Iteration Example

Let’s look at a famous example of iteration. We’re going to study the Fibonacci Sequence . That’s Fibonacci a mathematician from long ago and here is this sequence. It starts with 0 and then 1 and then every element after that is the sum of the previous two elements , so 0 and 1 make 1

, 1 and 1 make 2 make 3 , etc, and you can see that it starts getting fairly large before too long

Now Fibonacci did not invent this sequence , it was discussed and descibed by mathematicians long before him but he made it popular in the West and so we still refer to him every time we see this sequence of numbers.

Every Fibonacci number is associated with its index , the position in the Fibonacci sequence, so we usually call this the zeroth Fibonacci number, the first, the second, the third, the fourth and the fifth, etc. So the fifth Fibonacci number is 5,now it might appear in position six in this sequence but we call this one the zeroth Fibonacci number, since it’s zero, that’s just a convention . Now who cares about the Fibonacci sequence.That **certainly** has interesting **properties** . You can make what’s called the Golden Spiral by **tiling** together squares whose side lengths are Fibonacci numbers. So here’s a square of side 1

And then if you draw a spiral **going through** the intersection points of these squares, you get an ever-expanding spiral that looks particularly well balanced to the human eye. It’s also a spiral that people like to look for in nature . Here is a cabbage where somebody thinks they’ve discovered the golden spiral . olay enough about the Fibonacci sequence. Let’s figure out how to compute it using a while statement

Here’s a function that takes n the position or index in the Fibonacci sequence where we want that Fibonacci number and it computes the end Fibonacci number for n greater than or equal to 1

It does that by keeping track of various values and then executing a while statement, so when you’re designing an iterative function, one of the most important things to think about is what information you need to **keep track of** **in order to** perform the iteration, in this case, in order to compute that the next Fibonacci number is the sum of the previous and the **current** one we need to keep track of what those Fibonacci numbers are, so we start out at the beginning of the sequence the 0th and first Fibonacci numbers are 0 and 1, and then we also need to keep track of where we are in the sequence, so that’s why how we use k here.

K **keeps track of** the index and throughout the execution of this while statement k will be telling us what Fibonacci number is bound to the name curve for current, now the current Fibonacci number at the moment is the first one, this on, 0th first, but **as we execute this** while statement k will change and curr will change and k will tell us which Fibonacci number is curr, okay this says k less than n . We’re going to rebind pred and curr to be the next numbers in the sequence.

So the predecessor is bound to the current one and a new current value is computed

And here we use the definition of the Fibonacci sequence, the next Fibonacci number is the sum of the current one and its predecessor and we bind that to be the new current Fibonacci number which has an index one larger than before, so that’s why we bind k to be k + 1.

Now the structure of the while statement allows us to perform this computation many times until we found the n Fibonacci number, so we keep doing it until k is less than n and that means at the end k equals n , because we’re just adding 1 to k each time

So we found the nth Fibonacci number we’ve bound it to occur and we return it.

In the environment diagram for this example, we’re keeping track of the names pred, curr, n and k in a local frame for calling the fib function and never changes, so if we’re trying to find the fifth in Fibonacci number which is five then n would be five at the beginning pred and cur would be 0, and 1, and k would be one, now we’ll track which values are bound to these names every time we finish executing k equals k plus 1, so the first time through will rebind pred and curr to be the next two numbers in the sequence and say now curr points to the case Fibonacci number where that is the second Fibonacci number, there it is and then we execute the body of the while statement again marching pred and curl up the sequence and changing k at the same time , we’re going to do that four times now k equals n. So it’s not the case the k is less than n we’re finished executing the while statement and we return curr which is the fifth Fibonacci number five . Here’s a discussion question: what if in the body of fib, I had made the following change, I had bound pred , curr to 1 and 0, and k equals 0.

Is this alternative definition of fib the same are different from the original one , pause the video and think about that for a moment then we’ll discuss together.

This is still a correct implementation of fib for every n greater than or equal to 1, just like our old definition but it’s even better because it can compute the 0th Fibonacci number correctly

So we’ve found a particular input value where this does a better job than the old one, so in particular if n is 0 then curr is 0 which is the zeroth Fibonacci number, by the time we reach the while statement k will be 0 , n will be 0, and so the body will never be executed and will return 0 which is the zeroth Fibonacci number excellent. Our old version didn’t do that , but what about with n equals 5 . Well we start out with curr 0 and then we’re actually going to execute the body of the while statement five times instead of four because we started with k equals zero instead of k equals 1, so curr will be 0, and then 1 1 2 3 5 will still get the fifth Fibonacci number