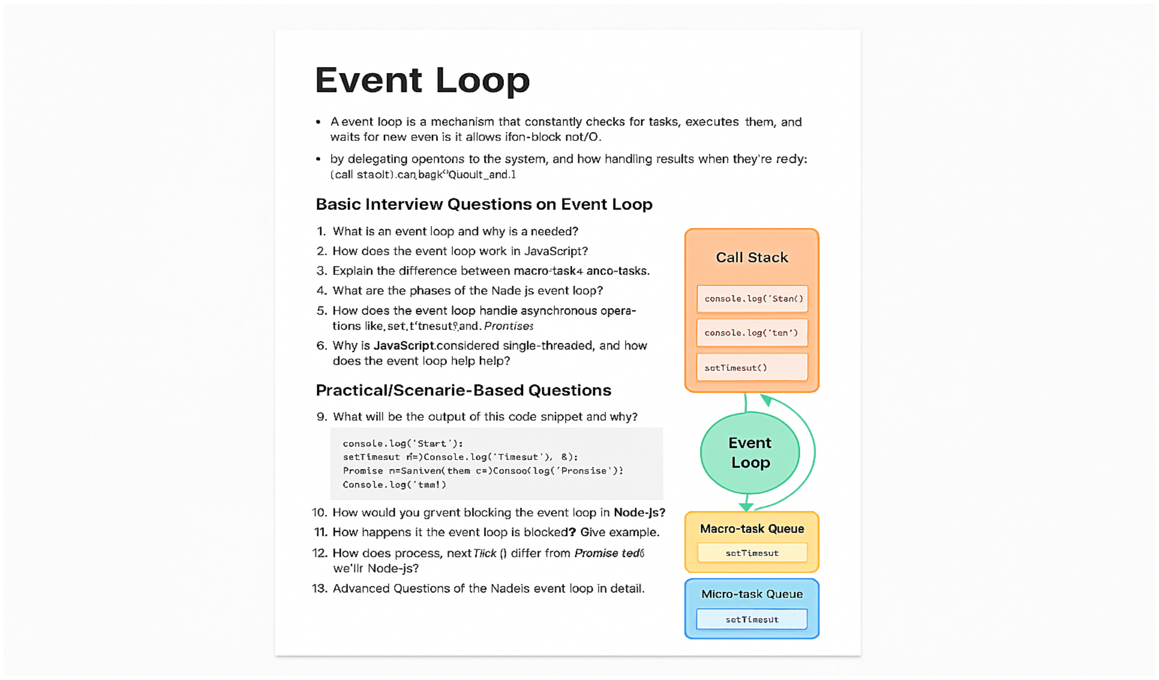
JavaScript Event Loop: Complete Interview Guide

Event Loop Cheat Sheet

- JavaScript is single-threaded and synchronous by default.
- Event Loop enables asynchronous behavior using queues.
- Queues:
* Macro-task Queue: setTimeout, setInterval, I/O callbacks.
* Micro-task Queue: Promise.then, process.nextTick.
- Execution Order:
1. Synchronous code.
2. All micro-tasks.
3. Next macro-task.



Event Loop

- A event loop is a mechanism that constantly checks for tasks, executes them, and waits for new even is it allows ifon-block not/O.
- by delegating opentons to the system, and how handling results when they're redy: (call staolt).can,bagk°Quoult_and.1

Basic Interview Questions on Event Loop

1. What is an event loop and why is a needed?
2. How does the event loop work in JavaScript?
3. Explain the difference between macro-task4 anco-tasks.
4. What are the phases of the Nade js event loop?
5. How does the event loop handie asynchronous operations like,set,t'tnesut?,and. *Prontises*
6. Why is **JavaScript**.considered single-threaded, and how does the event loop help help?

Practical/Scenarie-Based Questions

9. What will be the output of this code snippet and why?

```
console.log('Start'):
setTimesut n=)Console.log('Timesut'), 8):
Promise n=Saniven(them c=)Consoo(log('Pronsise'):
Console.log('tmm!)
```

10. How would you grvent blocking the event loop in **Node-Js?**
11. How happens it the event loop is blocked**?** Give example.
12. How does process, next *Tlick* () differ from *Promise ted6* we'llr Node-js?
13. Advanced Questions of the Nadeis event loop in detail.

10 Advanced Event Loop Questions with Explanations

1. Async/Await with setTimeout

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

async function demo() {
console.log('Inside async function');
await new Promise(resolve => setTimeout(resolve, 0));
console.log('After await');
}

demo();

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

Output: Start, Inside async function, End, After await. Explanation: await pauses inside async function, promise resolves after timeout.

2. Nested Promises and setTimeout

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

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

Promise.resolve().then(() => {
console.log('Promise 1');
Promise.resolve().then(() => console.log('Promise 2'));
});

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

Output: A, B, Promise 1, Promise 2, Timeout. Explanation: Micro-tasks run before macro-tasks.

3. process.nextTick vs Promise (Node.js)

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

process.nextTick(() => console.log('NextTick'));
Promise.resolve().then(() => console.log('Promise'));

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

Output: Start, End, NextTick, Promise. Explanation: nextTick runs before micro-tasks in Node.js.

4. Multiple async functions

```
async function first() {
console.log('First start');
await second();
console.log('First end');
}

async function second() {
console.log('Second start');
await Promise.resolve();
console.log('Second end');
}
```

```
first();
console.log('Global end');
```
Output: First start, Second start, Global end, Second end, First end.

5. Mixed setTimeout and async

```
setTimeout(() => console.log('Timeout 1'), 0);

Promise.resolve().then(() => {
console.log('Promise 1');
setTimeout(() => console.log('Timeout 2'), 0);
});

console.log('Sync');
```
Output: Sync, Promise 1, Timeout 1, Timeout 2.

6. Function with Promise and setTimeout

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

function demo() {
console.log('Inside function');
new Promise(resolve => {
setTimeout(resolve, 100);
console.log('promise');
});
console.log('After promise');
}

demo();

console.log('End');
```
Output: Start, Inside function, promise, After promise, End.

7. Async/Await chain

```
async function test() {
console.log('Step 1');
await Promise.resolve();
console.log('Step 2');
}

test();
console.log('Outside');
```
Output: Step 1, Outside, Step 2.

8. Nested setTimeout

```
setTimeout(() => {
console.log('Timeout 1');
setTimeout(() => console.log('Timeout 2'), 0);
}, 0);

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

Output: Sync, Timeout 1, Timeout 2.

9. Promise inside setTimeout

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

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

Output: Start, Timeout, Promise inside Timeout.

10. Async with multiple awaits

```
async function chain() {
console.log('First');
await Promise.resolve();
console.log('Second');
await Promise.resolve();
console.log('Third');
}

chain();
console.log('Outside');
```

Output: First, Outside, Second, Third.