- Material disponível em: https://github.com/DevilAnseSenior/Introdu-JavaScript

JavaScript

- **Linguagem de programação interpretada e estruturada;**
- **Script de alto nível e tipagem dinâmica fraca;**
- **Multi-paradigma(Protótipos, orientada a objetos, imperativo e funcional);**
- **Uma das três técnologias WWW;**
- **Criação de páginas Web Interativas;**

Primeiros passos com JavaScript

- **É uma linguagem usada para adicionar interatividade ao seu site(Ex.: Jogos, respostas quando botões são pressionados ou dados inseridos em formulários, estilo dinâmico, animações).**

Um exemplo "Olá mundo"

- **Para começar, veja como adicionar algum JS básico a sua página, criando um "olá mundo!".**

  1. **Primeiro crie um arquivo chamado index.html e escreva o seguinte trecho de código:**

```
<html>
    <head>
        <!--Configurações iniciais-->
        <title>Exemplo 1</title>
    </head>
    <body>
        <!--Estrutura do site-->

        <script src="scripts/main.js"></script>
    </body>
</html>
```

  2. **Nesta pasta crie uma pasta chamada scripts;**
  3. **Dentro da pasta crie um arquivo chamado main.js e escreva a instrução a seguir:**

```
console.log('Olá mundo!!!');
```

  4. **Para executar o código, abra o navegador use o comando CTRL + o e abra o arquivo index.js, na página principal abra as opções de desenvolvedor e verique a saída no console;**

- **O que aconteceu?**

- Basicamente fizemos com que uma mensagem fosse impressa no console.

Curso intensivo de fundamentos da linguagem: Sintaxe e tipos

- Vamos explicar alguns dos principais recursos do JS, para dar um melhor entendimento de como tudo funciona.

Declarações

- Existem três tipos de declarações em JavaScript.
  - var
    - Declara uma variável, opcionalmente, inicializando-a com um valor;
  - let
    - Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor;
  - const
    - Declara uma constante de escopo de bloco, apenas de leitura.

Variáveis

- São espaços na memória do computador onde são armazenados dados.
- Tem nomes simbólicos para valores na aplicação. Esse nomes obedecem determinadas regras:
  1. Deve começar com uma letra, underline(_) ou cifrão($);
  2. Os caracteres subsequentes podem ser numeros (0 - 9);
  3. Letras maiúsculas são diferidas das minúsculas.
- Variáveis são iniciadas com as palavras-chaves var, let ou const seguida por qualquer nome que você queira chamá-la:

```
var nome;
const PI;
var idade;
let minhaVariavel;
```

Nota: Ponto-e-virgula no final da linha indica onde uma instrução termina, colocá-las é uma boa prática;

Nota: JS é case sensitive, ou seja, 'minhaVariavel' é diferente de 'minhavariavel'.

- **Depois de declarar uma várivel, você pode dar um valor a ela:**

```
var nome;
nome = 'Bob';
```

- **Se quiser é possível fazer as duas operações na mesma linha:**

```
var genero = 'Masculino';
```

- **Você pode retornar o valor chamando a váriavel pelo nome:**

```
nome;
```

- **Depois de dar um valor a variável, tambem é possível mudá-lo:**

```
var nome = 'Bob';
nome = 'Steve';
```

- **Para uma entrada dinâmica de dados (através do teclado), usamos o comando prompt:**

```
var fruta = prompt("Informe a fruta desejada: ");
```

- **Nesse caso a variável fruta, receberá o conteudo digitado pelo usuário.**

Note que as variáveis tem diferentes tipos de dados:

Variáveis	Explicação	Exemplo
String	Sequência de texto é conhecida como string. Para mostrar que o valor é uma String, deve ser envolvida entre aspas.	var nome = 'Bob';
Number	Um número. Números não tem aspas.	var num = 10;
Boolean	Um valor verdadeiro ou falso. As palavras 'true' e 'false' são reservadas em JS e não precisam de aspas.	var bool = true;
	Uma estrutura que permite armazenar	var pessoa = [1, 'Bob', 10]

Variáveis	Explicação	Exemplo
Array	Uma estrutura que permite armazenar vários valores em uma única variável.	Cada item pode ser acessado: pessoa[0], pessoa[1], etc.
Object	Basicamente, qualquer coisa. Em JS tudo é objeto e pode ser armazenado em uma váriavel. Tenha isso em mente enquanto aprende.	var titulo = document.querySelector('h1');

- **Variáveis são necessárias para qualquer coisa interessante na programação. Se os valores não podessem mudar, nada dinâmico seria criado.**
- **JavaScript é uma linguagem fortemente tipada. Isso significa que você não precisa especificar tipo de dado de uma variável quando declará-la;**
- **São convertidos automáticamente conforme a necessidade, por exemplo:**

```
var answer = 42;
```

- **E depois, posso atribuir uma string para a mesma variável, por exemplo:**

```
answer = "Obrigado pelos peixes...";
```

Escopo de váriavel

- **Quando uma variavel é declarada fora de qualquer função, ela é chamada de variável global;**
- **Quando declarada dentro de uma função, é considerada variável local.**

Comentários

- **É possível colocar comentários em códigos JS:**

```
/*
    Tudo no meio é um comentário.
*/
```

- **Se o seu comentário tiver apenas uma linha, coloque dessa maneira:**

```
// Isto é um comentário
```

Operadores

- **Simbolo matemática que produz um resultado baseado em dois valores;**
- **Vejamos alguns deles:**

Operador	Explicação	Simbolo(s)	Exemplo
adição/concatenação	Usado para somar dois numeros ou juntar duas strings	+	6 + 9; "Olá " + "Mundo!"
subtrair, multiplicar, dividir	Fazem exatamente o que você espera que eles façam na matemática básica.	-, *, /	9 - 3; 8 * 2; 9 / 3;
operador de atribuição	Já vimos, ela associa um valor a uma váriavel.	=	var nome = 'Bob';
operador de igualdade	Faz um teste para ver se dois valores são iguais, retornando um resultado true/false (booleano)	===	var numero = 3; numero === 4;

negação, não igual(diferente)	Retorna o valor lógico oposto do sinal; transforma um true em false, etc. Quando usado junto com o operador de igualdade, o operador de negação testa se os valores são diferentes.	!, !==	"Não igual" dá basicamente o mesmo resultado da sintaxe diferente. Aqui estamos testando "É numero NÃO é igual a 3". Isso retorna false porque numero É igual a 3.
			var numero = 3;
			numero !== 3;

- **Há vários outros operadores para explorar, mas por enquanto esse são suficientes.**
- **Em expressões envolvendo valores numérico e string com o operador +, JavaScript converte valores numéricos para string. Ex.:**

```
x = "A resposta é " + 42;
y = 42 + " é a resposta.";
```

- **Nas declarações envolvendo outros operdores, JS não converte em valores numéricos para string. Ex.:**

```
"37" - 7;
"37" + 7;
```

Convertendo strings para números

- **No caso de um valor que representa um número está armazenado na memória como uma string, existem métodos para conversão.**
  - **parseInt()**
    - **Irá retornar apenas números inteiros, uso restrito para casas decimais.**
  - **parseFloat()**
    - **Irá retornar um número real, restrito a números com parte decimal.**
  - **Um método alternativo de conversão de um número em forma de string é com o operador + (operador soma):**

```
"1.1" + "1.1" = "1.11.1"
(+"1.1") + (+"1.1") = 2.2
// Nota: Parênteses usados para deixar legível, ele não é requerido.
```

Controle de Fluxo e Manipulação de Erro

Declaração em bloco

- **Utilizada para agrupar declarações, este bloco é delimitado por chaves:**

```
{
    declaracao_1;
    declaracao_2;
    .
    .
    .
    declaracao_n;
}
```

Exemplo

```
var x = 1;
{
    var x = 2;
}
console.log(x);
```

- **Este código exibe 2 pois a declação de var x antes do bloco.**

Condicionais

- **Permite testar se uma expressão retorna verdadeiro ou não;**
- **Executa um código alternativo, dependendo da situação;**
- **Uma forma comum de condicional é a instrução if... else. Declarada da seguinte maneira:**

```
if (condicao) {
    declaracao_1;
} else {
    declaracao_2;
}
```

- **Onde condicao pode ser qualquer expressão que seja avaliada como verdadeira ou falso. Ex.:**

```
var sorvete = 'chocolate';
if (sorvete === 'chocolate') {
    alert('Gosto desse sabor');
} else {
    alert('Bom, mas o meu sabor favorito é chocolate.');
}
```

- **A expressão dentro do if (...) é o teste - ela usa o operador de igualdade(como descrito anteriormente) para comparar a variável sorvete com a string chocolate para ver se elas são iguais;**
- **Se a comparação retorna true, o primeiro bloco é executado;**
- **Se for falsa, o primeiro bloco é ignorado e o segundo executado.**
- **Tambem é possível combinar declarações utilizando else if parar obter várias condições testadas em sequência. Ex.:**

```
if (condicao) {
    declaracao_1;
} else if (condicao_2) {
    declaracao_2;
} else if (condicao_n) {
    declaracao_n;
} else {
    declaracao_final;
}
```

- **Aqui conseguimos vizualizar claramente a declaração de blocos. Em geral, é uma boa prática prática, especialmente ao aninhar if's:**

```
if(condicao) {
    declaracao_1_executada_se_condicao_for_verdadeira;
    declaracao_2_executada_se_condicao_for_verdadeira;
} else {
    declaracao_3_executada_se_condicao_for_falsa;
    declaracao_4_executada_se_condicao_for_falsa;
}
```

  - **Recomenda-se não utilizar atribuições simples em uma expressão condicional porque o símbolo de atribuição poderia ser confundido com o de igualdade. Por exemplo, não utilize o seguinte código:**

```
if (x = y) {
    /* faça a coisa certa disgraçadão */
}
```

  - **Caso tenha que utilizar uma atribuição em uma expressão condicional, uma prática comum é colocar parênteses adicionais em volta da atribuição. Por exemplo:**

```
if((x = y)) {
    /*Faça a coisa certa*/
}
```

Valores avaliados como falsos

- **Os seguintes valores são avaliados como falsos:**
  - **false;**
  - **undefined;**
  - **null;**

- 0;
- NaN;
- string vazia("");

- Todos os outros valores, incluindo todos os objetos, são avaliados como verdadeiros quando passados para uma declaração condicional.

Declaração switch

- Permite que um programa avalie uma expressão e tente associar a um caso(case).
- Se o correspondente é encontrado, a operação associada é executada.
- Vejamos uma declaração switch:

```
switch (expressao) {
    case rotulo_1:
        declaracoes_1
        [break;]
    case rotulo_2:
        declaracoes_2
        [break;]
    ...
    default:
        declaracoes_padrao
        [break;]
}
```

- O programa primeiro procura um case com o rótulo correspondente;
- Se nenhum correspondente é encontrado, o programa procura pela clausula opcional default;
- A instrução break associada a cada clausula, garante que o programa sairá do switch assim que a correspondente for executada;

Exemplo

- No exemplo a seguir, se tipofruta for avaliada como "Banana", o programa faz a correspodência do valor com case "Banana" e executa a declaração associada. Quando o break é encontrado o programa termina.

```
switch (tipofruta) {
    case "Laranja":
```

```
        console.log("O quilo da laranja está R$0,59.<br>");
        break;
    case "Maçã":
        console.log("O quilo da maçã está R$0,32.<br>");
        break;
    case "Banana":
        console.log("O quilo da banana está R$0,48.<br>");
        break;
    case "Cereja":
        console.log("O quilo da cereja está R$3,00.<br>");
        break;
    case "Manga":
        console.log("O quilo da manga está R$0,56.<br>");
        break;
    case "Mamão":
        console.log("O quilo do mamão está R$2,23.<br>");
        break;
    default:
        console.log("Desculpe, não temos " + tipofruta + ".<br>");
}
console.log("Gostaria de mais alguma coisa?<br>");
```

Declaração e manipução de Error(Erros)

- **Você pode chamar uma exceção usando a declaração throw e manipulá-la usando try...catch.**

Tipos de exceções

- **Praticamente pode-se utilizar throw em qualquer objeto JS. Todavia, nem todos os objetos ativados throw são igualmente criados.**

Declaração throw

- **Use a declaração throw para lançar uma exceção. Quando lançada, devemos especificar a expressão que conterá o valor a ser lançado:**

```
throw expressão;
```

- **É possivel lançar qualquer tipo de expressão. O código a seguir lança varias exceções de tipos diferentes:**

```
throw "Error2"; //tipo string
throw 42; //tipo numérico
throw true; //tipo booleano
throw {toString: function() { return "Eu dou um objeto"; } };
```

Nota: É possivel especificar um objeto quando se lança uma expressão. É possível especificar essas propriedades de um objeto no bloco catch. O exemplo a seguir cria um objeto myUserException do tipo userException e o usa em uma declaração throw.

```
// Criando um objeto do tipo UserException
function UserException(mensagem) {
    this.mensagem = mensagem;
    this nome = "UserException";
}

// Realiza a conversão da exceção para uma string adequada quando
usada como uma string.
// (ex. pelo console erro)
UserException prototype.toString = function() {
    return this.name + ': "' + this.message + '"';
}

//Cria uma instância de um tipo de objeto e lança ela
throw new UserException("Valor muito alto");
```

Declaração try...catch

- **A declaração try...catch com um bloco de declarações em try, e especifica uma ou mais respostas para uma exceção lançada. Se uma exceção é lançada, a declaração try...catch pegá-a.**
- **O exemplo a seguir usa a declaração try...catch. O exemplo chama uma função que recupera o nome de um mês no array com base no valor passado para a função. Se o valor não corresponde ao numero de um mês (1-12), uma exceção é lançada com o valor "InvalidMonthNo" e as declarações no bloco catch define a váriavel monthName para unknown.**

```
function getMonthName(mo) {
    mo = mo - 1; // Ajusta o número do mês para o índice do array (1
= Jan, 12 = Dez)
    var months = ["Jan", "Fev", "Mar", "Abr", "May", "Jun", "Jul",
"Aug", "Sep", "Oct", "Nov", "Dez"];
    if (months[mo]) {
        return months[mo];
    } else {
        throw "InvalidMonthNo"; //lança uma palavra-chave aqui usada.
    }
}

try { // statements to try
    monthName = getMonthName(myMonth); // função poderia lançar
exceção
}
catch (e) {
    monthName = "unknow";
    logMyErrors(e); // passa a exceção para o manipulador de erro ->
sua função local.
}
```

O bloco catch

- **Você pode usar o bloco catch para lidar com todas as exceções que podem ser geradas no bloco try.**

```
catch (catchID) {
    declaracoes
}
```

- **O bloco catch é especificado por um identificador, que contem a especificação dada pelo throw; Esse identificador contem informações da exceção lançada. O identificador funciona enquando o bloco catch está está em execução.**
- **Por exemplo, o seguinte código lança uma exceção. Quando a exceção ocorre, o controle é transferido para o bloco catch.**

```
try {
    throw "MyException"; //Lança uma exceção
}
catch(e) {
    // declarações de lidar com as exceções
    logMyErrors(e); // passar a exceção para o manipulador de erro
}
```

O bloco finally

- **O finally contém instruções para executar após os blocos try e catch, mas antes das declarações seguinte a declaração try...catch. O bloco finally é executado com ou sem o lançamento de excessão. Se uma exceção é lançada, a declaração do bloco finally excuta, mesmo que nenhum catch seja processado.**

```
openMyFile();
try {
    writeMyFile(theData); // Isso pode lançar um erro
} catch(e) {
    handleError(e); // Se temos um erro temos que lidar com ele
} finally {
    closeMyFile(); //Sempre feche o recurso
}
```

- **Se o bloco finally retorna um valor, este valor se torna toda a entrada try-catch-finally, independente de quaisquer declarações:**

```
function f() {
    try {
```

```
        console.log(0);
        throw "bogus";
    } catch(e) {
        console.log(1);
        return true; // Essa declaração de retorno é suspensa
                     // até que o bloco finally seja concluído
        console.log(2); // não executa
    } finally {
        console.log(3);
        return false; // substitui o return anterior
        console.log(4); // não executa
    }
    // "return false" é executado agora
    console.log(5); // não executa
}
f(); // exibe 0, 1, 3; retorna false
```

- **Substituições de valores de retorno pelom bloco finally támbem se aplica a exceções lançadas ou re-lançadas dentro do bloco catch:**

```
function f() {
  try {
    throw "bogus";
  } catch(e) {
    console.log('captura interior "falso"');
    throw e; // essa instrução throw é suspensa até
             // que o bloco finally seja concluído
  } finally {
    return false; // substitui "throw" anterior
  }
  // "return false" é executado agora
}

try {
  f();
} catch(e) {
  // isto nunca é executado porque o throw dentro
  // do catch é substituído
  // pelo return no finally
  console.log('captura exterior "falso"');
}

// SAIDA
// captura interior "falso"
```

- **Tambem é possível aninhar declarações try...catch.**

Loops - Laços e iterações

- **Permite que uma parte do código continue executando repetidademente, até que determinada condição seja satisfeita.**

- **Existem várias formas de repetição em JS, mas eles na sua essência fazem a mesma coisa: repetem uma ação multiplas vezes(possível até repetir 0 vezes). Cada uma oferecem diferentes formas de iniciar e encerrar. Há situações onde é possível um problema utilizando um determinado tipo de laço do que outros.**
  - **for**
    - **Um laço for é repetido até que a condição especifica seja falsa. A declação for é feita da seguinte maneira:**

```
for([expressaoInicial]; [condicao]; [incremento]) {
    declaracoes.
}
```

    - **Vejamos um exemplo prático onde temos um jogo onde você manda o seu personagem andar X passos em uma direção e Y em outra; por exemplo, a ideia "vá 5 passos para o leste" pode ser expressa em um laço dessa forma:**

```
var passo;
for (passo = 0, passo < 5; passo++) {
    //Executa 5 vezes, com os valores de passsos de 0 a 4.
    console.log('Ande um passo para o leste.');
}
```

    - **Um loop for utiliza a inserção de três valores (argumentos):**

      1. **Um valor inicial**: Nesse caso estamos iniciando a contagem com 0, mas pode ser qualquer numero da minha escolha;
      2. **Uma condição de saída**: Aqui nos especificamos passo < 5 - O loop irá continuar até que passo não seja mais menor que 5.
      3. **Incremento:** Especificado como passo++. Significa "Adicione 1 ao passo".
  - **do... while**
    - **Repetirá até que a condição especifica seja falsa.**

```
do {
    declaracao_1;
    declaracao_2;
    declaracao_3;
} while(condicao);
```

- **Será executada uma vez antes da condição ser verificada. Caso a condição seja verdadeira, então o laço será executado novamente. Ao final de cada execução, a condicao é verificada. Quando for falsa a execução é encerrada. Exemplo:**
  - **A seguir o laço é executado pelo menos uma vez e irá executar até que i seja menor que 5.**

```
do {
    i += 1;
    console.log(i);
} while(i < 5);
```

while

- **Executa as intruções, desde que a condição especifica seja avaliada como verdadeira.**

```
while (condicao) {
    declaracao1;
    declaracao2;
    declaracao3;
}
```

- **Se a condição se torna falsa ele segue em frente.**
- **O teste ocorre antes do laço, assim o laço executará e testará a condição novamente. Ex:**
  - **O while a seguir executará enquando n for menor que três:**

```
n = 0;
x = 0;
while (n < 3) {
    n++;
    x += n;
}
```

  - **A cada iteração, o laço incrementa n e adiciona este valor para x. Portanto, x e n recebem os seguintes valores:**
    - Depois de executar pela primeira vez n = 1 e x = 1;
    - Depois da segunda vez: n = 2 e x = 3;
    - Depois da terceira vez: n = 3 e x = 6.

Instrução break

- **Usado para terminar laços e switchs;**
- **Exemplo:**
  - **O exemplo a seguir percorre os elementos de um Array até que ele encontre o indice do elemento que o possui um valor contido na variável valor:**

```
for(i = 0; i < a.length; i++) {
    if(a[i] == valor) {
        break;
    }
}
```

Instrução continue

- **Pode ser usada para reiniciar uma instrução while, do-while ou for e continuará a execução apartir da próxima iteração.**
- **Ao contrário da instrução break, continur não encerra a execução do laço. Em while, ele voltara pra a condição. Em for, ele pulará para a expressão de incrementação.**
- **Exemplo:**
  - O exemplo a seguir mostra um laço while utilizando continue que executará quando o valor de i for igual a 3. Desta forma, n recebe os valores um, três e doze.

```
var i = 0;
var n = 0;
while(i < 5) {
    i++;
    if (i==3) {
       continue;
    }
    n += i;
}
```

Instrução for...in

- **Executa iterações apartir de uma variável especifica;**
- **Percorre todas as propriedades de um objeto;**
- **Para cada propriedade distinta, o JS executará uma iteração.**

```
for(variavel in objeto) {
    declaracoes;
}
```

Instrução for...of

- **Cria um laço com objetos interativos((incluindo, Array, Map, Set, assim por conseguinte)), executando uma iteração para o valor de cada propriedade distinta.**

```
for(variavel of objeto) {
    declaracoes;
}
```

Funções

- **Maneira de encapsular funcionalidades reutilizaveis;**
- **Apresentada a necessidade, posso chamar a função pelo nome, ao invés de, reescrever o código inteiro;**
- **Já vimos um exemplo:**

```
alert('hello');
```

- **Essa função é pré-definida nos navegadores para ser usada quando quiser;**
- **Se ver alguma coisa com um nome de variável, mas com parênteses - () - depois, provavelmente é uma função;**
- **Geralmente tem argumentos;**
- **Esses colocados dentro dos parênteses e separados por vírgula.**
- **Tambem é possivel definirmos nossas próprias funções, façamos uma que multiplica dois numeros:**

```
function multiplicacao(num1, num2) {
    var resultado = num1 * num2;
    return resultado;
}
```

- **A definição de uma função(também chamada de declaração) consite no uso da palavra chave function, seguida por:**
    - **Nome da função;**
    - **Lista de argumentos da função, entre parênteses e separado por vírgulas;**
    - **Declarações JS que definem a função entre chaves { }.**

- **Por exemplo, a função *multiplicacao* recebe dois argumentos (num1, num2). A função consiste em executar um calculo através da variável *resultado*. A instrução *return* especifica o valor retornado por essa função.**
- **A definição de uma função não a executa. Definir a função é simplismente nomear a função e especificar o que fazer quando a função é chamada. Chamar a função executa realmente as ações especificadas com os parâmetros indicados. Por exemplo, tente executar essa função no console e teste com vários argumentos. Ex.:**

```
mutiplicacao(4,7);
mutiplicacao(20,20);
mutiplicacao(0.5,3);
```

Nota: A instrução return diz ao navegador que retorne a variável resultado da função. Isso é necessário pois, variáveis definidas dentro de funções só estão disponíveis dentro de funções.

- **Funções devem estar no escopo quando são chamadas, mas a declaração de uma função pode ser puxada para o topo (aparecem abaixo da chamada no código).**
- **O escopo de uma função é a função na qual ela é declarada, ou todo o programa se ela é declarada no nível superior.**

Nota: Isso funciona apenas quando usamos a definição padrão de função.

- **Para parâmetros primitivos (como um número) são passados para funções por valor; o valor é passado para a função, mas se a função altera o valor do parâmetro, esta mudança não reflete globalmente ou na função chamada.**
- **Váriaveis definidas no interior de uma função não podem ser acessadas de nenhum lugar fora dela.**
- **Mas, uma função pode acessar todas as váriaveis e funções criadas fora de escopo, ou seja, funções no ambito global podem acessar todas as váriaveis, enquanto uma função criada dentro de outra pode acessar os dados da sua função hospedeira.**

```
// Seguintes váriaveis definidas no escopo global
var num1 = 20,
    num2 = 3,
    nome = "Chamahk";

// Função definida no escopo global
function multiplica() {
    return num1 * num2;
}

multiplica(); // Retorna 60

// Um exemplo de função aninhada
function getScore() {
    var num1 = 2,
        num2 = 3;

    function add() {
        return nome + " scored " + (num1 + num2);
    }

    return add();
}

getScore(); // Retorna "Chamhk scored 5"
```

Expressão de função

- **Funções tambem podem ser criadas por uma expressão de função. Tal função pode ser anônima; não precisa ter um nome.**

```
var square = function(numero) { return numero * numero };
var x = square(4) //x recebe o valor 16
```

- **No entanto um nome pode ser fornecido com uma expressão de função e póde ser utilizado no interior da função para se referir a si mesma, ou em um debugger para identificar a função stack traces:**

```
var fatorial = function fac(n) { return n < 2 ? 1 : n * fac*(n-1) };

console.log(fatorial(3));
```

- **As expressões de funções são convenientes quando precisamos passala como argumento para outra função. O exemplo a seguir mostra uma função map sendo definida e, em seguida, chamada com uma função anônima como seu primeiro parâmetro:**

```
function map(f,a) {
    var result = [];
    var i;
    for(i = 0; i != a.length; i++)
        result[i] = f(a[i]);
    return result;
}
```

- **O código a seguir:**

```
map(function(x) {return x * x * x}, [0, 1, 2, 5, 10]);
```

- **Deve retornar [0, 1, 8, 125, 1000].**
- **Em JS, uma função pode ser definida com base numa condição. Por exemplo, a seguinte definição de função define minhaFuncao somente se num é igual a 0:**

```
var minhaFuncao;
if (num == 0) {
    minhaFuncao = function(objeto) {
        objeto.make = "Toyota";
    }
}
```

- **Um *método* é uma função invocada por um objeto.**

Array

- **Conjunto de valores ordenados que referenciado com um nome e um índice.**
- **Por exemplo, podemos ter um array chamado *emp* contendo nomes de funcionários com seus numeros indexados. Sendo assim, *emp[1]* poderia ser o funcionário número 1, *emp[2]* o funcionário numero 2 e assim por diante.**

- **JS não possui um tipo array. No entanto temos um objeto predefinido *Array* com métodos específicos para trabalhar com eles. Possui uma propriedade para determinar o tamanho dele e outras propriedades para utilização com funções regulares.**

Criando um array

- **As declarações a seguir criam arrays equivalentes:**

```
var arr = new Array(elemento0, elemento1, ..., elementoN);
var arr = Array(elemento0, elemento1, ..., elementoN);
var arr = [elemento0, elemento1, ..., elementoN];
```

- ***elemento0, elemento1, ..., elementoN* é uma lista de valores para os elementos do array. Quando esses valores são especificados, o array é inicializado com eles como elementos deste array. A propriedade do comprimento do array é definida pelo número de argumentos.**
- **A sintaxe utilizando colchete é chamado de "array literal" ou "inicializador de array". É uma abreviação de outras formas de criação array e é a forma preferida de criação.**
- **Para criar um array com tamanho diferente de zero, mas sem nenhum item, qualquer dos esquemas abaixo pode ser utilizado:**

```
var arr = new Array(comprimentoDoArray);
var arr = Array(comprimentoDoArray);

// Estes produzem exatamente o mesmo efeito
var arr = [];
arr.lenght = comprimentoDoArray;
```

Nota: No código acima, *comprimentoDoArray* deve ser um *número*. De outra maneira, um array com um unico elemento(o valor passado) será criado. Chamar arr.lenght retornará *comprimentoDoArray*, mas o array na verdade, contém elementos vazios(undefined). Executar um loop *for...in* no array, não retorna nenhum dos elementos do array.

- **Se você deseja inicializar um array com um único elemento, e este elemento é um *número*, você precisa usar a sintáxe dos colchetes. Quando um único valor de *número* é passado para o construtor Array(), ou para uma função, ele é interpretado como um *comprimentoDoArray*, e não como um elemento único.**

```
var arr = [42];     // Cria um array com apenas um elemento:
                    // o número 42.
var arr = Array(42) // Cria um array sem elementos
                    // e arr.lenght é definido como 42; isso é
                    // equivalente a:
var arr = [];
arr.length = 42;
```

Povoando um array

- **Você pode povoar um array (inserir elementos) a um array atribuindo valores aos seus elementos. Por exemplo,**

```
var emp = [];
emp[0] = 'Casey Jones';
emp[1] = 'Phil Lesh';
emp[2] = 'August West';
```

- **Tambem é possivel povoar um array na criação:**

```
var myArray = new Array('Olá', myVar, 3.14159);
var myArray = ['Manga', 'Maçã', 'Laranja'];
```

Referenciando os elementos de um array

- **Você pode referenciar os elementos de um array através do uso de elementos numéricos ordinais. Por exemplo, suponha que você definiu o seguinte array:**

```
var myArray = ['Vento', 'Chuva', 'Fogo'];
```

- **Você então se refere ao primeiro elemento do array como *myArray[0]* e ao segundo elemento como *myArray[1]*. O indice dos elementos sempre começa com 0.**

Iteração em arrays

- **Uma operação comum é a de iterar sobre os valores de um array, processando cada elemento de alguma maneira. A maneira mais simples para fazer isso é como segue:**

```
var cores = ['vermelho', 'verde', 'azul'];
for (var i = 0; i < cores.lenght; i++) {
    console.log(cores[i]);
}
```

- **O método *forEach()* disponibiliza um outro jeito de iterar sobre um array:**

```
var cores = ['vermelho', 'verde', 'azul'];
cores.forEach(function(cor) {
    console.log(cor);
});
// vermelho
// verde
// azul
```

Métodos dos arrays