Resumo JavaScript — Guia rápido para estudo

Material compacto com os pontos essenciais, exemplos práticos e exercícios — ideal para consulta rápida antes de exercícios passados pelo professor.

---

1. Tag `<script>`

- JavaScript é inserido entre `<script>` e `</script>`.
- `type="text/javascript"` é opcional hoje em dia.
- Scripts podem ser internos (no HTML) ou externos (`.js`), referenciados com `src`.
- Onde colocar o script:
  - No `<head>` (pode bloquear o carregamento da página). ○ No final do `<body>` (melhor para desempenho — DOM já carregado). ○ Com `defer` no `<script src="..." defer>` para carregar sem bloquear.

Exemplo (externo):

```
<!-- index.html -->
<!doctype html>
<html>q
<head>
  <meta charset="utf-8">
  <title>Exemplo</title>
</head>
<body>
  <h1 id="titulo">Olá</h1>
  <script src="app.js" defer></script>
</body>
</html> //
app.js
console.log('script carregado');
```

---

2. Saída de dados

- `innerHTML` altera conteúdo HTML (pode inserir tags). ▯ `innerText` altera apenas texto.
- `document.write()` escreve na página; se usado após o carregamento, substitui todo o HTML — evite em produção.
- `alert()` mostra caixa de alerta (bloqueante — uso limitado para testes).

- `console.log()` envia para o console (útil para depuração).

Exemplo — innerHTML vs innerText:

```
<div id="conteudo">Texto original</div>
<script>
  const el = document.querySelector('#conteudo');
  el.innerHTML = '<strong>Negrito via innerHTML</strong>';
  // el.innerText = '<strong>Negrito via innerText</strong>'; // mostraria as tags como texto
</script>
```

---

3. Entrada de dados

- window.prompt() pede texto ao usuário e retorna uma string ou null.

```
const nome = prompt('Digite seu nome:', 'Fulano'); if
(nome !== null) console.log('Olá, ' + nome);
```

---

4. Ponto e vírgula (;) e comentários

- Ponto e vírgula separa instruções; o JavaScript faz inserção automática de ponto e vírgula (ASI), mas é recomendado colocar `;` para evitar problemas.
- Comentários:
  - Linha: `// comentário` o
    Bloco: `/* comentário */`

---

5. Variáveis: `var`, `let`, `const`

- **`const`**: referência imutável — é obrigatório atribuir no momento da declaração. Não pode reatribuir a variável, mas **pode** alterar conteúdo de arrays/objetos referenciados.
- **`let`**: variável com escopo de bloco (ES6+) — use quando o valor mudará.
- **`var`**: escopo de função (antigo) — evitar por causa de hoisting e comportamento menos previsível; usar apenas para compatibilidade.

Exemplos:

```
const PI = 3.14; // não pode reatribuir PI = 3; let
contador = 0; contador++;
var x = 1; // tem hoisting e escopo de função

const arr = [1,2,3]; arr.push(4);
// permitido
```

```
// arr = [1]; // erro: não pode reatribuir
```

---

6. Escopo

- `var` é função-global (ou global se declarada fora de função).
- `let` e `const` têm escopo de bloco `{ ... }`.

Exemplo de escopo:

```
if (true) {
let a = 5;
  var b = 6;
}
// console.log(a); // ReferenceError console.log(b);
// 6 (porque var foge do bloco)
```

---

7. Operadores (resumo rápido)

- Aritméticos: `+ - * / % ++ --`
- Atribuição: `= += -= *= /=`
- Comparação: `==` (igualdade abstrata), `===` (igualdade estrita) — prefira `===` e `!==`.
- Lógicos: `&&` (e), `||` (ou), `!` (não)
- Ternário: `condição ? valorSeVerdadeiro : valorSeFalso`
- Concatenação de strings com `+` ou template strings: `` `Olá ${nome}` ``

Exemplo ternário e template string:

```
const idade = 18;
const maior = idade >= 18 ? 'sim' : 'não'; console.log(`É
maior? ${maior}`);
```

---

8. Boas práticas rápidas

- Sempre declare variáveis (`let`/`const`).
- Prefira `const` por padrão; use `let` só quando precisar reatribuir.
- Evite `var`.
- Use `===` e `!==` em comparações.
- Separe HTML, CSS e JS (use arquivos externos quando possível).
- Evite `document.write()` em código real.
- Adicione `defer` ao carregar scripts externos quando possível.

---

9. Exemplos práticos (úteis em exercícios)

1) Trocar o texto de um botão quando clicado

```
<button id="btn">Clique</button>
<script>
  document.querySelector('#btn').addEventListener('click', function() {
this.innerText = 'Clicado!';
  });
</script>
```

2) Somar dois números de campos de formulário

```
<input id="a" type="number" value="2">
<input id="b" type="number" value="3">
<button id="soma">Somar</button>
<div id="res"></div>
<script>
  document.querySelector('#soma').addEventListener('click', () => {
const a = Number(document.querySelector('#a').value);     const b
= Number(document.querySelector('#b').value);
    document.querySelector('#res').innerText = `Resultado: ${a + b}`;
  });
</script>
```

3) Exemplo de `const` com array (mutável internamente)

```
const lista = ['maçã', 'banana'];
lista.push('laranja'); // funciona
console.log(lista); // lista = [] //
erro
```

---

10. Exercícios para praticar (sem solução imediata)

1. Crie uma página com um campo `input` e um botão. Quando o usuário clicar, exiba em um `div` o texto convertido para maiúsculas.
2. Faça um contador: botão “+” incrementa, botão “-” decrementa. Mostre o valor atual.
3. Peça ao usuário, via `prompt()`, a sua idade e mostre no console se ele é maior de idade (>=18) usando ternário.
4. Substitua o conteúdo HTML de um `div` por uma lista `<ul>` criada dinamicamente a partir de um array.
5. Explique (em poucas linhas) a diferença entre `==` e `===` e dê um exemplo onde `==` retorna `true` e `===` retorna `false`.

---

11. Gabarito rápido (respostas sintéticas)

1. Use `toUpperCase()` no valor do input e `innerText` para mostrar.
2. Use `let contador = 0;` e dois `addEventListener` que fazem `contador++` e `contador--`, atualizando o `innerText`.
3. `const idade = Number(prompt('Idade?')); console.log(idade >= 18 ? 'Maior' : 'Menor');`
4. Criar `const ul = document.createElement('ul')`, fazer `array.forEach(item => { const li = document.createElement('li'); li.innerText = item; ul.appendChild(li); })`, e então `div.appendChild(ul)`.
5. `==` realiza coerção de tipos (por exemplo `0 == '0'` é `true`), `===` exige mesma identidade de tipo e valor (`0 === '0'` é `false`).
