Answer 1

Synchronous JavaScript

1. Instruction in synchronous code executes in a given sequence.
2. Each operation waits for the previous operation to complete its execution.
3. Most of the time JavaScript is used as Synchronous code.

Asynchronous JavaScript

1. Instructions in asynchronous code can execute in parallel.
2. Next operation can occur while the previous operation is still getting processed.
3. Asynchronous JavaScript is preferred in situations in which execution gets blocked indefinitely.

Answer 2

Application Programming Interfaces (APIs) are constructs made available in programming languages to allow developers to create complex functionality more easily. They abstract more complex code away from you, providing some easier syntax to use in its place.

As a real-world example, think about the electricity supply in your house, apartment, or other dwellings. If you want to use an appliance in your house, you plug it into a plug socket and it works. You don't try to wire it directly into the power supply — to do so would be really inefficient and, if you are not an electrician, difficult and dangerous to attempt.

APIs in client-side JavaScript:

Browser APIs

Third party APIs

Answer 3:

We may decide to execute a function not right now, but at a certain time later. That’s called “scheduling a call”.

There are two methods for it:

setTimeout allows us to run a function once after the interval of time.

setInterval allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

Eg of setTimeout

```
function sayHi() {
  alert('Hello');
}
setTimeout(sayHi, 1000);
```

Eg of setInterval

```
// repeat with the interval of 2 seconds

let timerId = setInterval(() => alert('tick'), 2000);

// after 5 seconds stop

setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
```

Answer 4

In JavaScript, there are two common ways to work with asynchronous operations: then/catch method chaining and async/await. Both methods can be used to handle promises, which are objects that represent the eventual completion (or failure) of an asynchronous operation.

```
const fetchAPI = async function(country1,country2,country3){
    try{
        const res1 = await fetch(`https://restcountries.com/v3.1/name/${country1}`)
        const res2 = await fetch(`https://restcountries.com/v3.1/name/${country2}`)
        const res3 = await fetch(`https://restcountries.com/v3.1/name/${country3}`)


        const data1 = await res1.json()
        const data2 = await res2.json()
        const data3 = await res3.json()
        console.log(data1[0].capital[0]);
        console.log(data2[0].capital[0]);
        console.log(data3[0].capital[0]);
        return "Done with fetchAPI"
    } catch(err){
        console.log(err);
        throw new Error("Custom Error")
    }
}


fetchAPI("canada", "germany", "russia")
```

Answer 5

Callback: A callback is a function that is passed as an argument to another function that executes the callback based on the result. They are basically functions that are executed only after a result is produced. Callbacks are an important part of asynchronous JavaScript.

Callback Hell: Callback Hell is essentially nested callbacks stacked below one another forming a pyramid structure. Every callback depends/waits for the previous callback, thereby making a pyramid structure that affects the readability and maintainability of the code.

Answer 6

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value. A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

JavaScript then() method

The then() method is used with the callback when the promise is successfully fulfilled or resolved.

The syntax of then() method is:

promiseObject.then(onFulfilled, onRejected);

JavaScript catch() method

The catch() method is used with the callback when the promise is rejected or if an error occurs. For example,

```
// returns a promise
let countValue = new Promise(function (resolve, reject) {
  reject('Promise rejected');
});

// executes when promise is resolved successfully
countValue.then(
   function successValue(result) {
      console.log(result);
   },
 )

// executes if there is an error
.catch(
   function errorValue(result) {
      console.log(result);
   }
);
```

JavaScript finally() method

You can also use the finally() method with promises. The finally() method gets executed when the promise is either resolved successfully or rejected. For example,

```
// returns a promise

let countValue = new Promise(function (resolve, reject) {

   // could be resolved or rejected

   resolve('Promise resolved');

});


// add other blocks of code

countValue.finally(

   function greet() {

      console.log('This code is executed.');

   }

);
```

Answer 7

Async: It simply allows us to write promises-based code as if it was synchronous and it checks that we are not breaking the execution thread. It operates asynchronously via the event loop. Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value.

Await: Await function is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result. It only makes the async block wait.

Answer 8

JavaScript try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements try and catch come in pairs:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will throw an exception (throw an error).

Answer 9

The fetch() method in JavaScript is used to request data from a server. The request can be of any type of API that returns the data in JSON or XML. The fetch() method requires one parameter, the URL to request, and returns a promise.

The fetch() method starts the process of fetching a resource from a server.

The fetch() method returns a Promise that resolves to a Response object.

fetch(file)

Answer 10

The async function declaration creates a binding of a new async function to a given name. The await keyword is permitted within the function body, enabling asynchronous, promise-based behavior to be written in a cleaner style and avoiding the need to explicitly configure promise chains.

Eg

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // Expected output: "resolved"
}

asyncCall();
```