

AS 12 PERGUNTAS DE JAVASCRIPT QUE MAIS CAEM NA ENTREVISTA.

Responda em 3 partes: **o que é**, **um exemplo em código** e **onde você usa**. Treine em voz alta, escrevendo o código no editor.

01 O que é closure?

Resposta que aprova: Uma função que lembra as variáveis do lugar onde foi criada, mesmo depois que esse lugar terminou de executar. Uso no dia a dia em callbacks, hooks do React e para esconder estado.

```
function criarContador() {  
  let total = 0;  
  return () => ++total;  
}  
const contar = criarContador();  
contar(); // 1  
contar(); // 2
```

Pergunta seguinte: “Onde você usa closure?” Em callbacks, em hooks do React e para esconder uma variável que ninguém de fora pode alterar.

COMO TREINAR

1. 2 perguntas por dia, em voz alta, escrevendo o código no editor.
2. Rode cada exemplo no console do navegador e mude uma linha para ver o que acontece.
3. Responda também a pergunta seguinte: é nela que o júnior trava.

02 var, let e const: qual a diferença?

Resposta que aprova: var tem escopo de função e pode ser redeclarada. let e const têm escopo de bloco. const não deixa reatribuir a variável.

```
for (var i = 0; i < 3; i++) {  
  setTimeout(() => console.log(i)); // 3, 3, 3  
}  
for (let j = 0; j < 3; j++) {  
  setTimeout(() => console.log(j)); // 0, 1, 2  
}
```

Pergunta seguinte: "const deixa o objeto imutável?" Não. const impede reatribuir a variável, mas o conteúdo do objeto ainda pode mudar. Para congelar, use Object.freeze.

03 Como funciona o event loop?

Resposta que aprova: O JavaScript executa uma coisa por vez, na pilha de chamadas. O que é assíncrono espera na fila e só roda quando a pilha esvazia. Promises vão para a fila de microtarefas, que roda antes da fila do setTimeout.

```
console.log('1');  
setTimeout(() => console.log('2'), 0);  
Promise.resolve().then(() => console.log('3'));  
console.log('4');  
// 1, 4, 3, 2
```

Pergunta seguinte: "Por que o 3 aparece antes do 2?" Porque a Promise entra na fila de microtarefas, que esvazia antes da fila de tarefas em que o setTimeout espera.

04 Promise ou async/await?

Resposta que aprova: É a mesma coisa. async/await é um jeito mais legível de escrever código com Promise. Erro se trata com try/catch, e chamadas independentes rodam em paralelo com Promise.all.

```
async function carregar() {
  try {
    const [user, pedidos] = await Promise.all([buscarUser(),
    buscarPedidos()]);
    return { user, pedidos };
  } catch (erro) {
    console.error('Falhou:', erro);
  }
}
```

Pergunta seguinte: “Qual a diferença entre Promise.all e Promise.allSettled?”

Promise.all falha assim que uma falhar. Promise.allSettled espera todas e devolve o resultado de cada uma.

05 O que é o this?

Resposta que aprova: Depende de como a função é chamada. Chamada como método, this é o objeto antes do ponto. Arrow function não tem this próprio: usa o do lugar onde foi criada.

```
const user = { nome: 'Ana', falar() { return this.nome; } };
user.falar();           // 'Ana'
const f = user.falar;
f();                    // this não é mais o user (em módulo, dá
erro)
```

Pergunta seguinte: “Como fixar o this?” Com bind, por exemplo user.falar.bind(user), ou usando arrow function no callback.

06 map, filter ou reduce?

Resposta que aprova: map transforma cada item, filter escolhe itens e reduce junta tudo em um valor só. Os três devolvem um valor novo e não alteram o array original.

```
const pedidos = [{ valor: 50, pago: true }, { valor: 30, pago: false }, { valor: 20, pago: true }];
const total = pedidos
  .filter(p => p.pago)
  .map(p => p.valor)
  .reduce((soma, v) => soma + v, 0); // 70
```

Pergunta seguinte: “map ou forEach?” map devolve um array novo. forEach só executa a função e devolve undefined.

07 Qual a diferença entre == e ===?

Resposta que aprova: == converte os tipos antes de comparar. === compara valor e tipo, sem converter. No dia a dia, use ===.

```
0 == '0'           // true
0 === '0'          // false
null == undefined  // true
null === undefined // false
```

Pergunta seguinte: “Quando == faz sentido?” Em valor == null, que pega null e undefined de uma vez. Fora isso, ===.

08 O que é hoisting?

Resposta que aprova: O JavaScript registra as declarações antes de executar o código. `var` começa como `undefined`. `let` e `const` ficam inacessíveis até a linha da declaração. Function declaration pode ser chamada antes.

```
console.log(a); // undefined
var a = 1;

console.log(b); // ReferenceError
let b = 2;

saudar(); // funciona
function saudar() { console.log('oi'); }
```

Pergunta seguinte: “E uma arrow function guardada em `const`?” Não pode ser chamada antes da linha em que foi declarada, porque é uma `const`.

09 null ou undefined?

Resposta que aprova: `undefined` é a variável que nunca recebeu valor. `null` é vazio de propósito, colocado por quem escreveu o código.

```
let x;           // undefined
const y = null;  // null
typeof undefined // 'undefined'
typeof null      // 'object' (um erro antigo da linguagem)
```

Pergunta seguinte: “Como dar um valor padrão?” Com `??`: valor `??` 'padrão' troca só `null` e `undefined`. O `||` troca também `0` e texto vazio.

10 Cópia rasa ou cópia profunda?

Resposta que aprova: O spread copia só o primeiro nível: objetos dentro do objeto continuam sendo os mesmos. structuredClone copia tudo.

```
const original = { nome: 'Ana', endereco: { cidade: 'Recife' } };
const rasa = { ...original };
rasa.endereco.cidade = 'Natal';      // muda o original também

const profunda = structuredClone(original);
profunda.endereco.cidade = 'Olinda'; // o original não muda
```

Pergunta seguinte: “E JSON.parse(JSON.stringify(obj))?” Funciona para dados simples, mas transforma Date em texto e perde undefined e funções. Prefira structuredClone.

11 O que é debounce?

Resposta que aprova: Esperar o usuário parar de disparar um evento para executar a função uma vez só. O exemplo clássico é a busca enquanto a pessoa digita.

```
function debounce(fn, ms) {
  let timer;
  return (...args) => {
    clearTimeout(timer);
    timer = setTimeout(() => fn(...args), ms);
  };
}

const buscar = debounce(termo => api.buscar(termo), 300);
```

Pergunta seguinte: “E throttle?” Throttle executa no máximo uma vez a cada intervalo, mesmo com o evento disparando sem parar. Bom para scroll.

12 for...in ou for...of?

Resposta que aprova: for...in percorre as chaves de um objeto. for...of percorre os valores de qualquer iterável: array, string, Map e Set.

```
const lista = ['a', 'b'];  
for (const i in lista) console.log(i); // '0', '1' (as chaves)  
for (const v of lista) console.log(v); // 'a', 'b' (os valores)
```

Pergunta seguinte: "Como percorrer um objeto com for...of?" Com Object.entries: for (const [chave, valor] of Object.entries(obj)).

PARA ESTUDAR MAIS

- [MDN: Closures \(pt-BR\)](#)
- [MDN: let \(pt-BR\)](#)
- [MDN: modelo de execução e event loop \(pt-BR\)](#)
- [MDN: async function \(pt-BR\)](#)
- [MDN: this \(pt-BR\)](#)
- [MDN: comparações de igualdade \(pt-BR\)](#)
- [MDN: structuredClone \(inglês\)](#)
- [javascript.info: event loop \(inglês\)](#)
- [Philip Roberts: What the heck is the event loop anyway? \(JSConf EU, em inglês\)](#)

O júnior decora a definição e trava na pergunta seguinte. Quem passa explica com um exemplo e conta onde usou.

Segue @lucasadvice