1. Introduction to JavaScript (10 minutes)

- **Overview of JavaScript**: Explain what JavaScript is, its history, and where it is commonly used (web development, server-side, etc.).
- **Setup**: Briefly show how to set up a basic HTML file and include JavaScript using the `<script>` tag.

2. Basic Syntax and Data Types (20 minutes)

- **Variables**: Introduction to `var`, `let`, and `const`. Discuss differences and best practices.
- **Data Types**: Explain primitive data types (`string`, `number`, `boolean`, `undefined`, `null`, `symbol`, and `bigint`).
- **Examples**: Simple console log examples showing different data types and variable declarations.

Example 1: Variable Declarations

```
// Declaring variables using var (older way)
var name = 'John Doe';
console.log(name); // Output: John Doe

// Declaring variables using let (preferred for variables that may change)
let age = 30;
console.log(age); // Output: 30

// Declaring constants using const (preferred for variables that should
not change)
const birthYear = 1994;
console.log(birthYear); // Output: 1994
```

Example 2: Differences between var, let, and const

```
// var is function-scoped and can be redeclared
var color = 'blue';
var color = 'red';
console.log(color); // Output: red

// let is block-scoped and cannot be redeclared within the same scope
let food = 'pizza';
// let food = 'pasta'; // This would throw an error
```

```
food = 'pasta'; // This is allowed because we are reassigning, not
redeclaring
console.log(food); // Output: pasta

// const is block-scoped and cannot be redeclared or reassigned
const pi = 3.14;
// pi = 3.14159; // This would throw an error
console.log(pi); // Output: 3.14
```

2. Data Types

Example 3: Different Data Types

```
// String
const greeting = 'Hello, world!';
console.log(greeting); // Output: Hello, world!

// Number
const year = 2024;
console.log(year); // Output: 2024

// Boolean
const isJavaScriptFun = true;
console.log(isJavaScriptFun); // Output: true

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

// Null
const noValue = null;
console.log(noValue); // Output: null

// Symbol (ES6+)
const uniqueId = Symbol('id');
console.log(uniqueId); // Output: Symbol(id)

// BigInt (ES2020)
const bigNumber = BigInt(123456789012345678901234567890);
console.log(bigNumber); // Output: 123456789012345678901234567890n
```

3. Type Checking

Example 4: Checking Data Types

```
// Checking the type of a variable using typeof
console.log(typeof greeting); // Output: string
console.log(typeof year); // Output: number
console.log(typeof isJavaScriptFun); // Output: boolean
console.log(typeof car); // Output: undefined
console.log(typeof noValue); // Output: object (this is a known quirk in
JavaScript)
console.log(typeof uniqueId); // Output: symbol
console.log(typeof bigNumber); // Output: bigint
```

3. Operators and Expressions (15 minutes)

- **Arithmetic Operators**: `+`, `-`, `*`, `/`, `%`, and `**`.
- **Assignment Operators**: `=`, `+=`, `-=`, etc.
- **Comparison Operators**: `==`, `===`, `!=`, `!==`, `>`, `<`, `>=`, `<=`.
- **Logical Operators**: `&&`, `||`, `!`.
- **Examples**: Write small snippets of code that demonstrate the use of these operators.

1. Arithmetic Operators

Arithmetic operators perform mathematical operations on numbers.

Example 1: Basic Arithmetic Operations

```
// Addition
const sum = 10 + 5;
console.log('10 + 5 =', sum); // Output: 10 + 5 = 15

// Subtraction
const difference = 10 - 5;
console.log('10 - 5 =', difference); // Output: 10 - 5 = 5

// Multiplication
const product = 10 * 5;
console.log('10 * 5 =', product); // Output: 10 * 5 = 50
```

```
// Division
const quotient = 10 / 5;
console.log('10 / 5 =', quotient); // Output: 10 / 5 = 2

// Modulus (Remainder)
const remainder = 10 % 3;
console.log('10 % 3 =', remainder); // Output: 10 % 3 = 1

// Exponentiation
const power = 2 ** 3;
console.log('2 ** 3 =', power); // Output: 2 ** 3 = 8
```

2. Assignment Operators

Assignment operators assign values to variables and can also perform operations.

Example 2: Using Assignment Operators

```
let number = 10; // Initial value assignment
console.log('Initial number:', number); // Output: Initial number: 10

// Add and assign
number += 5; // Equivalent to: number = number + 5;
console.log('After += 5:', number); // Output: After += 5: 15

// Subtract and assign
number -= 3; // Equivalent to: number = number - 3;
console.log('After -= 3:', number); // Output: After -= 3: 12

// Multiply and assign
number *= 2; // Equivalent to: number = number * 2;
console.log('After *= 2:', number); // Output: After *= 2: 24

// Divide and assign
number /= 4; // Equivalent to: number = number / 4;
console.log('After /= 4:', number); // Output: After /= 4: 6

// Modulus and assign
number %= 5; // Equivalent to: number = number % 5;
console.log('After %= 5:', number); // Output: After %= 5: 1
```

3. Comparison Operators

Comparison operators compare two values and return a boolean (`true` or `false`).

Example 3: Using Comparison Operators

```
// Equal to
console.log('10 == "10":', 10 == '10'); // Output: 10 == "10": true (loose
equality)

// Strict equal to
console.log('10 === "10":', 10 === '10'); // Output: 10 === "10": false
(strict equality)

// Not equal to
console.log('10 != "5":', 10 != '5'); // Output: 10 != "5": true (loose
inequality)

// Strict not equal to
console.log('10 !== "10":', 10 !== '10'); // Output: 10 !== "10": true
(strict inequality)

// Greater than
console.log('10 > 5:', 10 > 5); // Output: 10 > 5: true

// Less than
console.log('10 < 5:', 10 < 5); // Output: 10 < 5: false

// Greater than or equal to
console.log('10 >= 10:', 10 >= 10); // Output: 10 >= 10: true

// Less than or equal to
console.log('10 <= 5:', 10 <= 5); // Output: 10 <= 5: false
```

4. Logical Operators

Logical operators are used to combine or invert boolean values.

Example 4: Using Logical Operators

```
// AND operator
console.log('true && false:', true && false); // Output: true && false: false

// OR operator
console.log('true || false:', true || false); // Output: true || false: true

// NOT operator
console.log('!true:', !true); // Output: !true: false

// Combining logical operators
const age = 20;
const hasDrivingLicense = true;

// Checking multiple conditions with AND (&&)
console.log('Can drive?', age >= 18 && hasDrivingLicense); // Output: Can drive? true

// Checking multiple conditions with OR (||)
const hasPassport = false;
console.log('Can travel abroad?', hasDrivingLicense || hasPassport); // Output: Can travel abroad? true
```

4. Control Flow (20 minutes)

- **Conditional Statements**: `if`, `else if`, `else`, `switch`.
- **Loops**: `for`, `while`, `do...while`.
- **Examples**: Show examples of conditional statements and loops in practical scenarios, like checking a user's input or iterating over arrays.

1. Conditional Statements

Conditional statements allow the program to execute different blocks of code based on certain conditions.

Example 1: Using `if`, `else if`, and `else`

```
// Checking if a number is positive, negative, or zero
const number = 10;

if (number > 0) {
    console.log('The number is positive.'); // Output: The number is
positive.
} else if (number < 0) {
    console.log('The number is negative.');
} else {
    console.log('The number is zero.');
}
```

Example 2: Using switch

```
// Using switch to check the day of the week
const day = 3;
let dayName;

switch (day) {
    case 1:
        dayName = 'Monday';
        break;
    case 2:
        dayName = 'Tuesday';
        break;
    case 3:
        dayName = 'Wednesday';
        break;
    case 4:
        dayName = 'Thursday';
        break;
    case 5:
        dayName = 'Friday';
        break;
    case 6:
        dayName = 'Saturday';
        break;
    case 7:
        dayName = 'Sunday';
        break;
```

```
    default:
        dayName = 'Invalid day';
}

console.log('The day is:', dayName); // Output: The day is: Wednesday
```

JavaScript Fundamentals Part 2

2. Loops

Loops allow you to execute a block of code multiple times.

Example 3: Using for Loop

```
// Using a for loop to iterate over an array of numbers and calculate the
sum
const numbers = [1, 2, 3, 4, 5];
let sum = 0;

for (let i = 0; i < numbers.length; i++) {
    sum += numbers[i];
}

console.log('Sum of the array elements:', sum); // Output: Sum of the
array elements: 15
```

Example 4: Using while Loop

```
// Using a while loop to find the first number divisible by 3 and 5
let n = 1;

while (true) {
    if (n % 3 === 0 && n % 5 === 0) {
        console.log('First number divisible by 3 and 5 is:', n); //
Output: First number divisible by 3 and 5 is: 15
        break;
    }
    n++;
}
```

Example 5: Using `do...while` Loop

```
// Using a do...while loop to ask for a user's input until they provide a non-empty string
let userInput;
do {
    userInput = prompt('Enter a non-empty string:');
} while (!userInput);

console.log('You entered:', userInput); // Output depends on user input
```

5. Functions (20 minutes)

- **Function Declaration**: Explain how to declare functions using the `function` keyword.
- **Function Expressions and Arrow Functions**: Show the difference between function expressions and arrow functions.
- **Parameters and Return Values**: Demonstrate how functions can take parameters and return values.
- **Examples**: Write functions for common tasks like adding two numbers, converting temperatures, etc.

1. Function Declaration

A **function declaration** defines a function with a specified name using the `function` keyword.

Example 1: Function Declaration

```
// Function declaration to greet a user
function greet() {
    console.log('Hello, world!');
}

// Calling the function
greet(); // Output: Hello, world!
```

2. Function Expressions and Arrow Functions

A **function expression** defines a function as part of an expression. **Arrow functions** are a shorter syntax for writing function expressions, introduced in ES6.

Example 2: Function Expression

```
// Function expression to add two numbers
const add = function (a, b) {
```

```
    return a + b;
};
// Calling the function
console.log('Addition:', add(5, 3)); // Output: Addition: 8
```

Example 3: Arrow Function

```
// Arrow function to subtract two numbers
const subtract = (a, b) => a - b;

// Calling the function
console.log('Subtraction:', subtract(10, 4)); // Output: Subtraction: 6
```

Difference between Function Expressions and Arrow Functions:

- **Syntax**: Arrow functions provide a more concise syntax.
- **`this` Context**: Arrow functions do not have their own `this` context; they inherit `this` from the parent scope, which is different from regular function expressions.

3. Parameters and Return Values

Functions can take **parameters** (inputs) and return **values** (outputs).

Example 4: Function with Parameters and Return Value

```
// Function to convert Celsius to Fahrenheit
function convertCelsiusToFahrenheit(celsius) {
    return (celsius * 9) / 5 + 32;
}

// Calling the function with an argument
const fahrenheit = convertCelsiusToFahrenheit(30);
console.log('30 degrees Celsius is', fahrenheit, 'degrees Fahrenheit.');
// Output: 30 degrees Celsius is 86 degrees Fahrenheit.
```

Example 5: Arrow Function with Multiple Parameters

```
// Arrow function to calculate the area of a rectangle
const calculateArea = (width, height) => width * height;

// Calling the function
console.log('Area of the rectangle:', calculateArea(5, 10)); // Output:
Area of the rectangle: 50
```

Example 6: Function with Default Parameters

```
// Function with default parameters to calculate the power of a number
function power(base, exponent = 2) {
    return base ** exponent;
}

// Calling the function without providing the exponent argument
console.log('5 to the power of 2 is:', power(5)); // Output: 5 to the
power of 2 is: 25

// Calling the function with both arguments
console.log('3 to the power of 3 is:', power(3, 3)); // Output: 3 to the
power of 3 is: 27
```

Explanation:

1. **Function Declaration**:
    - Functions declared with the `function` keyword are hoisted to the top of their scope, meaning they can be called before they are defined in the code.
2. **Function Expressions and Arrow Functions**:
    - **Function Expressions**: These are functions defined inside an expression and are not hoisted, meaning they can only be called after they are defined.
    - **Arrow Functions**: These provide a shorter syntax for writing functions and do not have their own `this` context, which makes them useful for certain situations like methods inside objects or callback functions.
3. **Parameters and Return Values**:
    - **Parameters**: Functions can take one or more parameters to pass information into them.
    - **Return Values**: Functions can return a value using the `return` statement. If no `return` statement is provided, the function returns `undefined` by default.
    - **Default Parameters**: You can provide default values for parameters, which are used if no arguments are passed.

6. Arrays and Array Methods (20 minutes)

- **Creating Arrays**: How to create arrays using literals and constructors.
- **Accessing Elements**: Accessing, modifying, and adding elements to arrays.

- **Common Array Methods**: `push`, `pop`, `shift`, `unshift`, `slice`, `splice`, `forEach`, `map`, `filter`, `reduce`.
- **Examples**: Show practical examples of using array methods to manipulate data.

1. Creating Arrays

Arrays in JavaScript can be created using array literals or the `Array` constructor.

Example 1: Creating Arrays

```
// Creating an array using an array literal
const fruits = ['apple', 'banana', 'cherry'];
console.log('Fruits:', fruits); // Output: Fruits: ['apple', 'banana',
'cherry']

// Creating an array using the Array constructor
const numbers = new Array(1, 2, 3, 4, 5);
console.log('Numbers:', numbers); // Output: Numbers: [1, 2, 3, 4, 5]
```

2. Accessing and Modifying Elements

You can access array elements using their index and modify them directly.

Example 2: Accessing and Modifying Elements

```
// Accessing elements using index
console.log('First fruit:', fruits[0]); // Output: First fruit: apple

// Modifying an element
fruits[1] = 'blueberry';
console.log('Modified Fruits:', fruits); // Output: Modified Fruits:
['apple', 'blueberry', 'cherry']

// Adding an element
fruits[3] = 'date';
console.log('Added Element:', fruits); // Output: Added Element: ['apple',
'blueberry', 'cherry', 'date']

// Checking the length of an array
console.log('Number of fruits:', fruits.length); // Output: Number of
fruits: 4
```

3. Common Array Methods

Array methods provide various ways to manipulate and work with arrays.

Example 3: Using `push` and `pop`

```
// Adding elements to the end of an array using push
fruits.push('elderberry');
console.log('After push:', fruits); // Output: After push: ['apple',
'blueberry', 'cherry', 'date', 'elderberry']

// Removing the last element using pop
const lastFruit = fruits.pop();
console.log('Popped Fruit:', lastFruit); // Output: Popped Fruit:
elderberry
console.log('After pop:', fruits); // Output: After pop: ['apple',
'blueberry', 'cherry', 'date']
```

Example 4: Using `shift` and `unshift`

```
// Adding elements to the beginning of an array using unshift
fruits.unshift('fig');
console.log('After unshift:', fruits); // Output: After unshift: ['fig',
'apple', 'blueberry', 'cherry', 'date']

// Removing the first element using shift
const firstFruit = fruits.shift();
console.log('Shifted Fruit:', firstFruit); // Output: Shifted Fruit: fig
console.log('After shift:', fruits); // Output: After shift: ['apple',
'blueberry', 'cherry', 'date']
```

Example 5: Using `slice` and `splice`

```
// Using slice to create a shallow copy of part of an array
const someFruits = fruits.slice(1, 3);
console.log('Sliced Fruits:', someFruits); // Output: Sliced Fruits:
['blueberry', 'cherry']
```

```
// Using splice to remove and add elements
fruits.splice(2, 1, 'dragonfruit', 'elderberry'); // Removes 1 element at index 2 and adds 'dragonfruit' and 'elderberry'
console.log('After splice:', fruits); // Output: After splice: ['apple', 'blueberry', 'dragonfruit', 'elderberry', 'date']
```

Example 6: Using `forEach`

```
// Using forEach to iterate over an array
fruits.forEach(function(fruit, index) {
    console.log(`Fruit at index ${index}:`, fruit);
});
// Output:
// Fruit at index 0: apple
// Fruit at index 1: blueberry
// Fruit at index 2: dragonfruit
// Fruit at index 3: elderberry
// Fruit at index 4: date
```

Example 7: Using `map`

```
// Using map to create a new array with modified elements
const upperCaseFruits = fruits.map(fruit => fruit.toUpperCase());
console.log('Uppercase Fruits:', upperCaseFruits); // Output: Uppercase Fruits: ['APPLE', 'BLUEBERRY', 'DRAGONFRUIT', 'ELDERBERRY', 'DATE']
```

Example 8: Using `filter`

```
// Using filter to create a new array with elements that match a condition
const longNamedFruits = fruits.filter(fruit => fruit.length > 5);
console.log('Long-named Fruits:', longNamedFruits); // Output: Long-named Fruits: ['blueberry', 'dragonfruit', 'elderberry']
```

Example 9: Using `reduce`

```
// Using reduce to calculate the total length of all fruit names
const totalLength = fruits.reduce((accumulator, fruit) => accumulator + fruit.length, 0);
```

```
console.log('Total length of all fruit names:', totalLength); // Output:
Total length of all fruit names: 33
```

Explanation:

1. **Creating Arrays**:
    - Arrays can be created using literals ([]) or the Array constructor. It is usually preferred to use literals as they are more concise.
2. **Accessing and Modifying Elements**:
    - Elements in an array are accessed using their index, starting at 0. You can modify an element directly by assigning a new value to a specific index.
3. **Common Array Methods**:
    - **push and pop**: Used to add or remove elements from the end of an array.
    - **shift and unshift**: Used to add or remove elements from the beginning of an array.
    - **slice**: Returns a shallow copy of a portion of an array into a new array.
    - **splice**: Can be used to add, remove, or replace elements in an array.
    - **forEach**: Executes a function for each element in the array.
    - **map**: Creates a new array with the results of calling a function on every element.
    - **filter**: Creates a new array with all elements that pass a test defined in a function.
    - **reduce**: Reduces the array to a single value by executing a function for each element.