Contents

UNIT 5 – Introduction to JavaScript

5.0 Structure of Introduction to JavaScript

5.1 Learning Outcomes:

After the successful completion of this unit, the student will be able to:

- Explain the basics and key features of JavaScript.[L2]
- Describe the data types, input/output, control statements and loops, and string and array methods used in JavaScript.[L2]
- Describe the use of functions and pattern-matching concepts in developing client-side scripts for validation.[L2]

5.2 Overview of JavaScript

JavaScript is a versatile, interpreted scripting language that is used in developing web applications. It was created by Netscape in 1995 and has since become an essential tool for building interactive and dynamic websites. It is commonly used for creating interactive web pages, manipulating the Document Object Model(DOM), and handling events in web

pages. JavaScript is a weakly typed language(dynamic) and can be used for client-side as well as server-side development.

Key Features of JavaScript:

- It is supported by all major browsers and provides a built-in execution environment.
- It is a weakly typed language (dynamic language), which means variables, elements, and objects can be declared, altered, and modified during runtime.
- It is case-sensitive.
- It is event-driven, meaning it can respond to user actions like clicks, keyboard input, and mouse movements.
- It is single-threaded, meaning it processes one task at a time. However, asynchronous programming techniques can be used to handle multiple tasks concurrently.
- It is versatile and can be used for a wide range of applications, from creating web applications and client-side validation scripts to server-side applications using technologies like Node.js

5.3 JavaScript Syntax

JavaScript has a 'C' programming language-like syntax. JavaScript consists of variables, operators, strings and arrays, control statements, loops, functions, and many built-in methods.

There are two ways to embed JavaScript code in XHTML documents.

- Directly
- Indirectly

Embedding JavaScript Code directly in XHTML document:

It can be directly embedded in an XHTML document by using <script> tags within the <head> or <body> section of a web page.

Example: To display the message "Welcome to Graphic Era University" the statement can be written within the <script> tag as

```
<script type="text/javascript" >
     document.write("<h2> Welcome to Graphic Era University </h2>");
</script>
```

Here,

`document.write()` is used to write the output to the browser screen.

The `type="text/javascript"` attribute is used to specify the type of script to be included.

XHTML document: **"directjs.html"**

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
 <head>
 <title>Example of Embedding JavaScript Code directly in XHTML </title>
 </head>
  <body>
   <script type="text/javascript" >
     document.write("<h2> Welcome to Graphic Era University </h2>");
   </script>
</body>
</html>
```

Figure 5.1: Snippet of Embedding JavaScript Code directly in XHTML document

On opening the XHTML document **"directjs.html"** in the browser the output is :

Figure 5.2: Output of JavaScript Code given in Figure 5.1

Embedding JavaScript Code indirectly in XHTML document:

External JavaScript files having the extension of ".js" can be indirectly embedded in an XHTML document by using the <script> tags and specifying the external file names in its "src" attribute.

```
<script type="text/javascript"  src="display.js">
     displayfunc()
</script>
```

Here,

A call to the `displayfunc()` function is made from within the <script> tag. The function is present in the external JavaScript file `"display.js"`.

External JavaScript file: "display.js"

```
function displayfunc(){
 document.write("<h2> Welcome to Graphic Era University </h2>");
}
```

Figure 5.3: Snippet of External JavaScript Code

XHTML document: **"indirectjs.html"**

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
  <title>Example of Embedding External JavaScript Code in XHTML</title>
  <script type="text/javascript" src="display.js">
  </script>
  </head>
  <body>
   <script type="text/javascript" >
         displayfunc();     // call to javascript function
   </script>
</body>
</html>
```

Figure 5.4: Snippet of Embedding JavaScript Code indirectly in XHTML document

On opening the XHTML document **"indirectjs.html"** in the browser the output is :

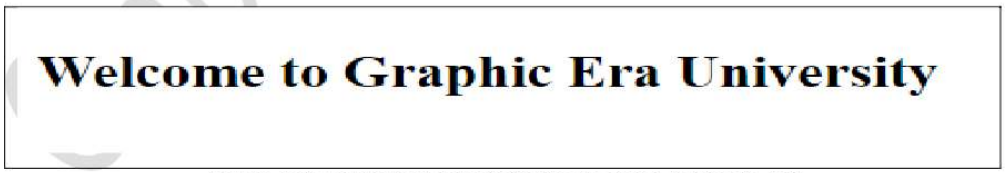

Figure 5.5: Output of JavaScript Code given in Figure 5.4

The external javascript has the advantage of hiding the scripts from the browser users. External javascripts also help in separating the computational javascript code from the layout and presentation provided by XHTML and CSS. Depending on the number of lines of javascript code either internal or external scripts can be used as per the convenience and the requirements of the user.

Comments:

Comments are used to add explanations and notes within the code. JavaScript supports two types of comments: Single line and multi-line comments

Single-line comment - // An example of a single-line comment

Multi-line comment - /* An example of
multi-line comment */

5.4 Variables and Data Types

JavaScript is a weakly typed (dynamic) language. Variables in JavaScript can be assigned and re-assigned values of all types, they are not directly associated with any particular data type.

Variables are used to store data. They are declared using the keywords var, let, or const.

Example:

```
var   x = 10;
let   y = 20;
const z = 100;
```

Rules to be followed while declaring variables:

- Variable names cannot begin with digits [0-9].
- They can begin with letters [a-z or A-Z], underscore ("_"), or dollar "$" sign, and subsequent characters can be digits or letters.
- They are case-sensitive

The primitive data types, operators, and expressions used in JavaScript are similar to programming languages like 'C' and C++.

JavaScript has several primitive data types –

- String: Text data, sequence of characters enclosed in quotes(single or double quotes).

  Example:
  ```
  var s1 = "Graphic" ;
  var s2 = 'Era' ;
  ```
- Number: Numeric data.

  Example:
  ```
  var x = 50 ;
  var y = 10.5 ;
  ```
- Boolean: Represents true or false.
- Undefined: Represents undefined value If a variable has been explicitly declared, but not assigned a value, then it is undefined

Example:
```
var x = 10;
var y;
var sum = x + y ;
```
here y is undefined since it is declared but not assigned a value earlier.

- Null: Represents null value i.e., no value at all. A variable is null if it has not been explicitly declared or assigned a value.

  Example:
  ```
  var a = 10;
  var sum = a + b ;
  ```
  here `b` is null since it is neither declared nor assigned a value earlier. The use of a null value results in a run-time error.

JavaScript has non-primitive data types -

- Object: Represents an instance, the members can be assigned and accessed dynamically.

  Example:
  ```
  var studobj = new Object();        // Creating an Object
  studobj.fname = "Mary";            // Initialize the fname property
  var x = studobj.fname;             // accessing fname property
  ```
- Array: An ordered collection of values.

  Example: `var ary = [10,20, "a", "b",50];`

JavaScript has various operators like 'C' programming language as follows -

Numeric operators → `"+","-","*","/","%","++","--"`, etc.
Logical Operators → &&, ||, !
Relational Operators → `"<", ">", ">=", "<=","===" , "==", "!=",` etc.
String Concatenation operator → "+"

It has various built-in Objects like Math, Date, and Number Objects that can be used in developing web applications

Math - provides various mathematical methods like – Math.min(), Math.max(), Math.pow(), Math.sqrt(), Math.sin(), Math.cos(), Math.log(), etc.

Date - provides various properties and methods like toLocaleString, getDate(), getMonth(), getDay(), getTime(), getHours(), getMinutes(), etc.

Number - MAX_VALUE, MIN_VALUE, NAN, PI, etc.

Conversions in JavaScript can be done in 2 ways - Implicit and Explicit conversions

Explicit conversions can be done by using – toString(), String() methods

5.5 Screen Output and Keyboard Input

JavaScript is interpreted by the browser when it finds the script in the XHTML document. The normal output of JavaScript is displayed on the browser screen. For Screen Output and Keyboard Input, the following methods are used in JavaScript.

- **document.write()**
- **console.log()**
- **prompt()**
- **alert()**
- **confirm()**

- **document.write()**

  The write() method is used to dynamically write content to a web page that gets displayed on the screen. It is mainly used for testing purposes. This method can contain XHTML tags in the arguments.

  Syntax :

  ```
  document.write(expr1, expr2, expr3,...,exprN);
  ```

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
  <!DOCTYPE html>
  <html>
    <head>
       <title> Example of document.write() method </title>
    </head>
    <body>
     <script type="text/javascript" >
      document.write("<h2> Graphic Era University </h2>");
      document.write(" <b> Welcome </b> <br/> ", " <br/> to",
                     " <h3> learning JavaScript </h3> ");
     </script>
  </body>
</html>
```

Figure 5.6: Snippet of using document.write() in JavaScript

Output:

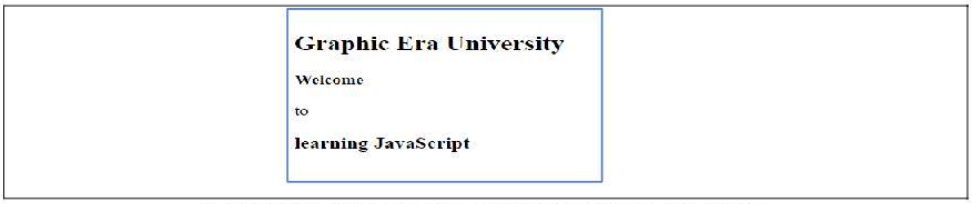

Figure 5.7: Output of using document.write in JavaScript

- console.log(): This method is used to print text or variables to the console, which is often accessible in web browser developer tools. It's a useful way to debug code and display information to the developer.

  Syntax :

  console.log("Content to be displayed!");

  Example :

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of console.log() method </title>
  </head>
  <body>
     <script type="text/javascript" >
      document.write("GEU - Example of console.log method ");
      let x = 100;
      console.log("Value of x is ", x);
     </script>
  </body>
</html>
```

Figure 5.8: Snippet of using console.log

Output:

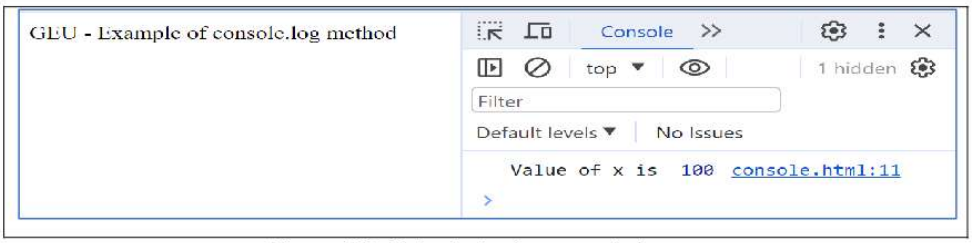

Figure 5.9: Output of using console.log

- prompt()

  In JavaScript, the prompt() method/function is used to obtain input from the user.

  This method is a built-in browser method that displays a prompt box that consists of :

  - A message[label] to the user, that is passed as a first parameter to the method
  - A textbox where the user can enter input. The second parameter passed to the method appears in the textbox.
  - Two buttons "OK" and "Cancel". The user can either enter the textual input in the textbox and click on "OK" or click on "Cancel".

  This method returns the string entered by the user or null if the "Cancel" button is clicked.

  Syntax :

```
var x = prompt([first parameter,][second parameter]);
```

  Both the parameters that are passed to the method are optional.

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of prompt() method </title>
  </head>
  <body>
   <script type="text/javascript" >
     let studname = "";

     studname = prompt("Enter your name","");
     document.write("Welcome to learning JavaScript at GEU :  ",
                    studname);
  </script>
</body>
</html>
```

Figure 5.10: Snippet of using prompt()

Output:

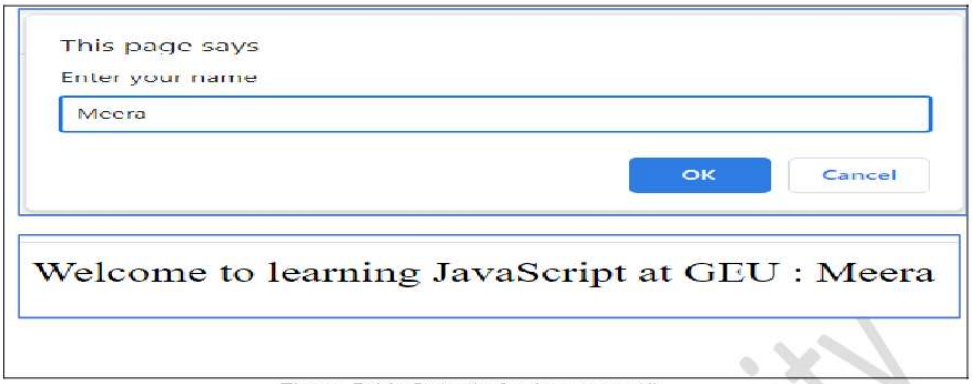

Figure 5.11: Output of using prompt()

In this example, when the JavaScript code is executed, a dialog box appears with the message "Enter your name:" and an input field i.e., a textbox. The user can type/enter their name, and the value they enter is stored in the "studname" variable and displayed using a document.write() method.

- alert()

  This method displays a popup with a message and an "OK" button.

  Syntax :

  ```
  alert(message);   // message is the content to be displayed to the user
  ```

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
     <title> Example of alert method </title>
  </head>
  <body>
     <script type="text/javascript" >
      alert("GEU - Example of alert method ");
      let a = 1000;
       alert("Value of a is " + a);
     </script>
  </body>
</html>
```

Figure 5.12: Snippet of using alert()

Output:

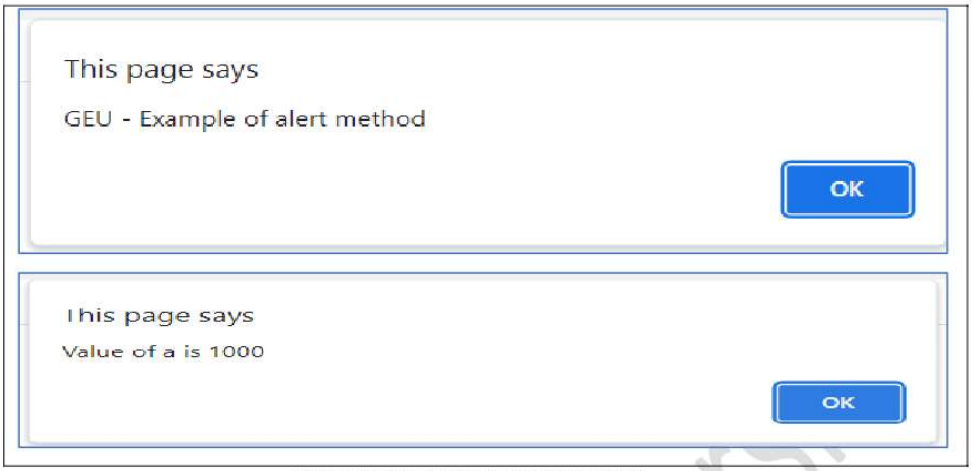

Figure 5.13: Output of using alert()

- confirm()

  In JavaScript, the confirm() method is used to display a dialog box with a message and two buttons: "OK" and "Cancel." It is often used to get/return a binary (yes/no or true/false) response from the user. The confirm() method returns true if the user clicks "OK" and false if the user clicks "Cancel."

  Syntax :

  ```
  confirm(message);    // message is the content to be displayed to the user
  ```

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head> <title> Example of confirm method </title> </head>
  <body>
    <script type="text/javascript" >
       let y = true;
       y = confirm("GEU - Example of confirm method ");
       document.write("Confirm method has returned the value
                      <b ", y,"</b>");
    </script>
  </body>
</html>
```

Figure 5.14: Snippet of using confirm()

Output:

Figure 5.15: Output of using confirm()

confirm() is commonly used in web applications to confirm actions that could have significant consequences, such as deleting a user's account, submitting a form, or proceeding with a critical operation. It's a simple way to obtain user confirmation for such actions.

5.6 Controls and Loops

JavaScript provides control structures and loops that allow you to manage the flow of your code and execute specific tasks repeatedly. These are fundamental to programming and are used extensively in a variety of applications.

Control Structures:

The execution of a program is controlled by the control structures based on specific conditions.

Conditional Statements (if, else if, else): Conditional statements are used to execute different code blocks depending on whether a specified condition evaluates to true or false.

if statement:

Syntax:

```
if (condition) {
    // Code to run if the condition is true
}
```

Example:

```
Let x = 10;
if (x<0) {
    document.write("\n", x, "is a negative number");
}
```

Figure 5.16: Snippet of using if statement

if else statement :

Syntax:
```
if (condition) {
   // Code to run if the condition is true
}
else {
  // Code to run if the condition is not true
}
```

Example:

```
if(x%2==0) {
    document.write("\n ", x, "is a even number");
}
else {
   document.write("\n ", x, "is an odd number");
}
```

Figure 5.17: Snippet of using if else statement

if else if statement :

Syntax:
```
if (condition) {
     // Code to run if the condition is true
  } else if (another condition) {
     // Code to run if the first condition is false and the second condition
    // is true
  } else {
    // Code to run if no conditions are true
  }
```

Example:

```
let percent = 68;
if(percent>=75) {
     document.write("\n First class with Distinction");
}
else if (percent >=60) {
     document.write("\n First class");
}
else {
     document.write("\n Second Class");
}
```

Figure 5.18: Snippet of using if else if statement

Switch Statement: A switch statement allows the execution of different code blocks based on different values of a single expression.

Syntax :

```
switch (expression) {
    case value1:
       // Code to run if the expression matches the value1
        break;
    case value2:
        // Code to run if the expression matches the value2
        break;
    default:
        // Code to run if the expression doesn't match any case
}
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head><title> Example of switch statement </title></head>
  <body>
   <script type="text/javascript" >
       let option = 3;
       switch (option) {
       case 1: document.write("Option is 1");
                break;
       case 2: document.write("Option is 2");
                break;
      case 3:
              document.write("Option is 3 - Exit Program");
              break;
       default:
                document.write("Invalid Option");
       }
  </script>
</body>
</html>
```

Figure 5.19: Snippet of using the switch statement

Loops:

for Loop: The for loop is used to repeatedly execute a block of code as long as a specified condition is true. The for is an entry-controlled loop.

Syntax :

```
for (initialization; condition; increment) {
    // Code to be executed in each iteration
}
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of for loop </title>
  </head>
  <body>
    <script type="text/javascript" >
        let i ;
        let sum = 0;
        for(i=1;i<=5;i++) {
           sum = sum + i;
        }
        document.write("\n Sum of Numbers from  1 to 5 is ", sum);
    </script>
  </body>
</html>
```

Figure 5.20: Snippet of using for..loop

while Loop: The while loop executes a block of code as long as a specified condition remains true. The while loop is called an entry-controlled loop as the condition is checked first and then the statements within the loop are executed.

Syntax:

```
while (condition) {
    // Code - statements to be executed in each iteration
}
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of while loop </title>
  </head>

  <body>
   <script type="text/javascript" >
      let i = 1 ;
      let sum = 0;

     while (i<=5){
        sum = sum + i;
        i= i + 1;
      }

      document.write("\n Sum of Numbers from  1 to 5 is ", sum);

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

Figure 5.21: Snippet of using while loop

do...while Loop: The do...while loop is similar to a while loop but guarantees that the code block will run/execute at least once, even if the condition is false. The do-while loop is known as an exit-controlled loop as the statements in the loop are executed first (at least once) and then the condition is checked.

Syntax:

```
do {
    // Code to be executed in each iteration
} while (condition);
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of do..while loop </title>
  </head>
  <body>
   <script type="text/javascript" >

            let i = 1 ;
            let sum = 0;
            do
            {
              sum = sum + i;
               i= i + 1;
            }while (i<=5);

            document.write("\n Sum of Numbers from  1 to 5 is ", sum);
  </script>
</body>
</html>
```

Figure 5.22: Snippet of using do..while loop

for...in Loop: It is used to loop through the properties of an object.

Syntax :

```
for (variable in object) {
    // Code to be executed for each property of the object
}
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
```

```
    <head>
         <title> Example of for-in loop </title>
    </head>
    <body>
     <script type="text/javascript" >

          var arycolors = ["red", "blue", "green"];
           for(var i in arycolors) {
              document.write("<br> ", i ," - color is ",
                                    arycolors[i]);
           }

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

Figure 5.23: Snippet of using for..in loop

for...of Loop (ES6): It is used to loop through the values of iterable objects like arrays and strings.

Syntax :

```
for (variable of iterable) {
    // Code to be executed for each value in the iterable objects
}
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of for-of loop </title>
  </head>
  <body>
   <script type="text/javascript" >
```

```
        var arycolors= ["red", "blue", "green"];
        for(var i of arycolors) {
           document.writeln("<br> color is - ", i);
        }
  </script>
</body>
</html>
```

Figure 5.24: Snippet of using for..of loop

These control structures and loops are the building blocks of structured programming in JavaScript. They allow us to make decisions, execute code conditionally, and perform repetitive tasks. Understanding how and when to use each of them is crucial for effective programming in JavaScript and other programming languages.

5.7 String Methods

JavaScript strings are used for storing and manipulating text. They are a sequence of characters. The characters are stored in the string starting with index 0.

There are 2 ways of creating strings in JavaScript.

- Using single or double quotes

  Examples:

  let s1 = 'Graphic Era' ;

  let s2 = "Graphic Era" ;

- Using new String() method

  Example:

  let s3 = new String("Graphic Era");

JavaScript provides a wide range of built-in methods for working with strings. These methods allow to manipulate and extract information from strings. Some commonly used string methods with examples are discussed here.

length: Returns the length of a string i.e., the number of characters in a string.

Example - const str1 = "Graphic Era!";

const len = str1.length; // length of the string is 11

charAt(index): Returns the character in the string at the specified index or position.

```
Example  -   const str1 = "Graphic Era";
             const char = str.1charAt(2);            // Character at position 2 is -  'a'
```

indexOf(substring): Returns the index of the first occurrence of the specified substring in the given string and returns -1 if not found.

```
Example  -   const str1 = "Graphic Era";
             const sindex = str1.indexOf("Era");     // "Era"  in starting at index 8
```

substring(startIndex, noofcharacters): Returns a substring from the specified starting index that is given as the first parameter and the number of characters that is given as the second parameter.

```
Example:  const str1 = "Graphic Era";
          const sub1 = str1.substring(0, 7);          // 'Graphic'
          const sub2 = str1.substring(8,3);          // 'Era'
```

slice(startIndex, endIndex): Returns a slice of the string from the start index to the end index (exclusive). Negative indices count from the end.

```
Example:  const str1 = "Graphic Era";
          const slice1 = str1.slice(0, 5);            // 'Graph'
          const slice2 = str1.slice(-3);              // 'Era'
```

lastIndexOf(substring): Returns the index of the last occurrence of the specified substring. Returns -1 if not found.

```
Example:  const str1 = "Graphic Era";
          const lastIndex = str1.lastIndexOf("a");      // 10
          con index1 = str1.lastindexOf("x");           // returns  -1 since "x" is not found
```

startsWith(substring) and endsWith(substring): Check if the string starts or ends with the specified substring, returning a boolean.

```
Example: const str1 = "Graphic Era";
          const startswith = str1.startsWith("Graphic");          // true
          const endswith  = str1.endsWith("Era");                 // true
          const endswith  = str1.endsWith("University");          // false
```

toUpperCase() and toLowerCase(): Convert the string to uppercase or lowercase.

Example: const str1 = "Graphic Era";

```
const upper = str1.toUpperCase();        // 'GRAPHIC ERA'
const lower = str1.toLowerCase();        // 'graphic era'
```

trim(): Removes whitespace (spaces, tabs, and line breaks) from both ends of the string.

Example: const str1 = " Graphic Era ";

```
const trimstr1 = str1.trim();              // 'Graphic Era'
```

replace(search, replace): Replaces the first occurrence of a specified substring with another substring.

Example: const str1 = "Graphic Era";

```
const replaced = str1.replace("Era", "University");    // 'Graphic University'
```

concat(string2, string3, ...): Combines two or more strings and returns a new string.

Example: const str1 = "Graphic Era";

```
const str2 = " University";
const str3 = str1.concat(str2);          // 'Graphic Era University'
```

Example: Using all the string methods discussed here

```
<?xml version="1.0" encoding="utf-8" ?>

<!DOCTYPE html>

<html>
<head>
    <title> Example of String methods in JavaScript </title>
</head>
<body>
   <script type="text/javascript" >


      let str1 = "Graphic Era";
      document.write("<h2> The given string is  : ", str1 , "</h2>");
      document.write("Example of various string methods used in JavaScript <b>
                     : Index starts from '0' </b><br/><br/>");


      // Finding the length of the given string
      let len = str1.length;      // 11
```

```
document.write("The length of the string is <b> : ", len, "</b>
                <br/><br/>");

// Finding the character at a specified position in the given string
const char = str1.charAt(2);     // 'a'
document.write("The character at position 2 in the string is <b> : ",
                char, "</b> <br/><br/>");

// Finding the starting index of a substring in the given string
const sindex = str1.indexOf("Era");   // 8
document.write("The index of 'Era' in the string is  <b> : ", sindex,
                "</b> <br/><br/> ");

// Returning a substring from the given string
const sub1 = str1.substring(0,7);     // 'Graphic'
document.write("The substring starting from index 0 of length 7
                characters in the given string is <b> : ", sub1,
                "</b> <br/><br/>");

// Slicing a substring from the given string
const slice1 = str1.slice(0, 5);    // 'Graph'
const slice2 = str1.slice(-3);      // 'Era'
document.write("Slicing the given string Slice 1 is <b> : ", slice1,
            "</b><br/><br/>");
document.write("Slicing the given string Slice 2 is <b> : ", slice2,
            "</b><br/><br/>");

// Finding the starting index and last index of a substring in the
//  given string
const sindex1 = str1.indexOf("a");   //  2
document.write("The first occurrence (index) of 'a' in the string is
                <b> : ", sindex1, "</b><br/><br/>");
const lastIndex = str1.lastIndexOf("a");
document.write("Last index of 'a' in the given string is <b> : ",
                lastIndex, "</b><br/><br/>");
const lastIndex1 = str1.lastIndexOf("x");
document.write("Last index of 'x' in the given string is <b> : ",
                lastIndex1, "</b><br/><br/>");
```

```
      // Finding a string starting with and ending with
      const startswith = str1.startsWith("Graphic");              // true
      const endswith1  = str1.endsWith("Era");                    // true
      const endswith2  = str1.endsWith("University");             // false
       document.write("The given string is starting with <b> 'Graphic'  : ",
                      startswith, "</b><br/><br/>");
       document.write("The given string is ending with <b> 'Era'  : ",
                      endswith1, "</b><br/><br/>");
       document.write("The given string is ending with <b> 'University'  : ",
                      endswith2, "</b><br/><br/>");

      // Converting string to Uppercase and Lowercase
       const upper = str1.toUpperCase();          // 'GRAPHIC ERA'
       const lower = str1.toLowerCase();          // 'graphic era'
       document.write("The given string in Uppercase is <b> : ", upper,
                     "</b><br/><br/>");
       document.write("The given string in Lowercase is <b> : ", lower,
                     "</b><br/><br/>");

       // Trimming a string from both ends
       const str2 = "   Graphic Era  ";
       const trimstr1 = str2.trim();              // 'Graphic Era'
       document.write("The given string after it is trimmed is <b> : '",
                      trimstr1, "' </b><br/><br/>");

       // Replacing the first occurrence of a string with the given string
       const repstr1=str1.replace("Era", "University"); // 'Graphic University'
       document.write("The given string after replacing is <b> : '", repstr1,
                      "' </b><br/><br/>");

       // Concatenating two strings
       const str3 = " University";
        const str4 = str1.concat(str3);          // 'Graphic Era University'
       document.write("The given string after concatenating is <b> : ", str4,
                      "</b><br/><br/>");
   </script>
</body>
</html>
```

Figure 5.25: Snippet of using string methods in JavaScript

Output:

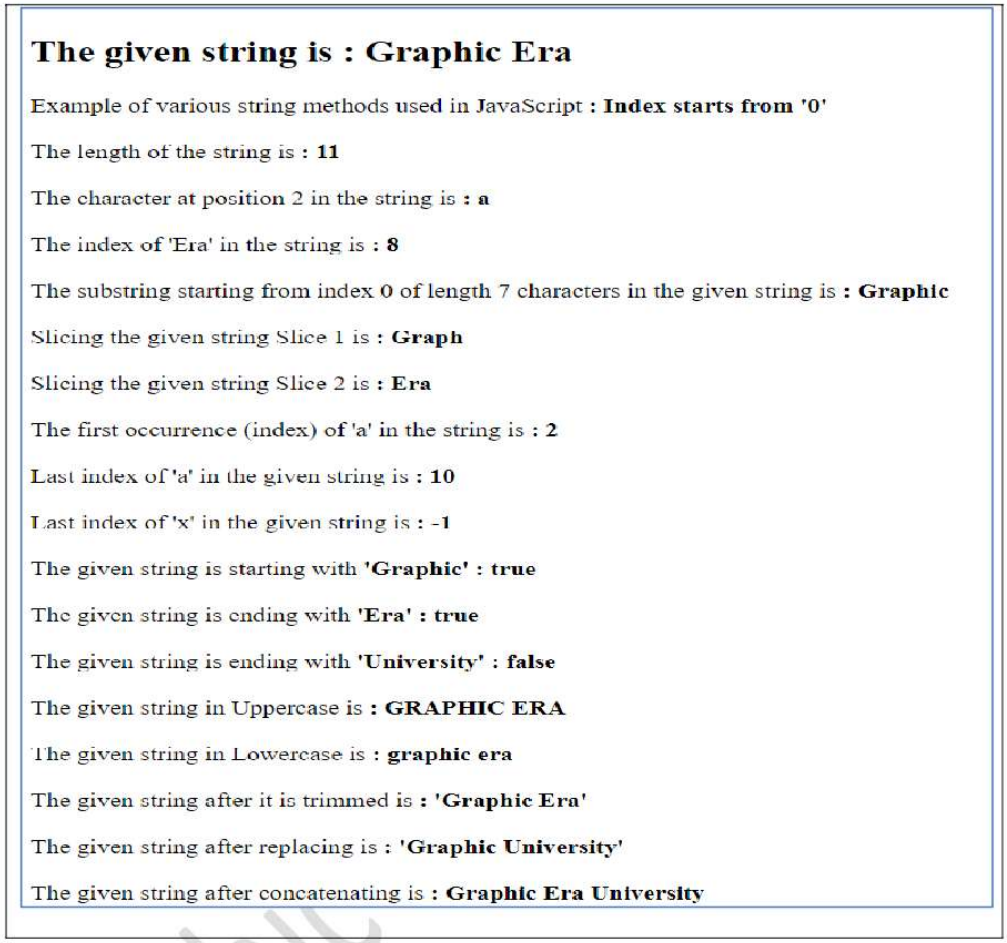

Figure 5.26: Output of using string methods in JavaScript

These are just a few of the many string methods available in JavaScript. They are incredibly useful for various string manipulation tasks in JavaScript programs.

5.8 Arrays

Arrays in JavaScript are ordered collections of data, which can include values of different data types. Arrays in JavaScript have dynamic length. The length/size of the array can be made to grow or shrink dynamically. Arrays are commonly used to store multiple items, making it easy to perform operations on the entire collection or individual elements within it. Arrays can contain a mix of data types, including numbers, strings, objects, and other arrays.

Here are some key concepts and examples related to arrays in JavaScript:

Creating Arrays:

In JavaScript, arrays can be created in the following ways -

- Using an array literal
- Using a new operator along with the Array() constructor

- Creating arrays using an array literal

Arrays can be created in JavaScript using square brackets [] and populating them with values.

```
Example:  const aryfruits = ["apple", "banana", "cherry", "grapes"];
          const arynumbers = [1, 2, 3, 4, 5];
          const arymixed = [1, "apple", true];
          const aryempty = [];
```

- Creating an array using a new operator along with the Array() constructor

Arrays can also be created in JavaScript using Array() constructor and populating them with values. When the Array() constructor has many values, they are considered as elements of the array and when only one value is passed as parameter to the Array() constructor it is considered as the size of the array with no elements in it.

Example 1: Creating an array of mixed data type

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes");
```

Here the array - "ary1" is created of length 6

Example 2: Creating an array of size 100

```
const ary1 = new Array(100);
```

Here the array - "ary1" is created of size 100 with no elements in it

Accessing Array Elements:

Elements in the array can be accessed using their index. The index starts at 0 for the first element.

Example 1:

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes");
var elmt1 = ary1[0];      // "apple"
var elmt2 = ary1[1];      // 100
var x = ary1[10];         // undefined
```

Size of the Array:

The length of the array is the highest subscript to which the value has been assigned plus one (1).

Example :

```
const ary1 = new Array("apple", 100);
```

If the array ary1 is declared and initialized with 2 elements and the following statement is executed

```
ary1[6] = 500;
```

then, the new size/length of the array will be 7.

highest subscript + 1 i.e. 6 + 1 = 7

All other elements that are not initialized will be undefined. i.e., ary[2], ary[3], ary[4], and ary1[5] will be undefined, whereas ary1[0], ary1[1] and ary1[6] will contain the values "apple", 100, and 500 respectively.

Verifying whether the variable is an array:

Array.isArray() method determines whether the parameter passed to the method it is an array or not. Returns true if it is an array and false otherwise.

Example :

```
const ary1 = new Array("apple", 100);
var bool1 = Array.isArray(ary1);          // returns true since ary1 is an array
var x = 10;
var bool2 = Array.isArray(x);             // returns false since x is not an array
```

Array Length:

The number of elements in an array can be determined by using the length property.

Example 1:

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes");
 const len =  ary1.length;        // 6
```

Modifying Array Elements:

The value of an element in an array can be changed by assigning a new value to it.

Example 1:

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes");
      ary1[1] = 500 ;                  // value 100 is replaced by 500
      ary1[4] = "orange" ;          //  value "cherry" is replaced by "orange"
```

Now the array consists of ["apple", 500, "banana", 200, "orange", "grapes"];

Adding and Removing Elements:

Arrays have several built-in methods for adding and removing elements – push(), pop(), shift() and unshift().

push(): Adds one or more elements to the end of an array.

Example 1:

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes");
    ary1.push("orange");        // "orange" is added
```

Now, array ary1 consists of ["apple", 100, "banana", 200, "cherry", "grapes", "orange"];

pop(): Removes and returns the last element from an array.

Example 1:

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes", "orange");
var elmt1 = ary1.pop();          // "orange" is removed
```

Now, ary1 consists of ["apple", 100, "banana", 200, "cherry", "grapes"];

unshift(): Adds one or more elements to the beginning of an array.

Example 1:

```
const ary1 = new Array("apple", 100, "banana", 200, "cherry", "grapes");
    ary1.unshift("watermelon");    // "watermelon"  is added
```

Now, ary1 consists of ["watermelon", "apple", 100, "banana", 200, "cherry", "grapes"];

shift(): Removes and returns the first element from an array.

Example 1:

```
const ary1 = new Array("watermelon", "apple", 100, "banana", 200, "cherry",
                       "grapes");
var elmt1 = ary1.shift();       // "watermelon"  is removed
```

Now, ary1 consists of ["apple", 100, "banana", 200, "cherry", "grapes"];

Iterating Through Arrays:

To loop through the elements in an array, various looping constructs like → for, for...of, and forEach can be used.

Example:

```
const ary1 = new Array("apple", 50, "banana", "grapes");
```

for loop: Loops through a block of code a number of times

// Iterating using - for loop

```
for (let i = 0; i < ary1.length; i++) {
        console.log(ary1[i]);
}
```

```
for (let i = 0; i < ary1.length; i++) {
        document.write(ary1[i]);
}
```

Figure 5.27: Snippet of iterating through an Array using for ..loop in JavaScript

for...of loop: Loops through the values of an iterable object

// Iterating using - for...of loop

```
for (const i of ary1) {
    console.log(i);
 }
```

```
for (const i of ary1) {
      document.write(i);
}
```

Figure 5.28: Snippet of iterating through an Array using for …of loop in JavaScript

forEach loop: It is used to iterate through all types of arrays

// Iterating using - forEach

```
ary1.forEach(function (x) {
        console.log(x);
});
```

```
ary1.forEach(function (x) {
    document.write(x);
});
```

Figure 5.29: Snippet of iterating through an Array using forEach loop in JavaScript

Array Methods:

JavaScript provides many built-in methods for manipulating arrays. Some common array methods include join(), slice(), concat(), reverse(), and sort().

join() – This method converts all the elements in an array to a string and concatenates them to a single string. The values in the string are separated by a comma "," if no parameter is passed to the join() method, otherwise, the separator passed as a parameter to the join() method is used as a delimiter. The original array remains unaffected.

Example:

```
const ary1 = new Array("apple", 50, "banana", "grapes", "orange");
var str1 = ary1.join(":");
```

The value of the string str1 will be → "apple:50:banana:grapes:orange".

slice() –

It slices the array object depending on the parameters passed to the slice() method and returns part of the array object. The returned array object contains elements of the array object through which it is called from the first parameter up to the second parameter but not including it. The original array on which it is called is unaffected.

Example:

```
const ary1 = new Array("apple", 50, "banana", "grapes", "orange");
  var ary2 = ary1.slice(1,3);
```

After slicing the new array ary2[] will consists of - 50,banana

concat() –

The concat() method catenates the actual parameters passed, to the array object on which the method is called. The parameters can be strings, numbers, and even other arrays. The parameters are appended to the array object on which the method is called. It returns a new array object after concatenating. The original array remains unaffected.

Example:

```
const ary1 = new Array("apple", 50, "banana", "grapes", "orange");
  var ary2 = new Array(100,200,"orange");
  var ary3 = ary1.concat(ary2);
```

The arrays ary1[] and ary2[] are unaffected after the concat() operation. The new array ary3[] contains the elements of ary1[] and ary2[].

The contents of ary3[] will now be → **apple,50,banana,grapes,orange,100,200,orange**

reverse() –

The contents of the array are reversed. While reversing the reversed contents can also be stored in another new array. The original array also gets affected during reversing, so to preserve the contents of the original array it can be assigned to another array before reversing by using the concat() method.

Example:

```
const ary1 = new Array("apple", 50, "banana", "grapes", "orange");
var tempary = ary1.concat();      // preserving the original array in temporary array
var ary2 = ary1.reverse();
```

The original array ary1[] and new array ary2[] will contain

→ **orange, grapes, banana, 50, apple**

Example using array methods join(), slice(), concat() and reverse()

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of array methods in JavaScript </title>
  </head>
  <body>

   <script type="text/javascript" >

     const ary1 = new Array("apple", 50, "banana", "grapes", "orange");
     document.write("<b> Example of array methods in JavaScript </b>
                    <br/> <br/>");

     document.write("<b> Initial array is :  ", ary1, "</b> <br/>");

     // Array.isArray() method
     document.write("<b> Array.isArray() method : </b> <br/>");
     var bool1 = Array.isArray(ary1);
      document.write("It is an array <b> --> ", bool1," </b> <br/>");

      var x = 10;
      var bool2 = Array.isArray(x);
      document.write("var x = 10 is an array <b> --> ", bool2,"
                     </b> <br/>");
      // join() method
      document.write("<br/><b>Join method </b> <br/>");
      var str1 = ary1.join(":");
      document.write("Array is converted to String using join(':')  <b>
                     --> ", str1 ," </b> <br/>");
      // slice() method
      document.write("<br/><b>Slice method </b> <br/>");
      var ary2 = ary1.slice(1,3);
      document.write("New array returned after using slice(1,3) <b>
                     --> ", ary2 ," </b> <br/>");
```

```
      document.write("Original array is unaffected <b> --> ", ary1,
                     "</b> <br/>");

      // concat() method
      document.write("<br/><b>Concat method </b> <br/>");
      document.write("Original array before concatenating <b> --> ",
                       ary1 ," </b> <br/>");
      var ary2 = new Array(100,200,"orange");

      var ary3 = ary1.concat(ary2);

      document.write("Second Array  <b> --> ", ary2 ," </b> <br/>");
      document.write("New Array after concatenating  <b> --> ", ary3,
                     " </b> <br/>");
      document.write("Original Array remains unaffected  <b> --> ",
                      ary1," </b> <br/>");

      // reverse() method
      document.write("<br/><b>Reverse method </b> <br/>");
      document.write("Array before reversing <b> --> ", ary1 ,"
                     </b> <br/>");
      var empary1 = ary1.concat();
      document.write("Temporary array preserved before reversing  <b>
                     --> ", tempary1 ," </b> <br/>");

      var ary5 = ary1.reverse();
      document.write("Original array is affected after reversing  <b>
                     --> ", ary1," </b> <br/>");
      document.write("New array after reversing  <b> --> ", ary5,
                     " </b> <br/>");
  </script>
</body>
</html>
```

Figure 5.30: Snippet of using array methods in JavaScript

Output: Various Array methods used in JavaScript

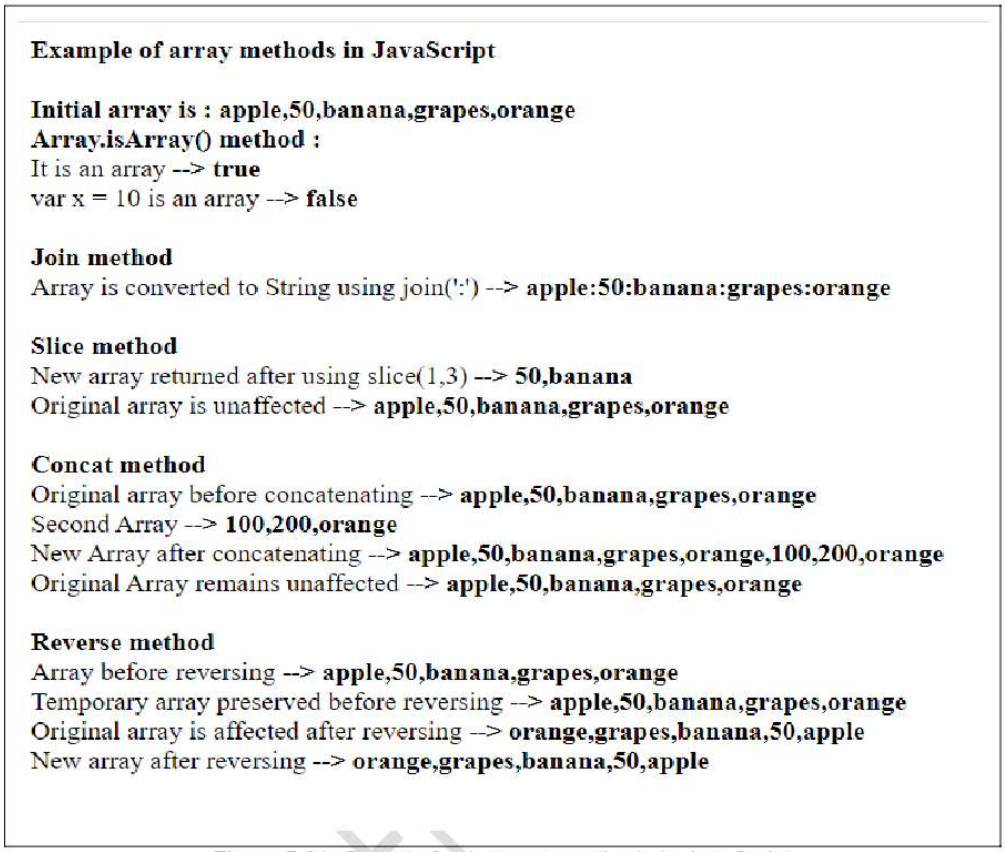

Figure 5.31: Output of using array methods in JavaScript

Sorting of Arrays in JavaScript

sort() –

This method sorts the array in ascending string order by default. While sorting the elements are converted to string and sorted in place which means that the changes are made in the original array and no copy is kept. Before sorting since the original array also gets affected it can be preserved by copying the contents to another array by using the concat() method.

To sort the elements in some order other than strings, the comparison function can be used. The comparison function compares all the elements of the array by passing two values at a time as parameters to the function.

Example

```
const ary1 = new Array(50,20,1000,10,500);
ary1.sort((a, b) => a - b );          // for sorting in ascending order
ary1.sort((a, b) => b - a );          // for sorting in descending order
```

The compare function returns a value.

- If the function returns a value < 0 then a is sorted before b (i.e., a comes before b)
- If the function returns a value > 0 then b is sorted before a (i.e., b comes before a)
- If the function returns a value == 0 then a and b remain unchanged

Example: Sorting elements in the array using sort() and even compare function

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
<head>
   <title> Example of Sorting  using sort() and compare function
   </title>
</head>
<body>
<script type="text/javascript" >
   let ary1 = new Array(50,20,1000,10,500);
    document.write("<h3> Initial array is :  ", ary1, " </h3>");
    document.write("<h3> Example of sorting arrays in JavaScript
                    using simple sort() method </h3>");
   var tempary = ary1.concat();
   ary1.sort();
   document.write("Original array is affected after sorting <b>--> ",
                  ary1 ," </b> <br/><br/>");
   // Sorting using compare function
   document.write("<h3> Example of sorting arrays in JavaScript using
                  sort() and compare function </h3>");
   ary1.sort((a, b) => b - a );      // descending order
   document.write("Array after sorting in descending order <b> --> ",
                  ary1 ," </b> <br/> <br/>");
   ary1.sort((a, b) => a - b );     // ascending order
   document.write("Array after sorting in ascending order <b> --> ",
                  ary1 ," </b> <br/>");
  </script>
</body>
</html>
```

Figure 5.32: Snippet of using sorting methods with arrays in JavaScript

Output:

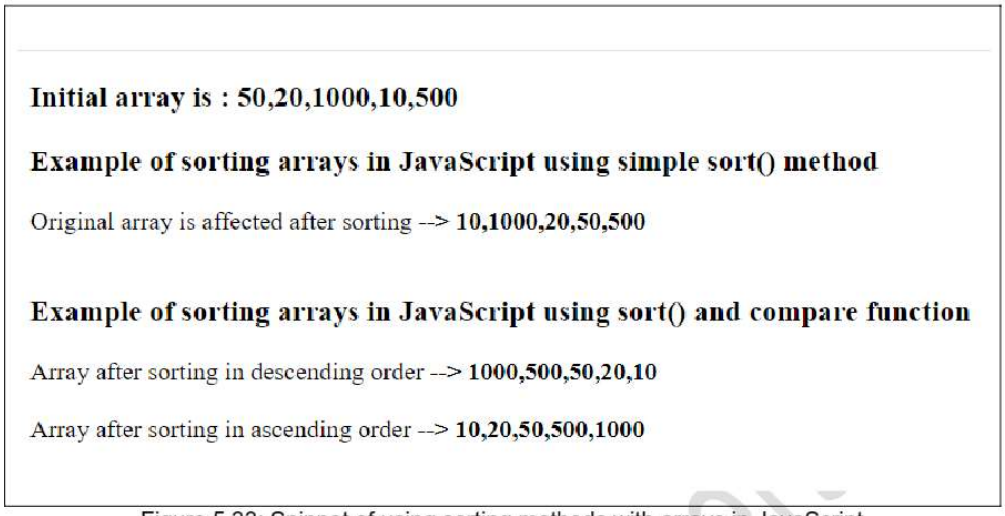

Figure 5.33: Snippet of using sorting methods with arrays in JavaScript

Arrays are versatile data structures in JavaScript and are widely used in many programming scenarios, such as storing lists of items, managing data, and performing various operations on collections of elements.

5.9 Functions

Functions in JavaScript are blocks of code that can be defined and called to perform specific tasks. They are similar to functions defined in 'C' and C++ programming languages. They are a fundamental concept in JavaScript and enable code reuse and organization. Defining functions reduce the time required to write many lines of code to perform similar task each time. It makes the programs compact due to less coding.

Defining Functions:

Functions are defined in JavaScript using the function keyword. A function consists of a function header and statements that describe the actions to be performed. Functions can take parameters, execute a block of code, and return a value.

Syntax:

```
function function-name([arg1, arg2, arg3, .... argN]){
    // code to be executed
}
```

Functions may have 0 or more arguments. Arguments are optional.

JavaScript function with no parameters/arguments

Functions can be defined without parameters(input values) to perform some actions.

Example:

```
// A simple function that has no parameters
function display() {
   document.write("JavaScript function with no arguments");
}
```

JavaScript function with parameters

Functions can accept parameters(input values) to perform actions or calculations based on those values.

Example:

```
// A function that takes parameters
function sumofargs(a, b) {
    var sum =  a + b;
    document.write("Sum of ", a," and b is: ", sum);
}

sumofargs(10, 20);   // function call
```

JavaScript function with parameters, and which returns a value

Functions can return values using the return statement. This value can be used in other parts of the code.

Example:

```
// A function that takes 2 parameters and returns a value
function sumofargs(a, b) {
    var sum =  a + b;
    return sum;
}

var sumab = sumofargs(a, b);   // function call
```

JavaScript anonymous function with parameters, and which returns a value

Functions can be defined without names, often called anonymous functions, and assign them to variables. These are commonly used in scenarios like callbacks and event handling.

Example:

```
// An anonymous function assigned to a variable (function expression)
const prod = function(x, y) {
    return x * y;
};

var prodxy = prod(2, 3);   // function call
```

JavaScript Arrow Functions (ES6):

Arrow functions provide a more concise way to define functions. They are often used in modern JavaScript.

Example:

```
const add = (a, b) => a + b;
const square = x => x * x;
const displaymsg = () => console.log("Welcome to GEU!");
```

Example: Using JavaScript function to calculate a factorial of a given number

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
<head>
   <script type="text/javascript" >
       //function to find factorial of a number n
       function fact(n)
       {
         var prod = 1;
          for(var i=1;i<=n; i++)
         {
            prod = prod * i;
          }
          document.writeln("<h3>Factorial of ", n , " is :  ",
                           prod,"</h3>");
       }
  </script>
</head>
```

```
<body>
 <b> Program to find factorial of a given number between 1 to 10 </b>
 <script language="javascript">
      var n;
       do
       {
         n=prompt("Enter a number between 1 to 10 "," ");
          // validating input - number between 1 to 10
          if ((n <= 0) || (n >=11)) {
             alert("Invalid Input");
          }
       }while(n<=0);
        fact(n);         // function call
  </script>
</body>
</html>
```

Figure 5.34: Program to find the factorial of a given n using JavaScript function

Output:

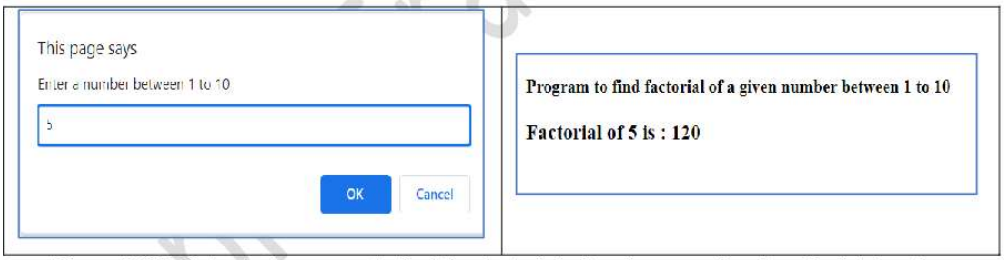

Figure 5.35: Output of program to find the factorial of a given n using JavaScript function

Functions- Scope of Variables:

Variables declared within a function are locally scoped and are not accessible outside the function unless explicitly returned. Variables declared outside functions are global and can be accessed within and outside functions.

Example:

```
function add(a, b) {
    const result = a + b;       // variable result has local scope
    return result;
}
console.log(result); // Throws an error, result is not defined outside the add function
```

Example: Usage of local and global variable in function in JavaScript

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
<head>
 <script type="text/javascript" >
   //function to demonstrate local and global variables in  JavaScript
  function display()
  {
    var a = 10;           // local variable in function
    n = n + 1;            // global variable accessed in function
    document.writeln("<h3>Value of local variable a in function is ",
                     a, "</h3>");
    document.writeln("<h3>Value of global variable n in function is ",
                     n ,"</h3>");
  }
 </script>
</head>

<body>
 <b> Program to demonstrate local and global variables in JavaScript </b>

 <script language="javascript">
   var n = 5;          // global variable declared and initialised
   document.writeln("<h3>Value of n before function call is ", n,
                     "</h3>");
   display();        // function call
   document.writeln("<h3>Value of global variable n after  function
                    call is ", n, "</h3>");
   document.writeln("<h3>Value of local variable a after  function call
                     is ", a, "</h3>");
</script>
</body>
</html>
```

Figure 5.36: Program to demonstrate the scope of local and global variables in JavaScript

Output:

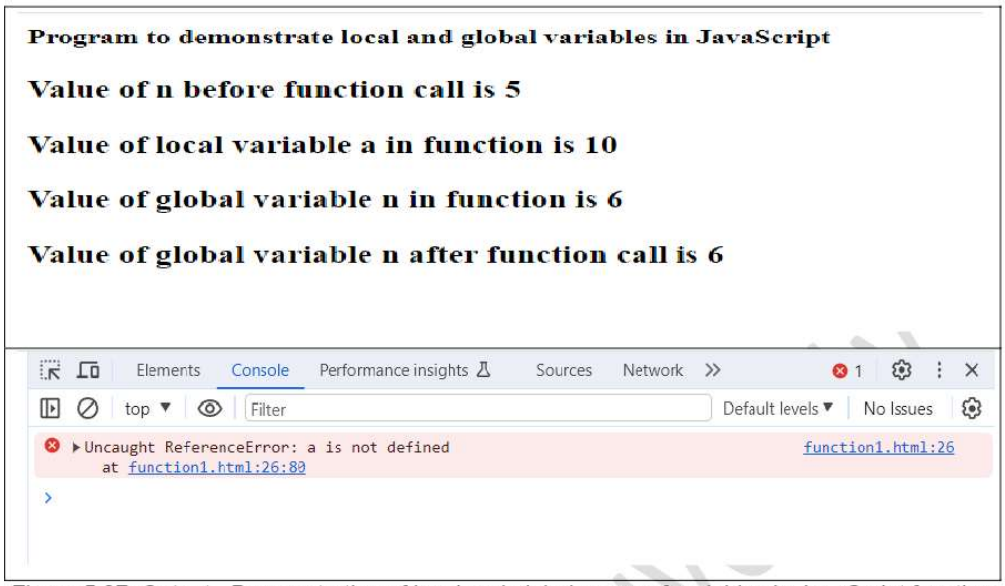

Figure 5.37: Output - Demonstration of local and global scope of variables in JavaScript function

5.10 Pattern Matching

Pattern matching is a powerful concept in programming that allows matching and manipulation of data based on specific patterns or structures. In JavaScript, pattern matching is often implemented using techniques like regular expressions and de-structuring.

Character and Character Classes in JavaScript

In JavaScript, regular expressions, characters, and character class patterns can be used to match specific characters or sets of characters in strings. These patterns are defined using various metacharacters and square brackets within regular expressions. Here's an overview of character and character class patterns as shown in Table 5.1.

Table 5.1: Pre-defined character classes in JavaScript

Name	Equivalent	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z0-9]	An alphanumeric character
\W	[^A-Za-z0-9]	Not an alphanumeric character
\s	[\t\n\f\r]	A whitespace character
\S	[^ \t\n\f\r]	Not a whitespace character

Character Patterns:

Single Character Match: To match a specific single character, simply include that character in the regular expression. For example, /a/ will match the character 'a' in a string.

```
let str1 = "apple";
let pattern = /a/;                  // matches the character 'a'
let result = pattern.test(str1);    // returns true since 'a' is found
```

Character Class Patterns:

Character Class [...]: A character class is defined using square brackets [...] and can match any one of the characters listed inside the brackets. For example, /[aeiou]/ matches any vowel.

```
let str2 = "apple";
let pattern = /[aeiou]/;            // matches the vowels ['a', 'e']
let result = pattern.test(str2)     // returns true since vowels are found
```

Negated Character Class [^...]: A negated character class matches any character not listed inside the brackets. For example, /[^0-9]/ matches any non-digit character.

```
let str3 = "Era123";
let pattern = /[^0-9]/;             // matches characters and not digits
let result = pattern.test(str3);    // returns true since characters are found
```

Ranges in Character Class [a-z]: The range of characters can be defined in a character class. For example, /[a-z]/ matches any lowercase letter.

```
let str4 = "WELCOME TO GEU";
let pattern = /[a-z]/g;             // matches lowercase characters
let result = pattern.test(str4);    // returns false since lowercase characters
                                    // are not found
```

Character Class Shorthand: There are shorthand representations for common character classes, such as \d for digits, \w for word characters (alphanumeric and underscore) and \s for whitespace characters.

Example 1:

```
let str1 = "Welcome 2024 Happy New Year";
let pattern = /\d\s\w/g;             // matches a digit, space and a word
let result = pattern.test(str1);     // returns true since pattern is found
```

Here, the pattern ["4 H"] is found

Example 2:

```
let str2 = "23OMC101";
let pattern = /^\d{2}OMC\d{3}/;
let result = pattern.test(str2);          // returns true
```

Here, the pattern matches 2 digits in the beginning, followed by the characters "OMC" and 3 digits. Here, {X} is used for repeating a pattern where "X" indicates the number of times to repeat a pattern.

These character and character class patterns allow to match specific characters or sets of characters in strings. They provide a flexible and powerful way to search for and manipulate data in JavaScript using regular expressions.

Anchor and pattern modifiers in JavaScript

In JavaScript, "anchor" and "pattern modifiers" usually refer to elements within regular expressions. Regular expressions are patterns used to match character combinations in strings, and they can include anchors and modifiers to refine the matching behavior.

Anchors and common pattern modifiers in JavaScript:

Anchors:

^ (Caret): The caret is used to anchor the pattern to the start of a string. For example, /^Learning/ will match any string that starts with "Learning"

```
const str1 = "Learning JavaScript";
const pattern = /^Learning/;          //true
```

$ (Dollar Sign): The dollar sign is used to anchor the pattern to the end of a string. For example, /JavaScript$/ will match any string that ends with "JavaScript"

```
const str1 = "Learning JavaScript";
const pattern = /JavaScript$/;        //true
```

\b (Word Boundary): The word boundary anchor is used to match a pattern only when it appears at a word boundary. For example, /\bwhite\b/ will match the word "white".

```
const str1 = "I have a white cat";
const pattern = /\bwhite\b/;      //true
```

Pattern Modifiers:

Pattern modifiers are used in regular expressions to change the way they match patterns. They are specified after the closing delimiter / and can include: i, g

i (Case Insensitive): The i modifier makes the pattern matching case-insensitive.

```
const str1 = "Graphic Era";
const pattern = /Era/i;        // matched ["Era"] ignores case
```

g (Global): The g modifier makes the pattern match globally, so it finds all occurrences in the string rather than just the first one.

```
const text = "apple, apple, banana";
const pattern = /apple/g;     // matches ["apple", "apple"]
```

These are some common anchors and pattern modifiers in JavaScript regular expressions that help to specify where and how to search for patterns in strings.

Regular Expressions:

Regular expressions are a way to describe and match patterns within strings. They are frequently used for tasks like text searching, validation, and manipulation.

JavaScript provides three methods for pattern matching and manipulation: search(), match(), and replace(). These methods are often used with regular expressions for more advanced pattern matching.

search() Method:

The search() method is used to search for a specified pattern within a string. It returns the index of the first occurrence of the pattern, or -1 if no match is found.

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of pattern search  method in JavaScript </title>
  </head>
  <body>
   <script type="text/javascript" >
```

```
      document.write("<h3> Graphic Era University </h3>");
      document.write("<h3> Example of pattern search  method in
                     JavaScript <h3> ");

      const str1 = "Graphic Era!";
      const pattern = /Era/i;     // case-insensitive search for "Era"
      const result = str1.search(pattern);

      document.write("<h3> Given String is : ", str1 ,"</h3>");
      document.write("<h3> Pattern to search is : 'Era' </h3>");
      if (result !== -1) {
            document.write("<h2> Pattern found at index : ",
                                 result, "</h2>");
      } else {
            document.write("<h2> Pattern not found. </h2>");
      }
  </script>
</body>
</html>
```

Figure 5.38: Program to demonstrate search pattern matching method in JavaScript

In this example, search() is used to find the index of the pattern "Era" (case-insensitive) within the string str1.

Output:

Graphic Era University

Example of pattern search method in JavaScript

Given String is : Graphic Era!

Pattern to search is : 'Era'

Pattern found at index : 8

Figure 5.39: Output of program to demonstrate search pattern matching method in JavaScript

match() Method:

The match() method is used to extract the matched substrings from a string based on a regular expression pattern. It returns an array of matched substrings or null if no matches are found.

Example 1:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of pattern match method in JavaScript </title>
  </head>
  <body>
   <script type="text/javascript" >
     document.write("<h3> Graphic Era University </h3>");
     document.write("<h3> Example of pattern match  method in
                      JavaScript <h3> ");

     const str1 = "The rose is red. The Rose is beautiful";

     const pattern = /rose/ig;  // Global match for "rose" and ignore case

     const arymatches = str1.match(pattern);
     document.write("<h3> Given String is : ", str1 ,"</h3>");
     document.write("<h3> Pattern to search is : ", pattern ,"</h3>");

     if (arymatches == null) {
       document.write("<h3> Match not found. </h3>");
     }
    else {
       document.write("<h3> Match(es) found : ", arymatches, "</h3>");
    }
  </script>
</body>
</html>
```

Figure 5.40: Program to demonstrate match() pattern matching method in JavaScript

Output:

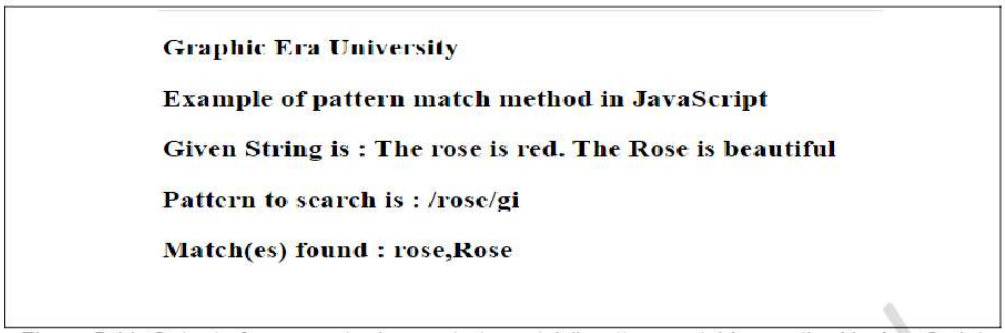

Figure 5.41: Output of program to demonstrate match() pattern matching method in JavaScript

In the program given in Figure 5.36. if only the pattern is changed to

```
const pattern = /rose/g;          // Global match for "rose"
```

The Output will be :

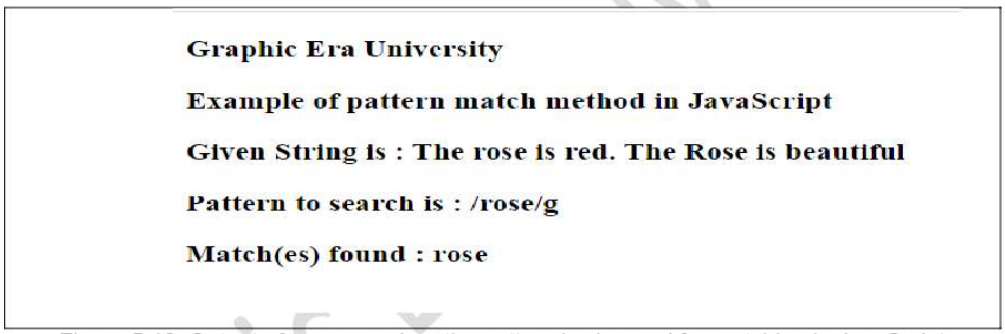

Figure 5.42: Output of program when the pattern is changed for matching in JavaScript

In the program given in Figure 5.36. if only the pattern is changed to

```
const pattern = /Lily/g;          // Global match for "Lily"
```

The Output will be :

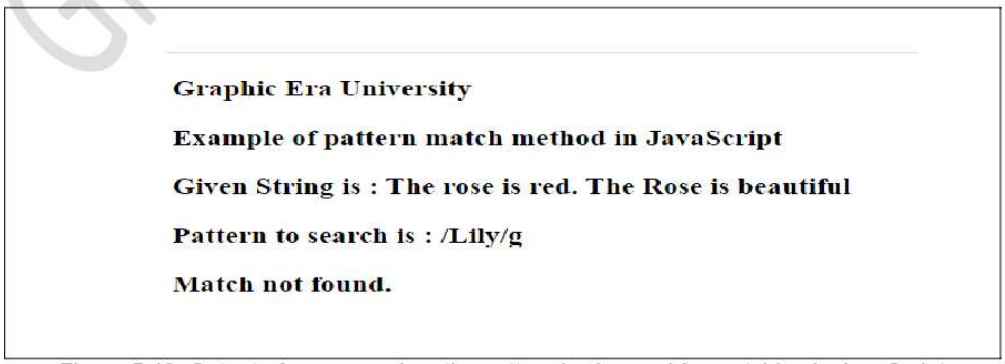

Figure 5.43: Output of program when the pattern is changed for matching in JavaScript

replace() Method:

The replace() method is used to replace a specified pattern within a string with another string or a function. It returns a new string with the replacements made.

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of pattern replace method in JavaScript </title>
  </head>
  <body>
   <script type="text/javascript" >
      document.write("<h3> Graphic Era University </h3>");
      document.write("<h3> Example of pattern replace  method in
                     JavaScript <h3> ");

      const str1= "The rose is white. The rose is cute!";
      const pattern = /rose/ig;   // Case-insensitive search for "rose"
      const replaceprtn = "puppy";
      const newstr = str1.replace(pattern, replaceprtn);

      document.write("<h3> Given String is : ", str1 ,"</h3>");
      document.write("<h3> Pattern to search is : ", pattern,
                     " </h3>");
      document.write("<h3> String for replacement is : ", replaceprtn,
                     " </h3>");
      if (newstr != null) {
       document.write("<h3> After replacement new string  is : ",
                          newstr, "</h3>");
      } else {
      document.write("<h3> Pattern not found - no replacement. </h3>");
      }
  </script>
</body>
</html>
```

Figure 5.44: Program to demonstrate replace() pattern matching method in JavaScript

In this example, replace() is used to replace the pattern "rose" (case-insensitive) with "puppy" in the string str1.

The Output is :

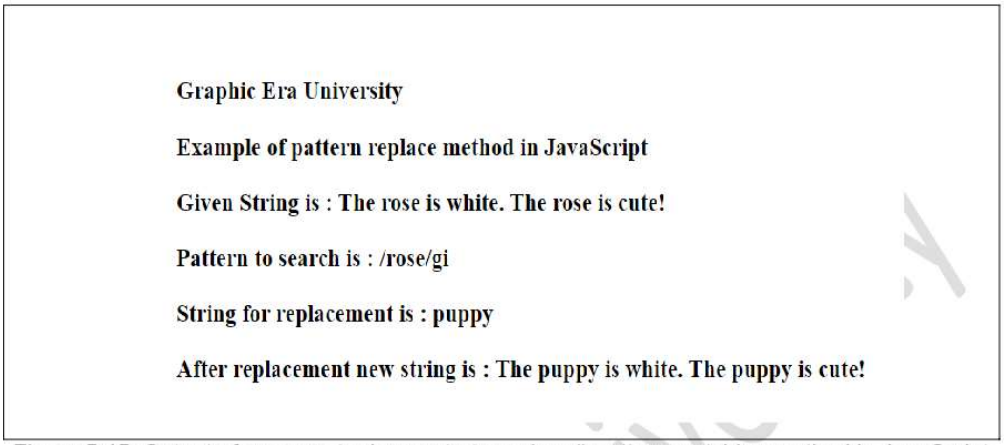

Figure 5.45: Output of program to demonstrate replace() pattern matching method in JavaScript

These methods are handy for various text-processing tasks, and they allow us to work with patterns and regular expressions in JavaScript to search for, extract, and replace data within strings.

5.11 Self-Assessment Questions

Q1. What is JavaScript? Discuss the key features of JavaScript. [6 marks, L2]

Q2. Explain the different ways of embedding JavaScript in XHTML documents with suitable examples. [8 marks, L2]

Q3. Discuss the declaration of variables and data types used in JavaScript. [6 marks, L2]

Q4. Explain the various screen output and Keyboard input methods used in JavaScript with examples for each. [8 marks, L1]

Q5. Explain different conditional statements used in JavaScript with suitable examples for each of them. [8 marks, L1]

Q6. Describe the use of various loops in JavaScript with suitable examples for each. [8 marks, L1]

Q7. How do you create arrays in JavaScript? Explain the various array methods in JavaScript with examples.[8 mark, L2]

Q8. Describe the various loops used to iterate through arrays in JavaScript with examples for each.[8 marks, L2]

Q9. Explain how arrays can be sorted in ascending and descending order by using the compare function in JavaScript with suitable examples. [8 marks, L2]

Q10. Discuss the use of functions with and without parameters with suitable examples. [8 marks, L2]

Q11. Explain pattern matching in JavaScript. Explain the various methods used for pattern matching in JavaScript with suitable examples.[8 marks, L2]

Q12. Write an external javascript function for reading 3 three numbers, using a prompt, and display the largest of them. Embed the script in an XHTML document and test the function for different inputs(numbers).

Q13. Write an external JavaScript function for validating a 12-digit Aadhar Number using pattern matching in JavaScript. Embed the script in an XHTML document and test the function for different inputs(numbers).

Q14. Create an XHTML document with the Javascript code that uses prompt() dialogs to ask a user for a name, enrolment number, and semester. Display all the information entered by the user.

Q15. Create an XHTML document with the Javascript code that uses prompt() dialog to ask a student for the semester he/she is studying. Validate the semester. For MCA the semester value should be between 1 to 4. Display appropriate error messages.

5.12 Self-Assessment Activities

A1. Write a JavaScript function called "isprime" that takes an integer as input and returns true if the number is prime (only divisible by 1 and itself), and false otherwise. Test the function with different inputs (numbers).

A2. Write a JavaScript function called "areaofcircle" that accepts the radius of a circle as a parameter and returns the area of the circle.

A3. Create a function named "reversestr" that takes a string as input and returns the reversed version of that string. For example, if the input is "welcome," the function should return "emoclew." Test the function with different input strings.

5.13 Multiple-Choice Questions

1. JavaScript is _____. [1 mark, L1]
   a) A weakly typed language (dynamic).
   b) A strongly typed language.
   c) A programming language used to build an operating system
   d) None of the above

2. ____ is not a valid data type in JavaScript. [1 mark, L1]
   a) String

b) Boolean

c) Float

d) Character

3. The _____ method is used to add an element to the end of an array in JavaScript. [1 mark, L1]

a) shift()

b) unshift()

c) push()

d) add()

4. The _____method is used to combine two or more arrays into a single array in JavaScript. [1 mark, L1]

a) unshift()

b) concat()

c) combine()

d) push()

5. The _____ method is used to convert an array of characters to a string in JavaScript. [1 mark, L1]

a) join()

b) concat()

c) push()

d) unshift()

6. Pattern matching in JavaScript is _____.[1 mark, L1]

a) A way to compare two strings

b) A method for sorting arrays

c) A technique to find and manipulate patterns in strings using regular expressions

d) A way to define variables

7. The ____modifier is used to perform a case-insensitive search in a regular expression in JavaScript.[1 mark, L1]

a) m

b) g

c) c

d) i

8. The ^ metacharacter in a regular expression _____. [1 mark, L1]
   a) Matches the preceding character 0 or more times
   b) Anchors the pattern to the start of a string
   c) Matches the end of a string
   d) Matches any word boundary

9. The purpose of the match() method in JavaScript regular expressions is ____. [1 mark, L1]
   a) To match a pattern and return an array of matched substrings
   b) To match a pattern and return a boolean value
   c) To match a pattern and return the index of the match
   d) To match a pattern and return an object

10. The scope of a variable declared inside a function is ____. [1 mark, L1]
    a) Global to the entire program
    b) Limited to the main program
    c) Local to the function
    d) Available to other functions

5.14 Key Answers to Multiple-Choice Questions

1. JavaScript is a weakly typed language (dynamic).[a]
2. Character is not a valid data type in JavaScript.[d]
3. push() method is used to add an element to the end of an array in JavaScript.[c]
4. The concat() method is used to combine two or more arrays into a single array in JavaScript.[b]
5. The join() method is used to convert an array of characters to a string in JavaScript.[a]
6. Pattern matching in JavaScript is a technique to find and manipulate patterns in strings using regular expressions.[c]
7. The i modifier is used to perform a case-insensitive search in a regular expression in JavaScript.[d]
8. The ^ metacharacter in a regular expression anchors the pattern to the start of a string.[b]
9. The purpose of the match() method in JavaScript regular expressions is to match a pattern and return an array of matched substrings.[a]
10. The scope of a variable declared inside a function is local to the function.[c]

5.15 Summary

JavaScript is a versatile, interpreted scripting language that is used in developing web applications. It is commonly used for creating interactive web pages, manipulating the Document Object Model(DOM), and handling events in web pages. The Key Features of JavaScript are: It is supported by all major browsers and provides a built-in execution environment, it is a weakly typed language (dynamic language), which means variables, elements, and objects can be declared, altered, and modified during runtime, it is case-sensitive, it is event-driven, meaning it can respond to user actions like clicks, keyboard input, and mouse movements, it is single-threaded, meaning it processes one task at a time, it is versatile and can be used for a wide range of applications, from creating web applications and client-side validation scripts to server-side applications using technologies like Node.js.

There are two ways to embed JavaScript code in XHTML documents - Directly and Indirectly. It can be directly embedded in an XHTML document by using <script> tags within the <head> or <body> section of a web page. External JavaScript files having the extension of ".js" can be indirectly embedded in an XHTML document by using the <script> tags and specifying the external file names in its "src" attribute.

JavaScript has several primitive data types like – String, Number, Boolean, Undefined, Null, and non-primitive data types like – Object and Arrays.

For Screen Output and Keyboard Input, the methods used in JavaScript are - `document.write()`, `console.log()`, `prompt()`, `alert()` and `confirm()`.

JavaScript provides control structures and loops that allow you to manage the flow of your code and execute specific tasks repeatedly.

JavaScript provides a wide range of built-in methods for working with strings. These methods allow to manipulate and extract information from strings. Some commonly used string methods are - `length`, `charAt(index)`, `indexOf(substring)`, `substring(startIndex, noofcharacters)`, `slice(startIndex, endIndex)`, `lastIndexOf(substring)`, `toUpperCase()`, and `toLowerCase()`, `trim()`, `concat(string2, string3, ...)` etc.

Arrays in JavaScript are ordered collections of data, which can include values of different data types. Arrays in JavaScript have dynamic length. The length/size of the array can be made to grow or shrink dynamically. Arrays can contain a mix of data types, including numbers, strings, objects, and other arrays. JavaScript provides many built-in methods for manipulating arrays. Some common array methods include push(), pop(), shift(), unshift(), join(), slice(), concat(), reverse(), sort() etc.

Functions in JavaScript are blocks of code that can be defined and called to perform specific tasks. They are a fundamental concept in JavaScript and enable code reuse and organization.

Pattern matching is a powerful concept in programming that allows matching and manipulation of data based on specific patterns or structures. In JavaScript, pattern matching is often implemented using techniques like regular expressions. JavaScript provides three methods for pattern matching and manipulation: search(), match(), and replace(). These methods are often used with regular expressions for more advanced pattern matching.

5.16 Keywords

- Weakly-typed
- Script
- Strings
- Arrays
- Functions
- Patterns
- Sorting
- Search
- Match
- Compare function
- Single-threaded

5.17 Recommended resources for further reading

a. Essential Reading

1. Sebesta, R. W. (2010). Programming the World Wide Web (6th ed.), Pearson education.
2. Subramanian, V. (2019). Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node (2nd Ed.), Apress.

b. Recommended Reading

1. DT Editorial Services. (2016). HTML 5: Covers CSS3, JavaScript, XML, XHTML, AJAX, PHP & jQuery: Black Book, Dreamtech Press.
2. Koroliova, E. W. I., (2018). MERN Quick Start Guide: Build Web applications with MongoDB, Express.js, React and Node, Packt.

--*--