Sobre Este Curso

Público Alvo

Programadores que têm como objetivo atuar como Desenvolvedor Front-End e criar interfaces para produtos digitais.

Pré-Requisitos

Conhecimentos de Lógica de Programação, HTML e CSS.

Desenvolvido por DJF Treinamentos e Consultoria e licenciado para
Apex Treinamentos de Alta Performance

Índice

	Anotações

Anotações	

Anotações

Copyright

As informações contidas neste material se referem ao curso de **Javascript** e estão sujeitas as alterações sem comunicação prévia, não representando um compromisso por parte do autor em atualização automática de futuras versões.

A **Apex** não será responsável por quaisquer erros ou por danos acidentais ou consequenciais relacionados com o fornecimento, desempenho, ou uso desta apostila ou os exemplos contidos aqui.

Os exemplos de empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares e eventos aqui representados são fictícios. Nenhuma associação a empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares ou eventos reais é intencional ou deve ser inferida.

A reprodução, adaptação, ou tradução deste manual mesmo que parcial, para qualquer finalidade é proibida sem autorização prévia por escrito da **Apex**, exceto as permitidas sob as leis de direito autoral.

Equipe

Conteúdos

- Felipe de Oliveira

Diagramação

- Fernanda Pereira

Revisão

- Fernanda Pereira

Histórico das Edições

Edição	Idioma	Edição
1ª	Português	Setembro de 2017

Anotações	

Capítulo 01 – Introdução

Objetivos:

Neste capítulo você irá aprender:

- O que é Javascript
- Javascript e a especificação ECMAScript
- Como incorporar Javascript nas páginas html

	Anotações

O que é Javascript

Javascript é uma linguagem interpretada baseada em Objetos e funções com tipagem dinâmica. JavaScript foi originalmente desenvolvido por Brendan Eich, da Netscape sob o nome de *Mocha*, posteriormente teve seu nome mudado para *LiveScript* e por fim *JavaScript*.

LiveScript foi o nome oficial da linguagem quando foi lançada pela primeira vez na versão beta do navegador Netscape 2.0 em setembro de 1995, mas teve seu nome mudado em um anúncio conjunto com a Sun Microsystems em dezembro de 1995 quando foi implementado no navegador Netscape versão 2.0B3. (https://pt.wikipedia.org/wiki/JavaScript)

Javascript e a especificação ECMAScript

O JavaScript é padronizado pela Ecma International — a associação Europeia para a padronização de sistemas de comunicação e informação (antigamente ECMA era um acrônimo para European Computer Manufacturers Association) para entregar uma linguagem de programação padronizada, internacional baseada em Javascript.

Esta versão padronizada de Javascript, chamada ECMAScript, comporta-se da mesma forma em todas as aplicações que suportam o padrão. As empresas podem usar a linguagem de padrão aberto para desenvolver a sua implementação de Javascript. O padrão ECMAScript é documentado na especificação ECMA-262.

Como incorporar Javascript nas páginas html

Em Podemos incorporar javascript a uma página html basicamente de duas formas:

- Criar o Script diretamente na página html.

Exemplo:

```
1  <!doctype html>
2  <html>
3  <head></head>
4  <body>
5  <script>
6  document.write("<h1>Bem Vindo ao Javascript</h1>");
7  </script>
8  <body>
9  </html>
```

Anotações	

- Criar um arquivo de script separado da página e referenciá-lo posteriormente na página em que desejamos utilizá-lo. Primeiro precisamos criar um arquivo com a extensão .js.

Exemplo:

```
1  //app.js
2  document.write("<h1>Bem Vindo ao Javascript</h1>");
```

Em seguida importamos o arquivo javascript informando sua localização na propriedade src (source) da tag javascript.

```
1  <!doctype html>
2  <html>
3  <head></head>
4  <body>
5  <script src="app.js"></script>
6  <body>
7  </html>
```

Anotações

Capítulo 02 – Tipos de Dados

Objetivos:

Neste capítulo você irá aprender:

- Tipos de Dados
- Tipos de Dados Primitivos
- Tipos de Dados Objeto

Anotações	

Tipos de Dados

Os tipos de dados em Javascript são divididos basicamente em 2 grupos denominados Tipos Primitivos e Tipos Objeto.

Tipos de Dados Primitivos

São tipos imutáveis, ou seja, que não mudam. São considerados tipos primitivos as estruturas de dados que não se enquadram como Objetos ou Função.

Em Javascript temos 6 tipos de dados primitivos:

Boolean

Representa um valor lógico verdadeiro(true) ou falso(false).

Exemplo:

```
var continua = true;
```

Null

Representa a atribuição de um valor nulo.

Exemplo:

```
var valorInicial= null;
```

Undefinide

Representa um valor não definido.

Exemplo:

```
var totalDePontos= undefined;
```

Number

Representa um valor numérico inteiro (int) ou real(float). Podem ser atribuídos valores reais entre $-(25^3 -1)$ and $25^3 -1)$.

Exemplo:

```
var pi= 3.1416;

var idade = 18;
```

Anotações

String

Representa um valor do tipo texto.

Exemplo:

```
var nomeCurso = 'Academia Frontend';
```

Symbol

Tipo de dado utilizado para representar propriedades de um objeto. Este tipo de dado foi incluído na especificação ES6 (EcmaScript 2015).

Exemplo:

```
var Obj= {};

Obj[Symbol('nomePropriedade')] = valor;
```

Tipos de Dados Objeto

Objetos são estruturas de dados que possuem propriedades e comportamentos. Essas estruturas são utilizadas para realizarmos o mapeamento de objetos da vida real para o mundo computacional.

Em um capítulo posterior entraremos em mais detalhes quanto a criação e utilização de objetos.

Operador Typeof

Para identificarmos o tipo de dado de um determinado valor podemos utilizar o operador typeof.

Exemplo:

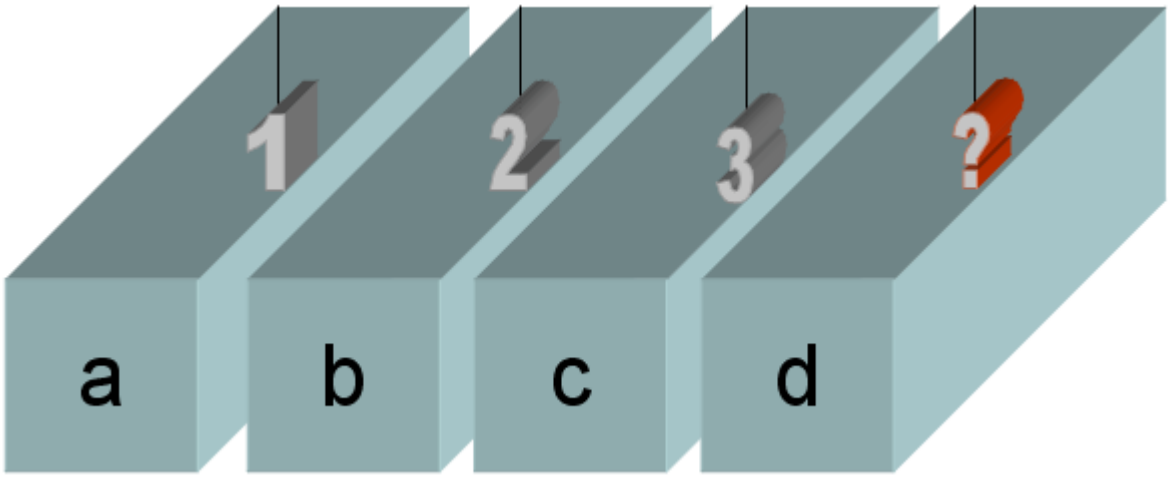
```
typeof 'teste' // 'string'

typeof  25 // 'number'

typeof  true // 'boolean'
```

Anotações	

Capítulo 03 – Declaração de Variáveis



Objetivos:

Neste capítulo você irá aprender:

- O que são variáveis
- Como o JavaScript trabalha com a tipo da variável
- Declaração de variáveis
- Regras de nomenclatura de um identificador
- Escopo de variáveis
- Conversão de dados

Anotações

O que são variáveis

Variáveis são identificadores utilizados para manipulação de endereços de memória utilizados para o armazenamento de dados de um sistema computacional.

Como o Javascript trabalha com a tipo da variável

Javascript realiza uma inferência dinâmica em tempo de execução para verificar o tipo de dado de uma variável. Isto significa que no momento da atribuição ele infere o tipo conforme o valor atribuído para a variável.

Declaração de variáveis

Em Javascript podemos declarar uma variável utilizando os símbolos **var**, **let** e **const** mais o nome do identificador.

Símbolo var

O símbolo **var** foi utilizado desde o início da linguagem para a definição de variáveis. Variáveis declaradas com o símbolo **var** possuem seu escopo definido dentro do contexto de execução e sua sintaxe é definida conforme exemplo abaixo:

Exemplo:

```
var nomeCurso=”Academia Frontend”
```

Símbolo let

O símbolo **let** foi introduzido na linguagem na especificação ES6 (EcmaScript 2015). Variáveis declaradas com o símbolo **let** possuem um escopo de bloco e devem ser preferencialmente utilizadas durante a definição de variáveis.

Exemplo:

```
let nomeCurso=”Academia Frontend”
```

Símbolo const

O símbolo **const** foi introduzido na linguagem na especificação ES6 (EcmaScript 2015). Variáveis declaradas com o símbolo **const** não podem ser alteradas e devem declarar o seu valor no momento da definição da variável.

Exemplo:

```
cons PI=3.1416;
```

Anotações	

Regras de nomenclatura de um identificador

Uma variável deve começar com uma letra, sublinhado(_) ou cifrão($), caracteres subsequentes podem ser letras de “a” até “z” ,”A” até “Z” ou dígitos(0-9). Javascript é Case Sensitive , ou seja, diferencia entre letras minúsculas e maiúsculas.

Escopo de variáveis

O escopo de uma variável indica em qual nível uma determinada variável é acessível.

Escopo de bloco

Variáveis declaradas dentro de um bloco de código específico, serão reconhecidas apenas dentro desse bloco. Este tipo de escopo é utilizado para variáveis definidas com símbolo *let*.

Escopo de contexto de execução

Variáveis declaradas dentro de uma função serão acessíveis em qualquer nível da função. Este tipo de escopo é utilizado para variáveis definidas com símbolo *var.*

Escopo global

Variáveis declaradas fora de blocos ou funções possuem o escopo global e podem ser acessadas em qualquer nível do código. Variáveis declaradas sem a utilização de símbolo (var,let ou const) são elevadas ao escopo global.

Conversão de Dados

Muitas vezes precisamos converter um tipo de dado em outro. Por exemplo, converter um valor do tipo texto em um inteiro ou um valor real, ou ainda converter um valor real em um valor do tipo texto. Para isso podemos utilizar as funções abaixo:

parseInt(String, radix)

Onde:
String é o texto a ser convertido;
radix é a representação numérica a ser utilizada(2-binária, 8- octal, 10-decimal)

Anotações

Exemplo:

var dez = parseInt(“10”,10); // retorna 10

var dez = parseInt(“10”,8); // retorna 8

parseFloat(String)

Onde:
String é o texto a ser convertido;

Exemplo:

var pi = parseFloat(“3.14”); // retorna 3.14

String(Object)

Onde:
Object é um objeto javascript

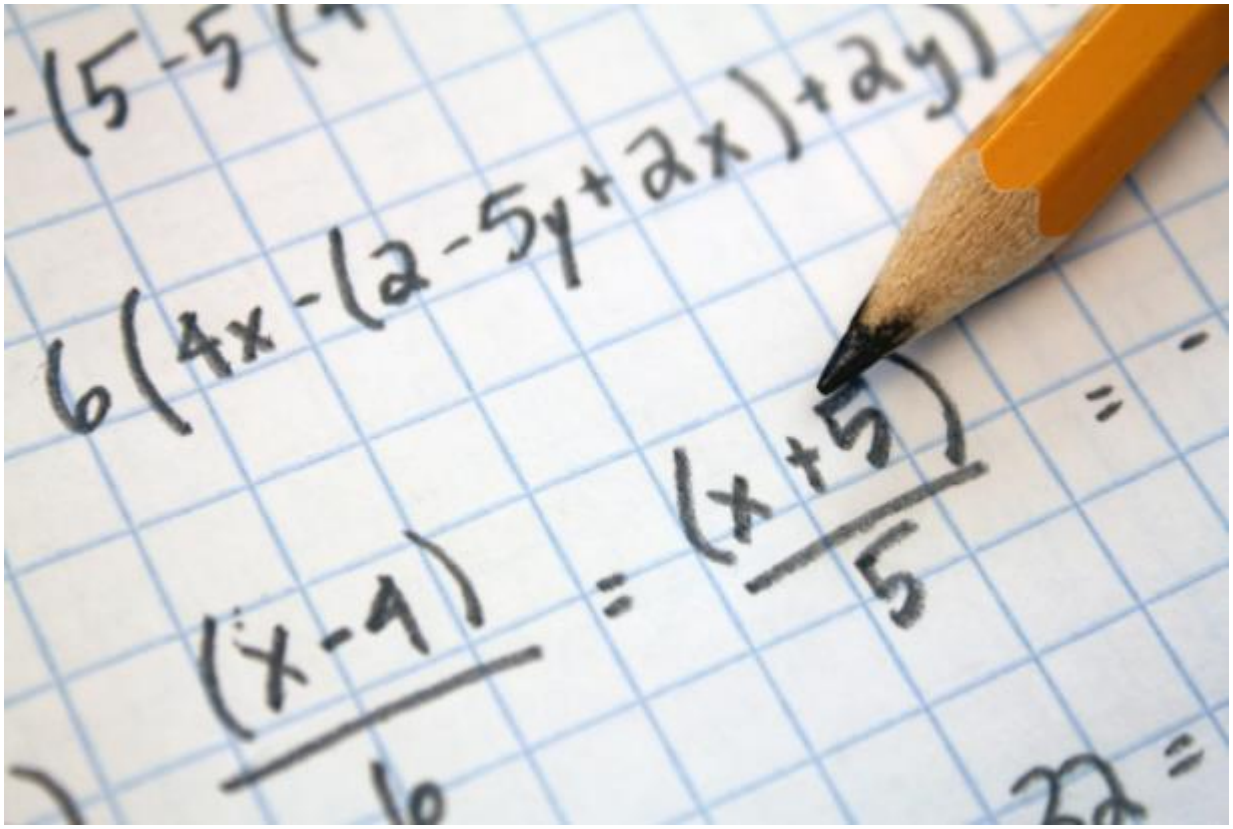
var teste = String(2.14); // retorna “2.14”

var teste = String(10); // retorna “10”

var teste = String({});// “[object Object]”

Anotações	

Capítulo 04 – Matemática em Javascript



Objetivos:

Neste capítulo você irá aprender:

- Operadores Matemáticos
- Precedência entre operadores
- Objeto Math
- Operadores Compostos

	Anotações

Operadores Matemáticos

Os operadores matemáticos em Javascript são:

+ Adição

- Subtração

* Multiplicação

/ Divisão

% Resto da divisão

++ Incremento

-- Decremento

Precedência entre operadores

A precedência de operadores determina a ordem em que os operadores são processados. Operadores com maior precedência são processados primeiro.

Em Javascript os operadores matemáticos seguem a seguinte precedência, da esquerda para a direita:

* / % + -

Objeto Math

Para algumas operações podemos utilizar algumas das funções do objeto Math.

Math.PI; // retorna o valor de Py.

Math.pow(5,2); //retorna o valor de 5 elevado na 2ª potência

Math.random(); retorna um número aleatório entre 0 e 1

Math.sqrt(16); // retorna o valor da raiz quadrada de 16

O operador de adição(+) também é utilizado para a concatenação de Strings.

Anotações	

Exemplo:

var primeiroNome = “João”;

var sobrenome = “Brasil”;

var nomeCompleto = primeiroNome + “ ”+sobrenome;

Operadores compostos

x += y; // equivale a x = x +y

x -= y;// equivale a x = x - y

x *= y;// equivale a x = x * y

x /= y;// equivale a x = x / y

x &= y;// equivale a x = x & y

x |= y;// equivale a x = x | y

Anotações

Capítulo 05 – Comparando valores

Objetivos:

Neste capítulo você irá aprender:

- Operadores de comparação
- Operadores lógicos

Anotações	

Operadores de Comparação

Os operadores de comparação utilizados em Javascript são:

< Menor que

<= Menor ou igual que

> Maior

>= maior ou igual a

!= diferente de

== igual a

Toda a comparação em Javascript retorna um valor boolean true (verdadeiro) ou false (false).

Exemplo:

```
2 < 10; // resulta em um valor true

219 < 10; // resulta em um valor false

10 > 2; // resulta em um valor true

5 != 10; // resulta em um valor true

5 == 5; // resulta em um valor true
```

Devemos ter um cuidado especial quando for realizar a comparação entre valores do tipo String e Number, pois este tipo de comparação também pode ser realizado.

```
2 == "2"; //resulta em valor true

10 > "100"; resulta em um valor false;
```

Os operadores diferente de (!=) e igual a (==) possuem uma outra versão que além de comparar os valores também fazem a comparação de seu tipos.

Anotações

!== Estritamente diferente de

=== Estritamente igual ª

Exemplo:

2 === 2; //resulta um valor true

2 === “2”;// resulta em um valor false

Operadores Lógicos

Os operadores lógicos servem para realizar a comparação entre duas expressões que retornam um valor boolean true ou false. O resultado desta comparação sempre será um valor lógico true ou false.

&& Operador E

Realiza a avaliação de duas proposições e retorna verdadeiro apenas se as duas proposições forem verdadeiras.

Exemplo:

var x = true;

var y = true;

var z = false;

x && y; // resulta em um valor true;

x && z; // resulta em um valor false;

|| Operador OU

Realiza a avaliação de duas proposições e retorna verdadeiro se uma das proposições forem verdadeiras.

Exemplo:

var x = true;

var y = true;

var z = false;

Anotações	

x || y; // resulta em um valor true;

x || z; // resulta em um valor true;

z || z; // resulta em um valor false;

! Operador NAO

Inverte o valor da proposição.

Exemplo:

var x = true;

var z = false;

!x; // resulta em um valor false;

!z; // resulta em um valor true;

^ Operador OU Exclusivo

Realiza a avaliação de duas proposições e retorna verdadeiro se apenas uma das duas proposições forem verdadeiras.

Exemplo:

var x = true;

var y = true;

var z = false;

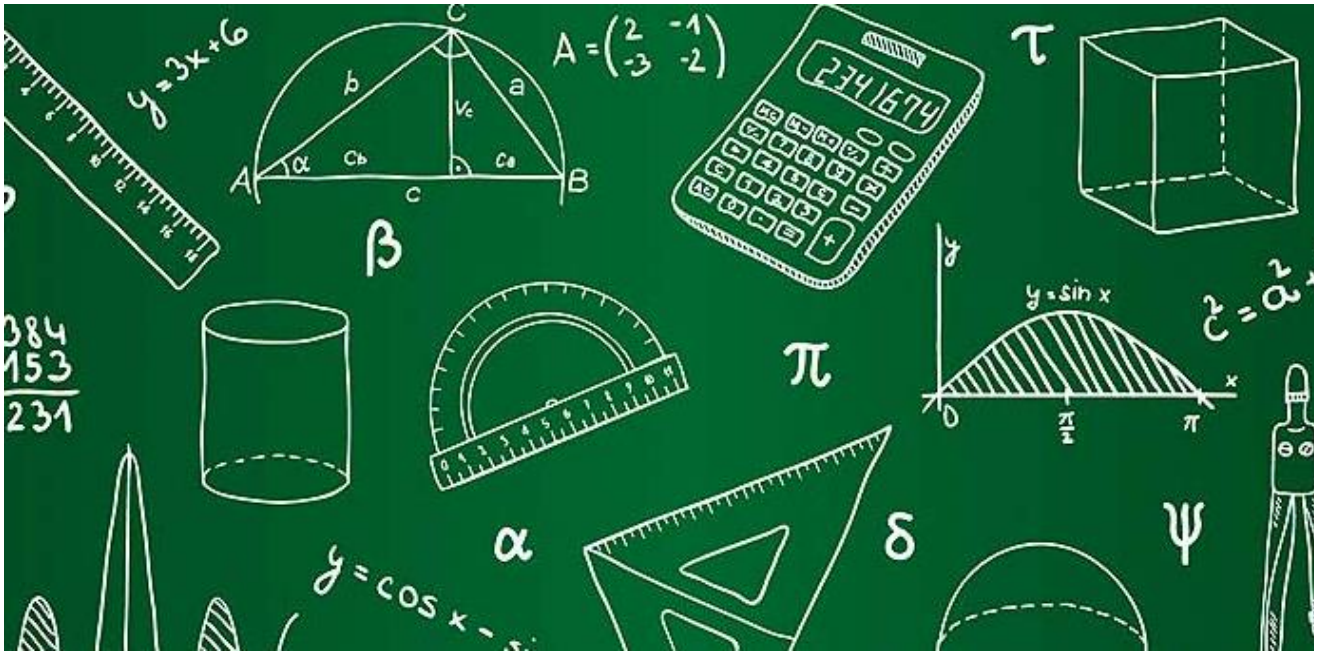
x ^ y; // resulta em um valor false;

x ^ z; // resulta em um valor true;

z ^ z; // resulta em um valor false;

	Anotações

Capítulo 06 – Funções



Objetivos:

Neste capítulo você irá aprender:

- O que são funções
- Declaração de funções
- Funções construtoras
- Funções anônimas

Anotações	

O que são Funções

Em Javascript funções são objetos de primeira classe, ou seja, eles são objetos e podem ser manipulados e repassados como qualquer outro objeto. Especificamente, eles são objetos de função.

Toda a função retorna algum valor. Caso a função possua a instrução return, a função retornará o valor informado por esta instrução. Caso a função seja uma função construtora, ela retornará uma nova instância do objeto. Caso seja uma função simples, retornará o valor undefined.

Declaração de Funções

Para declararmos uma função devemos utilizar a seguinte sintaxe:

```
function nome_da_funcao(lista_de_parametros){

        //corpo da função

}
```

Onde:

function → palavra reservada que indica que queremos criar uma função.

nome_da_funcao → identificador da função.

lista_de_parametros → lista de parâmetros que a função espera receber.

Exemplo:

```
function soma(a,b){

   return a+b;

}
```

Anotações

Funções Construtoras

Funções construtoras são funções utilizadas para criarmos novos objetos. Estas funções são definidas com a primeira letra maiúscula e são precedidas do operador **new**.

Exemplo:

```
function Pessoa(nome,sobrenome){
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.nomeCompleto = nome + ' '+sobrenome;
}

let joao = new Pessoa('João', 'Brasil');
console.log(joao.nomeCompleto)  //retorna 'João Brasil'
```

Funções Anônimas

Funções anônimas são funções criadas sem a definição de nome, normalmente utilizada como callback de outras funções.

Exemplo:

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

Anotações	

Capítulo 07 – Estruturas de Controle

Objetivos:

Neste capítulo você irá aprender:

- Operadores if/else
- Operador ternário
- Operador switch
- Operador while
- Operador for
- Operador for in
- Operador for of

	Anotações

Operador if/else

Quando queremos realizar algum desvio do fluxo conforme alguma condição podemos utilizar o comando if. Este comando recebe como parâmetro qualquer expressão que retorne true ou false. Caso o valor seja true ele será executado.

```
var desconto = true;

if(desconto){

        //código de desconto aqui

}
```

Podemos adicionar a instrução else que será executa caso o resultado da comparação do if seja falso.

```
var desconto = false;

if(desconto){

        //código de desconto aqui

} else{

        // código sem desconto aqui

}
```

Caso o if receba uma expressão que retorne um valor numérico, null ou undefined ele irá inferir os seguintes resultados;

0 →false

qualquer valor numérico != 0 → true

null → false

undefined → false;

Anotações	

Operador Ternário

O operador ternário é muito utilizado quando queremos atribuir um determinado valor a uma variável caso uma condição seja verdadeira ou um valor diferente caso a condição seja falsa.

var variavel = condição ? valor_caso_positivo : valor_caso_falso;

Ex:

var idade = 17;

var mensagem = idade >=18 ? “Maior” : “Menor”;

Operador switch

O operador **switch** avalia uma expressão combinando o valor da expressão para uma cláusula **case**, executando as instruções associadas ao **case**.

Sintaxe:

```
switch(expressão){
    case valor1:
        //instruções
        break;
    case valor2:
        //instruções
        break;
    default:
        //instruções;

}
```

Anotações

Exemplo:

```
let mesAtual = 'setembro'


switch(mesAtual){
    case 'janeiro':
        console.log('Escolhido o mês de Janeiro')
        break;
    case 'fevereiro':
        console.log('Escolhido o mês de Fevereiro')
        break;
    case 'março':
        console.log('Escolhido o mês de Março')
        break;
    case 'abril':
        console.log('Escolhido o mês de Abril')
        break;
    case 'maio':
        console.log('Escolhido o mês de Maio')
        break;
    case 'junho':
        console.log('Escolhido o mês de Junho')
        break;
    case 'julho':
        console.log('Escolhido o mês de Julho')
```

Anotações

```
        break;
    case 'agosto':
        console.log('Escolhido o mês de Agosto')
        break;
    case 'setembro':
        console.log('Escolhido o mês de Setembro')
        break;
    case 'outubro':
        console.log('Escolhido o mês de Outubro')
        break;
    case 'novembro':
        console.log('Escolhido o mês de Novembro')
        break;
    case 'dezembro':
        console.log('Escolhido o mês de Dezembro')
        break;
}
```

Operador while

O operador while é utilizado quando queremos repetir algum trecho de código enquanto uma determinada condição seja verdadeira. Ou seja, este operador recebe um valor lógico, avalia esse valor e executa um bloco de código caso o valor lógico seja verdadeiro.

Sintaxe:

```
while(expressão){
    //bloco de código
```

	Anotações

}

Exemplo:

```
var cont = 0;

while(cont < 10){

        console.log(cont);

        cont++;

}
```

Operador for

A estrutura de controle for é utilizada quando queremos repetir algum trecho de código em um número determinado de vezes.

Sua sintaxe é :

```
for(bloco de inicialização; bloco de validação; bloco de finalização ){      //corpo  }
```

Exemplo:

```
for(var count = 0; count< 1000; count++ ){

        Console.log(count);

}
```

Operador for in

Em Javascript utilizamos a estrutura for.. in quando queremos percorrer pelas propriedades **de um objeto.**

Sua sintaxe é:

```
for(variável in Objeto){//corpo do for}
```

Anotações	

Exemplo:

```
var contato = {nome:'Nome do Contato', email:'contato@email', telefone:'33221100'};

for (var prop in contato) {

  console.log("obj." + prop + " = " + contato[prop]);

}


// Saída:

// "obj.nome = Nome do Contato"

// "obj.email = contato@email"

// "obj.telefone = 33221100"
```

Operador for of

O operador for .. of foi adicionado no ES6 e tem por finalidade percorrer os valores de um objeto iterável (String,Array,Map,Set,etc.)

Exemplo:

```
let frutas = ['Maçã', 'Banana', 'Abacaxi'];

for(let fruta of frutas){

        console.log(fruta)

}
```

Resultado:

```
//Maça

//Banana

//Abacaxi
```

	Anotações

Capítulo 08 – Manipulação de Texto

Objetivos:

Neste capítulo você irá aprender:

- Forma literal
- Caracteres especiais
- Função length
- Função concat
- Função charAt
- Função indexOf
- Função trim
- Função substring
- Função split
- Função startWith
- Função endWith
- Função toLowerCase
- Função toUpperCase
- Função replace
- Template string

Anotações	

Forma literal

Para declarar um texto em Javascript de forma literal podemos utilizar aspas simples ou aspas duplas. No exemplo abaixo as tuas declarações estão corretas.

Exemplo:

```
let nome = 'João'
let email = "joao@teste.com"
```

Caracteres especiais

Caracteres especiais são os caracteres que não podemos escrever diretamente em um texto, mas ao invés disso devemos utilizar a notação de escape:

\0	o caractere NULL
\'	aspas simples
\"	aspas duplas
\\	barra invertida
\n	nova linha
\r	carriage return
\v	tab vertical
\t	tab
\b	backspace
\f	form feed

Anotações

Propriedade length

Retorna a quantidade de caracteres em um texto.

Exemplo:

```
let escola = 'Apex ensino';
console.log(escola.length)  // retorna 11
```

Função concat

Retorna um novo texto com o resultado da união do texto base com o texto passado como parâmetro da função.

Exemplo:

```
let nome = 'João'
let sobrenome = " da Silva"
let nomeCompleto = nome.concat(sobrenome)
console.log(nomeCompleto)  //escreve 'João da Silva'.
```

Função charAt

Retorna o caractere da posição informada.

Exemplo:

```
let escola = ‘APEX’;
console.log(escola.charAt(0)) //retorna a letra A (primeira letra do texto)
```

Anotações	

Função indexOf

Retorna o índice em que se encontra um determinado caracter. Caso o caracter informado não exista retorna o valor -1;

Exemplo:

```
let escola = 'Apex';

console.log(escola.charAt(3))  // retorna 'x'
```

Função trim

Retorna uma nova String sem os caracteres em branco do início e do fim do texto.

Exemplo:

```
let nome = '     Maria   ';

console.log(nome.trim()) ;  //retorna 'Maria'
```

Função substring

Retorna uma nova string a partir de um subconjunto da string original.

Exemplos:

```
console.log('Maria'.substring(2)); // retorna 'ria'

console.log('Maria'.substring(0,2)); // retorna 'Ma'
```

Função split

Cria um array com a separação do texto conforme o parâmetro informado.

Exemplo:

```
let linha= "Eva;eva@teste.com;33221100";

let dados = linha.split(';');

console.log(dados[0]); // //Eva
```

	Anotações

console.log(dados[1]); //eva@teste.com

console.log(dados[2]); // 33221100

Função startWith

Esta função foi adicionada na versão ES6 e determina se o texto inicia com os caracteres informados no parâmetro.

Exemplo:

Console.log('Apex'.startWith('A')); //true

Função endWith

Esta função foi adicionada na versão ES6 e determina se o texto finaliza com os caracteres informados no parâmetro.

Exemplo:

Console.log('Apex'.endWith('M')); //false

Função toLowerCase

Retorna o texto em caixa baixa.

Exemplo:

Console.log('JAVASCRIPT.toUpperCase()); // javascript'

Função toUpperCase

Retorna o texto em caixa alta.

Exemplo:

Console.log('javascript'.toUpperCase()); // JAVASCRIPT

Anotações	

Função replace

Retorna uma nova string com a substituição de parte do texto pelos parâmetros informados

Exemplo:

```
console.log('Cristiano'.replace('o','e')); //Cristiane
```

Template string

Template string é um recurso adicionado no ES6 e serve para criarmos templates através de textos que podem ser completados com valores de variáveis.

Para criarmos um template precisamos colocar o texto desejado entre dois sinais crase ``. Para incluirmos os valores de uma variável no template devemos utilizar o sinal ${}.

Exemplo:

```
function imprimirDadosContato(contato){
    let template = `
        Nome:${contato.nome}
        Email:${contato.email}
    `;
    console.log(template);
}

imprimirDadosContato({nome:'Maria',email:'maria@teste.com'});
```

Anotações

Capítulo 09 - Arrays

Objetivos:

Neste capítulo você irá aprender:

- O que é um array
- Como criar um array
- Como adicionar elementos em um array
- Como remover elementos de um array
- Como fazer cópias de partes de um array
- Como transformar arrays
- Como ordenar dados de um array

Anotações	

O que é um array

Arrays podem ser descritos como containers que armazenam outros containers. Em outras palavras, em Javascript são objetos que armazenam uma lista de outros objetos.

Para acessarmos os elementos que estão contidos em um array devemos utilizar o índice da posição em que o elemento está armazenado.

A indexação dos arrays inicia-se na posição 0 e vai até o tamanho do array -1. Para sabermos a quantidade de elementos que um array possui podemos utilizar a propridade length.

Como criar um array

Em Javascript podemos criar arrays de forma literal utilizando o operador colchetes, ou criando uma nova instância do Objeto Array;

```
var nomes = []; // os colhcetes indicam que a variável deve apontar para um array
```

ou

```
var contatos = new Array();
```

Como adicionar elementos em um array

Podemos utilizar a função push para incluir um elemento no final do array:

```
var nomes = ['Eva','Maria'];

nomes.push('Adão');  //['Eva','Maria','Adão'];
```

Podemos utilizar a função unshift para incluir um elemento no início do array:

```
var nomes = ['Eva','Maria'];

nomes.unshift('Adão');  // ['Adao','Eva','Maria'];
```

	Anotações

Podemos ainda incluir um elemento no array utilizando um índice:

```
var nomes = [];

nomes[0] = 'Adão';
```

Como remover elementos de um array

Para remover um elemento do início do array podemos utilizar a função shift():

```
var nomes = ['Adão','Eva'];

nomes.shift(); // remove  Adão.    ['Eva']
```

Para remover um elemento do final do array podemos utilizar a função pop():

```
var nomes = ['Adão','Eva'];

nomes.pop(); // remove  Eva.    ['Adão']
```

Para remover um elemento do meio do array podemos utilizar a função splice(index,qt):

```
var nomes = ['Adão','Eva','José','Maria'];

nomes.splice(2,1); // remove  José.    ['Adão','Eva','Maria']
```

Como fazer cópias de um array

Para copiarmos os dados de um array podemos utilizar a função slice():

```
var nomes = ['Adão','Eva','José','Maria'];

var nomes2 = nomes.slice();
```

Anotações	

Como transformar array

Para transformar um array em texto podemos utilizar a função join.

Exemplo:

```
var nomes = ['Adão','Eva','José','Maria'];

console.log(nomes.join('->'))//escreve
                                                    //Adão->Eva->José->Maria
```

Também podemos utilizar a função map para criar um novo array com base em uma transformação que queremos realizar no array original.

```
Let numeros = [2,3,4,5,6];

Let multiplicados = numeros.map(function(atual){return atual * atual})

console.log(multiplicados); //[ 4, 9, 16, 25, 36]
```

Como ordenar dados de um array

Para ordenar os dados de um array podemos utilizar a função sort.

Exemplo:

```
let numeros = [5,3,7,1,10]

numeros.sort(function(num1,num2){
        if(num1 > num2){
                return 1;
        }else if(num1 == num2){
                return 0;
```

	Anotações

```
        }else{
                return -1;
        }
});


console.log(numeros);   // [1, 3, 5, 7, 10]
```

Anotações	

Capítulo 10 - Object

Objetivos:

Neste capítulo você irá aprender:

- Como declarar uma Objeto
- Declarando um objeto de forma literal
- Declarando um objeto através de uma função construtora
- Declarando um objeto através de uma classe

	Anotações

Como declarar uma objeto

Em Javascript podemos declarar um objeto de várias formas. Aqui será apresentada a definição de um objeto de forma literal, utilizando uma função construtora e também através da definição de classe.

Declarando um objeto de forma literal

Para declararmos um objeto de forma literal basta utilizamos o operador Chave {}, sempre que atribuímos este operador a uma variável estamos criando um novo objeto.

Exemplo:

```
var Pessoa = {};

Pessoa.nome = 'João'; // adiciona a propriedade nome ao Objeto criado
```

Também podemos definir propriedades para um objeto no momento de sua criação;

Exemplo:

```
var Pessoa = {nome: 'Maria'};

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

Declarando um objeto através de uma função construtora

Para criarmos um objeto através de uma função construtora basta criarmos uma função e em seguida chamá-la utilizando o operador new.

Exemplo:

```
function Contato(nome){
        this.nome = nome;
}

var contato1 = new Contato('Manoel');

console.log(contato1);
```

Anotações	

Declarando um objeto através de uma classe

A declaração de classes foi adicionada no ES6 e segue a sintaxe conforme exemplo abaixo.

Exemplo:

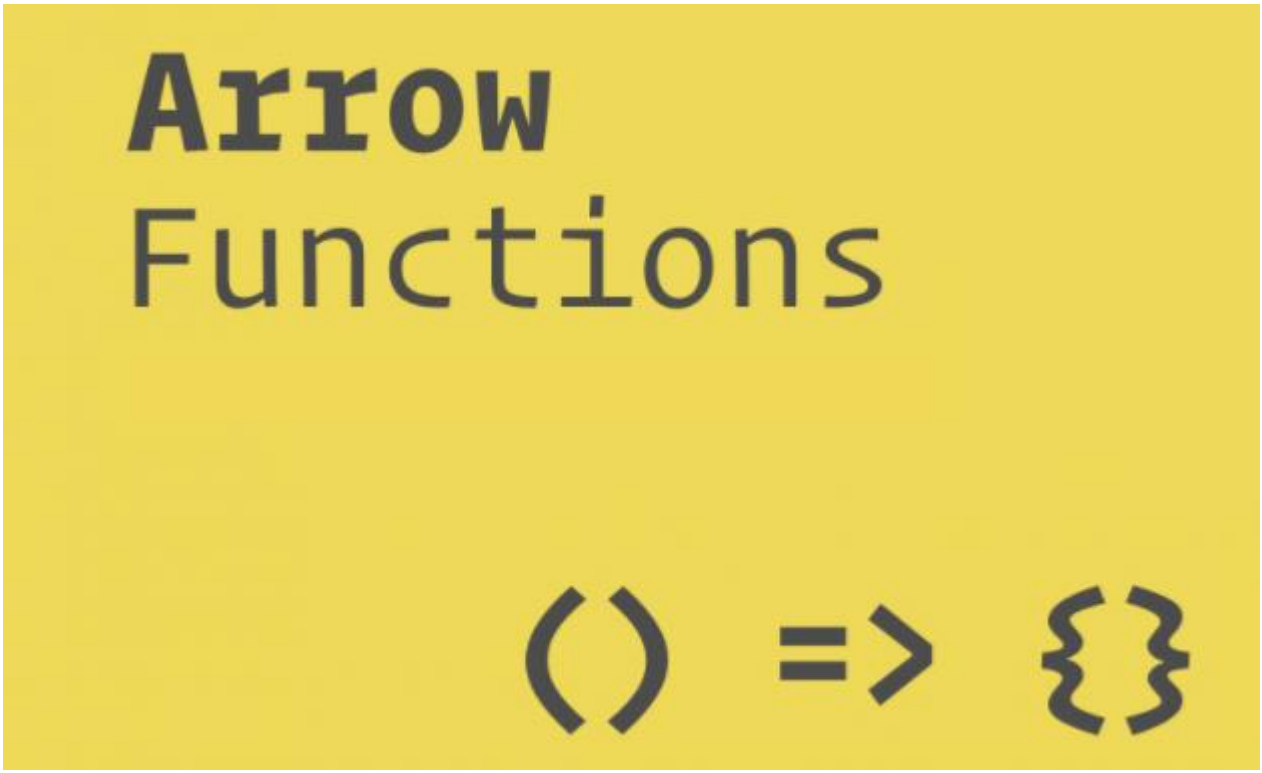
```
class Contato{
	constructor(name){
		this.name = name
	}
	 getTelefone(){
		return `Telefone do ${this.name}`
	}
}


var c1 = new Contato('Adão')
console.log(c1.name); //Adão
Console.log(c1.getTelefone()) //"Telefone do Adão"
```

Anotações

Anotações	

Capítulo 11 – Arrow Function



Objetivos:

Neste capítulo você irá aprender:

- Declaração e utilização de Arrow Function

Anotações

Declaração e utilização de Arrow Function

Arrow Function foi introduzida na versão ES6, e possui uma sintaxe mais curta que a declaração de uma função, outra diferença é que ela vincula o valor de this de maneira léxica.

Sintaxe

```
([param],[param])=>{
    //bloco de execução
}
```

Ou

```
param => comando_de_execução
```

Onde:

param: são os parâmetros necessários para a execução

Exemplos:

```
function callbackExec(callback){
    callback('Frontend nota 10');
}

callbackExec(mensagem =>console.log(mensagem));
```

Anotações	

```
class Contato{
	constructor(nome){
		this.nome = nome;
	}

	escreverNome(){
		window.setTimeout(()=>{console.log(this.nome)},100);
	}
}

var c1 = new Contato('Maria');
c1.escreverNome();
```

Anotações

Capítulo 12 – Manipulação do DOM

Objetivos:

Neste capítulo você irá aprender:

- Selecionar elementos do DOM
- Alterar propriedades de um elemento
- Alterar estilização de um elemento
- Como ler o texto de um elemento
- Como manipular elementos da página através de seu id
- Como manipular elementos da página através de sua classe
- Como manipular elementos da página através de sua tag

Anotações	

Introdução

Muitas vezes necessitamos buscar, alterar, adicionar ou remover informações contidas dentro de uma página html. Para que possamos realizar esse tipo de tarefa, o Javascript possui alguns métodos que podem no auxiliar na manipulação do Document Object Model (DOM).

O DOM é basicamente uma árvore de componentes com todas as suas ramificações, desta forma é possível navegar nesta árvore e encontrar as informações desejadas.

Para poder manipular o DOM, o Javascript nos fornece um objeto implícito chamado document. Este objeto possui uma série de funções para a manipulação do documento.

Para compreendermos alguns conceitos básicos vamos criar um documento html com a seguinte estrutura:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Título</title>
<style>
.azul{
        color: blue;
}
.verde{
        color: green;
}
</style>
</head>
<body>
<h1 id="titulo">Título grande</h1>
```

Anotações

```
<h3>Subtítulo</h3>

<h4>Nome</h4>

<ul id="listaNomes">

<li class="verde">Adão</li>

<li class="azul">Eva</li>

<li class="verde">João</li>

<li class="azul">Maria</li>

</ul>

<form>

        <label>Nome</label>

        <input name="txtNome">

        <button type="submit">Enviar</button>

</form>

</body>

<html>
```

Selecionar elementos do DOM

Manipulando elementos da página através de seu id

Para acessar os elementos da página através do seu id podemos utilizar a função document.getElmentById(). Esta função nos retorna um objeto que contém as informações do elemento desejado.

Exemplo:

```
var el = document.getElementById('titulo');

console.log(el.innerHTML); //  retorna  “Título grande”
```

Anotações	

Manipulando elementos da página através da sua classe

Para acessar os elementos da página através do seu id podemos utilizar a função document.getElementByClass (). Esta função nos retorna um objeto que contém as informações do elemento desejado.

```
var el = document.getElementByClass('verde'); //

console.log(el.item(0).className); // “verde”
```

Manipulando elementos da página através da sua tag

Para acessar os elementos da página através do seu id podemos utilizar a função document. getElementByTagName(). Esta função nos retorna um objeto que contém as informações do elemento desejado.

```
var el = document.getElementByTagName('ul'); //

console.log(el.item(0).childElementCount);  // 4

console.log(el.item(0).children[0].innerHTML); // Adão

console.log(el.item(0).children[3].innerHTML); // Maria
```

Buscando objetos através da função querySelector

A função querySelector busca o primeiro objeto no DOM que se encaixe com o seletor css informado.

Exemplos:

```
document.querySelector(‘#listaNomes’) //busca alista de nomes

document.querySelector(‘.azul’) //busca o primeiro item com a
                                //    classe  azul

document.querySelector(‘h3’) busca o primeiro h3
```

Anotações

Buscando objetos através da função querySelectorAll

A função querySelectorAll retorna uma lista de objetos no DOM que se encaixem com o seletor css informado.

Exemplos:

```
document.querySelectorAll('.azul') //busca os itens com a classe
                                   //azul

document.querySelectorAll('li') busca os itens da lista
```

Alterar estilização de um elemento

Para alterarmos a estilização de um elemento podemos alterar as propriedades de estilo através da propriedade style do objeto.

Exemplo:

```
let h3 = document.querySelector('h3');

h3.style['color'] = 'green';
```

Como ler o texto de um elemento

Para lermos o texto de um input podemos utilizar a propriedade value.

Exemplo:

```
console.log(document.querySelector('input').value);
```

Para lermos o texto contido em elemento html, como um parágrafo ou uma div por exemplo, podemos utilizar a propriedade innerText. Caso queira ler e manipular o html do elemento pode ser utilizada a propriedade innerHTML.

Exemplo:

```
console.log(document.querySelector('h3').innerText);

console.log(document.querySelector('listaNomes').innerHTML);
```

Anotações	

Capítulo 13 – Eventos

Objetivos:

Neste capítulo você irá aprender:

- O que são eventos
- Principais eventos
- Como se registrar para um evento
- Como deixar de ouvir um evento

	Anotações

O que são eventos

Eventos são funções executas pelo browser em decorrência de ações realizadas pelo usuário ou pelo próprio browser.

Principais eventos

Click - Disparado quando o mouse é pressionado sobre um elemento

Keydown - Disparado quando a tecla é pressionada para baixo.

Keyup - Disparado quando a tecla é liberada.

Keypress - Disparado após a tecla ser pressionada e liberada.

Focus - Disparado quando um elemento recebe o foco.

Blur - Disparado quando um elemento perde o foco.

MouseOver - Disparado quando o mouse está sobre o elemento

Input - Disparado quando um input, textarea ou select é alterado.

Load - Disparado quando o elemento é carregado.

Como se registrar para um evento

Para que possamos ser notificados sobre um determinado evento, devemos nos registrar no elemento passando uma função de notificação. Esse registro é feito através da função addEventListener.

Exemplo:

```
let listener= document.querySelector('button').addEventListener(
                                                    'click',function(evt){
    Console.log('Click no botão')
})
```

Anotações	

Como deixar de ouvir um evento

Para deixar de ouvir um evento, basta chamar a função removeEventListener, passando como parâmetros o nome do evento e o listener que se ligava ao evento.

Exemplo:

```
document.querySelector('button').removeEventListener('click',listener);
```

Anotações