JavaScript e JSON

Instrutor: Klaus Cavalcante

SERPRO

MOTIVAÇÃO

- Este curso tem como pré-requisitos HTML e CSS (básico)
- Mais importante que assistir às aulas, é se dedicar e fazer todos os exercícios propostos.
- Pratique o máximo possível (dentro e fora do curso)!
- Normalmente o sucesso em qualquer projeto depende de 1% de inspiração e 99% de transpiração, ou seja, um pouquinho de talento e muito trabalho duro!

MOTIVAÇÃO

- Links para cursos básicos de HTML
  - https://www.w3schools.com/html/default.asp
  - https://developer.mozilla.org/pt-BR/docs/Web/HTML
  - https://www.devmedia.com.br/html-basico-codigos-html/16596
  - https://www.tutorialspoint.com/html/

- Links para cursos básicos de CSS
  - https://www.w3schools.com/css/default.asp
  - https://developer.mozilla.org/pt-BR/docs/Web/CSS
  - https://www.tutorialspoint.com/css/

JavaScript

AGENDA

- Introdução
- Ambiente de desenvolvimento
- Sintaxe básica e tipos de dados
- Expressões e operadores
- Controle de fluxo e manipulação de erro
- Laços e iteração
- Funções
- Funções de seta (=>)

AGENDA

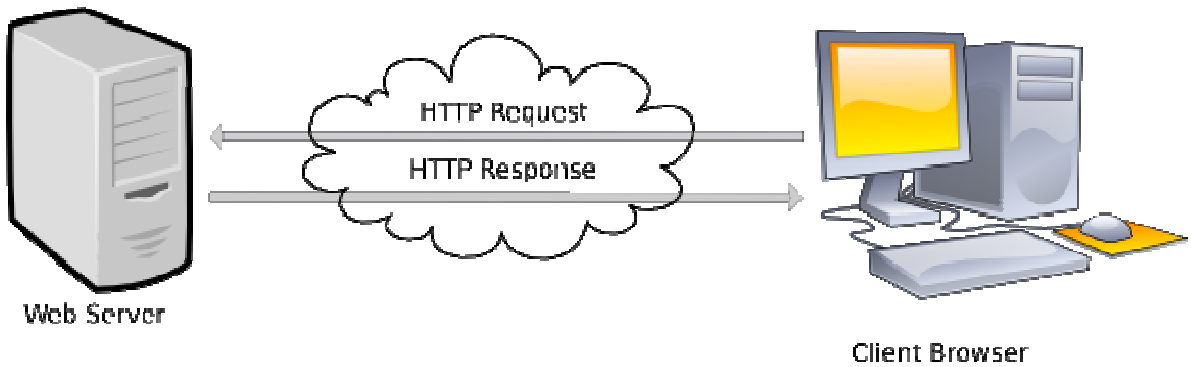
- Objetos
- API de JavaScript
- Programação Assíncrona
- Recursos do ES6

JavaScript

Introdução

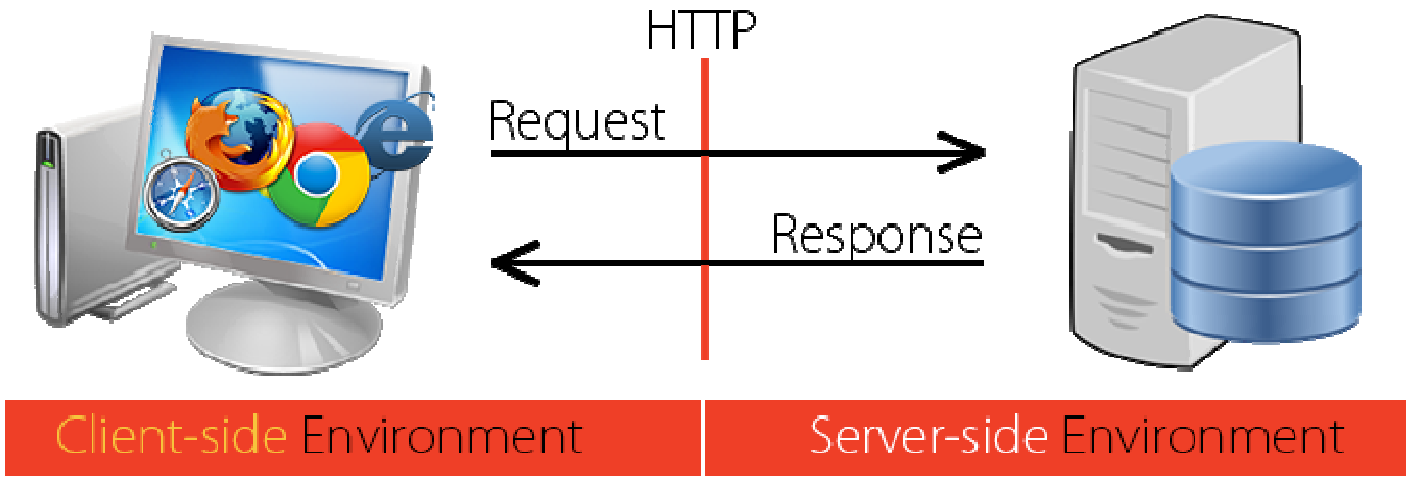
INTRODUÇÃO

- Sistemas web são aplicações que usam uma (ou mais) página(s) HTML como interface, sendo estes acessados através de um navegador (*browser*), e tendo como meio de comunicação o protocolo HTTP.



INTRODUÇÃO

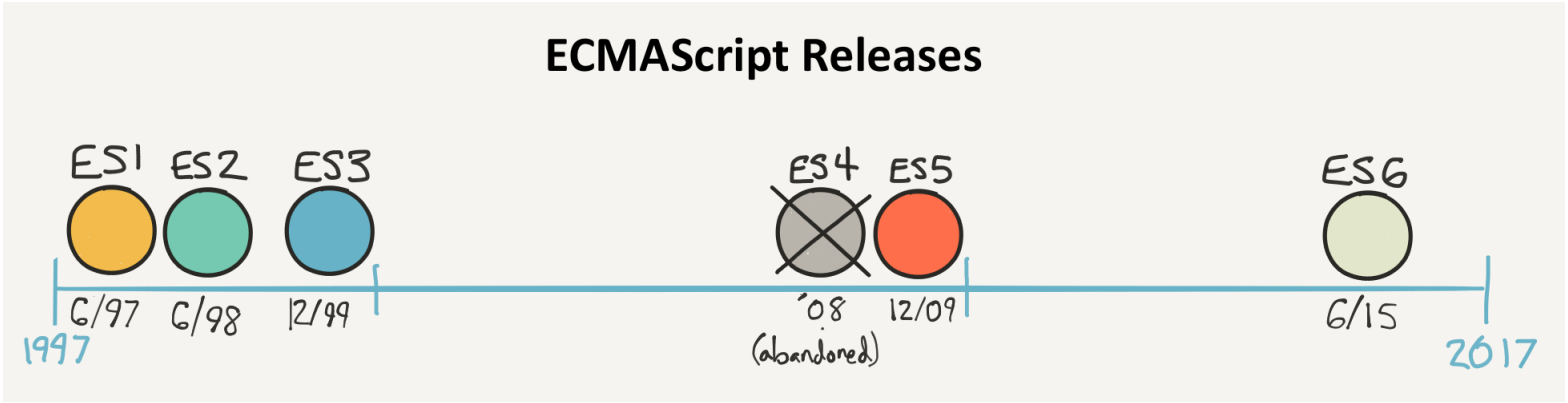
- O desenvolvimento Web utiliza linguagens de programação que são executadas no servidor (*server-side* – Servlets, JSP, JSF, PHP, ASP.NET) ou no navegador do cliente (*client-side* – JavaScript).

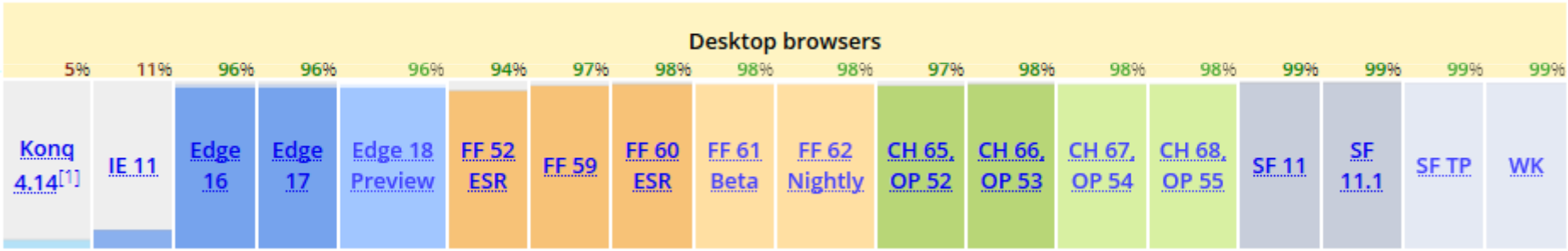


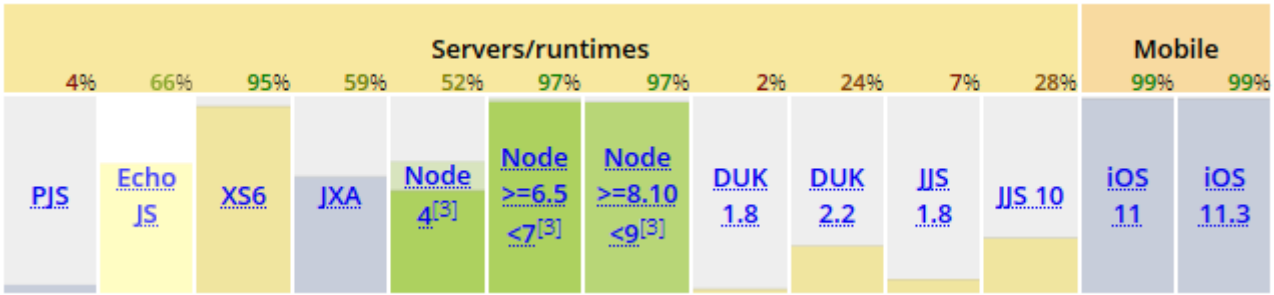
JAVASCRIPT

- JavaScript é a principal linguagem de programação *client-side* em ambiente Web.
- Criada inicialmente com o nome LiveScript, foi desenvolvida para rodar no navegador Netscape (1995).
- É baseada no ECMAScript e padronizada pela ECMA International nas especificações ECMA-262 e ISO/IEC 16262.
- Atualmente os browsers suportam 100% o ECMAScript 5 (ES5), porém os mais recentes já suportam quase completamente a versão 6 (ES6).

ECMA







Fonte: https://kangax.github.io/compat-table/es6 (20/05/2018)

JavaScript

Ambiente de Desenvolvimento

O QUE PRECISAMOS INSTALAR?

- NodeJS + NPM (https://nodejs.org)
- Editor de texto de sua preferência
- Navegador (*browser*) de sua preferência
- Bootstrap (opcional para o curso!)

FERRAMENTAS

- Plataforma para desenvolvimento de aplicações “server-side” usando JavaScript, ideal para prototipações rápidas de pequenas aplicações, e “backend” para aplicações REST
- Quando instalado, permite a criação/manutenção de aplicações através do gerenciador de pacotes NPM (Node Package Manager)
- Baseado na Engine V8 do Google Chrome

https://nodejs.org

INSTALANDO O NODEJS MANUALMENTE

```
$ sudo apt-get update
$ sudo apt-get install nodejs

// atualizar o node para a versão mais recente
$ sudo npm cache clean -f
$ sudo npm install -g n
$ sudo n stable
```

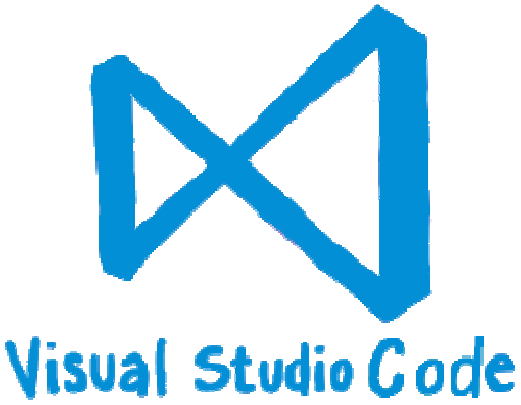
Fonte: https://davidwalsh.name/upgrade-nodejs

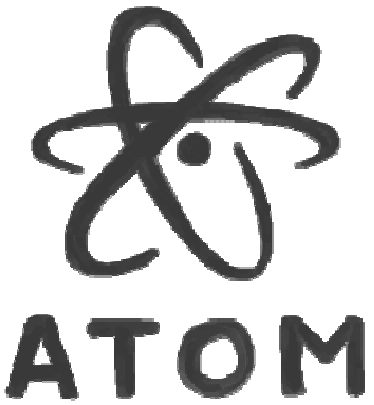
INSTALANDO O NODEJS USANDO O NVM

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/
  v0.33.8/install.sh | bash (após o comando, fechar e abrir
  um novo terminal!)
$ nvm install node
$ node -v (para verificar a instalação)
```

Fonte: https://github.com/creationix/nvm

EDITOR DE TEXTO





VISUAL STUDIO CODE

- Editor de texto gratuito e *open source* da Microsoft
- Possui suporte para diversas linguagens de programação
- https://code.visualstudio.com/download
- Shortcuts para Visual Studio Code
  - https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf

JavaScript

Primeiros Passos

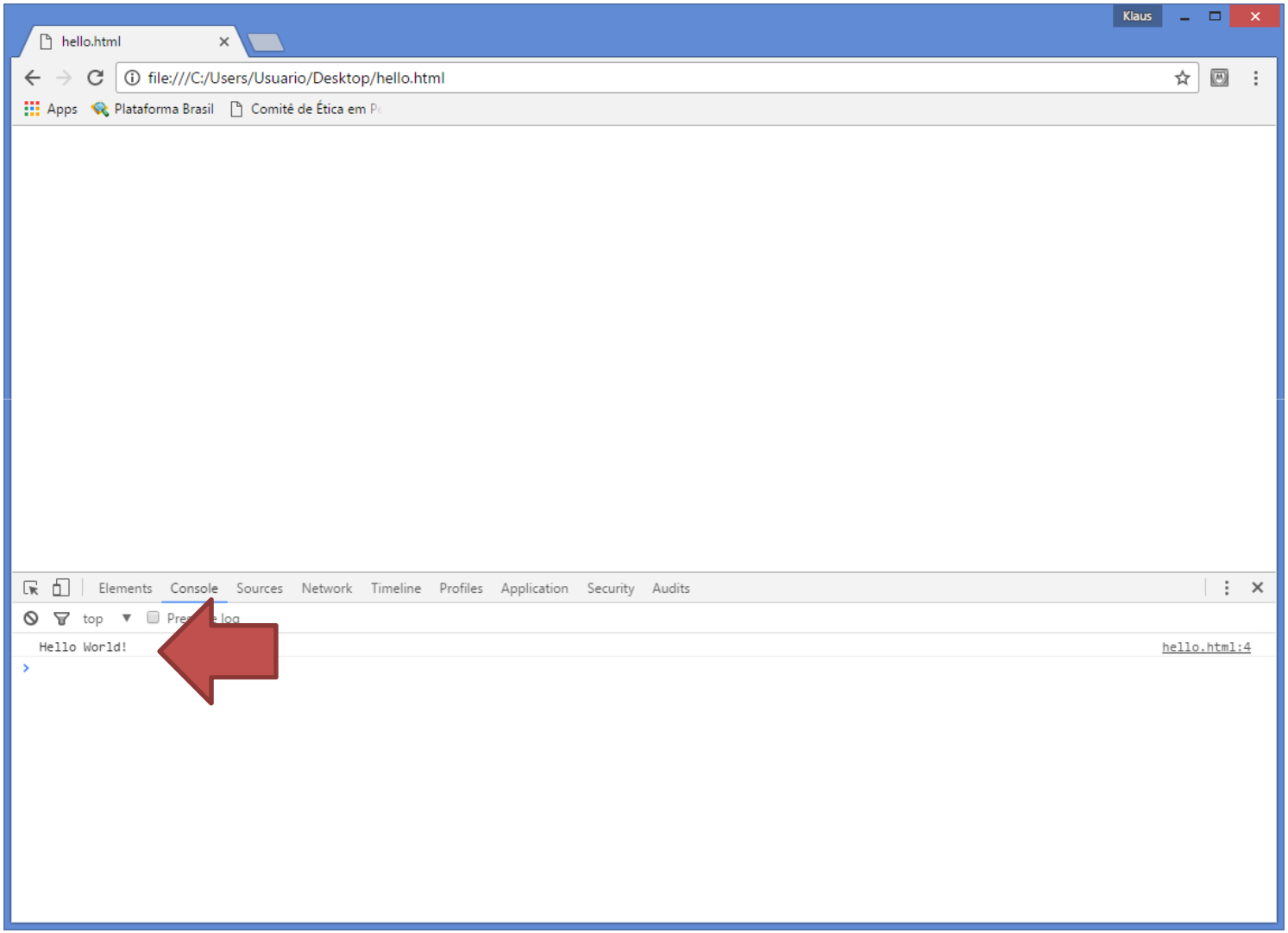
HELLO WORLD

```
<!DOCTYPE html>
<html>
<body>
    <script type="text/javascript">
      console.log('Hello World!');
    </script>
  </body>
</html>



# hello.html
```

HELLO WORLD (CONT.)



HELLO WORLD NO NODEJS

```
console.log('Hello World!');



# hello.js
```

```
$> node hello.js
Hello World!
$>_
```

JAVASCRIPT NA SEÇÃO <HEAD>

```
<html>
<head>
  <meta charset="UTF-8" />
  <script type="text/javascript">
    function exibir() {
      console.log('javascript na seção head!');
    }
  </script>
</head>
<body>
  <input type="button" onclick="exibir()" value="Exibir">/>
</body>
</html>

# load1.html
```

JAVASCRIPT NA SEÇÃO <BODY>

```
<html>
<head>
  <meta charset="UTF-8" />
</head>
<body>
  <input type="button" onclick="exibir()" value="Exibir"/>
  <script type="text/javascript">
    function exibir() {
      console.log('javascript na seção body!');
    }
</script>
</body>
</html>

# load2.html
```

JAVASCRIPT EM ARQUIVO .JS

```
<html><head>
  <meta charset="UTF-8" />
  <script type="text/javascript" src="load4.js"></script>
</head>
<body>
  <input type="button" onclick="exibir()" value="Exibir"/>
</body>
</html>

# load4.html
```

```
function exibir() {
  console.log('javascript em arquivo externo');
}


# load4.js
```

DICA

- O evento '*click*' é um dos eventos DOM existentes que são usados para tornar a página HTML dinâmica
- São usados nas *tags* HTML com o prefixo '*on*' (exs.: *onclick*, *oninput*, *onkeydown*, *onload*, etc.)
- Podem ser usados via JavaScript (addEventListener())
- Quais outros eventos posso usar?
  - http://developer.mozilla.org/en-US/docs/Web/Events

JavaScript

Sintaxe básica e tipos de dados

SINTAXE BÁSICA E TIPOS DE DADOS

- Comentários

```
// comentário de linha
/*
  comentário de bloco
*/
```

- Declarações

var: Declara uma variável global.

let: Declara uma variável global ou local de escopo de bloco.

const : Declara uma constante global ou local de escopo de bloco

SINTAXE BÁSICA E TIPOS DE DADOS

- Declarações (cont.)

```
var a = 10;
b = 20;

console.log(a);
console.log(b);

var c;
console.log(c);
console.log(d);


# variavel.js
```

SINTAXE BÁSICA E TIPOS DE DADOS

- Escopo de Variáveis

```
var a = 10;
let b = 20;

if (true) {
  var c = 30;
  let d = 40;
  const e = 50;
}
console.log(a);
console.log(b);
console.log(c);
console.log(d);
console.log(e);

# escopo.js
```

Em um bloco, variáveis 'let' ou 'const' têm escopo local!

SINTAXE BÁSICA E TIPOS DE DADOS

- *Hoisting* de variáveis

```
var a;
console.log(a);
console.log(b);

var b = 10;
console.log(b);


# hoisting.js
```

Não gera uma exception do tipo ReferenceError!

SINTAXE BÁSICA E TIPOS DE DADOS

- Tipos de dados

1. number (123 ou 120.50)
2. string ("exemplo" ou 'exemplo')
3. boolean (*true* ou *false*)
4. undefined (*undefined*)
5. function (funções)
6. object (objetos e *null*)

SINTAXE BÁSICA E TIPOS DE DADOS

- Tipos de dados (cont.)

console.log(?)	?	var a	var a = null	var a = undefined	var a = ''
a	ReferenceError	undefined	null	undefined	
typeof a	undefined	undefined	object	undefined	string
a == null	ReferenceError	true	true	true	false
a === null	ReferenceError	false	true	false	false
a == undefined	ReferenceError	true	true	true	false
a === undefined	ReferenceError	true	false	true	false
!a	ReferenceError	true	true	true	true

NodeJS v8.7.0

EXERCÍCIO 1

1. Criar uma página HTML que contenha um botão e um e que ao clicar no botão, incremente uma variável
2. Exibir em um o valor da variável
3. Dica: Para manipular o conteúdo do em JavaScript, utilize o método ***document.getElementById()*** para recuperar o respectivo objeto do DOM
4. Modificar página para que ao carregá-la, já seja exibido o valor da variável
5. Dica: Utilize o evento ***load*** na *tag* <body>
6. Incluir uma funcionalidade para decrementar o valor da variável
7. Incluir um <select> com as opções 1, 5 e 10 para ser usada como valor para incremento e decremento. Valor default: 1

JavaScript

Expressões e operadores

EXPRESSÕES E OPERADORES

Aritméticos	`+ - * / % ++ --`
Comparação	`== === != !== > >= < <=`
Lógicos	`&& || !`
Op. bits	`& | ~ ^ >> <<`
Atribuição	`= += -= *= /= %=`
Condicional ou Ternário	`? :`
Verificação de tipos	**`typeof`** `expressão`
Op. new	**`new`** `Tipo(param1, param2, ...);`
Op. delete	**`delete`** `obj ou obj.propriedade`
Op. in	`propriedade` **`in`** `obj`
Op. instanceof	`obj` **`instanceof`** `Tipo`

EXPRESSÕES E OPERADORES

- Exemplos

```
let x = 5;
console.log(x == 5);
console.log(x === 5);
console.log(x == '5');
console.log(x === '5');

console.log(typeof 'João');
console.log(typeof 3.14);
console.log(typeof false);
console.log(typeof [ 1, 2, 3, 4 ]);
console.log(typeof { nome:'João', idade: 34 });
console.log(typeof new Date());
console.log(typeof function () {});
console.log(typeof null);

# op1.js
```

EXPRESSÕES E OPERADORES

- Palavra-chave ***this***
  - O valor do *this* é determinado de acordo com o contexto usado

```
console.log(this === window); // objeto global (window em browsers)

function f1(){
  return this;
}
console.log(f1() === window); // objeto global (window em browsers)

let o1 = {
  prop: 37,
  f: function() { return this.prop; }
};
console.log(o1.f() === o1); // objeto o1


# this1.html
```

EXPRESSÕES E OPERADORES

- A palavra-chave ***this*** como parâmetro em *callbacks* de eventos

```
<html><head><meta charset="UTF-8" />
  <script type="text/javascript">
    function validarIdade(input, minimo, maximo) {
      if (input.value < minimo || input.value > maximo)
        document.getElementById('div').innerHTML = 'Idade inválida';
      else
        document.getElementById('div').innerHTML = '';
    }
  </script>
</head>
<body>
  <input type="text" onkeyup="validarIdade(this, 18, 100);" />
  <div id="div"></div>
</body>
</html>


# this2.html
```

A palavra-chave 'this' assume o objeto input do DOM

EXERCÍCIO 2

1. Criar uma página HTML que leia do usuário um valor em graus Fahrenheit, através de um <input>
2. Calcular o seu equivalente em graus Celsius e exibir os resultados em um . Fórmula => C = (F - 32)*5/9
3. Ao validar o campo, exibir a mensagem "Campo inválido!“, se este não for preenchido ou não corresponder a um valor em ponto flutuante válido.

EXERCÍCIO 3

1. Criar uma página HTML que leia do usuário os votos brancos, nulos e válidos de uma eleição, através de três <input>’s.
2. Calcular o percentual que cada um representa em relação ao total de eleitores e exibir os resultados em um .
3. Ao validar os campos, exibir a mensagem "Campo inválido!" para cada um dos campos que não foram preenchidos ou não correspondam a valores inteiros válidos.

EXERCÍCIO 4 (OPCIONAL)

1. Criar uma página HTML que implemente uma calculadora para op. aritméticas. Incluir três <input>’s, quatro botões (op. aritméticas) e um para saída de mensagens. Dois <input>'s servirão para os operandos e o terceiro (somente leitura) para a resposta da operação
2. Exibir a mensagem "Um ou mais operandos inválidos!" e abortar a operação caso o usuário não tenha digitado valores numéricos válidos
3. Exibir a mensagem "Operação inválida! Divisão por zero!", caso seja verificado tal cenário durante a operação.
4. Incluir botões para duas novas operações: resto de divisão e potência.
5. Incluir um botão de memória (M+) para armazenar o resultado atual calculado e outro botão (M-) para atribuir o valor da memória ao segundo <input>.

JavaScript

Controle de fluxo e manipulação de erro

CONTROLE DE FLUXO E MANIPULAÇÃO DE ERRO

- JavaScript possui as mesmas estruturas de condição:

  if e ***switch***

  ... e de tratamento de exceções de Java:

  throw e ***try-catch-finally***

CONTROLE DE FLUXO E MANIPULAÇÃO DE ERRO

- A novidade está no comando ***switch***:

```
let tipofruta = 'maçã';
switch (tipofruta) {
case 'laranja':
  console.log('O quilo da laranja está R$0,59.');
  break;
case 'banana':
  console.log('O quilo da banana está R$0,48.');
  break;
default:
  console.log('Desculpe, não temos ' + tipofruta + '.');
}
console.log('Gostaria de mais alguma coisa?');

# switch.js
```

Em JavaScript é possível testar strings!

EXERCÍCIO 5

1. Um hotel cobra R$ 50,00 reais a diária e mais uma taxa de serviços. A taxa de serviços é de:

   2,50 por dia, se número de diárias <15

   2,00 por dia, se número de diárias =15

   1,50 por dia, se número de diárias >15

2. Criar uma página HTML que leia do usuário a quantidade de dias que o hóspede ficou no hotel através de um <input>, e imprima a taxa de serviços e o total a pagar.
3. Criar um arquivo JavaScript externo para manter as funções de cálculo da taxa e do total a pagar.

EXERCÍCIO 6

1. Criar uma página HTML com um <select> que contenha 5 tipos de produtos, um <input> para a quantidade a ser comprada e um botão para calcular o total da compra, exibindo o valor em um <input> (somente leitura).
2. Incluir um <select> contendo a forma de pagamento (boleto-10%, cartão-5%) e calcular o desconto da compra de acordo com a forma escolhida.

EXERCÍCIO 7 (OPCIONAL)

1. Criar uma página HTML que leia do usuário o saldo de uma conta bancária e valor da operação, através de dois <input>'s, e o tipo de operação: depósito-1, saque-2, através de um <select>.
2. Calcular e mostrar o novo saldo em um terceiro <input>.
3. Exibir a mensagem "Conta com saldo negativo!", caso o saldo da conta fique negativo.

CONTROLE DE FLUXO E MANIPULAÇÃO DE ERRO

- A cláusula ***throw*** interrompe o fluxo de execução e levanta uma exceção
- As cláusulas ***try-catch*** em conjunto permitem capturar exceções levantadas pelos comandos executados.
- A cláusula ***finally*** é sempre executada em um bloco ***try-catch***, independente se houve ou não a captura de exceções. Seu uso é opcional.

CONTROLE DE FLUXO E MANIPULAÇÃO DE ERRO

- Cláusulas ***throw*** e ***try-catch***

```
function getMes(indice) {
  indice--; // Ajusta o mês (1 = Jan, 12 = Dec)
  let meses = ['Jan','Fev','Mar', ... ,'Out','Nov','Dez'];
  if (meses[indice]) {
    return meses[indice];
  } else {
    throw 'MesInvalido';
  }
}
try {
  console.log(getMes(15));
} catch (e) {
  console.log(e + ' ' + typeof e);
}
# throw.js
```

Não é uma boa prática de programação lançar exceções de tipos diferentes de Error!

CONTROLE DE FLUXO E MANIPULAÇÃO DE ERRO

- Cláusula ***finally***

```
function validar(valor) {
  try {
    if (valor.length === 0)
      throw 'Campo vazio';
  } catch (e) {
    console.log('Exceção capturada: ' + e);
  } finally {
    console.log('Este texto sempre será exibido!');
  }
}
validar('');

# finally.js
```

CONTROLE DE FLUXO E MANIPULAÇÃO DE ERRO

- Tipo ***Error*** (base para exceções definidas pelo usuário – ES6)

```
function tratandoErro() {
  throw new Error('Mensagem de erro!');
}

try {
  tratandoErro();
} catch (e) {
  console.log(e.name);
  console.log(e.message);
}

# error.js
```

EXERCÍCIO 8

1. Criar um arquivo JavaScript contendo uma função para validar se o parâmetro de entrada é um email válido do serpro (ou seja, termina com '@serpro.gov.br'). Caso não seja um email válido, lançar uma exceção contendo o texto "Email inválido! Digite novamente!".
2. Incluir outra função para autenticar email e senha passados como parâmetros de entrada. Obs1: Chame a função criada anteriormente para pré-validar o email. Obs2: Caso email e senha não correspondam a 'admin@serpro.gov.br' e '123456', lançar uma exceção contendo o texto "Login/senha inválidos!"
3. Criar uma página HTML que contenha dois <input>’s e um botão. Ao clicar no botão, validar login usando a função do arquivo externo. Caso o login seja validado, exibir a mensagem "Usuário autenticado com sucesso!". Tratar as exceções apropriadamente e exibir as mensagens em um .

JavaScript

Laços e iteração

ARRAYS

- *Arrays* em Javascript são objetos cujas estruturas são semelhantes àquelas encontrados em outras linguagens:

```
let array1 = [3, 5, 7];
console.log(array1[1]); // 5
console.log(array1.length); // 3

let array2 = [];
console.log(array2[1]); // undefined
console.log(array2.length); // 0

# array.js
```

LAÇOS E ITERAÇÃO

- Javascript também possui os mesmos comandos de repetição:

  for, ***while*** e ***do-while***

  ... e comandos desestruturadores de Java:

  break e ***continue***

  ... e possui mais duas variações do comando '***for***':

  for-of e ***for-in***

LAÇOS E ITERAÇÃO

- O comando ***for-of*** cria um laço para *Arrays* e objetos iterativos (*Maps* e *Sets*) percorrendo todos os valores de um objeto

```
let array = [3, 5, 7, 9];

for (let i of array) {
  console.log(i);
}

# for-of.js
```

- Em contrapartida, o comando ***for-in*** cria um laço percorrendo os nomes das propriedades de um objeto (será visto na seção sobre Objetos!)

EXERCÍCIO 9

1. Criar uma página HTML que leia do usuário uma string (através de um <input>) cujo conteúdo seja uma sequencia de números inteiros separados por vírgula.
2. Converter a string em um array de números com os valores recebidos
3. Dica: Quebre a string passada usando a função ***split()*** da API de String (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String)
4. Verificar se algum número está fora do intervalo válido (1-100). Caso afirmativo, exibir mensagem apropriada na tela.
5. Exiba os números em um separados por ':'

EXERCÍCIO 9 (CONT.)

6. Dica: Monte os valores de um array com a função ***join()*** da API de Array (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array)
7. Usando a estrutura ***for-of***, calcular a soma de todos os valores do array e exibir em um <input> somente leitura.

EXERCÍCIO 10

1. Uma empresa decidiu fazer um levantamento dos candidatos que se inscreveram para preenchimento de vaga no seu quadro de funcionários, utilizando processamento eletrônico. Criar uma página HTML que leia um conjunto de informações para cada candidato, contendo: idade, sexo (1-mas/2-fem), experiência anterior (1-sim/2-não). Calcule:

a) Qtde de candidatos e candidatas;

b) Média de idade dos homens com experiência;

c) Percentagem dos homens > 45 anos;

d) Qtde de mulheres com idade < 35 anos e com experiência;

e) Menor idade entre as mulheres com experiência no serviço.

Obs.: As informações acima devem ser calculadas a partir do primeiro candidato cadastrado.

EXERCÍCIO 11

1. Adaptar a página HTML criada no **Exercício 6** para exibir uma <table> dinâmica com todos os pedidos realizados, apresentando juntamente com o valor final a ser pago calculado para cada linha.
2. Dica: Consulte as propriedades e funções da API de *HTML DOM* (http://www.w3schools.com/jsref/dom_obj_all.asp) e *HTML Elements* (http://www.w3schools.com/jsref/default.asp) de JavaScript.

EXERCÍCIO 12 (opcional)

1. Criar uma página HTML para cadastrar os produtos de uma farmácia usando *arrays*. Cada produto possui os seguintes dados: código, nome, tipo (0-Genérico, 1-Marca), laboratório (0-Bayer, 1-LAFEPE, 2-Pfizer), preço de custo, preço de venda. O preço de venda será o preço de custo acrescido do percentual de lucro de acordo com a tabela abaixo:

	LAFEPE	Pfizer	Bayer
Genérico	10%	15%	20%
Marca	20%	35%	45%

EXERCÍCIO 12 (opcional - CONT.)

2. Exibir uma tabela <table> dinâmica com todos os produtos cadastrados.
3. Dica: Consulte as propriedades e funções da API de *HTML DOM* (http://www.w3schools.com/jsref/dom_obj_all.asp) e *HTML Elements* (http://www.w3schools.com/jsref/default.asp) de JavaScript.

JavaScript

Funções

FUNÇÕES

- Declaração de funções em JavaScript:

```
function <nome>(<argumentos>) {
  <corpo_da_função>
}
```

```
function fat(x) {
  if (x == 1)
    return 1;
  return x * fat(x-1);
}

console.log(fat(4));

# fat.js
```

FUNÇÕES

- Funções podem ser criadas através de expressões

```
let fatorial = function fat (n) {
  return n<2 ? 1 : n*fat(n-1);
};
console.log(fatorial(3));

let square = function (x) {
  return x*x;
};
console.log(square(3));
```

FUNÇÕES

- Funções podem ser criadas através de expressões (cont.)

```
function map(fun, array) {
  let result = [];
  for (i = 0; i != array.length; i++)
    result[i] = fun(array[i]);
  return result;
}
let valores = map(function (x) { return x*x*x; }, [0,1,2,5,10]);
console.log(valores);

# exp_fun.js
```

FUNÇÕES DE SETA (=>)

- Utilizadas para simplificar declarações de funções
- São intrinsecamente anônimas
- Sintaxe:
  - Sem parâmetros: `() => { ... }`
  - C/ um parâmetro: `p => { ... }` ou `(p) => { ... }`
  - C/ vários parâmetros: `(p1, p2, ..., pn) => { ... }`
  - Se o corpo da função for apenas um comando ***return***, podemos simplificá-la da seguinte forma:

    `(x) => { return x*x; }` equivale a `(x) => x*x` ou `x => x*x`

- ***Atenção: Largamente empregadas em Angular!***

FUNÇÕES DE SETA (=>)

```
let a = ['Hidrogênio','Hélio','Lítio','Berílio'];

console.log(a.map(function (s) { return s.length }));
console.log(a.map( s => s.length ));

let notas = [8, 4, 10, 5, 7];
let aprov1 = notas.filter(function (nota) {
  return nota >= 7;
});
console.log(aprov1);
let aprov2 = notas.filter(nota => nota >= 7);
console.log(aprov2);

# fun_seta.js
```

EXERCÍCIO 13

1. Adaptar a página HTML criada no **Exercício 9** para filtrar os números maiores que 50 e exibir em um . Utilizar função de seta na solução.
2. Dica: Filtre os elementos do array com a função ***filter()*** da API de Array (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array)
3. Exibir em um as raízes quadradas dos valores armazenados no array. Utilizar função de seta na solução.
4. Dica: Mapeie os elementos do array com a função ***map()*** da API de Array (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array)

EXERCÍCIO 14

1. Adaptar a página HTML criada no **Exercício 10**, para utilizar funções de seta no cálculo das questões **(a)** até **(e)**.
2. Dica: Filtre os dados dos arrays através da função ***filter()*** da API de Array (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array). Estude os possíveis parâmetros da função passada como argumento para ***filter()***.

JavaScript

Objetos

OBJETOS

- A linguagem JavaScript é projetada com base em um simples paradigma orientado a objeto
- Um objeto consiste em uma coleção de propriedades (chave e valor), similar a um mapa (*Map*)
- Um valor pode ser um número, string, valor booleano, função ou outro objeto
- O acesso ao valor de uma propriedade do objeto pode ser feito de duas formas:
  - **objeto.propriedade**
  - **objeto['propriedade']**

CRIANDO NOVOS OBJETOS (#1)

- Usando o tipo ***Object***

```
let carro1 = new Object();
carro1.marca = 'Renault';
carro1.modelo = 'Sandero';
carro1.ano = 2015;
carro1.toString = function () {
  return this.marca + ' ' + this.modelo + ' (' + this.ano + ')';
};
console.log(carro1.marca);
console.log(carro1['modelo']);
console.log('carro1 = ' + carro1.toString());

# objetos1.js
```

CRIANDO NOVOS OBJETOS (#2)

- Usando inicializadores de objetos

```
let carro2 = {
  marca: 'Fiat',
  modelo: 'Palio',
  ano: 2013,
  toString: function () {
    return this.marca + ' ' + this.modelo + ' (' + this.ano +
')'; }
};
console.log('carro2 = ' + carro2.toString());

# objetos2.js
```

CRIANDO NOVOS OBJETOS (#3)

- Usando uma função construtora

```
function Carro(marca, modelo, ano) {
  this.marca = marca;
  this.modelo = modelo;
  this.ano = ano;
  this.toString = () => this.marca + ' ' + this.modelo + ' (' +
this.ano + ')';
}
let carro3 = new Carro('Honda', 'Fit', 2016);
console.log('carro3 = ' + carro3.toString());

# objetos3.js
```

CRIANDO NOVOS OBJETOS (#4)

- Usando uma classe (ES6)

```
class Pessoa {
  constructor(nome) {
    this.nome = nome;
  }
  getNome() {
    return this.nome;
  }
}
let pessoa = new Pessoa('Zezinho');
console.log(pessoa.getNome());

# objetos4.js
```

INSPECIONANDO OBJETOS

- Através do comando de iteração ***for-in***, é possível percorrer as chaves das propriedades de um objeto, e não dos valores como no ***for-of***:

```
let carro = {
  marca: 'Fiat',
  modelo: 'Palio',
  ano: 2013,
};

for (let prop in carro) {
  console.log(prop);
}

# for-in.js
```

EXERCÍCIO 15

1. Criar um arquivo JavaScript para realizar as seguintes tarefas:
2. Usando as quatro formas de criação de objetos, implementar um tipo Pessoa (nome, idade e endereço), e instanciar/imprimir objetos Pessoa. Obs: O endereço também é um tipo que contém logradouro e número;
3. Implemente métodos ***toString()*** em cada um dos tipos criados e imprima os objetos Pessoa utilizando este método.
4. Criar uma página HTML que leia do usuário os dados de uma pessoa e um botão para imprimir suas informações através da função ***toString()***. Obs.: Todos os campos são obrigatórios

EXERCÍCIO 16

1. Adaptar a página HTML criada no **Exercício 14** para manter os dados dos candidatos em um *array* de objetos do tipo Candidato.

EXERCÍCIO 17 (OPCIONAL)

1. Adaptar a página HTML criada no **Exercício 11** para manter os dados dos pedidos em um *array* de objetos do tipo Pedido.

EXERCÍCIO 18 (OPCIONAL)

- Adaptar a página HTML criada no **Exercício 12** para manter os dados dos produtos da farmácia em um *array* de objetos do tipo Produto.

JavaScript

API de JavaScript

API DE JAVASCRIPT

- Strings (***String***)
  - concat(), replace(), substring(), toLowerCase(), toUpperCase(), ...

```
let str = new String('hello');
str = str.concat(' world!');
console.log(str);
str = str.replace('world', 'javascript');
console.log(str.toUpperCase());
console.log(str.toLowerCase());
console.log(str.substring(0, 2));

# string.js
```

API DE JAVASCRIPT

- Expressões Regulares (***Regexp***)
  - Normalmente usadas em buscas/atualização de Strings

```
let re = new RegExp('natal','i');
let str1 = 'Hoje é dia de Natal!';
console.log(str1.replace(re, 'São João'));
// Hoje é dia de São João

let str2 = 'Laranjas são redondas, e laranjas são deliciosas.';
console.log(str2.replace(/laranjas/gi, 'ameixas'));
// ameixas são redondas, e ameixas são deliciosas.
```

API DE JAVASCRIPT

- Expressões Regulares (***Regexp***) (cont.)
  - Normalmente usadas em buscas/atualização de Strings

```
let str3 = 'Klaus Cavalcante';
console.log(str3.replace(/^(\w+)\s(\w+)$/, '$2, $1'));
// Cavalcante, Klaus

# regexp.js
```

Significado dos caracteres especiais nas expressões regulares:
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/ Global_Objects/RegExp

EXERCÍCIO 19

1. Criar uma página HTML que contenha um <input> para armazenar um CEP. Deve ser aplicada uma máscara correspondente durante a digitação.
2. Dica: Usar o evento ***oninput*** do elemento <input> para tratar a digitação do CEP pelo usuário
3. Criar um novo <input> para armazenar um CPF. Aplicar a máscara correspondente durante a digitação.
4. Dica: Para substituir os caracteres digitados e aplicar uma máscara, considere utilizar a função ***replace()*** da API de String (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String) com uma expressão regular (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/RegExp).

API DE JAVASCRIPT

- Datas (***Date***)
  - getDate(), getMonth(), getFullYear(), getHours(), getMinutes(), getSeconds(), toDateString(), toLocaleDateString(), ...

```
let someday = new Date(2000, 5, 20);
console.log(someday.getDate() + '/' + (someday.getMonth()+1) +
'/' + someday.getFullYear());

let today = new Date();
console.log(today.toDateString());
console.log(today.toLocaleDateString());

# date.js
```

API DE JAVASCRIPT

- Números (***Number***)
  - Number.MAX_VALUE, Number.MIN_VALUE, parseInt(), parseFloat(), isFinite(), isInteger(), isNaN()

```
let num = Number.parseFloat('12345.6789');
console.log(num.toFixed());                  // 12346
console.log(num.toFixed(2));                 // 12345.68
console.log(num.toFixed(5));                 // 12345.67890
console.log(num.toPrecision(1));             // 1e+4
console.log(num.toPrecision(2));             // 1.2e+4
console.log(num.toPrecision(4));             // 1.235e+4

# number.js
```

API DE JAVASCRIPT

- Funções Matemáticas (***Math***)
  - sin(), cos(), tan(), round(), trunc(), floor(), ceil(), pow(), exp(), log10(), ...

```
function getRandomArbitrary(min, max) {
    return Math.round(Math.random() * (max - min) + min);
}
console.log(getRandomArbitrary(1, 60));

# math.js
```

API DE JAVASCRIPT

- Arrays (***Array***)

```
let arr1 = [42, 15, 10, 43, 79];
let arr2 = new Array(42, 15, 10, 43, 79);

console.log(arr1); // [ 42, 15, 10, 43, 79 ]
console.log(arr2); // [ 42, 15, 10, 43, 79 ]

arr2.push(17); // [ 42, 15, 10, 43, 79, 17 ]
arr2.pop(); // [ 42, 15, 10, 43, 79 ]

arr2.shift(); // [ 15, 10, 43, 79 ]
arr2.unshift(99); // [ 99, 15, 10, 43, 79 ]
```

API DE JAVASCRIPT

- Arrays (***Array***) (cont.)

```
arr2.sort();
// [ 10, 15, 43, 79, 99 ]

arr2.sort( (a, b) => {
  if (a === b) return 0;
  if (a < b) return 1;
  if (a > b) return -1;
});
// [ 99, 79, 43, 15, 10 ]
```

API DE JAVASCRIPT

- Arrays (***Array***) (cont.)

```
arr2.forEach((item, i) => {
  console.log('arr[' + i + ']=>' + item);
});
// arr[0]=>99 arr[1]=>79 arr[2]=>43 arr[3]=>15 arr[4]=>10

console.log(arr2.map((item, i) => item*10));
// [ 990, 790, 430, 150, 100 ]
console.log(arr2.filter((item, i) => item % 3 == 0)); // 99, 15

console.log(arr2.indexOf(43)); // 2
console.log(arr2.join(';')); // 99;79;43;15;10

# arrays.js
```

API DE JAVASCRIPT

- Arrays (***Array***) (cont.)
  - slice() x splice()

```
let arr = [ 4, 2, 8, 5 ];
let res1 = arr.slice();
console.log(res1); // [ 4, 2, 8, 5 ]

let res2 = arr.slice(1);
console.log(res2); // [ 2, 8, 5 ]

let res3 = arr.slice(1, 3);
console.log(res3); // [ 2, 8 ]
```

API DE JAVASCRIPT

- Arrays (***Array***) (cont.)
  - slice() x splice() (cont.)

```
let arr1 = [ 4, 2, 8, 5 ];
arr1.splice(2);
console.log(arr1); // [ 4, 2 ]

let arr2 = [ 4, 2, 8, 5 ];
arr2.splice(2, 1);
console.log(arr2); // [ 4, 2, 5 ]

let arr3 = [ 4, 2, 8, 5 ];
arr3.splice(2, 1, 10, 20);
console.log(arr3); // [ 4, 2, 10, 20, 5 ]
#slice_splice.js
```

API DE JAVASCRIPT

- Arrays (***Array***) (cont.)
  - some() x every()

```
[1,2,3].some(t => t === 1); // true
[2,3]  .some(t => t === 1); // false
[1]    .some(t => t === 1); // true
[]     .some(t => t === 1); // false

[1,2,3].every(t => t === 1); // false
[2,3]  .every(t => t === 1); // false
[1]    .every(t => t === 1); // true
[]     .every(t => t === 1); // true

#some_every.js
```

DICA

- Lodash (https://lodash.com) é uma biblioteca JavaScript que fornece diversas funções utilitárias para trabalhar com *arrays*, objetos, strings entre outros

EXERCÍCIO 20

1. Criar uma página HTML que contenha um <input> e um botão. Ao clicar no botão, verificar se o conteúdo do <input> possui mais de 3 caracteres. Caso afirmativo exibir o conteúdo num e caso contrário, exibir uma mensagem para o usuário.
2. Criar dois botões para tornar todas as letras do conteúdo do <input> maiúsculas e minúsculas respectivamente
3. Criar um <input> e um botão. Verificar se o conteúdo do segundo <input> é uma substring do primeiro
4. Criar um botão para verificar se o conteúdo do primeiro <input> é iniciado pelo conteúdo do segundo
5. Dica: Consulte as funções da API de String (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String)

EXERCÍCIO 21

1. Adaptar a página HTML do **Exercício 13** para permitir a exibição do *array* com qualquer separador (utilizar um <input> para sua digitação)
2. Incluir uma funcionalidade de inclusão de um item tanto no início quanto no final do *array*
3. Incluir a funcionalidade de ordenação de forma crescente e decrescente (utilizar um <select> para as opções)
4. Incluir um <input> apenas leitura com a data e hora atual com a seguinte formatação: (dd/mm/yyyy hh:mm:ss, preencher com zero os valores menores que 10 onde for necessário, ex.: 03/02/2017 10:07:17).

EXERCÍCIO 22

1. Criar uma página HTML para simular um sistema de sorteio da Mega Sena. A página deverá conter um painel (<fieldset>) contendo 60 <checkbox>'s (gerados dinamicamente), numerados de 1 a 60 no formato semelhante aquele utilizado pelas lotéricas para armazenar a aposta de um jogador.
2. Incluir um botão para sortear 6 números entre 1 e 60. Obs.: Descartar números repetidos durante o "sorteio".
3. Dica: Para gerar números aleatórios, utilize a função random() da API de Math (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math)

EXERCÍCIO 22 (CONT.)

4. Validar se o jogador escolheu exatamente 6 números do painel. Caso tenha selecionado um número diferente, exibir a mensagem "Jogo inválido! Por favor, selecionar 6 dezenas!".
5. Exibir em um a mensagem "Parabéns! Você acaba de ficar milionário!" , caso a aposta seja ganhadora, ou "Infelizmente não foi dessa vez! Jogue novamente!", caso contrário.
6. Dica: Para verificar se os números escolhidos batem com os sorteados, considere utilizar a função ***every()*** da API de Math (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math) que retorna 'true' se cada elemento no array satisfaz uma função de teste (que neste caso, significa o número escolhido estar entre os números sorteados).

JavaScript

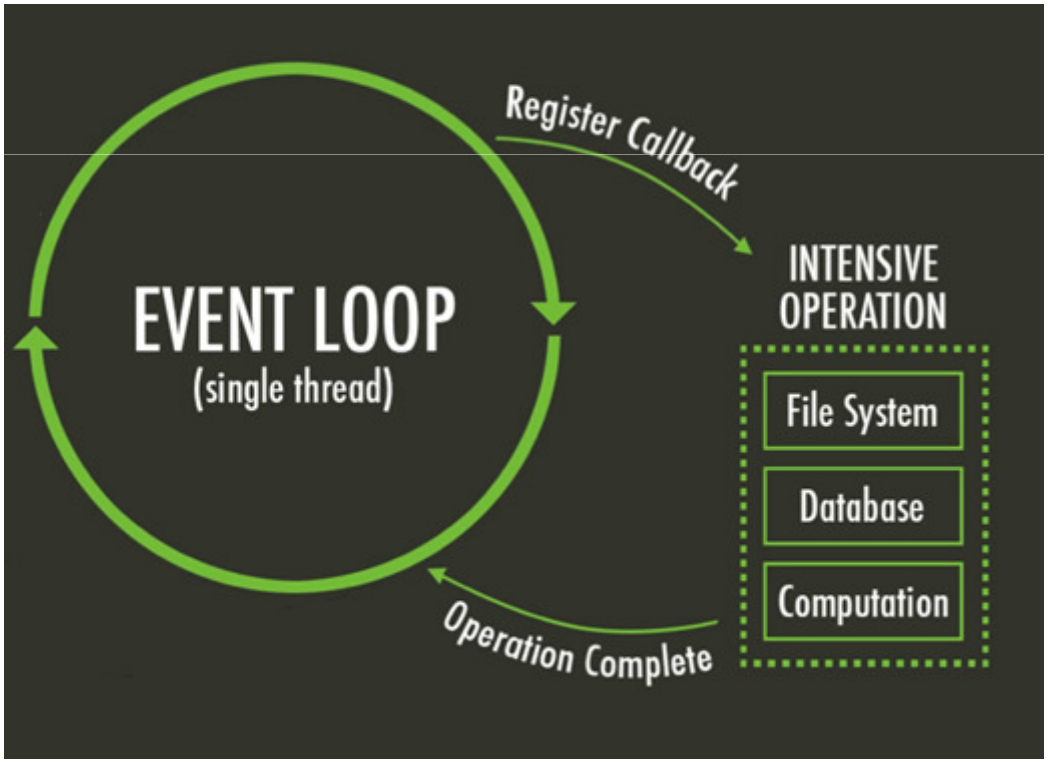
Programação Assíncrona

PROGRAMAÇÃO ASSÍNCRONA

- A programação assíncrona é um dos pontos principais da linguagem, uma vez que JavaScript roda em uma única *thread*
- Para evitar o bloqueio desta *thread*, operações demoradas como requisições HTTP, acesso a disco, ou a um banco de dados são tipicamente executadas de forma assíncrona
- O interpretador (*engine*) da linguagem utiliza outras *threads* para realizar estas operações, enquanto que o código que solicita a operação e o código do ***callback*** que trata o resultado executam na *thread* única dedicada ao código JavaScript

PROGRAMAÇÃO ASSÍNCRONA

- Esta thread única executa um *loop* (*event loop*) que possui uma fila com os eventos resultantes das operações assíncronas



PROGRAMAÇÃO ASSÍNCRONA

- A função *setTimeout()* registra um evento de "*timer*" que executa uma função após um tempo especificado

```
console.log('Antes da chamada do setTimeout()...');
setTimeout(function() {
  console.log('Chamada da função de callback!');
}, 3000);
console.log('Após chamada do setTimeout()...');
```

Qual a sequência de mensagens exibida no log do console?

```
# timeout.js
```

PROGRAMAÇÃO ASSÍNCRONA

- Usando *setTimeout()* para simular uma requisição assíncrona

```
class HttpFake {
  get(url, callback) {
    // simulando uma requisicao HTTP assíncrona
    setTimeout(function () {
      console.log('Resposta do servidor recebida após 3s...');
      // simulando resposta OK
      const response = { status: 200 }; // HTTP 200 OK
      callback(response);
    }, 3000);
  }
}

# async_http.js
```

PROGRAMAÇÃO ASSÍNCRONA

- Usando *setTimeout()* para simular uma requisição assíncrona

```
let http = new HttpFake();
console.log('Simulando uma requisição HTTP a um servidor backend...');
http.get('http://localhost:8080/api/cursos', function (response) {
  if (response.status === 200) {
    console.log('Tratando sucesso (status = ' + response.status + ')');
  } else {
    console.log('Tratando erro (status = ' + response.status + ')');
  }
});
console.log('Após a requisição HTTP...');

# async_http.js
```

PROMISES

- Uma *Promise* é um recurso muito usado para proc. assíncrono, pois permite a associação de *callbacks* para resultados de sucesso ou falha em eventos assíncronos

```
class HttpPromise {
  get(url) {
    return new Promise(function(resolve) {
      // simulando uma requisicao HTTP assíncrona
      setTimeout( function() {
        console.log('Resposta do servidor recebida após 3s...');
        // simulando resposta OK
        const response = { status: 200 }; // HTTP 200 OK
        resolve(response);
      }, 3000);
    });
  }
}
```

PROMISES

```
let http = new HttpPromise();
console.log('Simulando uma requisição HTTP a um servidor backend...');
let promise = http.get('http://localhost:8080/api/cursos');
promise.then(function (response) {
  console.log('Tratando sucesso (status = ' + response.status + ')');
});
console.log('Após a requisição HTTP...');

# async_promise.js
```

PROMISES

- Adicionando tratamento de falha/rejeição

```
...
    return new Promise(function(resolve, reject) {
      // simulando uma requisicao HTTP assíncrona
      setTimeout( function() {
        console.log('Resposta do servidor recebida após 3s...');
        // simulando resposta OK
        const response = { status: 404 }; // HTTP 404 NOT FOUND
        if (response.status === 200) {
          resolve(response);
        } else {
          reject(response);
        }
      }, 3000);
    });
```

PROMISES

```
let http = new HttpPromise();
console.log('Simulando uma requisição HTTP a um servidor backend...');
let promise = http.get('http://localhost:8080/api/cursos');
promise.then(function (response) {
  console.log('Tratando sucesso (status = ' + response.status + ')');
}, function (response) {
  console.log('Tratando erro (status = ' + response.status + ')');
});
console.log('Após a requisição HTTP...');
```

 # async_promise.js

JavaScript

Recursos do ES6

ES6 - TEMPLATE STRINGS

```
const multilinhas = `texto string linha 1
texto string linha 2
texto string linha 3`;

console.log(multilinhas);

const nome = 'Rex';
const idade = 2;
const frase = `Meu cachorro ${nome} tem ${idade*7} anos.`;

console.log(frase);

#es6_ts.js
```

ES6 - VALORES DEFAULT PARA FUNÇÕES

```
function soma(a = 0, b = 0) {
  return a + b;
}
console.log(soma(1, 2)); // 3
console.log(soma(1)); // 1

#es6_vdf.js
```

ES6 - GETTERS E SETTERS

```
class Pessoa {
  constructor(nome) {
    this._nome = nome;
  }
  get nome() {
    return this._nome;
  }
  set nome(valor) {
    this._nome = valor;
  }
}

let pessoa = new Pessoa('Zezinho');
pessoa.nome = 'Huguinho';
console.log(pessoa.nome);

#es6_getset.js
```

ES6 - HERANÇA

```
class Funcionario extends Pessoa {
  constructor(nome, departamento) {
    super(nome);
    this.departamento = departamento;
  }
  getDepartamento() {
    return this.departamento;
  }
}

let func = new Funcionario('Capt. Nascimento', 'SUPDE');
console.log(func.nome + ' - ' + func.getDepartamento());

#es6_heranca.js
```

ES6 - COLEÇÕES

```
let carros = new Set();
carros.add('Ferrari');

console.log(carros.has('Ferrari')); // true
console.log(carros.has('Honda'));   // false

carros.delete('Ferrari');

console.log(carros.has('Ferrari'); // false

let map = new Map();
map.set('server', 'http://git.serpro/');
map.set('project', 'sief-angular');
console.log(map.get('server') + map.get('project'));

#es6_colecoes.js
```

JSON

JSON

- JSON significa *JavaScript Object Notation*
- É um formato para armazenar e transportar dados
- É geralmente usado quando dados são trafegados entre a página *web* (*frontend*) e o servidor (*backend*) da aplicação
- É independente de linguagem
- É menos “verboso” que XML
- É auto-explicativo e fácil de entender
- É 99% semelhante a um objeto JavaScript!

JSON

- Dados em pares de chave/valor envoltos em chaves
- Itens separados por vírgulas
- Suporta objetos (que são envoltos em chaves)
- Suporta *arrays* (que são envoltos em colchetes)
- Regras
  - As chaves das propriedades sempre são envoltas em aspas duplas
  - Valores númericos/booleanos não precisam de aspas, com exceção de *strings*

JSON

- Formato JSON

```
{ "nome": "Zezinho", "idade": 30, "endereco": {
    "logradouro": "Rua sem Saída", "numero": 100
  }, "telefones": [ "9999-9999", "9999-9955"]
}
```

- Objeto JavaScript

```
{ nome: 'Zezinho', idade: 30, endereco: {
    logradouro: 'Rua sem Saída', numero: 100
  }, telefones: [ '9999-9999', '9999-9955']
}
```

JSON

- Convertendo objeto JavaScript para JSON
  - *JSON.stringify()*

```
let pessoa = { nome: 'Zezinho', idade: 30, endereco: {
logradouro: 'Rua sem Saida', numero: 100 }, telefones: [
'9999-9999', '9999-9955']};
let jsonPessoa = JSON.stringify(pessoa);
```

- Convertendo JSON para objeto JavaScript
  - *JSON.parse()*

```
let pessoa1 = JSON.parse(jsonPessoa);
let pessoa2 = JSON.parse('{ "nome": "Zezinho", "idade":
30, "endereco": { "logradouro": "Rua sem Saida", "numero":
100 }, "telefones": [ "9999-9999", "9999-9955"]}');
# json.js
```

EXERCÍCIO 23

1. Adaptar a página HTML do **Exercício 15** para incluir um botão que apresente num <textarea> o objeto Pessoa em formato JSON.
2. Incluir um botão para importar do <textarea> uma string no formato JSON de uma pessoa e carregue nos campos <input> do formulário.

EXERCÍCIO 24

1. Adaptar a página HTML criada no **Exercício 16** para incluir um botão que apresente num <textarea> o *array* de candidatos em formato JSON.

EXERCÍCIO 25 (OPCIONAL)

1. Adaptar a página HTML criada no **Exercício 17** para incluir um botão que apresente num <textarea> o *array* de pedidos em formato JSON.

EXERCÍCIO 26 (OPCIONAL)

- Adaptar a página HTML criada no **Exercício 18** para incluir um botão que apresente num <textarea> o *array* de produtos em formato JSON.

EXERCÍCIO FINAL

- Implementar um CRUD completo (*frontend* + *backend* + BD) para uma entidade escolhida qualquer, utilizando as seguintes tecnologias:
- Frontend:
  - HTML;
  - JavaScript; e,
  - JQuery (p/ as requisições AJAX ao *backend*)
- Backend:
  - NodeJS; e,
  - Express.
- Banco de Dados:
  - MySQL

EXERCÍCIO FINAL

Estrutura do projeto:

```
projetojs/
  +- package.json
  +- backend.js
  +- frontend/
       +- alterar.html
       +- incluir.html
       +- index.html
       +- listar.html
```

PASSOS PARA O DESENVOLVIMENTO DO PROJETO

1. Criação do projeto + instalação do servidor Web (Express)

   http://expressjs.com/en/starter/installing.html

2. Utilização do Express no projeto do *backend*

   http://expressjs.com/en/starter/hello-world.html

3. Link dos arquivos estáticos (HTML) do *frontend* com o Express

   http://expressjs.com/en/starter/static-files.html

4. Instalação e utilização do pacote NPM do MySQL

   https://www.npmjs.com/package/mysql

DICA: REQUISIÇÕES AJAX COM JQUERY

Instalação do JQuery (3.1.1 minified) no *frontend*

https://code.jquery.com

Chamada AJAX: `$.ajax({name:value, name:value, ... })`

http://www.w3schools.com/jquery/ajax_ajax.asp

http://api.jquery.com/jquery.ajax/

REFERÊNCIAS

1. http://www.w3schools.com/jsref/
2. http://www.w3schools.com/js/js_json_intro.asp
3. https://www.tutorialspoint.com/javascript/
4. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
5. https://developer.mozilla.org/en-US/docs/Web/JavaScript/ Reference
6. https://nodejs.org/
7. https://code.jquery.com/

Dúvidas?

Agradecemos pela atenção.

Klaus Cavalcante
klaus.cavalcante@serpro.gov.br

SERPRO