UNIT II

Introducing DHTML

DHTML Introduction

- DHTML explains different technologies that are required to make web pages dynamic and interactive.
- It is the combination of HTML, CSS, DOM and JavaScript that helps to create animate on the document.
- JavaScript is a scripting language that is used by DHTML to control, access and manipulate HTML elements.
- DOM defines a standard set of objects for HTML and standard way to access and manipulate them.
- CSS allows web developers to control the style and layout of web pages.

Continue..

- HTML- To define the **content** of web pages.
- CSS- To specify the **layout** of web pages
- JavaScript- To program the **behavior** of web pages.
- Dynamic HTML(DHTML):

  DHTML= HTML+ CSS+ JAVA SCRIPT

- We can enhance the web page by adding more dynamism and interactivity.
- JavaScript is an **interpreted, client side and object oriented scripting language**.

Continue..

- We can embed java script into an html document by using

<script>.....</script>

- **<script>** tags can be placed in either **body or in head**.
- When you want the java script to run while the web page is **loading**, place the <script> tag in **body** part.
- When you want the script to run only when the **user performs an action**, such as clicking a link, place the <script> tag in **head** part.

Client Side Scripting

- Java Script is a client side scripting.
- The web browser executes the client side scripting that resides at the users computer.
- The browser receives the page sent by the server and executes client side scripting.

A Script is a small program which is used with HTML to make web pages more attractive, dynamic and interactive, such as an alert popup window on mouse click.
Currently, the most popular scripting language is JavaScript used for websites.

We mainly use JavaScript to create

- websites
- web applications
- client-side applications using Node.js

but JavaScript is not limited to these things, and it can also be used to

- create mobile applications using tools like React Native
- create programs for microcontrollers and the internet of things
- create smartwatch applications

We can include java script code in html document using <script> …. </script> tag

Limitations of JavaScript:

We cannot treat JavaScript as a full-fledged programming language.
It lacks the following important features:
Client-side JavaScript does not allow the reading or writing of files.
This has been kept for security reason.
JavaScript cannot be used for networking applications because there is no such support available.
JavaScript doesn't have any multithreading or multiprocessor capabilities.

JavaScript is a programming language:

- **high level**: it provides abstractions that allow you to ignore the details of the machine where it's running on. It manages memory automatically with a garbage collector, so you can focus on the code instead of managing memory like other languages like C would need, and provides many constructs which allow you to deal with highly powerful variables and objects.
- **dynamic**: opposed to static programming languages, a dynamic language executes at runtime many of the things that a static language does at compile time. This has pros and cons, and it gives us powerful features like dynamic typing, late binding, reflection, functional programming, object runtime alteration, closures and much more. Don't worry if those things are unknown to you - you'll know all of them by the end of the course.
- **dynamically typed**: a variable does not enforce a type. You can reassign any type to a variable, for example, assigning an integer to a variable that holds a string.
- **loosely typed**: as opposed to strong typing, loosely (or weakly) typed languages do not enforce the type of an object, allowing more flexibility but denying us type safety and type checking (something that TypeScript - which builds on top of JavaScript - provides)
- **interpreted**: it's commonly known as an interpreted language, which means that it does not need a compilation stage before a program can run, as opposed to C, Java or Go for example. In practice, browsers do compile JavaScript before executing it, for performance reasons, but this is transparent to you - there is no additional step involved.
- **multi-paradigm**: the language does not enforce any particular programming paradigm, unlike Java for example, which forces the use of object-oriented programming, or C that forces imperative programming. You can write JavaScript using an object-oriented paradigm, using prototypes and the new (as of ES6) classes syntax. You can write JavaScript in a functional programming style, with its first-class functions, or even in an imperative style (C-like

Differences between Client Side Scripting and Server Side Scripting

Client Side Scripting	Server Side Scripting
1.It executes in the client side i.e., in browser	1.It executes in the server side.
2.It cant access the database.	2.It can access the database.
3.It cant access the file system in server.	3. It can access the file systems in server.
4.It is faster than server side scripting.	4.It is slower compared with client side scripting.
5.Eg:JavaScript, Jscript, VB Script	5.Eg: Servlets, JSP, PHP, .NET

Client-side benefits of JavaScript over VBScript

- JS can be embedded in the shockware Flash and Adobe PDF documents.
- VBScript is used for network administrators in the Windows environment.
- Benefits – JS is very fast bcz. Any function can run immediately without waiting for an answer from the server.
- Simple to learn and implement
- JS reduced the demand on the server as it is a client-side scripting language.

JavaScript Introduction

- A script is a program code that is written using a scripting language, which is a kind of programming language with less functionality.

Ex: JavaScript, VBScript, ASP and PHP.

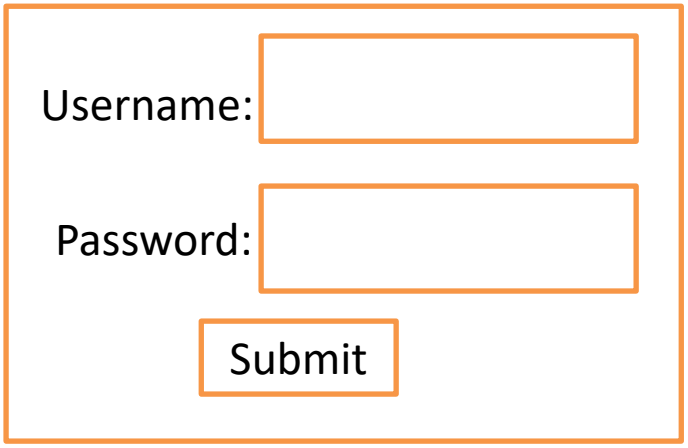
- JavaScript is most popular scripting language used to infuse dynamism and interactivity in web pages. Also called LiveScript.
- JS is interpreted, client-side and object-based scripting lang. that has various functionalities to enliven the static HTML web pages.

Uses of JavaScript

1. Validation- The main purpose of JavaScript is to validate web pages.
2. To reduce burden on the server.
3. To reduce network traffic
4. To minimize time.

Without JavaScript:

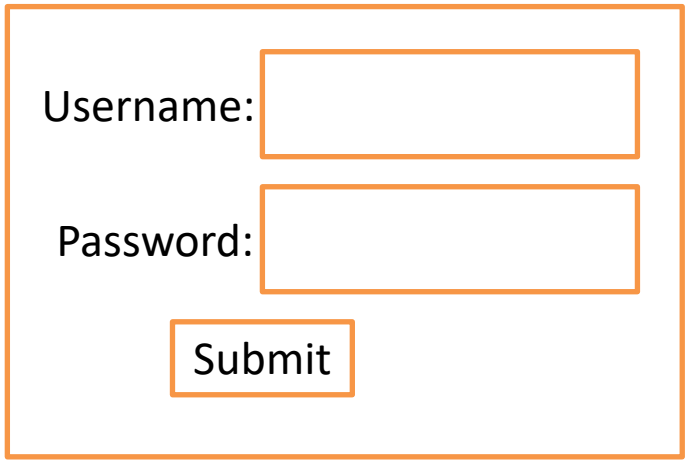
Gmail:



Here, unnecessarily request goes to web server resulting in network traffic and burden on server.

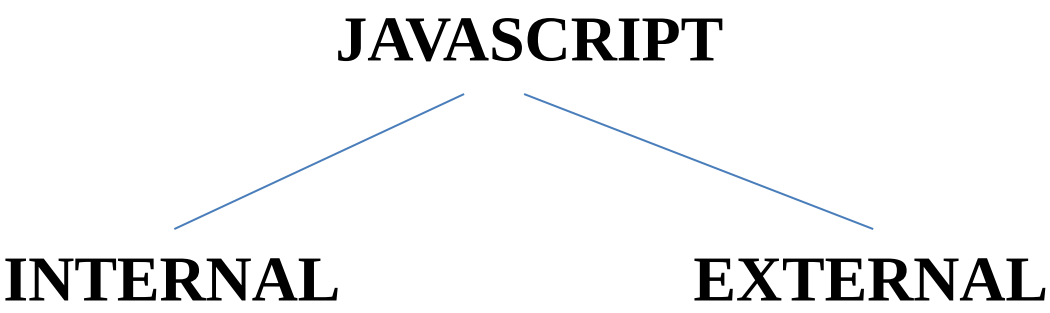
With JavaScript:

Gmail:



Client Side Scripting validates webpage in the browser itself. If everything is valid, then it sends request to server there by reducing network traffic and burden on server.

Use of JavaScript as



Embedding JS in an HTML page

- <script> </script> tag is used to embed JS in an HTML document and it treated as script code.
- Syntax:

<script type = “text/css” language=“JavaScript”>

Stmts.

</script>

- type attribute refers to the MIME type of the script. Mandatory attribute according go W3C.
- language attribute refers to the scripting lang. used to create scripts.
- src attribute refers to the URL of another file that has a script.

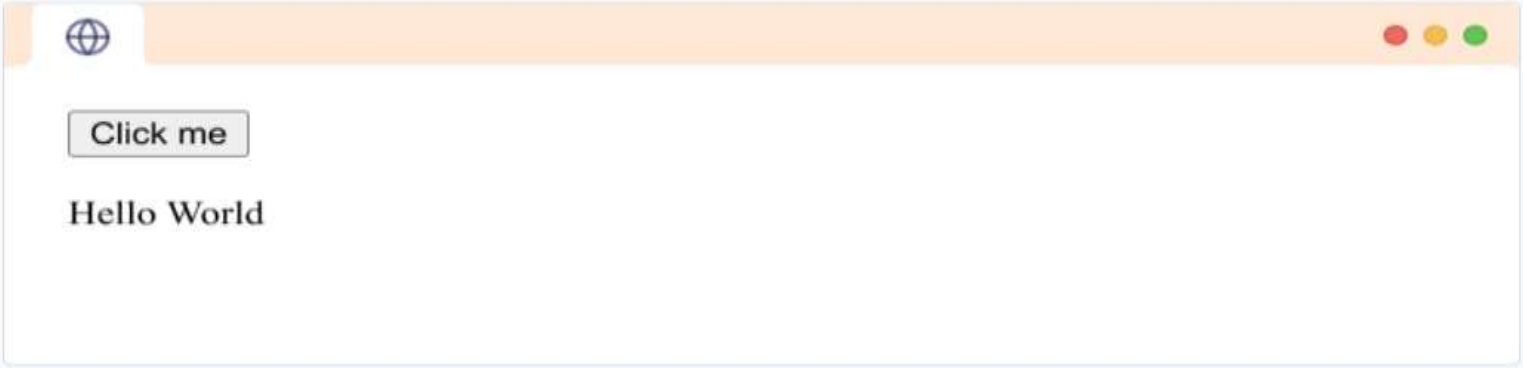
Internal JavaScript

```
<html >
<head>
<title> Internal Java Script </title>
 </head>
 <body> Running JS:<br/>
<script> alert("Hello World - Internal JS!");
   </script>
 </body>
</html>
```

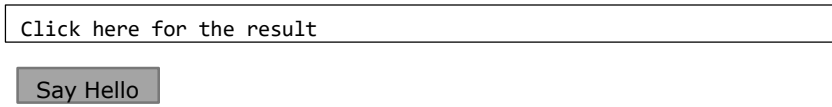
Adding an Internal Script

```
<head>
<title>
</title>
<script>
function showHiddenText()
{
 document.write("Hello World"; } </script>
</head>
<body>
<button onclick="showHiddenText()">Click me</button>
<p id="demo"></p>
</body>
```

Browser output(after click)



```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
  alert("Hello World")
}
//-->
</script>
</head>
<body>
Click here for the result
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```



Example Program

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<center>
<h1>JavaScript</h1>
<script type="text/javascript" language="javascript">
document.write("Creating a Script in an HTML Document");
</script>
</body>
</html>
```

Note: There is no limit to number of scripts that we can add in an HTML document.

External JavaScript

- Similar to External CSS.

cosmiclearn.js file:

```
alert("Hello World!");
```

```
<html>
<head>
 <title>Title</title> </head>
<body> Running JS:<br/>
 <script src="cosmiclearn.js"></script>
</body> </html>
```

You can put your JavaScript code in <head> and <body> section altogether as follows.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {  alert("Hello World")
} //-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Adding an External Script

We can also use an external javascript file in our HTML document.

To add an external script, we provide the location of the JS file to the src attribute of a <script> tag.

```
<script src="scripts/code.js"></script>
```

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

filename.js

```
function
  sayHello()
  {
  alert("Hell
  o World")
}
```

Variables

- Accepts any kind of data (int / float/ string/char).
- Default value is **Undefined**.

Syntax to declare a variable:

```
var   variable_name;
```

```
<html>
<body>
    <script>
        var a;
        document.write(a);
    </script>
</body>
</html>
```

Using Variables in JS

Example program:

```
<html>
<head>
<title>JavaScript variable Example</title>
</head>
<body bgcolor="pink">
<center>
<h1>JavaScript</h1>
<script type="text/javascript" language="javascript">
var countBooks = 50
document.write("Count books = “ + countBooks);
</script>
</body>
</html>
```

Example:

```
<html>
<head></head>
<body>
    <script>
        var a=10;
        var b=20.5;
        var c="hai";
        var d='a';
        a="hello";
        document.write(a+"<br>");
        document.write(b);
        document.write(c);
        document.write(d);
    </script>
</body>
</html>
```

JavaScript Operators

- Arithmetic (+,-,*,/,%,++, --)
- Assignment (=, +=,-=,*=,/=,%=, <<=,>>=,>>>=, &=,^=,!=)
- String (+)
- Comparison (==, ===,!==,>,<,>=,<=,?)
- Logical (&&,||,!)
- Type (typeof, instanceof)
- Bitwise (&, |,~,^,<<,>>,>>>)

JavaScript Operators Example

```
<html>
<body>
<script type = "text/javascript">
<!–
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";
document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);
document.write("a - b = ");
result = a - b; document.write(result); document.write(linebreak);
```

Continue..

```
document.write("a / b = ");
result = a / b;
document.write(result);
document.write(linebreak);
document.write("a % b = ");
result = a % b;
document.write(result); document.write(linebreak);
document.write("a + b + c = ");
result = a + b + c;
document.write(result); document.write(linebreak);
```

Continue..

```
a = ++a;

document.write("++a = ");

result = ++a; document.write(result);
document.write(linebreak);

b = --b;

document.write("--b = ");

result = --b; document.write(result);
document.write(linebreak); //-->

</script> /body>

</html>
```

Control statements in JS

- 3 type of control stmts.

a) Selection

b) Loops

c) Jump

- Selection – if, if-else, if-else-if, switch
- Loop – while, do while, for
- Jump – break, continue

IF - ELSE EXAMPLE

```
<html>
<body>
    <script>
        var a=10;
        if(a%2==0)
        {
        document.write(a+"is even number");
        }
        else
        {
        document.write(a+"is odd number");
        }
    </script>
</body>
</html>
```

Switch Case Example

```
<html>
<body>
    <script>
        var num=parseInt(prompt("Enter number"));
        var div=parseInt(prompt("Enter number"));
        var rem=num%div;
        switch(rem)
        {
        case 0:document.write("zero");break;
        case 1:document.write("one");break;
        case 2:document.write("two");break;
        case 3:document.write("three");break;
        case 4:document.write("four");break;
        case 5:document.write("five");break;
```

Continue…

```
case 6:document.write("six");break;
case 7:document.write("seven");break;
case 8:document.write("eight");break;
case 9:document.write("nine");break;
default:document.write("wrong");break;
        }
    </script>
</body>
</html>
```

Prompt() Dialogbox

- prompt()- It takes the input from keyboard.

  - It accepts strings

Syntax: prompt(message, default)

- **message:** It is an optional parameter. It is the text displays to the user. We can omit this value if we don't require to show anything in the prompt.
- **default:** It is also an optional parameter. It is a string that contains the default value displayed in the textbox.

parseInt() method

parseInt()- The **parseInt()** function parses a string argument and returns an integer of the specified radix

- Syntax: parseInt(*string, radix*)

String: The value to parse. If this argument is not a string, then it is converted to one using the ToString abstract operation. Leading whitespace in this argument is ignored.

radix (Optional) An integer between 2 and 36 that represents the *radix* (the base in mathematical numeral systems) of the *string*. Be careful—this does ***not*** default to 10! If the radix value is not of the Number type it will be coerced to a Number

Working With Loops

- Allows to execute a particular group of statements repeatedly.
- The number of times the group of statements is executed depends on a particular condition.

1. Using While Loop
2. Using Do-While Loop
3. Using For Loop

Using While Loop

- The group of statements that is to be executed is specified after condition. The group of statements keeps on executing until the condition is false.

```
<html>
<body>
    <script>
        var a=1;
        while(a<=10)
        {
        document.write(a);
        a++;
        }
    </script>
</body>
</html>
```

Using Do-While Loop

- The group of statements are executed at least once.

```
<script>
    var a=1;
    do{
        document.write(a);
        a++;
    } while(a<=10);
</script>
```

For Loop

- The for loop allows to execute block of statements for a pre determined number of times. The condition of for loop is placed at the beginning of the loop.

```
<script>
    var i;
    for(i=0;i<=10;i++)
    document.write(i);
</script>
```

Functions

- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas:
  (*parameter1, parameter2, ...*)
- The code to be executed, by the function, is placed inside curly brackets: **{}**
- function *name*(*parameter1, parameter2, parameter3*)
  {
  // code to be executed
  }

Functions - Example

```
<html>
<body>
    <script>
    function func(a,b,c)
    {
    var sum;
    sum=a+b+c;
    document.write(sum);
    }
    document.write("The function is to be called here");
    func(10,30,40);
    </script>
</body>
</html>
```

Returning Value From Function

```
<html>
   <body>
   <script>
   function func(a,b,c)
   {    var result;
   result=a*b/c;                return result;
   }
   document.write("The function is to be called here");
   var a=func(10,30,40);
   document.write(a);
   </script>
   </body>              </html>
```

Find factorial using recursion

```
<html>
<body>
    <script>
    function fact(a)
    {     if(a==0 || a==1)
                return 1;
          else
                return (a*fact(a-1));
    }
    document.write("The factorial is");
    var a=fact(5);       document.write(a);
    </script>    </body>    </html>
```

Using Break

```
<html>
<body>
    <script>
    function func()
    {
    var i;
    for(i=0;i<=10;i++)
    {
    if(i==5)
            break;
    document.write(i+"<br>");
    }                }   document.write("the function is called here");
    func();
    </script>                </body>          </html>
```

Using Continue

```
<html>
<body>          <script>
        function func()
        {
        var i;
        for(i=0;i<=10;i++)
        {
        if(i==5)
                continue;
        document.write(i+"<br>");
        }                   }
        document.write(“The output is:");
        func();
   </script>              </body>                  </html>
```

Event Handling In JavaScript

- Events are actions that happen when a user interacts with the page - like clicking an element, typing in a field, or loading a page.
- The browser notifies the system that something has happened, and that it needs to be handled. It gets handled by registering a function, called an event handler, that listens for a particular type of event.
- Syntax: <*element event*="***some JavaScript***">
- Example:
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>

Continue…

In a simple way--

- Events refers to action performed on web page by user.
- Event Handler is a function that handles particular event.
- Here are some examples of events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked

1. onload event

It occurs while loading the page

```
<html>
<head>
   <script type="text/javascript" language="javascript">
        function myfunc()
        {
        alert("Welcome");
        }
   </script>
</head>
<body onload="myfunc()">
</body>
</html>
```

2. onclick Event

It Execute a JavaScript when a button is clicked

```
<html>
<head>    </head>
<body bgcolor="silver">
    <form>
        <input type="text" id="field"><br><br>
        <button
        onclick="alert('Hi'+document.getElementById('field').
        value)">  Click  </button>
    </form>
</body>
</html>
```

3. onmouseover Event

It Execute a JavaScript when moving the mouse pointer onto an image

```
<html>
<head>
    <script type="text/javascript" language="javascript">
        function mouovr()
        {
        alert("Welcome to WT");
        }
    </script>
</head>
<body>
    <h1 onmouseover="mouovr()">HAI</h1>
</body>
</html>
```

4. onmouseout Event

It Execute a JavaScript when moving the mouse pointer out of an image

```
<html>
<head>
    <script type="text/javascript" language="javascript">
        function mouout()
        {
        alert("Welcome to WT");
        }
    </script>
</head>
<body>
    <h1 onmouseout="mouout()">HAI</h1>
</body>
</html>
```

5. onreset Event

Execute a JavaScript when a form is reset

```
<html>
<head>
    <script type="text/javascript" language="javascript">
        function rese()
        {
        alert("Are you sure to reset the data");
        }
    </script>     </head>
<body>
    <form onreset="rese()">
        <input type="text" value="Enter the name"><br>
        <input type="reset" value="reset">
    </form>                </body>                    </html>
```

6. onsubmit Event

Execute a JavaScript when a form is submitted

```
<html>
<head>
    <script type="text/javascript" language="javascript">
        function subm()
        {   alert("Are you sure to submit the data");       }
    </script>           </head>
<body>
    <form onsubmit="subm()">
        <input type="text" value="Enter the name"><br>
        <input type="submit" value="submit">
    </form>
</body>                 </html>
```

7. onfocus Event

Execute a JavaScript when an input field gets focus

```
<html>
<body>
    Enter your name:<input type="text"
    onfocus="onfoc(this)">
    <p>When iput field get focused, fucntion is
    executed</p>
    <script>
        function onfoc(x)
        {
        x.style.background="yellow";
        }
    </script>
</body>
</html>
```

8. onblur Event

Execute a JavaScript when a user leaves an input field

```
<html>
<body>
    <p>When you enter the input field, a function is triggeredz</p>
    Enter your name:<input type="text" id=“myInput”onfocus="onfoc()"
    onblur="onblu()">
    <script>
function onfoc()
{ document.getElementById(“myInput”).style.background="yellow"; }
function onblu()
{   document.getElementById(“myInput”).style.background=“red";   }
</script>
</body>
</html>
```

Event hadlers and event listeners

There are two ways to handle events in Java script

1. Event handler
2. Even listener

With event handler mechanism we can add one handler function to an event, but using event lister, we can add more than one handler function to an event.

Example:

```
element.addEventListener("click", function1);
 element.addEventListener("click", function2);
```

This is impossible with event handler properties because any subsequent attempts to set the property will overwrite earlier ones:

Handler (use on before event)

```
<!DOCTYPE html>
<html>
<head>
<title>Document</title>
</head>
<body>
<button onclick="display()"> Click here</button>
<h1 id="text"></h1>
<script>
function display() {
document.getElementById("text").innerText = "Hello";
};
</script>
</body>
</html>
```

Listener

```
<!DOCTYPE html>
<html>
<head>
        <title>Document</title>
</head>

<body>
    <button id="try">Click here1</button>
    <h1 id="text1"></h1>
        <script>
        document.getElementById("try").addEventListener("click", function () {
            document.getElementById("text1").innerText = “Hello";
        });

    </script>
</body>

</html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Document</title>
</head>
<body>
  <button id="try">Click here1</button> // single event, two actions
<h1 id="text1"></h1>
<h2 id="text2"></h1>
<script>
document.getElementById("try").addEventListener("click", function () {
document.getElementById("text1").innerText = "Hello";
document.getElementById("text2").innerText = "Hello";
});
</script>
</body>
</html>
```

```
<button onclick="alert('Hello, this is my old-fashioned event handler!');">
 Press me </button>
```

it is not a good idea to mix up your HTML and your JavaScript, as it becomes hard to read. Keeping your JavaScript separate is a good practice, and if it is in a separate file you can apply it to multiple HTML documents.

Arrays In JavaScript

- An array is a collection of elements, there are two ways to declare an array:

1.var array_name= new Array(size);

Eg: var a=new array(3);

2.var arr_name=new Array(initialization);

Eg:var b=new array(“hai”,”welcome”,”to”,”WT”).

Property

- **Length**: To know the length of the array.

Methods

- **concat()**: To concatenate the array elements
- **reverse()**:To reverse the elements.
- **sort()**: To sort the elements.
- **join()**: To join the elements.
- Join entity is treated as single entity whereas concat output is treated as multiple entities.

```
<html>
<body>
    <script>
        var i;
        var a=new Array(5);
        var b=new Array("hai",“welcome",”to”,“WT");
        a[0]=1;
        a[1]=2;
        a[2]=“hai";
        var n=a.length;
        for(i=0;i<n;i++)
            document.write(“Array is”+a[i]+"<br>");
        document.write(“Array length:"+a.length+"<br>");
        document.write(“Array concat:"+b.concat()+"<br>");
        document.write(“Array reverse:"+b.reverse()+"<br>");
        document.write(“Array sort:"+b.sort()+"<br>");
        document.write(“Array join:"+a.join());
    </script>
</body>
</html>
```

Objects In JavaScript

- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

Creating a JavaScript Object

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword `new`.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using `Object.create()`.

1. object literal

An object literal is a list of
name:value
pairs (like age:50) inside curly braces {}.

Example:

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

2. Using the JavaScript Keyword new

```
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

```
function Employee()
{
this.name = names[Math.floor(Math.random() *
names.length)];
this.age=1; this.level=1;
this.production=400;
 this.totalprod=0;
}

    var employee1 = new Employee();
```

Objects are mutable: They are addressed by reference, not by value.
If person is an object, the following statement will not create a copy of person:

```
const x = person;  // Will not create a copy of person.
```

The object x is **not a copy** of person. It **is** person.
Both x and person are the same object.
Any changes to x will also change person, because x and person are the same object.

```
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}

const x = person;
x.age = 10;       // Will change both x.age and
person.age
```

Clone object

```
const originalObject =
{
name: 'John Doe’,
age: 30,
address: { city: 'New York', country:
'USA', },
};
const clonedObject = Object.assign({},
originalObject);
console.log(clonedObject);
```

Define an object constructor

```
function User()
{
this.name = 'Bob’;
}
 var user1 = new User();
var user1 = new User();

function User (name, age)
 {
this.name = name;
this.age = age;
}
var user1 = new User('Bob', 25);
var user2 = new User('Alice', 27);
```

Constructor vs Object Literal

An object literal is typically used to create a single object whereas a constructor is useful for creating multiple objects:

```
//Object literal
let user = { name: 'Bob' }
```

```
//Constructor function
User() {
 this.name = 'Bob';
}
var user1 = new User();
 var user2 = new User();
```

Object Prototype

Properties and methods can be added to a constructor using a prototype:

```
//Constructor
function User()
{ this.name = 'Bob'; }
let user1 = new User();
 let user2 = new User();
//Adding property to constructor using prototype
User.prototype.age = 25;
console.log(user1.age); // 25
console.log(user2.age); // 25
```

Built-in Constructors

JavaScript has some built-in constructors, including the following:

```
var a = new Object();
var b = new String();
 var c = new String('Bob’)
var d = new Number();
var e = new Number(25);
var f = new Boolean();
var g = new Boolean(true);
```

- Create an object using `Object.create()`.

The **Object.create()** static method creates a new object, using an existing object as the prototype of the newly created object.

Object.create(proto)
Object.create(proto, propertiesObject)

Inheritance concept implemented using this

```js
const person = {
  isHuman: false,
  printIntroduction: function () {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  },
};

const me = Object.create(person);

me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
me.isHuman = true; // Inherited properties can be overwritten

me.printIntroduction();
// Expected output: "My name is Matthew. Am I human? true"
```

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

Write a JavaScript program to list the properties of a JavaScript object.
Sample object:
var student = {
name : "David Rayy",
sclass : "VI",
rollno : 12 };
Sample Output: name,sclass,rollno

```
let student =
 {
name : "David Rayy",  sclass : "VI",  rollno : 12,
};
console.log(Object.keys(student));
```

```
var student = {
        name : "David Rayy",
        sclass : "VI",
        rollno : 12 };
var arr = [];
for( item in student)
{
arr.push(item);
}
console.log(arr)
```

Write a JavaScript program to delete the rollno property from the following object. Also print the object before or after deleting the property.
Sample object:

```
var student = {
name : "David Rayy",
sclass : "VI",
rollno : 12 };
```

```
var student =  {
name : "David Rayy",   sclass : "VI",   rollno : 12  };
console.log(student);
delete student.rollno;
console.log(student);
```

Write a JavaScript program to get the length of a JavaScript object.
Sample object:
var student = {
name : "David Rayy",
sclass : "VI",
rollno : 12 };

```
var student = {
name: "David Rayy",
sclass: "VI",
rollno: 12};
console.log(Object.keys(student).length);
```

```
// nested object
const student = {
    name: 'John',
    age: 20,
    marks: {
        science: 70,
        math: 75
    }
}

// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}

// accessing property of marks object
console.log(student.marks.science); // 70
```

```
<!DOCTYPE html>
\<html> <body>
<div id ="demo"></div>
<script>
const person =
{
name: "John",
age: 30,
city: "New York"
};
var text = person.name + "," + person.age + "," +
person.city;
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

You can Display JavaScript object in HTML using **innerHTML** and getElementById,

```
<<!DOCTYPE html>
\<html> <body>
<div id ="demo"></div>
<script>
class person
{
constructor(name1,age1,city1)
{
this.name-name1;
this.age=age1;
this.city =city1;
}
show() {
  return this.name + "," + this.age + "," + this.city ;

}
}
let p = new person("aaa",23,"aa");

document.getElementById("demo").innerHTML = p.show();
</script>
</body>
</html>
```

```
class person
{
constructor(name1,age1,city1)
{
this.name=name1;
this.age=age1;
this.city =city1;
}
show() {
  return this.name + "," + this.age + "," + this.city

}
}
let p1 = new person('a',23,"aa");
let p2 = new person("a",23,"aa");
let myString1= JSON.stringify(p1);
let myString2= JSON.stringify(p2);

let text=" ";
text+=myString1;
text+=myString2;

console.log(text);
```

JSON.stringify(): object to string
JSON.parse(): String to object

```
{"name":"a","age":23,"city":"aa"}{"name":"a","age":23,"city":"aa"}
```

DATE OBJECT

```
<html>
<head>
    <script>
        var dt=new Date();
        var year=dt.getFullYear();
        var date=dt.getDate();
        var day=dt.getDay();
        var month=dt.getMonth();
        var days=new
        Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday");
        var months=new
        Array("January","February","March","April","May","June","July","August","Septemb
        er","October","November","December");
    </script>
</head>
<body>
    <script>
        document.write("Today is"+days[day]+" "+date+" "+months[month]+" "+year);
        document.write("<br>");
        document.write(dt.toString());
    </script>
</body>
</html>
```

Today isThursday 21 December 2023
Thu Dec 21 2023 08:27:47 GMT+0000 (Coordinated Universal Time)

Strings

In JavaScript, strings are immutable. That means the characters of a string cannot be changed.

```
let a = 'hello';
a[0] = 'H';
console.log(a); // "hello"
```

To use a multiline string, you can either use the + operator or the \ operator

```
// using the + operator
const message1 = 'This is a long message ' +
    'that spans across multiple lines' +
    'in the code.'

// using the \ operator
const message2 = 'This is a long message \
that spans across multiple lines \
in the code.'
```

```
const a = 'hello';
const b = new String('hello');

console.log(a); // "hello"
console.log(b); // "hello"

console.log(typeof a); // "string"
console.log(typeof b); // "object"
```

Method	Description
charAt(index)	returns the character at the specified index
concat()	joins two or more strings
replace()	replaces a string with another string
split()	converts the string to an array of strings
substr(start, length)	returns a part of a string
substring(start,end)	returns a part of a string
slice(start, end)	returns a part of a string
toLowerCase()	returns the passed string in lower case
toUpperCase()	returns the passed string in upper case
trim()	removes whitespace from the strings
includes()	searches for a string and returns a boolean value
search()	searches for a string and returns a position of a match

STRING OBJECT

```
<html>
<body>
    <script>
        var st="Welcome to Java Script Programming";
        var str="Happy Programming";
        document.write(st.length+"<br>");
        document.write(st.bold()+"<br>");
        document.write("The character at 11th position is"+st.charAt(11)+"<br>");
        document.write(st.concat(str)+"<br>");
        document.write(str.substring(0,17)+"<br>");
        document.write(st.toLowerCase()+"<br>");
        document.write(st.toUpperCase()+"<br>");
        if(st.match(/Java/))
        {
        document.write("The string contains Java"+"<br>");
        }
        if(str.search(/Happy/)!=-1)
        {
        document.write(str.replace(/Happy/,"Enjoy"));
        }
    </script>
</body>
</html>
```

Built In Functions

1.Alert:

To create an alert dialog box-error message

2.Prompt: Interact with user.

```
<html>
<body>
   <script>
        var no1=prompt("Enter first number");
        var no2=prompt("Enter second number");
        var sum=no1+no2;
        document.write(sum);
   </script>
</body>
</html>
```

```
<html>
<body>
    <script>
        var no1=parseInt(prompt("Enter first number"));
        var no2=parseInt(prompt("Enter second number"));
        var sum=no1+no2;
        document.write(sum);
    </script>
</body>
</html>
```

3.confirm: based on response

```
<html>
<body>
    <script>
        var n=confirm("Are you sure to delete this file");
        if(n)
        document.write("The document is deleted");
        else
        document.write("Welcome");
    </script>
</body>
</html>
```

4.Evaluation Function(**eval**);

To calculate with in the function.

```
<html>
<body>
    <script>
        eval("x=25;y=20;document.write(x+y)");
        document.write("<br>"+eval("20*4"));
    </script>
</body>
</html>
```

5.isFinite(): is a more reliable way to test whether a value is a finite number value, because it returns false for any non-number input

```
<html>
<body>
   <script>
        document.write(isFinite(345));
        document.write("<br>"+isFinite("HAI"));
   </script>
</body>
</html>
```

6.isNaN():

NaN means Not A Number.

```
<html>
<body>
    <script>
        document.write(isNaN(345));
        document.write("<br>"+isNaN("HAI"));
    </script>
</body>
</html>
```

7.Number(): **converts a value to a number**. If the value cannot be converted, NaN is returned.

Input : Number(true);

Number(false);

Output : 1 0

Input : Number("10 20");

Output : NaN Not a number is returned by the compiler.

Input : Number(new Date("2017-09-30"));

Output : 1506729600000

The **Number()** method above returns the number of milliseconds since 1.1.1970.

Input : Number("John"); Output : NaN

8.Escape and Unescape Functions:

The escape function is used to encode a specified string, it returns the hexadecimal encoding of an argument

```
<html>
<body>
    <script>
        document.write(escape("Welcome to WT"));
        document.write("<br>");
        document.write( unescape("Welcome%20to%20WT"));
    </script>
</body>
</html>
```

Welcome%20to%20WT
Welcome to WT

The unescape function is used to decode a specified string

- Write A JavaScript Program To Validate Username:

```
<html>
<head>
    <script type="text/javascript">
        function validation()
        {
            var a=document.form.name.value;
            if(a=="")
            {
                alert("Please enter your name");
                document.form.name.focus();
                return false;
            }
        }
```

```
</script>
</head>
<body>
    <form name="form" method="POST" onsubmit="return
    validation()">
        <table>
            <tr>
            <td>Your Name</td>
            <td><input type="text" name="name"></td>
            </tr>
            <tr><td>
            <input type="submit" name="sub" value="Submit">
            </td></tr>
        </table>
    </form>
</body>
</html>
```

Password Pattern

- (# Start of group
- (?=.*\d) # must contains one digit from 0-9
- (?=.*[a-z]) # must contains one lowercase characters
- (?=.*[A-Z]) # must contains one uppercase characters
- (?=.*[@#$%]) # must contains one special symbols in the list "@#$%"
- . # match anything with previous condition checking
- {6,20} # length at least 6 characters and maximum of 20
-) # End of group

- **Password Validation Program:**

```
<html>
<head>
    <script type="text/javascript">
        function checkForm(form)
        {
        if(form.pwd1.value != "")
         {
                if(form.pwd1.value.length < 6)
                {
                alert("Error: Password must contain at least six characters!");
                form.pwd1.focus();
                return false;
                }
                re = /[0-9]/;
                if(!re.test(form.pwd1.value))
                {
                alert("Error: password must contain at least one number (0-9)!");
```

```
        form.pwd1.focus();
        return false;
        }
        re = /[a-z]/;
        if(!re.test(form.pwd1.value))
        {
        alert("Error: password must contain at least one lowercase letter (a-z)!");
        form.pwd1.focus();
        return false;
        }
        re = /[A-Z]/;
        if(!re.test(form.pwd1.value))
        {
        alert("Error: password must contain at least one uppercase letter (A-Z)!");
        form.pwd1.focus();
        return false;
        }
}
```

```
        else
         {
                alert("Error: Please Enter Password!");
                form.pwd1.focus();
                return false;
        }
        alert("You entered a valid password: " + form.pwd1.value);
        return true;
        }
    </script>
</head>
<body>
    <form  onsubmit="return checkForm(this);">
        <p>Password: <input type="password" name="pwd1"></p>
        <p><input type="submit"></p>
    </form>
</body>
</html>
```

- **Username and Password Validation:**

```
<html>
<head>
    <script type="text/javascript">
        function checkForm(form)
        {
        if(form.username.value == "")
        {
                alert("Error: Username cannot be blank!");
                form.username.focus();
                return false;
        }
        re = /^\w+$/;
        if(!re.test(form.username.value))
         {
                alert("Error: Username must contain only letters, numbers and
                        underscores!");
                form.username.focus();
                return false;
        }
```

```
if(form.pwd1.value != "" && form.pwd1.value == form.pwd2.value)
{
if(form.pwd1.value.length < 6)
{
        alert("Error: Password must contain at least six characters!");
        form.pwd1.focus();
        return false;
}
if(form.pwd1.value == form.username.value)
{
        alert("Error: Password must be different from Username!");
        form.pwd1.focus();
        return false;
}
re = /[0-9]/;
if(!re.test(form.pwd1.value))
{
        alert("Error: password must contain at least one number (0-9)!");
        form.pwd1.focus();
        return false;
}
```

```
re = /[a-z]/;
if(!re.test(form.pwd1.value))
{
        alert("Error: password must contain at least one lowercase
        letter (a-z)!");
        form.pwd1.focus();
        return false;
}
re = /[A-Z]/;
if(!re.test(form.pwd1.value))
{
        alert("Error: password must contain at least one uppercase
        letter (A-Z)!");
        form.pwd1.focus();
        return false;
}
}
else
 {
        alert("Error: Please check that you've entered and confirmed your
        password!");
        form.pwd1.focus();
        return false;
}
```

```
            alert("You entered a valid password: " + form.pwd1.value);
            return true;
        }
</script>
</head>
<body>
    <form onsubmit="return checkForm(this);">
            <p>Username: <input type="text" name="username"></p>
            <p>Password: <input type="password" name="pwd1"></p>
            <p>Confirm Password: <input type="password"
            name="pwd2"></p>
            <p><input type="submit"></p>
    </form>
</body>
</html>
```

Adding Error Handling

An error is an action that is inaccurate or incorrect. There are three types of errors in programming which are discussed below:

- **Syntax error:** syntax error is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language or it is also a compile-time error
- **Logical error :** It is the most difficult error to be traced as it is the error on the logical part of the coding or error is a bug in a program that causes it to operate incorrectly and terminate abnormally (or crash).
- **Runtime error** : A runtime error is an error that occurs during the running of the program, also known as the exception.

Exception is **an unwanted or unexpected event,** which occurs during **the execution of a** program, i.e. at run time, that disrupts the normal flow of the program's instructions.
Exceptions can be caught and handled by the program

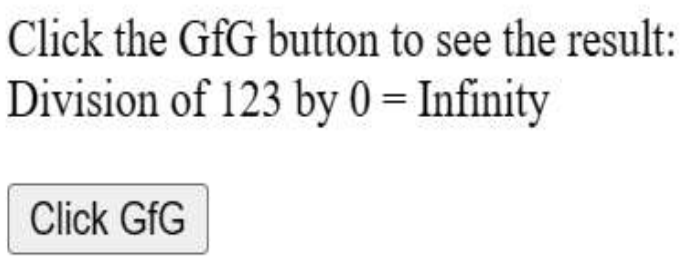
- JavaScript uses the try catch and finally to handle the exception and it also uses the throw operator to handle the exception.
- try have the main code to run and in the catch block if any error is thrown by try block will be caught and the catch block will execute.
- Finally block will always occur even if an error is thrown

```
try {
            // Here the main Code runs
            [break;]
    }
    catch ( exception e ){
        // The code will run when there is an exception
        [break;]
    }
```

Division by zero exception

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Document</title>
</head>
<body>
  <p id="gfg">
      Click the GfG button to see the result:
   </p>
   <form>
      <input type="button" value="Click GfG" onclick="First();" />
   </form>
   <script type="text/javascript">
      function First() {
         let a = 123
         let b = 0

         try {
            document.getElementById('gfg').innerHTML +=
               '<br> Division of ' + a + ' by ' + b + ' = ' + a / b
         }
         catch (e) {
            alert(e.description)
         }
      }
   </script>
</body>
</html>
```



```
<!DOCTYPE html>
<html lang="en">
<head>

  <title>Document</title>
</head>

<body>
  <script>
    function First() {
      let a = 123;
      let b = 0;
      try {
        if (b == 0) {
          throw "Do not divide by zero";
        }
      }
      catch (e) {
        document.getElementById('gfg').innerHTML
+=
          "<br>" + e + "<br>";
      }
      finally {
        document.getElementById('gfg').innerHTML
+=
          "  Finally block will always execute!";
      }
    }  </script>
```

```
<p id="gfg">Click the GfG button to see the
result:</p>
  <form>
    <input type="button" value="Click GfG"
onclick="First()" />
  </form>
</body>
</html>
```