Chapter 4

4. JavaScript

4.1. Introduction to JavaScript

JavaScript is *an object-based scripting language* which is lightweight and cross-platform. JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser. JavaScript is used to create interactive websites. It is mainly used for:

- ✓ Client-side validation,
- ✓ Dynamic drop-down menus,
- ✓ Displaying date and time,
- ✓ Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- ✓ Displaying clocks etc.

Example: <script> document.write("Hello World"); </script>

Where to put JavaScript code

JavaScript codes can be embedded to html code in three ways:

1. Between the head tag of html
2. Between the body tag of html
3. In .js file (external javaScript)

JavaScript code between the head tag

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

<html> <head>

<script type="text/javascript">

function msg(){ alert("Hello world"); } </script>

</head> <body> Welcome to JavaScript

<form><input type="button" value="click" onclick="**msg()**"/></form> </body> </html>

JavaScript code between the body tag

<html><head></head><body>

<script type="text/javascript"> alert("Hello World"); </script>

</body></html>

External JavaScript file

We can create external JavaScript file and embed it in many html page. It provides **code re usability** because single JavaScript file can be used in several html pages. An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript file that prints Hello World in a alert dialog box.

message.js

```
function msg(){ alert("Hello World");  }
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

<html> <head>

<script type="text/javascript" src="message.js"></script>

</head> <body> Welcome to JavaScript

<form><input type="button" value="click" onclick="**msg()**"/>

</form> </body> </html>

External JavaScript Advantages

Placing scripts in external files has some advantages:

- ✓ It separates HTML and code
- ✓ It makes HTML and JavaScript easier to read and maintain
- ✓ Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

JavaScript Output

JavaScript can "display" data in different ways:

- ✓ Writing into an HTML element, using innerHTML.
- ✓ Writing into the HTML output using document.write().
- ✓ Writing into an alert box, using window.alert().
- ✓ Writing into the browser console, using console.log().

Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method. The id attribute defines the HTML element. The innerHTML property defines the HTML content:

Example

```
<!DOCTYPE html><html><body>
<h1>My First Web Page</h1><p>My First Paragraph</p>
<p id="demo"></p><script>
document.getElementById("demo").innerHTML = 5 + 6;
</script></body></html>
```

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

For testing purposes, it is convenient to use document.write():

Example

```
<!DOCTYPE html><html><body>
<h1>My First Web Page</h1><p>My first paragraph.</p><script>
document.write(5 + 6); </script></body></html>
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**. The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

```
<!DOCTYPE html><html><body><h1>My First Web Page</h1>
<p>My first paragraph.</p><script>window.alert(5 + 6);</script></body></html>
```

Using console.log()

For debugging purposes, you can use the console.log() method to display data.

Example

```
<!DOCTYPE html><html><body>
<script>console.log(5 + 6);</script>
</body></html>
```

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with //. Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
```

Multi-line Comments

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript.

Example

```
/* The code below will change the heading with id = "myH"
and the paragraph with id = "myP" in my web page: */
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

4.2. Variables, Data Type, Operators

JavaScript Variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

- ✓ Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
- ✓ After first letter we can use digits (0 to 9), for example value1.
- ✓ JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
var x = 10;
var _value="sonoo";
```

Incorrect JavaScript variables

```
var  123=30;
```

var *aa=320;

Example:

<script>var x=10; var y=20; var z=x+y; document.write(z);</script>

Output of the above example: 30

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. Example:

<script> function abc(){ var x=10;//local variable } </script>

Or,

<script>If(10<13){ var y=20;//JavaScript local variable } </script>

JavaScript Global Variable

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

Example:

<script> var value=50;//global variable

function a(){ alert(value); } function b(){ alert(value); }

</script>

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. Now it can be declared inside any function and can be accessed from any function. For example:

```
function m(){
window.value=100;//declaring global variable by window object
}
function n(){
alert(window.value);//accessing global variable from other function
}
```

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

- ✓ Primitive data type
- ✓ Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands

JavaScript has the following operators:

- ✓ Arithmetic Operators
- ✓ Comparison (Relational) Operators
- ✓ Bitwise Operators
- ✓ Logical Operators
- ✓ Assignment Operators
- ✓ Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2

%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
In	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
New	creates an instance (object)
typeof	checks the type of object.
Void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

4.3. Control Structure, Arrays & Functions

Conditional Statements

Conditional statements are used to perform different actions based on different conditions. Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The if Statement

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
```

```
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

JavaScript Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The syntax of JavaScript switch statement is given below.

```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
......
  default:
 code to be executed if above values are not matched;
}
```

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript: for loop, while loop, do-while loop and for-in loop

For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

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

while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

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

do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

```
do{
   code to be executed
}while (condition);
```

for in loop

The **JavaScript for in loop** is used *to iterate the properties of an object*. The JavaScript for/in statement loops through the properties of an object:

Example

```
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
var x;
for (x in person) {  text += person[x];  }
```

The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement. The break statement can also be used to jump out of a loop.

The break statement breaks the loop and continues executing the code after the loop (if any):

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

The Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

```
var arrayname=[value1,value2.....valueN];
```

As you can see, values are contained inside [ ] and separated by , (comma).

Example:

```
<script>  var emp=["Abebe","Helen","Kibru"];
for (i=0;i<emp.length;i++){  document.write(emp[i] + "<br/>");  }
  </script>
```

The .length property returns the length of an array.

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
var arrayname=new Array();
```

Here, **new keyword** is used to create instance of array.Example:

```
<script>
var i;
var emp = new Array();
emp[0] = "Abebe";
emp[1] = "Helen";
emp[2] = "Kibru";
```

```
 for (i=0;i<emp.length;i++){  document.write(emp[i] + "<br>");  }
</script>
```

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly. Example:

```
<script>
var emp=new Array("Abebe","Helen","Kibru");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

JavaScript Array Methods

Let's see the list of JavaScript array methods with their description.

Methods	Description
concat()	It returns a new array object that contains two or more merged arrays.
copywithin()	It copies the part of the given array with its own elements and returns the modified array.
every()	It determines whether all the elements of an array are satisfying the provided function conditions.
fill()	It fills elements into an array with static values.
filter()	It returns the new array containing the elements that pass the provided function conditions.
find()	It returns the value of the first element in the given array that satisfies the specified condition.
findIndex()	It returns the index value of the first element in the given array that satisfies the specified condition.
forEach()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.
indexOf()	It searches the specified element in the given array and returns the index of the first match.

join()	It joins the elements of an array as a string.
lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
map()	It calls the specified function for every array element and returns the new array
pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
shift()	It removes and returns the first element of an array.
slice()	It returns a new array containing the copy of the part of the given array.
sort()	It returns the element of the given array in a sorted order.
splice()	It add/remove elements to/from the given array.
unshift()	It adds one or more elements in the beginning of the given array.

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

- **Code reusability**: We can call a function several times so it save coding.
- **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

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

JavaScript Functions can have 0 or more arguments.

Example

```
<script>
function msg(){  alert("hello! this is message");  }
</script>
```

```
<input type="button" onclick="msg()" value="call function"/>
```

JavaScript Function Arguments

We can call function by passing arguments. Example:

```
<script>
function getcube(number){  alert(number*number*number); }
</script>  <form>
<input type="button" value="click" onclick="getcube(4)"/> </form>
```

Function with Return Value

We can call function that returns a value and use it in our program. Let's see the example of function that returns value.

```
<script>
function getInfo(){  return "hello Students! How r u?";  }
</script>
<script> document.write(getInfo());  </script>
```

JavaScript Function Object

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax

```
new Function ([arg1[, arg2[, ....argn]],] functionBody)
```

Parameter

- **arg1, arg2, ... , argn** - It represents the argument used by function.
- **functionBody** - It represents the function definition.

JavaScript Function Methods

Method	Description
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
call()	It is used to call a function contains this value and an argument list.
toString()	It returns the result in a form of a string.

JavaScript Function Object Examples

Example 1:

```
<script>
var add=new Function("num1","num2","return num1+num2");
document.writeln(add(2,5));
</script>
```

Example 2:

```
<script>
var pow=new Function("num1","num2","return Math.pow(num1,num2)");
document.writeln(pow(2,3));
</script>
```

4.4. JavaScript Form Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must. The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

Form data that typically are checked by a JavaScript could be:

- has the user left required fields empty?
- has the user entered a valid e-mail address?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Example

In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long. Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;
  if (name==null || name==""){
  alert("Name can't be blank");
  return false;
}else if(password.length<6){
```

```
 alert("Password must be at least 6 characters long.");
 return false;
 }
}
</script>
<body>
<form name="myform" method="post" action="" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
```

JavaScript Retype Password Validation

```
<script type="text/javascript">
function matchpass(){
        var firstpassword=document.f1.password.value;
        var secondpassword=document.f1.password2.value;
        if(firstpassword==secondpassword){  return true;  }
        else{  alert("password must be same!");  return false;  }
}
</script>
<form name="f1" action="register.jsp" onsubmit="return matchpass()"> Password:<input type="
password" name="password" /><br/>
Re-enter Password:<input type="password" name="password2"/><br/>
<input type="submit">
</form>
```

JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
 document.getElementById("numl").innerHTML="Enter Numeric value only";    return false;
}else{    return true;    }
```

```
}
</script>
<form name="myform" onsubmit="return validate()" >
Number: <input type="text" name="num"><span id="numl"></span><br/>
<input type="submit" value="submit">
</form>
```

JavaScript validation with image

Let's see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```
<script>
function validate(){
var name=document.f1.name.value;
var password=document.f1.password.value;
var status=false;
if(name.length<1){
document.getElementById("namel").innerHTML="<img src='unchecked.gif'/> Please enter your
 name";
status=false;
}else{
document.getElementById("namel").innerHTML="<img src='checked.gif'/>";  status=true;
}
if(password.length<6){
document.getElementById("pwdl").innerHTML=
" <img src='unchecked.gif'/> Password must be at least 6 char long";
status=false;
}else{
document.getElementById("pwdl").innerHTML="<img src='checked.gif'/>";  }
return status;
}
</script>
  <form name="f1" action="#" onsubmit="return validate()">
<table><tr><td>Enter Name:</td><td><input type="text" name="name"/>
<span id="namel"></span></td></tr>
```

```
<tr><td>Enter Password:</td><td><input type="password" name="password"/>
<span id="pwdl"></span></td></tr>
<tr><td colspan="2"><input type="submit" value="register"/></td></tr>  </table>
</form>
```

JavaScript email validation

We can validate the email by the help of JavaScript. There are many criteria that need to be follow to validate the email id such as:

- ✓ email id must contain the @ and . character
- ✓ There must be at least one character before and after the @.
- ✓ There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 ||dotposition<atposition+2 ||dotposition+2>=x.length){
  alert("Please enter a valid e-mail address \n atpostion:
"+atposition+"\n dotposition:"+dotposition);  return false;
  }
}
</script>
<body>
<form name="myform"  method="post" action="#" onsubmit="return validateemail();">
Email: <input type="text" name="email"><br/>    <input type="submit" value="register">
</form>
```

4.5. JavaScript Events

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

```
<html> <head> <script type = "text/javascript">
  function sayHello() {  alert("Hello World");    }
</script></head> <body>
   <p>Click the following button and see result</p>
<form><input type="button" onclick="sayHello()" value="Say Hello"/>  </form></body>
</html>
```

onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example: The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function validation() {
          all validation goes here
          .........
          return either true or false
```

```
        }
      //-->
    </script>
  </head>

  <body>
    <form method = "POST" action = "t.cgi" onsubmit = "return validate()">
      .....
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<html>  <head> <script type = "text/javascript">
      <!--
        function over() {  document.write ("Mouse Over");    }
        function out() {  document.write ("Mouse Out");      }
      //-->
    </script> </head><body>
    <p>Bring your mouse inside the division to see the result:</p>
    <div onmouseover = "over()" onmouseout = "out()">
      <h2> This is inside the division </h2>
    </div> </body> </html>
```

4.6. String & Regular Expressions

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

1) By string literal

The string literal is created using double quotes. The syntax of creating string using string literal is given as: <script> var str="This is string"; document.write(str); </script>

2) By string object (using new keyword)

The syntax of creating string object using new keyword is given below:

<script> var stringname=new String("hello javascript string");

document.write(stringname); </script>

JavaScript String Methods

Methods	Description
charAt()	It provides the char value present at the specified index.
charCodeAt()	It provides the Unicode value of a character present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching a character from the last position.
search()	It searches a specified regular expression in a given string and returns its position if a match occurs.
match()	It searches a specified regular expression in a given string and returns that regular expression if a match occurs.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.
substring()	It is used to fetch the part of the given string on the basis of the specified index.
slice()	It is used to fetch the part of the given string. It allows us to assign positive as well negative index.
toLowerCase()	It converts the given string into lowercase letter.

toLocaleLowerCase()	It converts the given string into lowercase letter on the basis of host?s current locale.
toUpperCase()	It converts the given string into uppercase letter.
toLocaleUpperCase()	It converts the given string into uppercase letter on the basis of host?s current locale.
toString()	It provides a string representing the particular object.
valueOf()	It provides the primitive value of string object.

1) JavaScript String charAt(index) Method

The JavaScript String charAt() method returns the character at the given index.

<script>var str="javascript"; document.write(str.charAt(2)); </script>

2) JavaScript String concat(str) Method

The JavaScript String concat(str) method concatenates or joins two strings.

<script> var s1="javascript "; var s2="concat example";

var s3=s1.concat(s2); document.write(s3); </script>

3) JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

<script> var s1="javascript from javatpoint indexof";

var n=s1.indexOf("from"); document.write(n); </script>

4) JavaScript String lastIndexOf(str) Method

The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

<script> var s1="javascript from javatpoint indexof";

var n=s1.lastIndexOf("java"); document.write(n); </script>

5) JavaScript String toLowerCase() Method

The JavaScript String toLowerCase() method returns the given string in lowercase letters.

<script> var s1="JavaScript toLowerCase Example";

var s2=s1.toLowerCase(); document.write(s2); </script>

6) JavaScript String toUpperCase() Method

The JavaScript String toUpperCase() method returns the given string in uppercase letters.

<script> var s1="JavaScript toUpperCase Example";

var s2=s1.toUpperCase(); document.write(s2); </script>

7) JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

```
<script>  var s1="abcdefgh";  var s2=s1.slice(2,5);
document.write(s2);  </script>
```

8) JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

```
<script>  var s1="     javascript trim    ";  var s2=s1.trim();   document.write(s2);  </script>
```

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a **search pattern**. When you search for data in a text, you can use this search pattern to describe what you are searching for. A regular expression can be a single character, or a more complicated pattern. Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

/***pattern***/*modifiers*;

Example

```
var patt = /w3schools/i;
```

Example explained:

/w3schools/i is a regular expression.

w3schools is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace(). The search() method uses an expression to search for a match, and returns the position of the match. The replace() method returns a modified string where the pattern is replaced.

Using String search() With a String

The search() method searches a string for a specified value and returns the position of the match:

Example

Use a string to do a search for "W3schools" in a string:

```
var str = "Visit W3Schools!";
var n = str.search("W3Schools");
```

Using String search() With a Regular Expression

Example

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
var str = "Visit W3Schools";
var n = str.search(/w3schools/i);
```

The result in *n* will be: 6

Using String replace() With a String

The replace() method replaces a specified value with another value in a string:

```
var str = "Visit Microsoft!";
var res = str.replace("Microsoft", "W3Schools");
```

Use String replace() With a Regular Expression

Example

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

```
var str = "Visit Microsoft!";
var res = str.replace(/microsoft/i, "W3Schools");
```

The result in *res* will be: Visit W3Schools!

Note: Regular expression arguments (instead of string arguments) can be used in the methods above.

Regular expressions can make your search much more powerful (case insensitive for example).