Capítulo 4

Repetições

Programação com JavaScript & TypeScript

Antes de começar

Rodando programas no terminal

```
$ node meu_programa.js
$ npx tsx meu_programa.ts       # TypeScript
```

Lendo múltiplas entradas (útil para loops)

```
const rl = require("readline").createInterface({
  input: process.stdin, output: process.stdout
});

function perguntar(msg) {
  return new Promise(r => rl.question(msg, r));
}

const nome = await perguntar("Nome: ");
const idade = Number(await perguntar("Idade: "));
```

Laço for

Repetição com variável de controle — ideal quando sabemos quantas vezes repetir

```
for (let i = 1; i <= 10; i++) { ... }
```

início condição incremento

Crescente

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

Decrescente

```
for (let i = 10; i >= 1; i--) {
  console.log(i);
}
```

Passo customizado

```
for (let i = 0; i <= 20; i += 2) {
  console.log(i); // pares
}
```

Espaço para exemplo ao vivo — Tabuada

Laço while

Teste no início — repete enquanto a condição for verdadeira

Sintaxe

```
let i = 10;

while (i > 0) {
  console.log(i);
  i--;
}
```

Características

- Testa a condição ANTES de executar
- Os comandos podem NÃO ser executados
- Ideal quando não sabemos quantas repetições serão necessárias

Espaço para exemplo ao vivo

Laço do...while

Teste no final — garante no mínimo uma execução

Sintaxe

```
let senha;

do {
  senha = await perguntar("Senha: ");
  if (senha !== "1234") {
    console.log("Incorreta!");
  }
} while (senha !== "1234");
```

while vs do...while

	while	do...while
Teste	Início	Final
Execução mín.	0 vezes	1 vez
Uso típico	Leitura arquivos	Validação entrada

Espaço para exemplo ao vivo

break & continue

Comandos especiais para controlar o fluxo dentro de loops

break

Sai do laço imediatamente

```
for (let i = 1; i <= 100; i++) {
  if (i % 7 === 0) {
    console.log(`${i} é o 1º
      divisível por 7`);
    break;
  }
}
```

continue

Pula para a próxima iteração

```
for (let i = 1; i <= 10; i++) {
  if (i % 2 !== 0) {
    continue;
  }
  console.log(i);
  // imprime só pares
}
```

Contadores e acumuladores

Contador

Incrementa por valor constante (geralmente 1)

```
let numContas = 0;  // inicializa

numContas++;        // incrementa
numContas = numContas + 1; // idem
```

Acumulador

Incrementa por valor variável

```
let total = 0;      // inicializa

total += valor;     // acumula
total = total + valor; // idem
```

Regras fundamentais

1. Sempre inicializar antes do loop (geralmente com 0)
2. Receber a si mesmo + algum valor dentro do loop

Espaço para exemplo ao vivo — Soma de contas

Variável flag (sinalizadora)

Indica presença ou ausência de algo — otimiza loops

```
function ehPrimo(num) {
  let temDivisor = false;

  for (let i = 2; i <= num / 2; i++) {
    if (num % i === 0) {
      temDivisor = true;
      break;
    }
  }

  return num > 1 && !temDivisor;
}
```

Inicializa

false antes do loop

Sinaliza

true quando encontra

break

Sai imediatamente

Verifica

Usa o flag após o loop

Depuração de programas

Erros de sintaxe

Impedem o programa de rodar.
Ex: variável escrita errada, parêntese faltando

Erros de lógica

O programa roda, mas dá resultado errado.
Mais difíceis de encontrar!

Ferramentas

console.log() para inspecionar
Breakpoints no DevTools
Janela Watch

```
// Use console.log para inspecionar valores:
console.log("valor de i:", i, "estrelas:", estrelas);
```

Resumo — Capítulo 4

`for`	Variável de controle — sabe quantas vezes repete
`while`	Teste no início — pode não executar
`do...while`	Teste no final — executa ao menos 1 vez
`break / continue`	Interrompem ou pulam iteração no loop
`Contadores`	Variável + constante (ex: count++)
`Acumuladores`	Variável + variável (ex: total += valor)