**Assignment 5**

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

Synchronous and asynchronous are two different modes of communication or interaction in various systems, including computer programming, telecommunications, and general information exchange. Here's the difference between the two:

1. Synchronous: In synchronous communication or processing, the sender and receiver of information or the parties involved are required to be in sync or in direct coordination with each other. In this mode, the sender sends a request, and the sender waits for a response before proceeding further. The sender and receiver must both be available and active at the same time for communication to occur.
2. Asynchronous: In asynchronous communication or processing, the sender and receiver do not need to be active simultaneously. The sender can initiate a request or send a message, and then continue with other tasks without waiting for an immediate response. The receiver can process the request at their own pace and respond whenever they are ready. Asynchronous communication does not require the sender and receiver to be synchronized.

**Q.2 What are Web Apis ?**

Web APIs (Application Programming Interfaces) are sets of rules and protocols that allow different software applications to communicate and interact with each other over the internet. They enable applications to access and use the functionality, data, or services provided by another application or system.

Web APIs act as intermediaries, providing a standardized interface that allows developers to access and manipulate data or perform operations on remote servers or services. They define the methods, data formats, and rules for exchanging information between different software systems.

**Q.3 Explain SetTimeOut and setInterval ?**

setTimeout and setInterval are functions in JavaScript that allow you to execute code after a specific delay or at regular intervals, respectively. Here's an explanation of each:

1. setTimeout:

The setTimeout function is used to execute a specified piece of code or function after a specified delay (in milliseconds). It takes two parameters: a function or code to execute and a delay duration.

**Syntax:**

**setTimeout(function, delay);**

| **Example: console.log("Start"); setTimeout(function() {  console.log("Delayed code execution"); }, 2000); // Executes the function after a 2-second delay console.log("End"** |
| --- |
|  |

1. **setInterval:**

The setInterval function is used to repeatedly execute a specified piece of code or function at fixed intervals. It takes two parameters: a function or code to execute and a delay duration between each execution.

**Syntax:**

| **setInterval(function, delay);** |
| --- |

**Example:**

| **console.log("Start"); var counter = 0; var intervalId = setInterval(function() {  console.log("Interval code execution");  counter++;  if (counter === 5) {  clearInterval(intervalId); // Stops the interval after 5 executions  } }, 1000); // Executes the function every 1 second console.log("End");** |
| --- |

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

**In JavaScript, asynchronous code can be handled using various techniques to ensure smooth execution and proper handling of results or errors. Here are some commonly used approaches:**

**In JavaScript, asynchronous code can be handled using various techniques to ensure smooth execution and proper handling of results or errors. Here are some commonly used approaches:**

**Callbacks:**

**Callbacks are a traditional way of handling asynchronous code in JavaScript. A callback is a function that is passed as an argument to an asynchronous function. When the asynchronous operation completes, the callback function is invoked with the result or error.**

**Example:**

| **function fetchData(callback) {  setTimeout(function() {  const data = "Some data";  if (data) {  callback(null, data); // Pass null as the error and data } else {  callback("Error occurred", null); // Pass error as the first argument  }  }, 2000); }  function handleData(error, result) {  if (error) {  console.error("Error:", error);  } else {  console.log("Result:", result);  } }  fetchData(handleData);** |
| --- |

**2. Promises:**

Promises provide a more structured and readable way of handling asynchronous code. A promise represents the eventual completion or failure of an asynchronous operation and allows chaining of operations using then and catch methods.

**Example:**

| **function fetchData() {  return new Promise(function(resolve, reject) {  setTimeout(function() {  const data = "Some data";  if (data) {  resolve(data); // Resolve the promise with data  } else {  reject("Error occurred"); // Reject the promise with an error  }  }, 2000);  }); }  fetchData()  .then(function(result) {  console.log("Result:", result);  })  .catch(function(error) {  console.error("Error:", error);  });** |
| --- |

**Q.5 What are Callbacks & Callback Hell ?**

Callbacks are functions that are passed as arguments to other functions and are executed later when a particular operation or task is completed. They allow you to control the flow of execution in asynchronous code and handle the result or error once an operation is finished.

Callbacks are commonly used in JavaScript to handle asynchronous operations such as making API requests, reading files, or executing database queries. They are executed after the asynchronous operation completes, and they receive the result or error as parameters**.**

**Here's an example of a callback function used with the setTimeout function:**

**function greet(name, callback) {**

| **setTimeout(function() {  callback("Hello, " + name + "!");  }, 2000); } function displayGreeting(greeting) {  console.log(greeting); } greet("John", displayGreeting);** |
| --- |

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

**Promises are objects in JavaScript that represent the eventual completion (fulfillment or rejection) of an asynchronous operation. They provide a cleaner and more structured way of handling asynchronous code compared to callbacks. Promises have three states: pending, fulfilled, or rejected.**

**Here are three important methods of promises:**

1. then(): The then() method is used to handle the fulfillment or rejection of a promise. It takes two optional callback functions as arguments: onFulfilled and onRejected. The onFulfilled callback is executed when the promise is fulfilled, and it receives the fulfillment value. The onRejected callback is executed when the promise is rejected, and it receives the reason for rejection.

**Syntax:**

**promise.then(onFulfilled, onRejected);**

1. catch(): The catch() method is used to handle promise rejections. It is a shorthand for handling only the rejection without handling the fulfillment. It is typically appended at the end of promise chains to catch any errors that occur throughout the chain.

**Syntax:**

**promise.catch(onRejected);**

1. finally(): The finally() method is used to specify a callback function that is executed regardless of whether the promise is fulfilled or rejected. It allows you to perform cleanup operations or execute code that needs to run in both cases.

**Syntax:**

**promise.finally(onFinally);**

1. These three methods, then(), catch(), and finally(), provide powerful capabilities for handling the outcome of promises and allow you to write more readable and maintainable asynchronous code.

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

The async and await keywords are features introduced in ECMAScript 2017 (ES8) to simplify asynchronous programming in JavaScript. They provide a more synchronous-like syntax for working with promises, making asynchronous code easier to read and write.

1. **async Keyword:**

The async keyword is used to declare an asynchronous function. It can be placed before a function declaration or function expression. When a function is marked as async, it always returns a promise, which allows you to use await within the function.

**Example:**

| async function fetchData() {  // Asynchronous code using await  const result = await someAsyncOperation();  return result; } |
| --- |

1. **await Keyword:**

The await keyword can only be used within an async function. It is used to pause the execution of the function until a promise is resolved, and it allows you to retrieve the fulfilled value of the promise. While waiting for the promise to settle, the await expression does not block the entire JavaScript execution, allowing other tasks to continue**.**

**Example:**

| async function fetchData() {  const result = await someAsyncOperation();  return result; } |
| --- |

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

The purpose of the try and catch block in JavaScript is to handle and manage potential errors or exceptions that may occur during the execution of a block of code. The try block contains the code that might throw an error, and the catch block is responsible for handling and responding to the error if it occurs.

Here's why we need the try and catch block:

Error Handling: The try and catch block provides a structured way to handle errors and prevent them from causing the program to crash or behave unexpectedly. It allows you to gracefully handle errors and perform appropriate actions based on the type of error encountered.

Robustness: By implementing error handling using try and catch, you can make your code more robust and resilient. It enables you to anticipate and handle potential issues, ensuring that your code can recover or gracefully degrade in the face of errors.

Debugging: The try and catch block helps with debugging by providing a mechanism to capture and inspect errors. When an error occurs, the catch block is executed, allowing you to log or display relevant error messages, stack traces, or perform any necessary debugging tasks.

Graceful Failure: With error handling, you can provide a fallback or alternative behavior when an error occurs. Instead of crashing or stopping the execution, you can handle the error gracefully, display meaningful error messages to users, and continue with the execution of the program in a controlled manner.

**Example:**

| try {  // Code that might throw an error  const result = someOperation();  console.log(result); } catch (error) {  // Error handling  console.error("An error occurred:", error);  // Perform any necessary fallback or error recovery actions } |
| --- |

**Q.9 Explain fetch**

fetch is a built-in JavaScript function used to make HTTP requests and fetch resources from a server. It provides a modern and flexible way to handle network requests in web applications. The fetch function returns a promise that resolves to the response from the server.

Here's a basic syntax for using the fetch function:

| fetch(url)  .then(response => {  // Handle the response  })  .catch(error => {  // Handle any errors  }); |
| --- |

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

To define an asynchronous function in JavaScript using the async and await keywords, you simply need to prefix the function declaration or function expression with the async keyword. This allows you to use the await keyword within the function to pause the execution and wait for asynchronous operations to complete.

Here's the basic syntax for defining an asynchronous function using async/await:

| async function functionName() {  // Asynchronous code using await } |
| --- |

Here's an example of an asynchronous function that uses async/await to fetch data from an API:

| 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:', error);  } } |
| --- |

In this example, the fetchData function is defined as an asynchronous function using the async keyword. Inside the function, the await keyword is used to pause the execution and wait for the completion of asynchronous operations.

The await keyword is used before the fetch() function to wait for the response to be resolved, and then it is used again before response.json() to wait for the JSON parsing to complete. By using await, the code inside the function appears to be executing synchronously, enhancing its readability and maintainability.