Assigment for day 1

Q1 Explain the use of javascript(or what you can do using a javascript)

Ans:-Javascript is used by programmers across the world to create dynamic and interactive web content like applications and browsers. JavaScript is so popular that it's the most used programming language in the world, used as a client-side programming language by 97.0% of all websites.Some of the most popular websites are built with JavaScript, including Google, YouTube, and Facebook.

- Building Applications
- Developing Engaging Games
- Creating Interactive Websites

Q2 what is the difference between client-side and server-side

Ans:-Client-side means that the processing takes place on the user's computer. It requires browsers to run the scripts on the client machine without involving any processing on the server.

Server-side means that the processing takes place on a web server.

This processing is important to execute the tasks required by the user on the web. Since the client-side script is executed on the client's computer, it is visible to the client. On the other hand, the server-side script is executed in the server; hence, it is not visible to the users.

Differences between client-side and server-side

Client-side

Does not need interaction with the server

Runs on the user's computer

Reduces load on the server's proccessing unit

Languages used: HTML, CSS, JavaScript

Server-side

Requires interaction with the server

Runs on the web server

Allows the server to provide dynamic websites tailored to the user. Increases the processing load on server.

Languages used: PHP, ASP.net, Python

Q3 What is nodejs

Ans:-Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Q4 Explain scope in javascript

Ans:-Scope determines the accessibility(visibility) of variables. javascript has 3 types of scope:

- Block scope
- function scope
- Global scope

Block scope:-ES6 introduced two important new JavaScript keywords: **let** and **const**.These two keywords provide Block Scope in JavaScript.Variables declared inside a { } block cannot be accessed from outside the block. for example:-

```
{
        let x = 2;          // x cannot be used here
}
```

Variables declared with the **var** keyword can not have block scope.Variables declared inside a { } block can be accessed from outside the block. for example:-

```
{
var x = 2;          // x can be used here
}
```

Local Scope:-Variables declared within a JavaScript function, become local to the function.

```
// code here cannot use carName

 function myFunction() {
        let carName = "Audi";
```

```
    // code here can use carName

        }

// code here cannot use carName
```

They can only be accessed from within the function.Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.Local variables are created when a function starts, and deleted when the function is completed.

Function Scope:-JavaScript has function scope: Each function creates a new scope.Variables defined inside a function are not accessible (visible) from outside the function.Variables declared with var, let and const are quite similar when declared inside a function.They all have **Function Scope:**

Global Scope:-Variables declared Globally (outside any function) have Global Scope.Global variables can be accessed from anywhere in a JavaScript program.Variables declared with **var, let and const** are quite similar when declared outside a block.They all have Global Scope:

Q5 Javascript is asynchronous or synchronous

Ans:-Synchronous JavaScript: As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

Let us understand this with the help of an example.

```
<script>
    document.write("Hi"); // First
    document.write("<br>");
      document.write("Mayukh") ;// Second
        document.write("<br>");
      document.write("How are you"); // Third
</script>
```

Output:-

Synchronous JavaScript: As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

Let us understand this with the help of an example.

```
<script>
        document.write("Hi"); // First
        document.write("<br>");
    document.write("Mayukh") ;// Second
        document.write("<br>");
          document.write("How are you"); // Third
</script>
```

Output:

Hii

Mayukh

How are you

In the above code snippet, the first line of the code Hi will be logged first then the second line Mayukh will be logged and then after its completion, the third line would be logged How are you.

So as we can see the codes work in a sequence. Every line of code waits for its previous one to get executed first and then it gets executed.

Asynchronous JavaScript: Asynchronous code allows the program to be executed immediately where the synchronous code will block further execution of the remaining code until it finishes the current one. This may not look like a big problem but when you see it in a bigger picture you realize that it may lead to delaying the User Interface.

Let us see the example how Asynchronous JavaScript runs.

```
<script>
        document.write("Hi");
        document.write("<br>");
    setTimeout(() => {
                document.write("Let us see what happens");
        }, 2000);
    document.write("<br>");
```

```
    document.write("End");

    document.write("<br>");
</script>
```

Output:

Hi

End

let see what happens

So, what the code does is first it logs in Hi then rather than executing the setTimeout function it logs in End and then it runs the setTimeout function.

At first, as usual, the Hi statement got logged in. As we use browsers to run JavaScript, there are the web APIs that handle these things for users. So, what JavaScript does is, it passes the setTimeout function in such web API and then we keep on running our code as usual. So it does not block the rest of the code from executing and after all the code its execution, it gets pushed to the call stack and then finally gets executed. This is what happens in asynchronous JavaScript.

Q6 javascript is single-threaded or multi-threaded

Ans:- The JavaScript within a chrome browser is implemented by V8 engine.

The V8 engine has two parts:

Memory Heap

Call Stack

Memory Heap: It is used to allocate the memory used by your JavaScript program. Remember memory heap is not the same as the heap data structures, they are totally different. It is the free space inside your OS.

Call Stack: Within the call stack, your JS code is read and gets executed line by line.

Now, JavaScript is a single-threaded language, which means it has only one call stack that is used to execute the program. The call stack is the same as the stack data structure that you might read in Data structures. As we know stacks are FILO that is First In Last Out. Similarly, within the call stack, whenever a line of code gets inside the call stack it gets executed and move out of the stack. In this way, JavaScript is a single-thread language because of only one call stack.

JavaScript is a single-threaded language because while running code on a single thread, it can be really easy to implement as we don’t have to deal with the complicated scenarios that arise in the multi-threaded environment like deadlock.

Since, JavaScript is a single-threaded language, it is synchronous in nature. Now, you will wonder that you have used async calls in JavaScript so is it possible then?

So, let me explain to you the concept of async call within JavaScript and how it is possible with single-threaded language. Before explaining it you let's discuss briefly why we require the async call or asynchronous calls. As we know within the synchronous calls, all the work is done line by line i.e. first one task is executed then the second task is executed, no matter how much time one task will take. This arises the problem of time wastage as well as resource wastage. These two problems are overcome by asynchronous calls, where one doesn't wait for the one call to complete instead it runs another task simultaneously. So, when we have to do things like image processing or making requests over the network like API calls, we use async calls.

Now, coming back to the previous question of how to use async call within JS. Within JS we have a lexical environment, syntax parser, an execution context (memory heap and call stack) that is used to execute the JS code. But except these browsers also have Event Loops, Callback queue, and WebAPIs that is also used to run the JS code. Although these are not part of JS it also helps to execute the JS properly as we sometimes used the browser functions within the JS.

As you can see in the above diagram, DOM, AJAX, and Timeout are not actually part of JavaScript but the part of RunTime Environment or browser, so these can be run asynchronously within the WebAPI using the callback queue and again put in the call stack using event loop to execute.

Let us take an example to be very clear of the concept. Suppose we have the following piece of code that we want to execute in the JS run-time environment.

Example:

```
<script>

  console.log('A');

        setTimeout(() => {

        console.log('B');

      }, 3000);

          console.log('C');

</script>
```

Output:

A C B

Let's see why this happens as JavaScript is a single-threaded language so, the output should be A B C but it is not.

When JS tries to execute the above program, it places the first statement in the call stack which gets executed and prints A in the console and it gets to pop out of the stack. Now, it places the second statement in the call stack and when it tries to execute the statement, it finds out that setTimeout() doesn't belong to JS so it pops out the function and puts in the WebAPI to get executed there. Since the call stack is now again empty, it places the third statement in the stack and executes it thus prints C in the console.

In the meanwhile, the WebAPI executes the timeout function and places the code in the callback queue. The eventloop checks if the call stack is empty or not or whether there is any statement in the callback queue that needs to be executed all the time. As soon as the event loop checks that the call stack is empty and there is something in the callback queue that needs to be executed, it places the statement in the call stack and the call stack executes the statement and prints B in the console of the browser

Q7 Explain DOM in your own word

Ans:-DOM stands for Document Object Model. It is a programming interface that allows us to create, change, or remove elements from the document. We can also add events to these elements to make our page more dynamic.The DOM views an HTML document as a tree of nodes. A node represents an HTML element.

```
<!DOCTYPE html>

<html lang="en">

   <head>

      <meta charset="UTF-8">

      <meta name="viewport" content="width=device-width, initial-scale=1.0">

      <meta http-equiv="X-UA-Compatible" content="ie=edge">

      <title>DOM tree structure</title>

   </head>

   <body>

      <h1>DOM tree structure</h1>

         <h2>Learn about the DOM</h2>

   </body>

</html>
```

Our document is called the root node and contains one child node which is the <html> element. The <html> element contains two children which are the <head> and <body> elements.Both the <head> and

<body> elements have children of their own.