Aula 2

O que vamos aprender nessa aula:

- Diferença entre `var` e `let` .
- Coerção.
- Vetores
- Matrizes
- Laço de repetição.

Diferença entre `var` e `let` em JavaScript

Em JavaScript, `var` e `let` são palavras-chave utilizadas para declarar variáveis, mas possuem diferenças importantes.

Escopo de Bloco

A principal diferença entre `var` e `let` é o escopo de bloco em que as variáveis são declaradas.

Ao usar `var`, a variável é escopada ao contexto da função em que foi declarada. Isso significa que a variável pode ser acessada em qualquer lugar dentro da função, mesmo fora de blocos de código específicos.

Por outro lado, ao usar `let`, a variável é escopada ao bloco em que foi declarada. Isso significa que a variável só pode ser acessada dentro do bloco em que foi definida.

Exemplos

Exemplo 1:

```
function exemploVar() {
  if (true) {
    var x = 10;
  }
```

```
  console.log(x);
}

exemploVar();
```

Neste exemplo, mesmo que a variável `x` seja declarada dentro de um bloco condicional (`if`), ela pode ser acessada fora do bloco porque foi declarada com `var`. Isso ocorre porque `var` não respeita o escopo de bloco.

Exemplo 2:

```
function exemploLet() {
  if (true) {
    let y = 20;
  }
  console.log(y);
}

exemploLet();
```

Neste exemplo, a variável `y` é declarada dentro de um bloco condicional com `let`. Dessa forma, ela só pode ser acessada dentro do bloco em que foi declarada. Fora desse bloco, ocorrerá um erro de referência.

Ao utilizar `let`, podemos garantir um escopo mais restrito para as variáveis, evitando problemas de acesso indevido e facilitando a manutenção do código.

Exemplo de Diferença entre `var` e `let` em um Laço de Repetição `for`

Considere o seguinte exemplo de um laço de repetição `for`:

```
function exemploVar() {
  for (var i = 0; i < 5; i++) {
    setTimeout(function() {
      console.log(i);
    }, 1000);
  }
}

exemploVar();
```

Neste exemplo, estamos usando `var` para declarar a variável `i` dentro do laço `for`. Suponha que queremos imprimir o valor de `i` a cada segundo usando o `setTimeout`.

No entanto, quando executamos esse exemplo, veremos que o valor impresso será sempre 5, repetido 5 vezes.

Isso acontece porque, quando usamos `var`, a variável `i` é escopada à função em que foi declarada, não ao bloco de código dentro do laço `for`. Portanto, quando a função `setTimeout` é executada após 1 segundo, o valor de `i` já foi incrementado para 5 no final do laço `for`.

Agora, vamos ver como o uso de `let` faz a diferença:

```
function exemploLet() {
  for (let j = 0; j < 5; j++) {
    setTimeout(function() {
      console.log(j);
    }, 1000);
  }
}

exemploLet();
```

Neste exemplo, usamos `let` para declarar a variável `j` dentro do laço `for`. Quando executamos esse exemplo, veremos que o valor impresso será 0, 1, 2, 3, 4, cada um após 1 segundo.

Isso ocorre porque, ao usar `let`, a variável `j` é escopada ao bloco de código dentro do laço `for`. Portanto, cada iteração do laço `for` terá seu próprio escopo para `j`, preservando seu valor correto quando a função `setTimeout` é executada.

Dessa forma, ao usar `let` em laços de repetição `for`, podemos evitar problemas de acesso indevido e garantir o comportamento esperado do código.

Coerção em JavaScript

A coerção é o processo em que o JavaScript automaticamente converte um tipo de dado em outro tipo, quando ocorre uma operação entre valores de tipos diferentes. Isso pode acontecer de forma implícita, quando o JavaScript realiza a conversão automaticamente, ou de forma explícita, quando especificamos a conversão.

Existem dois tipos de coerção em JavaScript: coerção implícita e coerção explícita.

Coerção Implícita

A coerção implícita ocorre quando o JavaScript realiza automaticamente a conversão de tipos durante as operações. Por exemplo, quando você soma um

valor numérico com uma string, o JavaScript irá converter o valor numérico em uma string e concatenar as duas strings. Veja o exemplo abaixo:

```
let idade = 25;
let mensagem = "Eu tenho " + idade + " anos.";
console.log(mensagem);
```

Nesse exemplo, a coerção implícita converte o valor da variável `idade` de um número para uma string, para que possa ser concatenado com a string "Eu tenho" e a string "anos.".

Coerção Explícita

A coerção explícita ocorre quando especificamos manualmente a conversão de um tipo de dado para outro tipo. Isso pode ser feito usando funções de conversão específicas do JavaScript, como `Number()`, `String()`, `Boolean()`, entre outras. Veja o exemplo abaixo:

```
let numero = "10";
let numeroConvertido = Number(numero);
console.log(numeroConvertido);

let booleano = 0;
let booleanoConvertido = Boolean(booleano);
console.log(booleanoConvertido);
```

Nesse exemplo, usamos a função `Number()` para converter a string "10" em um número e a função `Boolean()` para converter o número 0 em um booleano.

Considerações sobre Coerção

Embora a coerção possa ser útil em certas situações, é importante ter cuidado ao lidar com ela, pois pode levar a resultados inesperados. Por exemplo, em operações de igualdade (`==`), o JavaScript realiza a coerção implícita para comparar valores de tipos diferentes, o que pode levar a comparações não intuitivas.

Por esse motivo, é recomendado utilizar o operador de igualdade estrita (`===`), que compara os valores e os tipos de dados de forma rigorosa, sem realizar coerção implícita.

Ao realizar operações que envolvam tipos de dados diferentes, é importante entender como a coerção funciona em JavaScript e considerar se é necessário realizar a conversão explícita para evitar comportamentos indesejados.

A coerção é um aspecto importante a ser compreendido para escrever código JavaScript robusto e evitar erros sutis que podem surgir devido a conversões automáticas de tipos.

Vetores em JavaScript

Em JavaScript, um vetor é uma estrutura de dados que armazena uma coleção de elementos sequencialmente. É uma forma de armazenar múltiplos valores em uma única variável.

Para criar um vetor em JavaScript, podemos usar a notação de colchetes (`[]`) e separar os elementos por vírgulas. Por exemplo:

```
let vetor = [10, 20, 30, 40, 50];
```

Podemos acessar os valores armazenados em um vetor usando a notação de colchetes e o índice do elemento desejado. Os índices em JavaScript começam em zero, ou seja, o primeiro elemento de um vetor tem o índice 0, o segundo elemento tem o índice 1 e assim por diante. Por exemplo:

```
let vetor = [10, 20, 30, 40, 50];
console.log(vetor[0]);
console.log(vetor[2]);
```

Nesse exemplo, o primeiro `console.log` imprime o valor do elemento com índice 0, que é 10, e o segundo `console.log` imprime o valor do elemento com índice 2, que é 30.

É importante lembrar que, se tentarmos acessar um índice que não existe no vetor, será retornado `undefined`. Portanto, devemos garantir que o índice esteja dentro dos limites do vetor.

Além disso, podemos modificar os valores de um vetor atribuindo um novo valor a um elemento específico usando a mesma notação de colchetes. Por exemplo:

```
let vetor = [10, 20, 30, 40, 50];
vetor[1] = 25;
```

```
console.log(vetor);
```

Nesse exemplo, atribuímos o valor 25 ao elemento com índice 1 do vetor, substituindo o valor original 20.

Os vetores são uma parte fundamental da programação em JavaScript, permitindo armazenar e acessar conjuntos de valores de forma eficiente. Ao entender como criar e acessar vetores, podemos lidar com dados de forma mais estruturada e realizar operações complexas em nossos programas.

Matrizes em JavaScript

Em JavaScript, uma matriz é uma estrutura de dados que armazena uma coleção de elementos em uma grade bidimensional. Ela é composta por linhas e colunas, permitindo armazenar e manipular conjuntos de valores de forma organizada.

Para criar uma matriz em JavaScript, podemos usar a notação de colchetes (`[]`) e separar os elementos por vírgulas. Cada elemento é uma lista separada por vírgulas que representa uma linha da matriz. Por exemplo:

```
let matriz = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

Nesse exemplo, criamos uma matriz de 3x3, onde cada elemento representa um número. A primeira linha contém os valores 1, 2 e 3, a segunda linha contém os valores 4, 5 e 6, e a terceira linha contém os valores 7, 8 e 9.

Podemos acessar os valores armazenados em uma matriz usando a notação de colchetes e os índices das linhas e colunas desejadas. Os índices em JavaScript também começam em zero. Por exemplo:

```
let matriz = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

console.log(matriz[0][0]);
console.log(matriz[1][2]);
```

Nesse exemplo, o primeiro `console.log` imprime o valor da primeira linha e primeira coluna da matriz, que é 1, e o segundo `console.log` imprime o valor da segunda linha e terceira coluna da matriz, que é 6.

É importante lembrar que devemos garantir que os índices estejam dentro dos limites da matriz, caso contrário, será retornado `undefined`. Além disso, podemos modificar os valores de uma matriz atribuindo um novo valor a um elemento específico usando a mesma notação de colchetes. Por exemplo:

```
let matriz = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

matriz[1][2] = 10;
console.log(matriz);
```

Nesse exemplo, atribuímos o valor 10 à segunda linha e terceira coluna da matriz, substituindo o valor original 6.

As matrizes são uma parte importante da programação em JavaScript, permitindo estruturar dados em uma grade bidimensional. Ao entender como criar e acessar matrizes, podemos trabalhar com dados de forma organizada e realizar operações complexas em nossos programas.