

Oficina de seleção

Conceitos Básicos de Javascript

Variáveis

No Javascript as variáveis “Armazenam dados”, As variáveis em JavaScript são frequentemente descritas como "armazenadoras de dados", mas essa definição pode ser enganosa.

Exemplo

```
let value = 42; -> number  
value = "Hello"; -> agora é uma string
```

Neste ponto, a variável value é criada e é apontada para um local na memória onde o valor numérico 42 está armazenado.

Quando você atribui "Hello" a value, a referência anterior (que era o número 42) é descartada, e value agora passa a apontar para um novo local na memória onde a string "Hello" está armazenada.

Ou seja, a variável value não "armazenou" esses dados em si, ela apenas referenciou o valor 42 inicialmente e, depois, passou a referenciar a string "Hello" em outro local de memória.

Diferenças entre **var**, **let** e **const**

O escopo se refere ao local onde podemos acessar a variável.

var

- **Escopo:** O var tem **escopo de função** ou **escopo global**. Se for declarado fora de uma função, é global, se for declarado dentro de uma função, é local àquela função. Não respeita o escopo de blocos (como if e for).
- **Redeclaração:** Pode ser redeclarado na mesma função ou escopo.

let

- **Escopo:** O let tem **escopo de bloco**. Isso significa que variáveis declaradas com let só existem dentro do bloco onde foram criadas (como dentro de if, for).
- **Redeclaração:** Não pode ser redeclarado no mesmo escopo, mas pode ser atualizado.

const

- **Escopo:** Como o let, o const também tem **escopo de bloco**.

- **Mutabilidade:** Uma variável declarada com const **não pode ser reatribuída**, ou seja, não pode mudar para outro valor depois de inicializada. No entanto, se o valor for um objeto ou array, o conteúdo interno pode ser alterado.

Operadores

Operadores são símbolos ou palavras especiais que informam ao JavaScript para realizar uma ação específica sobre valores ou variáveis.

| Operador Aritmético | Descrição |
|---------------------|------------------------------|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Modulo(resto de uma divisão) |
| ++ | Incremento |
| -- | Decremento |

| Operador de comparação | Descrição |
|------------------------|---------------------------|
| == | Igual a |
| === | Mesmo valor e tipo |
| != | Diferente |
| !== | Diferente em valor e tipo |
| > | Maior que |
| >= | Maior ou igual a |
| < | Menor que |
| <= | Menor ou igual a |

| Operador Lógico | Descrição |
|-----------------|-----------|
| && | E |
| | Ou |
| ! | Negação |

Tipos primitivos

No JavaScript existem 6 tipos de primitivos:

| Tipos primitivos | Descrição |
|------------------|--|
| Number | Representa números, como 10 ou 3.14. |
| String | Representa texto, que fica entre aspas, como "Olá, mundo!". |
| Boolean | Representa um valor verdadeiro (true) ou falso (false). |
| Null | Indica que não há valor; é como um espaço vazio. |
| Undefined | Significa que uma variável foi criada, mas ainda não recebeu um valor. |
| Symbol | Um valor único e imutável que pode ser usado como identificador de propriedades em objetos. |
| BigInt | Um tipo que permite trabalhar com números inteiros muito grandes, maiores do que o que o tipo Number pode armazenar. |

Instruções condicionais

Estruturas que permitem que o código tome decisões com base em condições específicas.

If e Else

A instrução if executa um bloco de código se a condição especificada for verdadeira. Você também pode usar else para executar um bloco de código se a condição for falsa.

else if

Você pode encadear múltiplas condições usando else if.

Com else if, o JavaScript para de verificar as condições assim que encontra a primeira que é verdadeira. Isso pode tornar o código mais eficiente, especialmente se as condições forem complexas ou custosas em termos de processamento.

Laços

Laços (ou loops) em JavaScript são estruturas que permitem repetir um bloco de código várias vezes, facilitando a execução de tarefas repetitivas.

for

O laço for é utilizado quando você sabe exatamente quantas vezes deseja iterar sobre um bloco de código. Ele é frequentemente usado para percorrer arrays.

while

O laço while executa um bloco de código enquanto a condição especificada for verdadeira. Ele é útil quando o número de iterações não é conhecido de antemão.

do...while

O laço do...while é semelhante ao while, mas garante que o bloco de código seja executado pelo menos uma vez, independentemente da condição.

Casos de uso

Use while quando:

Você não tem certeza se o bloco de código deve ser executado com base na condição inicial.

A condição pode ser falsa antes da execução, e você não quer que o código seja executado nesse caso.

Use do...while quando:

Você quer garantir que o bloco de código seja executado pelo menos uma vez.

A condição precisa ser verificada após a execução do bloco.

Funções

O que é uma função?

Uma função é um bloco de código que pode ser chamado e executado sempre que você precisar. Pense nela como uma receita: você a escreve uma vez, e pode usá-la várias vezes.

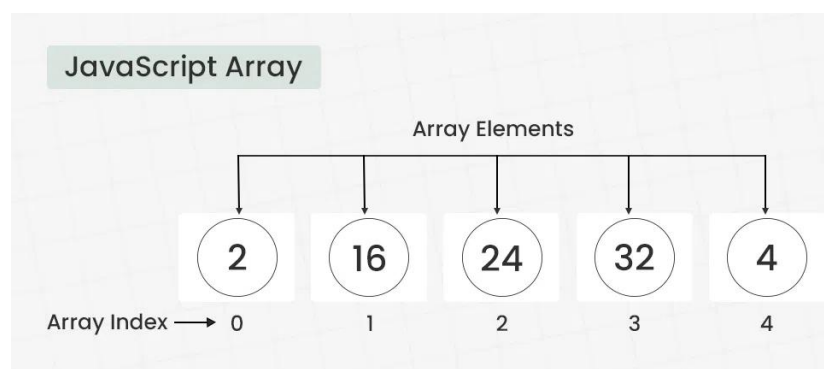
Exemplo:

```
function somar(a, b) {  
  return a + b;  
}  
  
let resultado = somar(5, 3);  
  
console.log(resultado)
```

- **Funções:** blocos de código reutilizáveis.
- **Parâmetros:** valores que você pode passar para a função.
- **Return:** permite que a função devolva um valor.

Arrays


Um **array** é uma lista de itens que podem ser de diferentes tipos (números, strings, objetos, etc.). Os itens são armazenados em uma ordem específica e podem ser acessados pelo seu índice (posição na lista).



fonte: geekforgeeks

Objetos

Um **objeto** é uma coleção de propriedades, onde cada propriedade tem uma chave e um valor. Os objetos são usados para armazenar dados relacionados.



```
const aluno = {nome : "Matheus",  
               idade : "23",  
               email : "matheushcastiglioni@gmail.com"  
};  
  
console.log(aluno.nome);  
console.log(aluno.idade);  
console.log(aluno.email);
```

fonte: Alura