

JavaScript



Book 5

ES6 and Beyond



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

Table of Contents

| | |
|--|----|
| Introduction to JavaScript: From Basics to Advanced..... | 4 |
| Why Learn JavaScript?..... | 4 |
| What Will You Learn in This Book?..... | 4 |
| 1. Getting Started with the Basics..... | 4 |
| 2. Core Concepts..... | 5 |
| 3. Intermediate Features..... | 5 |
| 4. Advanced Topics..... | 5 |
| 5. Real-world Applications..... | 6 |
| How This Book Works..... | 6 |
| Who Is This Book For?..... | 7 |
| Let's Get Started!..... | 7 |
| Introduction to Javascript..... | 8 |
| Why Learn JavaScript?..... | 8 |
| Hello World in JavaScript..... | 9 |
| Explanation:..... | 10 |
| JavaScript Variables and Data Types..... | 10 |
| JavaScript Variables..... | 10 |
| Declaring Variables..... | 11 |
| Variable Rules..... | 11 |
| Summary:..... | 13 |
| Variable Assignment and Reassignment..... | 14 |
| Task:..... | 14 |
| Data Types:..... | 14 |
| String (string):..... | 14 |
| 2. Number (number):..... | 15 |
| 3. Boolean (boolean):..... | 15 |
| 4. Undefined (undefined):..... | 16 |
| 5. Null (null):..... | 16 |
| Objects and Arrays (Complex Data Types)..... | 17 |
| Object (object):..... | 17 |
| Array (array):..... | 18 |
| Summary..... | 19 |
| Javascript Operators..... | 19 |
| 1. Arithmetic Operators..... | 20 |
| 2. Assignment Operators..... | 20 |
| 3. Comparison Operators..... | 21 |
| 4. Logical Operators..... | 21 |
| Control Statements in JavaScript..... | 22 |
| 1. Conditional Statements..... | 22 |
| if Statement..... | 22 |
| Example..... | 23 |
| if-else Statement..... | 24 |
| else if Statement..... | 25 |
| switch Statement..... | 27 |
| Summary..... | 28 |
| Task Todo..... | 28 |
| Task:..... | 29 |
| Looping Statements..... | 29 |

| | |
|---|----|
| for Loop..... | 29 |
| Tasks..... | 30 |
| while loop..... | 31 |
| Summary..... | 31 |
| Tasks..... | 32 |
| Javascript Functions..... | 33 |
| Basic Function Without Parameters..... | 33 |
| Function with Parameters..... | 34 |
| Arrow Functions..... | 35 |
| Arrow Function without Parameters:..... | 35 |
| Arrow Function with Parameters:..... | 36 |
| Explanation:..... | 37 |
| Summary:..... | 38 |
| Task ToDo..... | 38 |
| Practice Questions..... | 40 |
| Appendix 1..... | 42 |
| Useful Links..... | 42 |

Introduction to JavaScript: From Basics to Advanced

Welcome to **Book 5** – your comprehensive guide to mastering JavaScript!

Whether you're just starting out with programming or you're looking to level up your JavaScript skills, this book will take you on an exciting journey, covering everything from the foundations of JavaScript to its most advanced features.

Why Learn JavaScript?

JavaScript is the backbone of modern web development. It powers dynamic websites, mobile apps, and even server-side applications through Node.js. Mastering JavaScript means gaining the skills to create interactive web pages, perform asynchronous operations, manipulate data, and build full-fledged applications. Whether you want to work on the front-end with frameworks like React, Vue, or Angular, or dive into back-end development with Node.js, JavaScript is the language that opens countless opportunities.

What Will You Learn in This Book?

This book is designed to take you step by step, progressing from the basics of JavaScript to its most advanced concepts, in a structured way. Here's how we'll guide you:

1. Getting Started with the Basics

- **Syntax:** Learn how to write JavaScript code with proper syntax, variables, and basic operations.
- **Data Types:** Understand how JavaScript handles different types of data (strings, numbers, arrays, objects).

- **Control Flow:** Master loops, conditionals, and basic logic to control how your code behaves.

2. Core Concepts

- **Functions:** Understand how to create and work with functions, including function expressions, parameters, and return values.
- **Objects & Arrays:** Learn about these powerful structures, how to manipulate them, and store data.

3. Intermediate Features

- **Error Handling:** Master how to handle errors in JavaScript to make your applications more robust.
- **Event Handling:** Understand how to interact with user events (like clicks, key presses) to make dynamic applications.

4. Advanced Topics

- **Advanced Functions:** Get to grips with higher-order functions, arrow functions, and functional programming techniques.
- **Performance & Optimization:** Explore how to write efficient, performance JavaScript code, and optimize applications for better user experiences.

5. Real-world Applications

- **Frameworks & Libraries:** Dive into the basics of popular frameworks like React JS.
- **APIs & Web Requests:** Learn how to interact with third-party APIs to create powerful applications.

How This Book Works

This book is structured to take you on a progressive learning path, starting from the ground up. Each chapter builds on the previous one, with clear examples, exercises, and challenges to help you solidify your understanding. You'll find:

- **Practical Examples:** Real-world scenarios to demonstrate concepts.
- **Interactive Exercises:** Engaging challenges at the end of each chapter to practice what you've learned.
- **Key Takeaways:** Quick summaries of the most important concepts to remember.

By the end of this book, you will not only understand JavaScript but be able to write clean, efficient, and sophisticated applications. You'll have the skills to tackle complex problems, debug code effectively, and build amazing things on the web.

Who Is This Book For?

This book is perfect for:

- **Beginners:** If you've never written JavaScript before, don't worry! We'll start from the very basics.
- **Intermediate Developers:** If you already know some JavaScript but want to deepen your knowledge, we cover intermediate and advanced concepts in detail.
- **Aspiring Web Developers:** JavaScript is the language of the web, and learning it is essential if you want to build websites, applications, and APIs.

Let's Get Started!

Ready to take your JavaScript skills to the next level? Let's start by understanding the basic syntax and building your first JavaScript program.

To start working with Javascript we need to install VS Code Editor from

<https://code.visualstudio.com/>

Also install Node JS from <https://nodejs.org/en/learn/getting-started/how-to-install-nodejs>

Introduction to Javascript

JavaScript is one of the most popular and powerful programming languages used in web development. It's primarily used to make websites interactive and dynamic. Originally created to run in web browsers, JavaScript has since evolved into a language that can be used for full-stack development, mobile app development, and even desktop applications.

JavaScript is a **high-level, interpreted** language, meaning that you don't need to compile it, and it is executed directly by the browser or a runtime environment (like Node.js). It can interact with HTML and CSS, making it the heart of modern web development.

Why Learn JavaScript?

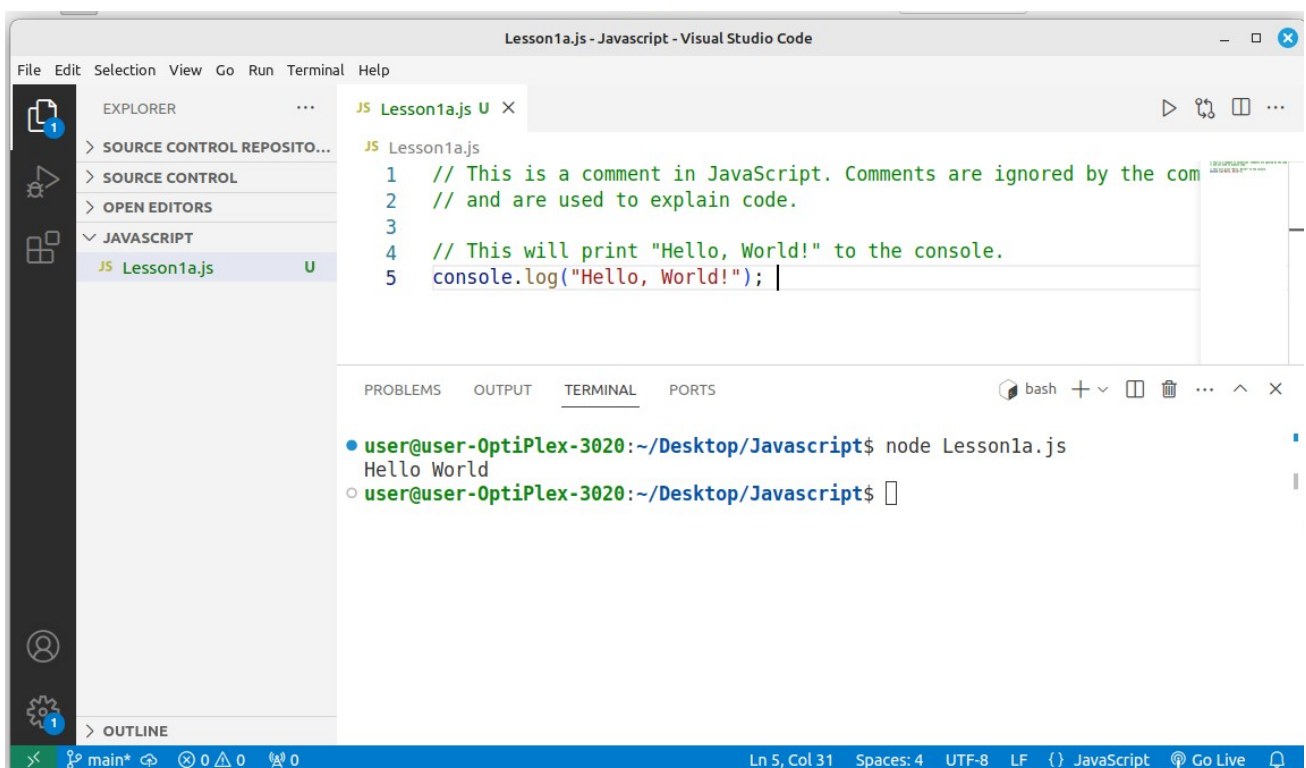
- **Interactivity:** JavaScript allows you to add interactivity to websites, such as handling user input, animations, and updates without reloading the page.
- **Versatility:** JavaScript is used for both **client-side** (in the browser) and **server-side** (with Node.js) development, making it a full-stack language.
- **Widely Used:** JavaScript is the most used programming language on the web, powering interactive websites, web applications, mobile apps, and even games.

Hello World in JavaScript

The "Hello, World!" program is the simplest example of any programming language, and it's a great way to start your journey in coding. It simply prints the message "**Hello, World!**" to the screen.

Here is the **Hello, World!** code in JavaScript:

In VS code Create a File named Lesson1a.py and write below code.



The screenshot shows the Visual Studio Code interface with a file named 'Lesson1a.js' open. The code in the editor is as follows:

```
1 // This is a comment in JavaScript. Comments are ignored by the com
2 // and are used to explain code.
3
4 // This will print "Hello, World!" to the console.
5 console.log("Hello, World!");
```

Below the editor, the TERMINAL panel shows the command 'node Lesson1a.js' being executed, resulting in the output 'Hello World'.

To run Your code use this command in terminal as shown above.

node Lesson1a.js

Explanation:

- `console.log()` is a built-in JavaScript function that outputs whatever is inside the parentheses to the console. The console is a tool that developers use to log messages, check errors, and test small snippets of code.
- "Hello, World!" is a string. In JavaScript, text is enclosed in double or single quotes.
- The `//` syntax is used for writing comments in JavaScript. Everything after `//` on that line is ignored by the program, and it's just there to explain or clarify the code.

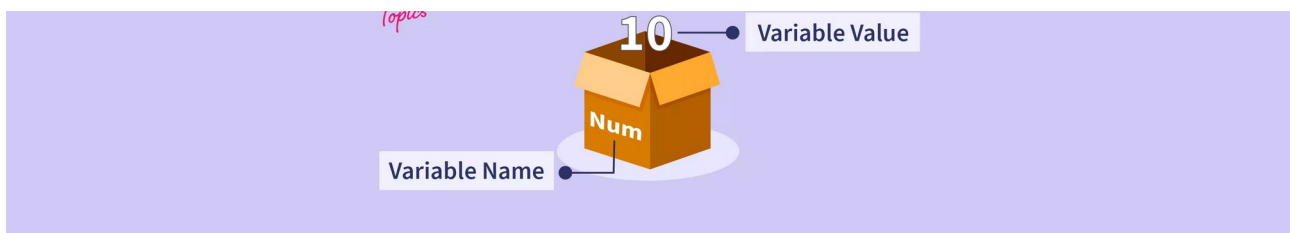
JavaScript Variables and Data Types

In JavaScript, **variables** and **data types** are essential concepts that you'll encounter frequently as you begin writing code. Let's explore both of these concepts in detail.

JavaScript Variables

A **variable** is a container used to store data values. In JavaScript, variables are used to hold various types of data, such as numbers, strings, booleans, and more. Once a variable is declared, you can use it to store, modify, and access the data.

Example



Declaring Variables

You can declare variables using one of the following keywords:

- **var**: An older way to declare variables, typically used in the past. It has function-level scope and can lead to unexpected behavior in modern JavaScript.
- **let**: The modern and preferred way to declare variables. It has block-level scope, which means it is limited to the block (like a function or a loop) where it is declared.
- **const**: Declares a constant value. Once assigned, a const variable cannot be reassigned. However, if the value is an object or array, the contents of the object can still be modified.

Variable Rules

In JavaScript, there are specific rules and best practices for declaring variables. Understanding these rules is important to avoid errors and write efficient code. Here are the key **rules** for declaring a variable in JavaScript:

1. Variable Declaration with **let**, **const**, and **var**:

- **let and const** are used to declare variables with block scope, while **var** is function-scoped.
- **const** is used to declare a constant, i.e., a variable whose value cannot be reassigned after initialization.

2. Variable Names (Identifiers) Rules:

- **Must start with a letter, underscore (_), or dollar sign (\$).**
 - Valid: `let name;`, `let _age;`, `let $price;`
 - Invalid: `let 1name;` (cannot start with a number).
- **Can contain letters, numbers, underscores, or dollar signs.**
 - Valid: `let user1;`, `let _value;`, `let $score;`
- **Cannot use JavaScript reserved keywords (Check Appendix 1) (such as `let`, `class`, `return`, etc.) as variable names.**
 - Invalid: `let for;`, `let function;`
- **Case-sensitive:** JavaScript is case-sensitive, so `age` and `Age` are considered different variables.
 - Example: `let age;` and `let Age;` are two separate variables.

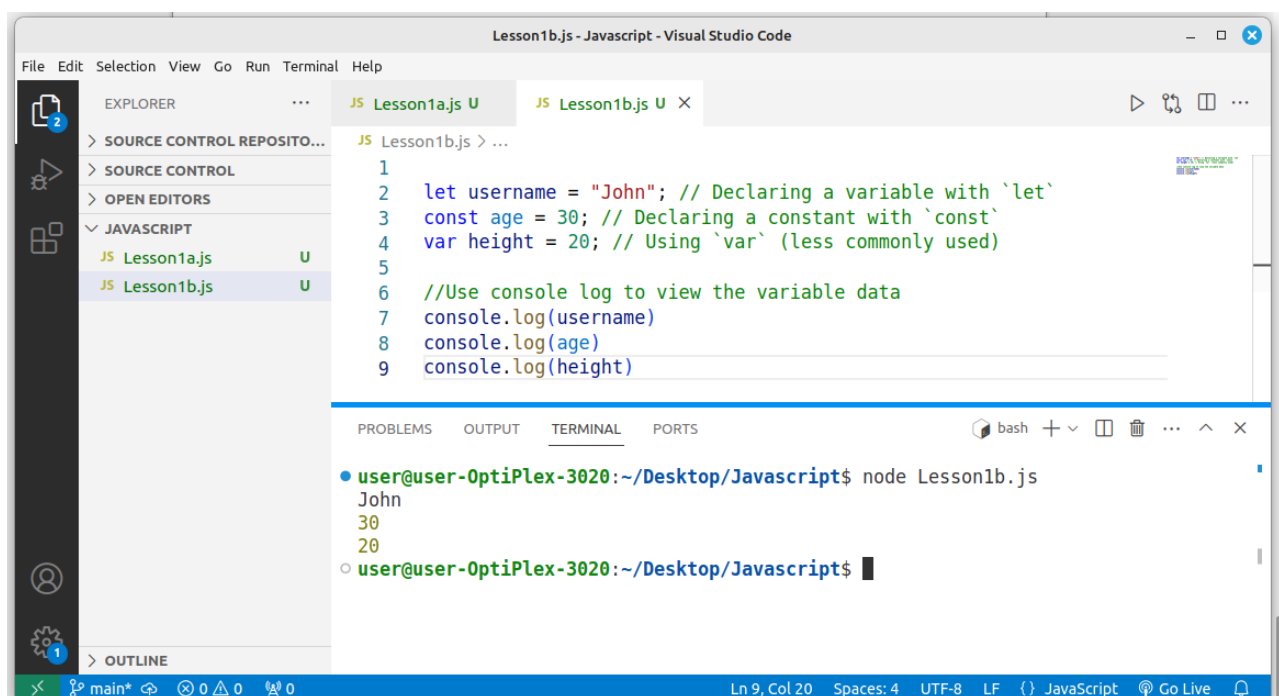
3. Naming Conventions:

- **Camel case** is generally used for variable names in JavaScript, where the first word is lowercase and each subsequent word is capitalized.
 - Example: `let userAge = 25;`
- **Avoid using all uppercase letters** for variable names, as that is usually reserved for constants.
 - Example: `const MAX_LENGTH = 100;`

Summary:

- Variables can be declared with `let`, `const`, or `var`.
- Variable names must start with a letter, underscore, or dollar sign and can contain letters, numbers, underscores, or dollar signs.
- You cannot use JavaScript reserved keywords as variable names.
- `let` and `const` have block-level scope, while `var` has function-level scope.
- Variables declared with `let` and `const` cannot be redeclared in the same scope.
- `const` must be initialized at the time of declaration.
- JavaScript variables are case-sensitive.

Example of declaring variables - **Lesson1b.js**



The screenshot displays the Visual Studio Code editor with a file named `Lesson1b.js` open. The Explorer sidebar on the left shows the file structure, including `Lesson1a.js` and `Lesson1b.js`. The main editor area shows the following JavaScript code:

```
1
2 let username = "John"; // Declaring a variable with `let`
3 const age = 30; // Declaring a constant with `const`
4 var height = 20; // Using `var` (less commonly used)
5
6 //Use console log to view the variable data
7 console.log(username)
8 console.log(age)
9 console.log(height)
```

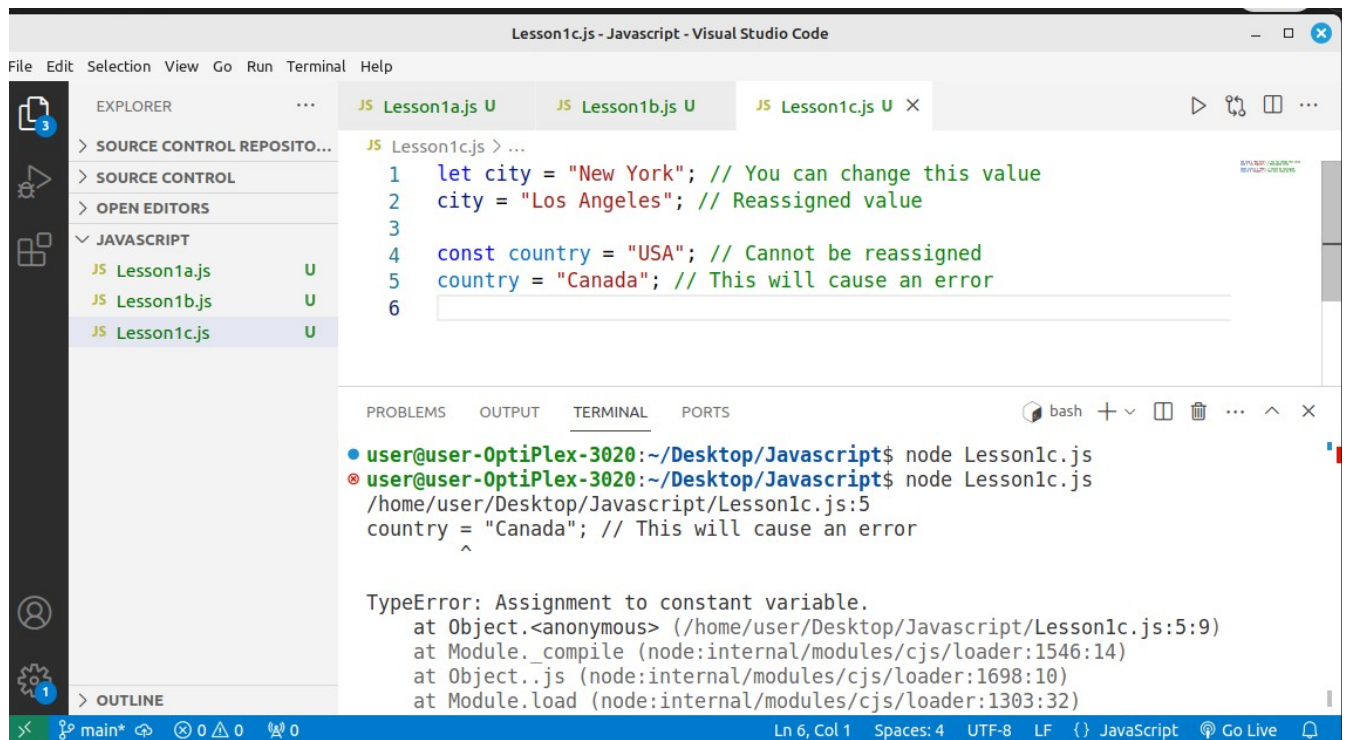
Below the code editor, the TERMINAL panel shows the command `node Lesson1b.js` being executed. The output of the program is displayed as follows:

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1b.js
John
30
20
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

The status bar at the bottom indicates the current line and column (Ln 9, Col 20), the number of spaces (4), the encoding (UTF-8), the line feed (LF), the language (JavaScript), and the Go Live button.

Variable Assignment and Reassignment

- **let** allows you to declare variables that can be reassigned to new values.
- **const** does not allow reassignment of the variable once a value has been assigned.



The screenshot shows the Visual Studio Code interface with a file named `Lesson1c.js` open. The code in the editor is as follows:

```
1 let city = "New York"; // You can change this value
2 city = "Los Angeles"; // Reassigned value
3
4 const country = "USA"; // Cannot be reassigned
5 country = "Canada"; // This will cause an error
6
```

The terminal window at the bottom shows the command `node Lesson1c.js` being executed, which results in a runtime error:

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1c.js
/home/user/Desktop/Javascript/Lesson1c.js:5
country = "Canada"; // This will cause an error
      ^
TypeError: Assignment to constant variable.
    at Object.<anonymous> (/home/user/Desktop/Javascript/Lesson1c.js:5:9)
    at Module._compile (node:internal/modules/cjs/loader:1546:14)
    at Object.<.> (node:internal/modules/cjs/loader:1698:10)
    at Module.load (node:internal/modules/cjs/loader:1303:32)
```

Task:

Students `ToDo 3` more `console.log();` Print your own messages

Data Types:

Data types are simple values that are directly assigned and used in the program.

String (string):

- Represents a sequence of characters (text).

- Strings are enclosed in either single quotes (' ') or double quotes (" ").



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar lists files: Lesson1a.js, Lesson1b.js, Lesson1c.js, and Lesson1d.js. Lesson1d.js is selected. The main editor displays the content of Lesson1d.js:

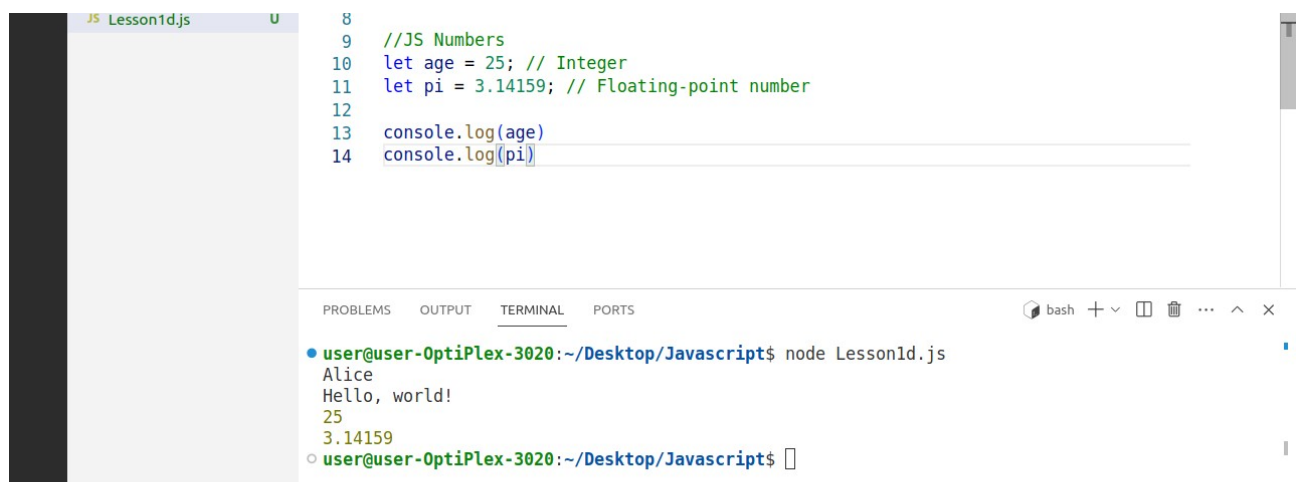
```
1 // JS strings
2 let username = "Alice";
3 let greeting = 'Hello, world!';
4
5 //Log
6 console.log(username)
7 console.log(greeting)
```

Below the editor, the TERMINAL panel shows the command `node Lesson1d.js` being executed, with the output:

```
Alice
Hello, world!
```

2. Number (number):

- Represents numeric values, both integers and floating-point numbers.



The screenshot shows the Visual Studio Code interface. The main editor displays the content of Lesson1d.js:

```
8
9 //JS Numbers
10 let age = 25; // Integer
11 let pi = 3.14159; // Floating-point number
12
13 console.log(age)
14 console.log(pi)
```

Below the editor, the TERMINAL panel shows the command `node Lesson1d.js` being executed, with the output:

```
Alice
Hello, world!
25
3.14159
```

3. Boolean (boolean):

- Represents a truth value: either true or false.

4.



```
13
16 //Boolean
17 let isAdult = true;
18 let isRegistered = false;
19
20 console.log(isAdult)
21 console.log(isRegistered)
```

PROBLEMS OUTPUT TERMINAL PORTS

user@user-OptiPlex-3020:~/Desktop/Javascript\$ node Lesson1d.js

Alice
Hello, world!
25
3.14159
true
false

user@user-OptiPlex-3020:~/Desktop/Javascript\$

Ln 21, Col 26 Spaces: 4 UTF-8 LF JavaScript Go Live

Saturday December 7, 2:50:09 PM

Undefined (undefined):

- A variable that has been declared but not assigned a value is automatically given the value undefined.



```
23 //Undefined
24 let someVariable;
25 console.log(someVariable); // Output: undefined
26
```

PROBLEMS OUTPUT TERMINAL PORTS

user@user-OptiPlex-3020:~/Desktop/Javascript\$ node Lesson1d.js

Alice
Hello, world!
25
3.14159
true
false
undefined

user@user-OptiPlex-3020:~/Desktop/Javascript\$

Ln 26, Col 1 Spaces: 4 UTF-8 LF JavaScript Go Live

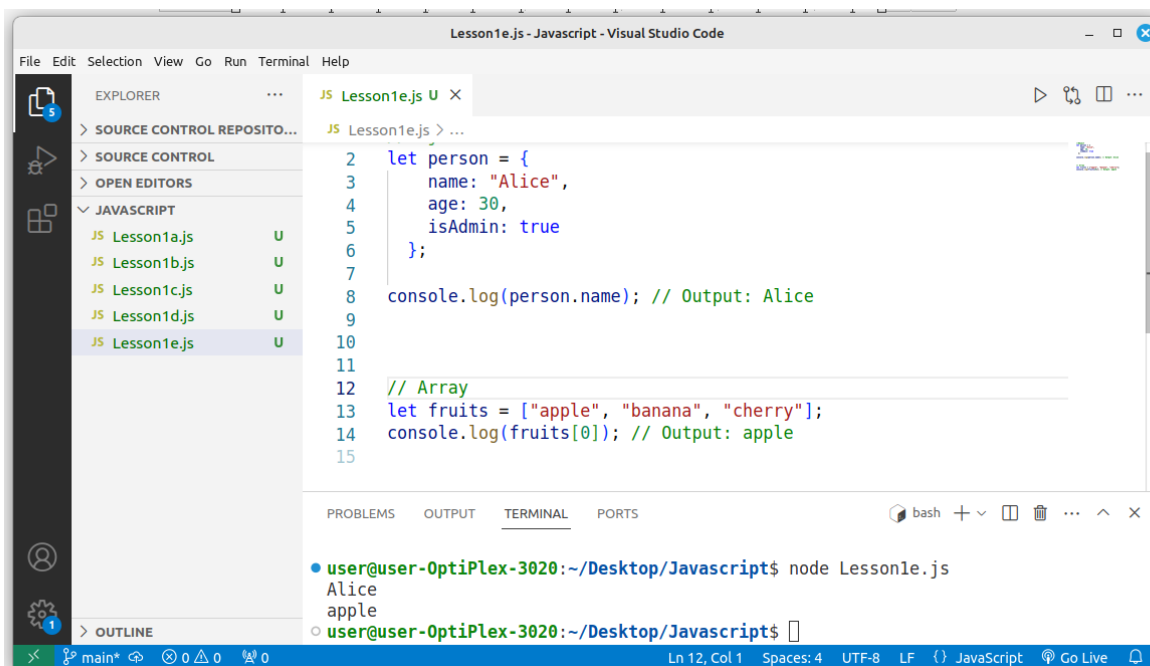
Saturday December 7, 2:51:00 PM

5. Null (null):

- Represents the intentional absence of any object value. It's a special value that represents "nothing."

Array (array):

- An array is a special type of object used to store a list of values in an ordered manner. Arrays are indexed by numbers starting from 0.



In JavaScript, the `typeof` operator is used to **determine the type of a variable or expression**. It returns a string indicating the type of the operand.

```
let num = 10;
```

```
console.log(typeof num); // Output: "number"
```

Summary

- **Variables** in JavaScript store data values and can be declared with `var`, `let`, or `const`.
- **Data types** include:
 - `string`: Text data.
 - `number`: Numeric values (integers and floats).
 - `boolean`: Logical values (true or false).
 - `undefined`: Variables declared but not assigned a value.
 - `null`: An intentional absence of a value.
- **Complex data types** like **objects** and **arrays** store collections of data.

By understanding how to declare variables and work with different data types, you can start writing more powerful and dynamic JavaScript code.

Javascript Operators

Operators are the same in all Programming Languages but the way of writing code or using the operators are different.

JavaScript operators are special symbols or keywords used to perform operations on values (known as operands). Operators can be used for mathematical calculations, comparisons, logical evaluations, assignment, and more.

Below are Javascript operators

1. Arithmetic Operators

These operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

| Operator | Description | Example | Result |
|----------|-----------------------|---------|--------|
| + | Addition | 5 + 3 | 8 |
| - | Subtraction | 5 - 3 | 2 |
| * | Multiplication | 5 * 3 | 15 |
| / | Division | 6 / 3 | 2 |
| % | Modulus (remainder) | 5 % 3 | 2 |
| ** | Exponentiation (ES6+) | 2 ** 3 | 8 |

2. Assignment Operators

Assignment operators are used to assign values to variables.

| Operator | Description | Example | Result |
|----------|---------------------------|------------|-------------------------|
| = | Assignment | let x = 5; | x = 5 |
| += | Addition Assignment | x += 3; | x = x + 3 (i.e., x = 8) |
| -= | Subtraction Assignment | x -= 2; | x = x - 2 |
| *= | Multiplication Assignment | x *= 4; | x = x * 4 |
| /= | Division Assignment | x /= 2; | x = x / 2 |
| %= | Modulus Assignment | x %= 3; | x = x % 3 |

3. Comparison Operators

These operators are used to compare two values and return a Boolean result (true or false).

| Operator | Description | Example | Result |
|----------|--------------------------|---------|--------|
| == | Equal to (value) | 5 == 5 | true |
| != | Not equal to (value) | 5 != 3 | true |
| > | Greater than | 5 > 3 | true |
| < | Less than | 5 < 8 | true |
| >= | Greater than or equal to | 5 >= 5 | true |
| <= | Less than or equal to | 3 <= 4 | true |

4. Logical Operators

These operators are used to perform logical operations, typically with Boolean values.

| Operator | Description | Example | Result |
|----------|-------------|---------------|------------|
| && | Logical AND | true && false | false |
| | Logical OR | true false | Logical OR |
| ! | Logical NOT | !true | false |

Control Statements in JavaScript

Control statements in JavaScript allow you to control the flow of your program. They let you make decisions, loop through data, and handle different situations. The three main categories of control statements are:

1. **Conditional Statements** - Making decisions in your code based on conditions.
2. **Looping Statements** - Repeating a block of code a number of times.
3. **Jump Statements** - Controlling the flow of the program based on certain conditions.

1. Conditional Statements

Conditional statements help your program to decide between different courses of action based on whether a condition is true or false.

if Statement

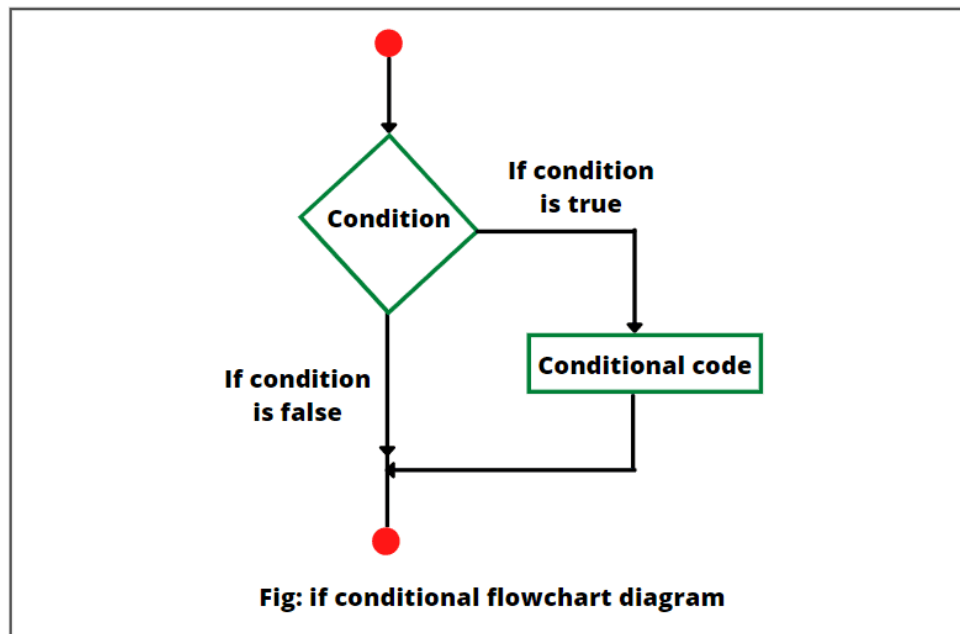
The if statement is used to execute a block of code only if a specific condition is true.

Syntax:

if (condition)

```
{  
    // code to be executed if the condition is true  
}
```

If Statement Flowchart



Example

```
Lesson2a.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
  > SOURCE CONTROL REPOSITO...
  > SOURCE CONTROL
  > OPEN EDITORS
  > JAVASCRIPT
    JS Lesson1a.js U
    JS Lesson1b.js U
    JS Lesson1c.js U
    JS Lesson1d.js U
    JS Lesson1e.js U
    JS Lesson2a.js U

  > OUTLINE

JS Lesson2a.js
1 //IF Statement
2 let age = 18;
3 if (age >= 18) {
4   console.log("You are an adult.");
5 }
6

TERMINAL
bash
● user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson2a.js
You are an adult.
○ user@user-OptiPlex-3020:~/Desktop/Javascript$
```

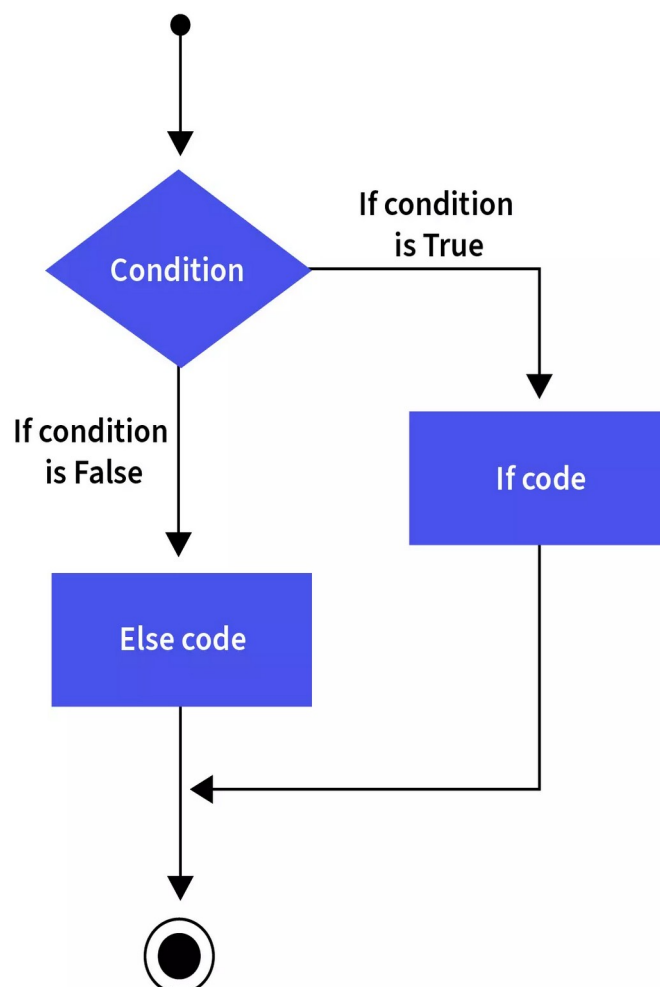
if-else Statement

The if-else statement allows you to specify a block of code to run if the condition is true and another block of code to run if the condition is false.

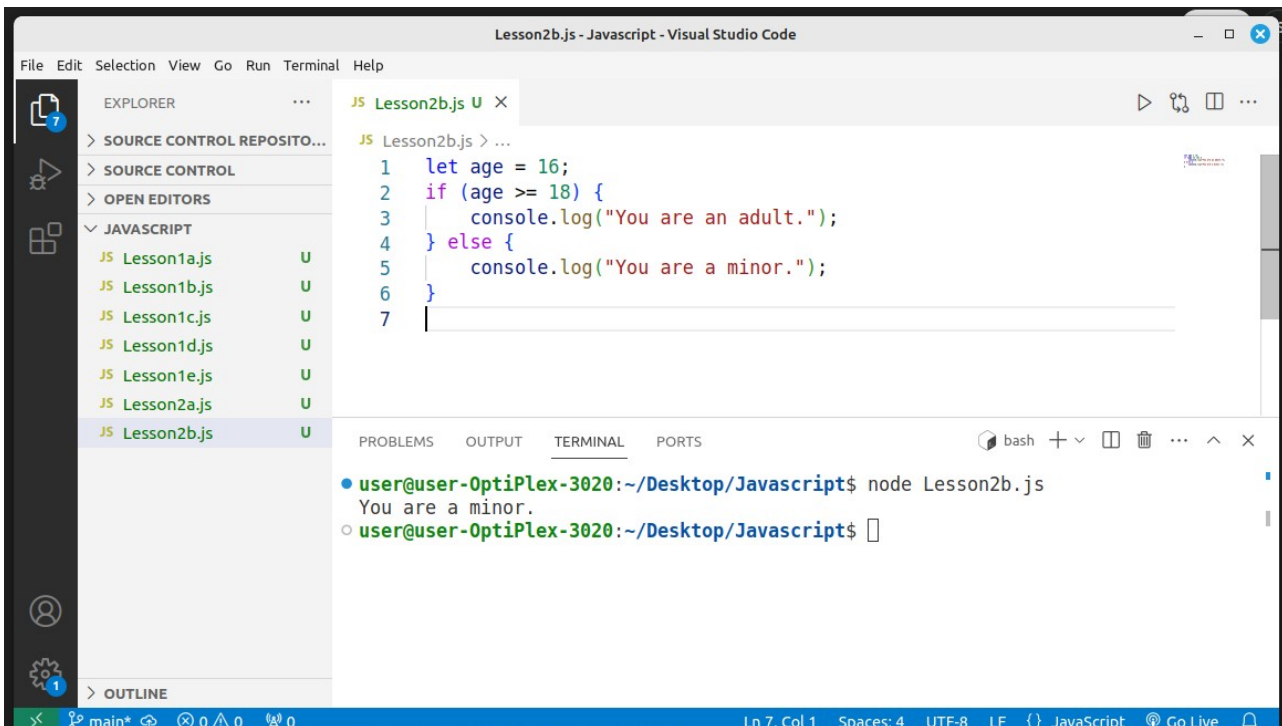
Syntax:

```
if (condition) {  
    // code to be executed if the condition is true  
} else {  
    // code to be executed if the condition is false  
}
```

If Else Flowchart



Example



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left lists several JavaScript files, with Lesson2b.js selected. The main editor displays the code for Lesson2b.js, which uses an if-else statement to check if a person is an adult or a minor based on their age. The terminal at the bottom shows the command 'node Lesson2b.js' being executed, resulting in the output 'You are a minor.'.

```
File Edit Selection View Go Run Terminal Help

EXPLORER
> SOURCE CONTROL REPOSITO...
> SOURCE CONTROL
> OPEN EDITORS
JAVASCRIPT
  JS Lesson1a.js U
  JS Lesson1b.js U
  JS Lesson1c.js U
  JS Lesson1d.js U
  JS Lesson1e.js U
  JS Lesson2a.js U
  JS Lesson2b.js U

PROBLEMS OUTPUT TERMINAL PORTS
bash
● user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson2b.js
You are a minor.
○ user@user-OptiPlex-3020:~/Desktop/Javascript$
```

```
1 let age = 16;
2 if (age >= 18) {
3     console.log("You are an adult.");
4 } else {
5     console.log("You are a minor.");
6 }
7
```

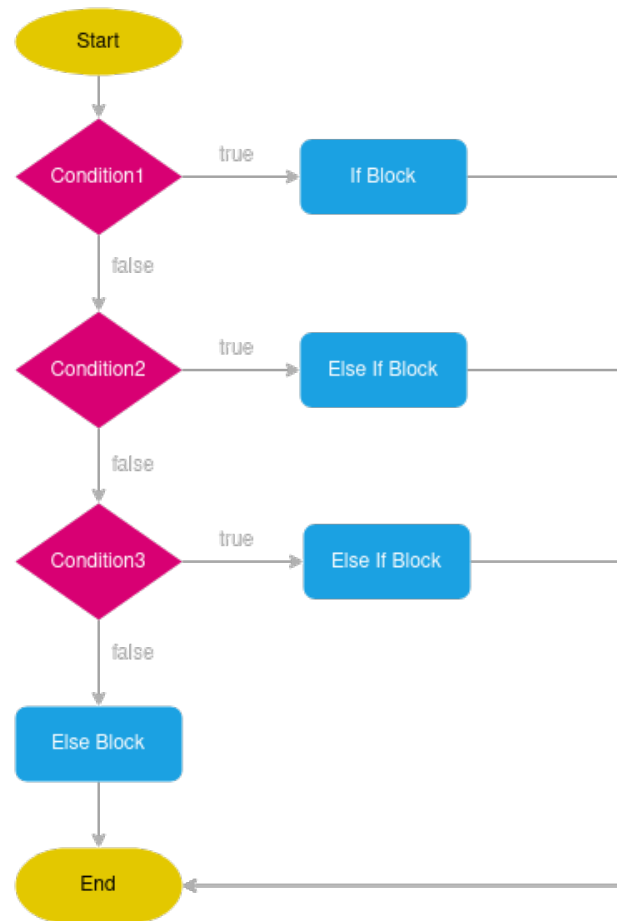
else if Statement

The else if statement is used to check multiple conditions if the first if condition is false. You can chain multiple else if statements to check for different conditions.

Syntax:

```
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition2 is true
} else {
    // code to be executed if none of the conditions are true
}
```

If Else - Else IF Flowchart



Example

The screenshot shows a Visual Studio Code window with a file named 'Lesson2c.js'. The code in the file is as follows:

```
1 let age = 21;
2 if (age >= 18 && age < 21) {
3   console.log("You are an adult but not old enough to drink.");
4 } else if (age >= 21) {
5   console.log("You are an adult and can drink alcohol.");
6 } else {
7   console.log("You are a minor.");
8 }
9
```

The terminal output shows the command `node Lesson2b.js` being executed, which results in the output `You are a minor.`

switch Statement

The switch statement evaluates an expression and executes code based on matching cases. It is a cleaner way of writing multiple if-else if conditions.

Syntax:

switch (expression) {

case value1:

 // code to be executed if expression equals value1

break;

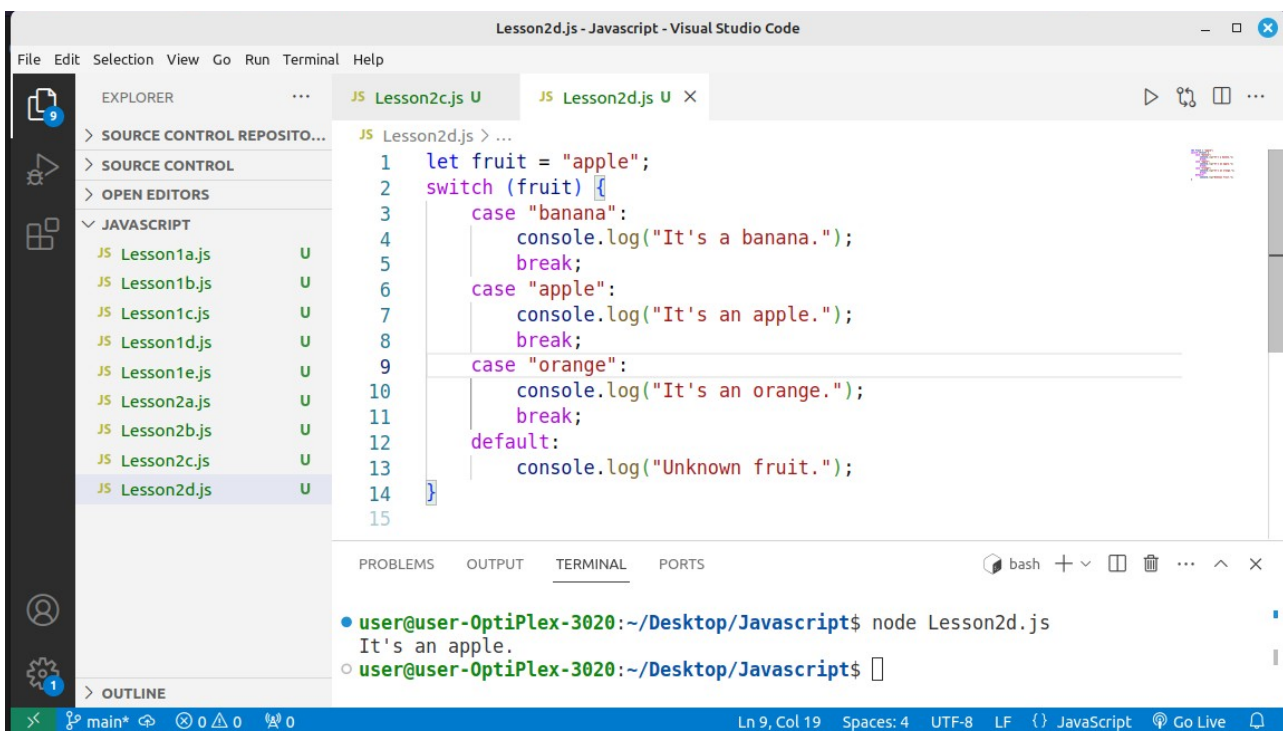
case value2:

 // code to be executed if expression equals value2

break;

default:

 // code to be executed if none of the above cases are true



The screenshot shows the Visual Studio Code editor with a file named 'Lesson2d.js' open. The code in the editor is as follows:

```
1 let fruit = "apple";
2 switch (fruit) {
3     case "banana":
4         console.log("It's a banana.");
5         break;
6     case "apple":
7         console.log("It's an apple.");
8         break;
9     case "orange":
10        console.log("It's an orange.");
11        break;
12    default:
13        console.log("Unknown fruit.");
14 }
15
```

The terminal at the bottom shows the command `node Lesson2d.js` being executed, resulting in the output `It's an apple.`

}

Summary

1. **if**: Executes code if a condition is true.
2. **if-else**: Executes one block of code if a condition is true, and another if it is false.
3. **else-if**: Checks multiple conditions in sequence if the previous conditions were false.
4. **switch**: Used to compare a variable against multiple values and execute the corresponding block of code.

These conditional statements are essential for controlling the flow of a program based on dynamic conditions. Let me know if you need further examples or explanations!

Task Todo

Example Scenario:

Assume you have the following tax brackets:

Tax Table:

| Salary Range | Tax Rate |
|--------------------|----------|
| \$0 - \$30000 | 10% |
| \$30001 - \$60000 | 15% |
| \$60001 - \$100000 | 20% |
| \$100001 and above | 25% |

Task:

The students will write a JS program that uses an **if-else if- else statement** to calculate the tax based on the provided salary.

Looping Statements

Looping statements are used to repeat a block of code multiple times based on a condition or a fixed number of iterations.

for Loop

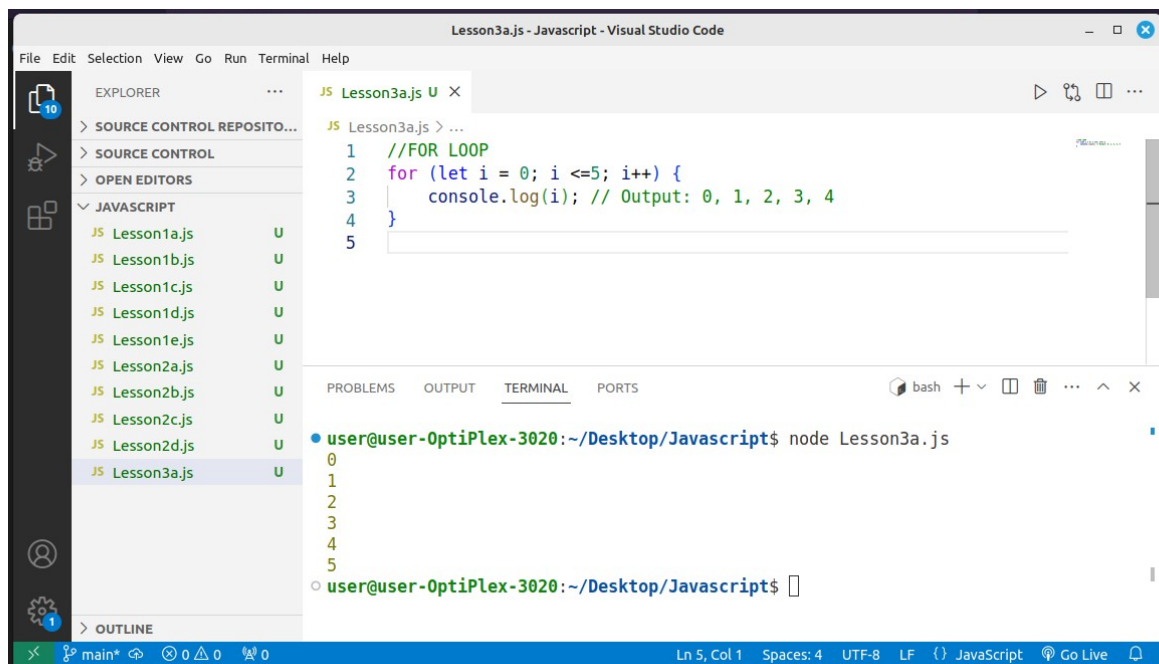
The for loop is typically used when you know beforehand how many times you want to repeat the block of code.

Syntax:

```
for (initialization; condition; increment) {  
    // code to be executed  
}
```

- **Initialization:** Executed before the loop starts
- **Condition:** Checked before each iteration. If true, the loop continues; if false, the loop stops.
- **Increment:** Executed after each iteration.

Example



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists several JavaScript files under the 'JAVASCRIPT' folder, with 'Lesson3a.js' selected. The main editor window displays the code for 'Lesson3a.js':

```
1 //FOR LOOP
2 for (let i = 0; i <=5; i++) {
3     console.log(i); // Output: 0, 1, 2, 3, 4
4 }
5
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson3a.js
0
1
2
3
4
5
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

Tasks

Task: Write a for loop to print all odd numbers from 1 to 19.

Task: Write a for loop to count down from 10 to 1 and print the numbers in the console.

Task: Write a for loop to find the largest number in the array: [10, 20, 4, 45, 99, 1].

Task: Write a for loop to print the multiplication table of 5 from 5 x 1 to 5 x 10.

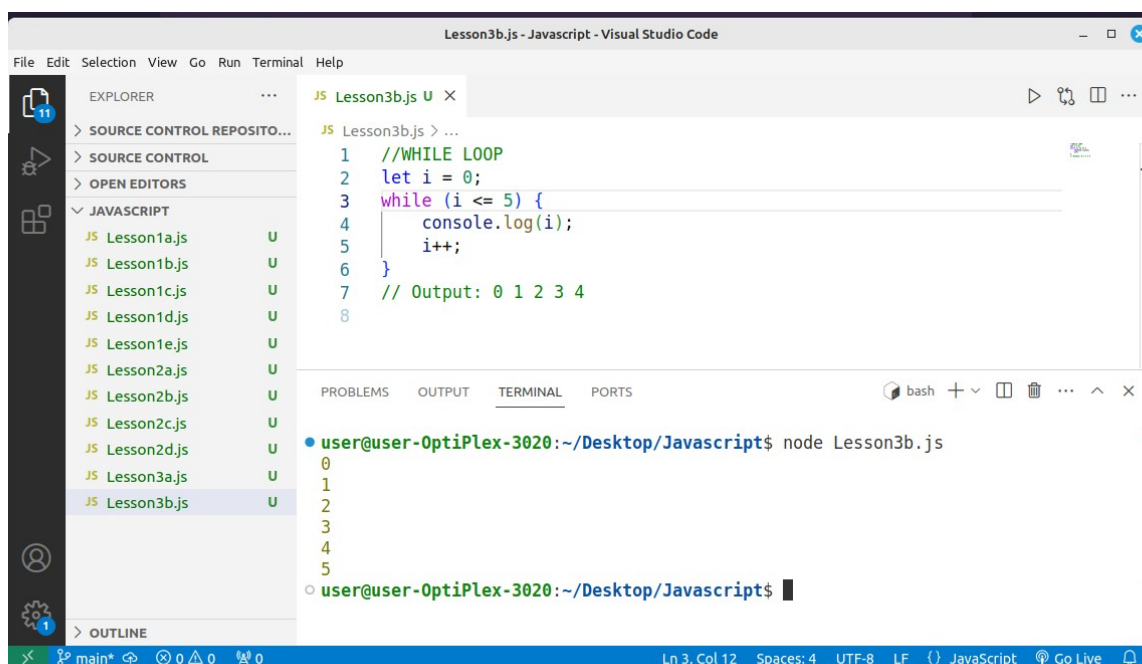
while loop

The while loop runs as long as the given condition evaluates to true.

Syntax:

```
while (condition) {  
  
    // code to be executed  
  
}
```

Example



The screenshot shows the Visual Studio Code editor with a file named Lesson3b.js. The code in the editor is as follows:

```
1 //WHILE LOOP  
2 let i = 0;  
3 while (i <= 5) {  
4     console.log(i);  
5     i++;  
6 }  
7 // Output: 0 1 2 3 4  
8
```

The terminal at the bottom shows the command `node Lesson3b.js` being executed, resulting in the output:

```
0  
1  
2  
3  
4  
5
```

Summary

- **for loop:** Best when the number of iterations is known in advance, and you can manage all the loop control logic (initialization, condition, and increment) in one place.

- **while loop:** Best when the number of iterations is unknown, and the loop should continue until a certain condition is no longer true.

Tasks

Task: Write a for loop to print all odd numbers from 1 to 19.

Task: Write a for loop to count down from 10 to 1 and print the numbers in the console.

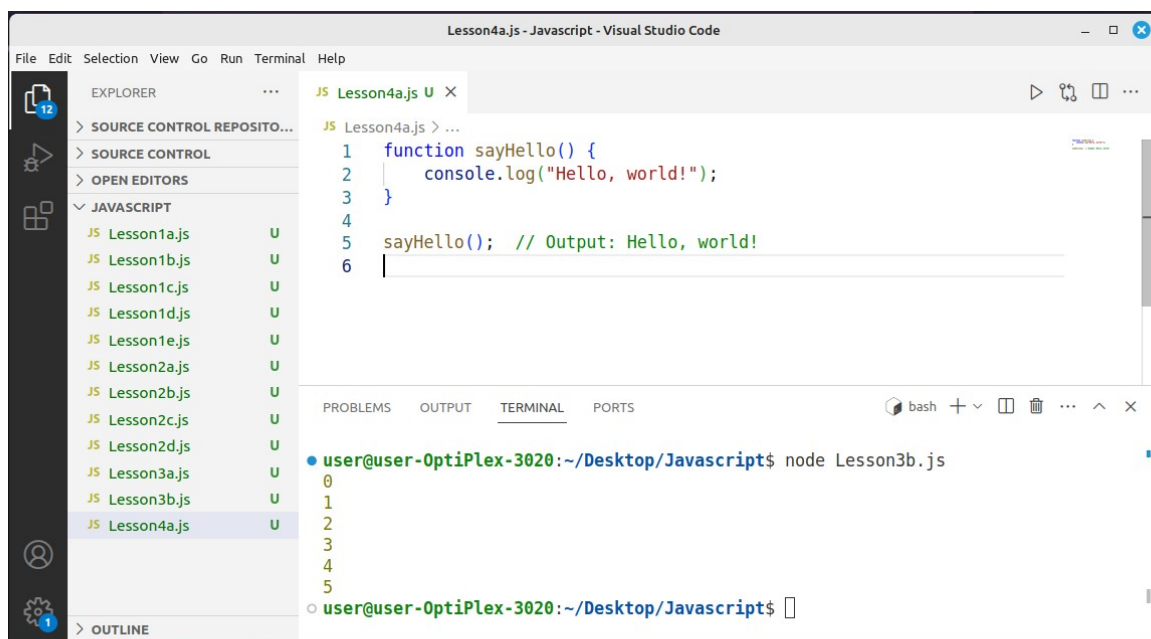
Javascript Functions

In JavaScript, functions are blocks of reusable code designed to perform a specific task. Functions allow you to group code into modular units, making your code more readable, maintainable, and reusable. Here's a comprehensive guide on how functions work in JavaScript:

Basic Function Without Parameters

A simple function can be created using the function keyword. This function doesn't accept any parameters and just performs a task, like printing something to the console.

Example:



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left lists several JavaScript files, with Lesson4a.js selected. The main editor displays the code for Lesson4a.js, which defines a function `sayHello()` that logs "Hello, world!" to the console and then calls itself. The Terminal panel at the bottom shows the command `node Lesson3b.js` being executed, resulting in the output "0", "1", "2", "3", "4", "5".

```
Lesson4a.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> SOURCE CONTROL REPOSITO...
> SOURCE CONTROL
> OPEN EDITORS
JAVASCRIPT
  JS Lesson1a.js U
  JS Lesson1b.js U
  JS Lesson1c.js U
  JS Lesson1d.js U
  JS Lesson1e.js U
  JS Lesson2a.js U
  JS Lesson2b.js U
  JS Lesson2c.js U
  JS Lesson2d.js U
  JS Lesson3a.js U
  JS Lesson3b.js U
  JS Lesson4a.js U
  > OUTLINE

JS Lesson4a.js > ...
1 function sayHello() {
2   console.log("Hello, world!");
3 }
4
5 sayHello(); // Output: Hello, world!
6

PROBLEMS OUTPUT TERMINAL PORTS
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson3b.js
0
1
2
3
4
5
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

In above example:

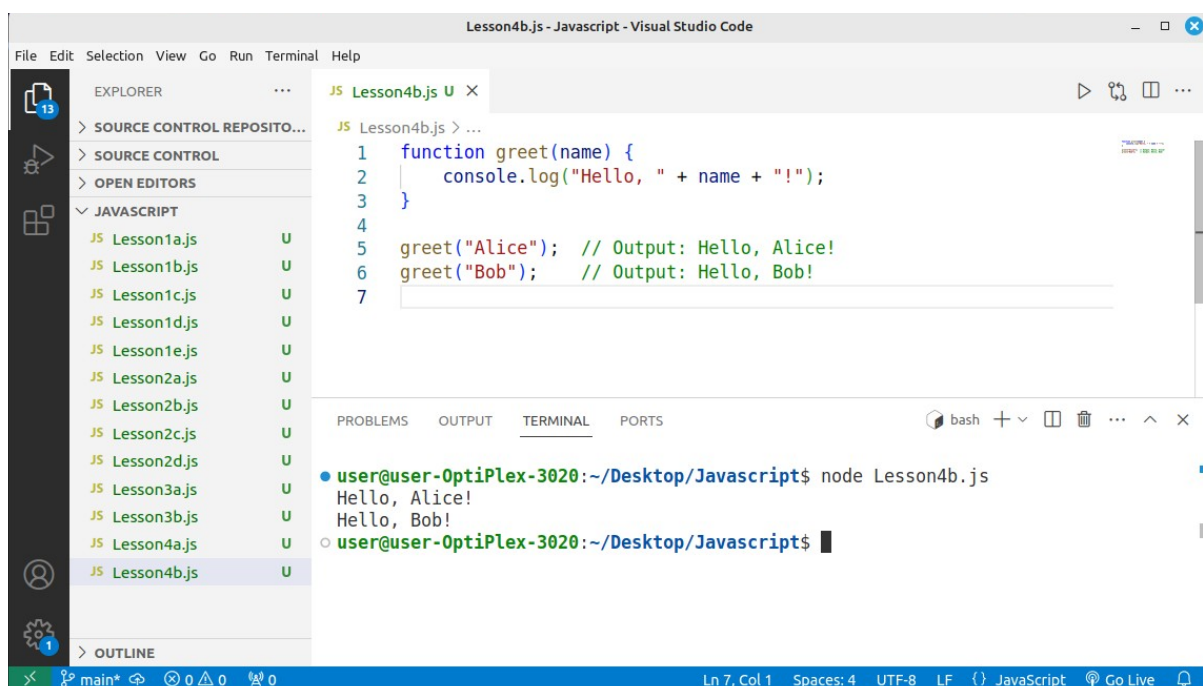
- The function `sayHello` doesn't take any parameters.

- When called, it prints "Hello, world!" to the console.

Function with Parameters

A function can accept parameters, which are variables you pass when calling the function. These parameters can be used inside the function to perform tasks dynamically.

Example:



The screenshot shows the Visual Studio Code editor with a file named `Lesson4b.js` open. The file contains the following JavaScript code:

```
1 function greet(name) {  
2   console.log("Hello, " + name + "!");  
3 }  
4  
5 greet("Alice"); // Output: Hello, Alice!  
6 greet("Bob");   // Output: Hello, Bob!  
7
```

The Explorer sidebar on the left shows a list of JavaScript files, with `Lesson4b.js` selected. The Terminal panel at the bottom shows the command `node Lesson4b.js` being executed, resulting in the output:

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson4b.js  
Hello, Alice!  
Hello, Bob!  
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

In this example:

- The function `greet` takes a parameter `name`.
- When called with an argument like `"Alice"`, it prints `"Hello, Alice!"`.

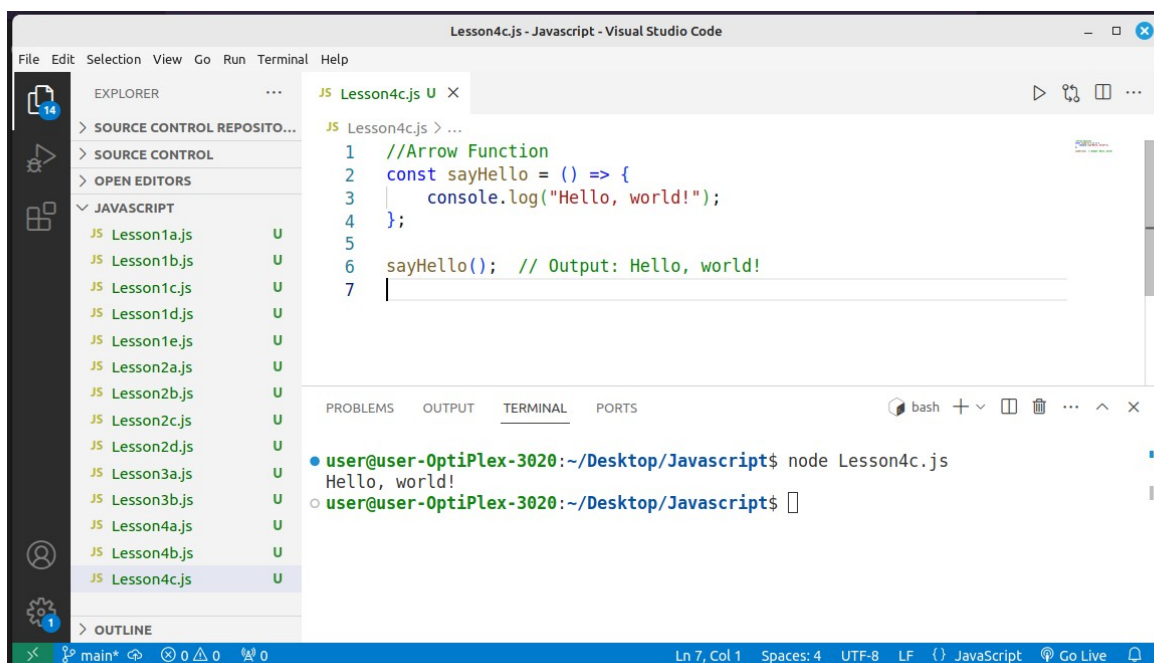
Arrow Functions

Arrow functions, introduced in ES6, provide a more concise syntax for writing functions. Arrow functions are especially useful in callback functions and allow you to write functions in a more compact way.

Arrow Function without Parameters:

An arrow function without parameters can be written as follows:

Example:



The screenshot shows the Visual Studio Code editor with a file named Lesson4c.js. The code defines an arrow function named sayHello and calls it. The terminal shows the command to run the file and the output.

```
1 //Arrow Function
2 const sayHello = () => {
3   console.log("Hello, world!");
4 };
5
6 sayHello(); // Output: Hello, world!
7
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson4c.js
Hello, world!
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

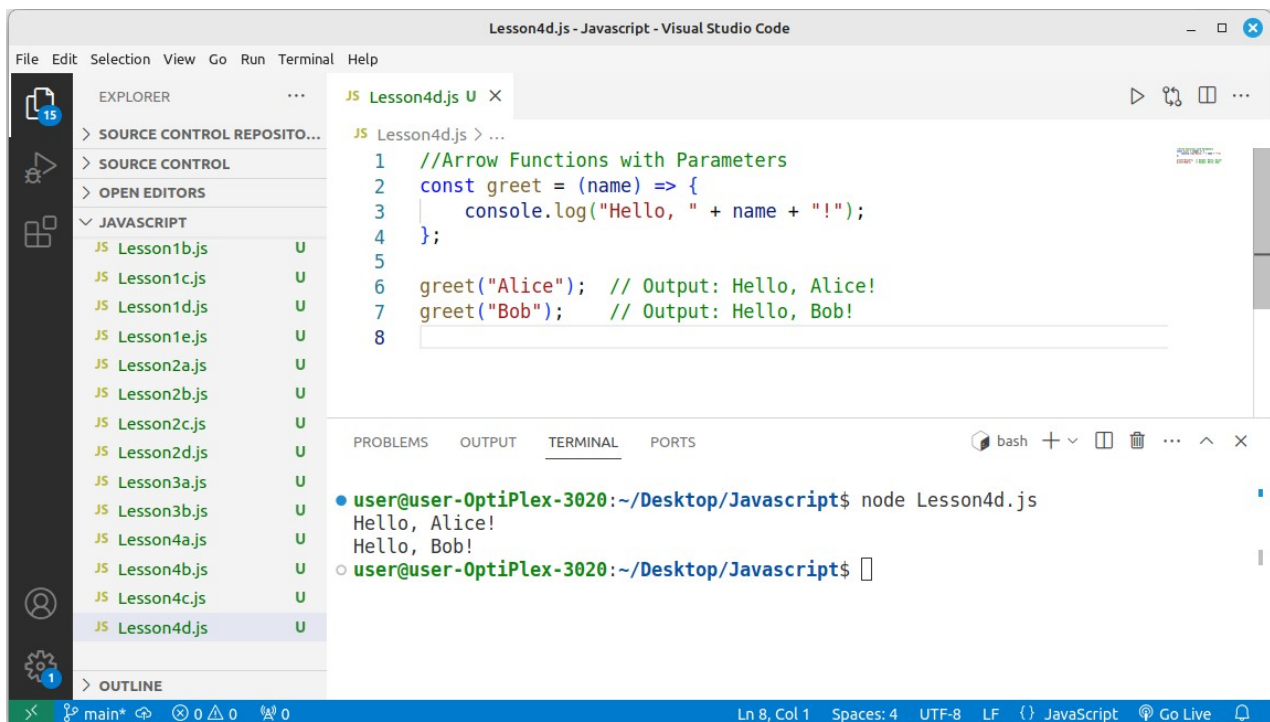
In this case:

- sayHello is assigned an arrow function.
- The function doesn't take any parameters, and it performs the same task as the first example: printing "Hello, world!".

Arrow Function with Parameters:

An arrow function with parameters follows a similar pattern but allows for dynamic input.

Example:



The screenshot shows the Visual Studio Code editor with a file named `Lesson4d.js` open. The code defines an arrow function `greet` that takes a parameter `name` and logs a greeting message. The function is then called with the strings "Alice" and "Bob". The terminal output shows the execution results.

```
1 //Arrow Functions with Parameters
2 const greet = (name) => {
3   console.log("Hello, " + name + "!");
4 };
5
6 greet("Alice"); // Output: Hello, Alice!
7 greet("Bob");   // Output: Hello, Bob!
8
```

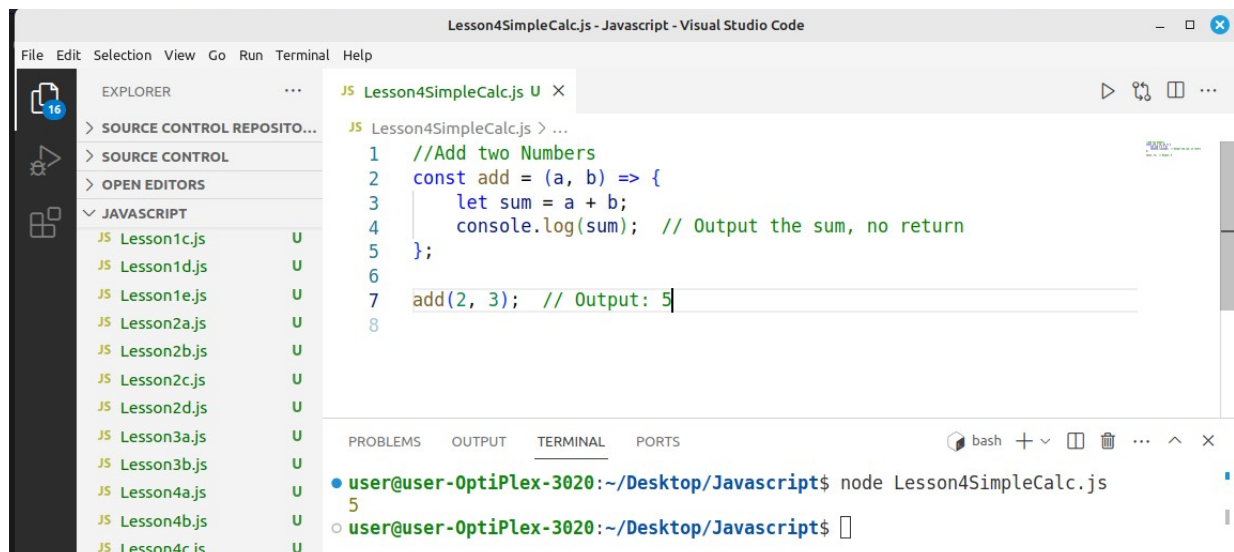
Terminal Output:

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson4d.js
Hello, Alice!
Hello, Bob!
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

In this case:

- The arrow function `greet` takes one parameter, `name`, and prints "Hello, <name>!".

Example:



The screenshot shows the Visual Studio Code editor with a file named `Lesson4SimpleCalc.js` open. The code defines an arrow function `add` that takes two arguments, `a` and `b`, calculates their sum, and logs it to the console. The function is then called with `add(2, 3)`. The terminal at the bottom shows the command `node Lesson4SimpleCalc.js` being executed, resulting in the output `5`.


```
1 //Add two Numbers
2 const add = (a, b) => {
3   let sum = a + b;
4   console.log(sum); // Output the sum, no return
5 };
6
7 add(2, 3); // Output: 5
8
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson4SimpleCalc.js
5
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

Explanation:

- **let sum = a + b;**: The sum is calculated and stored in the variable `sum`.
- **console.log(sum);**: Instead of returning the sum, we simply log the result to the console.

Example - Finding Body Mass Index - Arrow Function



The screenshot shows the Visual Studio Code editor with a file named `Lesson4SimpleBMICalc.js` open. The code defines an arrow function `calculateBMI` that takes `weight` and `height` as arguments, calculates the BMI, and logs it to the console. The function is then called with two sets of arguments: `calculateBMI(70, 1.75)` and `calculateBMI(50, 1.60)`. The terminal at the bottom shows the command `node Lesson4SimpleBMICalc.js` being executed, resulting in the output `22.857142857142858` and `19.531249999999996`.

```
1 const calculateBMI = (weight, height) => {
2   const bmi = weight / (height * height);
3   console.log(bmi);
4 };
5
6 // Example usage with direct arguments:
7 calculateBMI(70, 1.75); // Output: 22.857142857142858
8 calculateBMI(50, 1.60); // Output: 19.53125
9
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson4SimpleBMICalc.js
22.857142857142858
19.531249999999996
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

Summary:

- **Basic Function (no parameters):** A function that doesn't take any input.
- **Function with Parameters:** A function that accepts parameters to be used inside its body.
- **Arrow Function (no parameters):** A more concise way to write functions without parameters.
- **Arrow Function with Parameters:** A concise function syntax that also accepts parameters.

Task ToDo

Task: Write an arrow function **calculateSimpleInterest** that calculates the simple interest based on the given principal amount, rate of interest, and time period. The function should take three parameters:

- p: The initial amount of money (in dollars).
- r: The rate of interest (in percentage).
- t: The time period for which the interest is calculated (in years).

The formula for calculating simple interest is:

$$\text{Simple Interest (SI)} = (\text{Principal} * \text{Rate} * \text{Time}) / 100$$

Requirements:

1. The function should return the calculated simple interest.

2. Use arrow function syntax.
3. Ensure the function is flexible and accepts different values for the parameters.

Practice Questions

1. What is a variable in JavaScript, and how do you declare one?
2. What is the difference between var, let, and const in JavaScript?
3. What is the difference between undefined and null in JavaScript?
4. What is the purpose of the console.log() method in JavaScript?
5. What will be the result of declaring a variable without initializing it?
6. How do you declare a **constant** in JavaScript?
7. What does const do in JavaScript, and when can it be used?
8. What will be the output of the following code?

```
let a = 10;
```

```
let b = a;
```

```
b = 20;
```

```
console.log(a);
```

```
console.log(b);
```

9. What does the typeof operator do in JavaScript?
10. Why are comments used, and how can you make a comment in JavaScript?
11. What are the different types of data types in JavaScript?
12. What happens when you declare a variable with var in a loop, and why does it behave differently than let?
13. How do you access the first element of an array?

14. What is a **Boolean** in JavaScript, and what values can it have?
15. What is the difference between a **for** loop and a **while** loop?
16. What is the difference between an **object** and an **array** in JavaScript?
17. How can you add two numbers in JavaScript?
18. How do you create an object Type in JavaScript?
19. Write an arrow function **square** that takes a number as a parameter and logs its square
20. Create an arrow function **isEven** that takes a number as a parameter and logs a message if number is ODD or EVEN
21. Write an arrow function **celsiusToFahrenheit** that takes a temperature in Celsius as a parameter and converts it to Fahrenheit. Use the formula:
Fahrenheit = (Celsius * 9/5) + 32.

Appendix 1

Reserved keywords in JavaScript

| Reserved Words in JavaScript | | | |
|------------------------------|------------|------------|--------------|
| abstract | else | instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super | |

Useful Links

<https://www.w3schools.com/js/>

<https://www.microverse.org/blog/introduction-to-javascript-a-guide-for-beginners>

<https://careerfoundry.com/en/tutorials/web-development-for-beginners/an-introduction-to-javascript>