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

Ans:- Synchronous refers to events or processes that occur in a coordinated and time-dependent manner, where one task waits for another to complete before proceeding. In contrast, asynchronous refers to events or processes that can occur independently and concurrently, without requiring immediate coordination or waiting for each other to finish.

Q.2 What are Web Apis ?

Ans:- 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 define a set of methods and data formats that enable the exchange of information between systems. Web APIs provide a standardized way for developers to access and manipulate data or functionality from remote servers or services, enabling the integration of different applications, platforms, and technologies.

Q.3 Explain SetTimeOut and setInterval ?

Ans:- Both **setTimeout** and **setInterval** are functions provided by JavaScript that allow you to execute code after a certain delay. Here's an explanation of each:

1. **setTimeout**: This function is used to execute a specific block of code once after a specified delay. It takes two parameters: a function or code snippet to be executed, and the delay time in milliseconds. After the specified delay, the code inside the function is executed.
2. **setInterval**: This function is used to execute a specific block of code repeatedly at a fixed interval. It also takes two parameters:

a function or code snippet to be executed, and the interval time in milliseconds. The code inside the function will be executed repeatedly at the specified interval until **clearInterval** is called.

In summary, **setTimeout** is used for executing code once after a delay, while **setInterval** is used for executing code repeatedly at fixed intervals until explicitly stopped.

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

Ans:- In JavaScript, asynchronous code can be handled using various techniques. Here are a few commonly used methods:

1. Callbacks: Callback functions are a traditional way to handle asynchronous operations in JavaScript. A callback function is passed as an argument to an asynchronous function and is executed once the operation is complete. This allows you to specify the code that should run after the asynchronous operation finishes.
2. Promises: Promises provide a more structured approach to handle asynchronous code. A promise represents the eventual completion or failure of an asynchronous operation and allows you to chain operations and handle success or error cases using **.then()** and **.catch()** methods.
3. Async/await: Async/await is a modern approach introduced in ECMAScript 2017 (ES8) that simplifies asynchronous code by using keywords **async** and **await**. The **async** keyword is used to define an asynchronous function, and the **await** keyword is used to pause the execution of the function until the awaited promise is resolved. This provides a more synchronous-looking code structure while still executing asynchronous operations.

Q.5 What are Callbacks & Callback Hell ?

Callbacks are functions that are passed as arguments to other functions and are executed at a certain point or when a specific event occurs. They are commonly used in asynchronous programming to handle the result of an asynchronous operation.

Callback Hell refers to a situation where multiple asynchronous operations are nested within each other, resulting in deeply nested and hard-to-read code. This occurs when callbacks are used to handle asynchronous operations sequentially, leading to a pyramid-like structure of callbacks. It can make the code difficult to understand, maintain, and debug. It often arises when working with multiple asynchronous functions that depend on the results of each other, leading to a "pyramid" of nested callbacks.

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

Ans:- Promises are objects in JavaScript that represent the eventual completion (or failure) of an asynchronous operation. They provide a more structured way to handle asynchronous code compared to callbacks. Promises have three states: pending, fulfilled, or rejected.

Here are three methods commonly used with promises:

1. **then()**: The **then()** method is used to handle the fulfillment of a promise. It takes two optional callbacks as arguments: **onFulfilled** and **onRejected**. The **onFulfilled** callback is executed when the promise is fulfilled (resolved successfully), and it receives the resolved value as an argument. The **onRejected** callback is executed when the promise is rejected, and it receives the rejection reason as an argument.
2. **catch()**: The **catch()** method is used to handle promise rejections. It is equivalent to calling **then(null, onRejected)**,

allowing you to specify a callback function to handle any errors or rejections that occur during the promise chain.

3. **finally()**: The **finally()** method allows you to specify a callback that is executed regardless of whether the promise is fulfilled or rejected. This is useful for cleanup or finalization tasks that need to be performed after the completion of a promise, regardless of its outcome.

Q.7 What's async & await Keyword in JavaScript

Ans:- The **async** and **await** keywords in JavaScript are used together to simplify the handling of asynchronous code and make it look more like synchronous code.

- **async**: The **async** keyword is used to define an asynchronous function. It allows the function to use the **await** keyword inside it. An async function always returns a promise, and any value returned from the async function is automatically wrapped in a resolved promise.
- **await**: The **await** keyword is used inside an async function to pause the execution of the function until a promise is resolved or rejected. It can only be used within an async function. When encountering an **await** expression, the async function waits for the promise to settle and then resumes its execution, either with the resolved value or by throwing the rejection reason.

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

Ans:-

The purpose of the **try** and **catch** block in JavaScript is to handle and manage exceptions or errors that occur during the execution of code.

It provides a structured way to handle potential errors and prevents the code from abruptly terminating.

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

1. Error Handling: The **try** block allows you to enclose the code that might throw an exception. If an exception occurs within the **try** block, it is caught by the corresponding **catch** block. This enables you to handle the error gracefully, perform appropriate actions, and prevent the application from crashing.
2. Graceful Degradation: In situations where an error occurs, the **catch** block allows you to execute alternative logic or fallback mechanisms. It helps ensure that even if certain parts of the code fail, the application can continue to function or provide a meaningful response to the user.
3. Error Reporting: The **catch** block provides an opportunity to log or report the error details. This information is valuable for debugging and troubleshooting, as it helps developers identify and understand the cause of the error.

Q.9 Explain fetch

Ans:- **fetch** is a built-in JavaScript function used for making HTTP requests to fetch resources from a network. It provides a modern and flexible way to perform asynchronous network requests in browsers and Node.js.

Key points about **fetch**:

1. HTTP Requests: **fetch** allows you to send HTTP requests, such as GET, POST, PUT, DELETE, etc., to a specified URL.
2. Promises: **fetch** returns a Promise that resolves to the Response object representing the response to the request. This allows

you to use Promise-based techniques like **.then()** and **.catch()** to handle the response asynchronously.

3. Headers and Body: You can customize the request by providing options, including headers, request method, request body, and more.
4. Handling Response: The Response object provides methods to access and process the response data, such as **.json()** to parse JSON response, **.text()** to retrieve the response as text, and **.blob()** to handle binary data.

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

Ans:- To define an asynchronous function using **async/await** in JavaScript, you prefix the function declaration with the **async** keyword. This indicates that the function will contain asynchronous operations and will return a Promise.

In summary, by using the **async** keyword, you can define an asynchronous function in JavaScript and leverage **await** to pause the function's execution until asynchronous operations complete