🧠 Q1. Event Loop Reordering

```
console.log(1);
setTimeout(() => console.log(2));
Promise.resolve().then(() => console.log(3));
Promise.resolve().then(() => setTimeout(() => console.log(4)));
console.log(5);
```

> 🤔 *What's the exact output order and why?*

🧠 Q2. Nested Microtasks Starvation

```
Promise.resolve().then(function loop() {
  console.log('Microtask');
  Promise.resolve().then(loop);
});
setTimeout(() => console.log('Timeout'), 0);
```

> 🤔 *Will `"Timeout"` ever get printed? Why or why not?*

🧠 Q3. setInterval and Overlapping Microtasks

```
let i = 0;
const id = setInterval(() => {
  console.log(i);
  Promise.resolve().then(() => console.log('P' + i));
  if (++i === 3) clearInterval(id);
}, 0);
```

> 🤔 *What's the full output with ordering?*

🧠 Q4. Interleaved setTimeout & Promises Inside Interval

```
let count = 0;
setInterval(() => {
  console.log('Interval Start');
  setTimeout(() => console.log('Timeout'), 0);
  Promise.resolve().then(() => console.log('Promise'));
  if (++count === 2) process.exit();
}, 0);
```

> 🤔 *List all printed lines and explain the precise order.*

🧠 Q5. Nested Promises with setTimeout

```
setTimeout(() => {
  console.log(1);
  Promise.resolve().then(() => {
    console.log(2);
    setTimeout(() => console.log(3), 0);
```

```
  });
}, 0);
```

> 🧠 *What will be printed and in what order?*

🔹 Q6. Promise-Timeout-Promise Chain

```
Promise.resolve().then(() => {
  console.log('A');
  setTimeout(() => {
    console.log('B');
    Promise.resolve().then(() => console.log('C'));
  }, 0);
});
```

> 🧠 *Explain the internal queueing and output.*

🔹 Q7. Long Microtask Queue Blocking Timers

```
for (let i = 0; i < 100000; i++) {
  Promise.resolve().then(() => {});
}
setTimeout(() => console.log('Done'), 0);
```

> 🧠 *When does `"Done"` get printed, and why is it delayed?*

🔹 Q8. Chained Promises and Timer Placement

```
setTimeout(() => console.log('T1'));
Promise.resolve().then(() => console.log('P1'))
  .then(() => setTimeout(() => console.log('T2'), 0))
  .then(() => console.log('P2'));
```

> 🧠 *What logs and in what order?*

🔹 Q9. Clear Timeout After Promise Inside Interval

```
let id;
let i = 0;
id = setInterval(() => {
  Promise.resolve().then(() => {
    if (i++ === 1) clearInterval(id);
    console.log('Tick');
  });
}, 0);
```

> 🧠 *How many times does `"Tick"` log?*

🔹 Q10. Synchronous Loop Blocking Everything

```
setTimeout(() => console.log('Timeout'), 0);
for (let i = 0; i < 1e9; i++) {}
console.log('After loop');
```

> 🤔 *Why is `"Timeout"` delayed? When does it run?*

🔹 Q11. Resolved Promise Inside Timeout

```
setTimeout(() => {
  console.log('A');
  Promise.resolve().then(() => console.log('B'));
}, 0);
console.log('C');
```

> 🤔 *Output and internal event loop stages?*

🔹 Q12. Promise Inside Promise Inside setTimeout

```
setTimeout(() => {
  console.log('X');
  Promise.resolve().then(() => {
    Promise.resolve().then(() => {
      console.log('Y');
    });
    console.log('Z');
  });
});
```

> 🤔 *Output order?*

🔹 Q13. 0ms vs 100ms Timers and Promises

```
setTimeout(() => console.log('A'), 0);
setTimeout(() => console.log('B'), 100);
Promise.resolve().then(() => console.log('C'));
```

> 🤔 *What's the output if the environment is Node.js vs browser?*

🔹 Q14. Interval Drift Detection

```
let start = Date.now();
let count = 0;
const id = setInterval(() => {
  let drift = Date.now() - start - count * 10;
  console.log(`Drift: ${drift}ms`);
  if (++count > 5) clearInterval(id);
}, 10);
```

> 🤔 *Why does drift accumulate? Explain JS single-threaded behavior.*

🔹 Q15. Cancel Timer Before It Fires

```
let id = setTimeout(() => console.log('Boom'), 0);
clearTimeout(id);
Promise.resolve().then(() => console.log('Safe'));
```

 Does `"Boom"` ever print?

 Q16. Infinite Loop Inside setTimeout

```
setTimeout(() => {
  while (true) {}
}, 0);
console.log('End');
```

 What happens to the browser tab or Node process?

 Q17. setTimeout in Nested Microtasks

```
Promise.resolve().then(() => {
  console.log('A');
  return Promise.resolve().then(() => {
    console.log('B');
    setTimeout(() => console.log('C'), 0);
  });
});
```

 Output order?

 Q18. setInterval Starvation via while Loop

```
setInterval(() => console.log('Tick'), 0);
while (true) {}
```

 What do you observe in console?

 Q19. setTimeout in Resolved Promise and Vice Versa

```
Promise.resolve().then(() => {
  setTimeout(() => console.log('Timeout'));
});
setTimeout(() => {
  Promise.resolve().then(() => console.log('Promise'));
});
```

 Who wins the race: `"Timeout"` or `"Promise"`?

 Q20. Clear Timeout Inside Its Own Execution

```
const id = setTimeout(() => {
  console.log('Hi');
  clearTimeout(id);
}, 0);
```

Q21. The “0ms Timeout” Trap

```
console.log('A');
setTimeout(() => console.log('B'), 0);
Promise.resolve().then(() => console.log('C'));
setTimeout(() => console.log('D'), 0);
console.log('E');
```

Predict **the exact log order** and **why**.

Q22. Promise Chain + Timeout Explosion

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

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

Promise.resolve()
  .then(() => {
    console.log('p1');
    setTimeout(() => console.log('p2-timeout'), 0);
  })
  .then(() => {
    console.log('p3');
  });

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

Explain **why `p2-timeout` logs last**.

Q23. The Interval in the Microtask Maze

```
let i = 0;
const id = setInterval(() => {
  console.log(`Interval: ${i}`);
  Promise.resolve().then(() => console.log(`Microtask: ${i}`));
  if (++i >= 3) clearInterval(id);
}, 0);
```

Predict the full **log sequence**.

Q24. The Recursive Monster

```
function run() {
  console.log('Run Start');
  Promise.resolve().then(() => console.log('Promise Inside Run'));
  setTimeout(run, 0);
}
```

```
run();
```

 Describe **how many times** the promise logs if left running for 5 seconds.

Q25. Multiple Intervals with Conditional Clear

```
let count = 0;
const id = setInterval(() => {
  console.log('First Interval');
  if (++count === 2) clearInterval(id);
}, 0);

setInterval(() => {
  console.log('Second Interval');
}, 0);
```

 Predict the log order **up to 5 lines**.

Q26. The Microtask Infinite Loop? Or Not?

```
let i = 0;

function task() {
  if (i < 3) {
    Promise.resolve().then(() => {
      console.log('Micro', i);
      i++;
      task();
    });
  }
}

task();
```

 Will this code **block** the event loop? Why or why not?

Q27. Interval + Timeout = Mind Bender

```
setInterval(() => console.log('Interval'), 0);
setTimeout(() => console.log('Timeout'), 0);
```

 Will `Interval` or `Timeout` **always** run first? Why?

Q28. setTimeout in a Promise Chain

```
Promise.resolve().then(() => {
  console.log('First');
  setTimeout(() => console.log('Timeout'), 0);
});
```

```
Promise.resolve().then(() => console.log('Second'));
```

 Predict the log sequence.

Q29. Nested Timeouts in a Promise

```
Promise.resolve().then(() => {
  console.log('P1');
  setTimeout(() => {
    console.log('T1');
    setTimeout(() => console.log('T2'), 0);
  }, 0);
});
```

 Predict the output and explain the **timing**.

Q30. Interval Race Condition

```
let a = 0;
let b = 0;

setInterval(() => {
  console.log('A', a++);
}, 0);

setInterval(() => {
  console.log('B', b++);
}, 0);
```

 Can you predict if A or B **will ever consistently run before the other**?

Q31. setTimeout Starvation

```
while (true) {
  Promise.resolve().then(() => {});
}
```

 What happens to `setTimeout` scheduled in the same script? Explain why.

Q32. Clear Timeout After Trigger

```
let id = setTimeout(() => console.log('Boom'), 0);
clearTimeout(id);
```

 Will `Boom` ever log? What if the event loop is **blocked** after `clearTimeout`?

Q33. Interleaving Promises and Intervals

```
setInterval(() => console.log('I1'), 0);
Promise.resolve().then(() => {
  console.log('P1');
  setInterval(() => console.log('I2'), 0);
});
```

 What’s the log order, and will the intervals **compete**?

Q34. setTimeout Inside setInterval

```
let count = 0;
setInterval(() => {
  console.log('Interval');
  setTimeout(() => console.log('Timeout'), 0);
  if (++count > 2) process.exit();
}, 0);
```

 Predict logs for **2 iterations**.

Q35. setInterval Inside Promise

```
Promise.resolve().then(() => {
  console.log('P');
  setInterval(() => console.log('Interval'), 0);
});
```

 Will `Interval` ever run **before** `P`? Why?

Q36. Delayed ClearInterval

```
const id = setInterval(() => console.log('Running'), 0);
Promise.resolve().then(() => {
  setTimeout(() => clearInterval(id), 0);
});
```

 Predict how many times `Running` might log.

Q37. The Misleading Timeout Zero

```
setTimeout(() => console.log('Zero'), 0);
setTimeout(() => console.log('Zero2'), 0);
console.log('End');
```

 Are both `Zero` and `Zero2` **guaranteed** to run in order?

Q38. Promise Chain + Blocking Operation

```
Promise.resolve().then(() => console.log('Microtask'));
setTimeout(() => console.log('Timeout'), 0);
```

```
for (let i = 0; i < 1e9; i++) {}
```

 What happens, and why does the output **order matter**?

Q39. Self-Clearing Interval

```
const id = setInterval(() => {
  console.log('Tick');
  clearInterval(id);
}, 0);
```

 How many times does `Tick` log? Why?

Q40. Infinite Promises with Timeouts

```
function spam() {
  Promise.resolve().then(() => {
    console.log('Spam');
    setTimeout(spam, 0);
  });
}

spam();
```

 What happens? Will this **crash** your browser or Node.js process? Why or why not?