Welcome to CoGrammar WD Tutorial 3

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: [www.hyperiondev.com/support]
- Report a **safeguarding** incident: [www.hyperiondev.com/safeguardreporting]
- We would love your **feedback** on lectures: **Feedback on Lectures**

Skills Bootcamp 8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

Functions and Functional Programming

March 2024

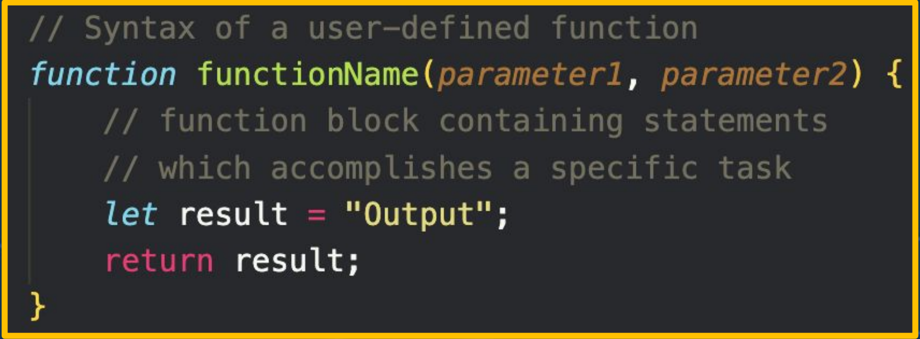
Lecture Overview

- → Functions
- → Scope
- → Hoisting
- → Higher Order Functions
- → Callback Functions
- → Function Composition

CoGrammar

Functions

- To declare a function in JavaScript, we use the **function** keyword.
- We have to provide a **name** for our function (using variable naming conventions), a list of **parameters** (placeholders for function inputs) in brackets and the **body** of the function in curly brackets
- We also need to add a **return statement** for functions that return a value. This is not necessary for all functions e.g. functions that modify a state.

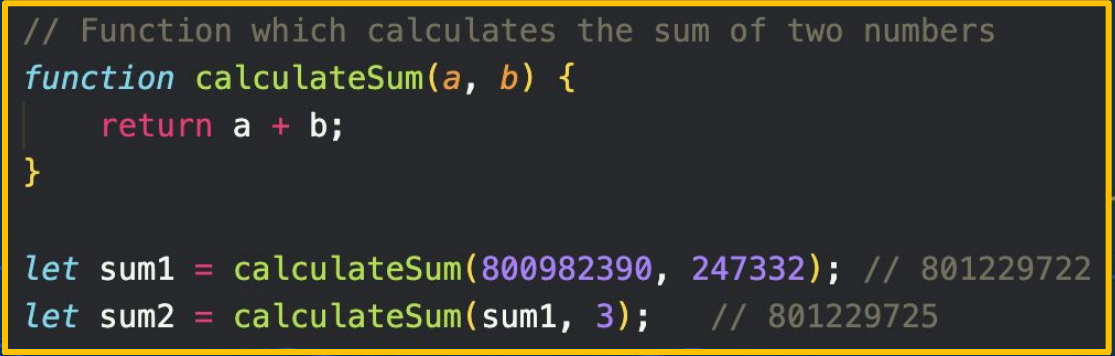


```
// Syntax of a user-defined function
function functionName(parameter1, parameter2) {
    // function block containing statements
    // which accomplishes a specific task
    let result = "Output";
    return result;
}
```

CoGrammar

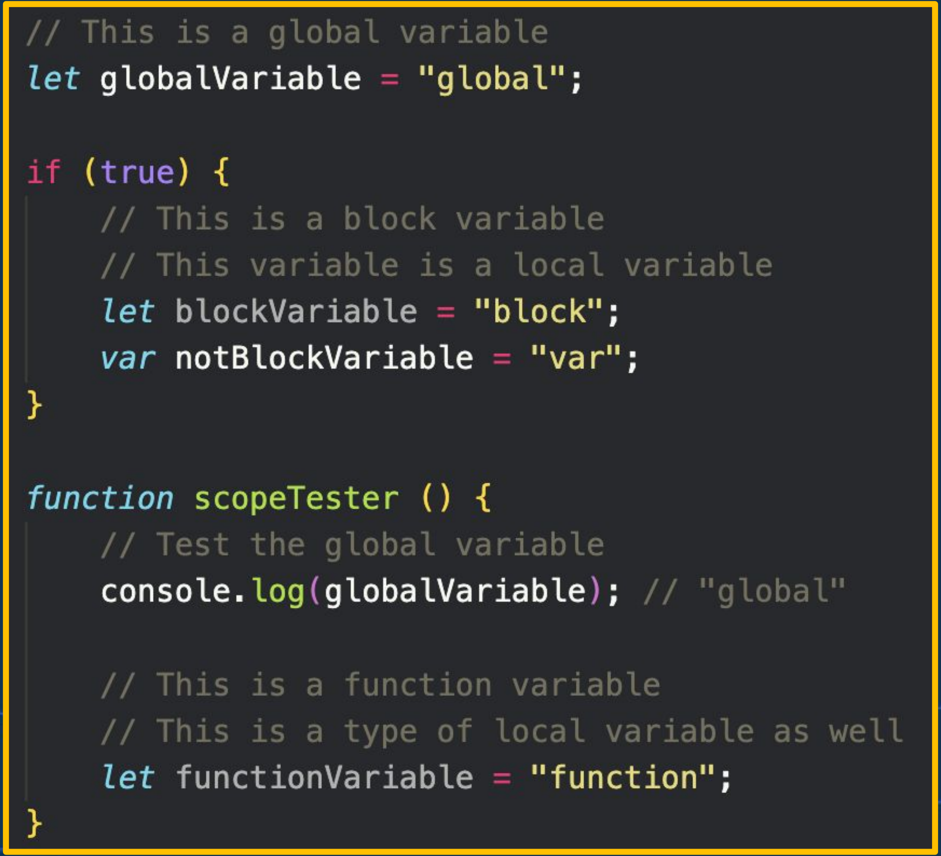
Functions

- After defining a function, we **call or invoke** it to use it in our code.
- We call a function with its name followed by a list of **arguments** enclosed in brackets, if required by the functions.
- **Arguments** are the input values provided to the function and take the place of the **parameters** defined in the function in the **same position**.



```
// Function which calculates the sum of two numbers
function calculateSum(a, b) {
    return a + b;
}

let sum1 = calculateSum(800982390, 247332); // 801229722
let sum2 = calculateSum(sum1, 3);    // 801229725
```

Scope



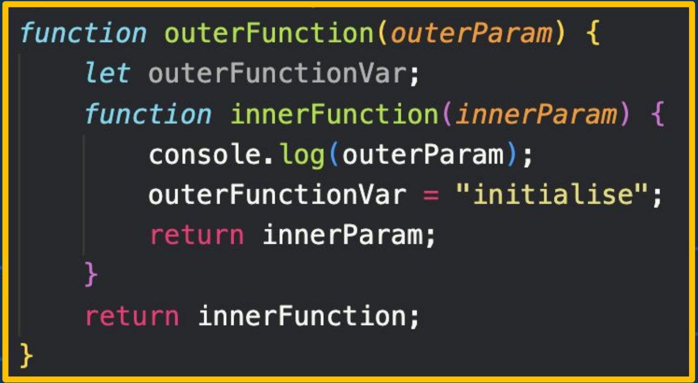
```javascript
// This is a global variable
let globalVariable = "global";

if (true) {
    // This is a block variable
    // This variable is a local variable
    let blockVariable = "block";
    var notBlockVariable = "var";
}

function scopeTester () {
    // Test the global variable
    console.log(globalVariable); // "global"

    // This is a function variable
    // This is a type of local variable as well
    let functionVariable = "function";
}
```

CoGrammar

Nested Functions

A function that is defined inside another function.

- The **nested function** is referred to as the **inner function** and the **containing function** is known as the **outer function**.
- Nested functions can only be called **within the containing function.**
- A nested function forms a **closure**, the function has its **own local variables and parameters** and is able to reference and use its containing **function's function variables and parameters**.



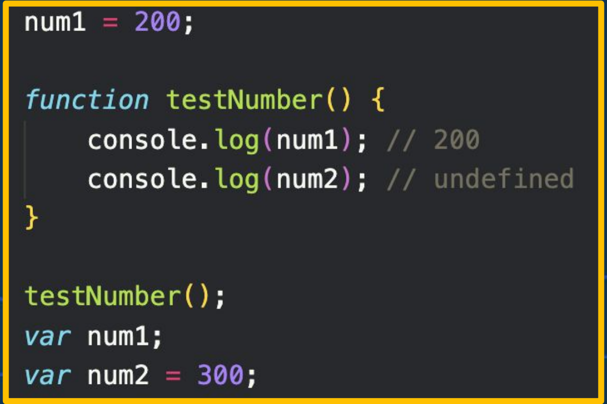
```
function outerFunction(outerParam) {
    let outerFunctionVar;
    function innerFunction(innerParam) {
        console.log(outerParam);
        outerFunctionVar = "initialise";
        return innerParam;
    }
    return innerFunction;
}
```

CoGrammar

Hoisting

A JavaScript mechanism where variable, function and class declarations are moved to the top of their scope, during the compilation phase.

- ❖ This process allows us to **access** variables **before they are declared**, without any errors preventing our code from running.
- ❖ It also allows us to **declare variables after we initialise** and use them.
- ❖ Only the **variable declaration** is moved, **not the initial binding**.



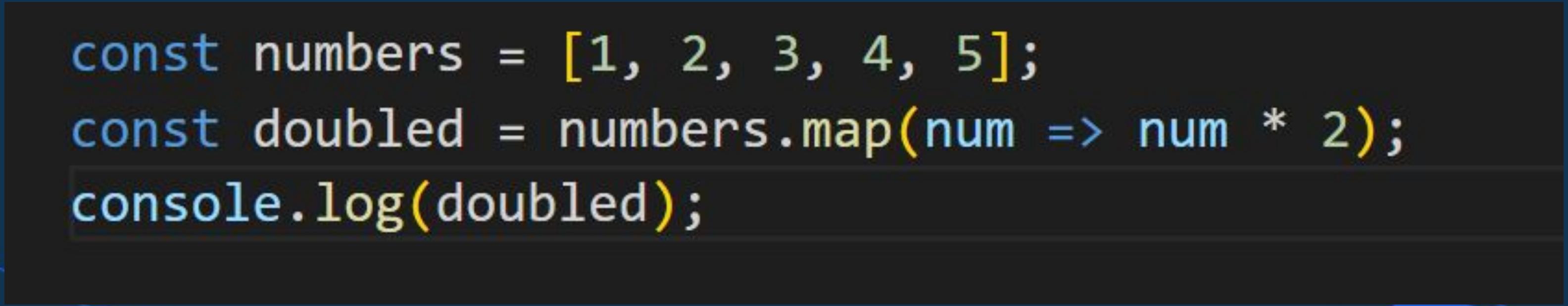
```
num1 = 200;

function testNumber() {
    console.log(num1); // 200
    console.log(num2); // undefined
}

testNumber();
var num1;
var num2 = 300;
```

CoGrammar

Higher Order Functions

Higher order functions are functions that can accept other functions as arguments or return functions as results.

- The **map()** function **applies** a provided function to each element of an array and returns a new array with the results.



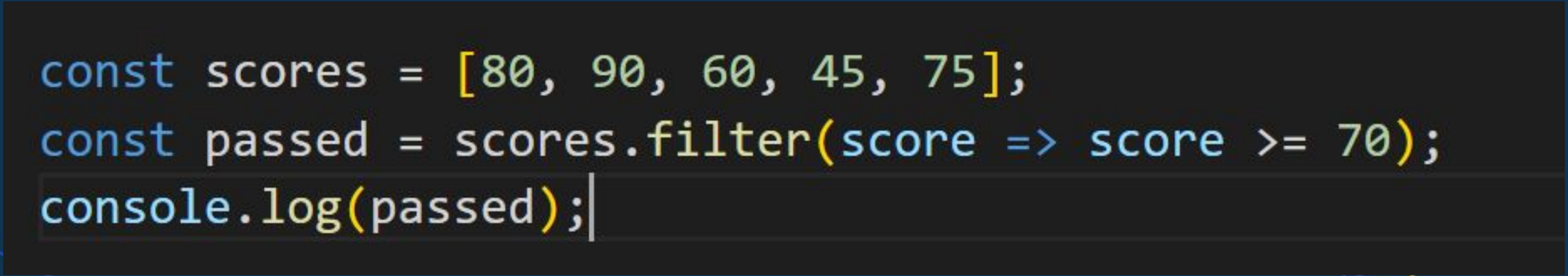
```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map(num => num * 2);
console.log(doubled);
```

CoGrammar

Higher Order Functions

Higher order functions are functions that can accept other functions as arguments or return functions as results.

❖ The **filter()** function creates a new array with all elements that pass the test implemented by the provided function.



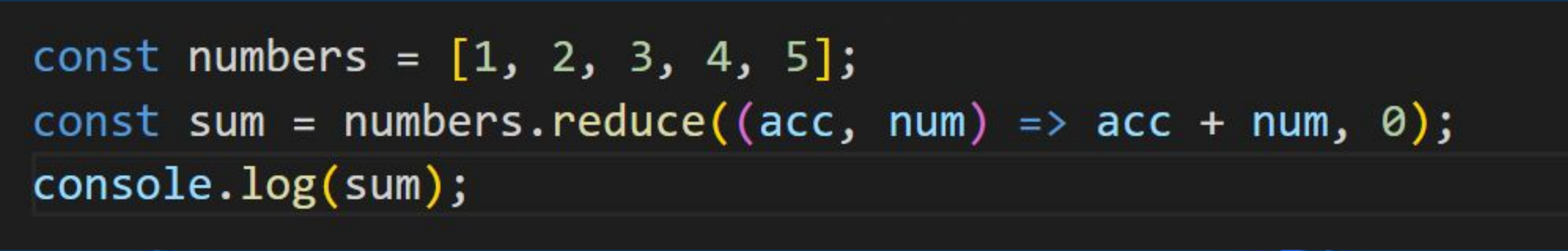
```
const scores = [80, 90, 60, 45, 75];
const passed = scores.filter(score => score >= 70);
console.log(passed);
```

CoGrammar

Higher Order Functions

Higher order functions are functions that can accept other functions as arguments or return functions as results.

❖ The **reduce()** function executes a **reducer** function on each element of the array, resulting in a single output value.

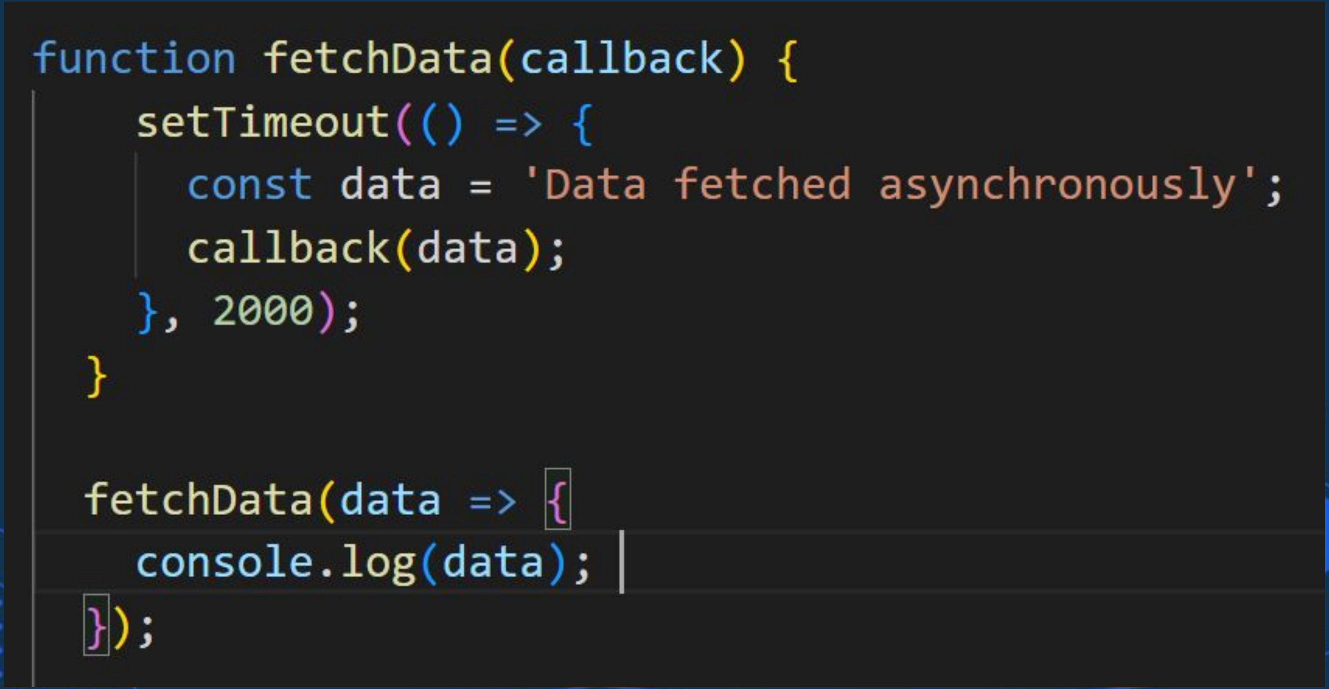


```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum);
```

CoGrammar

Callback Functions

Callback functions are functions passed as arguments to other functions and executed later.

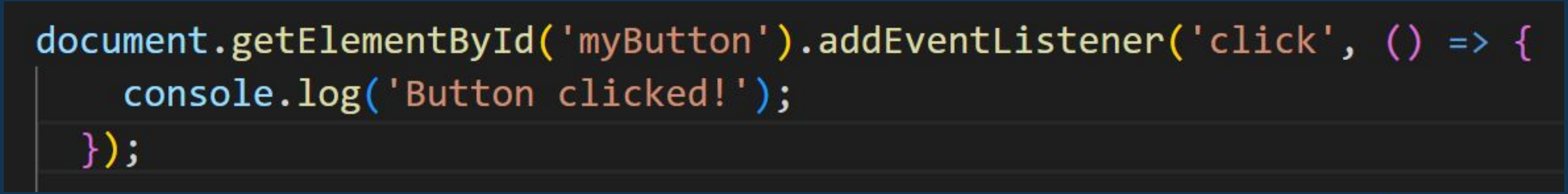


```
function fetchData(callback) {
    setTimeout(() => {
      const data = 'Data fetched asynchronously';
      callback(data);
    }, 2000);
  }

  fetchData(data => {
    console.log(data);
  });
```

CoGrammar

Callback Functions

Callback functions are functions passed as arguments to other functions and executed later.



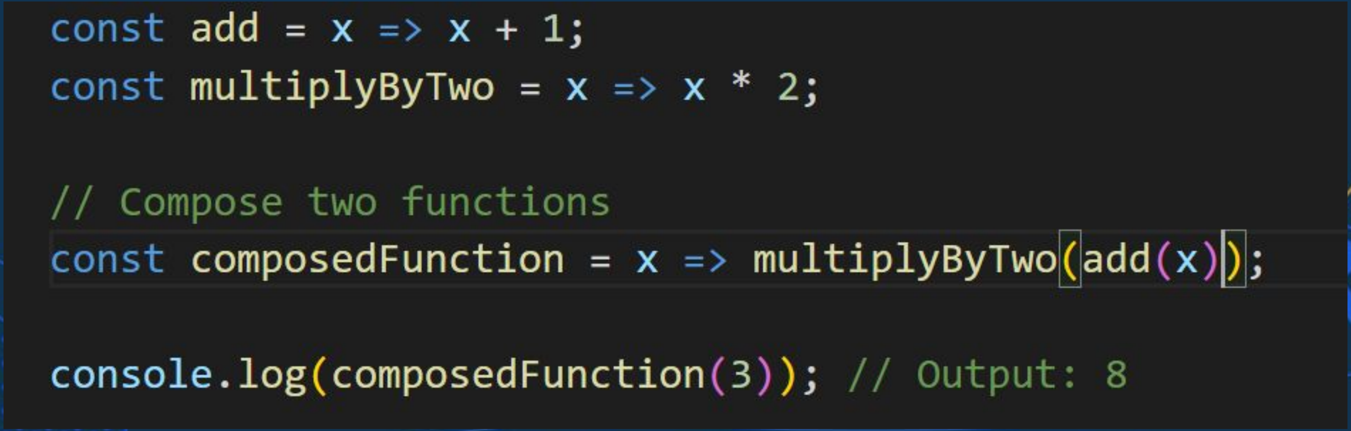
```
document.getElementById('myButton').addEventListener('click', () => {
    console.log('Button clicked!');
});
```

CoGrammar

Function Composition

Function composition is a technique used to combine multiple functions to create a new function.

- It involves **chaining** functions together, where the **output** of one function becomes the **input** of the **next**.
- Function **composition** enhances code modularity and readability by breaking down complex operations into smaller, composable units.



```
const add = x => x + 1;
const multiplyByTwo = x => x * 2;

// Compose two functions
const composedFunction = x => multiplyByTwo(add(x));

console.log(composedFunction(3)); // Output: 8
```

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar