A COMPLETE JAVASCRIPT

NOTES BY

- *Mohammad Faizan.*

Here you go and Check it out and Expend your Coding Journey With Mohammad Faizan.

Scroll Down

Here's some topics in the NOTES:

CONTENT OF JAVASCRIPT :

Content Page no.

Scroll Down

JavaScript Notes

Date: 11/10/2023

Introduction to Javascript:

JavaScript is a versatile programming language primarily used for web development. Created by Brendan Eich in 1995, it enables dynamic content and interactivity in web browsers, playing a crucial role in modern web development.

JavaScript: A Brief History

JavaScript, created by **Brendan Eich in 1995 at Netscape**, has a rich history and is a pivotal language in web development.

Evolution of Names:

Initially known as **Mocha**, it underwent transformations, becoming **LiveScript** and eventually adopting the name **JavaScript**.

Purpose:

JavaScript is a dynamic, versatile **client-side scripting language**. It empowers web developers to create interactive and dynamic content within web browsers.

Key Features:

- Enables front-end interactivity

- Essential for modern web development

- Follows the evolving **ECMAScript standard**

Significance:

JavaScript plays a crucial role in developing dynamic websites and is the backbone of popular front-end frameworks like : **React and Angular**.

In summary, JavaScript's journey from its inception to its current state reflects its continuous evolution and indispensable role in shaping the interactive web experience.

Node.js: Unveiling its Evolution

Birth of Node.js:

Node.js, born in 2009, was created by **Ryan Dahl**. It marked a groundbreaking shift by bringing JavaScript to the server-side.

JavaScript Everywhere:

Node.js allows developers to use JavaScript not only for client-side but also for **server-side programming**, promoting a unified language across the entire web development stack.

Event-Driven and Asynchronous:

Built on the **V8 JavaScript runtime engine**, Node.js introduced an event-driven, **non-blocking I/O model**, enhancing efficiency and scalability.

NPM Revolution:

The advent of **Node Package Manager (NPM)** transformed how developers manage dependencies, fostering a vibrant ecosystem of open-source libraries.

Widespread Adoption:

Node.js gained rapid adoption, becoming a go-to choice for building scalable and performant server-side applications.

Continued Innovation:

Ongoing updates and contributions from the open-source community ensure Node.js remains a dynamic and influential force in the realm of web development.

In summary, Node.js has not just revolutionized server-side development but has also shaped the way developers approach building scalable and efficient applications.

Node Package Manager (NPM): A Comprehensive Journey

Inception (2009):

- **Birth of NPM**: Coexisting with Node.js, NPM was introduced in 2009.

- **Purpose**: Initially designed as a package manager for Node.js.

Expansion (2010-2015):

- **Versatility: NPM's** role extended beyond Node.js, becoming the standard for managing frontend packages.

- **Broader Ecosystem**: Adoption soared, creating a vast repository of open-source packages.

Key Features:

- **CLI Empowerment:** NPM's Command-Line Interface streamlined package installation, version control, and script execution.

- **Scalability**: NPM's non-blocking I/O model contributed to scalable server-side applications.

Evolution (2016-2022):

- **Global Impact**: Worldwide adoption as the go-to package manager for JavaScript projects.

- **Updates and Enhancements:** Continuous evolution with regular updates, security enhancements, and community-driven contributions.

Beyond Basics:

- **NPM Scripts**: Integration of scripts in package.json, facilitating custom workflows.

- **Yarn Competition:** NPM faced competition from Yarn, spurring healthy innovation.

Current Landscape (2023):

- **Community-Driven**: NPM remains a pivotal force, fostering collaboration and innovation within the JavaScript ecosystem.

- **Indispensable Tool**: Integral to modern web development, ensuring efficient package management.

In summary, NPM's journey from its inception to its current state reflects its pivotal role in shaping the JavaScript ecosystem, with continuous evolution and a global impact.

Certainly! Here's a set of steps to set up Visual Studio Code (VS Code) for JavaScript development:

1. Install Visual Studio Code:

- Download and install Visual Studio Code from the official website: [Visual Studio Code](https://code.visualstudio.com/)

2. Install Node.js:

- JavaScript development often involves using Node.js. Install it from [Node.js official website](https://nodejs.org/)

3. Open VS Code:

- After installation, open VS Code.

4. Install Extensions:

- Install relevant extensions for JavaScript development. Some popular ones are:

- **ESLint:** For code linting and style checking.

- **Prettier:** For code formatting.

- **Debugger for Chrome:** If you're working with Node.js or web applications.

5. Create a New Project:

- Open a terminal in VS Code and navigate to the folder where you want to create your project.

- Run `npm init` to initialize a new Node.js project. Follow the prompts to set up your `package.json` file.

6. Install Packages:

- Install any necessary packages using `npm install`. For example :

npm install express

7. Create JavaScript Files:

- Create your JavaScript files in the project folder.

8. Configure ESLint and Prettier:

- Create an `.eslintrc` file for ESLint configuration and a `.prettierrc` file for Prettier configuration. Customize these files based on your preferences and project requirements.

9. Configure Debugger (if needed):

- If you're using the Debugger for Chrome extension, set up your `launch.json` configuration for debugging.

10. Start Coding:

- You're all set! Start coding your JavaScript project in VS Code.

Understanding VAR , LET & CONST :-

Javascript Provides three Keywords for declaring Variables : Var, Let & Const .

Each of these has its own use cases and behaviors.

VAR : The Old Way

In the Past , Var was the primary way to declare variables in Javascript. However, it has some issues and its generally recommended to avoid using it in Modern Code.

The Problem with Var is that it doesn't have block level scope, which can lead to Unexpected Behaviour.

```
function exampleVar() {
    if (true) {
        var localVar = "I am a var
variable";
        console.log(localVar); //
Output: I am a var variable
    }

    console.log(localVar); // Output:
I am a var variable
}

exampleVar();
```

LET : The New Way

Introduced in **ES6 (ECMASCRIPT2015)** , Let is Block-Level Scoped.Making it is a safer choice for declaring Variables within a Specific block. Using let Prevents Unintentional Variable leaks and Provides better Control over scope.

```
function exampleLet() {
    if (true) {
        let localLet = "I am a let
variable";
        console.log(localLet); //
Output: I am a let variable
    }

    // Uncommenting the line below
would result in an error:
    // console.log(localLet); //
Error: localLet is not defined
}

exampleLet();
```

CONST : The Constant

Use Cont to declare Constant Variable, whose value shouldn't change after initializing. Variables declare with Const are also block-Scope like Let , but they can't re-assignedbonce declared.

```
const myConstVariable = 10;
myConstVariable = 20; // This will
cause an error
```

Primitives Data types in JS :

》 These are the like basic building block of data.

》 They can hold one piece of information at a time.

》 Examples Include : (nn,bb,ss,u)

// **NUMBER :**

let num = 42 ;

//NULL :

Let box1 = null ;

// BOOLEANS :

Let isTrue = true ;

//BigInt :

Const BigInt = 123467483929292992827617 ;

// Symbol :

Const unique symbol = symbol (" I am a Symbol ");

// Undefined :

Let box2 = Undifined ;

Non - Primitives Data types :

》 These are more like containers that can hold many pieces of information.

》 They can hold collection of data or Even Complex Structure.

》 Examples Include objects , Array (List of items) and Functions.

//Object

```
// Define an object representing a
person
const person = {
  name: "Alice",
  age: 25,
  isStudent: true,
  greet: function() {
    console.log(`Hello, my name is $
{this.name} and I'm ${this.age} years
old.`);
  }
};

// Accessing properties of the person
object
console.log(person.name); // Output:
Alice
console.log(person.age); // Output:
25
```

// Array

```
// Define an array of colors
const colors = ["red", "green",
"blue", "yellow"];

// Accessing elements of the array
console.log(colors[0]); // Output:
red
console.log(colors[2]); // Output:
blue
```

// Function

```
function divide(d1, d2) {
  // Check if the denominator is not
zero to avoid division by zero
  if (d2 !== 0) {
    return d1 / d2;
  } else {
    return "Cannot divide by zero";
  }
}

// Example usage:
const result = divide(10, 2);
console.log(result); // Output: 5
```

Practice Set Test : 1

1. **Declare two variables having number data and add them in console.**

```
// Declare two variables with number
data
var num1 = 5;
var num2 = 10;

// Add the two variables
var sum = num1 + num2;

// Output the result to the console
console.log("The sum is: " + sum);
```

2. **Declare two variables having string and number data type and concatenate them in console.**

```
let strVariable = "Hello";
let numVariable = 42;

console.log(strVariable + " " +
numVariable);
```

3. **Declare a Const Variable and re-assign its value is it Possible to do it.**

```
const myConstVariable = 10;
myConstVariable = 20; // This will
cause an error
```

4. **Write a Program to differ Var and Let Keyword .**

》Var Keyword《

```
function exampleVar() {
    if (true) {
        var localVar = "I am a var
variable";
        console.log(localVar); //
Output: I am a var variable
    }

    console.log(localVar); // Output:
I am a var variable
}

exampleVar();
```

》Let Keyword《

```
function exampleLet() {
    if (true) {
        let localLet = "I am a let
variable";
        console.log(localLet); //
Output: I am a let variable
    }

    // Uncommenting the line below
would result in an error:
    // console.log(localLet); //
Error: localLet is not defined
}

exampleLet();
```

5. **Write a Program to make a list of 5 Opposite words.**

```
const oppositeWords = {
    "up": "down",
    "happy": "sad",
    "hot": "cold",
    "fast": "slow",
    "bright": "dim"
};

console.log("Opposite Words:");
for (const word in oppositeWords) {
    console.log(`${word} - $
{oppositeWords[word]}`);
}
```

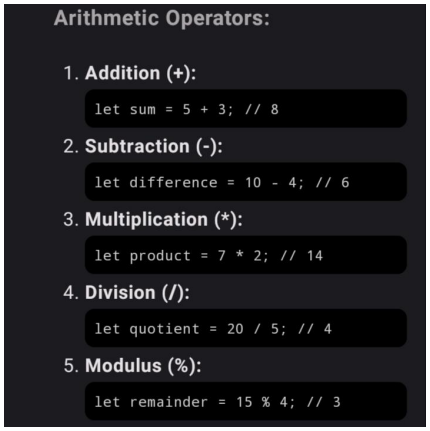
Operators and Expressions in JavaScript

Operators in JavaScript are symbols or keywords that perform operations on one or more operands. They are the fundamental building blocks for constructing expressions and manipulating values. JavaScript supports various types of operators, including:

Expressions are combinations of values, variables, and operators that, when evaluated, result in a single value. They are integral to JavaScript code and can include:

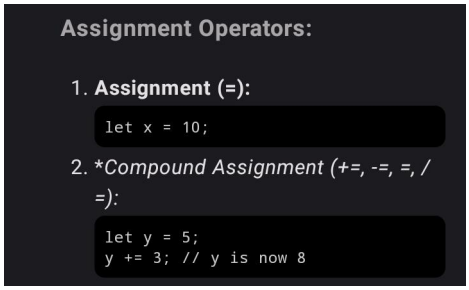
1. Arithmetic Operators:

- Perform mathematical operations such as addition, subtraction, multiplication, division, and modulus.



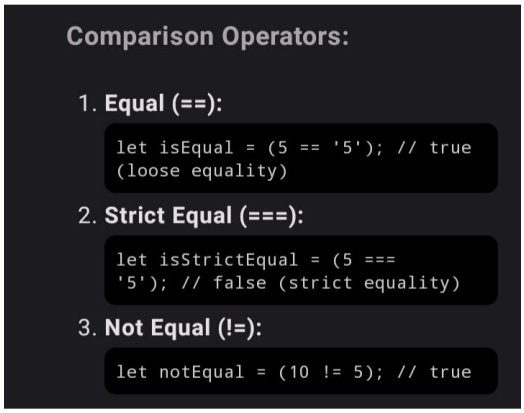
2. Assignment Operators:

- Assign values to variables and perform compound assignments.



3. Comparison Operators:

- Compare two values and return a boolean result, indicating equality, inequality, or the relationship between values.



4. Logical Operators:

- Combine or manipulate boolean values, allowing for logical operations such as AND, OR, and NOT.

Logical Operators:

1. **Logical AND (&&):**

```
let andResult = (true &&
false); // false
```

2. **Logical OR (||):**

```
let orResult = (true || false); //
true
```

3. **Logical NOT (!):**

```
let notResult = !true; // false
```

5. Conditional (Ternary) Operator:

- Provides a concise way to write an if-else statement in a single line.

Conditional (Ternary) Operator:

1. **Ternary Operator (condition ? expr1 : expr2):**

```
let age = 20;
let isAdult = (age >= 18) ?
'Yes' : 'No';
```

Practice Set Test : 2

1. **Write a JS Program to check whether the person's age lies between 18 and 24.**

```
function checkAge(age) {
    if (age >= 18 && age <= 24) {
        return "The person's age is
between 18 and 24.";
    } else {
        return "The person's age is
not between 18 and 24.";
    }
}

// Example usage:
const personAge = 20;
console.log(checkAge(personAge));
```

2. **Write a JS Program to check if the given number is even Or Odd .**

```
function checkEvenOrOdd(number) {
    if (number % 2 === 0) {
        return "The given number is
even.";
    } else {
        return "The given number is
odd.";
    }
}

// Example usage:
const givenNumber = 7;
console.log(checkEvenOrOdd(givenNumbe
r));
```

3. **Write a JS Program to check the given number is divisible by 2 and 3.**

```
function
checkDivisibilityBy2And3(number) {
    if (number % 2 === 0 && number %
3 === 0) {
        return "The given number is
divisible by both 2 and 3.";
    } else {
        return "The given number is
not divisible by both 2 and 3.";
    }
}

// Example usage:
const givenNumber = 12;
console.log(checkDivisibilityBy2And3(
givenNumber));
```

4. **Write a JS Program to check the given number is either divisible by 2 or 3.**

```
function
checkDivisibilityBy2Or3(number) {
    if (number % 2 === 0 || number %
3 === 0) {
        return "The given number is
divisible by either 2 or 3.";
    } else {
        return "The given number is
not divisible by either 2 or 3.";
    }
}

// Example usage:
const givenNumber = 9;
console.log(checkDivisibilityBy2Or3(g
ivenNumber));
```

5. **Write a JS Program to Print " You can drive " or "You can't Drive " based on age using Ternary Operator.**

```
function checkDrivingAge(age) {
    const result = age >= 18 ? "You
can drive" : "You can't drive";
    console.log(result);
}

// Example usage:
const personAge = 20;
checkDrivingAge(personAge);
```

For Loop in JavaScript

Definition:

- The `for` loop is used to repeatedly execute a block of code a specific number of times.

Syntax:

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

Example:

```
// Print numbers 1 to 5 using a for
loop
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

While Loop in JavaScript

Definition:

- The `while` loop continues to execute a block of code while a specified condition is true.

Syntax:

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

Example:

```
// Print numbers 1 to 5 using a while
loop
let counter = 1;
while (counter <= 5) {
  console.log(counter);
  counter++;
}
```

Notes:

For Loop:

- Initialization: Executed before the loop starts.

- Condition: Evaluated before each iteration. If false, the loop stops.

- Increment/Decrement: Executed after each iteration.

While Loop:

- Condition: Evaluated before each iteration. If false, the loop stops.

- Both loops are used for repetitive tasks, and the choice between them depends on the specific use case.

- Ensure the loop conditions are appropriately set to prevent infinite loops.

- Example code provided demonstrates looping through numbers; however, loops can be used for various tasks.

These loop structures are essential for controlling the flow of your JavaScript programs and performing repetitive tasks efficiently. Experiment with different loop conditions and actions to solidify your understanding.

Functions in JavaScript

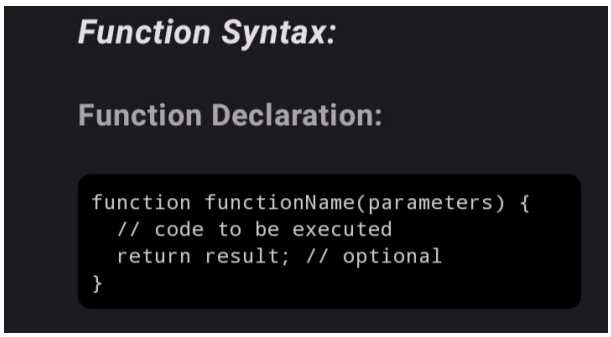
Definition:

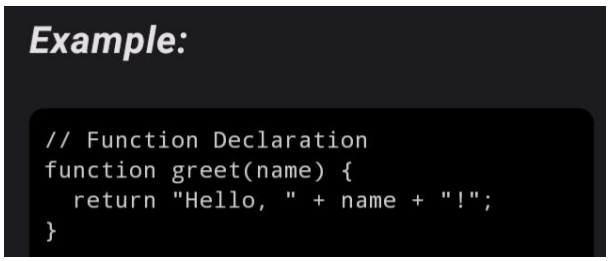
- A function is a block of reusable code that performs a specific task. Functions help in organizing and modularizing code.

Notes:

Function Declaration:

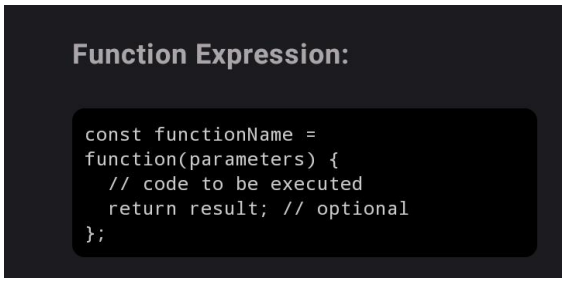
- Defined using the `function` keyword.
- Hoisted, can be called before the declaration.





Function Expression:

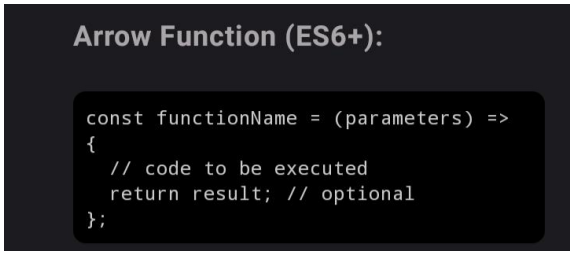
- Created by assigning an anonymous function to a variable.
- Not hoisted, must be defined before use.



```
// Function Expression
const addNumbers = function(a, b) {
  return a + b;
};
```

Arrow Function (ES6+):

- A concise syntax for function expressions.
- Lexical `this` binding.



Parameters:

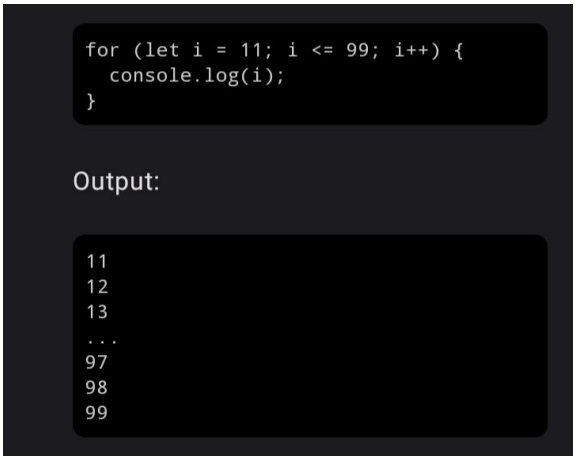
- Input values passed to a function.

Return Statement:

- Specifies the value the function returns.
- If omitted, the function returns `undefined`.
- Functions provide code reusability and encapsulation, improving maintainability.
- ES6 introduced arrow functions, offering a more compact syntax.
- Functions are first-class citizens in JavaScript, allowing them to be assigned to variables, passed as arguments, and returned from other functions.

Practice Set Test : 3

1. Write a JavaScript program that uses for loop to print numbers from 11 to 99 with output.

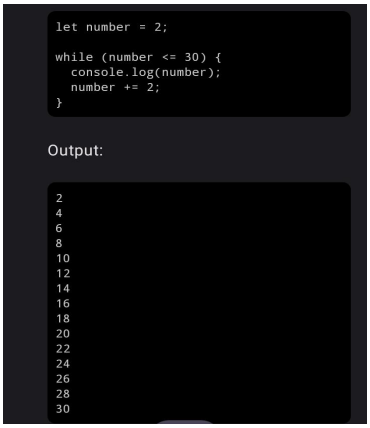


```
for (let i = 11; i <= 99; i++) {
  console.log(i);
}
```

Output:

```
11
12
13
...
97
98
99
```

2. Write a Javascript Program that uses While loop to print Even numbers from 2 to 30 with output.



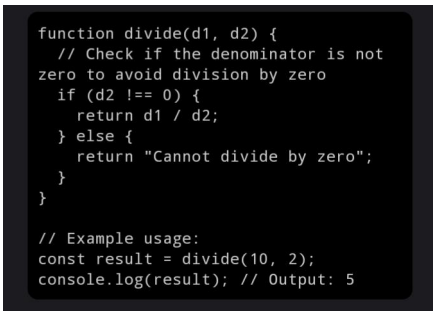
```
let number = 2;

while (number <= 30) {
  console.log(number);
  number += 2;
}
```

Output:

```
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
```

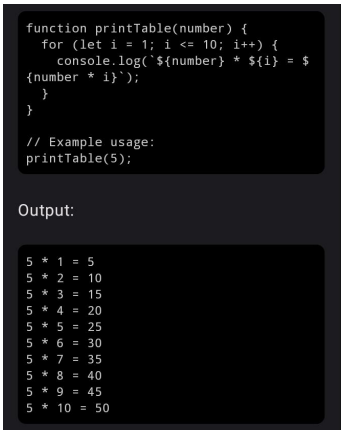
3. Write a function, divide that text two parameters, d1 and d2, and return their quotient with output.



```
function divide(d1, d2) {
  // Check if the denominator is not
zero to avoid division by zero
  if (d2 !== 0) {
    return d1 / d2;
  } else {
    return "Cannot divide by zero";
  }
}

// Example usage:
const result = divide(10, 2);
console.log(result); // Output: 5
```

4. Write a function printTable that takes number parameter and uses a for loop to print table of 5 with output.



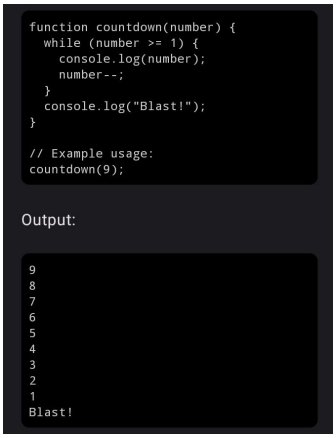
```
function printTable(number) {
  for (let i = 1; i <= 10; i++) {
    console.log(`${number} * ${i} = ${number * i}`);
  }
}

// Example usage:
printTable(5);
```

Output:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

5. Write a function countdown that takes a number parameter and uses a while loop to count down from 9 to 1 and print blast at the end with output.



```
function countdown(number) {
  while (number >= 1) {
    console.log(number);
    number--;
  }
  console.log("Blast!");
}

// Example usage:
countdown(9);
```

Output:

```
9
8
7
6
5
4
3
2
1
Blast!
```

Strings and String Methods in JavaScript

Introduction to Strings

Strings are a fundamental data type in JavaScript, used to represent sequences of characters. They are enclosed in single or double quotes, and can contain any combination of letters, numbers, symbols, and whitespace.

Common String Operations

JavaScript provides various built-in methods to manipulate and work with strings. Here are some of the most common string operations:

1. **Concatenation:** Joining two or more strings together. Example: (let message = "Hello" + " " + "World!";)
2. **Length:** Determining the number of characters in a string. Example: (let strLength = "JavaScript".length;)
3. **Accessing Characters:** Retrieving individual characters at specific positions. Example:(let firstChar = "Coding".charAt(0);)
4. **Indexing:** Accessing characters using bracket notation. Example: (let secondChar = "Coding"[1];)
5. **Searching:** Finding occurrences of a substring within a string. Example: (let index = "Welcome".indexOf("come");)
6. **Extracting Substrings:** Obtaining portions of a string. Example:(let subString = "Programming".slice(3, 7);)
7. **Case Conversion:** Changing the case of a string (uppercase, lowercase). Example: (let upperCase = "javascript".toUpperCase();)
8. **Trimming:** Removing whitespace from the beginning or end of a string. Example: (let trimmedString = " JavaScript ".trim();)

String Methods

JavaScript provides a rich set of built-in methods for manipulating strings. Here are some of the most useful methods:

1. **charAt():** Returns the character at a specified index.

```
let word = "JavaScript";

// Using charAt() to get characters
at specific indices
let firstChar = word.charAt(0); //
Gets the first character
let thirdChar = word.charAt(2); //
Gets the third character
let lastChar =
word.charAt(word.length - 1); // Gets
the last character

console.log(firstChar); // Outputs:
"J"
console.log(thirdChar); // Outputs:
"v"
console.log(lastChar); // Outputs:
"t"
```

2. **charCodeAt():** Returns the Unicode code point of the character at a specified index.

```
let text = "Hello, JavaScript!";

// Using charCodeAt() to get Unicode
of characters at specific indices
let unicodeAtIndex0 =
text.charCodeAt(0); // Gets the
Unicode at index 0
let unicodeAtIndex7 =
text.charCodeAt(7); // Gets the
Unicode at index 7
let unicodeAtIndex12 =
text.charCodeAt(12); // Gets the
Unicode at index 12

console.log(unicodeAtIndex0); //
Outputs: 72
console.log(unicodeAtIndex7); //
Outputs: 44
console.log(unicodeAtIndex12); //
Outputs: 112
```

3. **concat():** Joins two or more strings.

```
let firstName = "John";
let lastName = "Doe";

// Using concat() to concatenate
strings
let fullName = firstName.concat(" ",
lastName);

console.log(fullName); // Outputs:
"John Doe"
```

4. **indexOf():** Returns the first index of a substring within a string.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using indexOf() to find the index
of the first occurrence
let indexJavaScript =
sentence.indexOf("JavaScript");

console.log(indexJavaScript); //
Outputs: 0
```

5. **lastIndexOf():** Returns the last index of a substring within a string.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using lastIndexOf() to find the
index of the last occurrence
let lastIndexJavaScript =
sentence.lastIndexOf("JavaScript");

console.log(lastIndexJavaScript); //
Outputs: 21
```

6. **includes():** Checks if a substring exists within a string.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using includes() to check if a
substring is present
let includesJavaScript =
sentence.includes("JavaScript");

console.log(includesJavaScript); //
Outputs: true
```

7. **match():** Searches for a pattern or regular expression within a string.

```
let text = "The rain in Spain falls
mainly on the plain.";

// Using match() to find a pattern in
the string
let matchResult = text.match(/ain/g);

console.log(matchResult);
// Outputs: [ 'ain', 'ain', 'ain' ]
```

8. **replace():** Replaces a substring with a new substring.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using replace() to replace a
substring
let replacedSentence =
sentence.replace("JavaScript",
"Python");

console.log(replacedSentence);
// Outputs: "Python is fun. I love
JavaScript!"
```

```
let replacedAll = sentence.replace(/
JavaScript/g, "Python");

console.log(replacedAll);
// Outputs: "Python is fun. I love
Python!"
```

9. **search():** Searches for a pattern or regular expression within a string and returns the index of the first match.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using search() to find the index
of a pattern
let searchResult = sentence.search(/
JavaScript/);

console.log(searchResult); //
Outputs: 0
```

10. **slice():** Extracts a portion of a string.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using slice() to extract a portion
of the string
let slicedPortion = sentence.slice(4,
15);

console.log(slicedPortion); //
Outputs: "Script is f"
```

11. **split():** Splits a string into an array of substrings based on a delimiter character or regular expression.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using split() to split the string
into an array
let wordsArray = sentence.split(" ");

console.log(wordsArray);
// Outputs: [ 'JavaScript', 'is',
'fun.', 'I', 'love', 'JavaScript!' ]
```

12. **startsWith():** Checks if a string begins with a specified substring.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using startsWith() to check if a
string starts with a prefix
let startsWithJavaScript =
sentence.startsWith("JavaScript");

console.log(startsWithJavaScript); //
Outputs: true
```

13. **endsWith():** Checks if a string ends with a specified substring.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using endsWith() to check if a
string ends with a suffix
let endsWithJavaScript =
sentence.endsWith("JavaScript!");

console.log(endsWithJavaScript); //
Outputs: true
```

14. **toLowerCase():** Converts a string to lowercase.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using toLowerCase() to convert the
string to lowercase
let lowerCaseSentence =
sentence.toLowerCase();

console.log(lowerCaseSentence);
// Outputs: "javascript is fun. i
love javascript!"
```

15. **toUpperCase():** Converts a string to uppercase.

```
let sentence = "JavaScript is fun. I
love JavaScript!";

// Using toUpperCase() to convert the
string to uppercase
let upperCaseSentence =
sentence.toUpperCase();

console.log(upperCaseSentence);
// Outputs: "JAVASCRIPT IS FUN. I
LOVE JAVASCRIPT!"
```

16. **trim():** Removes whitespace from the beginning and end of a string.

```
let stringWithSpaces = "   Hello,
JavaScript!   ";

// Using trim() to remove leading and
trailing spaces
let trimmedString =
stringWithSpaces.trim();

console.log(trimmedString);
// Outputs: "Hello, JavaScript!"
```

Practice Set Test : 4

1. **Write a program having two variables and concatenate them using template literals in JavaScript.**

```
// Two variables
let firstName = "John";
let lastName = "Doe";

// Concatenating variables using
template literals
let fullName = `${firstName} $
{lastName}`;

// Output the result
console.log(fullName);
// Outputs: "John Doe"
```

2. **Create a Function having (InputText) as a Parameter and calculate the length of the strings.**

```
// Function to calculate the length
of a string
function
calculateStringLength(inputText) {
  return inputText.length;
}

// Example usage
let text = "Hello, JavaScript!";
let lengthOfText =
calculateStringLength(text);

// Output the result
console.log(`The length of the text
is: ${lengthOfText}`);
// Outputs: "The length of the text
is: 18"
```

3. **Write a Program that convert a string into Upper Case and lower Case.**

```
// Example usage
let originalString = "Hello,
JavaScript!";

// Convert to uppercase
let upperCaseString =
originalString.toUpperCase();
console.log(`Uppercase: $
{upperCaseString}`);

// Convert to lowercase
let lowerCaseString =
originalString.toLowerCase();
console.log(`Lowercase: $
{lowerCaseString}`);
```

4. **Write a Program that removes Extra spaces beside of a strings.**

```
// Example usage of trim()
let stringWithSpaces = "   Hello,
JavaScript!   ";

// Using trim() to remove leading and
trailing spaces
let trimmedString =
stringWithSpaces.trim();

// Output the result
console.log(`Original String: "$
{stringWithSpaces}"`);
console.log(`Trimmed String: "$
{trimmedString}"`);
```

5. **Write a Program to compare two String Values and check if true.**

```
// Function to compare two strings
function areStringsEqual(string1,
string2) {
  return string1 === string2;
}

// Example usage
let firstString = "Hello,
JavaScript!";
let secondString = "Hello,
JavaScript!";

// Check if the strings are equal
let result =
areStringsEqual(firstString,
secondString);

// Output the result
console.log(`Are the strings equal? $
{result}`);
// Outputs: "Are the strings equal?
true"
```

Arrays in JavaScript

What are Arrays?

Arrays are ordered collections of items. Each item in an array is called an element. Arrays are a versatile data structure that can be used to store a variety of data types, including numbers, strings, booleans, and even other arrays.

Creating Arrays

There are two main ways to create arrays in JavaScript:

1. **Using square brackets:**

JavaScript

const fruits = ["apple", "banana", "orange"];

2. Using the `new Array()` constructor:

Accessing Array Elements

You can access individual elements in an array using their index. Arrays are zero-indexed, which means that the first element has an index of 0, the second element has an index of 1, and so on.

Array Methods

JavaScript provides a variety of methods for working with arrays. Some of the most common methods include:

1. **`push()`**: Adds an element to the end of an array.

```
// Initialize an array
let fruits = ["Apple", "Banana",
"Orange"];

// Using push() to add an element to
the end
fruits.push("Mango");
console.log("After push:",
fruits); // Outputs: ["Apple",
"Banana", "Orange", "Mango"]
```

2. **`pop()`**: Removes the last element from an array and returns it.

```
// Using pop() to remove the last
element
let removedFruit = fruits.pop();
console.log("After pop:", fruits); //
Outputs: ["Apple", "Banana",
"Orange"]
console.log("Removed Fruit:",
removedFruit); // Outputs: "Mango"
```

3. **`unshift()`**: Adds an element to the beginning of an array.

```
// Using unshift() to add an element
to the beginning
fruits.unshift("Grapes");
console.log("After unshift:",
fruits); // Outputs: ["Grapes",
"Apple", "Banana", "Orange"]
```

4. **`shift()`**: Removes the first element from an array and returns it.

```
// Using shift() to remove the first
element
let shiftedFruit = fruits.shift();
console.log("After shift:",
fruits); // Outputs: ["Apple",
"Banana", "Orange"]
console.log("Shifted Fruit:",
shiftedFruit); // Outputs: "Grapes"
```

5. **slice()**: Returns a shallow copy of a portion of an array.

```
// Initialize an array
let numbers = [1, 2, 3, 4, 5];

// Using slice() to create a new
array from a portion
let slicedNumbers = numbers.slice(1,
4);
console.log("After slice:",
slicedNumbers); // Outputs: [2, 3, 4]
```

6. **splice()**: Removes or replaces elements from an array.

```
// Using splice() to insert and/or
remove elements
numbers.splice(2, 1, 6, 7);
console.log("After splice:",
numbers); // Outputs: [1, 2, 6, 7, 4,
5]
```

7. **forEach()**: Calls a function for each element in an array.

```
// Using forEach() to iterate over
array elements
console.log("Using forEach:");
numbers.forEach(function (number) {
  console.log(number);
});
// Outputs:
// 1
// 2
// 6
// 7
// 4
// 5
```

Using Loops in Arrays

Arrays are fundamental data structures in JavaScript, used to store collections of items. They are represented as an ordered sequence of values enclosed in square brackets ([]).

Loops are essential tools for iterating through arrays and performing operations on each element. JavaScript offers three primary loop types:

1. **For Loop:** The most common loop type, executing a block of code a specified number of times.

```
// Initialize an array
let fruits = ["Apple", "Banana",
"Orange", "Mango"];

// Using a for loop to iterate
through each element
console.log("Using for loop:");
for (let i = 0; i < fruits.length; i+
+) {
  console.log(fruits[i]);
}
```

```
Using for loop:
Apple
Banana
Orange
Mango
```

CODE OUTPUT

2. **While Loop:** A loop that continues to execute as long as a specified condition remains true.

```
// Initialize an array
let numbers = [1, 2, 3, 4, 5];

// Using a while loop to iterate
through each element
console.log("Using while loop:");
let i = 0;
while (i < numbers.length) {
  console.log(numbers[i]);
  i++;
}
```

CODE

```
Using while loop:
1
2
3
4
5
```

OUTPUT

Map , Filter and Reduce Methods

. **map():** Creates a new array by transforming each element of an array.

```
// Initialize an array
let numbers = [1, 2, 3, 4, 5];

// Using map() to create a new array
by doubling each element
let doubledNumbers =
numbers.map(function (number) {
  return number * 2;
});

// Output the result
console.log("Original Array:",
numbers);
console.log("Doubled Array:",
doubledNumbers);
```

CODE

```
Original Array: [1, 2, 3, 4, 5]
Doubled Array: [2, 4, 6, 8, 10]
```

OUTPUT

. **filter():** Creates a new array by filtering out elements that don't meet a certain condition.

```
// Initialize an array
let numbers = [1, 2, 3, 4, 5, 6, 7,
8, 9, 10];

// Using filter() to create a new
array with even numbers
let evenNumbers =
numbers.filter(function (number) {
  return number % 2 === 0;
});

// Output the result
console.log("Original Array:",
numbers);
console.log("Even Numbers:",
evenNumbers);
```

```
Original Array: [1, 2, 3, 4, 5, 6, 7,
8, 9, 10]
Even Numbers: [2, 4, 6, 8, 10]
```

CODE **OUTPUT**

. **reduce():** Reduces an array to a single value.

```
// Initialize an array
let numbers = [1, 2, 3, 4, 5];

// Using reduce() to calculate the
sum of all numbers
let sum = numbers.reduce(function
(accumulator, currentNumber) {
  return accumulator + currentNumber;
}, 0);

// Output the result
console.log("Original Array:",
numbers);
console.log("Sum of Numbers:", sum);
```

```
Original Array: [1, 2, 3, 4, 5]
Sum of Numbers: 15
```

CODE **OUTPUT**

Practice Set Test : 5

1. **Create an array and use a for loop to print each element in an array.**

```
// Create an array
let fruits = ["Apple", "Banana",
"Orange", "Mango"];

// Using a for loop to print each
element
console.log("Using for loop:");
for (let i = 0; i < fruits.length; i+
+) {
  console.log(fruits[i]);
}
```

```
Using for loop:
Apple
Banana
Orange
Mango
```

CODE **OUTPUT**

2. **Use . Map () method to double the all element of an array.**

```
// Create an array
let numbers = [1, 2, 3, 4, 5];

// Using map() to double each element
let doubledNumbers =
numbers.map(function (number) {
  return number * 2;
});

// Output the result
console.log("Original Array:",
numbers);
console.log("Doubled Array:",
doubledNumbers);
```

CODE

```
Original Array: [1, 2, 3, 4, 5]
Doubled Array: [2, 4, 6, 8, 10]
```

OUTPUT

3. **Filter all the element of an array greater than 17 and less than 25.**

```
// Create an array
let numbers = [12, 18, 20, 22, 30,
15, 25];

// Using filter() to get elements
greater than 17 and less than 25
let filteredNumbers =
numbers.filter(function (number) {
  return number > 17 && number < 25;
});

// Output the result
console.log("Original Array:",
numbers);
console.log("Filtered Array:",
filteredNumbers);
```

CODE

```
Original Array: [12, 18, 20, 22, 30,
15, 25]
Filtered Array: [18, 20, 22]
```

OUTPUT

4. **Use .Concat method to combine two arrays and returns a new array.**

```
// Create two arrays
let fruits1 = ["Apple", "Banana"];
let fruits2 = ["Orange", "Mango"];

// Using concat() to combine the
arrays
let combinedFruits =
fruits1.concat(fruits2);

// Output the result
console.log("Array 1:", fruits1);
console.log("Array 2:", fruits2);
console.log("Combined Array:",
combinedFruits);
```

CODE

```
Array 1: ["Apple", "Banana"]
Array 2: ["Orange", "Mango"]
Combined Array: ["Apple", "Banana",
"Orange", "Mango"]
```

OUTPUT

5. **Create an empty array and use .push ( ) method to add numbers from 1 to 5 to the array.**

```
// Create an empty array
let numbersArray = [];

// Using push() to add numbers from 1
to 5 to the array
for (let i = 1; i <= 5; i++) {
  numbersArray.push(i);
}

// Output the result
console.log("Updated Array:",
numbersArray);
```

CODE

```
Updated Array: [1, 2, 3, 4, 5]
```

OUTPUT

Javascript in the Browser :

There are two main ways to run JavaScript code in the browser process:

1. Embedding the code in an HTML page: This is the most common way to run JavaScript code. The code is written in a **<script>** tag inside of the HTML document. The browser will execute the code when the page is loaded.

There are primarily three ways to embed JavaScript code in an HTML document:

1. **Inline:** Embedding JavaScript directly within HTML using the `<script>` tag, as shown in the previous example.

```
<script>
    // Your JavaScript code here
</script>
```

2. **Internal:** Including JavaScript code in a separate file and linking it to the HTML document using the `<script>` tag with the `src` attribute.

```
<script src="your-script.js"></
script>
```

3. **External:** Using event attributes in HTML tags to include JavaScript directly. For example, using the `onclick` attribute to attach a function to a button click.

```
<button
onclick="yourFunction()">Click
me</button>
```

Each method has its use cases, and the choice depends on factors like code organization, reusability, and separation of concerns.

2. Using the browser's developer tools: The browser's developer tools provide a console where you can enter and execute JavaScript code. This is useful for debugging code or testing new ideas.

To open the developer tools, you can use one of the following shortcuts:

- **Windows/Linux: F12**
- **Mac: Command+Option+J**

Once the developer tools are open, you can find the console in the bottom panel. To execute code, simply type it into the console and press Enter.

JavaScript

```
console.log('Hello, world!');
```

The browser will execute the code and display the output in the console.

Exercise : 1

1. **Write a javascript Program and take User's Name and Greet them with their name.** (Hint : Prompt & Alert)

```
// Prompt the user for their name
var userName = prompt("Please enter
your name:");

// Greet the user with their name
alert("Hello, " + userName + "!
Welcome back!");
```

2. **Write a Javascript Program to calculate the area of a rectangle after taking input from the user.**(Hint : Prompt & Alert)

```
// Welcome message
console.log("Hey there! Let's
calculate the area of a rectangle.");

// Get user input for length
let length = parseFloat(prompt("Enter
the length of the rectangle:"));

// Get user input for width
let width = parseFloat(prompt("Enter
the width of the rectangle:"));

// Calculate the area
let area = length * width;

// Display the result
console.log(`The area of the
rectangle is: ${area}`);
```