UNIT-4

Client Side Scripting with JavaScript

Javascript is a programming language. This means developer has control over how the program is executed. Javascript files have a .js extension. Javascript is mostly known as the programming language of the web, which helps to change behaviour of web page. It provides control about what happens when the user interacts with it. To run the .js file in a HTML page we must refrence the javascript file using a <script> HTML tag. The <script> tags should go inside the page's <head> tag. The script tag has an src attribute that points to the javascript file.

Example:-
```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page.</title>
        <script type="text/javascript" src="file.js"></script>
    </head>
    <body>
        <h1>Page heading</h1>
        <p>Page content</p>
    </body>
</html>
```

JS Variables:

Javascript allows us to give values ~~lables.~~ labels. A label is called a variable and can store a single value. There are 3 ways to declare a javascript variable:

Using var:

Variables are containers for storing data (values). In this example x, y and z are variables, declared with var keyword:

Example:
```
var x=5;
var y=6;
var z=x+y;
```

From this example we can expect that x stores the value 5, y stores the value 6 and z stores the value 11. Here '=' is an assignment operator. Variables declared with var has functional Scope.

iii) Using let:-

Variables defined with let can not be redeclared. Variables defined with let must be declared before use. Variables defined with let have block scope.

Example:- let x=25;

Now if we try to redeclare x as x=0 then we will get syntax error saying 'x' has already been declared.

iii) Using Const:-

Variable defined with const cannot be redeclared, as well as cannot be reassigned. Variables defined with const have block scope.

Example: const PI= 3.14159

PI=3.14; // This will give an error.

PI = PI+10; // This will also give an error.

Javascript const variables must be assigned a value when they are declared:

```
const PI=3.14159;
```

✓ Correct

```
const PI;
PI=3.14159;
```

✗ Incorrect

⊛. JS Data Types:-

i) Number → A number type represents both integer and floating point numbers. There are many operations for numbers e.g, multiplication *, division /, addition +, subtraction −, and so on. Besides regular numbers, it can have special numeric values: Infinity and NaN.

ii) BigInt → In Js "number" type cannot represent integer values larger than $(2^{53}-1)$, or less than $-(2^{53}-1)$ for negatives. For most cases it's quite enough, but sometimes we need really big numbers which can be assigned with the help of BigInt.

iii) String → A string in JavaScript must be surrounded by quotes single or double, or backticks. Double and single quotes are "simple" quotes. Backticks are "extended functionality" quotes.

iv) Boolean → The boolean type has only two values: true and false. This type is commonly used to store yes/no values.

v) The "undefined" value → The meaning of undefined is "value is not assigned". If a variable is declared, but not assigned, then its value is undefined.

⊛ JS Statements:-

A computer program is a list of "instructions" to be "executed" by a computer. In programming language, these programming ~~stat~~ instructions are called statements. A JavaScript program is a list of programming statements. JavaScript statements are composed of values, operators, expressions, keywords and comments.

- Expressions → Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation. Expressions can be arithmetic expression (e.g, 10+3), string expression (eg. 'hello' + 'world'), logical expressions (a===20 && b===30) etc.

  (=> triple = compares both value and type)

Keywords → Keywords are the reserved words in JavaScript that we cannot use to indicate variable and function names. There are 63 keywords that JS provides to programmers like int, let, return, switch, if, else, for etc.

⊛. JS Operators:-

i) The assignment operator (=) assigns a value to a variable.

Example: `let x = 10;`

ii) The addition operator (+) adds two numbers.

Example:
```
let x = 5;
let y = 2;
let z = x+y;
```

iii) The multiplication operator (*) multiplies numbers.

Similarly we have Subtraction (-), Division (/), Modulus (%), Increment (++), Decrement (--) etc.

⊛. Flow Controls:-

In computer programming, flow control refers to the logic that decides whether certain blocks of code will be executed or not. The most common source of flow control in JavaScript is BEDMAS, which describes the order of operation in algebra. It stands for:

→ Brackets
→ Exponents
→ Division
→ Multiplication
→ Addition
→ Subtraction.

The brackets are computed first, then exponents, then division and so on...

In JS: // → specifies single line comment
/*...*/ → specifies multi line comment.

1) "If else" statement:-

```
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.
}
```

Example:

```
if (age > 18){
    citizen = "voter";
}
else{ citizen = "non-voter";
}
```

2) "Else if" statement:-

```
if(condition1){
    //block of code to be executed if condition1 is true.
}
else if (condition 2){
    //block of code to be executed if condition 1 is false and
    condition 2 is true
}
else{
    //block of code to be executed if the condition 1 is false
    and condition2 is false.
}
```

Note:- There can be more than one else if block.

Example:

```
if (time < 10){
        greeting = "Good morning";
}
else if (time < 20) {
        greeting = "Good day";
}
else{ greeting = "Good evening";
}.
```

3) Switch Statement:-

We use the switch statement to select one of many code blocks to be executed.

```
switch (expression){
        case x:
            //code block
            break;
        case y:
            //code block
            break;
        default:
            //code block.
}
```

Example:-

```
switch (new Date().getDay()){
        case 0:
            day = "Sunday";
            break;
        case 1:
            day = "Monday";
            break;
        case 2:
            day = "Tuesday";
            break;
        case 3:
            day = "Wednesday";
            break
        case 4:
            day = "Thursday";
            break;
        case 5:
            day = "Friday";
            break;
        case 6:
            day = "Saturday";
            break;
}
```

⊛. JS Looping Statements:-

1) For loop:

Syntax: for (initialization; test condition; iteration statement){
    //statements to be executed
}

(iteration statement → increment or decrement)

Example:
```
for (let i=0; i<5; i++){
    number = number + i;
}
```

2) For In Loop:

Syntax:
```
for (key in object){
    //body for for...in
}
```

Example:
```
const salaries = {Jack: 24000, Paul: 34000, Monica: 55000}

//using for...in
for (let i in salaries){
    salary = "$" + salaries[i];
}
```

3) For Of Loop:

Syntax:
```
for (variable of iterable){
    //code block to be executed
}
```

Example:-
```
const array1 = ['a', 'b', 'c'];
for (const element of array1){
    console.log(element);
}
```

(console.log is used for output such as printf in C)

4) While loop:

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

Syntax:
```
(condition) {
    //code of block to be executed
}
```

Example:
```
while (i<10) {
    text = text + "The number is" + i;
    i++;
}
```

5) do/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.

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

Example:
```
do{ text += "The number is" + i;
    i++;
}
while (i<10);
```

⊛. JS Functions:-

A JavaScript function is a block of code designed to perform a particular task and is executed when "something" invokes it (calls it). A JavaScript function is defined with the function keyword, followed by a function name, followed by parenthesis (). Function names can contain letters, digits, underscores and dollar signs (same rules as variables). The parenthesis may include parameter names separated by commas. The code to be executed by the function is placed inside curly brackets: {}.

Syntax:
```
function function_name (parameter1, parameter2, ...){
    //code to be executed
}
```

Example:
```
function myFunction (p1, p2){
    return p1*p2;
}
```

The code inside the function will execute when it is called/invoked. For example above function can be invoked/called as:

```
myFunction (10, 5);
```

When javascript reaches a return statement, the function will stop executing and the return value is "returned" back to the "caller"; Example: let x = myFunction (10,5); Now, the value returned by function above will return computed value to caller myFunction(10,5) which will then assign it to variable x. Now, x contains computed value 50.

⊛ Popup Boxes:-

Javascript Popup boxes are three types alert box, confirm box, prompt box. Javascript alert box (alert message display to a browser). Javascript confirm box (verify or accept some confirm message from user and display on browser). Javascript prompt box (fetch value from user and display on browser).

Alert box:

Syntax: alert ("Any message text");

Example:-
```
<!DOCTYPE html>
<html>
  <head> <title> JavaScript Alert Box </title>
  </head>
  <body>
    <input type="button" onclick="alert ("This is alert box");" value="Open Alert Box" />
  </body>
</html>
```

Confirm box:

Syntax: confirm ("Are you sure to perform this action.");

Example:- Same as above just replace onclick="confirm ("Are you sure to perform this action.");

Prompt box:

Syntax: prompt ("Enter some value", "default value");

Example:- Same as above just replace onclick part using syntax as; onclick="prompt ("Enter your lucky digit number", "2");"

⊛ Objects and properties:-

Objects are similar to variables but objects can contain many values of different types. The values are written as name:value pairs. It is a common practice to declare objects with the const keyword. The name:values pairs in Javascript objects are properties.

Example:- const person = {firstName: "John", lastName: "Doe", age: 50};

Spaces and line breaks are not important. An object definition can span multiple lines. For example same above example can also be written as;

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

We can access object properties in two ways:

objectName.propertyName OR objectName["propertyName"].

e.g. person.lastName e.g. person["lastName"].

The This Keyword:-

In a function definition, this refers to the "owner" of the function. The value of this, when used in an object, is the object itself. In a constructor function this does not have a value. It is a substitute for the new object. The value of this will ~~becomes~~ become the new object when a new object is created. Note that this is not a variable. It is a keyword. We cannot change the value of this.

⊛. Object Constructors:-

Sometimes we need a "blueprint" for creating many objects of same "type". The way to create an "object type", is to use an object constructor function. In the example below, function Person() is an object constructor function.

```
function Person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
```

Objects of same type are created by calling constructor function with the new keyword:

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

⊛. Arrays:- [Imp]

An array is a special variable, which can hold more than one value at a time. We can access the values of an array by referring to an index number. The index number of an array starts with 0. Arrays are special types of objects.

Syntax: const array-name = [item1, item2, ...];

Example: const cars = ["Saab", "Volvo", "BMW"];

Spaces and line breaks are not important. A declaration can span multiple lines:

```
const cars = [
    "Saab",
    "Volvo",
    "BMW"
];
```

We can also create an array, and then provide elements:

Example: const cars = [];
cars[0] = "Saab";
cars[1] = "Volvo";
cars[2] = "BMW";

Array Methods:

i) toString() → converts an array to a string of (comma seperated) array values. Example:- const fruits = ["Banana", "Orange", "Apple"];
myString = fruits.toString();

ii) join() → The join() method joins all array elements into a string. Example:- const fruits = ["Banana", "Orange", "Apple"];
mystring = fruits.join("*");

Result: *Banana*Orange*Apple.

iii) pop() → This removes the last ~~one~~ element from an array.
Example: fruits.pop(); will remove Apple from above fruit array.

iv) push() → This adds new element to the end of an array.

v) shift() → Removes the first array element and shifts all other elements to a lower index.

vi) unshift() → Adds the new element at begining of array, and shifts all other elements to a higher index.

⊛ Built-in Objects:-

Built-in objects are not related to any Window or DOM object model. These objects are used for simple data processing in the Javascript.

1) Math Object:- Math object is a built-in static object. It is used for performing complex math operations.

Method	Description

i) abs(x)
e.g: Math.abs(-4.7);
//returns 4.7
	Returns the absolute (positive) value of x

ii) round(x)
eg: Math.round(4.7);
//returns 5
	Returns the value of x rounded to its nearest integer.

iii) pow(x,y)
e.g: Math.pow(8,2);
//returns 64
	Returns the value of x to the power of y.

iv) sqrt(x)
e.g, Math.sqrt(64);
//returns 8
	Returns the square root of x.

v) random()
e.g: Math.random();
//returns a random number
	Returns a random number between 0 and 1.

2) Date Object:- Date objects are created with the new Date() constructor. There are 4 ways to create a new date object:

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

Examples:-

```
var d = new Date();
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
var d = new Date("October 13, 2014 11:13:00");
```

Methods		Description
Date()	→	Returns current date and time.
getDate()	→	Returns the day of the month.

getDay() → Returns the day of the week
getFullYear() → Returns the year.
getHours() → Returns the hour.
getMinutes() → Returns the minutes.
getSeconds() → Returns the seconds.
getMonth() → Returns the month.
setDate() → sets the day of the month.
setFullYear() → Sets the full year.

3) Strings :- JS strings are used for storing and manipulating text.
A JavaScript string is zero or more characters written inside quotes.
Example :- var x = "John Doe"; ← we can use single or double quotes

To find length of string we use built-in length property:
Example:- var txt = "ABCDEFGHIJ";
var txln = txt.length;

Strings can be defined as objects with the keyword new:
var firstName = new String ("John");

Methods	Description
charAt()	→ It returns the character at specified index.
charCodeAt()	→ It returns the ASCII code of the character at specified position.
concat()	→ It combines the text of two strings and returns new string.
indexOf()	→ It returns the index within the calling string object.
toLowerCase()	→ It returns the calling string value converted lower case.
toUpperCase()	→ It returns the calling string value converted to upper case.
slice()	→ It extracts a session of a string and returns new string.
split()	→ It splits a string object into an array of strings by seperating the string into substrings.

4) Numbers:- Unlike many other programming languages, JavaScript does not define different types of numbers like integers, short, floating-point etc. JavaScript numbers are always stored as

double precision floating point numbers. Numbers can be written with or without decimals.

Example: var x=3.14
var y=3;

Method	Description
toString()	→ This method returns a number as a string.
toFixed()	→ Returns a string, with the number written with a specified number of decimals.
toPrecision()	→ Returns a string, with a number written with a specified length.
valueOf()	→ Returns a number as a number.
Number()	→ Used to convert JS variables to numbers.

5) Booleans:- A JavaScript Boolean represents one of two values: true or false. We can use Boolean() function to find out if an expression (or a variable) is true.

Example: Boolean(10>9) //returns true.

The Boolean value of 0, -0, "" (i.e, empty string), undefined, null, false, NaN will return false.

Example: var x=0;
Boolean(x); //returns false.

Booleans can also be defined as objects with the keyword new:

Example: var y = new Boolean(false);

6) Regular Expressions:- A regular expression is a sequence of characters that forms a search pattern. 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;

Here, /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).

Regular Expression Patterns:-

Brackets are used to find range of characters:

[abc] → Find any of the characters between the brackets.
[0-9] → Find any of the digits between the brackets.
(x|y) → Find any of the alternatives seperated with |.

Metacharacters are characters with a special meaning:

\d → Find a digit
\s → Find a whitespace character.
\b → Find a match at the begining or at the end of word.
\uxxxx → Find the Unicode character specified by hexadecimal number xxxx.

Quantifiers define quantities:

n+ → Matches any string that contains at least one n.
n* → Matches any string that contains zero or more occurrences of n.
n? → Matches any string that contains zero or one occurrences of n.

Regular Expression Modifiers:-

i → Perform case-insensitive matching
g → Finds all matches rather than stopping after first match. (global match).
m → Perform multiline matching.

Methods	Description
test()	Searches a string for a pattern, and returns true or false depending on the result.

Example:-
```
var patt = /e/;
patt.test("The best things in life are free!");
```
⇒ Since there is an "e" in the string, the output will be: true.
The two lines in example can be shortened as;
```
/e/.test("The best things in life are free!");
```

exec() → Searches a string for a specified pattern, and returns found text as object.

Example:
```
/e/.exec("The best things in life are free!");
```

7) Window:-

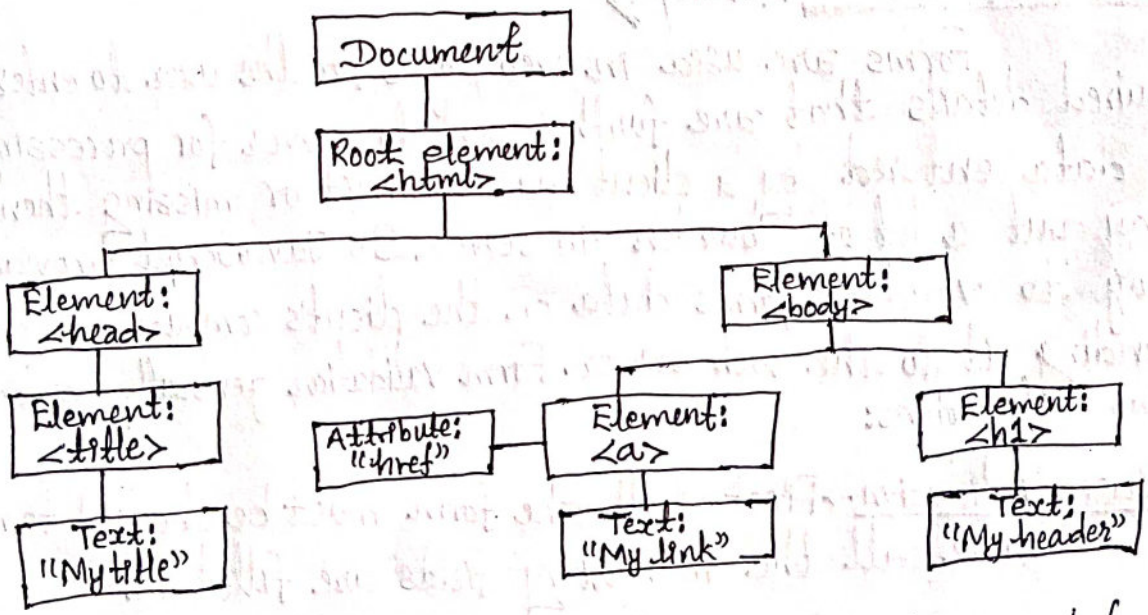
The Window object is supported by all browsers. It represents the browser's window. All global Javascript objects, functions, and variables automatically become members of the window object. Two properties can be used to determine the size of browser window. Both properties return sizes in pixels:

Window.innerHeight → Specifies inner height of browsers window.
Window.innerWidth → Specifies inner width of the browsers window.

Methods	Description
window.open()	Open a new window
window.close()	Close the current window
window.moveTo()	Move the current window
window.resizeTo()	Resize the cursent window.

8) DOM (Document Object Model): [Imp]

The HTML DOM is a standard for how to get, change, add or delete HTML elements. With the DOM, JavaScript gets all the power it needs to create dynamic HTML. When a web page is loaded, the browser creates a DOM of the page. The HTML DOM model is constructed as a tree of objects.



In the DOM, all HTML elements are defined as objects. A property is a value that we can get or set (like changing the content of HTML element). A method is an action we can do (like add or delete an HTML element).

Example:- The following example changes the content of innerHTML of the <p> element with id="demo".

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

Note: In this example, 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 getElementById method uses 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.

⊛. Form Validation: [V. Imp]

Forms are used in web pages for the user to enter their required details that are further sent to server for processing. If data entered by a client was incorrect or missing then it may put a lot of burden in server. So JavaScript provides a way to validate forms data on the client's computer before sending it to the web server. Form validation generally performs two functions:

Basic Validation → First of all, the form must be checked to make sure all the mandatory fields are filled in.

Data Format Validation → Secondly, the data that is entered must be checked for correct form and value.

Example: Form validation example combining all input types validation.
(OR Example to demonstrate form validation).

```
<html>
  <head>
    <title>Form Validation</title>
  </head>
  <body>
    <form name="myForm" onsubmit="return validation()">
      Username: <input type="text" id="name"> <br>
      Number: <input type="text" id="number"> <br>
      Gender: <input type="radio" name="gender" id="male">M
              <input type="radio" name="gender" id="female">F <br>
              <input type="submit" value="Submit">
    </form>
  //JS starts here just before end of body tag.
  <script type="text/javascript">
    function validation(){
        // Username or Name validation
    var name=document.getElementById("name").value;
      if(name==""){
          alert("Enter name");
      }
    var condition=/^[A-Z a-z]+$/;
    var result=name.match(condition);
      if(result==NULL){
          alert("Name format not valid.");
      }

    //Number Validation
    var number=document.getElementById("number").value;
    if(number==""){
        alert("Enter number");
    }
    else if(isNaN(number)){
        alert("Entered value is not a number");
    }
```

This is Regular expression pattern. This specifies name can contain letters from A-Z or a-z and any letter can repeat.

.match returns object value if it matches condition

Result contains NULL value if name.match(condition) is false or user entered invalid name. But if user enter valid name it will contain object having user input and other values.

```
// Radio validation.
    var male = document.getElementById("male").checked;
    var female = document.getElementById("female").checked;
        if (male==false && female==false){
            alert("Gender need to be selected");
        }
    }
}
</script>
</body>
</html>
```

.checked returns boolean value. either true → if checked, false → if not checked

validation function closed

Solution of these questions can be found on collegenote website in old questions and solutions section - if needed

#Important Questions Related to Form Validation: [Practice these questions].

1. Design a form containing textbox for username, password field for password, radio button for gender and checkbox for hobbies. Write a program for client-side validation of the form for the user name and password field as required fields, length of username should be 5, the radio button and checkbox should be checked.

2. Write HTML script for creating a form containing textbox for username, password field for password, and checkbox for Education Fields. Write a JavaScript function for the validation of the form for all of the fields as required. In addition length of username should at least 4, the password should start with digit and end with $.

⊛. Event Handling: [Imp]

Event handlers are JavaScript code that are not added inside the <script> tags, but rather inside the html tags, that execute JavaScript when something happens, such as pressing a button, moving mouse over a link, submitting a form etc. The basic syntax of these event handlers is:

```
name_of_handler="JavaScript code goes here".
```

For Example:

```
<a href="http://google.com" onclick="alert('hello')"> Google. </a>
```

When the above link is clicked, the user will first see an alert message before being taken to Google.

Some event handlers used in JavaScript:-

onclick → Used to invoke JS upon clicking.
onload → Used to invoke JS after page or image has finished loading.
onmouseout → Used to invoke JS if the mouse goes pass some link.
onmouseover → Used to invoke JS if the mouse passes by some link.
onunload → Used to invoke JS right after someone leaves page.

JavaScript code to demonstrate event handling:

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

⊛ Error Handling (Try/Catch):

There are three types of errors in programming:

❶ Syntax Errors :- Syntax errors occur at compile time in traditional programming languages and at interpret time in Javascript. For example the following line causes a syntatic error because it is missing a closing parenthesis.

```
<script type="text/javascript">
  window.print(;
</script>
```

❷ Runtime Errors:- Runtime errors, also called exceptions, occur during execution. For example, the following line causes a runtime error because the syntax is correct, but at runtime it is trying to call a method that does not exist.

```
<script type="text/javascript">
  window.printme();
</script>
```

iii) Logical Errors:- Logic errors can be the most difficult types of errors to track down. Logical errors occur when we make a mistake in the logic that drives our script and we do not get the result as we expected. We can catch those errors, because it depends on our requirement what type of logic we want to put in our program.

The try...catch...finally Statement:

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try... catch... finally construct to handle exceptions. Here is the try... catch... finally block syntax:

```
<script type="text/javascript">
    try {  //Code to run
           break;
    }
    catch (e) {
           //Code to run if an exception occurs.
           break;
    }
    [finally {
           //Code that is always executed regardless of
           // an exception occuring.
    }]
</script>
```

⊛. Handling Cookies:- [Imp]

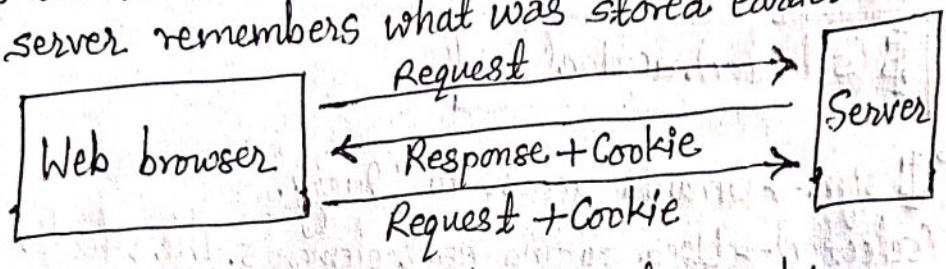
A cookie is variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JS, we can both create and retrive cookie values. Name cookie, Date cookie etc. are examples of cookies.

Use of Cookies:- Cookies let us to store user information in web pages. When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user. Cookies were invented to solve the problem "how to remember information about the user". When a user visits a web page his/her information can be stored in a cookie.

Next time when the user visits the page, the cookie "remembers" his/her name. Cookies are saved in name-value pairs like: username=Roshan

How it works?

Our server sends some data to visitor's browser in the form of cookie. The browser may accept the cookie. If it does, it is stored as a plain text record in the visitor's hard drive. Now, when visitor arrives at another page on our site, the browser sends the same cookie to the server for retrival. Once retrived, our server remembers what was stored earlier.



Example:- JavaScript code to set and get a cookie:

```
<!DOCTYPE html>
<html>
    <head> <title> Cookie </title> </head>
    <body>
        <input type="button" value="setCookie" onclick="setCookie()">
        <input type="button" value="getCookie" onclick="getCookie()">
        <script type="text/javascript">
            function setCookie(){
                document.cookie = "username=Roshan";
            }
            function getCookie(){
                if(document.cookie.length != 0){
                    alert(document.cookie);
                }
                else{
                    alert("Cookie not available");
                }
            }
        </script>
    </body>
</html>
```

jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It was designed to handle ~~browse~~ browser incompatibilities and to simplify HTML DOM manipulation, Event handling, animations and Ajax. There are several ways to use jQuery in our web site but simplest and fastest way is to include jQuery CDN.

jQuery Syntax:

```
$(selector).action()
```

where, $ sign → Grants access to jQuery.
(selector) → Used to find html elements. The selector may be id selector or class selector.
action() → It is used to perform actions on elements.

Examples:
$(this).hide() → hides the current element.
$(".test").hide() → hides all elements with class="test".

⊛ jQuery Selectors:

jQuery selectors allow us to select and manipulate HTML element(s). jQuery selectors are used to "find" (or select) HTML elements based on their name, id, class, attributes and more. All selectors in jQuery start with dollar sign and parenthesis: $(). The different types of jQuery selectors are:-

1) The element selector: The jQuery element selector selects elements based on the element name.

Example: When a user clicks on a button, all <p> elements will be hidden:

```
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
```

ii) The id selector: The jQuery id selector uses the id attribute of an HTML tag to find the specific element. To find an element with a specific id, write a hash character, followed by id of HTML element.

Example:- When the user clicks on a button, the element with id="test" will be hidden:

```
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
```

iii) The class selector: The jQuery class selector uses class attribute of an HTML tag to find the specific element. To find an element with a specific class, we write a period (dot) character, followed by name of class.

Example: When the user clicks on button, the element with class="test" will be hidden:

```
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
    });
});
```

⊛. jQuery Events:-

All the different visitors actions that a web page can respond to are called events. An event represents the precise moment when something happens.

Examples:

→ Moving a mouse over an element.
→ Selecting a radio button
→ Clicking on an element.

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload.

⊛. jQuery Effects:-

i) hide/show : With jQuery we can hide and show HTML elements with the hide() and show() methods:

Syntax: $(selector).hide(speed, callback);
$(selector).show(speed, callback);

The optional speed parameter specifies the speed of the hiding or showing and can take values: "slow", "fast", or milliseconds.
The optional callback parameter is a function to be executed after the hide() or show() method completes.

Example:
```
$("button").click(function(){
    $("p").hide(1000);
});
```

ii) fade:- With jQuery we can fade an element in and out of visibility. jQuery has fadeIn(), fadeOut(), fadeToggle(), fadeTo() fade methods. Let us take jQuery fadeIn() method.

Syntax: $(selector).fadeIn(speed, callback);

Example:
```
$("button").click(function(){
    $("p").fadeIn(1000);
});
```

iii) Slide:- The jQuery slide methods slide elements up and down. It creates sliding effect on elements. jQuery has slideDown(), slideUp(), slideToggle() methods. Let us take slideDown() method.

Syntax: $(selector).slideDown(speed, callback);

Example:
```
$("button").click(function(){
    $("#plane1").slideDown(1000);
});
```

iv) Animate: With jQuery we can create custom animations.

Syntax: $(selector).animate({parameters}, speed, callback);
The parameters specifies the CSS properties to be animated.

Example:-
```
$("button").click(function(){
    $("#div").animate({left:'250px'});
});
```

iv) stop(): The jQuery stop() method is used to stop animations or effects before it is finished. The stop() method works for all jQuery effect functions, including sliding, fading and custom animations.

Syntax: $(selector).stop(stopAll, goToEnd);

The optional stopAll parameter specifies whether also the animation queue should be cleared or not, default is false. The optional goToEnd parameter specifies whether or not to complete the current animation immediately, default is false.

Example:
```
$("button").click(function(){
    $("#div").stop();
});
```

v) Callback: A callback function is executed after the current effect is finished. Syntax: $(selector).hide(speed, callback);

Example:
```
$("button").click(function(){
    $("p").hide("slow", function(){
        alert("The paragraph is now hidden");
    });
});
```

vi) Chaining: Chaining allows us to run multiple jQuery commands, one after another, on the same element(s). To chain an action we simply append the action to the previous action. The following example chains together the css(), slideUp(), and slideDown() methods.

Example:
```
$("button").click(function(){
    $("#div").css("color", "red").slideUp(2000).slideDown(200
});
```

JSON:

JSON stands for JavaScript Object Notation. JSON is a syntax for storing and exchanging data. When exchanging data between browser and a server, the data can only be text. JSON is text and we can convert any JavaScript object into JSON, and send JSON to the server. We can also convert any JSON received from the server into JavaScript objects.

⊛. JSON Syntax:-

A JSON object looks something like this:

```
{
    "book":[
        {"id":"01",
         "language":"JavaScript",
         "edition":"third"
        },
        {"id":"02",
         "language":"Java",
         "edition":"second"
        }
    ]
}
```

→ Data is in name/value pairs consisting field name (in double quotes), followed by a colon, followed by a value:

Example: "edition": "third"

Note: JSON names require double quotes. JavaScript names don't.

→ Data is separated by commas.
→ Curly braces hold objects.
→ Square brackets hold arrays.

⊛. JSON Data Types:

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

→ a string e.g: {"name":"John"}
→ a number e.g: {"age":30}
→ an object (JSON) object e.g: {"employee":{"name":"John", "age":30}}
→ an array e.g:- {"employees":["John", "Anna", "Peter"]}
→ a boolean e.g: {"sale":true}
→ null e.g: {"middlename":null}

⊛. Parsing JSON:

We use the JavaScript function JSON.parse() to convert text into a JavaScript object. We should make sure that the text is written in JSON format, or else we will get a syntax error.

Example:-

```
<!DOCTYPE html>
<html>
<body>
    <h2>Create Object from JSON String</h2>
    <p id="demo"></p>
    <script type="text/javascript">
        var txt = '{"name":"John", "age":30, "city":"New York"}'
        var obj = JSON.parse(txt);
        document.getElementById("demo").innerHTML = obj.name + "," + obj.age;
    </script>
</body>
```

Output:

Create Object from JSON String
John,30

[Imp] Q. Write a Javascript code to create an array of elements {csit, it, bca, bim}. Display the array in body of HTML.

Solution:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Display array inside body.</title>
    </head>
    <body>
        <div id="demo">
        </div>
        <script type="text/javascript">
            var array = ["csit", "it", "bca", "bim"];
            document.getElementById("demo").innerHTML = array;
        </script>
    </body>
</html>.
```