TI academy

@tiacademybrasil

JAVASCRIPT

AULA 4 | Prof. Marcelo

JS

- Strings
  - for
  - for in
  - for of
  - while
- Manipulação de elementos - querySelectorAll
- Criar elementos

O que é um laço de repetição?

Laços de repetição ou interação permite que uma determinada rotina seja repetida um certo número de vezes.

A repetição é terminada quando chega até um limite ou até quando atinger alguma condição.

FOR

A instrução for irá criar um “loop” que contém **três expressões** dentro de parênteses e **separadas por ponto e vírgula**. Entre as chave ficará o código que será executado durante este loop.

```
for(var i=0; i <=0; i++) {

    //repete o código
}
```

FOR in

O laço de repetição **for...in** possibilita percorrer ou iterar sobre as propriedades de um objeto.

```
for ([indice] in [objeto]) {
  declaração
}
```

FOR of

O laço de repetição **for...of** acessa ou percorre uma propriedade de um objeto de forma direta.

```
for ([indice] of [objeto]) {
  declaração
}
```

While

A declaração while irá criar um laço que executa uma determinada rotina enquanto a condição está sendo avaliada como verdadeira.

A condição é avaliada antes da execução da rotina.

```
while(condição) {
   declaração
}
```

do while

O do...while irá cria um laço que irá se repetir **até que** o teste da condição seja **falsa**.

A condição é avaliada depois que o bloco de código é executado, resultando que uma declaração seja executada pelo menos uma vez.

```
do {
    codigo
}while(condição)
```

FORMATAÇÃO DE STRINGS

Strings

O javascript trata as strings como se fossem arrays, isso por ser de fato uma cadeia ou lista de caracteres. Exemplo:

```
var st = “javascript”;
console.log(st.lenght);
console.log(st[0]);
```

Manipulação de Strings

Existem diversas funções disponíveis para manipular strings, estas funções auxiliam no tratamento de strings em diversas ocasiões.

match()
O método match procura uma palavra em uma string. Existem alguns parâmetros de pesquisa que permite uma maior precisão conforme a necessidade.

Modificador	Descrição
i	Busca sem case-sensitive, ou seja, não diferencia letras maiúsculas de minúsculas
g	Diz ao método para encontrar todas as ocorrências da palavra e não parar na primeira encontrada, cada ocorrência é armazenada em uma posição do array
m	Pesquisa normal sem armazenar em forma de array

Ex.:

```
var str = “Curso de Javascript”;
console.log( srt.match(/Curso/gim);
```

Manipulação de Strings

search()

O método serach() procura pela ocorrência e retornando a posição na cadeia da string, a posição é em relação ao primeiro caractere da ocorrência.

```
var str = "Curso de Javascript";
console.log( str.search(/D/gi));
```

replace()

Este método substitui uma string por outra, simples assim, ele pesquisa a string informada, como no método “match” e a substitui por outra string nas aspas informada como segundo parâmetro

```
var lorem = "Lorem ipsum, dolor sit amet consectetur adipisicing, elit."+
            "Pariatur exercitationem harum, voluptatem, animi repellendus"+
            "sequi ratione corporis nemo eaque atque reiciendis, assumenda,"+
          "iusto quia rerum. Provident suscipit, repellendus expedita id?";

var mudaLdoLorem = lorem.replace(/e/gi,'X');

console.log(mudaLdoLorem)
```

Manipulação de Strings

localeCompare( )

O método *localeCompare* compara efetua a comparação entre duas strings, se estas forem iguais o retorno será **“0”** zero. O valores -1 e 1 podem ser esperados caso elas nãos sejam iguais.

```
var nome1="Comparar";
var nome2="Comparar";
var resultado=nome1.localeCompare(nome2);
console.log(resultado);
```

JS

Manipulação de Strings

toString()
O uso da toString irá converter um valor qualquer em uma um srtring.

```
var n = Number(10);
var nParaString = n.toString();
console.log(nParaString);
```

toLowerCase()
Faz a conversão de uma string inteira para caracteres minúsculos (caixa baixa):

```
var str = “TESTE”;
console.log(str.toLowerCase());
```

toUpperCase()
Faz a conversão de uma string inteira para caracteres maiúsculos (caixa alta).

JS

Manipulação de Strings

trim()

Faz a remoção de espaços antes e depois da string especificada.

```
var strComEspaco = 'palavra    ';
console.log(str.trim());
```

Number

Converte uma string em um número inteiro (parseInt) ou decimal (parseFloat), caso não consiga efetuar a conversão será retornando **NaN** (Not as Number).

JS

Conversão de valores para moeda

O método *toLocaleString*() irá retornar uma string com uma representação da moeda definida como no parâmetro **currency**.

style: O estilo do formato a ser utilizado. Os valores permitidos são "decimal" para formato de número simples, "currency" para formato monetário e "percent" para formato percentual; o padrão é "decimal".

currency: A moeda para usar na formatação monetária

```
var valor = var valor = 1.35;
console.log( valor.toLocaleString('pt-BR', { style: 'currency', currency: 'BRL' }));
```

querySelector e querySelectorAll e o método forEach

Diferença entre o querySelector e o querySelectorAll

O **querySelector** irá selecionar sempre o primeiro elemento de uma lista de elementos de uma determinada classe.

Já o **querySelectorAll** irá retornar uma *lista* dos elementos presentes no documento que estão no mesmo GRUPO DE SELETORES como home da classe ou mesmo o nome do elemento.

Diferença entre o querySelector e o querySelectorAll

```
<ul>
        <li class="lista">Item 1</li>
        <li class="lista">Item 2</li>
        <li class="lista">Item 3</li>
        <li class="lista">Item 4</li>
        <li class="lista">Item 5</li>
</ul>
```

document.**querySelector**(“ul > .lista ”) irá retorar apenas o primeiro elemento, o item 1.

```
<ul>
        <li class="lista">Item 1</li>
        <li class="lista">Item 2</li>
        <li class="lista">Item 3</li>
        <li class="lista">Item 4</li>
        <li class="lista">Item 5</li>
</ul>
```

document.**querySelectorAll**(“ul > .lista ”) irá retorar TODOS os elementos porém será necessário *iterar em laço,* nativamente pode-se usar o ***forEach( )***.

querySelectorAll - Exemplo

```
<ul>
        <li class="item">Item 1</li>
        <li class="item">Item 2</li>
        <li class="item">Item 3</li>
        <li class="item">Item 4</li>
        <li class="item">Item 5</li>
</ul>
```

```
const  itemlista  = document.querySelectorAll("ul > .item")

        itemlista.forEach( itens=>{
                itens.addEventListener('click',()=>{
                        alert(itens.textContent);
                })
        })
```

A constante **itemlista** armazenará todas as propriedades do elemento **li** presente em **ul** em uma **“lista”**
Para que se possa acessar as propriedades individuais de cada LI é necessário percorrer em um laço de repetição, nativamente usa-se o método **forEach** que irá receber uma função de callback.
O parâmetro itens irá receber os elementos enviados por forEach, apartir desse momento pode-se acessar os itens da lista individualmente.

Alguns métodos e interação com DOM

Onde estiver a “**el**” entenda qualquer elemento html.

el.: retorna atributos como type | value | name | id | class e outros
el.**innerHTML**: escreve ou retorna o texto. Se retornar traz estrutura html que possa estar no elemento.
el.**innerText**: escreve ou retorna todo o texto. Se retornar aplicações da classes internas são assumidas
el.**textContent**: escreve ou retorna todo o texto. Se retornar desconsidera qualquer tag e ou classe
el.**getAttribute**: captura um atributo - type - name - value etc.
el.**setAttribute**: adiciona um atributo dinamicamente
el.**appendChild**: adiciona um elemento filho no elemento pai
el.**classList**: add (adiciona uma classe)| remove(remove uma classe) | toggle (adiciona ou remove)
el.**type**: retorna o tipo do elemento
el.**key**: retonar o valor da tecla pressionada
el.**keyCod**:retorna o código da tecla pressionada
el.**style**: backgroundColor | border | margin | display ..etc.
el.**dataset:** *data attibutes* são tags atributos criadas e incorporadas no el htm e recuperadas como objeto

DATA ATTRIBUTE

Alguns métodos e interação com DOM

Os “data attributes” são “atributos de dados” que podem ser criados dentro de um elemento html.

Os atributos de um elemento como o input são limitados e não são personalizaveis em si mesmos. Mesmo com o uso de id ou classes por vezes é necessário incorporar algum dado no elemento que seja usual posteriormente.

Para atribuir um novo atributo bata usar o prefixo data-* onde o “*” aterisco irá representar qualquer nome que se possa referenciar, ex: **data-**codigo, **data-**idade, **data-**codigo e etc

<input type=”text” value=”” name=”nomeCliente” **data-codigoCliente**=”10” >

Para o JavaScripte “reconhecer” este atributo utiliza-se a referência dataset do elemento:

“elemento.dataset.codigoCliente”