SYNCHRONOUS AND ASYNCHRONOUS PROGRAMMING

Synchronous and asynchronous programming are two fundamental approaches to handling tasks and operations. They relate to how the code execution and flow are managed when dealing with potentially time-consuming operations like fetching data, making network requests, or performing other I/O operations.

Synchronous Programming:

In synchronous programming, tasks are executed one after the other in a sequential manner. Each task must wait for the previous one to complete before it can start. This can lead to "blocking" behavior, where the entire program or application pauses until a particular task finishes. Synchronous code is easier to read and reason about because the execution order is straightforward. However, it can become inefficient when dealing with operations that take a significant amount of time, as it can lead to unresponsiveness.

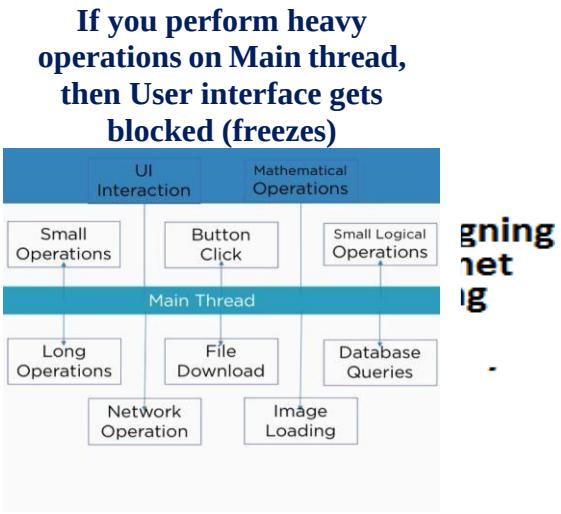
For an example Imagine you're at a fast-food restaurant, and you're the only customer. The cashier takes your order, prepares your food, and gives it to you. Everything happens step by step, and you have to wait for each task to finish before the next one starts. This is like synchronous programming. It does one thing at a time, and you have to wait for each task to complete before moving on to the next.

e.g.

```
function syncExample() {
    console.log("Task 1");
    console.log("Task 2");
    console.log("Task 3");
}
syncExample();
```

Asynchronous Programming:

In asynchronous programming, tasks are initiated and executed independently of the main program flow. This allows multiple tasks to be performed concurrently without waiting for each other to complete. Asynchronous operations are often used when dealing with tasks that might take time, like making ***network requests or reading files.*** JavaScript provides mechanisms like **callbacks, promises, and async/await to manage asynchronous operations**. These mechanisms help prevent blocking and improve the overall responsiveness of applications.



e.g.

```
function asyncExample() {
    console.log("Task 1");
    setTimeout(function() {
        console.log("Task 2 (after a delay)");
    }, 2000);
    console.log("Task 3");
```

```
}
asyncExample()
```

Asynchronous programming is crucial in scenarios where responsiveness and performance are important, such as web applications that need to handle multiple user interactions simultaneously. It allows non-blocking execution and improves user experience.

CALL BACK

A callback function in JavaScript is a function that is passed as an argument to another function and is intended to be executed at a later time or after a specific event occurs. Callback functions are commonly used in asynchronous programming to handle tasks that might take some time to complete, such as fetching data from a server, reading a file, or waiting for a user action.

e.g.

```
<script>
   function one(call_two) {
      console.log(" step 1 complete. please call step 2");
      call_two()
   }
   function two() {
      console.log(" step 2 ");
   }
   one(two)
</script>
```

Akhilesh Kumar Gupta
(Technical Training specialist)
TechCarrel LLP