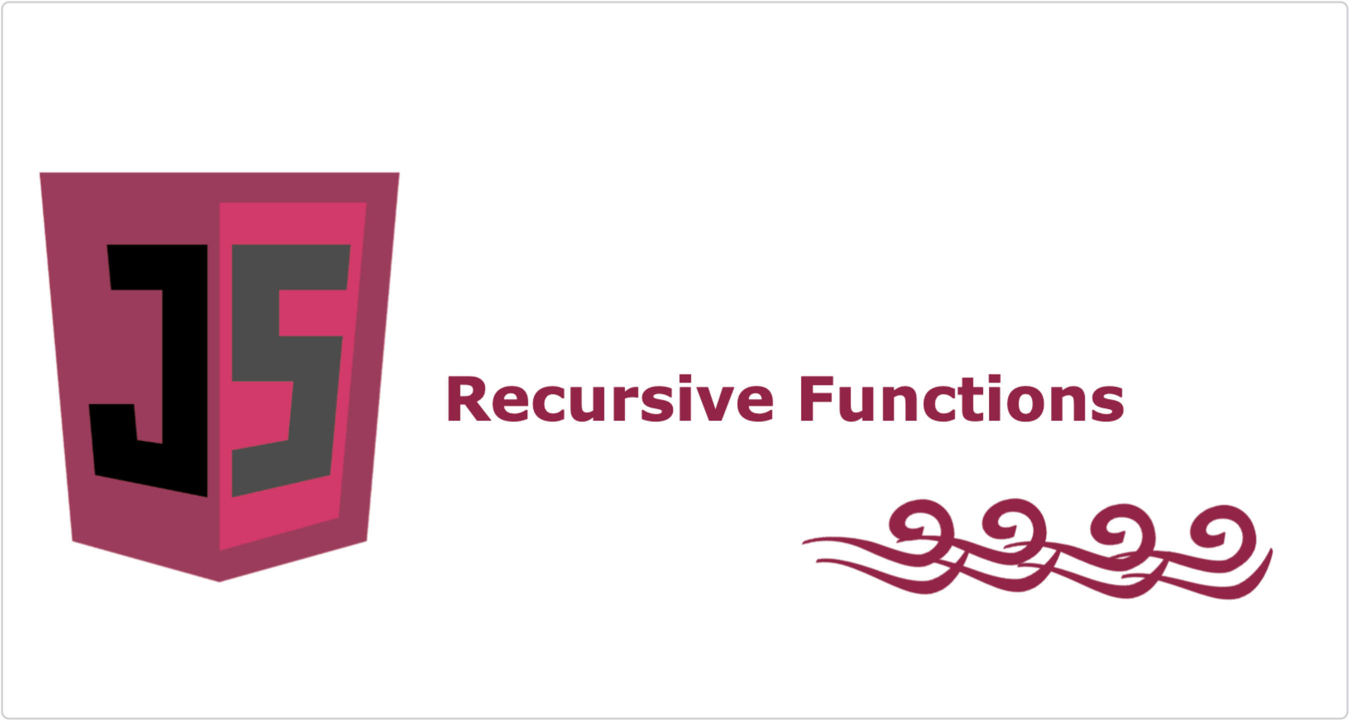
Recursive Functions

In this lesson, we meet a very special type of function called the recursive functions.
Let's begin!

WE'LL COVER THE FOLLOWING



A number of algorithmic problems can be solved with recursion when a function calls itself by name.

What is recursion?

Recursion is a method of function calling in which a function calls itself during execution.

There are problems that are naturally recursively defined. For example, the Fibonacci sequence is defined in a recursive way. By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

```
fibonacci(n) = fibonacci(n-2) * fibonacci(n-1)
```

Parts of recursion

In terms of programming, a recursive function must comprise two parts:

- **Base case**

  A recursive function must contain a base case. This is a condition for the termination of execution.

- **Recursive case**

  The function keeps calling itself again and again until the base case is reached.

Example

The following example computes the factorial of a number using recursion:

```
function fibonacci(index) {
      if (index == 0) return 0;
      if (index == 1) return 1;
      return fibonacci(index - 1) +
        fibonacci(index - 2);
    }

    for (var i = 0; i < 5; i++) {
      console.log(fibonacci(i));
    }
```

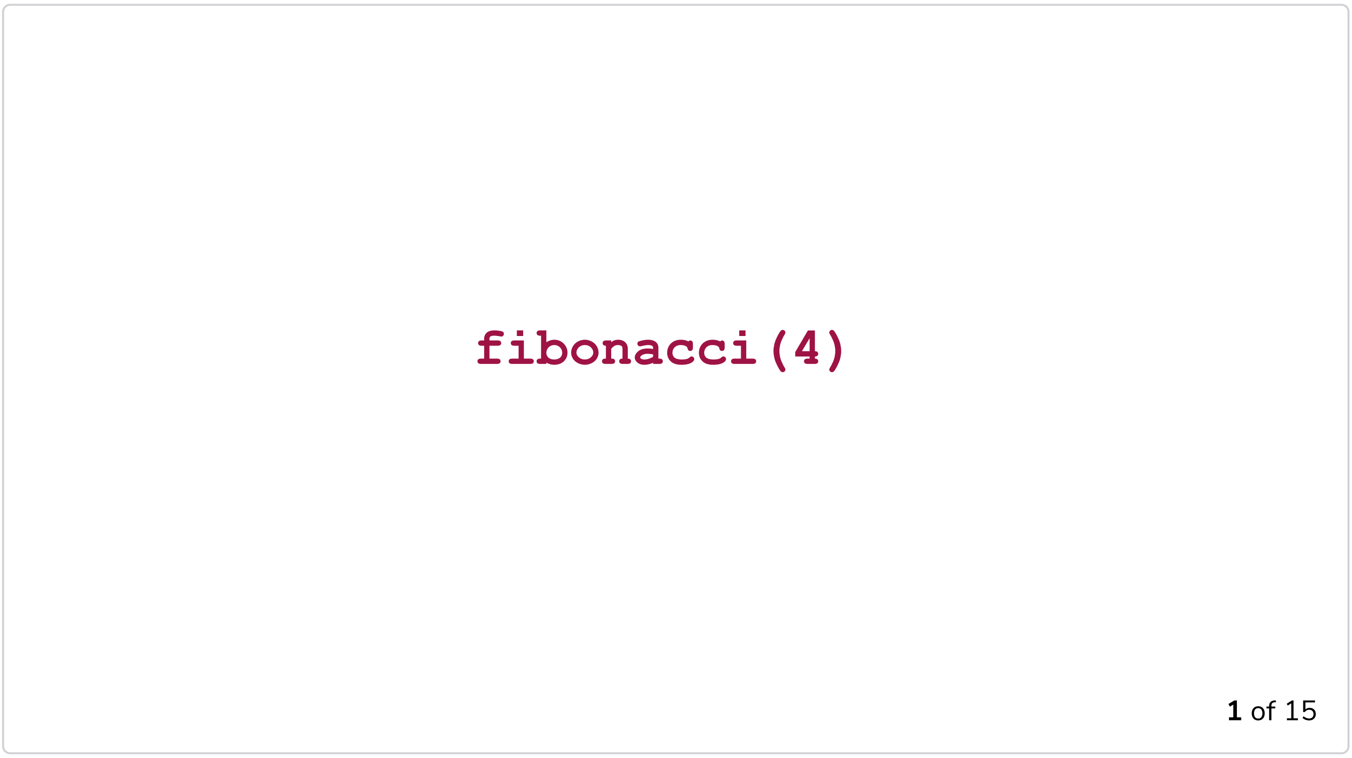
Explanation

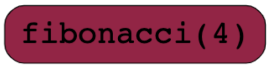
A recursive function, `fibonacci`, takes a **parameter `index` of type integer** and **returns** an integer, i.e., the nth term of the Fibonacci number.

The recursive function has two parts: the base case and the recursive case.

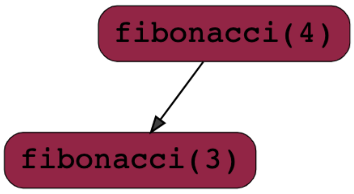
- **base case**
  - **lines two and three** take the `index` value and if it matches with 0, it returns 0; and if it matches with 1, returns 1
- **recursive case**
  - It decrements the value of the `index` by one. A recursive call is made with argument `index - 1` or `index - 2` and the function execution can't proceed until the recursive calls return.
  - **Line four** makes a recursive call with the argument `index - 1` and the result is computed prior to the `+` operator.
  - **Line five** makes a recursive call with the argument `index - 2` and the result is computed after the `+` operator.
- At the end, the result of both recursive calls is summed to get the actual output.

The following illustration explains the code through a recursion tree:

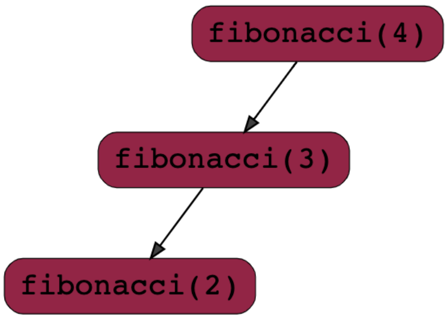



fibonacci(4)

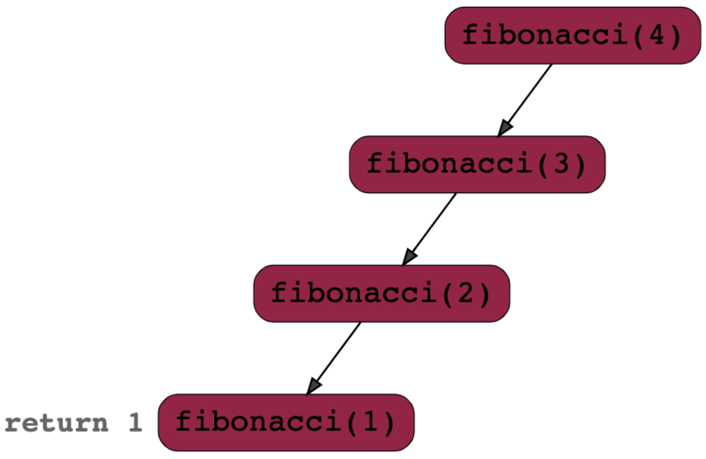
2 of 15


fibonacci(4)
fibonacci(3)

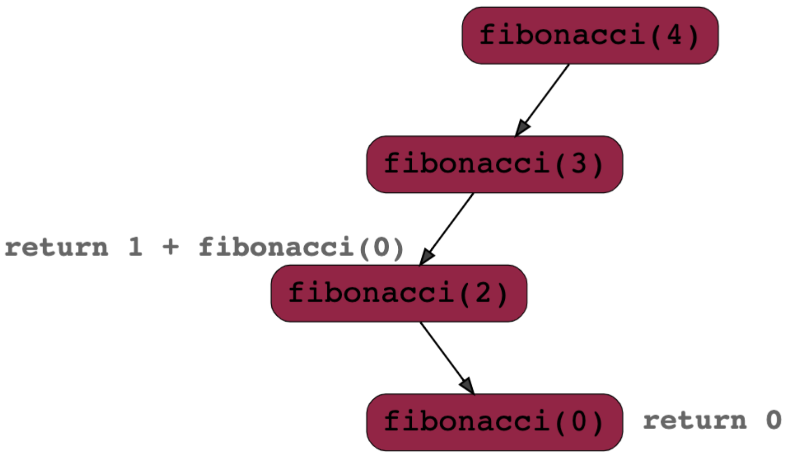
3 of 15


fibonacci(4)
fibonacci(3)
fibonacci(2)

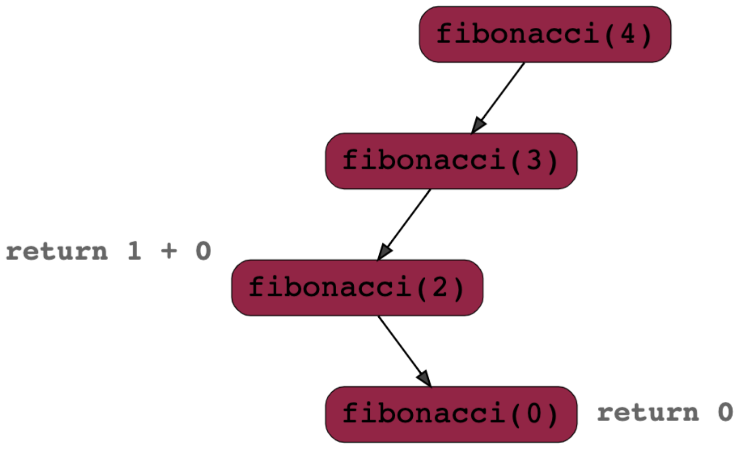
4 of 15



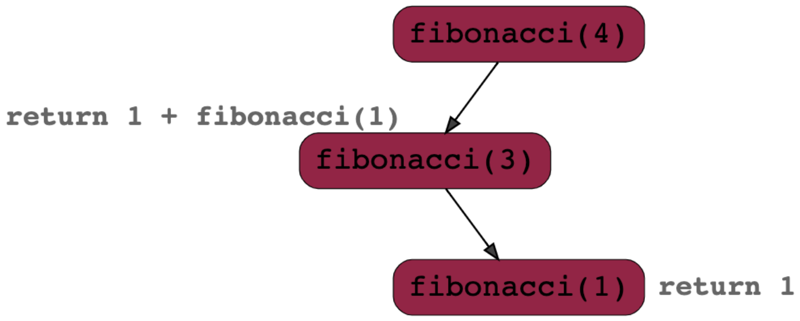
5 of 15



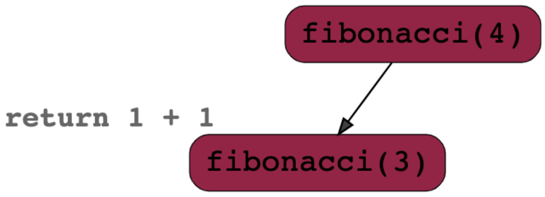
6 of 15

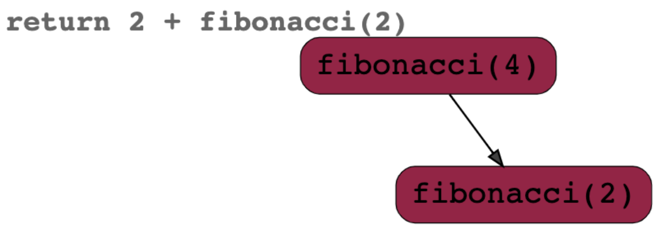

fibonacci(4)
fibonacci(3)
return 1 + 0
fibonacci(2)
fibonacci(0)
return 0

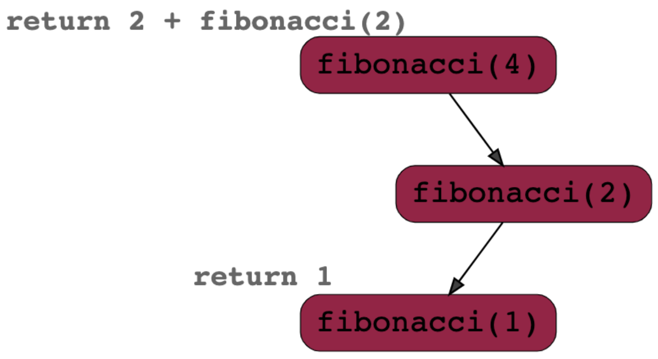
7 of 15

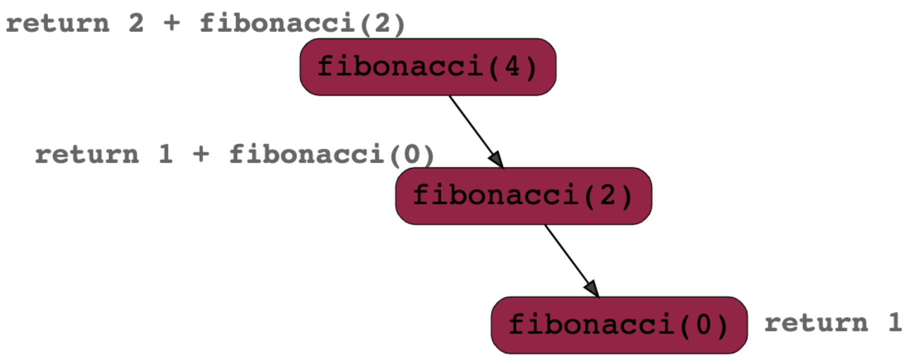

fibonacci(4)
return 1 + fibonacci(1)
fibonacci(3)
fibonacci(1)
return 1

8 of 15

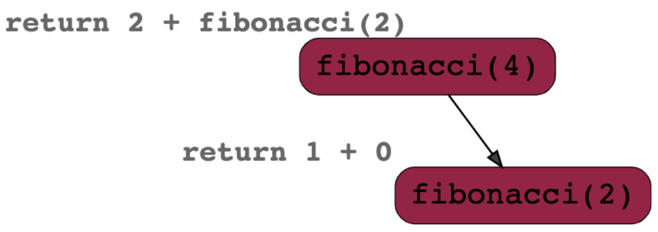

fibonacci(4)
return 1 + 1
fibonacci(3)


return 2 + fibonacci(2)
fibonacci(4)
fibonacci(2)

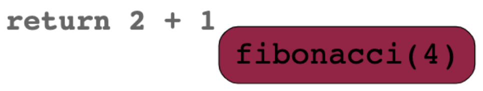

return 2 + fibonacci(2)
fibonacci(4)
fibonacci(2)
return 1
fibonacci(1)



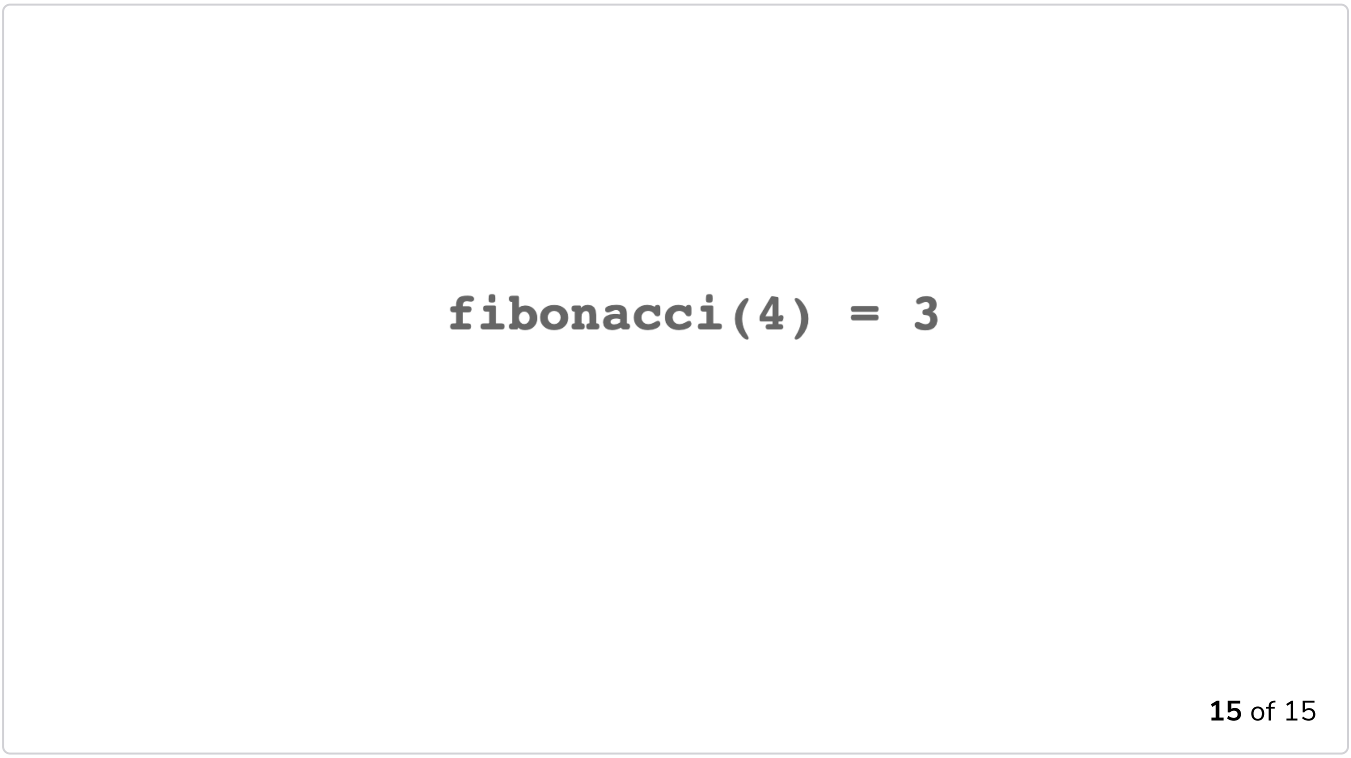
12 of 15



13 of 15



14 of 15


```
fibonacci(4) = 3
```

15 of 15

You can create a recursive function that produces the specified number of this sequence by its index, as shown in Listing 8-5.

Listing 8-5: Creating a recursive function

```
<!DOCTYPE html>
<html>
<head>
  <title>Recursive functions</title>
  <script>
    function fibonacci(index) {
      if (index == 0) return 0;
      if (index == 1) return 1;
      return fibonacci(index - 1) +
        fibonacci(index - 2);
    }

    for (var i = 0; i < 10; i++) {
      console.log(fibonacci(i));
    }
  </script>
</head>
<body>
  Listing 8-5: View the console output
</body>
</html>
```

This short code creates the 10 numbers in the Fibonacci sequence, as the output indicates. I removed line breaks and used commas to separate the numbers, there might be a (2) next to the 1 printed on console to indicate two 1s:

console

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

To describe this recursive function with a function expression, you need to provide a name for the function, otherwise, you cannot invoke it. Listing 8-6 shows the Fibonacci sequence defined with a function expression.

Listing 8-6: Recursive Fibonacci

```
<!DOCTYPE html>
<html>
<head>
  <title>Recursive function expression</title>
  <script>
    var fiboSeq =
    function fibo(index) {
      if (index == 0) return 0;
      if (index == 1) return 1;
      return fibo(index - 1) +
        fibo(index - 2);
    }

    for (var i = 0; i < 10; i++) {
      console.log(fiboSeq(i));
    }
  </script>
</head>
<body>
  Listing 8-6: View the console output
</body>
</html>
```

There is another way to use recursion with function expressions. The arguments object has a callee property, which is a pointer to the function itself.

Using `arguments.callee`, you can define the Fibonacci sequence like this:

```
var fiboSeq =
function (index) {

  if (index == 0) return 0;
  if (index == 1) return 1;
  return arguments.callee(index - 1) +
    arguments.callee(index - 2);
}
console.log(fiboSeq(4))
```

The value of `arguments.callee` is not accessible to a script running in strict mode and will cause an error.

In the *next lesson*, we will learn all about closures in JavaScript.