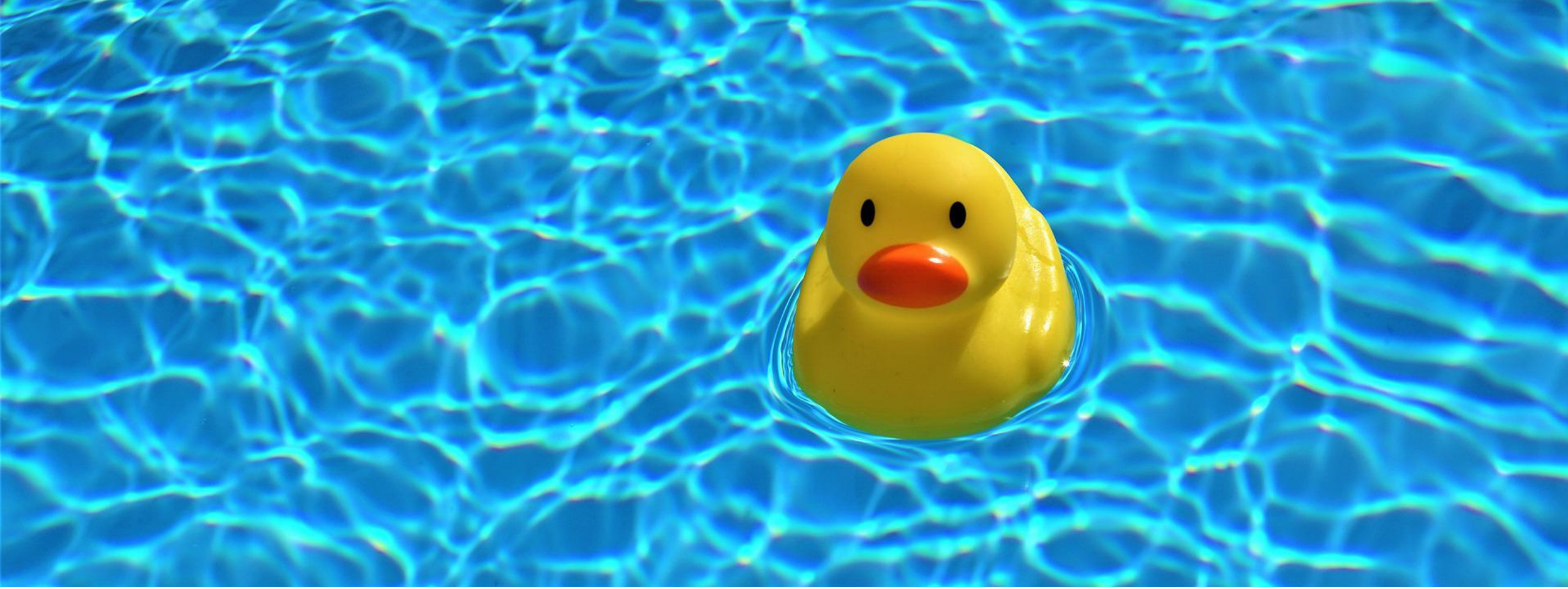
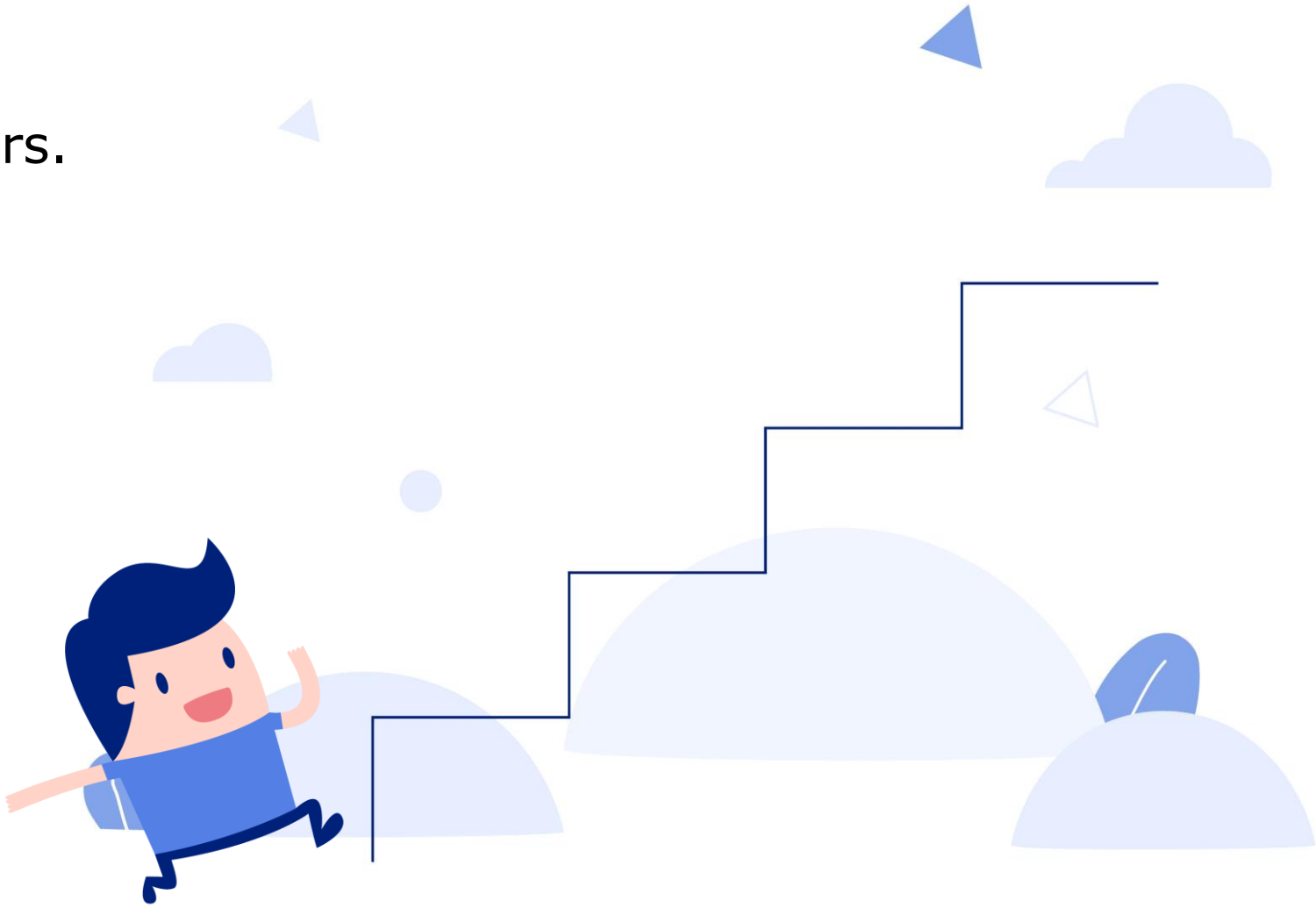


RECURSIVE FUNCTIONS EXAMPLE: CLIMBING STAIRS

IMAGINE STAIRS

You are walking up the stairs.



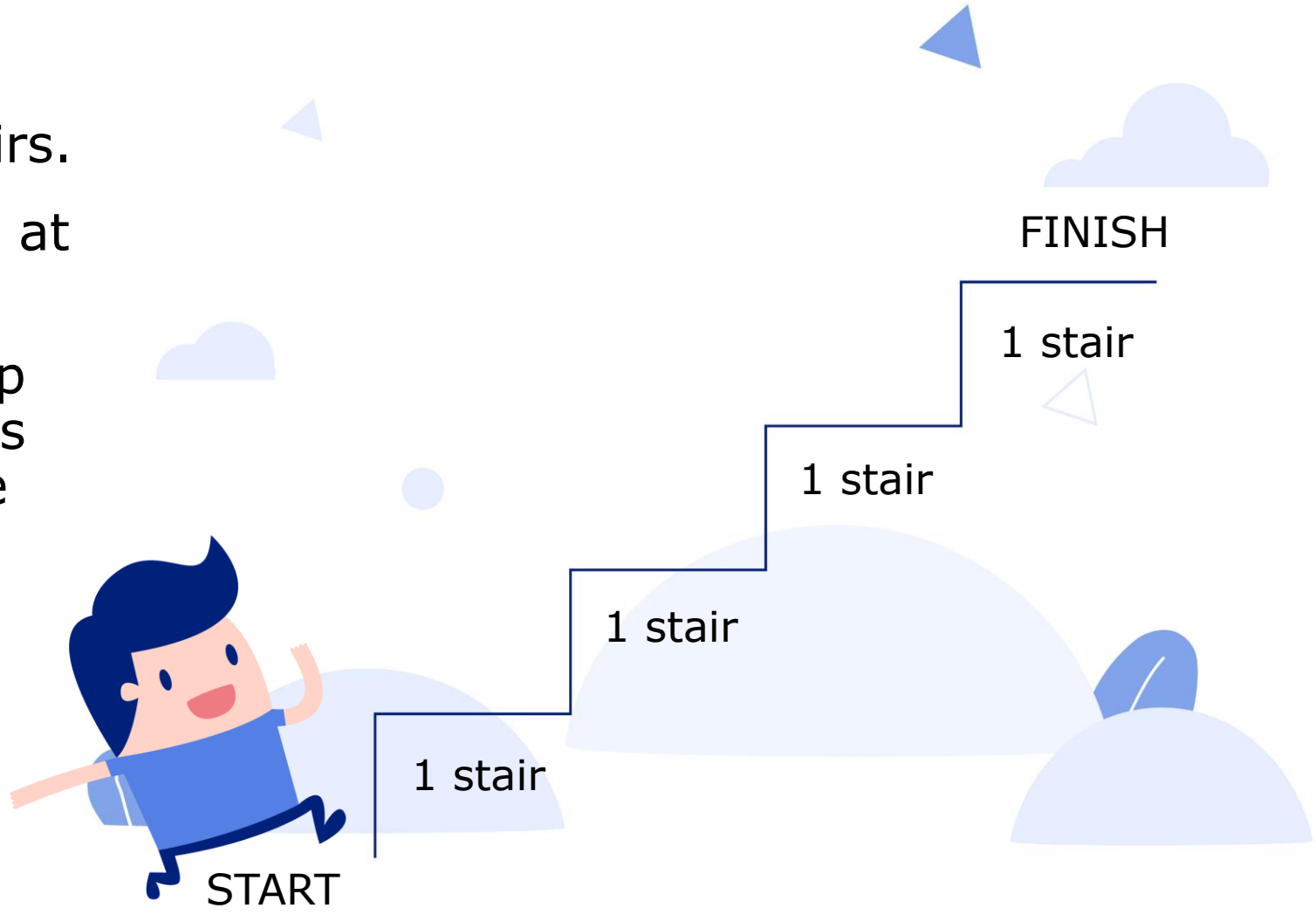
ALSO! REMEMBER: YOU ARE A COMPUTER :-)

IMAGINE STAIRS

You are walking up the stairs.

You can only take one step at a time.

Your goal is to be at the top of the staircase: this means your goal is to climb all the stairs, and have no more stairs left to climb.

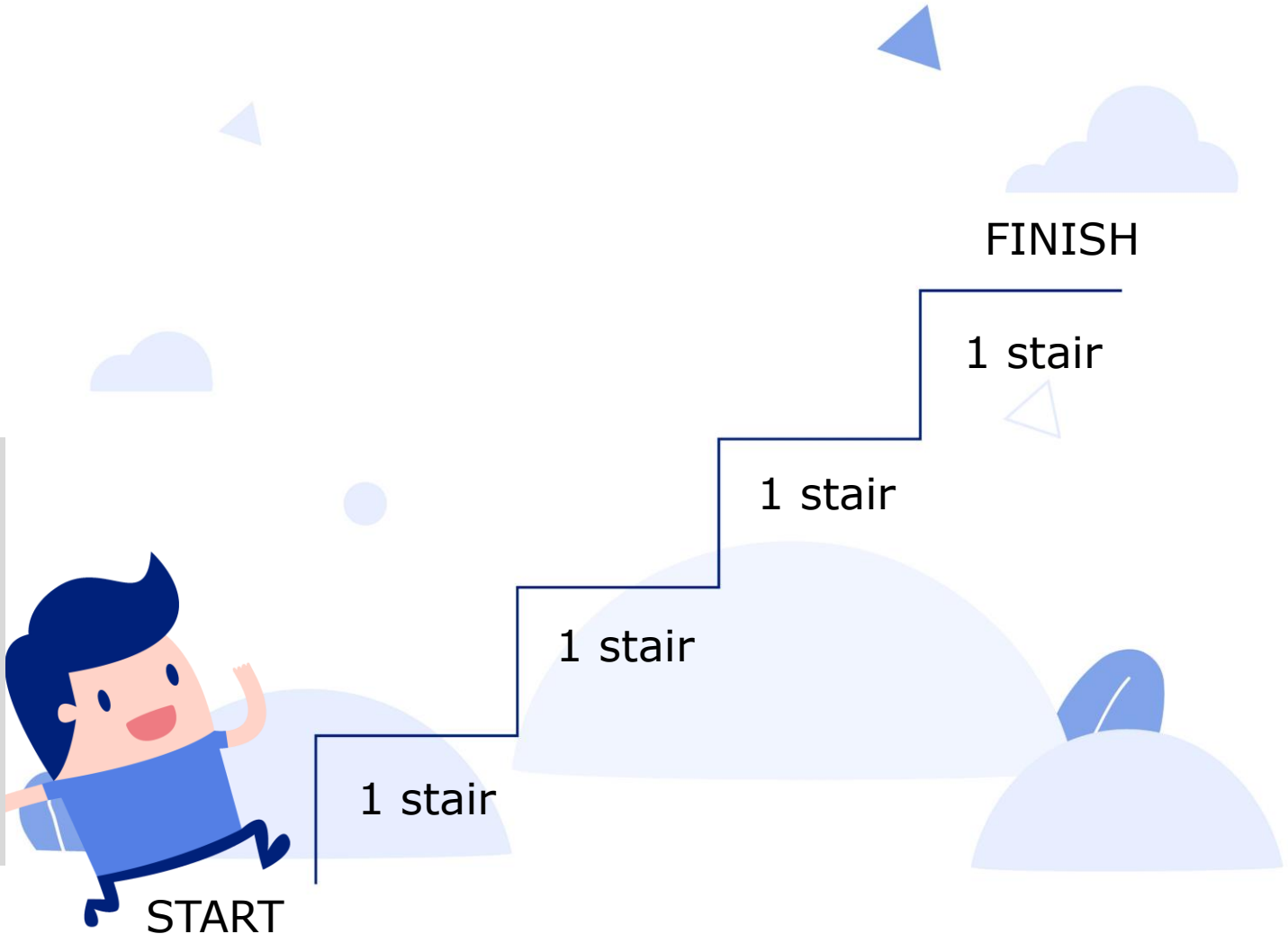


A SIMPLE FUNCTION AND A WHILE-LOOP

If you can climb one step at a time, then you can write a function called `climbStair()` that represents that one step at a time.

You can use a while-loop:

```
>> def climbStair():
>>       print(“you climbed one stair!”)
>> numStairs = 4
>> while numStairs >= 0:
>>       climbStair()
>>       numStairs -= 1
```



BUT WHAT IF YOU WANT IT ALL IN ONE FUNCTION?

What if you just want a function called `to_the_top(x)`, where x is the number of stairs you need to climb, and you climb one stair at a time?

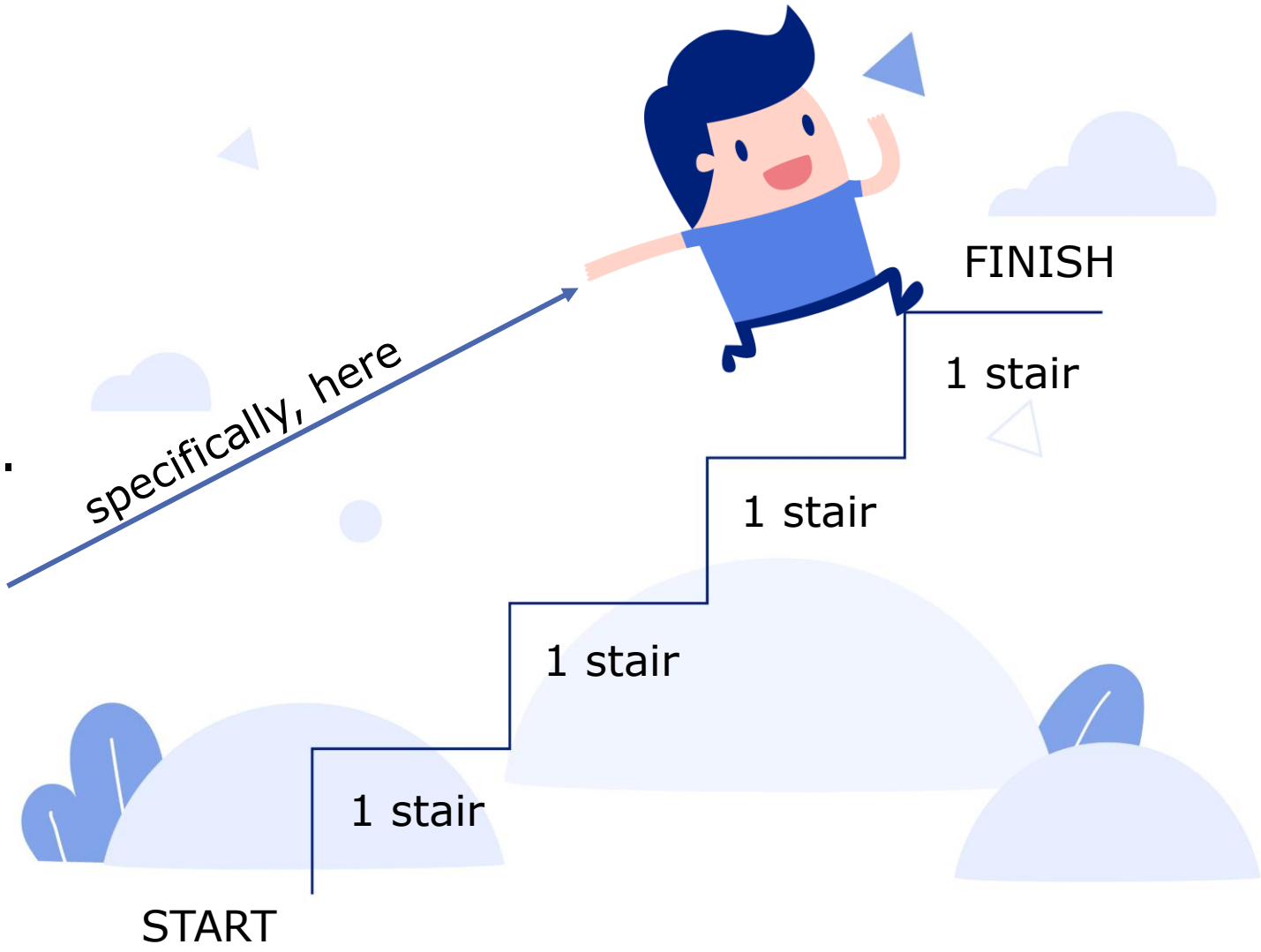
You can write a *recursive* function!

```
>> def to_the_top(numStairs):
>>   if x == 1:
>>      print(“congratulations, you are on top!”)
>>   else:
>>      return 1 + to_the_top(x-1)
```

ONE STAIR AT A TIME

Your function `to_the_top(n)` allows you to climb only one stair at a time. It will keep repeating that one action (climb one stair) until you reach the top of the staircase.

At some point, you will climb one stair, and then reach the top.



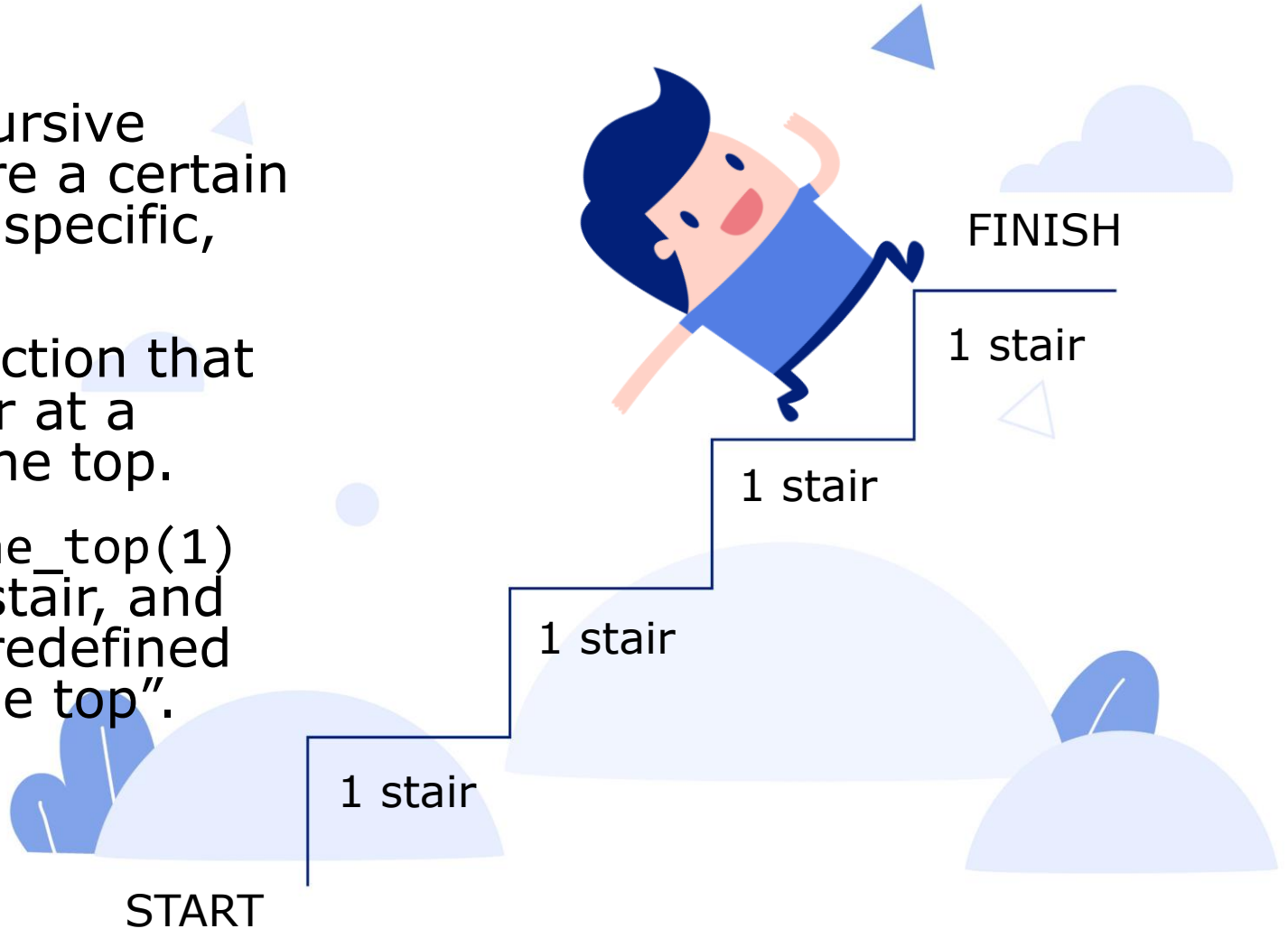
BASE CASE

A base case of the recursive function is a case where a certain parameter results in a specific, predefined result.

`to_the_top(n)` is a function that lets you climb one stair at a time, until you reach the top.

So we know that `to_the_top(1)` means you climb one stair, and then you will have a predefined result of "you are at the top".

This is your base case.



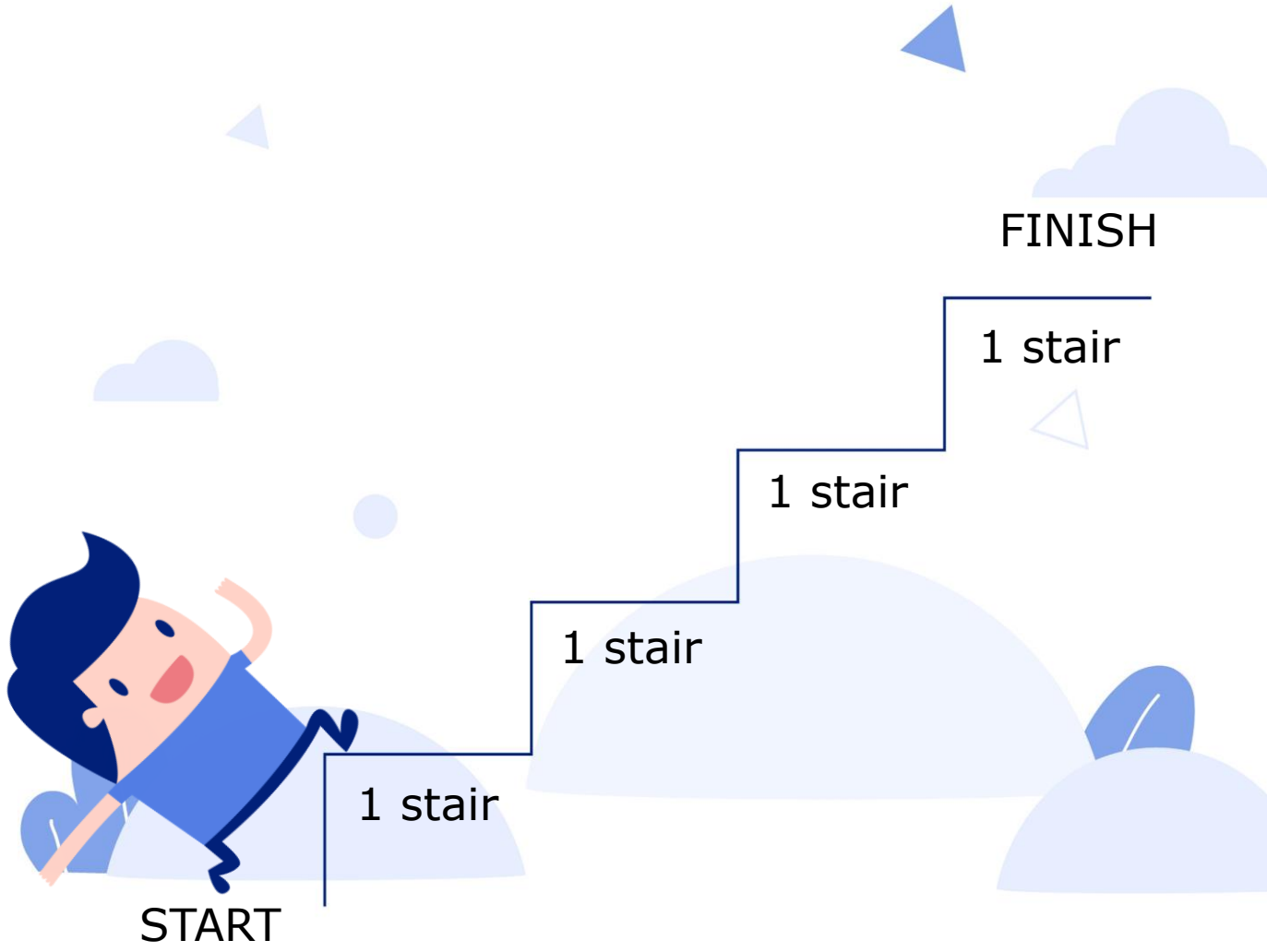
RECURSIVE CASE

Now things get a little bit more hairy, because you start at the bottom of the stairs! Not at the top.

As you prepare to reach the top, you ask yourself for the **first** time:

If I climb one stair, will I reach the top?

The answer is no. If you climb one stair, you will have more stairs left.

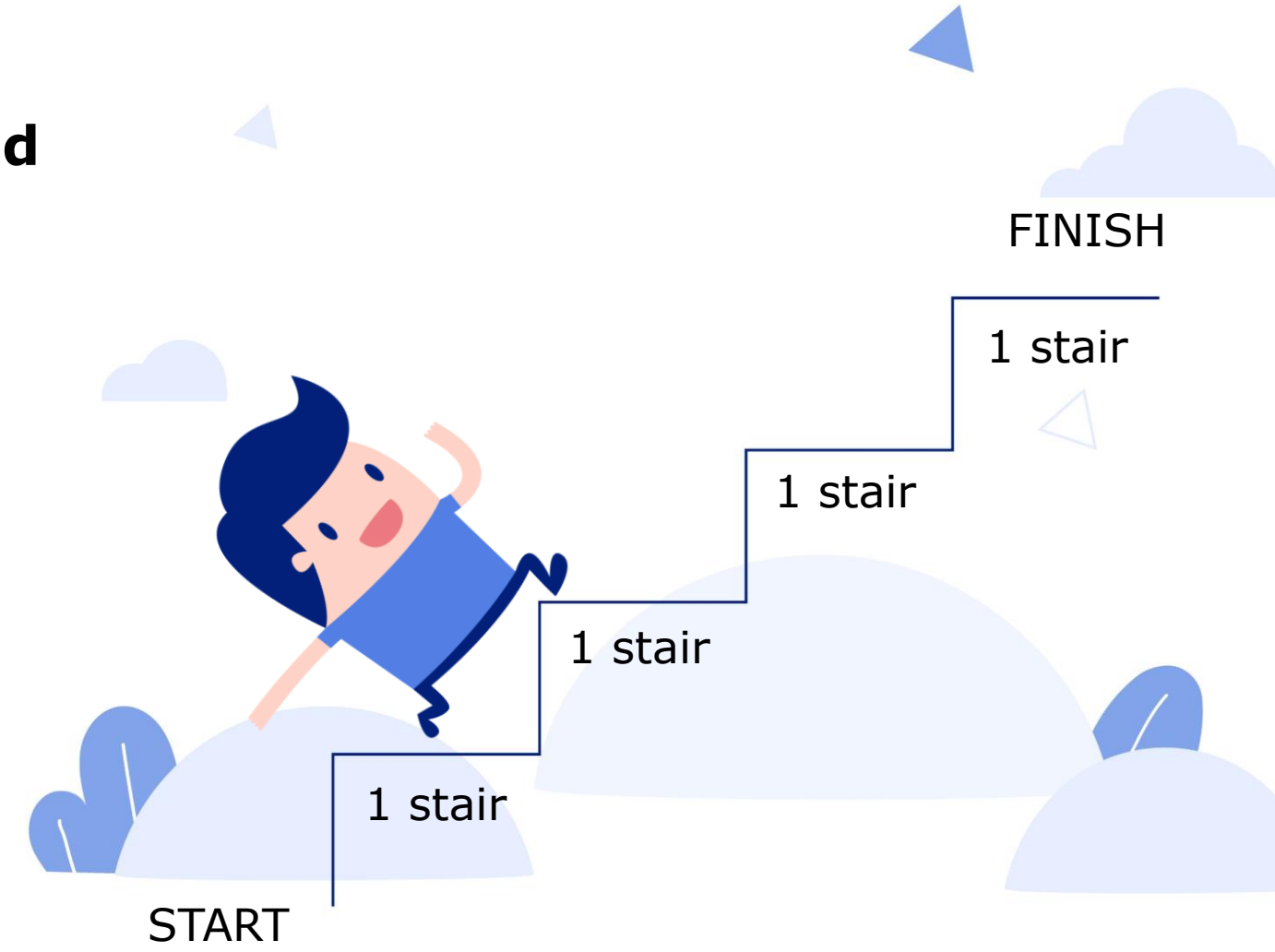


RECURSIVE CASE

So you ask again, for the **second** time:

If I climb one more stair, will I reach the top?

The answer is no. If you climb one more stair, you will have more stairs left.



RECURSIVE CASE

So you ask again, for the **third** time:

If I climb one more stair, will I reach the top?

The answer is no. If you climb one more stair, you will have more stairs left.

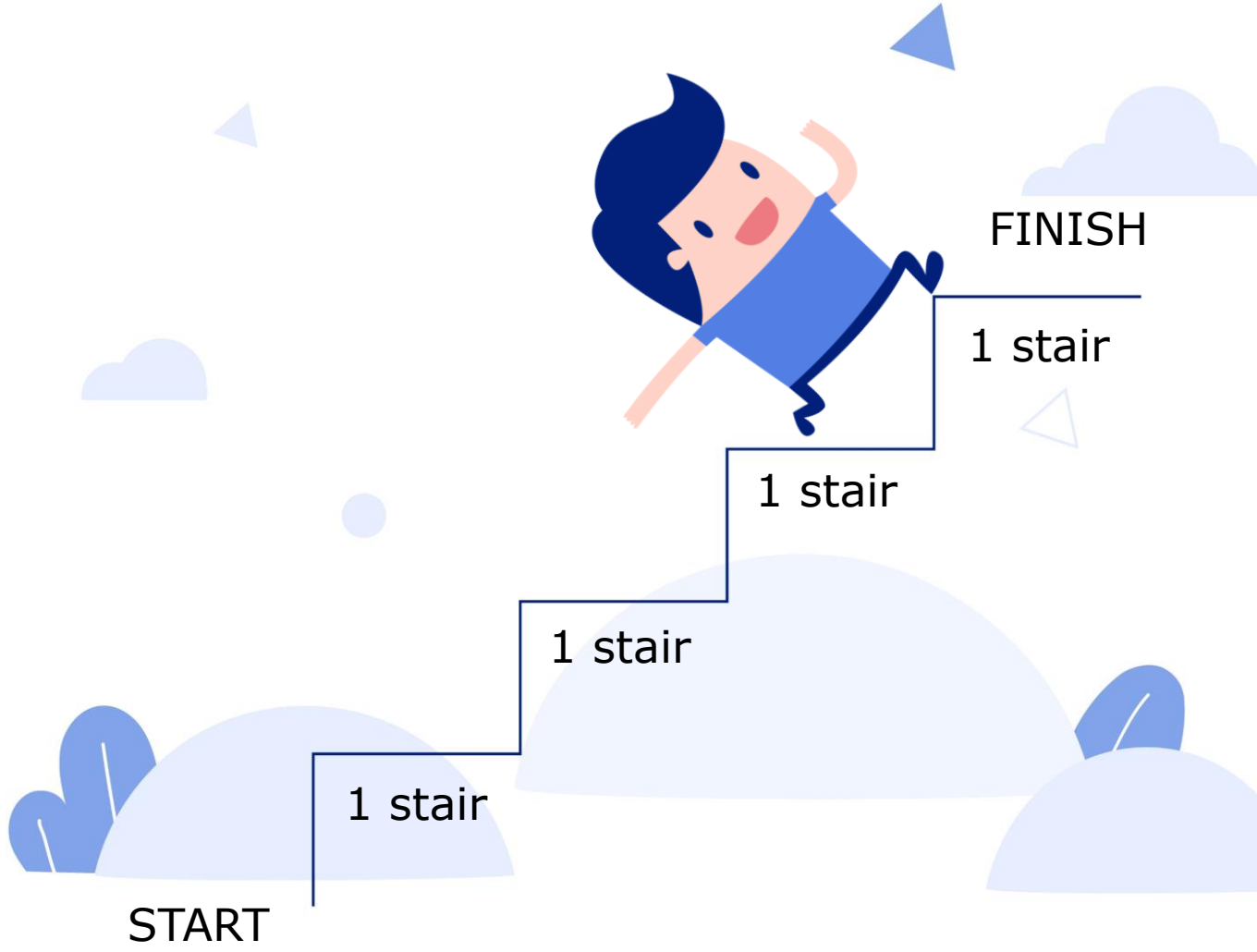
BASE CASE

So you ask again, for the **fourth** time:

If I climb one more stair, will I reach the top?

The answer is YES! If you climb one more stair, you will reach the top.

You reached your base case.



THE STACK

order in which you CLIMB

As you were thinking about climbing the stairs, you asked yourself the same question four times:

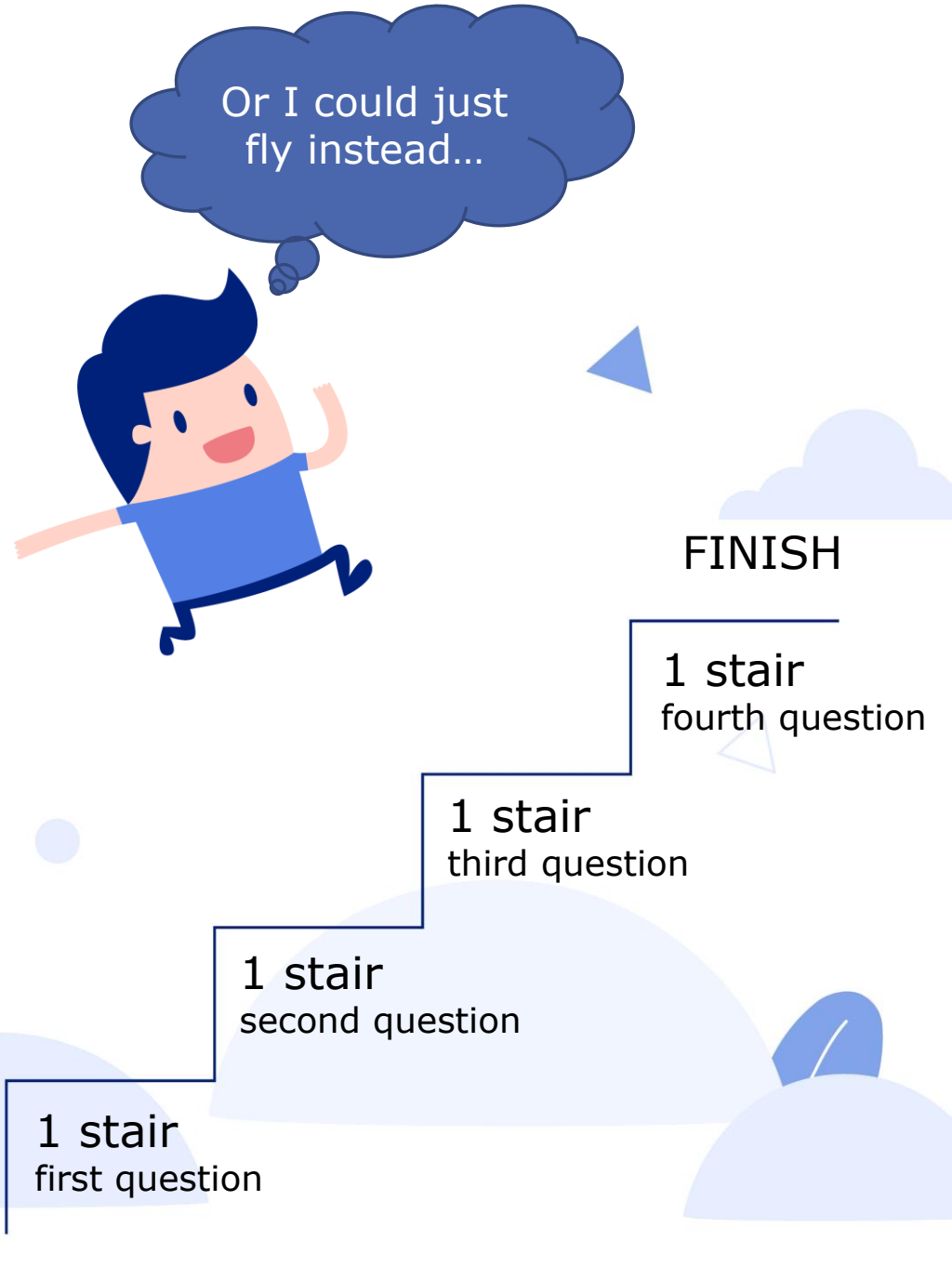
The **first** question: If I climb one stair, will I reach the top? The answer was no!
→ therefore, you execute your function (climb one stair) and then you keep climbing → remember that one stair, and keep climbing

The **second** question: if I climb one stair, will I reach the top? Answer was no!
→ therefore, you execute your function (climb one stair) and then you keep climbing → remember that one stair, and keep climbing

The **third** question: if I climb one stair, will I reach the top? Answer was no!
→ therefore, you execute your function (climb one stair) and then you keep climbing → remember that one stair, and keep climbing

The **fourth** question (base case): if I climb one stair, will I reach the top? Answer was yes!
→ therefore, you climb the stair and you reach the top!

How many stairs did you climb? Or: how many stairs do you remember climbing?



THE STACK

When we say remember that one stair, what we really mean is:

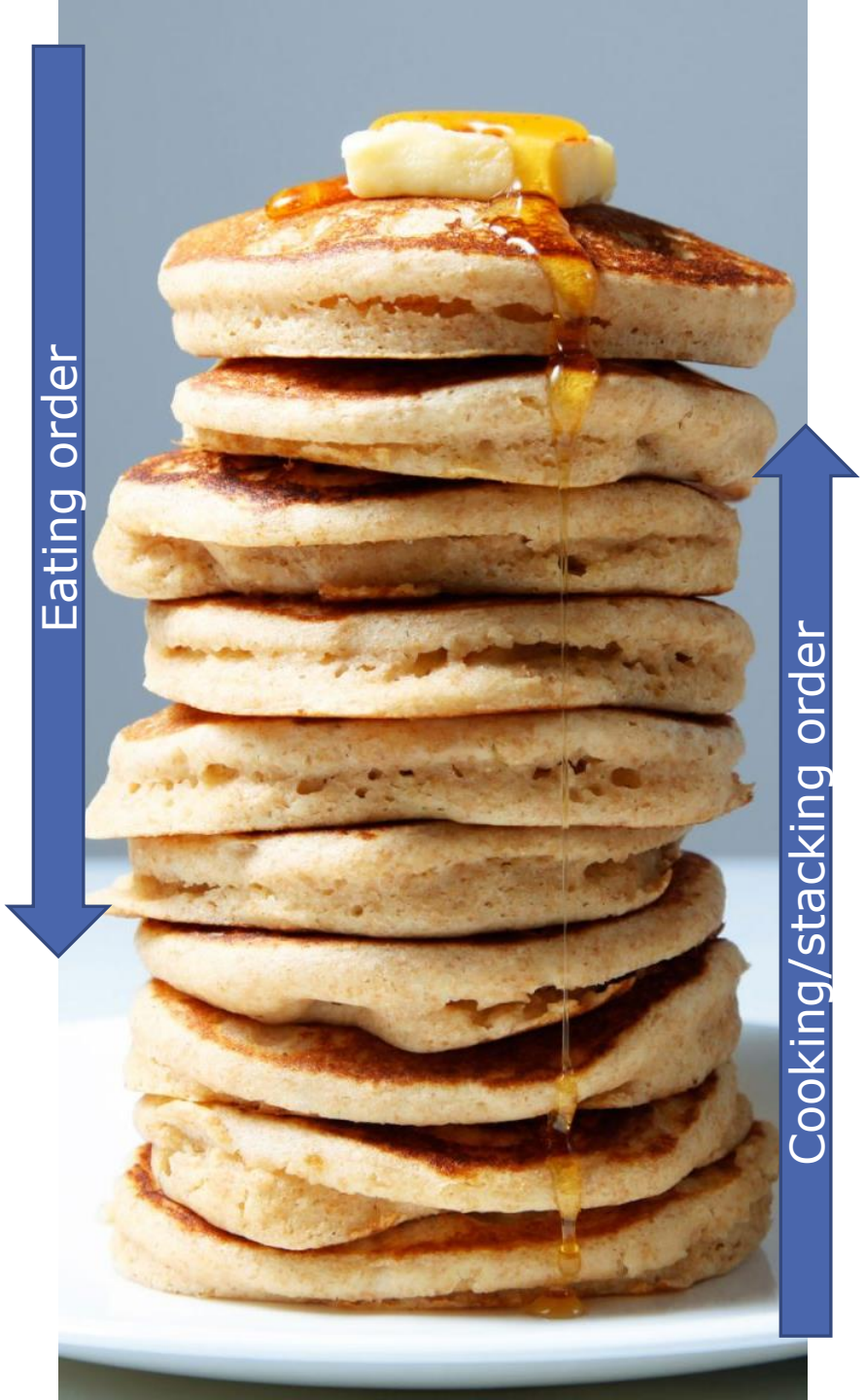
Put the content of the local variable on the stack, and come back to it later – after you reach the base case.

THE STACK

Every time we call on a recursive function and we peel away part of the problem, that little part we just peeled away is put on the stack.

A stack is a conceptual structure, which consists of homogeneous elements in a sequence.

The rule is ***last in, first out***: the last item/element to be put on the stack is also the first item/element to be retrieved. Kinda like pancakes on a stack: the pancake that is put on the stack last will also be the first one to be eaten.



LAST IN, FIRST OUT

You reached the top. How many stairs did you climb?
Or: how many stairs do you remember climbing?

The **fourth** question (base case): if I climb one stair, will I reach the top? Answer was yes!
→ therefore, you climb the stair and you reach the top!

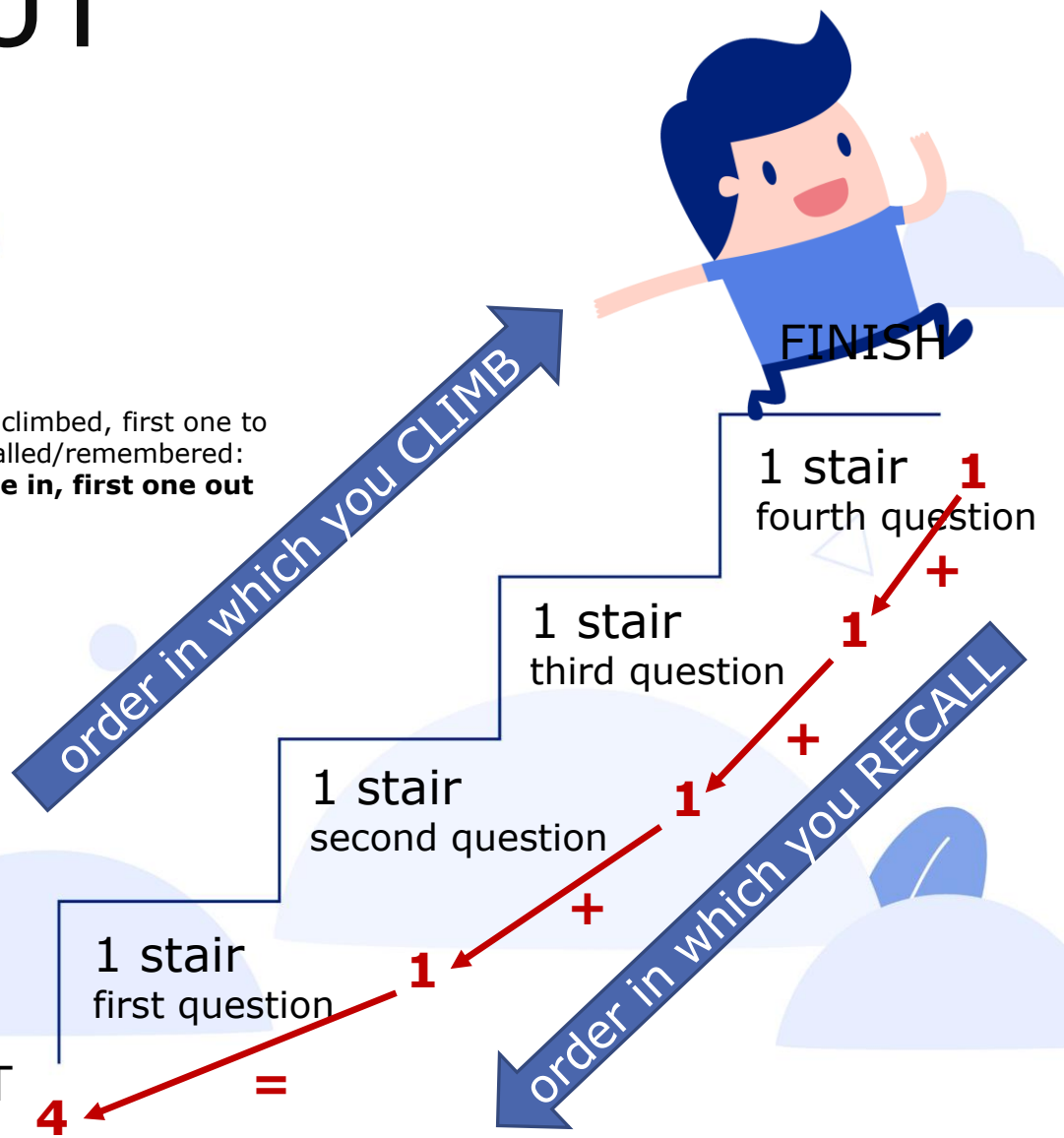
The **third** question: if I climb one stair, will I reach the top? Answer was no!
→ therefore, you climb one stair and then you keep climbing → remember that one stair, and keep climbing

The **second** question: if I climb one stair, will I reach the top? Answer was no!
→ therefore, you climb one stair and then you keep climbing → remember that one stair, and keep climbing

The **first** question: If I climb one stair, will I reach the top? The answer was no!
→ therefore, you climb one stair and then you keep climbing → remember that one stair, and keep climbing

last one climbed, first one to be recalled/remembered:
last one in, first one out

first one climbed, last one to be recalled/remembered:
first one in, last one out



STACK OVERFLOW

When your stack becomes too large – when you've used up more memory for the stack than your program is allowed to use – the stack overflows.

And then your code crashes and all your pancakes are all over the floor.