ANSWER 7-

The async and await keywords in JavaScript are used to handle asynchronous operations in a more synchronous-like manner. They provide a more readable and straightforward way to work with Promises and make asynchronous code appear and behave more like synchronous code.

async: The async keyword is used to define an asynchronous function. When a function is marked as async, it means that it will always return a Promise. Inside an async function, you can use the await keyword to pause the execution of the function until a Promise is resolved or rejected. The function will automatically wrap the returned value in a resolved Promise or the thrown error in a rejected Promise. Example:

```
async function fetchData() {
  // Asynchronous operation
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();

  return data;
}

fetchData().then(function(result) {
  console.log(result); // Handle the fetched data
});
```

In this example, the fetchData function is marked as async. Inside the function, the await keyword is used to pause the execution until the fetch Promise resolves and the response is received. Then, the await keyword is used again to pause the execution until the Promise returned by response.json() resolves and the data is parsed. Finally,

the function returns the data, which is wrapped in a resolved Promise.

await: The await keyword is used inside an async function to pause the execution until a Promise is fulfilled or rejected. It can only be used inside an async function. When the await keyword is used with a Promise, it "waits" for the Promise to settle and then returns the resolved value or throws an error if the Promise is rejected. The use of await allows you to write asynchronous code that looks and behaves more like synchronous code. Example:

```
async function fetchData() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();


  return data;
}
```

In this example, the await keyword is used to pause the execution of the fetchData function until the fetch Promise resolves. It waits for the response to be received before moving on to the next line of code.

Using async and await together helps simplify and improve the readability of asynchronous code by avoiding nested callbacks and providing a more sequential and synchronous-like flow of execution.