory-efficient, but it saves the developers' time.