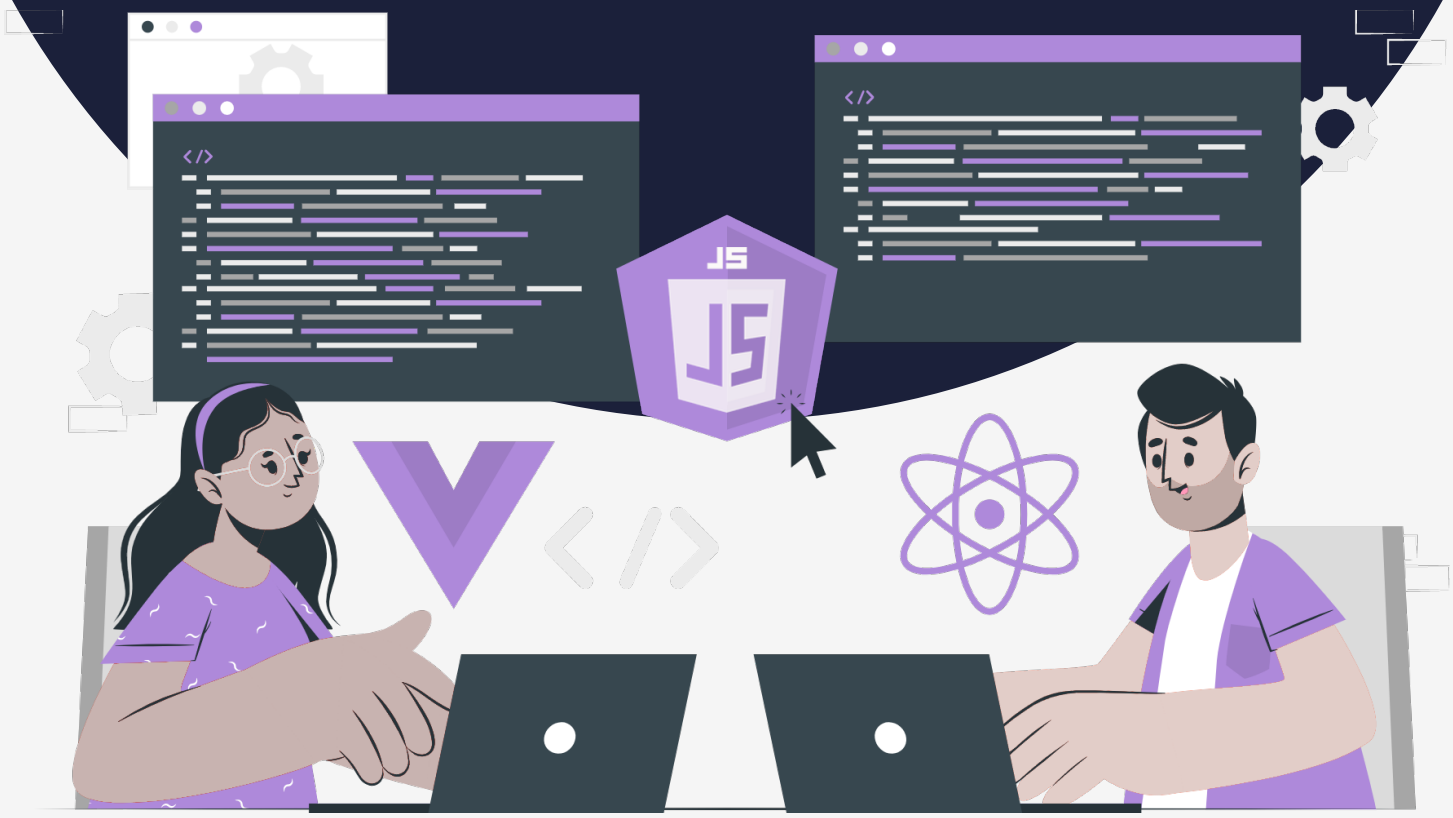
Lesson:

Single Thread



Topics Covered:

1. Introduction to threads.
2. Classification of programming languages based on threads.
3. Javascript is a single-threaded programming language.
4. Is being single-threaded a limitation in Javascript?
5. Is Javascript slow?

Imagine a company that needs to process customer orders. One person could be responsible for receiving the order and entering it into the system, while another could prepare the order for shipping. These tasks could be performed independently, and each person would be a separate "thread" of work.

In the same way in programming, a thread refers to a sequence of instructions that a computer can execute independently from other sequences of instructions in the same program. Think of a thread as a single unit of work or a single task that a computer can perform.

Threads are important because they allow programs to perform multiple tasks at the same time, which can improve the program's performance and responsiveness. For example, a web browser might use multiple threads to download web pages, render graphics, and respond to user input, all at the same time.

In the above-mentioned example, if the company only had one person to handle all the orders, the process would be slower and less efficient. But by having multiple people working on different aspects of the same task simultaneously, the company can process more orders in less time.

Similarly, in programming, using multiple threads to perform different tasks can improve the efficiency and speed of a program. However, it's important to manage these threads carefully to avoid issues.

Classification of programming languages based on threads.

Programming languages can be categorized as single-threaded or multi-threaded, depending on how they handle concurrent execution.

A single-threaded programming language, as the name suggests, can only execute one sequence of instructions at a time. This means that the program can only perform one task at a time, and any other tasks must wait until the first is complete. This can limit the performance and responsiveness of the program, particularly for tasks that involve long computations or I/O operations.

Examples of single-threaded programming languages include JavaScript, PHP, and Ruby.

On the other hand, a multi-threaded programming language can execute multiple threads simultaneously, allowing the program to perform multiple tasks concurrently. This can improve performance and responsiveness for tasks that can be parallelized, such as graphics rendering or data processing.

Examples of multi-threaded programming languages include C++, Java, and Python.

Javascript is a single-threaded programming language.

JavaScript is primarily a single-threaded programming language, meaning that it can only execute one sequence of instructions at a time.

This is because the JavaScript engine, which is responsible for interpreting and executing JavaScript code, uses a single call stack to manage the sequence of instructions that need to be executed. We will look into call stack in upcoming lectures.

Is being single-threaded a limitation in Javascript?

Being single-threaded is both a limitation and a benefit for JavaScript, depending on the situation and the alternatives available.

As a limitation, a single-threaded language can only execute one piece of code at a time. To avoid this, JavaScript supports asynchronous programming through features like callbacks, promises, and async/await. These features allow for non-blocking I/O operations, which means that the JavaScript engine can continue executing other instructions while waiting for I/O operations to complete. This can help to avoid blocking the main thread and keep the application responsive.

The benefit of being single-threaded is that it simplifies the programming model, making it easier to reason about program behavior and preventing complex errors that can occur in multi-threaded applications. JavaScript's event loop, which manages the execution of code, provides a simple and efficient way to handle asynchronous operations, such as I/O, without the need for explicit threading.

Event Loop.

By now we all know that JavaScript is a single-threaded programming language, which means that it can only execute one task at a time. However, it can handle asynchronous events using the event loop. The event loop is a mechanism that allows JavaScript to execute multiple tasks simultaneously without blocking the main thread.

Before understanding the event loop, let’s first have a look at the event queue. The event queue is a data structure that stores events in the order they are triggered. When an event is triggered, it is added to the end of the queue. The event loop continuously checks the event queue for new events and processes them one at a time.

Now that we know what is the event loop, let’s have a clear understanding of the event loop. The event loop is a mechanism that handles asynchronous events in JavaScript. It works by continuously checking the event queue for new events and processing them one at a time. When an event is triggered, it is added to the event queue, and the event loop begins to process the next event in the queue. This allows JavaScript to handle multiple events simultaneously without blocking the main thread.

There are multiple characters involved in the process like call stack which we will be discussing in upcoming lectures.

Let's now look at an example demonstrating an event loop.

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

setTimeout(() => {
  console.log("setTimeout");
}, 0);

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

In this example, we have three statements:

1. console.log('start'): This statement logs the string "start" to the console.
2. setTimeout(() => {console.log('setTimeout');}, 0): This statement sets a timer using the setTimeout function with a delay of 0 milliseconds. When the timer expires, the callback function passed as an argument is executed, which logs the string "setTimeout" to the console.
3. console.log('end'): This statement logs the string "end" to the console.

When this code is executed, the output is:

```
start
end
setTimeout
```

The flow of the event loop is as follows:

1. The first statement console.log('start') is executed, and the string "start" is logged to the console.
2. The second statement setTimeout(() => {console.log('setTimeout');}, 0) sets a timer with a delay of 0 milliseconds and registers a callback function to be executed when the timer expires. Since the delay is 0 milliseconds, the callback function is added to the event queue immediately.
3. The third statement console.log('end') is executed, and the string "end" is logged to the console.
4. The event loop continuously checks the event queue for new events. Since there is an event in the queue (the callback function registered by the setTimeout statement), it is executed, and the string "setTimeout" is logged to the console.

Event loop allows JavaScript to handle asynchronous tasks such as timers without blocking the main thread. The setTimeout function is used to simulate an asynchronous task, and the callback function is executed after all the synchronous tasks have been completed.

Is Javascript slow?

No, being single-threaded does not necessarily mean that JavaScript is slow. Single-threaded simply means that JavaScript can only execute one task at a time, but it does not decide the speed of the language.

In fact, JavaScript can be quite fast due to modern JavaScript engines, such as V8 used in Chrome and Node.js, which have advanced optimization techniques. These techniques allow JavaScript code to be executed at decent speeds, which is why JavaScript has become a popular language for both client-side and server-side applications.