async

In the context of async/await in JavaScript, the async keyword is used to define an asynchronous function. An asynchronous function is a function that can perform asynchronous operations and can use the await keyword to pause its execution while waiting for asynchronous operations to complete.

Here's the meaning of the async keyword:

Asynchronous Function Declaration

When you declare a function using the async keyword, you're indicating that the function contains asynchronous code and may use the await keyword inside it.

Implicit Promise

An async function always returns a Promise. The Promise resolves to the value returned by the function, or it's rejected with an error if an exception is thrown within the function.

Sequential-Looking Code

The use of async and await allows you to write asynchronous code that appears more similar to traditional synchronous code. This makes your code more readable and easier to reason about.

await

In JavaScript, the await keyword is used in conjunction with the async keyword to handle asynchronous operations in a more readable and sequential manner. It's typically used inside an async function and is used to pause the execution of the function until a Promise is resolved. This allows you to write asynchronous code that looks and behaves more like synchronous code.

Here's what the await keyword does:

Pauses Execution

When you use await before a Promise inside an async function, the execution of that function is paused at that point. This means that the function will stop executing until the awaited Promise is resolved or rejected.

Waiting for Promise Resolution

While the function is paused, JavaScript's event loop continues to process other tasks. When the awaited Promise is resolved, the execution of the async function resumes from where it was paused.

Capturing Promise Result

The value that the Promise resolves to is returned from the await expression. This allows you to capture the result of the asynchronous operation and use it directly in your code.

```
const getUser = (id)=>
{
  return new Promise((resolve, reject)=>
  {
    setTimeout(()=>
    {
      const user = { id, name : `User ${id}`};
      resolve(user);
    }, 2000)
  });
}

const getRepos = (username)=>
{
  return new Promise((resolve, reject)=>
  {
    setTimeout(()=>
    {
      const repos = { name :  username, repos: ["repo1", " repo2", " repo3", "
repo4"] };
      resolve(repos);
    }, 2000)
  });
}


const getCommits = (repo)=>
{
  return new Promise((resolve, reject)=>
  {
    setTimeout(()=>
    {
      const commits = { repo :  repo, commits: ["commit 1", " commit 2", "
commit 3", " commit 4"] };
      resolve(commits);
    }, 2000)
  });
}
```

```javascript
const fetchCommits = async (id) =>
{
  try
  {
    const user = await getUser(id);
    const repos = await getRepos(user.name);
    const commits = await getCommits(repos.repos[0]);
    console.log(`commits are ${commits.commits.toString()}`);
  }
  catch (error)
  {
    console.log(error.message)
  }
}


console.log("Before");

fetchCommits(1);

console.log("After");
```