JS asynchronous behaviour, event loop, callback queue and more:

The event loop and callback queue are concepts that are closely associated with the runtime environment in which JavaScript code is executed. They are not part of the core JavaScript language itself, but they are fundamental to the way asynchronous operations are managed in many JavaScript runtime environments, such as web browsers and Node.js.

Event Loop:
The event loop is a mechanism that manages the execution of code in an event-driven environment. It is a part of the runtime environment (like web browsers or Node.js) rather than being a part of the JavaScript language specification itself. The event loop's primary purpose is to continuously monitor the call stack and the callback queue, ensuring that asynchronous tasks are executed at the appropriate times.

Callback Queue:
Similarly, the callback queue is a construct used in the event-driven model of JavaScript runtime environments. When asynchronous tasks (like timers or network requests) complete, their associated callback functions are placed in the callback queue. These callbacks are executed by the event loop when the main execution stack is empty.

Both the event loop and callback queue play a crucial role in handling asynchronous code and ensuring that JavaScript remains non-blocking and responsive while waiting for asynchronous tasks to complete. They are integral to the runtime environment's implementation of JavaScript's concurrency model.

In summary, while the core JavaScript language does not define the event loop and callback queue, these concepts are vital components of many JavaScript runtime environments and are responsible for managing asynchronous operations and enabling the event-driven programming paradigm.

JavaScript Runtime Model Based on an Event Loop:
- JavaScript operates in a unique way when it comes to executing code and handling tasks. It uses something called an "event loop" to manage these tasks.
- The event loop is like a traffic controller that keeps an eye on things and makes sure everything runs smoothly. It handles executing code, dealing with events, and taking care of small tasks that need to be done.

Executing the Code:
- When you write JavaScript code, it needs to be executed so that the computer knows what to do. The event loop takes care of running this code step by step, making sure things happen in the right order.

Collecting and Processing Events:
- In a program, events are things like mouse clicks, keyboard presses, or other actions. The event loop watches out for these events and responds to them. For example, when you click a button on a web page, the event loop notices and triggers the appropriate action.

Executing Queued Sub-tasks:
- Sometimes, while the event loop is busy doing one thing, other small tasks (sub-tasks) might be waiting to be done. The event loop takes care of these tasks one by one, making sure nothing is left undone.

Difference from Other Languages:
- This way of handling things is different from how some other programming languages like C or Java work. They have their own ways of managing tasks and events, but JavaScript's event loop model is unique to how it operates.

Runtime vs. Core Language:
- The JavaScript runtime includes all the tools and mechanisms that make JavaScript work outside of just writing code. This includes the event loop, Web APIs (like timers and network requests), and other environment-specific features.
- The core JavaScript language is the set of instructions, keywords, and syntax rules that you use to write your code. It's the language itself without considering the specific environment in which it's running.

In summary, the JavaScript runtime is responsible for managing how JavaScript code gets executed and how it interacts with the environment, while the core JavaScript language is what you use to write your actual code. The event loop is a crucial part of the runtime that ensures things happen in the right order and handles various tasks and events.