Disciplina: Linguagem Web – javascript
Professora: Sandra Hedler de Amorim
E-mail: sandra-famorim@educar.rs.gov.br

Introdução a Linguagem JavaScript

JavaScript foi inicialmente conhecido como LiveScript, mas a Netscape mudou seu nome para JavaScript, possivelmente devido ao entusiasmo gerado pelo Java. O JavaScript apareceu pela primeira vez no Netscape 2.0 em 1995 com o nome LiveScript . O núcleo de propósito geral da linguagem foi incorporado ao Netscape, Internet Explorer e outros navegadores da web.

Javascript é uma linguagem de programação para web. JavaScript é muito fácil de implementar porque está integrado ao HTML. É aberto e multiplataforma. Até muito pouco tempo atrás o javascript era usado única e exclusivamente dentro do navegador, então todos os principais navegadores já vinham com o interpretador do javascript já pré-instalado dentro do próprio navegador. Então quando se acessava um site o que era carregado basicamente naquele era o HTML e o CSS, ou seja, tinha o interpretador HTML, CSS e interpretador javascript. Portanto, hoje em dia a máquina javascript como linguagem ela foi expandida um pouco, então agora temos acesso em codificar em javascript em outros locais que não somente nosso navegador. É uma linguagem de programação interpretada com recursos orientados a objetos.

Com tudo, o código é exatamente o mesmo, as funcionalidades são as mesmas e o que vai diferenciar é onde o javascript está rodando. Então no curso vai ser abordado com a expectativa que o javascript pode ser utilizado não somente no navegador, mas em vários outros tipos de dispositivos e equipamentos.

Existem muitas bibliotecas e estruturas Javascript úteis:

- Angular
- jQuery
- Node.js

Existem vários outros frameworks e bibliotecas Javascript disponíveis.

Aplicações de programação Javascript

Javascript é uma das linguagens de programação mais amplamente utilizadas (front-end, bem como back-end). Está presente em quase todas as áreas de desenvolvimento de software. Vou listar alguns deles aqui:

- Validação do lado do cliente - Isso é realmente importante para verificar qualquer entrada do usuário antes de enviá-la ao servidor e o Javascript desempenha um papel importante na validação dessas entradas no próprio front-end.
- Manipulando páginas HTML - Javascript ajuda na manipulação de páginas HTML em tempo real. Isso ajuda a adicionar e excluir qualquer tag HTML facilmente usando javascript e modificar seu HTML para alterar sua aparência com base em diferentes dispositivos e requisitos.

- Notificações do usuário - você pode usar Javascript para gerar pop-ups dinâmicos nas páginas da web para fornecer diferentes tipos de notificações aos visitantes do seu site.
- Carregamento de dados de back-end - Javascript fornece uma biblioteca Ajax que ajuda a carregar dados de back-end enquanto você está fazendo algum outro processamento. Isso realmente proporciona uma experiência incrível aos visitantes do seu site.
- Aplicativos de servidor - Node JS é construído no tempo de execução Javascript do Chrome para construir aplicativos de rede rápidos e escaláveis. Esta é uma biblioteca baseada em eventos que ajuda no desenvolvimento de aplicativos de servidor muito sofisticados, incluindo servidores da web.

Existem várias áreas onde muitos desenvolvedores de software estão usando Javascript para desenvolver ótimos sites e outros softwares.

Vantagens do JavaScript

Os méritos de usar JavaScript são:

Menos interação com o servidor - você pode validar a entrada do usuário antes de enviar a página para o servidor. Isso economiza o tráfego do servidor, o que significa menos carga no servidor.

Feedback imediato para os visitantes - Eles não precisam esperar o recarregamento da página para ver se esqueceram de inserir algo.

Maior interatividade - você pode criar interfaces que reagem quando o usuário passa o mouse sobre elas ou as ativa por meio do teclado.

Interfaces mais ricas - você pode usar JavaScript para incluir itens como arrastar e soltar componentes e controles deslizantes para fornecer uma interface rica aos visitantes do seu site.

ECMAScript 2015 (ou ES2015)

https://www.ecma-international.org/publications-and-standards/standards/

https://www.treinaweb.com.br/blog/qual-a-diferenca-entre-ecmascript-e-javascript/

https://www.revista-programar.info/artigos/ecmascript-2015-nova-versao-do-javascript/

Onde colocar o código JS?

O javascript pode ser executado no navegador de duas formas diferente, ou seja, pode organizar de duas formas distintas:

A primeira opção é utilizando uma tag chamada **script**.

Arquivo **index.html**

```
<!DOCTYPE html>
<html>
<head>
        <title> Curso Javascript </title>
```

```
        <link rel="stylesheet" href="style.css">
</head>
<body>
        <h1> Seja bem vindo(a) </h1>
        <script>
                document.write('Testando... javascript');
        </script>
</body>
</html>
```

O código escreve no documento (na própria página) a frase acima.

A segunda opção de escrever um código javascript é escrever em um arquivo separado, exatamente igual o CSS externo.

Exemplo: arquivo **script.js**

```
document.write('Opa, esta funcionando o js');

alert('Mais um exemplo para mostrar na tela uma janela...');
```

Arquivo **index.html**

```
<!DOCTYPE html>

<html>

<head>

        <title> Curso Javascript </title>

        <link rel="stylesheet" href="style.css">

        <script src="script.js"></script>

</head>

<body>

        <h1> Seja bem vindo(a) </h1>

</body>

</html>
```

Porém, qualquer coisa que você põem dentro da tag head, é a primeira coisa que é carregada quando sua página é carregada dentro do seu navegador. Conforme exemplo acima, vai dar um título para página carregar o CSS e depois o script.js e por último vai para a tag body e mostrar o conteúdo efetivamente.

Então o script inserido dentro do head se faz quando quer que o script seja executado antes da página.

Mas, nem todo caso precisa disso, porque, quanto mais coisas se colocar dentro do head, obrigatoriamente ele vai carregar antes do body.

Então quanto mais coisas colocar no head, mais lento o seu site vai carregar.

Então, pode-se tirar o javascript do head e colocar no fim do body conforme exemplo a seguir:

```
<!DOCTYPE html>

<html>

<head>

    <title> Curso Javascript </title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1> Seja bem vindo(a) </h1>

    <script src="script.js"></script>

</body>

</html>
```

Ou seja, primeiro vai carregar o que estiver dentro do head e depois pula para o body e carrega todo conteúdo que estiver dentro e por fim carrega o javascript, isso qualquer outra coisa que o javascript venha fazer, ele ira fazer depois que o site já esta carregado.

Isso normalmente, em 90% dos casos, é o que vai ser suficiente e benéfico para o usuário que esta acessando o site.

Então, o recomendado é colocar o javascript no final do body, por que não vai atrapalhar e executar tudo da melhor maneira.

Uma das funções do javascript é manipular os elementos que estão na tela, dentre eles, o h1 conforme exemplo.

Como usar o console?

O javascript funciona no google chrome, firefox e em qualquer outro navegador. O javascript ele vem junto com o próprio navegador, isso significa que pode executar javascript no próprio navegador, como fazer isso?

Executar, botão direito do mouse e ir em inspecionar elementos. Vai abrir varias abas como elements, console, sources...

A primeira mostra os elementos que estão sendo reinderizados na tela.

A segunda aba console, nela vão aparecer desde avisos no próprio javascript, também vão aparecer mensagens específicas, alertas de algumas coisas, e também a possibilidade de roda o próprio código javascript. Então não necessariamente tenha que criar um arquivo e executar. O programador pode simplesmente ir no console e executar o seu próprio código javascript, claro que não vai ficar salvo, quando atualizar a tela perde o código e logo a seguir o ponto de inserção fica a baixo pronto para o usuário se quiser digitar.

Então nesta área podem-se digitar códigos para testar e também ver o resultado de um código que foi executado, ou seja, possibilitando ver os erros.

Formas de mostrar (Output)

Quanto está digitando códigos javascript, tem quatro opções de mostrar informações para usuário final. Quais são essas opções?

Existe varias opções, mas as quatro principais são:

Primeira é você manipular elementos do HTML que está no arquivo, exemplo, você colocar um id neste elemento que se quer manipular:

```
<!DOCTYPE html>

<html>

<head>

    <title> Curso Javascript </title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1 id="titulo"> Seja bem vindo(a) </h1>

    <script src="script.js"></script>

</body>

</html>
```

E no arquivo script.js se faz a alteração para esse elemento de id chamada de titulo.

Exemplo:

```
document.getElementById("titulo").innerHTML="opa, tudo bem ?";
```

Nas próximas aulas, vai ser explicado com detalhes essa estrutura. Agora é só uma amostra de como é uma das formas.

Conforme exemplo, já houve uma manipulação de um elemento que já esta na tela.

A segunda, forma de fazer isso e usando o document.write, exemplo:

document.write("Algum texto");

Esse comando ele funciona e mantém outras coisas que estavam na tela quando esta sendo iniciado junto do código inicial. Simplesmente adicionou a tudo que já estava na tela inicial.

A primeira forma é manipulando um elemento que já está, a segunda forma, criando seu próprio elemento e a **terceira forma utilizando a janela do navegador**. Então tem que saber diferenciar a página e janela.

A página é todo conteúdo do seu site, para o javascript ele é o **document**, que é o documento basicamente. Mas também tem a apropria janela do navegador e o javascript consegue manipular ela, através do comando **window**, que vem de janela. Temos esses dois elementos diferentes.

Então tem coisas para mostrar para o usuário através da janela e através do documento. Vimos duas formas de mostrar coisas através do documento. E agora como mostrar através da janela:

window.alert("Mensagem de exemplo");

Esse resultado desse código (a janela), que mostra é a janela do navegador, ela não tem relação com o documento em si, mas sim com a janela do seu navegador como um todo.

Entao, pode-se utilizar window.alert() ou como atalho, só alert();

Então o document, precede da página, ou seja, altera alguma coisa na página e window se esta alterando alguma coisa na janela.

Temos uma quarta opção que se pode mostrar alguma coisa que é o console. Mas ele não é nem a janela, nem o documento.

No qual podemos ver erros, avisos, mensagens e etc.

console.log("A mensagem que inserida aqui, aparecerá no console");

Essas são 4 principais formas que tenho de exibir informações para o usuários.

Então quando quero exibir uma informação para o desenvolvedor, mas não quero exibir para o usuário final utilizo o console.

Então quando quero exibir alguma coisa para o usuário, ou se altera o documento ou window que é a janela ou alert().

Essas são dicas importantes para você aluno entender quando estiver desenvolvendo, saber por que esta utilizando um e outro num determinado momento.

Variáveis

Criar variáveis no javascript precisa declarar var e em seguida o nome dessa variável e um igual e o valor que se quer, exemplo:

var idade = 20;

A partir disto pode-se utilizar essa variável quando desejar.

alert(idade);

Quando esta declarando variável tipo texto coloca-se entre aspa simples ou aspas duplas, exemplo:

var nome = "Sandra";

alert(nome);

Nos casos acima entendemos como armazena variáveis, e a partir de agora como utiliza-las.

var x = 10;

var y = 20;

var total = x + y;

alert(total);

Podemos ver executando esse código com o operador de adição que fez a soma de 10 + 20, mostrando o valor 30 armazenado na variável total.

var nome = "Sandra";

var sobrenome = "Amorim";

var nomecompleto = nome + sobrenome;

alert(nomecompleto);

Aqui na variável nomecompleto, o operador de adição, como as variáveis envolvidas são do tipo texto, ele não faz uma adição e sim, juntar, o que chamamos de concatenação.

Para mostrar nome e sobrenome entre espaço podemos fazer o seguinte:

var nomecompleto = nome+" "+sobrenome;

alert(nomecompleto);

Outro exemplo:

var nome = "Sandra";

var idade = 30;

var total = nome + idade;

```
alert(total);
```

O resultado de saída do comando irá juntar, concatenar nome e idade. O javascript quando vê que um deles é do tipo texto não faz uma operação matemática neste caso, e sim uma concatenação.

```
var x = "10";

var y = "20";

var total = x + y;

alert(total);
```

Conforme valores das variáveis no exemplo anterior, elas não são mais do tipo numéricos e sim tipo texto por estarem entre aspas(" ");

Condicionais

É fazer com que o código seja executado de acordo com uma condição específica. É uma técnica primordial para qualquer linguagem que for aprender.

Para fazer uma condição, vamos criar uma variável chamada hora:

```
var hora = 10;
```

Vamos fazer com que dependendo desse horário apareça uma determinada condição, como

```
console.log("Bom dia");
```

ou

```
console.log("Boa noite");
```

dependendo do que esta nesta variável, entendeu.

```
if(hora < 12){
        console.log("Bom dia");
} else{
        console.log("Não é de manhã");
}
```

A segunda etapa o else, é opcional, usa se quiser, quando se quer, caso a condição não for satisfeita, que execute um determinado comando.

Outro exemplo:

```
var hora = 10;
```

```
if(hora < 12){
        console.log("Bom dia");
}

if(hora < 18){
        console.log("Boa tarde");
}

if(hora <= 23){
        console.log("Boa noite");
}
```

Conforme esse exemplo vai ser executados as três saídas, Bom dia, Boa tarde e Boa noite. Pois foi utilizado as três condições independentes.

Então, existe alternativas para não executar essas três opções, que é o que chamamos de else if.

```
var hora = 10
if(hora < 12){
        console.log("Bom dia");
} else if(hora < 18){
        console.log("Boa tarde");
} else if(hora <= 23){
        console.log("Boa noite");
}
```

Agora vamos para um próximo passo, que é uma condicional um pouco mais avançada.

Podemos fazer duas ou mais condições em uma condicional. Exemplo de como fazer duas condições em um if somente.

```
var hora = 14;
if (hora >=12 && hora <18){
```

```
        console.log("Boa tarde");
}
var hora = 12;


if(hora == 12 || hora == 18){
        console.log("Você está na hora do rush");
}
```

Variáveis

Vamos ver algumas alterações que houve a partir do ECMAScript de 2015, que foi lançado neste mesmo ano.

E claro que leva alguns anos para todos os navegadores atualizarem, só recentemente que essas variações foram utilizadas.

Você aprendeu até aqui definição de variáveis como:

```
var nome = "Sandra";
```

Agora vamos aprender definir variáveis de outros dois tipos pelo menos:

```
let nome = "Sandra";
```

```
const nome = "Sandra";
```

Qual é a diferença desses três tipos?

Quando define uma variável usando o var, acontece um processo chamado HOISTING, que essa variável vai ser enviado para o escopo geral da sua tela, no caso do seu javascript. Inclusive consegue utilizar essa variável, usando o window, uma variável global.

Exemplo:

```
var nome = "Sandra";

console.log(window.nome);
```

Ela funciona, porque ela foi transferida para o escopo geral do site. Isso significa que tem acesso a ela em qualquer área do seu sistema. Isso é bom, mas por um lado tem seus lados negativos.

Agora uma comparação ao let, utilizando a mesma variável.

```
let nome = "Sandra";
```

Então quando definimos a variável com o let, ela fica disponível apenas naquele escopo específico de código, naquela área específica de código que fica disponível a variável.

Se você tentar acessar ela em outro local, você não vai conseguir acessar, e muito menos com o window.

```
console.log(window.nome);
```

Executando, vai retornar como variável não definida.

Um outro exemplo claro para você entender:

```
var nome = "Sandra";

if(nome=="Sandra"){

        var idade = 50;

}

console.log(idade);
```

Mudando o valor da variável:

```
var nome = "Ana";

if(nome=="Sandra"){

        var idade = 50;

}

console.log(idade);
```

Vai dar como variável definida.

Mudando agora de var para let:

```
var nome = "Sandra"; //aqui também se quiser mudar para let, mas não vai interferir, e sim, principalmente em idade.

if(nome=="Sandra"){

        let idade = 50;

}

console.log(idade);
```

Executando, vai dar como variável idade não definida.

Isso não quer dizer que não entrou no if, isso mostra que ficou visível somente dentro do if essa variável. Ou seja, somente neste bloco de código.

Para executar essa variável idade, devemos fazer a seguinte alteração no código.

```
var nome = "Sandra";

if(nome=="Sandra"){

        let idade = 50;

        console.log(idade);

}
```

Essa, é a principal diferença entre o var e o let. O let só vai ser disponível naquele escopo específico.

Isso possibilita economia de memória, pois acaba de executar esse bloco, elas são tiradas da memória.

Quanto declaro assim:

```
var nome = "Sandra";

var nome = "Ana";

console.log(nome);
```

Executando, irá mostra Ana como saída.

Se fizermos isso com o let por exemplo

```
let nome = "Sandra";

let nome = "Ana";

console.log(nome);
```

Executando, irá mostrar que o identificador nome já foi declarado.

Então, quando declarar com o let, além do fator global do escopo, a variável, ela não pode ser redeclarada, ou seja, repetida no mesmo local, ou bloco.

Se você quiser conforme esse exemplo definir outro valor para o nome, poderá fazer o seguinte:

```
let nome = "Sandra";

nome = "Ana";
```

console.log(nome);

Essas são umas das diferenças de var e let.

E o const ?

O const como o nome já diz, estou declarando uma variável de valor constante.

const nome = "Sandra";

Não é uma constante mesmo, o que veremos mais tarde, que é outro tipo de variável específica.

Mas neste caso podemos até chamar de constante, com tudo é uma variável de valor constante.

const nome = "Sandra";

console.log(nome);

Com tudo agora, eu não posso modificar o valor dessa constante, exemplo:

const nome = "Sandra";

nome = "Ana";

console.log(nome);

Executando, mostra que a variável constante não pode receber um novo valor.

Uma vez definida fica com o mesmo valor.

A única exceção, é para arrays e objetos.

Exemplo, de um objeto:

const nome = {nome:'Sandra', sobrenome:'Amorim'};

console.log(nome);

Se quiser alterar o nome por exemplo, pode ser feito assim:

const nome = {nome:'Sandra', sobrenome:'Amorim'};

nome.nome = 'Ana';

console.log(nome);

O que não pode é alterar a estrutura da constante nome, por exemplo:

const nome = {nome:'Sandra', sobrenome:'Amorim'};

nome = {idade:20, hooby:'codigo'};

console.log(nome);

Porque, com isso esta redefinindo a variável com um valor totalmente diferente.

Ela é útil quando se está usando array e objetos. Porque, apesar de poder trocar os valores dentro dos arrays e objetos, você não pode trocar um objeto como todo.

Um outro exemplo:

```
var pessoa = {nome:'Sandra', sobrenome:'Amorim'};

pessoa = "Ana";

console.log(pessoa);
```

Conforme exemplo utilizando var, no momento que redefinir a variável, esta se trocando todo o nome e sobrenome pelo valor Ana.

Neste caso, trocamos uma variável que será um objeto por uma string.

Então tudo isso é possível com o var, já com o const não será possível.

```
const pessoa = {nome:'Sandra', sobrenome:'Amorim'};

pessoa = "Ana";

console.log(pessoa);
```

Esse exemplo já irá apontar erro. Isso de uma forma geral protege de você de forma acidental de modificar essas informações, o que é interessante.

Então a recomendação a partir de agora é você não usar mais var, e sim, utilizar let ou const dependendo da situação.

Então se você quer uma variável que vai ficar modificando o valor, usa let. Se você quer uma variável constante como o próprio nome já diz, usa o const.

Tipos de dados

Tipos de objetos que existe no javascript:

Quando declaramos var nome = "Sandra"; e dentro dela existe uma string, ou seja, existe um texto.

Quando declaramos var idade = 30; estamos declarando uma variável chamada idade com um valor do tipo numérico.

Existem outros tipos, por exemplo, quando você quer saber se uma determinada coisa está ligada ou desligada, sim ou não, ok ou não ok. existe o que chamamos de variáveis boolean, exemplo se você quer saber se sim ou não:

```
var salvo = true;
```

ou

var salvo = false;

Isso é o chamado valor boolean, que nada mais é sim ou não.

Porém, existe outras como por exemplo, array, objeto e funções, vários tipos específicos.

Mas como identificar um tipo específico?

Exemplo:

var cidade;

Quando não tem valor a variável, o tipo dela é indefinido ou undefined.

Mas vamos ver agora como ver o tipo delas:

var nome = "Sandra";

var idade = 30;

var salvo = true;

var cidade;

Usando o auxilio do console:

console.log(nome);

vai mostrar Sandra

console.log(idade);

vai mostrar 30

console.log(salvo);

vai mostrar true

console.log(cidade);

vai mostrar undefined, ou seja, indefinida.

Então, como sabemos qual é o tipo de uma determinada variável?

typeof nome;

irá retornar string, ou seja, é uma string.

typeof idade;

irá retornar number, ou seja, é um número.

typeof idade;

irá retorna boolean, ou seja, true ou false.

typeof cidade;

irá retornar undefined, ou seja, indefinida.

E existem outros tipos que será visto de uma forma mais avançada como objetos, arrays. Ainda existe um typeof tipo null, que basicamente é nulo. Que é muito comparável com o undefined.

Então, para verificar um tipo de uma variável utiliza o typeof.

Observe a seguir:

var tipo = typeof nome;

Neste caso, criou uma variável chamada tipo, para armazenar um valor que é o tipo da variável nome.

Então, não irá armazenar o valor, que bastaria digitar a linha de código assim:

var tipo = nome;

Para ver o tipo de dados que a variável utiliza se faz

var tipo = typeof idade;

console.log(tipo);

Comentários:

É uma forma de dizer, explicar, o que determinado código esta fazendo dentro do sistema.

São códigos que serão completamente ignorados pelo javascript. Serve única e exclusivamente para o programador entender o que está acontecendo nesta determinada linha ou linhas de códigos.

Existe 2 tipos de comentários que pode ser feito no javascript:

De uma linha e de múltiplas linhas.

Comentário de uma linha, coloca-se duas barra (//):

// definição de variáveis

var nome = "Sandra"; //nome do usuário

// pegando o TIPO da variável idade

var tipo = typeof idade;

Para comentários de múltiplas linhas, onde se pode estender um pouco mais a explicação no código mais detalhadas.

Coloca-se barra asterisco no início do comentário e asterisco barra ao final, exemplo:

/* primeira linha de comentário

e mais uma linha

e outra

e quantas outras quiser

*/

getElementById innerHTML

Comando que pega um elemento da tela através do Id desse elemento.

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8"/>
	<title> Curso Javascript </title>
	<link rel="stylesheet" href="style.css">
</head>
<body>
	<h1 id="titulo"> Seja bem vindo(a) </h1>
	<input id="campo" type="text" name="usuario" value = "Nome do usuário"/>
	<p id="paragrafo">Conteúdo qualquer... </p>
	<script src="script.js"></script>
</body>
</html>
```

//nome do arquivo externo abaixo

script.js

Comando a seguir pega dentro do documento (página) o elemento pelo id dele. E dentro dele podemos trabalhar com esse elemento com o innerHTML. O exemplo a seguir adiciona um valor, se esse valor for um número, por exemplo, adiciona direto sem aspas duplas.

document.getElementById("titulo").innerHTML = "olá mundo";

Ou pode-se fazer da seguinte forma: Selecionar o elemento, e armazenar esse elemento em uma variável, e a partir daí usa essa variável para alterar quando quiser.

Exemplo:

var titulo = document.getElementById("titulo");

titulo.innerHTML = "Outro título";

Com isso dá para entender como funciona esse processo.

O próximo exemplo ensina como manipular o elemento de id chamado de campo, ou seja, a tag input do arquivo html.

var campo = document.getElementById

Observação: a forma de manipular um valor de um campo é diferente de uma tag como o exemplo anterior no h1. No campo temos uma value, então, a forma de alterar é diferente.

Então, quando se quer alterar o ***conteúdo interno*** de uma tag usa o **innerHTML**. Quando se quer alterar o ***valor específico*** de um elemento usa o **value**.

campo.value = "Sandra";

Agora, quando tem uma tag qualquer, como h1, div, ou p, se usa o innerHTML

document.getElementById("paragrafo").innerHTML = "Novo conteúdo";

Ou então. salva esse conteúdo numa variável conforme exemplo a seguir:

var paragrafo = document.getElementById("paragrafo");

paragrafo.innerHTML = "Novo conteúdo";

Os exemplos anteriores, servem para mostrar o comando getElementById, innerHTML e value.

Funções (1/2)

Funções nada mais é que um conjunto de códigos que criamos e utilizamos quando quiser.

```
function alterar(){  //abre colchetes
        document.getElementById("titulo").innerHTML = "trocou o titulo";
        document.getElementById("campo").value = "trocou o campo";
}  //fecha colchetes
```

O comando a seguir mostra como utilizar essa função.

```
alterar();
```

Funções (2/2):

Nos exemplos anteriores usamos o comando alert(), que nada mais é que uma função.

Uma função pode ter parâmetros, que vai entre os parênteses.

Exemplo:

```
function alterar(titulo){
        document.getElementById("titulo").innerHTML = "trocou o titulo";
        document.getElementById("campo").value = "trocou o campo";
}
```

O parâmetro na função é titulo, que pode ter vários parâmetros em uma função.

E esse parâmetro se transforma em uma variável dentro da função.

Exemplo:

```
function alterar(titulo){
        document.getElementById("titulo").innerHTML = titulo;
        document.getElementById("campo").value = titulo;
}
```

Para passar um valor para esse parâmetro chamado titulo é muito simples. Usa a função e entre aspa, dependendo do tipo de valor.

Exemplo:

```
alterar("Titulo de exemplo");
```

O exemplo a seguir, mostra como criar uma função com mais de um parâmetro:

```
function somar(x,y){
        // var total = x + y; // se usar var, pode-se utilizar essa variável fora da função.
        let total = x + y; // com let o total é no local, dentro da função.
        document.getElementById("campo").value = total;
}

// console.log(total);
```

A função somar, está recebendo dois números e somando esses dois números e armazenando na variável total.

E logo depois usando essa variável total para exibir em algum lugar. No exemplo a variável

total armazena a soma desses dois parâmetros, colocando o resultado no elemento da página de id chamado campo.

Podem-se usar quantas funções quiser em um arquivo, e só utilizar a função que se quer.

Na função soma, conforme exemplo acima, se não quiser que mostre em um determinado campo e sim, somente mostrar o resultado desses dois números, pode-se fazer assim:

```
function somar(x,y){
        let total = x + y; // com let a variável total é local, dentro da função.
        return total; // vai retornar o valor que esta na variável total
}

var resultado = somar(10, 15);
console.log(resultado);
```

Usando a função acima de outras formas diferentes:

Evento de clique

Eventos na linguagem javascript representa a linguagem reagir a um determinado acontecimento.

Por exemplo, o ato de clicar em um botão significa um evento. Então, o javascript reage a esse evento.

Existe evento de mouse, que é, quando passa o mouse em cima de alguma coisa, da própria tela. O navegador está disparando acontecimentos, ou seja, disparando acontecimentos. E o javascript pode atrelar códigos específicos a esses eventos.

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8"/>

    <title> Curso Javascript </title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1 id="titulo"> Seja bem vindo(a) </h1>

    <input id="campo" type="text" name="usuario" value = "Nome do usuário"/>

    <p id="paragrafo"> Conteúdo qualquer... </p>

    <button onclick="clicou()"> Clique aqui... </button> //onclick, igual ao clicar, mostra quando o usuário clicar no botão executa determinado código

    <script src="script.js"></script>

</body>

</html>
```

A seguir, no arquivo script.js, tem a função clicou, que será executada conforme o exemplo acima do evento onclick.

```
function clicou(){

    //alert("Você clicou no botão");

    document.getElementById("titulo").innerHTML = "Obrigado";

}
```

Observação: Essa não é a única forma de executar eventos de onclick. Pode-se executar coisas dentro do onclick, ou seja, direto no elemento com esse evento, conforme exemplo:

<button onclick = "document.getelementById('titulo').innerHTML = 'Titulo qualquer...'"> Clique aqui... </button>

Ou seja, quando o usuário clicar no botão não vai executar uma função, mas sim um código javascript direto.

Outra possibilidade, é alterar o próprio botão, conforme o exemplo a seguir:

<button onclick = "this.innerHTML = 'Clicou'"> Clique aqui... </button>

A propriedade chamada this significa o próprio botão. Ou seja, é como utilizar uma id com getElementById().

O evento, em geral, não precisa ser em botões, pode-se aplicar em qualquer elemento HTML.

Exemplo:

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8"/>

    <title> Curso Javascript </title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1 id="titulo" onclick = "alert('Clicou')"> Seja bem vindo(a) </h1>

    <input id="campo" type="text" name="usuario" value = "Nome do usuário"/>

    <p id="paragrafo">Conteúdo qualquer... </p>

    <script src="script.js"></script>

</body>

</html>
```

Como utilizamos o evento de passar o mouse?

Exemplo:

```
<!DOCTYPE html>

<html>

<head>

        <meta charset="utf-8"/>

        <title> Curso Javascript </title>

        <link rel="stylesheet" href="style.css">

</head>

<body>

        <h1 id="titulo" onmouseover = "console.log('passou o mouse')"> Seja bem vindo(a) </h1>

// onmouseover, quando o mouse estiver sobre o elemento


        <input id="campo" type="text" name="usuario" value = "Nome do usuário"/>

        <p id="paragrafo">Conteúdo qualquer... </p>

        <script src="script.js"></script>

</body>

</html>
```

Outros eventos de mouse:

onmouseout = quando tirar o mouse

Podemos combinar eventos:

```
<h1 id="titulo" onclick="console.log('clicou')" onmouseover="console.log('passou o mouse')" onmouseout = "console.log('tirou o mouse')"> Seja bem vindo(a) </h1>
```

Esses são os eventos de clicks.

Evento de teclados:

onkeydown

onkeypress

onkeyup

Qual a diferença ?

onkeydown = evento vai acontecer quando pressionar a tela. Down, igual pra baixo.

onkeyup = evento acontece quando o usuário solta a tecla.

```
<input onkeyup = "digitou()" id="campo" type="text" name="usuario" value = "Nome do usuário"/>
```

```
function digitou(){
    console.log("você digitou");
}
```

Observação: Um detalhe muito importante, talvez o mais importante que onkeydown ou onkeyup é saber o que o usuário digitou. Como se faz? Sabemos que a função recebe parâmetros, então ela pode receber o próprio evento.

Exemplo:

```
function digitou(e){  //geralmente usa só o e, para representar que é um evento, mas poderia ser function digitou(evento)
    console.log(e);
}
```

A função tem um parâmetro para ser recebido. E esse parâmetro é enviado conforme exemplo abaixo:

```
<input onkeyup = "digitou(event)" id="campo" type="text" name="usuario" value = "Nome do usuário"/>
```

O comando event, trata do próprio evento, assim como comando this.

No momento que o usuário digitar, vai ocorrer um KeyboardEvent, ou seja, um evento de teclado, e mostra vários detalhes específicos desse evento, incluindo a numeração da tecla pressionada, ou seja, o keyCode.

No exemplo a seguir, só vai mostrar o console.log() se o usuário digitar a tecla ENTER.

```
function digitou(){
	if(e.keyCode == 13){ // tecla enter
		let texto = document.getElementById("campo").value;
		console.log(texto);
	}
}
```

A seguir, outro exemplo de só aparecer o console.log se o usuário digitar control + enter

```
function digitou(){
	if(e.keyCode == 13 && e.ctrlkey == true){ // tecla enter
		let texto = document.getElementById("campo").value;
		console.log(texto);
	}
}
```

Mudança de estilos

Usa-se muito em projetos reais, possibilita fazer muitas coisas legais. É uma das coisas mais importante para se fazer com javascript. Possibilita interação entre javascript com eventos do CSS e objetos.

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8"/>
	<title> Curso Javascript </title>
	<link rel="stylesheet" href="style.css">

</head>
<body>
	<h1 id="titulo" class="vermelho oculto"> Seja bem vindo(a) </h1>
```

```
        <input id="campo" type="text" name="usuario" value = "Nome do usuário"/>
        <p id="paragrafo">Conteúdo qualquer... </p>
        <script src="script.js"></script>
</body>
</html>
```

Exemplo do arquivo style.css

```
.vermelho {
        background-color: #ff0000;
        color: #fff;
}
.oculto {
        display: none;
}
.azul {
        background-color: #00ff00;
        color: #fff;
}
.verde {
        background-color: #00ff00;
        color: #000;
}
```

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8"/>
        <title> Curso Javascript </title>
        <link rel="stylesheet" href="style.css">
```

```
</head>
<body>
	<h1 id="titulo" class="vermelho oculto"> Seja bem vindo(a) </h1>
	<button onclick="azul( )"> Azul </button>
	<button onclick="vermelho( )"> Vermelho </button>
	<button onclick="verde( )"> Verde </button>
	<script src="script.js"></script>
</body>
</html>
```

No arquivo script.js esta as funções:

```
function azul( ) {
	limpar();
	document.geElementById("titulo").classList.add('azul');
}
function vermelho ( ) {
	limpar();
	document.geElementById("titulo").classList.add('vermelho');
}
function verde( ) {
	limpar();
	document.geElementById("titulo").classList.add('verdel');
}
function limpar( ) {
	document.geElementById("titulo").classList.remove('azul');
	document.geElementById("titulo").classList.remove('vermelhol');
	document.geElementById("titulo").classList.remove('verde');
}
```

Próximo exemplo:

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8"/>
	<title> Curso Javascript </title>
	<link rel="stylesheet" href="style.css">
</head>
<body>
	<h1 id="titulo"> Seja bem vindo(a) </h1>
	<div id="telefone"> 999999999 </div>
	<button onclick="mostrarTelefone(this)"> Mostrar telefone </button>
	<script src="script.js"></script>
</body>
</html>
```

No arquivo **script.js**

```
function mostrarTelefone(elemento ){
	elemento.style.display = 'none';
	document.getElementById("telefone").style.display = "block";
}
```

E no arquivo **style.css**:

```
#telefone {
	display: none;
}
```

Para praticar essas mudanças de estilos, sugiro que fazer um botão que ao clicar mostre um texto, botão para mostrar site ou uma div que ao clicar um botão mostre um site nela.

Array

É um tipo de variável que possibilita armazenar vários valores, as chamadas variáveis compostas.

```
let carros = ['Palio', 'Uno', 'Corolla', 'Ferrari'];


console.log(carros);
```

Se quiser mostrar a primeira propriedade (elemento) do array carros?

```
console.log(carros[0]);
```

O exemplo a seguir mostra como até um tempo atrás se usava para declarar array.

```
let carros = new Array('Palio' 'Uno', 'Corolla', 'Ferrari');
```

Também pode ver arrays sendo montado assim, eventualmente, um pouco mais espaçado:

```
let carros = [
        'Palio',
        'Uno',
        'Corolla',
        'Ferrari'
];
```

Pode-se usar qualquer tipo de dado em um array, incluindo funções e array dentro de array.

Exemplo:

```
let carros = [
        'Palio',
        'Uno',
```

```
	'Corolla',
	'Ferrari',
	function() {
		console.log('Testando 1, 2, 3...');
	}
];
```

Para executar essa função dentro do array carros, faz da seguinte forma:

```
carros[4]();
```

Outro exemplo de array dentro de array:

```
let ingredientes = [
	['uva', 'pera', 'maça'],
	['arroz', 'macarrão']
];

console.log(ingredientes);
```

ou selecionando um determinado elemento dentro de um array ou arrays:

```
console.log(ingredientes[1]);
```

ou

```
console.log(ingredientes[1][0]);
```

Objeto (1/2)

Objetos são muitos similares a arrays. A diferença é que um array é uma listagem numerada, e um objeto é uma listagem nomeada.

Num array, praticamente pode-se pegar um determinado item, pois são todos numerados.

Um objeto, são itens nomeados, onde específica o nome e o valor do item que se quer colocar.

Exemplo:

```
let carro = {
        nome:'Fiat',
        modelo:'Uno',
        peso:'800kg'
};
```

Dessa forma, foi criado o objeto, com três propriedades. No caso de um objeto, quando se quer acessar um valor não se faz como um array

```
console.log(carro[0]);
```

pois esse objeto não é numerado, e sim nomeado.

Exemplo de como acessar um valor no objeto:

```
console.log(carro['nome']);
```

ou assim:

```
console.log(carro.nome);
```

Agora você vai lembrar a referência das aulas anteriores como:

```
document.getElementById("titulo").style.display = "block";
```

ou seja, tem um objeto chamado document, dentro dele existe uma função chamada getElementById.

Assim como no array pode-se criar uma função dentro dele.

Exemplo:

```
let carro = {
        nome:'Fiat',
        modelo:'Uno',
        peso:'800kg',
        ligar:function(){  //não coloca-se o nome após function, pois o nome dela é ligar.
                console.log("VRUM VRUM!");
        }
};
```

Para acessar a função ligar:

```
carro.ligar();
```

Acrescentando mais uma função no objeto carro:

```
let carro = {
        nome:'Fiat',
        modelo:'Uno',
        peso:'800kg',
        ligar:function(){ //não coloca-se o nome após function, pois o nome dela nesse caso está antes, que é ligar.
                console.log("VRUM VRUM!");
        },

        acelerar:function(){
                console.log("Riiiiiihhhin");
        }
```

```
};


console.log("Modelo: "+carro.modelo);

carro.acelerar();
```

Objeto (2/2)

Voltando a questão do this, refere-se ao elemento ao qual esta usando no contexto. O this também existe dentro de um objeto.

O this não existe fora de um elemento, objeto, somente dentro. Conseguimos acessar o this de dentro de um objeto.

Através do de uma função consegue acessar os itens, propriedade ou até outras funções.

Exemplo:

```
let carro = {
        nome:'Fiat',
        modelo:'Uno',
        peso:'800kg',
        //na linha seguinte muda a propriedade do carro
        ligado:false,
        ligar:function(){
                this.ligado = true;
 /* quando a função ligar foi executada a propriedade ligar vai substituir false por  true */
                console.log("Ligando o "+this.modelo);
/*nesse exemplo foi usado o this, para acessar o próprio objeto, mostra o modelo do carro. */
                console.log("VRUM VRUM!");
        },


        acelerar:function(){
                // Então só vai mostrar o valor "Riiiiiihhhin", caso o carro esteja ligado
                if(this.ligado == true){
```

```
                console.log("Riiiiiihhhin");
            }
        }
};

console.log("Modelo: "+carro.modelo);
carro.ligar();
carro.acelerar();
```

Agora excluindo a linha de código que dá o processo de ligar o carro, veja o que acontece:

```
console.log("Modelo: "+carro.modelo);
carro.acelerar();
```

Neste caso, só vai mostrar o modelo, porque houve uma tentativa de acelerar o carro, nas o carro não está ligado.

Pode-se colocar uma linha de código que mostra mensagem para o usuário, que o carro não está ligado:

```
let carro = {
        nome:'Fiat',
        modelo:'Uno',
        peso:'800kg',
        //na linha seguinte muda a propriedade do carro
        ligado:false,
        ligar:function(){
            this.ligado = true; /* quando a função ligar foi executada a propriedade ligar vai subtituir false por  true */
            console.log("Ligando o "+this.modelo); /*nesse exemplo foi usado o this, para acessar o próprio objeto, mostra o modelo do carro. */
            console.log("VRUM VRUM!");
```

```
	},


	acelerar:function(){
		// Então só vai mostrar o valor "Riiiiiihhhin", caso o carro esteja ligado
		if(this.ligado == true){ /*neste caso, só consegue acelerar se o carro estiver ligado */
			console.log("Riiiiiihhhin");
		}else{
			console.log(this.nome+" "+this.modelo+" não está ligado");
		}
	}
};


console.log("Modelo: "+carro.modelo);
carro.ligar();
carro.acelerar();
```

Tudo isso até aqui foi feito com o objeto chamado carro. Dá uma visão que além de armazenar propriedades, mas também armazenar funções, fazer junções de funções, usar propriedades. Com isto dá para ter uma noção mais avançada de objeto.

Outro detalhe, pode-se colocar objeto dentro de um array.

Exemplo do array carros:

```
let carros = [
	{nome:'Fiat', modelo:'palio'},
	{nome:'Fiat', modelo:'Uno'},
	{nome:'Toyota', modelo:'Corolla'},
	{nome:'Ferrari', modelo:'Spide'}
];
```

Conforme o array criado, têm quatro itens.

console.log(carros);

Como acessar uma propriedade específica:

console.log(carros[2]);

Um modelo específico:

console.log(carros[2].modelo);

Ou então

console.log(carros[2]['modelo']);

O modo mais usado entre programadores é com o ponto. Exemplo: console.log(carros[2].modelo);

Uma observação:

Para identificar um **array** ou **objeto**, ***não esquecer***, é a forma que ele começa ou termina.

Um objeto começa e termina com chaves, e um array com colchetes, e não existe os " : " dois pontos. Em um array

simplesmente coloca os itens e ele vai ordenar, atribuir um número para cada item.

querySelector (querySelectorAll)

Um dos recursos novos do javascript que faz com que muitos recursos sejam feitos muito mais facilmente, sem muito trabalho. E com essas duas funções possibilita fazer muitas coisas de um modo fácil.

Arquivo index.html

```
<!DOCTYPE html>
<html>
<head>
        <title> Curso Javascript </title>
```

```
        <link rel="stylesheet" href="style.css">
</head>
<body>
        <h1> Seja bem vindo(a) </h1>
        <h3 class="subtitulo">Ingredientes</h3>
        <ul class="lista">
                <li>Farinha</li>
                <li>Chocolate</li>
                <li>Manteiga</li>
                <li>Ovos</li>
        </ul>

        <h3 class="subtitulo">Modo de preparo</h3>
        <script src="script.js"></script>
</body>
</html>
```

Arquivo script.js

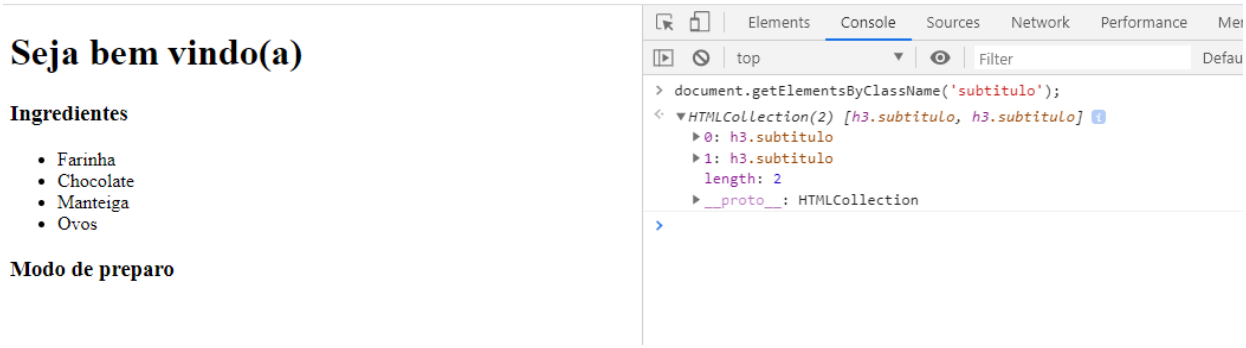
Para selecionar um item pelo id precisamos document.getElementById("..."), ou seja, esse elemento precisa ter um ID.

Existe uma outra função que é document.getElementsByClassName(''), onde colocamos entre parenteses a classe e essa função pega os elementos que tem essa classe específica.

Por exemplo vamos selecionar as tag h3 do index.html usando a classe desses elementos.

document.getElementsByClassName('subtitulo');

Como podem ver melhor via console essa seleção mostrada na imagem abaixo:



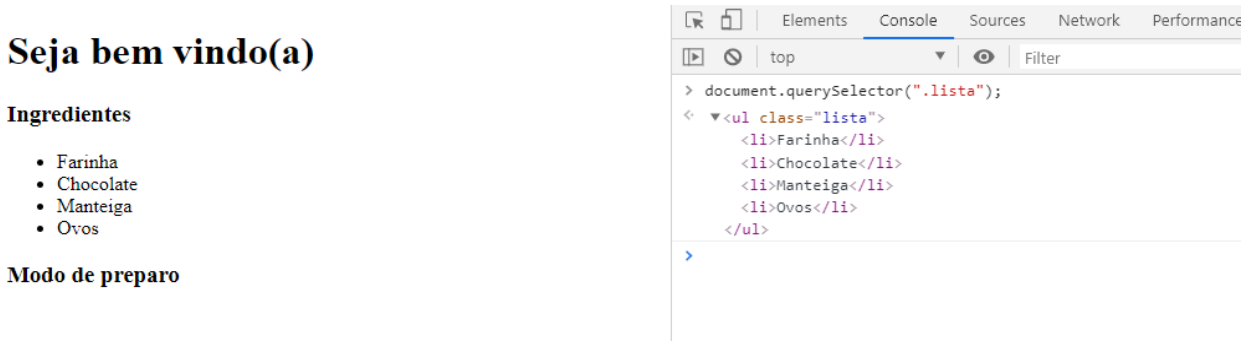
O resultado como podem ver é um array com os 2 itens que tem a classe subtitulo.

Mas temos outro recurso, ou melhor, dois recursos que são o **querySelector** e o **querySelectorAll**.

Vamos começar pelo querySelector(), que seleciona um elemento pela mesma forma que selecionamos no CSS. Por exemplo vamos selecionar a tag ul no arquivo index.html, basta colocar "." seguido do nome da classe desse elemento.

document.querySelector(".lista");

Observem que selecionou o elemento ou a tag ul com a classe lista:



Agora observem abaixo no arquivo index.html foi adicionado mais uma lista não ordenada tag ol, com a mesma classe lista:

Arquivo index.html

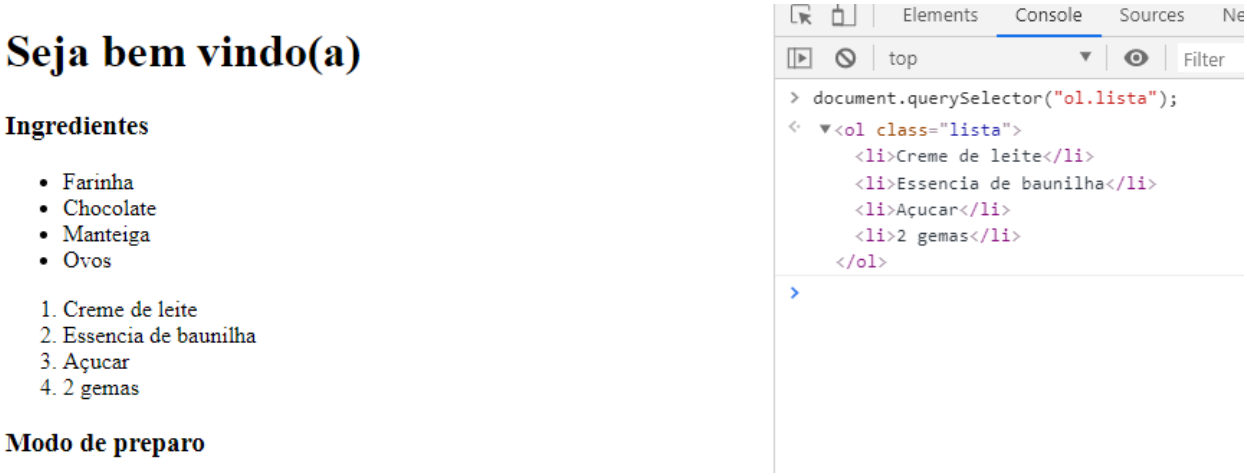
```
<!DOCTYPE html>
<html>
<head>
	<title> Curso Javascript </title>
	<link rel="stylesheet" href="style.css">
</head>
<body>
	<h1> Seja bem vindo(a) </h1>
	<h3 class="subtitulo">Ingredientes</h3>
	<ul class="lista">
		<li>Farinha</li>
		<li>Chocolate</li>
		<li>Manteiga</li>
		<li>Ovos</li>
	</ul>
	<ol class="lista">
		<li>Creme de leite</li>
		<li>Essencia de baunilha</li>
		<li>Açucar</li>
		<li>2 gemas</li>
	</ol>


	<h3 class="subtitulo">Modo de preparo</h3>
	<script src="script.js"></script>
</body>
</html>
```

E quero selecionar somente a tag ol, como faço já que temos duas tags como a mesma classe lista ?

Coloca-se o nome da tag, ponto e o nome da classe, como fariamos através do CSS.

document.querySelector("ol.lista");



Se vocês colocarem o cursor no array lista, podem ver que selecionou somente a tag da lista não ordenada(ol), como podem perceber na imagem abaixo:

Através dessa função document.querySelector(), também podemos selecionar pelo id, basta colocar assim:

document.querySelector("#idnome");

Ou seja, colocar o jogo da velha e o nome da id do elemento.