What is Asynchronous Programming? | JavaScript Tutorial In Hindi #34

In today's tutorial, we will explore the concept of asynchronous programming in JavaScript, and see why it is important, and how we can use it to effectively handle potential blocking operations such as fetching resources from a server. Asynchronous programming is a fairly advanced JavaScript topic, and you are advised to watch the previous tutorials first before going through this.

As we know that the JavaScript is single-threaded and can only handle one operation at a time. As there is a single execution thread for our program to run, a question arises that how to execute a long-running operation without blocking the thread of execution? Well, welcome to asynchronous programming.

What is the difference between synchronous and asynchronous programming?

In ***synchronous*** **programming**, one thing happens at a time. When we call a function that performs a long-running action, it returns a result when the action has finished. This stops the program for the time the action takes. In contrast, ***asynchronous*** **programming** allows multiple things to happen at the same time. When we start an action, the program continues to run. When the action finishes, the program is informed and gets the result.

Let's compare synchronous and asynchronous programming using an example: a program that fetches two resources from the network and then combines results.

In synchronous programming, where the request function returns only after it has done its work. To perform this task, we make the requests one after the other. Here the drawback is that the second request will be started only when the first has finished. Suppose the time taken by the first request is 12 seconds, and the time taken by the second request is 13 seconds, so the total time taken will be the sum of the two response times.

In asynchronous programming, the functions that perform a slow action takes an extra argument, a *callback function*. The action is started, and when it finishes, the callback function is called with the result.

For example, the setTimeout function waits a given number of milliseconds (a second is a thousand milliseconds) and then calls a function.

```
setTimeout(() => console.log("Tick"), 500);
```

Waiting is useful when doing something like updating an animation or checking whether something takes longer than a given amount of time.
Performing multiple asynchronous actions in a row using callbacks means that we have to keep passing new functions to handle the program's continuation after the actions.

Summary:-

Asynchronous code does not have to wait; the program can continue to run. The asynchronous programming makes it possible to express waiting for long-running actions without stopping the program during these actions. JavaScript usually implements this style of programming using callbacks.
Asynchronous programming is made easier by using promises, objects that represent actions that might complete in the future, and async functions, which allow us to write an asynchronous program as if it were synchronous.

Code website.html as described/written in the video

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <div class="container">
        <h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>
        <div id="myfirst" class="child red good" id="first">child 1

            <ul class="this" id='myul'>
                <li class="childul" id='fui'>this</li>
                <li class="childul">is</li>
                <li class="childul">a</li>
                <li class="childul">list </li>
                <li class="childul" id='lui'>of my dreams</li>
        </div>
        <div class="child">child 2</div>
        <div class="child red">child 3</div>
        <div class="child">child 4</div>
        <form action="none.html" method="post">
            <a href="//codewithharry.com">Go to Code With Harry</a>
            <br>
```

```
            <br>
            Search this website: <input type="text" name="Hello" id="">
            <button id="btn">Submit form</button>
            <!-- <input type="button" id='btn' value="submit"> -->
        </form>
    </div>
    <br>
    <div class="no">this is a dummy div1</div>
    <div class="no">this is a dummy div2</div>
    <div class="no">this is a dummy div3</div>
</body>
<!-- <script src="js/tut12.js"></script> -->
<!-- <script src="js/tut14.js"></script> -->
<!-- <script src="js/tut15.js"></script> -->
<!-- <script src="js/tut16.js"></script> -->
<!-- <script src="js/tut17.js"></script> -->
<!-- <script src="js/tut18.js"></script> -->
<!-- <script src="js/tut20.js"></script> -->
<!-- <script src="js/tut21.js"></script> -->
<!-- <script src="js/tut23.js"></script> -->
<!-- <script src="js/tut24.js"></script> -->
<!-- <script src="js/tut25.js"></script> -->
<!-- <script src="js/tut27.js"></script> -->
<!-- <script src="js/tut28.js"></script> -->
<!-- <script src="js/tut30.js"></script> -->
<!-- <script src="js/tut31.js"></script> -->
<!-- <script src="js/tut32.js"></script> -->
<script src="js/tut34.js"></script>
</html>
```

Code tut34.js as described/written in the video

Copy

```
console.log("This is tutorial 34");

setTimeout(() => {
    for (let index = 0; index < 4005; index++) {
        const element = index;
        console.log("This is index number" + index);
```

```
    }
}, 100);


console.log('done printing');
```

Previous Next

CodeWithHarry