

# Unidade 3

## Desenvolvendo Aplicativos Móveis com IONIC



# Introdução

Neste módulo veremos como preparar nosso ambiente de desenvolvimento para trabalhar com IONIC no desenvolvimento de aplicativos móveis, além de abordar uma pequena introdução às diversas tecnologias que vamos usar, como HTML, CSS, SCSS, JavaScript, TypeScript, Angular e Ionic.

## Objetivos

- Entender os conceitos básicos de HTML, CSS e SCSS;
- Entender os conceitos básicos de JavaScript e TypeScript;
- Usar ferramentas de desenvolvedor no Chrome para depurar e testar códigos;
- Preparar nosso ambiente de trabalho para o desenvolvimento de aplicativos móveis usando Ionic;
- Usar uma IDE de desenvolvimento (*Visual Studio Code*) para ganhar produtividade;
- Criar nosso primeiro projeto básico em Ionic.

## Conteúdo programático

Aula 01 – Introdução às tecnologias e preparando nosso ambiente de trabalho

Aula 02 – Preparando o ambiente e criando nosso aplicativo IONIC



Você poderá também **assistir às videoaulas** em seu celular! Basta apontar a câmera para os **QR Codes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no celular

# Aula 01 - Introdução às tecnologias e preparando nosso ambiente de trabalho

Olá pessoal, nesta aula vamos abordar de maneira simplificada uma série de novas tecnologias e linguagens de programação que passaremos a usar para desenvolver nossos aplicativos móveis.

Antes de iniciar, é importante saber que estas não são as únicas tecnologias que podem ser usadas para desenvolver aplicativos móveis, existem centenas de opções diferentes que podem ser usadas para isso, cada uma delas têm pontos positivos e negativos, não existe uma opção perfeita para todos os projetos. Para definir as tecnologias que serão usadas em um projeto, diversos fatores devem ser considerados, como as características da aplicação que será criada, a necessidade de integração com soluções já existentes, quais são os recursos usados no dispositivo móvel, se a aplicação deve ou não funcionar em plataformas diferentes, como Android e IOs (*iPhone*), o conhecimento da equipe envolvida no projeto, entre muitos outros fatores.

No nosso caso, a escolha desse conjunto específico de tecnologias levou em consideração alguns fatores, como:

- Apresentar um conjunto de tecnologias que propicie uma fácil adaptação para outros modelos de desenvolvimento, como o de aplicações para *web*, por exemplo;
- Apresentar tecnologias que são usadas no mercado de trabalho atualmente;
- Apresentar tecnologias que serão úteis em diversos projetos ao longo do curso e da carreira profissional;
- Apresentar tecnologias multiplataforma que possam ser usadas para criar aplicativos tanto para Android quanto para IOs;
- Usar um ambiente de desenvolvimento (trabalho) que funcione com recursos de máquina limitados (computadores mais simples);
- Usar ambientes de desenvolvimento que funcionem em Windows, Linux e Mac.

Por estes e diversos outros motivos foi escolhido o Ionic Framework, baseado em Angular para embasar o conjunto das tecnologias que serão usadas, são elas:

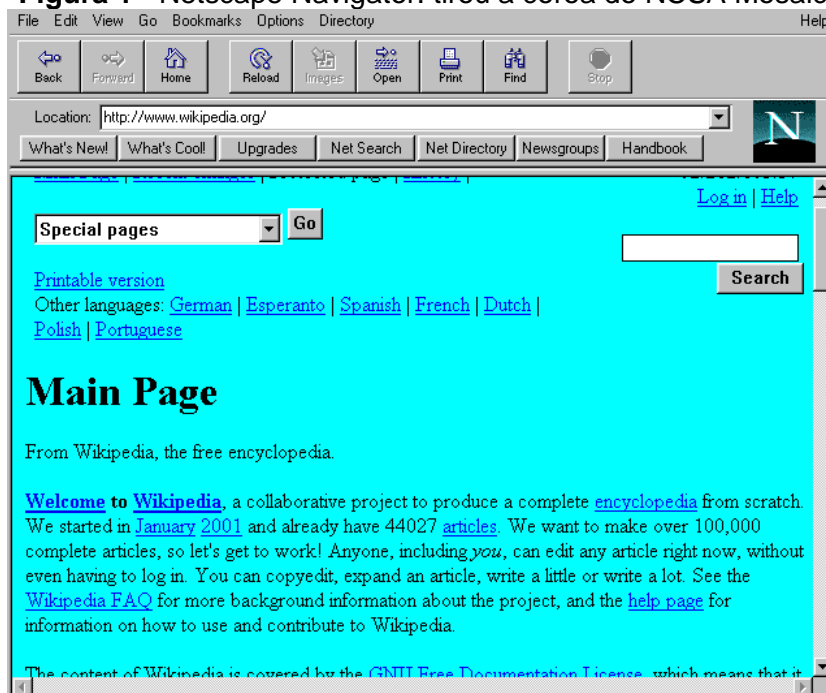
- **HTML** - abreviação para a expressão inglesa *HyperText Markup Language*, que significa: "Linguagem de Marcação de Hipertexto" é uma linguagem de marcação utilizada na construção de páginas na *Web*. Documentos HTML podem ser interpretados por navegadores. A tecnologia é fruto da junção entre os padrões HyTime e SGML. (Veja mais em: <https://pt.wikipedia.org/wiki/HTML>);
- **CSS** - *Cascading Style Sheets* é um mecanismo para adicionar estilo a um documento *web*. O código CSS pode ser aplicado diretamente nas *tags* ou ficar contido dentro das tags `<style>`. Também é possível, em vez de colocar a formatação dentro do documento, criar um link para um arquivo CSS que contém os estilos. (Veja mais em: [https://pt.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://pt.wikipedia.org/wiki/Cascading_Style_Sheets));
- **Sass e SCSS** - Sass é uma linguagem de folhas de estilo concebida inicialmente por Hampton Catlin e desenvolvida por Natalie Weizenbaum. Depois de suas versões iniciais, Weizenbaum e Chris Eppstein continuaram a estender Sass com SassScript, uma simples linguagem de script usada em arquivos Sass. A sintaxe mais recente, "SCSS", usa formatação de bloco, como a de CSS. Esta usa chaves para designar blocos de código e ponto e vírgula para separar linhas dentro de um bloco. Os arquivos com sintaxe de indentação e SCSS são tradicionalmente dados as extensões ".sass" e ".scss", respectivamente. (Veja mais em: [https://pt.wikipedia.org/wiki/Sass\\_\(linguagem\\_de\\_folhas\\_de\\_estilos\)](https://pt.wikipedia.org/wiki/Sass_(linguagem_de_folhas_de_estilos)));
- **JavaScript** - É uma linguagem de programação interpretada estruturada, de script em alto nível com tipagem dinâmica fraca e multiparadigma. Juntamente com HTML e CSS, o JavaScript é uma das três principais tecnologias da *World Wide Web*. (Veja mais em: <https://pt.wikipedia.org/wiki/JavaScript>);
- **TypeScript** - É um superconjunto de JavaScript desenvolvido pela Microsoft que adiciona tipagem e alguns outros recursos a linguagem. Anders Hejlsberg, arquiteto da linguagem C# e criador das linguagens Delphi e Turbo Pascal, trabalhou no desenvolvimento do TypeScript. (Veja mais em: <https://pt.wikipedia.org/wiki/TypeScript>);
- **Angular** - É uma plataforma de aplicações web de código-fonte aberto e *front-end* baseado em TypeScript liderado pela Equipe Angular do Google e por uma comunidade de indivíduos e corporações. Angular é uma reescrita completa do AngularJS, feito pela mesma equipe que o construiu. (Veja mais em: [https://pt.wikipedia.org/wiki/Angular\\_\(framework\)](https://pt.wikipedia.org/wiki/Angular_(framework)));

- **Ionic** - É um SDK de código aberto completo para desenvolvimento de aplicativo móvel híbrido criado por Max Lynch, Ben Sperry e Adam Bradley da Drifty Co. em 2013. A versão original foi lançada em 2013 e construída sobre AngularJS e Apache Cordova. (Veja mais em: [https://en.wikipedia.org/wiki/Ionic\\_\(mobile\\_app\\_framework\)](https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework))).

## Curiosidades

O JavaScript foi criado na década de 90, por Brendan Eich, a serviço da *Netscape*. Essa década foi um período de revolução, pois os *browsers* ainda eram estáticos.

**Figura 1** - Netscape Navigator: tirou a coroa do NCSA Mosaic



Fonte: disponível em: <http://shipit.resultadosdigitais.com.br/blog/javascript-1-uma-breve-historia-da-linguagem/>. Acesso em: 20 abr. 2021.

Veja mais informações em: <https://pt.wikipedia.org/wiki/JavaScript>, [https://pt.wikipedia.org/wiki/Hist%C3%B3ria\\_das\\_linguagens\\_de\\_programa%C3%A7%C3%A3o](https://pt.wikipedia.org/wiki/Hist%C3%B3ria_das_linguagens_de_programa%C3%A7%C3%A3o) e <https://pt.wikipedia.org/wiki/ECMAScript>

Para que seja possível criar projetos usando todas estas tecnologias é necessário um ambiente de compilação e execução dos nossos códigos. Como a base de programação escolhida para trabalhar é o JavaScript e TypeScript, o ambiente mais adequado para isso é o Node.js, que usa o “npm” como gerenciador de pacotes:

- **Node.js** - É um *software* de código aberto, multiplataforma, que executa códigos JavaScript no *back-end/servidor* e *front-end/interface*, baseado no V8 interpretador de JavaScript em C++ do Google, mantido pela fundação Node.js em parceria com a Linux Foundation. (Veja mais em: <https://pt.wikipedia.org/wiki/Node.js>);
- **npm** - É um gerenciador de pacotes para a linguagem de programação JavaScript. npm, Inc. é uma subsidiária do GitHub, que fornece hospedagem para desenvolvimento de *software* e controle de versão com o uso do Git. npm é o gerenciador de pacotes padrão para o ambiente de tempo de execução JavaScript Node.js. (Veja mais em: [https://pt.wikipedia.org/wiki/Npm\\_\(software\)](https://pt.wikipedia.org/wiki/Npm_(software)));
- **GitHub** - É uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos privados e/ou *Open Source* de qualquer lugar do mundo. GitHub é amplamente utilizado por programadores para divulgação de seus trabalhos ou para que outros programadores contribuam com o projeto, além de promover fácil comunicação através de recursos que relatam problemas ou mesclam repositórios remotos (issues, pull request). (Veja mais em: <https://pt.wikipedia.org/wiki/GitHub>);
- **Git** - É um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de *software*, mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo (Exemplo: alguns livros digitais são disponibilizados no GitHub e escrito aos poucos publicamente). O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos. (Veja mais em: <https://pt.wikipedia.org/wiki/Git>).

## Curiosidades

Como muitas coisas na vida, o Git começou com um pouco de destruição criativa e uma controvérsia de fogo. No início, o código fonte do Linux usava o BitKeeper (<http://www.bitkeeper.org/>) para gerenciar as versões do código, porém em 2005 esta ferramenta passou a ser paga, o que motivou Linux Torvalds a desenvolver a sua própria ferramenta baseada em lições aprendidas ao usar o BitKeeper. Em 2005 mesmo, a primeira versão do Git ficou disponível e evoluiu até hoje.

**Figura 2** - TED - Linus Torvalds about Linux and Git - 2016



Fonte: disponível em: <https://www.youtube.com/watch?v=Vo9KPk-ggKk>. Acesso em: 20 abr. 2021.

Veja mais informações em: <https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Uma-Breve-Hist%C3%B3ria-do-Git>

Para editar os códigos criados nestas linguagens de programação precisaremos de um Editor de Código ou IDE (*Integrated Development Environment*). Existem diversas IDEs que podem ser usadas para o desenvolvimento de aplicativos móveis, com diversos recursos diferentes, como IntelliJ IDEA, Android Studio, Eclipse, Netbeans, Zend Studio, etc. (Veja mais em: [https://pt.wikipedia.org/wiki/Ambiente\\_de\\_desenvolvimento\\_integrado](https://pt.wikipedia.org/wiki/Ambiente_de_desenvolvimento_integrado)), além de existirem bons editores de código, como Sublime Text, Atom, Notepad++, TextMate, Vim, UltraEdit, Coda, Brackets, CodeShare, etc. ([https://pt.wikipedia.org/wiki/Editor\\_de\\_texto](https://pt.wikipedia.org/wiki/Editor_de_texto)).



Considerando às tecnologias que vamos usar e a necessidade de um ambiente de trabalho que funcione em diversos sistemas operacionais, sem exigir muitos recursos do equipamento (computador), optamos pelo Visual Studio Code, que, apesar de ser classificado como editor de código-fonte, apresenta alguns recursos encontrados nas melhores IDEs:

- **Visual Studio Code** - É um editor de código-fonte desenvolvido pela Microsoft para Windows, Linux e macOS. Ele inclui suporte para depuração, controle Git incorporado, realce de sintaxe, complementação inteligente de código, *snippets* e refatoração de código. (veja mais em: [https://pt.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://pt.wikipedia.org/wiki/Visual_Studio_Code)).

Finalmente, para trabalhar com todas estas ferramentas e efetivamente construir nossos aplicativos móveis, teremos que usar um computador que opera através de um Sistema Operacional (veja mais em: [https://pt.wikipedia.org/wiki/Sistema\\_operativo](https://pt.wikipedia.org/wiki/Sistema_operativo)).

Existem diversos tipos de sistemas operacionais destinados a funcionalidades específicas, porém, nos computadores pessoais (*desktops*) três deles são os mais usados **Windows** ([https://pt.wikipedia.org/wiki/Microsoft\\_Windows](https://pt.wikipedia.org/wiki/Microsoft_Windows)), **Linux** (<https://pt.wikipedia.org/wiki/Linux>) e **MacOS** (<https://pt.wikipedia.org/wiki/MacOS>).

A quantidade de equipamentos rodando cada um desses sistemas operacionais têm variado ao longo dos anos, mas, dependendo do público alvo, enquanto o Windows ainda domina o mercado para uso pessoal, seguido por MacOS e Linux, nos servidores Web esta lógica se inverte. A maioria dos serviços que acessamos na internet estão rodando em um sistema operacional Linux (em suas diversas distribuições), por este motivo, a adoção do Linux também no *desktop* em empresas de tecnologia é cada vez maior.

Como profissionais da computação, é importante que vocês conheçam ao menos estes três sistemas operacionais, pois certamente vão se deparar com projetos que devem funcionar com um ou mais deles.

Já quando falamos nos sistemas operacionais para dispositivos móveis tem quase que exclusivamente o **Android** (<https://pt.wikipedia.org/wiki/Android>) e o **IOS** (<https://pt.wikipedia.org/wiki/IOS>).

Em nossas aulas teremos que usar um *Prompt* de Comandos (ou linha de comando) em modo textual para executar diversas operações. Esta é uma característica de muitos ambientes de desenvolvimento de sistemas, assim, vamos ver as opções para cada SO:



- **Linux Terminal (GNU Bash)** (Para quem usa Linux) - É um interpretador de comandos, um entre os diversos tradutores entre o usuário e o sistema operacional conhecidos como shell. Acrônimo para "Bourne-Again SHell"[2], o Bash é uma evolução retro compatível muito mais interativa do Bourne Shell (sh). Os shells Bourne permitem a execução de sequências de comandos inseridos diretamente na linha de comandos ("*prompt*") ou ainda lidos de arquivos de texto conhecidos como shell scripts. Neles, podem invocar comandos internos aos próprios shells (conhecidos como comandos "*builtins*") ou ainda arquivos binários ou *scripts* de terceiros arquivados nos dispositivos de armazenamento do sistema. (Veja mais em: <https://pt.wikipedia.org/wiki/Bash>).
- **Windows CMD** (Para quem usa Windows) - *Prompt* de Comando é um interpretador de linha de comando no OS/2 e de sistemas baseados no Windows NT. Ele é um comando análogo ao command.com do MS-DOS e de sistemas Windows 9x, ou de shells utilizados pelos sistemas Unix. (Veja mais em: <https://pt.wikipedia.org/wiki/Cmd.exe>).
- **Terminal.app** (Para quem usa macOS) - É o emulador de terminal incluído no sistema operacional macOS da Apple. O Terminal teve origem no NeXTSTEP e OPENSTEP, os sistemas operacionais predecessores do macOS. (Veja mais em: [https://pt.wikipedia.org/wiki/Terminal\\_\(macOS\)](https://pt.wikipedia.org/wiki/Terminal_(macOS))).

Em nossas videoaulas usaremos o Linux, especificamente o UBUNTU que executo em menu *Notebook*. Tenho apenas o Linux instalado em meu computador pessoal desde 2007 e funciona muito bem para mim. Porém, praticamente todos os comandos que vamos usar são muito parecidos, seja em Windows, Linux ou MacOS.

## HTML - Introdução

Ao acessar uma página web através de um navegador, ele é capaz de interpretar o código HTML e renderizá-lo de forma compreensível para o usuário final, exibindo textos, botões etc., com as configurações definidas por meio das diversas *tags* que essa linguagem dispõe.

Atualmente, a HTML *HyperText Markup Language* encontra-se na versão 5 e é padronizada pelo W3C (*World Wide Web Consortium*), (<https://www.w3schools.com/html/>)

uma organização internacional responsável por estabelecer padrões para a internet, como a linguagem XML, CSS e o SOAP.

Em HTML, a marcação (*markup*) é realizada através das *tags* delimitadas por “<” e “>”, a *tag* é iniciada por um nome padrão, e terminada pelo mesmo nome, porém com uma barra “/” após o elemento “<” por exemplo, a *tag* “html” inicia com <html> e é terminada com </html>, a *tag* “head” inicia com <head> e é terminada com </head>, outros exemplos <p> ... </p>, <h1> ... </h1>, <h2> ... </h2>, etc.

## Videoaula 1

Agora assista à videoaula que aborda um pouco sobre o HTML.

Disponível em: [https://www.youtube.com/watch?v=rM\\_hZUP9ynY](https://www.youtube.com/watch?v=rM_hZUP9ynY). Acesso em: 20 abr. 2021.



### Videoaula 1

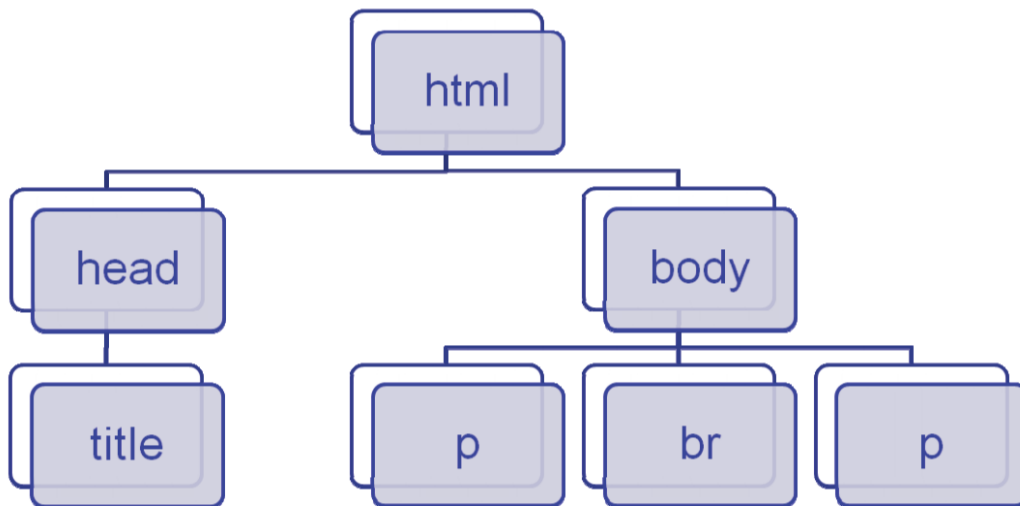
Utilize o QRcode para assistir!

Agora assista à videoaula que aborda um pouco sobre o HTML.



Os documentos HTML são estruturados e organizados em uma árvore de elementos que inicia com o “elemento HTML” (<html>), chamado de raiz da árvore de elementos.

**Figura 3** - Exemplo da árvore de elementos HTML



Fonte: adaptado de W3schools (2021)

Os arquivos que contêm documentos HTML têm a extensão “.html”. Estes arquivos podem conter *tags* html respeitando a estrutura definida, além de delimitadores como espaços, tabulações ou mudanças de linha, exemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Minha primeira página </title>
  </head>
  <body>
    <p> Olá mundo </p>
    <hr>
    <p> Olá a todos! </p>
  </body>
</html>
```

Um documento HTML possui a seguinte estrutura geral:

- Instrução DOCTYPE (identif. do tipo de documento);
- Elemento html (descrição do documento html);
- Elemento *head* (cabeçalho do documento);

- Elemento *body* (corpo do documento).

Instrução DOCTYPE:

- DOCTYPE não é uma *tag*, mas uma instrução;
- Deve ser a primeira linha do código;
- Indica qual a especificação deve ser utilizada para interpretar o documento;
- HTML5.

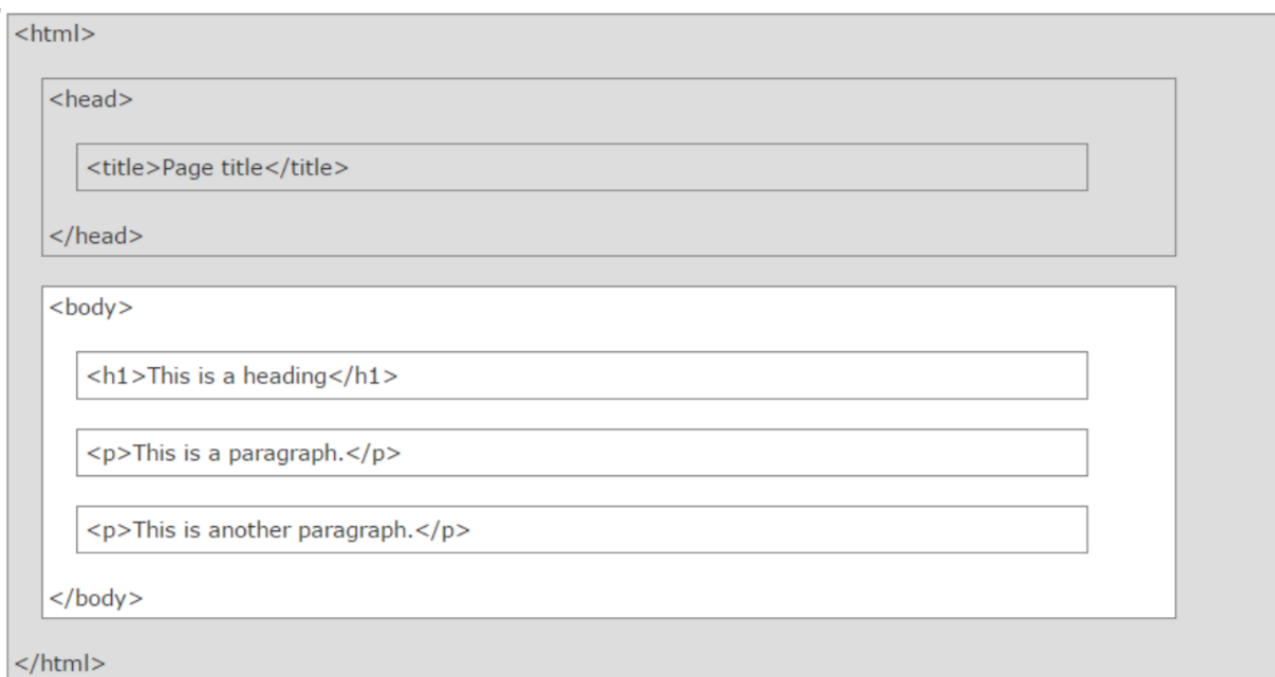
```
<!DOCTYPE html>
```

- HTML4:

```
<!DOCTYPE HTML PUBLIC  
"-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

Somente o elemento `<body>` é apresentado, pelo *browser*.

**Figura 4 – HTML**



Fonte: o autor (2021)

## Existem três tipos de elementos HTML:

- **Raw text** - Composto um conteúdo de texto entre *tags* html, por exemplo: `<tag>`  
conteúdo de texto `</tag>`
  - *Start tag* - `<tag>` - Delimitador do início do elemento. Pode incorporar qualquer quantidade de atributos HTML;
  - Conteúdo de texto (conteúdo de texto). Texto a ser apresentado, não permite a inclusão de outros elementos;
  - *End tag* - `</tag>` - delimitador do término do elemento;
  - Exemplo: `<p>Este é um parágrafo. </p>`.
- **Normal** - Composto geralmente por: *Start tag*, conteúdo e *end tag*, por exemplo: `<tag>` conteúdo `</tag>` sendo que:
  - *Start tag* - `<tag>`- pode incorporar qualquer quantidade de atributos HTML;
  - Conteúdo (conteúdo) - pode conter texto ou outros elementos;
  - *End tag* - `</tag>`- pode ser omitido em algumas situações.
  - Exemplo:

```
<form>
First name: <br>
<input type="text" name="firstname"> <br>
</form>
```

- **Void** - composto por somente por um *start tag*, não possui conteúdo e *end tag*, por exemplo `<tag>`, não contém elementos filhos, pode conter vários atributos. Alguns representam âncoras de localização de documentos externos, como imagens e especificações.
  - Exemplos:

```
<br>

<link rel="stylesheet" href="fancy.css" type="text/css" />
```

## Principais elementos HTML

```
<html>
<head> .... </head>
<body> .... </body>
</html>
```

Elemento “parágrafo” `<p> ... </p>`, usado dentro do elemento *body*:

```
<p> Bla bla bla bla </p>
<p> Bla bla bla bla </p>
```

Sequência de espaços, tabulações e mudança de linha na apresentação da página são tratados como um único espaço.

Exemplo:

```
<p> Este parágrafo contém
vários espaços,          tabulações e mudanças de linha,
                        mas o browser irá          ignorá-los.
</p>
```

Elemento “quebra de linha” (`<br>`), usado dentro do elemento *body*:

```
<br>
<p> Parágrafo com <br> quebra de linha </p>
```

Elemento “Linha divisória horizontal” (`<hr>`):

```
<p> Parágrafo antes da divisória </p>
<hr>
<p> Parágrafo depois da divisória </p>
```

Comentário (`<!-- ... -->`):

```
<!-- Este é um comentário -->
```

Exemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Exemplo de parágrafo </title>
  </head>
  <body>
    <p> Este é um parágrafo. </p>
    <br>
    <p> Este é outro parágrafo. </p>
    <hr>
    <p> Este é mais um parágrafo. </p>
  </body>
</html>
```

Elemento “título” (`<title>...</title>`) do documento é um metadado (não é apresentado na página), deve estar presente somente no elemento *head* (`<head>...</head>`) e deve ocorrer no máximo uma vez.

Elemento título de seções (`<h1>...</h1>`), (`<h2>...</h2>`) ... (`<h5>...</h5>`)

```
<body>
  <h1> Heading1 </h1>
  <h2> Heading2 </h2>
  <h3> Heading3 </h3>
  <h4> Heading4 </h4>
  <h5> Heading5 </h5>
</body>
```

Elemento de lista (`<ul>...</ul>`) (`<li>...</li>`) (*unorded list*) cria uma lista não ordenada de itens, cada item da lista é apresentado em uma linha, separada com *bulet*.

```
<ul>
  <li> UOL </li>
  <li> Folha </li>
  <li> Estadão </li>
```



```
</ul>
```

Elemento de lista ordenada (`<ol>...</ol>`) (`<li>...</li>`) (*orded list*) cria uma lista **ordenada** de itens, cada item da lista é apresentado em uma linha, separada com numeral (1, 2, 3 ...).

```
<ol>
  <li> UOL </li>
  <li> Folha </li>
  <li> Estadão </li>
</ol>
```

Elemento “imagem” (`<img ... >`) permite incluir um objeto tipo imagem à página, possui alguns atributos relevantes:

- Arquivo fonte (`src`);
- Texto alternativo (`alt`);
- Largura (`width`);
- Altura (`height`).

Exemplo:

```

```

Elemento link (`<a href="...">...</a>`) é um metadado que expressa relacionamento entre documentos, permite definir um objeto (texto ou imagem) que, quando selecionado, busca outro documento, possui alguns atributos relevantes:

- Atributo (`href`) (*hyperlink reference*): permite identificar a URL do documento a ser buscado.

Exemplo:

```
<a href="http://www.unifil.br">UniFil</a>
```

Praticamente todos os elementos HTML podem conter atributos que fornecem informações adicionais sobre um determinado elemento HTML. São sempre especificados no *tag* inicial do elemento, por exemplo: `<tag atributo1="valor1" atributo2="valor2" >...</tag>`

O atributo (`title`) permite definir um título a um elemento, este título não é apresentado, aparece na forma de um *tooltip* somente quando o mouse passa sobre o elemento.

Exemplo:

```
<p title="Threads">
Threads são linhas de execução sobre um determinado espaço de
endereçamento
</p>
```

Existem diversos atributos padronizados com comportamento pré-definido, como (`title`), (`href`), (`src`), (`alt`), etc. A definição dos atributos aceitos por padrão por cada *tag* HTML está na especificação do padrão da linguagem de marcação (<https://www.w3schools.com/html/>), porém um atributo presente em praticamente todas as *tags* que geram algum tipo de apresentação é o atributo (`style`).

O atributo (`style`) permite alterar o estilo padrão dos elementos HTML, como cor de fundo, cor do texto, tipo de fonte, tamanho, bordas, entre outros ([https://www.w3schools.com/html/html\\_styles.asp](https://www.w3schools.com/html/html_styles.asp)).

Neste exemplo, a cor do texto é alterada para vermelho:

```
<h2 style="color:red" >
Subtítulo HTML vermelho
</h2>
```

O atributo (`style`) possui uma relação direta com nosso próximo assunto neste módulo, os arquivos (.css). Um elemento HTML pode ter seu estilo alterado pelo atributo (`style`) ou por classes incluídas em folhas de estilo (CSS).

## Dica de leitura

CURSO DE HTML5 da <https://www.w3c.br/>. Os cursos de HTML5 do W3C Brasil visam apresentar as novidades que vêm sendo discutidas no WG HTML5 e que estão sendo implementadas na nova versão.

Disponível em: <https://www.w3c.br/Cursos/CursoHTML5>. Acesso em: 20 abr. 2021.

## CSS - Introdução

O CSS formata a informação entregue pelo HTML. Essa informação pode ser qualquer coisa: imagem, texto, vídeo, áudio ou qualquer outro elemento criado. Grave isso: CSS formata a informação.

Essa formatação na maioria das vezes é visual, mas não necessariamente. No CSS Aural, nós manipulamos o áudio entregue ao visitante pelo sistema de leitura de tela. Nós controlamos volume, profundidade, tipo da voz ou em qual das caixas de som a voz sairá. De certa forma você está formatando a informação que está em formato de áudio e que o visitante está consumindo ao entrar no site utilizando um dispositivo ou um sistema de leitura de tela. O CSS prepara essa informação para que ela seja consumida da melhor maneira possível.

## Videoaula 2

Agora assista esta videoaula que aborda um pouco sobre CSS na criação de páginas HTML.

Disponível <https://www.youtube.com/watch?v=BaPWRdUZQno>. Acesso em: 20 abr. 2021.



## Videoaula 2

Utilize o QRcode para assistir!

Agora assista esta videoaula que aborda um pouco sobre CSS na criação de páginas HTML.



Basicamente, para trabalhar com HTML + CSS, podemos usar três formas:

**CSS externos** - arquivos contendo a especificação CSS que são incluídos no HTML por meio de uma *tag* (`<link rel="stylesheet" href="meuArquivoDeEstilos.css">`).

Exemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="mystyle.css">
  </head>
  <body>
    <h1>Título de seção</h1>
    <p>Este é um parágrafo.</p>
  </body>
</html>
```

Arquivo (`mystyle.css`):

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}
```

**CSS internos** - definição das classes CSS dentro do arquivo HTML, na seção (**head**) tag (**style**), exemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-color: linen;
      }
      h1 {
        color: maroon;
        margin-left: 40px;
      }
    </style>
  </head>
  <body>
    <h1>Título de seção</h1>
    <p>Este é um parágrafo.</p>
  </body>
</html>
```

**CSS em linha** (*inline*) - Neste modelo, os estilos são definidos dentro do atributo (**style**) dos elementos.

```
<!DOCTYPE html>
<html>
  <body>
    <h1 style="color:blue;text-align:center;">Título de
seção</h1>
    <p style="color:red;">Este é um parágrafo.</p>
  </body>
</html>
```

Quando usamos CSS externos ou internos temos que identificar o elemento que deve ser formatado, para isso usamos seletores, e dentro do seletor incluímos as propriedades e valores, exemplo:

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

A propriedade é a característica que você deseja modificar no elemento. O valor é o atributo referente a esta característica. Se você quer modificar a cor do texto, o valor é um Hexadecimal, RGBA ou até mesmo o nome da cor por extenso.

Os seletores são a alma do CSS e você precisa dominá-los. É com os seletores que você irá escolher um determinado elemento dentro todos os outros elementos do site para formatá-lo. Boa parte da inteligência do CSS está em saber utilizar os seletores de uma maneira eficaz, escalável e inteligente.

O seletor representa uma estrutura. Essa estrutura é usada como uma condição para determinar quais elementos de um grupo de elementos serão formatados.

Seletores encadeados e seletores agrupados são a base do CSS. Você os aprende por osmose durante o dia a dia. Para você lembrar o que são seletores encadeados e agrupados, segue um exemplo de seletor encadeado:

```
div p strong a {  
    color: red;  
}
```

Este seletor formata o *link* ([a](#)), que está dentro de um (**strong**), que está dentro de (

p

) e que por sua vez está dentro de um (

div

).

Exemplo de seletor agrupado:

```
strong, em, span {  
    color: red;  
}
```

Você agrupa elementos separados por vírgula para que herdem a mesma formatação.

Estes seletores servem para cobrir suas necessidades básicas de formatação de elementos. Eles fazem o simples. O que vai fazer você trabalhar menos, escrever menos código CSS e também menos código Javascript são os seletores complexos.

Os seletores complexos selecionam elementos, talvez você precisaria fazer algum *script* em Javascript para poder marcá-lo com uma **CLASS** ou um **ID** para então você formatá-lo. Com os seletores complexos você consegue formatar elementos que antes eram inalcançáveis.

Imagine que você tenha um título (h1) seguido de um parágrafo (p). Você precisa selecionar todos os parágrafos que vem depois de um título h1. Com os seletores complexos você fará assim:

```
h1 + p {  
  color:red;  
}
```

Segue um pequeno resumo dos principais seletores:

- **elemento[atr]** - Elemento com um atributo específico;
- **elemento[atr="x"]** - Elemento que tenha um atributo com um valor específico igual a "x";
- **elemento[atr~="x"]** - Elemento com um atributo cujo valor é uma lista separada por espaços, sendo que um dos valores é "x";
- **elemento[atr^="x"]** - Elemento com um atributo cujo valor começa exatamente com *string* "x";
- **elemento[atr\$="x"]** - Elemento com um atributo cujo valor termina exatamente com *string* "x";
- **elemento[atr\*="x"]** - Elemento com um atributo cujo valor contenha a *string* "x";
- **elemento[atr|= "en"]** - Um elemento que tem o atributo específico com o valor que é separado por hífen começando com EN (da esquerda para direita);
- **elemento:root** - Elemento root do documento. Normalmente o HTML;
- **elemento:nthchild(n)** - Selecione um objeto N de um determinado elemento;
- **elemento:nth-lastchild(n)** - Seleciona um objeto N começando pelo último objeto do elemento;
- **elemento:empty** - Seleciona um elemento vazio, sem filhos. Incluindo elementos de texto;
- **elemento:enabled** / **elemento:disabled** - Seleciona um elemento de interface que esteja habilitado ou desabilitado, como *selects*, *checkbox*, *radio button* etc.
- **elemento:checked** - Seleciona elementos que estão checados, como *radio buttons* e *checkboxes*.



- **E > F** - Seleciona os elementos E que são filhos diretos de F.
- **E + F** - Seleciona um elemento F que precede imediatamente o elemento E.

A lista mais completa e atualizada dos seletores pode ser encontrada em <https://www.w3.org/TR/selectors-3/#selectors>.

Uma vez identificado o elemento que deve ser modificado através de um seletor adequado, passamos para a formatação deste elemento.

Cada elemento HTML possui um conjunto de características que podem ser tratadas, envolvendo posição, tamanho, forma, cores, bordas, sombras, ordem de apresentação, entre outros.

Um recurso muito usado é a definição de formatação condicionada a uma determinada ação, por exemplo, quando o *mouse* passar por cima do elemento:

```
a {
  color: white;
  background: gray;
}
a:hover {
  color: black;
  background: red;
}
```

Neste exemplo, o elemento ([a](#)) (*link*) tem o fundo cinza e a cor das letras brancas, quando o mouse passa por cima do elemento, ele é modificado, passando a ficar com o fundo vermelho e as letras pretas.

O problema é que a transição é muito brusca. O browser apenas modifica as características entre os dois blocos e pronto. Não há nenhuma transição suave entre os dois estados, para isso, usamos a propriedade *transition* juntamente com nossos seletores, por exemplo:

```
a {
  color: white;
  background: gray;
  -webkit-transition: 0.5s;
}
a:hover {
  color: black;
```

```
background: red;
-webkit-transition: 0.5s;
}
```

Para testar nosso código podemos usar o <https://jsfiddle.net/>, basta acessar e colar os códigos:

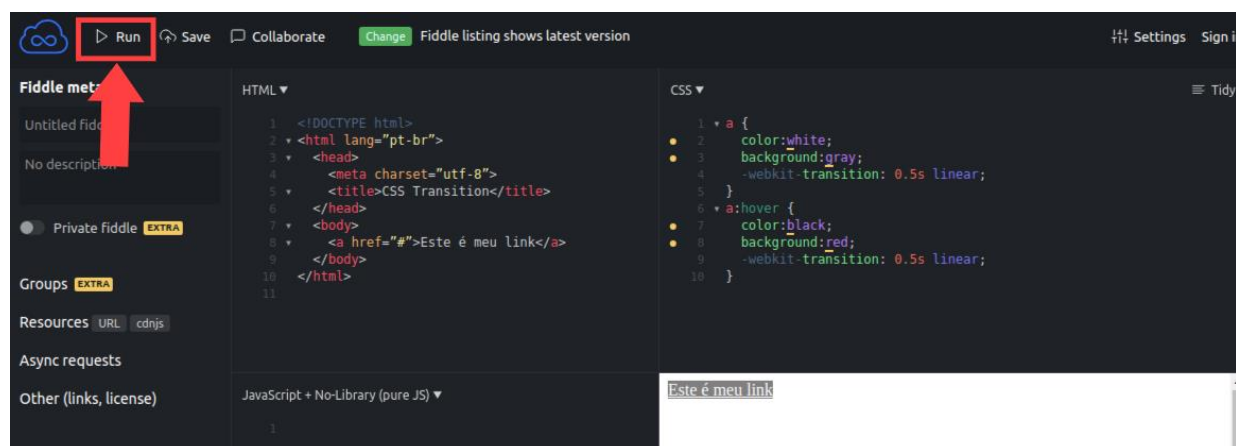
## HTML

```
<html lang="pt-br">
<head>
<meta charset="utf-8">
<title>CSS Transition</title>
</head>
<body>
<a href="#">Este é meu link</a>
</body>
</html>
```

## CSS

```
a {
  color:white;
  background:gray;
  -webkit-transition: 0.5s linear;
}
a:hover {
  color:black;
  background:red;
  -webkit-transition: 0.5s linear;
}
```

Exemplo:



Desta forma, adicionamos uma transição de meio segundo (**0.5s**) na modificação quando colocamos o *mouse* sobre o *link* e também quando retiramos o *mouse* de cima do *link*.

A propriedade **transition** trabalha de forma muito simples e inflexível. Você praticamente diz para o *browser* qual o valor inicial e o valor final para que ele aplique a transição automaticamente, controlamos praticamente apenas o tempo de execução. Para termos mais controle sobre a animação temos a propriedade **animation** que trabalha juntamente com a *rule* **keyframe**.

Basicamente você consegue controlar as características do objeto e suas diversas transformações definindo fases da animação. Observe o código abaixo e veja seu funcionamento:

## HTML

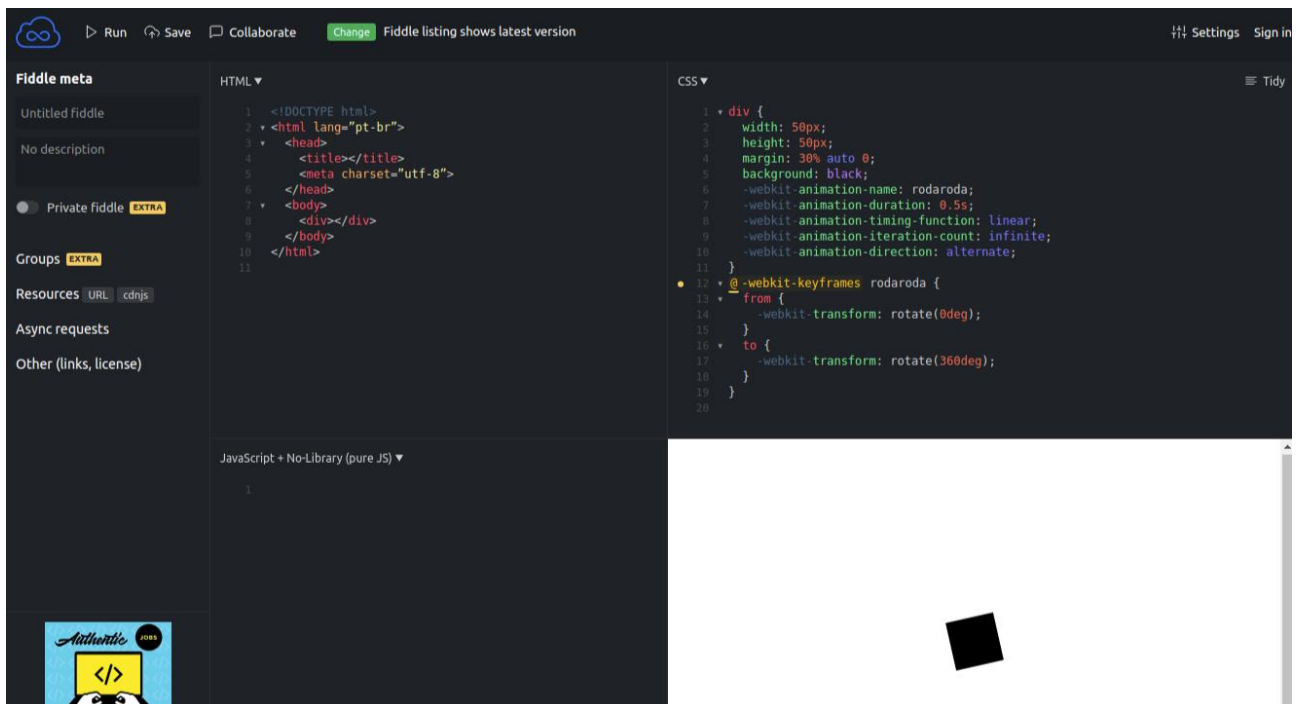
```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title></title>
    <meta charset="utf-8">
  </head>
  <body>
    <div></div>
  </body>
</html>
```

## CSS

```
div {
  width: 50px;
  height: 50px;
  margin: 30% auto 0;
  background: black;
  -webkit-animation-name: rodaroda;
  -webkit-animation-duration: 0.5s;
  -webkit-animation-timing-function: linear;
  -webkit-animation-iteration-count: infinite;
  -webkit-animation-direction: alternate;
}

@-webkit-keyframes rodaroda {
  from {
    -webkit-transform: rotate(0deg);
  }
  to {
    -webkit-transform: rotate(360deg);
  }
}
```

Execute o código usando o <https://jsfiddle.net/>:



Estas são apenas algumas das possibilidades do CSS, a documentação oficial está disponível em: <https://www.w3schools.com/css/default.asp>.

### Dica de leitura

CSS - CURSO W3C ESCRITÓRIO BRASIL da <https://www.w3c.br/>.

Disponível em: <https://www.w3c.br/Cursos/CursoCSS3/>. Acesso em: 20 abr. 2021.

Disponível em: <https://www.w3c.br/pub/Cursos/CursoCSS3/css-web.pdf>. Acesso em: 20 abr. 2021.

## JavaScript - Introdução

JavaScript é uma linguagem de *script* orientada a objetos, multiplataforma. É uma linguagem pequena e leve. Dentro de um ambiente de *host* (por exemplo, um navegador *web*), o JavaScript pode ser ligado aos objetos desse ambiente para prover um controle programático sobre eles.

JavaScript tem uma biblioteca padrão de objetos, como: *Array*, *Date*, e *Math*, e um conjunto de elementos que formam o núcleo da linguagem, tais como: operadores, estruturas de controle e declarações. O núcleo do JavaScript pode ser estendido para uma variedade de propósitos, complementando assim a linguagem:

O lado cliente do JavaScript estende-se do núcleo da linguagem, fornecendo objetos para controlar um navegador *web* e seu *Document Object Model* (DOM). Por exemplo, as extensões do lado do cliente permitem que uma aplicação coloque elementos em um formulário HTML e responda a eventos do usuário, como cliques do mouse, entrada de formulário e navegação da página.

O lado do servidor do JavaScript estende-se do núcleo da linguagem, fornecendo objetos relevantes à execução do JavaScript em um servidor. Por exemplo, as extensões do lado do servidor permitem que uma aplicação se comunique com um banco de dados, garantindo a continuidade de informações de uma chamada para a outra da aplicação, ou executar manipulações de arquivos em um servidor.

## Videoaula 3

Agora assista esta videoaula que aborda um pouco sobre JavaScript.

Disponível em: <https://www.youtube.com/watch?v=kk7VM9RfB3w>. Acesso em: 20 abr. 2021.



### Videoaula 3

Utilize o QRcode para assistir!

Agora assista esta videoaula que aborda um pouco sobre JavaScript.



Os códigos JavaScript podem ser inseridos na página HTML de duas formas:

- **Interno no documento:** para inserir o código direto na estrutura do HTML, utilizamos as tags `<script>` e `</script>`.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      //código JavaScript
    </script>
  </head>
  <body>
  </body>
</html>
```

- **Externo ao documento:** o código JavaScript também pode ser mantido em um arquivo separado do HTML. Para isso, deve-se referenciar tal arquivo na página.

O arquivo deve ser salvo com a extensão “.JS” e sua estrutura é a mesma utilizada quando o código é posto internamente no documento. Cabe aqui uma observação importante: a tag `<script>` requer a tag de fechamento separada, não podendo ser fechada em si própria como `<script type="...">/>`.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript"
src="meuArquivo.js"></script>
  </head>
  <body>
  </body>
</html>
```

## Sintaxe da linguagem JavaScript

JavaScript foi criada com base na ECMAScript e sua sintaxe é bastante semelhante a linguagens de alto nível muito utilizadas como C e Java, como veremos a seguir.

## Usando variáveis no JavaScript

Essa linguagem possui tipagem dinâmica, ou seja, não é necessário definir o tipo das variáveis ao declará-las, para isso basta usar a palavra reservada “var”.

```
var nome;
nome = "Fulano de Tal";
var idade = 30;
idade = 30 + 20 - 10*5;
```

## Trabalhando com funções

JavaScript fornece também suporte a funções, aliado às facilidades da tipagem dinâmica, o que torna a definição de métodos simples e prática. Para criar funções, utilizamos a palavra reservada **function**.



As funções podem receber parâmetros e retornar valores, mas o tipo de retorno e o tipo dos parâmetros não precisam ser previamente definidos.

```
function exibirMensagem()  
{  
  alert("Olá, seja bem-vindo(a)!");  
}
```

Para definir o valor de retorno da função, deve-se utilizar a palavra reservada `return` seguida do valor ou expressão do resultado.

```
function somar(A, B)  
{  
  return A + B;  
}
```

## Videoaula 4

Agora, assista à videoaula que aborda um pouco sobre JavaScript.

Disponível em: <https://www.youtube.com/watch?v=0T1bzia02dM>. Acesso em: 20 abr. 2021.



### Videoaula 4

Utilize o QRcode para assistir!

Agora, assista à videoaula que aborda um pouco sobre JavaScript.



## Estruturas de controle de fluxo

Assim como a maioria das linguagens de alto nível, JavaScript possui estruturas condicionais e de repetição para controle de fluxo.

```
if(condição 1) {  
    //ação se condição 1 verdadeira  
} else if (condição 2) {  
    //ação se condição 2 verdadeira  
} else {  
    //ação se nenhuma das condições for verdadeira  
}
```

O bloco `else` pode ser omitido, caso apenas uma condição precise ser avaliada.

O comando `switch` verifica o valor de uma variável e, para cada uma das opções, executa um conjunto de ações. Se nenhum dos valores for verificado, os comandos do bloco `default` são executados.

```
switch(variável) {  
    case valor1:  
        //ações caso valor1  
        break;  
    case valor2:  
        //ações caso valor2  
        break;  
    case valor3:  
        //ações caso valor3  
        break;  
    default:  
        //ações caso nenhum dos valores  
        break  
}
```

O bloco `default`, porém, pode ser omitido caso não exista uma ação padrão a ser executada se nenhuma das opções for observada.

A estrutura de repetição `while` é usada para executar um conjunto de ações enquanto uma condição for verdadeira. Quando a condição se tornar falsa, o bloco é

finalizado. A seguir, temos um exemplo que exibe uma mensagem para o usuário enquanto uma variável for menor que 5.

```
var contador = 0;
while(contador < 5) {
    alert("Olá");
    contador = contador + 1;
}
```

Uma outra estrutura muito semelhante é a `do ... while`, que executa um bloco de ações até que uma condição seja falsa. Porém, essa condição é avaliada após a execução dos comandos, fazendo com que estes sejam executados pelo menos uma vez.

```
var contador = 0;
do {
    alert("Olá");
    contador = contador + 1;
} while(contador < 5)
```

Por último, temos o comando `for`, que usa um contador para executar um bloco de ações uma determinada quantidade de vezes.

O entendimento dessa estrutura fica mais fácil se observarmos um exemplo prático. Neste exemplo, uma variável “contador” é inicializada com o valor zero, e enquanto for menor que 10, o laço *for* deve ser executado:

```
var contador;
for(contador = 0; contador < 10; contador++) {
    alert(contador);
}
```

## Manipulando eventos

A linguagem JavaScript também permite trabalhar com eventos dos elementos HTML como botões e caixas de texto. Eventos são disparados quando alguma ação é executada, como o clique em um botão ou a digitação de valores em um *input*.

No código JavaScript pode-se atribuir valores aos eventos como se fossem propriedades, desde que o valor atribuído seja um código a ser executado. Neste exemplo temos código de uma página com um botão que, ao ser clicado, exibe uma mensagem (*alert*).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
  </head>
  <body>
    <button onclick="alert('Você clicou no botão');">clique aqui</button>
  </body>
</html>
```

Vale notar que nesse caso as *tags* `<script>` não foram necessárias, pois o código encontra-se dentro do próprio *button*. Outro ponto a ser observado é que foram usadas aspas duplas para definir o código e aspas simples no interior do mesmo, quando precisamos escrever um texto dentro de um evento.

Caso o código a ser executado no evento seja muito extenso, é possível criar uma função para isso e associá-la ao evento em questão. Basta informar, no lugar do código, o nome da função (com parâmetros, caso existam).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <script type="text/javascript">
      function clique_botao() {
        alert("Você clicou no botão");
      }
    </script>
  </head>
  <body>
    <button onclick="clique_botao();">clique aqui</button>
  </body>
```

</html>

## Variáveis

Variáveis são espaços na memória do computador onde você pode armazenar dados. Você começa declarando uma variável com a palavra-chave “var” (menos recomendado) ou a palavra-chave “let”, seguida por qualquer nome que você queira chamá-la:

```
let minhaVARIavel = 'Bob';  
minhaVariavel = 'Steve';
```

Note que as variáveis podem conter valores com diferentes tipos de dados:

**Quadro 1** – Variáveis e suas explicações

Variável	Explicação	Exemplo
<i>String</i>	Uma sequência de texto é conhecida como uma <i>string</i> . Para mostrar que o valor é uma <i>string</i> , você deve envolvê-la em aspas.	<pre>let minhaVariavel = 'Bob';</pre>
<i>Number</i>	Um número. Números não tem aspas ao redor deles.	<pre>let minhaVariavel = 10;</pre>
<i>Boolean</i>	Um valor verdadeiro ou falso. As palavras <i>true</i> e <i>false</i> são palavras-chaves especiais em JS e não precisam de aspas.	<pre>let minhaVariavel = true;</pre>
<i>Array</i>	Uma estrutura que permite armazenar vários valores em uma única variável.	<pre>let minhaVariavel = [1, 'Bob', 'Steve', 10]; Acesse cada item do array dessa maneira: minhaVariavel[0], minhaVariavel[1], etc.</pre>

### Object

Basicamente, qualquer coisa. Tudo em JavaScript é um objeto e pode ser armazenado em uma variável. Tenha isso em mente enquanto aprende.

```
let minhaVariavel =  
document.querySelector('h1')  
;  
Todos os exemplos acima também.
```

Fonte: o autor (2021)

Então, por que precisamos de variáveis? Bom, variáveis são necessárias para fazer qualquer coisa interessante na programação. Se os valores não pudessem mudar, então você não poderia fazer nada dinâmico, como personalizar uma mensagem de boas-vindas, ou mudar a imagem mostrada em uma galeria de imagens.

## Comentários

Os comentários são, essencialmente, pequenos trechos de texto que podem ser adicionados entre os códigos e são ignorados pelo navegador. Você pode colocar comentários no código JavaScript, assim como em CSS:

```
/*  
Tudo no meio é um comentário.  
*/
```

Se o seu comentário não tiver quebras de linha, é mais fácil colocá-lo depois de duas barras como esta:

```
// Isto é um comentário.
```

## Operadores

Um operador é um símbolo matemático que produz um resultado baseado em dois valores (ou variáveis). No quadro a seguir, você pode ver alguns dos operadores mais simples, além de alguns exemplos para experimentar no console JavaScript.

**Quadro 2 - Operadores**

Operador	Explicação	Símbolo(s)	Exemplo
Adição	Usado para somar dois números ou juntar duas <i>strings</i> .	+	<code>6 + 9;</code> <code>"Ola " + "mundo!";</code>
Subtração, multiplicação, divisão	Fazem exatamente o que você espera que eles façam na matemática básica.	-, *, /	<code>9 - 3;</code> <code>8 * 2;</code> // no JS a multiplicação é um asterisco <code>9 / 3;</code>
Atribuição	Você já viu essa, ela associa um valor a uma variável.	=	<code>let minhaVariavel = 'Bob';</code>
Operador de igualdade	Faz um teste para ver se dois valores são iguais, retornando um resultado <i>true/false</i> (booleano).	===	<code>let minhaVARIavel = 3;</code> <code>minhaVariavel === 4;</code>
Negação, não igual	Retorna o valor lógico oposto do sinal; transforma um <i>true</i> em um <i>false</i> etc. Quando usado junto com o operador de igualdade, o operador de negação testa se os valores são diferentes.	!, !==	Para " <b>Negação</b> ", a expressão básica é <i>true</i> , mas a comparação retorna <i>false</i> porque a negamos:  <code>let minhaVariavel = 3;</code> <code>!(minhaVariavel === 3);</code>  "Não igual" dá basicamente o mesmo resultado com sintaxe diferente. Aqui estamos testando "é minhaVariavel NÃO é igual a 3". Isso retorna <i>false</i> porque minhaVariavel É igual a 3.



```
let minhaVariavel =  
3;  
minhaVariavel !==  
3;
```

Fonte: o autor (2021)

Há vários outros operadores para explorar, mas por enquanto esses são suficientes. Consulte para ver uma lista completa:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators>

### Nota:

Misturar tipos de dados pode levar a resultados estranhos quando uma operação é executada, portanto, tome cuidado para declarar suas variáveis corretamente, e você obterá os resultados esperados. Por exemplo, digite `'35' + '25'` no seu console. Por que você não teve o resultado esperado? Porque as aspas transformam os números em *strings*, você acabou concatenando *strings* em vez de somar os números. Se você digitar `35 + 25`, você obterá o resultado correto.

## Objeto, noções básicas

Um objeto é uma coleção de dados e/ou funcionalidades relacionadas (que geralmente consistem em diversas variáveis e funções — que são chamadas de propriedades e métodos quando estão dentro de objetos).

### Videoaula 5

Agora, assista esta videoaula que aborda um pouco sobre JavaScript.

Disponível em: <https://www.youtube.com/watch?v=Ep3xFyF6vUU>. Acesso em: 20 abr. 2021.



## Videoaula 5

Utilize o QRcode para assistir!

Agora, assista esta videoaula que aborda um pouco sobre JavaScript.



Como acontece com muitas coisas em JavaScript, a criação de um objeto geralmente começa com a definição e a inicialização de uma variável.

```
var pessoa = {};
```

Parabéns, você acabou de criar seu primeiro objeto. Tarefa concluída! Mas este é um objeto vazio, então não podemos fazer muita coisa com isso. Vamos atualizar nosso objeto para ficar assim:

```
var pessoa = {  
  nome: ['Bob', 'Smith'],  
  idade: 32,  
  sexo: 'masculino',  
  interesses: ['música', 'esquiar'],  
  bio: function() {  
    alert(this.nome[0] + ' ' + this.nome[1] + ' tem ' + this.idade + '  
anos de idade. Ele gosta de ' + this.interesses[0] + ' e ' +  
this.interesses[1] + '.');  
  },  
  saudacao: function() {  
    alert('Oi! Eu sou ' + this.nome[0] + '.');  
  }  
};
```

Agora você tem alguns dados e funcionalidades dentro de seu objeto e é capaz de acessá-los com uma sintaxe simples e agradável!

```

pessoa.nome
pessoa.nome[0]
pessoa.idade
pessoa.interesses[1]
pessoa.bio()
pessoa.saudacao()

```

Então, o que está acontecendo? Bem, um objeto é composto de vários membros, cada um com um nome (ex.: nome e idade vistos acima), e um valor (ex.: ['Bob', 'Smith'] e 32). Cada par nome/valor deve ser separado por uma vírgula e o nome e valor, em cada caso, separados por dois pontos. A sintaxe sempre segue esse padrão:

```

var nomeDoObjeto = {
  nomeMembro1: valorMembro1,
  nomeMembro2: valorMembro2,
  nomeMembro3: valorMembro3
};

```

O valor de um membro do objeto pode ser praticamente qualquer coisa. Em nosso objeto *pessoa*, temos uma *string*, um número, dois *arrays* e duas *functions*. Os primeiros quatro são data item (dados) e são referenciados como propriedades do objeto. Enquanto os dois últimos itens são funções que permitem que o objeto faça algo com esses dados. São chamados de métodos do objeto.

Um objeto como esse é chamado de objeto literal — ao pé da letra, escrevemos o conteúdo do objeto conforme o criamos. Isto está em contraste com objetos instanciados de classes, que veremos mais adiante.

É muito comum criar um objeto usando um objeto literal quando você deseja transferir uma série de itens de dados relacionados e estruturados de alguma maneira, por exemplo, enviando uma solicitação para o servidor para ser colocado em um banco de dados. Enviar um único objeto é muito mais eficiente do que enviar vários itens individualmente, e é mais fácil trabalhar com um *array*, quando você deseja identificar itens individuais pelo nome.

## Notação de ponto

Acima, você acessou as propriedades de objetos e métodos usando notação de ponto. O objeto nome (pessoa) atua como *namespace* (espaço de nomes) — ele deve ser digitado primeiro para que você acesse qualquer coisa encapsulada dentro do objeto. Depois você escreve um ponto, então o item que quer acessar — isso pode ser o nome de uma simples propriedade, um item de um *array* ou a chamada para um dos métodos do objeto, por exemplo:

```
pessoa.idade
pessoa.interesse[1]
pessoa.bio()
```

## Sub-namespaces

É até possível fazer o valor de um membro de um objeto ser outro objeto. Por exemplo, tente alterar o nome do membro de:

```
nome: ['Bob', 'Smith'],
```

Para:

```
nome : {
  primeiro: 'Bob',
  ultimo: 'Smith'
},
```

Aqui estamos efetivamente criando um sub-*namespace*. Parece difícil, mas não é — para acessar esses itens você apenas precisa encadear mais um passo ao final de outro ponto. Tente isso aqui no console:

```
pessoa.nome.primeiro
pessoa.nome.ultimo
```

Importante: nesse ponto você também precisará revisar seus métodos e mudar quaisquer instâncias de:

```
nome[0]  
nome[1]
```

Para:

```
nome.primeiro  
nome.ultimo
```

Caso contrário seus métodos não funcionarão.

## Notação de colchetes

Há outra forma de acessar propriedades do objeto — usando notação de colchetes. Ao invés desses:

```
pessoa.idade  
pessoa.nome.primeiro
```

Você pode usar:

```
pessoa['idade']  
pessoa['nome']['primeiro']
```

Fica muito parecido com a maneira que acessamos itens de um *array*, e é basicamente a mesma coisa, mas ao invés de usarmos um número de índice para selecionar um item, usamos o nome associado a cada valor. Não é por menos que objetos às vezes são chamados de **arrays associativos** — eles mapeiam *strings* a valores do mesmo modo que *arrays* mapeiam números a valores.

## “Setando” membros do objeto

Até agora nós apenas procuramos receber (ou apanhar) membros de objetos — podemos também “setar” (atualizar) o valor de membros de objetos simplesmente

declarando o membro que queremos “setar” (usando notação de ponto ou colchete), da seguinte forma:

```
peessoa.idade = 45;  
peessoa['nome']['ultimo'] = 'Cratchit';
```

Tente escrever as linhas acima e então apanhar seus membros novamente para ver como mudaram. Assim:

```
peessoa.idade  
peessoa['nome']['ultimo']
```

Não podemos apenas atualizar valores existentes de propriedades e métodos; podemos também criar membros completamente novos. Tente isso no console:

```
peessoa['olhos'] = 'castanho'.  
peessoa.despedida = function() { alert( "Adeus a todos!" ); }
```

Podemos testar nossos novos membros:

```
peessoa['olhos'];  
peessoa.despedida();
```

Um aspecto útil de notação de colchetes é que ele pode ser usado não apenas para “setar” valores dinamicamente, mas também nomes de membros. Vamos dizer que queremos que usuários possam armazenar tipos de valores personalizados em seus dados de 'peessoa', digitando o nome e o valor do membro em dois *inputs* de texto. Podemos obter esses valores dessa forma:

```
var myDataName = nameInput.value;  
var myDataValue = nameValue.value;
```

Podemos, então, adicionar esse novo nome e valor ao objeto peessoa, assim:

```
peessoa[myDataName] = myDataValue;
```

Para testar isso, tente adicionar as seguintes linhas em seu código, abaixo do fechamento das chaves do objeto `pessoa`:

```
var myDataName = 'altura';  
var myDataValue = '1.75m';  
pessoa[myDataName] = myDataValue;
```

Agora tente salvar e atualizar, entrando o seguinte no seu *input* de texto:

```
pessoa.altura
```

Adicionar uma propriedade a um objeto usando o método acima não é possível com a notação ponto, que só aceita um nome de membro literal, não aceita valor de variável apontando para um nome.

## O que é o "*this*"?

Você pode ter reparado algo levemente estranho em nossos métodos. Confira um exemplo abaixo:

```
saudacao: function(){  
  alert("Oi! Meu nome é " + this.nome.primeiro + ".");  
}
```

Você deve estar se perguntando o que é o "*this*". A palavra-chave *this* se refere ao objeto atual em que o código está sendo escrito — nesse caso, o *this* se refere a `pessoa`. Então, por que simplesmente não escrever `pessoa`? Como verá no artigo “Orientação a objeto em JavaScript para iniciantes”, quando começamos a criar funções construtoras etc., o *this* é muito útil — sempre lhe assegurará que os valores corretos estão sendo usados quando o contexto de um membro muda (exemplo: duas instâncias diferentes do objeto “`pessoa`” podem ter diferentes nomes, mas vão querer usar seu próprio nome ao usar a saudação).

Vamos ilustrar o que queremos dizer com um par de objetos “`pessoa`”:

```
var pessoa1 = {  
  nome: 'Chris',  
  saudacao: function() {  
    alert('Oi! Meu nome é ' + this.nome + '.');  
  }  
}  
  
var pessoa2 = {  
  nome: 'Brian',  
  saudacao: function() {  
    alert('Oi! Meu nome é ' + this.nome + '.');  
  }  
}
```

Neste caso, `pessoa1.saudacao()` gerará "Oi! Meu nome é Chris."; no entanto, `pessoa2.saudacao()` retornará "Oi! Meu nome é Brian.", mesmo que os códigos dos métodos sejam idênticos. Como dissemos antes, o *this* é igual ao código do objeto dentro dele — não é exatamente útil quando estamos escrevendo objetos literais na mão, mas é realmente incrível quando adicionamos objetos gerados dinamicamente (por exemplo, usando construtores).

### Nota

É útil pensar sobre como os objetos se comunicam passando mensagens - quando um objeto precisa de outro objeto para realizar algum tipo de ação, ele frequentemente enviará uma mensagem para outro objeto através de um de seus métodos e aguardará uma resposta, que reconhecemos como um valor de retorno.

## TypeScript - Introdução

O JavaScript, uma das linguagens de programação mais usadas do mundo, tornou-se a linguagem de programação oficial da *Web*. Os desenvolvedores o usam para escrever aplicativos multiplataforma que podem ser executados em qualquer plataforma e em qualquer navegador.



Embora o JavaScript seja usado para criar aplicativos multiplataforma, ele não foi concebido para aplicativos grandes envolvendo milhares ou até mesmo milhões de linhas de código. O JavaScript não tem alguns dos recursos de linguagens mais maduras presentes nos aplicativos sofisticados de hoje. As IDEs (editores de desenvolvimento integrado) podem ter dificuldades para gerenciar o JavaScript e manter essas grandes bases de código.

O TypeScript resolve as limitações do JavaScript, fazendo isso sem comprometer a principal proposta de valor do JavaScript: a capacidade de executar seu código em qualquer lugar e em todas as plataformas, navegadores e *hosts*.

## O que é o TypeScript?

O TypeScript é uma linguagem de *software* livre desenvolvida pela Microsoft. Ele é um superconjunto do JavaScript, o que significa que você pode continuar a usar as habilidades de JavaScript que já desenvolveu e adicionar determinados recursos que não estavam disponíveis para você anteriormente.

## Dicas de tipo

O principal recurso do TypeScript é o sistema de tipos. No TypeScript, você pode identificar o tipo de dados de uma variável ou parâmetro usando uma dica de tipo. Com as dicas de tipo, você descreve a forma de um objeto, o que proporciona uma melhor documentação e permite que o TypeScript valide se o código está funcionando corretamente.

Por meio da verificação de tipo estático, o TypeScript detecta, no início do desenvolvimento, problemas de código que o JavaScript normalmente não consegue detectar até que o código seja executado no navegador. Os tipos também permitem que você descreva o que seu código é destinado a fazer. Se você estiver trabalhando em uma equipe, um colega que chegue depois de você também poderá entender o código com facilidade.

Os tipos também potencializam os benefícios de inteligência e produtividade das ferramentas de desenvolvimento, como IntelliSense, navegação baseada em símbolos, ir

para definição, localizar todas as referências, preenchimento de instruções e refatoração de código.

A escritura de tipos pode ser opcional no TypeScript, porque a inferência de tipos torna seu código muito poderoso, mesmo sem que você escreva um código adicional. Se o TypeScript puder determinar o tipo de dados implicitamente (por exemplo, quando você atribuir um valor a uma variável usando `let age = 42`), ele inferirá automaticamente o tipo de dados.

## Outros recursos de código do TypeScript

O TypeScript tem recursos de codificação adicionais que você não encontrará em JavaScript:

- Interfaces;
- *Namespaces*;
- Genéricos;
- Classes abstratas;
- Modificadores de dados;
- Opcionais;
- Sobrecarga de função;
- Decoradores;
- Utilitários de tipo;
- Palavra-chave *readonly*.

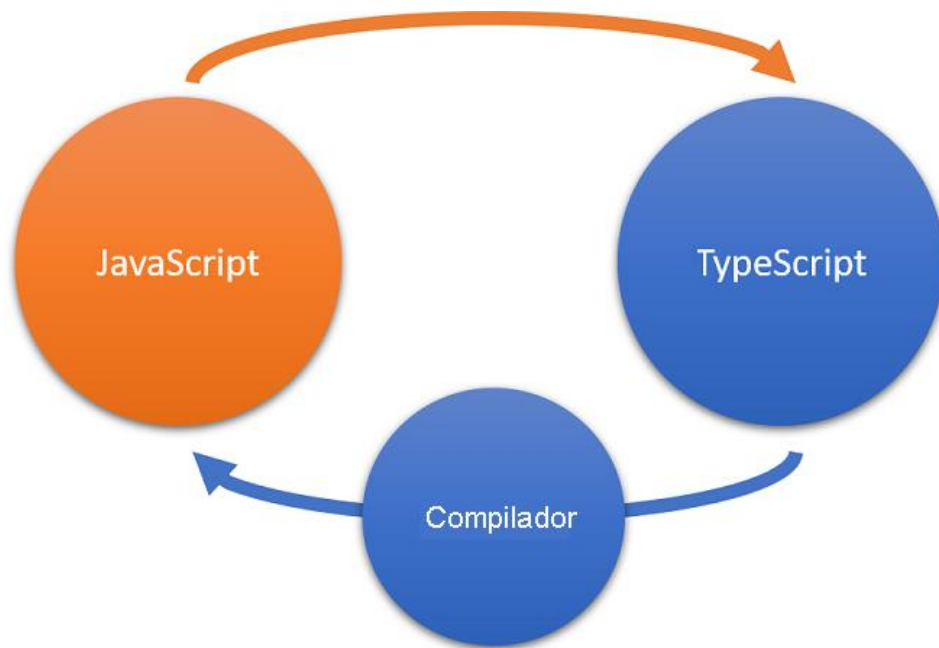
## Compatibilidade do TypeScript com o JavaScript

O TypeScript é um superconjunto estrito do ECMAScript 2015 (ECMAScript 6 ou ES6). Isso significa que todo o código JavaScript também é um código TypeScript, e um programa TypeScript pode consumir o JavaScript de maneira direta.

Os navegadores entendem apenas o JavaScript. Para que seu aplicativo funcione, ao escrevê-lo em TypeScript, você precisa compilar seu código e convertê-lo em JavaScript. Você transforma o código TypeScript em código JavaScript usando o compilador TypeScript ou um transcompilador compatível com TypeScript. O JavaScript

resultante é um código simples e limpo que pode ser executado em qualquer lugar em que o JavaScript é executado: em um navegador, no Node.js ou em seus aplicativos.

**Figura 5** – Compilador para TypeScript e JavaScript



Fonte: o autor (2021)

### **Nota**

Quando você trabalha com o TypeScript, lembre-se que, em quase todas as situações, o TypeScript será compilado (ou transformado) em JavaScript e o JavaScript será realmente executado pelo *runtime*. Você pode usar o TypeScript em todos os projetos que usam JavaScript.

## Aula 02 – Preparando o ambiente e criando nosso aplicativo IONIC

Agora nosso foco é inteiramente prático, vamos instalar nosso ambiente de trabalho com Visual Studio Code, Node.js e Ionic, teremos uma pequena introdução ao uso do terminal (*Prompt de Comandos*) e vamos criar nosso primeiro projeto em IONIC, entendendo os componentes básicos e o funcionamento das rotas para navegar entre as telas da nossa aplicação.

Antes de começarmos a desenvolver aplicações temos que entender que boa parte das atividades serão realizadas em uma interface de linha de comando (ILC), em inglês *command-line interface* (CLI).

**Figura 1** - Screenshot de uma sessão de *shell bash* em um terminal GNOME

```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.pmtpa.wikimedia.org (208.80.152.2) 56(84) bytes of data.
^C
--- text.pmtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 540.528/540.528/540.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 28:42 ..
drwxr-xr-x.  2 root root 4096 May 14 08:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x.  3 root root 4096 May 18 16:03 db
drwxr-xr-x.  3 root root 4096 May 18 16:03 empty
drwxr-xr-x.  2 root root 4096 May 18 16:03 games
drwxrwx--T.  2 root gdm  4096 Jun  2 18:39 gdm
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib
drwxr-xr-x.  2 root root 4096 May 18 16:03 local
lrwxrwxrwx.  1 root root    11 May 14 08:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 28:42 log
lrwxrwxrwx.  1 root root    10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x.  2 root root 4096 May 18 16:03 nis
drwxr-xr-x.  2 root root 4096 May 18 16:03 opt
drwxr-xr-x.  2 root root 4096 May 18 16:03 preserve
drwxr-xr-x.  2 root root 4096 Jul  1 22:11 report
lrwxrwxrwx.  1 root root    6 May 14 08:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt.  4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x.  2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
rpmfusion-free-updates                               | 2.7 kB      00:00
rpmfusion-free-updates/primary_db                    | 206 kB      00:04
rpmfusion-nonfree-updates                             | 2.7 kB      00:00
updates/metalink                                     | 5.9 kB      00:00
updates                                                | 4.7 kB      00:00
updates/primary_db                                   73% [=====] | 62 kB/s | 2.6 MB 00:15 ETA
```

Fonte: disponível em: [https://pt.wikipedia.org/wiki/Interface\\_de\\_linha\\_de\\_comandos](https://pt.wikipedia.org/wiki/Interface_de_linha_de_comandos). Acesso em: 20 abr. 2021.

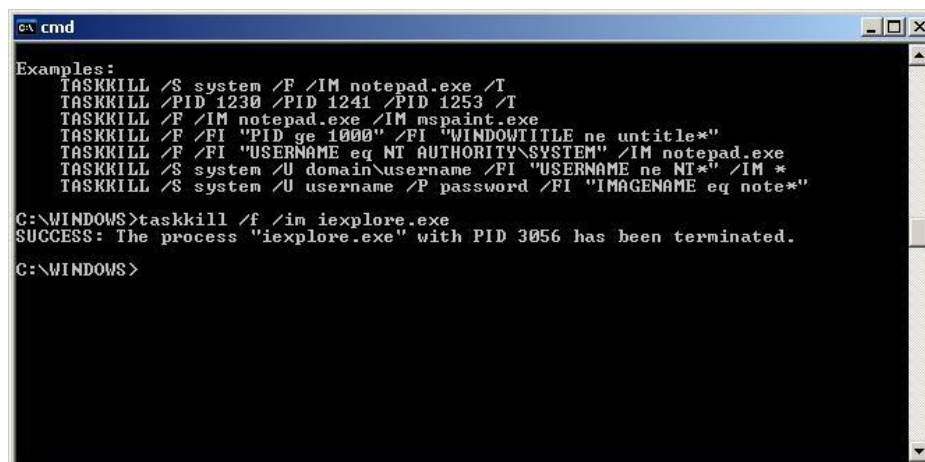
GNU Bash ou simplesmente Bash é um interpretador de comandos, um em meio a diversos tradutores, entre o usuário e o sistema operacional conhecidos como shell.

Shell script é o nome dado a um arquivo que será interpretado por algum programa tipo Shell. Atualmente existem vários programas tipo Shell. Além dos principais - sh e bash

-, existem também, ksh, zsh, csh e tcsh. Um Shell script (ou script em Shell) necessita basicamente do interpretador Shell.

Prompt de Comando (**cmd.exe**) é um interpretador de linha de comando no OS/2 e de sistemas baseados no Windows NT (incluindo Windows 2000, XP, Server 2003 e adiante até o mais recente Windows 10). Ele é um comando análogo ao command.com do MS-DOS e de sistemas Windows 9x, ou de shells utilizados pelos sistemas Unix.

**Figura 2** - cmd.exe do Windows mostrando a execução do Taskkill



```
cmd
Examples:
TASKKILL /S system /F /IM notepad.exe /T
TASKKILL /PID 1230 /PID 1241 /PID 1253 /T
TASKKILL /F /IM notepad.exe /IM mspaint.exe
TASKKILL /F /FI "PID ge 1000" /FI "WINDOWTITLE ne untitled*"
TASKKILL /F /FI "USERNAME eq NT AUTHORITY\SYSTEM" /IM notepad.exe
TASKKILL /S system /U domain\username /FI "USERNAME ne NT*" /IM *
TASKKILL /S system /U username /P password /FI "IMAGENAME eq notepad*"

C:\WINDOWS>taskkill /f /im iexplore.exe
SUCCESS: The process "iexplore.exe" with PID 3056 has been terminated.
C:\WINDOWS>
```

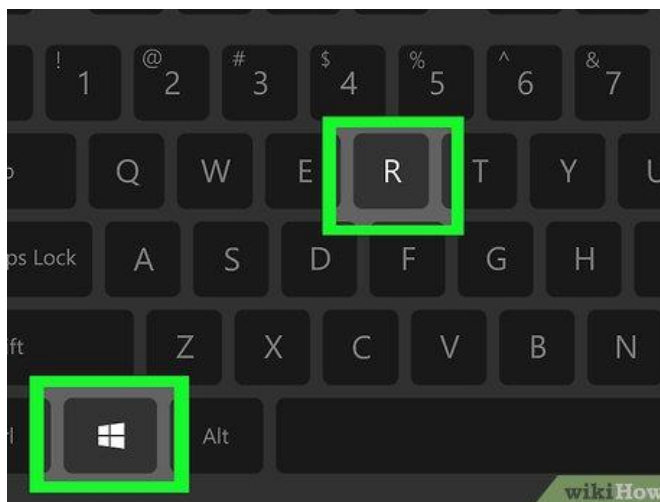
Fonte: disponível em: <https://pt.wikipedia.org/wiki/Command.exe>. Acesso em: 20 abr. 2021.

As interfaces baseadas em linhas de comando possibilitam uma infinidade de aplicações, realizando desde atividades mais simples (que é o nosso caso), até a gestão completa de servidores.

Considerando que a maior parte dos alunos irão trabalhar com Windows ou Linux em seus computadores pessoais, vamos ver alguns exemplos de comandos com os quais teremos que trabalhar.

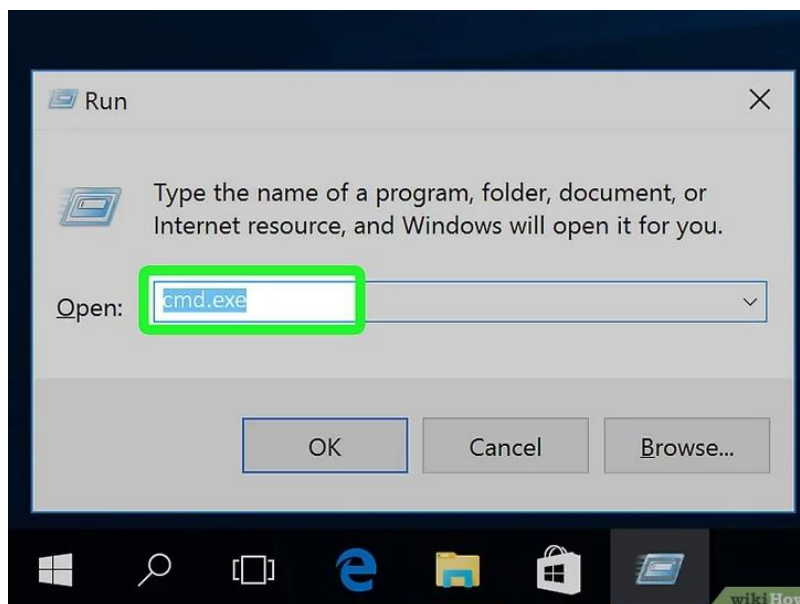
## Windows (CMD.EXE) - Introdução

**Figura 3** – Executando o cmd



Fonte: disponível em: <https://pt.wikihow.com/Abrir-o-Prompt-de-Comando-no-Windows>. Acesso em: 20 abr. 2021.

**Figura 4** – *Prompt* de comando



Fonte: disponível em: <https://pt.wikihow.com/Abrir-o-Prompt-de-Comando-no-Windows>. Acesso em: 20 abr. 2021.

## Linux Terminal (GNU Bash) - Introdução

Em um alto nível, computadores fazem quatro coisas:

- Rodam programas;
- Armazenam dados;
- Comunicam-se entre eles;
- Interagem com humanos.

A interação com humanos pode ser feita de diferentes maneiras, mas a que mais conhecemos é através de um teclado, *mouse* e monitor. Embora a maioria dos sistemas operacionais atuais se comuniquem com humanos através de janelas, ícones e botões, essa tecnologia só começou a se popularizar em meados dos anos 1980.

Este tipo de interface é chamado de **interface gráfica do usuário** (GUI, *graphical user interface*). Antes dessa fase, a interação com computadores se resumia a comandos digitados em um *terminal*, uma interface chamada de **interface por linha de comando** (CLI, *command-line interface*). A CLI consiste no processo em um **loop ler-avaliar-imprimir** (REPL, *read-evaluate-print loop*). Ou seja, quando um usuário digita um comando e aperta **Enter**, o computador lê, executa e imprime uma saída.

Com esta descrição, pode parecer que o usuário envia um comando direto para o computador, e o computador envia um resultado direto para o usuário. Na verdade, existe um programa intermediário, chamado de **shell**. O que um usuário digita é enviado para o shell, que então determina a execução pelo computador.

O shell é um programa como qualquer outro. O que ele tem de especial é que seu trabalho é executar outros programas ao invés de fazer cálculos propriamente ditos. O shell mais popular se chama **Bash**, e é o padrão na maioria das distribuições Linux.

Usar o Bash é muito mais parecido com programação do que com usar um mouse para executar programas. Alguns comandos podem parecer “estranhos” em um primeiro momento, e você pode se sentir como se estivesse voltando ao passado para usar o computador quando não existiam GUIs (apenas CLIs, como o DOS, por exemplo). No entanto, com apenas alguns comandos você será capaz de executar diversas tarefas rotineiras de maneira muito mais rápida, e principalmente automatizar tarefas (com um pouco mais de dedicação).

Possuir familiaridade com o shell é essencial para rodar uma variedade de ferramentas especializadas, como por exemplo, servidores remotos de alta-performance. Como clusters e computação em nuvem estão se tornando cada vez mais popular para análise de grandes bases de dados, ser capaz de interagir com estes sistemas é uma habilidade essencial.

### Videoaula 1

Agora, assista à videoaula que aborda sobre a instalação Node.js Instalação Windows.

Disponível em: <https://www.youtube.com/watch?v=kJD9r08Zhpk>. Acesso em: 20 abr. 2021.



#### Videoaula 1

Utilize o QRcode para assistir!

Agora, assista à videoaula que aborda sobre a instalação Node.js Instalação Windows.



## Node.js - Instalação - Windows

Você precisará de uma cópia do Node.js como um ambiente para executar o pacote do TypeScript. O Node.js, uma biblioteca do JavaScript para aplicativos do lado do servidor, não é necessária para aprender o TypeScript. Mas, ao instalá-lo, você também obtém o Gerenciador de Pacotes do Node, também chamado de npm, o gerenciador de pacotes da linguagem JavaScript. Em seguida, você usará o npm para instalar o pacote TypeScript.

Se você for um desenvolvedor de JavaScript, provavelmente o npm já estará instalado em seu computador. Você pode verificar isso abrindo a janela de *prompt* de comando e inserindo `npm version`. Se o npm estiver instalado, você verá a versão e uma lista de comandos do compilador e estará pronto para usar o npm.



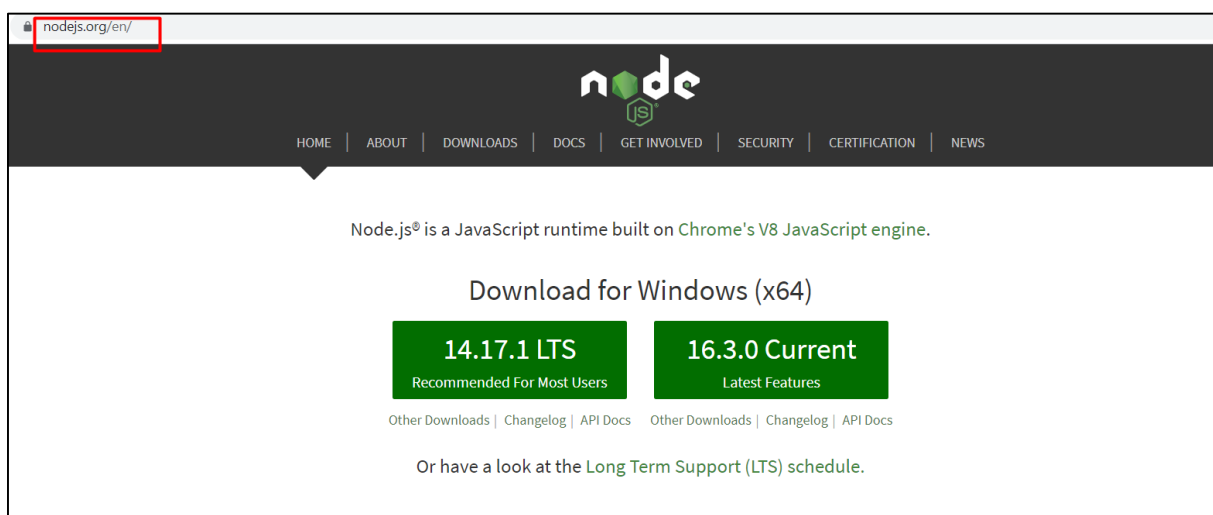
Se o npm não estiver instalado, você verá uma mensagem informando que o comando não foi reconhecido. Para instalar o Node.js:

Vá para a página de *Downloads* do Node.js.

[nodejs.org/en/](https://nodejs.org/en/)

Selecione qualquer uma das versões disponíveis do Node.js para baixá-la e instalá-la.

**Figura 5** – Baixando o Node.js



Fonte: disponível em: [nodejs.org/en/](https://nodejs.org/en/). Acesso em: 20 abr. 2021.

Após feito o *download*, executo o programa baixado da internet.

Podemos seguir com a instalação padrão até terminar a instalação.

Abra a janela do *prompt* de comando para verificar se o Node.js foi instalado com sucesso.

Execute o comando **node -v** ou **node -version**. Junto com o Node.js é instalado um grupo de pacotes. O Comando **NPM -V** verifica a versão destes pacotes instalados.

**Figura 6** – Consulta do comando npm

```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\adail>node -v
v13.6.0

C:\Users\adail>node --version
v13.6.0

C:\Users\adail>npm -v
6.13.4

C:\Users\adail>_
```

Fonte: o autor (2021)

## Videoaula 2

Assista à videoaula que aborda a instalação do Node.js.

Disponível em: [https://www.youtube.com/watch?v=90rBv2Cfq\\_s](https://www.youtube.com/watch?v=90rBv2Cfq_s). Acesso em: 20 abr. 2021.



### Videoaula 2

Utilize o QRcode para assistir!

Assista à videoaula que aborda a instalação do Node.js.



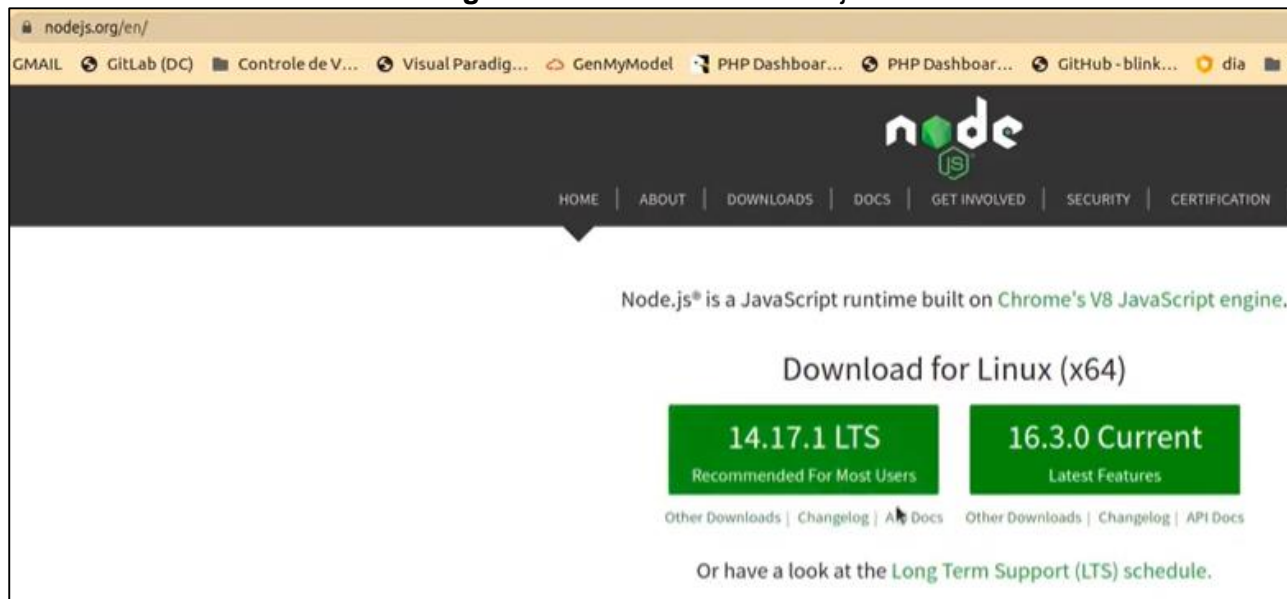
## Node.js - Instalação - Linux

O Linux que utilizaremos nesta instalação é o Ubuntu.

Primeiramente devemos acessar o endereço [nodejs.org](https://nodejs.org) e escolher a versão compatível com meu sistema operacional.

No nosso caso, faremos o *download* da versão 14.17.

**Figura 7 – Download do Node.js**



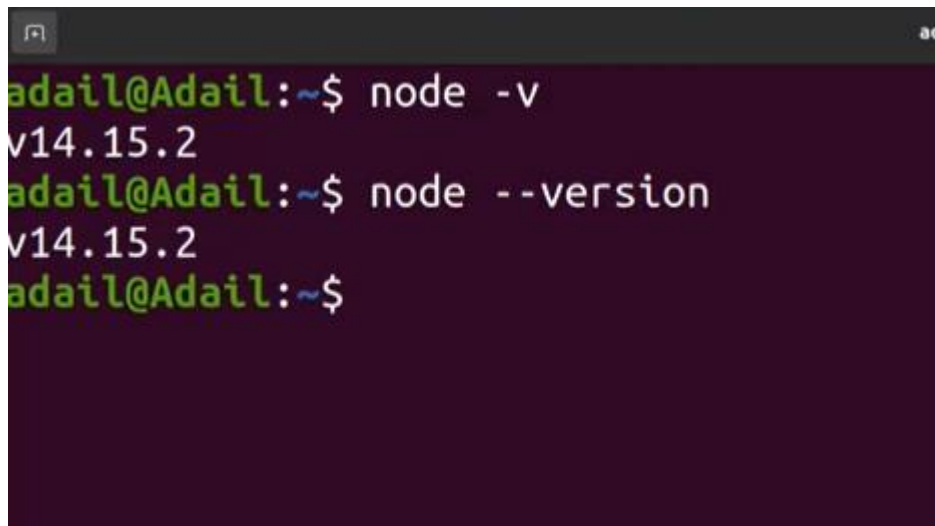
Fonte: disponível em: [nodejs.org/en/](https://nodejs.org/en/). Acesso em: 20 abr. 2021.

Assim que baixado, é necessário descompactar todo o conteúdo da pasta. Dentro da pasta, temos o arquivo **redme** que é utilizado como facilitador do processo com algumas instruções.

Em seguida, vamos executar o conteúdo da pasta BIN para realizar a instalação do *software*.

Para verificar se o *software* foi instalado corretamente, devemos informar o comando: **node -v** ou **node -version**.

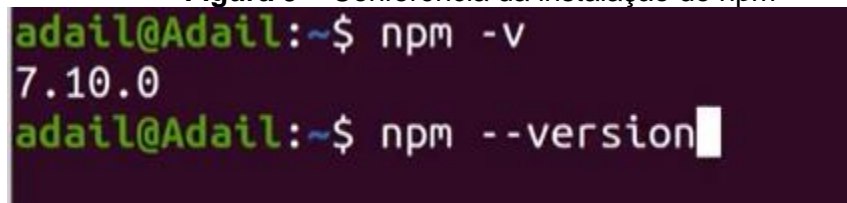
**Figura 8 – Consulta do software**

A terminal window with a dark purple background. The prompt is 'adail@Adail:~\$'. The first command is 'node -v', which outputs 'v14.15.2'. The second command is 'node --version', which also outputs 'v14.15.2'. The prompt returns to 'adail@Adail:~\$'.

Fonte: o autor (2021)

Outro detalhe que podemos verificar, é se foi instalado corretamente o NPM, através do comando: **npm -v** ou **npm --version**.

**Figura 9 – Conferência da instalação do npm**

A terminal window with a dark purple background. The prompt is 'adail@Adail:~\$'. The first command is 'npm -v', which outputs '7.10.0'. The second command is 'npm --version', which is partially visible with a cursor at the end.

Fonte: o autor (2021)

Para realizar a instalação via linha de comando, utilizar os comandos abaixo.

Primeiramente, devemos atualizar todas as bibliotecas de aplicações que podemos utilizar. Este processo, pode ser feito com o comando: **adail@Adail:~\$ sudo apt update**. Em seguida, o sistema solicitará a senha que faz parte do usuário com privilégio *root*.

Após terminar o processo, podemos enfim realizar a instalação via linha de comando.

O comando necessário é: **adail@Adail:~\$ sudo apt install nodejs**

Ao final, apenas para confirmação, verificar se a versão está correta.

```
adail@Adail:~$ node -v  
v14.15.2
```

### Videoaula 3

Assista à videoaula na qual será abordada a instalação Node.js no Linux. Disponível em:  
<https://www.youtube.com/watch?v=mVatUeEcy5Q>



#### Videoaula 3

Utilize o QRcode para assistir!

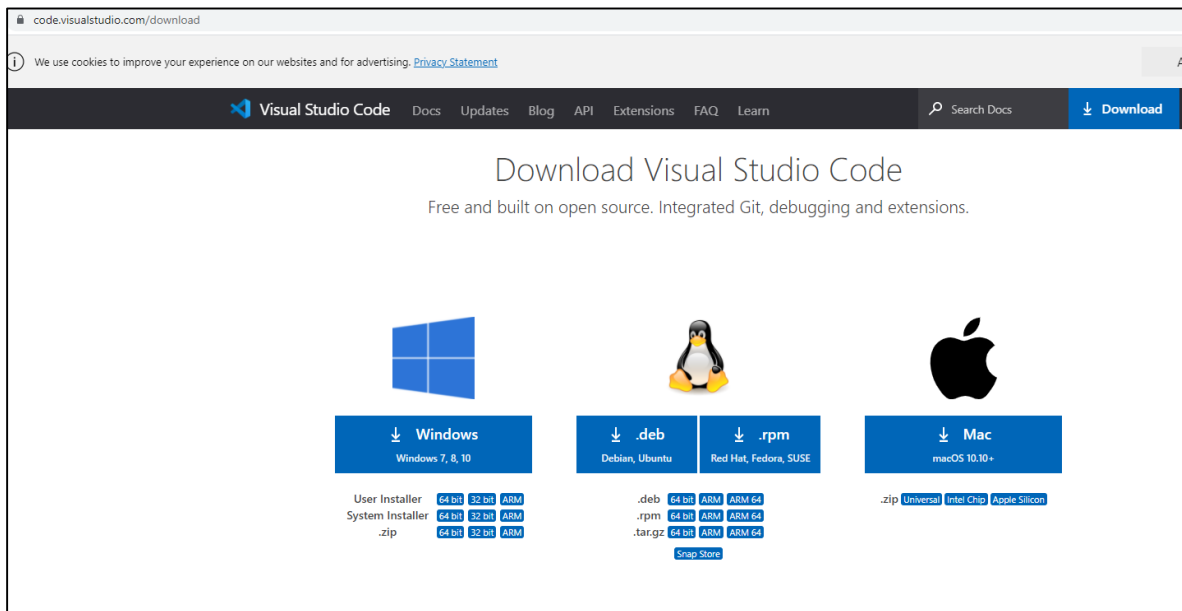
Assista à videoaula na qual será abordada a instalação Node.js no Linux.



## Visual Studio Code – Instalação

Para instalar o Visual Studio Code, precisaremos acessar a página (<https://code.visualstudio.com/download>) para fazer o *download* efetivo da ferramenta.

**Figura 10 – Instalando o Visual Studio Code**

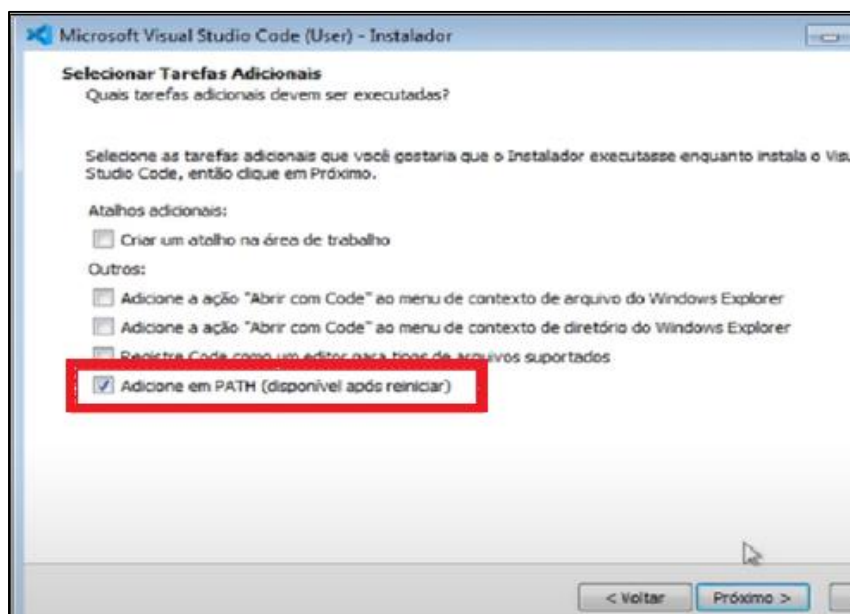


Fonte: disponível em: <https://code.visualstudio.com/download>. Acesso em: 20 abr. 2021.

Após feito o *download* do programa, é necessário clicar sob o arquivo baixado, para iniciar a instalação.

O único ponto que necessita de determinada atenção no ato de instalação, equivale que a cláusula: “Adicione em PATH (disponível após reiniciar)” deve estar marcada.

**Figura 11 - Adicione em PATH na instalação**



Fonte: Visual Studio Code instalador

O bacana desta ferramenta é poderemos utilizar para trabalho em Ionic e também com outras linguagens como PHP.

Após terminar a instalação, clique em Concluir e o programa será aberto.

É possível customizar o tema de acordo com a necessidade do usuário.

O *Visual Studio Code* pode ser acessado através do CMD, utilizando o comando: code.

```
C:\Users\adail>code .
```

Lembrando: O Visual Studio Code irá abrir na página a qual estiver aberta. No exemplo acima, será aberto editando a pasta c:\Users\adail, a qual estamos trabalhando.

#### Videoaula 4

Agora, assista à videoaula na qual abordaremos a instalação do VSCode no Windows.

Disponível em: [https://www.youtube.com/watch?v=90rBv2Cfq\\_s&list=PLAHrXe-C04uAOxDx2G7v1I5hVwE6ruoh5&index=19](https://www.youtube.com/watch?v=90rBv2Cfq_s&list=PLAHrXe-C04uAOxDx2G7v1I5hVwE6ruoh5&index=19). Acesso em: 20 abr. 2021.



#### Videoaula 4

Utilize o QRcode para assistir!

Agora, assista à videoaula na qual abordaremos a instalação do VSCode no Windows.



## Ionic - Instalação e primeiro projeto

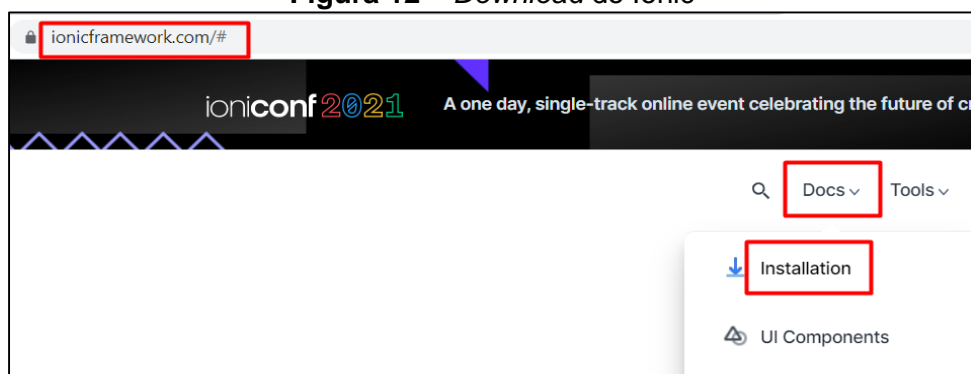
Esta é a última ferramenta que precisamos instalar para utilizar o desenvolvimento de aplicativos com Ionic. Entretanto, para realizar a instalação do Ionic, basicamente precisamos que o Node.js esteja instalado em nossa máquina.

Vá para a página de *download* do Ionic:

[ionicframework.com](https://ionicframework.com)

Acesse a aba **Docs** e em sequência **Installation**, para visualizar os procedimentos de instalação.

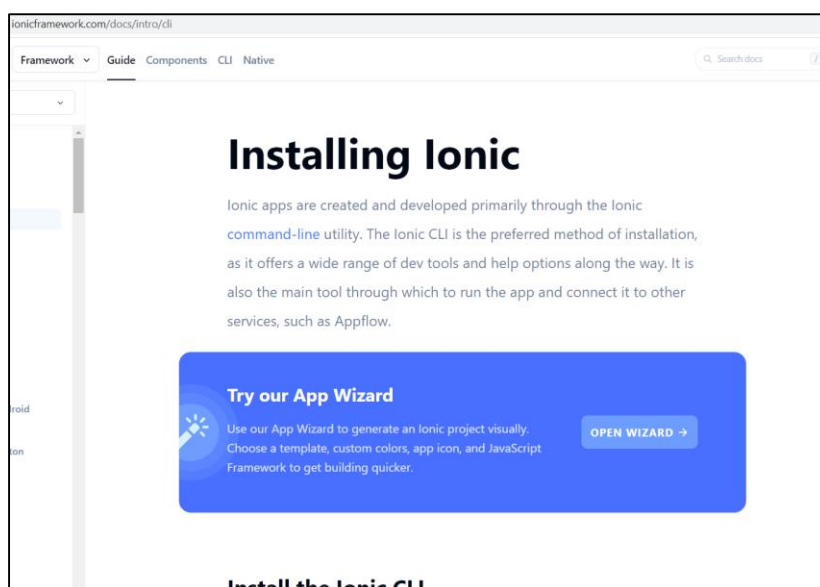
Figura 12 – Download do Ionic



Fonte: disponível em: ionicframework.com. Acesso em: 20 abr. 2021.

Você visualizará uma página como essa:

Figura 13 – Instalando Ionic



Fonte: disponível em: ionicframework.com. Acesso em: 20 abr. 2021.



Primeiramente deve ser verificado se está com a versão do Node.js instalada na máquina. Para isso, digite o comando abaixo:

**Figura 14** – Verificando a versão do Node.js

```
adail@Adail:~$ node -v
v14.15.2
adail@Adail:~$ npm -v
7.10.0
adail@Adail:~$
```

Fonte: o autor (2021)

Em seguida, devemos criar uma pasta para armazenar nossos projetos:

**Figura 15** – Criação de pasta para os projetos

```
adail@Adail:~$ mkdir aula-ddm
adail@Adail:~$ cd aula-ddm/
adail@Adail:~/aula-ddm$ ls
adail@Adail:~/aula-ddm$ dir
adail@Adail:~/aula-ddm$
```

Fonte: o autor (2021)

Agora, podemos executar o comando para instalação:

**Figura 16** – Instalando o Ionic

```
adail@Adail:~$ node -v
v14.15.2
adail@Adail:~$ npm -v
7.10.0
adail@Adail:~$ mkdir aula-ddm
adail@Adail:~$ cd aula-ddm/
adail@Adail:~/aula-ddm$ ls
adail@Adail:~/aula-ddm$ dir
adail@Adail:~/aula-ddm$ npm install -g @ionic/cli
```

Fonte: o autor (2021)

Em seguida, verifique se a versão está correta:

**Figura 17** – Verificando a versão do Ionic

```
adail@Adail:~/aula-ddm$ ionic -v
6.16.3
```

Fonte: o autor (2021)

A partir de então, criaremos nosso primeiro projeto em Ionic.

Após acessar a pasta de projetos na qual quero criar a aplicação, informo o comando para criação do nosso primeiro projeto.

**Figura 18** – Criando o primeiro projeto

```
adail@Adail:~/aula-ddm$ ionic start aula1

Pick a framework! 😊

Please select the JavaScript framework to use for your new app. To bypass this prompt next time, supply a value for the --type option.

? Framework: (Use arrow keys)
> Angular | https://angular.io
  React   | https://reactjs.org
  Vue     | https://vuejs.org
```

Fonte: o autor (2021)

Nesta aula, em específico, utilizamos o Ionic juntamente com o Angular para fazer a renderização. Deste modo, selecione o *Framework* que queira trabalhar.

**Figura 19** – Selecionando um *framework*

```
? Framework:
> Angular | https://angular.io
  React   | https://reactjs.org
  Vue     | https://vuejs.org
```

Fonte: o autor (2021)

Por fim, selecione o *template* que será usado. No nosso caso, selecionaremos o *template black*.

Figura 20 – Selecionando o *template*

```
? Starter template: (Use arrow keys)
> tabs | A starting project with a s
  sidemenu | A starting project with a s
the content area
  blank | A blank starter project
  list | A starting project with a l
  my-first-app | An example application that
```

Fonte: o autor (2021)

A instalação irá perguntar se é necessário acessar as abas nativas do telefone, como câmera, agenda, GPS etc. Clicaremos em Não, para ser mais otimizado.

Figura 21 – Continuação da instalação

```
? Starter template: blank
✓ Preparing directory ./aula1 in 1.22ms
✓ Downloading and extracting blank starter in 566.06ms
? Integrate your new app with Capacitor to target native iOS and Android
? (y/N) █
```

Fonte: o autor (2021)

A instalação será finalizada e algumas mensagens de aviso podem ser apresentadas.

Figura 22 – Mensagens de aviso da instalação

```
adali@Adali: ~/aula-ddm
✓ Preparing directory ./aula1 in 1.22ms
✓ Downloading and extracting blank starter in 566.06ms
? Integrate your new app with Capacitor to target native iOS and Android? N
Installing dependencies may take several minutes.

Ionic Appflow, the mobile DevOps solution by Ionic

Continuously build, deploy, and ship apps 🚀
Focus on building apps while we automate the rest 📦

👉 https://ion.link/appflow 👈

> npm i
[.....] / reify:fsevents: timing reifyNode:node modules/fseven
```

Fonte: o autor (2021)

A ferramenta irá solicitar caso seja necessário criar uma conta no Ionic. No nosso caso, não iremos criar.

**Figura 23** – Aviso quanto à criação de conta no Ionic

```
adail@Adail: ~/aula-ddm
run `npm fund` for details

14 vulnerabilities (4 moderate, 10 high)

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
> git init
Initialized empty Git repository in /home/adail/aula-ddm/aula1/.git/

Join the Ionic Community! ❤️

Connect with millions of developers on the Ionic Forum and get access to l
events, news updates, and more.

? Create free Ionic account? (y/N) █
```

Fonte: o autor (2021)

Por fim, o nosso projeto foi criado com sucesso.

**Figura 24** – Projeto criado

```
adail@Adail: ~/aula-ddm
create mode 100644 src/test.ts
create mode 100644 src/theme/variables.scss
create mode 100644 src/zone-flags.ts
create mode 100644 tsconfig.app.json
create mode 100644 tsconfig.json
create mode 100644 tsconfig.spec.json

Your Ionic app is ready! Follow these next steps:

- Go to your new project: cd ./aula1
- Run ionic serve within the app directory to see your app in the browser
- Run ionic capacitor add to add a native iOS or Android project using Capa
or
- Generate your app icon and splash screens using cordova-res --skip-config
--copy
- Explore the Ionic docs for components, tutorials, and more:
https://ion.link/docs
- Building an enterprise app? Ionic has Enterprise Support and Features:
https://ion.link/enterprise-edition
adail@Adail:~/aula-ddm$
```

Fonte: o autor (2021)



Em todos os momentos que criarmos um novo projeto em Ionic, é necessário acessar o endereço da pasta para trabalhar com o projeto.

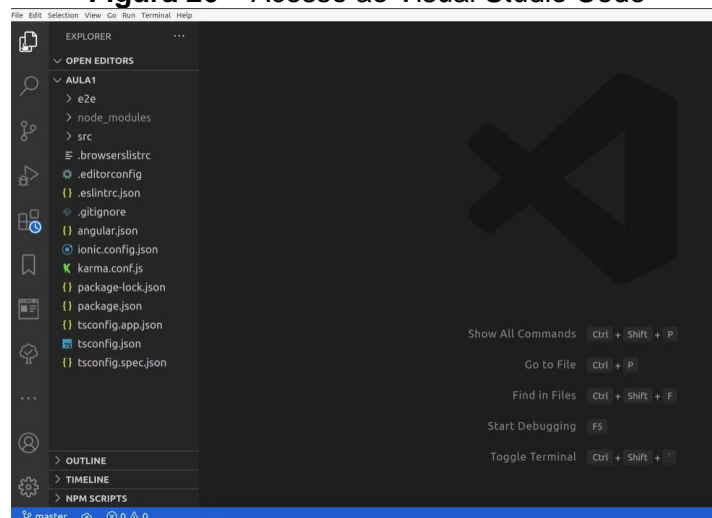
**Figura 25** – Acessando o endereço da pasta

```
drwxrwxr-x  3 adail adail 4096 jun 16 23:20 .  
drwxr-xr-x 153 adail adail 20480 jun 16 23:15 ..  
drwxr-xr-x  6 adail adail 4096 jun 16 23:23 aula1  
adail@Adail:~/aula-ddm$ cd aula1/  
adail@Adail:~/aula-ddm/aula1$
```

Fonte: o autor (2021)

Algumas subpastas foram criadas e será necessário acessar o Visual Studio Code.

**Figura 26** – Acesso ao Visual Studio Code



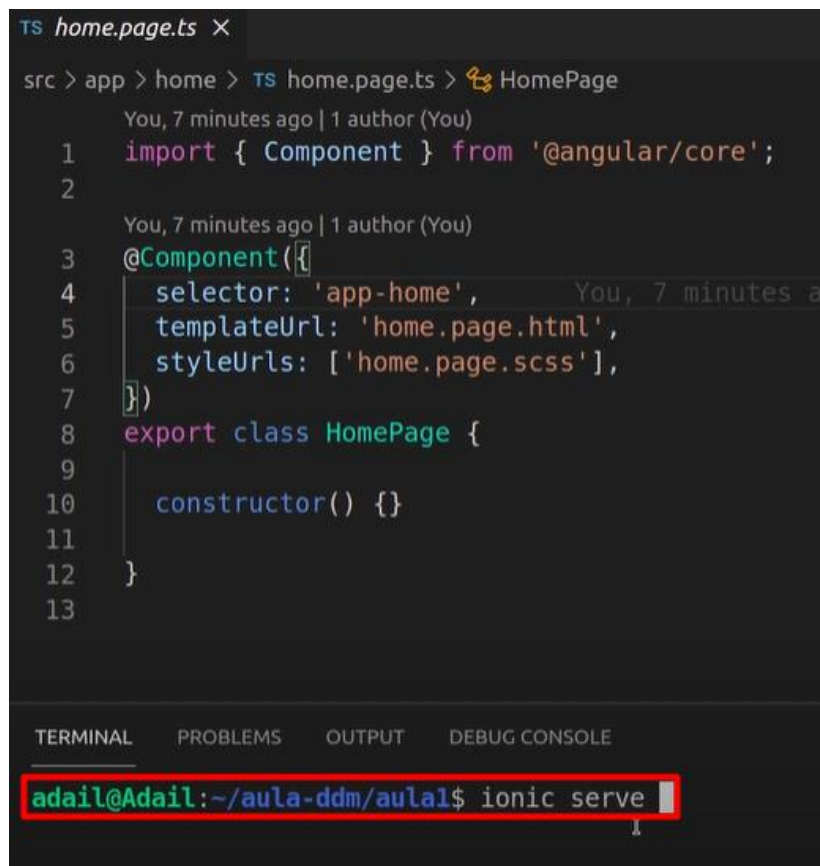
Fonte: Visual Studio Code

Em sequência podemos abrir um terminal de dentro do Visual Studio Code, clicando em terminal > *new terminal*.

Em seguida, realize a troca de bash para cmd.

- Comando para executar, ou seja, ver determinado projeto funcionando.

**Figura 27** – Executando comando



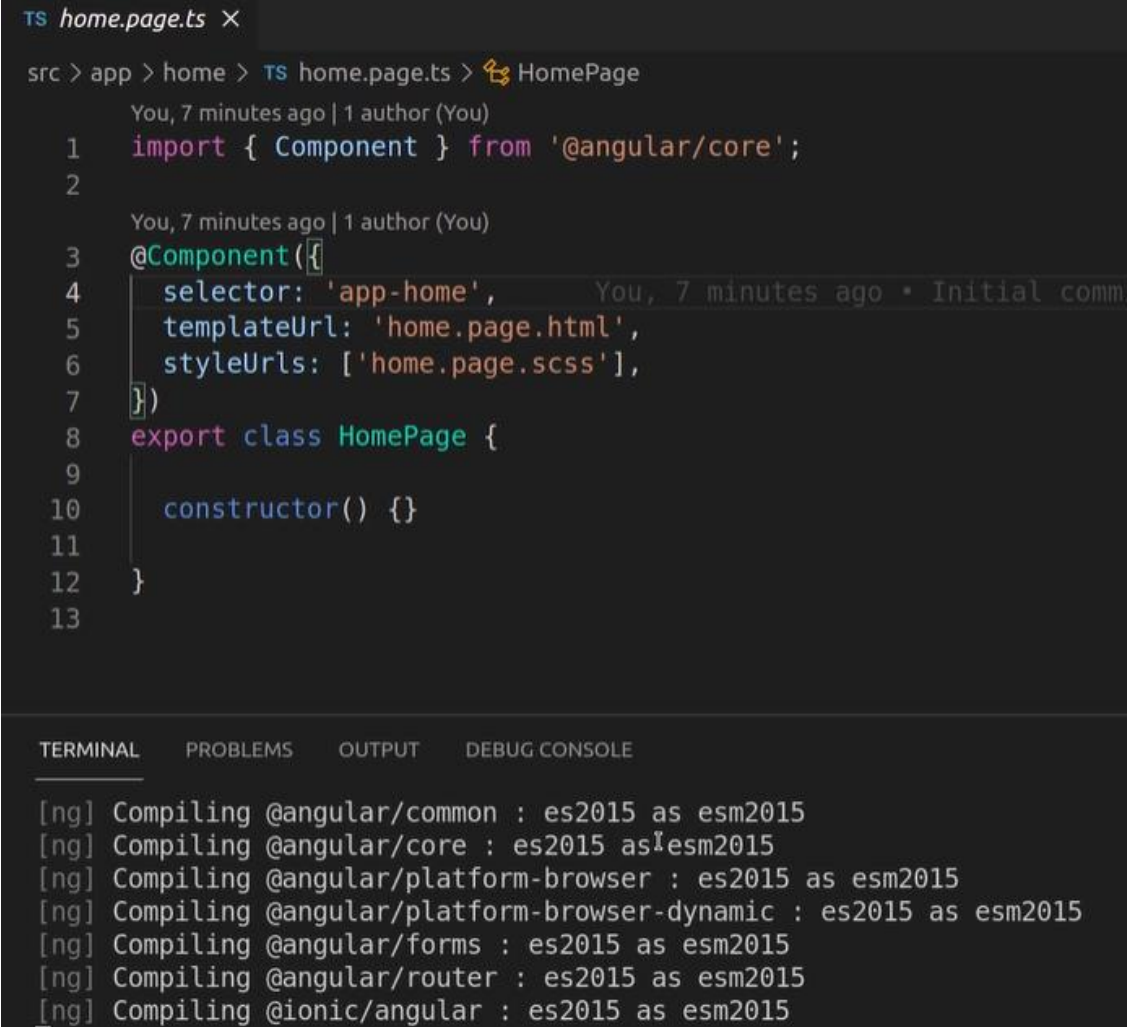
The image shows a code editor window with a file named `home.page.ts`. The code defines an Angular component named `HomePage`. It imports `Component` from `@angular/core` and uses the `@Component` decorator with the following properties: `selector: 'app-home'`, `templateUrl: 'home.page.html'`, and `styleUrls: ['home.page.scss']`. The component class `HomePage` has an empty `constructor()` method. Below the code editor, the `TERMINAL` tab is active, showing the command `adail@Adail:~/aula-ddm/aula1$ ionic serve` being executed. The terminal prompt and command are highlighted with a red rectangle.

```
TS home.page.ts X
src > app > home > TS home.page.ts > HomePage
You, 7 minutes ago | 1 author (You)
1 import { Component } from '@angular/core';
2
You, 7 minutes ago | 1 author (You)
3 @Component({
4   selector: 'app-home',
5   templateUrl: 'home.page.html',
6   styleUrls: ['home.page.scss'],
7 })
8 export class HomePage {
9
10   constructor() {}
11
12 }
13

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
adail@Adail:~/aula-ddm/aula1$ ionic serve
```

Em seguida, será compilado através do Angular.

Figura 28 – Compilando com Angular



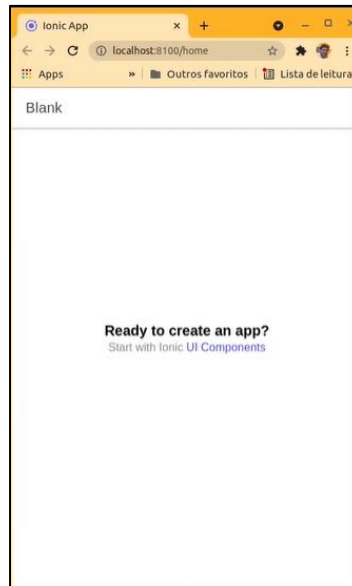
```
TS home.page.ts X
src > app > home > TS home.page.ts > HomePage
You, 7 minutes ago | 1 author (You)
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-home',
5   templateUrl: 'home.page.html',
6   styleUrls: ['home.page.scss'],
7 })
8 export class HomePage {
9
10   constructor() {}
11
12 }
13

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
[ng] Compiling @angular/common : es2015 as esm2015
[ng] Compiling @angular/core : es2015 as esm2015
[ng] Compiling @angular/platform-browser : es2015 as esm2015
[ng] Compiling @angular/platform-browser-dynamic : es2015 as esm2015
[ng] Compiling @angular/forms : es2015 as esm2015
[ng] Compiling @angular/router : es2015 as esm2015
[ng] Compiling @ionic/angular : es2015 as esm2015
```

Fonte: Visual Studio Code

E por fim a tela da nossa aplicação será aberta.

**Figura 29** – Aplicação em uso



Fonte: o autor (2021)

## **Ionic – Criar páginas e Rotas**

Daremos continuidade à aula passada criando novas páginas para a aplicação. Basicamente, instalamos o Ionic, executamos o Ionic start para criar uma aplicação do tipo Angular com o *template* do tipo *blank*.

Primeiramente devemos executar o comando:

```
adail@Adail:~/aula-ddm/aula1$ ionic serve --lab
```

na qual irá instalar o Ionic Lab.

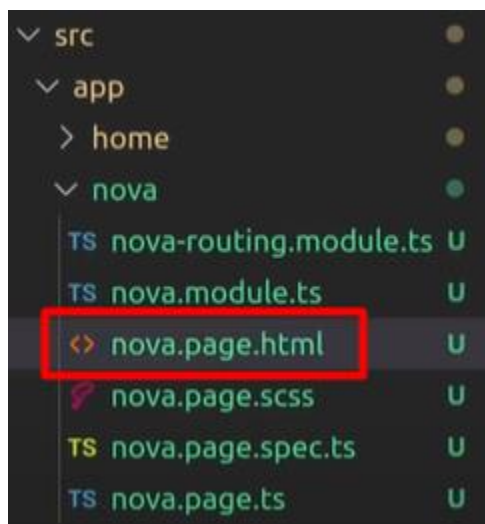
A partir do momento que temos a aplicação funcionando, é possível realizar algumas modificações nesta aplicação.

Para criar mais páginas na nossa aplicação, devemos criar em Terminal/New Terminal e em seguida informar o comando: ionic generate page + nome da página que queremos criar.

```
adail@Adail:~/aula-ddm/aula1$ ionic generate page nova
> ng generate page nova --project=app
```

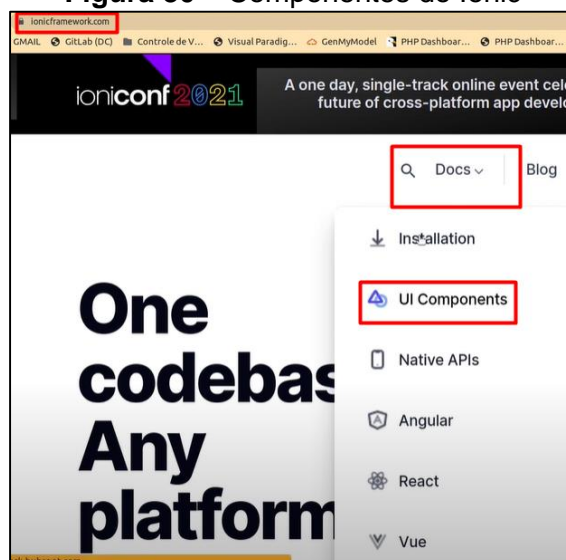


A seguir, podemos verificar a nova página criada:



Acessar a página de documentação do Ionic, cujo endereço é: [ionicframework.com](https://ionicframework.com) e em seguida clicar em Docs / UI Components.

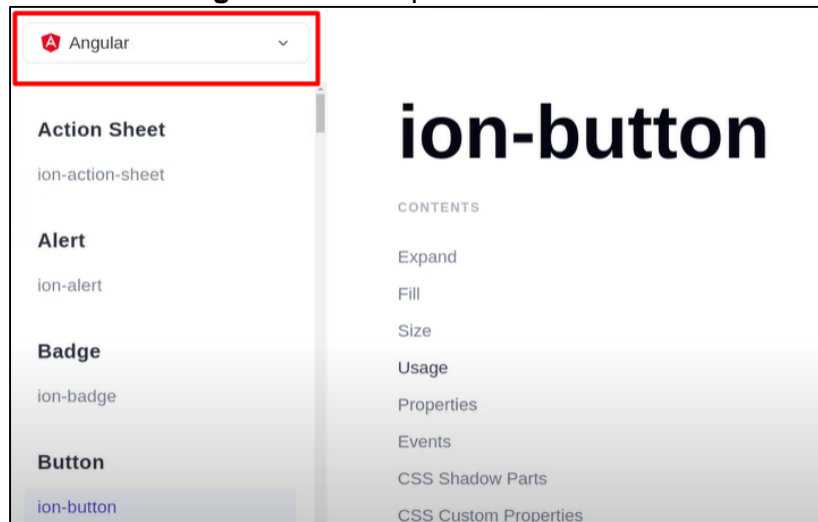
**Figura 30 – Componentes do Ionic**



Fonte: disponível em: [ionicframework.com](https://ionicframework.com). Acesso em: 20 abr. 2021.

No nosso caso, procuraremos o componente ion-button. Um ponto muito importante é sempre conferir se está selecionado como Angular.

**Figura 31** – Componente ion-button



Fonte: disponível em: ionicframework.com. Acesso em: 20 abr. 2021.

No corpo desta página, temos vários exemplos de botões para utilização. Exemplo:

```
<ion-button>Default</ion-button>
```

Vamos copiar este código acima e retornaremos para a nossa aplicação, colando dentro da nossa página criada.

**Figura 32** – Acessando a página criada



Fonte: Visual Studio Code

OBS: o que está escrito como *Default* diz respeito ao nome do botão. Este nome, pode ser alterado conforme a preferência.

Em seguida, inserir o atributo chamado **routerLink= "/nova"**

```
<ion-button routerLink="/nova" >Nova Pagina</ion-but
```

Desta forma, quando clicado no botão, será enviado para a página nova.

Em seguida, na nova página faremos o mesmo processo, apenas alterando o nome do botão sinalizando para o usuário retornar para a *Home*.

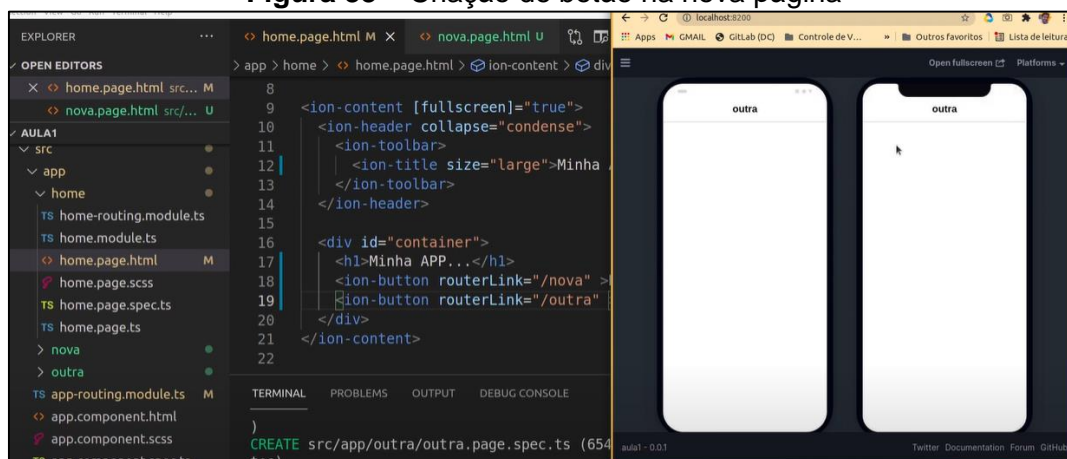
```
<ion-content>
<h1>Nova pagina</h1>
<ion-button routerLink="/home">Voltar</ion-button>
</ion-content>
```

Por agora, faremos a criação de outra página chamada “outra”, utilizando o comando já explicado nesta aula (ionic g page outra)

```
adail@Adail:~/aula-ddm/aula1$ ionic g page outr
a
```

Desta forma, realizaremos a criação do botão na nova página criada, conforme imagem abaixo:

**Figura 33 – Criação de botão na nova página**

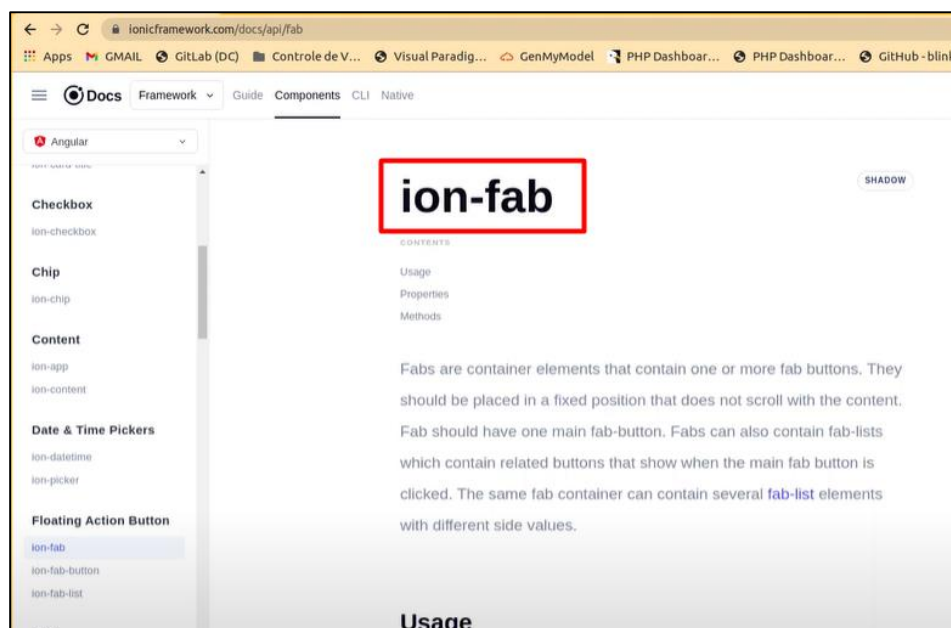


Fonte: Visual Studio Code

Podemos utilizar vários tipos de botões em nossas aplicações. Um outro exemplo de botão será demonstrado abaixo.

Primeiramente, navegar na documentação até ion-fab.

**Figura 34 – Acessando a documentação**



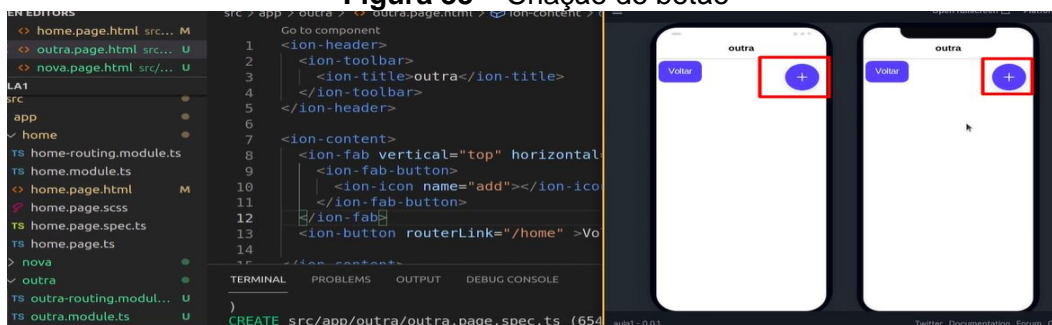
Fonte: disponível em: ionicframework.com. Acesso em: 20 abr. 2021.

Copiar o código:

```
<ion-fab vertical="top" horizontal="end" slot="fixed">  
  <ion-fab-button>  
    <ion-icon name="add"></ion-icon>  
  </ion-fab-button>  
</ion-fab>
```

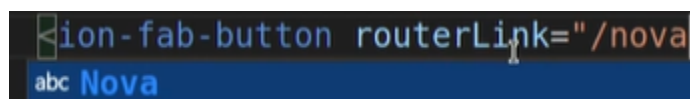
Colar dentro da página “Outra” que criamos a pouco. Desta forma, criaremos o botão conforme demonstrado abaixo:

Figura 35 – Criação do botão



Fonte: Visual Studio Code

Inserir o atributo para proporcionar ação ao botão.



## Videoaula 5

Agora, assista esta videoaula que aborda um detalhamento sobre como Criar Páginas e Rotas de direcionamento no IONIC.

Disponível em: <https://www.youtube.com/watch?v=URAKJYmIOC8>. Acesso em: 20 abr. 2021.



### Videoaula 5

Utilize o QRcode para assistir!

Agora, assista esta videoaula que aborda um detalhamento sobre como Criar Páginas e Rotas de direcionamento no IONIC.



## Videoaula 6

Agora, assista esta videoaula.

Disponível em: <https://www.youtube.com/watch?v=M0WuwU8ibv4>. Acesso em: 20 abr. 2021.



### Videoaula 6

Utilize o QRcode para assistir!

Agora, assista esta videoaula.



## Encerramento da Unidade

Considerando os pontos abordados nesta unidade, desde a introdução para as linguagens de programação até o ponto na qual criamos nosso primeiro aplicativo utilizando IONIC, destaco a importância e relevância para:

1. Definir qual linguagem de programação será utilizada na estruturação do projeto.
2. Abordar e estruturar a criação de ícones, páginas para diversas aplicações a qual deve sempre se colocar em prol de atender às expectativas do usuário final.

Agora que você completou esta unidade, você deve ser capaz de:

- Discutir a respeito de HTML;
- Discutir a respeito de CSS e SCSS;
- Discutir a respeito de JavaScript e TypeScript;
- Utilizar a IDE de desenvolvimento (Visual Studio Code);
- Criar um projeto básico com IONIC.

## Referências

FÉLIX, Rafael; SILVA, Everaldo Leme. **Arquitetura para computação móvel**. 2. ed. São Paulo: Pearson Education do Brasil, 2019.

FLATSCHART, Fábio. **HTML 5: Embarque Imediato**. Rio de Janeiro: Editora Brasport, 2011.

LEE, Valentino; SCHNEIDER, Heather; SCHELL, Robbie. **Aplicações móveis: arquitetura, projeto e desenvolvimento**. São Paulo: Pearson Makron Books, 2005.

MARINHO, Antonio Lopes (org.). **Desenvolvimento de aplicações para Internet**. São Paulo: Pearson Education do Brasil, 2016.

SEGURADO, Valquiria Santos (org.). **Projeto de interface com o usuário**. 1. ed. São Paulo: Pearson Education do Brasil, 2015.

SILVA, Diego (org). **Desenvolvimento para dispositivos móveis**. São Paulo: Pearson Education do Brasil, 2016.

SOMMERVILLE, Ian. **Engenharia de software**. Tradução Luiz Claudio Queiroz. Revisão técnica Fábio Levy Siqueira. 10. ed. São Paulo: Pearson, 2018. 768 p.

Publicações da SBC (<http://www.sbc.org.br/index.php?Itemid=196>)

Portal Periódicos. CAPES (<http://www.periodicos.capes.gov.br>)

Google Acadêmico (<http://scholar.google.com.br>)

SciELO (<http://www.scielo.br>)

arXiv.org (<http://arxiv.org>)

ACM Digital Library (<http://dl.acm.org>)

IEEE Xplore (<http://ieeexplore.ieee.org/Xplore/home.jsp>)

ScienceDirect (<http://www.sciencedirect.com>)



UNIFIL.BR