**Synchronous, Asynchronous And Condition Based Asynchronous**

***Java script is a single threaded language which compiles one task at time***

**Synchronous (Blocking) Code in TypeScript:**

Synchronous code in TypeScript executes tasks sequentially, one after the other. It can block the execution of subsequent code until a task is completed. Here's an example:

**function synchronousExample() {**

**console.log("Start");**

**for (let i = 0; i < 3; i++) {**

**console.log(`Task ${i}`);**

**}**

**console.log("End");**

**}**

**synchronousExample();**

In this example, the code inside **synchronousExample** runs sequentially, and each task completes before moving on to the next one. The output will be:

**Asynchronous (Non-Blocking) Code in TypeScript:**

Asynchronous code in TypeScript allows you to execute tasks concurrently or without blocking the execution of subsequent code. You commonly use mechanisms like callbacks, promises, and async/await to work with asynchronous operations. Here's an example using Promises:

function asynchronousExample() {

console.log("Start");

**const promise1 = new Promise((resolve) => {**

**setTimeout(() => {**

**console.log("Task 1");**

**resolve();**

**}, 1000);**

**});**

**const promise2 = new Promise((resolve) => {**

**setTimeout(() => {**

**console.log("Task 2");**

**resolve();**

**}, 500);**

**});**

**Promise.all([promise1, promise2]).then(() => {**

**console.log("End");**

**});**

**}**

**asynchronousExample();**

**two asynchronous tasks (represented by Promises) are executed concurrently.**

In TypeScript, you can also use **async** and **await** to write asynchronous code in a more synchronous-looking style while still achieving non-blocking behavior