```
console.clear();

/*

=====================================================

     Structure of the Event Loop in Javascript

=====================================================

The Event Loop is a fundamental mechanism in JavaScript that enables asynchronous behavior in a single-threaded environment. It ensures that code runs non-blocking, even when handling tasks like API calls, timers or user interactions.

Key Components of the Event Loop:

1. Call Stack

A stack where synchronous functions are pushed and executed one at a time. Functions are removed from the stack once their execution completes.

2. Microtask Queue

A queue for asynchronous tasks like .then() from Promises and MutationObserver callbacks. These tasks are executed immediately after the call stack is empty – before any macrotasks.

3. Macrotask Queue (Callback Queue / Task Queue)

A queue for tasks such as: setTimeout, setInterval setImmediate (Node.js) fetch callbacks (after the microtask that resolves them). Macrotasks are executed after all microtasks are cleared.

🔁 How the Event Loop Works:

JavaScript starts by executing all synchronous code in the call stack.

Once the call stack is empty:

It first checks the microtask queue and executes all microtasks.

```

```
After the microtask queue is empty:

It picks the next task from the macrotask queue and executes it.

This process repeats continuously — this cycle is called the event loop.

*/

// This message logs immediately during the synchronous phase of execution.

console.log("Waiter Order Received...");

// setTimeout is a Web API function that schedules a callback function to run after a delay (2000ms).

// This is added to the Web API environment and will be queued into the macrotask queue after 2 seconds.

setTimeout(() => {

    console.log("Pizza is ready.");  // Will run after the current call stack is empty and delay has passed.

}, 2000);

// `fetch` sends a request to an API endpoint. It returns a Promise and once resolved `.then()` is scheduled as a microtask (Promises are microtasks).

fetch("https://dummyjson.com/products/1")

    .then(() => console.log("API Response Received."));

// This `Promise.resolve()` resolves immediately and schedules its `.then()` callback in the microtask queue.

Promise.resolve().then(() => console.log("Quick Billing Done."));

// This is also a synchronous operation, so it runs immediately after the first console.log.

console.log("Serving Water...");
```