# TEXTO DE APOIO

AULA 5

## Web Mobile

**Professor** Pedro Henrique Cacique Braga

EaD | Universidade Presbiteriana Mackenzie

# Sumário

# JAVASCRIPT PARA FRONT-END

Criar uma página web estática já é uma tarefa simples para você, não é mesmo? Agora começaremos a deixá-la um pouco mais dinâmica.

A linguagem Javascript é a principal utilizada para trabalhos com o front-end. Por ser uma linguagem compilada e orientada a objetos, é bem comum e de fácil acesso aos desenvolvedores.

Atualmente, a linguagem pode ser utilizada também no back-end, em sua forma pura ou usando alguns frameworks bem estabelecidos.

Javascript é uma linguagem que divide opiniões entre os desenvolvedores. Ela foi criada inicialmente para dar vida aos sites. Trata-se de uma linguagem interpretada. Os programas nesta linguagem são conhecidos como scripts e baseados na especificação ECMAScript.

Ela começou com um nome diferente: LiveScript, mas logo foi alterado para JavaScript, provavelmente para pegar carona na fama de uma linguagem já estabelecida: o Java (ainda que sejam linguagens bem distintas).

O JS hoje roda tanto no navegador, por meio de uma máquina virtual quanto no servidor, usando uma Javascript engine.

O que faz do Javascript uma linguagem única e amplamente utilizada?

1. Tem completa integração com HTML/CSS.

2. É uma linguagem simples (há controvérsias).

3. Suportada pela maioria dos navegadores e habilitada por padrão.

Mas a sintaxe desta linguagem ainda não supre todas as necessidades dos diferentes contextos em que ela pode ser utilizada. Recentemente, surgiram muitas outras linguagens que são convertidas para JS antes de serem executadas no navegador. Assim, é possível trabalhar com diferentes sintaxes e diferentes paradigmas de programação nos diferentes contextos em que o Javascript é necessário.

São algumas das linguagens existentes:

- CoffeScript – tende a simplificar um pouco a sintaxe de JS.

- TypeScript – concentra-se em adicionar tipos de dados específicos, para suportar sistemas mais complexos.

- Flow – também adiciona alguns tipos de dados; desenvolvida pelo Facebook.

- Dart – é uma linguagem que tem sua própria engine e roda em ambientes externos aos navegadores; desenvolvida pelo Google.

- Brython – é uma transposição de python para Javascript.

Estas são apenas algumas das linguagens “derivadas” de JS, mas existem muitas outras, cada uma com um contexto específico de uso.

## EDITORES

Existem muitos editores que suportam a linguagem. O termo IDE (Integrated Development Environment) se refere a um editor que consegue dar suporte a todo o processo de desenvolvimento de um tipo de sistema. Trata-se de um ambiente de desenvolvimento completo, não apenas um editor.

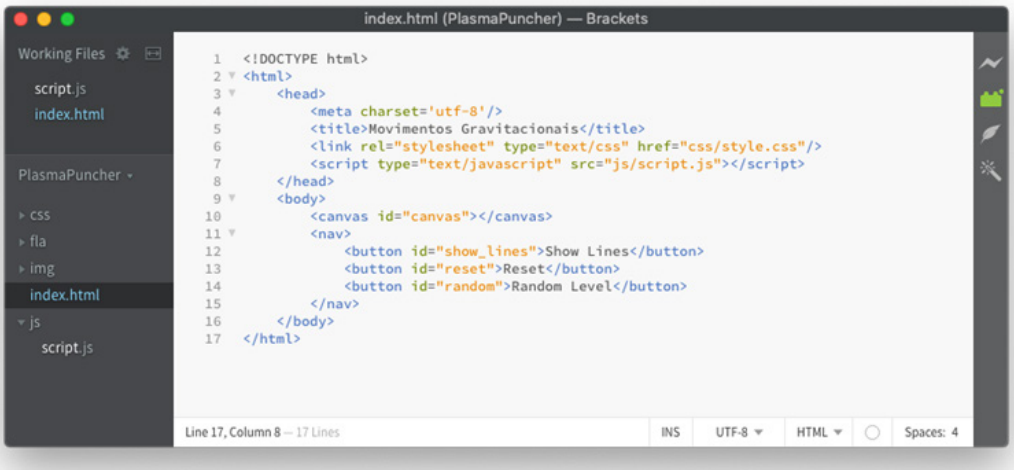
Na linha dos IDEs, pode-se citar o Visual Studio Code e o WebStorm, como bons exemplos. Entretanto, por ser uma linguagem de simples interpretação, existem alguns editores mais “leves”, que não são considerados IDEs poderosos, mas fazem bem o trabalho de edição e interpretação do código.

Nesta linha, podemos citar: Atom, Brackets, Sublime Text e Notepad++

Existem inúmeros editores, assim como existem inúmeras linguagens baseadas em JS. Dessa forma , você pode escolher o editor que mais te agrada em termos de interface e de usabilidade.

Eu, particularmente, gosto muito do Brackets (Figura 1), por ter uma interface bem simples, mas permitir a execução, em tempo real, de diferentes templates para formatação de cores. Há o benefício de ser open source, assim posso instalar diferentes plugins ou até criar o meu.

**Figura 1 – Brackets**




Você pode, ainda, optar por algum editor online, para poder programar de qualquer lugar, usando qualquer dispositivo e sistema operacional.

Um bom editor online que serve também como portfólio, pois tem uma boa comunidade envolvida, é o **CodePen.**

## CONEXÃO COM O HTML

Comecemos com a conexão do Javscript com o HTML. Podemos fazê-la de algumas maneiras. A mais simples é usando o código incorporado, assim:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset='utf-8'/>
        <title>Teste</title>

        <script type="application/javascript">
            console.log("Hello World!");
        </script>

    </head>
    <body>

    </body>
</html>
```

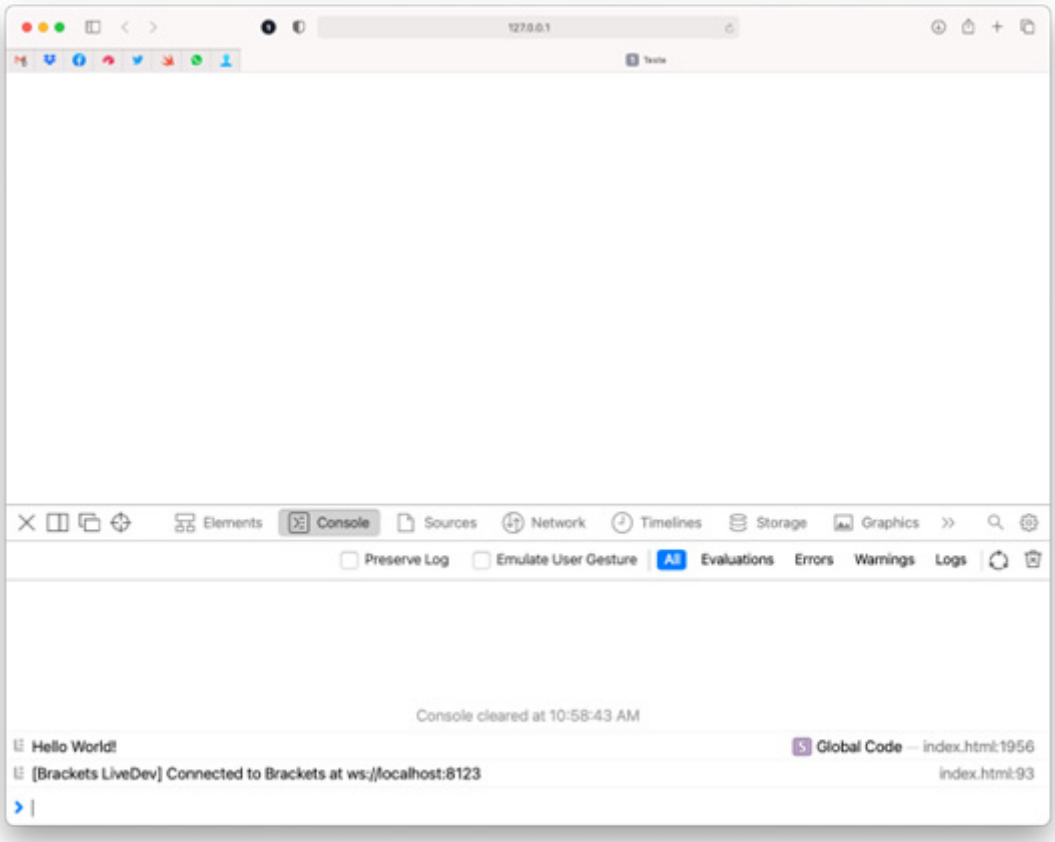
Veja que acrescentamos uma tag <script> dentro da tag <head>. Lembre-se de que, na tag <head>, usamos os atributos de configuração da página e, na tag <body>, aquilo que é visível no corpo da página.

Ao acrescentarmos o código no head, estamos dizendo que este script será executado no carregamento da página, antes mesmo que o body seja construído.

Você deve ter percebido que utilizamos uma função do Javascript muito comum: o log no console. Esta função serve para mostrar alguma informação no console de desenvolvimento do navegador. Ou seja, é uma ferramenta de desenvolvimento, a qual não é apresentada para o usuário final.

Para abrir o console em um navegador, é possível clicar com o botão direito em uma parte vazia da tela e escolher a opção inspecionar elemento. Dependendo do navegador, você pode ter de habilitar esta opção primeiro em um dos seus menus. A Figura 2 apresenta o console do navegador safari, quando executado o código anterior.

**Figura 2 – Console**

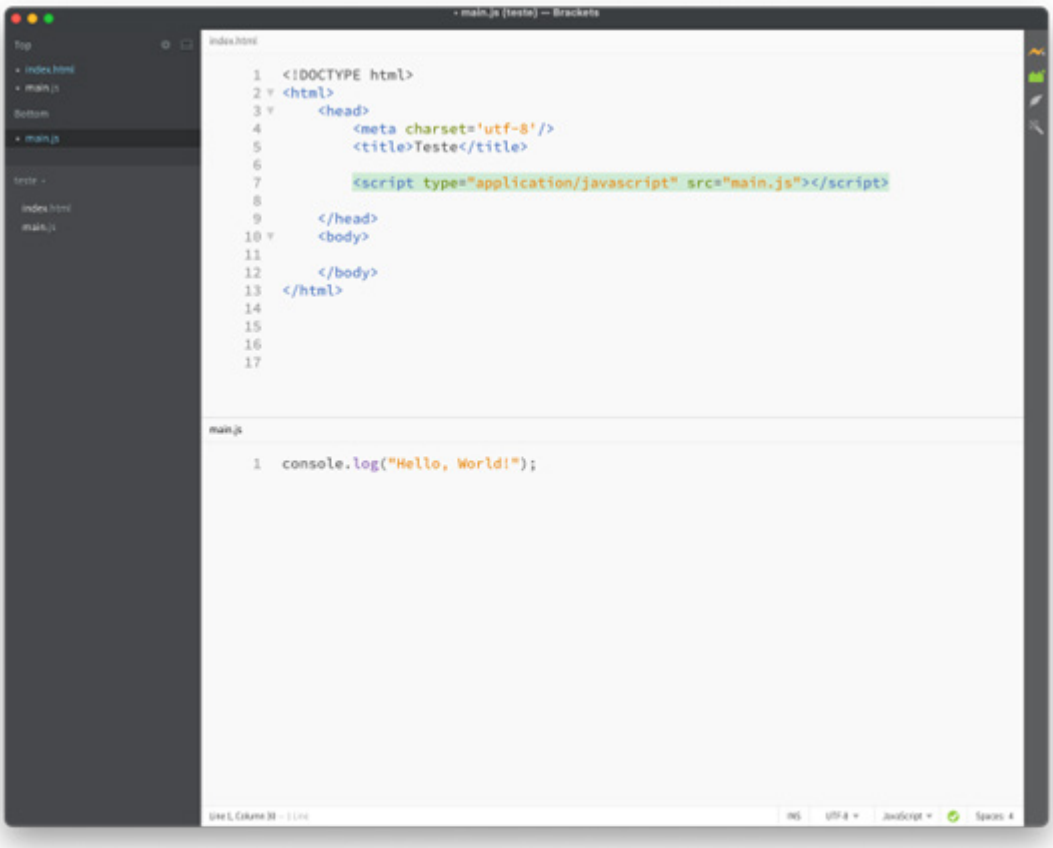



Esta primeira forma de inserção de Javascript é bem simples e usada sobretudo

para códigos pequenos. Quanto maior o código escrito, maior o tamanho do arquivo. Portanto, pode ser interessante trocarmos esta abordagem pelo link para um arquivo dedicado ao Javascript.

Veja, na Figura 3, que, em vez de colocarmos todo o código dentro do HTML, fizemos uma ligação com outro arquivo criado. Para isso, criamos um arquivo chamado main.js (você pode usar o nome que quiser, com a extensão js no final) e o conectamos na tag <script>.

**Figura 3 – arquivo js externo**




Na imagem, temos os dois arquivos abertos ao mesmo tempo em uma divisão de tela, proporcionada pelo Brackets.

# O BÁSICO DA LINGUAGEM

Variáveis e constantes

Começaremos a desenvolver alguns trechos bem simples de código, ok? Para declarar uma variável, use a palavra reservada var seguida do nome da sua variável. Lembre-se do ponto e vírgula ao final da instrução.

```
var nome;
```

Para atribuir um valor a esta variável, use o sinal de igualdade.

```
nome = "Pedro";
```

Você pode, ainda, declarar e instanciar uma variável em uma única linha.

```
var sobrenome = "Cacique";
```

Caso o valor desta variável não se altere em seu script, opte por uma constante. Isso pode ajudar no armazenamento e tornar o carregamento dos dados mais rápido. Para isso, use as palavras reservadas const ou let.

```
const PI = 3.14;
```

Veja que, em JS, não determinamos o tipo de dados para uma variável ou constante. Seu tipo será inferido pelo valor a ela atribuído.

## OPERAÇÕES BÁSICAS

Você pode trabalhar com todas as operações aritméticas básicas usando as variáveis criadas. Utilize os símbolos:

+ (soma)

- (subtração)

* (multiplicação)

/ (divisão)

% (resto ou módulo)

Qual é o valor da variável c a seguir?

```
var a = 5;

var b = 2;

var c = a % b;
```

Se você disse que a variável c recebe o valor 1, você acertou. Neste caso, estamos usando o resto da divisão de 5 por 2, que é 1.

## COMENTÁRIOS

Para adicionar comentários ao seu script e ignorá-los na interpretação do código, utilize os caracteres // para comentário de linha e /* ... */ para comentário de bloco.

```
// este é um comentário de linha

/*

Este é um

bloco comentado

*/
```

## ESTRUTURAS CONDICIONAIS

A linguagem conta com as principais estruturas de desvio de fluxo, ou tomadas de decisão: IF, IF... ELSE e SWITCH.

Utilize a opção IF se apenas o caso em que uma condição é satisfeita for desejado. Caso queira tratar também o caso quando a condição não é satisfeita, use IF... ELSE

```
if (a%b == 0 ) {

 console.log("número par");

} else {

console.log("número ímpar");

}
```

Veja que, neste trecho de código, estamos usando o operador módulo para dizermos se um número é par ou ímpar. Afinal, todo número par dividido por 2 apresenta resto 0, não é mesmo?

Utilize SWITCH quando quiser tratar casos pontuais específicos.

```
switch(a){
    case 1:
        console.log("professor");
    break;
    case 2:
        console.log("diretor");
    break;
    default:
        console.log("professor");
}
```

## ESTRUTURAS DE REPETIÇÃO

Um **FOR** é uma estrutura de repetição contável.

```
for (var i = 0; i < 5; i++) {
    console.log(i);
}
```

Esta talvez seja a estrutura de repetição mais simples e mais utilizada. Veja que são necessários, para sua estrutura, três parâmetros: a variável contadora, seu ponto de parada e seu incremento.

No caso acima, a estrutura escreve no console cinco números, variando de 0 a 4, incrementando de um em um.

Caso precise de uma estrutura não contável, utilize um WHILE para realizar um bloco de condições enquanto uma condição é válida.

```
var cont = 0;

while(cont < 5 ){

    console.log(cont);

    cont++;

}
```

E, se precisar realizar o teste da condição antes de executar o bloco, use o DO... WHILE.

```
var cont2 = 0;

do{

    console.log(cont2);

    cont2++;

} while(cont2 < 5 )
```

## FUNÇÕES

Para criar um bloco de instruções que deve ser reutilizado durante o script, crie funções com ou sem retorno.

```
function soma(var a, var b){

    return a + b

}
```

Você consegue perceber como a linguagem Javascript se assemelha às outras linguagens de programação que você está estudando? Veja que, até aqui, nós vimos a sintaxe das estruturas de lógica de programação que podem ser aplicadas a qualquer linguagem.

Utilize este material sempre que precisar relembrar algum detalhe!

# WEBSERVICES

Um webservice é um pequeno serviço online que tem uma finalidade única. Ele pode estar relacionado à autenticação ou às consultas em bancos de dados, por exemplo, mas não se limita a isso.

Pense em um webservice como um pequeno fragmento de código que garante acesso a um sistema maior.

Eles são comumente usados para consultas a bancos de dados ou partes específicas do sistema.

Uma API (Application Programming Interface) normalmente tem acesso por meio de uma URL (Uniform Resource Locator). Para cada serviço, temos uma URL diferente, direcionando o sistema a uma rota específica. É o que chamamos de endpoints.

Por exemplo, existe uma API chamada covid19api que traz informações diárias sobre a pandemia do coronavírus. O link a seguir representa um endpoint específico para receber os dados de um único país; neste caso, o Brasil.

https://api.covid19api.com/dayone/country/brazil

Veja que os parâmetros da URL indicam parâmetros que o serviço precisa para fazer a busca no seu banco de dados e retornar a informação específica.

Esta consulta retorna um json com os dados diários do nosso país, em um arquivo no formato JSON (Javascript Object Notation). Perceba que cada registro do banco de dados foi transformado em um objeto no JSON.

Um objeto JSON contém propriedades que são sempre representadas em pares ordenados, de nome e valor. Assim, a primeira propriedade se chama Country, por exemplo, e seu valor é a string (texto) Brazil.

Veja que este objeto organiza as informações necessárias para a compreensão da informação diária.

```
{
    "Country": "Brazil",
    "CountryCode": "BR",
    "Province": "",
    "City": "",
    "CityCode": "",
    "Lat": "-14.24",
    "Lon": "-51.93",
    "Confirmed": 4315687,
    "Deaths": 131210,
    "Recovered": 3723206,
    "Active": 461271,
    "Date": "2020-09-12T00:00:00Z"
}
```


O que precisamos saber sobre a notação de objetos do JSON? O primeiro ponto já conhecemos: pares ordenados! Outro ponto importante é que um objeto está sempre encapsulado entre chaves.

Os nomes das propriedades devem sempre estar entre aspas, seguidos de dois pontos, isso indica que, em seguida, teremos o seu valor. O valor pode ser numérico (sem aspas) ou textual (com aspas). Utilize vírgulas para separar as propriedades.

```
"CountryCode": "BR",
```

Para representar um array de objetos, utilize colchetes e separe os blocos dos objetos por vírgulas. No exemplo a seguir, suprimimos o conteúdo total dos objetos para simplificar o texto e focar apenas no que nos interessa no momento.

```
[{
    "Country": "Brazil",
                ...
    "Date": "2020-09-11T00:00:00Z"
},
{
    "Country": "Brazil",
                ...
    "Date": "2020-09-12T00:00:00Z"
}]
```

Veja, ainda, que todos os objetos devem apresentar a mesma estrutura para garantir a coesão dos dados transferidos.

É comum que as APIs apresentem páginas com informações dos dados, assim podemos saber exatamente o que esperar ao chamarmos um endpoint.

Para que uma API possa ser utilizada, é importante que haja uma manutenção de seus dados e seu serviço, assim, outras aplicações podem contar sempre com seu funcionamento.

Sempre garanta que seu app tenha um plano B caso a API não esteja mais funcionando. Não dependa totalmente de terceiros.

## CONSUMINDO OS DADOS DA API

Ok! Agora veremos como podemos trazer os dados do JSON para nossa aplicação.

Criamos uma função chamada getJSON que recebe dois parâmetros: a url do endpoint e uma função callback, que será chamada ao receber os dados.

```
function getJSON(url, callback) {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);
    xhr.responseType = 'json';
    xhr.onload = function () {
        if (xhr.status === 200) {
            console.log('Dados recebidos
                                com sucesso!');
            callback(xhr.response);
        } else {
            console.log('Problema ao conectar
                    com a API: ' + xhr.status);
        }
    }
    xhr.send();
}
```

Perceba que podemos passar uma função inteira como parâmetro de outra, no Javascript!

Esta função é responsável por abrir uma requisição via web, com base no protocolo HTTP. Veja que criamos uma variável xhr que recebe um novo pacote de mensagem HTTP.

Em seguida, determinamos seus parâmetros:

1. O tipo de requisição (GET) e a url.

2. O tipo de resposta: JSON.

Em seguida, criamos uma função anônima que será a resposta do evento de carregar a mensagem. Nesta função, precisamos verificar primeiro o código de resposta. Existe uma tabela de códigos de resposta HTTP. O código 200 significa sucesso! Ou seja, é possível verificar o corpo da mensagem para buscar a informação solicitada.

Uma vez verificado o sucesso, podemos chamar nossa função de callback, passando para ela os dados necessários. O exemplo passa nosso endpoint para a função getJSON e uma função anônima que fará algo com a resposta.

Perceba que estipulamos que a função callback sempre pedirá um parâmetro com todos os dados recebidos.

Por exemplo, se quisermos mostrar, no console, os resultados obtidos nesta consulta, podemos criar um loop para percorrer o conteúdo da variável data e apresentá-los individualmente no console.

```
var url = "https://api.covid19api.com/dayone/
country/brazil";

getJSON(url, function (data) {


});



getJSON(url, function (data) {
    for(var i=0; i<data.length; i++){
        console.log(data[i]);
    }
});
```

Agora, o que será feito com cada informação depende da proposta do seu site. Você pode mostrar apenas a informação com o nome do país, por exemplo. Para isso, acesse a propriedade Country de cada objeto.

Já sabemos como consumir os dados de uma API. Agora pratique! Assista à videoaula e faça os desafios propostos para consolidar o que você aprendeu!