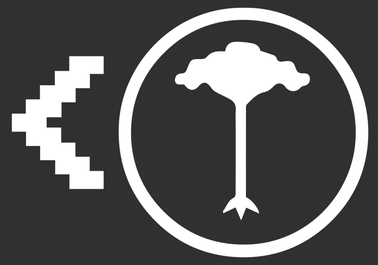
TRILHA FRONTEND

UX HTML CSS JS



Dia 4

- JavaScript
- Frameworks
- Boas práticas
- Testes unitários

Hands On
- Aplicar o javascript no site

Desafio
- Vai parar por aqui?

JavaScript - O que é?

JavaScript é uma linguagem criada pela Netscape em 1995 como uma extensão do HTML para o Netscape Navigator 2.0, tinha como principal função a manipulação de elementos HTML e a validação de formulários. Já foi chamado de Mocha e LiveScript até a Sun Microsystems batizá-lo de JavaScript.

JavaScript é uma linguagem interpretada. Isso quer dizer que ela precisa de um interpretador, que no caso do Javascript, é o navegador ou o NodeJS.

O NodeJS utiliza o V8 JavaScript Engine, o interpretador de JavaScript open source implementado pelo Google em C++ e utilizado pelo Chrome.

JavaScript - ECMAScript

ECMAScript é a linguagem de script que forma a base do JavaScript. ECMAScript é padronizada pela ECMA International.

Desde 2012, todos os navegadores modernos suportam o ECMAS-cript 5.1. Navegadores antigos suportam os ECMAScript 3.

Em 2015 a especificação para o ES6/ES2015 foi aprovada, mas ela ainda não é suportada integralmente pela maioria dos navegadores.

kangax.github.io/compat-table/es6/

JavaScript - Variáveis

A declaração de variável começa com a palavra-chave var/let/const, seguida por qualquer nome que você queira chamá-la, mas com algumas restrições:

- O primeiro caractere deve ser uma letra ASCII (maiúscula ou minúscula) ou um caractere de sublinhado (_).
- Números não podem ser usados como o primeiro caractere.
- Os caracteres subsequentes devem ser letras, números ou sublinhados (_).
- O nome da variável não deve ser uma palavra reservada.

Obs: let e var tem diferenças de escopo.

JavaScript - Fracamente tipada

Uma variável em JavaScript pode ter seu tipo de dados alterado durante a execução do programa.

Isso permite a realização de conversões de tipos (cast). Isso não significa que ela não possa fazer conversões ou que ainda a linguagem não forneça suporte a conversões.

JavaScript - Scoping

No JavaScript, existem escopos que delimitam a utilização das váriáves declaradas.

Blocos de repetição como laços (for, while) e condições (if, switch), não delimitam escopo, diferente de outras linguagens.
OBS: Varíaveis declaradas com o ‘let’ do ECMAScript 2015 respeitam o escopo em laços de repetição e condição.

O escopo é delimitado dentro de blocos de funções, ou seja, as variáveis declaradas dentro de uma função só estarão acessíveis dentro dela.

JavaScript - Hoisting

Em tempo de execução, todas as declarações de variáveis e funções são movidas para o ínicio de cada escopo em que essas declarações estão. Isso é o Hoisting (elevação).

Sabendo disso, é uma boa prática declarar todas as váriáveis juntas no começo do arquivo, evitando confusão ao se declarar uma variável depois de alguma função ou operação que altere seu valor.

JavaScript - Comentários

Comentários em códigos JavaScript são iguais ao CSS.

```
/*
Várias linhas
*/

// Linha única
```

JavaScript - Operadores

Um operador é um símbolo matemático que pode atuar em dois ou mais valores (ou variáveis) e produzir um resultado.

- Adição/concatenação: +
- Subtração: -
- Multiplicação: *
- Divisão e módulo: / , %
- Atribuição: =
- Identificação: ==, ===
- Negação: !, !==

JavaScript - Condicionais

São estruturas de código que te permitem testar se uma expressão retorna verdadeiro ou falso, e então executar diferentes linhas de código dependendo do resultado.

A forma mais comum de condicional é:

```
if (expressão) {

} else {

}
```

JavaScript - Laços e iterações

Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes.

- for
- do...while
- while
- for...in
- for...of
- forEach

JavaScript - Funções

Funções são uma maneira de encapsular funcionalidades que você tem que usar mais de uma vez.

Funções geralmente tem argumentos — pedaços de dados que elas precisam para fazerem o trabalho delas.

Por exemplo, a função alert() faz com que uma caixa pop-up apareça dentro da janela do browser, mas nós precisamos passar uma string como argumento para dizer qual mensagem deve ser escrita na caixa pop-up.

JavaScript - Funções (Declaration Vs Expression)

```
helloOpensanca();
goodByeOpensanca();

//Expression
function helloOpensanca(){
    console.log('Hello OpenSanca');
}

//Declaration
var goodByeOpensanca = function(){
    console.log('Good Bye OpenSanca');
}
```

JavaScript - Funções anônimas

São funções que são declaradas em tempo de execução.

São chamadas anônimas porque não possuem um nome da mesma forma que as funções normais possuem.

```
var opensanca = function() {
    return 'Hello Open Sanca!'
}
```

JavaScript - Funções de primeira classe

Funções em JavaScript são objetos de primeira classe.

Isso quer quer dizer que são um tipo especial de objeto que podem fazer tudo que um objeto normal faz.

Você pode atribuir função a uma variável, passar função como parâmetros de outra função, como atributos de objetos, ou seja, tudo o que você faz com qualquer outro tipo.

JavaScript - This

O domínio da keyword “this” em JavaScript é essencial na construção de objetos, herança e eventos.

o "this" é utilizado para não repetir o nome do sujeito da ação.

Fazendo uma comparação com a língua portuguesa, ele é semelhante ao uso dos pronomes ele, este, aquele que usamos quando não queremos massivamente repetir o sujeito da oração.

JavaScript - Call, Apply e Bind

Normalmente o “this” é automaticamente atribuído, mas você pode alterar seu valor por várias razões como reaproveitamento de código, por exemplo.

Existem 3 métodos de função capazes de alterar o valor “this”: Call, Apply e Bind.

JavaScript - Call

Este é o primeiro método de funções capaz de alterar o valor “this”. O primeiro parâmetro que recebe é o valor de “this” que que será atribuído à função. Os demais parâmetros são os parâmetros da função que invoca o método Call.

```
var nome = 'OpenSanca ';
function trilhaFrontendCall(dia1, dia2, dia3, dia4) {
   console.log(this.nome + dia1 + dia2 + dia3 + dia4);
}
trilhaFrontendCall('UX ', 'HTML ', 'CSS ', 'Javascript ');
trilhaFrontendCall.call({nome: 'Trilha Frontend '}, 'UX ', 'HTML ', 'CSS ',
'Javascript ');
```

JavaScript - Apply

Apply é o segundo método de funções capaz de alterar o valor “this”. Ele funciona exatamente como o método Call, porém seu segundo parâmetro recebe um Array ou Array-like dos parâmetros da função.

```
var nome = 'OpenSanca ';
function trilhaFrontendApply(dia1, dia2, dia3, dia4) {
   console.log(this.nome + dia1 + dia2 + dia3 + dia4);
}
trilhaFrontendApply('UX ', 'HTML ', 'CSS ', 'Javascript ');
trilhaFrontendApply.apply({nome: 'Trilha Frontend '}, 'UX ', 'HTML ', 'CSS ',
'Javascript ');
```

JavaScript - Bind

O terceiro método Bind se difere bastante do Call e Apply. Ele não executa a função, mas retorna uma outra.

O primeiro argumento recebe o valor do “this” a ser usado na função a ser retornada.

Os demais argumentos são os parâmetros que terão valores permanentemente atribuídos dentro da função a ser retornada.

JavaScript - Closures

Closure é dar continuidade à vida de variáveis locais de uma função.

Uma função interna enclausura uma variável local de uma função externa.

Assim, mesmo quando a função externa é invocada, a variável local interna permanece viva dentro da função interna, mantendo seu valor.

JavaScript - Eventos

São estruturas de código que percebem as coisas que acontecem no browser e permitem que você rode um código em resposta.

O exemplo mais óbvio é o evento de clique, que é disparado pelo browser quando o mouse clica em algo.

```
document.querySelector('html').onclick = function() {
    alert(‘Clicou!');
}
```

JavaScript - Strict Mode

O strict mode habilita semânticas diferentes do código normal:

- Elimina alguns erros silenciosos fazendo-os lançar exceções.
- Evita equívocos que dificultam que motores JavaScript realizem otimizações: código strict mode pode às vezes ser feito para executar mais rápido que código idêntico não-strict mode.
- Proíbe algumas sintaxes que provavelmente serão definidas em versões futuras do ECMAScript.

JavaScript - IIFE

Immediately Invoked Function Expression
É uma função em Javascript executada logo que definida, também conhecida como função imediata.

São úteis para realizar encapsulamento. As variáveis em Javascript têm como escopo a função pela qual elas foram definidas.

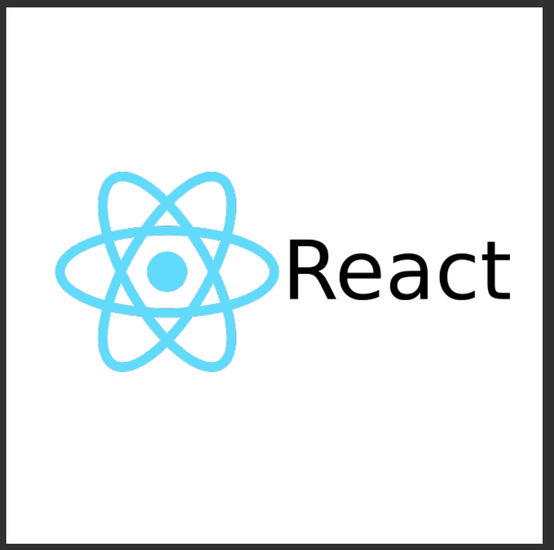
Ao criar uma função anônima com execução imediata, podemos criar um escopo temporário para nossas funções e variáveis. Com isso, evitamos poluição no nosso escopo global e possíveis conflitos de variáveis ou funções com o mesmo nome.

JavaScript - Orientação a Objetos

JavaScript tem fortes capacidades de programação orientada a objetos, apesar de ocorrerem algumas discussões devido às diferenças da orientação a objetos no JavaScript em comparação com outras linguagens.

No JavaScript, a programação é baseada em protótipos, um estilo de programação orientada a objetos na qual não temos presença de classes. Em vez disso, a reutilização de comportamento (herança nas linguagens baseadas em classes) é realizada através de um processo de decorar (ou expandir) objetos existentes que servem como protótipos.

JavaScript - Frameworks



JavaScript - Boas Práticas

Mantenha a consistência de estilo de código
Use um padrão de indentação em todo o projeto. Em projetos com mais de uma pessoa desenvolvendo no mesmo código, use o ‘.editorconfig’.

Mantenha o código sempre no mesmo idioma
Adote um idioma no início do projeto e siga com ele até o fim na criação de variáveis, funções, comentário, etc.

Utilize o modo restrito: 'use strict'.
Isso ajudará q evitar erros comuns de sintaxe pois exceções irão mostrar problemas no código.

JavaScript - Boas Práticas

Faça um código fácil de ser compreendido.
Use nomes de variáveis e funções auto explicativos e simples. Crie algum padrão e mantenha em todo o projeto.

Evite o uso de variáveis globais.

Escreva comentários quando necessário.
É comum ouvir "Um bom código não precisa de explicação", mas na prática em projetos maiores, procure explicar a finalidade do seu código. Muitas pessoas, de diferentes níveis, podem ter que trabalhar no seu código e nem sempre elas tem experiência, tempo ou conhecimento do negócio para entender tudo.

JavaScript - Boas Práticas

Evite misturar tecnologias.
Estilize seu html com CSS e não com javascript. Se for usar frameworks, evite usar mais de um que tenham as mesmas funcionalides (Ex: Jquery e YUI).

Use sintaxes abreviadas. Exemplo:

```
// Use
var margem = altura || 10;
```

```
// Em vez de
if(altura === undefined){
   var margem = altura;
} else {
   var margem = 10;
}
```

JavaScript - Boas Práticas

Modularize seu código
Evite escrever funções, trechos de código muito longos, ou aninhados. Procure separar regras e evite códigos repetidos.

Utilize os eventos de forma correta.
Ao escrever seu código e registrar comportamentos e eventos na interação do usuário é importante ter cuidado na escolha correta dos eventos. Um caso comum de erro é vincular uma função no clique do botão enviar de um formulário, em vez de chamar a ação no submit do formulário.

JavaScript - Boas Práticas

Evite muitos aninhamentos
Facilite o entendimento e manutençao dos seus códigos. Utilize as práticas anteriores e evite um código do tipo:

Otimize loops
Existem várias maneiras de fazer loops, uma melhores que outras. Uma técnica simples para melhorar ainda mais os loops for() nativos é fazer cache de variáveis de comparação, evitando executar o .length a cada iteração.

JavaScript - Boas Práticas

Minimize acessos ao DOM.
Evite ao máximo acessar o DOM para buscar ou inserir informações. Coisas simples podem minimizar o acesso, por exemplo, se alguma função precisa atualizar uma tabela no seu HTML, em vez de percorrer <td> por <td> alterando os valores, pode ser mais eficiente criar uma função que reescreva toda a tabela, e depois simplesmente troque a antiga pela nova.

Não reinvente a roda.
Use bibliotecas e frameworks estáveis, que possuem estrutura conhecida, mas antes de inserir libraries e plugins, verifique a real necessidade, o suporte e compatibilidade.

JavaScript - Boas Práticas

Código de desenvolvimento não é código de produção.
Minifique, remova comentários, concatene os javascripts antes de mandar para produção. Existem diversas ferramentas para isso, inclusive online. Tenha uma versão de desenvolvimento, normal, comentada e uma para o site em produção.

Conheça e utilize algum design pattern
https://addyosmani.com/resources/essentialjsdesignpatterns/book/

Escreva testes unitários

JavaScript - Testes unitários

A criação de programas complexos é apenas uma questão de dividi-los em unidades menores e depois juntá-las.

O teste unitário é basicamente o teste dessas unidades menores, aquelas que iremos juntar.

Basicamente os testes unitários ajudam você a pensar com as expectativas de seu código de uma forma organizada, minimizam o risco e o esforço ao mudar o código e incentivam o design modular.

JavaScript - Testes unitários

```
//Arquivo js
function multiplicaDoisNumeros(valor1, valor2){
    return valor1 * valor2;
}

//Arquivo js de teste
describe('Função que multiplica dois números', function () {

    var mult = multiplicaDoisNumeros;

    it('verifica se estamos multiplicando corretamente', function () {

        expect(mult).toBeDefined();
        expect(mult( 3, 7 )).toBe( 21 );

    });
});
```

HANDS ON!

Referências

http://jstherightway.org/
http://tableless.com.br/o-que-nodejs-primeiros-passos-com-node-js/
https://kangax.github.io/compat-table/es6/
http://opensource.locaweb.com.br/locawebstyle-v2/manual/praticas-padroes/javascript/
http://programadorobjetivo.co/call-apply-e-bind-em-javascript/
http://programadorobjetivo.co/closures/
http://opensource.locaweb.com.br/locawebstyle-v2/manual/praticas-padroes/javascript/
https://ericdouglas.github.io/2014/02/25/06-testando-javascript-com-jasmine-e-karma/

Obrigado!

Até a próxima!