JavaScript - Basics

Unit - 2 : Content

- **JavaScript Introduction**
    - Overview of JavaScript
    - Variables
    - Data Types
    - Functions
    - Arrays
    - **Objects in JavaScript**: Boolean, Number, String, Date, Math
    - Regular Expressions

JavaScript Introduction

- **Overview of JavaScript**
  - JavaScript is a high-level, interpreted scripting language primarily used for client-side web development.
  - It was introduced in 1995 by Netscape and has since become one of the most widely used programming languages.
  - JavaScript is known for its flexibility, as it allows developers to create dynamic and interactive web applications.
- **Importance in Web Development**
  - JavaScript plays a crucial role in modern web development, enabling the creation of dynamic user interfaces and responsive web pages.
  - It is used for tasks such as form validation, DOM manipulation, event handling, and asynchronous programming.
  - With the advent of frameworks like React, Angular, and Vue.js, JavaScript has become even more essential for building complex web applications.

Features of JavaScript

- **Dynamic Typing**
  - JavaScript is dynamically typed, meaning variable types are determined at runtime rather than compile time.
  - This allows for flexibility but can also lead to unexpected behavior if types are not properly managed.
- **Prototype-based Inheritance**
  - In JavaScript, objects inherit properties and methods directly from other objects through prototypes.
  - This differs from class-based inheritance in languages like Java or C++, providing a more flexible object model.
- **First-class Functions**
  - Functions in JavaScript are treated as first-class citizens, meaning they can be assigned to variables, passed as arguments, and returned from other functions.
  - This functional programming paradigm enables powerful techniques like higher-order functions and closures.

Features of JavaScript (cont'd)

- **Event-driven Programming**
  - JavaScript is inherently event-driven, allowing developers to respond to user interactions and system events.
  - This asynchronous programming model is essential for building interactive web applications.
- **DOM Manipulation**
  - JavaScript provides powerful APIs for manipulating the Document Object Model (DOM), which represents the structure of a web page.
  - Developers can dynamically update the content, style, and structure of web pages in response to user actions or other events.
- **Asynchronous Programming**
  - JavaScript supports asynchronous programming through mechanisms like callbacks, promises, and async/await.
  - This allows non-blocking execution of code, making it possible to handle tasks such as fetching data from servers or performing animations without freezing the user interface.

Include JS in HTML: Inline JavaScript

JavaScript code is directly embedded within the HTML document using the <script> tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Inline JavaScript Example</title>
</head>
<body>

<h1>Inline JavaScript Example</h1>

<script>
    // Inline JavaScript code
    document.write("Hello, World!");
</script>

</body>
</html>
```

Include JS in HTML: Internal JavaScript

JavaScript code is placed within the <script> tag inside the HTML document's <head> or <body> section. tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Internal JavaScript Example</title>
    <script>        // Internal JavaScript code
        function greet() {
            alert("Hello, World!");
        }
    </script>
</head>
<body>
<h1>Internal JavaScript Example</h1>
<button onclick="greet()">Click me</button>
</body>
</html>
```

Include JS in HTML: External JavaScript

JavaScript code is stored in an external .js file and linked to the HTML document using the `<script>` tag's src attribute.

```
<!-- index.html -->
<html lang="en">
<head>
    <title>External JavaScript Example</title>
    <script src="script.js"></script>
</head>
<body>
<h1>External JavaScript Example</h1>
<p id="demo"></p>
<button onclick="displayDate()">Click me</button>
</body>
</html>
```

```
// script.js
function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
```

JavaScript Variables I

- **Declaration and Initialization**
  - JavaScript variables are declared using the `var`, `let`, or `const` keywords.
  - The `var` keyword was traditionally used for variable declaration, but `let` and `const` introduced in ES6 provide block-scoping and immutability, respectively.
  - Example:
    - `var x = 5;`
    - `let y = 'Hello';`
    - `const PI = 3.14;`
- **Data Types**
  - JavaScript variables can hold various data types, including numbers, strings, booleans, objects, arrays, functions, and symbols.
  - Unlike many other programming languages, JavaScript is dynamically typed, meaning the type of a variable is inferred at runtime.

JavaScript Variables II

- **Variable Scope**
  - Variables declared with `var` are function-scoped, whereas variables declared with `let` and `const` are block-scoped.
  - Block-scoped variables are only accessible within the block they are defined in, providing better control over variable scope and reducing the risk of unintended side effects.
- **Hoisting**
  - In JavaScript, variable declarations are hoisted to the top of their containing scope during compilation.
  - However, only the declaration is hoisted, not the initialization. This means that variables declared with `var` are initialized with `undefined` until their value is assigned.

Rules for JavaScript Variable Names

- Variable names in JavaScript must begin with a letter (a-z, A-Z), underscore (_), or dollar sign ($). They cannot begin with a number.
- After the first character, variable names can include letters, numbers, underscores, or dollar signs.
- JavaScript variable names are case-sensitive, meaning `myVar` and `myvar` are treated as different variables.
- Certain words are reserved in JavaScript and cannot be used as variable names (e.g., `class`, `function`, `return`, etc.).
- Variable names should be meaningful and descriptive to improve code readability.

Data Types

- JavaScript variables can hold various data types, including numbers, strings, booleans, objects, arrays, functions, and symbols.
- Unlike many other programming languages, JavaScript is dynamically typed, meaning the type of a variable is inferred at runtime.

`We can know the type of a variable using typeof Operator`

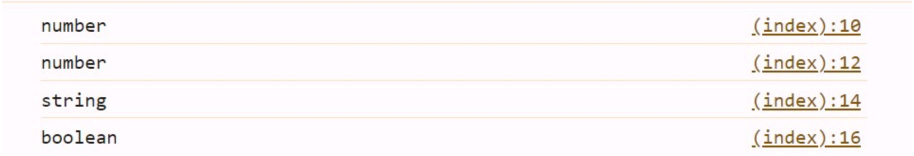
Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello, World!</title>
   </head>
  <body>
      <h1 id="vtype"></h1>
      <script type="text/javascript">
      var a=4;
      console.log(typeof(a));//number
      var b=5.6;
```

Data Types

```
      console.log(typeof(b));//number
      var c='SNIST';
      console.log(typeof(c));//string
      var d=true;
      console.log(typeof(d));//number
      </script>
  </body>
</html>
```

Output:



```
number                                              (index):10
number                                              (index):12
string                                              (index):14
boolean                                             (index):16
```

Objects in JavaScript: Functions

- In JavaScript, functions are first-class objects, which means they can be treated like any other object.
- The Function object is a global object and can be constructed using the `Function()` constructor or by simply declaring a function using the function declaration or function expression syntax.
- Functions in JavaScript are used to define reusable blocks of code that can be invoked (called) with different arguments.
- They are often used to encapsulate functionality, organize code, and promote reusability.
- Functions in JavaScript can be anonymous or named, and they can accept parameters and return values.
- The Function object in JavaScript provides several properties and methods that can be used to manipulate functions programmatically.

Function Declaration vs. Function Expression

In JavaScript, there are two common ways to define functions:

1. **Function Declaration**:
   - Declared using the `function` keyword followed by the function name and function body.
   - Can be invoked before they are declared due to hoisting.
   - Example:

```
function greet(name) {
    return `Hello, ${name}!`;
}
```

2. **Function Expression**:
   - Defined using a variable assignment where the value is a function.
   - Cannot be invoked before they are defined.
   - Example:

```
const greet = function(name) {
    return `Hello, ${name}!`;
};
```

Properties of the Function Object

The Function object in JavaScript has several properties, including:

- `length`: Specifies the number of arguments expected by the function.
- `name`: Returns the name of the function.
- `prototype`: Allows the addition of properties and methods to the function's prototype object.

These properties can be accessed using dot notation or bracket notation.

Methods of the Function Object

The Function object in JavaScript also provides several methods, including:

- `apply()`: Calls a function with a given this value and arguments provided as an array.
- `bind()`: Creates a new function that, when called, has its this keyword set to a specified value.
- `call()`: Calls a function with a given this value and arguments provided individually.
- `toString()`: Returns a string representing the source code of the function.

These methods can be used to manipulate functions dynamically at runtime.

Example: Using Function Object Methods

```
let car={
 name:"TATA"
}
function displaycar(color,price){
    console.log("car name is "+this.name+" ,Color is "+
 color+" and Price is "+price)
}
displaycar.call(car,"Red",1200000);
displaycar.apply(car,["Green",2400000]);
var newcar=displaycar.bind(car,"White",1000000);
newcar();
console.log(displaycar.toString());
```

This example showcases the use of `call()` ,`apply()` and `bind()` methods to manipulate the function `displaycar()`.

Objects in JavaScript: Array

- **Overview**
  - The Array object represents ordered collections of values in JavaScript.
  - It is used for storing and manipulating lists of data.

Method	**Description**
`concat()`	Joins two or more arrays and returns a new array.
`copyWithin()`	Copies array elements within the array, to and from specified positions.
`entries()`	Returns a new Array Iterator object that contains the key/value pairs for each index in the array.
`every()`	Checks if every element in an array pass a test.
`fill()`	Fills all the elements of an array from a start index to an end index with a static value.
`filter()`	Creates a new array with all elements that pass the test implemented by the provided function.
`find()`	Returns the value of the first element in an array that satisfies the provided testing function.

Objects in JavaScript: Array

Example:

```
  <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
   initial-scale=1.0">
    <title>Arrays</title>
</head>
<body>
    <script type="text/javascript">
    let fruits = ['apple', 'banana', 'cherry','apple'];
    console.log("Array Length : "+fruits.length);
    let iterator = fruits.entries();
    for (let entry of fruits) {
        console.log(entry);
    }

let startswithA=fruits.every(function(fruit){
```

Objects in JavaScript: Array

Example: (Continued...)

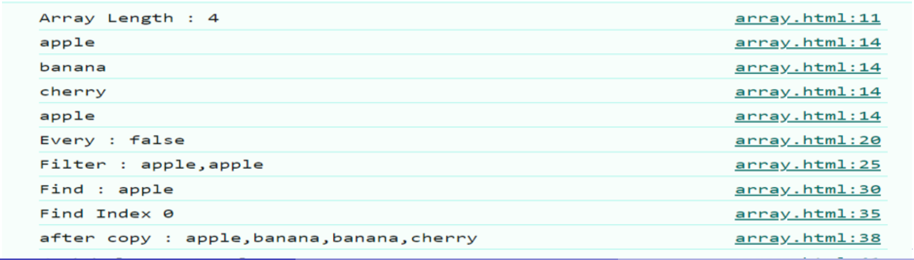
```
        return fruit[0]=="a";
    });
    console.log("Every : "+startswithA);

    var x=fruits.filter(function(fruit){
        return fruit[0]=="a";
    });
    console.log("Filter : "+x);

    var x1=fruits.find(function(fruit){
        return fruit[0]=="a";
    });
    console.log("Find : "+x1);

    var x1=fruits.findIndex(function(fruit){
        return fruit[0]=="a";
    });
    console.log("Find Index "+x1);
```

Objects in JavaScript: Array

Example: (Continued...)

```
    fruits.copyWithin(2,1,3);
    console.log("after copy : "+fruits);

    fruits.fill(2);
    console.log(fruits);
    </script>
</body>
</html>
```

Output:



```
Array Length : 4                                   array.html:11
apple                                              array.html:14
banana                                             array.html:14
cherry                                             array.html:14
apple                                              array.html:14
Every : false                                      array.html:20
Filter : apple,apple                               array.html:25
Find : apple                                       array.html:30
Find Index 0                                       array.html:35
after copy : apple,banana,banana,cherry            array.html:38
```

Objects in JavaScript: Array (continued)

- **Other Array methods:** `findIndex()`, `flat()`, `flatMap()`, `forEach()`, `includes()`, `indexOf()`, `join()`, `keys()`, `lastIndexOf()`, `map()`, `pop()`, `push()`, `reduce()`, `reduceRight()`, `reverse()`, `shift()`, `slice()`, `some()`, `sort()`, `splice()`, `toLocaleString()`, `toString()`, `unshift()`, `values()`

- **Example**

```
let fruits = ["Apple", "Banana", "Orange"];
fruits.push("Mango");
console.log(fruits); // Output: ["Apple", "Banana",
    "Orange", "Mango"]
console.log(fruits.join(", ")); // Output: "Apple,
    Banana, Orange, Mango"
```

Objects in JavaScript

- **Introduction**
  - Objects are one of the most fundamental concepts in JavaScript.
  - They are used to store collections of data and methods that operate on that data.
  - Objects in JavaScript are dynamic, meaning they can be modified at runtime by adding or removing properties and methods.
- **Prototypes and Prototypal Inheritance**
  - In JavaScript, objects can inherit properties and methods from other objects through prototypes.
  - Every JavaScript object has a prototype chain, which allows it to inherit properties and methods from its prototype object.
  - Prototypal inheritance is a key feature of JavaScript that allows for code reusability and polymorphism.

Objects in JavaScript: Creating Objects

- Objects in JavaScript can be created using object literals, constructor functions, or the `Object.create()` method.
- Example:
  - Object Literal: `let person = { name: 'Vijay', age: 38 };`
  - Constructor Function:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}
let person = new Person('Vijay', 38);
```

  - `Object.create()`:

```
let person = Object.create(null);
person.name = 'Vijay';
person.age = 38;
```

Objects in JavaScript: Properties and Methods

- Objects in JavaScript consist of key-value pairs, where keys are called properties and values can be of any data type.
- Methods are functions stored as object properties.
- Properties and methods can be accessed using dot notation or bracket notation.
- Example:

```
let person = {
    name: 'Vijay',
    age: 38,
    sayHello: function() {
    console.log('Hello, my name is ' + this.name);
    }
};
console.log(person.name); // Output: Vijay
person.sayHello(); // Output: Hello, my name is Vijay
```

Objects in JavaScript: Boolean

- **Overview**
  - The Boolean object represents one of two values: `true` or `false`.
  - It is used for logical operations and conditional expressions.
  - The Boolean value of 0, -0, ""(empty string), undefined, null, false, NaN is **false**.

Method	Description
`valueOf()`	Returns the primitive value of the Boolean object.
`toString()`	Returns a string representation of the Boolean object.

- **Example**

```
let boolObj = new Boolean(true);
console.log(boolObj.valueOf()); // Output: true
console.log(boolObj.toString()); // Output: "true"
```

Objects in JavaScript: Number

- **Overview**
  - The Number object represents numerical values in JavaScript.
  - It can represent integers, floating-point numbers, and special numeric values like `NaN` and `Infinity`.

Method	Description
`valueOf()`	Returns the primitive value of the Number object.
`toString()`	Returns a string representation of the Number object.
`toFixed()`	Returns a string representing the Number object using fixed-point notation.

- **Example**

```
let numObj = new Number(3.14159);
console.log(numObj.toFixed(2)); // Output: "3.14"
```

Objects in JavaScript: String

- **Overview**
  - The String object represents textual data in JavaScript.
  - It is used for manipulating and processing strings of characters.

Method	**Description**
`charAt()`	Returns the character at the specified index.
`charCodeAt()`	Returns the Unicode value of the character at the specified index.
`concat()`	Joins two or more strings and returns a new string.
`indexOf()`	Returns the index within the calling String object of the first occurrence of the specified value.
`lastIndexOf()`	Returns the index within the calling String object of the last occurrence of the specified value.
`replace()`	Searches a string for a specified value or regular expression and returns a new string where the specified values are replaced.

Objects in JavaScript: String (continued)

Example:(Continued...)

```
    <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=<device-width>,
   initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script type="text/javascript">
        let str="Welcome to WT Technologies Training.";
        for(var i=0;i<str.length;i++){
            document.write(" "+str[i]);
        }
        document.write('<br/>');
        document.write("Character at 5 : "+str.charAt(5));

        document.write('<br/>');
```

Objects in JavaScript: String (continued)

Example:(Continued...)

```
        document.write("Character Code at 4 : "+str.
    charCodeAt(4));

        str[0]="v";
        document.write('<br/>');
        document.write("1st index not modified  : "+str);//
    string is immutable

        var newstr=str.concat("SNIST welcomes you.");
        document.write('<br/>');
        document.write("concatenated String is :"+newstr);

        var index=str.indexOf("W");
        document.write('<br/>');
        document.write("Index of W character :"+index);

        var index=str.lastIndexOf("W");
        document.write('<br/>');
```

Objects in JavaScript: String (continued)

Example:(Continued...)

```
        document.write("Last Index of W character :"+index);

        var newstr=str.replace("W","V");
        document.write('<br/>');
        document.write("Replaced One In The String :"+newstr
    );

        var newstr=str.replaceAll("W","V");
        document.write('<br/>');
        document.write("Replaced All In The String :"+newstr
    );

        var newstr=str.slice(1,6);
        document.write('<br/>');
        document.write("Sliced String :"+newstr);

        var splitstr=str.split(" ");
        document.write('<br/>');
        document.write("Splitted String length:"+splitstr
```

Objects in JavaScript: String (continued)

Example:(Continued...)

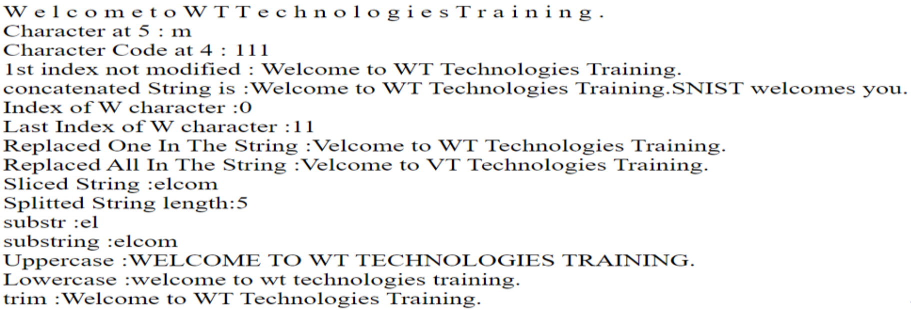
```
      document.write("Splitted String length:"+splitstr.
  length);

      var newstr=str.substr(1,2);
      document.write('<br/>');
      document.write("substr :"+newstr);

      var newstr=str.substring(1,6);
      document.write('<br/>');
      document.write("substring :"+newstr);

      var newstr=str.toUpperCase();
      document.write('<br/>');
      document.write("Uppercase :"+newstr);

      var newstr=str.toLowerCase();
      document.write('<br/>');
      document.write("Lowercase :"+newstr);
```

Objects in JavaScript: String (continued)

Example:(Continued...)

```
        var newstr=str.trim();
        document.write('<br/>');
        document.write("trim :"+newstr);
    </script>
</body>
</html>
```

Output:



```
W e l c o m e t o W T T e c h n o l o g i e s T r a i n i n g .
Character at 5 : m
Character Code at 4 : 111
1st index not modified : Welcome to WT Technologies Training.
concatenated String is :Welcome to WT Technologies Training.SNIST welcomes you.
Index of W character :0
Last Index of W character :11
Replaced One In The String :Velcome to WT Technologies Training.
Replaced All In The String :Velcome to VT Technologies Training.
Sliced String :elcom
Splitted String length:5
substr :el
substring :elcom
Uppercase :WELCOME TO WT TECHNOLOGIES TRAINING.
Lowercase :welcome to wt technologies training.
trim :Welcome to WT Technologies Training.
```

Objects in JavaScript: Date

- **Overview**
  - The Date object represents dates and times in JavaScript.
  - It is used for working with dates, times, and time intervals.

Method	**Description**
`getDate()`	Returns the day of the month (1-31) for the specified date.
`getDay()`	Returns the day of the week (0-6) for the specified date.
`getFullYear()`	Returns the year (4 digits for dates between 1000 and 9999) for the specified date.
`getHours()`	Returns the hour (0-23) in the specified date according to local time.
`getMilliseconds()`	Returns the milliseconds (0-999) in the specified date.
`getMinutes()`	Returns the minutes (0-59) in the specified date.
`getMonth()`	Returns the month (0-11) in the specified date.
`getSeconds()`	Returns the seconds (0-59) in the specified date.

Objects in JavaScript: Date

Example:

```
    <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
   initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script type="text/javascript">
        var d=new Date();
        document.write('<br/>');
        document.write("Current Date : "+d);

        var x=d.getDate();
        document.write('<br/>');
        document.write("Current Day : "+x);
```

Objects in JavaScript: Date

Example:(Continued...)

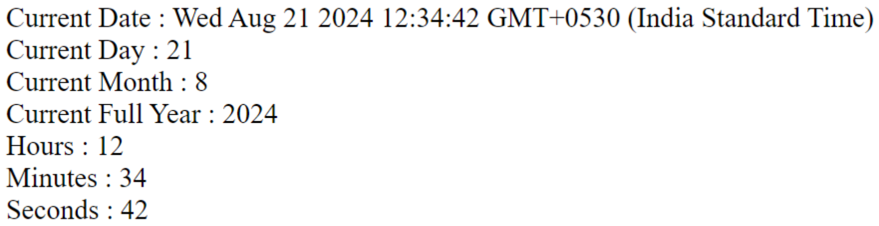
```
        var x=d.getMonth();
        document.write('<br/>');
        document.write("Current Month : "+(x+1));

        var x=d.getFullYear();
        document.write('<br/>');
        document.write("Current Full Year : "+x);

        var x=d.getHours();
        document.write('<br/>');
        document.write("Hours : "+x);


        var x=d.getMinutes();
        document.write('<br/>');
        document.write("Minutes : "+x);
```

Objects in JavaScript: Date

Example:(Continued...)

```
            var x=d.getSeconds();
            document.write('<br/>');
            document.write("Seconds : "+x);

      </script>
</body>
</html>
```

Output:



```
Current Date : Wed Aug 21 2024 12:34:42 GMT+0530 (India Standard Time)
Current Day : 21
Current Month : 8
Current Full Year : 2024
Hours : 12
Minutes : 34
Seconds : 42
```

Objects in JavaScript: Date (Continued)

- **Other date methods”** getTime(), getTimezoneOffset(), getUTCDate(), getUTCDay(), getUTCFullYear(), getUTCHours(), getUTCMilliseconds(), getUTCMinutes(), getUTCMonth(), getUTCSeconds(), setDate(), setFullYear(), setHours(), setMilliseconds(), setMinutes(), setMonth(), setSeconds(), setTime(), setUTCDate(), setUTCFullYear(), setUTCHours(), setUTCMilliseconds(), setUTCMinutes(), setUTCMonth(), setUTCSeconds()
- **Example**

```
    var x=d.getUTCDate();
    document.write('<br/>');
    document.write("Current UTC Date : "+x); // 21

    var x=d.getUTCHours();
    document.write('<br/>');
    document.write("Hours : "+x);// 7
```

Objects in JavaScript: Math

- **Overview**
  - The Math object provides mathematical constants and functions in JavaScript.
  - It is used for performing mathematical operations such as trigonometry, logarithms, and exponentiation.

Method	**Description**
`abs()`	Returns the absolute value of a number.
`acos()`	Returns the arccosine of a number.
`acosh()`	Returns the hyperbolic arccosine of a number.
`asin()`	Returns the arcsine of a number.
`asinh()`	Returns the hyperbolic arcsine of a number.
`atan()`	Returns the arctangent of a number.
`atan2()`	Returns the arctangent of the quotient of its arguments.
`atanh()`	Returns the hyperbolic arctangent of a number.
`cbrt()`	Returns the cube root of a number.

Objects in JavaScript: Math (continued)

- **Other math methods:** `ceil()`, `clz32()`, `cos()`, `cosh()`, `exp()`, `expm1()`, `floor()`, `fround()`, `hypot()`, `imul()`, `log()`, `log10()`, `log1p()`, `log2()`, `max()`, `min()`, `pow()`, `random()`, `round()`, `sign()`, `sin()`, `sinh()`, `sqrt()`, `tan()`, `tanh()`, `trunc()`
- **Example**

```
console.log(Math.abs(-5)); // Output: 5
console.log(Math.sqrt(25)); // Output: 5
console.log(Math.sin(Math.PI / 2)); // Output: 1 (
    sine of 90 degrees)
```

Regular Expression

- **Overview**
  - Regular expressions (regex) are patterns used to match character combinations in strings.
  - They are used for searching, replacing, and validating strings based on patterns.

Method	**Description**
`exec()`	The exec() methods searches a string for a specified value and returns the first match. or else returns null
`test()`	The test() method tests for a match in a string and returns true or false.
`match()`	The match() method retrieves the matches of a pattern in a string. It returns an array of matches or null if no match is found.
`search()`	The search() method searches for a pattern in a string and returns the index of the first match. If no match is found, it returns -1.

Regular Expression

Method	Description
`replace()`	The replace() method searches for a pattern in a string and replaces it with a replacement string.
`split()`	The split() method splits a string into an array of sub-strings using the pattern as a delimiter.

Regular Expression

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
   initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script type="text/javascript">
        //1. i: case- insensitive
        var text="Welcome to my world or your World";
        var pattern=/welcome/i;
        var result=text.match(pattern);
        console.log(result);// Welcome

        // email pattern search
        var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
```

Regular Expression

Example: (Continued..)

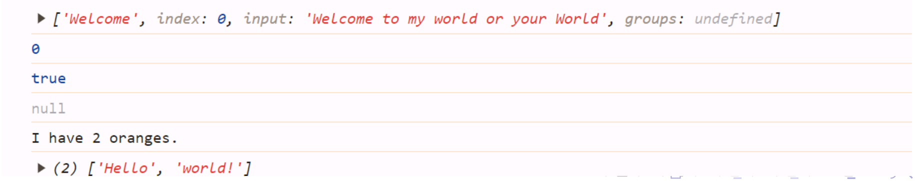
```
var email="vijaykumar.b@sreenidhi.edu.in";
var result=email.search(emailPattern);
console.log(result);

// email pattern test
var emailPattern1 = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
var email="vijaykumar.b@sreenidhi.edu.in";
var result=emailPattern1.test(email);
console.log(result);

// email pattern exec
var emailPattern2 = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
var email="vijaykumar.bsreenidhi.edu.in";
var result=emailPattern2.exec(email);
console.log(result);

var pattern = /apples/;
var str = "I have 2 apples.";
```

Regular Expression

Example: (Continued..)

```
    var result = str.replace(pattern, "oranges");
    console.log(result); // "I have 2 oranges."

    var pattern = /\s+/; // Matches one or more spaces
    var str = "Hello world!";
    var result = str.split(pattern); // ["Hello", "world!"]
    console.log(result);
    </script>
</body>
</html>
```

Output:



```
▶ ['Welcome', index: 0, input: 'Welcome to my world or your World', groups: undefined]
0
true
null
I have 2 oranges.
▶ (2) ['Hello', 'world!']
```

Questions

Short Answer Questions

1. Explain the role of JavaScript in web development.
2. What is the difference between let, const, and var in JavaScript?
3. What is the difference between a function declaration and a function expression?
4. How do you create an array in JavaScript?
5. What is a regular expression, and how is it used in JavaScript?

Questions

Long Answer Questions

1. Discuss the evolution of JavaScript from its inception to its current role in modern web development. How has JavaScript's role expanded beyond client-side scripting, and what are some of the major frameworks and technologies built on JavaScript today?
2. Explain the concept of variable scope in JavaScript. How do global, function, and block scopes differ, and how do closures relate to variable scope? Discuss the implications of scope in writing maintainable and bug-free code.
3. Discuss the importance of array manipulation in JavaScript. How can array methods like push, pop, shift, unshift, splice, and slice be used to add, remove, or modify elements within an array? Provide examples to illustrate their usage.
4. Explain the concept of regular expressions in JavaScript. How can regular expressions be used for pattern matching and string manipulation? Provide examples of how regular expressions can be used to validate user input, search for patterns within a string, and perform search-and-replace operations.

Thank You