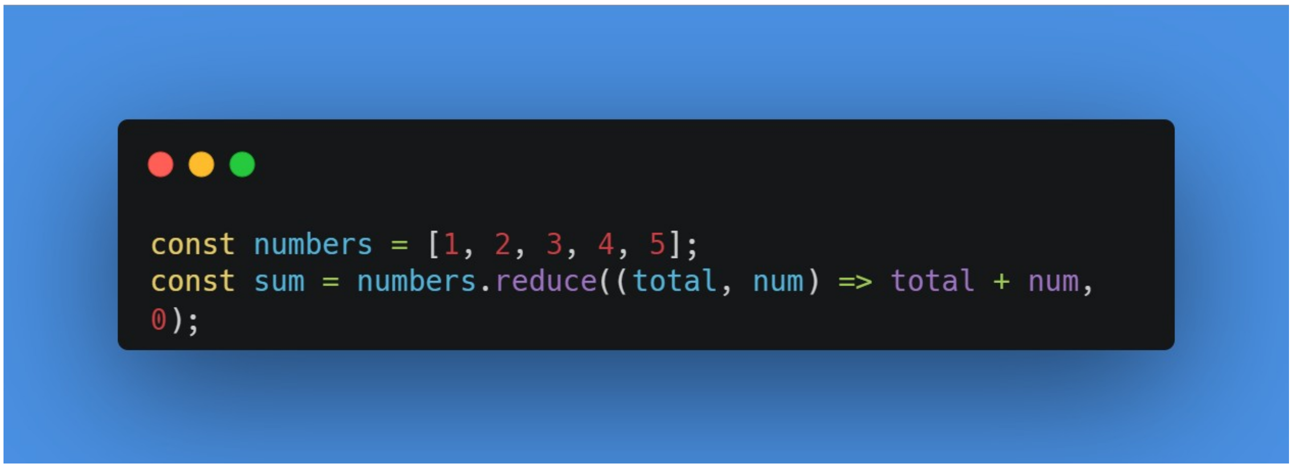
JavaScript

Book 5

ES6 and Beyond



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

https://github.com/modcomlearning/Javascript

MODCOM
Institute of Technology

Table of Contents

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. **ES6**, also known as **ECMAScript 2015**, is the sixth version of the ECMAScript standard, which is the specification that JavaScript follows. 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/

Node.js is a **JavaScript runtime environment** that allows you to run JavaScript code **outside** of a web browser.

We will be using Nodejs to run our JS Files.

Install Node JS from https://nodejs.org/en/learn/getting-started/how-to-install-nodejs or https://www.geeksforgeeks.org/install-node-js-on-windows/

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?

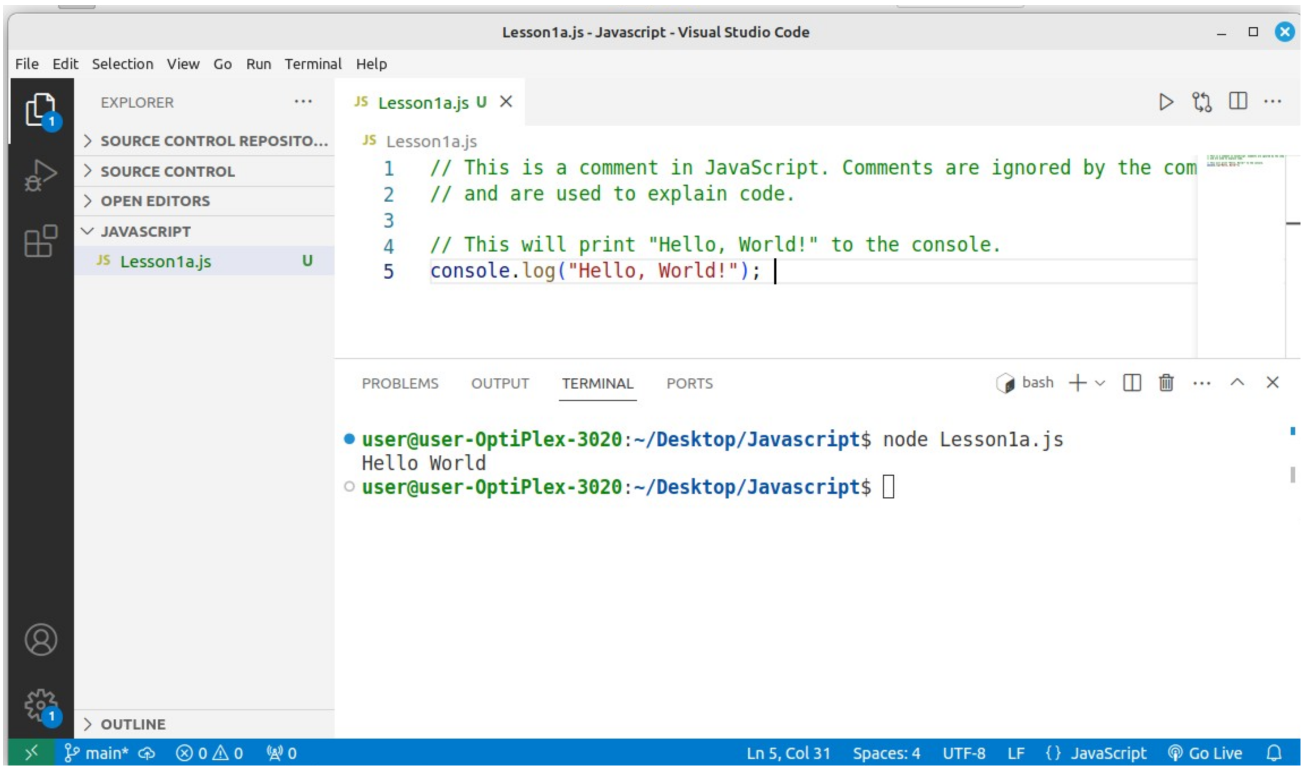
- **Interactivity**: JavaScript allows you to add interactivity to websites, such as handling user input, animations, and updates without reloading the page.
- **Versatility**: JavaScript is used for both **client-side** (in the browser) and **server-side** (with Node.js) development, making it a full-stack 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.


Lesson1a.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SOURCE CONTROL REPOSITO...
SOURCE CONTROL
OPEN EDITORS
JAVASCRIPT
Lesson1a.js U
OUTLINE
PROBLEMS OUTPUT TERMINAL PORTS
bash
main* ⊗ 0 ⚠ 0 0
Ln 5, Col 31 Spaces: 4 UTF-8 LF {} JavaScript Go Live

```javascript
// This is a comment in JavaScript. Comments are ignored by the com
// and are used to explain code.

// This will print "Hello, World!" to the console.
console.log("Hello, World!");
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1a.js
Hello World
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

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

To log is to print or output something

node Lesson1a.js

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

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.

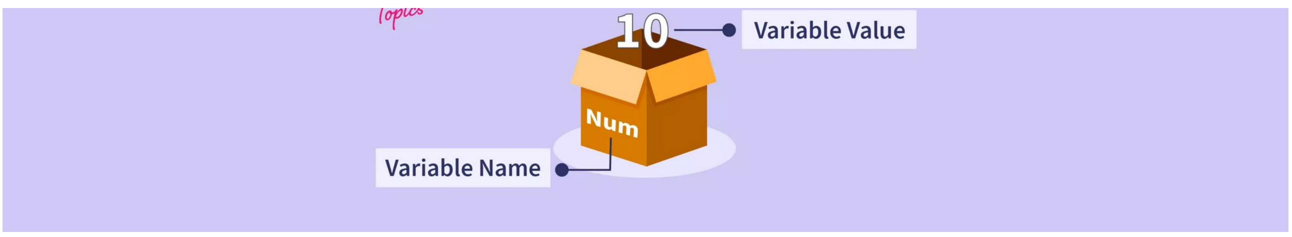
JavaScript Variables and Data Types

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.
    - `let` is less secure and unsafe.
- **let**: The modern and preferred way to declare variables.
    - `let` is more safer and secure.
- **const**: Declares a constant value.

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 and const.

- **let** is used to declare variables
- **const** is used to declare a constant, i.e., a variable whose value cannot be changed

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:**

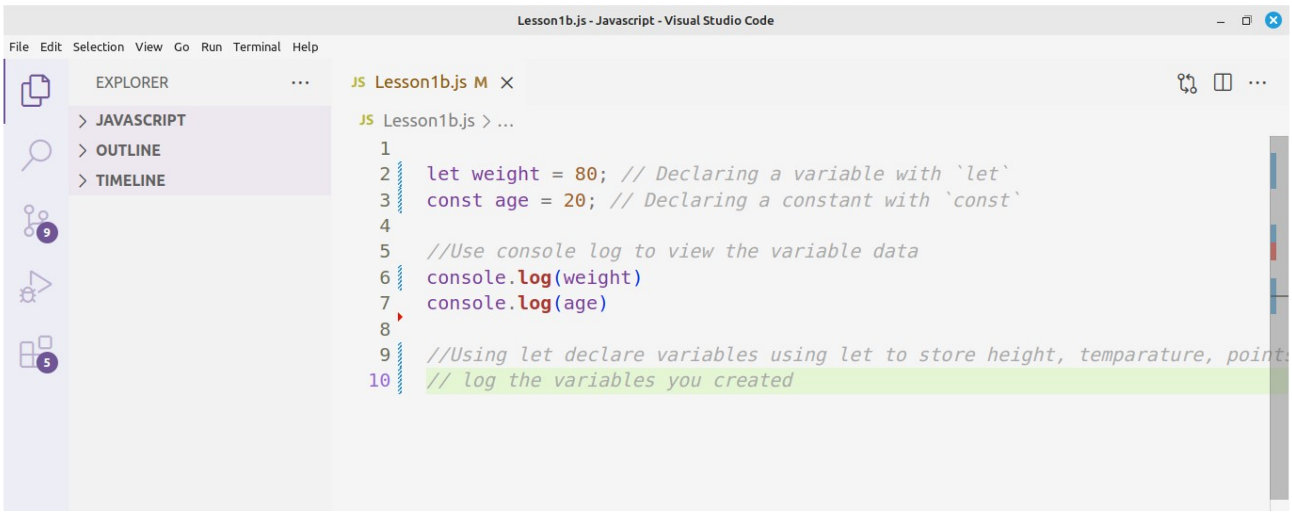
- **Camel case** is generally used for variable names in JavaScript, where the first word is lowercase and each subsequent word is capitalized.
  - Example: let userAge = 25;

- **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 are used to declare variables
- Variables declared with const cannot be changed
- JavaScript variables are case-sensitive.

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


Lesson1b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson1b.js M
Lesson1b.js > ...

```javascript

let weight = 80; // Declaring a variable with `let`
const age = 20; // Declaring a constant with `const`

//Use console log to view the variable data
console.log(weight)
console.log(age)

//Using let declare variables using let to store height, temparature, point
// log the variables you created
```

TODO

Students to Create Lesson1c.js

1. Using let declare variables using let to store height, temperature, points

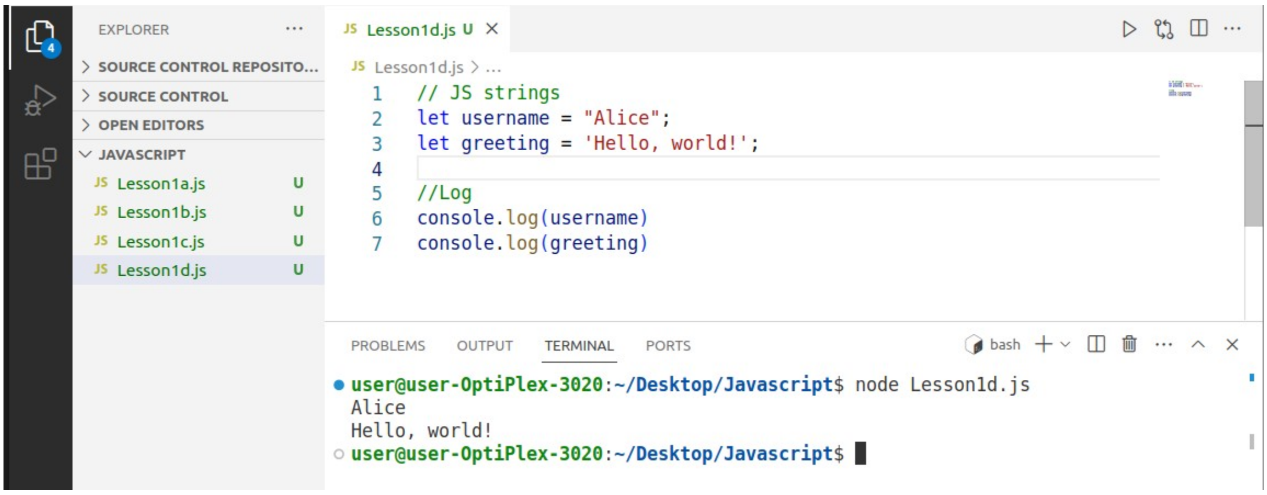
2. Using **console.log** to print the variables you created.

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 (" ").

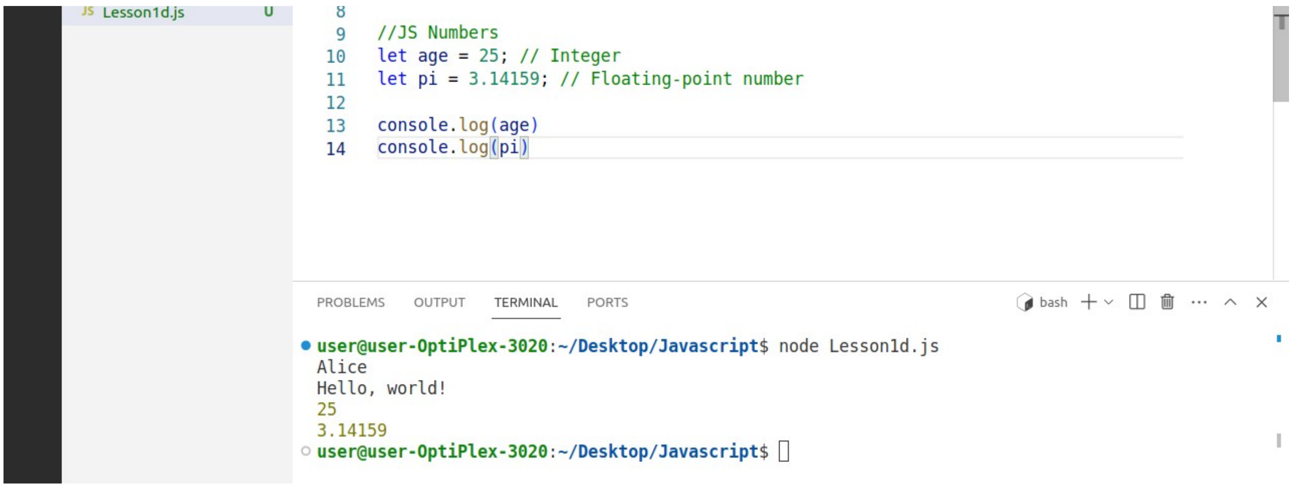


```
// JS strings
let username = "Alice";
let greeting = 'Hello, world!';

//Log
console.log(username)
console.log(greeting)
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1d.js
Alice
Hello, world!
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

2. Number (number):

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

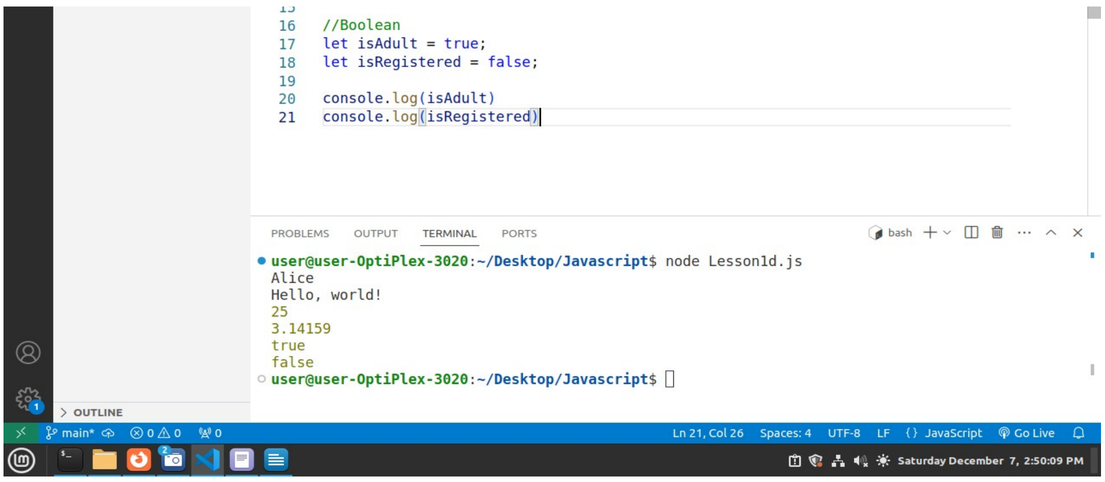


```

//JS Numbers
let age = 25; // Integer
let pi = 3.14159; // Floating-point number

console.log(age)
console.log(pi)
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1d.js
Alice
Hello, world!
25
3.14159
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

3. Boolean (boolean):

- Represents a truth value: either true or false.



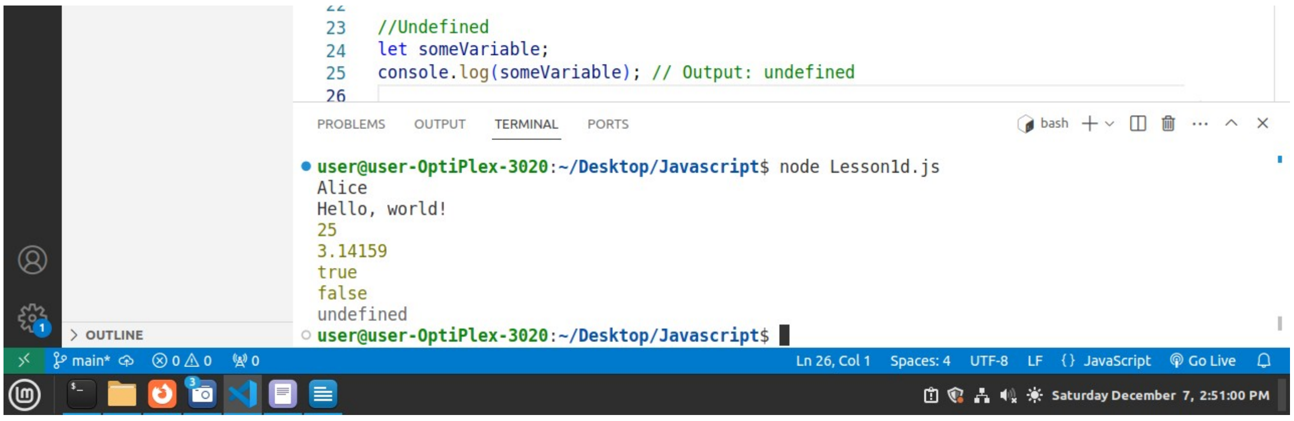
```
//Boolean
let isAdult = true;
let isRegistered = false;

console.log(isAdult)
console.log(isRegistered)
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1d.js
Alice
Hello, world!
25
3.14159
true
false
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

PROBLEMS OUTPUT TERMINAL PORTS
bash
OUTLINE
main*
Ln 21, Col 26 Spaces: 4 UTF-8 LF JavaScript Go Live
Saturday December 7, 2:50:09 PM

4. Undefined (undefined):

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



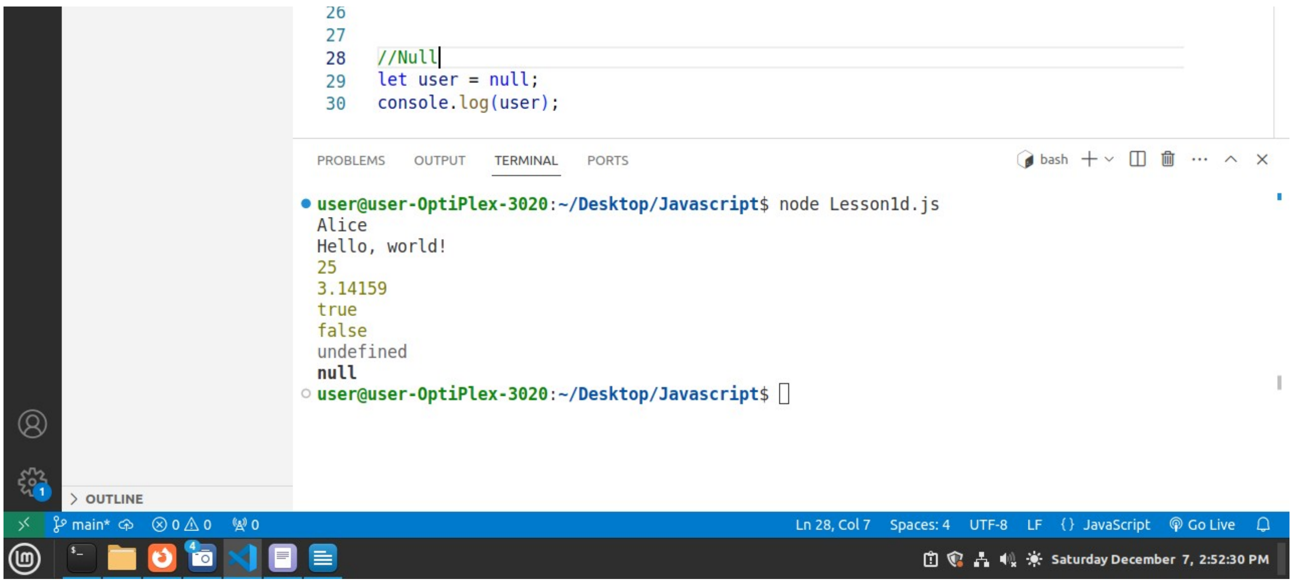
```
//Undefined
let someVariable;
console.log(someVariable); // Output: undefined
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1d.js
Alice
Hello, world!
25
3.14159
true
false
undefined
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

PROBLEMS OUTPUT TERMINAL PORTS
bash
OUTLINE
main*
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."



```
//Null
let user = null;
console.log(user);
```

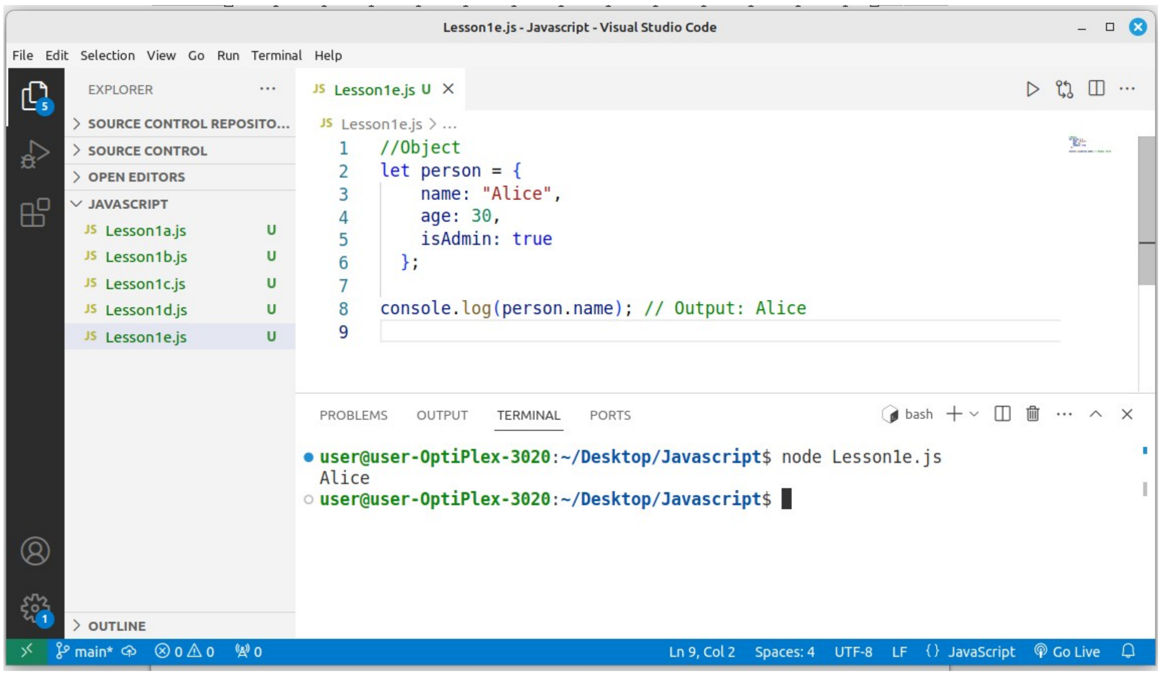
```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1d.js
Alice
Hello, world!
25
3.14159
true
false
undefined
null
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

Objects and Arrays (Complex Data Types)

Besides the data types, JavaScript also has **objects** and A**rrays**, which are more complex data structures.

Object (object):

- An object is a collection of key-value pairs where each key (also called a property) is associated with a value. The value can be any valid JavaScript data type.

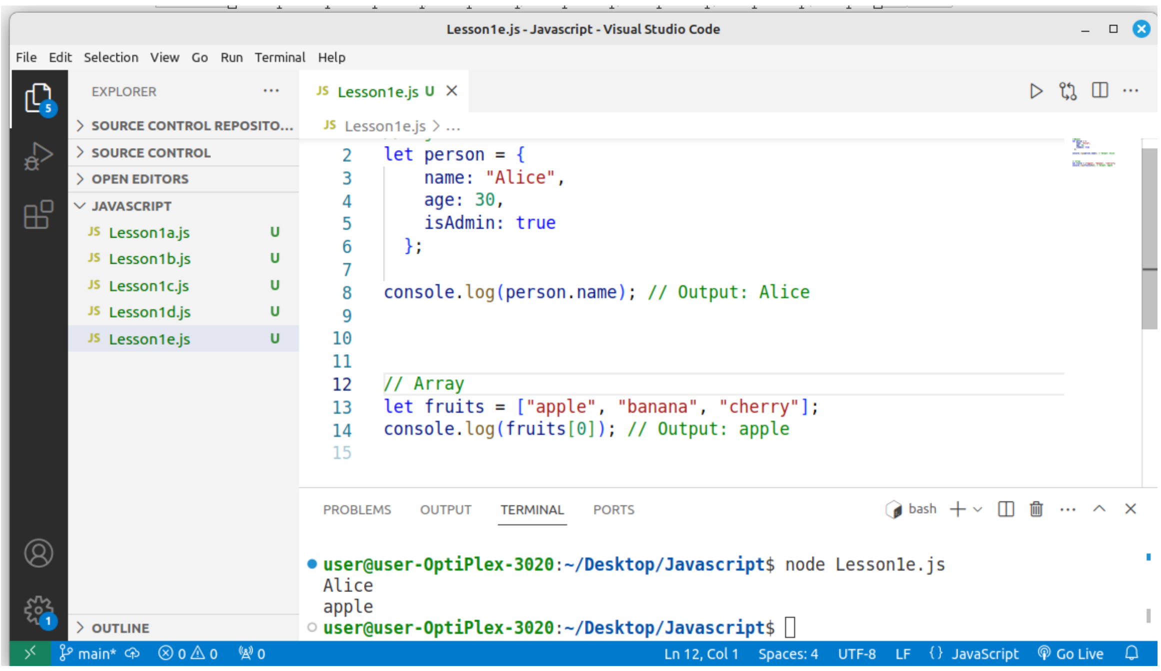

Lesson1e.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SOURCE CONTROL REPOSITO...
SOURCE CONTROL
OPEN EDITORS
JAVASCRIPT
Lesson1a.js U
Lesson1b.js U
Lesson1c.js U
Lesson1d.js U
Lesson1e.js U
OUTLINE
PROBLEMS OUTPUT TERMINAL PORTS
main* 0 0 0
Ln 9, Col 2 Spaces: 4 UTF-8 LF JavaScript Go Live

```
//Object
let person = {
    name: "Alice",
    age: 30,
    isAdmin: true
  };

console.log(person.name); // Output: Alice
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1e.js
Alice
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

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.


Lesson1e.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SOURCE CONTROL REPOSITO...
SOURCE CONTROL
OPEN EDITORS
JAVASCRIPT
Lesson1a.js U
Lesson1b.js U
Lesson1c.js U
Lesson1d.js U
Lesson1e.js U
OUTLINE
PROBLEMS OUTPUT TERMINAL PORTS
main* 0 0 0
Ln 12, Col 1 Spaces: 4 UTF-8 LF JavaScript Go Live

```
let person = {
    name: "Alice",
    age: 30,
    isAdmin: true
  };

console.log(person.name); // Output: Alice



// Array
let fruits = ["apple", "banana", "cherry"];
console.log(fruits[0]); // Output: apple
```

```
user@user-OptiPlex-3020:~/Desktop/Javascript$ node Lesson1e.js
Alice
apple
user@user-OptiPlex-3020:~/Desktop/Javascript$
```

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 A**rrays/Lists** 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.

Task

1. Students to Create 3 String variables to store username, password and email and log them in the console.

2. Students to Create an Array of 5 Counties and log the second county in the console using its index.

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

Operator	Description	Example	Result
+=	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

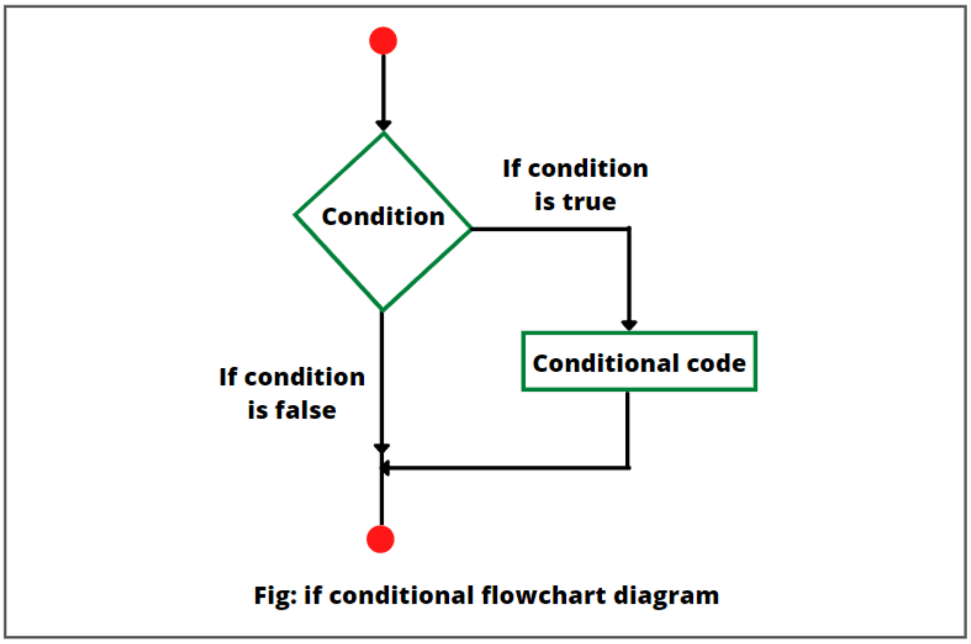
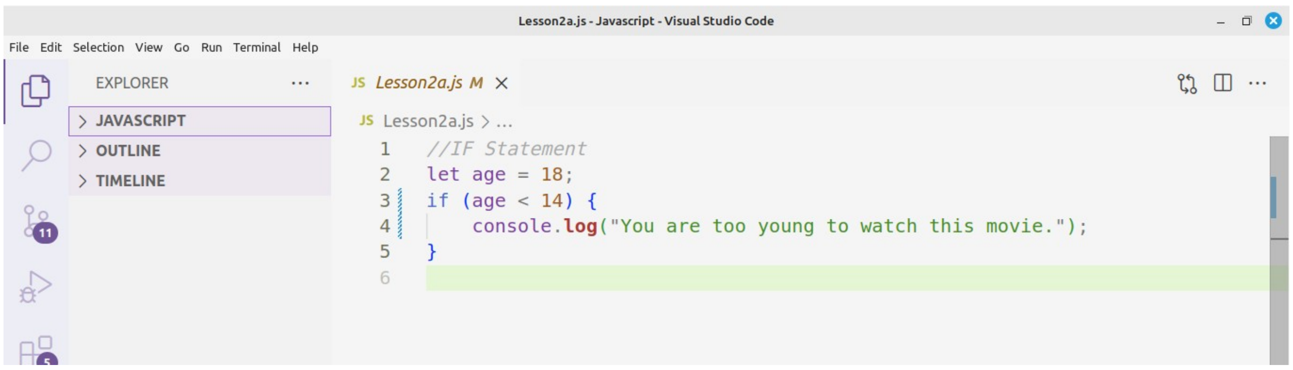


Fig: if conditional flowchart diagram

Example


Lesson2a.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson2a.js M
Lesson2a.js > ...

```javascript
//IF Statement
let age = 18;
if (age < 14) {
    console.log("You are too young to watch this movie.");
}
```

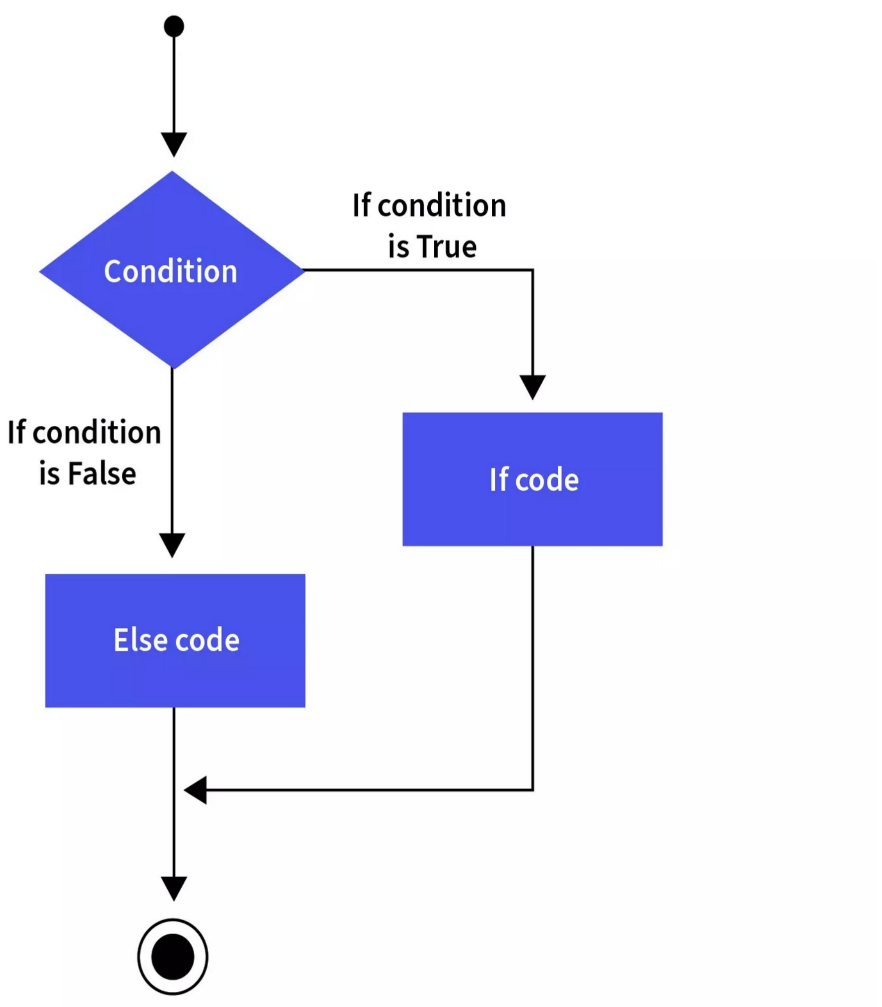
if-else Statement

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.

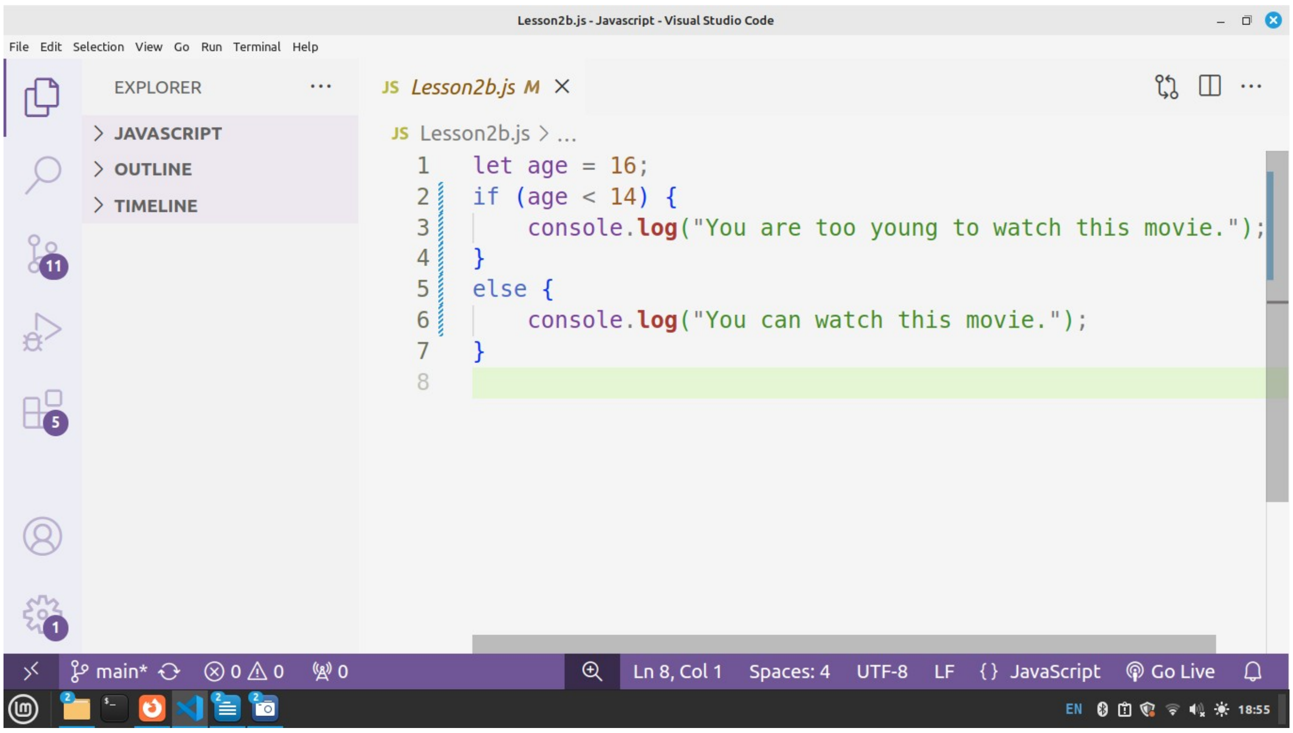
Syntax:

```
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



Lesson2b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson2b.js M
Lesson2b.js > ...
main*
Ln 8, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live

```
let age = 16;
if (age < 14) {
    console.log("You are too young to watch this movie.");
}
else {
    console.log("You can watch this movie.");
}
```

else if Statement

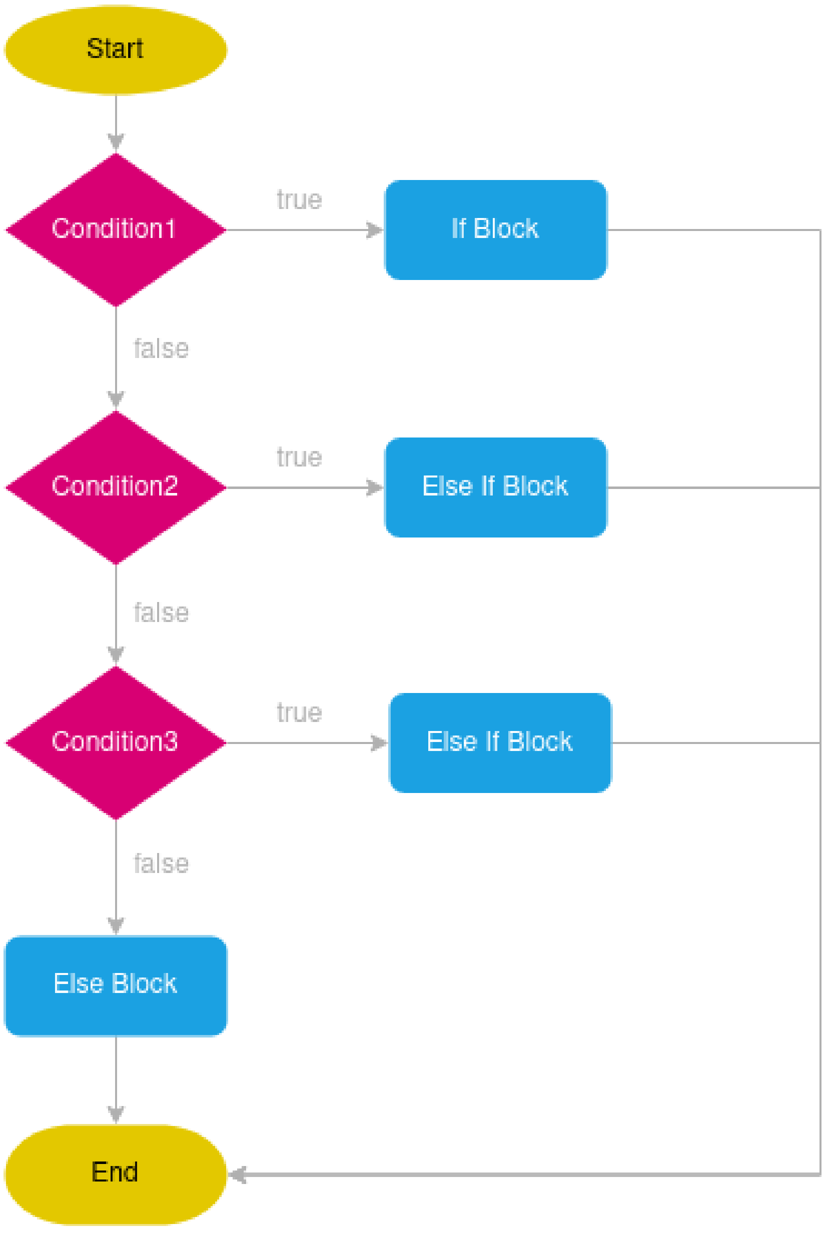
The else if statement is used to check multiple conditions if the first if condition is false.

You can chain multiple else if statements to check for different conditions.

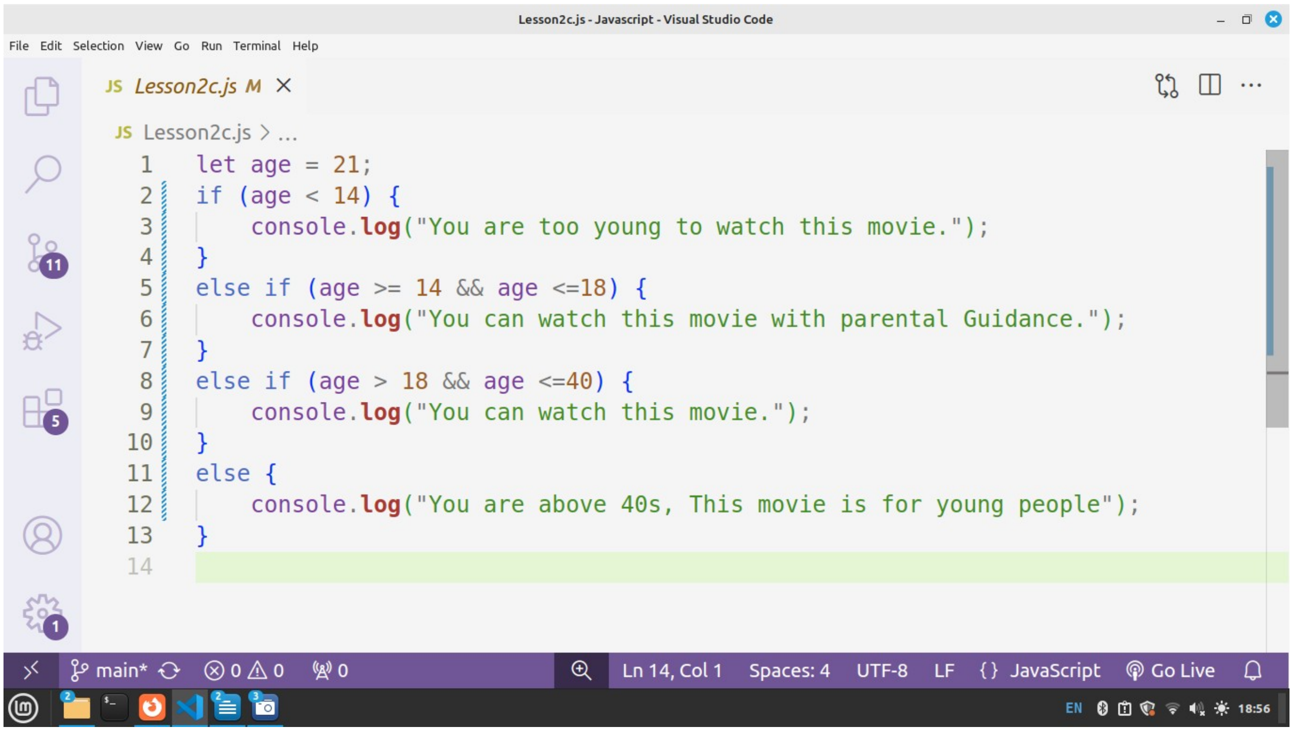
Syntax:

```
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 - Next


Lesson2c.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Lesson2c.js M
Lesson2c.js > ...
main*
Ln 14, Col 1
Spaces: 4
UTF-8
LF
JavaScript
Go Live

```javascript
let age = 21;
if (age < 14) {
    console.log("You are too young to watch this movie.");
}
else if (age >= 14 && age <=18) {
    console.log("You can watch this movie with parental Guidance.");
}
else if (age > 18 && age <=40) {
    console.log("You can watch this movie.");
}
else {
    console.log("You are above 40s, This movie is for young people");
}
```

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.

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 distance table

Distance Table:

Distance	**Amount To Pay**
0 - 100	Pay 5 USD
101 - 500	Pay 10 USD
501 - 1000	Pay 20 USD
1001 and above	Pay 40 USD

Task:

The students will write a JS program that uses an **if-else if- else statement** to calculate the Amount to Pay based on the distance traveled .

Iterative/Looping Statements

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

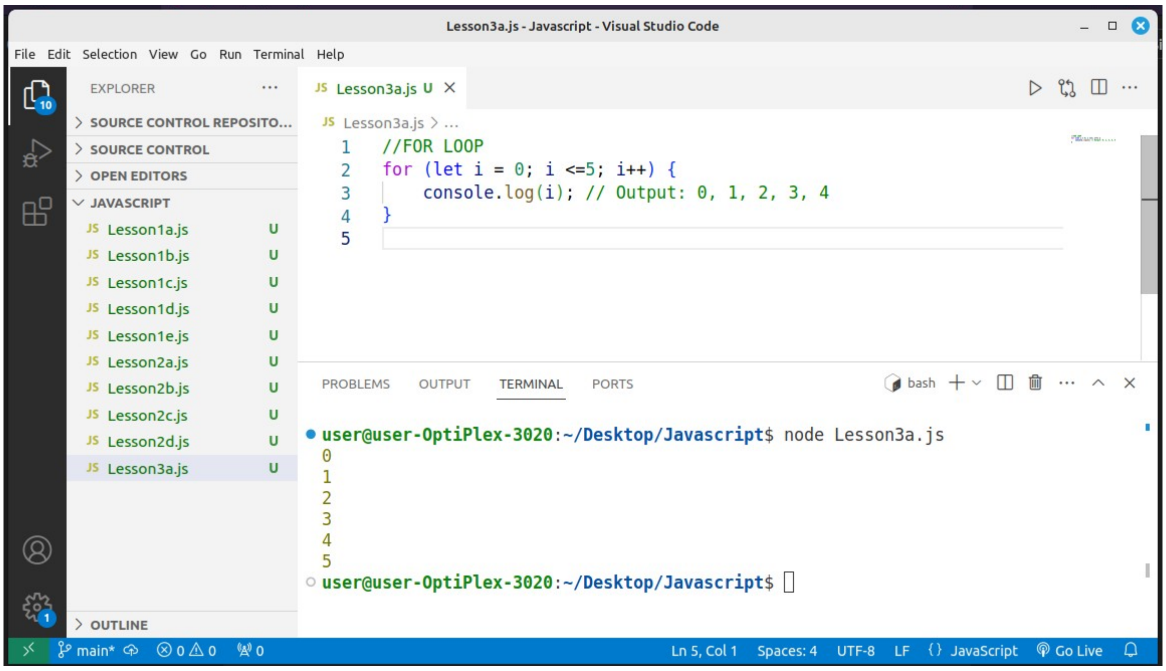
for Loop

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

```
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



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

```

```
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.

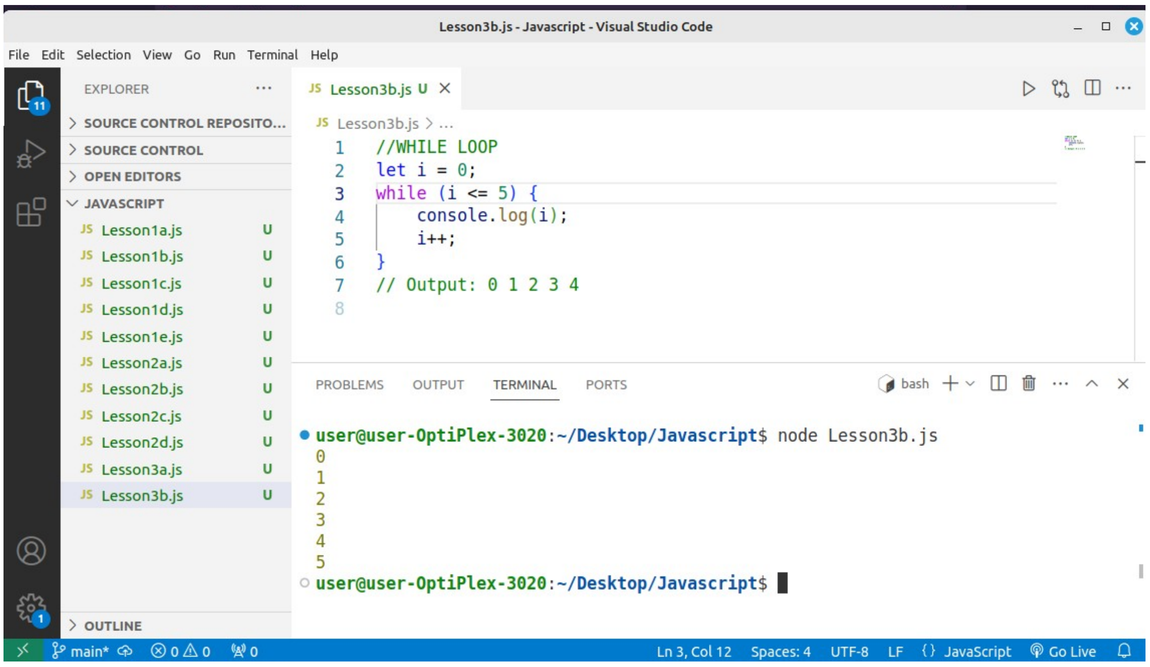
While loop

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

Syntax:

```
while (condition) {

    // code to be executed

}
```

Example



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

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

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.

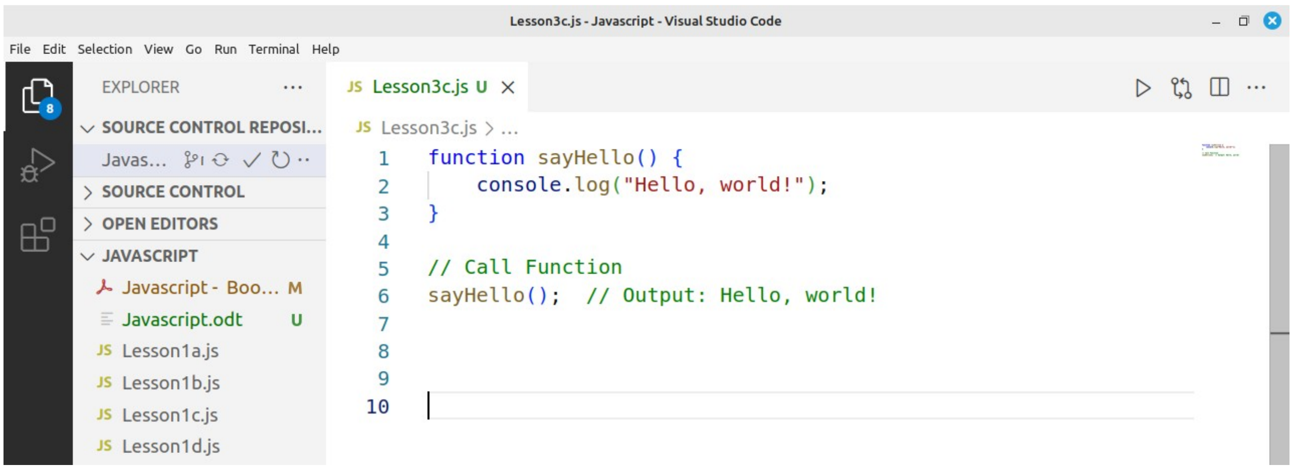
Javascript Functions

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:

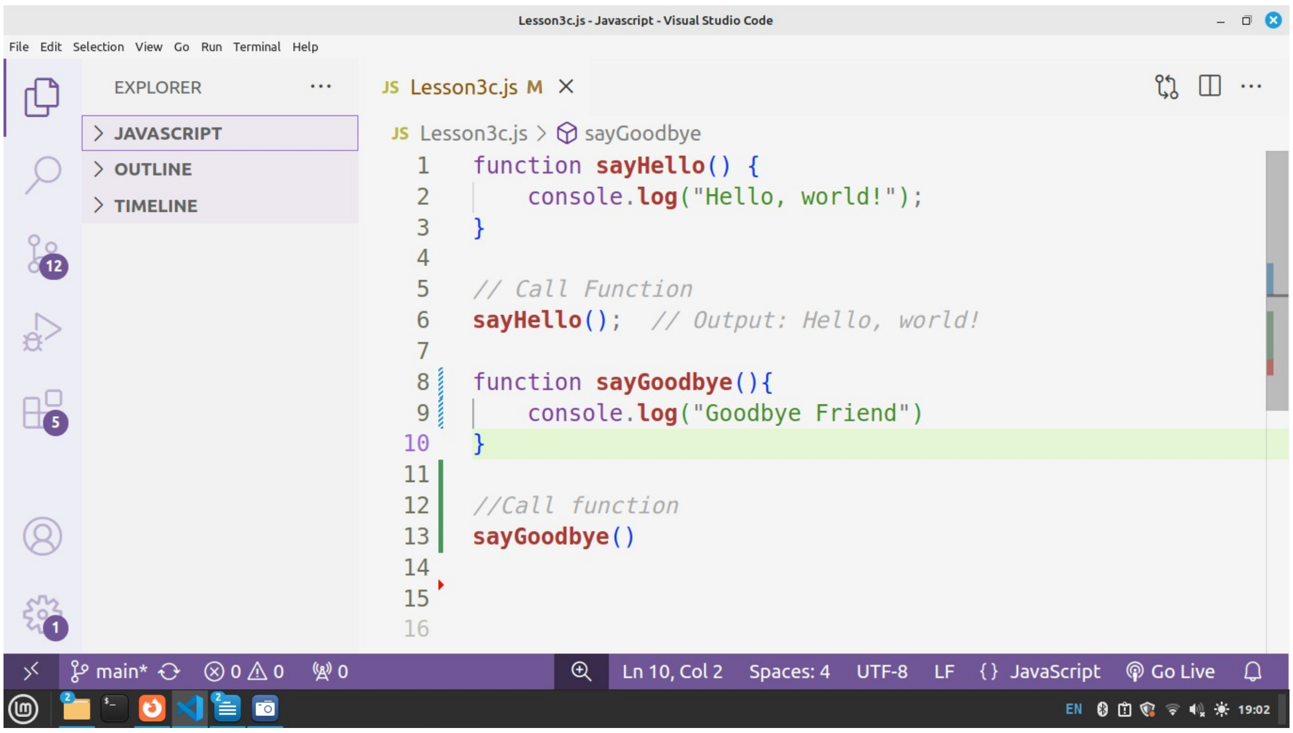


```
function sayHello() {
    console.log("Hello, world!");
}

// Call Function
sayHello();  // Output: Hello, world!
```

In above example:

- The function sayHello doesn't take any parameters/values
- When called, it prints "Hello, world!" to the console.
- Above function has name sayHello(), its called a **named function.**

Below we add another function to add two numbers
In **Lesson3c.js,** Add another Function named **sayGoodbye()**


Lesson3c.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson3c.js M
Lesson3c.js > sayGoodbye
main*
Ln 10, Col 2 Spaces: 4 UTF-8 LF {} JavaScript Go Live

```
function sayHello() {
    console.log("Hello, world!");
}

// Call Function
sayHello();  // Output: Hello, world!

function sayGoodbye(){
    console.log("Goodbye Friend")
}

//Call function
sayGoodbye()
```

Task:

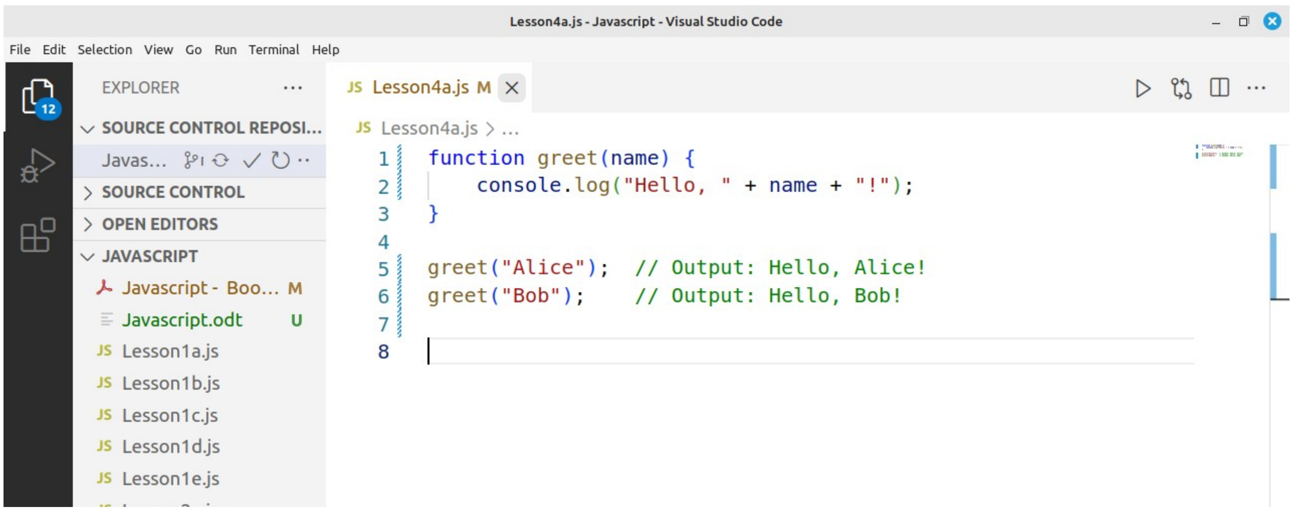
1. Students to create a function named sayGoodmorning(), the function to print/log a message “Goodmorning Dear!”. Call this function

2. Students to create a function named myCountry(), the function to print/log 5 messages explaining about myCountry. Call this function.

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:

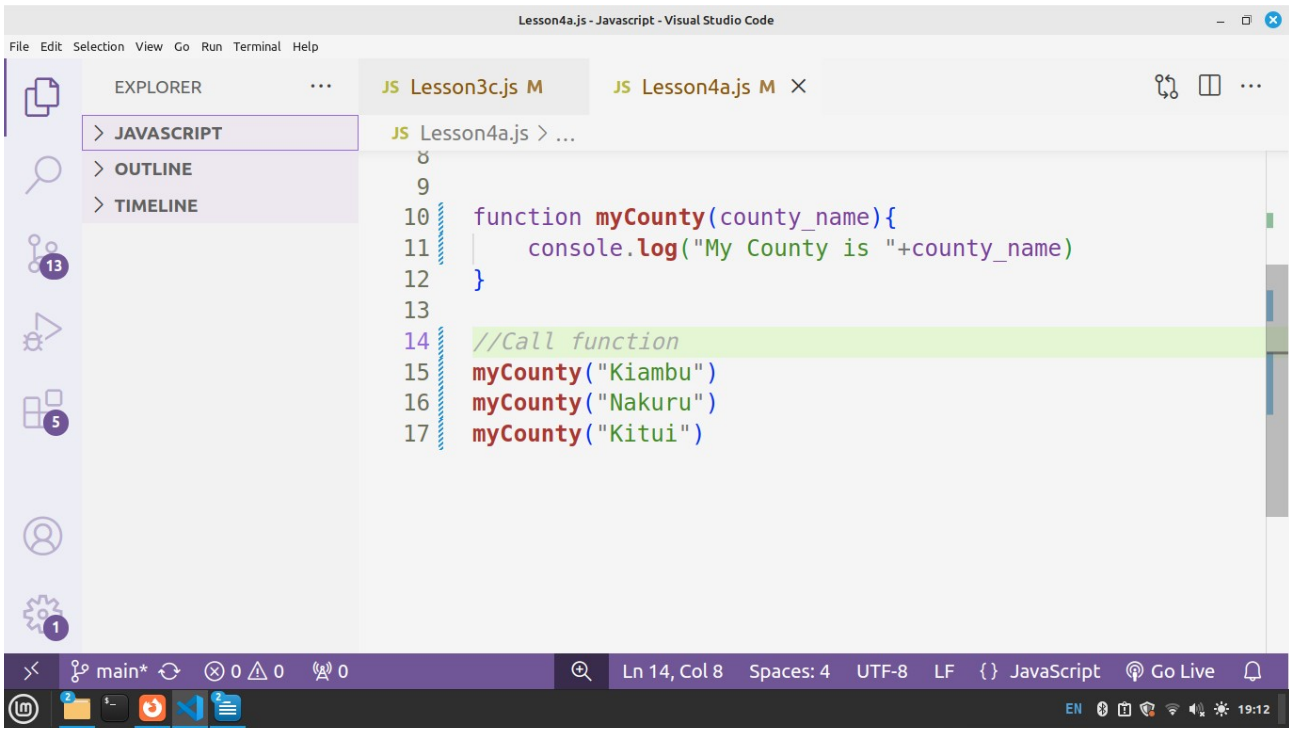


```
function greet(name) {
    console.log("Hello, " + name + "!");
}

greet("Alice");  // Output: Hello, Alice!
greet("Bob");    // Output: Hello, Bob!

```

In this example:

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

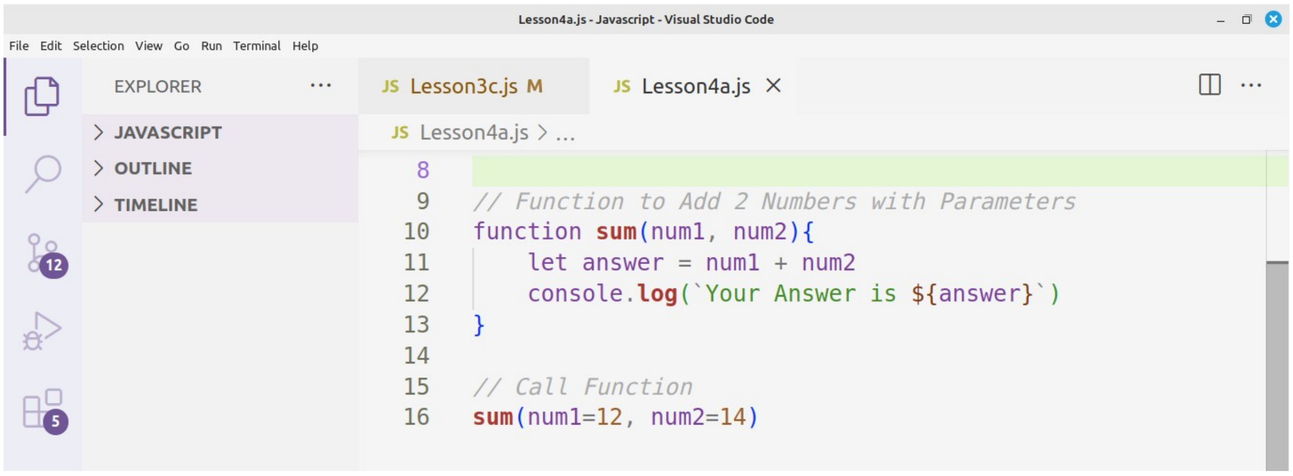
Lets redo the sum example with Parameters , Add myCounty() **function**.



```javascript
function myCounty(county_name){
    console.log("My County is "+county_name)
}

//Call function
myCounty("Kiambu")
myCounty("Nakuru")
myCounty("Kitui")
```

A function `myCounty` is defined with one parameter `county_name`.

- Inside the function, console.log prints "My County is " followed by county_name.
- The function is called three times with different arguments: "**Kiambu**", "**Nakuru**", and "**Kitui**".


Lesson4a.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson3c.js M
Lesson4a.js

```javascript

// Function to Add 2 Numbers with Parameters
function sum(num1, num2){
    let answer = num1 + num2
    console.log(`Your Answer is ${answer}`)
}

// Call Function
sum(num1=12, num2=14)
```

Here's a brief explanation of the updated code:

- **Function with Parameters**: function sum(num1, num2) defines a function that takes two parameters, num1 and num2.
- **Addition**: let answer = num1 + num2 adds the values of num1 and num2 and stores the result in answer.
- **Output**: console.log(\Your Answer is ${answer}`)` prints the result to the console using string interpolation.
- **Function Call**: sum(num1=12, num2=14) calls the function and passes 12 as num1 and 14 as num2, resulting in the output Your Answer is 26.

Task:

1. Students to do a function to subtraction two numbers

2. Students to do a function to multiply two numbers

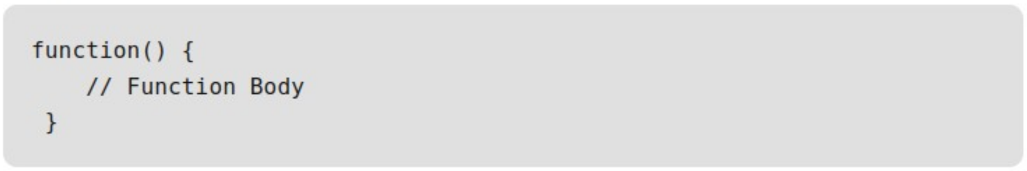
What are Anonymous Functions?

An anonymous function is simply a function that does **not have a name**.

In JavaScript, you normally use the **function keyword followed by a name** to declare a function. However, in an anonymous function, the name is **omitted.**

Syntax

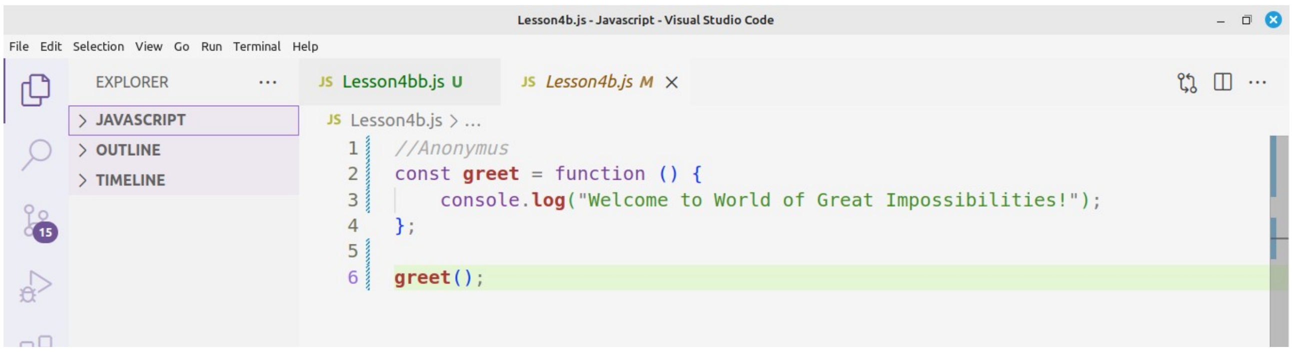
The below-enlightened syntax illustrates the declaration of an anonymous function using the normal declaration:



```
function() {
    // Function Body
 }
```

Example 1:

Defining an anonymous function that prints a message to the console. The function is then stored in the ***greet*** variable. We can call the function by invoking ***greet().***

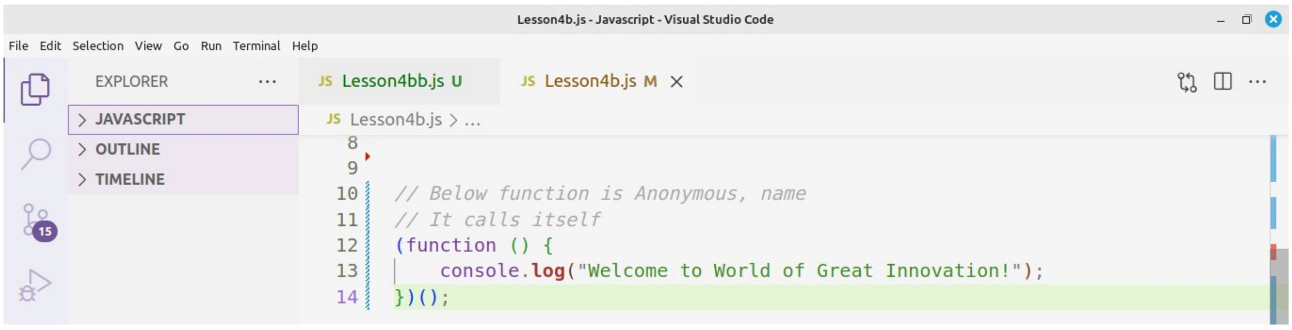

Lesson4b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson4bb.js U
Lesson4b.js M
Lesson4b.js > ...

```
//Anonymus
const greet = function () {
    console.log("Welcome to World of Great Impossibilities!");
};

greet();
```

Self-Executing Anonymous Functions

Another common use of anonymous functions is to create self-executing functions (also known as IIFE – Immediately Invoked Function Expressions). These functions run immediately after they are defined.

Example 4: Creating a self-executing function.


Lesson4b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson4bb.js U
Lesson4b.js M
Lesson4b.js > ...

```


// Below function is Anonymous, name
// It calls itself
(function () {
    console.log("Welcome to World of Great Innovation!");
})();
```

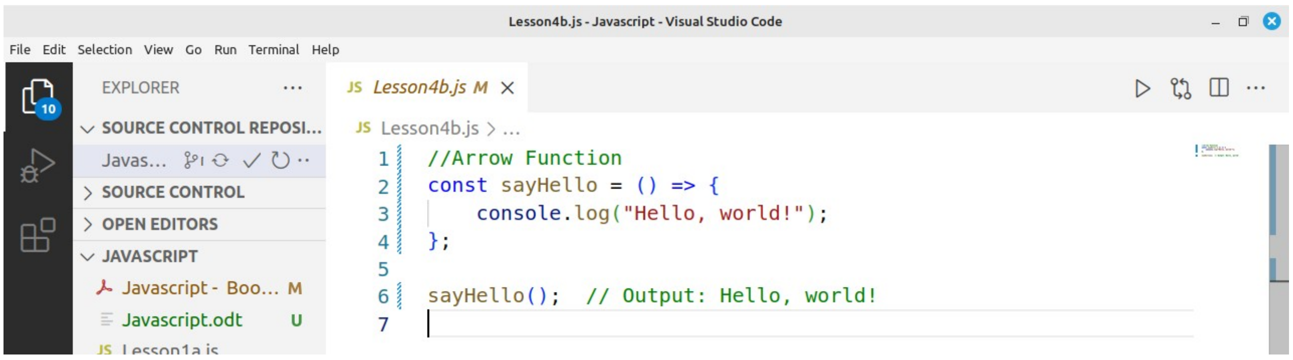
Arrow Functions

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 - Next:



```
//Arrow Function
const sayHello = () => {
    console.log("Hello, world!");
};

sayHello();  // Output: Hello, world!

```

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!".

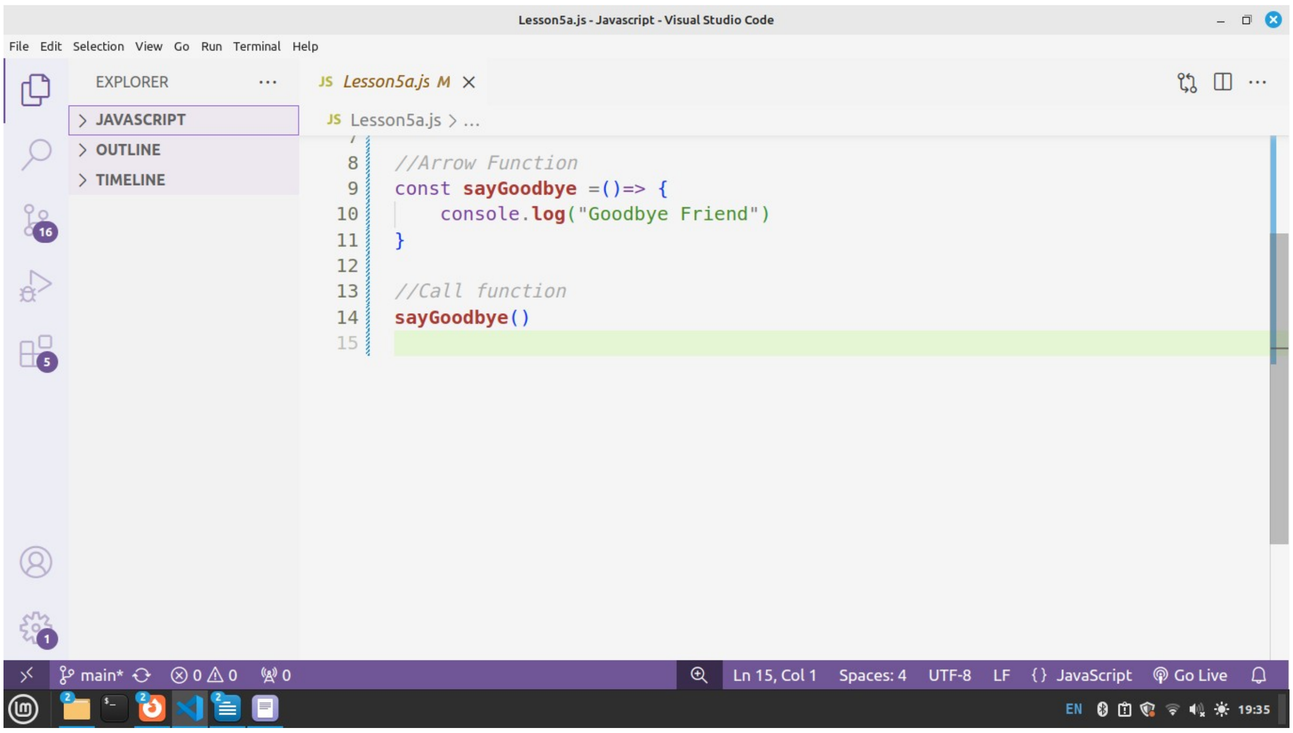
To compare above arrow function with older functions we did in **Lesson3c.js,** the difference is the function definition.

```
function sayHello() {
    console.log("Hello, world!");
}
```

Below is the Converted to Arrow function and looks like below. NB: Only the function definition is Changed

```
const sayHello = () => {
    console.log("Hello, world!");
    }
```

Lets now try to create a function **sayGoodbye function** with Arrow function Approach.



```
//Arrow Function
const sayGoodbye =()=> {
    console.log("Goodbye Friend")
}

//Call function
sayGoodbye()
```

Lesson5a.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson5a.js M
Lesson5a.js > ...
main*
Ln 15, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live
EN 19:35

Task:

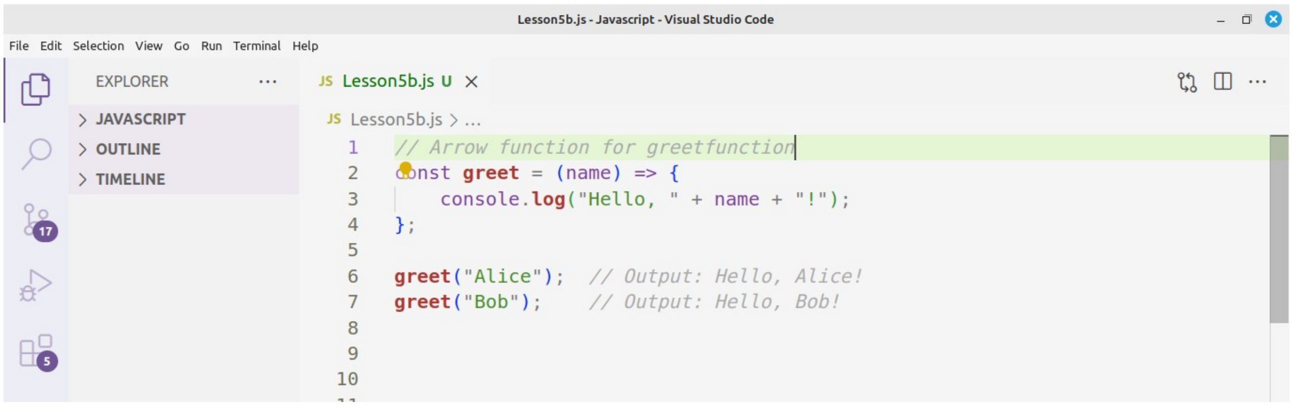
1. Students to create a arrow function named sayGoodmorning(), the function to print/log a message “Goodmorning Dear!”. Call this function

2. Students to create an arrow function named myCountry(), the function to print/log 5 messages explaining about myCountry. Call this function.

Arrow Function with Parameters:

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

Example1 - Next:

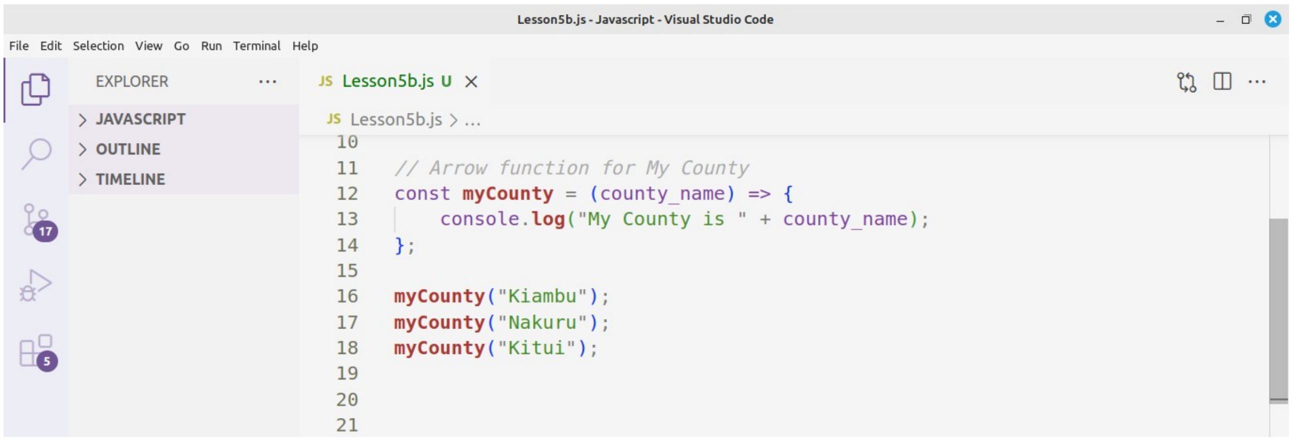

Lesson5b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson5b.js U

```
// Arrow function for greetfunction
const greet = (name) => {
    console.log("Hello, " + name + "!");
};

greet("Alice");  // Output: Hello, Alice!
greet("Bob");    // Output: Hello, Bob!
```

In this case:

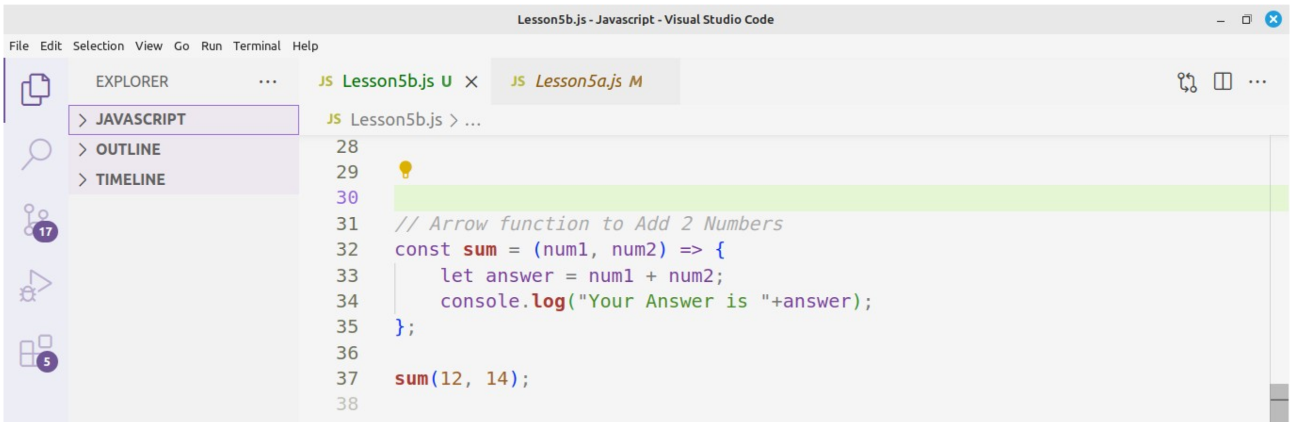
- The arrow function greet takes one parameter, name, and prints "Hello, <name>!".

Example2:


Lesson5b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson5b.js U

```
// Arrow function for My County
const myCounty = (county_name) => {
    console.log("My County is " + county_name);
};

myCounty("Kiambu");
myCounty("Nakuru");
myCounty("Kitui");
```

Example3:


Lesson5b.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson5b.js U
Lesson5a.js M

```


// Arrow function to Add 2 Numbers
const sum = (num1, num2) => {
    let answer = num1 + num2;
    console.log("Your Answer is "+answer);
};

sum(12, 14);
```

Brief Explanation:

- **Arrow Function**: The function is defined as an arrow function: const sum = (num1, num2) => { ... }.
- **Parameters**: num1 and num2 are passed as parameters to the function, allowing dynamic input values.
- **Addition**: The sum of num1 and num2 is calculated and stored in answer.
- **Output**: The result is printed to the console using template literals (console.log("Your Answer is "+answer).
- **Function Call**: sum(12, 45) calls the function with 12 as num1 and 45 as num2, and prints Your Answer is 57.

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

1. Students to do a function to subtraction two numbers

2. Students to do a function to multiply two numbers

3. 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:

Simple Interest (SI) = (Principal * Rate * 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.

Understanding JavaScript Imports, Exports

In modern JavaScript development, modular programming has become a key practice to write clean, maintainable, and reusable code. One of the most powerful features of JavaScript for modularity is **imports** and **exports**. In this chapter, we'll explore how to import different kinds of files and modules in JavaScript.

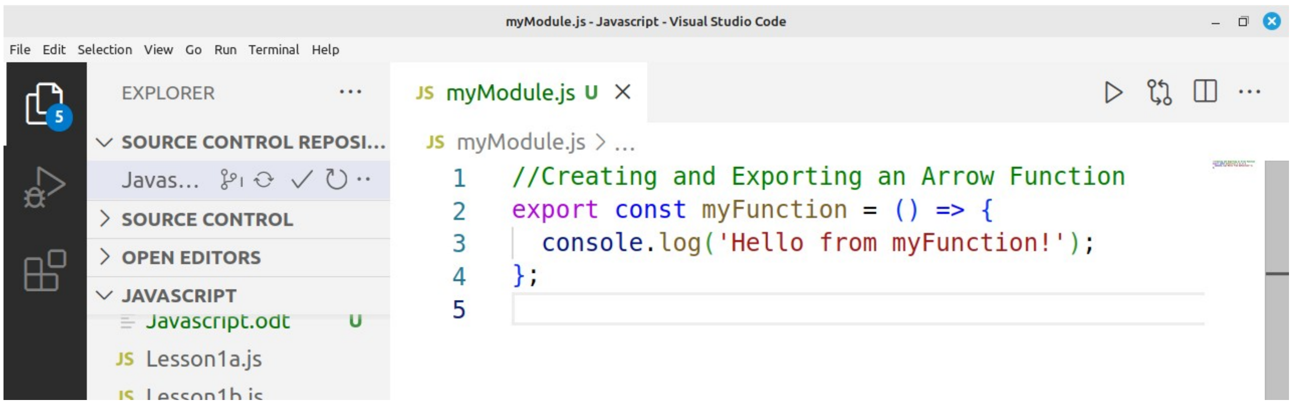
1. What are Imports and Exports?

Before diving into how to import files, it's important to understand **exports**. In JavaScript, the **export** keyword is used to make functions, objects, or variables available for other files to use.

The **import** keyword is used in other JavaScript files to bring these exports into their own scope.

The **import** keyword is used in other JavaScript files to bring the exported Files into another File.

Create a File named **myModule.js** and write below code.

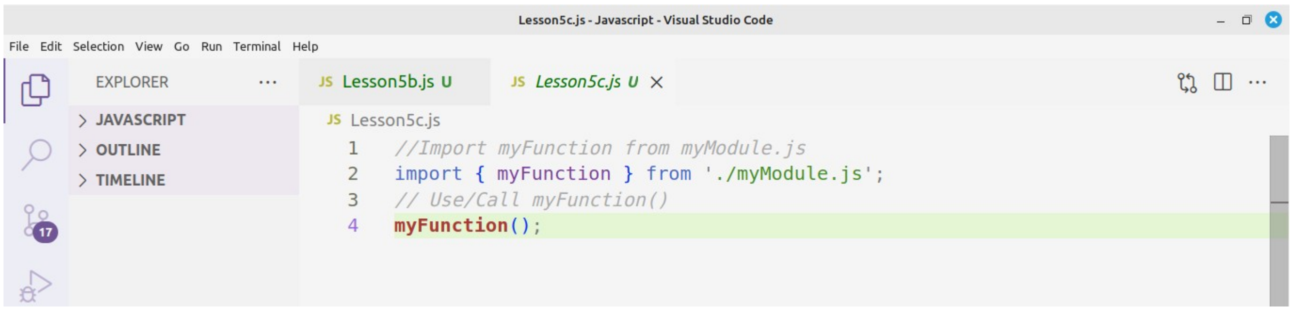


```
//Creating and Exporting an Arrow Function
export const myFunction = () => {
  console.log('Hello from myFunction!');
};
```

In this example, the function **myFunction** is exported from a file called **myModule.js**, Now we can **import** it in another File.

myModule.js is known as a module as it contains JS functions

Next, Create another File and named it **Lesson5a.js** and import the **myFunction** from **myModule.js** File


Lesson5c.js - Javascript - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
JAVASCRIPT
OUTLINE
TIMELINE
Lesson5b.js U
Lesson5c.js U
Lesson5c.js
17

```
//Import myFunction from myModule.js
import { myFunction } from './myModule.js';
// Use/Call myFunction()
myFunction();
```

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?

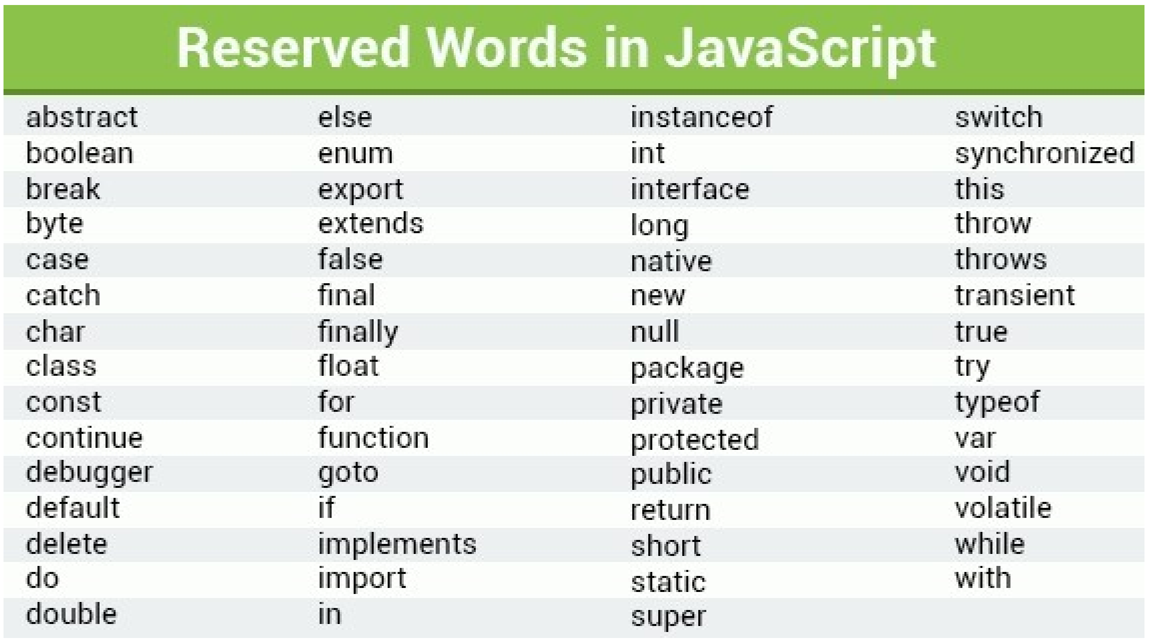
19. Write an arrow function **square** that takes a number as a parameter and logs its square

20. Create an arrow function **isEven** that takes a number as a parameter and logs a message if number id ODD or EVEN

21. Write an arrow function **celsiusToFahrenheit** that takes a temperature in 22. 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