Synchronous(Blocking) vs Asynchronous(Non Blocking)

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

In this program when the first line executes, the program is blocking and second line has to wait until first line finishes its execution.

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

setTimeout(()=>
{
  console.log("Reading a user from database");
}, 2000);

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

On the other hand, asynchronous code does not block the execution of next line.

JavaScript is single-threaded, meaning it can only execute one piece of code at a time. Asynchronous operations allow you to perform tasks without blocking the main thread, which keeps the application responsive. Asynchronous operations are typically managed using mechanisms like callbacks, Promises, and async/await.

There are three patterns to handle asynchronous code.

1. Callbacks
2. Promises
3. Async-Await

Callbacks

```
const getUser = (id, cb)=>
{
  setTimeout(()=>
  {
    console.log("Reading a user from the database ...");
    const user = { id, name : `User ${id}`};
    cb(user);
  }, 2000)
}
console.log("Before");

getUser(1, (user)=>
{
  console.log(user);
});

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

Callback Hell

```
const getUser = (id, cb)=>
{
  setTimeout(()=>
  {
    const user = { id, name : `User ${id}`};
    cb(user);
  }, 2000)
}

const getRepos = (username, cb)=>
{
  setTimeout(()=>
  {
    const repos = { name :  username, repos: ["repo1", " repo2", " repo3",
" repo4"] };
    cb(repos);
  }, 2000)
}

const getCommits = (repo, cb)=>
{
  setTimeout(()=>
  {
    const commits = { repo :  repo, commits: ["commit 1", " commit 2", "
commit 3", " commit 4"] };
    cb(commits);
  }, 2000)
}


console.log("Before");

getUser(1, (user)=>
{
  getRepos(user.name, (repos)=>
  {
    getCommits(repos.repos[0], (commits)=>
    {
      console.log(`commits are ${commits.commits.toString()}`);
    })
  })
});

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

Problems with callbacks

Callbacks can lead to callback hell or "pyramid of doom" when dealing with multiple asynchronous operations. This can make the code hard to read and maintain. To address these challenges, modern JavaScript introduced Promises and async/await.