AULA 2/4

JS: function

```
function add(x, y) {
    var total = x + y;
    return total;
}

Se nada for explicitamente retornado, o valor de retorno é
undefined.

Passagem de parâmetros:
    > add()
    NaN // You can't perform addition on undefined
    > add(2, 3, 4)
    5 // added the first two; 4 was ignored
```

JS: function

```
function add() {
    var sum = 0;
    for (var i = 0, j = arguments.length; i < j; i++) {
        sum += arguments[i];
    }
    return sum;
}
    > add(2, 3, 4, 5)
14
```

JS: function expression (funções anônimas)

- Acontece quando uma função é definida como o conteúdo de uma variável

```
var somaDoisNumeros = function (numero1, numero2) {
    return numero1 + numero2;
};
somaDoisNumeros(10, 20);
```

JS: function expression (funções anônimas)

```
var avg = function () {
    var sum = 0;
    for (var i = 0, j = arguments.length; i < j; i++) {
        sum += arguments[i];
    }
    return sum / arguments.length;
}

avg(5,10,5)
6.66666...
```

JS: funções com temporizador

```
// executa a minhaFuncao daqui um segundo
setTimeout(minhaFuncao, 1000);


// executa a minhaFuncao de um em um segundo
var timer = setInterval(minhaFuncao, 1000);


// cancela execução
clearInterval(timer);
```

JS: Template Literals

- Na hora de montar código JS é possível fazer uso do Template Literals
  - Este mecanismo ajuda na escrita de código de forma mais fluida

```
let var = `I counted ${3+4} sheep`;
```

- Tudo que estiver dentro do ${ } será interpretado como uma expressão JS.

- ATENÇÃO
  - É usado o character back-tick (`)
  - Se tentar utilizar aspa simples ou dupla, será lido como String.

JS: Spread

- Uso dos 3 pontinhos (...) in JS
- Algumas funções não aceitam que você passe arrays, sendo necessário passar os valores de forma individual
  - Por exemplo, a Math.max()
    - Dentro de um conjunto de valores, irá retornar o maior.
    - Math.max(1, 5, 77, 3, 55) □ Resultado: 77

```
const nums = [1,2,3,4,5]
Math.max(nums) // retorna NaN

Math.max(...nums) // retorna 5
// Mesmo que Math.max(1,2,3,4,5)
```

JS: Spread

- O operador spread (...) permite a iteração (como Array ou String) ser expandida em locais onde múltiplos elementos ou argumentos são esperados.

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const mergedArray = [...array1, ...array2];
// Retorna 1 Array:
// [1, 2, 3, 4, 5, 6]


const mergedArray = [array1, array2];
// Retorna 2 Arrays:
// [ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
```

JS: Spread

- Também permite concatenar elementos.

```
let cats = ['cat1', 'cat2'];
let dogs = ['dog1', 'dog2']
allPets = [...cats, ...dogs]
// [ 'cat1', 'cat2', 'dog1', 'dog2' ]
```

- Spread também pode ser usado em objetos, copiando as propriedades de um objeto para outro.

```
felino = {pernas: 4, familia: 'Felidae'}
novoObjeto = {...felino, cor: 'yellow'}
// { pernas: 4, familia: 'Felidae', cor: 'yellow' }
```

JS: Spread

- Também é usado para juntar dois objetos
  - let caninosFelinos = {...caninos, ...felinos}
    - Soma os atributos dos dois objetos em um novo

- Quando há atributos com o mesmo nome, o valor do último parâmetro ganha, pois substitui o anterior.

```
felino = {pernas: 4, familia: 'Felidae'}
canino = {dentes: 36, familia: 'Canidae’}

caninoFelino = {...felino, ...canino}

// { pernas: 4, familia: 'Canidae', dentes: 36 }
```

JS: arguments

- Dentro de toda função JS temos um atributo chamado arguments
  - O atributo arguments armazena todos os parâmetros que foram passados para a função
    - arguments[0] para acessar o primeiro parâmetro
    - arguments[1] para o segundo, etc

```
function fun(x, z){
    console.log(arguments)
    return x+z
}

fun(10, 165)
// [Arguments] { '0': 10, '1': 165 }
```

JS: arguments

- Ao utilizar diretamente o atributo arguments, algumas funções não estão disponíveis, como o .reduce()
  - Por isso, é uma boa prática alocar os parâmetros de entrada da função para uma outra variável

```
function fun(...nums){
    total = nums.reduce((total, valor) => total + valor)
    return total
}

console.log (fun(1, 2, 3, 4, 5))
```

JS: arguments

- Também é possível pegar alguns parâmetros e alocar o restante para a última variável

```
function fun(num1, num2, ...nums){
    console.log(`num1: ${num1}, num2: ${num2}. Outros numeros: ${nums}`)
   }

   console.log(fun(1, 2, 3, 4, 5))
```

JS: arrow function

- Há uma forma facilitada para declarar funções no JS... As arrow functions.

```
//ANTES
var oldWay = function (name, nickname) {
    return 'My name is ' + nickname + ', ' + name;
};

console.log(oldWay('James Bond', 'Bond'));
    // My name is Bond, James Bond
```

```
//COM ARROW FUNCTION
let newWay = (name, nickname) => {
    return 'My name is ' + nickname + ', ' + name;
};

console.log(newWay('James Bond', 'Bond'));
    // My name is Bond, James Bond
```

JS: arrow function

```
let newWay2 = (name, nickname) => 'My name is ' + nickname +
    ', ' + name;
console.log(newWay2('James Bond', 'Bond'));
// My name is Bond, James Bond
```

CLASSES

JS: classes

- JavaScript não possuía classes.
  - Atualmente há!!!

- Funcionalidade semelhante é obtida através de protótipos de objetos.
  - JavaScript usa funções como classes.
  - A palavra reservada **new** cria um novo objeto e o atribui a palavra chave **this** de dentro do escopo da função invocada.
    - Pode-se então adicionar atributos a esse objeto.

JS: construtores

```
function Person(first, last) {
    this.first = first;
    this.last = last;
    this.fullName = function () {
        return this.first + ' ' + this.last;
    };
    this.fullNameReversed = function () {
        return this.last + ', ' + this.first;
    };
}

var s = new Person("Lemmy", "Kilmister");
Console.log(s)
Console.log(s.fullName)
Console.log(s.fullName())
Console.log(s.fullNameReversed())
```

JS: construtores

```
var Pessoa = function (nome, email) {
    console.log("criando nova pessoa");
    console.log(typeof (this));
    this.nome = nome;
    this.email = email;
}

// criando nova pessoa
var joao = new Pessoa("João da Silva", "joao@da.silva");
console.log(joao.nome); // João da Silva
console.log(joao.email); // joao@da.silva
```

JS: construtores

```
var Curso = function (nome) {
    this.nome = nome;
    return "curso " + nome;
}


// Invocando como função
var stringParaCS01 = Curso("CS01");
typeof (stringParaCS01); // "string"
console.log(stringParaCS01); // curso CS01


// Invocando como construtor
var objetoParaWD47 = new Curso("WD47");
typeof (objetoParaWD47); // object
console.log(objetoParaWD47.nome); // WD47
```

JS: Classes modernas

- Classes com o *Class*

- No JavaScript moderno, você pode criar classes usando a palavra-chave class, introduzida no ES6 (ECMAScript 2015).

- Classes fornecem uma maneira mais clara e estruturada de criar objetos e trabalhar com herança, tornando o código mais legível e fácil de manter.

```
class NomeDaClasse {
  // Construtor: inicializa propriedades
  constructor(param1, param2) {
    this.propriedade1 = param1;
    this.propriedade2 = param2;
  }
```

JS: Classes modernas

```
class NomeDaClasse {
  // Construtor: inicializa propriedades
  constructor(param1, param2) {
    this.propriedade1 = param1;
    this.propriedade2 = param2;
  }
  // Método da instância
  metodoExemplo() {
    console.log(`Propriedade1 é: ${this.propriedade1}`);
  }
  // Método estático (acessado diretamente pela classe, não pela instância)
  static metodoEstatico() {
    console.log("Este é um método estático.");
  }
}

// Criando uma instância da classe
const instancia = new NomeDaClasse("valor1", "valor2");
instancia.metodoExemplo(); // Chamada do método da instância
// Chamando um método estático
NomeDaClasse.metodoEstatico();
```

JS: Classes modernas - Get e Set

```
class Produto {
  constructor(nome, preco) {
    this.nome = nome;
    this._preco = preco; // Convenção para propriedade "privada"
  }

  // Getter
  get preco() {
    return `R$ ${this._preco.toFixed(2)}`;
  }

  // Setter
  set preco(novoPreco) {
    if (novoPreco > 0) {
      this._preco = novoPreco;
    } else {
      console.log("O preço deve ser maior que zero.");
    }
  }
}

const produto = new Produto("Notebook", 2500);
console.log(produto.preco); // "R$ 2500.00"
produto.preco = 3000;
console.log(produto.preco); // "R$ 3000.00"
```

A depender da versão do JS, usa-se o #variavel para declarar uma variável privada.

Obs.: O Devtools possui uma permissão especial, se tentar acessar uma variável privada diretamente no Devtools você vai conseguir.
Teste em um arquivo HTML.

JS: Classes modernas - Herança

```
class Animal {
  constructor(nome) {
    this.nome = nome;
  }

  falar() {
    console.log(`${this.nome} faz um som.`);
  }
}

class Cachorro extends Animal {
  falar() {
    console.log(`${this.nome} late.`);
  }
}

const dog = new Cachorro("Rex");
dog.falar(); // "Rex late."

const animal = new Animal("Cobra")
animal.falar(); // "Cobra faz um som."
```

JS: protótipo

- Qualquer atributo ou função adicionado ao protótipo de uma dessas funções ficará disponível em qualquer objeto do tipo gerado por elas.

```
String.prototype.paraNumero = function () {
    if (this == "um") {
        return 1;
    }
}
console.log("um".paraNumero()); // 1
```

JS: protótipo

```
var Pessoa = function (nome, email) {
    this.nome = nome;
    // verifica se o e-mail foi preenchido
    if (email) {
        this.email = email;
    }
}


Pessoa.prototype.email = "contato@ufc.br"
var ricardo = new Pessoa("Ricardo");
console.log(ricardo.email); // contato@ufc.br


var joao = new Pessoa("Joao da Silva",
    "joao@da.silva");
console.log(joao.email); // joao@da.silva
```

JS: protótipo

```
var Pessoa = function (nome, email) {
    this.nome = nome;
    // verifica se o e-mail foi preenchido
    if (email) {
        this.email = email;
    }
};


Pessoa.prototype.fala = function () {
    console.log("Olá, meu nome é " + this.nome + " e meu email é
    “ + this.email);
    };


Pessoa.prototype.anda = function () {
    console.log("Estou andando");
};
```

EXERCÍCIO JS-Classes

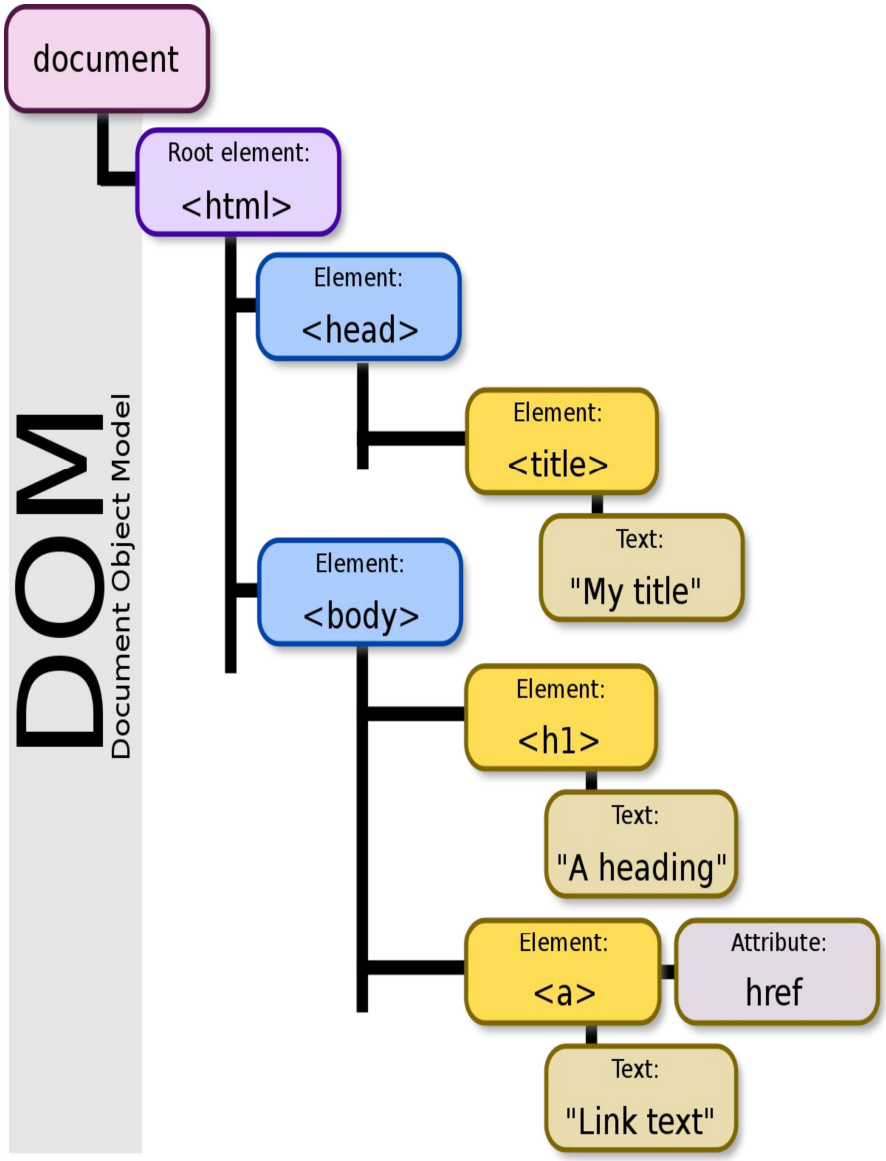
- Crie 3 classes no JavaScript usando Class: Animal, Gato, Cachorro e Pato. As classes Gato, Cachorro e Pato devem herdar da classe Animal.
- A classe Animal possui os atributos: nome e tutor. e os métodos: EmitirSom e Comer.
- Crie também os métodos Gets e Sets para cada atributo da classe, permitindo resgatar e alterar os valores dos atributos.
- Cada animal deve emitir o som respectivo, por exemplo “Au au”, “Miau” ou “Quack”.
- Instancie alguns objetos para testar as classes.

Document Object Model (DOM)

Document Object Model (DOM)

- O Document Object Model (DOM) é uma interface de programação que permite acessar os elementos HTML usando JavaScript. Com o acesso ao componente HTML é possível alterar estrutura, estilo e conteudo.

- No DOM a página web é representada como nós e objetos.



JS: Acessando DOM

- No console do chrome:
  - console.dir(document)
    - Para listar o objeto de forma visual
  - document.all
    - Listar todo o código HTML da página

DOM: Acessando DOM

- Formas de acessar os elementos HTML
  - document.getElementById
  - document.getElementsByClassName()
  - document.getElementsByName()
  - document.getElementsByTagName()

  - document.querySelector()
  - document.querySelectorAll()

DOM: getElementById()

```
<p id="tituloPrincipal">Este é o título principal da minha
página</p>

const elem = document.getElementById("tituloPrincipal");
elem.style.color = 'red';
elem.style.backgroundColor = 'blue'
```

ATENÇÃO !!!

Diferente do CSS, o JavaScript não permite hifens (-), por isso, alguns atributos possuem seu nome renomeado.

Por exemplo,
em JS usa-se **backgroundColor** ao invés de **background-color**

DOM: getElementsByClassName()

Permite selecionar todos os elementos que possuem uma ou mais classes.
Eles podem ser acessados como um array.

```
<span class="orange fruit">Orange Fruit</span>
<span class="orange juice">Orange Juice</span>
<span class="apple juice">Apple Juice</span>
<span class="foo bar">Something Random</span>
```

```
// getElementsByClassName somente seleciona os elementos que
possuem as duas classes (orange e juice)
const allOrangeJuiceByClass =
document.getElementsByClassName("orange juice");
let result = document.getElementsByClassName('orange juice');
result[0].style.color = 'red'
```

DOM: getElementsByName()

Permite selecionar todos os elementos que possuem o **atributo name** especificado.
Eles podem ser acessados como um array.

```
<body>
<input type="hidden" name="up" />
<input type="hidden" name="down" />
</body>

const up_names = document.getElementsByName("up");
console.log(up_names[0].tagName); // Resultado: "input"
```

DOM: getElementsByTagName()

Permite selecionar todos os elementos que pertencem a determinada **tag HTML**.
Eles podem ser acessados como um array.

```
const todosElementosTagP = document.getElementsByTagName("p");
const tamanho = todosElementosTagP.length;

console.log(tamanho);

console.dir(todosElementosTagP);

console.log(todosElementosTagP[0].innerText);
```

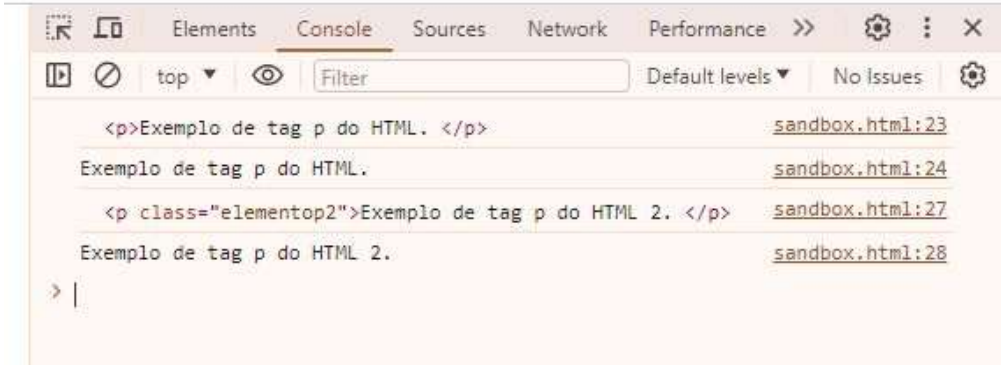
DOM: querySelector()

Permite selecionar **o primeiro** elemento que satisfaz a query, ou seja, o primeiro elemento que possui o seletor ou grupo de seletores especificados.
Deve-se utilizar a sintaxe do CSS.

```
<h1>Exemplo de site</h1>
<p>Exemplo de tag p do HTML. </p>
<p class="elementop2">Exemplo de tag p do HTML 2. </p>
```

```
let minhaPrimeiraTagP = document.querySelector("p");
console.log(minhaPrimeiraTagP);
console.log(minhaPrimeiraTagP.innerText);

let minhaTagPByClass = document.querySelector(".elementop2");
console.log(minhaTagPByClass);
console.log(minhaTagPByClass.innerText);
```



DOM: querySelector()

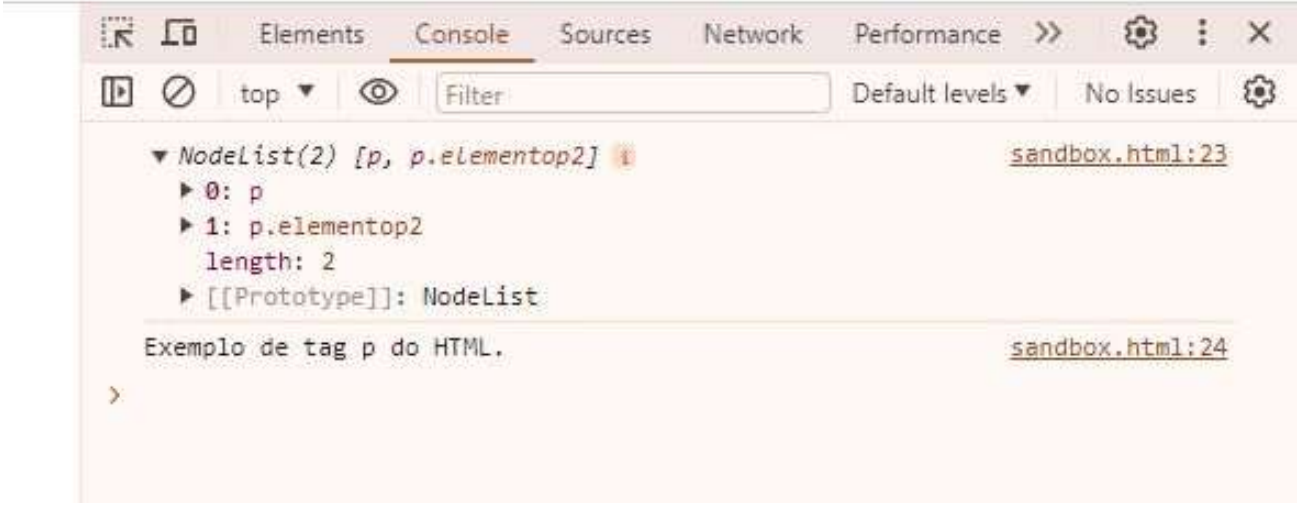
- Principais propriedades do objeto retornado no querySelector.

DOM: querySelectorAll()

Permite selecionar **TODOS** que satisfaz a query.
Deve-se utilizar a sintaxe do CSS.

```
<h1>Exemplo de site</h1>
<p>Exemplo de tag p do HTML. </p>
<p class="elementop2">Exemplo de tag p do HTML 2. </p>
```

```
let todasTagP = document.querySelectorAll("p");
console.log(todasTagP);
console.log(todasTagP[0].innerText);
```



DOM: querySelectorAll()

Outro exemplo

```
<p>Exemplo de tag p do HTML. <a href="http://exemplo...">meu
link</a> </p>
<p class="elementop2">Exemplo de tag p do HTML 2. </p>

todosLinksDentroDeP = document.querySelectorAll('p a')
// todos os elementos 'a' que estão dentro de tags 'p'

console.log(todosLinksDentroDeP);
console.log(todosLinksDentroDeP[0].href);
```

E mais um...

```
checkBox =
document.querySelector('input[type="checkbox"]')
//Seleciona o primeiro checkbox da pagina.

console.log(checkBox);
console.log(checkBox.checked);

//Se usar o querySelectorAll, não esquecer de acessar usando arrays.
checkboxs[0].checked.
```

JS: Outras formas de acessar os elementos

- document.getAttribute() e document.setAttribute()
  - É o mesmo que acessar meuObjeto.nomeAtributo.
  - Exemplo:
    - document.querySelector('img').src = 'xxx'
    - é o mesmo que meuObjeto.setAttribute('src', 'xxx').

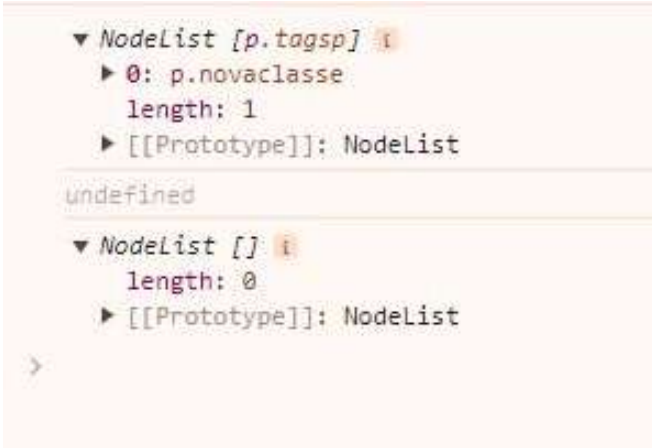
```
<p class="tagsp">Exemplo de tag p do HTML.</p>
<p class="elementop2">Exemplo de tag p do HTML 2. </p>
```

```
// Alterando a classe de um objeto
tagP = document.querySelectorAll('.tagsp')

console.log(tagP);
console.log(tagP.class);

tagP[0].setAttribute('class', 'novaclasse');
novaTagP = document.querySelectorAll('.tagsp')
console.log(novaTagP);

//Antes tinhamos um objeto p com a classe tagsp, agora não temos mais.
```



JS: Outros detalhes...

- Ao alterar o style de um elemento por JavaScript, é como se inseríssemos um *CSS in-line*, ou seja, individual para cada elemento.
  - Mas e se precisarmos redefinir atributos de vários elementos?

- Podemos incluir ou excluir classes de elementos usando o **classList**.
  - Adiciona ou remove classes de um objeto.
  - Ao adicionar uma classe, o elemento herda todas as características da classe.
  - Por exemplo, ".purple" define a cor purple para o objeto.
  - Para adicionar: **h2.classList.add('purple')**
  - Ou **h2.classList.toggle('purple').**
  - O toggle adiciona ou remove, se já tiver ele remove, se não tiver, ele adiciona.

JS: Outros detalhes...

- innerHTML
  - Texto em formato HTML
- innerText vs textContent
  - **innerText**: sensitivo ao contexto, só mostra o que está sendo exibido no momento da query
  - **textContent**: devolve tudo, o que está sendo visto ou não.
- Por exemplo, se definir um texto com "***display: none***", ele irá sumir do innerText, mas não do textContent.

Resumo:
funções, classes, construtores e seleção de elementos.

EXERCÍCIO

- Usando 2 <input>, receber preço e quantidade e calcular o valor total (preço * quantidade) e exibir o valor total abaixo.

- Quando um deles for alterado, alterar automaticamente o valor total.

EXERCÍCIO

- Crie uma página web que mostra uma lista de alunos usando <ul> e <li>. Após criar pelo menos 6 nomes de alunos, crie o código CSS e Javascript para alterar a cor do texto para vermelhos dos alunos que possuem uma posição par na lista.

Dica: use o querySelectorAll e o forEach(function(aluno, index))