

Introdução a Programação com Javascript

Prof. Alisson G. Chiquitto¹

¹WebDev ALFA

Desenvolvimento de Aplicações para Web e Dispositivos Móveis

Umuarama, 2017



Outline

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

Apresentação pessoal



Alisson Gaspar Chiquitto

Amante de tecnologias Open Source e compartilhamento do conhecimento.
Trabalho com desenvolvimento de Sistemas para Internet e Aplicativos para
Dispositivos Móveis. Nas horas vagas estudo.

Apresentação pessoal

Formação Acadêmica

- 2007 - Graduação em Análise e Desenvolvimento de Sistemas (UNIPAR - Cianorte)
- 2009 - Pós-graduação em Desenvolvimento de Sistemas para internet (UNIPAR - Cianorte)
- 2012 - MBA em Gerencia de Projetos e Governança de TI (UNIPAR - Umuarama)
- 2015 - Mestre em Ciência da Computação (UEM - Maringá)

Apresentação pessoal

Experiência profissional

- Desenvolvimento de Aplicações para Internet (desde 2003);
- Desenvolvimento de Aplicações para Dispositivos Móveis (desde 2015);
- Professor (desde 2011);



Código: Calculadora em Javascript



Esta calculadora resolve os seus problemas. É só descompactar e usar. Totalmente em .htm .É so vc ter no seu pc um navegador instalado

Autor:	Alisson Gaspar Chiquitto
Data de inclusão:	28/01/2002
Último acesso:	21/10/2015
Downloads:	3.763
Avaliação média:	★★★★☆ (baseado em 28 votos)

!015);

Baixar Código

Apresentação pessoal

Contato

- chiquitto@gmail.com;
- github.com/chiquitto;

- chi
- git



Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6













node



ionic



CoffeeScript

{JSON}

node



TypeScript

AJAX



ionic



CoffeeScript

{JSON}





Javascript

- JavaScript é uma linguagem de programação interpretada;
- Javascript é uma linguagem que adiciona interatividade à páginas *HTML*;
- Foi originalmente implementada como parte dos navegadores web para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor, controlando o navegador, realizando comunicação assíncrona e alterando o conteúdo do documento exibido.

Inserção de Javascript no HTML

Para adicionar códigos JavaScript à uma página devemos usar a tag `<script>`. Existem três maneiras de se inserir JavaScript em um documento HTML:

inline inserir o script diretamente na seção body do documento.

Trata-se de prática não recomendada de acordo com o princípio da separação das camadas de desenvolvimento;

incorporado inserir o script na seção head do documento;

externo escrever o script em um arquivo externo e carrega-lo no HTML;

Javascript Inline

Inserir script inline é uma prática que pertence ao passado e deve ser evitada pelo desenvolvedor comprometido com os Padrões Web. Ao escrevermos scripts dentro da marcação HTML, estaremos misturando as camadas de marcação e comportamento, dificultando a manutenção e o entendimento dos códigos.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Hello World</title>
</head>
<body onload="window.alert('Ola mundo');">

</body>
</html>
```

Javascript Incorporado

Cria-se uma tag `<script>`, então, coloca-se o código JavaScript dentro dessa tag.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Hello World</title>
  <script>
    window.alert('Ola mundo');
  </script>
</head>
<body>

</body>
</html>
```

Javascript Externo

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Hello World</title>
</head>
<body>
  <script src="js-externo.js"></script>
  <script src="http://site.com.br/arquivo.js"></script>
</body>
</html>
```

Note o uso do atributo `src` para indicar o caminho para o arquivo que contém o *script*.

Comentários no Javascript

É possível fazer dois tipos de comentários: um linha ou de várias linhas.

```
// Comentario de uma linha
```

```
/*  
Comentario de varias linhas  
*/
```


Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

Identificadores

Os identificadores são utilizados para identificar uma variável ou função.

Quando criamos identificadores temos de levar em consideração algumas regras específicas do *JavaScript*:

- Os identificadores são *case-sensitive*. Isso significa que nomes com letras maiúsculas são diferentes de nomes com letras minúsculas: Nome é diferente de nome.
- Palavras: embora a maioria dos navegadores reconheça caracteres especiais, o recomendado é o uso de apenas letras;
- Números: desde que sejam precedidos por uma ou mais letras;
- Underline (_) e cifrão (\$): são permitidos em qualquer posição, mas pouco usados.

Declaração de variáveis

A declaração de uma variável é feita através da palavra chave **var**, seguido pelo seu identificador. Opcionalmente podemos inicializar a variável atribuindo a ela um valor inicial.

```
// Declaracao de variavel
var i; // Uma variavel
var a, b; // Duas ou mais variaveis

// Inicializar (atribuir valor) a variavel
i = 0;
a = 1, b = 5;

// Declarar e inicializar a variavel
var i = 0;
var a = 1, b = 5;
```

Declaração de variáveis

```
// Identificadores validos
var pais = 'Brasil';
var populacao2015 = 175000000;
var $ = 'cifrao';
var $10 = 'cifrao 10';
var $a = 'cifrao a';
var _ = 'underscore';
var _10 = 'underscore 10';
var _a = 'underscore a';

// Identificadores invalidos
var 3 = 5;
var 1pais = 'EUA';
var nome completo = 'Fujiro Nakombi';
```

Tipos de variáveis

Primitivos:

- Números (*Number*);
- Sequência de caracteres (*String*);
- Booleano (*Boolean*);
- Nulo (*null*);
- Indefinido (*undefined*);

Compostos:

- Objeto (*Object*);

Tipo: Number

O tipo *Number* é utilizado para representar números, inteiros ou ponto flutuante.

```
var centavos = 35;  
var pi = 3.14;  
var saldo = -100;  
var pi2 = -3.14;
```

Tipo: String

O tipo *String* é utilizado para representar cadeias de caracteres, e o valor deve estar contido entre um par de apóstrofo/aspas.

```
var nome = "Josefina";  
var dia = 'Segunda-feira';  
var mes = 'Dezembro';
```

Tipo: Boolean

O tipo *Boolean* é utilizado para representar valores booleanos (verdadeiro/falso). Os valores devem ser representados pelas palavras chave `true` e `false`.

```
var jsLegal = true;  
var isSeculoXX = false;  
var mostrarValor = false;  
var maisLocoQueOBatman = true;
```


Tipo: null

Em *JavaScript* o valor `null` é nada.

```
var nada = null;
```

Tipo: undefined

Em *JavaScript* uma variável é `undefined` quando ela não possui valor.

```
var i;  
var a = undefined;
```

Tipo: Object

Objetos são como uma espécie de "super variáveis" que armazenam uma "coleção de valores" referenciados por um identificador.

Alguns tipos de variáveis de outras linguagens, no *JavaScript*, são considerados Objetos, como é o caso do vetor (*array*).

Tipo: Object

Array

Os *arrays* no *JavaScript*, são os conjuntos de zero ou mais valores, separados por vírgula. Os valores contidos em um *array* recebem um índice sequencial começando com zero.

Tipo: Object

Criando array

```
var vazio = new Array();  
var numeros = new Array(1, 2, 3, 4);  
var letras = new Array('a', 'b', 'c');
```

Notação simplificada:

```
var vazio = [];  
var numeros = [1,2,3,4];  
var letras = ['a', 'b', 'c'];
```

Tipo: Object

Acessando elementos do array

Com o nosso *array* criado podemos visualizar cada uma das posições individualmente colocando o índice dentro de colchetes:

```
var frutas = ['abacate', 'abacaxi', 'laranja'];  
  
console.log(frutas[0]); // abacate  
console.log(frutas[1]); // abacaxi  
console.log(frutas[2]); // laranja
```

Tipo: Object

Alterando valores de um array

Também podemos alterar o valor de cada posição da seguinte forma:

```
var frutas = ['abacate', 'abacaxi', 'laranja'];  
  
console.log(frutas[0]); // abacate  
  
frutas[0] = 'melancia';  
  
console.log(frutas[0]); // melancia
```

Tipo: Object

Criando Objetos literais

Um objeto literal é composto por um par de chaves "{ }", que envolve uma ou mais propriedades.

Cada propriedade segue o formato "nome:valor" e devem ser separadas por vírgula.

```
var cliente = {  
  nome: 'Fujiro Nakombi',  
  idade: 50,  
  email: 'fnakombi@gmail.com'  
};
```


Tipo: Object

Acessando propriedades

Após ter criado um objeto, você vai precisar acessar os valores que ele armazena. Podemos acessar (ou se preferir: "recuperar") os valores guardados em um objeto, de duas maneiras: utilizando notação de ponto ou notação de colchetes. Veja um exemplo:

```
var cliente = {  
  nome: 'Fujiro Nakombi',  
  idade: 50,  
  email: 'fnakombi@gmail.com'  
};  
  
// Fujiro Nakombi  
console.log(cliente.nome);  
console.log(cliente['nome']);
```

Tipo: Object

Alterando e adicionando propriedades

Basta acessar a propriedade que deseja alterar e atribuir o novo valor.

```
var cliente = {  
  nome: 'Fujiro Nakombi',  
  idade: 50,  
  email: 'fnakombi@gmail.com'  
};  
  
console.log(cliente.nome); // Fujiro Nakombi  
  
cliente.nome = 'Fujiro Jr. Nakombi';  
cliente.fone = '(11) 99999-1234';  
  
console.log(cliente.nome); // Fujiro Jr. Nakombi  
console.log(cliente.fone); // (11) 99999-1234
```

Tipo: Object

Exemplo: Lista de clientes

```
var clientes = [  
  {nome: 'Jose Java', cidade: 'Javalandia'},  
  {nome: 'Luiz Soares', cidade: 'Salvador'}  
];  
  
console.log(clientes[0].nome); // Jose Java  
console.log(clientes[1].nome); // Luiz Soares
```

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores**
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

Outline - Subseção

3 Operadores

- Operadores de Concatenação
- Operadores Aritméticos
- Operadores de Atribuição
- Operadores de Comparação
- Operadores de Incremento/Decremento
- Operadores Lógicos

Operador de concatenação

- O operador + pode concatenar ou somar;
- Se os dois valores da operação forem números, então o operador faz a soma;
- Caso contrário faz a concatenação.

```
var a;  
a = 1 + 1; // 2  
a = 1 + '1'; // 11  
a = '1' + '1'; // 11  
a = 1 + 'a'; // 1a  
a = 'a' + 'a'; // aa
```

Outline - Subseção

3 Operadores

- Operadores de Concatenação
- Operadores Aritméticos
- Operadores de Atribuição
- Operadores de Comparação
- Operadores de Incremento/Decremento
- Operadores Lógicos

Operadores aritméticos

Para as operações matemáticas básicas são utilizados os seguintes:

- + Soma (ou concatenação);
- − Subtração;
- * Multiplicação;
- / Divisão;
- % Módulo (Resto);

Operadores aritméticos

Exemplos

```
var a = 10, b = 5, c;
```

```
c = a + b; // 15
```

```
c = a - b; // 5
```

```
c = a * b; // 50
```

```
c = a / b; // 2
```

```
c = a % b; // 0
```

```
c = a % 3; // 1
```

Outline - Subseção

3 Operadores

- Operadores de Concatenação
- Operadores Aritméticos
- Operadores de Atribuição
- Operadores de Comparação
- Operadores de Incremento/Decremento
- Operadores Lógicos

Operadores de Atribuição

Operadores de atribuição são utilizados para atribuir valores às variáveis.

- = Atribuição;
- + = Soma e atribuição;
- = Subtração e atribuição;
- * = Multiplicação e atribuição;
- / = Divisão e atribuição;
- % = Módulo (Resto) e atribuição;

Operadores de Atribuição

Exemplos

```
var a = 10;
```

```
a += 5; // a = a + 5
```

```
a -= 5; // a = a - 5
```

```
a *= 5; // a = a * 5
```

```
a /= 5; // a = a / 5
```

```
a %= 5; // a = a % 5
```

Outline - Subseção

3 Operadores

- Operadores de Concatenação
- Operadores Aritméticos
- Operadores de Atribuição
- Operadores de Comparação
- Operadores de Incremento/Decremento
- Operadores Lógicos

Operadores de Comparação

Operadores de comparação comparam dois valores e retornam um valor booleano.

- `==` Igualdade: verdadeiro se os dois valores comparados forem iguais;
- `!=` Diferente: verdadeiro se os dois valores comparados forem diferentes;
- `>` Maior que: verdadeiro se o valor da esquerda for maior que o da direita;
- `<` Menor que: verdadeiro se o valor da esquerda for menor que o da direita;
- `>=` Maior que ou Igual: verdadeiro se o valor da esquerda for maior ou igual que o da direita;
- `<=` Menor que ou Igual: verdadeiro se o valor da esquerda for menor ou igual que o da direita;

Operadores de Comparação

Exemplos

```
var a = 10, b = 5, c;
```

```
c = (a == b); // false
```

```
c = (a != b); // true
```

```
c = (a > b); // true
```

```
c = (a < b); // false
```

```
c = (a >= b); // true
```

```
c = (a <= b); // false
```

Operadores de Comparação

- `===` Idêntico: verdadeiro se os dois valores forem iguais E do mesmo tipo;
- `!==` Não idêntico: verdadeiro se os dois valores forem diferentes OU de tipos diferentes;

Operadores de Comparação

Exemplos

```
1 === 1; // true
1 === '1'; // false
1 === 10; // false
1 === '10'; // false
```

```
1 !== 1; // false
1 === '1'; // true
1 === 10; // true
1 === '10'; // true
```

Outline - Subseção

3 Operadores

- Operadores de Concatenação
- Operadores Aritméticos
- Operadores de Atribuição
- Operadores de Comparação
- Operadores de Incremento/Decremento
- Operadores Lógicos

Operadores de Incremento/Decremento

Incrementam (+1) e decrementam (-1) o valor de variáveis.

`a++` Retorna a e então faz o incremento;

`a--` Retorna a e então faz o decremento;

`++a` Incrementa a e então retorna o novo valor;

`--a` Decrementa a e então retorna o novo valor.

Operadores de Incremento/Decremento

Exemplos

```
// a = a + 1
a++; ++a;

// a = a - 1
a--; --a;

// console.log(a); a = a + 1;
console.log(a++);

// a = a + 1; console.log(a);
console.log(++a);

// console.log(a); a = a - 1;
console.log(a--);

// a = a - 1; console.log(a);
console.log(--a);
```

Outline - Subseção

3 Operadores

- Operadores de Concatenação
- Operadores Aritméticos
- Operadores de Atribuição
- Operadores de Comparação
- Operadores de Incremento/Decremento
- Operadores Lógicos

Operadores Lógicos

Realizam operações lógicas (Tabela Verdade) e retornam um valor booleano.

`&&` E;

`||` OU;

`!` Negação;

Operadores Lógicos

Exemplos

```
var t = true,  
    f = false,  
    c;  
  
c = (t && f); // false  
c = (t || f); // true  
  
c = !t; // false  
c = !f; // true
```

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle**
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

Estruturas de controle

São comandos da linguagem que permitem desviar o fluxo do programa, dependendo de um teste.

Estrutura if

- O `if` permite a execução condicional de um bloco de instruções;
- A expressão será sempre avaliada como um *boolean*;
- Se a expressão for avaliada como `true`, o bloco de instruções será executado.

```
var a = 1;
var b = 10;

// Se A > B
if (a > b) {
    console.log("A > B");
}
```

Estrutura if..else

else

O else estende a instrução if para executar outras instruções caso a expressão do if retornar false.

```
var numero = 17;

if (numero >= 18) {
  console.log("Numero >= 18");
} else {
  console.log("Numero < 18");
}
```

Iteração

Iteração

1. Ato de iterar; repetição;
2. Iteração é o processo chamado na programação de repetição de uma ou mais ações.

Estrutura While

“Repetição de um trecho de código. Geralmente utilizado quando **não** se sabe de antemão a quantidade de iterações.”

- Loop mais simples do JS;
- Executa o bloco de instruções enquanto \$expr retornar true;

```
while (expr) {  
    // instrucoes  
}
```

Estrutura While

Exemplo de uso

```
var i = 2;
while (i < 10) {
  console.log(i);
  i+=2;
}
```

Estrutura For

Executam um bloco de código por um determinado número de vezes.
Geralmente utilizado quando sabe-se de antemão a quantidade de iterações.

- `expr1` é sempre executado no início do laço;
- antes de cada iteração, `expr2` é avaliado: se `true`, executa o bloco de instruções, senão a execução do laço termina;
- após cada iteração, `expr3` é executada;

```
for (expr1; expr2; expr3) {  
    // instrucoes  
}
```

Estrutura For

Exemplo de uso

```
for (i = 2; i < 10; i+=2) {  
    console.log(i);  
}
```


Estrutura For..in

Iteram sobre propriedades de um objeto/array.

```
var cliente = {nome:"John",  
sobrenome:"Doe", idade:25};  
  
for (var x in cliente) {  
  console.log(x + ':' + cliente[x]);  
}
```

Estrutura For..in

```
var lista = [  
    'Skype',  
    'Libre Office',  
    'Sublime Text'  
];  
  
for (var i in lista) {  
    console.log(lista[i]);  
}
```

Estrutura Switch

Útil quando se deseja fazer comparações de igualdade.

```
switch(expr) {  
  case coincidencia1:  
    // instrucoes 1  
  case coincidencia2:  
    // instrucoes 2  
    break;  
  default:  
    // instrucoes 3  
}
```

- O valor de **expr** é testado, na ordem, contra os valores das constantes especificadas nos comandos case;
- Quando uma **coincidência** for encontrada, a sequência de comando associada àquele case será executada até que o comando **break** ou o **fim do comando switch** seja alcançado;
- O comando default é executado se nenhuma *coincidência* for detectada;
- O default é opcional.

Estrutura Switch

Exemplo de uso

```
var n = 2;

switch(n) {
  case 2:
  case 4:
    console.log('par');
    break;
  case 1:
  case 3:
    console.log('impar');
    break;
  default:
    console.log('desconhecido');
}
```

```
var n = 10;

switch(n) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
    console.log('<=5');
    break;
  default:
    console.log('>5');
}
```

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções**
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

Funções

"Funções são formas de modularizar uma ou mais linhas de código de maneira que possam ser executadas em diferentes momentos do script/aplicação quando necessário."

"A função poderia ser definida como um conjunto de instruções que permitem processar as variáveis para obter um resultado."

"As funções são métodos de economizar tempo e trabalho para ações que irão se repetir."

Sintaxe de funções

- Uma função JS é definida pelo uso de `function`, seguida pelo seu nome, seguida por parenteses `()`.
- Os nomes de funções seguem as mesmas regras de variáveis;
- Os parenteses podem incluir uma lista de parâmetros separados por vírgula;

```
function escreve(a) {  
    console.log(a);  
}
```

Invocação de funções

A invocação consiste em colocar o nome da função seguido pelos parênteses. Isso faz com que o código dentro do corpo da nossa função seja executado.

```
function escreve(a) {  
  console.log(a);  
}  
  
escreve('Hello');  
escreve('World');
```


Instrução return

Quando a execução de uma função alcança uma instrução **return** duas ações são tomadas:

1. a execução da função é interrompida;
2. o valor indicado pelo **return** é retornado para o "chamador" da função.

```
function soma(a, b) {  
    return a + b;  
}  
  
// c recebe 3  
var c = soma(1, 2);
```

Escopo de variáveis

Existem dois 2 tipos de escopos:

- Local** São as variáveis definidas dentro de funções. Elas são acessíveis somente dentro das funções;
- Global** São as variáveis definidas fora de qualquer função. Elas podem ser acessados de qualquer local/momento.

Escopo de variáveis

```
var a;  
function iniciar() {  
  console.log(a); // undefined  
  console.log(b); // undefined  
  
  a = 10;  
  var b = 50;  
  
  console.log(a); // 10  
  console.log(b); // 50  
}  
  
console.log(a); // undefined  
console.log(b); // erro  
  
iniciar();  
  
console.log(a); // 10  
console.log(b); // erro
```

Funções em variáveis

```
var soma = soma(a, b) {  
  return a + b;  
}
```

```
// c recebe 3  
var c = soma(1, 2);
```

Funções anônimas como argumentos

```
function animacao(x, y, posFim) {  
  // logica de animacao  
  // mover (x, y);  
  
  // apos a finalizacao da animacao  
  // executar uma tarefa  
  posFim('Fim!');  
}  
  
animacao(0, 0, function(s) {  
  console.log(s);  
});
```

Funções como argumentos

```
function animacao(x, y, posFim) {  
  // logica de animacao  
  // mover (x, y);  
  
  // apos a finalizacao da animacao  
  // executar uma tarefa  
  posFim('Fim!');  
}  
  
function mensagem(s) {  
  console.log(s);  
}  
  
animacao(0, 0, mensagem);
```

Funções built-in

parseFloat (a)

Retorna o valor numérico de ponto flutuante da *string* a.

parseInt (a)

Retorna o valor inteiro da *string* a.

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript**
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

Outline - Subseção

6 Objetos built-in do Javascript

- String
- Number
- Array
- Math
- Date

Objeto String

- o objeto `String` permite trabalhar com um conjunto de caracteres;
- envolve o tipo primitivo *string* oferecendo métodos que auxiliam a manipulação;
- JS automaticamente converte faz a conversão entre tipo primitivo e objeto;

```
var s = new String("Texto");  
var s = "Texto";
```

Objeto String: Propriedades

`length` Comprimento da String;

Objeto String: Propriedades

```
var s = "Alisson Chiquitto";  
  
console.log(s.length); // 17
```

Objeto String: Métodos I

String.indexOf(searchValue[, fromIndex])

Retorna o índice da primeira ocorrência de searchValue.

String.replace(pattern, replacement)

Substitui pattern por replacement.

String.split(separator)

Divide a String em um *array*.

String.substr(start[, length])

Retorna uma porção de String a partir de start.

Objeto String: Métodos II

String.substring(indexA, indexB)

Retorna uma porção de String entre as posições indexA e indexB.

String.toLowerCase()

Retorna uma versão em caixa baixa de String.

String.toUpperCase()

Retorna uma versão em caixa alta de String.

Objeto String: Métodos

```
var s = new String("República do Brasil");

console.log(s.indexOf("a")); // 8
console.log(s.replace("Brasil", "Congo")); // República do Congo
console.log(s.split(" ")); // ["República", "do", "Brasil"]
console.log(s.substr(10)); // do Brasil
console.log(s.substring(10, 12)); // do
console.log(s.toLowerCase()); // república do brasil
console.log(s.toUpperCase()); // REPÚBLICA DO BRASIL
```

Outline - Subseção

6 Objetos built-in do Javascript

- String
- Number
- Array
- Math
- Date

Objeto Number

- o objeto `Number` permite trabalhar com valores numéricos;
- envolve o tipo primitivo *number* oferecendo métodos que auxiliam a manipulação;
- JS automaticamente converte faz a conversão entre tipo primitivo e objeto;

```
var n = new Number(10);  
var n = new Number(3.14);  
var n = 10;  
var n = 3.14;
```

Objeto Number

Number.toFixed(digits)

Retorna uma *string* com um número específico de dígitos decimais.

Number.toLocaleString()

Retorna uma *string* com a versão local do número.

Objeto Number: Métodos

```
var n = new Number(3.14);  
  
console.log(n.toFixed()); // 3  
  
// pt_BR = 3,14  
// en = 3.14  
console.log(n.toLocaleString());
```

Outline - Subseção

6 Objetos built-in do Javascript

- String
- Number
- **Array**
- Math
- Date

Objeto Array

- o objeto Array permite armazenar múltiplos valores em uma única variável;

```
var a = new Array(10, 20, 30, 40);
```

Objeto Array: Propriedades

`length` Comprimento do Array;

Métodos de Array I

Array.join(glue)

Junta os elementos em uma String.

Array.pop()

Remove o último elemento de Array e o retorna.

Array.push(...values)

Adiciona um ou mais valores no final do Array e retorna o novo comprimento do Array.

Métodos de Array II

Array.shift()

Remove o primeiro elemento de Array e o retorna.

Array.unshift(...values)

Adiciona um ou mais valores no início do Array e retorna o novo comprimento do Array.

Objeto Array: Métodos

```
var a = new Array(9,8,7,6);

console.log(a.join("-")); // 9-8-7-6
console.log(a.pop()); // [9,8,7] => 6
console.log(a.push(5, 4)); // [9,8,7,6,5,4] => 6
console.log(a.shift()); // [8,7,6] => 9
console.log(a.unshift(10)); // [10,9,8,7,6] => 5
```

Outline - Subseção

6 Objetos built-in do Javascript

- String
- Number
- Array
- **Math**
- Date

Objeto Math

- o objeto `Math` fornece propriedades e métodos para funções matemáticas;
- todos os métodos/propriedades de `Math` são estáticos.

Objeto Math: Propriedades

PI Valor de PI (aproximadamente 3,1415);

Objeto Math: Métodos I

Math.abs(number)

Valor absoluto de um número;

Math.ceil(number)

Arredonda frações para cima.

Math.floor(number)

Arredonda frações para baixo.

Math.max(...numbers)

Retorna o maior valor de numbers.

Objeto Math: Métodos II

Math.min(...numbers)

Retorna o menor valor de `numbers`.

Math.pow(base, exponent)

Retorna o calculo de base elevado a `exponent`.

Math.random()

Retorna um valor aleatório ($0 \leq \textit{number} < 1$);

Math.round(number)

Retorna o valor inteiro mais próximo a `number`.

Objeto Math: Métodos

```
console.log(Math.PI); // 3.14...

console.log(Math.abs(10)); // 10
console.log(Math.abs(-10)); // 10

console.log(Math.ceil(3.14)); // 4
console.log(Math.ceil(-3.14)); // -3

console.log(Math.floor(3.14)); // 3
console.log(Math.floor(-3.14)); // -4

console.log(Math.max(1,5,7)); // 7
console.log(Math.min(1,5,7)); // 1

console.log(Math.pow(2, 3)); // 8
console.log(Math.pow(2, -1)); // 0.5

console.log(Math.random(2, -1));

console.log(Math.round(1.4)); // 1
console.log(Math.round(1.5)); // 2
```

Outline - Subseção

6 Objetos built-in do Javascript

- String
- Number
- Array
- Math
- Date

Objeto Date

- o objeto Date permite a obtenção/manipulação de valores temporais.

```
// Data/Hora atual
var d = new Date( );

// timestamp_milliseconds
// Sun May 08 2016 17:22:01 GMT-0300 (BRT)
var d = new Date(1462738921000);

var d = new Date('2014-08-09');

// year,month,date[,hour,minute,second,millisecond ]
// Wed Jan 15 2014 00:00:00 GMT-0200 (BRST)
var d = new Date(2014,00,15);
```

Objeto Date: Métodos para pegar valores I

Date.getDate()

Retorna o número do dia no mês.

Date.getDay()

Retorna o número do dia na semana (0=Domingo).

Date.getFullYear()

Retorna o ano.

Date.getHours()

Retorna a hora.

Objeto Date: Métodos para pegar valores II

`Date.getMinutes()`

Retorna os minutos.

`Date.getMonth()`

Retorna o número do mês, iniciando em 0. (0=Janeiro, 11=Dezembro).

`Date.getSeconds()`

Retorna os segundos.

`Date.getTime()`

Retorna o *timestamp* milissegundos.

Objeto Date: Métodos para pegar valores

```
// year, month, date, hour, minute, second  
var d = new Date(2016, 04, 14, 13, 14, 15);  
  
console.log(d.getDate()); // 5  
console.log(d.getDay()); // 6  
console.log(d.getFullYear()); // 2016  
console.log(d.getHours()); // 13  
console.log(d.getMinutes()); // 14  
console.log(d.getMonth()); // 4  
console.log(d.getSeconds()); // 15  
console.log(d.getTime()); // 1463242455000
```

Objeto Date: Métodos para definir valores I

Date.setDate(day)

Define o número do dia no mês.

Date.setFullYear(year)

Define o ano.

Date.setHours(hours)

Define a hora.

Date.setMinutes(minutes)

Define os minutos.

Objeto Date: Métodos para definir valores II

Date.setMonth(month)

Define o número do mês, iniciando em 0. (0=Janeiro, 11=Dezembro).

Date.setSeconds(seconds)

Define os segundos.

Date.setTime(time)

Define o *timestamp* milisegundos.

Objeto Date: Métodos para definir valores

```
var d = new Date();  
  
d.setDate(15);  
d.setFullYear(1986);  
d.setHours(9);  
d.setMinutes(30);  
d.setMonth(4);  
d.setSeconds(0);  
  
// 15/05/1986 09:30:00  
console.log(d.toLocaleString());
```

Objeto Date: Métodos com valores localizados I

Date.toLocaleDateString()

Retorna uma *string* com a data.

Date.toLocaleTimeString()

Retorna uma *string* com a hora.

Date.toLocaleString()

Retorna uma *string* com a data/hora.

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM**
- 8 Eventos
- 9 HTML Api
- 10 ES6

HTML DOM

Cada página web reside dentro da janela do navegador que pode ser considerado um objeto.

Um objeto `Document` representa o documento HTML que é exibido na janela. O objeto `Document` possui várias propriedades que referem a outros objetos que permitem o acesso e modificação das suas propriedades.

HTML DOM

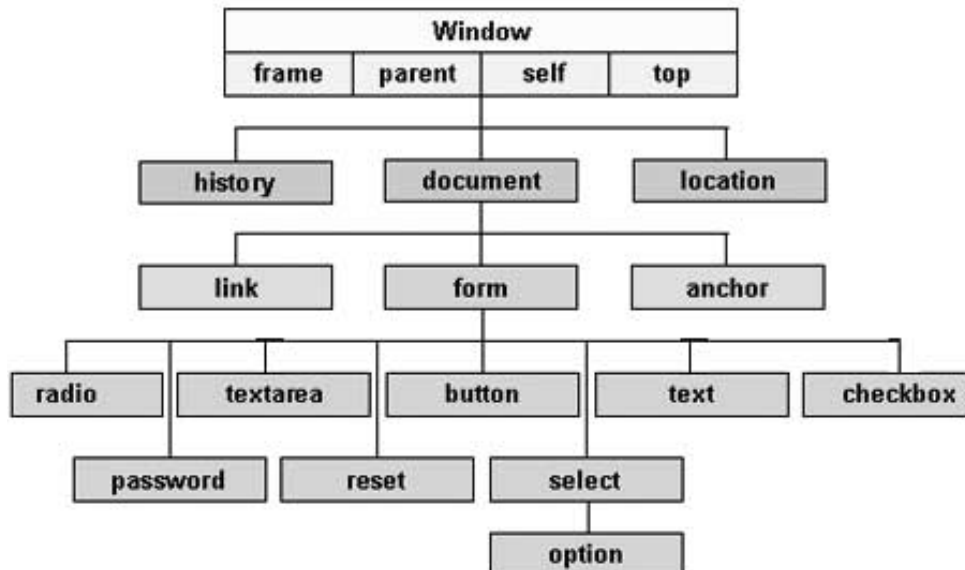
A maneira e a forma que os dados do HTML são acessados e modificados são chamados de DOM (*Document Object Model*).

Os objetos são organizados em uma hierarquia. A estrutura hierárquica se aplica à organização dos objetos em um documento web.

Objeto Window Acessado pela variável `window`. Topo da hierarquia;

Objeto Documento Acessado pela variável `document`. Representa o documento HTML carregado em `Window`.

HTML DOM



HTML DOM: Nodes

No HTML DOM, tudo é um *node* (nó):

- o *Document* em si é um *Document node*;
- os elementos HTML são *Element nodes*;
- os atributos HTML são *Attribute nodes*;

Outline - Subseção

7 HTML DOM

- Objeto Document
- Objeto Element
- Objeto Style

HTML DOM: document

- Quando um documento HTML é carregado no navegador, ele se torna um objeto Document.
- O objeto Document é o *node* do HTML que contém todos os outros *nodes*.
- Podemos acessar o objeto Document via:
 - ▶ propriedade `window.document`;
 - ▶ variável `document`;

HTML DOM: document I

Métodos/Propriedades

Document.addEventListener(event, handler)

Adiciona um manipulador de evento ao evento.

Element.appendChild(element)

Adiciona um elemento filho ao documento.

Document.getElementById(id)

Retorna o elemento que possui o valor especificado no id.

HTML DOM: document II

Métodos/Propriedades

Document.getElementsByClassName(className)

Retorna um `NodeList` contendo os elementos com o nome de classe especificado.

Document.getElementsByTagName(tagName)

Retorna um `NodeList` contendo os elementos com o nome de *tag* especificado.

Document.querySelector(selector)

Retorna o primeiro elemento que coincide com o Seletor CSS especificado no documento.

HTML DOM: document III

Métodos/Propriedades

Document.querySelectorAll(selector)

Retorna um `NodeList` contendo todos os elementos que coincide com o Seletor CSS especificado no documento.

Document.getElementById()

```
<ul id="lista">  
  <li>Item 1</li><li>Item 2</li>  
  <li>Item 3</li><li>Item 4</li>  
</ul>
```

```
var elemento = document.getElementById('lista');  
elemento.style.background='red';
```

Document.querySelector()

```
<ul id="lista">  
  <li>Item 1</li><li>Item 2</li>  
  <li>Item 3</li><li>Item 4</li>  
</ul>
```

```
var elemento = document.querySelector('#lista li');  
elemento.style.background='red';
```

Document.querySelectorAll()

```
<ul id="lista">
  <li>Item 1</li><li>Item 2</li>
  <li>Item 3</li><li>Item 4</li>
</ul>
```

```
var elementos = document.querySelectorAll('#lista li');

for (var i= 0; i < elementos.length; i++) {
  elementos[i].style.background='red';
}
```

Outline - Subseção

7 HTML DOM

- Objeto Document
- Objeto Element
- Objeto Style

HTML DOM: element

- O objeto `element` representa um elemento HTML;
- Objetos `element` podem ter *childs* (filhos);
- Um objeto `NodeList` representa uma lista de *nodes*;
- Elementos possuem *attributes nodes*.

HTML DOM: element I

Métodos/Propriedades

Element.addEventListener(event, handler)

Adiciona um manipulador de evento ao evento.

Element.appendChild(element)

Adiciona um elemento filho ao elemento.

Element.className

Define ou retorna os nomes de classes.

Element.getElementsByClassName(className)

Retorna um NodeList contendo os elementos com o nome de classe especificado.

HTML DOM: element II

Métodos/Propriedades

Element.innerHTML

Define ou retorna o conteúdo do elemento.

Element.getElementsByTagName(tagName)

Retorna um `NodeList` contendo os elementos com o nome de *tag* especificado.

Element.querySelector(selector)

Retorna o primeiro elemento que coincide com o Seletor CSS especificado no elemento.

HTML DOM: element III

Métodos/Propriedades

Element.querySelectorAll(selector)

Retorna um `NodeList` contendo todos os elementos que coincide com o Seletor CSS especificado no elemento.

Element.style

Define ou retorna o valor dos estilos do elemento.

Element.innerHTML

```
<h1 id="tit"></h1>
```

```
var e = document.getElementById('tit');  
e.innerHTML = 'Titulo da pagina';  
e.style.background = 'red';
```

Element.querySelector()

```
<h1>Titulo <small>Texto pequeno</small></h1>  
<h2>Titulo <small>Texto pequeno</small></h2>
```

```
var h1 = document.querySelector('h1');  
var h1small = h1.querySelector('small');  
h1small.innerHTML = 'nonono';
```

Outline - Subseção

7 HTML DOM

- Objeto Document
- Objeto Element
- Objeto Style

HTML DOM: style

- representa uma propriedade de estilo individual do elemento;
- pode ser acessado através de `Element.style`;
- pode manipular qualquer propriedade de estilo (CSS) do elemento;

HTML DOM: style

Nomes das propriedades

- os nomes das propriedades do objeto `style` são os mesmos das propriedades do CSS;
- nomes de propriedades CSS que contém hífen (-) são modificados de acordo com o padrão *lowerCamelCase* (remove-se o hífen, e a próxima letra é alterada para caixa alta).

Exemplos:

`border` \Rightarrow `border`

`background` \Rightarrow `background`

`border-top-color` \Rightarrow `borderTopColor`

`background-color` \Rightarrow `backgroundColor`

Element.style

```
<h1 id="tit">Titulo</h1>
```

```
// var h1 = document.querySelector('h1');  
// var h1 = document.querySelector('#tit');  
var h1 = document.getElementById('tit');
```

```
h1.style.backgroundColor = '#FF0000';  
h1.style.textAlign = 'center';  
h1.style.fontSize = '18px';  
h1.style.margin = '0';  
h1.style.marginBottom = '1em';  
h1.style.padding = '0';
```


Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos**
- 9 HTML Api
- 10 ES6

HTML Dom Events

Eventos do HTML DOM permitem o Javascript registrar diferentes manipuladores de eventos em elementos HTML.

Os eventos são disparados por ações do usuário, e então os manipuladores são invocados.

Cada manipulador invocado recebe um parâmetro Event.

Objeto Evento I

Métodos/Propriedades

Event.target

O elemento que invocou o evento.

Event.preventDefault()

Cancela o evento (se ele for cancelável). Significa que a ação padrão do evento não irá ocorrer.

Lista de eventos disponíveis

- W3schools HTML DOM Events
- Tutorials Point Javascript Events

window.onload

O evento `load` do objeto `window` é invocado quando todos os arquivos do documento foram carregados e o DOM esta pronto.

```
window.addEventListener('load', function(e) {  
    // Carregou os arquivos  
    console.log(e);  
});
```

```
function init(e) {  
    // Carregou os arquivos  
    console.log(e);  
}  
window.addEventListener('load', init);
```

Eventos de Mouse

```
document.getElementById('id').addEventListener('click', function(e) {  
    window.alert('click');  
});
```

Eventos de Mouse

```
var e = document.getElementById('id');

e.addEventListener('mousedown', function(e) {
  // usuario pressionou o botao do mouse
  console.log('mousedown');
});

e.addEventListener('mouseup', function(e) {
  // usuario soltou o botao do mouse
  console.log('mouseup');
});
```

Eventos de Teclado

```
document.addEventListener('keydown', function(e) {  
    // usuario esta mantendo uma tecla pressionada  
    console.log('keydown');  
});  
  
document.addEventListener('keypress', function(e) {  
    // usuario pressionou uma tecla  
    console.log('keypress');  
});  
  
document.addEventListener('keyup', function(e) {  
    // usuario soltou uma tecla  
    console.log('keyup');  
});
```


Eventos de Formulários

```
var f = document.getElementById('form');  
  
f.addEventListener('submit', function(e) {  
    // usuario submeteu um formulario  
    console.log('submit');  
});
```

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api**
- 10 ES6

APIs do HTML

O HTML possui vários recursos que são disponibilizados via API. Alguns deles são:

- File API;
- Geolocation API;
- Web Notification API;
- Canvas;
- Validação de formulários;
- Audio & Vídeo;
- History;

File API

- Ler arquivos em JavaScript usando as APIs do arquivo
- W3C File API

Geolocation API

- W3Schools Geolocation API
- Tutorials Point - HTML5 - Geolocation
- HTML5 Doctor - Plot a location on a Google Map using getPosition

Web Notification API

- Tableless: Web Notifications API
- Loopinfinito: Web Notifications API
- Site Point: An Introduction to the Web Notifications API
- HTML5Rocks: Using the Notifications API
- Web Notifications API Demo

Canvas API

- W3Schools: HTML5 Canvas
- HTML5 CANVAS EXAMPLE
- MDN: Canvas tutorial
- 21 Ridiculously Impressive HTML5 Canvas Experiments

Form Validation API

- The Art of Web: HTML: HTML5 Form Validation Examples
- SitePoint: HTML5 Form Validation
- HTML5Rocks: Constraint Validation: Native Client Side Validation for Web Forms
- MDN: Data form validation
- W3Schools: JavaScript Validation API

Outline - Seção

- 1 Introdução
- 2 Variáveis
- 3 Operadores
- 4 Estruturas de controle
- 5 Funções
- 6 Objetos built-in do Javascript
- 7 HTML DOM
- 8 Eventos
- 9 HTML Api
- 10 ES6

- ES6, ECMAScript 6 ou ES2015, é simplesmente a mais nova versão do JavaScript.
- Na verdade, o nome mais usado atualmente é ES2015. A ideia do comitê responsável (conhecido como TC39) pelas atualizações da linguagem é justamente fazer um *release* anual.

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Destruturamento
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Arrow Function

```
([param], [param]) => {  
  statements  
}
```

```
param => expression
```

- Uma expressão *arrow function* possui uma sintaxe mais curta quando comparada com expressões de função (function expressions);
- *Arrow functions* sempre são anônimas.

Arrow Function

Argumentos

```
let agora = function() { return new Date() }  
let agora = () => new Date()  
  
let contador = function(s) { return s.length }  
let contador = s => s.length  
let contador = (s) => s.length  
  
let somador = function(a, b) { return a + b }  
let somador = (a, b) => a + b
```

- Ausência de argumentos precisa ser indicada com ();
- Para apenas um argumento os parentêses não são requeridos;
- Para mais de um argumento os parentêses são requeridos;

Arrow Function

Instruções

```
let somador = function(a, b) { return a + b }  
let somador = (a, b) => a + b  
  
let subtrator1 = function(a, b) {  
  let c = a - b;  
  return c;  
}  
  
let subtrator2 = (a, b) => {  
  let c = a - b;  
  return c;  
}
```

- Uma única expressão, não exige chaves. Neste caso, a expressão é também o valor de retorno da função;
- Múltiplas declarações precisam ser envolvidas entre chaves, ;

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Desestruturamento
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Template Strings

- Nova maneira de criar *strings* e com novos recursos, como *strings multiline* e formatação por substituições de variáveis (Interpolação de Expressões).
- Para criar esse tipo de *string* devemos utilizar o caracter ' (apóstrofo craseado).
- Podemos ter quebras de linhas(multiline) sem a necessidade de concatenar.

Template Strings

```
var teste = `  
  texto multiline  
  texto  
  texto  
`;  
;
```

Também podemos fazer substituições utilizando `${}`. Sendo que entre `{}` podemos ter qualquer variável ou expressão que retorne algum valor.

```
var nome = "Chiquitto", dia = "hoje";  
'Oi ${nome}, será que ${dia} vai chover?'  
// Oi Chiquitto, será que hoje vai chover?
```

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- **Desestruturamento**
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Desestruturamento

Desestruturamento (*Destructuring Assignment*) ajuda a evitar a necessidade de variáveis temporárias quando lida-se com objetos e arrays.

O Desestruturamento torna possível extrair dados de *arrays* ou objetos em variáveis distintas.

Desestruturamento

```
var a, b, rest;  
[a, b] = [10, 20];  
console.log(a); // 10  
console.log(b); // 20  
  
[a, b, ...rest] = [10, 20, 30, 40, 50];  
console.log(a); // 10  
console.log(b); // 20  
console.log(rest); // [30, 40, 50]  
  
({a, b} = {a: 10, b: 20});  
console.log(a); // 10  
console.log(b); // 20
```

Desestruturamento

```
var a, b;  
  
function vetor() {  
    return [10,20];  
}  
  
[a, b] = [10, 20];  
console.log(a); // 10  
console.log(b); // 20  
  
[a, b] = vetor();  
console.log(a); // 10  
console.log(b); // 20
```

Desestruturamento

```
var a, b, rest;

function vetor() {
  return [10, 20, 30, 40, 50];
}

[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(a); // 10
console.log(b); // 20
console.log(rest); // [30, 40, 50]

[a, b] = vetor();
console.log(a); // 10
console.log(b); // 20
console.log(rest); // [30, 40, 50]
```

Desestruturamento

```
var nome, email;
var cliente = {
  nome: 'Alisson Chiquitto',
  email: 'chiquitto@gmail.com'
};
function funcao() {
  return {
    nome: 'Alisson Chiquitto',
    email: 'chiquitto@gmail.com'
  };
}

({nome, email} = cliente);
console.log(nome); // Alisson Chiquitto
console.log(email); // chiquitto@gmail.com

({nome, email} = funcao());
console.log(nome); // Alisson Chiquitto
console.log(email); // chiquitto@gmail.com
```

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Desestruturamento
- **Parâmetros Default de Funções**
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Parâmetros pré-definidos

Os parâmetros predefinidos de uma função permitem que parâmetros sejam inicializados com valores iniciais caso *undefined* ou nenhum valor seja passado.

```
function ola(nome = 'James Bond', apelido = 'Bond') {  
  console.log('Meu nome é ' + apelido + ', ' + nome);  
}  
  
ola();  
// Meu nome é Bond, James Bond  
  
let getPrecoFinal = (preco, imposto = 0.5) => preco + preco * imposto;  
getPrecoFinal(500); // 750
```

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Desestruturamento
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Operador Spread / Rest

O operador `...` é chamado de operador *spread* ou *rest*, dependendo de como e onde é usado.

Spread Operator

Quando usado com qualquer iterável, ele age como *"spread"* (espalhamento) em elementos individuais.

O operador faz o espalhamento de elementos de uma coleção iterável (como um *array* ou mesmo uma *string*) em elementos literais e parâmetros de função individuais.

```
var params = [ "hello", true, 7 ]
```

```
// [ 1, 2, "hello", true, 7 ]
```

```
var other = [ 1, 2, ...params ]
```

```
function valores(a, b, c) {
```

```
  console.log(a); // hello
```

```
  console.log(b); // true
```

```
  console.log(c); // 7
```

```
}
```

```
valores(...params);
```

Rest Operator

O outro uso comum de `...` é juntar uma série de valores em um *array*. Nesse caso, o operador é chamado de "*rest*".

A sintaxe de *Rest Parameter* permite representar um número indefinido de argumentos em um *array*.

```
function contador(...args) {  
  console.log(args.length);  
}  
  
contador('a'); // 1  
contador('a', 'b'); // 2  
contador('a', 'b', 'c'); // 3
```

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Desestruturamento
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Escopo por bloco

Para ter acesso a variáveis de escopo, precisamos utilizar ao invés de *var* temos que utilizar a palavra reservada *let* para variáveis e *const* para constantes.

```
if(true){  
  let variavel = "teste";  
  const CONSTANTE = "1111111";  
  //so fica disponivel dentro do escopo do bloco  
}  
  
//fora nao existe mais  
console.log(variavel);// ReferenceError: variavel is not defined  
console.log(CONSTANTE);// ReferenceError: CONSTANTE is not defined
```

Escopo por bloco

```
for(var i = 0; i <= 5; i++) {  
  console.log(i);  
};  
console.log(i); // 6  
  
for(let k = 0; k <= 5; k++) {  
  console.log(k);  
};  
  
console.log(k); // Uncaught ReferenceError: k is not defined
```


Escopo por bloco

```
// ERRADO
if (algumaCoisa) {
  let valor = true;
}
// Uncaught ReferenceError: algumaCoisa is not defined
console.log(valor);

// CERTO
let valor;
if (algumaCoisa) {
  valor = true;
}
console.log(valor); // true
```

Escopo por bloco

```
let teste = 1;
if (true) {
  let teste = 2; // OK pois está em outro escopo
  if(true){
    let teste = 3 // OK pois está em outro escopo
    console.log(teste); // 3
  }
  console.log(teste); // 2
}
console.log(teste); // 1
```

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Desestruturamento
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- **Orientação a Objetos**
- Promises

Orientação a Objetos

- Permite a criação de classes;

Orientação a Objetos

Classes

```
class Poligono {  
  constructor(id, x, y) {  
    this.id = id;  
    this.mover(x, y);  
  }  
  mover(x, y) {  
    this.x = x;  
    this.y = y;  
  }  
}
```

Orientação a Objetos

Herança

```
class Quadrado extends Poligono {
  constructor(id, x, y, base) {
    super(id, x, y);
    this.base = base;
  }
  calcularArea() {
    return this.base * this.base;
  }
}

class Triangulo extends Poligono {
  constructor(id, x, y, base, altura) {
    super(id, x, y);
    this.base = base;
    this.altura = altura;
  }
  calcularArea() {
    return this.base * this.altura / 2;
  }
}
```

Orientação a Objetos

Herança/Conversão para String

```
class Poligono {
  constructor(id, x, y) {
    this.id = id;
    this.mover(x, y);
  }
  toString() {
    return 'Poligono(${this.id})';
  }
}
class Quadrado extends Poligono {
  toString() {
    return 'Quadrado>' + super.toString();
  }
}
```

Orientação a Objetos

Instanciando e utilizando

```
let q = new Quadrado(3);
q.calcularArea(); // 9

let t = new Triangulo(3,4);
t.calcularArea(); // 6

q instanceof Quadrado; // true
q instanceof Triangulo; // false
q instanceof Poligono; // true

t instanceof Quadrado; // false
t instanceof Triangulo; // true
t instanceof Poligono; // true
```


Orientação a Objetos

Elementos estáticos

```
class Quadrado extends Poligono {  
  static quadradoPadrao() {  
    return new Quadrado('id', 1, 2, 10);  
  }  
}  
  
let q = Quadrado.quadradoPadrao();  
let area = q.calcularArea(); // 100
```

Orientação a Objetos

Get/Set

```
class Triangulo extends Poligono {
  constructor(id, x, y, base, altura) {
    super(id, x, y);
    this._base = base;
    this._altura = altura;
  }
  calcularArea() {
    return this._base * this._altura / 2;
  }
  set base(base) {
    this._base = base;
  }
  get base() {
    return this._base;
  }
  set altura(altura) {
    this._altura = altura;
  }
  get altura() {
    return this._altura;
  }
}
```

Outline - Subseção

10 ES6

- Arrow Functions
- Template Strings
- Desestruturamento
- Parâmetros Default de Funções
- Operador Spread / Rest
- Escopo por bloco
- Orientação a Objetos
- Promises

Promises

Uma *promise* é um objeto que espera (**pendente**) por uma operação assíncrona ser completa, e quando se completa (**definida**), a *promise* se torna **atendida** ou **rejeitada**.

Uma promessa pode ser:

atendida a ação relacionada à promessa teve sucesso;

rejeitada a ação relacionada à promessa falhou;

pendente a ação ainda não foi atendida nem rejeitada;

definida a ação foi atendida ou rejeitada local/momento;

Promises

A forma padrão de se criar uma *Promise* é usando o construtor `new Promise()` que aceita um manipulador (*handler*) que é dado 2 funções como parâmetros. O primeiro manipulador (geralmente nomeado `resolve`) é uma função pra chamar com o futuro valor quando ele estiver pronto; e o segundo manipulador (geralmente chamado de `reject`) é uma função pra chamar se a *Promise* não conseguir resolver o valor futuro, e for rejeitada.

```
var p = new Promise(function(resolve, reject) {  
  if (/* condition */) {  
    resolve(/* value */); // resolvida com sucesso  
  } else {  
    reject(/* reason */); // erro, rejeitada  
  }  
});
```

Promises

Toda *promise* tem um método chamado `then()`, que recebe um par de *callbacks*. O primeiro *callback* é chamado se a *promise* for resolvida, enquanto o segundo é chamado se for rejeitada.

```
p.then((val) => console.log("Promise Resolvida", val),  
      (err) => console.log("Promise Rejeitada", err));
```

Promises

O valor de retorno de `then()` vai ser passado como valor pro próximo `then()`.

```
var hello = new Promise(function(resolve, reject) {  
  resolve("Hello");  
});  
  
hello.then((str) => `${str} World`)  
  .then((str) => `${str}!`)  
  .then((str) => console.log(str)) // Hello World!
```

Promises





Quando retornar uma *promise*, o valor resolvido da *promise* vai ser passado para o próximo *callback* para encadeá-los efetivamente. Essa é uma forma simples de evitar o "*callback hell*"

```
var p = new Promise(function(resolve, reject) {
  resolve(1);
});






var EventualmenteAdicional = (val) => {
  return new Promise(function(resolve, reject){
    resolve(val + 1);
  });
}

p.then(EventualmenteAdicional)
  .then(EventualmenteAdicional)
  .then((val) => console.log(val)) // 3
```


Referências I

-  Silva, Maurício Samy
Javascript - Guia do Programador
Novatec, 2010.
-  Paulo Silveira, Adriano Almeida
Lógica de Programação: Crie seus primeiros programas usando Javascript e HTML
Casa do Código, 2014.
-  Eric T. Freeman, Elisabeth Robson
Head First JavaScript Programming
O'Reilly Media, 2014.
-  Douglas Crockford
Javascript The Good Parts
O'Reilly Media, 2008.

Referências II

-  ECMAScript Programming Language
<http://www.ecmascript.org/>
-  Tableless: Um guia para iniciantes na área de web.
<http://tableless.github.io/iniciantes/manual/js/inserindo-js.html>
-  MDN: Introdução Javascript
<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Introduction>
-  W3Schools: JavaScript Tutorial
<http://www.w3schools.com/js/>
-  Tutorials Point: JavaScript Tutorial
http://www.tutorialspoint.com/javascript/javascript_built_in_functions.htm

Referências III

-  The HTML 5 JavaScript API Index
<http://html5index.org/>
-  <https://developer.mozilla.org/en-US/docs/Web/API>
<https://developer.mozilla.org/en-US/docs/Web/API>
-  ECMAScript 6 Features
<https://github.com/lukehoban/es6features>
-  ES6 Arrow functions Sample
<https://googlechrome.github.io/samples/arrows-es6/>
-  Chrome Platform Status
<https://www.chromestatus.com/samples>

Referências IV



Arrow Functions

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Arrow_functions



Novos recursos do JavaScript com o ES6 (ES2015)

<https://victorvhpg.github.io/2016/11/12/es6.html>



ES6 para humanos

<https://github.com/alexmoreno/ES6-para-humanos>



ECMAScript 6 — New Features: Overview & Comparison

<http://es6-features.org>



ECMAScript® 2017 Language Specification

<https://tc39.github.io/ecma262/>