**Assignment 4 – PPT Web Dev**

Q.1 What’s difference between Synchronous and Asynchronous?

The main difference between synchronous and asynchronous programming lies in how tasks or operations are executed and how they affect the flow of the program:

Synchronous: In synchronous programming, tasks are executed one at a time, in sequential order. Each task must complete before the next one can start. The program waits for each task to finish before moving on to the next line of code. This means that the execution is blocked or halted until a task completes.

console.log("Start");

fetchData(); // This task may take some time

console.log("End");

Asynchronous: In asynchronous programming, tasks can start, run, and complete independently of each other. The program does not wait for a task to complete before moving on to the next one. Asynchronous operations are typically non-blocking, allowing the program to perform other tasks concurrently or asynchronously.

console.log("Start");

fetchDataAsync(); // This task runs asynchronously

console.log("End");

Q.2 What are Web Apis ?

Web APIs (Application Programming Interfaces) are sets of rules and protocols that define how different software components can interact and communicate with each other over the web. They enable developers to access and utilize specific functionality and data from web services, libraries, frameworks, or platforms.

Q.3 Explain SetTimeOut and setInterval ?

Both `setTimeout` and `setInterval` are JavaScript functions used for scheduling the execution of code after a certain delay or at regular intervals.

1. setTimeout: The `setTimeout` function is used to execute a piece of code or a function after a specified delay (in milliseconds). It allows you to schedule a single execution.

setTimeout(() => {

console.log("Delayed code execution");

}, 2000); // Execute after 2000 milliseconds (2 seconds)

```

2. setInterval: The `setInterval` function is used to repeatedly execute a piece of code or a function at a specific interval. It allows you to schedule recurring executions.

setInterval(() => {

console.log("Repeated code execution");

}, 1000); // Execute every 1000 milliseconds (1 second)

Q.4 how can you handle Async code in JavaScript ?

To handle asynchronous code in JavaScript we can use:

1. callbacks: Pass a function as an argument to an asynchronous function, which gets executed once the operation completes.
2. promises: Promises represent the eventual completion or failure of an asynchronous operation. Chain .then() to handle successful completion and .catch() to handle errors.
3. async/await: The async keyword is used to define an asynchronous function, and the await keyword is used to pause the execution until a promise settles. This allows writing asynchronous code in a more synchronous-like manner.

Q.5 What are Callbacks & Callback Hell ?

Callbacks are functions that are passed as arguments to other functions and are executed once a certain task or operation is completed. They are commonly used in asynchronous programming to handle the result of an asynchronous operation.

Callback Hell refers to a situation where there are multiple levels of nested callbacks, making the code difficult to read, understand, and maintain. It occurs when asynchronous operations depend on the results of previous asynchronous operations, leading to a pyramid-like structure of callbacks.

Q.6 What are Promises & Explain Some Three Methods of Promise?

Promises are a feature in JavaScript that allow you to handle asynchronous operations in a more organized and structured way. A promise represents the eventual completion (or failure) of an asynchronous operation and allows you to chain multiple asynchronous operations together.

Promises have three states:

1. Pending: The initial state of a promise. It represents that the asynchronous operation is still in progress and hasn't been fulfilled or rejected yet.

2. Fulfilled: The state of a promise when the asynchronous operation is successfully completed. It means that the promised value is available and can be accessed.

3. Rejected: The state of a promise when the asynchronous operation encounters an error or fails. It indicates that the promised value could not be delivered as expected.

Promises have several methods to handle asynchronous operations:

1. .then(): The `.then()` method is used to handle the fulfillment of a promise. It takes two optional arguments: a callback function for the success case and a callback function for the failure case (commonly known as `.catch()`). It allows you to chain multiple `.then()` methods together to perform successive operations.

myAsyncFunction()

.then((result) => {

// Handle success

})

.catch((error) => {

// Handle error

});

2. .catch(): The `.catch()` method is used to handle the rejection of a promise. It is used to catch any errors that occur during the execution of the promise chain. It can be appended to the end of a `.then()` chain to handle any errors that occur in any previous steps.

myAsyncFunction()

.then((result) => {

// Handle success

})

.catch((error) => {

// Handle error

});

3. .finally(): The `.finally()` method is used to specify a callback function that should be executed regardless of whether the promise is fulfilled or rejected. It allows you to perform cleanup operations or actions that need to be done after the promise completes.

myAsyncFunction()

.then((result) => {

// Handle success

})

.catch((error) => {

// Handle error

})

.finally(() => {

// Cleanup or final actions

})

Q.7 What’s async & await Keyword in JavaScript?

The async keyword is used to define an asynchronous function. It allows the function to implicitly return a promise that resolves to the return value. Inside an async function, we can use the await keyword to pause the execution until a promise is settled.

The await keyword is used to pause the execution of an async function until a promise settles (either resolves or rejects). It can only be used inside an async function. The await expression waits for the completion of a promise and unwraps its resolved value.

Q.8 Explain Purpose of Try and Catch Block & Why do we need it?

The purpose of the try and catch blocks in JavaScript is to handle and manage exceptions or errors that may occur during the execution of code. The try block is used to enclose the code that might throw an exception, and the catch block is used to specify the code to be executed when an exception is thrown.

The main reasons why we need try and catch blocks are as follows:

* Error Handling
* Preventing Program Termination
* Graceful Recovery
* Error Logging

Q.9 Explain fetch.

fetch is a built-in web API in modern browsers that provides an easy and powerful way to make HTTP requests and interact with web servers. It allows us to send network requests and handle responses using promises, simplifying the process of retrieving data or making API calls in JavaScript.

The fetch function takes a URL as its first argument and returns a promise that resolves to the response from the server. It supports various options to configure the request, such as headers, request methods, and body data.

Q.10 How do you define an asynchronous function in JavaScript using async/await?

To define an asynchronous function using async/await in JavaScript, we prefix the function declaration with the async keyword. This indicates that the function will contain asynchronous code and will implicitly return a promise.

e.g-

async function myAsyncFunction() {

// Asynchronous code

}

Within an async function,we can use the await keyword to pause the execution of the function until a promise is settled. This allows you to write asynchronous code in a more sequential and synchronous-like manner.

e.g

async function fetchData() {

try {

const response = await fetch('https://api.example.com/data');

const data = await response.json();

console.log(data);

} catch (error) {

console.error(error);

}

}