



# CURSO DE PROGRAMAÇÃO FULL STACK

Programação Front-end e Back-end



## Sumário

Introdução ao Git e GitHub .....	8
1.1 O que é Git? .....	8
1.2 Como instalar o Git .....	9
1.3 Primeiros passos o Git.....	10
1.4 O que é GitHub?.....	11
1.4.1 Criar conta Github.....	12
1.4.2 Criar repositório Git.....	15
1.4.3 Clonar repositório Git.....	18
1.5 Comandos básicos do Git.....	18
1.5.1 git init.....	19
1.5.2 git clone .....	19
1.5.3 git add.....	21
1.5.4 git status .....	21
1.5.5 git diff.....	22
1.5.6 git commit.....	22
1.5.7 git reset .....	23
1.5.8 git rm.....	23
1.5.9 git mv .....	24
1.6 Branch e Merge .....	24
1.6.1 git branch.....	25
1.6.2 git checkout .....	25
1.6.3 git merge.....	25
1.6.4 git log.....	25
1.6.4 git stash .....	25
1.6.5 git tag.....	26
1.7 Compartilhar e Atualizar Projetos.....	26
1.7.1 git fetch .....	26
1.7.1 git pull.....	27
1.7.2 git push .....	27
Introdução ao HTML .....	29

2.1 O que é HTML e suas tags? .....	29
2.2 Fundamentos do texto em HTML .....	30
2.2.1 O básico: Cabeçalhos e Parágrafos .....	30
2.3 Tipos de tag.....	31
2.4 Estrutura básica.....	32
2.4.1 <!DOCTYPE html>.....	33
2.4.2 <html> .....	33
2.4.3 <body>.....	33
2.4.4 <script> .....	33
2.4.5 <head>.....	34
2.4.6 <meta> .....	34
2.4.7 <link> .....	34
2.4.8 <style> .....	34
2.5 HTML: Tags semânticas .....	34
2.5 HTML: Tags sem semântica.....	36
2.6 HTML: Comentários .....	36
2.7 Elementos inline .....	37
2.7.1 <span>.....	38
2.7.2   .....	38
2.7.3 <a>.....	38
2.7.4 <img loading="lazy"> .....	38
2.7.5 <audio> .....	39
2.8 Elementos block level .....	39
2.8.1 <header> .....	39
2.8.2 <main> .....	40
2.8.3 <footer> .....	40
2.8.4 <section> .....	41
2.8.5 <article> .....	41
2.8.6 <aside> .....	41
2.8.9 <nav> .....	41
2.8.10 <div> .....	42
2.8.11 <p> .....	42
2.8.12 <h1>, <h2><h6> .....	42

2.8.13 <hr> .....	42
2.8.14 <video> .....	42
2.9 Elementos de um formulário .....	43
2.9.1 <form> .....	43
2.9.1 <input> .....	44
2.9.2 <textarea> .....	45
2.9.3 <button> .....	45
2.9.4 <label> .....	45
2.10 O que são atributos HTML? .....	46
2.10.1 class .....	46
2.10.2 id .....	47
2.10.3 src .....	47
2.10.4 alt .....	47
2.10.5 href .....	48
2.10.6 lang .....	48
2.10.7 target .....	48
Introdução ao CSS .....	49
3.1 Para que serve o CSS? .....	50
3.2 Sintaxe CSS .....	50
3.2.1 Módulos CSS .....	52
3.2.2 Especificações CSS .....	52
3.2.3 Especificações CSS .....	53
3.3 Como o CSS é estruturado .....	54
3.3.1 Folha de Estilos Externa .....	54
3.3.2 Folha de Estilos Interna .....	56
3.3.3 Estilos Inline .....	57
3.4 Seletor CSS .....	58
3.4.1 O que é um seletor .....	58
3.4.2 Lista de seleção .....	59
3.4.3 Tipos de seletores .....	60
3.5 Unidades no CSS .....	61
3.5.1 Medidas absolutas .....	62
3.5.1 Medidas relativas .....	62

3.5.1 Medidas absolutas no CSS .....	63
3.5.2 Medidas Relativas no CSS .....	67
3.6 Flexbox CSS .....	76
3.6.1 O que é o Flexbox .....	76
3.6.2 Elementos .....	76
3.6.3 Propriedades para o elemento-pai .....	78
3.6.4 Propriedades para elementos-filhos .....	89
Introdução ao Javascript .....	94
4.1 História do JavaScript .....	94
4.2 Bibliotecas JS, jogos e aplicações .....	95
4.3 Variáveis no JavaScript .....	96
4.3.1 Tipos de variáveis JavaScript .....	96
4.3.2 Tipos de variáveis JavaScript .....	97
4.3.3 var .....	98
4.3.4 let .....	99
4.3.5 const .....	99
4.4 Tipos de dados no JavaScript .....	101
4.5 Hello world em JavaScript .....	101
4.6 Funções JavaScript .....	103
4.7 JavaScript: Estrutura condicional .....	105
4.7.1 if .....	105
4.7.2 else .....	107
4.7.3 else if .....	108
4.7.4 if ternário .....	108
4.7.5 Operador && .....	109
4.8 JavaScript: Estruturas de repetição .....	110
4.8.1 O que é estrutura de repetição? .....	110
4.8.2 Quando devemos usar as estruturas de repetição? .....	110
4.8.3 While .....	111
4.8.4 Do While .....	114
4.8.5 For .....	116
4.8.6 Foreach .....	118
Banco de dados Relacional (SQL) .....	121

<b>5.1 Quais são os principais tipos de banco de dados? .....</b>	<b>123</b>
5.1.1 Oracle .....	123
5.1.2 SQL Server .....	124
5.1.3 MySQL .....	124
5.1.4 PostgreSQL .....	125
5.1.5 NoSQL .....	125
<b>5.2 Linguagem de Definição de Dados (DDL) .....</b>	<b>125</b>
5.2.1 Criando tabelas no MySQL .....	127
5.2.2 Alterando a tabela .....	129
<b>5.3 Linguagem de Manipulação de Dados (DML) e Linguagem de Transação de Dados (DTL) .....</b>	<b>130</b>
5.3.1 Inserindo registros .....	130
5.3.2 Alterando registros .....	131
5.3.3 Excluindo registros .....	132
<b>5.4 Linguagem de Consulta de Dados (DQL) .....</b>	<b>133</b>

# **Versionamento e gestão de código fonte**

## Introdução ao Git e GitHub

Neste capítulo, você vai aprender o que é o **Git**, um sistema de controle de versão distribuído que permite gerenciar e colaborar em projetos de software. Você também vai conhecer o **GitHub**, uma plataforma online que hospeda repositórios Git e facilita o compartilhamento e a integração de código. Além disso, você vai aprender os conceitos e comandos básicos do Git, como criar e clonar repositórios, **adicionar e remover arquivos**, fazer **commits** e **pushs**, e usar **branches** e **merges**. Por fim, você vai ver como usar o **Git Flow**, um fluxo de trabalho que define boas práticas para organizar e manter projetos Git.



### 1.1 O que é Git?

Git é um **sistema de controle de versão distribuído**, ou seja, uma ferramenta que permite **registrar e acompanhar as mudanças feitas em um conjunto de arquivos ao longo do tempo**. Com o Git, você pode criar um histórico das alterações que realizou em seu projeto, bem como voltar para versões anteriores se necessário. Além disso, você pode trabalhar em diferentes ramificações (branches) do seu projeto, criando e testando novas funcionalidades sem afetar o código principal. E o melhor de tudo, você pode fazer isso de forma distribuída, ou seja, sem depender de um servidor central. Cada



cópia do seu projeto é um repositório Git completo, que contém todo o histórico de alterações e pode ser sincronizado com outras cópias remotas.

## 1.2 Como instalar o Git

Para usar o Git, você precisa instalar o cliente Git no seu computador. O cliente Git é um programa que permite executar os comandos do Git e se comunicar com os repositórios remotos. A instalação do cliente Git varia de acordo com o seu sistema operacional. Veja a seguir os passos para cada um deles:

- **Windows:** Acesse o site <https://git-scm.com/> e baixe o instalador do Git. Execute o arquivo baixado e siga as instruções na tela. Você pode aceitar as configurações padrão ou alterá-las de acordo com a sua preferência. Após a instalação, você pode abrir o Git Bash, um terminal que permite usar o Git no Windows, ou usar o Git a partir de qualquer outro terminal, como o PowerShell ou o CMD.
- **Linux:** Dependendo da sua distribuição Linux, você pode instalar o Git usando um gerenciador de pacotes, como o apt, o yum, o pacman ou o zypper. Por exemplo, no Ubuntu, você pode executar o seguinte comando no terminal: `sudo apt install git`. Após a instalação, você pode usar o Git a partir de qualquer terminal.
- **MacOS:** Acesse o site <https://git-scm.com/download/mac> e baixe o instalador do Git. Execute o arquivo baixado e siga as instruções na tela. Você pode aceitar as configurações padrão ou alterá-las de acordo com a sua preferência. Após a instalação, você pode abrir o Terminal e usar o Git a partir dele.

Para verificar se o Git foi instalado corretamente, execute o seguinte comando no terminal:

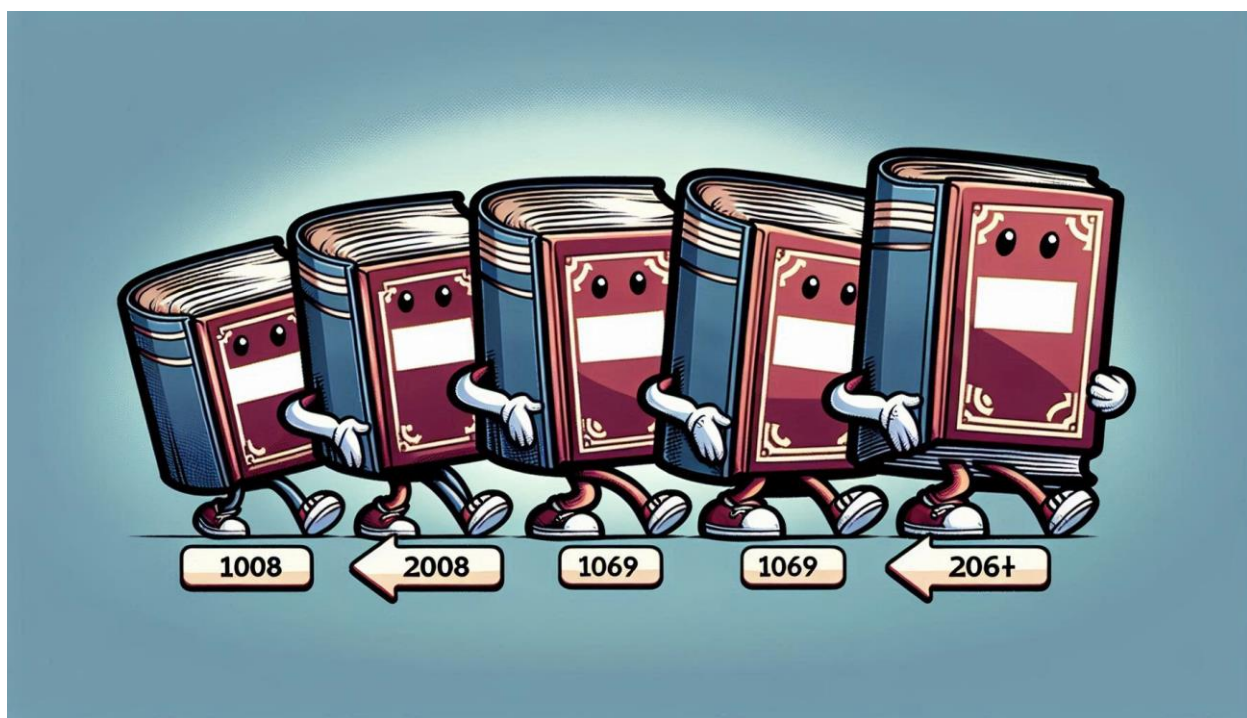
**git --version**

Você deverá ver a versão do Git que você instalou, por exemplo, **git version 2.33.0**.

Se você não vir a versão do Git, verifique se o caminho do Git está adicionado à variável de ambiente PATH do seu sistema. Você também pode consultar a documentação oficial do Git em <https://git-scm.com/doc> para mais detalhes sobre a instalação e configuração do Git.

### 1.3 Primeiros passos o Git

Vamos começar a entender o Git com uma analogia simples. Imagine que você está escrevendo um livro sobre programação e quer controlar as versões do seu texto. Uma forma simples seria salvar o arquivo com um nome diferente cada vez que fizer uma alteração significativa, por exemplo: **livro\_v1.docx**, **livro\_v2.docx**, **livro\_v3.docx**, e assim por diante. Dessa forma, você teria um histórico das versões do seu livro e poderia compará-las ou restaurá-las quando quisesse.



No entanto, esse método tem alguns problemas:

**Espaço no disco:** Guardar vários arquivos duplicados ocupa muito espaço.

**Precisão:** Você não sabe exatamente o que mudou de uma versão para outra, apenas que houve uma alteração.

**Segurança:** Se você perder ou danificar o seu computador, pode perder todo o seu trabalho.

Uma forma melhor de controlar as versões do seu livro seria usar o Git. Com o Git, você não precisa salvar vários arquivos com nomes diferentes, mas sim registrar as mudanças que fez em cada arquivo, chamadas de **commits**. Cada commit tem um **identificador único**, uma **mensagem explicativa** e uma **referência ao commit anterior**, formando uma cadeia de commits que representa o **histórico do seu projeto**.

Com o Git, você pode:

- Visualizar os commits que fez.
- Comparar as diferenças entre eles.
- Voltar para qualquer commit que quiser, sem perder as alterações posteriores.

Além disso, você pode criar **branches** (ramificações) do seu projeto para desenvolver novas funcionalidades ou corrigir erros sem interferir no código principal, que fica no branch master. Depois, você pode juntar os branches com o comando merge, que combina as alterações feitas em cada um, resolvendo possíveis conflitos. Dessa forma, você tem mais flexibilidade e segurança para trabalhar no seu projeto.

#### 1.4 O que é GitHub?

GitHub é uma plataforma online que hospeda repositórios Git, facilitando o compartilhamento e a integração de código entre desenvolvedores. Com o GitHub, você pode armazenar seus projetos Git na nuvem, acessá-los de qualquer lugar e colaborar com outros desenvolvedores. O GitHub oferece vários recursos que tornam o seu trabalho mais fácil e produtivo, como:

- **Interface Gráfica:** Visualize e gerencie seus repositórios Git de forma intuitiva.
- **Ferramentas de Colaboração:** Crie e gerencie issues, pull requests, code reviews e outros aspectos do desenvolvimento de software.
- **Integração com Outros Serviços:** Conecte-se facilmente a plataformas como Visual Studio Code, Heroku, Travis CI, Slack, entre outras.
- **Comunidade Ativa:** Participe de uma comunidade com milhões de desenvolvedores que compartilham e contribuem para projetos de código aberto.

Esses recursos e muitos outros fazem do GitHub uma ferramenta essencial para qualquer desenvolvedor, seja iniciante ou experiente.

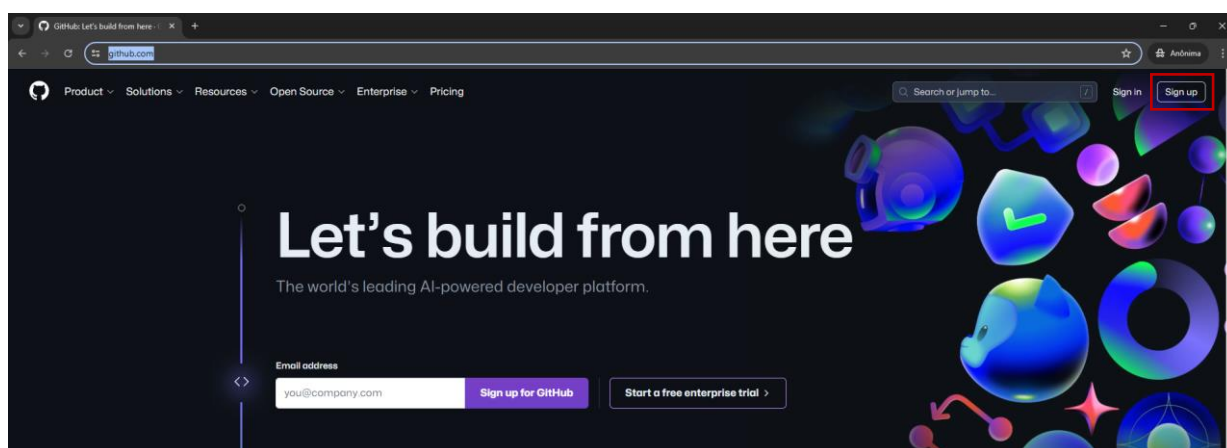
### Benefícios do GitHub

- **Armazenamento na Nuvem:** Seus projetos ficam seguros e acessíveis de qualquer lugar.
- **Colaboração Facilitada:** Trabalhe com outros desenvolvedores de forma eficiente, utilizando ferramentas integradas.
- **Visibilidade e Portfólio:** Compartilhe seu trabalho com a comunidade, construa um portfólio e receba feedback de outros profissionais.
- **Automação de Processos:** Utilize integrações para automatizar testes, implantações e outras tarefas repetitivas.

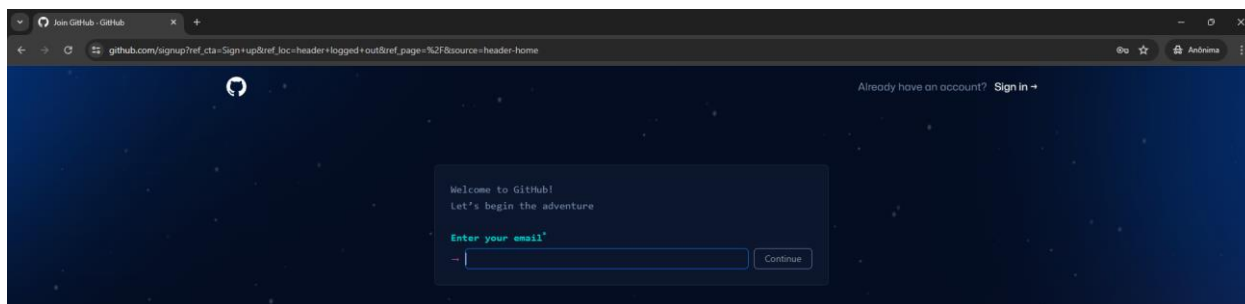
Ao usar o GitHub, você não só melhora sua produtividade, mas também se conecta a uma rede global de desenvolvedores, o que pode abrir portas para novas oportunidades e aprendizados.

#### 1.4.1 Criar conta Github

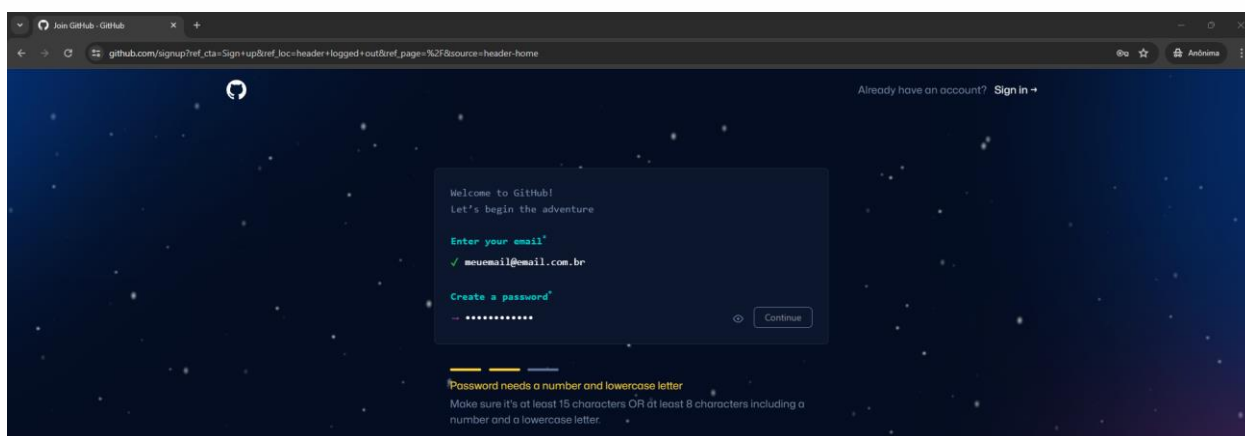
Para usar o GitHub, você precisa criar uma conta gratuita no site <https://github.com/>. Acesse o endereço acima e clique na opção “**Sign up**”, no canto superior direito da tela.



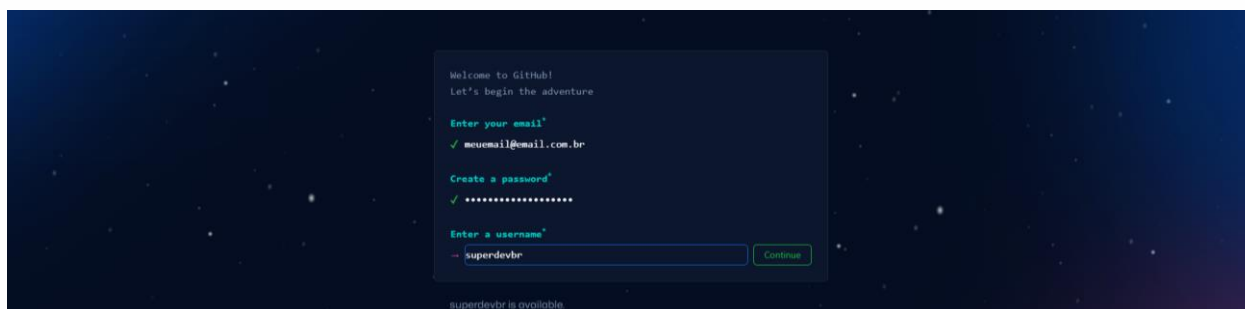
Na próxima etapa você deverá informar um e-mail válido, que será utilizado posteriormente para realizar o acesso à sua conta GitHub. Após preencher o e-mail, clique no botão “**Continue**”.



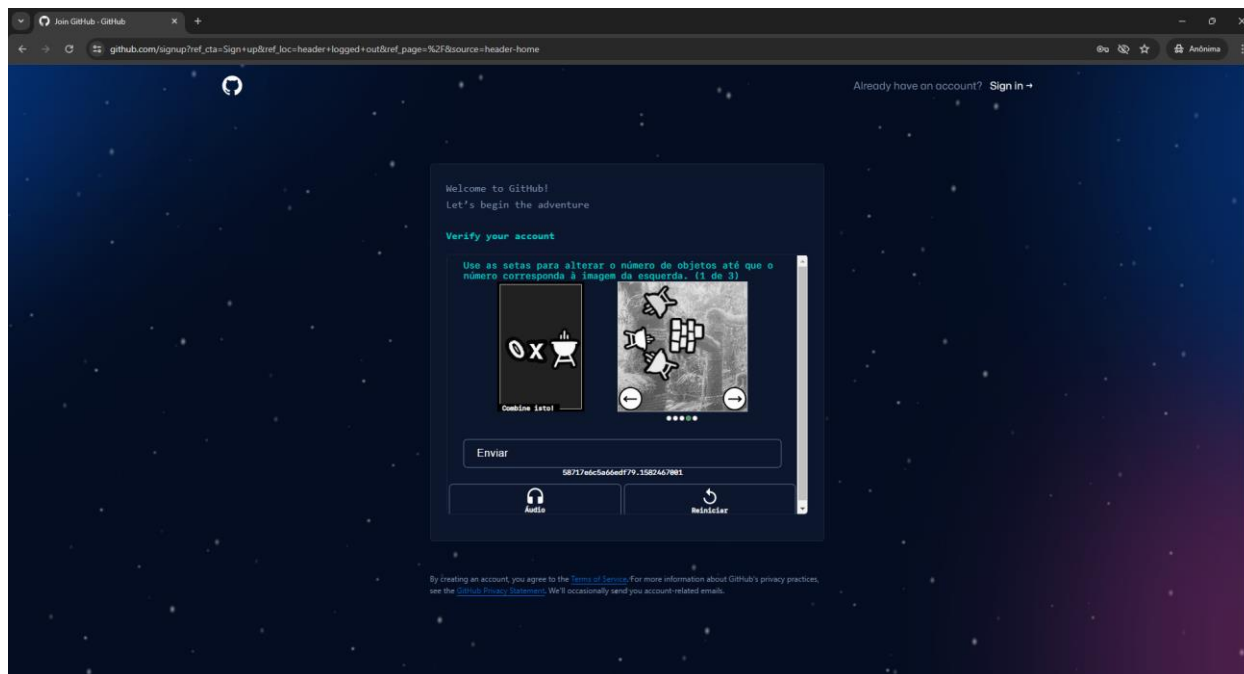
Na próxima etapa, você deverá informar uma senha. Dica: A senha ideal, tem ao menos 8 caracteres, com letras maiúsculas, minúsculas, números e caracteres especiais.



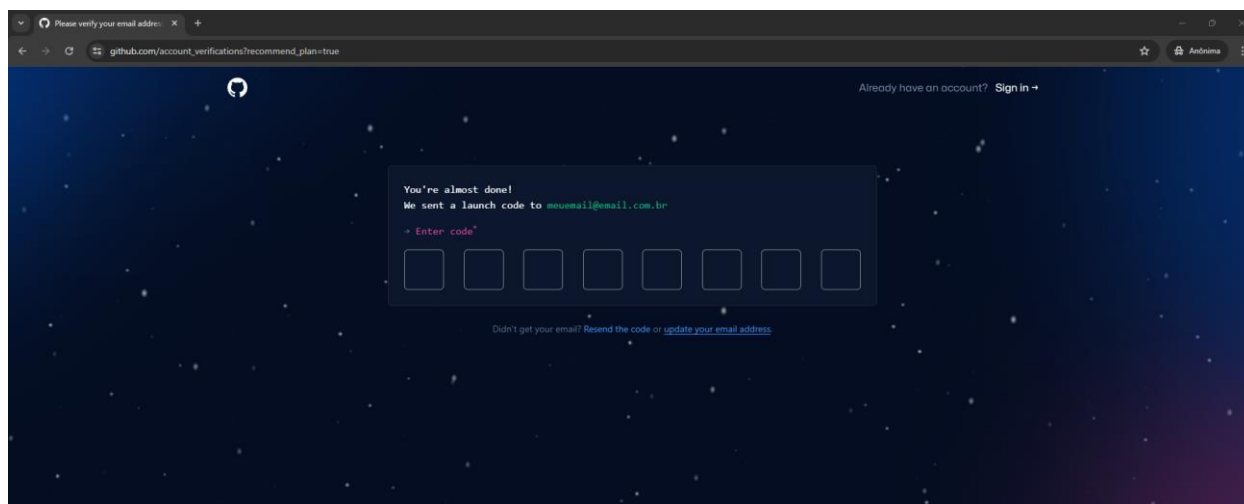
Por fim você poderá escolher seu nome de usuário (username) para o Github. O username contribui para a sua privacidade. Você pode informar ele ao invés do e-mail para ser adicionados a organizações, repositórios e projetos, deixando o seu e-mail privado.



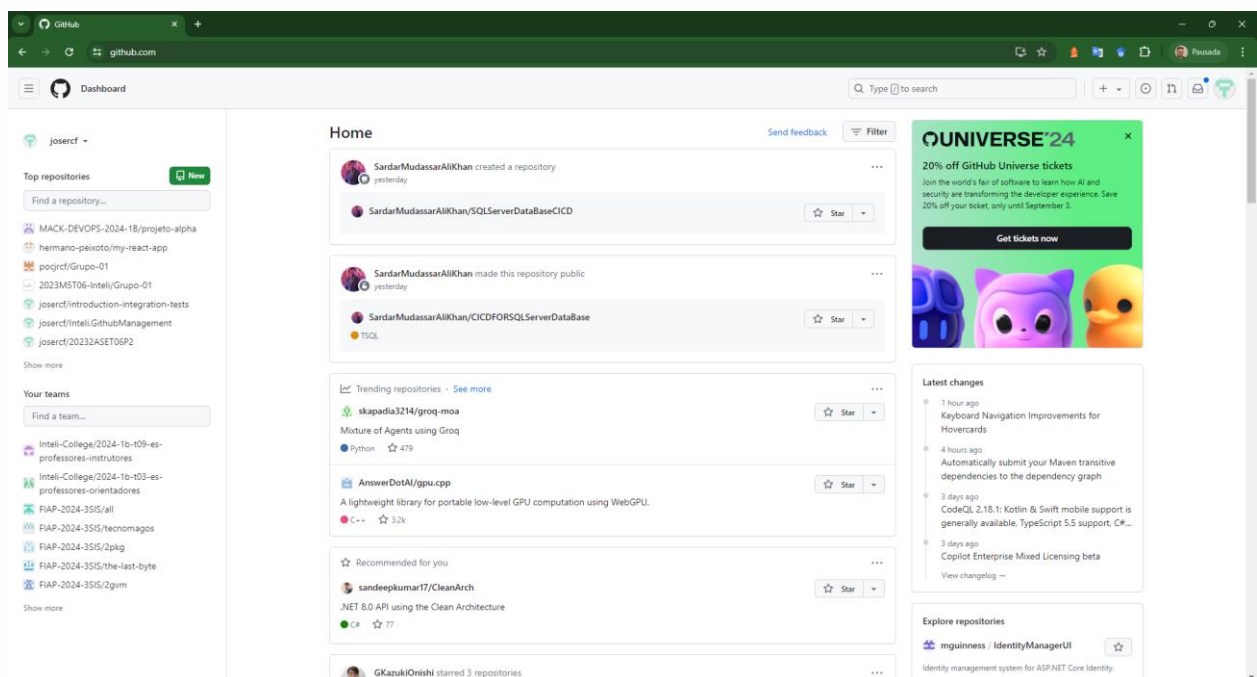
Após a etapa anterior, você precisará resolver um enigma para garantir que você é uma pessoa real.



Após resolver o desafio, um código será enviado para o e-mail que você informou na etapa anterior. Insira o código recebido no campo



Após seguir estas etapas, você deverá ter acesso a uma tela semelhante a tela abaixo. É a página principal do seu Github. No decorrer do curso você vai interagir bastante com este portal e se familiarizar com algumas funcionalidades.



A sua tela provavelmente estará vazia, pois a sua conta acabou de ser criada e você não possui conexões, repositórios ou projetos. Mas não se preocupe, ao longo do curso e da sua carreira como desenvolvedor, você terá a oportunidade de explorar essas funcionalidades e criar muitos repositórios.

### 1.4.2 Criar repositório Git

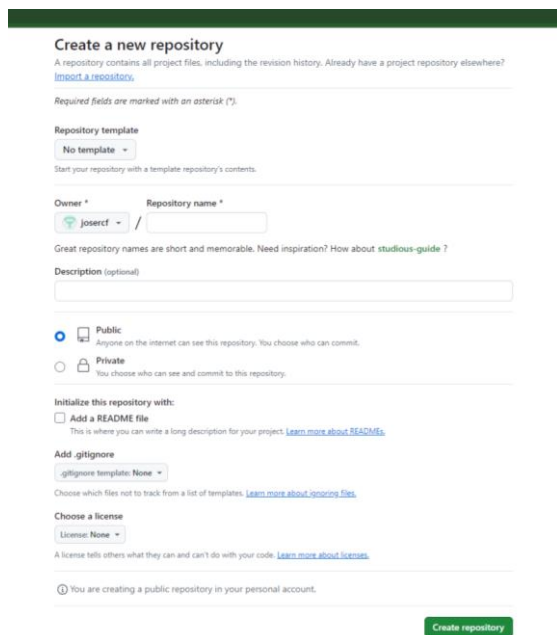
Um repositório git é um espaço onde você pode **armazenar**, **gerenciar** e **versionar** os **arquivos de um projeto**. Um repositório git contém o histórico de todas as alterações feitas nos arquivos, permitindo que você recupere versões anteriores, compare diferenças, faça ramificações e fusões, e colabore com outros desenvolvedores. Você pode criar um repositório git localmente no seu computador, ou remotamente em um serviço online como o GitHub.

No GitHub, você pode criar uma conta gratuita e ter acesso a vários recursos, como issues, pull requests, code review, integração contínua, e muito mais. O GitHub também tem uma comunidade de milhões de desenvolvedores, que contribuem para projetos open source de diversas áreas e tecnologias. Você pode usar o GitHub para aprender, ensinar, e se inspirar com outros projetos.



Para criar um repositório git usando o GitHub, você precisa seguir alguns passos:

Com a sua conta logada, acesse <https://github.com/new>. Após isso, preencha as informações solicitadas:

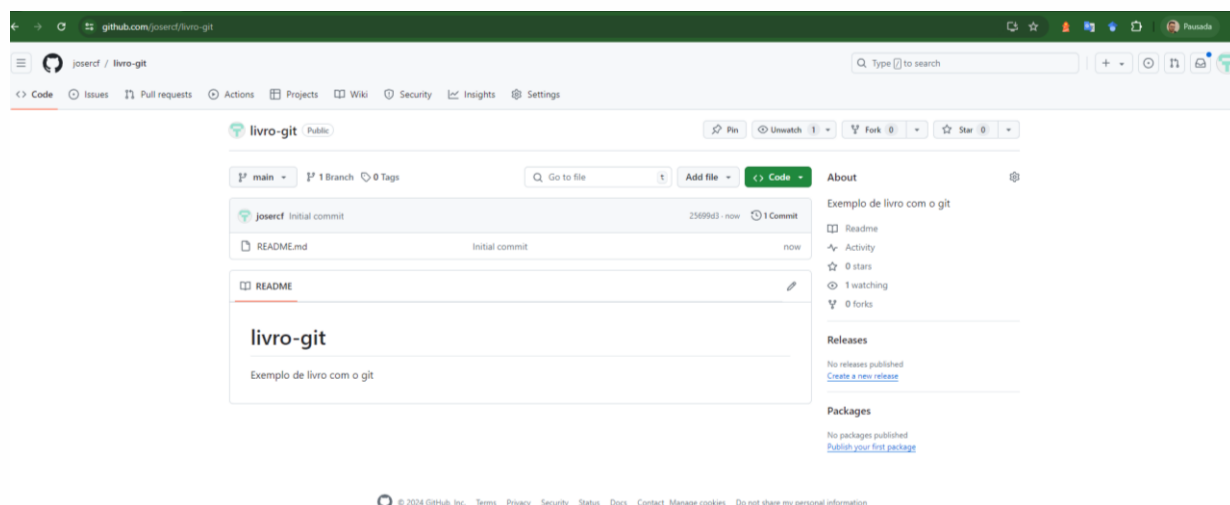


The screenshot shows the GitHub 'Create a new repository' page. It includes fields for 'Repository template' (set to 'No template'), 'Owner' (set to 'joserctf'), and 'Repository name'. There is a 'Description (optional)' text area. Under 'Visibility', 'Public' is selected. The 'Initialize this repository with' section has 'Add a README file' checked. The 'Add .gitignore' section has '.gitignore template: None' selected. The 'Choose a license' section has 'License: None' selected. A green 'Create repository' button is at the bottom right.

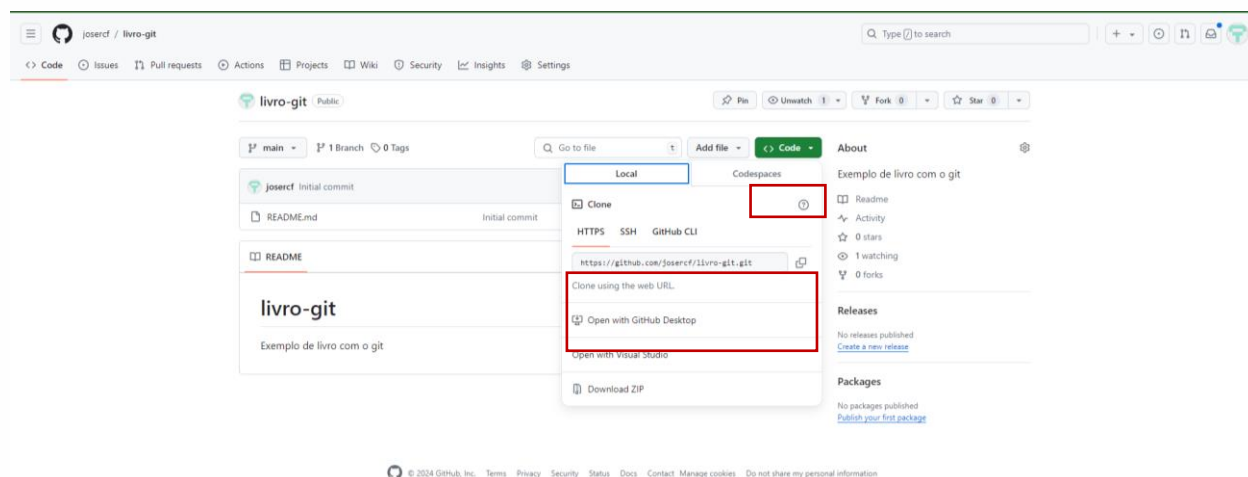
- **Owner:** Você pode participar de organizações e ter permissões de criar repositórios nelas. Você pode criar repositórios na sua própria org.
- **Repository Name:** Nome do repositório
- **Description:** Descrição breve do que será armazenado no repositório.
- **Visibilidade:** Public ou Private, define se o repositório pode ser visto e clonado por qualquer pessoa com acesso à internet ou se será privado e acessível somente mediante convite.
- **Add readme file:** Readme é um ou mais arquivos que possuem instruções e orientações sobre o projeto.
- **Add .gitignore:** Gitignore é um arquivo que nos permite informar ao git quais arquivos devem ser ignorados dentro do repositório. Isso serve, por exemplo, para evitar salvar senhas.
- **Choose a license:** Você pode definir uma licença para orientar como o seu código pode ser usado/modificado.



Depois de preencher as informações, clique no botão "Create repository" para criar o repositório. Você vai ver uma tela como esta:

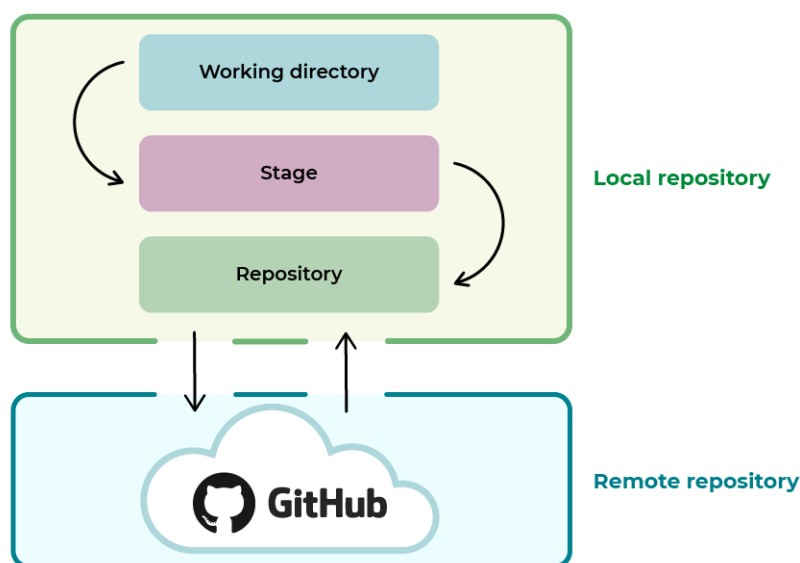


Nessa tela, você pode ver o endereço do seu repositório, que é formado pelo seu nome de usuário, seguido de uma barra (/), e o nome do repositório. Por exemplo, o endereço do nosso repositório é josercl/livro-git. Você pode copiar esse endereço clicando no botão "Code" e escolhendo a opção "HTTPS", que é o protocolo mais comum para acessar repositórios git.



### 1.4.3 Clonar repositório Git

O repositório que você acabou de criar está armazenado em um servidor remoto do GitHub, na nuvem. Isso significa que ele pode ser acessado por qualquer pessoa que tenha a URL e as permissões adequadas. No entanto, para trabalhar com o código do repositório, você precisa ter uma cópia dele no seu computador local, onde você pode editar, adicionar e remover arquivos.



Para fazer isso, você precisa clonar o repositório, ou seja, criar uma cópia exata dele na sua máquina. O comando **git clone** serve justamente para isso: ele baixa todos os arquivos e o histórico do repositório remoto para uma pasta no seu computador, que passa a se chamar de repositório local. Você só precisa fazer isso uma vez, na primeira vez que for usar o repositório.

### 1.5 Comandos básicos do Git

Nesta seção, vamos aprender os comandos básicos do Git que você precisa saber para começar a usar esse poderoso sistema. Vamos ver como **clonar** um repositório, isto é, copiar um projeto existente de um servidor remoto para o seu computador local; como **adicionar**, **remover** e modificar arquivos no seu repositório local; como fazer **commits**

das suas alterações e registrá-las no histórico do projeto; e como **sincronizar** o seu repositório local com o repositório remoto, enviando e recebendo as mudanças feitas por você ou por outros colaboradores.

### 1.5.1 git init

Você pode obter um projeto Git utilizando duas formas principais. A primeira faz uso de um projeto ou diretório existente e o importa para o Git. A segunda clona um repositório Git existente a partir de outro servidor.

### 1.5.2 git clone

Agora, você pode clonar o repositório no seu computador, usando o comando **`git clone`** seguido do endereço que você copiou. Por exemplo, no terminal, você pode digitar:

#### Inicializando um Repositório em um Diretório Existente

Caso você esteja iniciando o monitoramento de um projeto existente com Git, você precisa ir para o diretório do projeto e digitar

```
git init
```

Isso cria um novo subdiretório chamado `.git` que contem todos os arquivos necessários de seu repositório — um esqueleto de repositório Git. Neste ponto, nada em seu projeto é monitorado.

#### Primeira versão

Caso você queira começar a controlar o versionamento dos arquivos existentes (diferente de um diretório vazio), você provavelmente deve começar a monitorar esses arquivos e fazer um commit inicial. Você pode realizar isso com poucos comandos

```
touch .gitignore
```

```
git add .gitignore
```

```
git commit -m "Versão inicial do projeto"
```

Bem, nós iremos repassar esses comandos em um momento. Neste ponto, você tem um repositório Git com arquivos monitorados e um commit inicial.

### **git clone [URL]**

```
Windows PowerShell
PS E:\Projetos\Fullstack> git clone https://github.com/josercf/livro-git.git
Cloning into 'livro-git'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
PS E:\Projetos\Fullstack> |
```

Esse comando vai criar uma cópia do repositório na pasta atual, com o mesmo nome do repositório. Você pode entrar nessa pasta usando o comando ``cd`` seguido do nome do repositório. Por exemplo:

### **cd livro-git**

```
Windows PowerShell
PS E:\Projetos\Fullstack> cd .\livro-git\
PS E:\Projetos\Fullstack\livro-git> |
```

Pronto, você já tem um repositório git criado no GitHub e clonado no seu computador. A partir daqui você pode **adicionar**, **modificar** e **remover** arquivos, e **sincronizar** as suas alterações com o repositório remoto, usando os comandos do git.

Depois, você pode criar ou clonar repositórios Git no GitHub, e sincronizá-los com o seu computador usando os comandos do Git.

Você também pode usar o GitHub Desktop, um aplicativo que facilita o uso do Git e do GitHub no seu computador, sem precisar digitar comandos no terminal.

### 1.5.3 git add

Quando um repositório é inicialmente clonado, todos os seus arquivos estarão monitorados e inalterados porque você simplesmente os obteve e ainda não os editou. Conforme você edita esses arquivos, o Git passa a vê-los como modificados, porque você os alterou desde seu último commit. Você seleciona esses arquivos modificados e então faz o commit de todas as alterações selecionadas e o ciclo se repete.

#### Monitorando Novos Arquivos

Para passar a monitorar um novo arquivo, use o comando **git add**. Para monitorar o arquivo **README**, você pode rodar isso:

```
git add README
```

Se você rodar o comando **git status**, você pode ver que o seu arquivo **README** agora está sendo monitorado. Os arquivos monitorados serão os que faram parte do commit.

### 1.5.4 git status

A principal ferramenta utilizada para determinar quais arquivos estão em quais estados é o comando:

```
git status
```

O comando lhe mostra em qual branch você se encontra. Vamos dizer que você adicione um novo arquivo em seu projeto, um simples arquivo README. Caso o arquivo não exista e você execute git status, você verá o arquivo não monitorado dessa forma:

```
# On branch master
# Untracked files:
#   (use "git add {file}..." to include in what will be committed)
#
# README
nothing added to commit but untracked files present (use "git add" to track)
```

Você pode ver que o seu novo arquivo **README** não está sendo monitorado, pois está listado sob o cabeçalho "**Untracked files**" na saída do comando status. Não monitorado significa basicamente que o Git está vendo um arquivo que não existia na última captura (commit); o Git não vai incluí-lo nas suas capturas de commit até que você o diga explicitamente que assim o faça. Ele faz isso para que você não inclua acidentalmente arquivos binários gerados, ou outros arquivos que você não tem a intenção de incluir. Digamos, que você queira incluir o arquivo **README**, portanto vamos começar a monitorar este arquivo.

### 1.5.5 git diff

Se o comando git status for muito vago — você quer saber exatamente o que você alterou, não apenas quais arquivos foram alterados — você pode utilizar o comando.

```
git diff
```

Apesar do comando git status responder essas duas perguntas de maneira geral, o git diff mostra as linhas exatas que foram adicionadas e removidas — o patch, por assim dizer.

Se você quer ver o que selecionou que irá no seu próximo commit, pode utilizar:

```
git diff --cached
```

### 1.5.6 git commit

Armazena o conteúdo atual do índice em um novo commit, juntamente com uma mensagem de registro do usuário que descreve as mudanças.

Se usa o commit depois de já ter feito o git add, para fazer o commit:

```
git commit -m "Mensagem"
```

Para commitar também os arquivos versionados mesmo não estando no Stage basta adicionar o parâmetro -a

```
git commit -a -m "Mensagem"
```

Refazendo commit quando esquecer de adicionar um arquivo no Stage:

`git commit -m "Mensagem" --amend`

O amend é destrutivo e só deve ser utilizado antes do commit ter sido enviado ao servidor remoto.

### 1.5.7 git reset

Em qualquer fase, você pode querer desfazer alguma coisa. Aqui, veremos algumas ferramentas básicas para desfazer modificações que você fez. Cuidado, porque você não pode desfazer algumas dessas mudanças. Essa é uma das poucas áreas no Git onde você pode perder algum trabalho se fizer errado. Para voltar ao último commit:

**`git reset --hard HEAD~1`**

Para voltar ao último commit e mantém os últimos arquivos no Stage:

**`git reset --soft HEAD~1`**

Volta para o commit com a hash XXXXXXXXXXXX:

**`git reset --hard XXXXXXXXXXXX`**

Recuperando commit apagado pelo git reset

Para visualizar os hashes

**`git reflog`**

E para aplicar:

**`git merge {hash}`**

### 1.5.8 git rm

Para remover um arquivo do Git, você tem que removê-lo dos arquivos que estão sendo monitorados (mais precisamente, removê-lo da sua área de seleção) e então fazer o commit. O comando **`git rm`** faz isso e remove o arquivo do seu diretório para você não ver ele como arquivo não monitorado (untracked file) na próxima vez.

### **git rm -f {arquivo}**

Se você modificou o arquivo e já o adicionou na área de seleção, você deve forçar a remoção com a opção -f. Essa é uma funcionalidade de segurança para prevenir remoções acidentais de dados que ainda não foram gravados em um snapshot e não podem ser recuperados do Git.

### **1.5.9 git mv**

Diferente de muitos sistemas VCS, o Git não monitora explicitamente arquivos movidos. É um pouco confuso que o Git tenha um comando mv. Se você quiser renomear um arquivo no Git, você pode fazer isso com

```
git mv arquivo_origem arquivo_destino
```

e funciona. De fato, se você fizer algo desse tipo e consultar o status, você verá que o Git considera que o arquivo foi renomeado.

No entanto, isso é equivalente a rodar algo como:

```
mv README.txt README
git rm README.txt
git add README
```

O Git descobre que o arquivo foi renomeado implicitamente, então ele não se importa se você renomeou por este caminho ou com o comando mv. A única diferença real é que o comando mv é mais conveniente, executa três passos de uma vez. O mais importante, você pode usar qualquer ferramenta para renomear um arquivo, e usar add/rm depois, antes de consolidar com o commit.

## **1.6 Branch e Merge**

Um branch no Git é simplesmente um leve ponteiro móvel para um dos commits. O nome do branch padrão no Git é master. Como você inicialmente fez commits, você tem um branch principal (master branch) que aponta para o último commit que você fez. Cada vez que você faz um commit ele avança automaticamente.



### 1.6.1 git branch

O que acontece se você criar um branch? Bem, isso cria um ponteiro para que você possa se mover. Vamos dizer que você crie um branch chamado testing. Você faz isso com o comando git branch:

### 1.6.2 git checkout

Com o git checkout você pode mudar de branch, caso a branch ainda não exista você poderá passar o parâmetro -b para criar.

### 1.6.3 git merge

Suponha que você decidiu que o trabalho na tarefa #53 está completo e pronto para ser feito o merge no branch **master**. Para fazer isso, você fará o merge do seu branch **iss53**, bem como o merge do branch **hotfix** de antes. Tudo que você tem a fazer é executar o checkout do branch para onde deseja fazer o merge e então rodar o comando **git merge**:

### 1.6.4 git log

Depois que você tiver criado vários commits, ou se clonou um repositório com um histórico de commits existente, você provavelmente vai querer ver o que aconteceu. A ferramenta mais básica e poderosa para fazer isso é o comando:

### 1.6.4 git stash

Muitas vezes, quando você está trabalhando em uma parte do seu projeto, as coisas estão em um estado confuso e você quer mudar de branch por um tempo para trabalhar em outra coisa. O problema é, você não quer fazer o commit de um trabalho incompleto somente para voltar a ele mais tarde. A resposta para esse problema é o comando git stash.

Você quer mudar de branch, mas não quer fazer o commit do que você ainda está trabalhando; você irá fazer o stash das modificações. Para fazer um novo stash na sua pilha, execute:

**git stash**

Seu diretório de trabalho estará limpo. Neste momento, você pode facilmente mudar de branch e trabalhar em outra coisa; suas alterações estão armazenadas na sua pilha. Para ver as stashes que você guardou, você pode usar

### **git stash list**

Você pode aplicar aquele que acabou de fazer o stash com o comando mostrado na saída de ajuda do comando stash original: **git stash apply**. Se você quer aplicar um dos stashes mais antigos, você pode especificá-lo, assim: **git stash apply stash@{2}**. Se você não especificar um stash, Git assume que é o stash mais recente e tenta aplicá-lo.

### **1.6.5 git tag**

Git tem a habilidade de criar tags em pontos específicos na história do código como pontos importantes. Geralmente as pessoas usam esta funcionalidade para marcar pontos de release (v1.0, e por aí vai). Nesta seção, você aprenderá como listar as tags disponíveis, como criar tags, e quais são os tipos diferentes de tags. Para listar as **tags** execute:

#### **git tag**

Para criar uma **tag** basta executar o seguinte comando, caso não queira criar a **tag** anotada, somente retire os parâmetros **-a** e **-m**.

```
git tag -a v1.4 -m 'my version 1.4'
```

## **1.7 Compartilhar e Atualizar Projetos**

### **1.7.1 git fetch**

Para pegar dados dos seus projetos remotos, você pode executar:

```
git fetch origin
```

Esse comando vai até o projeto remoto e pega todos os dados que você ainda não tem. Depois de fazer isso, você deve ter referências para todos os branches desse remoto, onde você pode fazer o merge ou inspecionar a qualquer momento.

### 1.7.1 git pull

Incorpora as alterações de um repositório remoto no branch atual. Em seu modo padrão, **git pull** é uma abreviação para **git fetch** seguido de **git merge FETCH\_HEAD**. Por exemplo, se eu estiver em uma branch chamada **develop** e quiser atualizar caso haja atualizações remotamente:

```
git pull origin develop
```

### 1.7.2 git push

O **git push** é o comando em que você transfere commits a partir do seu repositório local para um repositório remoto. É a contrapartida do **git fetch**, que busca importações e comprometem as agências locais, utilizando o **git push** as exportações comprometem as filiais remotas.

Para fazer isso, você executa **git push [nome\_do\_repositório\_remoto] [nome\_da\_sua\_branch\_local]**, que vai tentar fazer que o **[nome\_do\_repositório\_remoto]** receba a sua Branch **[nome\_da\_sua\_branch\_local]** contendo todos seus commits com alterações. Por exemplo:

```
git push origin develop
```

# Frontend

## Introdução ao HTML

HTML (Linguagem de Marcação de HiperTexto) é o bloco de construção mais básico da web. Define o significado e a estrutura do conteúdo da web. Outras tecnologias além do HTML geralmente são usadas para descrever a aparência/apresentação (CSS) ou a funcionalidade/comportamento (JavaScript) de uma página da web.

"Hipertexto" refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. Links são um aspecto fundamental da web. Ao carregar conteúdo na Internet e vinculá-lo a páginas criadas por outras pessoas, você se torna um participante ativo no world wide web.

### Pré-requisitos

Antes de iniciar este módulo, você não precisa de nenhum conhecimento prévio sobre HTML, mas deve ter pelo menos uma familiaridade básica em utilizar computadores e utilizar a web passivamente (por exemplo, apenas navegando e consumindo conteúdo).

### 2.1 O que é HTML e suas tags?

O HTML existe desde 1991 e atualmente está na versão 5, que veio recheada de recursos e funcionalidades que trazem melhorias para o desenvolvimento web.

Sua responsabilidade principal é demarcar a **estrutura** de uma página da web.

Essa estrutura do HTML é formada por um conjunto de elementos, ou seja, os hipertextos, que se conectam entre si formando a página.

Os elementos HTML ou também chamados de tags HTML, são utilizados para informar ao navegador que tipo de estrutura é essa que está sendo construída, podendo ser títulos, parágrafos, imagens, links, entre outros.

Dessa forma, para que um documento seja interpretado pelo navegador, é necessário que o arquivo tenha a extensão .html e, a partir disso, poderá ser exibido por qualquer navegador web.

As tags são formadas por uma estrutura própria, iniciam com o sinal “menor que”, em seguida vem o nome daquele elemento e por fim, o sinal “maior que”.

Podem ser dispostas em tags que precisam de fechamento e tags que fecham sozinhas (self-closing).

O fechamento de uma tag será definido com uma barra (/), sendo que no caso das tags de autofechamento, não há necessidade da presença desse caractere.



```
<!--Tag que precisa de fechamento-->  
<h1>Olá Mundo!</h1>  
<!--Tag de auto fechamento-->  
, assim:

<p>Eu sou um parágrafo, oh sim, eu sou. </p>

Cada título deve ser envolvido em um elemento de título:

<h1>Eu sou o título da história. </h1>

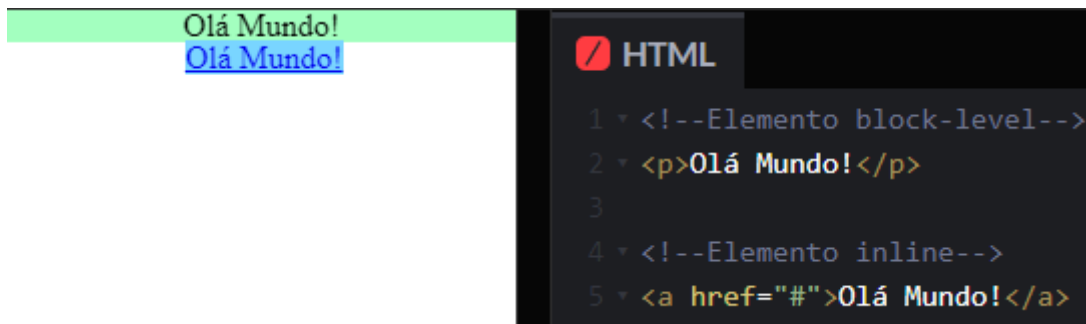
Existem seis elementos de título - <h1>, <h2>, <h3>, <h4>, <h5> e <h6>. Cada elemento representa um nível diferente de conteúdo no documento; <h1> representa o título principal, <h2> representa subtítulos, <h3> representa subsubtítulos, e assim por diante.

## 2.3 Tipos de tag

As tags podem ser categorizadas inicialmente em dois tipos: em “nível de bloco” (block-level) e “em linha” (inline).

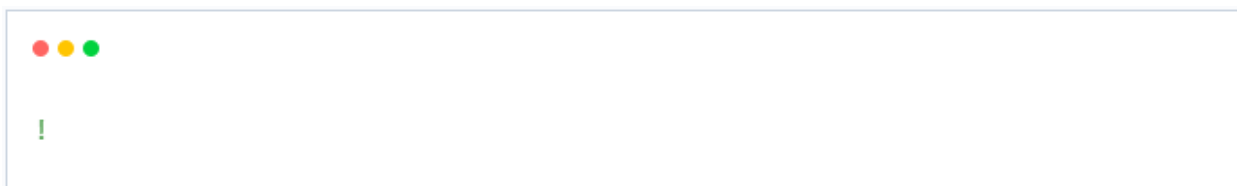
Um elemento ao nível de bloco ocupa toda a largura de seu elemento pai, que também chamamos de elemento container, criando assim um “bloco”.

Já os elementos inline, geralmente usamos para demarcação de conteúdo de texto.



## 2.4 Estrutura básica

O VS Code utiliza por padrão o [Emmet Abbreviations](#), que traz o aparecimento automático de linhas de código que fazem parte da estrutura básica do HTML ao digitar o ponto de exclamação. Dessa forma:



Resultado:





```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

#### 2.4.1 <!DOCTYPE html>

Informa ao navegador que esse documento é do tipo HTML e indica sua versão. Quando está escrito apenas html, indica que é a mais recente.

#### 2.4.2 <html>

Representa a raiz do documento, serve com um container que engloba todos os outros elementos HTML.

#### 2.4.3 <body>

É onde fica todo o conteúdo de texto, imagem e vídeos, em que o usuário ou usuária vê e interage, nele os conteúdos são estruturados pelas demais tags do HTML.

#### 2.4.4 <script>

Esse elemento contém instruções de script ou aponta para um arquivo de script externo por meio do atributo src.

#### 2.4.5 <head>

Compreende as informações do documento que serão interpretadas pelo navegador (metadados). Como, por exemplo, título do documento, links para folhas de estilo, etc.

#### 2.4.6 <meta>

Define metadados, ou seja, informações sobre dados de um documento HTML. As tags vão dentro do elemento e são usadas para especificar o conjunto de caracteres, o autor ou autora do documento, as configurações da janela de visualização etc.

#### 2.4.7 <link>

É uma tag vazia, que contém apenas atributos e faz a relação do documento HTML com recursos externos.

É comumente usado para vincular uma folha de estilo externa, também é usada para definir o favicon da página (ícone da aba do navegador), como outros recursos.

#### 2.4.8 <style>

Essa tag é usada para declarar estilos (CSS) para um documento.

### 2.5 HTML: Tags semânticas

Tags semânticas são tags que possuem um significado, que dão sentido à informação de texto ao navegador e buscadores. Como, por exemplo: utilizar a tag **<header>** para cabeçalhos ou **<article>** para dar um significado de artigo para aquele bloco de texto, até mesmo **<p>** para indicar que aquele texto é um parágrafo.

É uma boa prática tentar sempre utilizar essas tags semânticas para ajudar no entendimento do código, além de ajudar muito no SEO do site (Otimização para motores de busca, é o que ajuda o seu site a se rankear melhor nos buscadores como o Google).

#### Alguns dos benefícios de se escrever a marcação semântica:

- Os mecanismos de pesquisa considerarão seu conteúdo como palavras-chave importantes para influenciar os rankings de pesquisa da página.

- Os leitores de tela podem usá-lo como uma placa de sinalização para ajudar usuários com deficiência visual a navegar em uma página
- Encontrar blocos de código importantes é significativamente mais fácil do que procurar divs sem fim, com ou sem classes de semântica ou de nome espaçado.
- Sugira ao desenvolvedor o tipo de dados que serão preenchidos
- A nomeação semântica reflete a nomeação adequada do elemento/componente personalizado

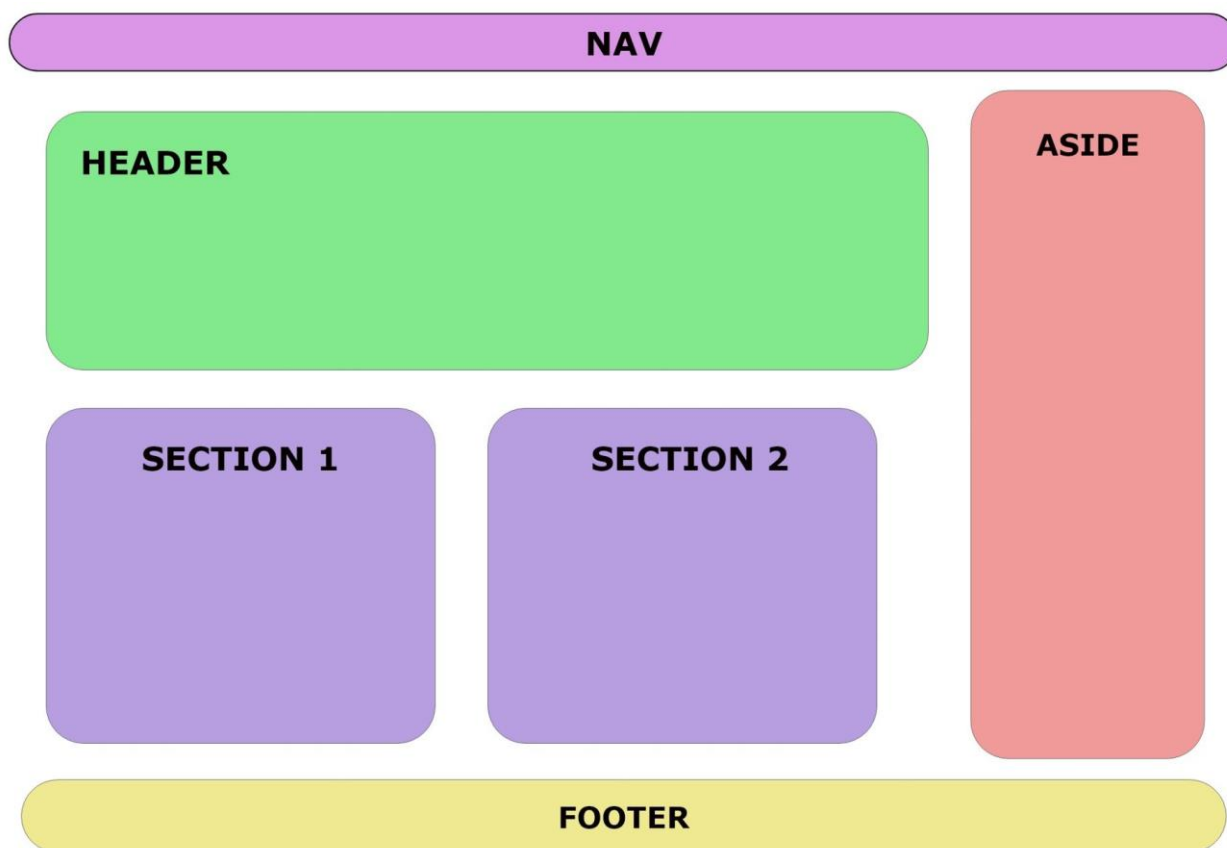
Ao abordar qual marcação usar, pergunte a si mesmo: "Quais elementos melhor descrevem/representam os dados que vou preencher?" Por exemplo, é uma lista de dados? ordenado, não ordenado? é um artigo com seções e uma parte de informações relacionadas? lista as definições? é uma figura ou imagem que precisa de legenda? deve ter um cabeçalho e rodapé, além do cabeçalho e rodapé em todo o site? etc.

## **Elementos semânticos**

Estes são alguns dos aproximadamente 100 elementos semânticos disponíveis:

- |             |           |
|-------------|-----------|
| • <header>  | • <aside> |
| • <main>    | • <nav>   |
| • <footer>  | • <ol>    |
| • <section> | • <ul>    |
| • <article> | • <li>    |

Exemplo de estruturação com tags semânticas:



## 2.5 HTML: Tags sem semântica

As tags que não possuem semântica não definem um significado para aquele texto, normalmente são utilizadas apenas para fins de separação e estilização. Veja logo abaixo a lista de algumas tags sem semântica:

- `<div>`
- `<span>`
- `<b>`
- `<i>`

## 2.6 HTML: Comentários

Comentários no HTML ou em qualquer outra linguagem são notas que podem ser incluídas no código-fonte para descrever o que quiser.

Assim, não modificam o programa executado e servem somente para ajudar a pessoa que está desenvolvendo a organizar melhor os seus códigos.

Para comentar algum código no HTML você deve envolvê-lo entre `<!-- Seu código aqui -->`.

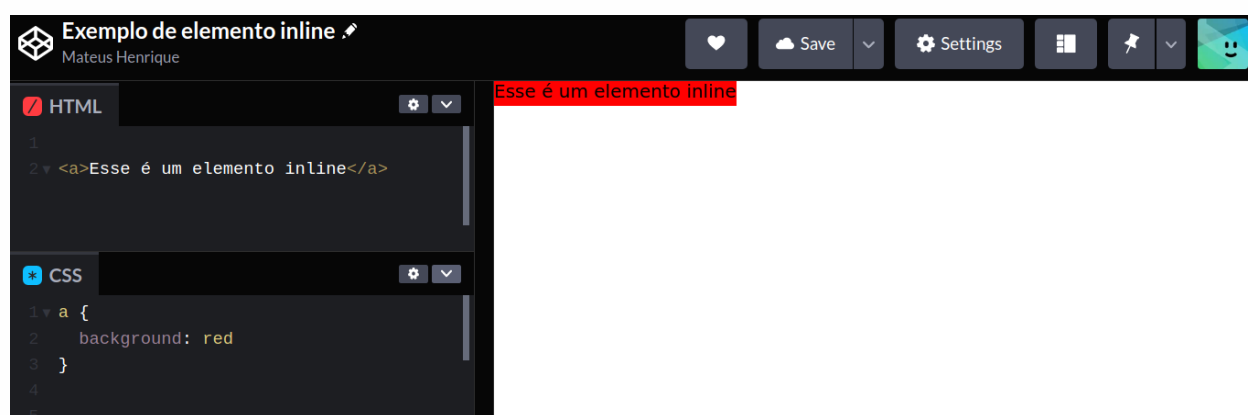
Por exemplo:

```
<!--  
  
<div>  
  <p>Esse código será ignorado pelo navegador</p>  
</div>  
  
-->
```

## 2.7 Elementos inline

"Inline" é uma categorização dos elementos do HTML, os elementos inline podem ser exibidos em nível de bloco ou outros elementos inline. Eles ocupam somente a largura de seu conteúdo.

`<a>Esse é um elemento inline</a>`



Como você pode ver logo acima, a tag `a` é um elemento inline, ou seja, ele está ocupando somente o tamanho total do conteúdo, que nesse caso é o texto.

Logo abaixo você pode conferir alguns desses elementos inline:

### 2.7.1 <span>

Muito parecido com a tag sem semântica <div>, porém é um elemento em linha que atua como um container genérico para agrupar conteúdo de texto.

```
<span>Olá, mundo!</span>
```

### 2.7.2 <br>

Essa tag produz uma quebra de linha em um determinado ponto do documento.

Exemplo de utilização:

```
<span>Olá <br> meu nome <br> é <br> Joãozinho</span>
```

### 2.7.3 <a>

É um elemento âncora que define um hiperlink, que vincula páginas web, arquivos, endereços de emails, ligações na mesma página. Essa tag com o atributo href, indica o destino do link. Também é possível criar âncoras para textos na mesma página com o href informando o id do elemento destino.

Exemplo de utilização:

```
<span>Clique <a href="https://www.google.com" target="blank">aqui</a> para ser redirecionado para a página do google.</span>
```

### 2.7.4 <img loading="lazy">

Utilizada para colocar imagens no site, deve-se utilizar o atributo src, colocando entre as aspas o link ou caminho do arquivo.

Exemplo de utilização:

```

```

### 2.7.5 <audio>

Utilizada para inserir áudios no site, tendo como principais atributos: **src**, recebendo como valor o link ou diretório do áudio, **controls** caso queira que seja possível controlar o áudio, **autoplay** para definir que o áudio de tocar automaticamente quando entrar no site, **type** recebendo como valor o tipo do áudio.

Exemplo de utilização:

```
<audio src="" controls></audio>
```

## 2.8 Elementos block level

Os elementos em “nível de bloco” ocupam todo o espaço do seu elemento pai (container).

### 2.8.1 <header>

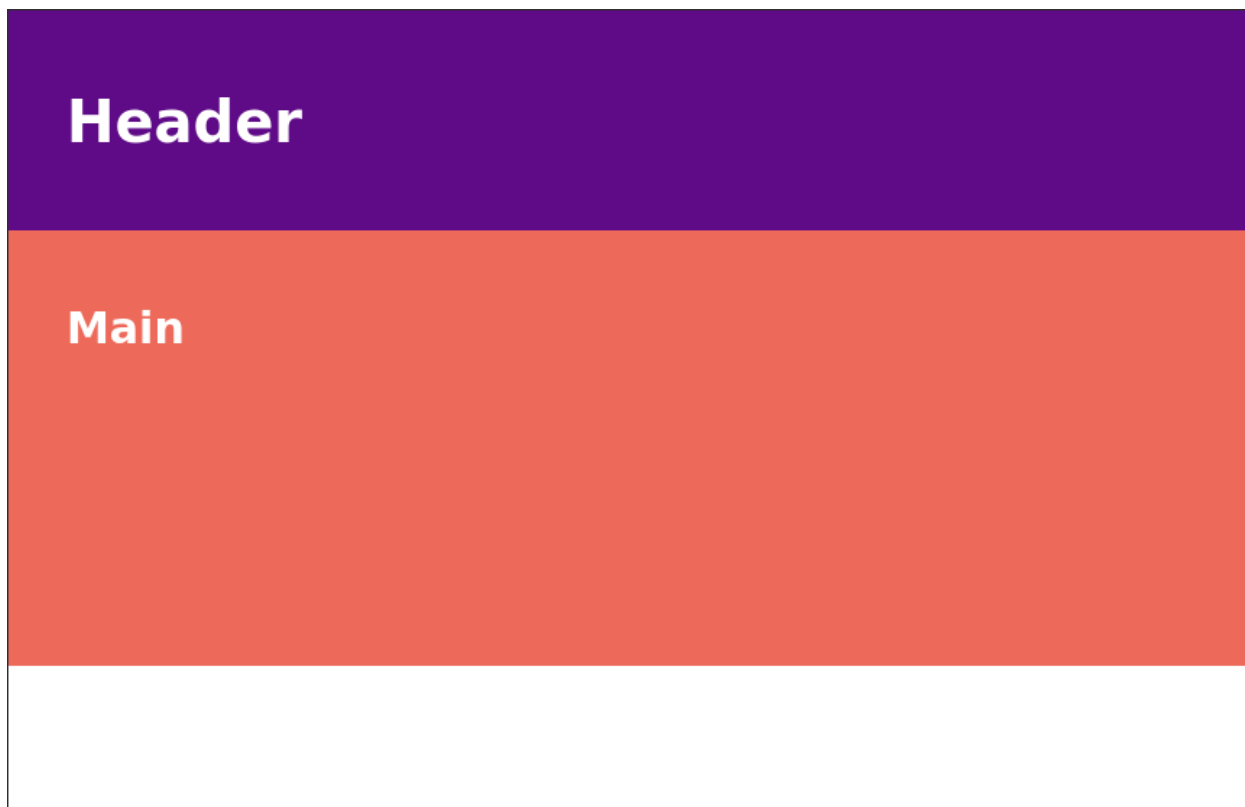
Define o cabeçalho da página, geralmente no cabeçalho identificamos o site, com a logo.



**Header**

### 2.8.2 <main>

Compreende o conteúdo principal do corpo da página.



### 2.8.3 <footer>

Define o final da página ou conteúdo, o rodapé, geralmente colocamos contatos, redes sociais, endereço, informações “institucionais” no geral.





**Header**

**Main**

**Footer**

#### **2.8.4 <section>**

Dentro do conteúdo principal, essa tag compreende uma seção da página.

#### **2.8.5 <article>**

Inclui um artigo da página, muito utilizado em blogs e páginas de criação de conteúdo, também indica o principal conteúdo de texto da página.

#### **2.8.6 <aside>**

Representa uma seção que faz referência a outro conteúdo da página, como uma definição, uma explicação extra, avisos, biografia do autor, ou seja um conteúdo complementar.

#### **2.8.9 <nav>**

Contempla o menu de navegação das páginas do site, e dentro inserimos a listas e links com a tag `<a href=""></a>`.

### 2.8.10 <div>

Assim como as outras tags, também funciona como um container, porém a grande diferença é que a div não tem valor semântico, é apenas uma divisão na página para fins de layout.

### 2.8.11 <p>

Representa um parágrafo.

### 2.8.12 <h1>, <h2><h6>

A família de cabeçalhos ou headings, define os **títulos** da página. Esse grupo de elementos possuem um alto valor semântico e uma organização hierárquica, indo de <h1> até <h6>, sendo o h1 de maior valor semântico e o h6 de menor valor semântico.

Olá Mundo!

Olá Mundo!

Olá Mundo!

Olá Mundo!

Olá Mundo!

Olá Mundo!

#### HTML

```
1 <h1>Olá Mundo!</h1>
2 <h2>Olá Mundo!</h2>
3 <h3>Olá Mundo!</h3>
4 <h4>Olá Mundo!</h4>
5 <h5>Olá Mundo!</h5>
6 <h6>Olá Mundo!</h6>
```

### 2.8.13 <hr>

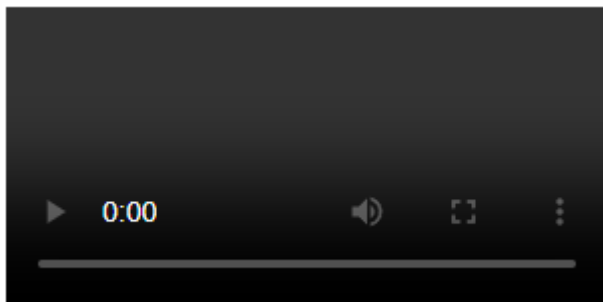
Essa tag constrói uma linha horizontal entre elementos, representa semanticamente uma quebra de conteúdo.

### 2.8.14 <video>

Utilizada para inserir vídeos no site, a tag possui atributos como width recebendo como valor a largura do vídeo em pixels, height recebendo como valor a altura do vídeo em

píxeis, caso não for informada esses atributos, será utilizado a largura e altura padrão do vídeo, controls (quando presente nos permite controlar o video), src recebendo como valor o link ou diretório do arquivo de vídeo.

```
<video src="" controls></video>
```



## 2.9 Elementos de um formulário

Formulários HTML são um dos principais pontos de interação entre um usuário e um web site ou aplicativo. Eles permitem que os usuários enviem dados para o web site. Na maior parte do tempo, os dados são enviados para o servidor da web, mas a página da web também pode interceptar para usá-los por conta própria.

### 2.9.1 <form>

A tag <form> indica que estamos iniciando um formulário, recebe como principais atributos **method** que recebe como valor o método http que esse formulário irá executar (get, post) e **action** que especifica para onde enviar os dados do formulário quando um formulário é enviado.

A tag <form> em HTML é usada para criar um formulário em uma página da web. Ela define uma seção do documento HTML que contém elementos interativos, como campos de entrada de texto, caixas de seleção, botões, etc., que permitem aos usuários inserirem dados ou interagir com o site.

Ela possui vários atributos, usados para especificar como os dados do formulário devem ser processados e enviados para o servidor web:

- **action:** especifica para onde os dados do formulário devem ser enviados quando o formulário é submetido.
- **method:** especifica o método [HTTP](#) a ser usado ao enviar os dados do formulário. Os métodos mais comuns são “GET” e “POST”.

`<form></form>`

- **enctype:** especifica como os dados do formulário devem ser codificados antes de serem enviados para o servidor.

### 2.9.1 <input>

É um campo para que o usuário ou usuária possa inserir algum texto, data, número, cor etc. possui como principais atributos **type**, que recebe como valor o tipo do input.

A tag input em um formulário HTML é um elemento usado para criar campos de entrada interativos, nos quais as pessoas usuárias podem inserir dados. Esta tag é essencial para a construção de formulários web. Ela pode ser usada para criar vários tipos de campos de entrada, dependendo do valor do atributo type especificado.

Alguns dos tipos mais comuns de campos de entrada input incluem:

- **text:** cria um campo de texto simples, onde é possível inserir texto livremente.
- **password:** cria um campo de senha, onde os caracteres digitados são ocultos.
- **checkbox:** cria uma caixa de seleção que permite selecionar uma ou mais opções.
- **radio:** cria botões de rádio, onde é possível selecionar apenas uma opção de um grupo de opções.
- **submit:** cria um botão de envio que permite que as pessoas usuárias enviem o formulário.
- **file:** cria um campo de seleção de arquivo, que permite enviar arquivos para o servidor.
- **email, number, date, entre outros:** tipos específicos de campos de entrada que fornecem validação e funcionalidades específicas para tipos de dados comuns.

### 2.9.2 <textarea>

Representa uma caixa de texto, útil quando você quer permitir à pessoa usuária informar um texto extenso em formato livre, como um comentário ou formulário de retorno.

```
<textarea></textarea>
```

### 2.9.3 <button>

Um botão clicável, que possui como principais atributos type, que caso receba submit como valor e esteja dentro de um formulário, irá submeter o formulário.

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

### 2.9.4 <label>

Veremos agora o que é label HTML, uma tag muito importante para os campos de formulários.

Ela especifica qual o “rótulo” do input (a que se refere o input, como, por exemplo, envolvê-la em um texto “Nome completo”), e ajuda na experiência do usuário e usuária durante a utilização e preenchimento do formulário.

```
<label>Nome completo</label>
```

Veja um exemplo de formulário criado utilizando as tags que aprendemos:

Nome completo

Digite seu elogio

Enviar

## 2.10 O que são atributos HTML?

Atributos HTML (em inglês html attributes") são palavras especiais usadas dentro da tag de abertura para controlar o comportamento do elemento.

Os atributos HTML são um modificador de um tipo de elemento HTML. Com eles, podemos identificar melhor um elemento, informar qual arquivo aquela tag deve utilizar, indicar o tipo de um campo de texto etc.

Há dois tipos de atributos no HTML, os globais, aceitos por todas as tags, como, por exemplo: class, id, lang, style e algumas outras que você pode conferir na documentação; e existem os específicos, que somente algumas tags possuem, como: src, disabled, href, label, etc.

A estrutura de um atributo é:

nome="valor"

```
<p class="intro">Esse é um parágrafo com atributo :D</p>
```

Onde **class** é o nome do atributo (como class, src, styled, id) e **intro** é o valor daquele atributo.

### 2.10.1 class

**class="NomeDaClasse"**

Classes são como classificações de uma tag/elemento, para estilizar uma tag específica, ou um conjunto de tags no CSS.

Também é possível usar no JavaScript para selecionar uma tag específica. Veja alguns exemplos.

**HTML:**

```
<h1 class="titulo">Mergulhe em Tecnologia!</p>
```

**CSS:** neste momento, você não precisa se preocupar tanto com o css. Você irá aprender no próximo capítulo.

```
.titulo {  
  font-size: 21px;  
  color: #fff;  
  background: blue;  
}
```

No código acima, selecionamos o css com o seletor de classes usando ponto antes do nome da classe (.classe), e então aplicamos um tamanho de fonte, cor e cor de fundo.

---

**Mergulhe em Tecnologia!**

### 2.10.2 id

**id="NomeDoido"**

É utilizado para identificar de forma única um elemento naquela página HTML, como o destino de âncoras, labels e outras funcionalidades neste sentido.

### 2.10.3 src

**src="Link ou diretório da mídia"**

Comumente utilizado para indicar para a tag qual arquivo ou mídia utilizar. Recebe valores como links (<https://google.com/minhaimagem.jpeg>) ou o nome de um arquivo já presente no projeto (/minhaimagem.jpeg).

```

```

### 2.10.4 alt

**alt="Texto alternativo"**

O atributo alt fornece informações alternativas para uma imagem se um usuário ou usuária, por algum motivo, não puder visualizá-la (devido à conexão lenta, um erro no atributo src ou a utilização de um leitor de tela).

```

```

### 2.10.5 href

#### href="Url"

Para a tag <a>, o atributo href especifica a URL da página para a qual o link vai.

```
<p>Clique <a href="https://www.alura.com.br/">aqui</a> e acesse o google</p>
```

### 2.10.6 lang

#### lang="Linguagem"

O atributo lang especifica o idioma do conteúdo da tag.

Os exemplos comuns são “en” para inglês, “es” para espanhol, “fr” para francês e assim por diante.

### 2.10.7 target

Esse atributo abre o link do documento em uma nova janela ou aba.



## Introdução ao CSS

Cascading Stylesheets — ou CSS — é a primeira tecnologia que você deve aprender após o HTML. Enquanto o HTML é utilizado para definir a estrutura e semântica do seu conteúdo, o CSS é usado para estilizá-lo e desenhá-lo. Por exemplo, você pode usar o CSS para alterar a fonte, cor, tamanho e espaçamento do seu conteúdo, dividi-lo em múltiplas colunas, ou adicionar animações e outros recursos decorativos.

No capítulo anterior vimos o que é HTML, e como ele é usado para fazer marcação de documentos. Estes documentos serão legíveis em um navegador web. Títulos serão mais largos do que textos comuns, parágrafos quebram em uma nova linha e tendo espaços entre eles. Links são coloridos e sublinhados para distingui-los do resto do texto. O que você está vendo é o estilo padrão do navegador - vários estilos básicos que o navegador aplica ao HTML, para garantir que ele será legível mesmo se não for explicitamente estilizado pelo autor da página web.

## Browser defaults

The browser will style HTML documents using an internal stylesheet. This ensures that headings are larger than normal text, links are highlighted and structures such as lists and tables are understandable.

Paragraphs are spaced out. List items get a bullet or number, [Links are highlighted and underlined](#).

- Item One
- Item Two

## A level 2 heading

You can change all of this with CSS.

No entanto, a web seria um lugar chato se todos os web sites tivessem estilos iguais ao mostrado na imagem acima. Usando CSS você pode controlar exatamente a aparência dos elementos HTML no navegador, apresentando a sua marcação com o design que desejar.

### 3.1 Para que serve o CSS?

Como falamos antes, CSS é uma linguagem para especificar como documentos são apresentados aos usuários — como eles são estilizados, dispostos etc.

Um documento é normalmente um arquivo texto estruturado usando uma linguagem de marcação — HTML é a linguagem de marcação mais comum, mas você também pode encontrar outras, como SVG ou XML.

Apresentar um documento para um usuário significa convertê-lo para um formato utilizável pelo seu público. Browsers, como Firefox, Chrome, ou Edge , são projetados para apresentar documentos visualmente, por exemplo, em uma tela de computador, projetor ou impressora.

Um navegador web é às vezes chamado de user agent, o que, basicamente, significa um programa de computador que representa uma pessoa por trás do sistema. Navegadores web são o principal tipo de agente do usuário que nos referimos quando falamos sobre CSS, contudo, ele não é o único. Há outros agentes de usuário disponíveis — tais como aqueles que convertem documentos HTML e CSS para PDF a serem impressos.

O CSS pode ser usado para estilizar um documento muito básico de texto — por exemplo, alterando a cor e tamanho dos títulos e links. Pode ser usado para criar layout — por exemplo, transformando uma simples coluna de texto em um layout com uma área de conteúdo principal e um sidebar (barra lateral) para as informações relacionadas. Pode até ser usado para efeitos tais como animação. Dê uma olhada nos links deste parágrafo, para ver exemplos específicos.

### 3.2 Sintaxe CSS

CSS é uma linguagem baseada em regras. — Você define regras especificando grupos de estilo que devem ser aplicados para elementos particulares ou grupos de elementos

na sua página web. Por exemplo, "Quero que o título principal, na minha página, seja mostrado como um texto grande e de cor vermelha."

O código seguinte mostra uma regra CSS muito simples, que chegaria perto do estilo descrito acima:



```
h1 {  
  color: red;  
  font-size: 5em;  
}
```

A regra é aberta com um [seletor](#). Isso *seleciona* o elemento HTML que vamos estilizar. Neste caso, estamos estilizando títulos de nível um ([<h1>](#)).

Temos, então, um conjunto de chaves { }. Dentro deles, haverá uma ou mais **declarações**, que tomam a forma de pares **propriedade** e **valor**. Cada par especifica uma propriedade do(s) elemento(s) que estamos selecionando e, em seguida, então um valor que gostaríamos de atribuir à propriedade

Antes dos dois pontos, temos a propriedade, e, depois, o valor. CSS [properties](#) possui diferentes valores permitidos, dependendo de qual propriedade está sendo especificado. Em nosso exemplo, temos a propriedade color, que pode tomar vários [valores para cor](#). Também temos a propriedade font-size. Essa propriedade pode ter vários [unidades de tamanho](#) como um valor.

Uma folha de estilo CSS conterá muitas regras tais como essa, escrita uma após a outra.



Você constatará que rapidamente aprende alguns valores, enquanto outros precisará pesquisar.

### 3.2.1 Módulos CSS

Como existem tantas coisas que você pode estilizar com CSS, a linguagem é dividida em módulos. Verá referência a esses módulos à medida que explora o MDN e muita das páginas da documentação são organizadas em torno de um módulo em particular.

Nesse ponto você não precisa se preocupar muito sobre como o CSS é estruturado. No entanto, isso pode tornar fácil achar informação se, por exemplo, você estiver ciente de que uma determinada propriedade provavelmente será encontrada entre outras coisas semelhantes e estiver, portanto, provavelmente na mesma especificação.

Para um exemplo específico, vamos voltar ao módulo Backgrounds e Borders — você pode achar que isso tem um senso lógico para as propriedades background-color e border-color serem definidas neste módulo. E, você está certo!

### 3.2.2 Especificações CSS

Todas as tecnologias de padrões web (HTML, CSS, JavaScript, etc.) são definidos em documentos gigantes chamados especificações (ou simplesmente "specs"), que são publicados por organizações de padrões (tais como W3C, WHATWG, ECMA, ou Khronos) e definem precisamente como essas tecnologias devem se comportar.

Com CSS não é diferente — ele é desenvolvido por um grupo dentro do W3C chamado CSS Working Group. Esse grupo é formado por representantes de fornecedores de

navegadores web e outras companhias que tem interesse em CSS. Também existe outras pessoas, conhecidas como peritos convidados (invited experts), que agem como vozes independentes; eles não são associados como um membro de alguma organização.

Novas características CSS são desenvolvidas, ou especificadas, pelo CSS Working Group. Às vezes, porque um navegador em particular está interessado em alguma capacidade, outras vezes, porque designers web e desenvolvedores estão perguntando por uma característica, e, algumas vezes, porque o Working Group em si tem identificado uma necessidade. O CSS está em constante desenvolvimento, com novas peculiaridades ficando disponíveis. Contudo, uma ideia chave sobre CSS é que todos trabalham pesado para nunca alterar as coisas de uma maneira que não quebrem os sites antigos. Um site construído no ano 2000, usando um CSS limitado da época, deverá ainda ser utilizável em um navegador moderno!

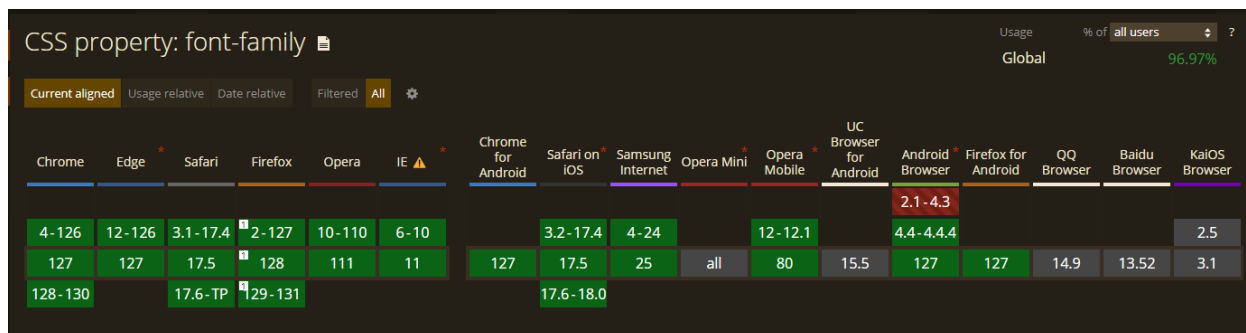
Como iniciante no CSS, é provável que você ache as especificações CSS impressionantes — eles são direcionados a engenheiros para implementar suporte aos recursos nos agentes de usuário (navegadores), não para desenvolvedores lerem com o intuito de entender CSS. Muitos desenvolvedores experientes preferem consultar a documentação do MDN ou outros tutoriais. No entanto, vale a pena saber que eles existem, entender a relação entre o CSS que você usa, suporte ao navegador (veja abaixo), e os specs (especificações).

### **3.2.3 Especificações CSS**

Uma vez que o CSS tenha sido especificado, então se torna útil para nós, em termos de desenvolvimento de páginas web, apenas se um ou mais navegadores implementá-los. Isso significa que o código foi escrito para transformar as instruções do nosso arquivo CSS em algo que possa ser mostrado na tela. É inusitado implementarem uma característica ao mesmo tempo, e, geralmente, existe uma lacuna na qual se pode usar parte do CSS em alguns navegadores e em outros não. Por esse motivo, ser capaz de verificar o estado da implementação é útil. Para cada página de propriedade no MDN,

pode-se ver o estado dela, que se está interessado. Assim, você saberá se pode usá-la em uma página.

A seguir, é apresentado o gráfico de dados compat para propriedade CSS font-family.



### 3.3 Como o CSS é estruturado

Agora que você tem uma ideia sobre o que é o CSS e seu uso básico, é hora de olhar um pouco mais a fundo das estruturas da linguagem em si. Nós já conhecemos muitos conceitos discutidos aqui, entretanto, você pode voltar para qualquer um em específico, se achar algum dos próximos conceitos um tanto confusos

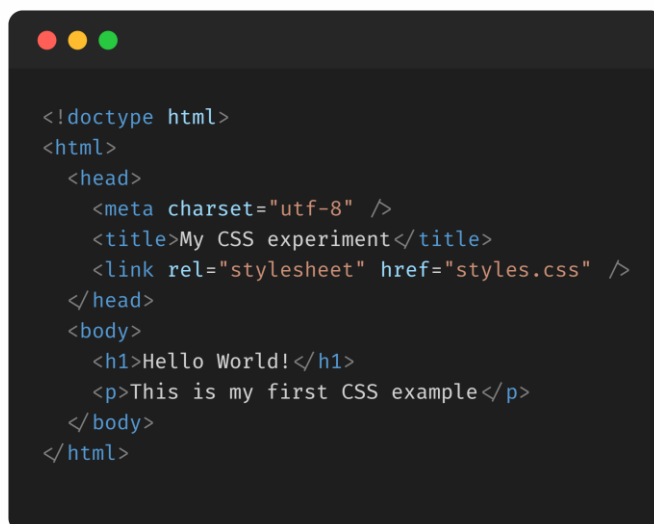
Aplicando CSS no seu HTML

A primeira coisa que você vai olhar é, os três métodos de aplicação do CSS em um documento.

#### 3.3.1 Folha de Estilos Externa

Em Começando com o CSS nós linkamos uma folha de estilos externas em nossa página. Isso é o método mais comum utilizado para juntar CSS em um documento, podendo utilizar tal método em múltiplas páginas, permitindo você estilizar todas as páginas como elas folha de estilos. Na maioria dos casos, as diferentes páginas do site vão parecer bem iguais entre si e por isso você pode usar as mesmas regras para o estilo padrão da página.

Uma folha de estilos externa é quando você tem seu CSS escrito em um arquivo separado com uma extensão **.css**, e você o refere dentro de um elemento `<link>` do HTML:



```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My CSS experiment</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

O arquivo CSS deve se parecer com algo nesse estilo:



```
h1 {
  color: blue;
  background-color: yellow;
  border: 1px solid black;
}

p {
  color: red;
}
```

O atributo href do elemento [`<link>`](#), precisa fazer referência a um arquivo em nosso sistema de arquivos.

No exemplo abaixo, o arquivo CSS está na mesma pasta que o documento HTML, mas você pode colocá-lo em outro lugar e reajustar o caminho marcado para encontrá-lo, como a seguir:

```
<!-- Dentro de um subdiretório chamado styles dentro do diretório atual -->
<link rel="stylesheet" href="styles/style.css" />

<!-- Dentro de um subdiretório chamado general, que está em um subdiretório chamado styles, dentro do diretório atual -->
<link rel="stylesheet" href="styles/general/style.css" />

<!-- Suba um nível de diretório, depois entre em um subdiretório chamado styles -->
<link rel="stylesheet" href="../../styles/style.css" />
```

### 3.3.2 Folha de Estilos Interna

Uma folha de estilos interna é usada quando você não tem um arquivo CSS externo, mas, ao contrário, coloca seu CSS dentro de elemento `<style>` localizado no `<head>` do documento HTML.

Deste modo, seu HTML se parecerá assim:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My CSS experiment</title>
    <style>
      h1 {
        color: blue;
        background-color: yellow;
        border: 1px solid black;
      }

      p {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```



Isso pode ser útil em algumas circunstâncias (talvez você esteja trabalhando em um sistema de gerenciamento de conteúdo - CMS - onde não tem permissão para modificar diretamente os arquivos CSS), entretanto isso não é tão eficiente quanto o uso de folhas de estilo externas — em um website, o CSS precisaria ser repetido em todas as páginas e atualizado em vários locais sempre que mudanças fossem necessárias.

### 3.3.3 Estilos Inline

Estilos inline são declarações CSS que afetam apenas um determinado elemento, inserido em um atributo `style`:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My CSS experiment</title>
  </head>
  <body>
    <h1 style="color: blue;background-color: yellow;border: 1px solid black;" >
      Hello World!
    </h1>
    <p style="color:red;">This is my first CSS example</p>
  </body>
</html>
```

**Por favor, não utilize isso a menos que seja estritamente necessário!** É péssimo para manutenção (você precisará atualizar a mesma informação diversas vezes em cada documento), além do que, mistura sua informação de estilização do CSS com sua informação de estrutura HTML, tornando seu código de difícil leitura e compreensão. Manter diferentes tipos de código separados torna o trabalho muito mais fácil para todos os que trabalham no código.

Existem alguns lugares onde o estilo embutido é mais comum, ou mesmo aconselhável. Você pode ter que recorrer ao uso deles se seu ambiente de trabalho for realmente restritivo (talvez o seu CMS permita apenas que você edite o corpo do HTML). Você

também os verá sendo muito usados em e-mails em HTML de modo a obter compatibilidade com o maior número possível de clientes de e-mail

### 3.4 Seletor CSS

Não é possível falar de CSS sem conhecer os seletores. Um seletor é o modo pelo qual nós apontamos para alguma coisa no nosso documento HTML para aplicar os estilos a ela. Se os seus estilos não forem aplicados, então é provável que o seu seletor não esteja ligado àquilo que você pensa que ele deveria.

Cada regra CSS começa com um seletor ou uma lista de seletores para informar ao navegador em qual elemento ou elementos as regras devem ser aplicadas. Todos os exemplos a seguir são válidos como seletores ou listas de seletores.



O **seletor CSS** é parte da regra do CSS que lhe permite selecionar qual elemento(s) vai receber o estilo pela regra.

#### 3.4.1 O que é um seletor


Você já conheceu os seletores. Um seletor CSS é a primeira parte de uma regra CSS. É um padrão de elementos e outros termos que informam ao navegador quais elementos HTML devem ser selecionados para que os valores de propriedade CSS dentro da regra sejam aplicados a eles. O elemento ou elementos que são selecionados pelo seletor são referidos como o *assunto do seletor*.

```
h1 {  
  color: blue;  
  background-color: yellow;  
}  
  
p {  
  color: red;  
}
```

Em CSS, os seletores são definidos na especificação dos seletores CSS; como qualquer outra parte do CSS, eles precisam ter suporte em navegadores para funcionarem. A maioria dos seletores que você encontrará são definidos na especificação de Seletores de nível 3, que é uma especificação madura, portanto, você encontrará um excelente suporte de navegador para esses seletores.

### 3.4.2 Lista de seleção

Se você tiver mais de um item que usa o mesmo CSS, os seletores individuais podem ser combinados em uma lista de seletores para que a regra seja aplicada a todos os seletores individuais. Por exemplo, se eu tiver o mesmo CSS para um `h1` e também para uma classe de `.special`, poderia escrever isso como duas regras separadas.



```
h1 {  
  color: blue;  
}  
  
.special {  
  color: blue;  
}
```

Eu também poderia combiná-los em uma lista de seletores, adicionando uma vírgula entre eles.



```
h1, .special {  
  color: blue;  
}
```

O espaço em branco é válido antes ou depois da vírgula. Você também pode achar os seletores mais legíveis se cada um estiver em uma nova linha.

### 3.4.3 Tipos de seletores

Existem alguns agrupamentos diferentes de seletores e saber qual tipo de seletor você pode precisar o ajudará a encontrar a ferramenta certa para o trabalho. A seguir, examinaremos os diferentes grupos de seletores com mais detalhes.

#### 3.4.3.1 Seletores de tipo, classe e ID

Este grupo inclui seletores que têm como alvo um elemento HTML, como um `<h1>`.

```
h1 {  
  
}
```

Também inclui seletores que direcionam uma classe:

```
.box {  
  
}
```

ou um ID:

```
#unique {  
  
}
```

#### 3.4.3.2 Seletores de atributos

Este grupo de seletores oferece diferentes maneiras de selecionar elementos com base na presença de um determinado atributo em um elemento:

```
a[title] {  
  
}
```

Ou até mesmo faça uma seleção com base na presença de um atributo com um valor específico:

```
a[href="https://example.com"]  
  
{
```

```
}
```

#### 3.4.3.3 Pseudo classes e pseudo-elementos

Este grupo de seletores inclui pseudoclasses, que definem o estilo de certos estados de um elemento. A pseudoclasse `:hover`, por exemplo, seleciona um elemento apenas quando ele está sendo passado pelo ponteiro do mouse:

```
a:hover {  
  
}
```

Também inclui pseudoelementos, que selecionam uma determinada parte de um elemento em vez do próprio elemento. Por exemplo, `::first-line` sempre seleciona a primeira linha de texto dentro de um elemento (a `<p>` no caso abaixo), agindo como se a tivesse `<span>` sido colocado em volta da primeira linha formatada e então selecionado.

```
p::first-line {  
  
}
```

#### 3.4.3.4 Combinadores

O grupo final de seletores combina outros seletores para direcionar os elementos em nossos documentos. O seguinte, por exemplo, seleciona parágrafos que são filhos diretos de `<article>` elementos usando o combinador filho (`>`):

```
article > p {  
  
}
```

### 3.5 Unidades no CSS

Quando começamos a lidar com desenvolvimento web, mais especificamente **HTML e CSS**, é bastante comum que fiquemos presos às ferramentas que já conhecemos e temos familiaridade.

Porém, isso pode se tornar um problema devido ao grande crescimento da Web e, com isso, o surgimento de novos problemas e consequentemente novas soluções. Esse é uma situação corriqueira quando lidamos com unidades de medidas no **CSS**, pela grande variedade, acabamos deixando de lado parte das existentes e não utilizamos, de fato, todo o poder que temos na mão.

Existem algumas unidades que provavelmente você já está acostumado, como o famoso pixel! Nesse post, abordaremos todas as unidades de medidas presentes atualmente e como elas podem nos ajudar durante sua jornada como desenvolvedor web. Antes de prosseguirmos, precisamos entender qual a diferença entre **medida absoluta** e **medida relativa**.

### 3.5.1 Medidas absolutas

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

Essas medidas são fixas e não mudam de acordo com as especificações do dispositivo. Esse tipo de medida é indicado para quando conhecemos perfeitamente as características físicas e as configurações das mídias onde serão exibidos nossos projetos.

### 3.5.1 Medidas relativas

Essas são as que normalmente não estamos habituados. Essas medidas são calculadas tendo como base uma outra unidade de medida definida, como por exemplo

**Em** e o **rem** (veremos mais sobre essas duas medidas no decorrer do post). O uso delas é mais apropriado para que possamos fazer ajustes em diferentes dispositivos garantindo um layout consistente e fluido em diversas mídias.

Devido ao fato de que essas medidas são calculadas pelo browser baseando-se em outra unidade, elas tendem a ser bastante flexíveis. Ou seja, podemos ter resultados diferentes de acordo com o ambiente.

Para começarmos a caminhar pelas medidas existentes, optei por começar pelas medidas absolutas, uma vez que estamos, normalmente, mais familiarizados com essas. Como diria meu amigo Flávio Almeida, bem começado, metade feito!

### **3.5.1 Medidas absolutas no CSS**

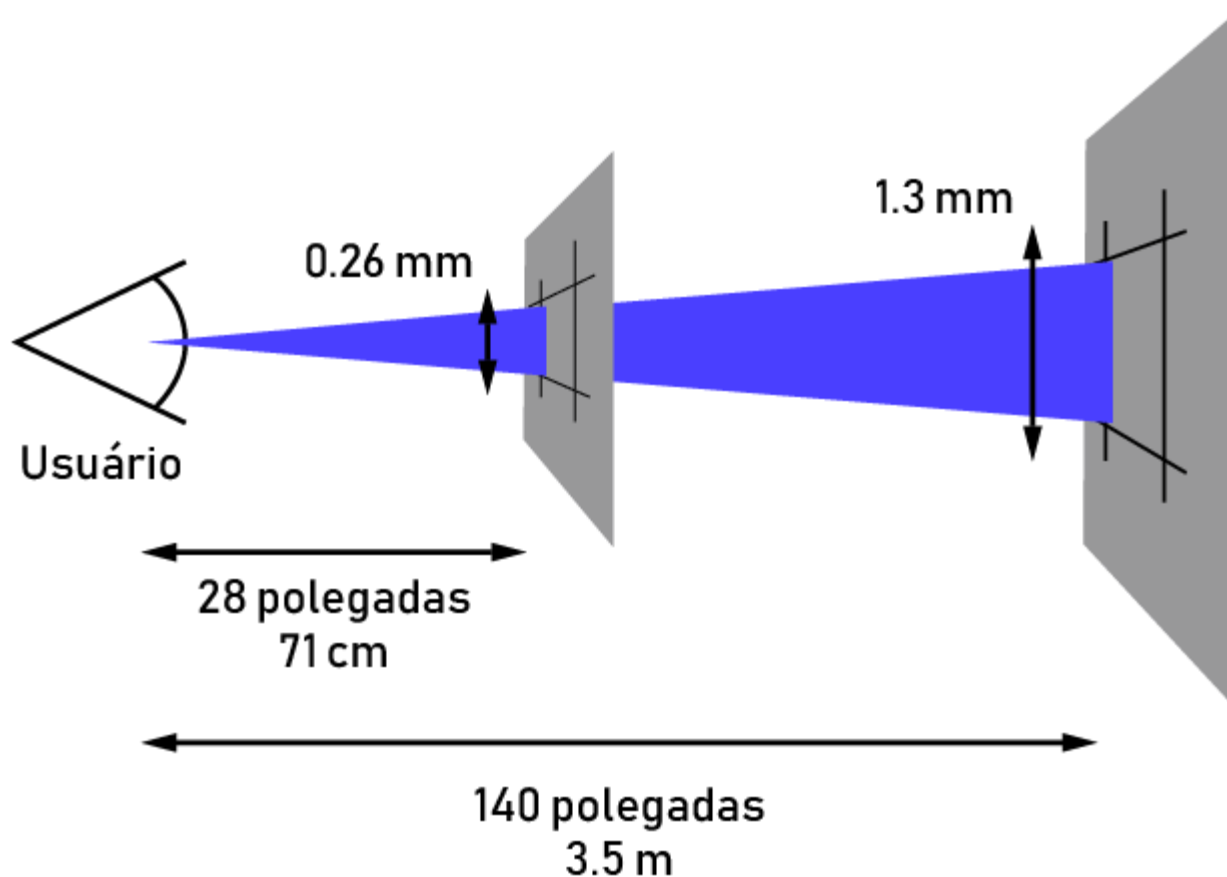
#### **3.5.1.1 Pixel (px)**

Provavelmente você já conhece ou ouviu falar desse rapaz chamado Pixel. Pixel nada mais é do que os pequenos pontinhos luminosos da tela do seu monitor, celular, televisão etc. Logo, o pixel é o menor elemento em um dispositivo de exibição!

Essa é uma medida bastante famosa para os web designers, grande parte dos desenvolvedores web usam o pixel como unidade principal de seus projetos.

Um detalhe que poucos conhecem é que na verdade, o pixel do CSS NÃO é realmente um pixel da tela do dispositivo (hardware), e sim o que chamamos de pixel de referência que geralmente é maior do que o pixel real. O que acaba por torná-lo numa medida abstrata onde é necessário controlar o mapeamento desse pixel de referência para o pixel do hardware (acontece por debaixo dos panos!).

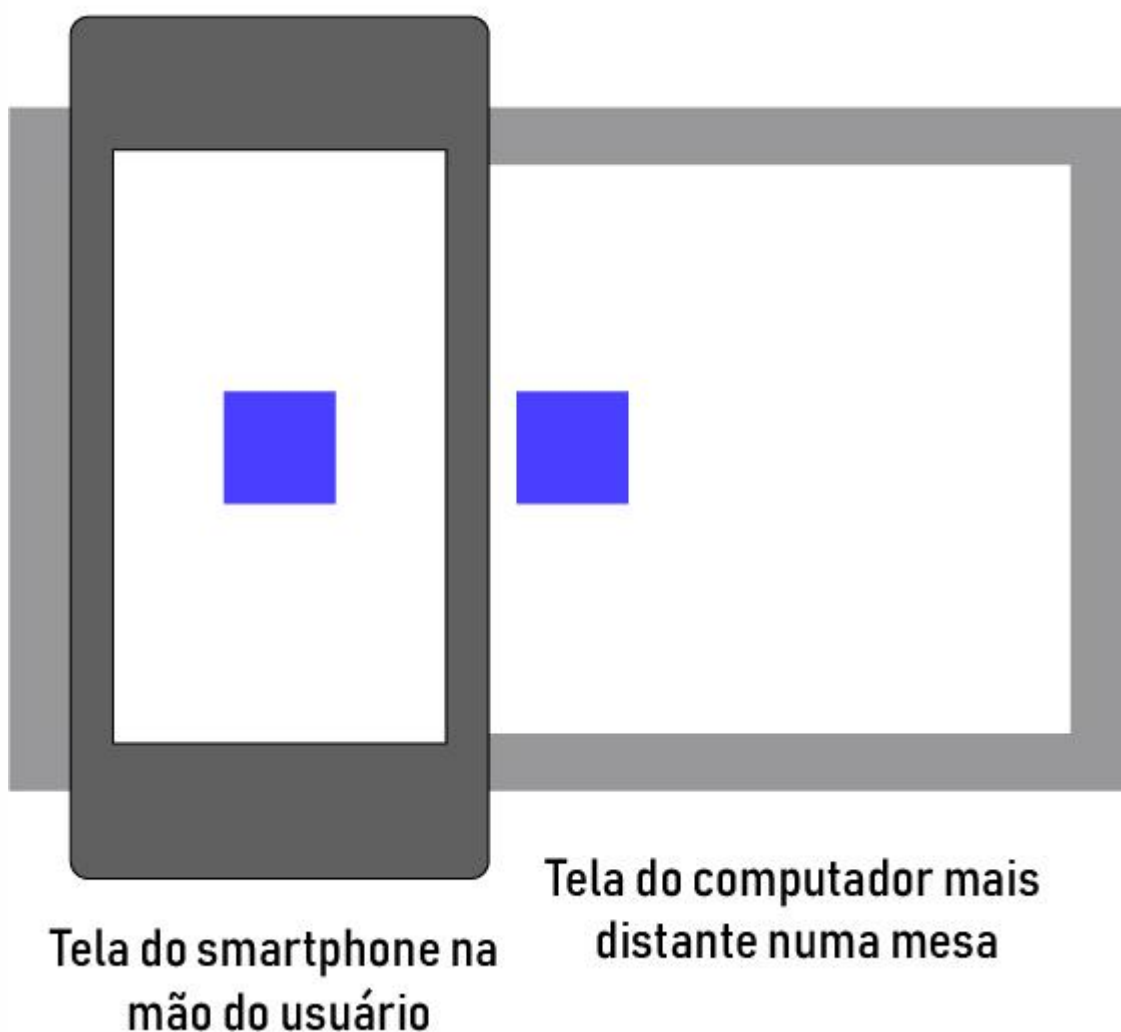
A definição de pixel de referência no CSS é o ângulo visual(0.0213deg) de um pixel em um dispositivo com a densidade de 96dpi a uma distância de um braço do leitor (28 polegadas), veja na imagem abaixo:



O benefício desse pixel de referência é que ele leva a proximidade da tela em consideração, ao usarmos um celular que seguramos próximos de nós, o pixel de referência terá o tamanho semelhante ao de um monitor mais distante de nós, por exemplo.



## Visão do ponto de vista do usuário



Portanto, não existe esse papo de que o pixel é "perfeito". Só se estivermos desenvolvendo um site para um mesmo dispositivo, com um mesmo tamanho de tela e que usa o mesmo navegador, mas sabemos que não é assim que funciona no mundo lá fora!

Um ponto interessante de se comentar é que recentemente o **bootstrap 4** deixou de utilizar **PX** e migrou para **REM**, além disso, o uso do pixel nos dá a sensação de que esse é igual ao pixel do hardware, o que pode gerar confusão para novos desenvolvedores.

Um dos aspectos mais importantes para esses Web Designers é a **escalabilidade e adaptabilidade de um layout**, ou seja, a medida em que as unidades aumentam de uma maneira previsível e razoável, seu layout deve ser capaz de se adequar a essas mudanças.

De um lado, a ideia de se manter o aspecto (tamanho) em diversos dispositivos pode parecer atrativa (cuidado com o pixel de referência!), mas do outro lado, temos consequências negativas quando estamos lidando com dispositivos de baixa resolução (blurry rendering).



#### 3.5.1.2 Points (pt)

A **próxima unidade é Point**. Definitivamente essa unidade é mais conhecida pelos designers, principalmente os que estudam tipografia.

Essa medida é geralmente utilizada em propriedades relacionadas a fonte do seu projeto. Sua abreviação se dá com a marcação de **pt** e seu uso não é tão comum, você provavelmente verá essa unidade muito raramente.

Geralmente espera-se que essa medida seja utilizada em folhas de estilo para impressões, quando se precisa ter certeza do tamanho da fonte utilizada. Não é recomendada para a estilização em tela!

#### 3.5.1.3 in (inches/polegadas)

Polegada ou inch em inglês é mais uma unidade de medida que conhecemos do mundo das medidas absolutas - geralmente vemos elas quando queremos comprar uma nova TV ou monitor, mas essa unidade também existe no mundo Web.

Apesar de existirem, elas não costumam ser utilizadas em projetos, uma vez que não existem um uso prático para elas (podemos atingir os mesmos resultados utilizando outras unidades)

#### 3.5.1.4 in Centímetro e Milímetro (cm / mm)

Nós brasileiros, que adotamos o sistema métrico, conhecemos bem essas duas medidas, que são bastante utilizadas no dia a dia. Apesar de bastante comuns, tanto centímetro e milímetro são pouco usadas no CSS. Assim como o pt, o uso dessas duas é esperado para folhas de estilo para impressões (medidas mais precisas), evitando que elas sejam aplicadas para exibições em tela.

#### 3.5.1.5 Paica (pc)

Também uma unidade pouco usada no mundo web, a **Paica** também vem para o CSS sendo herdada da tipografia. Por não ser uma unidade amplamente conhecida, ela acaba sendo fadada ao esquecimento, mas é sempre importante conhecermos todas as ferramentas que estão à nossa disposição. A relação entre as unidades absolutas é:

$$1\text{ in} = 2,54\text{cm} = 25,4\text{mm} = 72\text{pt} = 6\text{pc}$$

### 3.5.2 Medidas Relativas no CSS

#### 3.5.2.1 Ems (em)

Nossa primeira unidade relativa é bastante famosa no mundo **CSS**. Dificilmente você achará algum navegador que não tenha suporte para essa medida, que está presente desde os primórdios. Até para o IE, nós teríamos que usar a versão abaixo da 3.0 para que tivéssemos algum problema.

Esse definitivamente é um dos pontos que fazem o **em** tão popular. O segundo ponto, com certeza se dá a facilidade de criar layouts fluídos e responsivos.

Mas como funciona esse tal de **em**? Essa unidade muda para os elementos filhos de acordo com o tamanho da fonte (font-size) do elemento pai, então vamos lá. Digamos que temos o seguinte html, me permitindo a licença poética de utilizar a tag style:

```
<style>
  #pai{
    font-size: 16px;
  }

  #filho{
    font-size: 2em;
  }
</style>

<div id="pai">
  div pai
  <div id="filho">
    div filho
  </div>
</div>
```

Acima, temos uma div pai onde estou definindo um **font-size** de 16px, dentro dessa div, temos uma única div filha. Como havia mencionado, o tamanho definido para a fonte impactará no **em** dos elementos filhos.

Nesse nosso caso, para a div mais interna (id=filho), **1em será igual a 16px**, seguindo a lógica, **2em será igual a 32px** e assim por diante. Podemos colocar valores como 1.5 também! Nesse nosso caso, **1.5em será igual a 24px**. Quando expressamos tamanhos como margin, padding utilizando **em**, isso significa que eles serão relativos ao tamanho da fonte do elemento pai.

Portanto, de acordo com o tamanho da fonte utilizada em determinado elemento, os elementos filhos serão redimensionados de forma a obedecer a referência a esse tamanho de fonte!

Uma técnica bastante utilizada consiste justamente em fazer uso desse poder do **em** componentizando nossos elementos. A ideia é que a alteração do tamanho da fonte do elemento pai faça com que todo o componente se modifique e redimensione baseando-se nesse novo valor.

Apesar de divertido, o motivo de utilizarmos essa técnica não é para que o usuário tenha um slider e altere o tamanho da fonte. Mas sim para facilitar a manutenção do componente como um todo, sem ter que sofrer alterando valores de todas as partes do componente. Bem legal né?

O último ponto que devemos nos atentar ao usar o **em** é que quando usamos essa medida, nós temos que considerar o **font-size** de todos os elementos pai. Por exemplo, se tivéssemos uma terceira div mais interna no nosso exemplo anterior e definirmos o tamanho da fonte para **2em**, nesse caso esses **2em seriam 64px**, uma vez que o font-size do elemento pai foi definido sendo **32px(2em)**! Pegou o pulo do gato?

Isso tende a se complicar quando estamos falando de 5, 6, 7 divs aninhadas, provavelmente não será muito divertido calcular isso! Mas a boa notícia é que temos uma unidade que nos ajuda a resolver esse probleminha.

### 3.5.2.2 Rems (rem, "root em")

O **REM** vem como sucessor do **EM** e ambos compartilham a mesma lógica de funcionamento (font-size), porém a forma de implementação é diferente. Enquanto o **em** está diretamente relacionado ao tamanho da fonte do elemento pai, o **rem** está relacionado com o tamanho da fonte do elemento root (raiz), no caso, a **tag**.

O fato de que o **rem** se relaciona com o **elemento raiz** resolve aquele problema que tínhamos com diversas divs (elementos) aninhados, uma vez que não haverá essa "herança" de tamanhos, lembra?! Ou seja, não precisaremos ter dor de cabeça tendo que realizar cálculos, uma vez que nos baseamos na tag raiz.

Exemplificando, sabemos que a tag html é a tag raiz de todo documento html. Dito isso, se definirmos que o **font-size** desse elemento será de 18px, então **1rem = 18px**, **2rem = 36px** e assim por diante... Normalmente os browsers especificam o tamanho default da fonte do elemento root (raiz) sendo 16px, então guarde isso no coração! Mesmo essa unidade sendo mais tranquila de se trabalhar, ela não era muito utilizada para design responsivo, o que de primeira pode soar um tanto quanto estranho.

O motivo para isso é o suporte para essa medida. O chrome e o firefox suportavam tranquilamente, assim como o Opera e o Safari, porém, antigamente grande parte dos usuários utilizavam o IE, mais especificamente o IE 8, e esse browser não lidava muito bem com os **rems**, isso fazia com que os desenvolvedores precisassem optar por alguma unidade diferente, em muitos casos, o próprio **em**.

Como disse acima, o valor base é 16px, e isso pode acabar gerando dificuldades para que encontremos alguns tamanhos padrões que costumam ser utilizados. Por exemplo, como faríamos para atingir um tamanho de **10px** utilizando **rem**? Precisamos calcular.

## BASE 16PX

```
10px = 0.625rem
12px = 0.75rem
14px = 0.875rem
16px = 1rem
18px = 1.125rem
```

e assim por diante, realmente não são números muito “amigáveis” ou convenientes porém, podemos lançar mão de um pequeno truque para nos ajudar (62,5%)

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

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

p{
    font-size: 2.4rem; /*equivalente a 24px*/
}
```

Repare que dessa forma, o valor em pixel será sempre o valor definido em rem vezes 10! Fica mais conveniente, concorda?

Apesar de parecer uma boa ideia, devemos ter cuidado com essas abordagens, uma vez que ela forçará que você reescreva todos os font-size do seu site, então tome cuidado!

Existe uma terceira visão sobre isso tudo. Essa solução utiliza px, em e rem de maneira bem definida. A ideia consiste em definir o font-size do elemento root em pixel, módulos utilizando rem e elementos interiores aos módulos utilizando em, facilitando a manipulação do tamanho global que naturalmente escalará o tamanho para os módulos (utilizando rem) e esses por sua vez escalarão os elementos interiores (que utilizam em e referenciam ao elemento pai).

### 3.5.2.3 Porcentagem (%)

Apesar de não ser uma unidade de medida, a porcentagem costuma ser bastante utilizada quando falamos de layout responsivo e fluido, por isso, não poderia deixá-la passar.

A porcentagem permite que criemos módulos que sempre vão se readaptar para ocupar a quantidade especificada. Por exemplo, se definirmos um elemento tendo um tamanho de 50%, independente do dispositivo em questão, esse módulo sempre ocupará metade do espaço que lhe cabe (caso esteja dentro de algum outro elemento).

Repare que se alterarmos o tamanho da div container, o elemento interior a ela se redimensionará de forma a sempre ocupar a porcentagem especificada no CSS!

Veja também que o slider está alterando tanto a largura quanto a altura do elemento, mas se alterarmos cada um separadamente, também funcionará! Visto esse exemplo, podemos dizer que a porcentagem tem um comportamento um tanto parecido ao nosso já conhecido **em**, já que ele se relaciona diretamente com o tamanho da propriedade do elemento pai.

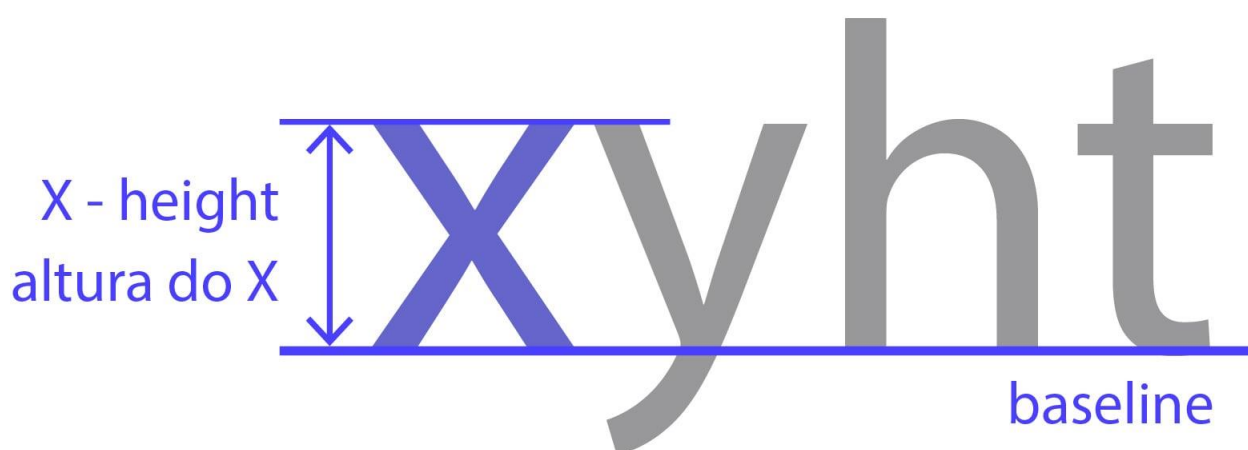
Portanto, ao trabalharmos com a porcentagem, temos o mesmo problema que tínhamos com o **em**, quanto mais elementos aninhados, mais complicado será de definirmos exatamente o tamanho, por isso, tenha cuidado quando utilizá-la!

### 3.5.2.4 Ex

Talvez você nunca tenha ouvido falar dessa unidade do CSS, mas ela existe. Diferentemente da forma como a **EM** e a **REM** funcionavam, essa unidade não se

relaciona com o tamanho da fonte (font-size), mas com qual fonte está sendo utilizada naquele momento (font-family), mais especificamente ao tamanho do caractere **x** dessa fonte em questão (x-height).

Como o browser sabe esse valor? Esse valor pode vir diretamente com a fonte, o browser pode medir o caractere em caixa baixa (lower case) e se esses dois não funcionarem, o browser estipula um valor de **0.5em** para **1ex**.



Com isso, se quando mudamos o tamanho da fonte (depende do elemento) o **em** e o **rem** mudam, dessa vez, quando alteramos completamente a fonte, o **ex** mudará. O uso dessa unidade está mais presente em ajustes tipográficos, nos dando um controle mais preciso quando o padrão definido para algumas tags não se adequa corretamente ao nosso layout.

### 3.5.2.5 Ch

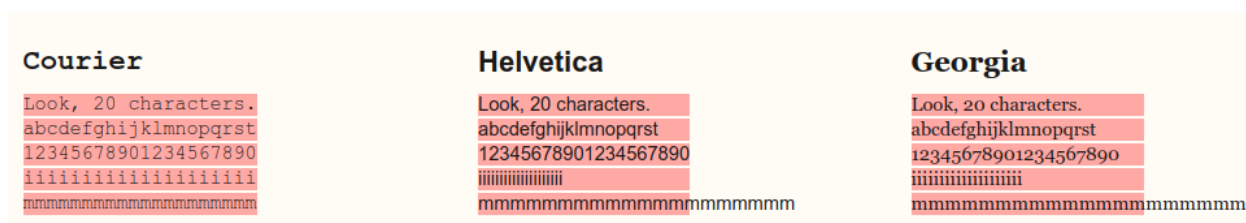
Também uma unidade pouco conhecida, o **ch** (character unit) é definida na documentação como sendo a "medida avançada" da largura do caractere zero ("0").

Existe uma discussão antiga onde se debateu bastante sobre essa unidade e o que realmente seria "medida avançada", você pode acompanhar aqui. A ideia é que um elemento com, por exemplo **100ch** de largura poderá comportar uma string de 100



caracteres dessa determinada fonte, caso essa fonte seja monospace (todos os caracteres têm o mesmo tamanho).

É comum acharmos definições que dizem que a frase acima se aplica para qualquer fonte, porém isso está errado. Como mencionei acima, a regra de **1ch = 1 caractere** se aplica apenas se a fonte usada for **monospace** (largura fixa). Fontes com a largura variável, qualquer caractere pode ser mais largo ou menos largo que o zero ("0"), como podemos ver na imagem abaixo:



Como podemos analisar, a tipografia Courier (monospace) obedece a regra acima, porém as outras duas não! O que podemos tirar após observações é que normalmente **1ch** é 20% - 30% mais largo, porém isso não é uma verdade absoluta e deve ser observado para cada fonte que você deseja aplicar. Por isso, tome cuidado com o uso!

### 3.5.2.6 Vw (viewport width)

Essa medida faz parte das medidas mais atuais e do futuro do CSS. Viewport units.

Como escrito no título, **vw** significa viewport width, mas o que é viewport?

Viewport nada mais é que a área visível de uma página web para o seu usuário, essa viewport pode variar de acordo com o dispositivo, sendo menor em celulares e maior em desktops.

Antigamente, quando não existiam tablets e celulares capazes de acessar sites, todas as webs pages eram pensadas para a tela de um computador, com tamanho fixo e design estático. Com a chegada desses dispositivos móveis, essas páginas eram grandes demais para serem exibidas nesses aparelhos, o que tornava muito difícil a navegação.

A primeira solução partiu dos browsers desses dispositivos, eles adotavam um comportamento de retirar o zoom de forma que o site inteiro coubesse na tela do

aparelho, definitivamente não era o ideal, mas uma solução rápida. No HTML5, foi introduzido uma maneira para que os desenvolvedores conseguissem alterar a viewport através da **tag**, corrigindo esse problema de usabilidade relacionado aos dispositivos móveis, mas isso é assunto para outra postagem!

Voltando para o nosso querido **vw**, essa unidade se relaciona diretamente com a largura da viewport, onde **1vw** representa **1%** do tamanho da largura dessa área visível. A diferença entre **vw** e a % é bem semelhante a diferença entre **em** e **rem**, onde a % é relativa ao contexto local do elemento e o **vw** é relativo ao tamanho total da largura da viewport do usuário.

#### 3.5.2.7 Vh (viewport height)

Essa unidade funciona da mesma forma que o **vw**, porém dessa vez, a referência será a altura e não a largura. Existem diversos exemplos práticos e interessantes de uso dessas duas unidades.

#### 3.5.2.8 Vmin (viewport minimum)

Essa unidade também se relaciona com as dimensões da viewport, mas com um porém. Anteriormente quando vimos **vh** e **vw** precisávamos escolher se gostaríamos de nos basear na altura (**vh**) ou na largura (**vw**) da viewport.

Diferentemente das anteriores, o **vmin** utilizará como base a menor dimensão da viewport (altura x largura), vamos ao exemplo.

Imagine que estamos trabalhando com uma viewport de 1600px de altura e 900px de largura. Nesse caso, **1vmin** terá o valor de **9px** (1% da menor dimensão!), caso tenhamos **100vmin**, esse será igual a **900px**!

No caso acima, a menor dimensão foi a da largura, porém se tivéssemos 300px para altura e 1400px para largura, nosso valor de referência seria o 300px! Sempre a **menor** dimensão!

### 3.5.2.9 Vmax (viewport maximum)

Seguindo a mesma base lógica da unidade anterior, o **vmax** terá como valor de referência a maior dimensão da viewport. Ou seja, utilizando o mesmo exemplo, se tivermos 1600px de altura e 900px de largura, **1vmax** será equivalente a **16px**!

No segundo exemplo ocorrerá a mesma inversão, tendo 300px para altura e 1400px para largura, **1vmax** será equivalente a **14px**. Dessa vez sempre será a **maior** dimensão!

## Conclusão

Como podemos perceber, existem várias unidades que podemos utilizar no mundo web, mas sempre surgem aquelas perguntas de quando tenho que utilizar? Qual a melhor? etc...

Não existe resposta certa nem errada para essas perguntas, infelizmente não temos uma regra de ouro para todas as situações. O uso dessas unidades depende de diversos fatores como equipe, preferência, familiaridade e assim por diante.

Entretanto, é importante que você como programador fullstack tenha todas essas ferramentas no seu cinto de utilidades já que nunca sabemos quando precisaremos utilizar.

Definitivamente a inclusão das unidades da **viewport** foram positivas para a web quando estamos lidando com layout flexível, cabe a você começar a colocá-las em uso durante seu dia a dia!

No mais, espero de coração que esse post tenha agregado algum conhecimento e facilitado sua jornada rumo ao saber.

## 3.6 Flexbox CSS

O Flexbox tem como meta ser um modo mais eficiente para criar leiautes, alinhar e distribuir espaços entre itens em um container, mesmo quando as dimensões destes itens são desconhecidas e/ou dinâmicas (daí o termo "flex").

Vamos aprender os fundamentos do CSS Flexbox para alinhamento e posicionamento, e como utilizar suas funcionalidades corretamente.

### 3.6.1 O que é o Flexbox

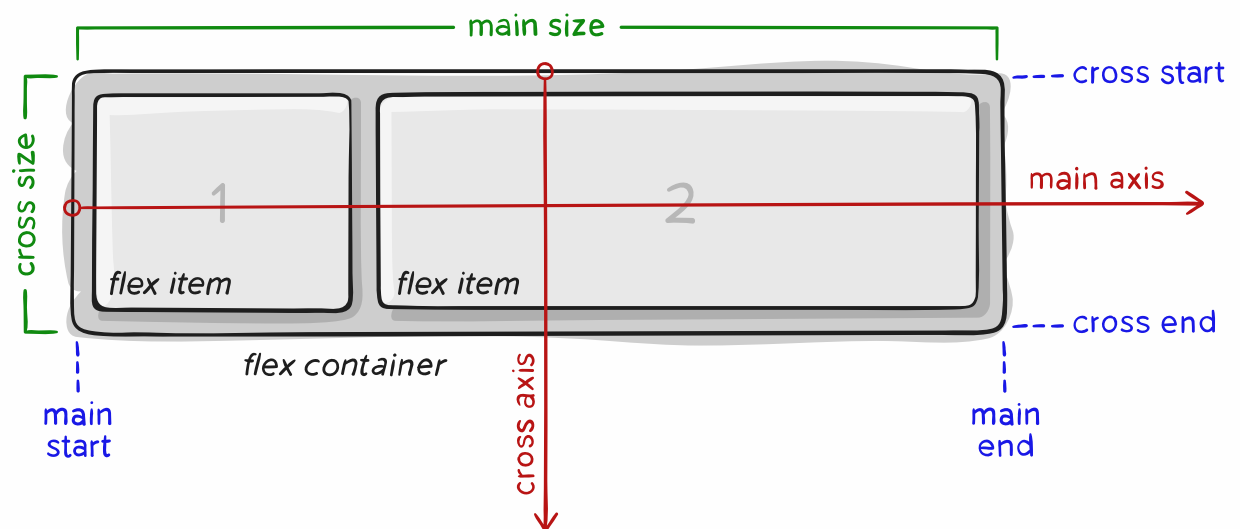
Por muito tempo, as únicas ferramentas disponíveis para criar leiautes em CSS e posicionar elementos com boa compatibilidade entre browsers eram `float` e `position`. Porém, essas ferramentas possuem algumas limitações muito frustrantes, especialmente no que diz respeito à responsividade. Algumas tarefas que consideramos básicas em um leiaute, como centralização vertical de um elemento-filho com relação a um elemento-pai ou fazer com que elementos-filhos ocupem a mesma quantidade de espaço, ou colunas terem o mesmo tamanho independente da quantidade de conteúdo interno, eram impossíveis ou muito difíceis de serem manejadas com floats ou position, ao menos de forma prática e `flexível`.

A ferramenta Flexbox (de Flexible Box) foi criada para tornar essas tarefas mais simples e funcionais: os filhos de um elemento com Flexbox podem se posicionar em qualquer direção e pode ter dimensões flexíveis para se adaptar.

### 3.6.2 Elementos

O Flexbox é um módulo completo e não uma única propriedade; algumas delas devem ser declaradas no container (o elemento-pai, que chamamos de *flex container*), enquanto outras devem ser declaradas nos elementos-filhos (os *flex itens*).

Se o leiaute "padrão" é baseado nas direções block e inline, o leiaute Flex é baseado em direções "flex flow". Veja abaixo um diagrama da especificação, explicando a ideia central por trás do leiaute Flex.



Os ítems serão dispostos no leiaute seguindo ou o eixo principal ou o transversal.

- Eixo principal: o eixo principal de um *flex container* é o eixo primário e ao longo dele são inseridos os *flex items*. **Cuidado:** O eixo principal não é necessariamente horizontal; vai depender da propriedade *flex-direction* (veja abaixo).
- *main-start* | *main-end*: os *flex items* são inseridos dentro do container começando pelo lado *start*, indo em direção ao lado *end*.
- Tamanho principal: A largura ou altura de um *flex item*, dependendo da direção do container, é o tamanho principal do ítem. A propriedade de tamanho principal de um *flex item* pode ser tanto *width* quanto *height*, dependendo de qual delas estiver na direção principal.
- Eixo transversal: O eixo perpendicular ao eixo principal é chamado de eixo transversal. Sua direção depende da direção do eixo principal.
- *cross-start* | *cross-end*: Linhas flex são preenchidas com ítems e adicionadas ao container, começando pelo lado *cross start* do *flex container* em direção ao lado *cross end*.

- *cross size*: A largura ou altura de um *flex item*, dependendo do que estiver na dimensão transversal, é o *cross size* do item. A propriedade *cross size* pode ser tanto a largura quanto a altura do item, o que estiver na transversal.

**Flex container** é o elemento que envolve sua estrutura. Você define que um elemento é um Flex Container com a propriedade `display` e valores `flex` ou `inline-flex`.

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

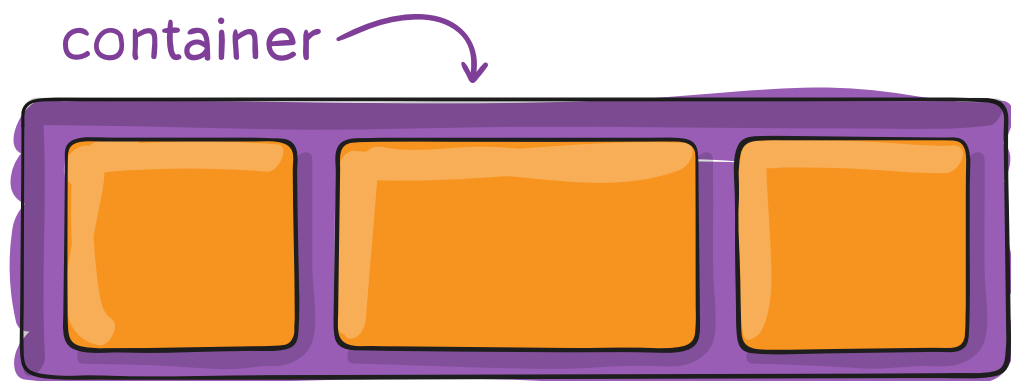
---

```
.flex-container {
  display: flex;
}
```

**Flex Item** são elementos-filhos do flex container.

**Eixos ou Axes** são as duas direções básicas que existem em um Flex Container: **main axis**, ou eixo principal, e **cross axis**, ou eixo transversal.

### 3.6.3 Propriedades para o elemento-pai



Quando utilizamos o *Flexbox*, é muito importante saber quais propriedades são declaradas no elemento-pai (por exemplo, uma *div* que irá conter os elementos a serem alinhados) e quais serão declaradas nos elementos-filhos. Abaixo, seguem propriedades que devem ser declaradas utilizando o elemento-pai como seletor (para alinhar elementos-filhos):

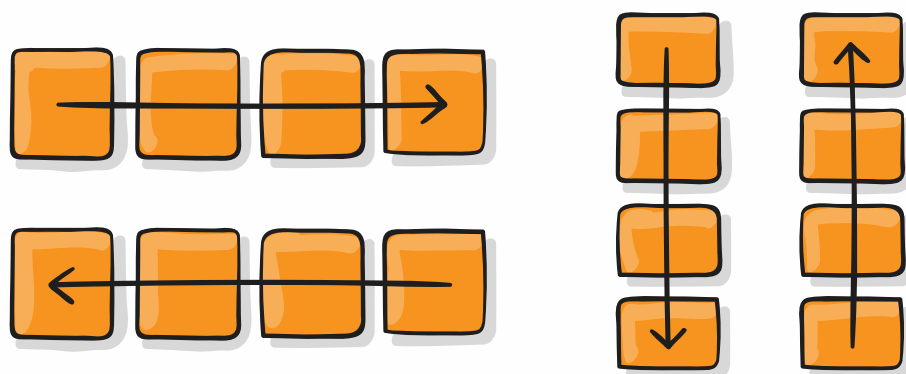
### **display**

Esta propriedade define um *flex container*, inline ou block dependendo dos valores passados. Coloca todos os elementos-filhos diretos num contexto Flex.

```
.container {  
  display: flex; /* or inline-flex */  
}
```

Note que a propriedade de CSS `columns` não tem efeito em um *flex container*.

#### **3.6.3.1 flex-direction**

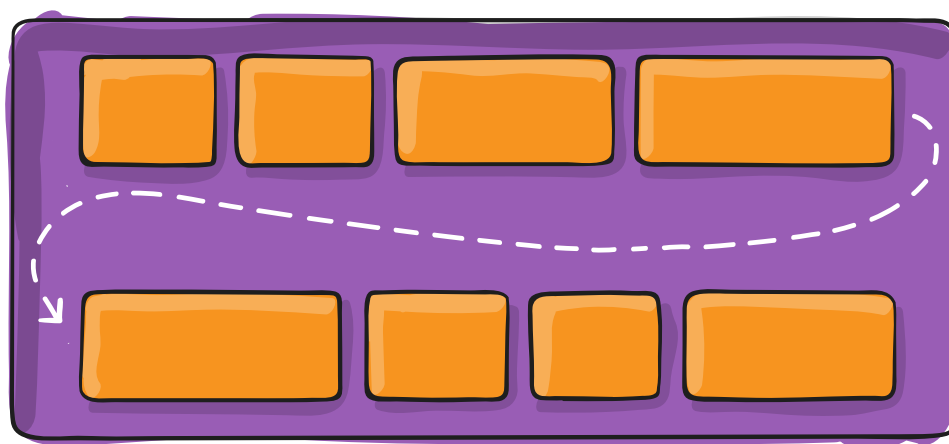


Estabelece o eixo principal, definindo assim a direção em que os *flex items* são alinhados no *flex container*. O Flexbox é (com exceção de um wrapping opcional) um conceito de layout de uma só direção. Pense nos *flex items* inicialmente posicionais ou em linhas horizontais ou em colunas verticais.

```
.flex-container {  
  flex-direction: row | row-reverse | column | column-reverse;  
}
```

- **row** (padrão): esquerda para a direita em **ltr** (left to right), direita para a esquerda em **rtl** (right to left)
- **row-reverse**: direita para a esquerda em **ltr**, esquerda para a direita em **rtl**
- **column**: mesmo que **row**, mas de cima para baixo
- **column-reverse**: mesmo que **row-reverse** mas de baixo para cima

### 3.6.3.2 flex-wrap



Por padrão, os *flex items* vão todos tentar se encaixar em uma só linha. Com esta propriedade você pode modificar esse comportamento e permitir que os itens quebrem para uma linha seguinte conforme for necessário.

```
.flex-container {
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

- **nowrap** (padrão): todos os *flex items* ficarão em uma só linha
- **wrap**: os *flex items* vão quebrar em múltiplas linhas, de cima para baixo
- **wrap-reverse**: os *flex items* vão quebrar em múltiplas linhas de baixo para cima



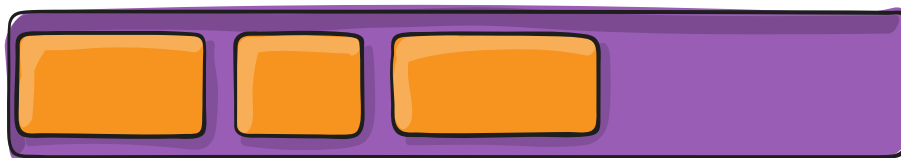
### 3.6.2.3 *flex-flow*

A propriedade **flex-flow** é uma propriedade *shorthand* (uma mesma declaração inclui vários valores relacionados a mais de uma propriedade) que inclui **flex-direction** e **flex-wrap**. Determina quais serão os eixos principal e transversal do container. O valor padrão é **row nowrap**.

```
.flex-container {  
  flex-flow: row nowrap | row wrap | column nowrap | column  
wrap;  
}
```

### 3.6.3.4 justify-content

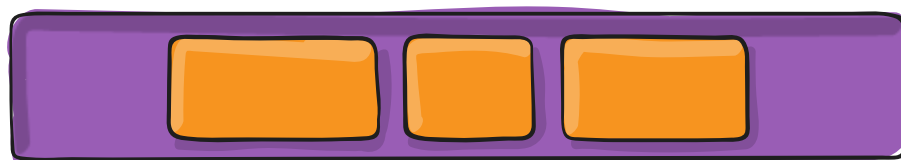
flex-start



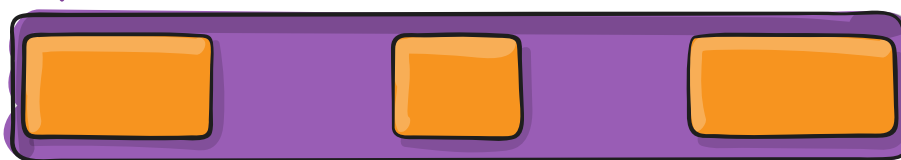
flex-end



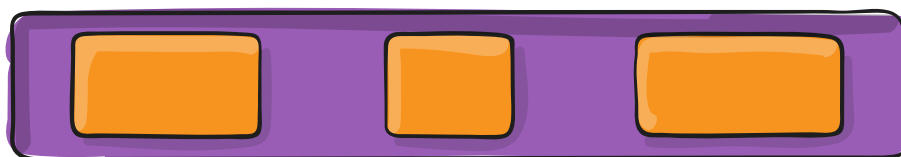
center



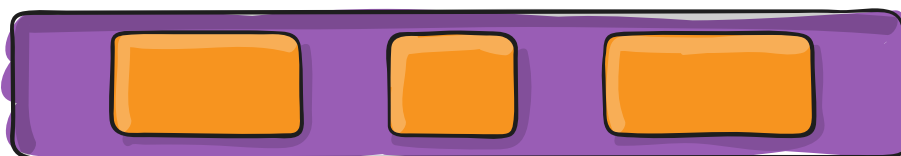
space-between



space-around



space-evenly



Esta propriedade define o alinhamento dos itens ao longo do eixo principal. Ajuda a distribuir o espaço livre que sobrar no container tanto se todos os flex items em uma linha são inflexíveis, ou são flexíveis, mas já atingiram seu tamanho máximo. Também exerce algum controle sobre o alinhamento de itens quando eles ultrapassam o limite da linha.

```
.flex-container {  
  justify-content: flex-start | flex-end | center | space-  
between | space-around | space-evenly;  
}
```

- **flex-start** (padrão): os itens são alinhados junto à borda de início (start) de acordo com qual for a `flex-direction` do container.
- **flex-end**: os itens são alinhados junto à borda final (end) de acordo com qual for a `flex-direction` do container.
- **start**: os itens são alinhados junto à borda de início da direção do `writing-mode` (modo de escrita).
- **end**: os itens são alinhados junto à borda final da direção do `writing-mode` (modo de escrita).
- **left**: os itens são alinhados junto à borda esquerda do container, a não ser que isso não faça sentido com o `flex-direction` que estiver sendo utilizado. Nesse caso, se comporta como `start`.
- **right**: os itens são alinhados junto à borda direita do container, a não ser que isso não faça sentido com o `flex-direction` que estiver sendo utilizado. Nesse caso, se comporta como `start`.
- **center**: os itens são centralizados na linha.
- **space-between**: os itens são distribuídos de forma igual ao longo da linha; o primeiro item junto à borda inicial da linha, o último junto à borda final da linha.
- **space-around**: os itens são distribuídos na linha com o mesmo espaçamento entre eles. Note que, visualmente, o espaço pode não ser igual, uma vez que todos os itens têm a mesma quantidade de espaço dos dois lados: o primeiro item

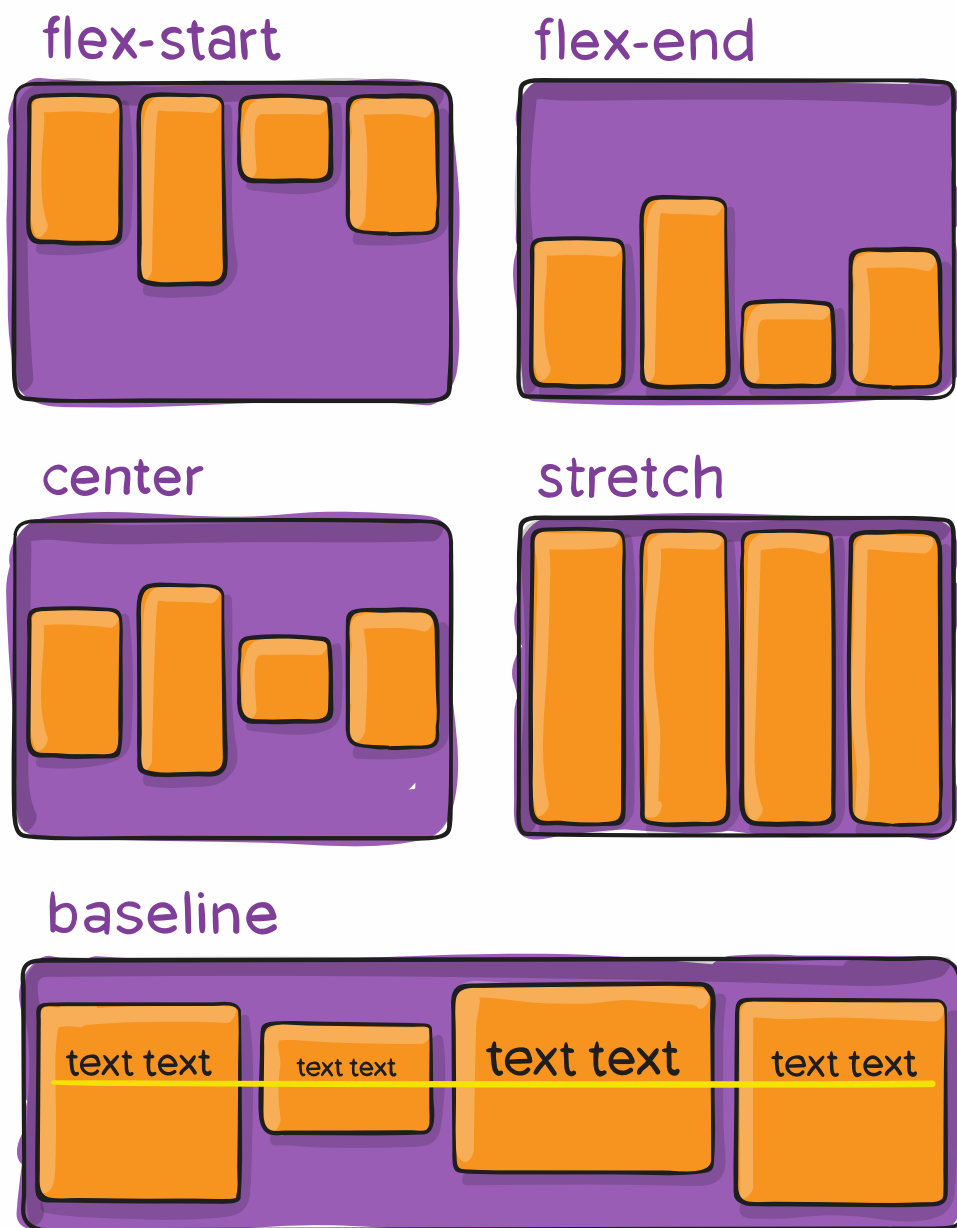
vai ter somente uma unidade de espaço junto à borda do container, mas duas unidades de espaço entre ele e o próximo item, pois o próximo item também tem seu próprio espaçamento que está sendo aplicado.

- **space-evenly:** os itens são distribuídos de forma que o espaçamento entre quaisquer dois itens da linha (incluindo entre os itens e as bordas) seja igual.

Nota: o suporte dado pelos navegadores para estes valores é difuso. Por exemplo, space-between não tem suporte em nenhuma versão do Edge (até a elaboração deste tutorial) e start/end/left/right ainda não foram implementados no Chrome. Para tabelas detalhadas, consulte o MDN. Os valores mais seguros são flex-start, flex-end e center.

Também existem duas palavras-chave adicionais que você pode usar em conjunto com estes valores: **safe** e **unsafe**. Safe garante que, independente da forma que você faça esse tipo de posicionamento, não seja possível "empurrar" um elemento e fazer com que ele seja renderizado para fora da tela (por exemplo, acima do topo), de uma forma que faça com que o conteúdo seja impossível de movimentar com a rolagem da tela (o CSS chama isso de "perda de dados").

### 3.6.3.5 align-items



Define o comportamento padrão de como *flex items* são alinhados de acordo com o eixo transversal (*cross axis*). De certa forma, funciona de forma similar ao `justify-content`, porém no eixo transversal (perpendicular ao eixo principal).

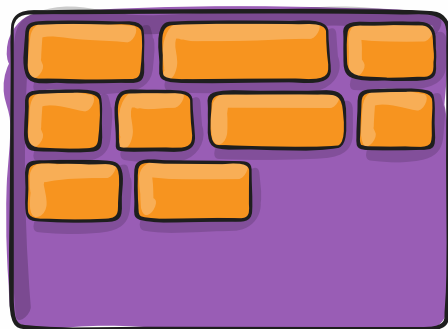
```
.flex-container {  
  align-items: stretch | flex-start | flex-end | center |  
baseline;  
}
```

- **stretch (padrão):** estica os itens para preencher o container, respeitando o `min-width/max-width`).
- **flex-start/ start / self-start:** itens são posicionados no início do eixo transversal. A diferença entre eles é sutil e diz respeito às regras de `flex-direction` ou `writing-mode`.
- **center:** itens são centralizados no eixo transversal.
- **baseline:** itens são alinhados de acordo com suas baselines.

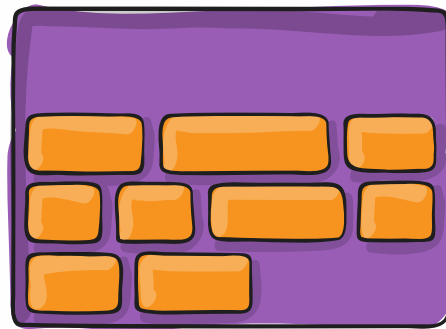
Os modificadores `safe` e `unsafe` podem ser usados em conjunto com todas essas palavras-chave (favor conferir o suporte de cada navegador) e servem para prevenir qualquer alinhamento de elementos que faça com que o conteúdo fique inacessível (por exemplo, para fora da tela).

### 3.6.3.6 align-content

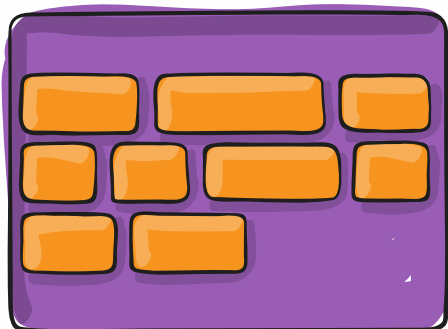
flex-start



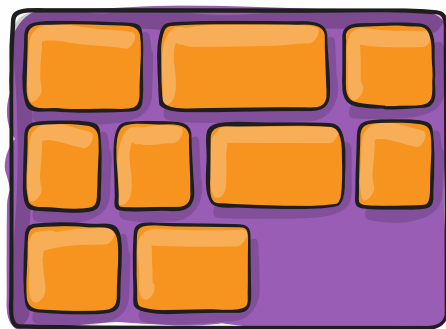
flex-end



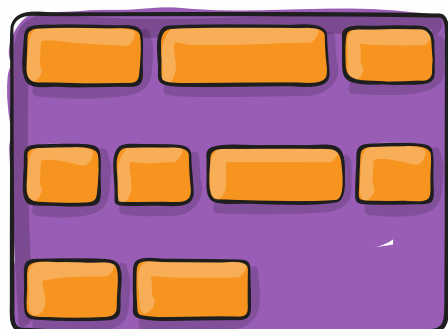
center



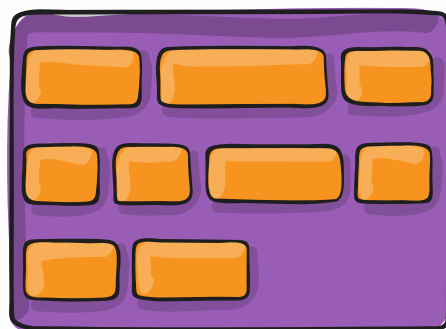
stretch



space-between



space-around



Organiza as linhas dentro de um flex container quando há espaço extra no eixo transversal, similar ao modo como **justify-content** alinha itens individuais dentro do eixo principal.

**Importante:** Esta propriedade não tem efeito quando há somente uma linha de flex items no container.

```
.flex-container {  
  align-content: flex-start | flex-end | center | space-  
between | space-around | stretch;  
}
```

- **flex-start / start:** itens alinhados com o início do container. O valor (com maior suporte dos navegadores) `flex-start` se guia pela `flex-direction`, enquanto `start` se guia pela direção do `writing-mode`.
- **flex-end / end:** itens alinhados com o final do container. O valor (com maior suporte dos navegadores) `flex-end` se guia pela `flex-direction`, enquanto `end` se guia pela direção do `writing-mode`.
- **center:** itens centralizados no container.
- **space-between:** itens distribuídos igualmente; a primeira linha junto ao início do container e a última linha junto ao final do container.
- **space-around:** itens distribuídos igualmente com o mesmo espaçamento entre cada linha.
- **space-evenly:** itens distribuídos igualmente com o mesmo espaçamento entre eles.
- **stretch (padrão):** itens em cada linha esticam para ocupar o espaço remanescente entre elas.

Os modificadores `safe` e `unsafe` podem ser usados em conjunto com todas essas palavras-chave (favor conferir o suporte de cada navegador) e servem para prevenir qualquer alinhamento de elementos que faça com que o conteúdo fique inacessível (por exemplo, para fora da tela).



### 3.6.4 Propriedades para elementos-filhos

A seguir, veremos propriedades que devem ser declaradas tendo como seletor os elementos-filhos, ou seja:

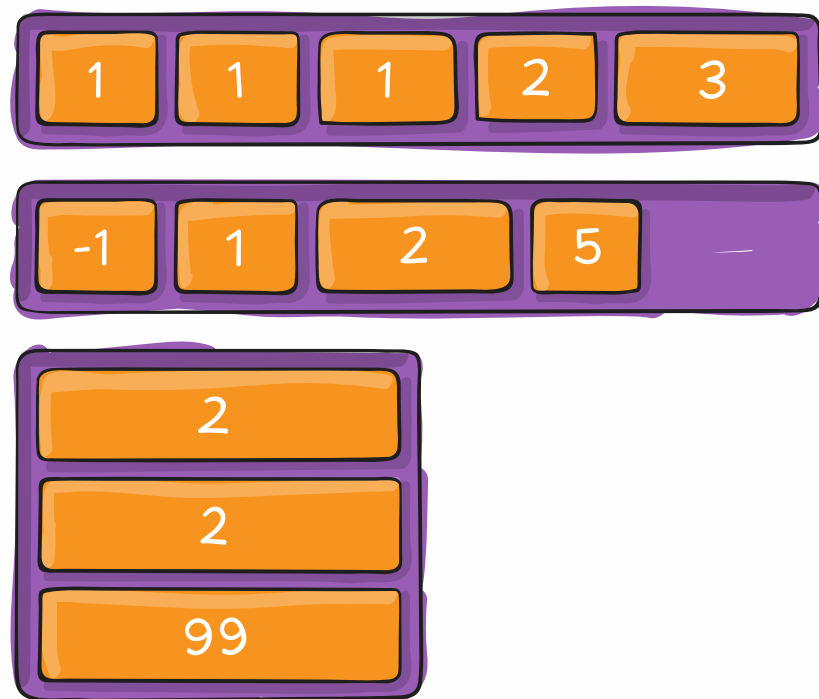
```
<div class="flex-container">  
  <div class="flex-item">1</div>  
  <div class="flex-item">2</div>  
  <div class="flex-item">3</div>  
</div>
```

Isso significa que, onde existe um elemento-pai com propriedade *flex* (o *flex-container*), é possível atribuir propriedades flex específicas também para os elementos-filhos (*flex-item*).

Você pode definir as propriedades abaixo para apenas um dos elementos-filhos através de um identificador, como uma classe específica.

#### 3.6.4.1 order

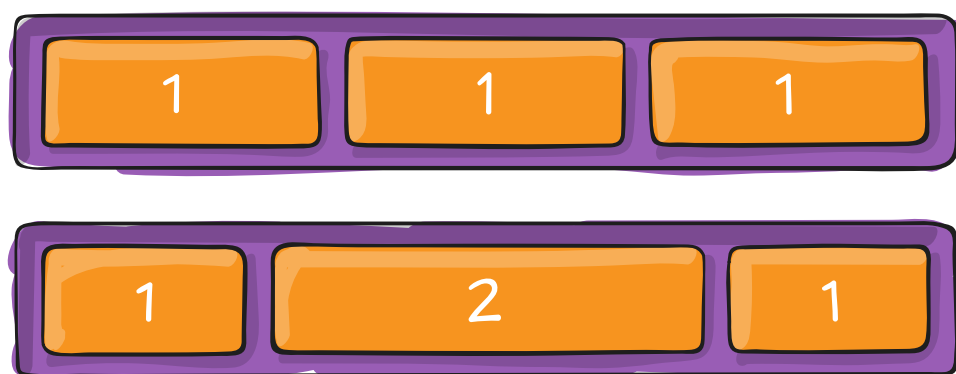
Determina a ordem em que os elementos aparecerão.



Por padrão os flex items são dispostos na tela na ordem do código. Mas a propriedade `order` controla a ordem em que aparecerão no container.

```
.flex-item {
  order: <número>; /* o valor padrão é 0 */
}
```

#### 3.6.4.2 *flex-grow*



Define a habilidade de um flex item de crescer, caso necessário. O valor dessa propriedade é um valor numérico sem indicação de unidade, que serve para cálculo de proporção. Este valor dita a quantidade de espaço disponível no container que será ocupado pelo item.

Se todos os itens tiverem `flex-grow` definido em 1, o espaço remanescente no container será distribuído de forma igual entre todos. Se um dos itens tem o valor de 2, vai ocupar o dobro de espaço no container com relação aos outros (ou pelo menos vai tentar fazer isso).

```
.flex-item {
  flex-grow: <numero>; /* o valor default (padrão) é 0 */
}
```

Valores negativos não são aceitos pela propriedade.

#### 3.6.4.3 *flex-shrink*

Define a habilidade de um flex item de encolher, caso necessário.

```
.flex-item {
  flex-shrink: <número>; /* o valor padrão é 0 */
}
```

Valores negativos não são aceitos pela propriedade.

#### 3.6.4.4 flex-basis

Define o tamanho padrão para um elemento antes que o espaço remanescente do container seja distribuído. Pode ser um comprimento (por exemplo, 20%, 5rem, etc) ou uma palavra-chave. A palavra-chave `auto` significa "observe minhas propriedades de altura ou largura" (o que era feito pela palavra-chave `main-size`, que foi depreciada). A palavra-chave `content` significa "estabeleça o tamanho com base no conteúdo interno do item" - essa palavra-chave ainda não tem muito suporte, então não é fácil de ser testada, assim como suas relacionadas: `max-content`, `min-content` e `fit-content`.

```
.flex-item {
  flex-basis: flex-basis: | auto; /* o valor padrão é auto */
}
```

Com o valor de 0, o espaço extra ao redor do conteúdo não é considerado. Com o valor de `auto`, o espaço extra é distribuído com base no valor de `flex-grow` do item.

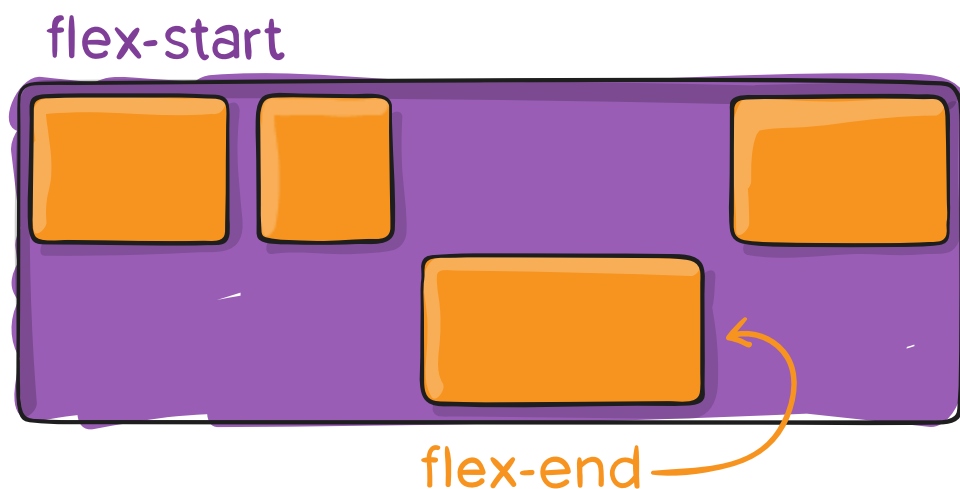
#### 3.6.4.5 flex

Esta é a propriedade *shorthand* para `flex-grow`, `flex-shrink` e `flex-basis`, combinadas. O segundo e terceiro parâmetros (`flex-shrink` e `flex-basis`) são opcionais. O padrão é 0 1 auto, mas se você definir com apenas um número, é equivalente a 0 1.

```
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

**É recomendado que você utilize esta propriedade *shorthand*** ao invés de definir cada uma das propriedades em separado. O *shorthand* define os outros valores de forma inteligente.

### 3.6.4.6 align-self



Permite que o alinhamento padrão (ou o que estiver definido por `align-items`) seja sobrescrito para itens individuais.

Por favor veja a explicação da propriedade `align-items` para entender quais são os possíveis valores.

```
.item {  
  align-self: auto | flex-start | flex-end | center | baseline  
  | stretch;  
}
```

Dica: Você pode aprender mais e praticar os conceitos aprendidos neste capítulo neste site: [Flexbox Froggy](#)

# Programação

## Introdução ao Javascript

Quando você está pesquisando sobre o mercado de programação é muito comum se deparar com o nome JavaScript. E não é à toa, pois essa linguagem está presente em toda a Web e você provavelmente lida com ela diariamente enquanto usa seu navegador (e até mesmo fora dele!).

O JavaScript, ou JS, como muitos chamam carinhosamente, é uma linguagem de programação interpretada de alto nível que, segundo a Pesquisa de Desenvolvedores do Stack Overflow de 2022, é a mais popular no mundo. Isso se deve principalmente ao fato de que o JavaScript é a linguagem padrão que os navegadores interpretam e que com HTML (“HyperText Markup Language” ou linguagem de Marcação de HiperTexto, utilizada nos navegadores) e CSS (“Cascading Style Sheets”, folhas de estilo em cascata, em Português) formam a base de toda a Web.

### 4.1 História do JavaScript

Agora que sabemos o que é o JavaScript, que tal entendermos mais sobre a sua história?

A década de 90 foi marcada por uma **grande disputa no mercado dos navegadores**. Nesse momento da história, a *Netscape*, responsável pelo *Netscape Navigator*, buscava se destacar na “guerra dos navegadores”. Para isso, planejava construir algo que pudesse deixar a navegação mais dinâmica, o que para a época seria um enorme **diferencial**, pois para **carregar uma página simples levava muito tempo**.

Assim, em 1995, a *Netscape* contratou o desenvolvedor **Brendan Eich** para criar uma **linguagem de script** que trouxesse “vida” às páginas Web. Desse modo, foi criada uma linguagem que foi chamada de *LiveScript*.

Após uma parceria com a *Sun Microsystems*, empresa responsável pela criação da linguagem Java, o nome *LiveScript* foi **alterado** para **JavaScript**. Essa foi uma jogada de *marketing* feita com o objetivo de chamar atenção para o JavaScript através da fama que o Java já tinha conquistado naquele momento.

É sempre válido lembrar que a **única semelhança entre Java e JavaScript é o nome**, já que são linguagens **totalmente** diferentes, ok?

Em 1997, em associação ao *ECMA* (acrônimo para *European Computer Manufacturers Association*), foi criada uma **padronização do JavaScript**, chamada de *ECMAScript*, para garantir o crescimento da linguagem seguindo algumas normas.

E assim surgiu o que hoje conhecemos como JavaScript.

## 4.2 Bibliotecas JS, jogos e aplicações

Como vimos anteriormente, a comunidade do JavaScript é muito colaborativa, com isso, surgem as bibliotecas da linguagem.

De forma resumida, as bibliotecas são pedaços de código pronto que vão trazer alguma funcionalidade ou resolver algum problema.

A ideia de utilizar as bibliotecas, também chamadas de “libs”, é reutilizar códigos já existentes e não perder tempo tentando escrever algo que já foi escrito por outra pessoa. Então, funcionalidades complexas, como trabalhar com dados, animações e cálculos matemáticos, podem ser simplificadas de maneira rápida.

A linguagem JavaScript também está presente no desenvolvimento de jogos. Hoje, apenas com HTML, CSS e JavaScript já é possível criar jogos que rodam nos navegadores de internet. Embora também existam as Game Engines feitas em JavaScript, que podem ser utilizadas para desenvolver jogos multiplataformas, isto é, jogos que podem ser jogados em diferentes dispositivos e sistemas operacionais sem a necessidade de comprar uma versão diferente para cada plataforma.

As Game Engines são programas que facilitam o processo do desenvolvimento do jogo, então, ela irá fazer as partes mais complicadas da criação, como renderizar gráficos, detectar colisões, fazer animações, além de suporte para sons, inteligência artificial, gerenciamento de arquivos, entre outras.

Além disso, diversas aplicações hoje em dia possuem pelo menos uma parte feita em JavaScript, por exemplo:

Paypal;

LinkedIn;

Netflix;

Uber; e

GoDaddy.

Em todas essas aplicações o JavaScript foi utilizado em algum momento, seja no navegador ou no aplicativo. Como, por exemplo, a Netflix que implementou o JavaScript nas suas aplicações para diminuir o tempo de espera dos usuários.

### 4.3 Variáveis no JavaScript

No ambiente de programação utilizamos **variáveis**, que **são espaços na memória do computador que o programa em execução reserva**. Usamos esse espaço reservado para guardar informações, realizar operações aritméticas, dentre várias outras aplicações.

#### 4.3.1 Tipos de variáveis JavaScript

Quando trabalhamos com JavaScript podemos usar três tipos de variáveis:

var;

let; e

const.

É fundamental sabermos a diferença entre elas para aplicarmos corretamente o seu uso no dia a dia.

Para começar, precisamos entender que o JavaScript possui uma peculiaridade chamada hoisting, isso significa que quando usamos uma variável do tipo var, ela é içada (levantada) para o topo do seu escopo. Vamos ver um exemplo:

```
console.log(cor) // saída: undefined  
var cor = "amarelo"
```



No exemplo acima, estamos usando uma variável do tipo `var` chamada `cor`, antes mesmo de declararmos o seu valor. Levando em conta a particularidade do *hoisting*, quando utilizamos o tipo `var`, não há erros de compilação, mas a saída da nossa impressão é `undefined` (em português significa “indefinido”, ou seja, a variável não foi inicializada).

Esse tipo de situação com *hoisting*, pode não ser o ideal quando queremos tornar nosso **código mais limpo e coeso**.

Uma solução para evitar possíveis confusões é utilizar variáveis do tipo `let`, e o mesmo algoritmo terá um resultado diferente. Observe o seguinte exemplo:

```
console.log(cor) // saída: Uncaught ReferenceError: Cannot
access 'cor' before initialization
let cor = "amarelo"
```

Nesse caso, ao tentar usar a variável `cor` antes de declararmos o seu valor, recebemos um **erro de referência** nos informando que não podemos acessar a variável antes que ela seja declarada.

Beleza! Então com `let` nós garantimos que erros causados pelo *hoisting* não existam, contudo, ainda assim pode ocorrer de declararmos uma variável do tipo `let` e usarmos antes que ela seja inicializada (antes de darmos um valor a ela), e seu resultado será `undefined`.

Para esses casos, quando temos uma variável que deve ser inicializada na sua declaração e que sabemos que seu valor não irá mudar, usamos o **const**. Veja abaixo um exemplo:

```
const nome = "Maria"
console.log(nome)
```

#### 4.3.2 Tipos de variáveis JavaScript

Em JavaScript, toda variável é “**elevada/içada**” (*hoisting*) até o topo do seu contexto de execução. Esse mecanismo move as variáveis para o topo do seu escopo antes da execução do código.

No nosso exemplo acima, como a variável `mensagemDentroDolf` está dentro de uma *function*, a declaração da mesma é elevada (*hoisting*) para o topo do seu contexto, ou seja, para o topo da *function*.

É por esse mesmo motivo que “é possível usar uma variável antes dela ter sido declarada”: em tempo de execução a variável será elevada (*hoisting*) e tudo funcionará corretamente.

### 4.3.3 var

Considerando o conceito de *hoisting*, vamos fazer um pequeno teste usando uma variável declarada com `var` antes mesmo dela ter sido declarada:

```
void function() {  
    console.log(mensagem);  
}();  
var mensagem;
```

No caso da palavra-chave `var`, além da variável ser içada (*hoisting*) ela é automaticamente inicializada com o valor `undefined` (caso não seja atribuído nenhum outro valor).

Ok, mas qual é o impacto que temos quando fazemos esse tipo de uso?

Imagine que nosso código contenha muitas linhas e que sua complexidade não seja algo tão trivial de compreender.

Às vezes, queremos declarar variáveis que serão utilizadas apenas dentro de um pequeno trecho do nosso código. Ter que lidar com o escopo de função das variáveis declaradas com `var` (escopo abrangente) pode confundir a cabeça até de programadores mais experientes.

Sabendo das "complicações" que as variáveis declaradas com `var` podem causar, o que podemos fazer para evitá-las?

#### 4.3.4 let

Foi pensando em trazer o escopo de bloco (tão conhecido em outras linguagens) que o ECMAScript 6 se destinou a disponibilizar essa mesma flexibilidade (e uniformidade) para a linguagem.

Através da palavra-chave `let` podemos declarar variáveis com escopo de bloco. Vamos ver:

```
var exibeMensagem = function() {  
  if(true) {  
    var escopoFuncao = 'Banana';  
    let escopoBloco = 'Maça';  
  
    console.log(escopoBloco); // Banana  
  }  
  console.log(escopoFuncao); // Maça  
  console.log(escopoBloco);  
}
```

Qual será a saída do código acima?

```
exibeMensagem(); // Imprime 'Banana', 'Maça' e dá um erro
```

Veja que quando tentamos acessar uma variável que foi declarada através da palavra-chave `let` fora do seu escopo, o erro *Uncaught ReferenceError: escopoBloco is not defined* foi apresentado.

Portanto, podemos usar tranquilamente o `let`, pois o escopo de bloco estará garantido.

#### 4.3.5 const

Embora o `let` garanta o escopo, ainda assim, existe a possibilidade de declararmos uma variável com `let` e ela ser *undefined*. Por exemplo:

```
void function() {  
  let mensagem;  
  console.log(mensagem); // Imprime undefined  
}();
```

Supondo que temos uma variável que queremos garantir sua inicialização com um determinado valor, como podemos fazer isso no JavaScript sem causar uma inicialização *default* com *undefined*?

Para termos esse tipo de comportamento em uma variável no JavaScript, podemos declarar constantes por meio da palavra-chave `const`. Vamos dar uma olhada no exemplo:

```
void function() {  
    const mensagem = 'Olá';  
    console.log(mensagem); // Olá  
    mensagem = 'Oi';  
}();
```

O código acima gera um *Uncaught TypeError: Assignment to constant variable*, pois o comportamento fundamental de uma constante é que uma vez atribuído um valor a ela, este não pode ser alterado.

Assim como as variáveis declaradas com a palavra-chave `let`, constantes também tem escopo de bloco.

Além disso, constantes devem ser inicializadas obrigatoriamente no momento de sua declaração. Vejamos alguns exemplos:

```
// constante válida  
const idade = 18;  
  
// constante inválida: onde está a inicialização?  
const pi;
```

No código acima temos o exemplo de uma constante `idade` sendo declarada e inicializada na mesma linha (constante válida) e um outro exemplo onde o valor não é atribuído na declaração de `pi` (constante inválida) ocasionando o erro *Uncaught SyntaxError: Missing initializer in const declaration*.

É importante utilizar `const` para declarar nossas variáveis, porque assim conseguimos um comportamento mais previsível, já que o valor que elas recebem não podem ser alterados.

## 4.4 Tipos de dados no JavaScript

É comum ouvirmos falar que o JavaScript é uma linguagem de **tipagem fraca**, mas você sabe o que isso significa?

No JavaScript a **tipagem é dinâmica**, por isso não é necessário declarar o tipo de uma **variável** — basta atribuir seu valor e a linguagem assume se é uma sequência de caracteres, números, indefinido ou outras. Para entender melhor esse conceito, observe o seguinte exemplo de código:

```
let variavelDeTexto = 'texto qualquer';
let variavelNumerica = 123;
let variavelIndefinida;
let variavelBooleana = true;

console.log(typeof variavelDeTexto) //saída: string
console.log(typeof variavelNumerica) //saída: number
console.log(typeof variavelIndefinida) //saída: undefined
console.log(typeof variavelBooleana) //saída: boolean
```

No código acima, temos a criação de variáveis, do tipo *String*, numérica, booleana e uma indefinida. Depois disso, utilizamos uma função do próprio JavaScript, que é o *typeof*, que irá nos **retornar o tipo da variável que foi passada como parâmetro**, assim, as saídas serão: *String*, *number*, *undefined* e *boolean*. Podemos perceber que esses tipos foram reconhecidos de maneira dinâmica pelo JavaScript.

## 4.5 Hello world em JavaScript

Quando estamos iniciando em uma nova linguagem, é uma **tradição** começarmos imprimindo na tela o famoso “**Olá, mundo!**”, e para darmos esse passo importante em JavaScript, é imprescindível que você prepare seu ambiente, realizando o download e instalando um editor de código como o Visual Studio Code, ou outro de sua preferência.

### Passo 01

Preparado o ambiente, vamos lá! O JavaScript funciona em conjunto com HTML (que significa “*HyperText Markup Language*” ou linguagem de marcação de texto, utilizada nos navegadores), dessa maneira, precisamos criar um arquivo com extensão `.html`

(index.html, por exemplo), e dentro desse arquivo precisamos usar o corpo de uma página HTML, da seguinte maneira:

```
<!DOCTYPE html>
<html>

</html>
```

## Passo 02

Agora, perceba que o HTML é uma linguagem de marcação, ou seja, utiliza marcações (**etiquetas** ou *tags*) **para informar ao navegador o que é cada coisa dentro dele**. E para o código JavaScript, também temos uma *tag*. Vamos usar a *tag* script, dentro da *tag* html, para guardar nosso código, veja abaixo:

```
<!DOCTYPE html>
<html>
    <script>
    </script>
</html>
```

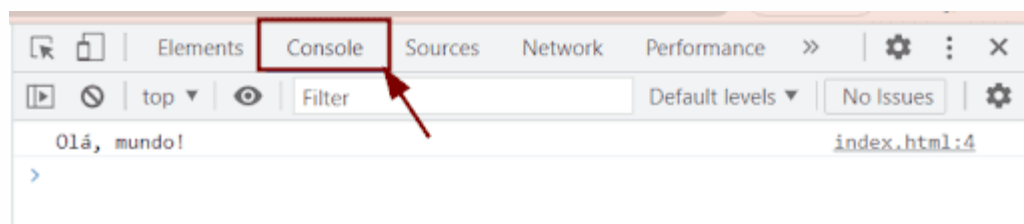
## Passo 03

E, claro, agora precisamos imprimir o nosso “Olá, mundo!” e, para isso, usaremos o `console.log()`, que vai **imprimir na aba de console do navegador a mensagem que quisermos**. O código final fica da seguinte maneira:

```
<!DOCTYPE html>
<html>
    <script>
        console.log("Olá, mundo!")
    </script>
</html>
```

## Passo 04

Depois de concluir e salvar seu código, abra a pasta do arquivo e abra o arquivo .html no seu navegador, espere abrir e clique com o botão direito em qualquer área da tela, escolha a opção **inspecionar**. Deve abrir a seguinte aba:



Na opção "Console", você pode observar a frase “Olá, mundo!” impressa, é **dessa maneira que visualizamos as impressões** que vamos utilizar em nosso **código** JavaScript.

## 4.6 Funções JavaScript

### Passo 01

Em JavaScript, assim como em outras linguagens de programação, nós trabalhamos com **funções**, que nada mais são do que um trecho de código que pode ser invocado (chamado) em outro momento no seu algoritmo. Para criar uma função em JavaScript, é bastante simples, basta escrever da seguinte forma:

```
function nomeDaFuncao() {  
  //o que a função faz (corpo da função)  
}
```

### Passo 02

Na prática, imagine que queremos criar uma função para *somar*  $6 + 4$ , podemos dizer que o nome da função é *soma* e no **corpo da função**, ou seja, aquilo que ela vai fazer, podemos usar um `console.log(6 + 4)` para escrever o resultado da soma no *console*. Vamos ver abaixo como o código ficaria:

```
<!DOCTYPE html>  
<html>  
  <script>  
    function soma() {  
      console.log(4 + 6)  
    }  
  </script>  
</html>
```

### Passo 03

Mas note que se você testar apenas isso, nada vai aparecer no console do seu navegador. *E por quê?* Como disse anteriormente, as funções precisam ser **invocadas**, chamadas em algum momento do seu código. Portanto, você deve chamar a função pelo seu nome `soma()`, veja abaixo:

```
<!DOCTYPE html>
<html>
  <script>
    function soma() {
      console.log(4 + 6)
    }
    soma()
  </script>
</html>
```

E agora sim, você deve conseguir ver o resultado 10 no seu *console*.

#### Passo 04

Outro ponto interessante é que funções também podem receber **parâmetros**, **argumentos** (dados), que são passados para a função no momento da sua chamada. Então, agora imagine que queremos uma função que some qualquer número, podemos usar um argumento `x` e outro `y` que serão somados dentro do `console.log`, da seguinte maneira:

```
<!DOCTYPE html>
<html>
  <script>
    function soma(x, y) {
      console.log(x + y)
    }
  </script>
</html>
```

Assim, podemos realizar somas distintas apenas *passando* diferentes valores na chamada da função, veja como ficaria o código:

```
<!DOCTYPE html>
<html>
  <script>
    function soma(x, y) {
      console.log(x + y)
    }
  </script>
</html>
```



```

    }
    soma (4, 6)
    soma (3, 5)
</script>
</html>

```

Nesse algoritmo a saída seria respectivamente 10 e 8.

## Passo 05

Repare que as funções que aprendemos acima podem ser chamadas dentro da própria *tag* script dentro do HTML, ou dentro de um arquivo JavaScript *linkado* ao HTML, como aprendemos anteriormente. Porém, indo um pouco além, **que tal chamarmos a função quando apertamos um botão na página HTML?** Para isso, podemos usar uma *tag* HTML chamada *button*, e dentro dela usarmos uma propriedade chamada *onclick()* (que significa “ao clicar”) que recebe uma função, e será invocada quando clicarmos no botão. Veja abaixo como ficaria:

```

<!DOCTYPE html>
<html>
  <button onclick="soma(4, 6)">Me aperte</button>
  <script>
    function soma(x, y) {
      console.log(x + y)
    }
  </script>
</html>

```

Agora, ao clicarmos no botão, a função `soma(4, 6)` é chamada e a soma de  $4 + 6$  aparece no *console* do nosso navegador.

## 4.7 JavaScript: Estrutura condicional

As estruturas condicionais permitem que um programa execute diferentes comandos de acordo com as condições estabelecidas. Elas estão presentes em diversas linguagens de programação e todo profissional da área precisa saber como utilizá-las.

### 4.7.1 if

A estrutura condicional `if` permite ao JavaScript executar um trecho de código somente se uma determinada condição for verdadeira.

Ela pode ser escrita das seguintes formas:

- Quando há mais de uma linha de código a ser executado devemos usar `{ e }` para delimitar esse bloco de código:

```
if ( condicao ) {  
  // códigos que serão executados  
  // códigos que serão executados  
  // códigos que serão executados  
}
```

Quando há só uma linha de código para ser executado não precisamos usar `{ e }`, fica opcional:

```
if ( condicao)  
  // código que vai ser executado;
```

ou

```
if ( condicao) // código que vai ser executado;
```

O código dentro da estrutura `if` só será executado se a **condição** for verdadeira. Por exemplo:

```
var preco = 101;  
if ( preco > 100 )  
  console.log("Desconto liberado");  
  
// vai imprimir "Desconto liberado"
```

Nesse exemplo, a instrução `console.log("Desconto liberado")` será executada somente se o valor da variável `preco` for maior que 100.

Repare que, dessa forma, se a condição for falsa a instrução `console.log("Desconto liberado")` será ignorada.

#### 4.7.2 else

Existem casos em que precisamos executar um código caso uma condição seja verdadeira ou um outro, caso ela seja falsa. Para isso utilizamos a palavra-chave `else`.

Alguns exemplos:

```
if ( condicao )
// se condição for verdadeira executa
else
// se condição for falsa executa
```

Para um bloco de códigos:

```
if ( condicao ) {
// códigos que serão
// executados caso a
// condição seja verdadeira
} else
// códigos que serão
// executados caso a
// condição seja falsa
}
```

Veja um exemplo, onde verificamos a variável `preco` e caso o seu valor seja maior que 100 a mensagem `console.log("Desconto liberado")` será impressa. Caso contrário será impresso `console.log("Nenhum desconto foi liberado")`.

```
var preco = 20;
if ( preco > 100 )
    console.log("Desconto liberado");
else
    console.log("Nenhum desconto foi liberado");
```

```
// vai imprimir "Nenhum desconto foi liberado"
```

#### 4.7.3 else if

Como vimos, utilizar if e else permite ao JavaScript executar um código dentre duas opções. Porém, há casos em que devemos testar uma nova condição antes de executar o trecho de código alternativo. Uma forma de escrever essa verificação é utilizando else if.

```
var preco = 70;
if ( preco > 100 ) {
  console.log("Desconto de 10% liberado");
} else if ( preco > 50 ) {
  console.log("Desconto de 5% liberado");
} else {
  console.log("Nenhum desconto foi liberado");

  // vai imprimir "Desconto de 5% liberado"
}
```

No exemplo acima temos três alternativas de códigos para serem executados:

1. No primeiro caso, a instrução `console.log("Desconto de 10% liberado")` será executada se o valor da variável `preco` for maior que 100.
2. Se não for esse o caso, uma nova verificação será feita e se o valor da variável `preco` for maior que 50 a instrução `console.log("Desconto de 5% liberado")` será executada.
3. Caso nenhuma dessas situações seja verdadeira a instrução `console.log("Nenhum desconto foi liberado")`.

#### 4.7.4 if ternário

A estrutura if possui uma forma mais compacta, chamada de if ternário. Sua sintaxe é a seguinte:

```
condicao ? código1 : código2
```

Veja um exemplo:

```
var nome;  
console.log( nome ? 'Olá ' + nome : 'Digite um nome' );  
// vai imprimir 'Digite um nome'
```

Esse código tem o mesmo efeito deste:

```
var nome;  
if ( nome ) {  
  console.log("Olá " + nome);  
} else {  
  console.log("Digite um nome");  
}  
  
// vai imprimir 'Digite um nome'
```

Contudo, a primeira forma, ternária, utiliza menos linhas para realizar a mesma verificação.

#### 4.7.5 Operador &&

O operador && possui um comportamento chamado curto-circuito que torna possível executar um código de forma similar ao if.

Veja um exemplo:

```
var valor = 650;  
if ( valor > 100 ) console.log("Pode parcelar a compra sem juros");  
// vai imprimir "Pode parcelar a compra sem juros"
```

Esse código pode ser escrito utilizando && da seguinte forma:

```
var valor = 650;  
( valor > 100 ) && console.log("Pode parcelar a compra sem juros") ;  
// vai imprimir "Pode parcelar a compra sem juros"
```

Veja um outro exemplo, onde atribuímos um valor a uma variável, baseado na condição `preco > 100`.

```
var preco = 20;  
var permiteParcelar = preco > 100 && true;  
console.log(permiteParcelar);  
// vai imprimir false
```

Estas são as variações da estrutura `if` que o JavaScript nos oferece. O uso de cada uma delas vai depender de como você vai codificar a sua aplicação.

## 4.8 JavaScript: Estruturas de repetição

As estruturas de repetição estão muito presentes na vida profissional de uma pessoa desenvolvedora de software. Isso porque sempre precisamos trabalhar com vários tipos de listagens e a partir disso, realizamos tarefas de ordenação, filtragem e até modificação dos valores.

Mesmo sendo uma ferramenta tão corriqueira para o trabalho de programação, as pessoas que desenvolvem precisam ficar atentas a utilização dessas repetições, já que o mau uso pode acabar afetando a performance da aplicação.

### 4.8.1 O que é estrutura de repetição?

Uma estrutura de repetição é uma lógica que repete as ações de um mesmo bloco de código por um número de vezes determinado a partir do tipo de estrutura de repetição.

É fundamental que a pessoa iniciante em programação entenda os conceitos das estruturas de repetição, já que elas são responsáveis pelos diversos processos envolvendo dados.

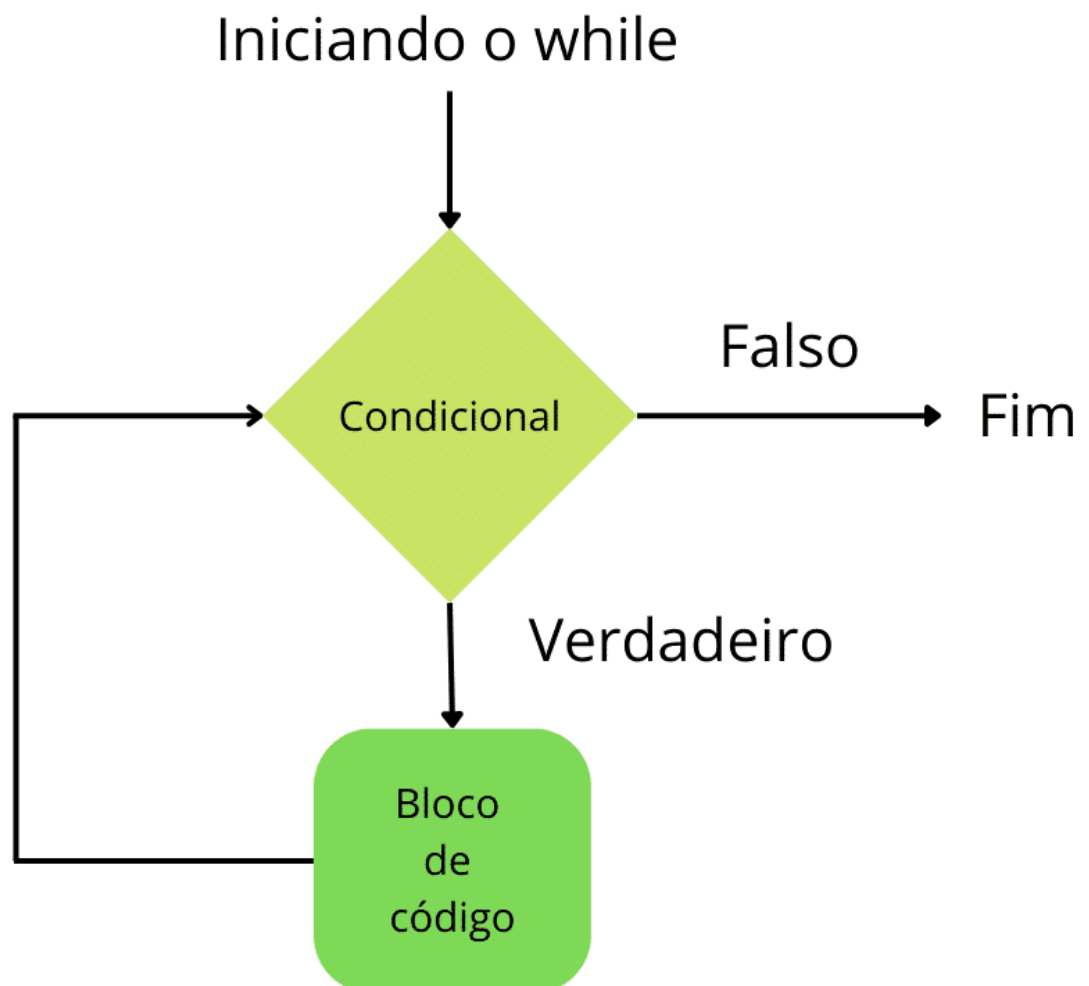
### 4.8.2 Quando devemos usar as estruturas de repetição?

No geral, as estruturas de repetição nos auxiliam na hora de trabalhar com listagens. Por exemplo: se você tem uma lista onde cada item é referente a uma pessoa usuária de um sistema interno, com as estruturas de repetição é possível realizar tanto uma operação

que encontre um ou mais itens que possuam um dado semelhante, como também é possível ordenar essa lista pelos nomes de cada pessoa usuária.

Em alguns outros casos também é possível criar lógicas dentro da estrutura de repetição para que a mesma seja terminada antes do previsto. Essa prática é muito comum em sistemas de comando de linha.

#### 4.8.3 While



While é uma **estrutura de repetição** que utiliza uma **estrutura condicional** para **verificar se a repetição deve ou não continuar**.

A partir das operações de dentro do bloco de código, a condicional precisa ter o seu valor alterado em algum momento, se não pode ocorrer um grave problema de loop infinito, **onde as tarefas do código são realizadas diversas vezes sem parar e com a consequência de travar o programa**.

### Sintaxe da estrutura de repetição While

Para descrevermos a sua sintaxe, utilizaremos a ideia de pseudocódigo:

```
condicional <- VERDADEIRO;
contador <- 0
ENQUANTO condicional == VERDADEIRO
    SE contador == 5 ENTÃO
        condicional = FALSO
    FIM SE;
    contador <- contador + 1
FIM ENQUANTO;
```

O funcionamento do While é bem simples. Contamos apenas com um código que verifica a condicional e o bloco de código conforme o exemplo abaixo.

#### Exemplo com código

Aqui, vamos escrever o while na linguagem Javascript:

```
let condicional = true
let contador = 0

console.log("Iniciando while")

while (condicional == true) {
    if (contador == 5) {
        condicional = false
    }
    contador = contador + 1
    console.log("Contador: ", contador)
}
```



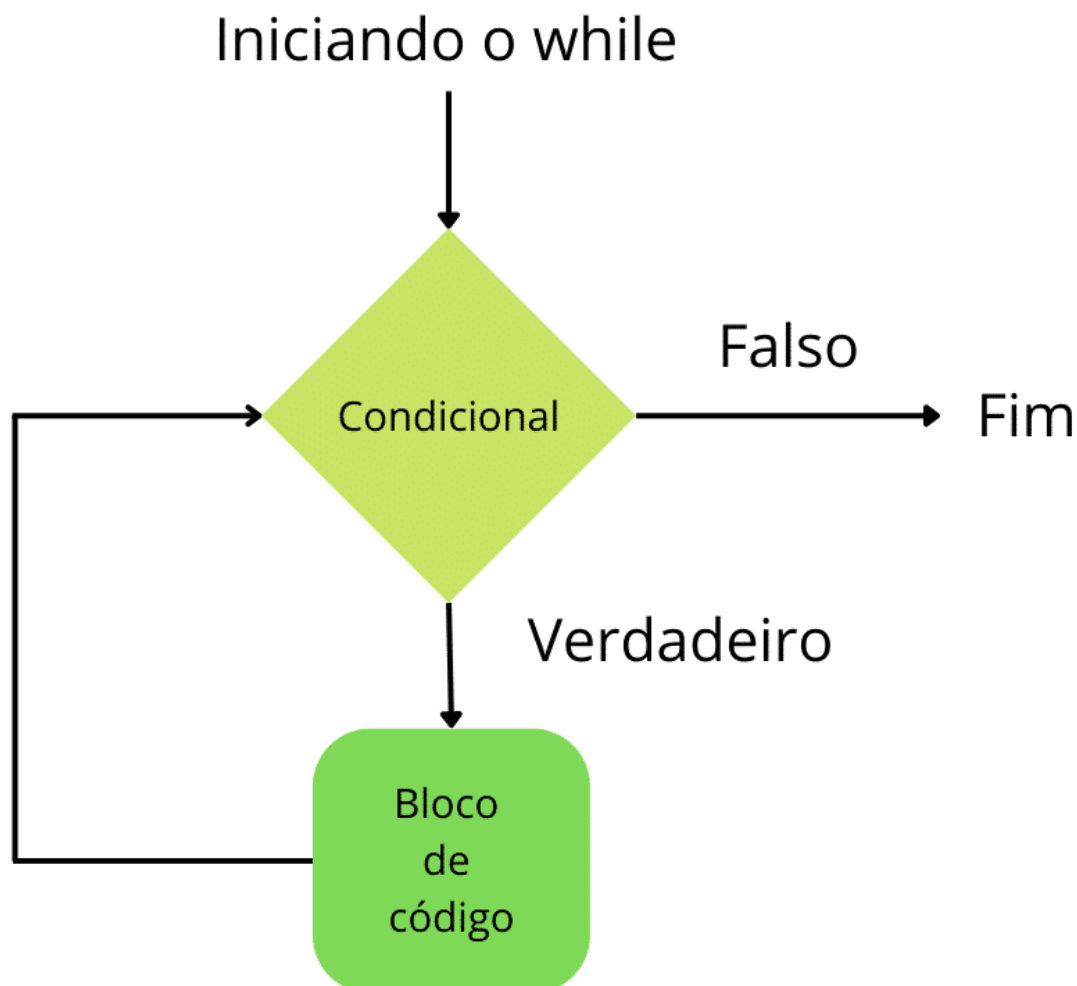
```
console.log("Terminando while")
```

Veja que a estrutura possui uma sintaxe muito simples, também. Adicionamos alguns comentários para ficar mais fácil a visualização do programa no momento de sua execução.

### **Quando usar a estrutura de repetição While?**

Utilize o while sempre que você precisar realizar operações com um tamanho indefinido de passos. **Lembre-se de se certificar que o código em repetição contém alguma lógica que modifique a condicional!**

#### 4.8.4 Do While



Do While, diferentemente do While convencional, começa executando o bloco de código pelo menos uma vez. A partir disso, é checado se a repetição deve ou não continuar.

#### Sintaxe da estrutura de repetição Do While

Sua sintaxe é descrita da seguinte maneira:

```
condicional <- VERDADEIRO;
contador <- 0
REPITA
  SE contador == 5 ENTÃO
    condicional = FALSO
  FIM SE;
  contador <- contador + 1
ENQUANTO condicional == VERDADEIRO;
```

### Exemplo com código

Em Javascript, temos o Do While assim:

```
let condicional = true
let contador = 0

console.log("Iniciando do while")

do {
  if (contador == 5) {
    condicional = false
  }
  contador = contador + 1
  console.log("Contador: ", contador)
} while (condicional == true)

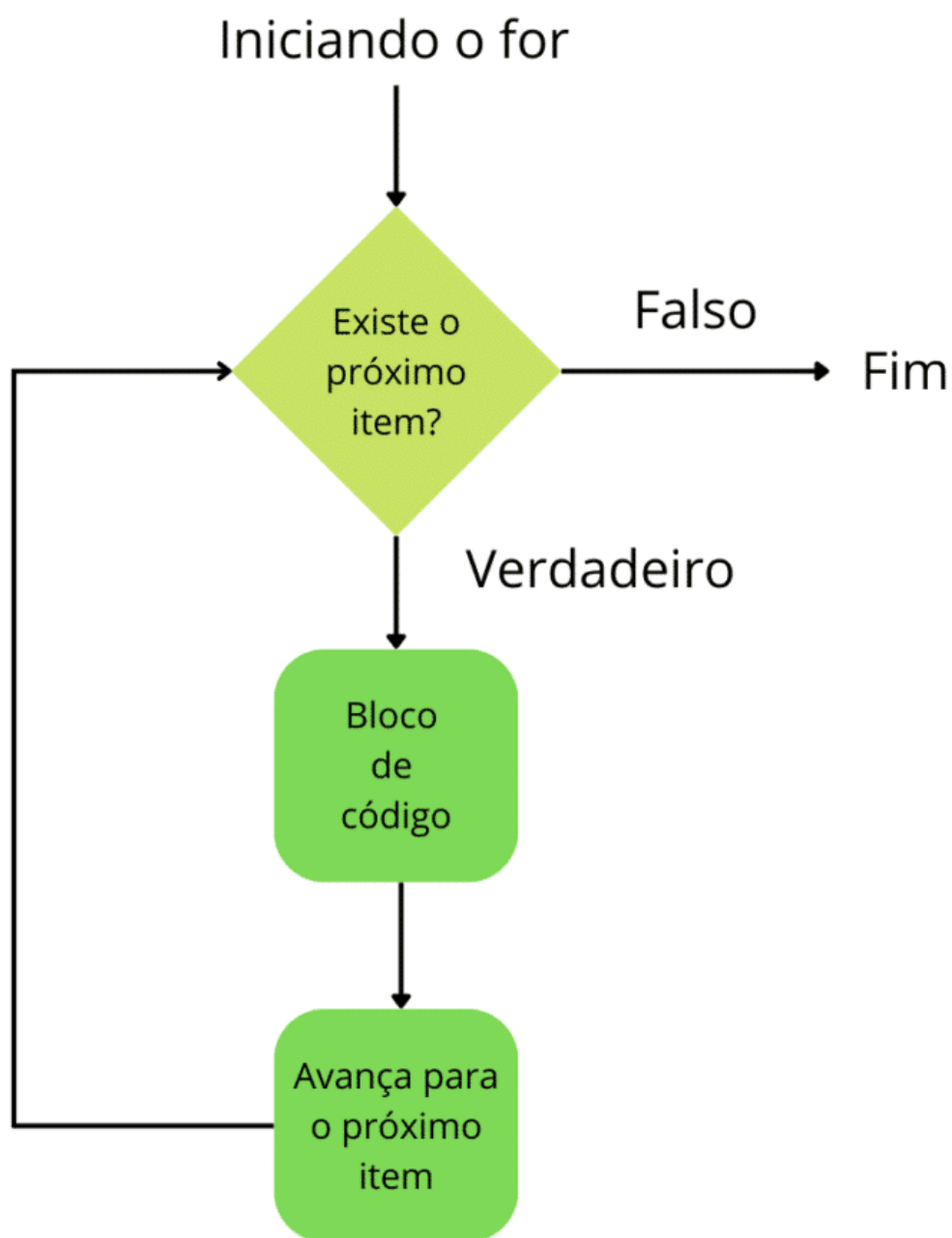
console.log("Terminando do while")
```

Neste exemplo, temos um resultado muito parecido com a estrutura While. Mas atenção, a verificação da condicional só é realizada ao final de cada execução do bloco de código, ao invés do começo.

### Quando usar a estrutura de repetição Do While?

A repetição Do While é utilizada em sistemas onde é preciso executar o bloco de código pelo menos uma vez, necessariamente.

#### 4.8.5 For



Na estrutura de repetição For temos três principais pilares a definir:

1. uma variável contadora;
2. uma condicional baseada na variável;
3. um pedaço de código que é executado ao final de cada repetição.

Além disso, por ser uma estrutura com uma sintaxe mais rígida, dificilmente teremos problema de repetição infinita, algo que ocorre em outros tipos de estrutura.

### Sintaxe da estrutura de repetição For

Utilizamos a palavra **PARA** como forma de repetição For, em [pseudocódigo](#).

```
PARA contador <- 0 ATE 5 FAÇA  
  <Bloco de instruções>  
FIM-PARA
```

### Exemplo com código

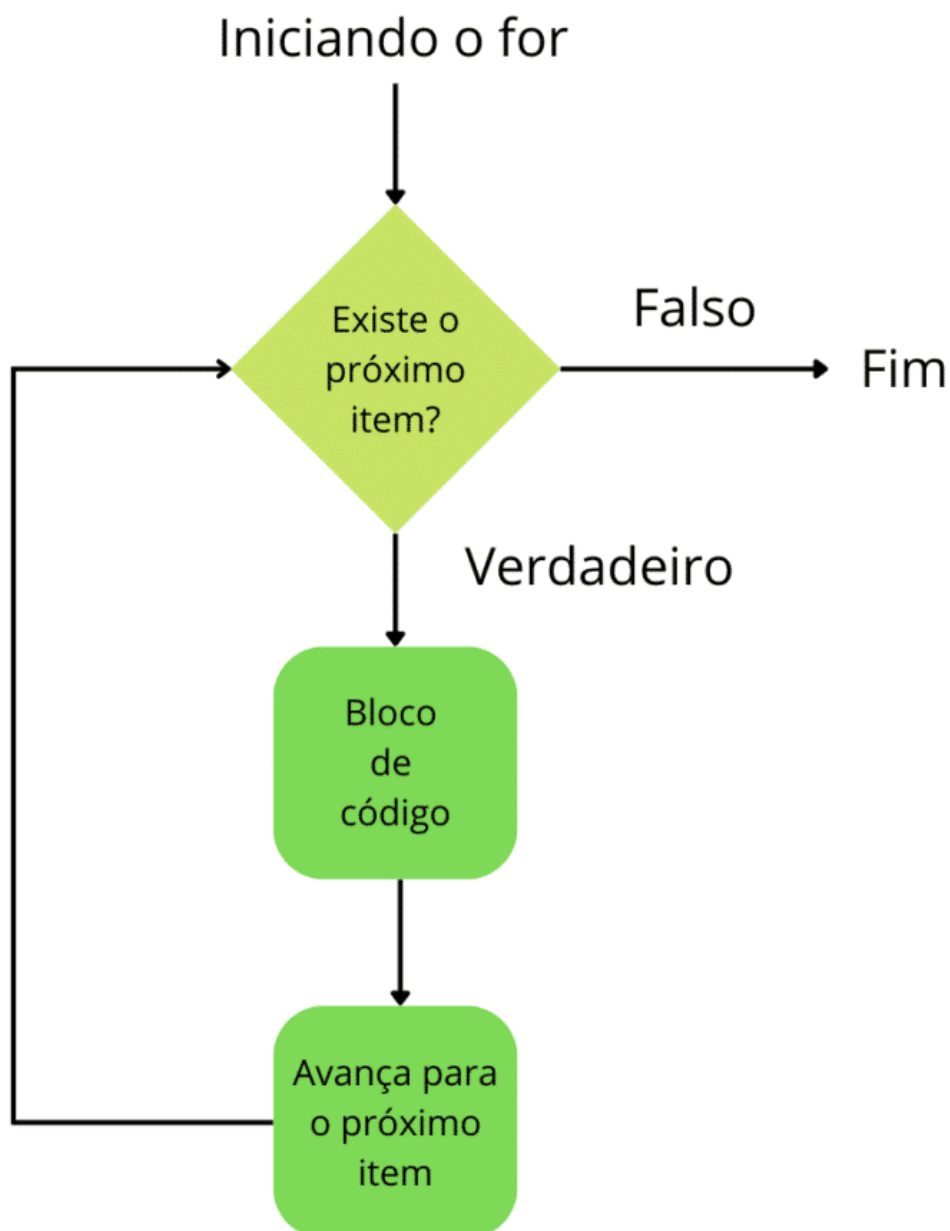
Dentro da estrutura For, definimos a variável contadora, qual a estrutura condicional dela e a operação que é realizada para modificá-la:

```
console.log("Iniciando for")  
  
for (let contador = 1; contador <= 5; contador = contador + 1) {  
  console.log("Contador: ", contador)  
}  
  
console.log("Terminando for")
```

### Quando usar a estrutura de repetição For?

Você deve utilizar a estrutura For quando souber quantas repetições são necessárias para completar a sua lógica. De certo modo, são mais usados em listagens, já que, a partir do tamanho da lista, podemos percorrer cada item.

#### 4.8.6 Foreach



O Foreach é uma estrutura utilizada para vasculhar uma listagem com um número indefinido de itens do começo ao fim. Quando chegamos ao final da lista, a repetição é terminada.

### Sintaxe da estrutura de repetição Foreach

Uma sintaxe que pode ser considerada válida em pseudocódigo é a seguinte:

```
lista <- [1, 2, 3, 4, 5]

PARA CADA valor DA lista
  <Bloco de instruções>
FIM-PARA-CADA
```

### Exemplo com código

Em Javascript, temos o método padrão **forEach** para toda variável que é considerada uma array.

```
const items = [1, 2, 3, 4, 5]

console.log("Iniciando foreach")

items.forEach((valor) => {
  console.log(valor)
})

console.log("Terminando foreach")
```

# Banco de dados



## Banco de dados Relacional (SQL)

O banco de dados é a organização e armazenagem de informações sobre um domínio específico. De forma mais simples, é o agrupamento de dados que tratam do mesmo assunto, e que precisam ser armazenados para segurança ou conferência futura.



É comum que empresas tenham diversas informações que precisam ser organizadas e disponibilizadas dentro do negócio para que sejam consultadas posteriormente pela equipe e pela gerência.

Por isso, é interessante ter um sistema de gerenciamento de banco de dados, SGBD, para conseguir manipular as informações e tornar a rotina da empresa muito mais simples.

Hoje, existem diversos tipos de SGBDs, e cada um é adequado para uma necessidade dos clientes. São os mais comuns: Oracle, DB2, MySQL, SQL Server, PostgreSQL e outros.

Se a sua empresa tem um site em WordPress ou em alguma outra plataforma, o banco de dados é fundamental para manter o bom funcionamento e a praticidade no dia a dia do negócio.

A Structured Query Language (em português – Linguagem de Consulta Estruturada) ou chamado pela abreviação SQL, é conhecida comercialmente como uma “linguagem de consulta” padrão utilizada para manipular bases de dados relacionais. Por ser uma linguagem padrão, é utilizada em inúmeros sistemas, como: MySQL; SQL Server; Oracle; Sybase; DB2; PostgreSQL; Access etc.

Cada sistema pode usar um “dialetto” do SQL, como T-SQL (utilizado por SQL SERVER), PL/SQL (Oracle), JET SQL (Access) entre outros.

No entanto, o SQL possui muitos outros recursos além de consulta ao banco de dados, como meios para a definição da estrutura de dados, para modificação de dados no banco de dados e para a especificação de restrições de segurança.

Esses recursos em SQL são divididos em cinco partes, sendo:

1. Data Definition Language (Linguagem de Definição de Dados), conhecido pela abreviação **DDL**;
2. Data Manipulation Language (Linguagem de Manipulação de Dados), conhecido pela abreviação **DML**;
3. Data Query Language (Linguagem de Consulta de Dados), conhecido pela abreviação **DQL**;
4. Data Control Language (Linguagem de Controle de Dados), conhecido pela abreviação **DCL**;
5. Data Transaction Language (Linguagem de Transação de Dados), conhecida pela abreviação **DTL**.

## 5.1 Quais são os principais tipos de banco de dados?

Existem várias opções de bancos de dados disponíveis no mercado. Mas, antes de trabalharmos cada uma delas, é preciso entender a diferença entre os bancos de dados relacionais e os não relacionais.

Os bancos de dados relacionais são criados no paradigma da orientação a conjuntos. Dessa forma, os dados que ali estão disponíveis serão armazenados em tabelas. Cada tabela terá atributos e linhas ou registros responsáveis por organizar essas informações.

São comumente utilizados para dados tabulares, que possuem sua inserção muito mais simples e permite, também, a recuperação de forma mais prática no dia a dia. A linguagem utilizada nesse formato é de SQL, Structured Query Language. Portanto, se você quer utilizar um banco de dados relacionais, é preciso se atentar a esse detalhe.

Eles são ideais para CRMs, ERPs ou até mesmo gerenciamento financeiro das empresas.

Já os bancos de dados não relacionais são responsáveis por atender a demandas que os bancos relacionais não conseguem suprir. Um exemplo de demandas são aqueles dados mistos, onde se misturam tabelas, imagens e mapas, por exemplo, que não poderão ser tabulados em colunas e linhas de tabela.

Suas soluções são baseadas em armazenamento na nuvem. A linguagem utilizada nesse formato é NoSQL, Not Only SQL.

### 5.1.1 Oracle

O Oracle Database é o sistema de gestão de banco de dados mais utilizado no mundo. Trabalha com a linguagem SQL, e garante a segurança e diversos recursos para seus clientes e usuários.

Uma das vantagens desse modelo é a facilidade para ser instalado nas mais diversas plataformas, sendo compatível com BIM AIX, IBM VMS, Windows, Linux, Unix e HP/UX.

No entanto, é interessante investir em um bom hardware para que o desempenho não seja prejudicado.

Outra vantagem do Oracle é a sua documentação. Ela é extremamente detalhada e, por isso, os desenvolvedores terão muito mais conhecimento dos recursos disponíveis na plataforma.

Além disso, o Oracle oferece recursos de segurança e performance que garantem a qualidade do trabalho e a tranquilidade dos usuários, se tornando a melhor alternativa para grandes empresas ou negócios que possuem requisitos mais complexos.

### **5.1.2 SQL Server**

O SQL Server, criado pela Microsoft, é muito conhecido e utilizado no mercado. A linguagem usada nessa ferramenta é o T-SQL, e oferece recursos avançados e diferenciados para facilitar a atualização de dados e o armazenamento das informações de forma segura e confiável.

O SQL Server atua com sistemas integrados de criptografia, permitindo que a visualização ou alteração das informações sejam feitas apenas pelas pessoas responsáveis, o que garante ainda mais segurança e tranquilidade para os usuários e empresários.

É uma alternativa comumente utilizada em lojas online, instituições governamentais, bancos e indústrias dos mais diversos portes.

### **5.1.3 MySQL**

O MySQL é um banco de dados relacional que pertence à Oracle. Uma das características mais marcantes desse modelo é o fato de se tratar de um Open Source. Utiliza a linguagem SQL e funciona com as licenças de software comercial e livre.

O MySQL se destaca pelo seu fácil uso e uma estrutura de segurança e confiabilidade que permitiu que empresas e aplicativos baseados na internet utilizassem seus recursos. Dentre os principais usuários estão o Google, Facebook, Youtube, Twitter e NASA.

#### **5.1.4 PostgreSQL**

O PostgreSQL também é um gerenciador de banco de dados relacional Open Source, comumente utilizado para sistemas online, como Skype, Apple e o Metrô de São Paulo.

É considerada uma das alternativas mais avançadas do mercado, com recursos diferenciados e complexos, que permite que os usuários consigam ter maior facilidade de acessos e integridade transacional. Essa alternativa exige uma solução em hardware potente para não prejudicar o desenvolvimento. O PostgreSQL possui uma capacidade de suportar um grande fluxo de dados, garantindo a segurança e estabilidade, além de um alto desempenho por um valor ainda mais acessível.

#### **5.1.5 NoSQL**

O NoSQL é um sistema de banco de dados não relacional, conforme explicado anteriormente. Hoje, esse termo é comumente utilizado por pessoas que produzem conteúdos por dispositivos, redes sociais e outros tipos de funcionalidades web, que exigem a gestão de dados em diferentes formatos.

O uso de bancos de dados NoSQL permitirá que você tenha maior escalabilidade e maior economia no dia a dia, pois, ao contrário de alguns que já apresentamos ao longo deste artigo, não exige um hardware muito potente.

O NoSQL também possui sua manutenção muito mais simples e prática, o que pode permitir que a equipe seja reduzida e os gastos revisados, se tornando uma alternativa muito interessante para grandes empresas.

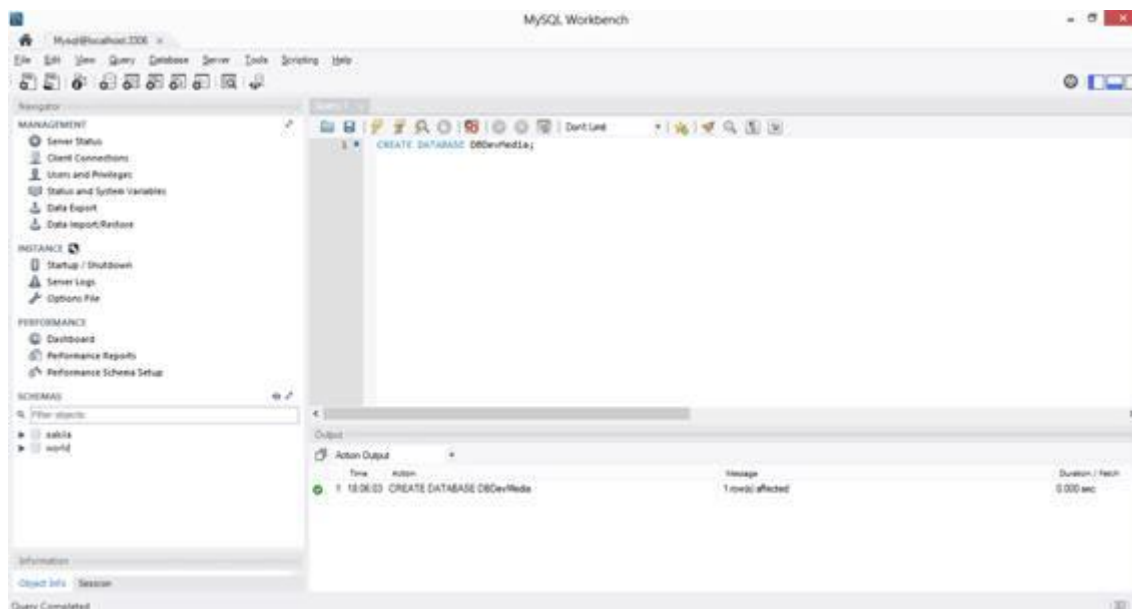
### **5.2 Linguagem de Definição de Dados (DDL)**

Ao criarmos nosso banco de dados com as tabelas explicitando seus tipos de dados a cada campo, sua(s) chave(s) primaria(s) e estrangeiras, índices, regras e etc., temos para isso a criação e alteração de estruturas que definem como os dados serão armazenados. Logo, quando falamos de comando do tipo DDL estamos falando de comandos do tipo CREATE, ALTER e DROP (criar, alterar e excluir, respectivamente).

Para criar o banco de dados DBDevMedia utilizaremos a sintaxe CREATE, conforme o código a seguir:

```
CREATE DATABASE DBDevMedia;
```

Ao executá-lo teremos o mesmo resultado da **Figura 8**.



Podemos complementar o nosso código com a sintaxe opcional IF NOT EXISTS, que **permite ao MySQL verificar se o nome escolhido esteja sendo utilizado no servidor**, evitando que retorne um erro com a possível existência de dois bancos com o mesmo nome em um mesmo **servidor MySQL**:

```
CREATE DATABASE IF NOT EXISTS DBDevMedia;
```

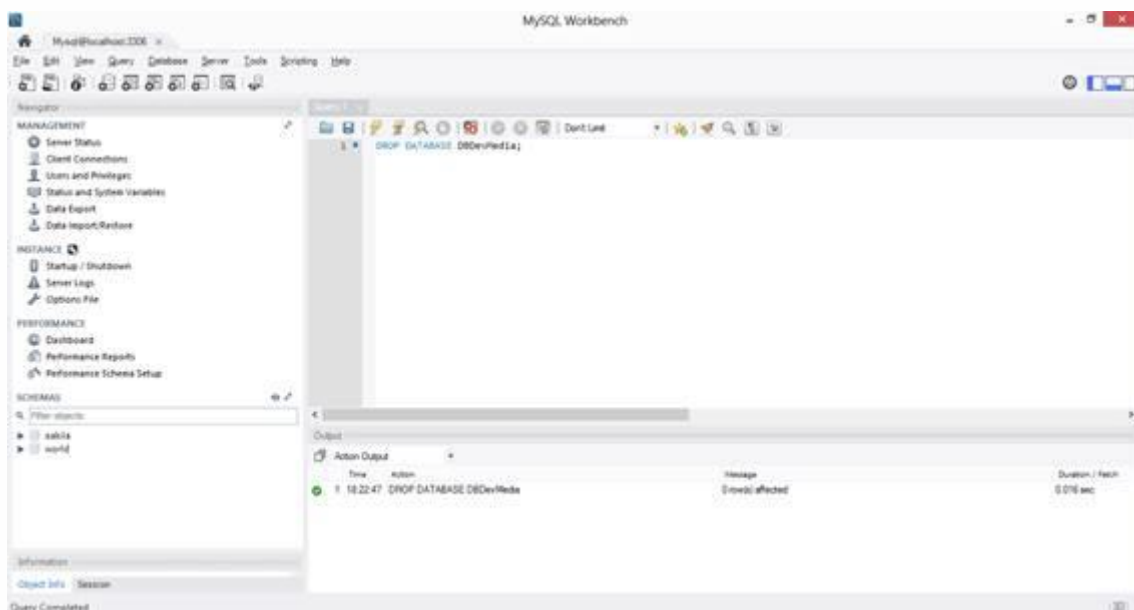
Para visualizar uma lista com todos os bancos de dados existentes no servidor, use o comando:

```
SHOW DATABASES;
```

Observe que todos os **comandos em MySQL** sempre termina com “;” no final. Essa sintaxe é obrigatória para que o MySQL possa entender o termino do comando.

Para remover os bancos de dados existentes no servidor, utilize o comando a seguir, mas atenção, pois uma vez executado, a ação é **irreversível**:

```
DROP DATABASE DBDevMedia;
```



### 5.2.1 Criando tabelas no MySQL

Dada a grande quantidade de parâmetros aceitos, a declaração `CREATE TABLE` é uma das mais **complexas no MySQL**.

Vamos começar selecionando o banco de dados que ganhará a nova tabela usando a sintaxe:

```
USE DBDevMedia;
```

De acordo com a documentação disponível pela Oracle, a sintaxe simplificada seria:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
(create_definition, ...)
```

A parte de declaração que se encontra entre colchetes é opcional:

- **TEMPORARY**: Indica que a tabela criada será temporária, ou seja, ela expira assim que sua **sessão no MySQL** terminar. Use-a sempre que estiver fazendo testes.
- **IF NOT EXISTS**: Verifica a prévia existência da tabela e evita uma interrupção do script causada por erro. Como o **MySQL é case sensitive**, tabelas com nomes iguais, mas usando letras em caixa alta, como em `tbl_name` e `Tbl_name`, são consideradas tabelas totalmente diferentes.

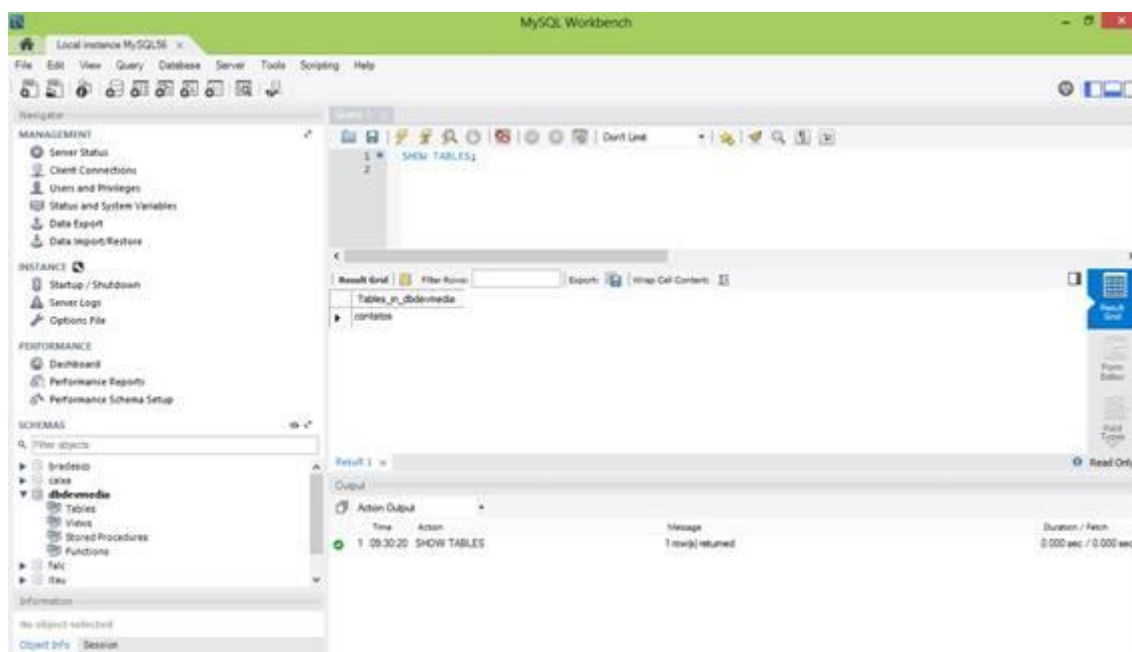
Uma tabela é composta por uma ou mais colunas, cada qual com suas definições.

Vamos começar pela criação de uma agenda telefônica. A tabela contatos terá a seguinte estrutura da **Listagem 1**.

```
CREATE TABLE contatos (
    nome VARCHAR(50) NOT NULL,
    telefone VARCHAR(25) NOT NULL
);
```

Para verificar se a tabela foi criada use o comando:

```
SHOW TABLES;
```



Podemos melhorar um pouco mais a tabela contatos, ao acrescentar mais alguns campos, como sobrenome dos contatos, DDD, data de nascimento e e-mail. Antes de criar uma tabela, com o mesmo nome, vamos remover a anterior usando o comando:

```
DROP TABLE contatos;
```

Agora, vamos criar a tabela, conforme a **Listagem 2**.

```
CREATE TABLE IF NOT EXISTS contatos (
    nome VARCHAR(20) NOT NULL,
```



```
sobrenome VARCHAR(30) NOT NULL,  
ddd INT(2) NOT NULL,  
telefone VARCHAR(9) NOT NULL,  
data_nasc DATE NULL,  
email VARCHAR(30) NULL);
```

A chave primária é o que torna a linha ou o registro de uma tabela único. Geralmente, é utilizada uma sequência automática para a geração dessa chave para que ela não venha a se repetir. Em nosso caso, o `nro_contato` será único, com uma sequência numérica que identificará o registro.

A cláusula `auto_increment` é utilizada para incrementar automaticamente o valor da chave primária. Por padrão, essa cláusula inicia com 1. Porém, se houver a necessidade de iniciar por outro valor, podemos fazer como no exemplo a seguir:

```
CREATE TABLE contatos AUTO_INCREMENT=100;
```

### 5.2.2 Alterando a tabela

Imagine que sua tabela já contenha dados armazenados e você precisa acrescentar mais um campo (chamado Ativo) na tabela de contatos.

Conhecidamente pensaríamos em usar o *drop table* para excluir a tabela e recriá-la com o novo campo, mas perder os dados é algo inviável.

Nossa solução é utilizar a sintaxe `ALTER TABLE`, que permite alterar a estrutura da tabela existente. Por exemplo, você pode adicionar ou deletar colunas, criar ou remover índices, alterar o tipo de coluna existentes, ou renomear coluna ou tabelas. Você também pode alterar o comentário para a tabela e tipo de tabela.

Para adicionar colunas use o comando `ADD`, seguido do nome e dos atributos da coluna que será adicionada e, da sua posição dentro da tabela com o auxílio do parâmetro `AFTER`. Assim, para adicionarmos a coluna ativo, usaremos o código a seguir:

```
ALTER TABLE contatos  
ADD ativo SMALLINT NOT NULL AFTER email;
```

Para ver o resultado das alterações, dê o comando:

```
DESCRIBE contatos;
```

Para alterar os atributos e nome de colunas usamos o parâmetro **CHANGE**, seguido da denominação da coluna a ser alterada e dos novos atributos. Para mudar os atributos da coluna nome, utilizaremos a seguinte sintaxe:

```
ALTER TABLE contatos  
CHANGE telefone telefone CHAR(9) NOT NULL;
```

Vocês devem ter percebido que a palavra “telefone” foi utilizada duas vezes. Isso ocorre porque se indica primeiro a coluna e depois seus novos atributos, e o nome da coluna é um de seus atributos.

Para mudar o nome da coluna e manter seus demais atributos usamos a sintaxe a seguir:

```
ALTER TABLE contatos  
CHANGE telefone fone VARCHAR(9) NOT NULL;
```

### 5.3 Linguagem de Manipulação de Dados (DML) e Linguagem de Transação de Dados (DTL)

A **Linguagem de Manipulação de Dados**, ou **DML**, é o grupo de comandos responsáveis pela manipulação de dados em um banco de dados; isto geralmente implica em inserir, editar ou excluir linhas em tabelas SQL.

#### 5.3.1 Inserindo registros

Depois da tabela pronta precisamos agora de registros em nosso banco de dados. Para esse exemplo não vamos usar nenhuma aplicação para inserir esses dados, mas sim diretamente pelo SGBD através de comando SQL.

Vamos fazer o primeiro **INSERT** na tabela **contatos** com o comando **INSERT INTO contatos**. Entre parênteses informaremos em quais colunas queremos inserir os registros e depois devemos informar qual o valor para cada coluna, como mostra a Listagem 3.

```
INSERT INTO contatos (nome
```

```
,sobrenome
,ddd
,telefone
,data_nasc
,email
,ativo)
VALUES('Bruno'
,'Santos'
,11
,999999999
,'2015-08-22'
,'contato@dominio.com.br'
,1);
```

Se você quiser inserir em todos os campos da tabela, não é necessário descrever quais serão populados. Apenas não se esqueça de conferir se os valores estão na sequência correta, como na **Listagem 4**, onde omitimos estes campos. O SGBD subentende que todos os campos serão populados.

```
INSERT INTO contatos VALUES('Bruno'
,'Santos'
,11
,999999999
,'2015-08-22'
,'contato@dominio.com.br'
,1);
```

Observe que em nenhum momento foi mencionado o campo `nro_contato` ou acrescentado um valor diretamente, isso porque este campo foi definido como `auto_increment`, desta forma, o campo recebe o valor automaticamente.

### 5.3.2 Alterando registros

Para alterar os registros usamos o comando `UPDATE`.

No exemplo anterior inserimos um sobrenome errado. Para corrigir usamos a sintaxe da Listagem 5.

```
UPDATE contatos SET  
sobrenome= 'Nascimento' WHERE nro_contato= 100;  
commit;
```

Podemos atualizar mais de um campo de uma vez só, separando com “,” , como mostra a Listagem 6.

```
UPDATE contatos SET  
sobrenome= 'Nascimento'  
, ddd= 015  
, telefone= '0123456789'  
WHERE nro_contato = 100  
commit;
```

Perceba que, além do UPDATE utilizamos o SET para informar qual campo que queremos alterar. O WHERE indica a condição para fazer a alteração e, em seguida, o commit diz ao SGBD que ele pode realmente salvar a alteração do registro. Se, por engano, fizemos o UPDATE incorreto, antes do commit podemos reverter a situação usando a instrução SQL rollback, da seguinte maneira:

```
UPDATE contatos SET  
sobrenome= 'Nascimento' WHERE nro_contato= 100;  
rollback;
```

Com isso, o nosso SGBD vai reverter a última instrução. Porém, se tiver a intenção de utilizar o rollback, faça-o antes de aplicar o commit, pois se você aplicar o UPDATE ou qualquer outro comando que necessite do commit, não será possível reverter.

As instruções commit e rollback são tratadas pela **Linguagem de Transação de Dados (DTL)**.

### 5.3.3 Excluindo registros

Para deletar algum registro usamos a instrução SQL DELETE. Diferente do DROP, ele deleta os registros das colunas do banco de dados.

O DROP é usado para excluir objetos do banco, como tabelas, colunas, *views* e *procedures*, enquanto, o delete deletará os registros das tabelas, podendo excluir apenas uma linha ou todos os registros. Desta maneira, vamos apagar o primeiro registro da tabela contatos usando o seguinte comando:

```
DELETE FROM contatos WHERE nro_contato= 100;  
commit;
```

Para deletar todos os registros da tabela de clientes usamos o comando:

```
DELETE FROM contatos;  
commit;
```

Observe que, ao empregar o DELETE você também deve usar o commit logo após a instrução. Da mesma maneira, podemos também utilizar o rollback para não efetivar uma exclusão de dados incorretos.

Além do DELETE, podemos eliminar os dados usando a instrução SQL TRUNCATE, que não necessita de commit. Nem o rollback pode reverter à operação.

Isso ocorre porque, quando você utiliza o DELETE, o SGBD salva os seus dados em uma tabela temporária e, quando aplicamos o rollback, ele a consulta e restaura os dados. Já o TRUNCATE não a utiliza, o SGBD faz a eliminação direta. Para usar esse comando utilizar a sintaxe a seguir:

```
TRUNCATE TABLE contatos;
```

Essa instrução não pode ser usada dentro da cláusula WHERE.

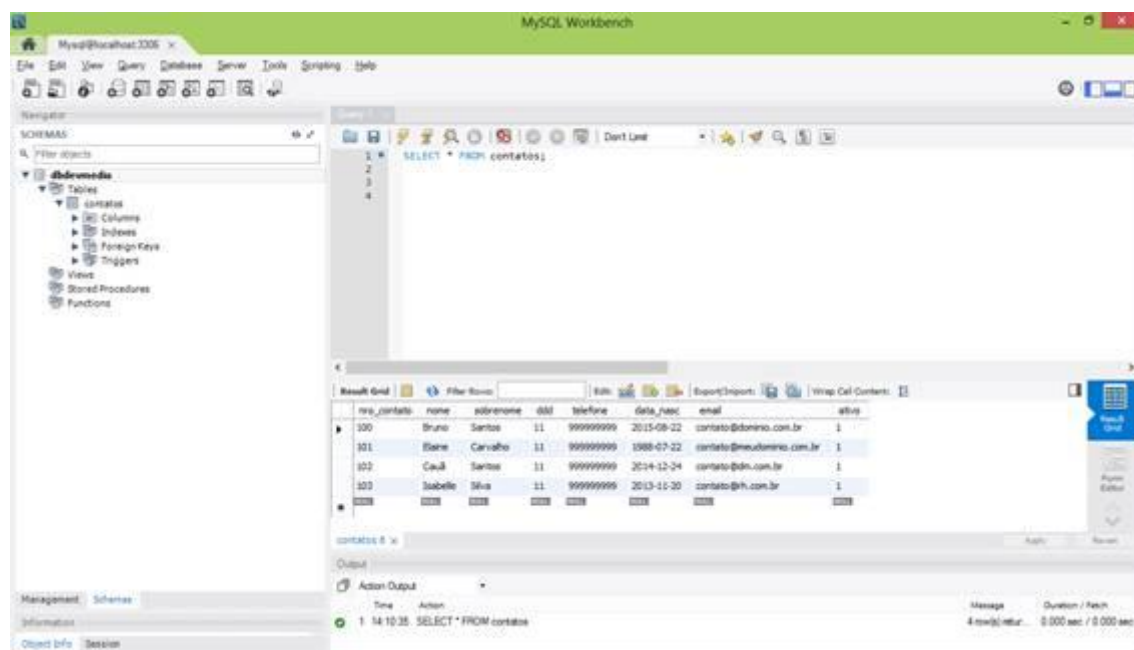
## 5.4 Linguagem de Consulta de Dados (DQL)

O objetivo de armazenar registros em um banco de dados é a possibilidade de recuperar e utilizá-los em relatórios para análises mais profundas. Essa recuperação é feita através de consultas.

O comando SQL utilizado para fazer consultas é o SELECT. Selecionando os dados, devemos dizer ao SGBD de onde queremos selecionar, através do comando FROM.

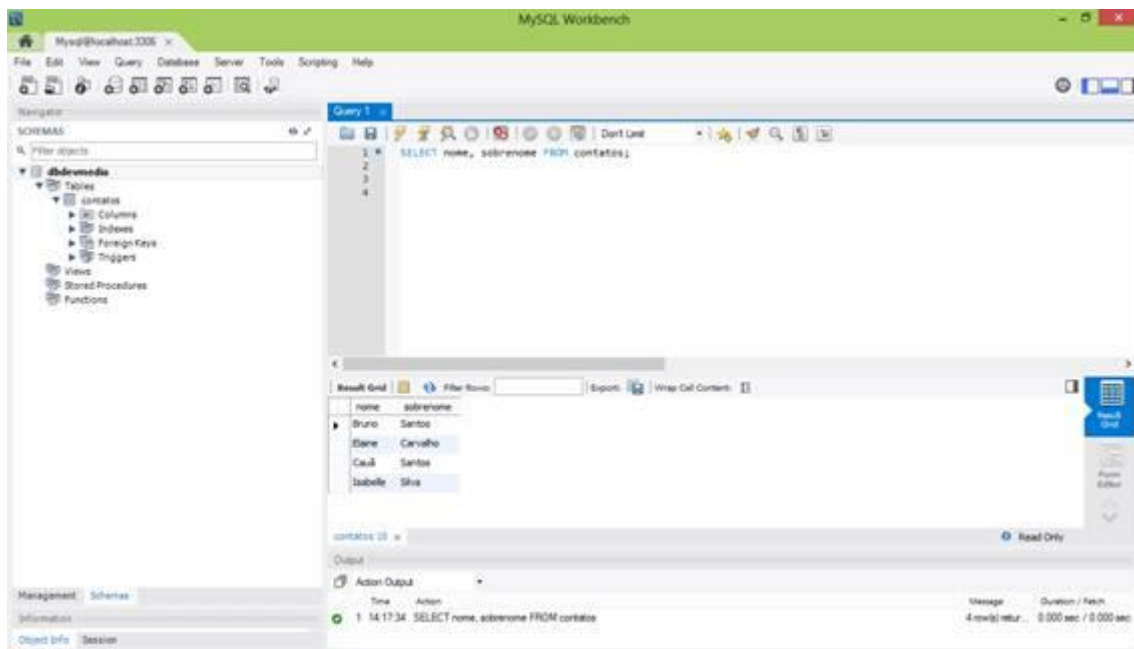
Como exemplo, vamos selecionar os registros da tabela de contato. Quando não queremos selecionar um ou vários campos específicos, utilizamos o asterisco (\*):

```
SELECT * FROM contatos;
```



Se quisermos selecionar os registros dos campos **nome** e **sobrenome**, usamos a sintaxe:

```
SELECT nome, sobrenome FROM contatos;
```



Ainda podem surgir situações que necessitem selecionar apenas um registro. Neste caso, utilizamos o WHERE

Vamos selecionar o cliente com uma cláusula que deve ter nro\_contato= 101:

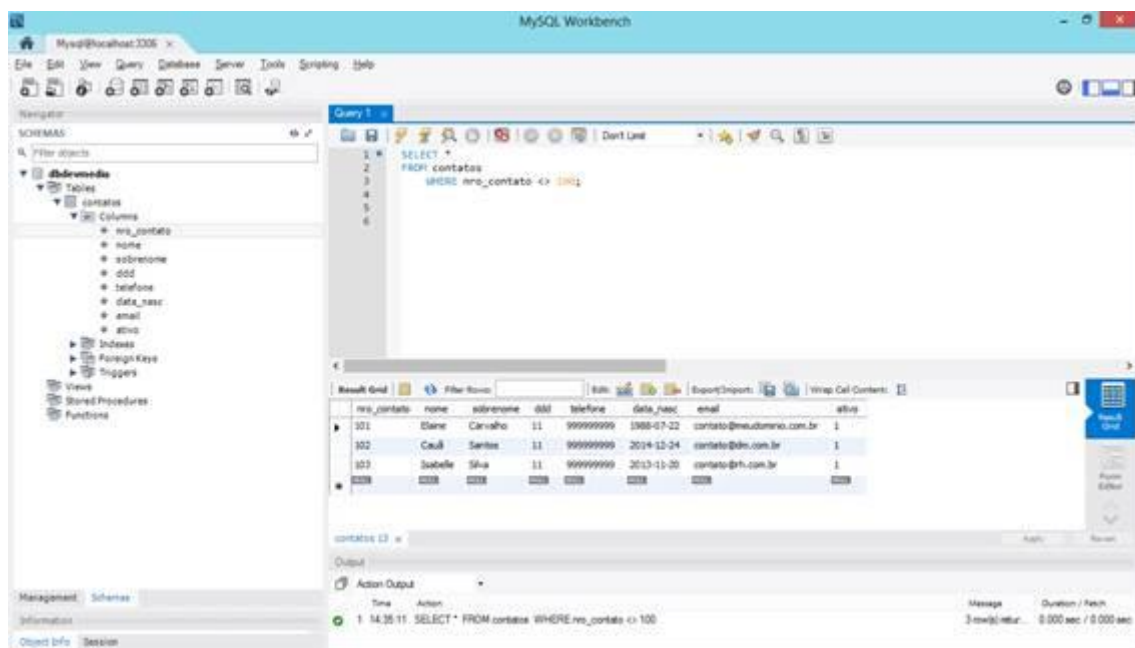
```
SELECT nome, sobrenome
FROM contatos
WHERE nro_contato= 100;
```

Para colunas do tipo texto será necessário colocar o valor entre aspas simples, assim dizemos ao SGBD que estamos querendo comparar o valor com uma coluna do tipo texto:

```
SELECT nome, sobrenome
FROM contatos
WHERE nome= 'Bruno';
```

E se quiséssemos todos os clientes que sejam diferentes de '100'? Fariamos uma consulta utilizando o **operador do MySQL** diferente <>:

```
SELECT nome, sobrenome
FROM contatos
WHERE nro_contato<> 100;
```



Além dos operadores de comparação = e <>, temos os seguintes operadores:

- >: maior;
- <: menor;
- >=: maior e igual;
- <=: menor e igual.

A cláusula DISTINCT retorna apenas uma linha de dados para todo o grupo de linhas que tenha o mesmo valor. Por exemplo, executando a consulta a seguir:

```
SELECT DISTINCT sobrenome FROM contatos;
```

Os valores retornados são apenas três, pois Santos se repete duas vezes:

Santos

Carvalho

Silva

Já a cláusula ALL é o oposto de DISTINCT, pois retorna todos os dados. Observe a consulta a seguir:



```
SELECT ALL sobrenome FROM contatos;
```

Repare que o resultado a seguir apresenta o sobrenome Santos duas vezes:

Santos

Carvalho

Santos

Silva

A cláusula `ORDER BY` retorna os comandos em ordem ascendente (ASC) ou descendente (DESC), sendo o padrão ascendente. Vejamos um exemplo:

```
SELECT nome FROM contatos ORDER BY nome DESC;
```

Repare que os nomes são retornados em ordem decrescente

Isabelle

Elaine

Cauã

Bruno

A cláusula `LIMIT [início,] linhas` retorna o número de linhas especificado. Se o valor *início* for fornecido, aquelas linhas são puladas antes do dado ser retornado. Lembre-se que a primeira linha é 0.

```
SELECT * FROM contatos LIMIT 3,1;
```

O resultado da consulta será:

```
103 Isabelle Silva 11 999999999 2013-11-20 contato@rh.com.br
```

Para incrementar as consultas podemos usar algumas funções. A seguir apresentaremos as mais comuns:

- A função `ABS` retorna o valor absoluto do número, ou seja, só considera a parte numérica, não se importando com o sinal de positivo ou negativo do mesmo. Por exemplo: `ABS(-145)` retorna 145;

- A função BIN considera o binário de número decimal. Por exemplo: BIN(8) retorna 1000;
- A função CURDATE() / CURRENTDATE() retorna a data atual na forma YYYY/MM/DD. Por exemplo: CURDATE() retorna 2002/04/04;
- A função CURTIME() / CURRENTTIME() retorna a hora atual na forma HH:MM:SS. Por exemplo: CURTIME() retorna 13:02:43;
- A função DATABASE retorna o nome do banco de dados atual: Por exemplo: DATABASE() retorna DBDevMedia;
- A função DAYOFMONTH retorna o dia do mês para a data dada, na faixa de 1 a 31. Por exemplo: DAYOFMONTH("2004-04-04") retorna 04;
- A função DAYNAME retorna o dia da semana para a data dada. Por exemplo: DAYNAME("2004-04-04") retorna Sunday;
- A função DAYOFWEEK retorna o dia da semana em número para a data dada, na faixa de 1 a 7, onde o 1 é domingo. Por exemplo: DAYOFWEEK("2004-04-04") retorna 1;
- A função DAYOFYEAR retorna o dia do ano para a data dada, na faixa de 1 até 366. Por exemplo: DAYOFYEAR("2004-04-04") retorna 95;
- A função FORMAT(NÚMERO, DECIMAIS) formata o número nitidamente com o número de decimais dado. Por exemplo: FORMAT(5543.00245,2) retorna 5.543.002,45

A função LIKE merece um destaque especial, pois faz uma busca sofisticada por uma substring dentro de uma string informada. Temos, dentro da função LIKE, os seguintes caracteres especiais utilizados em substrings:

- %: busca zero ou mais caracteres;
- \_: busca somente um caractere.

Vamos a alguns exemplos:

```
SELECT nome From contatos Where nome like 'B%';
```

O caractere '%' nessa consulta indica que estamos procurando nomes que possuem a inicial B, ou seja, com base na nossa tabela contatos, o retorno será apenas Bruno.

```
SELECT nome From contato Where nome like '_a%';
```

O caractere '\_' na consulta indica que estamos procurando nomes nos quais a letra A é a segunda letra do nome, ou seja, o retorno será apenas Cauã.

```
SELECT nome From contato Where nome like '%o';
```

A consulta buscou nomes em que a última letra é o caractere 'O', ou seja, teremos como retorno apenas Bruno.

Outra função importante para retorno de consultas é Left, que retorna os primeiros caracteres à esquerda de uma string. Sua sintaxe é apresentada a seguir:

```
LEFT(string,tamanho)
```

A consulta a seguir retornará os três primeiros caracteres à esquerda dos registros da coluna nome:

```
SELECT LEFT(nome,3) from contatos
```

O resultado será:

Bru

Ela

Cau

Isa

A função Right é semelhante a função Left, mas esta retorna os últimos caracteres à direita de uma string. Sua sintaxe também é semelhante:

```
RIGHT(string1,tamanho)
```

Repare que na consulta a seguir são retornados os quatro últimos caracteres à direita dos nomes da tabela contatos:

```
SELECT RIGHT(nome,4) From contatos;
```

O resultado será:

runo

aine

Cauã

ele