Week 10 WDD330

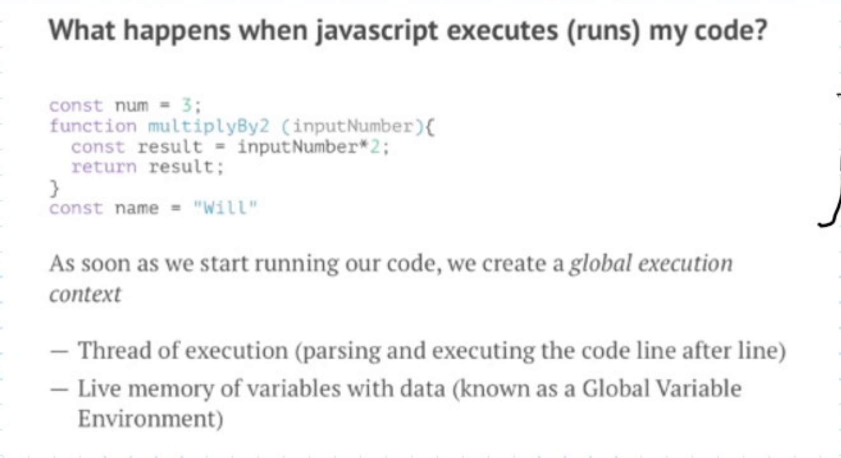
martes, 20 de junio de 2023 14:25

[illegible]

Functions, Execution Context and the Call Stack

From <https://www.youtube.com/watch?v=exrc_rLj5iw&ab_channel=Codesmith>

WHEN JS EXECUTES MY CODE:
- PARSES THROUGH LINE BY LINE
- STORE STUFF IN MEMORY (GLOBAL)
-



What happens when javascript executes (runs) my code?

```
const num = 3;
function multiplyBy2 (inputNumber){
  const result = inputNumber*2;
  return result;
}
const name = "Will"
```

As soon as we start running our code, we create a *global execution context*

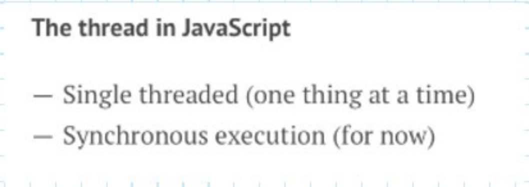
— Thread of execution (parsing and executing the code line after line)
— Live memory of variables with data (known as a Global Variable Environment)

① STORE IN MEMORY ALL VARIABLES
- NUM = 3
- MULTIPLYBY2 [F]
- NAME = "WILL"

DOESN'T GO INSIDE THE FUNCTION UNTIL IT'S CALLED

- EXECUTION CONTEXT = THE SPACE WHERE WE EXECUTES CODE AND WHILE GOING LINE BY LINE, STORES STUFF IN MEMORY

- AS SOON AS WE WRITE OUR CODE WE CREATE A GLOBAL EXECUTION CONTEXT



The thread in JavaScript

— Single threaded (one thing at a time)
— Synchronous execution (for now)

RUNNING/CALLING/INVOKING A FUNCTION

- IS NOT THE SAME AS DEFINING
- WHEN YOU EXECUTE A FUNC YOU CREATE A NEW EXECUTION CONTEXT:
  ① IT GOES LINE BY LINE THROUGH THE FUNCTION
  ② STORE IN LOCAL MEMORY THE LINES OF THE FUNC. (VARIABLE ENVIRONMENT)

IF WE CALL

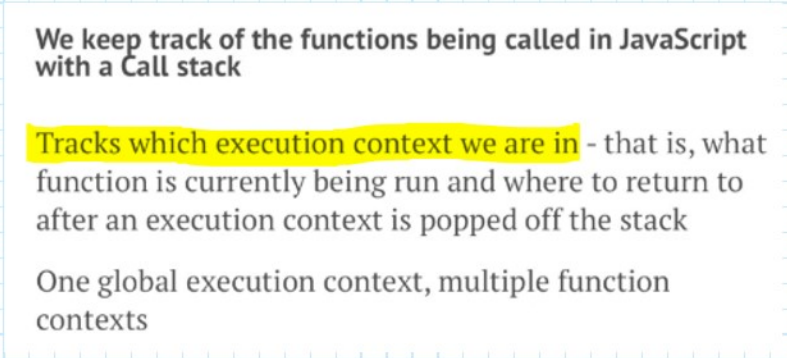
```
CONST OUTPUT = MULTIPLYBY2(4);
```

- BY DEFAULT, OUTPUT WILL HAVE UNDEFINED VALUE UNTIL THE FUNCTION RETURN ITS RESULT.
- THE FUNCTION WILL BE EXECUTED IN A LOCAL EXECUTION CONTEXT
- IT WILL GO LINE BY LINE AND STORE STUFF IN $\underbrace{\text{LOCAL MEMORY}}$ ↓

- INPUT NUM PARAM = 4
(EXECUTES MULTIPLICATION) - RESULT = 8
(RETURN THE VALUE) - RETURN 8

- THEN OUTPUT WILL ADQUIRE THE VALUE OF 8

JS HAS A TRACK WHICH EXECUTION CONTEXT IT'S IN, WHERE IT WAS AN WHERE TO GO BACK TO. THIS IS CALLED CALL STACK

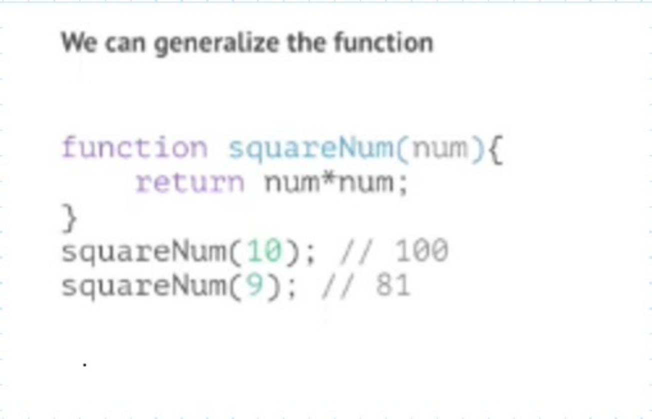

We keep track of the functions being called in JavaScript with a Call stack

Tracks which execution context we are in - that is, what function is currently being run and where to return to after an execution context is popped off the stack

One global execution context, multiple function contexts

JavaScript the Hard Parts: How to Understand Callbacks & Higher Order Functions

From <https://www.youtube.com/watch?v=viQz4nUUnpw&ab_channel=Codesmith>

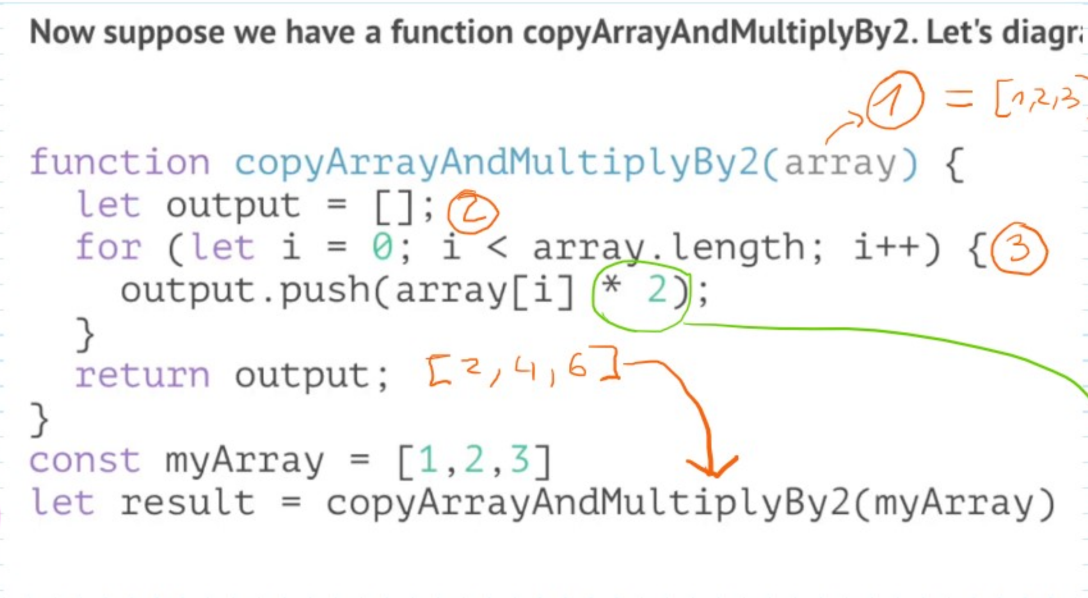

We can generalize the function

```
function squareNum(num){
    return num*num;
}
squareNum(10); // 100
squareNum(9); // 81
```

- THIS FUNCTION IS REHUSABLE
- WE CAN APPLY OUR MULTIP. FUNCTIONALITY TO WHEN WE RUN OUR FUNCTION, NOT WHEN WE DEFINE IT

• HIGHER ORDER FUNCTIONS FOLLOW THE SAME PRINCIPLE

GlobAl



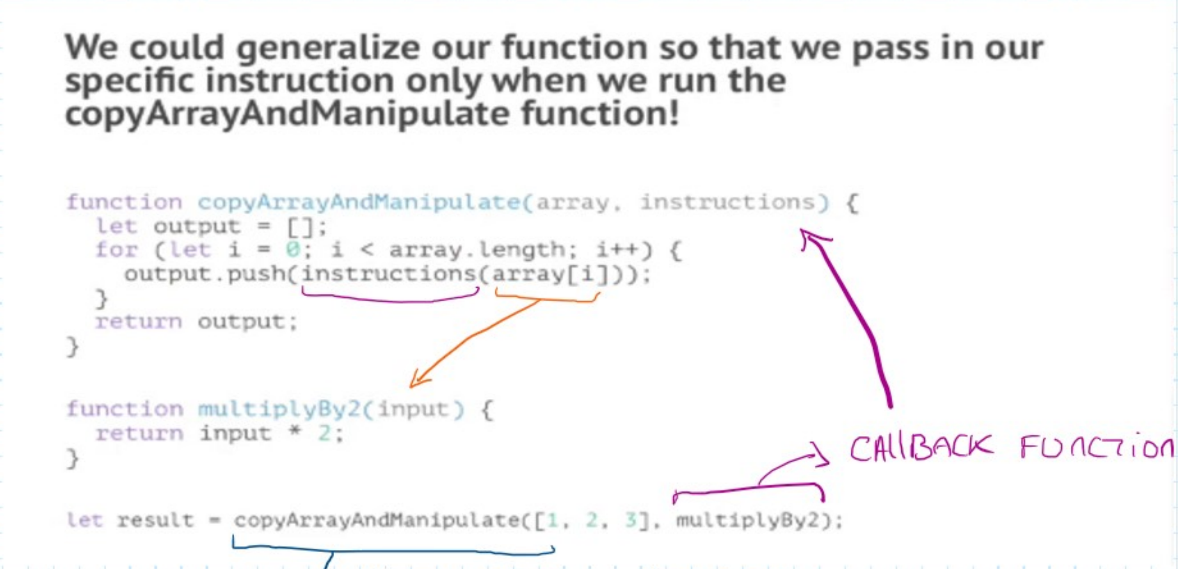
Now suppose we have a function copyArrayAndMultiplyBy2. Let's diagr

$\boxed{1} = [1,2,3]$

```
(1) function copyArrayAndMultiplyBy2(array) {
      let output = []; (2)
      for (let i = 0; i < array.length; i++) { (3)
        output.push(array[i] * 2);
      }
      return output;  [2,4,6]
    }
(2) const myArray = [1,2,3]
(3) let result = copyArrayAndMultiplyBy2(myArray)
```

(4) RESULT = [2,4,6]

IF WE NEED MORE OPERATIONS, WE CAN generalize THE FUNCTION TO BECOME MORE REUSABLE AND NOT REPEAT



We could generalize our function so that we pass in our specific instruction only when we run the copyArrayAndManipulate function!

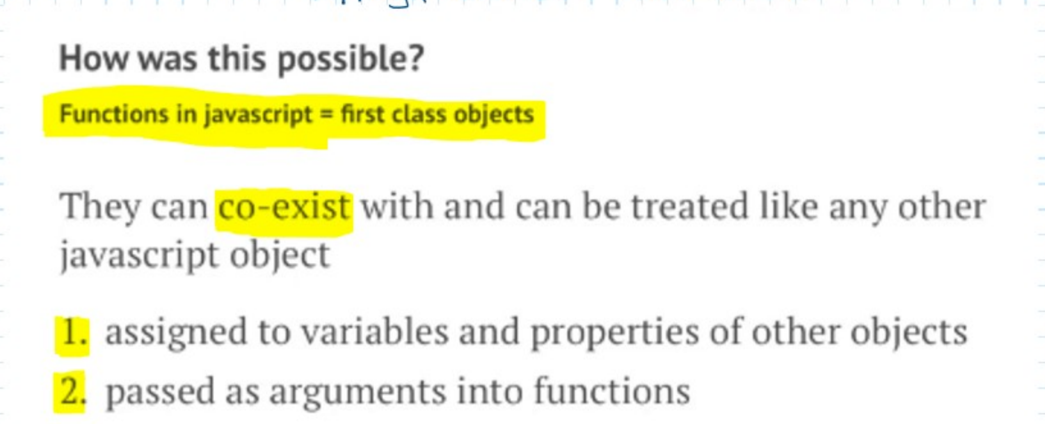
```
function copyArrayAndManipulate(array, instructions) {
  let output = [];
  for (let i = 0; i < array.length; i++) {
    output.push(instructions(array[i]));
  }
  return output;
}

function multiplyBy2(input) {
  return input * 2;
}

let result = copyArrayAndManipulate([1, 2, 3], multiplyBy2);
```

CALLBACK FUNCTION

HIGH ORDER FUNCTION



How was this possible?

Functions in javascript = first class objects

They can co-exist with and can be treated like any other javascript object

1. assigned to variables and properties of other objects
2. passed as arguments into functions

1. assigned to variables and properties of other objects
2. passed as arguments into functions
3. returned as values from functions

- I CAN'T DECLARE AN OBJECT AND THEN INVOKE IT
  BUT FUNCTIONS CAN (EVEN IF THEY ARE FUNCTIONS)

High - ORDER FUNCTIONS

- TAKES IN A FUNCTION OR PASSES OUT A FUNCTION
- JUST A TERM TO DESCRIBE THIS FUNCTIONS (BUT THEY WORK THE SAME AS OTHER FUNCTIONS)

CALLBACKS AND HIGHER ORDER FUNCTIONS SIMPLIFY OUR CODE AND KEEP IT **DRY**.

THEY ALLOW US TO RUN ASYNCHRONOUS CODE.