1. O que é o loop de eventos e qual sua importância no Node.js?

O **loop de eventos** é o mecanismo que permite ao Node.js executar código de forma **assíncrona e não bloqueante**, mesmo com apenas **uma única thread**.

Ele **verifica constantemente** se há tarefas (como leitura de arquivos ou respostas de rede) prontas para serem executadas, e **chama os callbacks** correspondentes.

Importância:

- Permite lidar com **várias conexões ao mesmo tempo**.
- Evita que o código fique parado esperando operações demoradas.
- Torna o Node.js ideal para **aplicações web rápidas e escaláveis**.

2. Diferencie CommonJS de ES Modules

CommonJS é o sistema antigo de módulos usado no Node.js. Nele, usamos require() para importar e module.exports para exportar. Ele funciona de forma **síncrona** (tudo é carregado imediatamente).

ES Modules (ou ESM) é o sistema moderno, padrão do JavaScript. Usa import e export e funciona de forma **assíncrona**, o que permite melhor desempenho em aplicações modernas.

Hoje, o Node.js suporta os dois, mas **ES Modules** é o recomendado para novos projetos.

3. O que são Streams e Buffers?

Buffers são blocos de dados binários usados pelo Node.js para armazenar temporariamente dados brutos — como arquivos, imagens ou dados recebidos da rede. Eles são úteis para lidar com dados que não estão em formato de texto.

Streams são formas de **ler ou escrever dados de maneira contínua**, aos poucos, sem precisar carregar tudo na memória de uma vez. Eles permitem processar grandes volumes de dados com eficiência, como ao transmitir um vídeo ou ler um arquivo muito grande.

Resumo:

- **Buffer**: armazena dados binários temporariamente.
- **Stream**: processa dados em partes, economizando memória.

4. Três tipos de Streams e suas aplicações:

1. **Readable Stream (Leitura)**
   Permite **ler dados aos poucos**, como ao ler um arquivo com fs.createReadStream() ou receber dados de uma requisição HTTP.
2. **Writable Stream (Escrita)**
   Permite **escrever dados aos poucos**, como ao salvar dados em um arquivo com fs.createWriteStream() ou enviar resposta para o cliente HTTP.
3. **Duplex Stream (Leitura e Escrita)**
   Faz **leitura e escrita ao mesmo tempo**, como em conexões de rede (net.Socket).

5. Diferença entre erros de runtime e erros de sintaxe:

- **Erro de sintaxe** acontece quando o código está **escrito de forma errada** e **não pode ser interpretado**. Exemplo: esquecer um parêntese ou uma vírgula. O programa **nem chega a rodar**.
- **Erro de runtime** acontece **durante a execução** do código, mesmo que ele esteja corretamente escrito. Exemplo: tentar acessar uma variável que não existe ou dividir por zero.

6. Quais são as vantagens de usar Streams para processamento de dados?

1. **Baixo uso de memória**: Streams processam os dados em partes, sem precisar carregar tudo de uma vez na memória.
2. **Alta performance**: São mais rápidos e eficientes para lidar com grandes volumes de dados, como arquivos grandes ou vídeos.
3. **Processamento contínuo**: Permitem começar a trabalhar com os dados **enquanto ainda estão sendo recebidos**, sem esperar o final.
4. **Escalabilidade**: Ideal para sistemas que precisam lidar com muitos dados ou múltiplas requisições ao mesmo tempo.

7. Como o módulo fs pode ser usado para manipular arquivos?

O módulo fs (**File System**) do Node.js permite **ler, escrever, criar, deletar e manipular arquivos e diretórios**.

Ler:

```
const fs = require('fs');

fs.readFile('arquivo.txt', 'utf8', (err, data) => {
 if (err) throw err;
 console.log(data);
});
```

Escrever:

```
fs.writeFile('novo.txt', 'Conteúdo novo', (err) => {
 if (err) throw err;
 console.log('Arquivo salvo!');
});
```

Apagar:

```
fs.unlink('arquivo.txt', (err) => {
 if (err) throw err;
 console.log('Arquivo deletado!');
});
```

8. O que acontece quando um erro não é tratado em um stream?

Quando um erro **não é tratado** em um stream, ele pode causar a **interrupção do processo** e o **encerramento do programa** com uma exceção (uncaughtException).

```
const fs = require('fs');

const stream = fs.createReadStream('arquivo_inexistente.txt');

stream.on('data', chunk => {
 console.log(chunk);
```

```
});


// Sem tratamento de erro — pode causar falha no programa!
```

Correto:

```
stream.on('error', err => {
  console.error('Erro no stream:', err.message);
});
```

Qual é a função do http no Node.js?

O módulo http permite criar **servidores web** e **fazer requisições HTTP** diretamente no Node.js, sem precisar de bibliotecas externas.

10. Explique o papel do Thread Pool no Node.js

O **Thread Pool** no Node.js é um grupo de threads em segundo plano usado para **executar tarefas pesadas ou bloqueantes**, como:

- Acesso ao sistema de arquivos
- Criptografia
- Compressão
- DNS e operações de rede não assíncronas

Apesar do Node.js rodar JavaScript em **uma única thread**, ele usa a **libuv**, que gerencia o Thread Pool (com 4 threads por padrão), para lidar com essas tarefas sem travar o event loop.