PART B

11.Difference between Client side JavaScript and Server side JavaScript?

Client-side scripting	Server-side scripting
Source code is visible to the user.	Source code is not visible to the user because its output of server-sideside is an HTML page.
Its main function is to provide the requested output to the end user.	Its primary function is to manipulate and provide access to the respective database as per the request.
It usually depends on the browser and its version.	In this any server-side technology can be used and it does not depend on the client.
It runs on the user's computer.	It runs on the webserver.
There are many advantages linked with this like faster. response times, a more interactive application.	The primary advantage is its 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 used in web development in which scripts run on the client's browser.	It is a technique that uses scripts on the webserver to produce a response that is customized for each client's request.
HTML, CSS, and javascript are used.	PHP, Python, Java, Ruby are used.
No need of interaction with the server.	It is all about interacting with the servers.
It reduces load on processing unit of the server.	It surge the processing load on the server.

12.In which location cookies are stored on the hard disk?

This depends on the user's browser and OS.

- In the case of Netscape on Windows OS, its stored in cookies.txt.
- In IE, each cookie is stored in a file and has is named as username@website.txt.

13.What's the difference between event.preventDefault() and event.stopPropagation() methods in JavaScript?

event.preventDefault() Method	event.stopPropagation() Method
Prevent the default action of browsers taking on that event.	Prevent further propagation of current events by parent or child elements.
It is a method present in the Event interface.	This method is also present in the Event interface.

For example, it prevents the browser from following a link.	It can not stop the default behavior of the browser.
Its syntax is -: **event.preventDefault();**	Its syntax is -: **event.stopPropagation();**
This method does not take any parameters	This method also does not take any arguments in its definition
Its supported browsers are -: chrome, firefox, safari, opera, etc	Its supported browsers are -: chrome, firefox, safari, opera, etc
It does not return value	It does not have any return type
Its uses the DOM version of DOM Level 3 Events	Its uses the DOM version of DOM Level 2 Events

14. How to set the cursor to wait in JavaScript?What is this [[[]]]?

We can directly use the **CSS** property to display the cursor as waiting but since we need the dynamic impact on showing the cursor, we would be doing this with the help of plain **JavaScript**.

Some common examples where waiting is applied on the cursor −

- While processing a payment, so the user does not initiate the cursor twice.
- While downloading a file so that no multiple files are downloaded for the same process.
- Whenever the user clicks the button it will tell the user to wait and will not let him execute any other action. We will use the addEventListener() function from JavaScript for achieving this functionality. With the help of this, we would be able to control the behavior of events like **click, hover**, etc.

- **#Filename: index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .box {
      display: flex;
      padding-top: 20px;
    }
    #btn {
      height: 50px;
      width: 100px;
      border-radius: 10px;
      background-color: gray;
      font-size: 1.1rem;
    }
  </style>
</head>
<body>
  <h1 style="color: green;">Welcome to Tutorials Point</h1>
  <p>The cursor will go into waiting on clicking this button.</p>
  <div class="box">
    <button id="btn">Click me</button>
  </div>
  <script>
    document.getElementById("btn")
      .addEventListener("click", function() {
        document.body.style.cursor = "progress";
```

```
            document.getElementById("btn")
              .style.backgroundColor = "gray";
            document.getElementById("btn")
              .style.cursor = "progress";
        });
    </script>
</body>
</html>
```

15.What is the difference between View state and Session state? ViewState	SessionState
Maintained at page level only.	Maintained at session level.
View state can only be visible from a single page and not multiple pages.	Session state value availability is across all pages available in a user session.
It will retain values in the event of a postback operation occurring.	In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration.
Information is stored on the client's end only.	Information is stored on the server.
used to allow persistence of page-instance-specific data.	used for the persistence of user-specific data on the server's end.
ViewState values are lost/cleared when new page is loaded.	SessionState can be cleared by programmer or user or in case of timeouts.

16.What are the pop-up boxes available in JavaScript?

javaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

window.alert("*sometext*");

The `window.alert()` method can be written without the window prefix.

Example

```
alert("I am an alert box!");
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

window.confirm("*sometext*");

The window.confirm() method can be written without the window prefix.

Example

```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
}
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

window.prompt("*sometext*","*defaultText*");

The window.prompt() method can be written without the window prefix.

Example

```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```

17.How to submit a form using JavaScript by clicking a link?

<!DOCTYPE html>

<html>

```
<body>
    <h2 style="color:green">GeeksforGeeks</h2>
    <b>Submit form details</b>

    <form id="form__submit" action="form.php" method="post">
        <label>NAME: </label><br />
        <input type="text" name="name" /><br />
        <label>AGE: </label><br />
        <input type="number" name="age" /><br />
        <label>CITY: </label><br />
        <input type="text" name="city" /><br /><br />
        <a href="#" onclick="submitForm()">Submit Here</a>
    </form>

    <script>
```

```
            function submitForm() {

                let form =
document.getElementById("form__submit");

                form.submit();

            }

        </script>

</body>



</html>
```

18. How to validate a form in JavaScript?

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
```

}

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php" onsubmit="return
validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

19.How to validate email in JavaScript?

```
function checkEmail() {

    var email = document.getElementById('txtEmail');

    var filter = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-
Z0-9]{2,4})+$/;

    if (!filter.test(email.value)) {

    alert('Please provide a valid email address');

    email.focus;

    return false;
```

}

}

- Call the function on Email textbox

20.What is the requirement of debugging in JavaScript?

In debugging, generally we set breakpoints to examine each line of code step by step. There is no requirement to perform this task manually in JavaScript.

JavaScript provides **debugger** keyword to set the breakpoint through the code itself. The **debugger** stops the execution of the program at the position it is applied. Now, we can start the flow of execution manually. If an exception occurs, the execution will stop again on that particular line.

- <script>
- x = 10;
- y = 15;
- z = x + y;
- debugger;
- document.write(z);
- document.write(a);
- </script>

Part-A

6.How to empty an array in JavaScript?

here are multiple ways to clear/empty an array in JavaScript. You need to use them based on the context. Let us look at each of them. Assume we have an array defined as −

```
let arr = [1, 'test', {}, 123.43];
```

Substituting with a new array −

```
arr = [];
```

Setting length prop to 0 –

```
arr.length = 0
```

Splice the whole array

```
arr.splice(0, arr.length)
```

7. How would you use a closure to create a private counter?.

A closure is a combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function.

Counter.js

```
// Global function which would form

// closure with modify function

function counter() {




// Private counter variable

let count = 0;




// To increment the value of counter
```

```
function increment() {
        count++;
}

// To decrement the value of counter
function decrement() {
        count--;
}

// Modify function forms closure
// here which is used outside
function modify(val) {

        // To check increment or decrement
```

```
        // button has been clicked
        if (val === "1") increment();
        else if (val === "0") decrement();

        // Return the counter
        return count;
}

// Returning to make it available
// outside counter function
return modify;
}

// Storing the closure modify
```

```
const closure = counter();



// This function handles the button

// click, objButton to get value

function counterHandler(objButton) {



// Storing the value return by modify

let count = closure(objButton.value);



// Getting div by it's id

// and modifying its inner html

document.getElementById("counter_div")

        .innerHTML = "<h2>" + count + "</h2>";

}
```

8. How does the this keyword work? Provide some code examples?

In JavaScript, the `this` keyword refers to an **object**.

Which object depends on how `this` is being invoked (used or called).

The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.
Alone, `this` refers to the **global object**.
In a function, `this` refers to the **global object**.
In a function, in strict mode, `this` is `undefined`.
In an event, `this` refers to the **element** that received the event.
Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

- How would you create a private variable in JavaScript?

Private Variables creation in functions: Whenever we deal with functions we always try to make variables private which then helps not to directly access variables further avoids updating these values too.

<script>

```
function carDetails() {
    let kms = 0;
    let speed = 0;
    let speedUp = (intialSpeed) => {
        speed += intialSpeed;
        kms += speed;
    };
    let totalkmsDriven = () => kms;
    return { speedUp, totalkmsDriven };
}

let car_object = carDetails();
car_object.speedUp(7);
car_object.speedUp(9);
```

```
        console.log(car_object.totalkmsDriven());



        // Undefined, since it is made private:

        console.log(car_object.kms);
</script>
```

10. Write a recursive function that performs a binary search

```
<script>
let recursiveFunction = function (arr, x, start, end) {



        // Base Condition

        if (start > end) return false;



        // Find the middle index

        let mid=Math.floor((start + end)/2);
```

```
        // Compare mid with given key x
        if (arr[mid]===x) return true;


        // If element at mid is greater than x,
        // search in the left half of mid
        if(arr[mid] > x)
                return recursiveFunction(arr, x, start, mid-1);
        else

                // If element at mid is smaller than x,
                // search in the right half of mid
                return recursiveFunction(arr, x, mid+1, end);
}

// Driver code
let arr = [1, 3, 5, 7, 8, 9];
```

```
let x = 5;



if (recursiveFunction(arr, x, 0, arr.length-1))

        document.write("Element found!<br>");

else document.write("Element not found!<br>");



x = 6;



if (recursiveFunction(arr, x, 0, arr.length-1))

        document.write("Element found!<br>");

else document.write("Element not found!<br>");

</script>
```