Microtasks vs macrotasks

🔄 Microtasks vs Macrotasks in JavaScript — In Detail

In JavaScript, understanding **microtasks** and **macrotasks** is crucial for mastering asynchronous behavior, especially when dealing with `Promises`, `async/await`, `setTimeout`, and more.

🧠 What Are Tasks in JavaScript?

JavaScript is **single-threaded**, but it can handle asynchronous operations using the **Event Loop**.

The Event Loop manages:

- 📦 **Macrotasks** (a.k.a. tasks)
- 🔬 **Microtasks** (a.k.a. jobs or microtask queue)

📦 Macrotasks

Macrotasks include:

- `setTimeout`
- `setInterval`
- `setImmediate` (Node.js)
- I/O operations (e.g., file reads)
- UI rendering tasks

They are scheduled in the **macrotask queue** and executed **one at a time** in each event loop tick.

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

🔬 Microtasks

Microtasks include:

- `.then()` callbacks from `Promises`
- `async/await` operations
- `queueMicrotask()`
- `MutationObserver` callbacks

They are scheduled in the **microtask queue** and are executed **immediately after the currently executing script finishes and before any macrotask**.

```
Promise.resolve().then(() => {
  console.log("Microtask: Promise.then");
});
```

🔁 Event Loop Execution Order

In One Event Loop Tick:

1. 🧠 Execute currently running code (main thread)
2. 🔬 Run **all** microtasks in the microtask queue
3. 🧱 Run **one** macrotask from the macrotask queue
4. 🔁 Repeat

🔍 Example: Microtasks vs Macrotasks

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

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

Promise.resolve().then(() => {
```

```
  console.log("Microtask: Promise.then");
});

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

🔽 Output:

```
Script start
Script end
Microtask: Promise.then
Macrotask: setTimeout
```

👉 Even though `setTimeout` is set to 0 ms, the **Promise resolves first** because **microtasks run before macrotasks**.

🛠️ queueMicrotask()

You can explicitly queue a microtask like this:

```
queueMicrotask(() => {
  console.log("Microtask: via queueMicrotask()");
});
```

This is helpful when you want something to run *after* the current synchronous block, but *before* any macrotask.

🧪 Advanced Example

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

setTimeout(() => {
  console.log("setTimeout 1");
}, 0);

Promise.resolve().then(() => {
```

```
  console.log("Promise 1");

  queueMicrotask(() => {
    console.log("queueMicrotask inside Promise");
  });
});

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

Output:

```
Start
End
Promise 1
queueMicrotask inside Promise
setTimeout 1
```

🧬 Summary Table

Feature	Type	Runs When
`setTimeout`	Macrotask	In the next event loop tick
`setInterval`	Macrotask	Repeatedly on event loop ticks
`Promise.then`	Microtask	After current code, before macrotask
`async/await`	Microtask	Same as Promise
`queueMicrotask`	Microtask	Immediately after current call

🧠 Why It Matters

- Avoids unexpected execution order
- Helps with debugging async code
- Critical in frameworks (e.g., React batch updates, DOM mutations)

If you'd like a live coding example or want to test it interactively, I can help create a CodePen or JSFiddle demo.