Aula 1

O que vamos aprender nessa aula:

- Sistema de Tipos no javascript
- Estados
- Constantes e Variáveis (valores únicos) e sua relação com o conceito matemático
- Uso de funções nativas da linguagem
- Coerções (mudança de tipos)
- Operadores e aritmética

Sistema de Tipos no JavaScript

O JavaScript possui um sistema de tipos dinâmico, o que significa que as variáveis podem conter diferentes tipos de dados durante a execução do programa. Alguns dos principais tipos de dados em JavaScript são:

- **Number**: representa valores numéricos, como 1, 3.14, ou -5.
- **String**: representa sequências de caracteres, como "Olá, mundo!".
- **Boolean**: representa os valores verdadeiro (`true`) e falso (`false`).
- **Array**: representa uma coleção de elementos ordenados.
- **Object**: representa uma estrutura de dados complexa, contendo propriedades e métodos.

Aqui estão alguns exemplos de declaração de variáveis em JavaScript:

```
let idade = 25; // Number

let nome = "Maria"; // String

let isStudent = true; // Boolean

let numeros = [1, 2, 3, 4, 5]; // Array
```

```
let pessoa = { nome: "João", idade: 30 }; // Object
```

É importante lembrar que o JavaScript também permite a conversão implícita e explícita entre os diferentes tipos de dados, através de operações chamadas de coerções.

Estado em JavaScript

No contexto do JavaScript, o termo "estado" refere-se ao valor atual de uma variável em um determinado momento durante a execução do programa. Uma variável pode ter seu estado alterado ao longo do tempo, à medida que o programa é executado e as operações são realizadas. O estado de uma variável é fundamental para o funcionamento de muitos algoritmos e lógicas de programação.

Em JavaScript, podemos atribuir valores a uma variável usando a palavra-chave `let` ou `const`. Essas variáveis podem ser atualizadas e seu estado pode ser alterado ao longo do programa. No entanto, também é possível usar a palavra-chave `const` para criar constantes, que são variáveis cujo valor não pode ser alterado após a sua atribuição inicial.

Por exemplo, vamos considerar a seguinte variável em JavaScript:

```
let contador = 0;
```

Neste caso, a variável `contador` possui um estado inicial de 0. Ao longo do programa, podemos alterar o estado dessa variável incrementando ou decrementando seu valor:

```
contador = contador + 1; // incrementa o valor do contador em 1
contador = contador - 1; // decrementa o valor do contador em 1
```

Dessa forma, o estado da variável `contador` é atualizado conforme as operações são realizadas.

É importante compreender o conceito de estado em JavaScript para garantir que as variáveis tenham os valores corretos em cada etapa do programa e para acompanhar as mudanças de valor ao longo da execução.

Constantes e Variáveis (valores únicos) e sua relação com o conceito matemático

Em programação, as constantes e variáveis desempenham um papel semelhante aos conceitos matemáticos de constantes e variáveis.

Constantes são valores fixos que não mudam ao longo do programa. Eles são semelhantes às constantes matemáticas, como π (pi) ou e, que têm um valor constante e não podem ser alteradas. Em JavaScript, podemos criar constantes usando a palavra-chave `const`. Por exemplo:

```
const PI = 3.14159;
const GRAVITY = 9.8;
```

Variáveis são valores que podem mudar ao longo do programa. Elas são semelhantes às variáveis matemáticas, onde podemos atribuir diferentes valores a elas. Em JavaScript, podemos criar variáveis usando a palavra-chave `let`. Por exemplo:

```
let x = 5;
let y = 10;
```

Podemos usar constantes e variáveis em expressões matemáticas e operações aritméticas. Por exemplo:

```
const raio = 5;
let area = Math.PI * raio * raio;
let circunferencia = 2 * Math.PI * raio;
```

Nesse exemplo, a constante `raio` representa o raio de um círculo e as variáveis `area` e `circunferencia` armazenam os resultados dos cálculos. Podemos usar essas constantes e variáveis em expressões matemáticas para realizar cálculos.

Entender a relação entre constantes, variáveis e conceitos matemáticos é fundamental para escrever programas que envolvam cálculos e operações matemáticas de forma precisa e eficiente.

Uso de funções nativas do JavaScript

O JavaScript possui uma ampla variedade de funções nativas que são fornecidas pela própria linguagem. Essas funções nativas são chamadas diretamente no código e permitem executar diversas tarefas e operações com facilidade.

Algumas das funções nativas mais comumente usadas no JavaScript incluem:

- `console.log()`: Essa função é usada para exibir mensagens no console do navegador ou em um ambiente de desenvolvimento. É útil para depuração e para exibir informações durante a execução do programa.
- `alert()`: Essa função é usada para exibir uma caixa de diálogo com uma mensagem para o usuário. É útil para fornecer informações importantes ou solicitar ação do usuário.
- `prompt()`: Essa função é usada para exibir uma caixa de diálogo que permite que o usuário insira um valor. É útil para solicitar entrada do usuário e capturar dados fornecidos por ele.
- `Math.random()`: Essa função retorna um número pseudoaleatório entre 0 e 1. É útil quando você precisa gerar valores aleatórios em seu programa.
- `Math.round()`: Essa função arredonda um número para o inteiro mais próximo. É útil quando você precisa arredondar números com precisão.
- `Math.floor()` e `Math.ceil()`: Essas funções arredondam um número para baixo e para cima, respectivamente. Elas são úteis quando você precisa arredondar números para o próximo número inteiro menor ou maior.
- `parseFloat()`: Essa função é usada para converter uma string em um número de ponto flutuante (float). Ela analisa o valor da string e retorna um número decimal. Por exemplo:
- `parseInt()`: Essa função é usada para converter uma string em um número inteiro (integer). Ela analisa o valor da string e retorna um número inteiro. Por exemplo:

Essas são apenas algumas das muitas funções nativas disponíveis no JavaScript. Ao utilizar essas funções, você pode realizar uma ampla gama de tarefas, desde exibir informações na tela até executar cálculos complexos.

O conhecimento e uso adequado das funções nativas do JavaScript podem ajudar a tornar seu código mais eficiente e eficaz, permitindo que você aproveite ao máximo a linguagem.

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."; // A coerção implícita converte a idade em uma string
console.log(mensagem); // Output: Eu tenho 25 anos.
```

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); // Coerção explícita para converter a string "10" em um número
console.log(numeroConvertido); // Output: 10 (agora é um número)

let booleano = 0;
let booleanoConvertido = Boolean(booleano); // Coerção explícita para converter o número 0 em um booleano
console.log(booleanoConvertido); // Output: false (agora é um booleano)
```

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.

Operadores e Aritmética em JavaScript

Em JavaScript, os operadores são símbolos especiais que realizam operações em valores e variáveis. A aritmética em JavaScript envolve o uso desses operadores para realizar cálculos matemáticos.

Alguns dos operadores aritméticos mais comuns em JavaScript incluem:

- **Adição (+)**: Realiza a adição de dois valores.
- **Subtração (-)**: Realiza a subtração de dois valores.
- **Multiplicação (*)**: Realiza a multiplicação de dois valores.
- **Divisão (/)**: Realiza a divisão de dois valores.
- **Resto (%)**: Retorna o resto da divisão de dois valores.
- **Incremento (++)**: Aumenta o valor de uma variável em 1.
- **Decremento (--)**: Diminui o valor de uma variável em 1.

Aqui está um exemplo de uso de operadores aritméticos em JavaScript:

```
let x = 5;
let y = 2;

let soma = x + y; // 7
let subtracao = x - y; // 3
```

```
let multiplicacao = x * y; // 10
let divisao = x / y; // 2.5
let resto = x % y; // 1

x++; // Incrementa o valor de x em 1 (x é igual a 6)
y--; // Decrementa o valor de y em 1 (y é igual a 1)
```

Além dos operadores aritméticos, JavaScript também possui outros operadores, como operadores de atribuição, operadores de comparação e operadores lógicos, que permitem a realização de diferentes tipos de operações.

Ao utilizar operadores e realizar operações aritméticas em JavaScript, é importante levar em consideração as regras de precedência dos operadores, que determinam a ordem em que as operações são executadas.

Compreender os operadores e a aritmética em JavaScript é fundamental para realizar cálculos e manipulações de valores de forma eficiente e correta em seus programas.