What is JavaScript?

- HTML and CSS concentrate on a static rendering of a page; things do not change on the page over time, or because of events.
- To do these things, we use scripting languages, which allow content to change dynamically.
- Not only this, but it is possible to interact with the user beyond what is possible with HTML.
- Scripts are programs just like any other programming language; they can execute on the client side or the server.

Advantages of client side scripting

- The web browser uses its own resources, and eases the burden on the server.
- It has fewer features than server side scripting.
- It saves network bandwidth.

Disadvantages of client side scripting

- Code is usually visible.
- Code is probably modifiable.
- Local files and databases cannot be accessed.
- User is able to disable client side scripting.

Differentiate between server side and client side scripting languages

Client side scripting	Server side scripting
Source code is visible to user.	Source code is not visible to user because it's output of server side is a HTML page.
It usually depends on browser and it's version.	In this any server side technology can be use and it does not depend on client.
It runs on user's computer.	It runs on web server.
There are many advantages link with this like faster. response times, a more interactive application.	The primary advantage is it's ability to highly customize, response requirements, access rights based on user.

It does not provide security for data.	It provides more security for data.
It is a technique use in web development in which scripts runs on clients browser.	It is a technique that uses scripts on web server to produce a response that is customized for each clients request.
HTML, CSS and javascript are used.	PHP, Python, Java, Ruby are used.

What can we build using JavaScript ?
JavaScript is a widely-used programming language. Given below are some domains/products that can be built using JavaScript:

- **Websites:** JavaScript helps us to add behavior of our website. It helps users to interact with the website. For eg. clicking on buttons, saving details, uploading details on the website, etc.
- **Web Servers:** We can make robust server applications using JavaScript. To be precise we use JavaScript frameworks like Node.js and Express.js to build these servers.
- **Game Development:** In Game Development industry, JavaScript is used widely. With the addition of HTML5 Canvas, it's now possible to make 2D and 3D games in JavaScript very efficiently.
- **3D Drawings:** JavaScript in addition with HTML Canvas is used to make three-dimensional graphics.
- **Mobile Apps:** Mobile applications are the most popular modes of communicating these days. JavaScript also used to design mobile applications. There are many JavaScript frameworks using which we can make android, IOS, and hybrid apps.
- **Smart watch Apps:** The popular smart watch maker Pebble has created Pebble.js, a small JavaScript framework that allows a developer to create an application for the Pebble line of watches in JavaScript.

Why to learn JavaScript ?
JavaScript is the most popular and hence the most loved language around the globe. Apart from this, there are abundant reasons to learn it. Below are a listing of few important points:

- **No need of compilers:** Since JavaScript is an interpreted language, therefore it does not need any compiler for compilations.
- **Used both Client and Server-side:** Earlier JavaScript was used to build client-side applications only, but with the evolution of its frameworks namely Node.js and Express.js, it is now widely used for building server-side applications too.
- **Helps to build a complete solution:** As we saw, JavaScript is widely used in both client and server-side applications, therefore it helps us to build an end-to-end solution to a given problem.
- **Used everywhere:** JavaScript is so loved because it can be used anywhere. It can be used to develop websites, games or mobile apps, etc.
- **Huge community support:** JavaScript has a huge community of users and mentors who love this language and take it's legacy forward.

What is difference between Java script and JAVA?

- Java is a statically typed language; JavaScript is dynamic.
- Java is class-based; JavaScript is prototype-based.
- Java constructors are special functions that can only be called at object creation; JavaScript "constructors" are just standard functions.
- Java requires all non-block statements to end with a semicolon; JavaScript inserts semicolons at the ends of certain lines.
- Java uses block-based scoping; JavaScript uses function-based scoping.
- Java has an implicit this scope for non-static methods, and implicit class scope; JavaScript has implicit global scope.

Embedded JavaScript

- JavaScript can be embedded in an HTML document.
- To embed it in HTML you must write:

```
<script type="text/javascript">

</script>
```

- The script tag has effect of the stopping the JavaScript being printed out as well as indentifying the code enclosed.
- The JavaScript can be placed in the head section of your HTML or the body.

```
<html>

<body>

<script type="text/javascript">
        document.write("<h1>This is a
        heading</h1>");

</script>

</body>

</html>
```

- The Scripts placed in the body section are executed as the page loads and can be used to generate the content of the page.
- As well as the body section, JavaScript can also be placed in the head part.

- The advantages of putting a script in there are that it loads before the main body.

External JavaScript

- If you want to use the same script on several pages it could be a good idea to place the code in a separate file, rather than writing it on each.
- That way if you want to update the code, or change it, you only need to do it once.
- Simply take the code you want in a separate file out of your program and save it with the extension
  .js.

```
<html>

<body>

<script src="myScript.js"></script>

</body>

</html>
```

JavaScript Variables

- Variables in JavaScript behave the same as variables in most popular programming languages (C, C++, etc) do, but in JavaScript you don't have to declare variables before you use them.
- A variable's purpose is to store information so that it can be used later. A variable is a symbolic name that represents some data that you set.
- When using a variable for the first time it is not necessary to use "**var**" before the variable name.
- Variable names must begin with a letter.
- Variable names are case sensitive (y and Y are different variables).

var x=5; var y=6; var z=x+y many variables in one statement. Just start the statement with var and separate the;

- You can declare variables by comma:

```
varname="Doe",
age=30,
job="carpenter";
var name="Doe",

age=30,
job="carpenter";
```

- Variable declared without a value will have the value **undefined**.
- If you re-declare a JavaScript variable, it will not lose its value.
- The value of the variable *carname* will still have the value "Volvo" after the execution of the following two statements.

```
Varcarname="Volvo";
varcarname;
```

JavaScript Conditions

- Conditional statements are used to perform different actions based on different conditions.
- In JavaScript we have the following conditional statements:
    - o **if statement** - use this statement to execute some code only if a specified condition is true
    - o **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
    - o **if...else if .. else statement** - use this statement to select one of many blocks of code to be executed
    - o **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement

- Use the if statement to execute some code only if a specified condition is true.

```
if (condition)
{
        code to be executed if condition is true
}
```

If...else Statement

- Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

```
if (condition)
{
code to be executed if condition is true
}
```

```
else
 {
 code to be executed if condition is not true
 }
```

If...else if...else Statement

- Use the if....else if...else statement to select one of several blocks of code to be executed.

```
if (condition1) {
 code to be executed if condition1 is true
 }
else if (condition2) {
 code to be executed if condition2 is true
 }
else {
 code to be executed if neither condition1 nor condition2 is true
 }
```

Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

```
switch(n)
{
case 1:
  execute
  code block
  1 break;
case 2:
  execute
  codeblock
  2 break;

default:

  code to be executed if n is different from case 1 and 2

}
```

The *default* Keyword

- Use the *default* keyword to specify what to do if there is no match.

JavaScript Loops and Repetition

- Loops can execute a block of code a number of times.
- Loops are handy, if you want to run the same code over and over again, each time with a different value.

Different Kinds of Loops

- JavaScript supports different kinds of loops:
    - **for** - loops through a block of code a number of times
    - **for/in**- loops through the properties of an object
    - **while** - loops through a block of code while a specified condition is true
    - **do/while** - also loops through a block of code while a specified condition is true

The For Loop

- The for loop is often the tool you will use when you want to create a loop.
- The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3)

 {

 the code block to be executed

 }
```

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.

```
for (var i=0; i<5; i++)

 {
```

```
x=x + "The number is " + i + "<br>";
}
```

The For/In Loop

- The JavaScript for/in statement loops through the properties of an object.

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

The While Loop

- The while loop loops through a block of code as long as a specified condition is true.

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

The Do/While Loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

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

JavaScript Objects

- JavaScript has several built-in objects, like String, Date, Array, and more.
- An object is just a special kind of data, with **properties** and **methods**.

Accessing Object Properties

- Properties are the values associated with an object.
- The syntax for accessing the property of an object is below

objectName.propertyName

- This example uses the length property of the String object to find the length of a string:

```
var
message="Hello
World!"; var
x=message.length
;
```

Accessing Objects Methods

- Methods are the actions that can be performed on objects.
- You can call a method with the following syntax.

objectName.methodName()

- This example uses the toUpperCase() method of the String object, to convert a text to uppercase.

```
var
message="Hello
world!"; var
x=message.toUpper
Case();
```

Creating JavaScript Objects

- With JavaScript you can define and create your own objects.
- There are 2 different ways to create a new object:
  - Define and create a direct instance of an object.
  - Use a function to define an object, then create new object instances.

Creating a Direct Instance

- This example creates a new instance of an object, and adds four properties to it:

```
person=new Object();
person.firstname="Narendra";
person.lastname="Modi";
person.age=24;
person.eyecolor="blue";
```

User- Defined Objects/How user defined objects are created in JavaScript?

- JavaScript allows you to create your own objects.
- The first step is to use the *new* operator.

  Var myObj= new Object();

- This creates an empty object.
- This can then be used to start a new object that you can then give new properties and methods.
- In object- oriented programming such a new object is usually given a constructor to initialize values when it is first created.
- However, it is also possible to assign values when it is made with literal values.

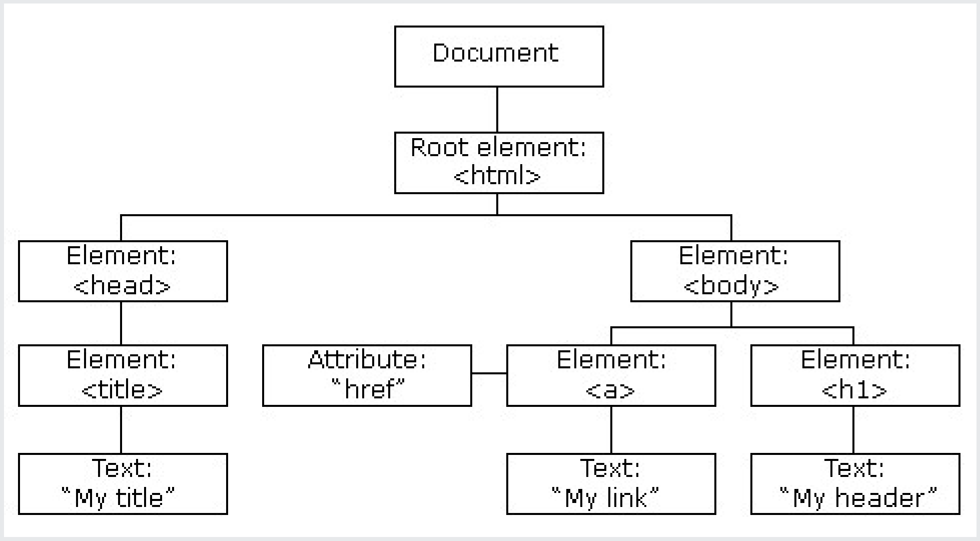
```
<!DOCTYPE html>
<html>
<body>
<script language=”JavaScript” type=”text/JavaScript”> person={
firstname: "Ketan", lastname: "Chavda", age: 24,
eyecolor: "blue"
}
document.write(person.firstname + " is " + person.age + " years old.");
</script>
</body>
</html>
```

Explain document object in JavaScript.

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



- There are lots of objects that descend from the *document* object forming a large sub-tree known as the Document Object Model (DOM), which has become standardized.
- The *document* object represents the HTML displayed in window.
- Using this object it is possible to access information about the document itself and also about the information content.
- Using this object it is possible to control certain parameters of the current document like background, foreground and link colors.

VarmyObj= new Object();
document.bgColor = "#9F2020";
documet.fgcolor = "#FAF519";

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Page</h2>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

In the example above, getElementById is a **method**, while innerHTML is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the innerHTML property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

Finding HTML Elements

Method	**Description**
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Property	**Description**
element.innerHTML = *new html content*	Change the inner HTML of an element

element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	**Description**
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	**Description**
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Why do you need validation? Show the use of regular expression in JavaScript to validate the email address with example.

- The idea behind JavaScript form validation is to provide a method to check the user entered information before they can even submit it.
- JavaScript also lets you display helpful alerts to inform the user what information they have entered incorrectly and how they can fix it.

JavaScript Email Address validation using Regular Expression

- Using Regular expression we do checking for valid information.

```
<html>
<head>
<title>JavaScript Forms</title>
</head>
<body>
<form method=”post” name=”getinfo” onSubmit=”return processForm()”>
<input type=”text” name=”email”/>
<input type=”submit” value=”log in” name=”Login”/>
</form>
<script language=”JavaScript” type=”text/JavaScript”> functionprocessForm()
{
Var myform= document.getinfo; var check= myform.email.value;
    Document.write(testEmail(check));
}
Function testEmail(chkMail)
```

```
{
varemailpattern = “^*\\w-_\.]*[\\w-_\.]\@[\\w]\.+[\\w]+[\\w+$”; var regex = new
     RegExp(emailpattern); returnregex.test(chkMail);
}
</script>
</body>
</html>
```

- This will check for a valid email as describe.
- The testEmail() function returns true or false.

- Thewayitdeterminesthisendresultisbuiltonthetemplate/patterninthestring *emailpattern.*
- This is used to work out the order of expected characters, how many times they repeat and specially occurring punctuation.
- The string in this case that is used as template is:

  “^*\\w-_\.][\\w-_\.]\@[\\w]\.+[\\w]+[\\w+$”*

- The first section is:

  ^[\\w-_\.]

- This sequence, beginning with ^, means check the first character is a word character represented by
  \\w.
- The next part is :

  **[\\w-_\.]*

- The * means that the next series of characters described can be represented many times or notat all.
- The characters themselves are the same as before; that is word characters, underscore, hyphen or period.

  \@[\\w]\.+

- This section begins by checking for the @ character.
- Following this should ne the word characters and then at least one ‘dot’.
- In other words it would not accept a dot straight after the @ character.
- The last part is :

[\\w]+[\\w]$

- The first set in square brackets makes sure that there are some characters after the dot and the last part checks that the last character is a word character.
- Inthisprogramafterthestringisdeclared,aregularexpressionobjectiscreatedwiththe pattern:

  var regex = new RegExp(emailpattern);

- Thepatterncanthenbetestedagainsttheincomingparameterwithobject'stestmethod:

  returnregex.test(chkMail);

- This will return true or false depending on whether there is a match or not.

Common HTML Events

Here is a list of some common HTML events:

Event	**Description**
Onchange	An HTML element has been changed
Onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element

onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

Event	**Description**	**Belongs To**
abort	The event occurs when the loading of a media is aborted	UiEvent, Event
afterprint	The event occurs when a page has started printing, or if the print dialogue box has been closed	Event

animationend	The event occurs when a CSS animation has completed	AnimationEvent
animationiteration	The event occurs when a CSS animation is repeated	AnimationEvent
animationstart	The event occurs when a CSS animation has started	AnimationEvent
beforeprint	The event occurs when a page is about to be printed	Event
beforeunload	The event occurs before the document is about to be unloaded	UiEvent, Event
blur	The event occurs when an element loses focus	FocusEvent
canplay	The event occurs when the browser can start playing the media (when it has buffered enough to begin)	Event
canplaythrough	The event occurs when the browser can play through the media without stopping for buffering	Event

change	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)	Event
click	The event occurs when the user clicks on an element	MouseEvent
contextmenu	The event occurs when the user right-clicks on an element to open a context menu	MouseEvent
copy	The event occurs when the user copies the content of an element	ClipboardEvent
cut	The event occurs when the user cuts the content of an element	ClipboardEvent
dblclick	The event occurs when the user double-clicks on an element	MouseEvent
drag	The event occurs when an element is being dragged	DragEvent
dragend	The event occurs when the user has finished dragging an element	DragEvent

dragenter	The event occurs when the dragged element enters the drop target	DragEvent
dragleave	The event occurs when the dragged element leaves the drop target	DragEvent
dragover	The event occurs when the dragged element is over the drop target	DragEvent
dragstart	The event occurs when the user starts to drag an element	DragEvent
drop	The event occurs when the dragged element is dropped on the drop target	DragEvent
durationchange	The event occurs when the duration of the media is changed	Event
ended	The event occurs when the media has reach the end (useful for messages like "thanks for listening")	Event
error	The event occurs when an error occurs while loading an external file	ProgressEvent, UiEvent, Event

focus	The event occurs when an element gets focus	FocusEvent
focusin	The event occurs when an element is about to get focus	FocusEvent
focusout	The event occurs when an element is about to lose focus	FocusEvent
fullscreenchange	The event occurs when an element is displayed in fullscreen mode	Event
fullscreenerror	The event occurs when an element can not be displayed in fullscreen mode	Event
hashchange	The event occurs when there has been changes to the anchor part of a URL	HashChangeEvent
input	The event occurs when an element gets user input	InputEvent, Event
invalid	The event occurs when an element is invalid	Event

keydown	The event occurs when the user is pressing a key	KeyboardEvent
keypress	The event occurs when the user presses a key	KeyboardEvent
keyup	The event occurs when the user releases a key	KeyboardEvent
load	The event occurs when an object has loaded	UiEvent, Event
loadeddata	The event occurs when media data is loaded	Event
loadedmetadata	The event occurs when meta data (like dimensions and duration) are loaded	Event
loadstart	The event occurs when the browser starts looking for the specified media	ProgressEvent
message	The event occurs when a message is received through the event source	Event

mousedown	The event occurs when the user presses a mouse button over an element	MouseEvent
mouseenter	The event occurs when the pointer is moved onto an element	MouseEvent
mouseleave	The event occurs when the pointer is moved out of an element	MouseEvent
mousemove	The event occurs when the pointer is moving while it is over an element	MouseEvent
mouseover	The event occurs when the pointer is moved onto an element, or onto one of its children	MouseEvent
mouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children	MouseEvent
mouseup	The event occurs when a user releases a mouse button over an element	MouseEvent
Mousewheel	Deprecated. Use the wheel event instead	WheelEvent

offline	The event occurs when the browser starts to work offline	Event
online	The event occurs when the browser starts to work online	Event
open	The event occurs when a connection with the event source is opened	Event
pagehide	The event occurs when the user navigates away from a webpage	PageTransitionEvent
pageshow	The event occurs when the user navigates to a webpage	PageTransitionEvent
paste	The event occurs when the user pastes some content in an element	ClipboardEvent
pause	The event occurs when the media is paused either by the user or programmatically	Event
play	The event occurs when the media has been started or is no longer paused	Event

playing	The event occurs when the media is playing after having been paused or stopped for buffering	Event
Popstate	The event occurs when the window's history changes	PopStateEvent
progress	The event occurs when the browser is in the process of getting the media data (downloading the media)	Event
ratechange	The event occurs when the playing speed of the media is changed	Event
resize	The event occurs when the document view is resized	UiEvent, Event
reset	The event occurs when a form is reset	Event
scroll	The event occurs when an element's scrollbar is being scrolled	UiEvent, Event
search	The event occurs when the user writes something in a search field (for <input="search">)	Event

seeked	The event occurs when the user is finished moving/skipping to a new position in the media	Event
seeking	The event occurs when the user starts moving/skipping to a new position in the media	Event
select	The event occurs after the user selects some text (for <input> and <textarea>)	UiEvent, Event
show	The event occurs when a <menu> element is shown as a context menu	Event
stalled	The event occurs when the browser is trying to get media data, but data is not available	Event
Storage	The event occurs when a Web Storage area is updated	StorageEvent
submit	The event occurs when a form is submitted	Event
suspend	The event occurs when the browser is intentionally not getting media data	Event

timeupdate	The event occurs when the playing position has changed (like when the user fast forwards to a different point in the media)	Event
toggle	The event occurs when the user opens or closes the <details> element	Event
touchcancel	The event occurs when the touch is interrupted	TouchEvent
touchend	The event occurs when a finger is removed from a touch screen	TouchEvent
touchmove	The event occurs when a finger is dragged across the screen	TouchEvent
touchstart	The event occurs when a finger is placed on a touch screen	TouchEvent
transitionend	The event occurs when a CSS transition has completed	TransitionEvent

unload	The event occurs once a page has unloaded (for <body>)	UiEvent, Event
volumechange	The event occurs when the volume of the media has changed (includes setting the volume to "mute")	Event
waiting	The event occurs when the media has paused but is expected to resume (like when the media pauses to buffer more data)	Event
wheel	The event occurs when the mouse wheel rolls up or down over an element	WheelEvent

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	Onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form

blur	Onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	Onload	When the browser finishes the loading of the page
unload	Onunload	When the visitor leaves the current webpage, the browser unloads it
resize	Onresize	When the visitor resizes the window of the browser

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers
- JSON is language independent *

*
The JSON syntax is derived from JavaScript object notation, but the JSON format is text only.

Code for reading and generating JSON exists in many programming languages.

Why Use JSON?

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

`JSON.parse()`

JavaScript also has a built in function for converting an object into a JSON string:

`JSON.stringify()`

Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs (aka key/value pairs).

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example

```
"name":"John"
```

JSON names require double quotes.

JSON - Evaluates to JavaScript Objects

The JSON format is almost identical to JavaScript objects.

In JSON, *keys* must be strings, written with double quotes:

JSON

```
{"name":"John"}
```

In JavaScript, keys can be strings, numbers, or identifier names:

JavaScript

```
{name:"John"}
```

JSON Values

In **JSON**, *values* must be one of the following data types:

- a string
- a number
- an object
- an array
- a boolean
- null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

- In JSON, *string values* must be written with double quotes:
- JSON
- `{"name":"John"}`
- In JavaScript, you can write string values with double *or* single quotes:
- JavaScript
- `{name:'John'}`

JavaScript Objects

Because JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

Example

```
person = {name:"John", age:31, city:"New York"};
```

You can access a JavaScript object like this:

```
<!DOCTYPE html>
<html>
<body>

<h2>Access a JavaScript object</h2>
<p id="demo"></p>

<script>
const myObj = {name:"John", age:30, city:"New York"};
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

You can access a JavaScript object like this also:

```
<!DOCTYPE html>
<html>
<body>

<h2>Access a JavaScript object</h2>
<p id="demo"></p>

<script>
const myObj = {name:"John", age:30, city:"New York"};
document.getElementById("demo").innerHTML = myObj["name"];
</script>

</body>
</html>
```