eFront - HTML, CSS e JavaScript

Introdução

Um desenvolvedor front-end precisava dominar só o HTML, o CSS e JavaScript para ter uma boa colocação no mercado. Claramente, outras coisas eram incluídas nesse "só", como por exemplo: responsividade, semântica do HTML, acessibilidade, performance entre outras. Mas tudo girava em torno das três tecnologias principais.

A web mudou. Entraram outras habilidades como versionamento de arquivos, automatização de tarefas, pré-processadores, bibliotecas, frameworks, NodeJS e gerenciador de pacotes, para citar alguns. Mas a base de tudo ainda gira em torno das três tecnologias principais.

Isso quer dizer que não importa a época ou a tecnologia, a base sempre vai ser a mesma. Então antes de pular para o tão famoso framework, temos que aprender a base dele para realmente entender o que está por trás dos panos. Sei que encontrar material de qualidade e com uma linguagem simples para quem está começando não é uma tarefa muito fácil. Com essa premissa em mente, criei esse e-book onde irei ensinar alguns fundamentos básicos do front-end.

Fundamentos do HTML

O que é HTML

HTML, ou Hypertext Markup Language é uma linguagem de marcação (não de programação) da web - cada vez que você carrega uma página da web, você está carregando um código HTML. Pense em HTML como o esqueleto de uma página da web, ele é responsável pelos textos, links, listas e imagens - ele oferece conteúdos.

Iniciando com HTML

HTML é escrito em arquivos .html. Para criar uma página HTML é fácil, entre em seu editor de código (se não tiver recomendo o visual studio code) e salve o arquivo em branco como index.html (você pode nomeá-lo como quiser, mas não esqueça do .html).

A estrutura inicial do seu html será essa:

```
<!DOCTYPE html>
<html>
 <head></head>
 <body></body>
</html>
```

<!doctype html> - Essa tag (não tem fechamento) informa ao seu navegador que o arquivo faz parte de um documento HTML5.
<html> - Representa a raiz de um documento HTML.
<head> - O head contém informações sobre sua página, mas não é o conteúdo que vai aparecer na sua página. Ela conterá coisas como: links para folhas de estilo (CSS), título da página, links de fontes e meta tags e entre outros.
<body> - No body contém todo o conteúdo que vai aparecer na sua página. Todo o código que você queira apresentar na página deverá estar dentro dele.

Sintaxe

Os elementos HTML são escritos usando tags. Todas as tags tem uma chave de abertura e fechamento. A tag de fechamento tem uma barra após o primeiro colchete.

Os elementos HTML são escritos usando tags. Todas as tags tem uma chave de abertura e fechamento. A tag de fechamento tem uma barra após o primeiro colchete.

```
<tag>iuricode</tag>
```

Por exemplo, se você deseja criar um parágrafo, usaremos as chaves de abertura e fechamento .

```
<p>Programador sem café é um poeta sem poesia</p>
```

Os elementos podem entrar dentro de outros elementos:

```
<pai>
 <filho>Esta tag está dentro de outra tag</filho>
</pai>
```

Indentação adequada

As quebras de linha entre suas tags são super importantes para escrever um bom código HTML.

```
...
<body>
 <h1>Aqui temos um título</h1>
 <p>E aqui um parágrafo</p>
</body>
...
```

Abaixo temos um indentação não recomendada:

```
<!DOCTYPE html> <html> <head> </head> <body> <h1>Não faça isso!</h1> </body>
</html></code>
```

Títulos

As tags de títulos são representadas em até seis níveis. <h1> é o nível mais alto e <h6> é o mais baixo. Quanto maior for o nível da tag, maior vai ser o tamanho da fonte e sua importância no SEO.

```
<h1>Título do h1</h1>
<h2>Título do h2</h2>
<h3>Título do h3</h3>
<h4>Título do h4</h4>
<h5>Título do h5</h5>
<h6>Título do h6</h6>
```

Textos

As tags de texto, definem diferentes formatações para diversos tipos de texto. Desde estilos de fonte, parágrafos, quebra de linha ou até mesmo spans. Enfim iremos conhecê-las:

<p> - Sendo a principal tag de texto, é usada para criar um parágrafo.
 - Mesmo tendo a sua funcionalidade parecida com o uso dos parágrafos, os spans geralmente são utilizados para agrupar uma pequena informação.
<b> - Deixa o texto em negrito.
<i> - Deixa o texto em itálico.
<hr> - Cria uma linha horizontal.
 - Adiciona importância (no SEO) no texto.

Imagem

Para adicionar uma imagem na página é bem simples, vejamos:

```
<img src="nomeDaPasta/nomeDoArquivo.formato" alt="descrição sobre a imagem">
```

O src vem de source, ele é atributo da tag <img>, nele vai conter o caminho da imagem que será inserida por você. Além dele, temos o atributo alt. Recomendo que você sempre coloque o atributo alt para que pessoas com deficiência saibam (através de um leitor) do que se trata a imagem na página. Alguns pontos importantes:

A tag img não tem a chave de fechamento.
Se você tiver a imagem de uma pasta, deve colocar o caminho dela dentro do src.

Div

A tag é utilizada para dividir elementos para fins de estilos (usando class ou id). Ele deve ser utilizado somente quando não tiver outro elemento de semântica.

Mesmo tendo a sua funcionalidade parecida com o que são geralmente utilizados para guardar uma pequena informação. A é usada para uma divisão de um conteúdo pois, como o uso dela ajuda a quebrar os elementos em linhas, deixando melhor a visualização.

```
<div>
 <p>Sou uma div</p>
</div>
```

A (link)

A tag <a> (ou elemento âncora) é um hiperlink, que é usado para vincular uma página a outra. O conteúdo dentro de cada <a> precisará indicar o destino do link. O atributo mais importante da tag <a> é o href, que indica o destino do link. Se a tag <a> não tiver o atributo href, ela será apenas um espaço reservado para um hiperlink.

Por padrão, os links aparecem da seguinte forma em todos os navegadores:

Um link não visitado está sublinhado e azul;
Um link visitado é sublinhado e roxo;
Um link ativo está sublinhado e vermelho;

```
<a href=”www.iuricode.com”>Portfólio</a>
```

Comentário no HTML

Para adicionar um comentário em um código HTML, é preciso utilizar as tags próprias para esse propósito, nosso é <!-- -->.
Portanto, todo o código que estiver dentro dessa tag não será executado.

```
<!-- sou um comentário -->
```

Table

A criação de uma tabela em uma página HTML é feita com a tag <table>.

```
<table></table>
```

Entretanto, somente essa tag não é o suficiente para formatar a tabela da maneira correta, pois ela precisa de outras tags para exibir a formatação adequada. Confira quais são eles nos próximos tópicos.

TR

Em todas as tabelas existem linhas. Para isso, utilizamos a tag <tr>, ela pode ser inserida em diferentes áreas da tabela, como no cabeçalho, no corpo e no rodapé.

```
<tr></tr>
```

TH

A tag <th> é utilizada para inserir o cabeçalho na tabela, ela pode ser inserida em diferentes áreas da tabela. Na região do corpo da tabela, ela serve para dar destaque e identificar cada coluna específica.

```
<th></th>
```

TD

```
<td></td>
```

Além das linhas, toda tabela precisa de uma coluna. Isso significa que é preciso adicionar a tag <td> em todas as linhas para criar as colunas desejadas e inserir o conteúdo a ser exibido.

Outras tags da tabela

- Definindo a legenda de uma tabela: <caption>
- Criando um rodapé: <tfoot>
- Especifica um grupo de uma ou mais colunas em uma tabela para formatação: <colgroup>
- Agrupa o conteúdo do cabeçalho em uma tabela: <thead>
- Agrupa o conteúdo do corpo em uma tabela: <tbody>
- Agrupa o conteúdo do rodapé em uma tabela: <tfoot>

Exemplo simples de uma tabela:

```
<table>
 <tr>
  <th>Nome</th>
  <th>eBook</th>

 </tr>
 <tr>
  <td>Iuri</td>
  <td>eFront</td>
 </tr>
</table>
```

Lista

Vamos agora falar um pouco sobre as listas (ordenadas e desordenadas) e como elas funcionam no HTML. As listas são muito importantes quando queremos listar itens na página.

O elemento <li> é usado para representar um item que faz parte de uma lista. Este item deve estar contido em um elemento pai: uma lista ordenada (<ol>), uma lista desordenada (<ul>).

Lista ordenada

As listas ordenadas (ou numeradas) são usadas para indicar alguma sequência ou numeração.

```
<ol>
 <li>html</li>
 <li>css</li>
 <li>javascript</li>
 <li>front-end</li>
</ol>
```

Caso você queira deixar em ordem alfabética é simples, coloque o atributo type="a" dentro da tag <ol>.

```
<ol type="a">
 <li>html</li>
 <li>css</li>
 <li>javascript</li>
 <li>front-end</li>
</ol>
```

Você também pode deixar com algarismos romanos com o atributo type="I".

Listas desordenadas

As listas não numeradas são usadas para listar itens, sem se preocupar com sua sequência. Chamamos apenas de lista de marcadores. Ela segue o mesmo padrão da ordenada apenas mudando de <ol> para <ul>. Em seu visual ele mudará de números para pontos.

```
<ul>
 <li>html</li>
 <li>css</li>
 <li>javascript</li>
 <li>front-end</li>
</ul>
```

Formulário

Formulários são um dos principais pontos de interação entre o usuário e sua página. Um formulário HTML é feito de um ou mais widgets. Esses widgets podem ser campos de texto, caixas de seleção, botões, checkboxes, radio buttons e entre outros elementos HTML.

Para construir o nosso formulário de contato, vamos utilizar os seguintes elementos: <form>, <label>, <input>, <textarea> e <button>.

Form

O <form> é o elemento que define o formulário e os atributos que definem a maneira como esse formulário se comporta.

```
<form action="/pagina-processa-dados-do-form" method="post"></form>
```

Informações:
O atributo action especifica para onde enviar os dados do formulário quando um formulário é enviado.

O atributo method especifica o método HTTP a ser usado ao enviar dados do formulário (ele pode ser get ou post).

Label

O elemento <label> é a maneira formal de definir um rótulo. Esse elemento é importante para acessibilidade - quando implementados corretamente, os leitores de tela falarão o rótulo de um elemento de formulário juntamente com as instruções relacionadas.

```
<label>Nome:</label>
```

Input

O elemento <input> é usado para criar controles interativos para formulários baseados na web para receber dados do usuário. A semântica de um <input> varia consideravelmente dependendo do valor de seu atributo type.

```
<input type=”email”></input>
```

No input temos o atributo type. Esse atributo define o tipo do nosso input.
Exemplo: o type="email" ele define que o campo aceita só endereço de e-mail.

Alguns tipos de inputs

O tipo de controle a ser exibido. O tipo padrão é text, se este atributo não for especificado. Os valores possíveis são:

- Um botão sem comportamento padrão: type=”button”.
- Uma caixa de marcação: type=”checkbox”.
- Um controle para especificar cores: type=”color”.
- Um controle para inserir uma data: type=”date”.
- Um campo para editar um endereço de e-mail: type=”email”.
- Um controle que permite ao usuário selecionar um arquivo: type=”file”.
- Um campo de texto cujo valor é ocultado: type=”password”.
- Um campo de texto com uma só linha para digitar termos de busca: type=”search”.

Fieldset

O elemento <fieldset> é usado para agrupar elementos em um formulário.

```
<fieldset></fieldset>
```

Legend

O elemento <legend> representa um rótulo para o conteúdo do <fieldset>.

```
<legend>Contato</legend>
```

Exemplo simples de um formulário

```
<form action="/pagina-processa-dados-do-form" method="post">
 <label>Nome:</label>
 <input type="text"/>

 <label>Email:</label>
 <input type="email">

 <label>Mensagem:</label>
 <textarea></textarea>
</form>
```

Tags semânticas

Na vida sempre temos uma forma correta de fazer as coisas, no HTML não é diferente! As tags semânticas além de deixar o código melhor para o SEO, ela ajuda outros desenvolvedores a entender seu código só batendo o olho.

Informações:
As tags semânticas não têm nenhum efeito na apresentação na página.
As tags semânticas tem significado e deixam seu conteúdo claro.

Exemplo de elementos não semânticos: e .

Exemplo de elementos semânticos: <form>, <table>, <nav>, <aside>, <article>, <footer>, <section> e entre outros.

Veja que é amplo, mas a tag <footer> dá significado (que é o rodapé).

Portanto ao invés de:

```
<div>Sou o rodapé</div>
```

Seria melhor:

```
<footer>Sou o rodapé</footer>
```

Section

O elemento <section> representa uma seção em um documento HTML, geralmente com um título, quando não existe um elemento semântico mais específico para representá-lo.

```
<section></section>
```

Article

O elemento <article> representa uma composição independente em um documento. Isso é, não precisa do resto do site para contextualizar. Este poderia ser um simples card, um artigo de revista ou jornal, um post de um blog, ou qualquer outra forma de conteúdo independente.

Para um melhor SEO a tag <article> precisa de um título (<h1> - <h6>) acompanhado.

```
<article></article>
```

Aside

O elemento <aside> representa uma seção de uma página que consiste de conteúdo que poderia ser considerado separado do conteúdo principal. Essas seções são, muitas vezes, representadas como barras laterais.

Um bom exemplo de uso são os sites de notícias, a maioria tem uma seção lateral para ver outras notícias que não estão relacionadas com a notícia principal da página.

```
<aside></aside>
```

Header

O elemento <header> (de cabeçalho) representa um contêiner para conteúdo introdutório (muitas vezes usado na primeira seção da página).

```
<header></header>
```

Footer

O elemento <footer> representa um rodapé para o seu conteúdo de seção.

```
<footer></footer>
```

Nav

O elemento <nav> (de navegação) representa uma seção de uma página que aponta para outras páginas ou para outras áreas da página, ou seja, uma seção com links de navegação. Pense nela como um agrupador de links.

```
<nav></nav>
```

Main

O elemento <main> define o conteúdo principal da página. Ele representa o conteúdo mais importante da página, que está diretamente relacionado ao tópico central do documento.

```
<main></main>
```

Meta tags

SEO (Search Engine Optimization) é o conjunto de técnicas responsáveis por fazer com que determinada página de um site apareça nas primeiras páginas do Google. Quanto melhor for o SEO de um site, maiores as chances de que a página apareça logo de cara para quem pesquisa por determinado assunto.

A tag <meta> define metadados (informações) sobre um documento HTML. As tags <meta> sempre vão dentro do elemento <head> e normalmente são usadas para especificar o conjunto de caracteres, descrição da página, palavras-chave, autor do documento e configurações da janela de visualização.

Informações:

- Os metadados não serão exibidos na página, mas podem ser analisados por máquina.
- Os metadados são usados por navegadores (como exibir conteúdo ou recarregar a página), mecanismos de busca e outros serviços da web.

Meta tag - Title

Title define o título do documento. Ele é mostrado na aba da página do navegador.

```
<title>Iuri Silva</title>
```

Meta tag - Description

Description contém um curta e precisa descrição do conteúdo da página, vários navegadores usam esta meta como descrição padrão da página quando é marcada.

```
<meta name=”description” content=”Descrição sobre a página”>
```

Meta tag - Charset

Este atributo define a codificação de caracteres usados na página. O conjunto de caracteres UTF-8 cobre quase todos os caracteres e símbolos do mundo.

```
<meta charset=”UTF-8”>
```

Meta tag - Keywords

Keywords é uma meta de palavras-chaves associadas ao conteúdo da página, contendo strings separadas por vírgulas.

```
<meta name=”keywords” content=”css, html, js”>
```

Meta tag - Author

Author define o nome do autor do documento.

```
<meta name=”author” content=”Iuri”>
```

Fundamentos do CSS

O que é CSS

CSS ou folhas de estilo em cascata é a linguagens de marcação (não de programação) responsável por adicionar estilos em nossas páginas web, como cores, tamanhos, posicionamentos e entre outros. Sem ele, as páginas são apenas um grupo de textos e links.

Etapa 1: Criar um arquivo CSS. A primeira coisa que precisamos fazer é criar um arquivo CSS do qual nossa página HTML possa obter seu estilo. Então em seu editor de código basta você criar um novo arquivo chamado style.css.

Etapa 2: Linkar seu CSS no HTML. Depois de criado precisamos conectar nosso arquivo style.css no documento HTML. Dentro da tag <head> do documento HTML, vamos adicionar uma tag <link> para conectar a nossa nova folha de estilo.

```
<link rel="stylesheet" href="style.css">
```

Certifique-se de que sua página HTML e sua folha de estilo CSS estão no mesmo nível de pasta, caso ele esteja dentro de uma pasta basta colocar o nome da pasta antes do nome do arquivo separado com uma barra (/).

```
<link rel="stylesheet" href="nomeDaPasta/style.css">
```

Etapa 3: Chame e dê estilo aos elementos. Experimente selecionar um elemento e estilizá-lo!

```
h1 {
 color: #00f;
 font-size: 26px;
}
```

Acabamos de chamar todas tags <h1> do nosso documento HTML e adicionar o tamanho da fonte para 26px e com a cor azul (sempre atualize a página em seu navegador assim que aplicar algo novo em seu arquivo CSS ou HTML).

Sintaxe

A sintaxe do CSS é bem simples.
Primeiro precisamos indicar um seletor (pela tag, id ou class) e dentro das chaves inserimos os comandos referente à formatação, que são as propriedade e valor.

```
seletor {
  propriedade: valor;
}
```

Comentários no CSS

Para adicionar um comentário em nosso documento CSS, é preciso utilizar /* */.
Portanto, todo o código que estiver dentro dessa tag não será executado.

```
div {
  /* color: blue; */
}
```

Variáveis

Uma grande aplicação tem uma quantidade muito grande de CSS, e com uma quantidade de repetição de valores muito frequente. Por exemplo, a mesma cor pode ser usada em vários lugares diferentes em nosso documento, caso seja requerido uma substituição na cor, teríamos que procurar todos os lugares que essa cor foi adicionada. Já com variáveis CSS a história muda. Ela permite que um valor seja guardado em uma variável, para ser referenciado em muitos outros lugares. Isso é ótimo para nós desenvolvedores pois, caso seja solicitado a substituição de trocar o valor dessa variável, apenas um lugar será substituído para que todo nosso documento sofra alteração.

Declarando e utilizando variáveis

São configuradas usando esta notação:

```
:root {
 --cor: black;
}
```

O :root se equipara à raiz de uma árvore, que representa o documento.

E são acessadas usando a função var():

```
h1 {
 var(--cor);
}
```

Aplicando fontes do Google Fonts

Para colocar uma fonte em sua página primeiro é preciso chamar ele no seu HTML, para isso vamos entrar no site do Google Fonts. Logo em seguida, procure a fonte que você deseja e clique no botão "Select this style" para assim adicionar os estilos de fontes que você deseja (perceba que quando você selecionar uma vai abrir uma aba na lateral direita).

Nessa aba vai ter duas opções para aplicar a fonte em sua página, uma é o <link> e o outro @import. O <link> é para importar pelo seu documento HTML e o import é para o seu documento CSS. Você pode escolher qualquer um, isso vai com seu gosto, mas importar pelo HTML nos trás uma melhor performance.

Perceba que logo depois de selecionar a opção <link> no campo de baixo é apresentado um código HTML e uma propriedade CSS, o font-family. É exatamente ele que usaremos em nossa página quais seletores irão receber a nossa fonte.

Em nosso documento HTML dentro da tag <head> coloque o código HTML apresentado no site do Google Fonts e em documento CSS, coloque o
código CSS (a propriedade font-family).

```
* {
  font-family: 'Nome da sua fonte aqui';
}
```

Aplicando reset em nossa página

Agora vamos desfazer os padrões dos navegadores... “mas espera, o que isso quer dizer?”. Quer dizer que os navegadores tem um padrão, e alguns desses padrões não são legais para nós desenvolvedores, então vamos tirar alguns deles.

Zerando os espaçamentos das nossas páginas: Quando criamos uma página HTML, por padrão, nosso site tem um espaçamento em volta da página. Para tirar esse padrão usamos duas coisas: o padding (espaçamento dentro do conteúdo) e o margin (espaçamento fora do conteúdo).

```
body {
  padding: 0;
  margin: 0;
}
```

E por último, vamos aplicar o box-sizing: border-box. Esse é super importante na criação dos elementos pois, quando criamos um container sempre usamos uma largura e altura (no nosso exemplo vai ser 300px em cada um) mas caso você queira colocar um espaçamento interno nele (padding) de 30px, o elemento vai deixar de ser 300px e será 330px.

Mas a gente não quer isso, não é? Queremos que o elemento continue 300px e com um espaçamento interno. É aí que entra o nosso querido box-sizing: border-box;

```
* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
```

PX e REM

O CSS oferece um número de unidades diferentes para a expressão de comprimento. Iremos citar só duas delas: PX (medida absoluta) e REM (medida relativa).

Medidas Absolutas: Essas são as mais comuns que vemos no dia a dia. São medidas que não estão referenciadas a qualquer outra unidade, ou seja, não dependem de um valor de referência. São unidades de medidas definidas pela física, como o píxel, centímetro, metro, etc...

Medidas Relativas: O uso delas é mais apropriado para que possamos fazer ajustes em diferentes dispositivos garantindo um layout consistente e fluido em diversas mídias.

PX

Definir unidades de texto em pixels traz uma desvantagem. Quando o usuário tenta mudar o tamanho do texto pelo browser, não aumenta o texto pelo simples motivo de que o texto está definido em pixels. Um problema sério de acessibilidade. É por isso que muitos devs preferem definir o tamanho do texto utilizando outras medidas em vez de trabalhar com pixels.

```
h1 {
 font-size: 18px;
}
```

REM

O nome rem significa root em. Se você definir na raiz do seu site font-size: 14px, 1rem sempre será 14px em absolutamente qualquer elemento.

```
h1 {
  font-size: 1rem;
}
```

Como disse acima, o valor base é 16px, e isso pode acabar gerando dificuldades para que encontremos alguns tamanhos padrões que costumam ser utilizados. Por exemplo, como

faríamos para atingir um tamanho de 10px utilizando rem? Precisamos calcular. Ficar calculando esses números não é algo muito "amigável", porém, podemos adicionar um pequeno truque para nos ajudar (62,5%).

```
html {
  font-size: 62, 5%;
}

h1 {
  font-size: 1.2rem; /*equivalente a 12px*/
}
```

Repare que dessa forma, o valor em pixel será sempre o valor definido em rem vezes 10! Exemplo: 1.2rem = 12px, 2.6rem = 26px, 5.6rem = 56px....

RGB e HEX

RGB e HEX são formatos de composição de cores digitais. Após compreender esses temas você pode escolher um dos formatos para utilizá-la mais frequentemente nos projetos.

RGB

O processo é simples: como na vida real onde você mistura cores para obter uma outra cor como resultado, você faz a mesma coisa como o RGB.

Sua sintaxe utiliza a soma de 3 valores: Red, Green e Blue. Onde o valor máximo de todas as cores é 255.

```
h1 {
  color: rgb(255, 200, 10);
}
```

Graças a função RGBA (red-green-blue-alpha) podemos adicionar opacidade no formato RGB.

```
h1 {
  color: rgba(255, 200, 10, 0.4);
}
```

O alpha define a opacidade como um número entre 0,0 (totalmente transparente) e 1,0 (totalmente opaco).

HEX

Hex, ou Hexadecimal, tem sintaxe muito mais curta que o RGB. O código Hex consiste em seis letras ou números precedidos por um “#”. Os seus dois primeiros valores representam a intensidade do vermelho, terceiro e quarto a intensidade de verde e os dois últimos a intensidade de azul.

```
h1 {
  color: #00ff00;
}
```

Você deve se perguntar “okay, mas se eu tiver mais de um elemento <p> e quiser mudar a cor do texto só de um?”.

É aí que entram os identificadores id e class:

- As classes são uma forma de identificar um grupo de elementos. Através delas, pode-se atribuir formatação a VÁRIOS elementos de uma vez.
- Os ids são uma forma de identificar UM elemento, e devem ser únicas para cada elemento. Através delas, pode-se atribuir formatação a um elemento em especial.

Para fazermos referência a uma classe usando um . (ponto) com o nome da classe.

```
.souClass {
  color: #f00;
}
```

E para fazermos referência a um id usando um # (hashtag) com o nome do id.

```
#soulD {
 color: #f00;
}
```

Outros seletores

Dependendo da estrutura da nossa aplicação, selecionar elementos acabam sendo complexos de certa forma, mas existem algumas formas de selecionar tags, id e class. Vou apresentar somente três delas.

Seleciona o primeiro parágrafo que vem exatamente após a div:

```
div + p {
 color: #f00;
}
```

Seleciona todos os elementos <p> com a class “iuricode”:

```
p.iuricode {
 color: #f00;
}
```

Seleciona todos os parágrafos que são filhos da div:

```
div > p {
 color: #f00;
}
```

Padding e Margin

As propriedades de preenchimento CSS são usadas para gerar espaço em torno ou dentro do conteúdo.

Padding

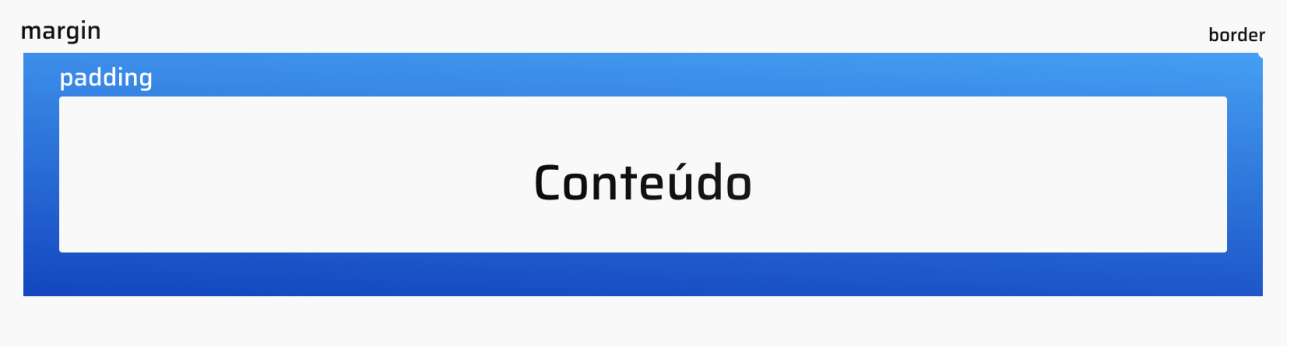
A propriedade de padding é usada para gerar espaço em torno do conteúdo. O preenchimento limpa uma área ao redor do conteúdo (dentro da borda) de um elemento.

Margin

A propriedade margin é usada para gerar espaço em torno de elementos. A propriedade margin define o tamanho do espaço de fora da borda.

Exemplo:

Padding é o espaço entre o conteúdo e a borda, enquanto margin é o espaço fora da borda



Como definir

Elas podem ser aplicadas nos quatros lados de um elemento: superior, direita, inferior e esquerda (nessa ordem).
No exemplo vamos usar o margin mas também pode ser aplicada no padding.

Margem de 20px iguais nos quatros lados do elemento:

```
.class {
  margin: 20px;
}
```

Margem superior e inferior de 5px e margem esquerda e direita de 10px:

```
.class {
  margin: 5px 10px;
}
```

Você também pode definir os espaços individualmente (mas lembre-se sempre da ordem):

```
.class {
  margin: 0px 5px 10px 15px;
}
```

Display

Basicamente todos os elementos têm um valor padrão para sua propriedade display, a maioria dos elementos tem seu display configurado em block ou inline.

Block

O elemento block, não aceita elementos na mesma linha que ele, ou seja, quebra a linha após o elemento, e sua área ocupa toda a linha onde ele é inserido.
Alguns elementos que têm como padrão block: , <h1> até <h6>, <p>, <form>, <header>, <footer>, <section>, <table>.

Inline

O elemento inline não inicia em uma nova linha nem quebra de linha após o elemento, pois ele ocupa somente o espaço de seu conteúdo.
Alguns elementos que têm como padrão inline: , <a>, <img>.

A propriedade position específica como um elemento será posicionado na tela, podemos até posicionar em um ponto específico controlando com os parâmetros top, right, bottom e left.

São quatro tipos de posicionamento disponíveis: static, relative, fixed e absolute. A única configuração que não permite escolher um posicionamento para o elemento é static.

- top - Desloca o elemento na vertical (Y), o valor é a distância do elemento com o topo.
- right - Desloca o elemento na horizontal (X), o valor é a distância do elemento com a borda direita.
- bottom - Desloca o elemento na vertical (Y), o valor é a distância do elemento com a borda inferior.
- left - Desloca o elemento na horizontal (X), o valor é a distância do elemento com a borda esquerda.

Static

Este é o valor padrão de todos os elementos HTML, neste posicionamento os elementos não podem ser controlados por top, right, bottom e left, e não tem seu posicionamento afetado pelo posicionamento de outros elementos.

Relative

Um elemento com relative tem seu posicionamento relacionado com o elemento anterior.

Absolute

Um elemento com absolute tem seu posicionamento relacionado com o elemento pai e não com o elemento anterior, desta maneira elementos anteriores não irão afetar seu posicionamento.

Fixed

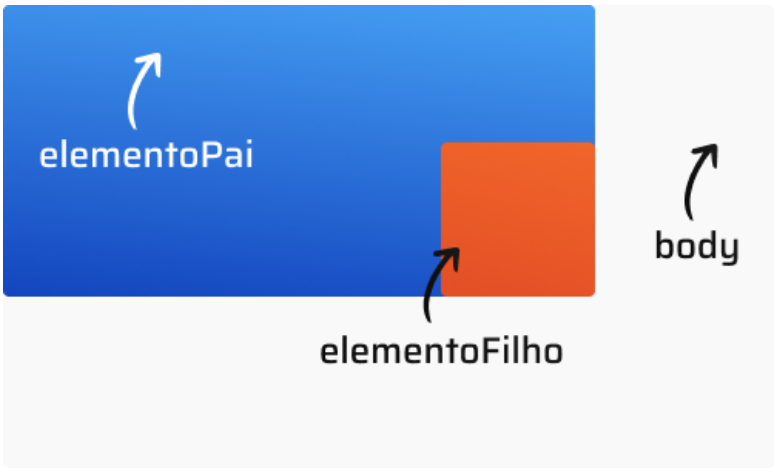
O elemento com fixed tem o mesmo comportamento do absolute, só que como o nome já diz, ele fica fixo na tela, isso é, mesmo se acontecer a rolagem ele ficará fixado na página.

Exemplos ilustrativos:


```
.elementoFilho {
 position: absolute;
 bottom: 0;
 right: 0;
}
```

Nesse exemplo, o nosso retângulo roxo é o body da nossa página. Por padrão, sempre que colocamos position absolute em um elemento ele irá se referenciar ao seu ancestral mais próximo (que no nosso caso é body). Além do retângulo roxo, temos nosso quadrado verde, ele é o nosso elemento com position absolute, o elemento fica posicionado de forma absoluta em relação ao fluxo do documento, mas de forma relativa ao seu ancestral.

Que tal um exemplo melhor?



```css
.elementoPai {
  position: relative;
}
```

```css
.elementoFilho {
  position: absolute;
  bottom: 0;
  right: 0;
}
```

Perceba que nosso elementoFilho continua com as mesmas propriedades, porém adicionamos o elementoPai e atribuímos a ele um position relative.
O que isso muda? Muda tudo! Lembra que eu falei que o position absolute em um elemento ele irá se referenciar ao seu ancestral mais próximo? Foi o que aconteceu! Como o elementoPai é o ancestral mais próximo do elementoFilho, obrigatoriamente o elementoFilho irá se referir a ele.

E como ficou nosso HTML?

Perceba que nossa classe elementoFilho está dentro da classe elementoPai, por isso que ele se referenciou a ele.

```html
<body>
  <div class="elementoPai">
    <div class="elementoFilho"></div>
  </div>
</body>
```

Okay Iuri, quer dizer que se o elementoFilho ficar fora do elementoPai ele irá se referenciar ao body? Sim, isso mesmo.

Flexbox

Flexbox são regras de CSS que configuram o alinhamento, posicionamento e distribuição ele elementos através de regras em uma caixa pai, responsável por organizar o conteúdo

dentro do espaço que ela possui. Facilitando, assim, a implementação de sites responsivos.

justify-content

A propriedade justify-content define como o navegador distribui o espaço entre e ao redor dos itens de conteúdos ao longo do eixo principal de um contêiner flexível e do eixo embutido de um contêiner de grade. A propriedade só terá efeito se a mesma tiver com a propriedade display flex.

```
.elementoPai {
 display: flex;
 justify-content: flex-start;
}
```

flex-start (padrão)

Os itens são postos no começo do eixo principal.

flex-end

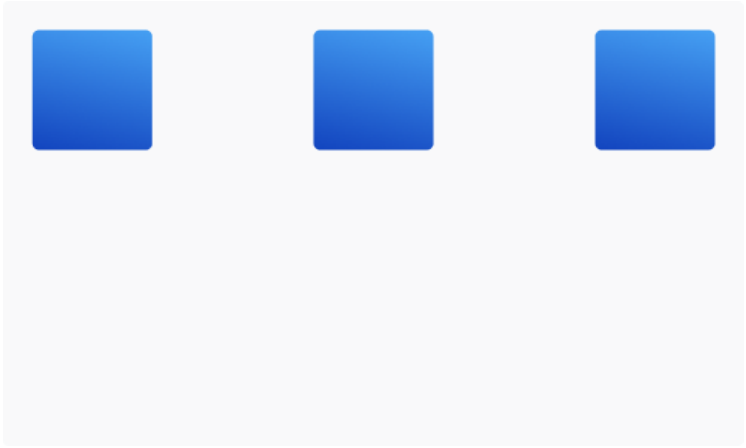
Os itens são alocados no final do eixo principal.

flex-center

Os itens são alinhados no meio do eixo.

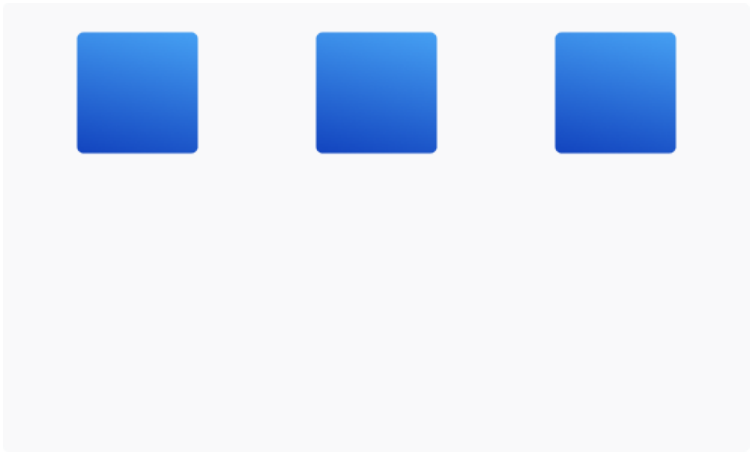
space-between

O primeiro item fica no começo do eixo e o último no fim, os restantes ficam alinhados entre si no meio.



space-around

O espaço vazio antes do primeiro e depois do último item é igual a metade do espaço entre cada par de itens adjacentes.



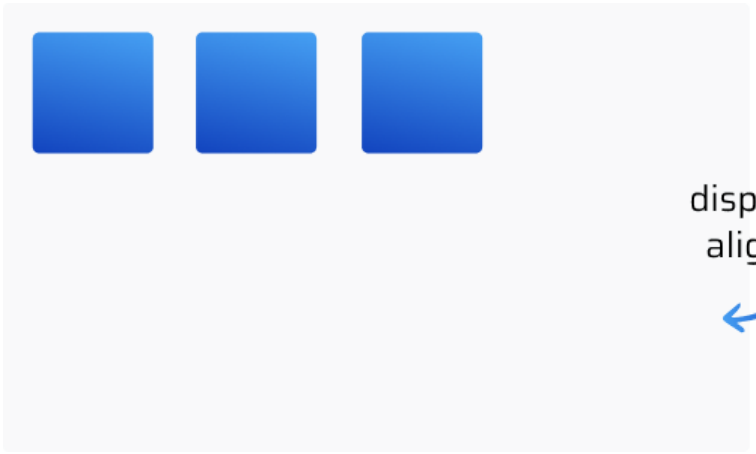
space-evenly

Os itens são colocados no eixo principal com espaçamentos equivalentes entre si.

align-items

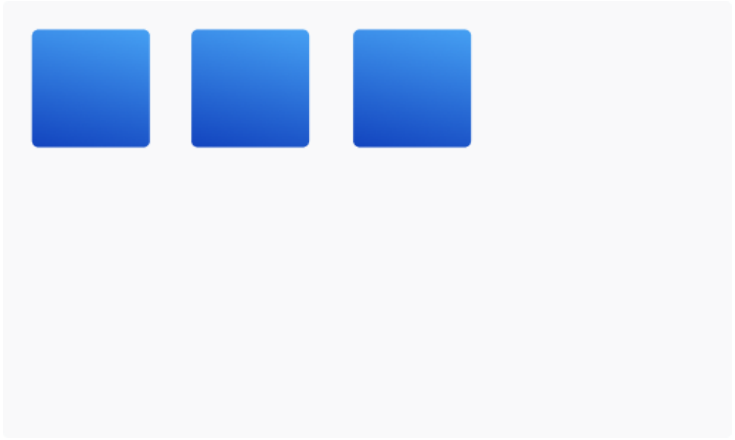
Nós usamos justify-content para alinhar o item no eixo principal, que neste caso é o eixo indo horizontalmente. Para centralizar nossa caixa nós usamos a propriedade align-items para alinhar nosso item no eixo transversal, que neste caso é o eixo do bloco indo verticalmente.

```
.elementoPai {
 display: flex;
 align-items: flex-start;
}
```



flex-start

Os itens são postos no começo do eixo transversal.



flex-end

Os itens são alocados no final do eixo transversal.

stretch

Os itens crescem igualmente.

center

Os itens são alinhados no meio do eixo.

baseline

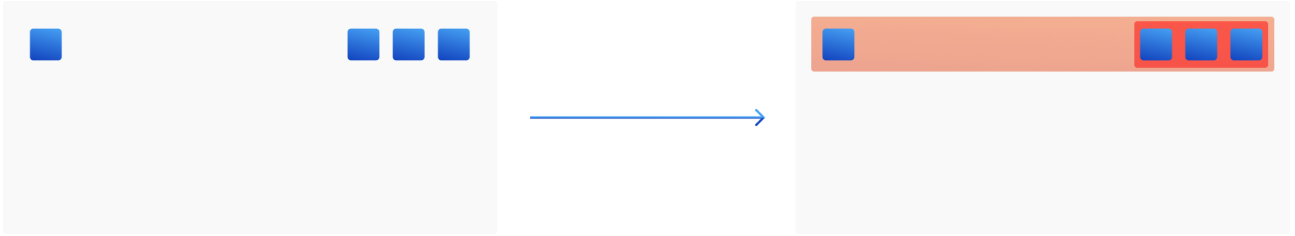
Alinha os itens de acordo com a linha base da tipografia.

Outras propriedades

Existem outras propriedades do flexbox como flex-basis, flex-direction, flex-grow, flex-shrink, flex-wrap, order, gap, align-self e entre outros. Caso você queira ver outras propriedades, clique aqui e leia mais.

Posicionamento com flexbox

Quando falamos em posicionamento, primeiro temos que pensar qual vai ser a estrutura HTML e assim conseguimos trabalhar com flexbox no CSS.



A estrutura acima segue o mesmo padrão de um menu, certo? Perceba a forma que separei a estrutura dele.

A parte laranja é o nosso menu (nav), dentro desse menu temos duas coisas, um quadrado azul na esquerda e um grupo de quadrados na direita. Para posicional eles precisamos agrupar esses quadrados azuis na direita pois, caso isso não ocorra, seria difícil posicional esses elementos.

Veremos esse exemplo em uma aplicação real.


```css
nav {
 display: flex;
 align-items: center;
 justify-content: space-between;
}
```

Para replicar esse menu do Instagram precisaríamos de 4 tags HTML (tirando o elementos em vermelho) para posicionar os elementos.

É claro que isso leva a outros fatores como tamanhos, alturas, espaçamentos e entre outras coisas. Mas se você está utilizando flexbox para criar uma estrutura e posicionar elementos, essa sempre vai ser a base.

O que é um site responsivo?

Sites responsivos são aqueles que adaptam o tamanho das suas páginas ao tamanho das telas que estão sendo exibidos, como as telas de celulares.

Qual a forma de aplicar?

Temos algumas formas de aplicar responsividade em nossos sites, as mais utilizadas são com o uso de frameworks e media queries. Se você está fazendo um site e tem um prazo curto para entregar ele, recomendo o uso de frameworks pois já está tudo pronto para um layout responsivo.

Quero deixar responsivo com media queries, como eu faço?

A responsividade é um estilo, então o media queries será aplicado em sua folha de estilo CSS. Mas antes temos que entender que temos alguns tipos de media, eles são chamados de media types.
Os types definem para que tipo de media um certo código CSS será aplicado.

Media type

Os types mais utilizados são:

- all: usado para todos os dispositivos de tipo de mídia.
- print: usado para impressoras.
- screen: usado para telas de computador, tablets, smartphones e etc.
- speech: usado para leitores de tela que “lêem” a página em voz alta.

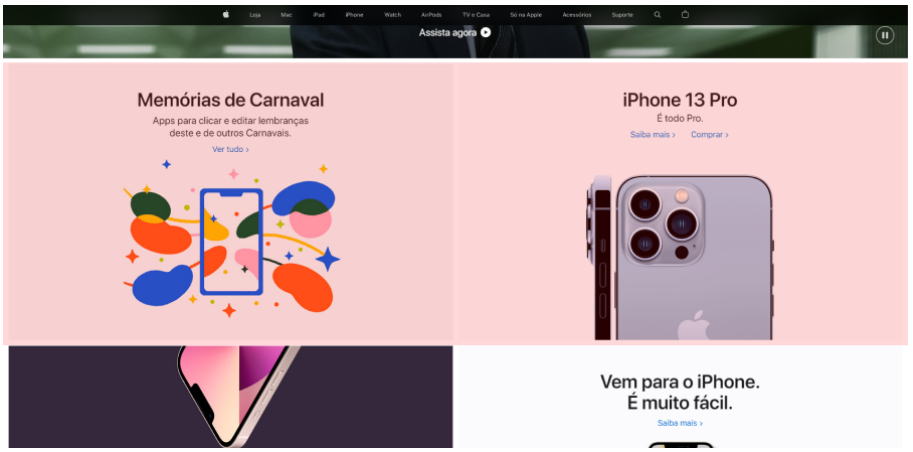
Mas como eu faço para aplicar?

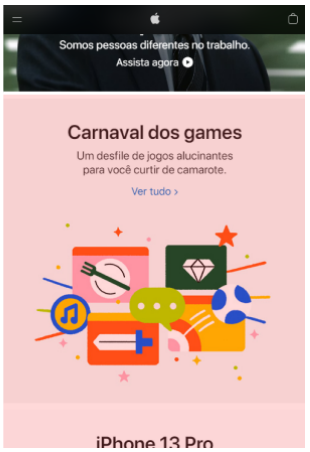
O exemplo a seguir altera a cor de fundo da página para roxo se a janela de visualização tiver 480 pixels de largura ou menos.

```
@media screen and (max-width: 480px) {
  body {
    background-color: #00ff00;
  }
}
```

Sempre que vamos fazer algo no media queries, temos antes que dizer que é para um media que estamos desenvolvendo para isso colocamos @media. Logo em seguida o tipo dessa media (no nosso caso foi o screen) e um operador lógico (no nosso caso foi o and). Entre os parênteses passamos o tamanho da largura que aquele media irá ser aplicado. Perceba que usamos o max, nesse caso estamos dizendo que será aplicado quando tiver 480 pixels de largura ou menos.

Vamos usar o site da Apple como um exemplo real de responsividade.





Veja como o site da Apple aplica responsividade em suas páginas. A grande sacada foi aumentar o tamanho dos card para quando chegar em um determinado tamanho para dispositivos mobile.

```
@media screen and (max-width: 480px) {
 .card {
  width: 100%;
 }
}
```

Uma coisa super importante é que você não precisa escrever todas as propriedades de uma class ou id dentro do seu @media, somente o que você quer que mude. Que no caso o tamanho de cada card terá o tamanho da resolução do dispositivo.

Observação: caso seja feito com flexbox você precisa colocar a propriedade flex-wrap: wrap;. Ela define se os itens ficam na mesma linha ou se podem ser quebrados em várias linhas. Que no caso dos card da Apple os card quebraram a linha e foram para baixo.

Padrões de resoluções

- Desktops: (min-width: 1281px)
- Laptops, desktops: (min-width: 1025px) and (max-width: 1280px)
- Tablets, ipads: (min-width: 768px) and (max-width: 1024px)
- Tablets de baixa resolução, celulares: (min-width: 481px) and (max-width: 767px)
- A maioria dos smartphones móveis: (min-width: 320px) and (max-width: 480px)

Fundamentos do JavaScript

O que é JavaScript

JavaScript, ou JS é uma linguagem de programação, sendo a principal linguagem client-side em navegadores web. Ela é responsável pelo dinamismo de uma página, ou seja, promove a interação com o usuário. Um exemplo clássico de JavaScript são os menus mobile, quando clicamos nele é feito uma interação com o usuário (no caso abrindo os itens do menu).

Iniciando

JavaScript é escrito em arquivos .js. Para criar um documento JavaScript é fácil, basta salvar o arquivo como script.js (lembro que você pode nomeá-lo como quiser, mas não esqueça do .js) dessa forma teremos nosso documento semipronto para o uso.

Agora iremos criar a conexão do JavaScript com nosso arquivo HTML. Antes do fechamento da tag <body> de seu documento HTML, vamos adicionar uma tag <script> para conectá-los. Isso é, na tag <script> atribuímos o atributo src="" (que vem do inglês source), esse atributo é responsável em fazer a conexão com outros arquivos.

O que são variáveis?

Variáveis são lugares na memória onde colocamos valores para podermos trabalhar com eles posteriormente. As variáveis são um dos fatores para mantermos o código dinâmico, fácil de ser lido e compreendido Isso é, uma vez armazenado um valor em uma memória podemos utilizar seu valor ao longo do nosso código.

Definindo uma variável

No JavaScript não há necessidade de declarar o tipo da variável, mas isso não significa que ela não tem tipo. Isso torna o JavaScript uma linguagem de tipagem dinâmica, o tipo é inferido pelo valor do dado e a checagem (type checking) é feita em tempo de execução

(runtime). Então se você tiver uma variável chamado “nome” com o valor de “iuri” e ao longo do nosso código mudar seu valor para 10, para o JavaScript isso está certo. Isso tem seus lados bons e ruins.

Por exemplo, essa é a declaração de uma variável:

```
const nome = "Iuri"; // Está correto
nome = 10; // Está correto também
// Resultado final de nome é 10
```

Observação: o // são comentários. Comentários são linhas de códigos não executáveis.

A barra dupla // serve para comentar uma linha de código.
Começa com /* e termina com */ podemos comentar várias linhas de código em JavaScript.

Ok Iuri, mas o que é esse const em nosso código?

Bem, para declarar uma variável temos três opções: var, let e const.

- Uso do var: Uma variável declarada com var possui o que chamamos de escopo de função. Isso significa que se criarmos uma variável deste tipo dentro de uma função, você pode modificar em qualquer parte desta função, mesmo se criar outra função dentro dela.

- Uso do let: Diferente de var, declarar como let leva em conta o bloco de código onde foi declarada. Isso significa que se a declararmos dentro de um uma função, ela será “enxergada” apenas dentro deste escopo.

- Uso do const: Uma variável const é usada para definir uma constante. Diferente de var e let, as variáveis de const não podem ser atualizadas nem declaradas novamente.

Que tal um exemplo?

var

```
function exibirNome() {
  var nome = "Iuri";
}
console.log(nome); // Erro: nome não está definido
```

Let

```
let nome = "Iuri";
if (true) {
  let nome = "Kenay";
  console.log(nome); // Retornará "Kenay"
}
console.log(nome); // Retornará "Iuri"
```

Por que isso não retorna um erro? Porque as duas instâncias são tratadas como variáveis diferentes, já que são de escopos diferentes.

const

```
const nome = "Iuri";
nome = "Kenay"; // Erro: atribuição a uma variável constante.
```

Assim como as declarações de let, os consts sofrem o hoisting para o topo do escopo, mas não são inicializadas.

String

Uma string é uma sequência de caracteres usados para representar texto. São declaradas usando aspas simples ou aspas duplas.

```
var nome = "Iuri";
// Ou
```

```
var nome = 'Iuri';
```

Number

Number é um tipo de dado numérico. O JavaScript não diferencia números inteiros, em ponto flutuante, double...

```
var numero = 10;
var pi = 3.14;
```

Boolean

Um booleano é um tipo de dado lógico que pode ter apenas um de dois valores possíveis: true (verdadeiro) ou false (falso).

```
var javascript = true;
var java = false;
```

Object

Objetos são tipos de dados especiais que podem armazenar vários valores ao mesmo tempo.

```
var pessoa = {
 nome: "Iuri",
 idade: 21,
 frontend: true,
};
```

Function

Uma função é um conjunto de instruções que executa uma tarefa ou calcula um valor.

```
function nome() {
 console.log("Iuri");
}
```

Undefined

Uma variável que não foi atribuída a um valor específico, assume o valor undefined (indefinido).

```
var texto
```

Null

O valor null é um valor nulo ou "vazio" (exemplo: que aponta para um objeto inexistente).

```
var texto = null;
```

Operadores de atribuição

“Um operador de atribuição atribui um valor ao operando à sua esquerda baseado no valor do operando à direita. O operador de atribuição básica é o igual (=), que atribui o valor do operando à direita ao operando à esquerda. Isto é, x = y atribui o valor de y a x.”
- MDN Web Docs

Operador	Operador encurtado
Atribuição	x = y
Atribuição de adição	x += y
Atribuição de subtração	x -= y
Atribuição de multiplicação	x *= y
Atribuição de divisão	x /= y
Atribuição de resto	x %= y

Operadores aritméticos

“Operadores aritméticos tomam valores numéricos como seus operandos e retornam um único valor numérico. Os operadores aritméticos padrão são os de soma (+), subtração (-),

multiplicação (*) e divisão (/). Estes operadores trabalham da mesma forma como na maioria das linguagens de programação quando utilizados com números”
- MDN Web Docs

Operador	Descrição
Módulo (%)	Operador binário. Retorna o inteiro restante da divisão dos dois operandos.
Incremento (++)	Operador unário. Adiciona um ao seu operando.
Decremento (--)	Operador unário. Subtrai um de seu operando.
Negação (-)	Operador unário. Retorna a negação de seu operando.
Adição (+)	Operador unário. Tenta converter o operando em um número, sempre que possível.
Operador de exponenciação (**)	Calcula a base elevada á potência do expoente, que é, base expoente

Operadores de comparação

“Um operador de comparação compara seus operandos e retorna um valor lógico baseado em se a comparação é verdadeira. Na maioria dos casos, se dois operandos não são do mesmo tipo, o JavaScript tenta convertê-los para um tipo apropriado. Isto geralmente resulta na realização de uma comparação numérica. ”
- MDN Web Docs

Operador	Descrição
Igual (==)	Retorna verdadeiro caso os operandos sejam iguais.
Não igual (!=)	Retorna verdadeiro caso os operandos não sejam iguais.
Estritamente igual (===)	Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo.
Estritamente não igual (!==)	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.
Maior que (>)	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.
Maior que ou igual (>=)	Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita.
Menor que (<)	Retorna verdadeiro caso o operando da esquerda seja menor que o da direita.
Menor que ou igual (<=)	Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita.

If

As estruturas condicionais são utilizadas para verificar uma condição e definir se algo deve ou não acontecer.

Podemos representar condições em JavaScript utilizando o comando de estrutura condicional if (se) da seguinte maneira:

```
if (condicao) {
  // O código será executado caso a condição seja verdadeira
}
```

Agora você deve estar se perguntando “Mas Iuri, e se essa condição for falsa?”. Nesse nosso exemplo, se a condição for falsa o fluxo irá continuar sem executar nosso código dentro do if.

Else

Agora, caso você queira executar algo quando a condição for falsa, utilizamos o else (se não). O else sempre será executado caso o if seja falso.

```
if (condicao) {
  // O código será executado caso a condição seja verdadeira
} else {
  // O código será executado caso a condição seja falsa
}
```

Else if

No contexto de estruturas condicionais....vamos imaginar que temos um sistema de verificação de nome, e você queria verificar apenas se dois nomes existem em nosso sistema. Você não concorda comigo que fazer dois if e else para verificar os dois nomes ficaria algo repetitivo? Ainda mais que nosso else iria retornar a mesma coisa nas duas condições.

O else if (se não se) nos ajuda a fazer mais do que uma condição, ou seja se a condição anterior não for verdadeira verifique uma nova condição.

Exemplo de else if

```
if (condicao) {
  // O código será executado caso a condição seja verdadeira
} else if (condicaoDois) {
  // O código será executado caso a primeira condição seja falsa
} else {
  // O código será executado caso todas as condições sejam falsas
}
```

O que é function?

Uma função, ou Functions é um conjunto de instruções que executa uma tarefa.

Imagine que você tenha duas tarefas, uma é executar a soma de dois números e a outra é mostrar o nome do usuário no sistema. Percebeu que essas tarefas são duas funções diferentes? Imagine essas duas tarefas espalhadas em nosso código, iria atrapalhar muito.

Definindo

A definição da função (também chamada de declaração de função) consiste no uso da palavra-chave function, seguida por:

- Nome da função.
- Lista de argumentos para a função, entre parênteses e separados por vírgulas.
- Declarações JavaScript que definem a função, entre chaves { }.

Por exemplo, o código a seguir define uma função simples chamada somar:

```
function somar(numero) {
  return numero * numero;
}
console.log(somar(10));
```

Veja que a função de somar recebe um argumento chamado numero. Isso significa que a nossa função está esperando o valor de numero (que no exemplo é 10) para multiplicar por si mesmo (numero * numero). A declaração return especifica o valor retornado pela função.

O que é Objeto?

Imagine que você tenha uma variável chamada pessoa e queira ter uma coleção de propriedades, como nome, idade, cidade... Isso é possível? Claro que é, isso é chamado de objeto. Objeto é uma coleção de propriedades, sendo cada propriedade definida como uma sintaxe de chave: valor. A chave pode ser uma string e o valor pode ser qualquer informação.

Por exemplo:

```
const pessoa = {
 nome: "Iuri",
 idade: 21,
};
```

Acessando as propriedades

Para acessar uma propriedade de um objeto, use uma das duas notações (sintaxes):

- Notação de ponto (objeto.propriedade).
- Notação de array (objeto[“propriedade”]).

```
pessoa.nome; //Resultado: Iuri
pessoa["nome"]; //Resultado: Iuri
```

O que é Array?

Arrays, ou matrizes são lista de objetos. Elas são objetos que contêm múltiplos valores armazenados em uma lista.

Se nós não tivéssemos arrays, teríamos que armazenar cada item em uma variável separada. Isso é, uma variável chamada, frutas1, outras chamadas fruta2, fruta3, fruta4.... imagine se fosse 100 frutas. Isto seria muito mais longo de escrever e acessar.

Criando um array

Arrays são construídas de colchetes, os quais contém uma lista de itens separada por vírgulas.
Vamos supor que queremos armazenar a nossa lista de frutas, nós temos o seguinte código.

```
var frutas = ["banana", "maçã", "uva", "kiwi"];
```

Acessando a lista

Você pode acessar itens individuais em um array usando a notação de colchetes, da mesma forma que você acessa as letras em uma string.

```
frutas[2]; // Resultado: "uva"
```

Observação: o índice do array começa sempre em 0.

Métodos de array

Arrays nos fornece vários métodos para facilitar nosso trabalho em certas situações. Com cada método que vamos abordar não estão disponíveis em objetos ou qualquer outra coisa além de Arrays. Veremos os seguintes métodos de Array (é bom lembrar que existem outros métodos de array):

- concat()
- join()
- push()
- pop()
- shift()
- slice()

- splice()
- reverse()

concat()

O método “concat()”, que serve para concatenar dois arrays, no exemplo vamos concatenar o array front com o array back.

```
var front = ["html", "css", "javascript"];
var back = ["java", "php", "node"];

front = front.concat(back);
console.log(front); // Resultado será um array com os elementos dois arrays
```

join()

O método “join()” puxa elementos de um array e lista no formato de string.

```
var front = ["html", "css", "javascript"];

front = front.join("-");
console.log(front); // Resultado será as propriedades do array separar com um traço
```

push()

O método “push()” serve para adicionarmos elementos no final do array.

```
var front = ["html", "css", "javascript"];

front.push("react");
console.log(front); // Resultado será um novo elemento no final do array
```

pop()

O método “pop()” remove o último elemento de um array.

```
var front = ["html", "css", "javascript"];

front.pop();
console.log(front); // Resultado será a remoção do último elemento do array
```

shift()

O método “shift()” remove o primeiro elemento do array.

```
var front = ["html", "css", "javascript"];

front.shift();
console.log(front); // Resultado será a remoção do primeiro elemento do array
```

reverse()

O método “reverse()” iverte a ordem dos elementos do array.

```
var front = ["html", "css", "javascript"];

front.reverse();
console.log(front); // Resultado será reversão dos elementos do array
```

unshift()

O método “unshift()” adiciona um elemento no início de um array.

```
var front = ["html", "css", "javascript"];

front.unshift('next');
console.log(front); // Resultado será um novo elemento no início do array
```

O que são Loops?

Laços de repetição, ou loops como o próprio nome já diz, ele faz com que blocos de códigos sejam repetidos diversas vezes até que certa condição seja cumprida.

Um loop geralmente possui um ou mais dos seguintes itens:

- O contador, que é inicializado com um certo valor - este é o ponto inicial do loop.
- A condição de saída, que é o critério no qual o loop para - geralmente o contador atinge um certo valor.
- Um iterador, que geralmente incrementa o contador em uma pequena quantidade a cada loop, sucessivamente, até atingir a condição de saída.

Loop for

O primeiro que você usará na maior parte do tempo, é o loop for, ele tem a seguinte sintaxe:

```
for (inicializador; condição - saída; expressão - final) {
  // Código para executar
}
```

No exemplo do for temos:

1. A palavra-chave for, seguido por parênteses.
2. Dentro do parênteses temos três itens, separados por ponto e vírgula:

- O inicializador— geralmente é uma variável configurada para um número, que é incrementado para contar o número de vezes que o loop foi executado. É também por vezes referido como uma variável de contador.

- A condição-saída — como mencionado anteriormente, aqui é definido quando o loop deve parar de executar. Geralmente, essa é uma expressão que apresenta um operador de comparação, um teste para verificar se a condição de saída foi atendida.

- A expressão-final — isso sempre é avaliado (ou executado) cada vez que o loop passou por uma iteração completa. Geralmente serve para incrementar (ou, em alguns casos, decrementar) a variável do contador, aproximando-a do valor da condição de saída.

for na prática

```
for (let i = 1; i <= 10; i++) {
 console.log(i); // Saída: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
}
```

Explicando: dentro do nosso laço for temos o “let i=1;” ele é o nosso inicializador, ele basicamente está falando que nossa variável de inicialização é o “i” que tem o valor de 1. Logo em seguida temos o “i<=10”, ele é o condição-saída, como o próprio nome já diz, ele é a condição que irá executar em nosso loop até que seja cumprida. Como o valor inicial de i é 1 ele será executado até seu valor ser igual a 10.
E por último temos o “i++”, ele é o expressão-final, ou seja, é o responsável pelo incremento da nossa variável i até chegar em 10.

Loop while

“while funciona de maneira muito semelhante ao loop for, exceto que a variável inicializadora é definida antes do loop, e a expressão final é incluída dentro do loop após o código a ser executado - em vez de esses dois itens serem incluídos dentro dos parênteses. A condição de saída está incluída dentro dos parênteses, que são precedidos pela palavra-chave while e não por for.” - MDN Web Docs

```
inicializador;
while (condição - saída) {
 // Código
 expressão - final;
}
```

Loop do ... while

```
inicializador;
do {
 // Código
 expressão - final;
} while (condição - saída);
```

Nesse caso, a condição-saída é avaliada depois que o bloco de código é executado, ou seja, o código dentro do ”do” será executado pelo menos uma vez antes de ter certeza que a condição-saída é verdadeira.

O que é DOM?

“O Document Object Model (DOM) é uma interface de programação para os documentos HTML e XML. Representa a página de forma que os programas possam alterar a estrutura do documento, alterar o estilo e conteúdo. O DOM representa o documento com nós e objetos, dessa forma, as linguagens de programação podem se conectar à página.” - MDN Web Docs

O que são eventos no JavaScript?

Os eventos são ações que são realizadas em um determinado elemento da página, seja ele um texto ou uma imagem, por exemplo. Muitas das interações do usuário que está visitando sua página com o conteúdo do seu site podem ser consideradas eventos.

Principais eventos

- onload: Disparado quando documento é carregado.
- onunload: Disparado quando documento é descarregado de janela ou de frame.
- onsubmit: Disparado quando formulário é submetido.
- onreset: Disparado quando formulário é "limpado" via botão de reset.
- onselect: Disparado quando texto é selecionado numa área de entrada de texto.
- onblur: Disparado quando elemento de entrada do formulário perde o foco, e quando texto fora do elemento é selecionado.
- onchange: Disparado quando elemento perde o foco e foi modificado.
- onclick: Disparado quando botão de formulário é selecionado via click do mouse.
- onfocus: Disparado quando o elemento recebe foco: clicando o mouse dentro do elemento ou entrando no mesmo via Tab.
- ondblclick: Disparado quando ocorre um click duplo do mouse.
- onmousedown: Disparado quando mouse é pressionado enquanto está sobre um elemento.
- onmouseup: Disparado quando mouse é despressionado.

- onmouseover: Disparado quando cursor do mouse é movido sobre elemento
- onmouseout: Disparado quando mouse é movido fora do elemento onde estava
- onkeydown: Disparado quando tecla é pressionada.
- onmousemove: Disparado quando mouse é movido enquanto sobre elemento.

Como usar os eventos

Existem diversas maneiras de se aplicar esses eventos aos elementos HTML, são elas:

- Inline.
- Em um arquivo externo, usando um manipulador de eventos.

Qual é melhor? Bem, usar a maneira de arquivo externo deixa nosso documento HTML limpo de código JavaScript. Já com inline, aplicamos diretamente na tag HTML o evento JavaScript.

```
<button onclick="alert('Oi')">Sou um evento inline</button>
```

O exemplo anterior está funcionando corretamente, aos clicarmos no botão irá disparar um evento chamado “onclick” que irá abrir uma simples caixas de diálogo (alert) porém, estamos adicionando JavaScript junto com código HTML. Irei reproduzir o mesmo evento em um arquivo JavaScript.

```
document.querySelector("button").addEventListener("click", () => {
  alert("oi");
});
```

Sei que olhando esse código parece que utilizar eventos inline é mais fácil e rápido, mas estamos utilizando só um exemplo, imagina uma aplicação cheia de animações e interações do usuário, você mesmo ficaria perdido e terá um código JavaScript maior.

Tenha calma que irei explicar cada parte desse código futuramente.

Selecionando elementos HTML

Seletores são métodos que selecionam tags HTML (e seu conteúdo), seja pelo id, class ou pela própria tag. Isso é muito útil quando queremos alterar a DOM, seja adicionando novos elementos ou alterando o que já existe nela. Veremos os seguintes métodos:

- getElementById()
- getElementsByClassName()
- querySelector()
- querySelectorAll()

Observação: todas as propriedades, métodos e eventos disponíveis para manipular e criar páginas são organizados em objetos, por exemplo, o objeto document representa o próprio documento.

getElementById

Este método retorna um elemento correspondente ao id passado como parâmetro. Veja o exemplo abaixo.

```
let eFront = document.getElementById("ebook");
```

Esse comando cria uma variável chamada eFront e armazena nessa variável o elemento cujo id é igual ao id passado como parâmetro. Caso este id não exista dentro do código HTML, o retorno da função é null.

getElementsByClassName

Como o próprio nome já diz, essa função vai retornar os elementos que possuem uma mesma classe passada.

```
let eFront = document.getElementsByClassName("ebook");
```

Enquanto a função anterior retorna um único elemento, está retorna uma HTMLCollection (uma coleção) de elementos.

querySelector

Diferente dos outros métodos, este método utiliza “.” para indicar a seleção de uma classes, “#” para indicar seleção de ids ou a própria tag.

```
let teste1 = document.querySelector("#souId");
let teste2 = document.querySelector(".souClass");
let teste3 = document.querySelector("div");
```

Um ponto interessante do querySelector() só retorna o primeiro elemento encontrado que tem a mesma seleção.

querySelectorAll

Este método é similar ao método querySelector que também utiliza os seletores do CSS, porém retorna uma NodeList com todos os elementos que correspondem ao seletor criado.

```
let teste = document.querySelectorAll("div"); // Irá retornar todas as divs da página
```

Interações do usuário

Existem duas formas comuns (existem outras formas) de adicionar estilos CSS em nossa página com JavaScript:

- Estilos Inline.
- Classes CSS a elementos.

A propriedade style em JavaScript retorna o estilo de um elemento na forma de um objeto CSSStyleDeclaration que contém uma lista de todas as propriedades de estilos. Dessa forma conseguimos adicionar estilos CSS em nossa aplicação através do JavaScript.

Adicionando estilos inline com JavaScript

```
var button = document.querySelector("button");
```

```
button.addEventListener("click", () => {
  button.style.backgroundColor = "red";
});
```

Perceba que diferente das propriedades do CSS, no JavaScript não temos o “-” (hífen) e substituímos ele pela convenção de nomenclatura lowerCamelCase.

Dessa forma adicionamos background-color do CSS em nosso botão de forma de estilos inline. Mas veja que isso leva a um problema que tivemos com o JavaScript dentro do HTML.

Adicionando classes CSS a elementos

Utilizando a propriedade classList você pode obter, definir ou remover classes CSS facilmente de um elemento. O exemplo a seguir mostrará como adicionar uma classe chamada “outraCor” em um elemento com id="iuricode".

```
var elementoDiv = document.getElementById("iuricode");

elementoDiv.addEventListener("click", () => {
  elementoDiv.classList.toggle("outraCor");
});
```

Uma observação: eu criei uma classe no CSS que tem a propriedade background-color, assim quando ela for chamada através do click, irá mudar a cor do elemento selecionado (que no caso é a div com o id=”iuricode”).

addEventListener

O método addEventListener (ou conhecido como evento de escuta) permite configurar funções a serem chamadas quando um evento acontece, que em nosso exemplo, quando um usuário clica em um botão. No exemplo a seguir mostrar que o elemento nomeElemento está sendo escutado caso o usuário faça um click.
Sintaxe do addEventListener

```
alvo.addEventListener(evento, função);
```

- alvo: é o elemento HTML ao qual você deseja adicionar os eventos.
- evento: é onde você especifica qual é o tipo de evento que irá disparar a ação/função. Uma coisa interessante no evento é que diferente do JavaScript no HTML, ele não tem o “on” dos eventos como onclick, onchange, onblur...
- função: específica a função a ser executada quando o evento é detectado. Em nossos exemplos foram utilizados a sintaxe arrow function que deixa a expressão da nossa função mais curta e anônima. Mas você pode simplesmente aplicar uma outra função código no lugar.

```
var nomeElemento = document.getElementById("div");

nomeElemento.addEventListener("click", () => {
  ...
});
```

O arrow function não terá uma explicação mais afundo pois ele é um recursos introduzido no ES6 e estamos apenas abordando os fundamentos.

Consumindo API com Axios

O que é API?

API, ou Application Programming Interface, é um conjunto de definições e protocolos para integrar softwares de aplicações. Um exemplo clássico de API é a do Google Maps. Por meio de seu código, conseguimos ter acesso a localização, muitas outras aplicações utilizam os dados do Google Maps adaptando-o da melhor forma.

O que é Axios?

Axios é uma biblioteca JavaScript baseada em promessas (promises) para interceptar requisições (requests) HTTP. Esses interceptadores são úteis quando queremos pegar ou alterar requisições HTTP. Cada requisição HTTP pode usar um dos métodos de requisição existente, temos sete tipos de requisição, mas nesse ebook iremos abordar somente o método GET para consultar os dados em uma API.

Adicionando o Axios

A primeira coisa a se fazer quando estamos trabalhando com uma biblioteca ou framework, é procurar o CDN da biblioteca para importa-lá em nossa aplicação.

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script src=”main.js”></script>
```

Dessa forma temos acesso aos recursos do Axios.

Uma observação importante: a importação do CDN tem que vir antes do nosso arquivo script que iremos utilizar para consumir a API (isso se aplica a qualquer biblioteca).

Adicionando a API

Em nosso exemplo iremos utilizar a API de usuário do GitHub.

```
axios.get("https://api.github.com/users/iuricode");
```

Primeiro, você importa o Axios para que possa ser usado na aplicação. Em seguida, executa uma requisição utilizando o GET através do método get() para obter uma promessa que retorna um objeto de resposta que pode dar sucesso ou erro, dependendo da resposta da API.

```
axios
 .get("https://api.github.com/users/iuricode")
 .then(function (response) {
  console.log(response.data);
  console.log(response.data.login);
 })
 .catch(function (error) {
  console.log(error);
 });
```

Ok Iuri, mas o que é esse then?

Lembra que eu falei que o Axios trabalha com promessas? Exatamente isso que está acontecendo! Caso a promessa for cumprida, o argumento then() será chamado; se a promessa for rejeitada, o argumento catch() será chamado.

Dentro do argumento then() estamos imprimindo duas coisas no console do navegador: response.data e response.data.login.
O response.data está imprimindo todos os dados da nossa API no console, já o response.data.login está imprimindo o nosso username do GitHub.

Axios é a única solução?

Não! Não precisamos do Axios para consumir uma API, temos nativamente no JavaScript o fetch(). Ele é perfeitamente capaz de reproduzir as principais funcionalidades do Axios! Mas o Axios é uma biblioteca que traz bastante benefícios quando estamos trabalhando com APIs.