[CT JS] Aula 1

Apresentação

Ambiente de desenvolvimento

Opções

- Navegador
- Editor de texto (VSCode)
- Codepen (https://codepen.io/)

♣ Conteúdo

▼ Sistema de Tipos

- Linguagens Estáticas:
  - Verificação em tempo de compilação, erros verificados antes de executar o projeto.
  - Geralmente é necessário declarar os tipos das variáveis para auxiliar o compilador na verificação de consistencia.
  - C, C++, Java
- Linguagens Dinâmicas:
  - Verificação em tempo de execução, ou seja, os erros são identificados quando o programa é executado
  - Sem a necessidade de declarar tipos
  - Python, JS e Ruby

> 💡 JavaScript é uma linguagem fracamente tipada e dinâmica

Tipos (6) de acordo com a documentação oficial ECMAScript (https://262.ecma-international.org/5.1/#sec-8)

- **Undefined:** Qualquer variável que não possua um valor atribuido terá o valor *undefined.* É o valor padrão atribuido automaticamente a variáveis que ainda não foram definidas ou não tem um valor atribuido.
- **Null**: É um valor atribuido de forma proposital para indicar que uma variável não possui valor por enquanto.
- **Boolean**: True/False
- **String**: Usado para inserção de valores textuais. Cada elemento da string é tratado de forma unitária, tendo suas posições indexadas a partir da posição 0.
- **Number**: Usado para valores inteiros, decimais, NaN e Infinity
  - **NaN**: Quando temos erros matemáticos (Not-a-number), é um resultado que não é válido. Para verificar podemos usar o `isNaN()`
  - **Infinity**: É um valor especial que representa o infinito podendo ser positivo ou negativo `-Infinity`
- **Object**: É uma coleção de propriedades com chave e valor

---

Referências

https://dev.to/zakariaelk/undefined-vs-not-defined-vs-null-185g

Constantes e variáveis

- ***var***
  - **Escopo de Função**: São visiveis em toda a função em que foram declaradas, mesmo fora dos blocos de código.
  - **Hoisting**: As variáveis são elevadas (hoisted) para o topo do escopo que foram definidas, ou seja, podemos usar a variável antes de sua declaração.

```
console.log(x); // Saída: undefined var x = 5;
console.log(x); // Saída: 5
```

  - **Reatribuição**: Pode receber novos valores e permite que seja redistribuida (pode ser declarada novamente)
- ***let***
  - **Escopo de bloco:** É visivel apenas dentro do bloco em que foi declarada
  - **Hoisting**: Elevada para o topo do bloco mas não pode ser usada antes da declaração
  - **Reatribuição**: Pode ser reatribuida (receber um novo valor) mas não permite redistribuição (não pode ser declarada novamente)
- ***const***
  - **Escopo de bloco**: É visível apenas dentro do bloco em que foi declarada
  - **Hointing**: Elevada para o topo do bloco mas não pode ser usada antes da declaração
  - **Imutabilidade**: São constantes que não podem ser reatribuidas ou redefinidas após a inicialização, contudo se forem objetos as propriedades podem ser modificadas.

Funções nativas da linguagem

- **console**: Objeto que fornece métodos para interagir com a saída do console do ambiente de execução. Algumas das funções uteis:
  - log(): mensagens informativas de depuração;
  - error(): mensagens de erro
  - warn(): mensagens de alerta
  - table(): exibe array em formato de tabela com os indices correspondentes
- **parseFloat()**: conversão de string em float. O Javascript utiliza o ponto como separador decimal independente da localização do ambiente, isso se deve a utilização do padrão internacional para facilitar a interoperabilidade entre ambientes

```
let numeroComVirgula = "3,14"; let numero =
parseFloat(numeroComVirgula.replace(",", "."));
console.log(numero); // Saída: 3.14
```

- **parseInt()**: conversão de string em inteiro
- **String**:
  - **replace**: substituição de trechos do texto
  - **concat**: concatenar strings
  - **includes**: verificar a existencia de palavra
- **Math**: Não afeta o próprio numero, sempre retorna uma nova instância
  - **floor**: arredonda pra baixo, retorna o numero inteiro menor ao fornecido.
  - **round**: arredonda o número para o inteiro mais próximo `para baixo < 0.5, para cima ≥ 0.5`
  - **ceil**: arredonda para cima, retorna o numero inteiro maior ou igual ao fornecido.

Coerções (casting)

Conversão de um tipo de dado em outro. No JS podemos ter esta conversão de forma explicita ou implicita.

- **Casting implícito:** Acontece automaticamente quando o JavaScript tenta converter um tipo de dado em outro. Por exemplo, em operações aritméticas, o JavaScript pode realizar casting implícito para garantir que os operandos tenham o mesmo tipo antes da operação.

```
let numero = 10; let texto = "20"; let resultado = numero + texto; // Aqui ocorre um casting implícito de 'numero' para string console.log(resultado); // Saída: "1020"
```

- **Casting explícito**: Neste cenário temos a intervenção do desenvolvedor para converter um tipo de dado em outro usando funções ou operadores específicos. Por exemplo, você pode usar `parseInt()` ou `parseFloat()` para converter uma string em um número.

```
let texto = "123"; let numero = parseInt(texto); // Casting explícito de 'texto' para número console.log(numero); // Saída: 123
```

  - **Number()**: Também é um caso de casting explícito no qual o valor é convertido para o tipo primitivo number. Caso a string possuir caracteres não numérico retorna NaN.

> 💡 **Number()** sempre irá retornar NaN caso encontre algum caracter não numérico na string, já o **parseInt()** e o **parseFloat()** tentarão extrair o máximo de dígitos

Operações e Aritmética

O JS segue as regras padrão de precedencia matemática:

- Parenteses (não temos [ ] ou { })
- Operadores de exponenciação (**)
- Operadores de multiplicação (*), Divisão (/) e módulo (%)
- Operadores de Adição (+) e Subtração (-)

Quando temos operações de mesma precedência, a avaliação deve ser realizada da esquerda para a direita.

♧ Exercícios

> Link para alunos
> [CT JS] Exercícios - Aula 1

1. Declare duas variáveis, `a` e `b`, atribuindo a elas valores booleanos diferentes. Em seguida, crie uma condição que verifica se ambas são verdadeiras.

```
let a = true; let b = false; let resultado = a && b; console.log("Ambas são verdadeiras:", resultado);
```

2. Declare duas variáveis numéricas, `c` e `d`, e realize uma operação matemática entre elas. Imprima o resultado no console.

```
let c = 5; let d = 3; let resultado = c + d; console.log(`Resultado da operação: ${resultado}`);
```

3. Declare uma variável `e` contendo uma string de sua escolha. Em seguida, concatene essa string com outra e imprima o resultado.

```
let e = "Olá"; let outraString = " Mundo!"; let resultadoConcatenacao = e + outraString; console.log(resultadoConcatenacao);
```

4. Declare duas variáveis, `f` e `g`, ambas inicializadas como `undefined`. Imprima no console se essas variáveis são estritamente iguais.

```
let f = undefined; let g = undefined; console.log("f e g são est
ritamente iguais:", f === g);
```

5. Declare um objeto vazio chamado `c`. Adicione propriedades a esse objeto, como `nome` e `idade`, e imprima o objeto no console.

```
let c = { nome: "João", idade: 25 }; console.log(c);
```

6. Declare uma array vazia chamada `d`. Adicione alguns elementos a essa array e, em seguida, imprima o comprimento da array.

```
let d = ["item1", "item2", "item3"]; console.log("Comprimento da
array:", d.length);
```

7. Declare uma variável `e` inicializada como `null`. Imprima no console o tipo de `e` utilizando o operador `typeof`.