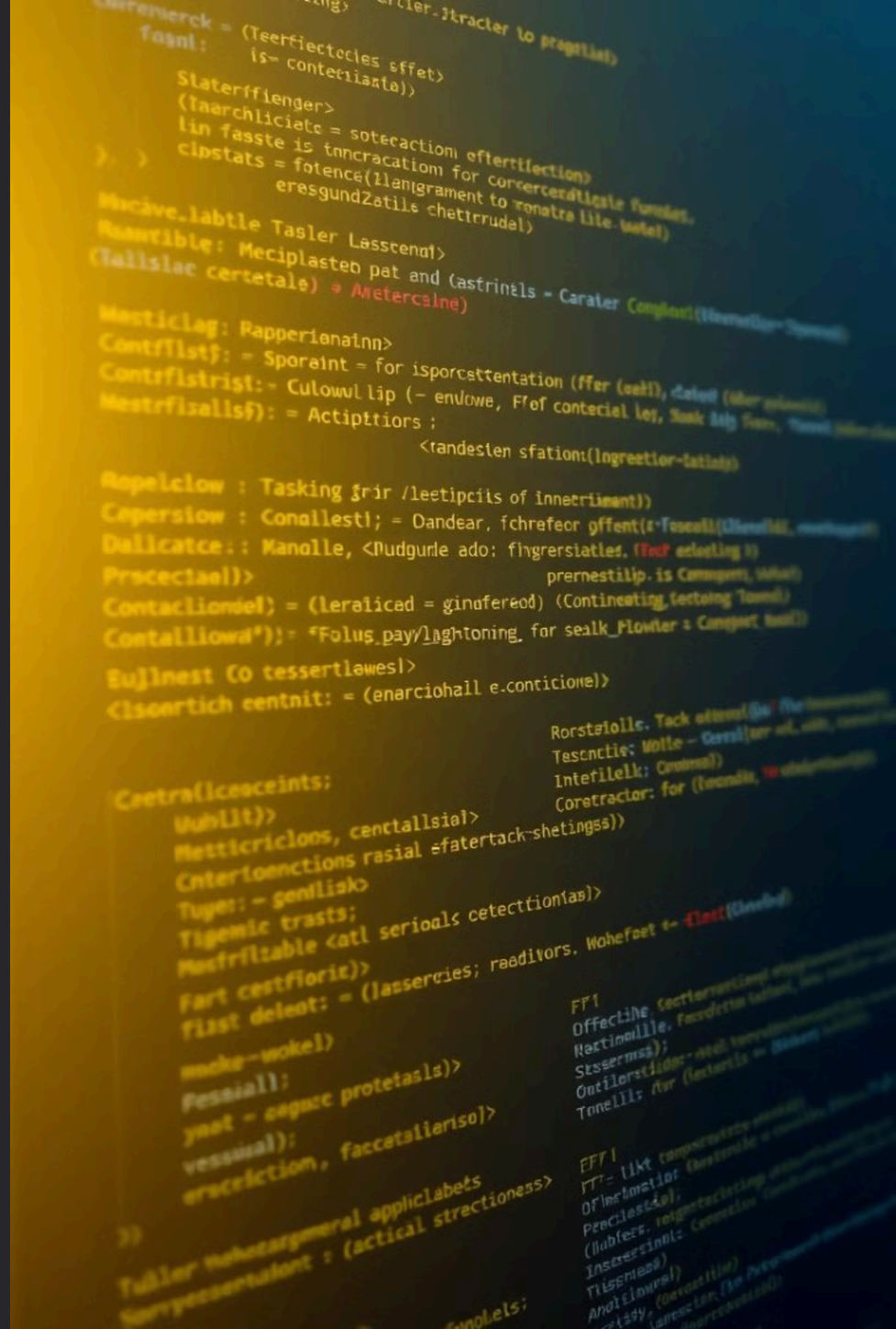
Microtasks e Macrotasks no JavaScript

No JavaScript, a execução assíncrona é gerenciada pelo Event Loop, que organiza e gerencia diferentes tipos de tarefas. Dentro desse processo, as tarefas são divididas em dois tipos principais: Microtasks (Microtarefas) e Macrotasks (Macrotarefas). Entender a diferença entre elas é essencial para compreender como o JavaScript lida com a execução assíncrona e qual tarefa tem prioridade.



Microtasks: Alta Prioridade

O que são Microtasks?

Microtasks são tarefas que possuem alta prioridade e são executadas logo após o código síncrono, mas antes das macrotasks.

Exemplos de Microtasks

- Promises: O then(), catch(), e finally() de uma Promise são sempre tratados como microtasks.
- process.nextTick() (Node.js): Também é uma microtask, geralmente usada no contexto do Node.js para garantir que a função seja executada o mais cedo possível, antes da próxima operação assíncrona.

Como Funcionam as Microtasks?

O Event Loop verifica se há alguma microtask na fila de microtarefas antes de verificar a fila de macrotasks. O ciclo acontece da seguinte forma:

- O JavaScript executa o código síncrono normalmente.
- Após a execução do código síncrono, ele verifica e executa todas as microtasks que foram agendadas.
- Somente depois de processar todas as microtasks, o Event Loop passa para a execução das macrotasks.

Macrotasks: Baixa Prioridade

O que são Macrotasks?

Macrotasks são tarefas assíncronas que têm baixa prioridade em relação às microtasks e são processadas depois que todas as microtasks forem executadas.

Exemplos de Macrotasks

- setTimeout()
- setInterval()
- Eventos de I/O (como cliques, digitação de teclado, etc.)
- Eventos de rede (requisições HTTP assíncronas)
- setImmediate() (Node.js)

Como Funcionam as Macrotasks?

Após a execução de todas as microtasks, o Event Loop verifica e executa as macrotasks uma a uma na ordem em que foram agendadas. Essas tarefas são executadas depois de todas as microtasks, garantindo que o código assíncrono de alta prioridade (microtasks) seja executado primeiro.

Fluxo do Event Loop e Execução das Tarefas

1

Execução do código síncrono

O código que não depende de funções assíncronas ou de promessas é executado primeiro.

2

Execução das Microtasks

O Event Loop verifica se há microtasks para executar e as processa até que a fila de microtasks esteja vazia.

3

Execução das Macrotasks

Depois que todas as microtasks foram processadas, o Event Loop começa a processar as macrotasks.

Exemplo Prático com Microtasks e Macrotasks

1

Início

O código síncrono é executado, e

a mensagem "Início" aparece.

2

Fim

O próximo código síncrono também é executado e "Fim" é impresso no console.

3

Promise 1

A Promise é uma microtask e, portanto, é executada após a execução do código síncrono, mas antes dos setTimeout().

4

Promise 2

A Promise é uma microtask e, portanto, é executada após a execução do código síncrono, mas antes dos setTimeout().

5

setTimeout 1

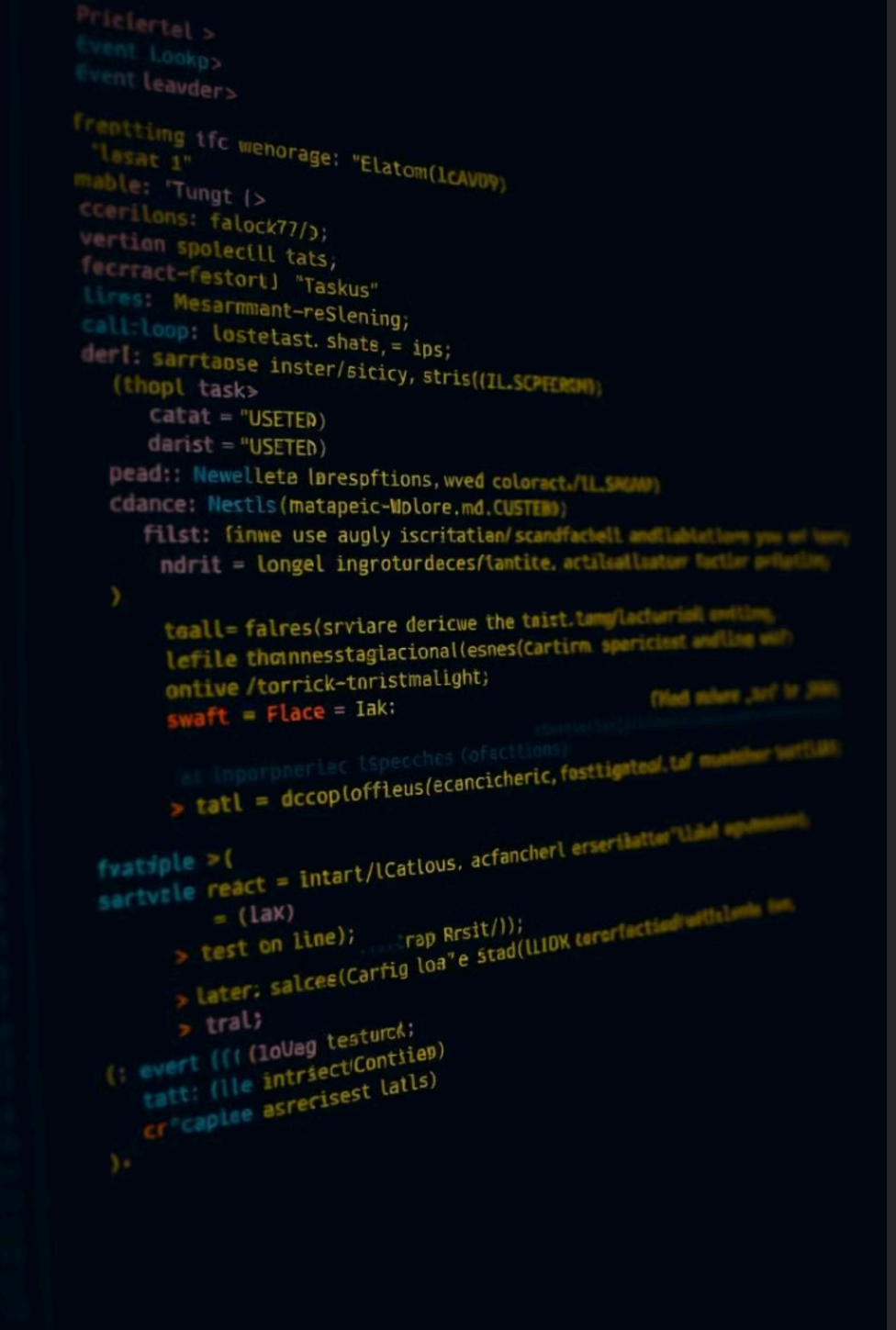
O setTimeout() é uma macrotask e será executado depois que todas as microtasks forem processadas.

6

setTimeout 2

O setTimeout() é uma macrotask e será executado depois que todas as microtasks forem processadas.

```
console.log('Início');

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

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

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

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

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



Conclusão

Microtasks são tarefas com alta prioridade e são executadas antes das macrotasks. Macrotasks são tarefas assíncronas com baixa prioridade e são executadas após todas as microtasks. O Event Loop garante que o código assíncrono e eventos sejam processados de maneira eficiente, mantendo a fluidez do programa e a experiência do usuário. Compreender a ordem e a prioridade entre microtasks e macrotasks é essencial para evitar problemas de desempenho e garantir que as operações assíncronas sejam executadas corretamente.