Instituto Telles

Material do Aluno

Técnico em Desenvolvimento *Web* e Cibersegurança

Front-End: princípios

Preparação para *Front End*
Ambiente e ferramentas de trabalho
HTML5, CSS3 e Design

Instituto Telles

Parcerias:

SEDUC
Secretaria de Estado da Educação

SECTI
Secretaria de Estado de Ciência, Tecnologia e Inovação

GOVERNO DE GOIÁS
O ESTADO QUE DÁ CERTO

Preparação para Front End
Ambiente e ferramentas de trabalho (PARTE 1)

Sumário

Preparação para Front End
Ambiente e ferramentas de trabalho

INTRODUÇÃO

Seja muito bem-vindo à sua jornada pelos princípios fundamentais do desenvolvimento *front-end*! Este material representa o ponto de partida para explorar o vasto mundo que molda a interface visual e a experiência do usuário em projetos digitais.

Ao iniciarmos esta trajetória, você estará prestes a desvendar os mistérios que envolvem a transição de diferentes linguagens de programação. Além disso, exploraremos editores de código *on-line* e compreenderemos a importância da escolha entre **quais sistemas utilizar para desenvolver o seu produto**.

Prepare-se para mergulhar nos sistemas de controle de versão, onde abordaremos não apenas o conceito de códigos, mas também a sua aplicação em contextos individuais e colaborativos, especialmente em plataformas de hospedagem de projetos. Estamos aqui para tornar esse percurso de aprendizado acessível e envolvente para todos os participantes, independentemente do nível de conhecimento técnico. Vamos lá?

Esta não é apenas um material didático, é a porta de entrada para um universo repleto de desafios e descobertas. Ao longo dessa caminhada, você não apenas adquirirá habilidades técnicas essenciais, mas também experimentará a satisfação de transformar conceitos abstratos em interfaces interativas. Estamos empolgados para iniciar essa jornada com você. Vamos explorar, aprender e criar juntos neste volume dedicado aos Princípios do *front-end*!

CAPÍTULO 01

Preparação para *front-end*

O que esperar deste capítulo:

- Aplicar os conceitos de lógica de programação em uma linguagem de programação (Python);
- Conhecer o ambiente on-line Google Colab para escrever e executar códigos de programação na linguagem Python.

1.1 Transição do *no-code* para linguagem de programação

Reflita sobre a pergunta abaixo.

Isso já aconteceu com você?

Alguma vez você se deparou com uma ideia brilhante para criar um aplicativo, mas ficou sem saber por onde começar? Ou talvez tenha sentido o desejo de desenvolver um jogo, mas pensou que o processo poderia ser bastante complexo?

Isso é comum ao tentar transformar uma ideia inovadora em um aplicativo funcional. Esse processo pode parecer confuso no início, mas vamos explorar algumas ferramentas fáceis de usar, conhecidas como *no-code* (sem-código), que não exigem muita habilidade em programação.

Nos últimos anos, ferramentas ***no-code*** tornaram-se a porta de entrada para muitos aspirantes a desenvolvedores, empreendedores e criativos digitais. A revolução *no-code* simplificou o processo de desenvolvimento, permitindo que ideias se

materializassem com facilidade, independentemente do conhecimento prévio em programação. Nesta introdução, destacamos como essa revolução **democratizou o desenvolvimento digital**, capacitando uma gama diversificada de indivíduos a criar e inovar.

Um bom exemplo desse processo é o jogo Little Misfortune, criado e lançado pelo estúdio independente sueco Killmonday Games. O jogo narra as aventuras e explorações sombrias de uma criança de oito anos que tem o desejo de conquistar o Prêmio da Felicidade Eterna para a sua mãe.

Para saber mais, acesse:.

Fonte: Vídeo *No-code: O futuro da programação será sem códigos?*, do canal Olhar Digital. Disponível em: https://youtu.be/bNsDTCYA6n0. Acesso em: 21 fev 2024.

Após ver o vídeo, tente responder:

A. Você acredita que o futuro da programação pode ser *no-code*?

B. Qual é a sua impressão sobre as ferramentas *no-code*?

As ferramentas *no-code* podem ser bons recursos para muitos desenvolvedores iniciantes. Entretanto, à medida que exploramos o cenário *no-code*, torna-se evidente que certos projetos demandam **mais do que essas ferramentas podem oferecer.** A busca por **personalização**, a **complexidade crescente dos projetos** e a **necessidade de maior controle** impulsionam a decisão de trilhar o caminho da programação tradicional. Este subtema não apenas expõe essas limitações, mas também apresenta a programação como a chave para desbloquear um mundo de possibilidades ilimitadas.

No-code vs. *Custom code*: o duelo de estratégias

Disponível em: freepik-http://tinyurl.com/3u993wym. Acesso em: 29 jan. 2024.

O desenvolvimento de *software* abrange três abordagens principais: *custom code* (código personalizado), *low-code* (código reduzido) e *no-code* (sem código).

Qual é a diferença?

Custom code	Low code	No-code
É traduzido como "código personalizado". Refere-se à prática de escrever códigos manualmente para atender a requisitos específicos ou realizar personalizações avançadas em uma aplicação.	Traduzido como "baixo código", proporciona uma abordagem intermediária, oferecendo uma interface visual para a construção de aplicações, mas também permitindo a inclusão de código para personalizações avançadas.	Traduzidas como "sem código", as ferramentas são projetadas para permitir que usuários criem aplicações sem a necessidade de codificação manual.

Fonte: elaborado pelo autor (2024).

Cada abordagem atende a diferentes níveis de complexidade, proporcionando diversas opções para o desenvolvimento de *software* de acordo com as necessidades e habilidades dos usuários. A escolha da abordagem mais adequada para um caso específico envolve diversos fatores, como a **complexidade do projeto**, o **nível de conhecimento do usuário**, **custos**, **tempo** e **prazos**.

Além disso, para determinar a melhor abordagem para o seu projeto, é fundamental explorar minuciosamente cada uma das três categorias. Então, que tal colocar em prática os conhecimentos adquiridos até aqui?

Vamos praticar!

DESAFIO PRÁTICO

Transição de *no-code* para linguagem de programação

Desafio: CodeCraft Academy: a jornada da inovação

Descrição

Imagine-se no coração pulsante da CodeCraft Academy, uma instituição pioneira que abraça a convergência entre criatividade e tecnologia. Você, talentoso aprendiz, embarca em uma jornada épica de transformação digital, encarregado de aprimorar o ecossistema *no-code* para enfrentar desafios de desenvolvimento mais complexos. A CodeCraft Academy está evoluindo, e é chegada a hora de dar um salto de inovação, transacionando do conforto do *no-code* para a profunda compreensão da linguagem de programação.

Objetivos

Entender as diferentes bases para desenvolvimento:

- Entender a importância e as diferenças entre no-code e programação tradicional;
- Elaborar um documento considerando as diferenças e explicando o porquê de utilizar cada uma das formas de desenvolvimento.

Orientações

- Inicie a exploração do universo no-code, destacando tanto as vantagens quanto as limitações dessa abordagem. Após compreender a importância dessa transição, concentre-se nos objetivos propostos. Se você já possui experiência ou conhece pessoas que atuam nessas formas de programação, compartilhe relatos que abordem os pontos de vista ao trabalhar com ambas as modalidades de desenvolvimento.

1.2 Editores de código *on-line*

Disponível em: freepik-http://tinyurl.com/yyrcm3ux. Acesso em: 29 jan. 2024.

O que são editores de código *on-line*?

No coração do desenvolvimento de *software*, uma revolução silenciosa está transformando a forma como pensamos e lidamos com o código. Os editores de código *on-line* estão liderando essa mudança, inaugurando uma nova era de desenvolvimento remoto acessível e colaborativo.

Imagine um mundo onde a criação de *software* não está restrita a um computador específico, mas pode acontecer em qualquer lugar com uma conexão à internet. É nesse cenário **dinâmico** e **globalizado** que os editores *on-line* trazem inovação, eliminando

obstáculos e redefinindo a experiência de codificação.

Principais editores de código *on-line*:

Repl.it

- **Interação, experimentação e aprendizado**: oferece ambientes de programação interativos que permitem aos usuários experimentar, testar e aprender programação de maneira prática. É uma plataforma especialmente útil para iniciantes, proporcionando uma abordagem *hands-on* para o desenvolvimento de código;
- **Suporte a várias linguagens de programação:** suporta uma ampla variedade de linguagens de programação, permitindo que os usuários escolham a linguagem que desejam aprender ou praticar. Isso torna a plataforma versátil e acessível a diferentes interesses e níveis de habilidade.

CodeSandbox

- **Ambiente robusto para tecnologias *front-end*:** projetado para o desenvolvimento de tecnologias *front-end*, proporcionando um ambiente completo e robusto para a criação de aplicações *web* interativas. Se destaca por oferecer ferramentas específicas para o desenvolvimento do lado do cliente, como React, Vue e Angular;

- **Fácil compartilhamento de projetos e colaboração em tempo real:** oferece a capacidade de compartilhar projetos de forma simples e colaborar em tempo real. Isso é particularmente útil para equipes de desenvolvimento que precisam trabalhar em conjunto em projetos *front-end*.

Glitch

- **Ágil na criação e colaboração de aplicativos:** essa plataforma facilita o desenvolvimento e a colaboração em projetos *web*. Adotando uma abordagem ágil, permite que os desenvolvedores experimentem e colaborem de maneira eficiente;
- **Intuitivo e simplificado:** destaca-se por sua interface amigável, tornando o desenvolvimento web mais acessível. Projetado para ser intuitivo, especialmente

para desenvolvedores que desejam criar rapidamente sem se preocupar com configurações complicadas.

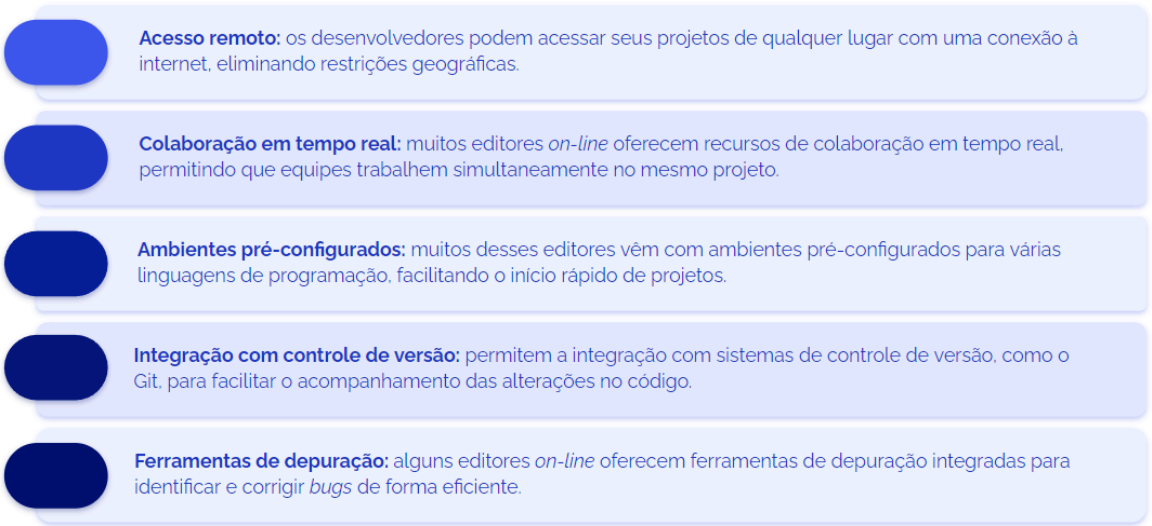
Visual Studio Code *on-line*

- **Versão on-line do poderoso Visual Studio Code:** é uma versão baseada na *web* do popular editor de código Visual Studio Code. Oferece uma experiência de desenvolvimento familiar e poderosa diretamente no navegador, permitindo que os desenvolvedores acessem suas configurações, extensões e projetos de qualquer lugar;

- **Familiar e flexível:** replica a experiência do Visual Studio Code ao manter sua interface amigável e suporte a extensões. A grande vantagem é a conveniência de ser acessado diretamente no navegador, eliminando a necessidade de instalação local. Isso é especialmente útil para desenvolvedores que buscam flexibilidade em seus ambientes de desenvolvimento.

Principais características dos editores de código *on-line*

Os editores de código *on-line* são ferramentas versáteis que proporcionam uma abordagem **flexível** e **colaborativa** para o desenvolvimento de *software* e são uma escolha atrativa para desenvolvedores que estão em busca de eficiência, colaboração e acesso simplificado a ambientes de desenvolvimento. Confira, a seguir, as suas principais **características**.



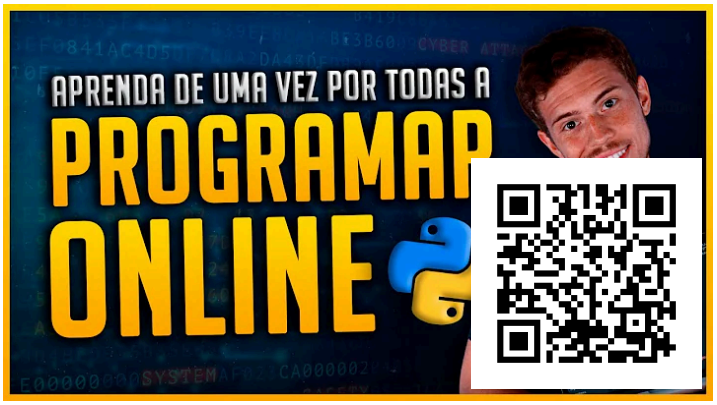
Fonte: elaborado pelo autor (2024).

GOOGLE COLAB + Python

O Google Colab destaca-se como uma inovação significativa para atividades de programação. Ele proporciona uma plataforma *on-line* dedicada à programação, especialmente voltada para a linguagem Python, e sua grande vantagem é o fato de dispensar a necessidade de instalação de qualquer *software* em seu computador.

A linguagem Python é como um idioma para o programador contemporâneo. Sua facilidade de compreensão e a vasta gama de bibliotecas disponíveis tornam a criação de programas e projetos mais acessível. Se a sua intenção é explorar o universo da programação, o Google Colab e Python surgem como excelentes alternativas.

Saiba mais no vídeo:



Fonte: Vídeo *Como Programar em Python Online (Google Colab) - Sem Instalar no Computador, do* canal Hashtag Programação. Disponível em: https://youtu.be/5zr9HWWs8nI. Acesso em: 2 fev. 2024.

Depois de assistir ao vídeo, responda:

A. Qual é a principal vantagem de utilizar a ferramenta Google Colab ao programar *on-line*?

B. Há limitações ao utilizar o Google Colab com Python? Quais?

Agora que você já entendeu um pouco mais do Google Colab e as suas principais definições, siga os exemplos abaixo, fazendo as aplicações.

```
# Boas-vindas
print ("Olá aluno! Este é o Google Colab, uma ferramenta online para atividades em Python.")
print ("Você pode realizar suas atividades sem a necessidade de instalar uma IDE.")
print ("Conforme explicado no material sobre editores online, esta ferramenta é ideal para começar a jornada em Python. Divirta-se!")
print (" abaixo irei deixar alguns exercícios para que vocês entendam a logica ")

#oq é print

#Python é uma função fundamental que permite a exibição de informações na saída padrão, geralmente no console.
#Ele é utilizado para imprimir mensagens, variáveis, resultados de operações ou qualquer conteúdo que você deseje visualizar durante a execução do programa.

#A sintaxe básica envolve o uso da palavra-chave print seguida  pelos elementos que você deseja exibir, separados por vírgulas.
#por exemplo:
print("Olá, Mundo!")

 #exibirá a mensagem "Olá, Mundo!" no console.

# Exercício 1: Condicionais
idade = 20
if idade >= 18:
    print("Você é maior de idade.")
else:
    print("Você é menor de idade.")

# Exercício 2: Looping
for i in range(3):
    print(f"Isto é o loop número {i + 1}.")

# Exercício 3: Explicação sobre o print
mensagem = "Python é incrível!"
print("A mensagem é:", mensagem)

# Exercício 4: Looping com condicional
for num in range (1, 6):
    if num % 2 == 0:
        print(f"{num} é um número par.")
    else:
        print(f"{num} é um número ímpar.")

# Exercício 5: Utilizando condicional para imprimir uma mensagem
temperatura = 25
if temperatura > 30:
    print("Está muito quente!")
elif temperatura < 15:
    print("Está um pouco frio.")
else:
```

```
    print("A temperatura está agradável.")

# Exercício 6: Looping com condição de parada
contador = 0
while contador < 4:
    print(f"Contagem: {contador}")
    contador += 1

# Exercício 7: Função com parâmetro
def saudacao(nome):
    return f"Olá, {nome}! Bem-vindo ao mundo Python."

print(saudacao("João"))

# Exercício 8: Utilizando uma lista
frutas = ["Maçã", "Banana", "Uva"]
for fruta in frutas:
    print("Gosto de", fruta)

# Exercício 9: Imprimir números pares de 0 a 8
print("Números pares de 0 a 8:")
for num in range(0, 9, 2):
    print(num)

# Exercício 10: Condição com operadores lógicos
numero = 25
if numero > 10 and numero < 30:
    print("O número está entre 10 e 30.")
```

Fonte: elaborado pelo autor(2024).

DESAFIO PRÁTICO

Introdução à Python com Google Colab

Desafio: Jornada inicial com Python no Google Colab

Descrição

Como membro da TechLab Academy, os novos aprendizes são desafiados a aprofundar os seus conhecimentos em Python no ambiente colaborativo do Google Colab. A tarefa consiste em criar um *script* dinâmico que não apenas cumprimenta o usuário, mas também incorpora recursos avançados, como entrada personalizada, manipulação de

dados e visualização gráfica. Este desafio oferece uma experiência mais complexa para explorar a versatilidade do Python e a eficácia do Google Colab.

Objetivos

- Implementar um *script* em Python no Google Colab;
- Apresentar visualmente os resultados das operações por meio de gráficos básicos;
- Entender o uso do Google Colab.

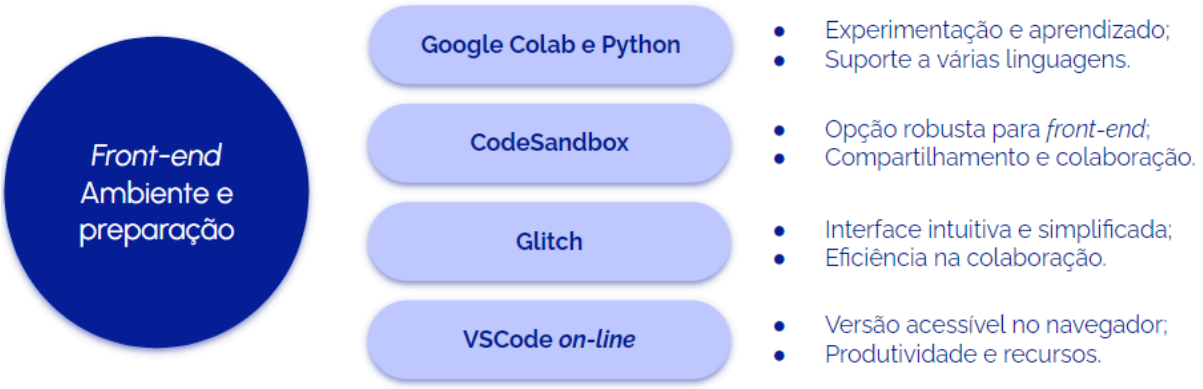
Orientações

➔ Explore o ambiente Google Colab, inserindo e executando células de código com elementos simples;

➔ Perceba como o Colab é prático para iniciantes, mesmo ao enfrentar desafios mais complexos, e faça uso da utilização extensiva para comentários explicativos no código.

RESUMO

Durante a nossa jornada, exploramos ambientes e plataformas para aprimorar o desenvolvimento, destacando ferramentas essenciais:



Fonte: elaborado pelo autor (2024).

Essas ferramentas, quando aliadas a tecnologias preexistentes, formam um arsenal versátil, proporcionando conhecimento prático e habilidades essenciais para prosperar em um ambiente digital em constante evolução. Que essa diversidade de recursos amplie significativamente as suas possibilidades no universo do desenvolvimento *web*.

ATIVIDADE DE FIXAÇÃO

1. Você já se deparou com uma situação em que ferramentas *no-code* não atenderam totalmente às suas necessidades de desenvolvimento? Como essa limitação motiva a exploração da programação tradicional?

2. Ao considerar o seu projeto atual ou algum que você queira desenvolver, quais benefícios específicos você acredita que a programação tradicional poderia oferecer em comparação com abordagens *no-code*?

3. Você já enfrentou dificuldades ao tentar entender a sintaxe de uma nova linguagem de programação? Como superou esse desafio?

4. Você já colaborou com outros desenvolvedores em tempo real? Como a experiência de edição colaborativa pode facilitar o desenvolvimento?

5. Em que situações você optaria por usar uma plataforma para computador em vez de um editor de código *web*? Como essa escolha poderia impactar o seu fluxo de trabalho?

6. Qual plataforma *no-code* é especialmente projetada para o desenvolvimento *front-end*, com destaque para tecnologias como React, Vue e Angular?

7. Você consegue pensar em algum projeto famoso em que foi utilizada uma ferramenta *no-code*? Se sim, como ela foi aplicada?

8. Na sua opinião, quais características são essenciais em um editor de código *on-line* para tornar a colaboração eficiente?

9. Você já utilizou ferramentas *no-code* em algum projeto? Se sim, quais foram os benefícios imediatos? Houve desafios significativos?

10. Ao refletir sobre os projetos anteriores, como a escolha entre *no-code* e *custom code* influenciou a escalabilidade e a manutenção a longo prazo desses projetos?

CAPÍTULO 02

Introdução ao desenvolvimento *web*

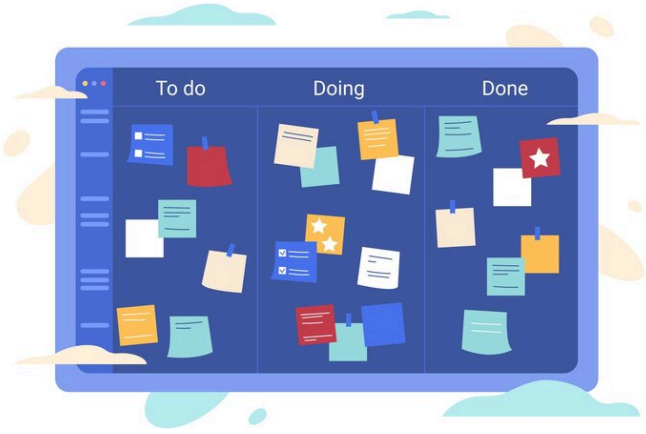
O que esperar deste capítulo:

- Compreender as diferenças, vantagens e situações de uso para facilitar a escolha entre Ambientes de Desenvolvimento Integrado (IDEs) e editores de código;
- Instalar, configurar e conhecer o ambiente de trabalho do editor de código VSCode.

2.1 O universo das IDEs: o poder além dos editores

Como você costuma se organizar no seu dia a dia? Você prefere usar um bloco de notas, um caderno para fazer anotações ou um aplicativo de celular?

Em determinados projetos, é crucial empregar ferramentas para coordenar as atividades de cada membro. Nessas situações, uma das abordagens mais comumente adotadas é o Kanban, que utiliza cartões para representar diversas tarefas, permitindo o acompanhamento dos status "A fazer", "Em andamento" e "Concluído".



Disponível em: freepik-http://tinyurl.com/58u4n5pr. Acesso em: 30 jan. 2024.

E quando o projeto é o desenvolvimento de um *software*, em que são utilizados códigos dentro de uma linguagem de programação? O que fazer?

Uma das soluções para este problema é o uso de um *Integrated Development Environment* (IDE), ou Ambiente de Desenvolvimento Integrado no português.

Um Ambiente de Desenvolvimento Integrado (IDE) é uma ferramenta completa projetada para **simplificar o processo de desenvolvimento de *software***. Essa plataforma unificada oferece uma variedade de funcionalidades essenciais para os desenvolvedores, reunindo diversas ferramentas em um único ambiente. O principal objetivo de um IDE é **otimizar** e **agilizar** o ciclo de vida do desenvolvimento, proporcionando uma experiência integrada que abrange desde a escrita e edição de código até a compilação, depuração e implementação.

Disponível em: freepik-http://tinyurl.com/bddc9k7x. Acesso em: 30 jan. 2024.

Um IDE típico inclui:

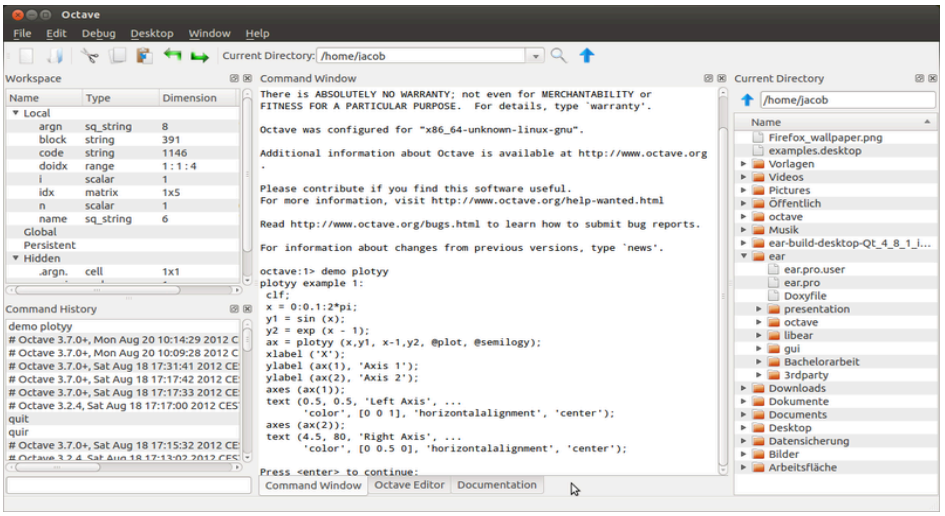
1. Um editor de código robusto que oferece recursos como **realce de sintaxe**, **autocompletar, refatoração** e **sugestões inteligentes**, contribuindo para a eficiência do desenvolvedor.
2. **Ferramentas de compilação e depuração** diretamente no ambiente, permitindo que o desenvolvedor identifique e corrija erros com facilidade.
3. Gerenciamento de projetos, controle de versão, e até mesmo, suporte a testes e documentação, fornecendo uma solução completa para as demandas complexas do desenvolvimento de *software*.

Fonte: elaborado pelo autor (2024).

As vantagens da utilização de um IDE são multifacetadas. Ele promove a **consistência** e a **eficiência no desenvolvimento**, oferecendo um ambiente unificado para todos os **aspectos do projeto**. A automação de tarefas rotineiras, como compilação e depuração, reduz erros humanos e aumenta a produtividade. Além disso, as sugestões inteligentes de código e as ferramentas de refatoração contribuem para a manutenção e evolução do código de maneira mais simplificada. Para equipes de desenvolvimento, a

padronização proporcionada por um IDE facilita a colaboração e a compreensão do código entre membros da equipe.

Ficou curioso para saber como pode ser a interface de um IDE? Confira o ambiente de desenvolvimento da linguagem de programação **GNU Octave**.



Disponível em: http://tinyurl.com/25e9f8tf. Acesso em: 30 jan. 2024.

Embora os IDEs ofereçam uma ampla gama de benefícios, é essencial notar que a escolha de um ambiente de desenvolvimento deve ser feita de maneira **consciente**, considerando as necessidades específicas do projeto e as preferências da equipe de desenvolvimento. Alguns IDEs são mais especializados para determinadas linguagens de programação, enquanto outros oferecem suporte mais amplo. A adaptabilidade e a flexibilidade são aspectos críticos a serem considerados na escolha de um IDE para garantir que ele atenda tanto a requisitos específicos quanto ao fluxo de trabalho geral da equipe de desenvolvimento.

Assista ao vídeo à seguir:

Vídeo *7 editores incríveis para programar online que você deveria experimentar*, do car V. Disponível em: https://youtu.be/40QyA5f3Qo0. Acesso em: 2 fev. 2024.

Depois de assistir ao vídeo, responda:

- Quais pontos positivos das IDEs on-line que você aproveita?
- Quais são os prós e contras que você conseguiu observar em trabalhar somente com IDEs on-line?

Ferramentas de desenvolvimento:

Durante a sua jornada inicial com Python no Google Colab, você pode se deparar com **diferentes ferramentas de desenvolvimento**. Entretanto, é essencial entender o funcionamento de outras plataformas que os desenvolvedores utilizam para projetos mais extensos ou específicos.

Existem várias IDEs populares que oferecem ambientes robustos para o desenvolvimento de *software*, a sua escolha pode depender:

- do tipo de projeto;
- da linguagem de programação;
- das preferências da equipe.

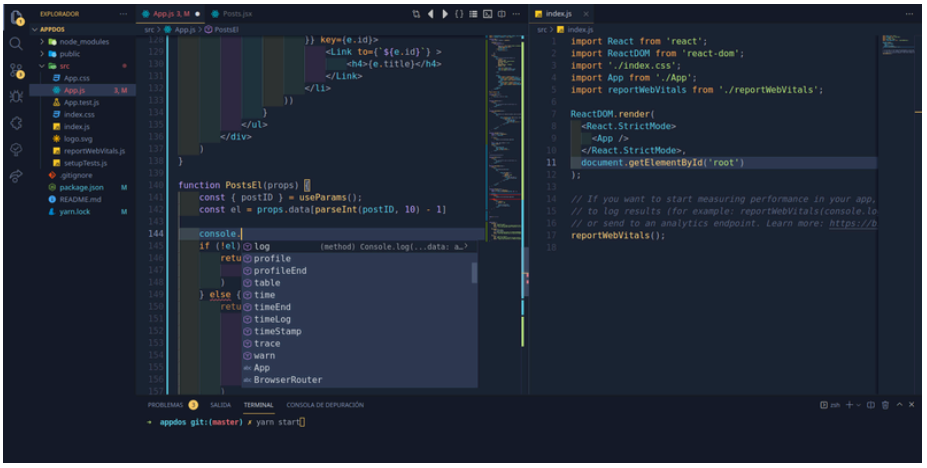
Abaixo estão alguns dos principais IDEs:

IDE	Linguagem suportada	Empresas conhecidas que a utilizam
IntelliJ IDEA	Focado em Java, mas também suporta outras linguagens.	JetBrains, Google, Airbnb.
Visual Studio (VS)	Oferece suporte a várias linguagens, incluindo Python.	Microsoft, Stack Overflow, Accenture.
PyCharm	É especializado em desenvolvimento de Python.	Spotify, Twitter e Pinterest.
Visual Studio Code (VSCode)	Embora seja mais um editor de código, o VSCode é altamente extensível e amplamente utilizado para desenvolvimento em várias linguagens.	Microsoft, Facebook e Slack.

Fonte: elaborado pelo autor (2024)

Ferramentas híbridas e editores de código

Além dos IDEs, há também ferramentas híbridas e editores de código que proporcionam flexibilidade e desempenho ágil. Um exemplo notável é o Visual Studio Code (VSCode), um editor de código altamente extensível e amplamente adotado para o desenvolvimento.



Visual Studio Code em execução no Ubuntu com complementos personalizados. Disponível em: http://tinyurl.com/3t3tyjtm. Acesso em: 30 jan. 2024.

Ao explorar essas ferramentas, lembre-se de que cada uma tem as suas próprias vantagens e situações ideais de uso. A escolha da plataforma certa dependerá das

necessidades do seu projeto, preferências da sua equipe e linguagem de programação utilizada.

Vantagens e desvantagens dos editores de código *on-line*

✔ Vantagens

- **Acessibilidade global:** os editores de código *on-line* permitem que desenvolvedores acessem e trabalhem em projetos de qualquer lugar, promovendo a colaboração remota;
- **Facilidade de compartilhamento:** compartilhar código é simples e direto, seja para fins de revisão de código, suporte técnico ou colaboração em projetos *open source*;
- **Ambiente isolado:** ambientes pré-configurados fornecem uma experiência de desenvolvimento consistente, evitando problemas de compatibilidade e configurações locais complexas;
- **Agilidade no desenvolvimento:** início rápido de projetos sem a necessidade de configurações locais extensas, resultando em maior agilidade no desenvolvimento.

✖ Desvantagens

- **Dependência de conexão à internet:** a necessidade de uma conexão estável à internet pode ser um desafio em ambientes onde a conectividade é limitada;
- **Segurança:** questões de segurança podem surgir ao compartilhar código sensível em ambientes *on-line*. É crucial escolher plataformas confiáveis e entender as políticas de segurança;
- **Limitações de personalização:** algumas ferramentas podem ser limitadas em termos de personalização se comparadas com ambientes locais.

Diferenças entre IDEs e editores de código

Agora que já exploramos um pouco sobre os IDES e os editores de código, chegou a hora de conhecer as suas **principais diferenças**.

Um IDE é como uma "suíte completa" para programadores: ele combina várias

funcionalidades essenciais em um único ambiente. Imagine como se fosse um *kit* com todas as ferramentas necessárias para construir uma casa. Já os editores de código são mais **leves** e **focados**, principalmente na edição do código fonte. São como ferramentas específicas, como uma chave de fenda personalizada para certos tipos de tarefas.

Veja a tabela a seguir:

Diferenças	IDE	Editores de código
Complexidade e recursos	Ambientes mais robustos, que oferecem uma gama maior de funcionalidades para desenvolvimento, como depuradores, compiladores, gerenciadores de projeto, entre outros. **São ideais para projetos grandes e complexos**.	São mais leves e centrados na edição de texto, frequentemente disponibilizando extensões para agregar funcionalidades específicas. **São especialmente indicados para projetos menores ou para aqueles que valorizam a simplicidade**.
Facilidade de uso	Podem ter uma curva de aprendizado mais íngreme devido à grande quantidade de recursos. São ideais para desenvolvedores que lidam com projetos complexos e precisam de todas as ferramentas integradas.	Tendem a ter uma curva de aprendizado mais suave e são frequentemente escolhidos por desenvolvedores que preferem simplicidade e personalização.
Performance	Podem consumir mais recursos do sistema devido à sua natureza integrada e às várias funcionalidades em execução.	Geralmente, têm uma pegada mais leve e podem ser mais eficientes em termos de recursos.
Finalidade	Projetados para oferecer um ambiente completo de desenvolvimento, incluindo *design*, construção e depuração.	Voltados principalmente para a edição e organização de código-fonte, deixando outras tarefas sob a responsabilidade de ferramentas externas ou *scripts*.

Além do que você aprendeu até aqui, é importante ficar ligado nos seguintes tópicos na hora de escolher entre um **IDE** ou um **editor de código**:

✓ Visão integrada e funcionalidades essenciais

Os Ambientes de Desenvolvimento Integrado (IDEs) proporcionam uma visão abrangente do processo de desenvolvimento, unificando não apenas a escrita de código, mas também ferramentas essenciais para compilação, depuração e gerenciamento de projetos. Proporcionam uma experiência fluida, integrando de maneira coesa cada etapa do ciclo de desenvolvimento.

✓ O mundo diversificado dos IDEs

Dentre as opções disponíveis, destacam-se o PyCharm e o Eclipse, cada um com as suas características particulares. O PyCharm, da JetBrains, é reconhecido por sua especialização em Python, proporcionando uma experiência dedicada e otimizada para os desenvolvedores desta linguagem. Já o Eclipse, é uma ferramenta de código aberto, que oferece flexibilidade e suporte para diversas linguagens de programação, tornando-se uma escolha versátil.

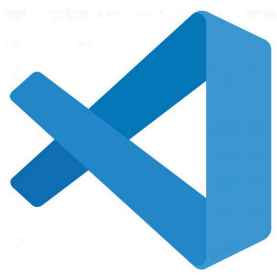
✓ Explorando o VSCode: o IDE de código aberto

O Visual Studio Code (VSCode) se destaca como uma opção popular e de código aberto no mundo do desenvolvimento. Criado pela Microsoft, o VSCode é conhecido por sua leveza, capacidade de personalização e comunidade ativa. Sua flexibilidade e suporte a diversas extensões o tornam ainda mais atraente para os desenvolvedores modernos.

Fonte: elaborado pelo autor (2024).

2.2 Desvendando os segredos do Visual Studio Code

Em 2015, a Microsoft lançou uma jóia da engenharia de *software*: o Visual Studio Code. Uma resposta direta à crescente necessidade de uma ferramenta leve, poderosa e multiplataforma para desenvolvedores. O VSCode rapidamente conquistou corações e mentes, tornando-se um pilar na caixa de ferramentas de desenvolvimento.



Logo do Visual Studio Code. Disponível em: logowik-http://tinyurl.com/3hzuust3. Acesso em: 30 jan. 2024.

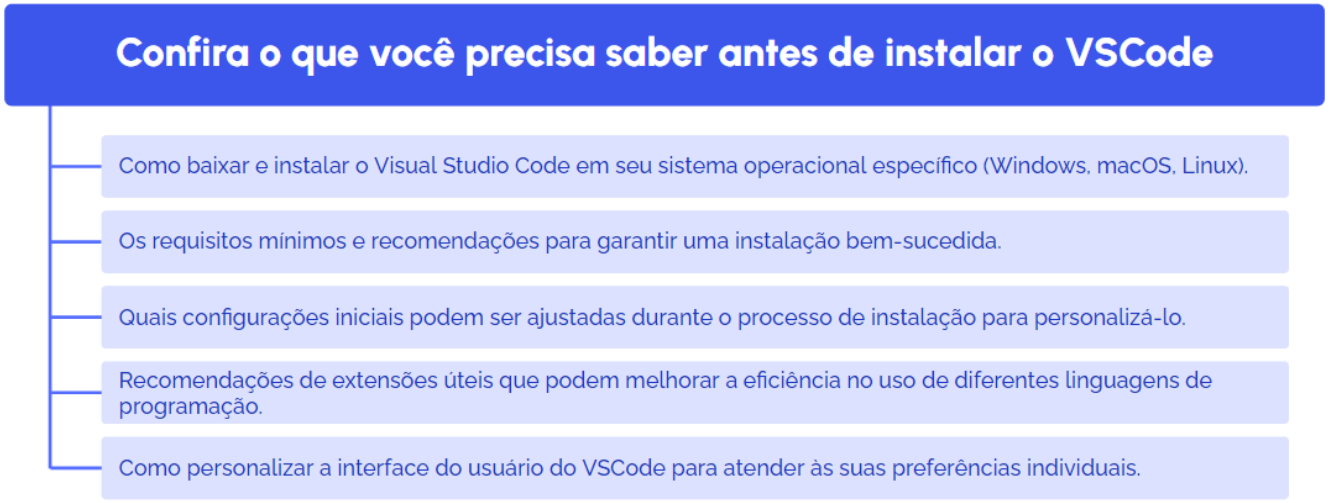
Ao longo dos anos, o VSCode não apenas acompanhou a evolução das

linguagens de programação, mas liderou o caminho. Ele é conhecido pelo seu suporte robusto para diversas linguagens, além da sua integração com Git, depuração avançada e comunidade ativa.

Diferente dos IDEs pesados, o VSCode optou por uma abordagem mais **leve**, mantendo a agilidade sem sacrificar a funcionalidade. A capacidade de personalização extrema, desde temas até a disposição dos painéis, permite uma **experiência única** para cada desenvolvedor.

Com ele, é possível não apenas escrever, editar e modificar o código-fonte, mas também colaborar em projetos de forma conjunta, permitindo que várias pessoas editem o código simultaneamente e facilitando a colaboração remota. Além disso, ele oferece suporte para o desenvolvimento em diversas linguagens de programação, aproveitando a funcionalidade do VSCode Online em várias linguagens.

Vamos nos **aprofundar** nesta plataforma? Para isso, será necessário que você **instale** o programa em seu computador.

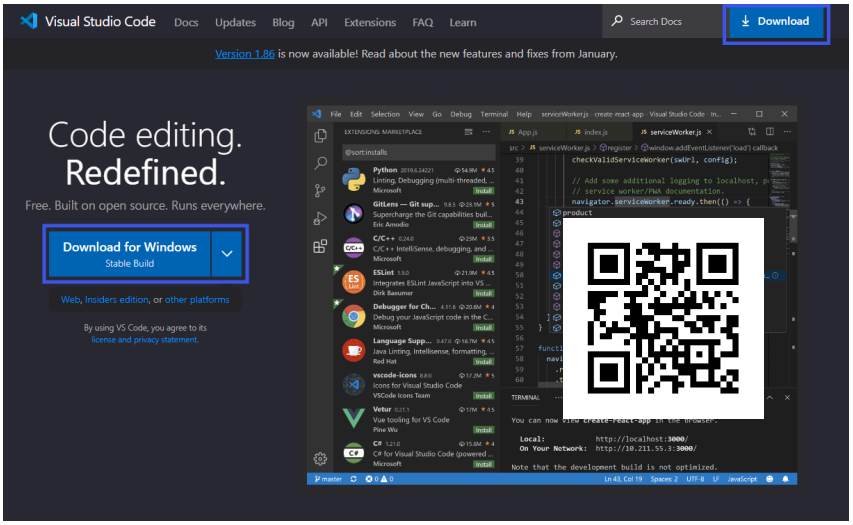


Fonte: elaborado pelo autor (2024).

Você pode baixá-lo seguindo as próximas etapas descritas ou o tutorial no YouTube:

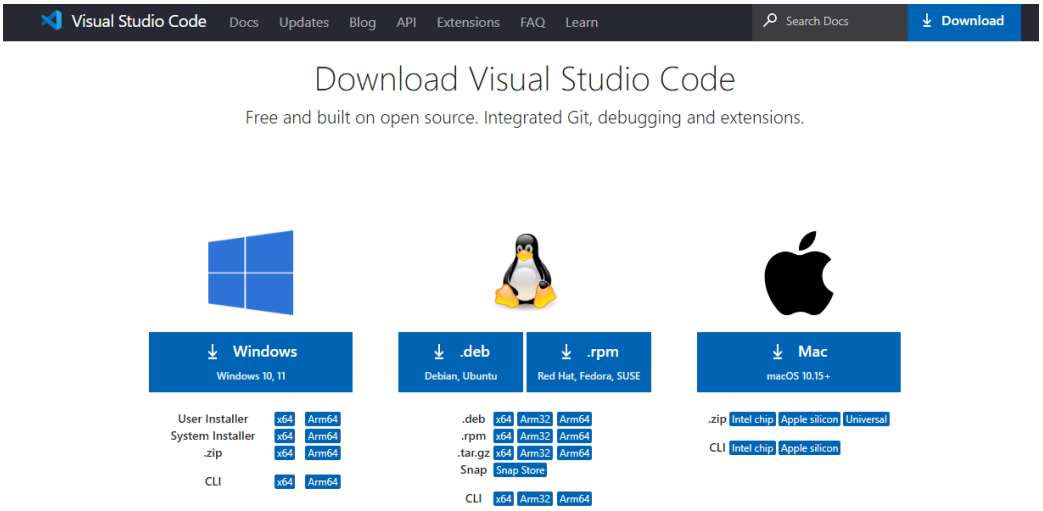
Fonte: Vídeo *Como baixar e instalar o Visual Studio Code (VS Code) - 2023*, do canal Fotografia Nunes Almeida. Disponível em: https://youtu.be/FWnZBah0WLc. Acesso em: 2 fev. 2024

Para realizar o download, é necessário que você acesse o *site* oficial no QR Code a seguir e baixe o programa de acordo com o sistema operacional do seu dispositivo. Veja a captura de tela:



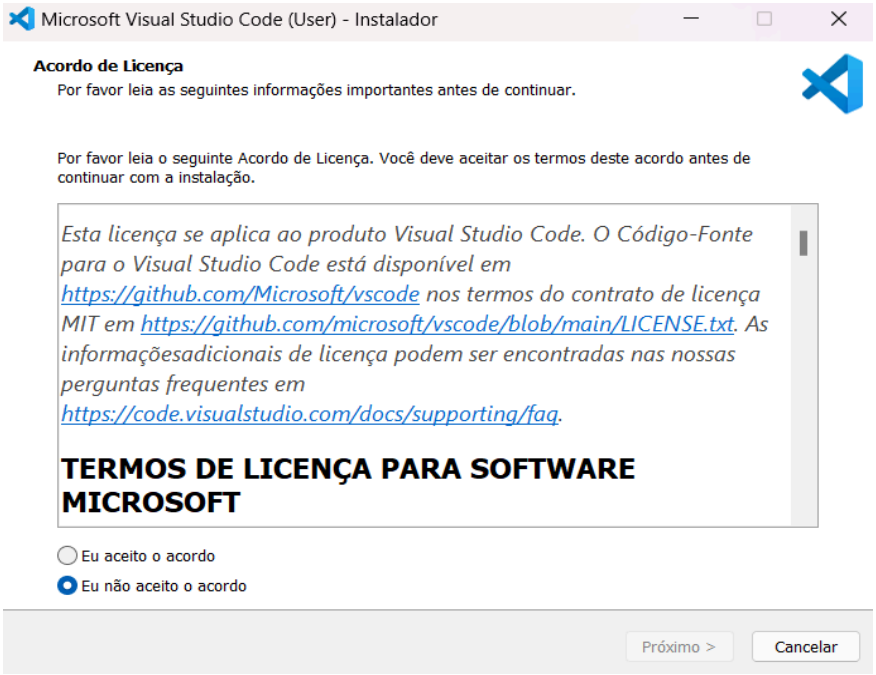
Fonte: elaborado pelo autor (2024).

Caso o seu sistema operacional seja o Windows, também é possível visualizar um botão de download direto. Ao acessar o botão de download, você pode baixar o arquivo para diferentes sistemas operacionais.



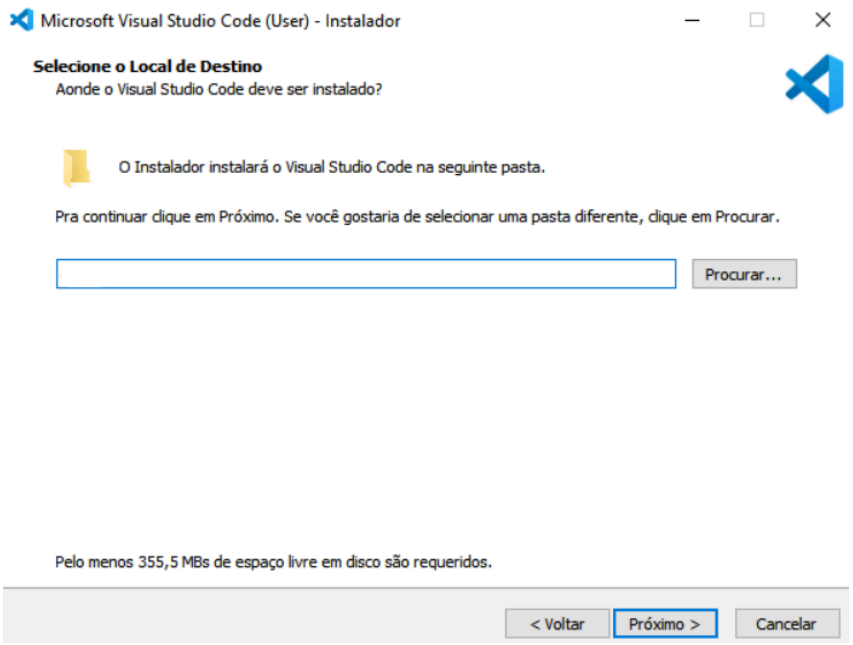
Fonte: elaborado pelo autor (2024).

Após concluir o *download*, abra o arquivo para iniciar o processo de instalação. Para prosseguir, será necessário concordar com os termos de uso.



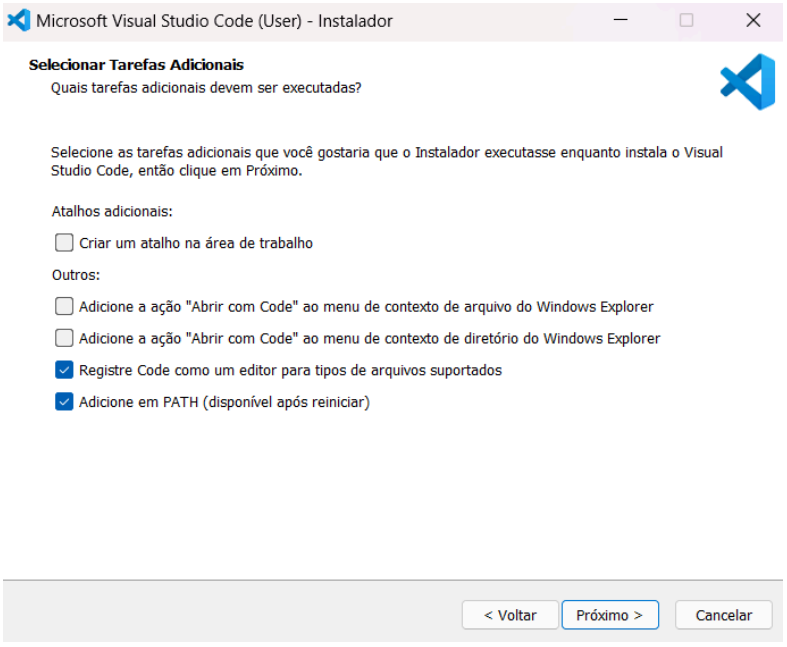
Fonte: elaborado pelo autor (2024).

Em seguida, escolha a pasta em seu computador onde deseja instalar o Visual Studio Code e selecione o diretório de instalação do Visual Studio Code em seu computador.



Fonte: elaborado pelo autor (2024).

Agora, **preste atenção em um ponto importante**: selecione a opção "Adicionar ao PATH". Isso é necessário para que o *software* esteja disponível nas variáveis de ambiente. Copie as configurações do exemplo abaixo para garantir o funcionamento adequado.

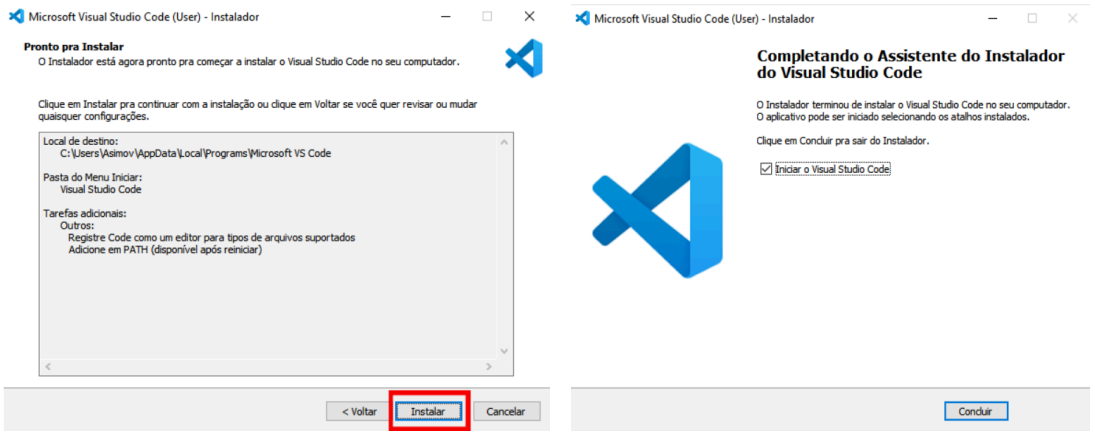


Configurações adicionais do Visual Studio Code que precisam ser selecionadas.

Fonte: elaborado pelo autor (2024).

O próximo passo é revisar se todas as configurações estão corretas. Se estiverem, clique em "Instalar". Caso precise fazer ajustes, clique em "Voltar".

Após a instalação, se tudo ocorrer conforme esperado, clique em "Concluir" para abrir o editor.

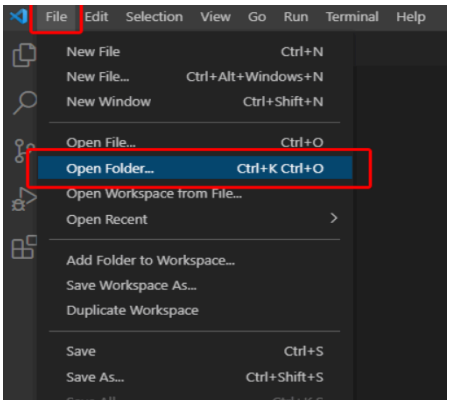


Fonte: elaborado pelo autor (2024).

Configurando o primeiro projeto no VSCODE

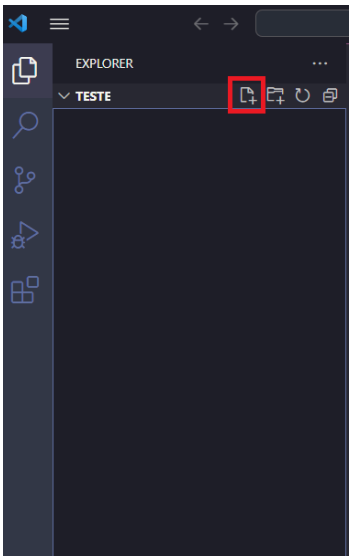
Após instalar e configurar o VS Code, você está pronto para criar o seu primeiro projeto. Vamos lá?

O primeiro passo é criar uma pasta em um local de fácil acesso no seu computador. Como no exemplo abaixo, dê um nome à sua pasta para armazenar os nossos códigos. No VS Code, vá em "File" → "Open Folder" e selecione a pasta criada anteriormente, assim como demonstrado na imagem.



Fonte: elaborado pelo autor (2024).

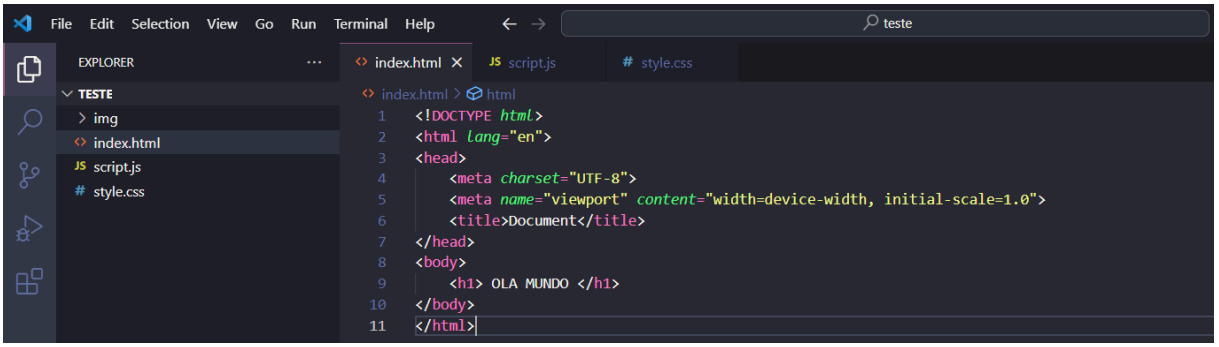
Para criar os primeiros arquivos, clique em "New File" (ícone à esquerda nas opções da pasta) na aba do Explorer ou acesse "File" → "New File" e crie três arquivos com os seguintes nomes: `index.html`, `style.css` e `script.js`. Os arquivos serão exibidos no Explorer, à esquerda do programa, onde você poderá manipulá-los juntamente com outros arquivos do projeto.



Fonte: elaborado pelo autor (2024).

Após a criação, o arquivo estará visível na tela principal do VS Code.

Exemplo de arquivo na tela principal:



Fonte: elaborado pelo autor (2024).

Fique ligado nos principais atalhos do VSCode:

Atalhos do VSCode:

Ctrl + P (Cmd + P no macOS): abrir uma paleta de comandos.

Ctrl + Shift + N (Cmd + Shift + N): abrir uma nova janela.

Ctrl + Shift + P (Cmd + Shift + P): abrir a paleta de comandos.

Ctrl + ` (Cmd + `): abrir o terminal integrado.

Ctrl + B (Cmd + B): alternar a visibilidade da barra lateral.

Ctrl + W (Cmd + W): fechar a janela do editor ativo.

Ctrl + Shift + W (Cmd + Shift + W): fechar a janela do VSCode.

Ctrl + K Ctrl + W (Cmd + K Cmd + W): fechar todas as janelas do editor.

Ctrl + K Ctrl + S (Cmd + K Cmd + S): teclas de atalho.

Ctrl + F4 (Cmd + F4): fechar o editor.

Ctrl + K Ctrl + C (Cmd + K Cmd + C): adicionar comentários no código.

Fonte: elaborado pelo autor (2024).

Integração do JavaScript, HTML e CSS no Visual Studio Code

A integração eficiente entre JavaScript, HTML e CSS é essencial para um desenvolvimento *front-end* coeso. Nesta seção, exploraremos as melhores práticas para criar, vincular e otimizar esses arquivos no ambiente do Visual Studio Code (VSCode).

Ao iniciar um projeto *front-end*, é crucial que os arquivos estejam organizados de maneira estruturada. No VSCode, podemos facilmente criar arquivos HTML, CSS e JavaScript.

Ao pressionar "!", obtemos toda a base necessária para a construção de projetos em HTML. Este é um atalho especialmente útil no Visual Studio Code, que agiliza o processo de criação de código HTML4.

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

</body>
</html>
```

Fonte: elaborado pelo autor (2024).

Geralmente, os arquivos são nomeados dessa forma no Visual Studio Code:

Fonte: elaborado pelo autor (2024).

No contexto do desenvolvimento *web*, determinados arquivos ocupam diferentes funções. Confira:

- o arquivo "index.html" representa a estrutura editável em HTML, onde a marcação e o conteúdo da página são definidos;
- "script.js" desempenha o papel de armazenar todas as funções escritas em JavaScript, proporcionando a lógica e a interatividade dinâmica para o *site*;
- o "style.css" é responsável por consolidar o estilo do código, abrangendo elementos como cores, margens, fontes e outras características visuais, conferindo uma apresentação estética e coesa à página da *web*.

Esses três arquivos, em conjunto, desempenham papéis fundamentais na construção e no *design* de uma experiência visualmente atraente e funcional para o seu *site*.

```
<!DOCTYPE html>
<html lang="en">
  <!-- É fundamental ressaltar que os arquivos devem ser linkados ao HTML da seguinte forma: -->

<!-- No modelo à esquerda, temos a tag "link", que irá conectar o HTML ao arquivo CSS.
     O item em verde, "href", fornece a referência do arquivo a ser buscado.
     Por padrão, ele vem vazio, ex: href"".
     Dentro dessas aspas, você pode usar o atalho "Ctrl + Espaço" para visualizar os arquivos já criados em seu workspace. -->
<link rel="stylesheet" href="style.css">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>

<!-- Aqui importamos o arquivo script de forma semelhante ao CSS, mas usando a tag "script" para buscar o arquivo de origem. -->
<script src="script.js"></script>
</html>
```

Fonte: elaborado pelo autor (2024).

Vinculando arquivos no VSCode

Após criar os arquivos, é essencial vinculá-los corretamente para garantir que o

HTML tenha acesso aos estilos e funcionalidades definidos no CSS e JavaScript, respectivamente.

- CSS: no arquivo HTML, utilize a *tag* <link> dentro da seção <head> para vincular o arquivo CSS, garantindo uma aplicação consistente de estilos.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <!-- essa é a maneira onde vamos linkar o css ao html fazendo seu estilo ser importado,
    aprender sobre caminhos no html é algo muito importante então nunca se esqueça que nomes devem ser exatamente iguais
    para ligar os arquivos, alem disso caso precise voltar uma pagina utilize ../ '-->
    <title>Seu Título</title>
</head>
<body>

<!-- nesse espaço colocamos o conteudo visual ( um exemplo classico e escrever atraves de um H1 o famoso ola mundo! ) -->

<h1> ola mundo </h1>

</body>
</html>
```

Fonte: elaborado pelo autor (2024).

- JavaScript: coloque a *tag* <script> antes do fechamento da *tag* <body> ou no final do arquivo HTML para garantir que o código JavaScript seja carregado após o HTML.

```html
<!-- Antes do fechamento da tag body -->
<script src="script.js"></script>
</body>
</html>

<!-- No final do arquivo HTML -->
<script src="script.js"></script>
```

Fonte: elaborado pelo autor (2024).

Dicas para uma integração eficiente

- Ao vincular arquivos, use caminhos relativos para evitar problemas ao mover o projeto para diferentes diretórios;
- Divida o CSS e o JavaScript em módulos ou pastas para facilitar a manutenção;
- Para uma experiência de desenvolvimento mais dinâmica, considere a extensão Live Server no VSCode. Isso permitirá atualizações automáticas no navegador conforme você edita os arquivos;
- Explore extensões no Marketplace do VSCode que ofereçam suporte específico

para HTML, CSS e JavaScript, facilitando a detecção de erros e a sugestão de código.

Vamos entender o propósito de cada uma dessas *tags* HTML:

<head>

A *tag* <head> é uma seção do HTML que contém informações sobre o documento, mas não é exibida diretamente na página. Ela pode incluir elementos como metadados, *links* para arquivos externos e configurações diversas.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemplo de Página HTML</title>
    <link rel="stylesheet" href="style.css">
    <!-- Outros elementos da head podem ser adicionados conforme necessário -->
</head>
```

Fonte: elaborado pelo autor (2024).

<meta>

É usada para fornecer metadados sobre o documento HTML. Ela não possui um conteúdo visível na página, mas desempenha um papel crucial em aspectos como codificação de caracteres, descrição da página para motores de busca, configuração da *viewport* para dispositivos móveis, entre outros.

```
<!-- Define o conjunto de caracteres para a página -->
<meta charset="UTF-8">

<!-- Configura a largura da tela e a escala inicial para dispositivos móveis -->
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- Descrição da página para motores de busca -->
<meta name="description" content="Descrição da página aqui">

<!-- Palavras-chave relacionadas à página para motores de busca -->
<meta name="keywords" content="palavra-chave1, palavra-chave2, palavra-chave3">

<!-- Autor da página -->
<meta name="author" content="Nome do Autor">

<!-- Instrui os motores de busca a indexarem e seguirem os links na página -->
<meta name="robots" content="index, follow">

<!-- Atualiza automaticamente a página após 5 segundos, redirecionando para https://example.com -->
<meta http-equiv="refresh" content="5;url=https://example.com">
```

Fonte: elaborado pelo autor (2024).

Exemplo de uso para definir a codificação de caracteres e a *viewport*:

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Fonte: elaborado pelo autor (2024).

<link>

É usada para vincular recursos externos ao documento HTML. Isso inclui folhas de estilo (CSS), ícones (favicon), fontes e até mesmo feeds RSS. Um uso comum é vincular um arquivo CSS para aplicar estilos à página.

Exemplo de uso para vincular uma folha de estilo CSS:

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

Fonte: elaborado pelo autor (2024).

<title>

É usada para definir o título da página, que geralmente é exibido na barra de título do navegador. É uma parte fundamental para melhorar a acessibilidade e fornecer informações contextuais sobre o conteúdo da página.

Exemplo de uso para definir o título da página:

```
<title>Meu Site Incrível</title>
```

Fonte: elaborado pelo autor (2024).

<body>

Envolve todo o conteúdo visível da página HTML. É nessa seção que você coloca textos, imagens, *links*, formulários, *scripts* etc. O conteúdo dentro da *tag* <body> é o que será exibido na página quando ela for acessada por um navegador.

Exemplo básico de estrutura de uma página HTML com a *tag* <body>:

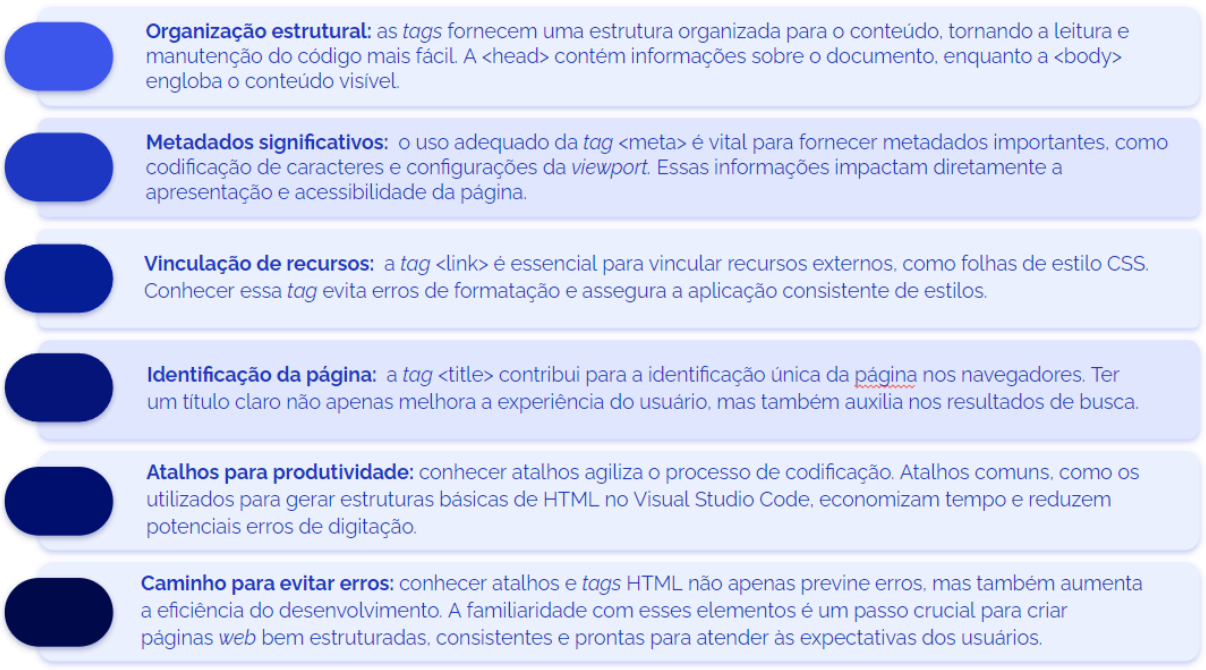
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Minha Página</title>
</head>
<body>
    <h1>Olá, Mundo!</h1>
    <p>Este é o conteúdo da minha página.</p>
</body>
</html>
```

Fonte: Elaborado pelo autor (2024).

Evitando erros e melhorando a eficiência

Entender os atalhos e *tags* HTML é essencial para desenvolvedores *front-end*, pois facilita a criação de páginas *web* eficientes e sem erros. Conhecer os elementos fundamentais, como <head>, <meta>, <link>, <title>, e <body>, proporciona uma base sólida para estruturar e configurar documentos HTML.

Veja as principais dicas para **não errar** na hora de trabalhar no VSCode:



Fonte: Elaborado pelo autor (2024).

Ao longo dessa jornada, você adquiriu conhecimentos sobre o uso do VSCode para assegurar uma experiência positiva na elaboração dos seus projetos. Com esta ferramenta, é possível simplificar tarefas, integrar-se a sistemas de controle de versão e personalizar o ambiente de desenvolvimento conforme suas preferências. Estas habilidades são essenciais para aprimorar a eficiência e tornar o processo de desenvolvimento mais efetivo.

E aí, preparado para colocar todo o seu conhecimento em prática? Resolva o desafio a seguir!

DESAFIO PRÁTICO

Desenvolvimento *web* e Visual Studio Code (VSCode)

Desafio: A jornada do desenvolvedor versátil

Descrição

Você faz parte de uma *start-up* de tecnologia especializada em soluções de gerenciamento de projetos. Durante o projeto, a equipe enfrenta desafios na eficiência do desenvolvimento de *software*. Atualmente, os desenvolvedores utilizam diferentes IDEs e editores de texto, o que leva a inconsistências no código, além de dificuldades na colaboração e na manutenção do código.

Agora, você deve implementar o Visual Studio Code (VS Code) como a principal ferramenta de desenvolvimento, visando melhorar a colaboração, padronizar o ambiente de desenvolvimento e aumentar a eficiência da equipe. Sabendo que a linguagem utilizada pela empresa é a CSS, será que você está preparado para aceitar o desafio?

Objetivos

- **Padronizar o ambiente:**
    - padronizar a configuração do VS Code para a linguagem CSS, incluindo extensões e configurações recomendadas;
    - criar arquivos de configuração compartilhados para facilitar a replicação do ambiente em novas máquinas.

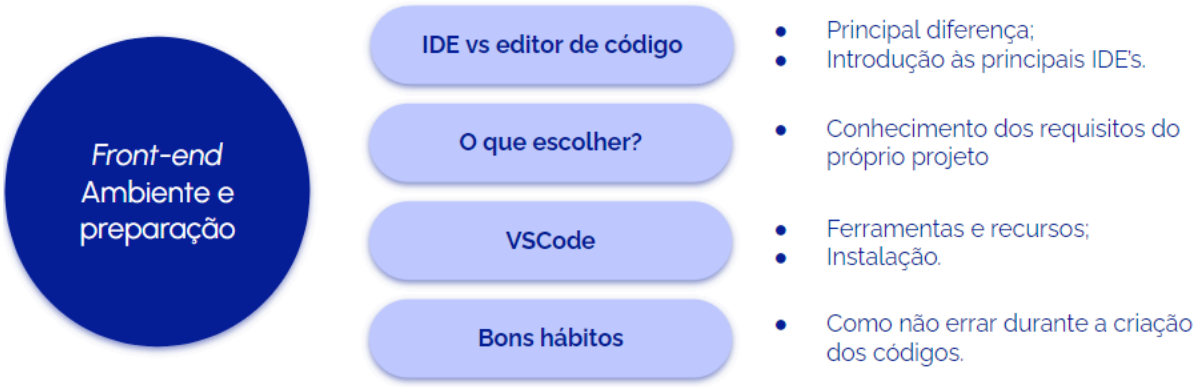
- **Promover treinamento e suporte:**
    - realizar treinamentos para a equipe acerca das funcionalidades e melhores práticas do VS Code;
    - estabelecer um sistema de suporte para ajudar os desenvolvedores na transição e solucionar problemas relacionados ao VS Code.

Orientações

- ➔ Para a configuração do VSCode, consulte a documentação oficial VSCode Setup, disponível em: https://code.visualstudio.com/docs/setup/setup-overview;
- ➔ Pesquisa ativamente em fóruns e comunidades ***on-line*** para resolver desafios específicos enfrentados durante a configuração e operação do VSCode.

Durante este material didático, foi possível aprender de maneira abrangente sobre as nuances fundamentais entre Ambientes de Desenvolvimento Integrado (IDEs) e editores de código, ambos elementos essenciais para os desenvolvedores modernos.

Alguns conceitos fundamentais foram explorados:



Fonte: elaborado pelo autor (2024).

ATIVIDADE DE FIXAÇÃO

1. Quais são as principais diferenças entre um IDE e um editor de código? Em quais situações você escolheria um em vez do outro?
2. Liste três vantagens do VSCode em comparação com outras ferramentas de desenvolvimento.
3. Descreva o processo de instalação do VSCode em seu sistema operacional específico.
4. Quais extensões do VSCode você considera essenciais para o desenvolvimento *web*?
5. Como você configura o VSCode para integrar-se a um repositório CSS local?
6. Explique como personalizar as configurações do VSCode para atender às suas preferências de desenvolvimento.
7. Qual é a função da barra de *status* no VSCode, e como ela pode auxiliar durante o desenvolvimento?
8. Quais são as etapas para depurar um programa usando as ferramentas integradas do VSCode?
9. Como você abriria múltiplos terminais no VSCode e por que isso pode ser útil?
10. Descreva um cenário em que você escolheria um IDE específico em vez do VSCode para um projeto.

CAPÍTULO 03

Controle de versão Git e Github

O que esperar deste capítulo:

- Compreender o conceito de **controle de versão** como uma prática para documentar um histórico de trabalho e facilitar a organização em projetos colaborativos;
- Utilizar os comandos fundamentais do Git para colaborar de forma independente com repositórios locais.

3.1 O que é controle de versão?

Vamos imaginar que você está trabalhando em um projeto de escrita colaborativa com uma amiga chamada Ana. Vocês estão criando um documento em um processador de texto, e há várias versões sendo editadas ao mesmo tempo. Aqui está um exemplo simples para ilustrar o conceito de controle de versão:

Disponível em: freepik-http://tinyurl.com/3mw6rpau. Acesso em: 30 jan. 2024.

Sem controle de versão	Com controle de versão
• Vocês compartilham o documento por *e-mail* ou mensagens instantâneas; • Ana faz algumas alterações e envia o documento de volta para você; • Agora, você tem duas versões diferentes do documento - a sua original e a versão editada por Ana; • Se cada um continuar editando de forma independente, pode se tornar desafiador controlar as alterações, resultando em conflitos inesperados.	• Vocês decidem usar um sistema de controle de versão, como o Git (uma ferramenta comumente usada para este fim); • Inicialmente, vocês criam uma "cópia de trabalho" do documento (um repositório); • Ana faz alterações e "commita" (registra) essas mudanças (criando uma versão identificada); • Você pode, então, "atualizar" sua cópia de trabalho para incluir as alterações de Ana; • Em caso de conflito, como quando ambos tentam editar a mesma parte do documento, o sistema de controle de versão auxilia na gestão e resolução organizada desses conflitos.

Como você pode perceber, o controle de versão é um sistema que permite rastrear as alterações feitas nos arquivos de um projeto ao longo do tempo. Ele fornece um histórico detalhado das modificações, permitindo que desenvolvedores colaborem de maneira eficiente e mantenham a integridade do código, o que facilita a identificação e resolução de problemas.

O Git, lançado em 2005 por Linus Torvalds, é a magia por trás desse processo. Ele foi projetado para ser rápido, eficiente e capaz de lidar com projetos de grande escala, como o **kernel** do Linux. Sua abordagem distribuída, que permite a cada desenvolvedor ter uma cópia completa do histórico do projeto, proporciona flexibilidade e robustez.

Disponível em: http://tinyurl.com/34rhfacb. Acesso em: 30 jan. 2024.

O GitHub, por sua vez, é uma plataforma *web* baseada em Git que oferece recursos colaborativos e sociais para projetos de *software*. Nele, os desenvolvedores

podem hospedar seus repositórios, facilitando o compartilhamento de código e a colaboração entre equipes distribuídas globalmente. Além disso, o GitHub oferece funcionalidades como ***pull requests***, que simplificam o processo de revisão de código, e ***issues***, que permitem o rastreamento de tarefas, *bugs* e melhorias.

Quer saber qual é a principal diferença entre o Git e o GitHub? Veja o infográfico a seguir.

Fonte: elaborado pelo autor (2024).

Curiosidade: Por que o símbolo do GitHub é um gato com oito patas?

Na verdade, ele personifica a noção de multitarefa e versatilidade, simbolizando a flexibilidade do GitHub ao lidar com uma variedade de projetos e colaborações. Além disso, o Octocat, como é conhecido, é frequentemente retratado segurando um **garfo**. Dentro do contexto do GitHub, um "*fork*" (garfo, em inglês) representa uma cópia de um repositório, evidenciando a capacidade dos desenvolvedores de criar cópias de projetos existentes para contribuir com suas próprias modificações.

Disponível em: http://tinyurl.com/3fcpm7us. Acesso em: 30 jan. 2024.

Principais recursos do GitHub:

Termo	Descrição
Repositório	Espaço no GitHub onde o código de um projeto é armazenado. Cada projeto, geralmente, tem o seu próprio repositório.
Commits	Alterações ou conjunto de alterações no código, cada uma acompanhada por uma mensagem descritiva que oferece contexto.
Branches	Ramificações do código principal que permitem desenvolvimento paralelo de recursos sem interferir no código principal.
***Pull Request* (PR)**	Propostas de alteração que permitem sugerir modificações, iniciar discussões e solicitar a mesclagem no código principal.
Colaboração	Outras pessoas podem contribuir para o seu projeto, adicionando novos recursos, corrigindo bugs ou fazendo melhorias. Cada colaborador pode ter a sua própria cópia do projeto, mas todos trabalham juntos no mesmo código.
Segurança e *backup*	Por estar na nuvem, o seu projeto é protegido contra perdas acidentais. Se algo acontecer ao seu computador, o seu código ainda estará seguro no GitHub.

Benefícios e desvantagens de utilizar o GitHub

Utilizar o GitHub traz uma série de benefícios que tornam o desenvolvimento de *software* mais eficiente e colaborativo. Confira a seguir:

Benefícios de usar o GitHub

Colaboração facilitada: permite que a equipe trabalhe simultaneamente no mesmo projeto. Isso significa que todos podem contribuir e interagir.

***Pull Requests* para discussão:** é como enviar um convite à equipe para examinar suas contribuições. Isso permite a discussão e sugestão de melhorias antes de incorporar as suas alterações.

Rastreamento de mudanças: registra cada alteração feita no projeto. Caso algo dê errado, você pode retroceder e corrigir.

***Branches* para experimentação:** você pode criar "ramificações" para trabalhar em novas ideias ou corrigir problemas sem interferir diretamente no projeto principal.

Epicentro para projetos abertos: esse aspecto cria uma comunidade global de desenvolvedores, permitindo a criação de *software* de alta qualidade.

Fonte: elaborado pelo autor (2024).

Embora o GitHub seja uma ferramenta valiosa, também apresenta algumas desvantagens que devem ser consideradas.

Desvantagens de usar o GitHub

Dependência da internet: a colaboração eficiente depende da conectividade à internet, o que pode ser uma limitação em ambientes com acesso restrito.

Curva de aprendizado: para iniciantes, o Git pode ter uma curva de aprendizado maior, especialmente quando se trata de resolver conflitos e entender conceitos avançados.

Limitações em repositórios: contas gratuitas no GitHub têm limitações em relação a repositórios privados. Por isso, projetos maiores podem precisar de planos pagos.

Fonte: elaborado pelo autor (2024).

Empresas que utilizam o GitHub

- **Microsoft**

  A própria Microsoft, proprietária do GitHub desde 2018, utiliza extensivamente a plataforma para diversos projetos, incluindo o desenvolvimento do Visual Studio Code, TypeScript e muitos outros;

- **Facebook**

  O Facebook emprega o GitHub para colaborações em projetos como o React, uma biblioteca JavaScript para a construção de interfaces de usuário;

- **Google**

  A gigante da tecnologia, Google, utiliza o GitHub para inúmeros projetos de código aberto, como o TensorFlow, uma biblioteca de aprendizado de máquina;

- **Netflix**

  A Netflix, líder em *streaming* de conteúdo, adota o GitHub para desenvolvimento colaborativo e rastreamento de alterações em suas aplicações e ferramentas internas;

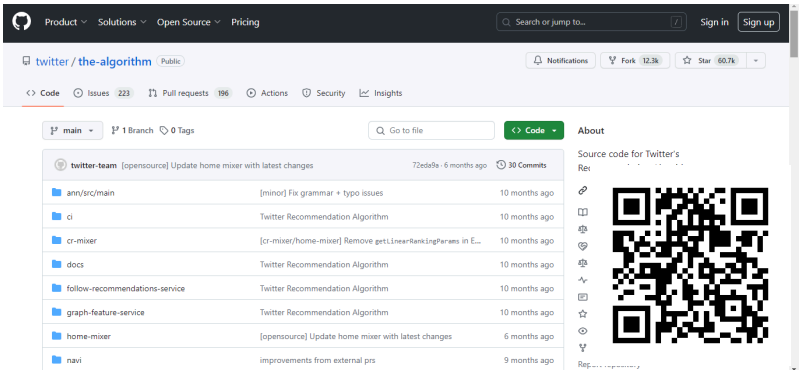
- **Spotify**

  A plataforma de *streaming* de música Spotify utiliza o GitHub para seus projetos, facilitando a colaboração e o desenvolvimento contínuo de novos recursos;

- **Amazon Web Services (AWS)**

  A divisão de serviços em nuvem da Amazon, a AWS, faz amplo uso do GitHub para desenvolvimento colaborativo e para fornecer acesso a ferramentas e SDKs.

O GitHub tornou-se um pilar central no ecossistema de desenvolvimento de *software*, e a sua adoção por empresas líderes evidencia a sua importância para equipes que buscam eficiência, colaboração e transparência em seus processos de desenvolvimento. Ficou curioso para saber como funciona uma **biblioteca compartilhada** no GitHub? Acesse o código aberto do X (antigo Twitter) no *link* ou QR Code à seguir.



Disponível em: http://tinyurl.com/3rvtzbhs. Acesso em: 30 jan. 2024.

Após acessar o código, tente responder:

A. Na sua opinião, quais são os benefícios em utilizar bibliotecas de código aberto, como as oferecidas pelo Twitter, em seus projetos de desenvolvimento?

B. Quais são os principais desafios ou dificuldades que você acredita que os desenvolvedores podem enfrentar ao utilizar a biblioteca de código aberto do Twitter?

Colaboração efetiva com Git e repositórios remotos: um exemplo

Imagine que você está trabalhando em um projeto de grupo, no qual cada membro é responsável por uma parte. Agora, suponha que um dos seus colegas fez algumas alterações no código que estavam sob a sua responsabilidade, e essas mudanças acabaram causando problemas.Fonte: elaborado pelo autor (2024).

Fonte: elaborado pelo autor (2024).

Explorando o GitHub: entendendo o fluxo de trabalho básico

1. Crie um clone de um repositório existente para começar a trabalhar no seu ambiente local.

```
git clone https://github.com/usuario/nome-do-repositorio.git
```

Fonte: elaborado pelo autor (2024).

2. Crie **branches** para isolar o trabalho em novos recursos ou correções de *bugs*, evitando conflitos com o código principal.

```
git branch nome-da-nova-branch
```

Fonte: elaborado pelo autor (2024).

3. Faça **commits** incrementais, cada um com uma mensagem clara e descritiva explicando as mudanças realizadas.

```
git commit -m "Mensagem descritiva aqui"
```

Fonte: elaborado pelo autor (2024).

4. Abra um **Pull Request** para revisão. Os membros da equipe podem revisar o código, fornecer *feedbacks* e discutir alterações antes da mesclagem.

5. Após a revisão bem-sucedida, o **Pull Request** pode ser mesclado no código principal, incorporando as alterações. Certifique-se de manter o seu ambiente local atualizado com as mudanças feitas por outros membros da equipe.

GitHub e hospedagem em nuvem: ampliando as suas fronteiras

O que é hospedagem em nuvem?

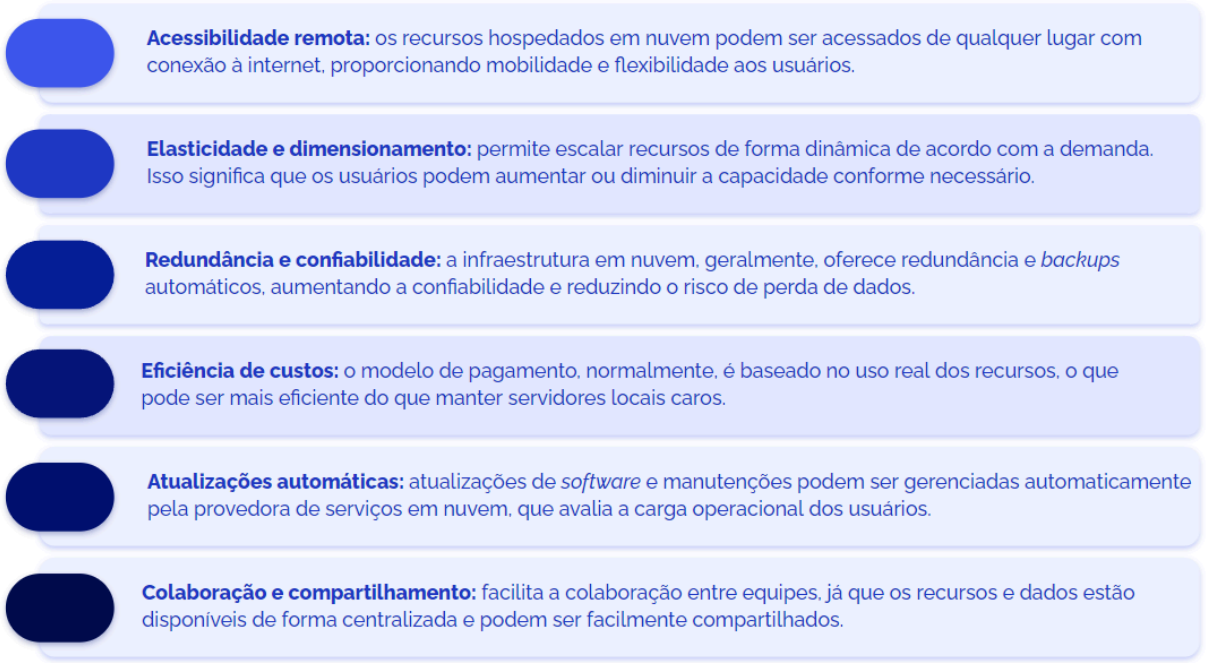
Imagine que você tem um computador em casa. Nele, você armazena fotos, documentos e outros arquivos importantes. Quando você precisa acessar esses arquivos de outro lugar, é possível que tenha dificuldades, porque seu computador está em casa.

Mas existe uma solução para este problema: a hospedagem em nuvem.

Hospedagem em nuvem para controle de versão refere-se à prática de **armazenar, gerenciar e colaborar** em repositórios de controle de versão que estejam em servidores remotos, geralmente fornecidos por serviços em nuvem. Isso oferece várias vantagens em comparação à hospedagem local ou em servidores próprios.

Esse tipo de hospedagem está comumente associado a serviços como Amazon Web Services (AWS), Microsoft Azure ou Google Cloud Platform, nos quais é possível hospedar e executar aplicativos, serviços e infraestrutura de forma escalável na nuvem. Essas plataformas fornecem uma ampla gama de recursos, que incluem máquinas virtuais, armazenamento em nuvem, bancos de dados, entre outros.

Vamos explorar os principais conceitos relacionados à hospedagem em nuvem:



Fonte: elaborado pelo autor (2024).

E aí, conseguiu perceber a importância de incorporar o Git e o GitHub em seus projetos? Com essas ferramentas, é possível gerenciar de maneira eficiente o controle de versão do seu código, o que significa que você pode rastrear todas as alterações feitas ao longo do tempo. Isso não apenas proporciona uma visão clara do histórico de desenvolvimento, mas também concede a capacidade de reverter para versões anteriores do seu projeto, caso seja necessário.

Agora que você aprofundou um pouco mais o seu conhecimento sobre este assunto, vamos praticar?

DESAFIO PRÁTICO

Controle de Versão, Git, GitHub

Desafio: Projeto de Desenvolvimento Colaborativo

Descrição

Você e a sua equipe estão iniciando um novo e ambicioso projeto de desenvolvimento de *software*. Com várias funcionalidades planejadas e uma equipe distribuída, torna-se evidente a necessidade de um sistema eficiente de controle de versão. O Git e o GitHub serão os seus aliados nesta jornada, fornecendo uma base sólida para colaboração e gerenciamento de código.

Objetivos

Configuração inicial:

- criar um repositório vazio no GitHub para o novo projeto;
- clonar o repositório para o seu ambiente local.

Desenvolvimento de funcionalidades:

- cada membro da equipe deve escolher uma funcionalidade específica para desenvolver;
- criar uma **branch** para cada funcionalidade usando o padrão **funcionalidade-nome**.

Colaboração e mesclagem:

- todos os membros da equipe deverão trabalhar simultaneamente em suas funcionalidades;

- cada membro deve realizar um **commit** das alterações em sua **branch** após o desenvolvimento;
- um membro deve ser designado para mesclar as funcionalidades no **branch** principal ("*main*") usando o método de merge.

Resolução de conflitos:

- criar conflitos, deliberadamente, em uma área do código entre duas funcionalidades durante a mesclagem;
- instruir os membros da equipe a resolverem esses conflitos, garantindo uma mesclagem bem-sucedida.

***Tags* e releases:**

- criar uma *tag* para marcar a versão do projeto após a conclusão de uma interação significativa;
- documentar as alterações notáveis em um arquivo "CHANGELOG.md".

Revertendo alterações:

- simular a necessidade de reverter uma alteração específica que causou problemas;
- utilizar o Git para reverter a alteração e criar um **commit**.

Orientações

- ➔ Mantenha a comunicação constante entre os membros da equipe. Utilize questões no GitHub, comentários em **commits** e outras ferramentas colaborativas;
- ➔ Utilize práticas de *branch naming* consistentes para facilitar a identificação das funcionalidades;
- ➔ Certifique-se de que todos compreendam o processo de resolução de conflitos e a importância da comunicação durante esse processo;
- ➔ Utilize o arquivo `CHANGELOG.md` como uma ferramenta de comunicação para destacar as alterações e melhorias realizadas em cada versão.

RESUMO

Nesta seção, abordamos um tema fundamental para o desenvolvimento de *software*: controle de versão e Git. Exploramos conceitos essenciais para desenvolvedores, como a importância de uma estrutura robusta para registrar e gerenciar alterações em projetos, bem como a relevância da coordenação do trabalho em equipe e da facilitação da integração de contribuições.

Veja o mapa mental a seguir com os principais pontos trabalhados neste tema:

Fonte: elaborado pelo autor (2024).

ATIVIDADE DE FIXAÇÃO

1. Utilizar versionamento no desenvolvimento de sistema é fundamental. Explique o porquê dessa afirmativa e cite quais ferramentas podem ser utilizadas.

2. Você acabou de criar um novo arquivo para o seu projeto no Git. Qual comando você deve usar para começar a rastrear as alterações nesse arquivo?

    a. git add
    b. git commit
    c. git init

d. git push

3. Descreva três benefícios específicos ao utilizar o Git como sistema de controle de versão.

4. Você fez algumas alterações em um arquivo, mas deseja desfazer as modificações e retornar ao estado anterior. Qual comando Git você deve usar?

   a. git commit --amend
   b. git reset
   c. git revert
   d. git branch

5. Você deseja verificar as diferenças entre o seu código atual e o último **commit**. Qual comando Git você deve usar?

   a. git diff
   b. it status
   c. git log
   d. git branch

6. Qual é a diferença entre os comandos **git commit** e **git push**? Dê um exemplo de cada.

7. Descreva como criar e trabalhar em um novo **branch** utilizando comandos Git.

8. Você quer enviar as suas alterações locais para o repositório remoto no Github. Qual comando você deve usar?

   a. git pull
   b. git commit
   c. git push
   d. git merge

9. Você trabalha em uma equipe de desenvolvimento e deseja mesclar as alterações de uma ramificação (**branch**) específica em sua ramificação atual. Qual comando você deve utilizar?

    a. git merge
    b. git pull
    c. git branch
    d. git fetch

10. Imagine que você está liderando um projeto de desenvolvimento de *software* com uma equipe de desenvolvedores. Durante o ciclo de desenvolvimento, surge a necessidade de implementar uma nova funcionalidade significativa, que pode impactar partes cruciais do código existente. Explique como a utilização de um sistema de controle de versão, como o Git, e a adoção de boas práticas de versionamento podem ser fundamentais para gerenciar e integrar essa nova funcionalidade de maneira eficiente e segura.

CAPÍTULO 04

Hospedagem de projetos

O que esperar deste capítulo:

- Ter conhecimento sobre plataformas de hospedagem de projetos em nuvem, com ênfase no GitHub;
- Dominar os principais comandos do Git para colaborar de maneira eficaz com equipes de desenvolvedores e repositórios remotos.

4.1 Hospedagem de projetos e suas implicações no desenvolvimento de *software*

Imagine que está escolhendo onde sua equipe vai morar, mas em vez de casas, são locais virtuais para guardar seu código. Esses espaços são como "residências" digitais para os projetos de *software*. Não se trata apenas de um local para armazenar linhas de código, mas de um ambiente onde todos podem colaborar, gerenciar alterações e utilizar ferramentas úteis para impulsionar o crescimento do projeto.

Nessa "casa", todos podem colaborar, fazer alterações no código e usar ferramentas úteis para contribuir com a evolução do projeto.

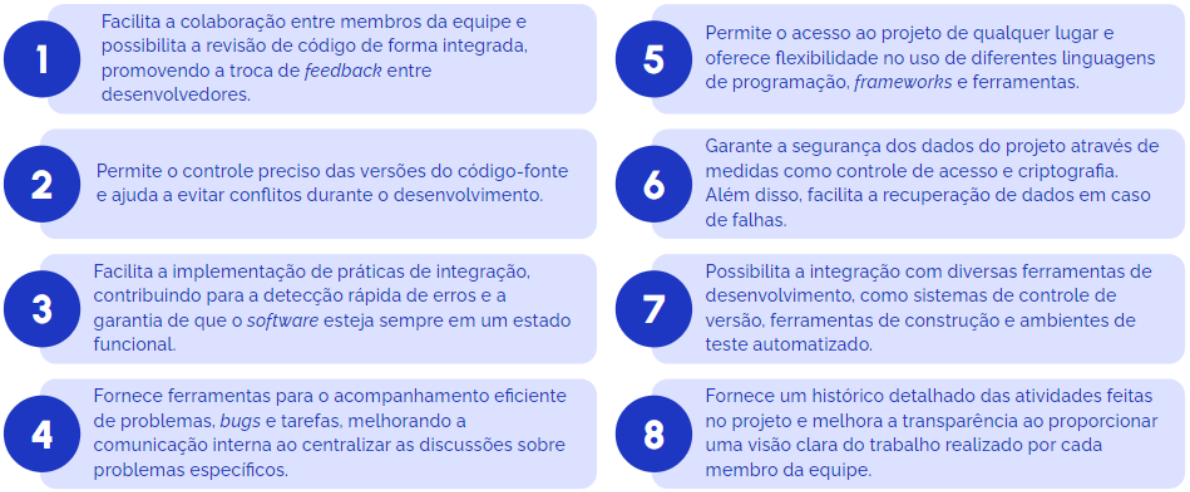
Fonte: elaborado pelo autor (2024).

Assim como escolher uma casa, a seleção da plataforma para hospedar o código é crucial, pois cada uma possui suas próprias características. A decisão não se limita apenas ao gosto pessoal, mas também à estratégia que influenciará a forma como as equipes colaboram e impulsionará o desenvolvimento dos projetos. Grandes empresas de tecnologia optam por plataformas específicas para inovar, demonstrando o impacto significativo dessa escolha.

Vamos explorar esses ambientes digitais, compreender as ferramentas que oferecem e entender como escolhê-los corretamente para fazer toda a diferença no desenvolvimento do projeto. Estão preparados?

Benefícios da hospedagem de projetos

A hospedagem de projetos oferece uma série de benefícios valiosos para equipes de desenvolvimento de *software*, proporcionando um ambiente digital colaborativo e eficiente. Entre as principais vantagens estão:



Fonte: elaborado pelo autor (2024).

Principais ferramentas de hospedagem

GitHub	É uma plataforma na qual você armazena o seu código e pode colaborar com outras pessoas de qualquer lugar do mundo.
	Exemplo: o GitHub é como um "Google Docs" para desenvolvedores. Assim como várias pessoas podem editar um documento simultaneamente no Google Docs, vários desenvolvedores podem contribuir para um projeto ao mesmo tempo no GitHub, independente de onde estejam.
GitLab	Oferece integração contínua, automatizando processos como compilação e teste do código, garantindo que tudo funcione.
	Exemplo: funciona como um "verificador automático". Assim como é possível verificar a ortografia automaticamente ao escrever no Word, o GitLab verifica se o código está funcionando corretamente.
Bitbucket	É uma plataforma que oferece controle de versão centralizado para rastrear e gerenciar alterações em projetos.
	Exemplo: é como se o Bitbucket fosse uma "linha do tempo" para o seu código, onde você pode visualizar todas as mudanças feitas ao longo do tempo, assim como você pode ver diferentes versões de um documento no Dropbox.
Azure DevOps	É uma plataforma de serviços e ferramentas fornecida pela Microsoft para apoiar o ciclo de vida completo do desenvolvimento de *software.*
	Exemplo: semelhante ao Dropbox, que mantém um histórico de versões de documentos, a hospedagem DevOps garante um registro claro e acessível de cada modificação realizada no código-fonte.

Fonte: elaborado pelo autor (2024)

Empresas reconhecidas e suas escolhas

Grandes empresas de tecnologia escolhem cuidadosamente as suas plataformas de hospedagem de projetos: Confira as principais a seguir.

- **GitHub:**
  Microsoft, Facebook, Google e Spotify.
- **GitLab:**
  Alibaba, Siemens e IBM.
- **Bitbucket:**
  Atlassian, Airbnb e Audi.

Explorando a Amazon Web Services (AWS)

Você provavelmente já ouviu falar da plataforma da Amazon, não é mesmo? A empresa é conhecida por sua assistente virtual, a Alexa, que prontamente responde a todas as nossas perguntas, e pelo Prime Vídeo, onde podemos desfrutar de maratonas das nossas séries favoritas.

Mas você sabia que a Amazon também se destaca em outra área? Eles oferecem um serviço de computação em nuvem chamado Amazon Web Services (AWS). A AWS é uma plataforma de serviços em nuvem que oferece uma vasta gama de recursos, incluindo computação, armazenamento, banco de dados, aprendizado de máquina, análise de dados, Internet das Coisas (IoT) e muito mais.

Vamos explorar isso mais profundamente nos tópicos a seguir.

✓ Amazon S3 para armazenamento de objetos

Ao realizar compras *on-line*, você já se perguntou onde todas as imagens de produtos e vídeos ficam armazenados? O Amazon S3 desempenha o papel de um depósito virtual, preservando todos os elementos visuais que compõem o *site*, desde fotografias de produtos até vídeos.

✓ AWS CodeCommit para hospedagem de repositórios Git privados

Se você já usou o Google Docs ou organizou pastas no seu computador para armazenar seus trabalhos, o AWS CodeCommit opera de maneira semelhante, mas é especificamente projetado para códigos. Ele fornece um local confiável para armazenar e gerenciar suas criações de código de forma segura e eficiente.

✓ AWS CodeBuild para integração contínua

Pense em quando você realiza uma compra *on-line* e tudo ocorre de forma instantânea. O AWS CodeBuild atua como um assistente virtual, verificando minuciosamente se todos os códigos estão em ordem, assegurando que tudo esteja perfeito, semelhante a um *check-up*.

Fonte: elaborado pelo autor (2024).

Assim como a Amazon revolucionou a experiência de compras on-line, a Amazon Web Services (AWS) está inovando significativamente o campo para os novos desenvolvedores, introduzindo uma nova era de flexibilidade e confiabilidade no desenvolvimento de software. E então, está pronto para explorar?

GitHub Pages: uma visão abrangente

Fonte: elaborado pelo autor (2024).

Quando você cria um projeto no GitHub, normalmente, as pessoas podem ver o seu código. O GitHub Pages vai além disso, ele utiliza o seu código e transforma-o em um *site* que qualquer pessoa pode visitar apenas digitando um endereço no navegador.

É uma maneira fácil e gratuita de compartilhar e publicar conteúdo *on-line*, tornando-se especialmente útil para projetos de código aberto, portfólios pessoais, documentação e páginas estáticas. Vamos imaginar: se você pudesse criar um *site* inteiramente novo, sobre o que ele seria?

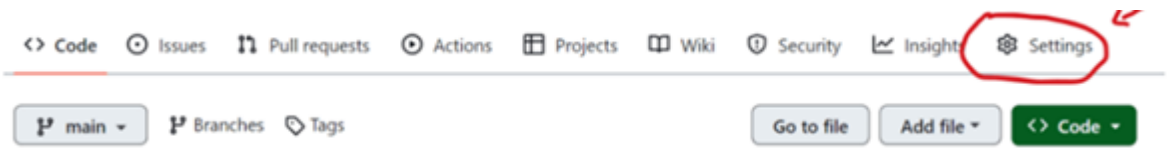
Principais características:

Conteúdo estático	Ideal para *sites* com HTML, CSS e JavaScript, pois não suporta servidores ou ambientes dinâmicos. Aqui, por exemplo, não seria possível disponibilizar um espaço onde as pessoas poderiam adicionar informações.
Integração com JEKYLL	Em vez de criar páginas dinâmicas em tempo real, como fazem os sistemas mais complexos, o Jekyll pré-compila as suas páginas e as transforma em HTML puro. Isso é perfeito para *sites* mais simples e eficientes.
Domínio personalizado	Adicionar um domínio personalizado ao seu GitHub Pages significa substituir o endereço padrão fornecido pelo GitHub (seunome.github.io/seuprojeto) por um nome de domínio personalizado, como www.seusite.com. Isso proporciona uma aparência mais profissional e personalizada para o seu *site*, além de facilitar a memorização do endereço.

Fonte: elaborado pelo autor (2024)

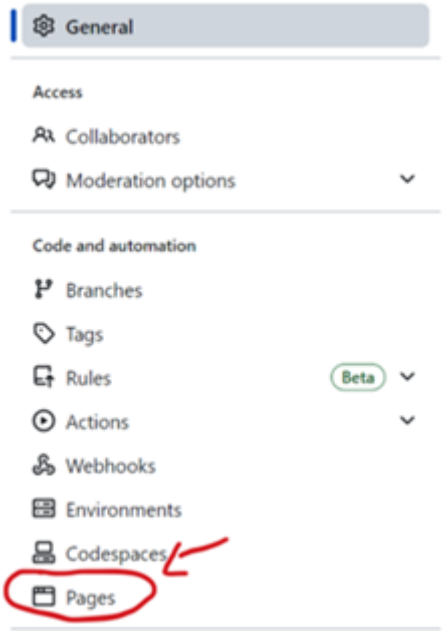
Como usar o GitHub Pages na prática:

1. Navegue até o repositório desejado e clique na aba **Settings**.



Fonte: elaborado pelo autor (2024).

2. Na coluna esquerda, encontre e clique na opção **Pages**.



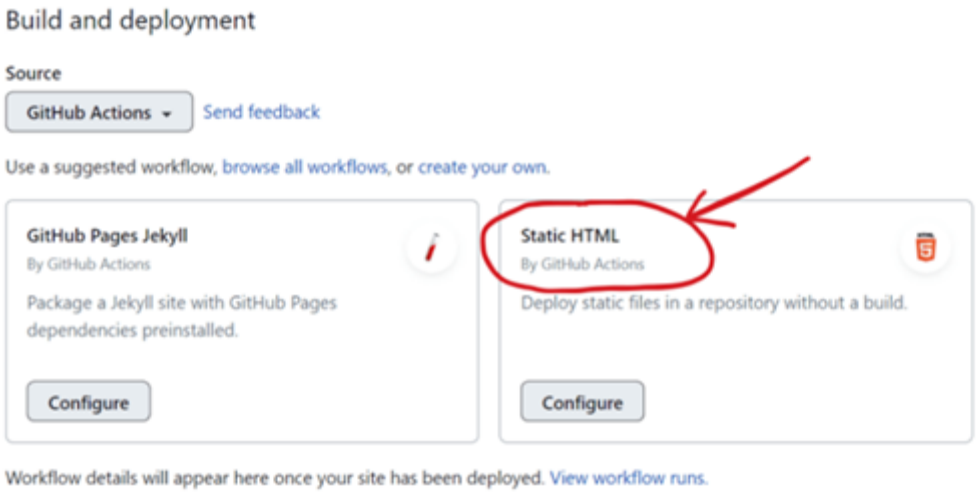
Fonte: Elaborado pelo autor (2024).

3. Escolha entre **Deploy por Branch** ou **GitHub Actions**.

- Para **Deploy por Branch**, selecione a **branch** principal (geralmente, **main**) e clique em **Save**.

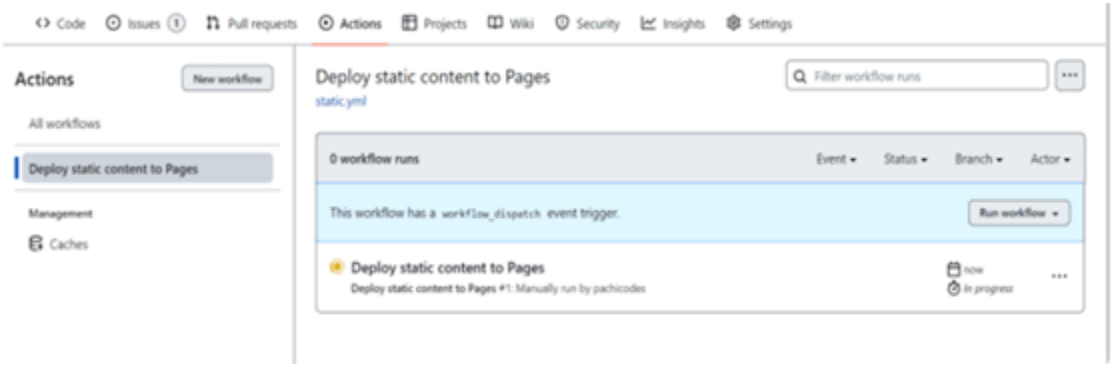
Fonte: elaborado pelo autor (2024).

- Para **GitHub Actions**, escolha um fluxo de trabalho baseado no código em seu repositório e selecione **HTML estático**. Depois, clique em **Configurar**.



Fonte: elaborado pelo autor (2024).

Você será direcionado a um fluxo de trabalho pré-criado. Revise o YAML conforme necessário e confirme as alterações.



Fonte: elaborado pelo autor (2024).

Após a ativação, o GitHub Pages gera automaticamente uma URL para o seu *site*. A URL segue o formato https://seunomeusuario.github.io/nomerepositorio. Este é o endereço onde o seu *site* será acessível publicamente.

E então, conseguiu entender um pouco mais sobre hospedagem de *sites*? Aqui, pudemos explorar diferentes aspectos desse tema, como GitHub Pages e desenvolvimento *web*. Ao entender esses conceitos, você está preparado para criar, hospedar e manter os seus próprios projetos.

Para **praticar** o que você aprendeu, que tal um desafio? Vamos lá!

DESAFIO PRÁTICO

GitHub Pages, desenvolvimento *web*, hospedagem de *sites*

Desafio: Aventura no GitHub Pages

Descrição

Você é um aspirante a desenvolvedor *web* e está empolgado com a ideia de compartilhar o seu primeiro projeto *on-line*. Você acabou de criar um portfólio pessoal para exibir as suas habilidades e projetos, porém, está um pouco perdido sobre como torná-lo acessível na *web*. É aí que entra o GitHub Pages.

Objetivos

- **Configuração inicial do GitHub Pages:** ativar o GitHub Pages para o seu repositório, permitindo que o seu portfólio seja acessível *on-line*;
- **Customização do domínio (opcional):** se você possui um domínio personalizado, dever aprender a configurá-lo para apontar para o seu GitHub Pages, dando ao seu portfólio uma aparência mais profissional;
- **Atualização contínua:** compreender como as atualizações no seu repositório refletem automaticamente no seu *site*. Certificar-se de que qualquer alteração que você fizer no código seja refletida no *site* hospedado.

Orientações

→ **Ativação do GitHub Pages:** explore as configurações do seu repositório no GitHub e ative o GitHub Pages na seção apropriada. Isso criará automaticamente uma **branch** chamada **gh-pages** para hospedar seu *site*;

- **URL de Acesso:** após a ativação, o seu portfólio estará disponível em uma URL no formato: https://seunomeusuario.github.io/nomerepositorio. Compartilhe essa URL para que outros possam ver o seu trabalho;
- **Customização de domínio (opcional):** se você deseja um toque personalizado, pesquise sobre como configurar um domínio personalizado para o seu GitHub Pages. Isso, geralmente, envolve configurar registros DNS no provedor do seu domínio;
- **Atualizações contínuas:** entenda como as atualizações feitas no seu repositório, como adição de projetos ou alterações no *design*, são refletidas automaticamente no seu *site*. Isso incentiva uma prática de desenvolvimento iterativa.

Controle de Versão

O controle de versão é como um diário do seu código, registrando sua evolução ao longo do tempo. Imagine iniciar um projeto e, ao longo de várias semanas, ver como cada linha de código evoluiu. Este é o poder do controle de versão.

Ao registrar alterações, você cria *checkpoints*. Se algo der errado, você pode facilmente voltar a um estado anterior. Vamos explorar o Git, uma ferramenta líder, e entender como os seus comandos básicos, como **commit** e **log**, podem fornecer um histórico claro e organizado.

```
$ git init          # Inicia um repositório Git
$ git add arquivo   # Adiciona alterações para o próximo commit
$ git commit -m "Primeira versão"   # Registra as alterações
$ git log           # Visualiza o histórico
```

Fonte: elaborado pelo autor (2024).

Criando um repositório remoto:

O GitHub permite hospedar o seu projeto em nuvem, facilitando a colaboração. Utilize o código abaixo para criar repositórios remotos e conectar nossos projetos locais a eles.

```
$ git remote add origin https://github.com/seu-usuario/seu-projeto.git
$ git push -u origin master
```

Fonte: elaborado pelo autor (2024).

Trabalhando em equipe

A colaboração é essencial. Com o código a seguir, é possível clonar repositórios, criar **branches** para recursos específicos e fundir essas mudanças de volta ao projeto principal.

```
$ git clone https://github.com/usuario/projeto.git
$ git branch nova-feature    # Cria um novo branch
$ git checkout nova-feature  # Troca para o novo branch
$ git merge master            # Funde as alterações de volta ao branch
```

Fonte: elaborado pelo autor (2024).

Explorar branches, resolver conflitos, usar *tags* e releases são práticas essenciais para gerenciar o desenvolvimento de software com o Git. A seguir, você poderá investigar alguns exemplos de como eles podem ser utilizados nos códigos de versão.

Explorando branches

```
# Criar uma nova branch
git branch nova-funcionalidade

# Mudar para a nova branch
git checkout nova-funcionalidade

# (ou alternativamente, criar e mudar para a nova branch em um único comando)
git checkout -b nova-funcionalidade

# Realizar alterações e commitar na nova branch
git add .
git commit -m "Implementar nova funcionalidade"

# Voltar para a branch principal
git checkout main

# Mesclar as alterações da nova branch na branch principal
git merge nova-funcionalidade


```

Fonte: elaborado pelo autor (2024).

Resolução de conflitos

```
# Durante um merge, se ocorrer um conflito, o Git indicará os arquivos com conflitos
# Abra esses arquivos e resolva manualmente os conflitos

# Após resolver os conflitos, adicione os arquivos alterados ao staging
git add <arquivo1> <arquivo2>

# Continue com o processo de merge
git merge --continue

# Ou, em caso de desistência do merge
git merge --abort

```

Fonte: elaborado pelo autor (2024).

Tags e releases

```
# Criar uma tag para uma versão específica
git tag -a v1.0 -m "Versão 1.0"

# Listar todas as tags
git tag

# Enviar as tags para o repositório remoto
git push origin --tags
```

Fonte: elaborado pelo autor (2024).

Essas são operações fundamentais no fluxo de trabalho com o Git, especialmente em projetos colaborativos ou quando você precisa gerenciar diferentes versões do seu código. Lembre-se de consultar a documentação oficial do Git e do GitHub para obter detalhes específicos e práticas recomendadas.

Integração contínua:

Para integração contínua, considerando o GitHub Actions, você pode criar um arquivo chamado .github/workflows/main.yml com o seguinte conteúdo:

```
name: CI/CD

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v2

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '14'

    - name: Install Dependencies
      run: npm install

    - name: Run Tests
      run: npm test

    - name: Deploy to Production
      run: npm run deploy
      # Adapte este comando conforme necessário para a implantação do seu projeto

```

Fonte: elaborado pelo autor (2024).

YAML Ain't Markup Language (YAML)

Esta é uma linguagem de marcação de dados utilizada comumente para escrever configurações e dados estruturados de uma forma fácil de ler, tanto para humanos quanto para máquinas. Ao contrário de outras linguagens de marcação, como XML e JSON, a YAML é projetada para ser mais legível e menos verbosa.

Características principais da YAML

Simplicidade e legibilidade

- A YAML utiliza uma sintaxe simples que se baseia em espaçamentos e indentação para estruturar os dados;
- Não requer o uso de caracteres especiais, como colchetes ou chaves, o que torna os arquivos mais concisos e compreensíveis.

Hierarquia por espaçamento

- A hierarquia dos dados é determinada pela indentação. Itens indentados sob um mesmo nível são considerados parte do mesmo bloco de dados.

Uso em configurações e dados estruturados

- A YAML é frequentemente utilizada para escrever configurações de aplicativos, *scripts* de automação, arquivos de manifesto em sistemas de orquestração, e em outros contextos em que a estrutura de dados precisa ser especificada.

Exemplo simples de YAML:

```yaml
# Um exemplo de um arquivo YAML básico
nome: John Doe
idade: 30
cidade: Nova York
hobbies:
  - Leitura
  - Caminhadas
```

Fonte: elaborado pelo autor (2024).

Neste exemplo, temos um conjunto simples de dados representando informações sobre uma pessoa. As chaves (por exemplo, nome, idade, cidade) são seguidas por seus valores, e a hierarquia é indicada pela indentação.

Aplicações do YAML:

- Configuração de *software*: muitas ferramentas e *frameworks* utilizam arquivos YAML para configuração, como Docker Compose, Kubernetes e, o já mencionado, GitHub Actions;
- Intercâmbio de dados: YAML é frequentemente utilizada para estruturar dados que precisam ser compartilhados entre sistemas.

Hey, parabéns por ter chegado até aqui! Neste tópico sobre controle de versão e uso de YAML, você explorou conceitos fundamentais para o gerenciamento eficiente de projetos de desenvolvimento de *software*. Além disso, abordamos a importância de sistemas como Git, destacando a capacidade de rastrear alterações, colaborar de forma eficiente e controlar versões para garantir a integridade do código.

Está preparado para praticar? Vamos lá!

DESAFIO PRÁTICO

Colaboração em desenvolvimento com Git e GitHub

Desafio: Projeto colaborativo de desenvolvimento

Descrição

Você integra uma equipe de desenvolvedores que estão trabalhando em um projeto *web* inovador. O trabalho está em constante evolução, com diferentes funcionalidades sendo desenvolvidas simultaneamente por membros da equipe. O Git e o GitHub são as ferramentas escolhidas para gerenciar o controle de versão e facilitar a colaboração eficiente.

Objetivos

Configuração inicial:

- clonar o repositório principal do projeto para o seu ambiente local;
- criar e mudar para uma nova **branch** chamada **nova-funcionalidade**.

Desenvolvimento de nova funcionalidade:

- fazer alterações significativas no código para implementar uma nova funcionalidade no projeto;
- realizar **commits** frequentes durante o desenvolvimento, usando mensagens descritivas.

Resolução de conflitos:

- imaginar que um colega da equipe também trabalhou na mesma área do código. Criar um cenário de conflito modificando a mesma linha de código nos dois **branches**;
- resolver os conflitos manualmente, mantendo as alterações importantes de ambos os desenvolvedores.

Branches e merge:

- Criar uma nova **branch** chamada **correcao-bug** para simular a correção de um *bug*;
- implementar correções na nova **branch** e, em seguida, realizar o **merge** dessa **branch** de volta à **branch** principal (**main**).

***Tags* e releases:**

- após uma série de melhorias, criar uma *tag* para marcar a versão 1.0 do projeto;
- listar todas as *tags* existentes e verificar se a *tag* criada está presente.

Integração contínua:

- configurar um *pipeline* de integração contínua usando GitHub Actions no arquivo .github/workflows/main.yml;
- o *pipeline* deve realizar testes automatizados sempre que houver um **push** na **branch main**.

Orientações

- Utilize os comandos **git clone, git branch, git checkout, git add, git commit, git merge, git tag, git push, git pull** conforme necessário para realizar cada objetivo;
- Experimente simular cenários realistas, como conflitos e correções de *bugs*, para ganhar experiência prática com resolução de problemas;
- Explore a documentação do GitHub para entender melhor as configurações e personalizações possíveis ao usar o GitHub Actions.

RESUMO

Neste tópico, exploramos estratégias para maximizar o potencial da hospedagem de projetos utilizando o GitHub como plataforma principal para gerenciamento colaborativo de código-fonte. Durante a aula, foi possível abordar os seguintes temas:

Fonte: elaborado pelo autor (2024).

ATIVIDADE DE FIXAÇÃO

1. Quais são os benefícios de utilizar plataformas de hospedagem em nuvem, especialmente o GitHub, para projetos de desenvolvimento de *software*?
2. Descreva o passo a passo para criar um novo repositório no GitHub.
3. Como os desenvolvedores podem clonar um repositório remoto do GitHub para as suas máquinas locais usando o Git?
4. Descreva a importância dos **branches** no contexto da colaboração com o GitHub.
5. Quais são as etapas para criar e enviar um **pull request** no GitHub?
6. Qual é a função do comando **git fetch** ao trabalhar com repositórios remotos?
7. Como sincronizar alterações locais com o repositório remoto utilizando o comando **git pull**?
8. Explique como resolver conflitos ao realizar uma operação de **merge** no Git.
9. Quais práticas ajudam a manter um histórico de **commit** claro e compreensível em um projeto colaborativo no GitHub?
10. Por que é essencial revisar e discutir alterações por meio de **pull requests** antes de integrá-las ao projeto principal?

Referências

FORBELLONE, André L. V.; EBERSPÄCHER, Henri F. *Lógica de programação:* a construção de algoritmos e estruturas de dados com aplicações em Python. 4. ed. São Paulo: Bookman, 2022.

RAMALHO, Luciano. *Python fluente*: programação clara, concisa e eficaz. São Paulo: Novatec, 2015.

CRUZ, Felipe. *Python:* escreva seus primeiros programas. São Paulo: Casa do código, 2015.

LUTZ, Mark; ASCHER, David. *Aprendendo Python*. Porto Alegre: Bookman, 2007.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. *Algoritmos:* Lógica para Desenvolvimento de Programação de Computadores. 29. ed. São Paulo: Érica, 2019.

BORGES, Luiz E. *Python para desenvolvedores.* São Paulo: Novatec, 2014.

ZIVIANI, Nivio. *Projeto de algoritmos com implementações em Pascal e C.* 3. ed. São Paulo: Cengage Learning, 2010.

LUTZ, Mark. *Learning Python*: powerful object-oriented programming. 5. ed. O'Reilly Media, 2013.

CHACON, Scott; STRAUB, Ben. *Pro Git.* 2. ed. Apress, 2014.

HTML5, CSS3 e design (PARTE 2)

Sumário

Front-end: princípios - VOL2

INTRODUÇÃO

Ao criarmos um *site*, nós queremos que ele seja atrativo e fácil de navegar, não é mesmo?

Para atingir esse objetivo, é fundamental entender sobre *front-end*, que é como adquirir superpoderes digitais, pois permite que você cuide da aparência das páginas *web* e como as pessoas interagem com o conteúdo *on-line*. Isso é possível por meio da criação de interfaces atraentes e intuitivas, que impactam diretamente a experiência do usuário. Afinal, um *site* bem projetado mantém as pessoas interessadas e envolvidas, mas, se a navegação for confusa ou desinteressante, os visitantes podem rapidamente perder o interesse.

Ainda que a gente não perceba, o *front-end* está em todos os lugares no nosso dia a dia digital, desde as redes sociais e lojas *on-line* até os sites de notícias. Um exemplo é no aplicativo do Instagram: ele está presente quando rolamos a tela, que exibe imagens de maneira suave e agradável. Já em lojas *on-line*, é possível percebê-lo quando encontramos produtos com facilidade e conseguimos finalizar as nossas compras de maneira fluida. Tudo isso é resultado do *design* de *front-end*.

Dessa forma, cada vez que você navega na internet, está interagindo com o trabalho do *front-end*. É a magia por trás dos *layouts* atraentes, dos botões clicáveis e das transições suaves.

Aprender sobre *front-end* é uma jornada empolgante para quem deseja transformar a sua criatividade em algo tangível na *web*. Desenvolver esta habilidade vai abrir portas para você construir o futuro visual da internet! **E aí, vamos começar?**

CAPÍTULO 01

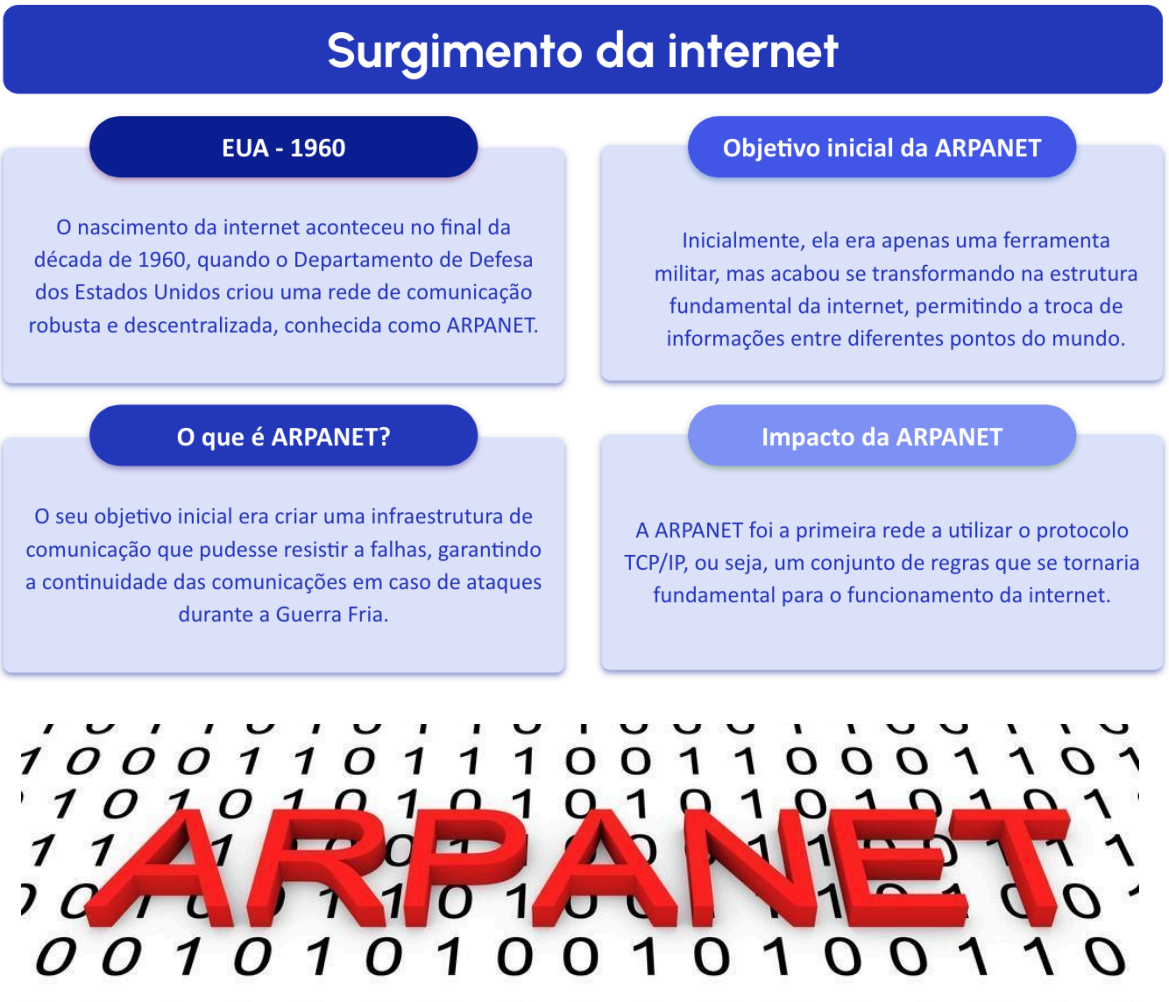
Introdução ao HTML5

O que esperar deste capítulo:

- Entender sobre o HTML e sobre a sua evolução;
- Trabalhar com a estrutura do HTML e com as suas bases.

1.1 Como tudo começou

É impossível começar a falar sobre o desenvolvimento *web* sem entender como tudo começou: ou seja, o surgimento da internet. "Se liga" na **imagem a seguir** para aprender um pouquinho sobre essa história!



Para ampliar os seus conhecimentos sobre o surgimento da internet, acesse o *link* abaixo (ou escaneie o QR *code*) para assistir ao vídeo e, depois, tente responder às perguntas a seguir.

A História da Internet! História da Tecnologia

Fonte: Vídeo *A História da Internet! História da Tecnologia*, do canal do YouTube TecMundo. Disponível em: https://youtu.be/pKxWPo73pX0. Acesso em: 5 fev. 2024.

Vamos pensar um pouco?

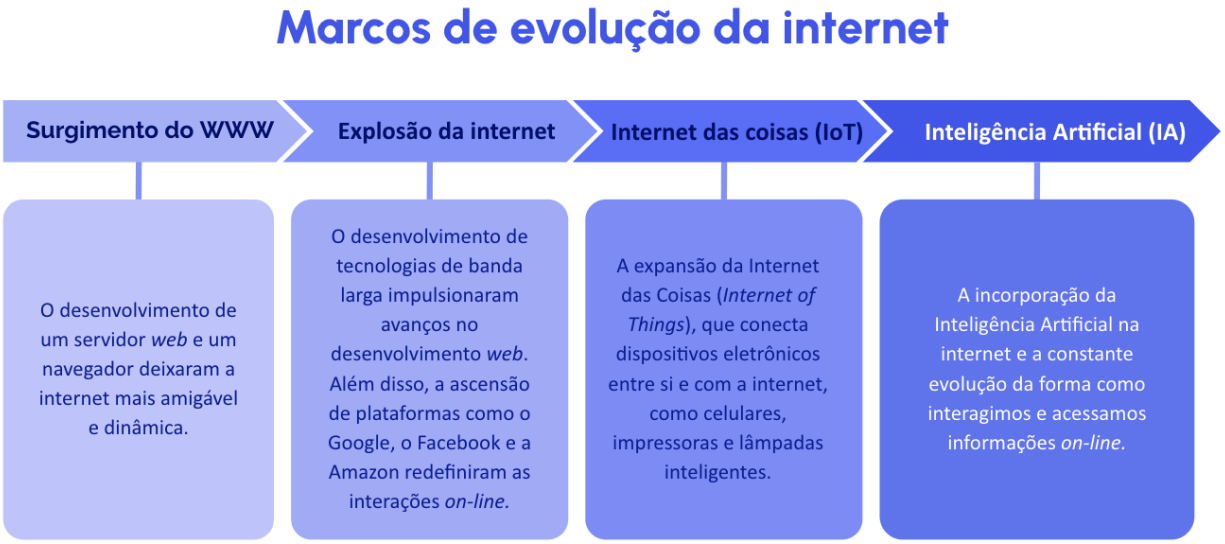
Após assistir ao vídeo, responda às questões abaixo:

- quais foram os marcos importantes na transformação da internet de uma ferramenta militar para um meio global de comunicação?
- como a internet permitiu a troca de informações entre diferentes pontos do mundo?
- quais são as consequências e as oportunidades atuais resultantes do surgimento e da evolução da internet?

Desde o início do desenvolvimento da internet na década de 1960, passando pela comunicação entre as diferentes máquinas na década de 1970 e pelas primeiras trocas de *e-mails* em 1980, o **desenvolvimento web** vem sendo marcado por constantes inovações tecnológicas.

Com a chegada da *World Wide Web* (WWW), tudo foi ficando mais dinâmico, por assim dizer. Tim Berners-Lee, um físico e cientista da computação, criou um servidor *web* e um navegador, tornando a internet bem mais amigável.

A partir disso, nós temos alguns marcos da evolução da internet. Eles são:



Linguagens como HTML, CSS e JavaScript entraram em cena, tornando as páginas mais vivas. Depois, o HTML5 e o CSS3 chegaram dando "um tapa" no *design*. Já os *frameworks*, como jQuery, Angular, React e Vue, ajudaram a criar aplicativos *web* mais atraentes. Hoje em dia, com a Internet das Coisas, a Inteligência Artificial e tudo mais, a internet não para de se transformar! E a gente muda junto com ela, surfando nessa onda digital.

1.2 HTML o que é e sua evolução

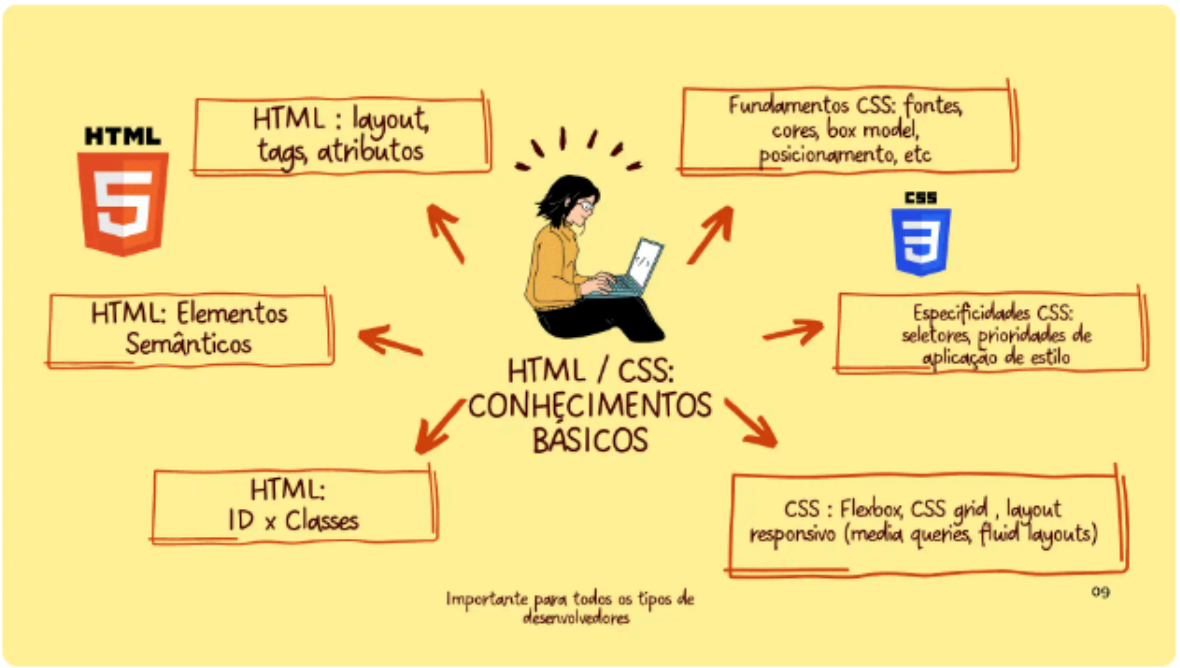
Antes de a gente poder começar a planejar os *layouts* e conteúdos e a dar vida às estruturas das páginas que temos em mente, é preciso pensar quais ferramentas serão usadas e quais são as suas finalidades. O primeiro passo consiste em entender como um documento ou um *layout* deve ser codificado para que ele seja entendido por um navegador.

É importante saber que os documentos disponíveis na internet, independentemente da temática que eles abordam (como notícias, entretenimento, ciência, comércio etc.), são estruturados através de uma **linguagem**, chamada **Linguagem de Marcação de Hipertexto**, ou seja, **HTML** (do inglês, *Hypertext Markup*

Language). Essa linguagem é um mecanismo para adicionar marcas com algum significado a um texto.

Disponível em:<freepik-http://tinyurl.com/2ukhjzja>. Acesso em: 06 jan. 2024.

É importante notar que o **HTML não é uma linguagem de programação**, algo que muitos desenvolvedores iniciantes acabam se confundindo. Essa **linguagem de marcação** é a espinha dorsal da estrutura da *World Wide Web*, sendo usada para criar e organizar conteúdo na *web*, permitindo que os navegadores interpretem e exibam informações de maneira formatada. Nela, os elementos são marcados com *tags* que indicam a estrutura e a função de cada parte do conteúdo.



Disponível em: <http://tinyurl.com/4nau87cj>. Acesso em: 08 jan. 2024.

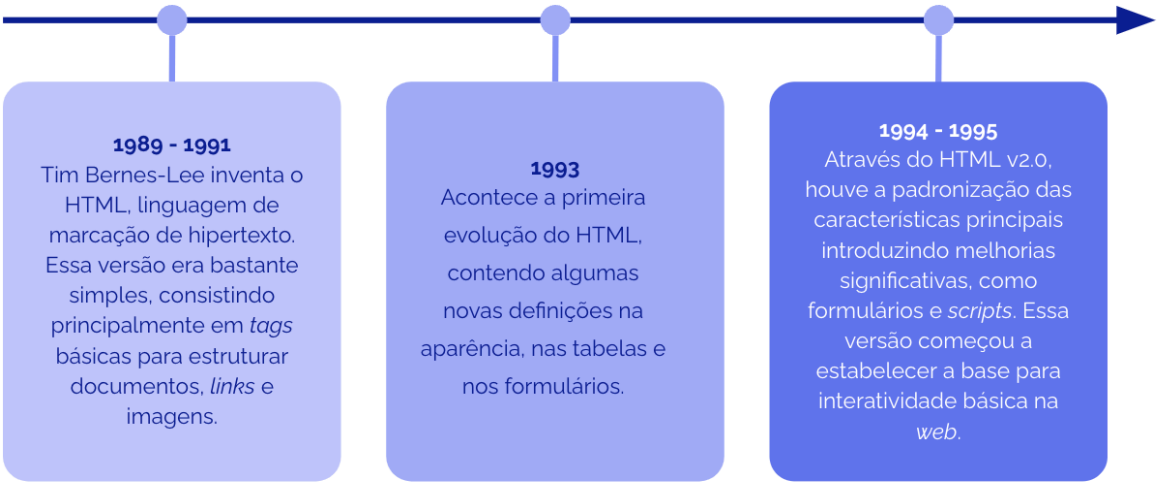
Lançado no ano de 1991, a linguagem HTML ganhou novos recursos e novas melhorias, como a possibilidade de trabalhar com diversos tipos de *frameworks*, o que não era permitido nas suas primeiras versões. Nelas, só era possível utilizar editores de textos simples para a elaboração das páginas *web*.

Já uma linguagem de programação de fato envolve **estruturas especializadas** que dependem do uso de variáveis simples e compostas, condições, laços e até objetos em questões mais complexas. Todas essas especificações não fazem parte das estruturas do HTML nem do CSS. Nesses casos, o HTML trabalha fundamentado nas marcas e nas etiquetas e o CSS é baseado nos seletores, nas propriedades e nos valores, o que veremos mais à frente.

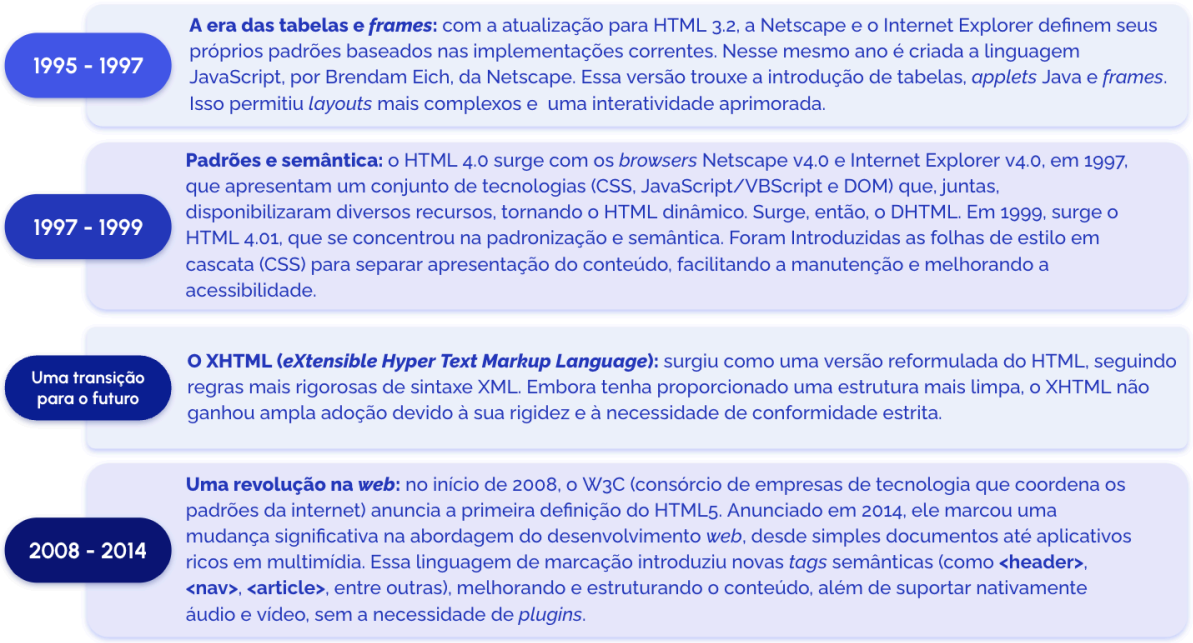
HTML e a sua evolução

Assim como muitas linguagens, o HTML passou (e passa) por atualizações e ajustes das suas estruturas ao longo do tempo. Isso permite o seu aprimoramento, trazendo consigo novas interações e, até mesmo, a correção de pequenos *bugs* nas suas estruturas. Essa evolução reflete a necessidade de atender às demandas crescentes da *web*, proporcionando maior funcionalidade e melhor usabilidade e suporte para as tecnologias mais recentes.

Vamos entender um pouco mais sobre essa evolução?



Fonte: Elaborado pelo autor (2023).



Fonte: Elaborado pelo autor (2023).

Alguns recursos notáveis do HTML5 são:

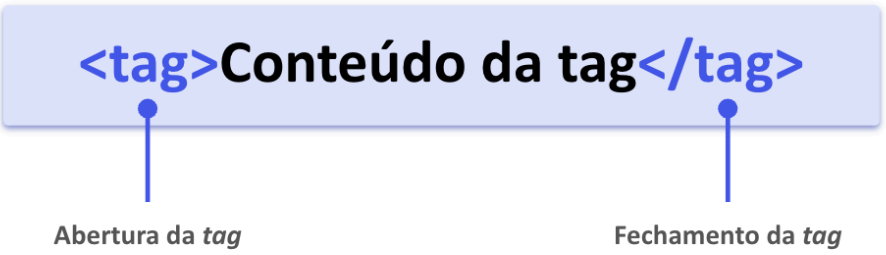
- **APIs avançadas**: geolocalização, armazenamento local, *canvas* para gráficos etc.;
- **semântica aprimorada**: *tags* específicas para estruturar o conteúdo, facilitando a compreensão por máquinas e melhorando a acessibilidade;
- **suporte nativo para mídia**: integração direta de áudio e vídeo, reduzindo a dependência de *plugins* (como o Flash);
- **WebSockets e Web Workers**: melhoria na comunicação entre o navegador e o servidor, possibilitando experiências mais interativas.

A evolução do HTML5 representa uma jornada contínua para tornar a internet mais poderosa, acessível e capaz de suportar uma variedade crescente de aplicativos e conteúdos. Com o avanço da tecnologia, é certo que novas versões e padrões continuarão a moldar o futuro do desenvolvimento *web*.

1.3 Estrutura básica de um documento HTML5

O HTML5 é a quinta (e mais recente) versão do HTML. Ela traz consigo uma estrutura básica simplificada e mais poderosa para a criação de páginas *web* e possui uma sintaxe limpa e uma semântica que facilita o desenvolvimento e melhora a acessibilidade. Para iniciarmos a elaboração de um desenvolvimento, é preciso compreender a sua estrutura básica, entendendo como trabalhar com ela de duas maneiras diferentes.

Para trabalhar com o HTML, é preciso entender os seus elementos fundamentais, ou seja, as ***tags***, pois são elas que irão estruturar e formatar o conteúdo em uma página *web*. A sua composição é definida por **colchetes angulares**, que são utilizados, em sua maioria, como a abertura e o fechamento de uma *tag*.



Vale ressaltar que algumas *tags* podem possuir propriedades e, até mesmo, serem fechadas nelas mesmas, como vamos observar mais à frente.

O primeiro passo é entender e trabalhar com uma estrutura simples, limpa e que retorna as informações dentro das necessidades mais básicas.

Entendendo as *tags* fundamentais

Declaração de início e fim do HTML: <HTML>

A *tag* **<html> </html>** é utilizada para dizer ao navegador onde iniciar a interpretação do documento HTML. Ela sempre será a *tag* inicial e a final da estrutura como um todo.

```
<html>
<head>···
</head>
<body>

</body>
</html>
```

Fonte: Elaborado pelo autor (2023).

Declaração do cabeçalho: <head>

No cabeçalho, que é marcado pelas *tags* **<head></head>**, estão marcadas as informações que serão usadas pelo navegador. Por exemplo:

- **<title>Sou o título da página</title>**: define o **título** da página *web*, que fica visível na aba do navegador.

Ela precisa estar definida **após a *tag* inicial** e **sempre antes da *tag* usada para a definição do conteúdo do documento HTML**.

```
<head>
    <title>Document</title>
</head>
```

Fonte: Elaborado pelo autor (2023).

Declaração de início do corpo da página HTML: <body>

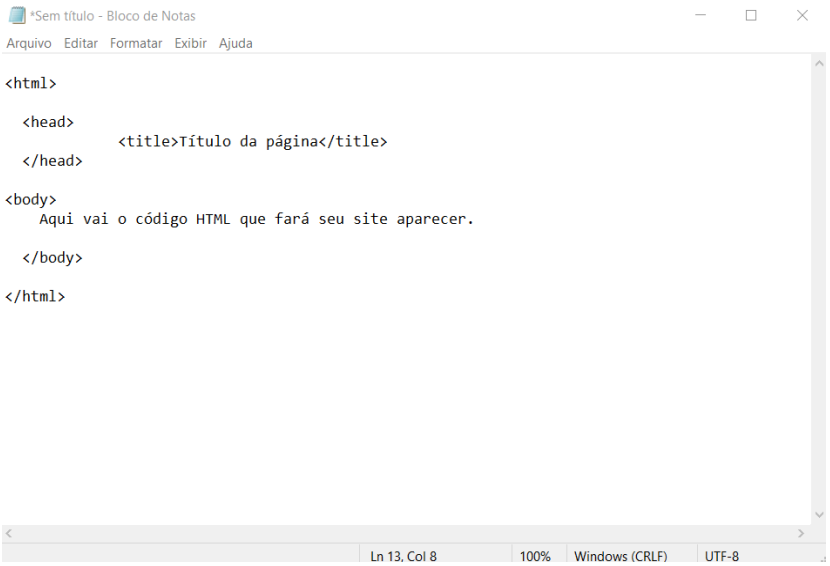
A *tag* **<body>** marca o início do documento. Ou seja, toda a informação do documento deve estar dentro do ***body***. Ela sempre deverá ser utilizada **após o cabeçalho** e sua *tag* de fechamento sempre deverá ficar no final, antes da *tag* de fechamento **</html>**.

```
<html>
<head>
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Fonte: Elaborado pelo autor (2023).

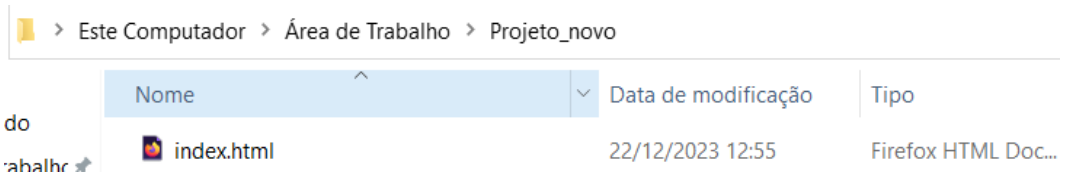
Depois de você entender o funcionamento dessas *tags*, já é possível elaborar páginas simples e sem muitos recursos. Esse primeiro momento é **fundamental** para entender as suas bases e, com isso, aprimorar as suas habilidades aos poucos. Para isso, você vai utilizar um editor de texto simples e sem muitos recursos — nesse caso, nós utilizaremos o Bloco de Notas. Acompanhe o passo-a-passo a seguir:

- acesse o Bloco de Notas e, nele, utilize as *tags* com a estrutura presente na imagem a seguir:



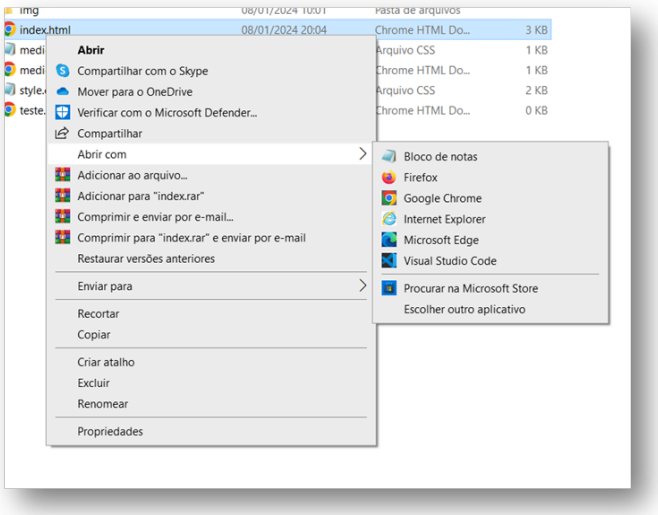
Fonte: Elaborado pelo autor (2023).

- após elaborar essa primeira estrutura, salve a sua página. Por padrão, toda primeira página do desenvolvimento deve ser salva como **index.html.** A **extensão HTML** é o que irá definir que é uma página *web*;
- outro fator importante é que, sempre que for trabalhar com o desenvolvimento de um *site*, você deve definir o local em que irá salvar o documento e as demais informações. Ou seja, é preciso criar uma pasta com o nome do projeto e salvar o documento dentro dela;



Fonte: Elaborado pelo autor (2023).

- depois de realizar os primeiros passos para identificar a sua página, encontre o seu arquivo **index.html** e abra-o no seu navegador padrão. Assim, você terá uma apresentação igual à tela a seguir:



Fonte: Elaborado pelo autor (2023).

Fonte: Elaborado pelo autor (2023).

- verifique as informações do código executado e compare-o com o resultado apresentado.

```
<HTML>

<HEAD>

<TITLE>Seu título aqui</TITLE>

</HEAD>
<BODY>
Este é o código utilizado para elaborar a página

</BODY>
</HTML>
```

Fonte: Elaborado pelo autor (2023).

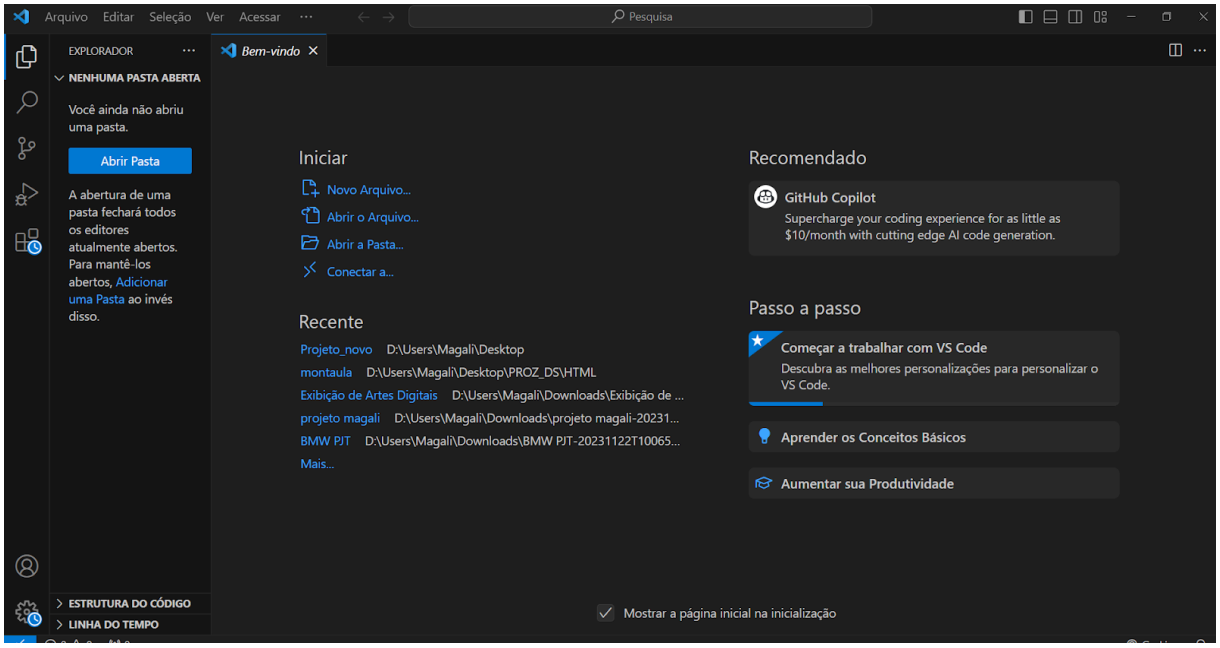
Entendendo os demais elementos da estrutura básica:

Já conseguimos perceber que o HTML, assim como um documento de texto básico, permite realizar várias melhorias na sua estrutura. Para utilizá-lo, precisamos entender, ao menos, o seu funcionamento básico. Ou seja, é preciso saber iniciar todo o processo, além de deixá-lo pronto para se ajustar à nossa rebuscada língua portuguesa e às suas regras de acentuação.

A partir de agora, vamos trabalhar com um **editor de texto** utilizado por desenvolvedores que nos ajudará a entender muitos outros recursos no decorrer do aprendizado: o Visual Code Studio.

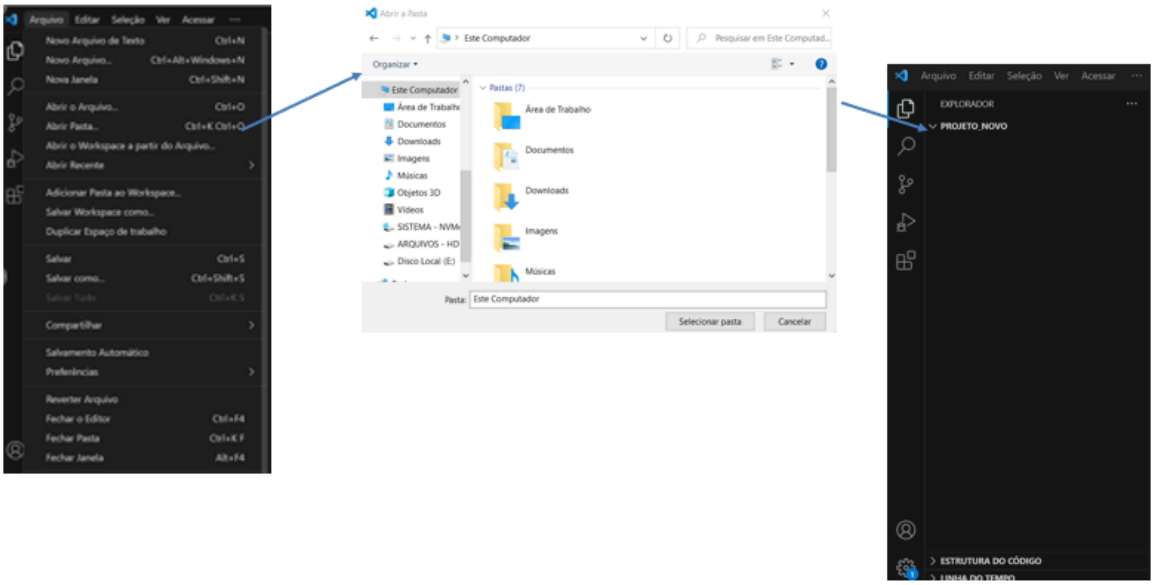
Sendo assim, vamos ao passo-a-passo apresentado a seguir.

- Acesse o **Visual Code;**



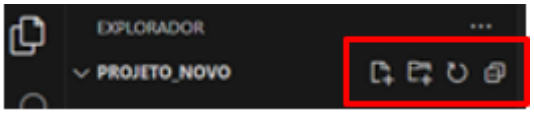
Fonte: Elaborado pelo autor (2023).

- Em seguida, crie uma **pasta para o armazenamento dos conteúdos** que serão desenvolvidos. Isto é, as páginas dentro do projeto, os arquivos das linguagens utilizadas, as imagens e as demais mídias que possam ser usadas no projeto;
- Depois, acesse o Visual Code, clique em **Arquivo** e, depois, em **Abrir Pasta,** para abrir a pasta que foi criada;



Fonte: Elaborado pelo autor (2023).

- A seguir, **crie o primeiro arquivo**, que, assim como já estudamos, será chamado de **index.html.** Para isso, ao lado do nome da pasta do projeto, terá uma sequência de ícones que correspondem respectivamente a:



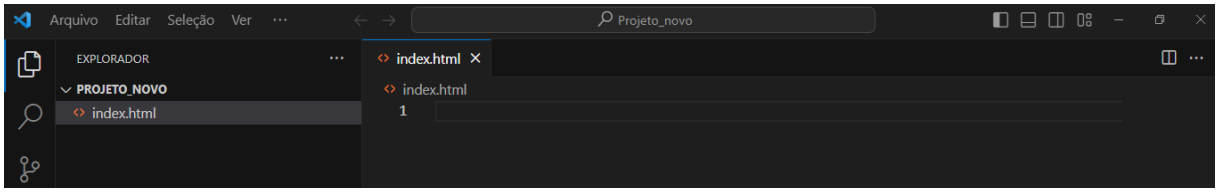
Fonte: Elaborado pelo autor (2023).

  - criar um novo arquivo;
  - criar uma nova pasta;
  - atualizar o explorador;
  - recolher as pastas do explorador.

- Clique na opção de **criar novo arquivo** e crie o arquivo **index.html**.

IMPORTANTE: não se esqueça de que devemos utilizar **apenas letras minúsculas** e que **não devemos usar acentos nem colocar espaços**.

Observe que o arquivo criado já aparece aberto na área principal do editor e o arquivo com extensão **.html** aparecerá com um pequeno ícone **<>,** indicando que foi criado no formato correto.



Fonte: Elaborado pelo autor (2023).

Para criar o **código base** do HTML na área principal, é possível trabalhar com duas opções:

- a primeira delas é digitar **HTML** e clicar na opção **HTML5**;
- como segunda opção, você pode digitar somente um ponto de exclamação **(!)** e, em seguida, pressionar a tecla ***Enter***. Assim, o código será gerado automaticamente e, conforme as necessidades do desenvolvimento e do desenvolvedor, poderá passar por algumas alterações.

Fonte: Elaborado pelo autor (2023).

Apesar de algumas dessas *tags* já serem conhecidas, vamos entender cada uma das linhas que ainda não estudamos, pois esse é um fator importante para que você possa dar sequência ao seu desenvolvimento de projetos.

Tags ou Atributos	De
<!DOCTYPE HTML>	Essa declaração é usada para es de página que eles irão renderiz a **exibirem as páginas da *web* c uma vez**, no topo da página,
<html lang="en">	Essa é a *tag* principal, **<html>,** sobre o seu **conteúdo.** Nesse **"pt-br"**, o que significa que **bra**
<meta charset="UTF-8">	Essa *tag* adiciona ao documento A remoção dela pode causar **e** letra
<meta name="viewport" content="width=device-width, initial-scale=1.0">	Essa *tag* indica que o conteúdo **todo o espaço disponível d**

ATIVIDADE DE FIXAÇÃO

1. Qual é o significado da sigla **HTML**?
   a. *Hyperlink and Text Markup Language.*
   b. *HyperText Markup Language.*
   c. *High-Level Text Management Language.*
   d. *Hyper Transfer Markup Language.*

2. O que a linguagem HTML define em uma página *web*?
   a. O seu estilo.
   b. A sua estrutura e o seu conteúdo.
   c. A sua interatividade.
   d. A sua navegação.

3. Na década de 1970, a ARPANET deu os primeiros passos para conectar instituições acadêmicas. Qual foi o objetivo inicial dessa rede?
   a. Facilitar o comércio *on-line*.
   b. Conectar universidades e instituições de pesquisa.
   c. Criar uma rede exclusiva para o governo.
   d. Desenvolver jogos *on-line.*

4. Explique o papel do HTML na construção de páginas *web* e a sua relação com o CSS.

5. Como se deu o início do HTML? Quais foram os principais marcos da sua evolução?

6. Descreva a estrutura básica de um documento HTML, indicando o propósito de cada uma das suas *tags* principais.

7. Qual é a diferença entre as *tags* de abertura e de fechamento no HTML? Dê exemplos.

8. Ao criar uma página *web* para um novo projeto, você se depara com a necessidade de estabelecer uma estrutura sólida e organizada. Explique a importância da estrutura principal do HTML nesse contexto, abordando as principais *tags* estruturais. De que forma essas *tags* (como **<html>, <head>, <body>** etc.) desempenham papéis cruciais na organização do conteúdo e na formatação da página? Além disso, discuta como uma estrutura HTML bem definida não apenas facilita a leitura do código, mas, também, contribui para uma experiência de usuário mais eficiente e acessível.

9. Qual é a finalidade da *tag* **<title>**?

10. Qual é a finalidade dos atributos definidos dentro das *tags* **lang** e **"UTF-8"**?

DESAFIO PRÁTICO

Estrutura básica de um documento HTML5

Desafio: Estruturando um projeto.

Descrição

Considerando os conhecimentos que você adquiriu durante as aulas, reflita: você já se sente preparado para começar a explorar o mercado de trabalho e as oportunidades disponíveis, especialmente agora que está imerso no campo da tecnologia? Imagine que, após criar um perfil no LinkedIn e indicar que está estudando e

possui habilidades de nível iniciante em desenvolvimento *web*, você encontra, algumas semanas depois, uma oportunidade de emprego como *trainee*, que exige conhecimentos básicos na área. Ao ler o anúncio da vaga, você descobre que um dos requisitos do processo seletivo inclui a criação de uma **estrutura básica** para um projeto *web*, que deverá ser disponibilizada em um repositório *on-line.*

Objetivos

- Demonstrar conhecimento nas questões básicas de construção do projeto;
- Criar a estrutura de uma página desde o início;
- Salvar o início do projeto em um repositório para ter a sequência do trabalho futuramente.

Orientações

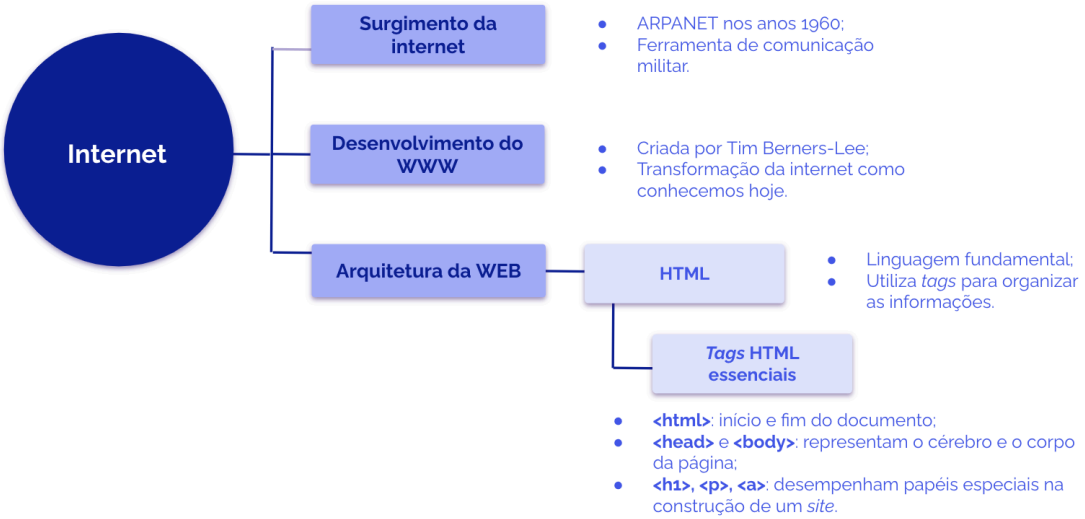
- Organize seu projeto criando uma pasta na qual todos os elementos serão salvos;
- Inicie o projeto criando um arquivo base seguindo o padrão de desenvolvimento estabelecido;
- Utilize o editor de código VSCode para criar a estrutura básica em HTML;
- Defina o título da página como **Projeto Inicial**;
- No corpo da página HTML, faça uma descrição do que você aprendeu até o momento sobre essa linguagem;
- Após concluir as etapas anteriores, salve o projeto e direcione-o para um repositório.

Exploramos o início do desenvolvimento *web*, destacando o surgimento da internet e a sua evolução desde os anos 1960 e que, com a chegada da *World Wide Web*

(WWW) e com as contribuições de Tim Berners-Lee, o desenvolvimento *web* tornou-se mais dinâmico. Também entendemos que o HTML (*Hypertext Markup Language*) atua como a espinha dorsal da *web*, permitindo a criação e a organização do conteúdo de maneira formatada.

Nós também aprendemos que o HTML, lançado em 1991, evoluiu ao longo do tempo, ganhando recursos e melhorias, como a introdução de *frameworks*. Diferenciando-se de linguagens de programação, ele foca em marcas e etiquetas para estruturar o conteúdo. Abordamos, ainda, que a evolução do HTML5 apresenta recursos notáveis, como APIs avançadas, semântica aprimorada e suporte nativo para mídias, como áudio e vídeo.

Também trabalhamos com o processo de criação de um arquivo HTML usando um editor de texto, como o Visual Code, e vimos exemplos de *tags* para uma compreensão básica do código HTML.



CAPÍTULO 02

Primeiros passos em HTML

O que esperar deste capítulo:

- Aprender sobre as *tags* essenciais e as suas aplicações;
- Compreender as *tags* estruturais e como elaborar uma página com elas.

2.1 Trabalhando com as *tags* essenciais

Antes de a gente se aprofundar em detalhes sobre as *tags* essenciais, é preciso entender que um desenvolvedor precisa dar muita importância à clareza dos seus códigos. Em um primeiro momento, pode parecer algo sem importância ou necessidade, mas, com o passar do tempo, você vai ver que são informações que terão grande destaque no desenvolvimento *web* e em outros desenvolvimentos.

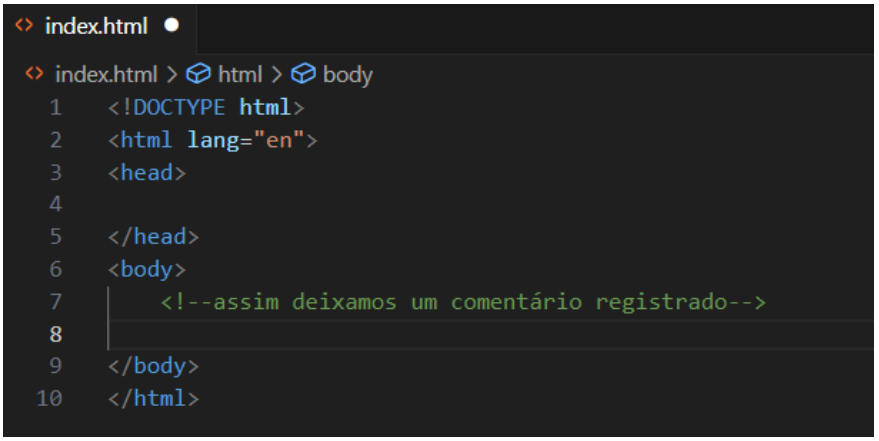
Nesse caso, nós estamos falando sobre os **comentários** no decorrer do código. Eles são importantes devido a alguns fatores, como:

- o esquecimento **do que** foi feito e **por que** algo foi feito de alguma forma;
- códigos muito extensos;
- a agilidade no entendimento do código;
- a facilidade para outros desenvolvedores entenderem como o código foi elaborado.

Por motivos como esses, torna-se crucial ter o hábito de documentar, sempre que for possível, o que está sendo feito, minimizando problemas futuros. Além disso, é importante destacar que **tudo** dentro de um comentário **não** aparece na página, independente do que foi descrito nele, podendo ser uma *tag* ou um texto.

Para fazer um comentário em HTML, deve-se usar a seguinte sintaxe:

<!--assim deixamos um comentário registrado-->



Fonte: Elaborado pelo autor (2023).

Uma outra coisa fundamental é entender a diferença entre o *layout* e os dados da página. O ***layout*** é toda a estrutura, tudo o que será posicionado e que podemos determinar o tamanho, as cores e os formatos, entre outras definições. Já os **dados** são os conteúdos que serão inseridos, como os textos, as tabelas e os formulários.

Por exemplo: se você desejar colocar uma tabela de valores ou horários, por exemplo, o conteúdo da tabela se refere aos **dados**. Já o ***layout*** diz respeito a onde ela será colocada, qual espaço ela irá ocupar, aos tipos de letras e à sua cor.

Tag <html> </html>

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

</body>
</html>
```

Fonte: Elaborado pelo autor (2023).

Assim como vimos anteriormente, a *tag* **<html>** é a estrutura inicial de qualquer documento HTML e indica o início e o fim do código. Ela envolve todo o conteúdo da página *web*, fornecendo o ponto de partida para a construção da estrutura.

O único item que fica fora dessa *tag* é a descrição **<!doctype html>**, que, assim como já foi estudado, define o tipo de página que os navegadores irão renderizar.

Tag <head> </head>

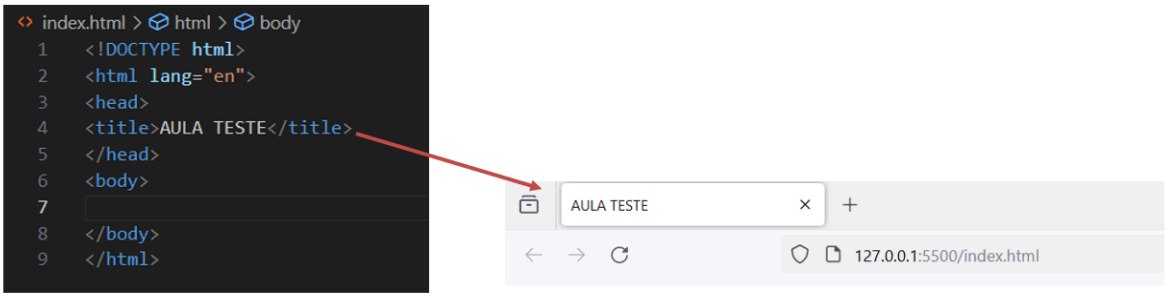
Assim como o próprio nome indica, a *tag* **<head>** representa o cabeçalho da página. Esta parte não fica totalmente visível para o usuário, visto que a maioria das suas configurações são feitas para a estruturação da página e as suas demais relações.

No entanto, outras *tags* podem ser inseridas dentro dela, como **<title>, <meta>, <link>,** e **<script>**.

Tag <title> </title>

A *tag* **<title>** representa o título da página. Ou seja, é através dela que é possível definir as informações que ficam no topo da página, especificamente a **definição da guia**. É importante destacar que ela só pode ser usada **dentro** da *tag* **<head>**.

Exemplo: defina uma página com o título "**AULA TESTE**". Depois de salvar, abra o seu arquivo **index.html** e observe o título da página na guia do navegador.



Fonte: Elaborado pelo autor (2023).

Tag <meta>

Apesar da *tag* <meta> não promover nenhuma mudança estrutural ou qualquer tipo de questão visual na página, ela é importante para o processo de indexação feita pelos *sites* de busca. Isso acontece porque um dos fatores que cresceu consideravelmente na *web* foi a colocação das páginas no topo dos resultados de buscas, tanto que surgiu um novo profissional no mercado, especializado em otimização dos metadados para uma

melhor indexação por parte dos *sites* de busca. Esta especialidade é conhecida como **SEO (*Search Engine Optimization* ou Otimização para Motores de Busca)**.

Metadados referem-se à informação sobre uma informação, permitindo especificar uma variedade de detalhes relevantes para os motores de busca nos quais desejamos indexar o *site*. Isso inclui uma descrição da página e as suas palavras-chave, bem como a identificação do autor do documento, entre outros dados considerados pertinentes.

Exemplo de uso da *tag* <meta>:

Normalmente, é a partir da palavra-chave que os sistemas de busca na rede procuram uma página ou um *site*. Para usá-la, é preciso criar atributos junto à tag. Vejam o exemplo da imagem abaixo, com comentários referentes às suas utilidades.

```
<meta charset="UTF-8">
<!--define o padrão de acentuação, ou seja, nesse caso seguirá
o padrão do português-brasileiro-->

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<!--definições do padrão de tamanho que será mostrada a página-->

<meta http-equiv="X-UA-Compatible" content="IE=edge"/>
<!--compatibilidade do navegador-->
```

Fonte: Elaborado pelo autor (2023).

Tag <link>

Normalmente, a *tag* **<link>** é utilizada para incluir recursos externos em uma página *web*. Frequentemente, ela é associada à vinculação de outros arquivos na página e é a partir dela que serão vinculados os arquivos de estilo CSS, arquivos JavaScript e arquivos PHP.

Como exemplo, podemos citar o *link* para vincular um arquivo CSS:

```
<link href="caminhodoarquivo" rel="stylesheet" type="text/tipodotexto"/>
```

Fonte: Elaborado pelo autor (2023).

Veja a utilidade de cada atributo dentro da *tag* **<link>** a seguir:

- **href:** este atributo especifica o caminho do arquivo que será vinculado. No contexto de folhas de estilo, o valor do **href** é o endereço do arquivo CSS que será aplicado à página;
- **rel:** indica o tipo de relação entre o documento atual e o recurso vinculado. No caso de folhas de estilo, o valor "***stylesheet***" é utilizado para indicar que o arquivo vinculado é uma folha de estilo;
- **type:** define o tipo de conteúdo do recurso vinculado. Normalmente, em um contexto de folhas de estilo CSS, o valor "**text/css**" é utilizado.

Tag <script>

A *tag* **<script>** é usada para incorporar ou referenciar *scripts*, geralmente escritos em JavaScript, que fornecem funcionalidades dinâmicas e interatividade às páginas *web*.

```
<script type=”text/tipodotexto” script desejado></script>
```

Fonte: Elaborado pelo autor (2023).

Dentro dessa tag, o atributo **type** especifica o tipo de conteúdo do *script*. Anteriormente, o valor "**text/javascript**" era comumente utilizado, mas, atualmente, para *scripts* em JavaScript, ele é considerado opcional. Porém, é uma boa prática incluir **type="text/javascript"** por questões de compatibilidade.

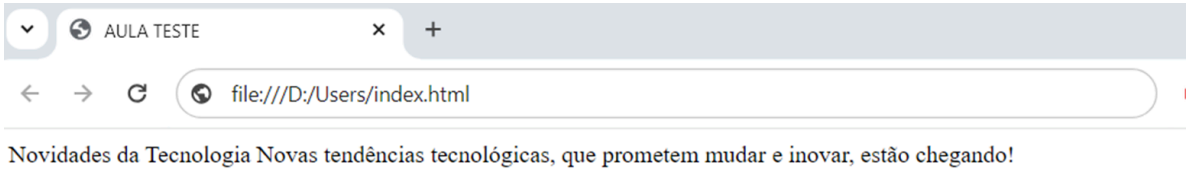
Dentro da *tag* **<script>**, você deve inserir o código JavaScript desejado, que pode ser fornecido diretamente no documento HTML ou referenciar um arquivo externo usando o atributo **src**.

Tag <body> </body>

Assim como o próprio nome declara, a *tag* **<body>** representa o corpo da página. Dessa forma, é onde **todo o conteúdo da página** será inserido, ou seja, tudo que ficará visível no navegador. Veja o exemplo da declaração em HTML e o resultado a seguir.

```
index.html > html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>···
21  </head>
22  <body>
23      Novidades da Tecnologia
24  Novas tendências tecnológicas, que prometem mudar e inovar, estão chegando!
25
26  </body>
27
28  </html>
```

Fonte: Elaborado pelo autor (2023).



Fonte: Elaborado pelo autor (2023).

Analisando as imagens anteriores, é possível considerar que temos uma página (ou o início de um *site*) sendo desenvolvida. Porém, o que não fica tão claro é que ela apresenta um texto simples, sem maiores detalhes de imagens e configurações.

É importante definir a diferença entre um **site** e uma página: um *site* é um conjunto de páginas, que, por sua vez, apresentam apenas um entre os outros conteúdos do *site*. Sendo assim, um *site* só existe quando todas as páginas pertencentes a ele são interligadas.

2.2 *Tags* estruturais ou elementos semânticos

Na linguagem HTML, as *tags* estruturais são aquelas que ajudam a criar divisões no documento, possibilitando a criação de "unidades funcionais". Elas são usadas durante a criação do *layout* e têm grande utilidade ao realizar a estilização pelo CSS. Alguns exemplos que você vai ver com frequência para a estruturação e estilização das páginas são as *tags* ** **, **** e os atributos ***id*** e ***class.***

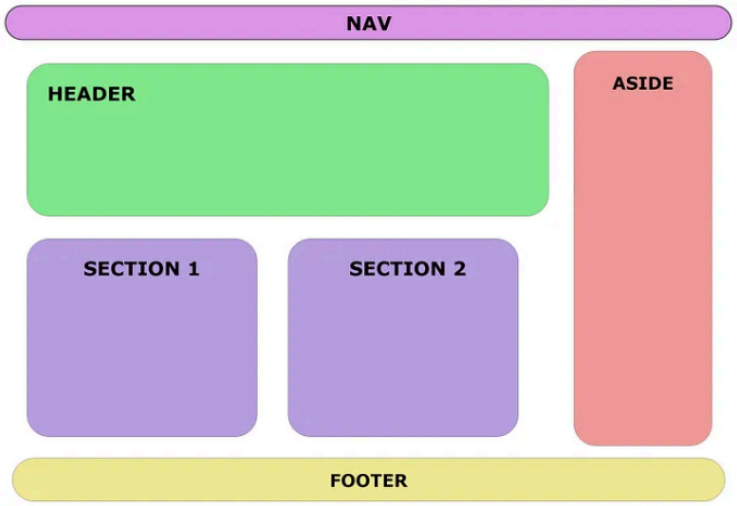
Essas *tags* também são conhecidas como elementos semânticos, que, em HTML, desempenham um papel fundamental na estruturação e no significado de uma página *web*. A semântica refere-se ao estudo do significado e, no contexto da *web*, os elementos

semânticos são aqueles que contribuem para a compreensão e interpretação correta do conteúdo por parte dos navegadores, motores de busca e, principalmente, pelos usuários.

Lembra daquele comentário sobre os motores de busca? Então, a *web* semântica leva o SEO do seu *site* a um outro patamar! Confira o exemplo abaixo:

Disponível em:<http://tinyurl.com/4639z4fd>. Acesso em 08 jan. 2024.

Diferente dos elementos não semânticos, que são utilizados apenas para definir a apresentação visual da página, os elementos semânticos são escolhidos com base no significado lógico e estrutural do conteúdo. Isso não apenas facilita a leitura e a compreensão do código por desenvolvedores, mas, também, melhora a acessibilidade e a indexação por motores de busca. Veja abaixo um modelo de estruturação de elementos dentro de uma página HTML:



Disponível em: <http://tinyurl.com/yc34bx4s>. Acesso em 28 dez. 2023.

Na imagem, é possível ver os "blocos" de conteúdos em que as informações estarão disponibilizadas. Como exemplo, nós temos o *footer,* ou o "rodapé", em que informações como o endereço e os meios de contato podem estar na página.

Existem algumas *tags* utilizadas no HTML que têm a finalidade de ajustar e melhorar a estrutura da página, cada uma com a sua particularidade. Outro aspecto importante das *tags* estruturais é que, normalmente, elas estão consolidadas dentro de *frameworks* de desenvolvimento *web*, como o Bootstrap. Veja os tipos e exemplos na imagem abaixo:

Observe a tabela a seguir para compreender as funções básicas de algumas das *tags* destacadas na imagem que vimos anteriormente.

Tags	Funções
<main>	Essa *tag* representa o conteúdo principal da página ou a funcionalidade central da aplicação, excluindo os cabeçalhos, os rodapés e as barras laterais.
<section>	Essa *tag* é utilizada para dividir ou segmentar o conteúdo de uma página em seções distintas. Cada seção pode ter seu próprio cabeçalho e conteúdo.
<article>	Essa *tag* delimita um artigo em sua página. É frequentemente empregada em *blogs* ou páginas de conteúdo, permitindo que o artigo seja reutilizado ou distribuído de forma independente.
<aside>	É utilizada para marcar conteúdos relacionados ao conteúdo principal da página, mas que podem ser considerados secundários ou não essenciais para a compreensão do conteúdo principal, podendo conter informações como barras laterais, anúncios, *links* relacionados, entre outros.
Outras *tags*	As *tags* **<footer>**, **<header>** e **<nav>** serão trabalhadas mais a frente, em um contexto mais pertinente a elas.

Outras tags estruturais

Tag

Essa *tag* tem como função definir uma ou mais divisões da página, sendo utilizada, normalmente, para agrupar elementos que se relacionam entre si. Imagine-a como uma caixa na qual tudo que for colocado dentro dela irá possuir as mesmas características. Ela funciona, portanto, como um *container* para conteúdo de fluxo. Uma vez que não possui valor semântico, a *tag* ** ** é muito usada para organizar melhor o conteúdo.

Ela é utilizada quando se pretende isolar parte da página e aplicar um estilo CSS ou efeitos JavaScript.

```
<body>
    <div>
        <h1 id="titulo"> Novidades da Tecnologia</h1>
        <p id="subtitulo"> Tecnologia & Inovações</p>
        <br>
    </div>
```

Fonte: Elaborado pelo autor (2023).

Fonte: Elaborado pelo autor (2023).

Tag

A *tag* **** tem como função diferenciar ou agrupar um bloco de texto de uma *tag* de parágrafo. Ou seja, geralmente, ela é usada quando se pretende isolar uma parte do texto e aplicar um estilo CSS ou um efeito JavaScript.

Apesar dessa *tag* ter uma funcionalidade e características parecidas com os parágrafos, ela costuma ser utilizada apenas para pequenas informações (como legendas de um formulário ou de uma imagem) ou para formar um *container*.

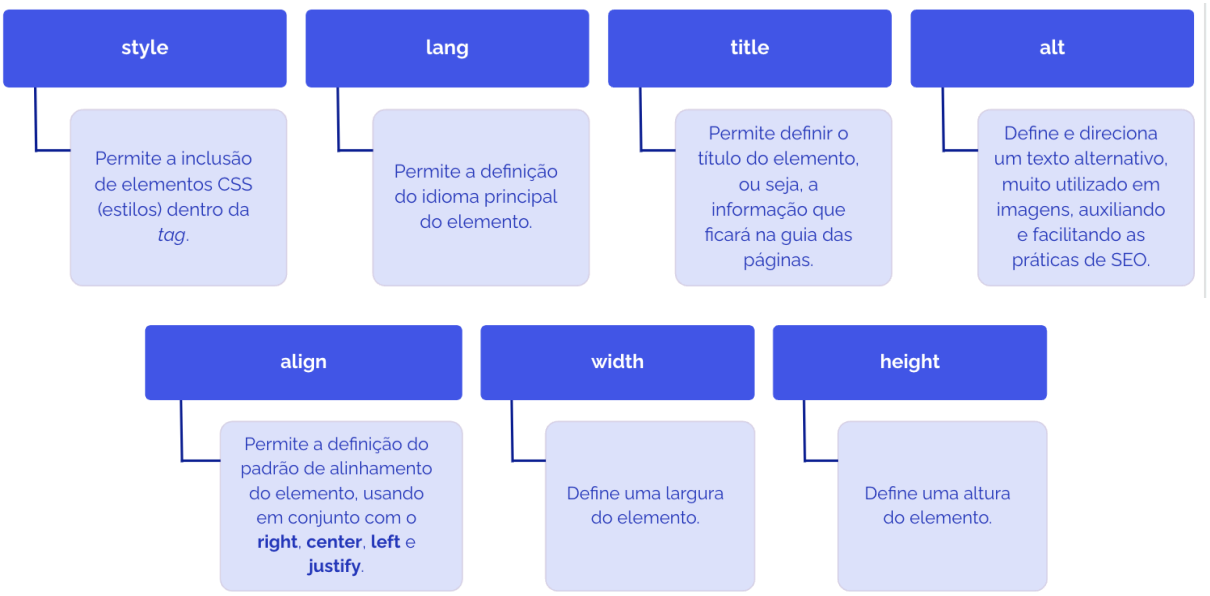
```
</head>
<body>
    asdfsfdfsdfasdfjsjflasjflkasjflasdfaljflsjflskdjfslfj<br>
     <span>box model</span> <br>
    fsldfjskdfjalsdjflsdjflsdjflasjdflsjflsjdfskldjlsf<br>

</body>
</html>
```

Fonte: Elaborado pelo autor (2023).

Atributos

Além das *tags* estruturais, nós também temos os **atributos,** que podem ser usados junto com elas. Os atributos são usados para personalizar as *tags*, modificando a sua estrutura ou funcionalidade. Ou seja, eles são utilizados para atribuir uma **classe** ou **id** a um elemento.

Esses atributos são utilizados com frequência no desenvolvimento *web* e, por isso, é fundamental que você consiga entender como eles devem ser aplicados.



Fonte: Elaborado pelo autor (2023).

Os atributos **id** e **class** têm uma pequena diferença entre si, que são:

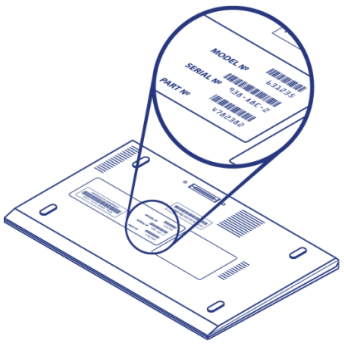
- **class:** uma classe pode ser utilizada para um ou mais elementos, sendo, portanto, um atributo que se identifica de uma forma geral;
- **id:** um *id* deve ser único, ou seja, ele só pode ser atribuído a um único elemento, sendo um identificador único.

Para entender melhor, observe o exemplo abaixo:

Código de barras

Um código de barra pode indicar o preço e a quantidade em estoque, informações relacionadas a todos os produtos do mesmo tipo.

código de barra = *class*



Número de série

Já o número de série é único do produto, não trazendo consigo informações gerais.

número de série = *id*

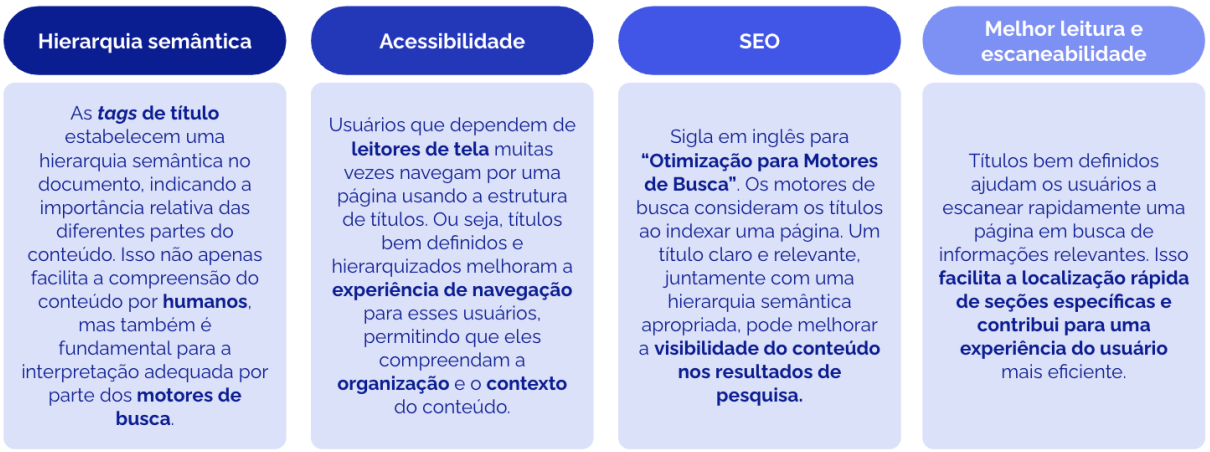
É importante reforçar que essa diferença ficará bem mais clara quando iniciarmos o uso de estilos em CSS.

2.3 *Tags* de texto e título

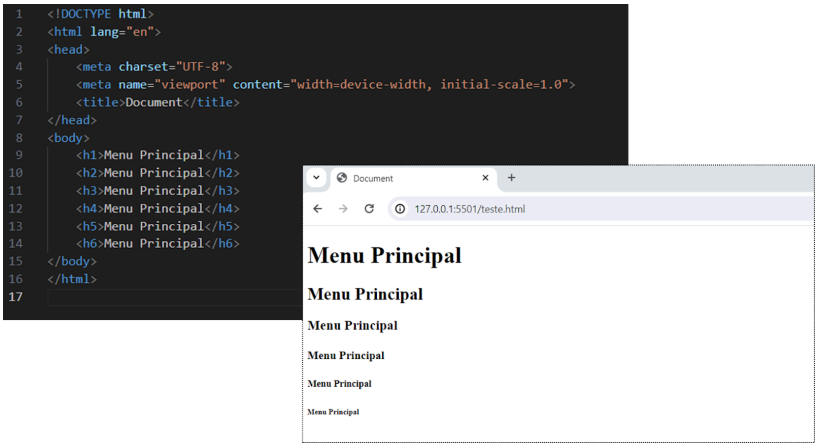
Tags de título

Em HTML, as *tags* de título são elementos usados para estruturar e identificar a hierarquia de informações em uma página *web*. Isso os torna cruciais, principalmente para identificar o título principal da página, bem como os títulos de seções e subseções. Elas são representadas pelos elementos de **<h1>** a **<h6>**, sendo **<h1>** o título mais importante e **<h6>** o menos importante.

A importância das *tags* de título é muito mais que formatação visual, pois elas desempenham um papel crucial na acessibilidade, na otimização para motores de busca e na compreensão semântica do conteúdo. Veja alguns fatores importantes sobre essas *tags* a seguir:



Para entender melhor, observe o exemplo abaixo:



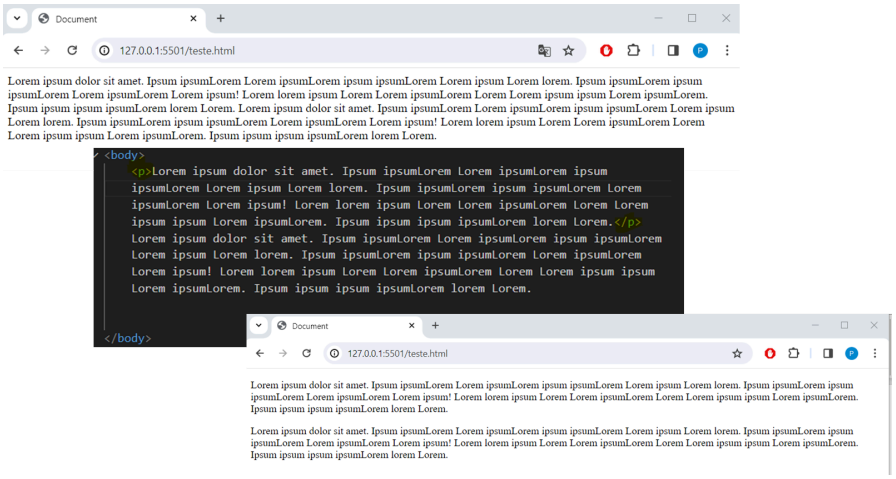
Fonte: Elaborado pelo autor (2023).

- **<h1></h1>**: título de maior valor hierárquico;
- **<h2></h2>;**
- **<h3></h3>;**
- **<h4></h4>;**
- **<h5></h5>;**
- **<h6></h6>**: título de menor valor hierárquico.

Tags de texto

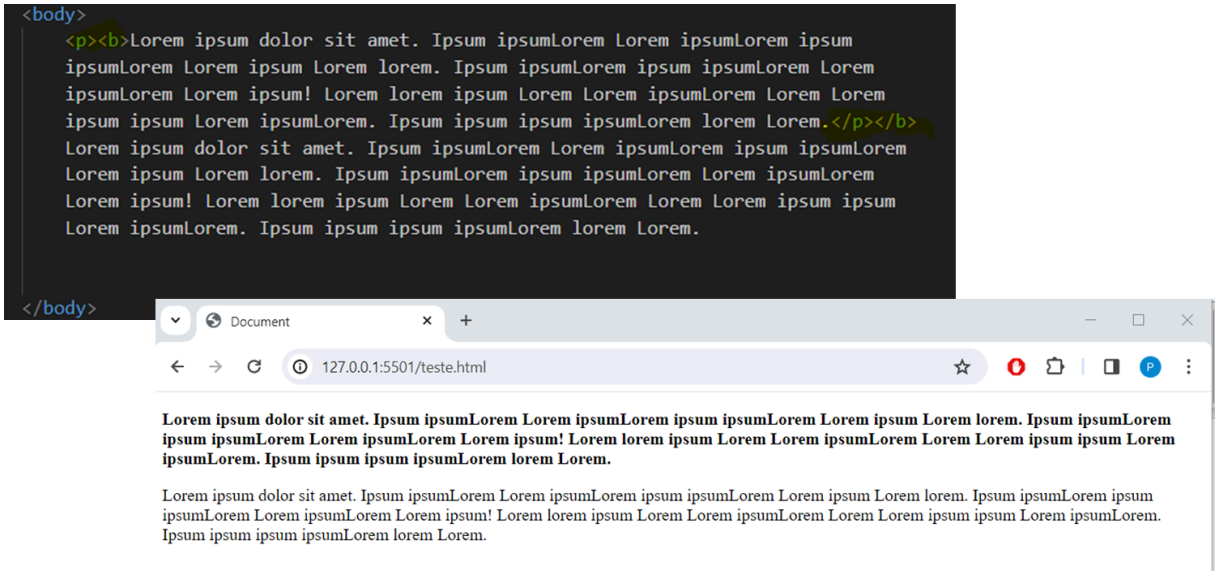
Esse tipo de *tag* define os aspectos do estilo do texto, como estilos de fonte, parágrafos, quebras de linhas etc. Vamos descobrir como elas podem ser utilizadas?

- **<p>:** *tag* de texto que compõe parágrafos;



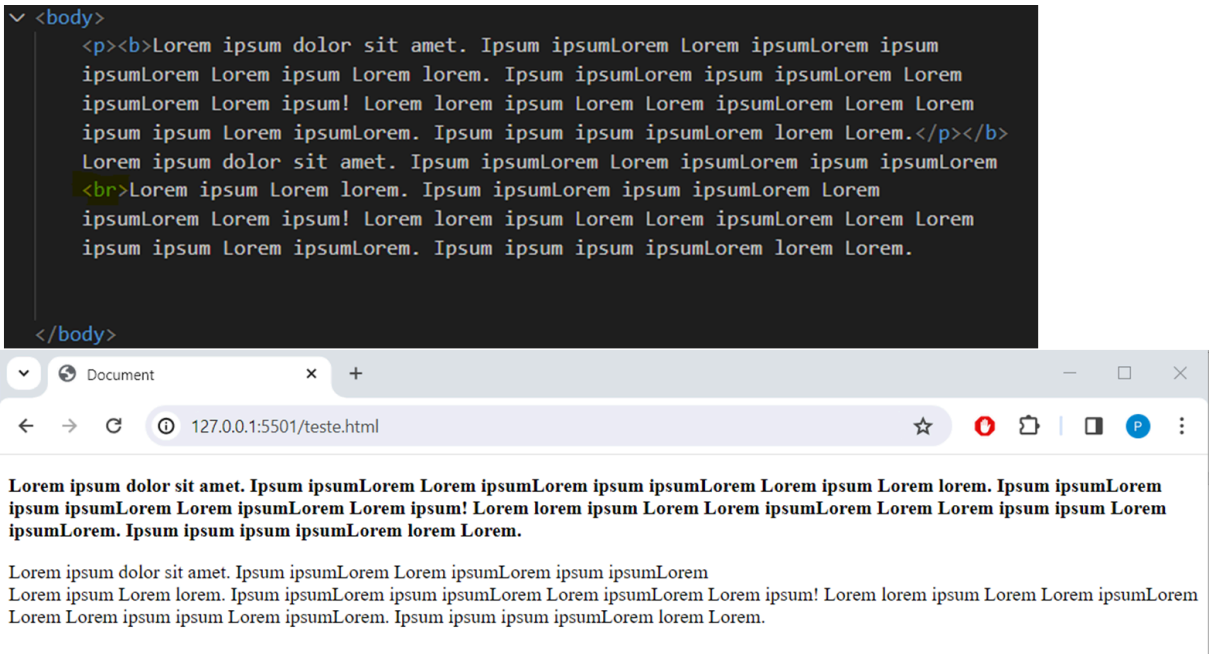
Fonte: Elaborado pelo autor (2023)

- **<pre>:** *tag* utilizada para representar texto pré-formatado. Ela é bastante utilizada para inserir códigos;
- **<i>:** essa *tag* transforma o conteúdo do texto em itálico;
- **<b>:** essa *tag* transforma o conteúdo do texto em negrito;



Fonte: Elaborado pelo autor (2023)

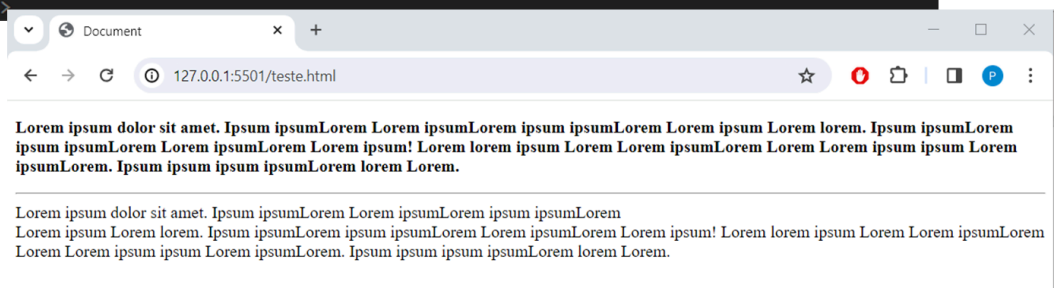
- **
:** essa *tag* executa a função de quebra de linha. Ela não precisa de uma *tag* de fechamento;



Fonte: Elaborado pelo autor (2023)

- **<hr>:** essa *tag* forma uma linha horizontal na página. Ela não precisa de uma *tag* de fechamento.

```
<body>
    <p><b>Lorem ipsum dolor sit amet. Ipsum ipsumLorem Lorem ipsumLorem ipsum
    ipsumLorem Lorem ipsum Lorem lorem. Ipsum ipsumLorem ipsum ipsumLorem Lorem
    ipsumLorem Lorem ipsum! Lorem lorem ipsum Lorem Lorem ipsumLorem Lorem Lorem
    ipsum ipsum Lorem ipsumLorem. Ipsum ipsum ipsum ipsumLorem lorem Lorem.</p></b>
    <hr>
    Lorem ipsum dolor sit amet. Ipsum ipsumLorem Lorem ipsumLorem ipsum ipsumLorem
    <br>Lorem ipsum Lorem lorem. Ipsum ipsumLorem ipsum ipsumLorem Lorem
    ipsumLorem Lorem ipsum! Lorem lorem ipsum Lorem Lorem ipsumLorem Lorem Lorem
    ipsum ipsum Lorem ipsumLorem. Ipsum ipsum ipsum ipsumLorem lorem Lorem.


</body>
```



Fonte: Elaborado pelo autor (2023).

2.4 Conjunto de listas

```
<nav>
    <h1>Menu Principal</h1>
    <ul>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>
    </ul>
</nav>
```

Fonte: Elaborado pelo autor (2023)

Tags <nav>

Como já falamos anteriormente, em HTML, é possível trabalhar com vários tipos de elementos semânticos. Entre eles, nós temos a *tag* **<nav>**, que define um conteúdo de navegação. Geralmente, existem *links* para outras páginas ou áreas importantes do *site* nas seções dessa *tag* e o seu uso contribui para uma estrutura semântica clara,

facilitando a identificação da navegação por leitores de tela e motores de busca, sendo muito utilizada em conjunto com listas e na criação de menus.

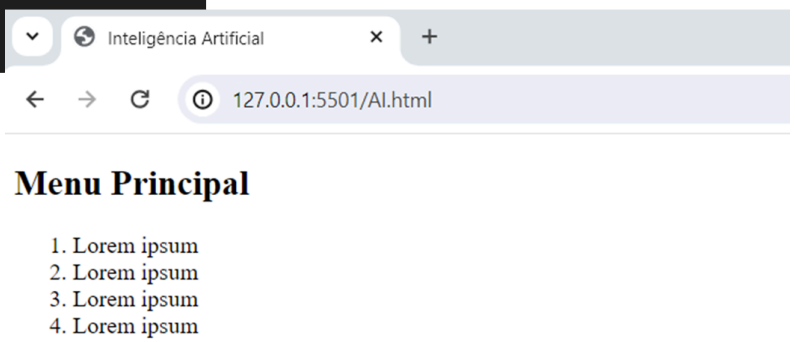
Porém, a *tag* **<nav>** não se limita a produzir efeitos visuais apenas com o HTML. A sua grande utilidade se revela ao ser combinada com o CSS, especialmente na organização dos menus.

Listas ordenadas e não ordenadas

As listas são utilizadas para apresentar informações de maneira sequencial e hierárquica, indicando uma ordem específica entre os itens. Elas podem ser ordenadas e não ordenadas, e o que diferencia os tipos de listas são as seguintes *tags*:

- **<li>:** cada item é representado por **<li>** (item de lista) e é automaticamente representado pela ordenação ou marcadores, dependendo somente do tipo de lista usada;
- **<ol>:** quando a *tag* é declarada desta forma, define-se a lista como ordenada. Ou seja, a ordem poderá ser classificada por números ou letras. Veja os exemplos abaixo:

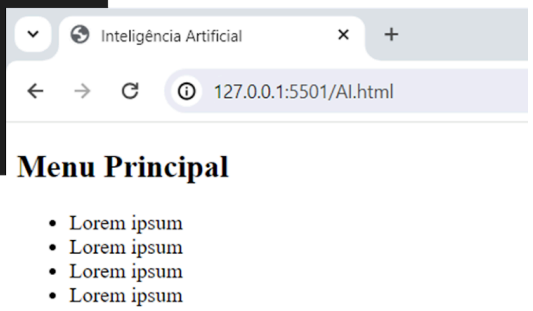
```
<nav>
    <h1>Menu Principal</h1>
    <ol>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>
        <li>Lorem ipsum</li>

    </ol>
```



Fonte: Elaborado pelo autor (2023)

- **<ul>:** quando a *tag* é declarada desta forma, define-se a lista como não ordenada. Ou seja, a ordem será representada por marcadores, como *bullets*. Veja o exemplo abaixo:

```
<h1>Menu Principal</h1>
<ul>
    <li>Lorem ipsum</li>
    <li>Lorem ipsum</li>
    <li>Lorem ipsum</li>
    <li>Lorem ipsum</li>

</ul>
```



Fonte: Elaborado pelo autor (2023)

ATIVIDADE DE FIXAÇÃO

1. Qual *tag* HTML é usada para criar uma lista não ordenada?
    a. **<ul>**
    b. **<ol>**
    c. **<li>**
    d. **<list>**

2. O que a *tag* **<p>** representa em HTML?
    a. Parágrafo.
    b. *Link*.
    c. Lista.
    d. Imagem.

3. Qual *tag* deve ser utilizada para criar um título principal em HTML?
    a. **<head>**
    b. **<title>**
    c. **<h1>**
    d. **<main>**

4. Qual das alternativas apresenta exemplos de elementos semânticos em HTML?
    a. ** ** e ****
    b. **<header>** e **<nav>**
    c. **<table>** e **<form>**
    d. **
** e **<hr>**

5. Qual é a função da *tag* **<nav>** em HTML?
    a. Criar uma lista de navegação.
    b. Definir o cabeçalho da página
    c. Formatar texto como negrito
    d. Adicionar uma imagem à página

6. Como adicionar uma linha horizontal a uma página em HTML?
    a. **<line>**
    b. **<hr>**
    c. **<linebreak>**
    d. **<horizontal>**

7. Fale sobre a *tag* **<meta>** e sobre a sua importância.

8. Ao desenvolver o conteúdo de um *blog*, você quer destacar a seção que contém a informação principal do artigo. Qual elemento semântico HTML é mais apropriado para essa finalidade?
    a. ** **
    b. **<section>**
    c. **<article>**
    d. **<main>**

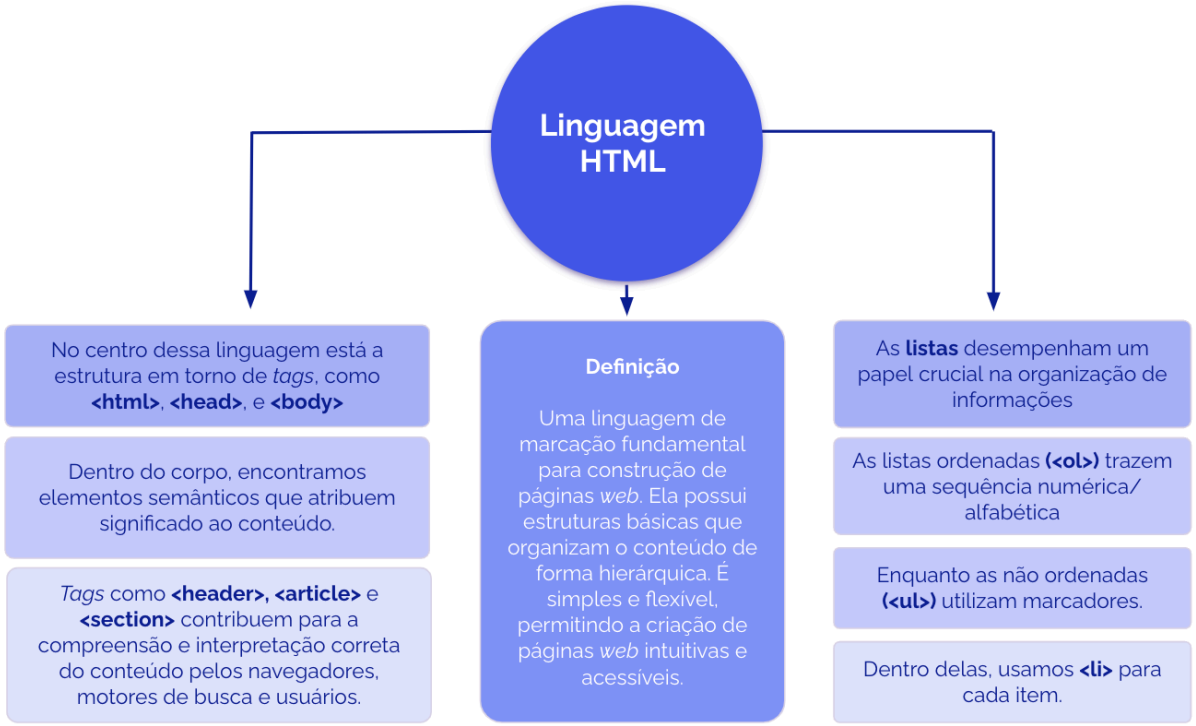
9. Explique os atributos da *tag* **<link>**.

10. Você está criando uma página *web* e precisa definir corretamente as estruturas básicas do HTML. Qual é a *tag* que você deve usar para indicar o início do documento HTML?
    a. **<start>**
    b. **<html>**
    c. **<head>**
    d. **<body>**

RESUMO

Estudamos que as linguagens de marcação, como o HTML e o CSS, desempenham papéis fundamentais na criação de páginas vivas e que o HTML, mesmo que não seja uma linguagem de programação, é crucial para estruturar o conteúdo da *web*. Também abordamos que no HTML5, a sua versão mais recente, surgiram melhorias, como APIs avançadas, uma semântica aprimorada e o suporte nativo para mídia e vimos que a estrutura básica do HTML é composta por *tags,* como **<html>**, **<head>**, e **<body>**.

Além disso, nós compreendemos que *tags* como ** ** e ****, juntamente com os atributos como **class** e **id**, desempenham papéis importantes na estruturação e na estilização das páginas. Também entendemos que as *tags* de título (**<h1>** a **<h6>**) e as *tags* de texto (**<p>, <i>, <b>,
, <hr>** etc.) são fundamentais para as hierarquias e para a formatação de texto, enquanto as listas ordenadas (**<ol>**) e não ordenadas (**<ul>**) organizam informações de maneira sequencial ou hierárquica.



CAPÍTULO 03

Estruturando outras *tags* de conteúdo

O que esperar deste capítulo:

- Dominar a marcação HTML produzindo estruturas e *layouts* de páginas da *web*;
- Criar *layouts* com conjunto de *tags* estruturais;

3.1 Trabalhando com *link* em geral

Quando trabalhamos com a linguagem HTML, percebemos que os *links* são recursos de grande importância no desenvolvimento *web*, pois são eles que permitem a **interconexão** entre as diferentes páginas de um *site* e, até mesmo, da *web*. Essa capacidade de vincular páginas é fundamental para a estrutura e a navegação na internet e, por isso, é importante entender como trabalhar com os *links*, para que você, enquanto desenvolvedor, consiga criar páginas que possibilitem uma experiência ao usuário fluida durante a utilização das páginas na internet.

O que são *links* em HTML?

Os *links*, também conhecidos como **âncoras** ou ***hiperlinks***, são elementos HTML que possibilitam a navegação entre páginas, recursos e conteúdos *on-line*. Eles podem **direcionar o usuário** para diferentes seções de uma página, outros documentos HTML, imagens, vídeos ou, até mesmo, para *sites* externos. No entanto, nós devemos sempre considerar que, quando um *link* direcionar para uma outra página, será necessário descrever a URL (endereço) existente em seu *site*, pois, caso contrário, a página retornará uma mensagem de **erro 404**.

Agora que você já entendeu o que são os *links* e qual é a sua função no desenvolvimento *web*, vamos entender como eles são criados.

Nós utilizamos a *tag* HTML **<a>**, abreviação de **"*anchor*" (âncora)**, para criar os *links*. Uma vez criado, será preciso estruturar o *link* e isso é feito incluindo o texto ou o elemento que servirá como redirecionador, utilizando o atributo **href** (*Hypertext Reference*). Feito isso, o *link*, ao ser clicado, executará a função de redirecionar o usuário para o endereço dentro do atributo **href**.

Veja como fica a sintaxe de um *link* para a página a seguir.

Exemplo

Fonte: Elaborado pelo autor (2024)

Geralmente, os *links* têm uma cor azul e são apresentados em uma página com um sublinhado sobre o nome, indicando que o usuário pode clicar sobre ele. Porém, esses padrões podem ser alterados por meio do CSS (*Cascading Style Sheets*), assunto que será abordado mais adiante.

Atributo href

Assim como já estudamos anteriormente, o atributo principal da *tag* **<a>** é o **href="".** Com ele, nós podemos redirecionar o usuário para outras páginas ou recursos, como documentos, vídeos etc. Existem três formas de se trabalhar o **href** para realizar o redirecionamento. Elas são:

- **redirecionamento interno**: redireciona para um elemento existente dentro da mesma página. No entanto, é necessário utilizar como recurso o atributo **id**, que descreve para onde o *link* será direcionado. Essa forma é muito utilizada quando se tem muitas seções em uma mesma página;

```
1 <a href="#secao1">Link para a seção 1</a>
```

Disponível em: <http://tinyurl.com/mruf6he3>. Acesso em 04 de jan. 2024.

- **redirecionamento local**: utilizado para páginas contendo o mesmo domínio, ou seja, entre páginas do mesmo *site*;

```
<a href="teste.html'>HOME</a>
```

Fonte: Elaborado pelo autor (2024)

- **redirecionamento global**: utilizado para páginas de outros domínios, ou seja, páginas de fora do *site*.

```
<a href="http://google.com">Google</a>
```

Fonte: Elaborado pelo autor (2024)

Atributo target

Um outro atributo utilizado juntamente com a *tag* **<a>** é o **target**. Ele informa ao navegador como o redirecionamento deverá ocorrer, ou seja, se a página será aberta na mesma janela ou aba do navegador, ou em uma nova janela ou aba.

Para compor a sintaxe de uso do atributo **target,** será necessário atribuir quais valores ou parâmetros eles terão, permitindo o direcionamento desejado. Vamos conhecer quais são eles a seguir:

- **_blank:** abre a página em uma nova janela ou aba;

```
<a href="http://google.com" target="_blank">Google</a>
```

Fonte: Elaborado pelo autor (2024)

- **_self:** abre a página na mesma janela ou aba;
-

```
<a href="http://google.com" target="_self">Google</a>
```

Fonte: Elaborado pelo autor (2024)

- **_parent:** abre a página na mesma janela do *link*;

```
<a href="http://google.com" target="_parent">Google</a>
```

Fonte: Elaborado pelo autor (2024)

- **_top:** cancela todos os demais *frames* e abre a nova página no mesmo navegador.

```
<a href="http://google.com" target="_top">Google</a>
```

Fonte: Elaborado pelo autor (2024)

Atributo download e links em imagens

Dentre as outras possibilidades de *link,* nós podemos, ainda, usá-lo para redirecionar o usuário para um *download.* Além disso, também é possível fazer com que uma determinada imagem seja atribuída com um *link*.

Para utilizar o atributo para *download*, o primeiro passo é saber onde o arquivo que deverá ser baixado está armazenado. Ou seja, será necessário descrever o caminho em que o arquivo se encontra hospedado para que o *link* redirecione o usuário para o *download* de um arquivo. Com esse atributo, podemos definir, também, um nome padrão para o arquivo que será baixado. Por exemplo: **Google.** Nesse caso, podemos perceber que a sintaxe HTML indica que o *download* será de uma imagem e que ela está localizada na pasta **img**.

Uma outra forma possível de se trabalhar com os *links* é **em blocos.** Ou seja, uma vez que o elemento **<a>** não possui nenhum impedimento com a inserção de blocos dentro dele, podemos inserir um *link* em uma imagem. No exemplo abaixo, foi feito o *link* de uma imagem com o *site* do Google. Além disso, também é possível perceber que foi feito o dimensionando da imagem, definindo a sua largura (*width*) e a sua altura (*height*).

```
<a href="https://www.Google.com/" target="_blank"><img width="30" height="30" src="img/img2.jpg"> </a>
```

Fonte: Elaborado pelo autor (2024)

3.2 Inserindo imagens

Até agora, nós já vimos muitos recursos que nos permitem elaborar uma página. Porém, sabemos que uma página *web* não é feita somente de textos e *links*. Dentre outros aspectos, é preciso considerar os tipos de usuários que irão acessar a página ou o *site*. Por isso, é necessário tornar o *site* atrativo para diversos públicos e as imagens são excelentes recursos para incrementar o *layout* e o visual de uma página, tornando-a visualmente agradável para quem acessa.

Porém, antes de entendermos como vincular uma imagem, é preciso saber de algumas informações bastante relevantes.

Dicas importantes para o uso de imagens no desenvolvimento *web*

Ao se trabalhar com imagens, é essencial conhecer os formatos existentes, pois cada um possui aplicações e características próprias, que implicam em vantagens e desvantagens. Mas, para tornar nosso estudo mais produtivo, vamos focar nos formatos mais utilizados para a criação de *sites*, que são o **JPG** (ou **JPEG**) e o **PNG**.

JPG ou JPEG

O formato JPEG gera arquivos pequenos e que ocupam pouco espaço em disco. Isso é especialmente importante ao colocar o *site* no ar, pois permite que a página fique mais leve, carregando as imagens rapidamente.

PNG

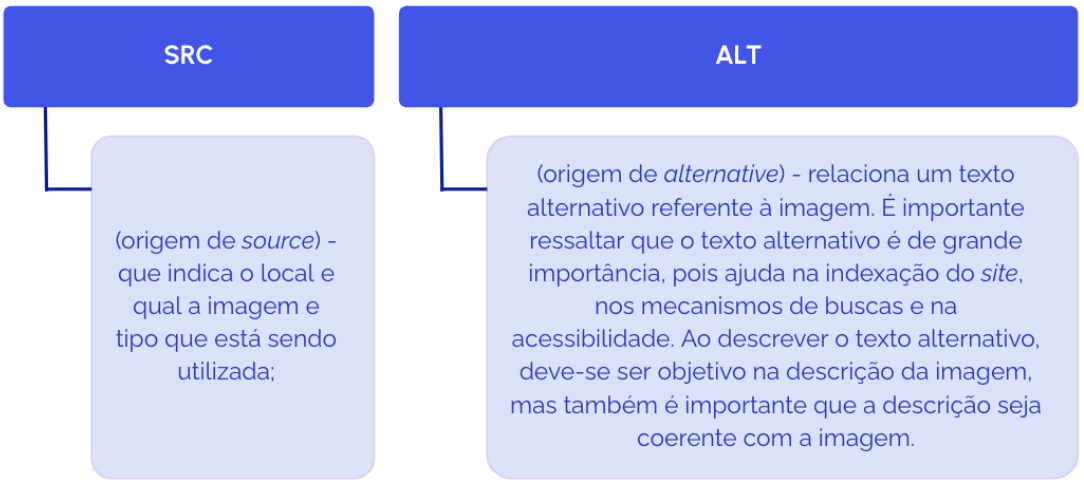
A sua principal característica é a capacidade de configurar a opacidade da imagem. Ou seja, esse formato permite deixar a imagem transparente ou limitar a sua transparência.

Para ajudar com seus estudos, veja o quadro abaixo, em que estão listadas as principais diferenças entre esses formatos:

Disponível em:<http://tinyurl.com/m7jbjkzk>. Acesso em: 08 jan. 2024.

Trabalhando com as imagens

Agora, nós vamos entender como inserir uma imagem em uma página. Para isso, vamos usar a *tag* **<img>**, que é responsável por carregar a imagem desejada, seja ela da pasta do projeto e ou de *links* externos. Seus atributos básicos são:



Fonte: Elaborado pelo autor (2024)

```
<img src="img/img2.jpg" alt="Pessoas trabalhando no computador"/>
```

Fonte: Elaborado pelo autor (2024)

Ícone de favoritos

Outro recurso muito utilizado e que tem relação com imagens são os ***favicon***, pequenos **ícones que aparecem ao lado dos nomes dos *sites*** que visitamos.

Fonte: Elaborado pelo autor (2024)

Para implementar o *favicon,* é necessário possuir o arquivo do ícone desejado. Comumente, esses arquivos estão no formato **ICO**, embora outros formatos, como **PNG** e **SVG**, sejam aceitáveis para a sua implementação.

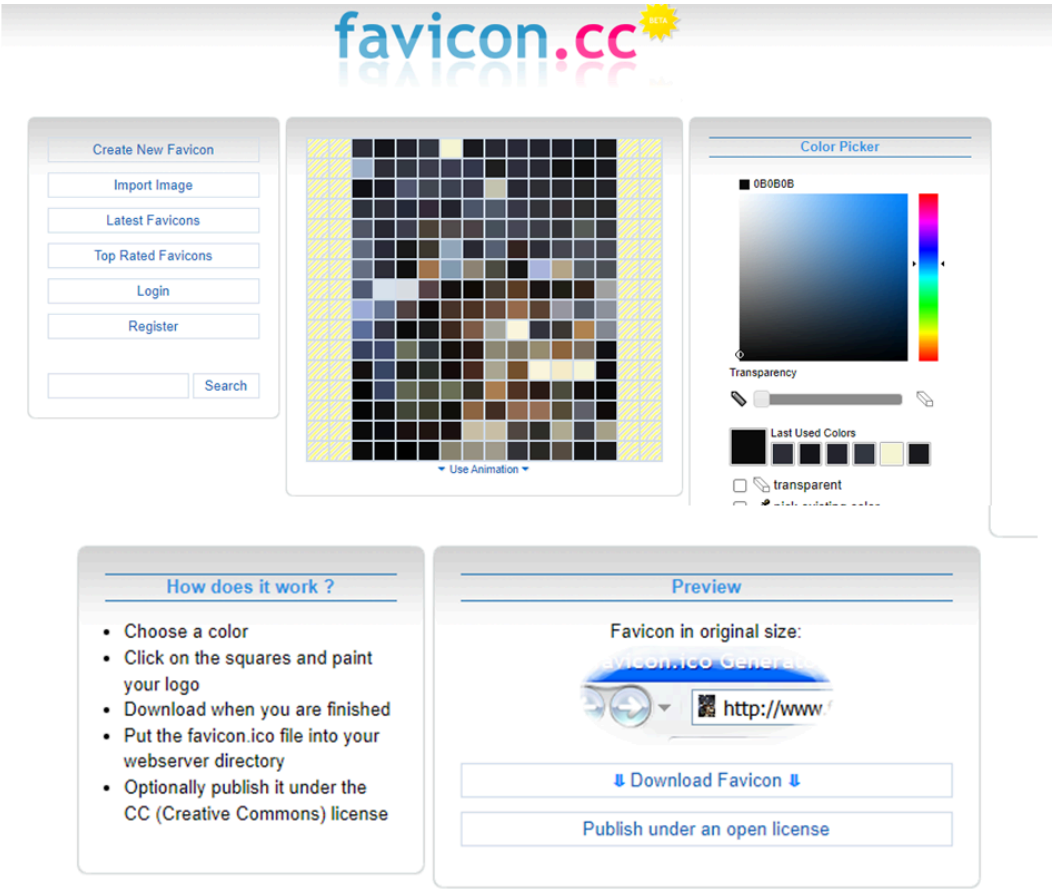
Para criar o seu *favicon*, acesse o *site* **favicon.cc**.

Disponível em: <https://www.favicon.cc/>. Acesso em: 04 jan. 2024.

Em seguida, faça o *upload* da imagem clicando em ***Import Image*.** Na sequência, escolha o ficheiro, procure pela sua imagem e, após encontrá-la, clique em *upload*:

Disponível em: <https://www.favicon.cc/>. Acesso em: 04 jan. 2024.

Depois que o *upload* estiver concluído, uma tela com a imagem *pixelada* será apresentada. Em seguida, clique no botão "***Download Favicon***" para baixar o ícone. Veja na imagem abaixo:



Disponível em: <https://www.favicon.cc/>. Acesso em 04 de jan. 2024.

Após salvar o arquivo **favicon.ico** na pasta IMG do seu projeto, você deve descrever o *link* para que funcione junto ao título que você definiu com o atributo **<title>.** Para isso, siga as informações abaixo:

Essa linha de comando deve estar dentro da *tag* **<head>**, mas deverá ficar fora do atributo **<title>**.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Novidades da Tecnologia</title>
<link rel="icon" type="image/icon" href="img/icon.ico"/>

</head>
```

Fonte: Elaborado pelo autor (2024)

Agora, vamos entender melhor a utilidade de cada atributo dentro da *tag* **<link>** com base na imagem acima:

- **rel**: indica o tipo de relação entre o documento atual e o recurso vinculado. No caso do exemplo, **"icon"** significa que está sendo referenciado um ícone;
- **type**: define o tipo de conteúdo do recurso vinculado. No exemplo, uma imagem que parte de um ícone criado;
- **href**: este atributo especifica o caminho do arquivo que será vinculado.

DESAFIO PRÁTICO

Estrutura básica de um documento HTML5

Desafio: Aprimorando o projeto.

Descrição

Após passar nos testes solicitados, você garantiu uma vaga para trabalhar com o desenvolvimento *web*. A sua primeira tarefa é verificar a correção das estruturas de listas, *links* e imagens em um *site* em desenvolvimento. Além disso, é necessário analisar o *layout* de todas as páginas do *site* antes de avançar para o próximo estágio de desenvolvimento.

Objetivos

- Demonstrar conhecimento com estruturas de listas, *links* e imagens;
- Trabalhar com diferentes tipos de inserção de *links* e imagens;
- Salvar o projeto em um repositório com as devidas alterações realizadas.

Orientações

- Organize o projeto criando uma pasta específica para armazenar as imagens que serão utilizadas;
- Utilize o projeto criado no desafio prático anterior como referência. Busque em *sites* que disponibilizam imagens de domínio público pelo menos três imagens relacionadas ao seu projeto;
- Faça o *download* das imagens e salve-as na pasta **IMG**;
- Integre as imagens ao seu projeto, incluindo uma delas como *link* para o *site* de origem;
- Adicione dois *links* para *sites* distintos que, ao serem clicados, devem abrir novas abas de navegação.

3.3 *Layout* de tabela

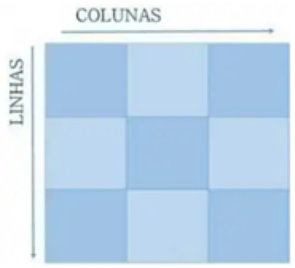
Com certeza você já deve ter criado uma tabela usando o Word ou o Excel, não é mesmo? Elas são muito usadas para separar e categorizar informações, deixando a apresentação do seu conteúdo mais organizada. Agora, imagine uma escola que conta com milhares de alunos. Como a coordenação da instituição consegue lidar de forma eficiente com a identificação de todos os alunos matriculados? Certamente você deve ter pensado em uma tabela no estilo do Excel, em que cada coluna contém informações como nome do aluno, turma e número de matrícula. Dessa forma, o controle e organização das informações dos alunos matriculados se torna muito mais efetiva.

Na imagem a seguir, podemos entender como as tabelas são formadas.

NOME	IDADE	PROFISSÃO
Eduardo	25	Motorista
Camila	32	Gerente de Vendas
Wesley	18	Ator

Fonte: Elaborado pelo autor (2024)

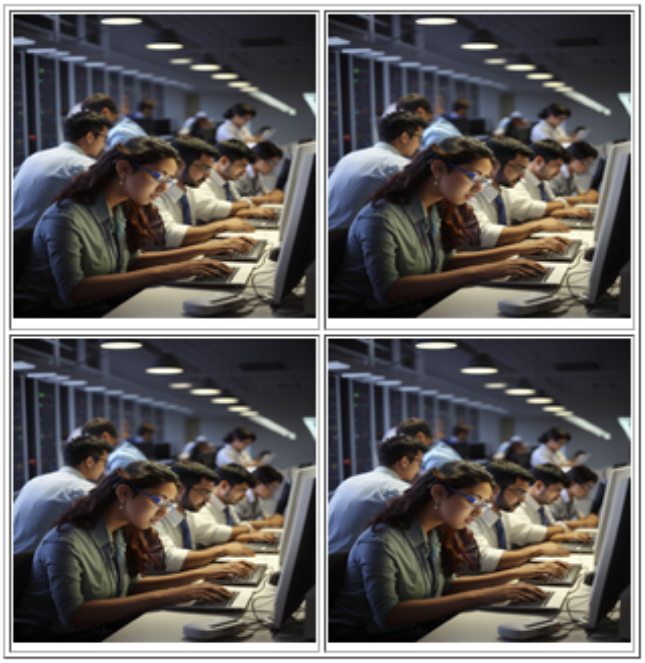
Podemos perceber que tabelas são, na verdade, um quadro dividido em **linhas e colunas**. A **intersecção** (ou o encontro) de uma linha com uma coluna é chamada de **célula**.



Disponível em: <https://www.homehost.com.br/blog/criar-sites/tabela-html/>. Acesso em: 04 jan. 2024.

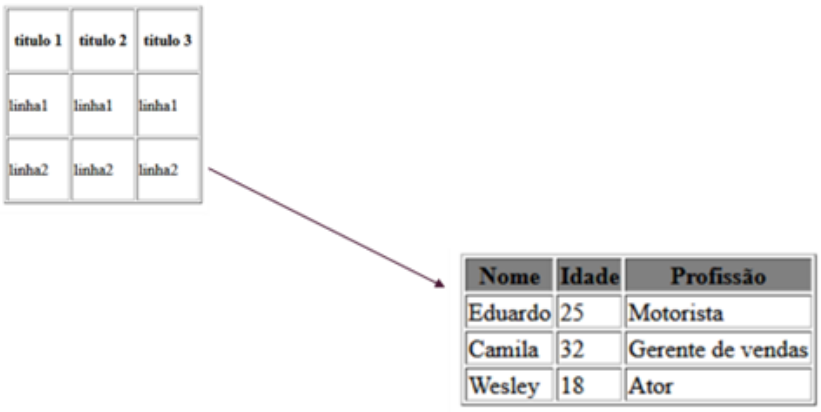
Com o uso de tabelas, nós podemos apresentar várias informações de maneira organizada e isso também se aplica ao HTML, pois elas permitem recursos interessantes nessa linguagem, principalmente na organização de *layouts*. Porém, vale lembrar que é preciso tomar cuidado para não colocar muitas tabelas nas páginas, já que isso tornaria o *site* poluído visualmente. Vamos conhecer alguns exemplos de como utilizar tabelas de forma eficaz em sua página?

- Para organizar fotos:



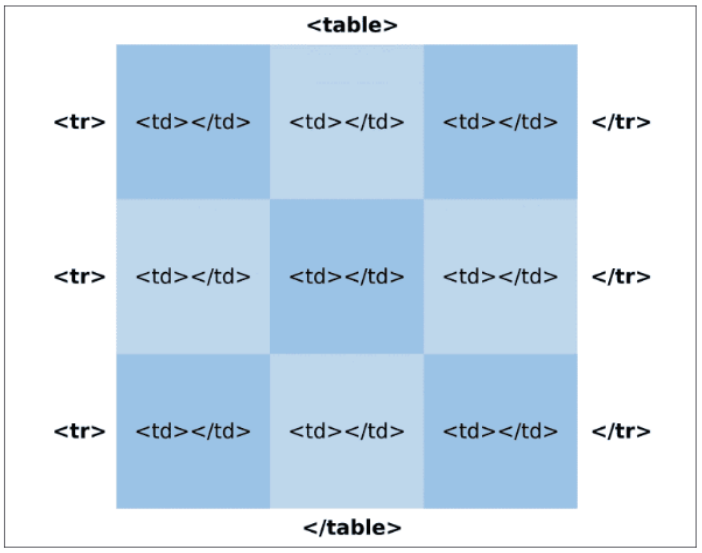
Fonte: Elaborado pelo autor (2024)

- Para definição de tabelas, como preços, lista de produtos, entre outras possibilidades:



Fonte: Elaborado pelo autor (2024)

- *Tags* para criar tabelas:



Disponível em: <https://www.homehost.com.br/blog/criar-sites/tabela-html/>. Acesso em: 04 jan. 2024.

Para iniciar a criação de uma tabela, a *tag* utilizada é a **<table></table>** e as demais *tags* que formam a estrutura da tabela deverão ser trabalhadas dentro dela. As *tags* utilizadas para criar linha e coluna são, respectivamente:

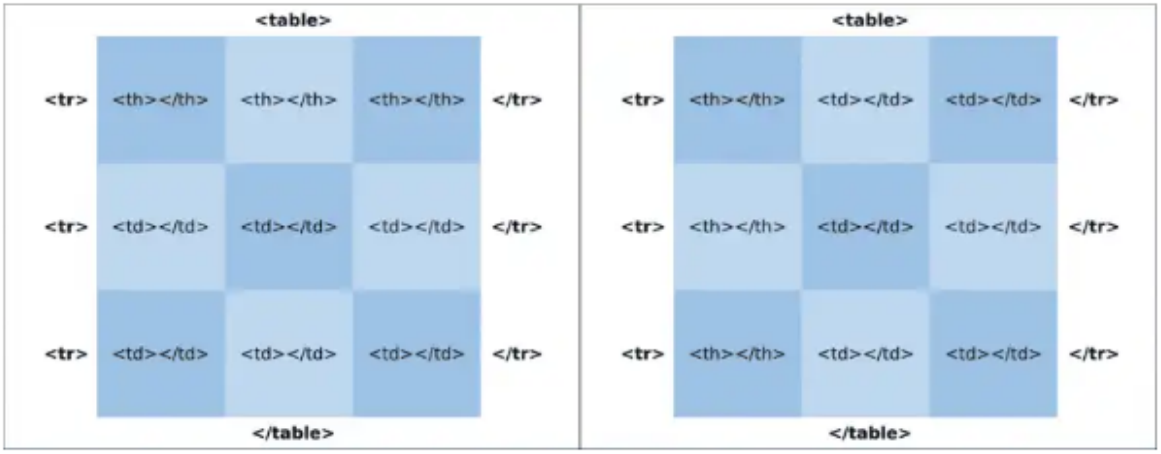
- ***tag* <tr>**: que significa *table row*, usada para definir uma linha;
- ***tag* <td>**: que significa *table data*, usada para criar uma célula;

Para melhor entendimento, faça o exemplo abaixo no Visual Code:

```
<table>
    <tr>
        <td>Nome</td>
        <td>Idade</td>
        <td>Profissão</td>
    </tr>
    <tr>
        <td>Ted</td>
        <td>22</td>
        <td>Estudante</td>
    </tr>
    <tr>
        <td>Ralf</td>
        <td>26</td>
        <td>Designer</td>
    </tr>
</table>
```

Fonte: Elaborado pelo autor (2024)

- ***tag* <th>**: utilizadas para incluir células que representam os títulos das tabelas. Elas apresentam um destaque ao conteúdo em relação às demais células;



Disponível em: <https://www.homehost.com.br/blog/criar-sites/tabela-html/>. Acesso em: 04 jan. 2024.

Para melhor entendimento, faça o exemplo abaixo no Visual Code:

```
<table>
        <tr>
            <th>Nome</th>
            <th>Idade</th>
            <th>Profissão</th>
        </tr>
        <tr>
            <td>Ted</td>
            <td>22</td>
            <td>Estudante</td>
        </tr>
        <tr>
            <td>Ralf</td>
            <td>26</td>
            <td>Designer</td>
        </tr>
```

Fonte: Elaborado pelo autor (2024)

3.4 Estrutura de formulários

No decorrer do desenvolvimento de uma página ou de um *site,* sempre surge a necessidade de inserir alguma interação com o usuário através de um elemento. Ao falar em interação, estamos nos referindo à necessidade de apresentar algo ao usuário, como uma pergunta, permitindo que ele se manifeste. Nesse caso, fornecendo uma resposta.

Uma das formas de interação, ainda que inicial, é a elaboração de formulários. Esse recurso pode ser usado para realizar a **coleta de informações** de acordo com a necessidade e do conteúdo do *site*, como realizar cadastros, receber opiniões etc. Veja como fica um modelo de formulário de cadastro em uma página *web*:

Fonte: Elaborado pelo autor (2024)

O exemplo acima possui praticamente todas as *tags* e todos os atributos para a construção de vários tipos de formulários. Agora, acesse o Visual Studio e acompanhe cada um dos passos, pois vamos aprender a desenvolver um formulário para você entender bem como utilizar esse recurso.

A primeira ação a ser feita no Visual Studio é abrir um **novo arquivo** dentro do seu projeto, com o nome de **Form.html**. Lembre-se que a criação de um arquivo e a estrutura inicial do HTML você já conhece. Após essa primeira ação, para iniciar um formulário, **incluímos as t*ags* <form> </form>**.

```
form.html > ...
1   <!DOCTYPE html>
2   <html lang="pt=br">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Formulario</title>
7   </head>
8   <body>
9       <form>
10
11      </form>
12  </body>
13  </html>
14
```

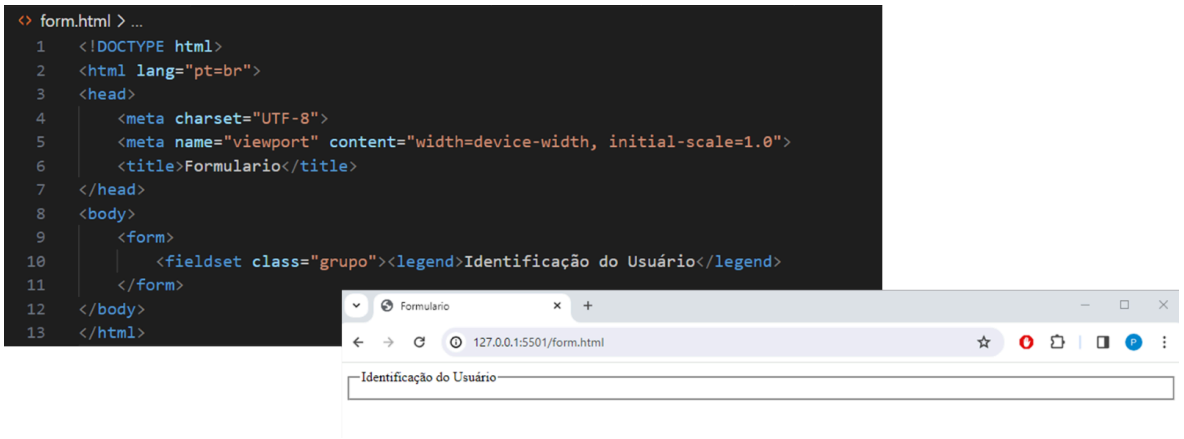
Fonte: Elaborado pelo autor (2024)

Agora, vamos começar a **incluir o conteúdo do formulário**. Digamos que você queira que o usuário envie informações sobre ele, ou seja, a **identificação do usuário.** Para isso, devemos **criar os campos de preenchimento** utilizando um conjunto de *tags* e atributos. Para tal, reproduza os elementos que estão apresentados a seguir:

- **fieldset**: permite colocar um conjunto de campos dentro de um quadro. Juntamente com essa *tag*, nós podemos utilizar uma **class**;
- **<legend>**: permite identificar o conjunto de campos.

```
<fieldset class="grupo"><legend>Identificação do Usuário</legend>
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

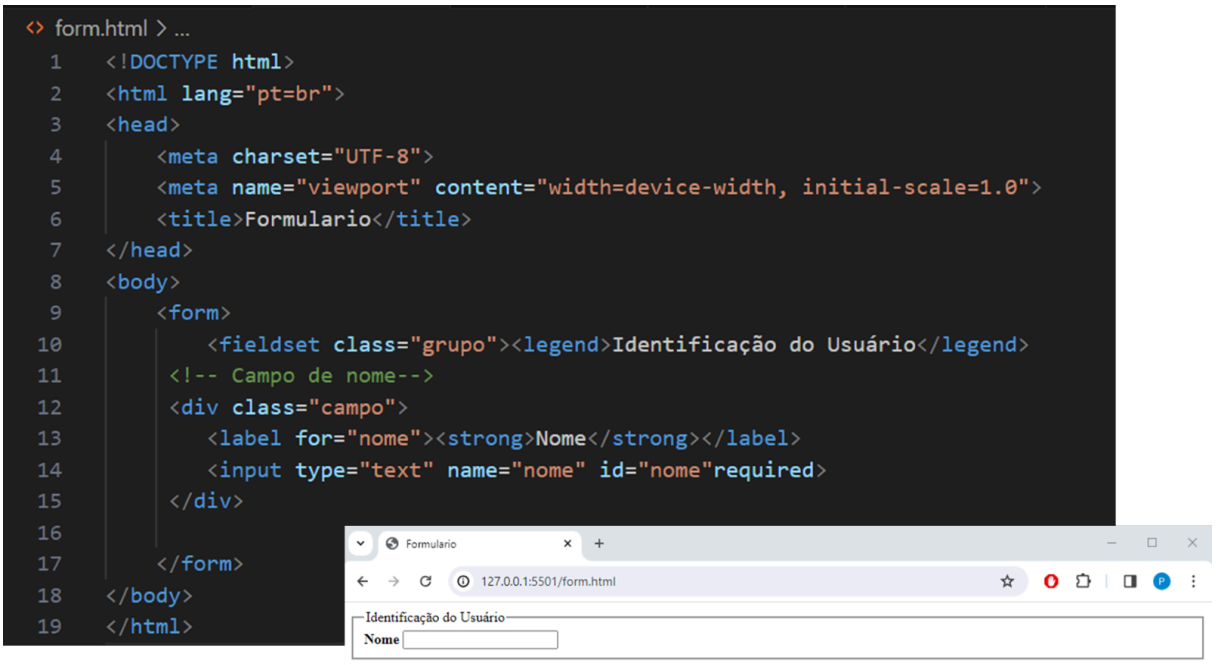
Para criar os campos em que o usuário irá inserir seus dados, vamos utilizar as seguintes *tags* e atributos:

- *tag* **<label>**: essa *tag* define qual será o nome do campo e o atributo **for** liga o nome à caixa de texto criada;
- *tag* ****: identifica o campo;
- *tag* **<Input>**: cria a caixa de texto;
- **type**: define a informação que será inserida na caixa de texto;
- **name** e **ID**: são atributos importantes para relacionar as informações entre o campo e o conteúdo da caixa de texto. Isso será essencial no processo de envio dos dados.

No Visual Studio, siga o exemplo abaixo para criar uma página em que o usuário deve inserir seus dados de identificação.

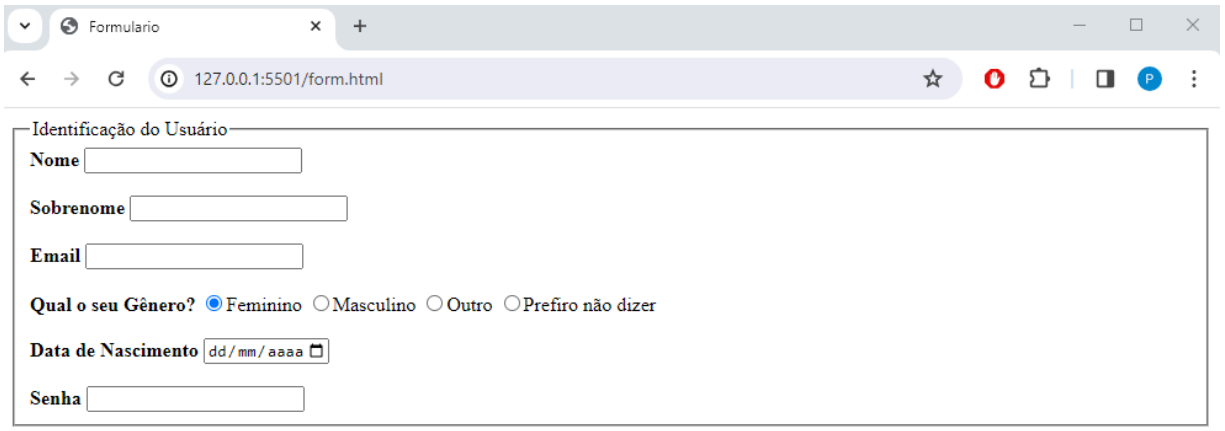
```
<!-- Campo de nome→
      <div class="campo">
      <label for="nome"><strong>Nome</strong></label>
      <input type="text" name="nome" id="nome"required>
      </div>
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

Com essas *tags* e esses atributos, é possível criar os demais campos de identificação, conforme a imagem a seguir:



Fonte: Elaborado pelo autor (2024)

Para realizar essa parte completa, siga, no Visual Studio, os exemplos dos códigos e explicações da *tag* **<input>**, como você pode ver a seguir.

- **<input type="text">**: define um campo que receberá qualquer caractere;

```
<!-- Campo de sobrenome-->
     <div class="campo">
     <label for="sobrenome"><strong>Sobrenome</strong></label>
     <input type="text" name="sobrenome" id="sobrenome" required>
     </div><br>
```

Fonte: Elaborado pelo autor (2024)

- **<input type="email">**: define um campo que receberá caracteres e verificará se ele consiste em um *e-mail* válido;

```
<!-- Campo de email-->
     <div class="campo">
     <label for="email"><strong>Email</strong></label>
     <input type="email" name="email" id="email" required>
     </div><br>
```

Fonte: Elaborado pelo autor (2024)

- **<input type="radio" name="genero" value="feminino" checked>**: permite a opção de definir um dos itens, em que **name** indica que somente um campo poderá ser selecionado por vez e o **checked** para deixar um campo já selecionado;

```
<!-- Campo de Gênero-->
<div class="campo">
     <label><strong>Qual o seu Gênero?</strong></label>
     <label>
     <input type="radio" name="genero" value="feminino" checked >Feminino
     </label>
     <label>
     <input type="radio" name="genero" value="masculino">Masculino
     </label>
     <label>
     <input type="radio" name="genero" value="outro">Outro
     </label>
     <label>
     <input type="radio" name="genero" value="ndizer">Prefiro não dizer
     </label>
</div><br>
```

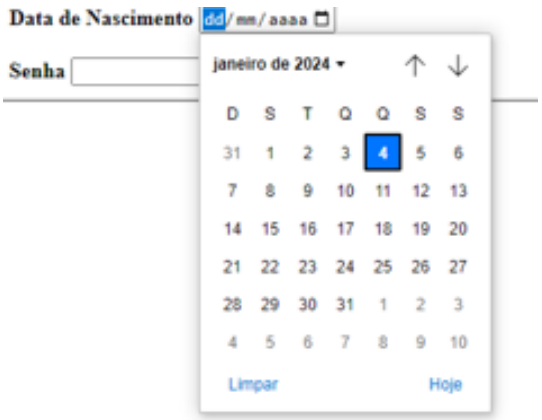
Fonte: Elaborado pelo autor (2024)

- **<input type="date" min="1920-01-01" max="2040-01-01">:** define um campo com um calendário, em que será definida a menor data e a maior data possível de ser colocada pelo usuário;

```
<!-- dtnascimento-->
<div class="campo">
     <label               for="dtnasc"><strong>Data                de
Nascimento</strong></label>
     <input  type="date"  name="dtnasc"  id="dtnasc"  min="1920-01-01"
max="2040-01-01">
```

Fonte: Elaborado pelo autor (2024)

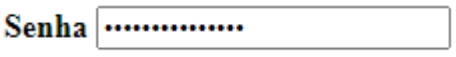
A partir do código descrito anteriormente, o resultado que aparecerá na página será um calendário, como podemos ver na imagem abaixo.



Fonte: Elaborado pelo autor (2024)

- **<input type="password">**: define um campo que receberá informações de senha, não permitindo a visualização dela;

```
<!-- Campo senha-->
 <div class="campo">
     <label for="senha"><strong>Senha</strong></label>
     <input type="password" name="senha" id="senha" required>
</div>
```



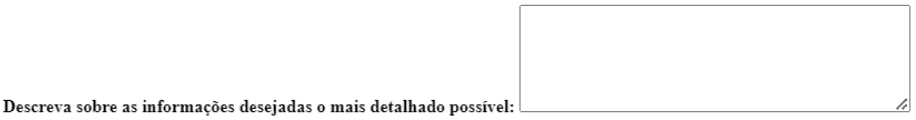
Fonte: Elaborado pelo autor (2024)

- **<textarea>**: essa *tag* tem como principal característica ser uma área de preenchimento de texto. Ou seja, permite que o usuário escreva um texto ou uma mensagem no seu interior. Ela também traz opções para que o usuário redimensione seu tamanho (**resize**);

```
<!-- Caixa de texto -->
   <div class="campo">
      <label for="mensagem"><strong>Descreva sobre as informações desejadas o mais detalhadamente possível: </strong></label>
      <textarea rows="6" style="width: 26em" id="experiencia" name="experiencia"></textarea>
</div>
```

Fonte: Elaborado pelo autor (2024)

A partir deste código, a página desenvolvida é a seguinte:



Fonte: Elaborado pelo autor (2024)

- *tag* **<button>**: define o botão para o envio das informações. Porém, ele deverá ser configurado para fazer uma ligação com um banco de dados, que receberá as informações.

```
<!-- Botão para enviar o formulário -->
<button class="botao" type="submit" onsubmit="">Enviar</button>
```

Fonte: Elaborado pelo autor (2024)

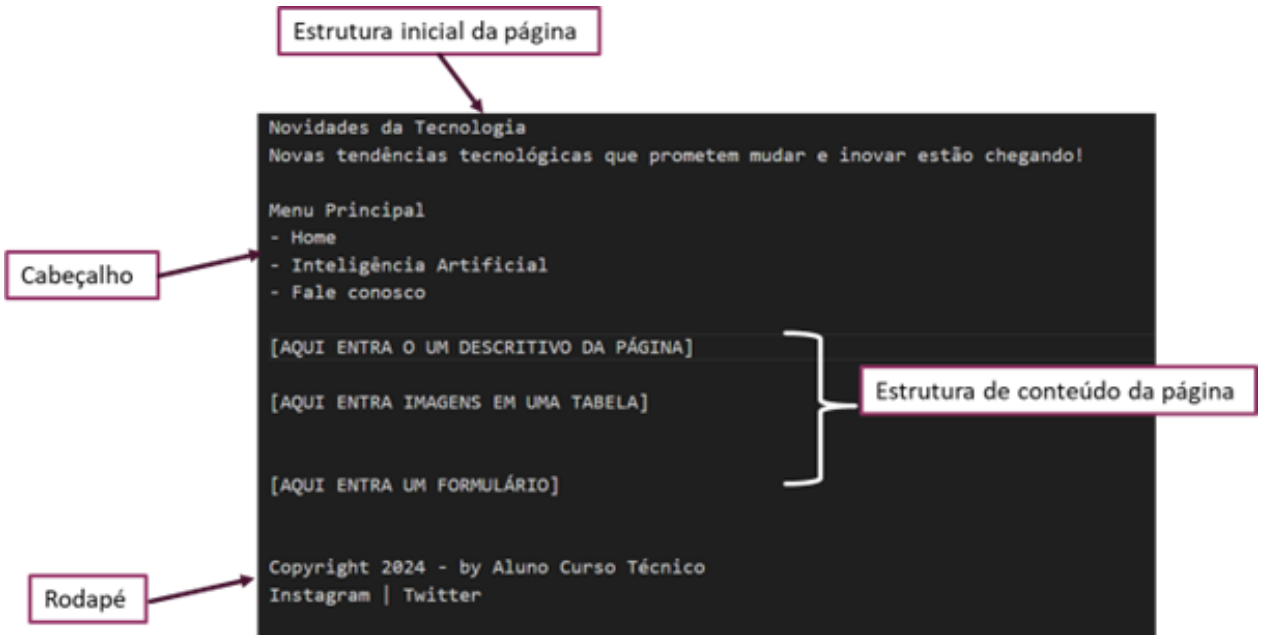
3.5 Criando uma página do zero

Esse é o momento em que vamos colocar todo o nosso aprendizado em prática. Para isso, construiremos uma página do zero usando as *tags* necessárias para cada ponto da página.

A página que criaremos terá a seguinte estruturação:

- estrutura inicial da página;
- **cabeçalho** – para isso utilizaremos a *tag* **<header>**, que tem como finalidade definir um cabeçalho. Portanto, todo conteúdo que estiver dentro dela faz parte de um cabeçalho, podendo ser utilizado dentro de outras sessões. Não confunda com a *tag* **<head>**;
- **estrutura de conteúdo da página** – nesse caso utilizaremos de algumas *tags* e alguns atributos importantes, além de recursos como a inserção de imagens, o uso de tabelas, o formulário, entre outros;
- **rodapé** – para isso, utilizaremos a *tag* **<footer>**, que define de forma mais organizada o rodapé da página. Geralmente, essa *tag* é utilizada no final da página.

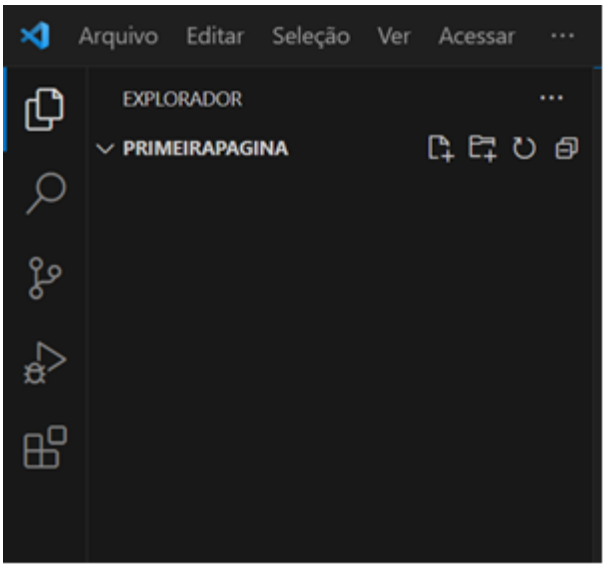
O primeiro ponto a se definir deve ser o tema que será trabalhado. Nesse caso, abordaremos, de forma rápida, algumas informações sobre **Tecnologia**. Então, vamos lá!



Fonte: Elaborado pelo autor (2024)

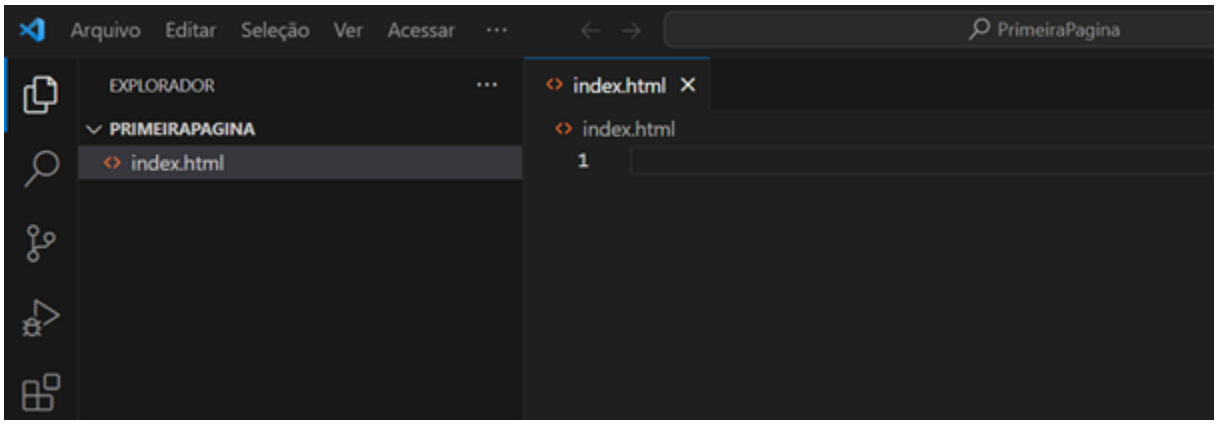
Estrutura inicial da página

Para iniciar o projeto, abra o Visual Studio e, em seguida, abra uma nova janela. Nela, você terá a opção de abrir uma pasta para este projeto. Ao abri-la, dê a ela o nome de **PrimeiraPagina.**



Fonte: Elaborado pelo autor (2024)

Na sequência, crie um novo arquivo chamado **index.html.** Esse será nosso arquivo principal.



Fonte: Elaborado pelo autor (2024)

Agora, nós vamos criar a estrutura inicial da página. Faça as devidas alterações da linguagem e altere o título da página para **inovação**. Utilize os recursos (**!** e **enter**) para colocar a estrutura inicial.

Fonte: Elaborado pelo autor (2024)

Construindo o cabeçalho

Para o cabeçalho, vamos trabalhar com algumas *tags* importantes, como:

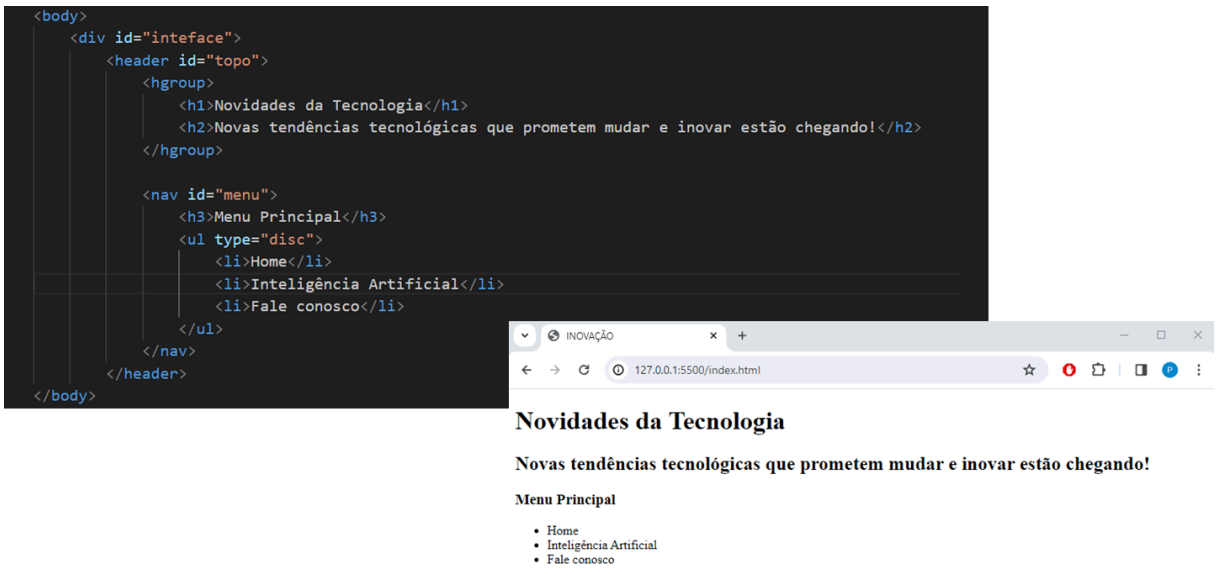
- *tag* **<header>**, que tem como finalidade delimitar a área do cabeçalho;
- *tag* **<hgroup>**, que destina-se a agrupar cabeçalhos de diferentes níveis para uma seção do documento. No caso, quando usamos as *tags* **<h1>** a **<h6>;**
- *tag* **<nav>**, que define o conteúdo de navegação. Ou seja, normalmente, o menu da página;
- *tags* de listas, que definem os menus em uma ordem;
- atributo **id**, que é importante para definições de elementos e para a estilização desses pelo CSS.

Importante: como ainda não estamos trabalhando com CSS, a nossa página terá um visual muito básico. Porém, cada um desses elementos será importante para agregar ao CSS posteriormente.

```
<body>
    <div id="interface">
    <header id="topo">
```

```
            <hgroup>
                  <h1>Novidades da Tecnologia</h1>
                  <h2>Novas tendências tecnológicas que prometem
mudar e inovar estão chegando!</h2>
            </hgroup>

            <nav id="menu">
                  <h3>Menu Principal</h3>
                  <ul type="disc">
                        <li>Home</li>
                        <li>Inteligência Artificial</li>
                        <li>Fale conosco</li>
                  </ul>
            </nav>
      </header>
</body>
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

Definindo o conteúdo da página

Para iniciarmos o conteúdo da página, nós vamos acrescentar primeiro o texto descritivo da página, conforme definido na sua estrutura geral. Abaixo, segue o texto que deverá ser utilizado:

> *"As tendências e inovações tecnológicas estão sempre bombando, trazendo um show de novidades que fazem nossos olhos brilharem! Imagina só: inteligência artificial deixando tudo mais esperto, como aqueles assistentes que respondem até antes de a gente perguntar. A realidade aumentada é outra estrela do momento, transformando nosso mundinho com informações extras superlegais. E não podemos esquecer da revolução 5G, que vai deixar a internet mais rápida do que um flash! Essas novas tecnologias estão dando um upgrade na nossa vida, trazendo um futuro cheio de surpresas e facilitações. É só ficar de olho e se preparar para essa festa high-tech! "*

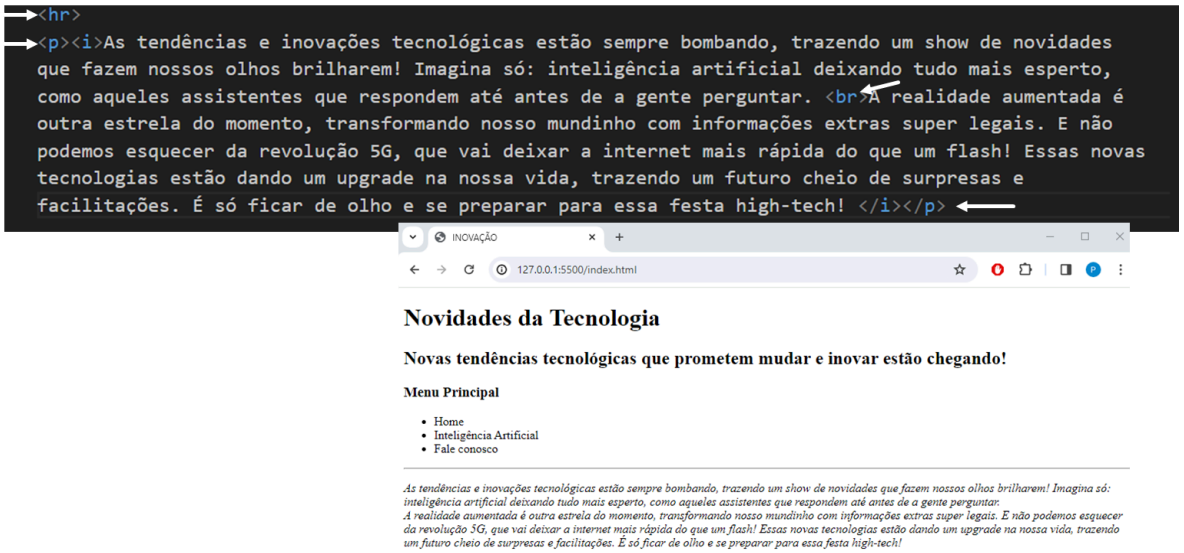
Para trabalhar esse texto, utilizaremos:

- a *tag* **<p>** para definir parágrafos;
- a *tag* **<hr>** para inserir uma linha separando o texto de outros elementos;
- a *tag* **<i>** para formatar o texto em itálico.

No Visual Studio, faça de acordo com a imagem abaixo:

```
<hr>
<p><i>As tendências e inovações tecnológicas estão sempre bombando, trazendo um show de novidades que fazem nossos olhos brilharem! Imagina só: inteligência artificial deixando tudo mais esperto, como aqueles assistentes que respondem até antes de a gente perguntar. <br>A realidade aumentada é outra estrela do momento, transformando nosso mundinho com informações extras superlegais. E não podemos esquecer da revolução 5G, que vai deixar a internet mais rápida do que um flash! Essas novas tecnologias estão dando um upgrade na nossa vida, trazendo um futuro cheio de surpresas e facilitações. É só ficar de olho e se preparar para essa festa high-tech! </i></p>
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

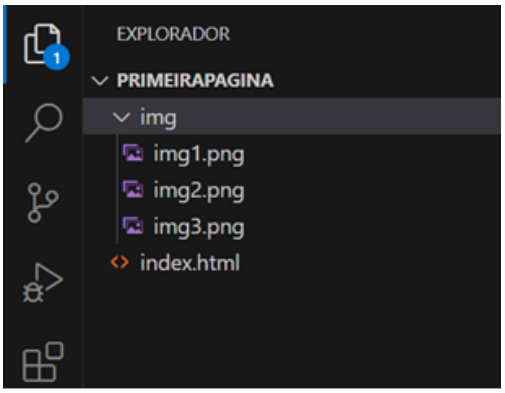
O próximo passo será inserir imagens na página, que deverão ficar organizadas dentro de uma tabela. Você lembra dessas *tags* e como fazê-las? Vamos relembrar!

- *Tag* **<img>**: utilizada para inserir imagem, junto a ela há o atributo *src*;
- *Tag* **<table>**: utilizada para inserir uma tabela e as *tags* **<tr>** e **<td>**, que significam, respectivamente, *table row*, que é usada para definir uma linha, e *table data*, que é usada para criar uma célula.

Importante: para fazer esse processo, será preciso fazer o *download* de três imagens relacionadas à tecnologia. Se você tiver dúvidas sobre *sites* com imagens públicas e como realizar esse processo, verifique com o(a) seu(ua) professor(a).

Acompanhando a imagem a seguir, insira as imagens já baixadas. Abaixo, você terá as linhas de código para construir a tabela e inserir as imagens.

Vale ressaltar que é preciso criar uma pasta dentro do projeto **PrimeiraPagina** chamada **IMG** para armazenar as imagens baixadas. Esse processo manterá uma organização no projeto e facilitará a inserção das imagens.

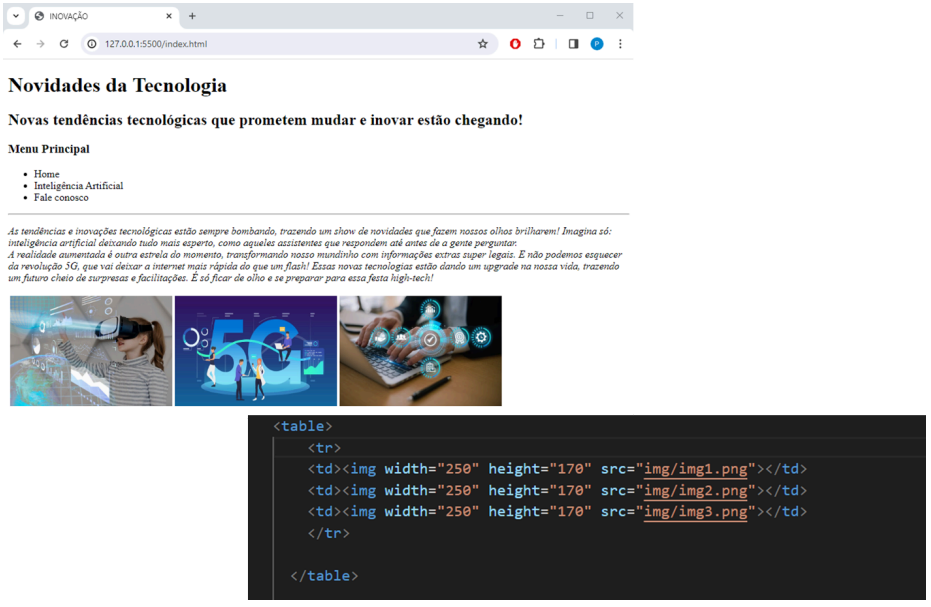


Fonte: Elaborado pelo autor (2024)

Feito isso, siga para a construção das linhas de códigos abaixo, depois do texto já inserido.

```
<table>
      <tr>
      <td><img width="250" height="170" src="img/img1.png"></td>
      <td><img width="250" height="170" src="img/img2.png"></td>
      <td><img width="250" height="170" src="img/img3.png"></td>
      </tr>

  </table>
```

Fonte: Elaborado pelo autor (2024)



Disponível em: <Katemangostar-http://tinyurl.com/27v64f2y> <freepik-http://tinyurl.com/4dnbvyz2> <freepik-http://tinyurl.com/3xat99dx>. Acesso em: 06 jan. 2024. Fonte: Elaborado pelo autor (2024)

Para finalizar o conteúdo da página, vamos criar um formulário simples, em que o usuário poderá deixar sugestões, elogios e outras informações.

Nesse caso, utilizaremos as *tags* para construção de formulário como **<form>**, **<fieldset>**, **<label>** e **<input>**.

Para isso, siga os procedimentos abaixo:

1. Crie a base do formulário:

```
<form>
    <fieldset class="grupo"><legend>Entre em contato</legend>



</fieldset>
</form>
```

Fonte: Elaborado pelo autor (2024)

A partir deste código, o seguinte resultado aparecerá na página:

Fonte: Elaborado pelo autor (2024)

2. Na sequência, será inserido um campo com opções de escolha sobre o assunto que o usuário deseja tratar. Nesse caso, nós vamos usar a *tag* **<input type="checkbox">**, permitindo, assim, a escolha de dados. Dessa forma, faça as linhas de comandos abaixo:

```
<!-- Campo para escolha do tipo de contato com opções para marcar (checkbox) -->
<div id="check">
    <label><strong>Selecione qual o tipo de informação deseja enviar:</strong></label><br><br>
    <input type="checkbox" id="assunto1" name="assunto1" value="Sugestoes">
    <label for="assunto1"> Sugestões</label>
    <input type="checkbox" id="assunto2" name="assunto2" value="elogio">
    <label for="assunto2"> Elogio</label>
    <input type="checkbox" id="assunto3" name="assunto3" value="outros">
    <label for="assunto3"> Outros</label>
</div>
```

Fonte: Elaborado pelo autor (2024)

Fonte: Elaborado pelo autor (2024)

Lembre-se de que essas informações deverão ficar dentro da t*ag* <fieldset>.

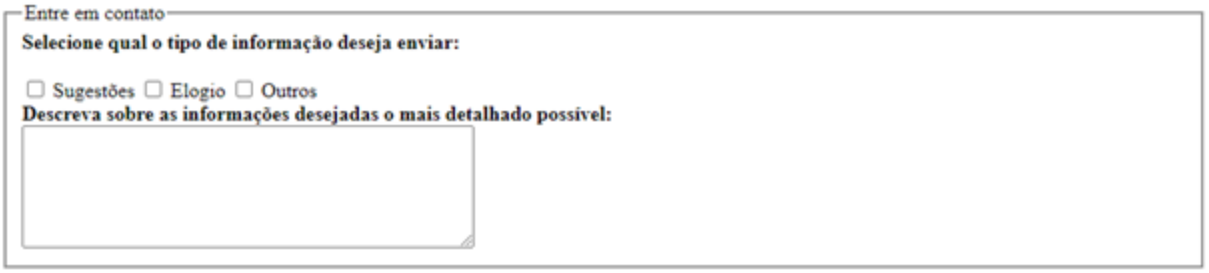
3. Para finalizar, insira uma caixa de texto, em que o usuário poderá digitar as suas informações. Nesse caso, utilize a *tag* **<textarea>**.

Siga o modelo a seguir:

```
<!-- Caixa de texto -->
<div class="campo">
    <label for="mensagem"><strong>Descreva sobre as informações desejadas o mais detalhado possível:
    </strong></label>
    <br>
    <textarea rows="6" style="width: 26em" id="experiencia" name="experiencia"></textarea>
</div>
```

Fonte: Elaborado pelo autor (2024)

Na imagem abaixo, é possível ver o resultado do código:



Fonte: Elaborado pelo autor (2024)

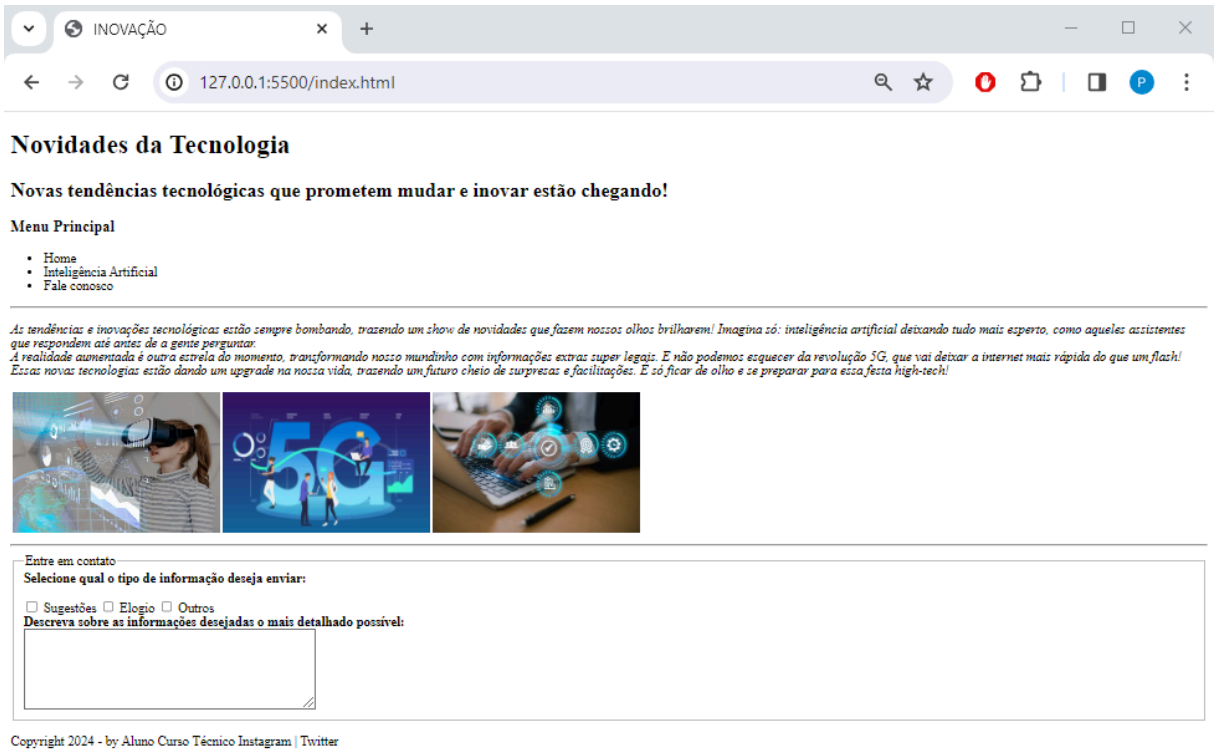
Construindo o rodapé

Para definir o rodapé, nós vamos trabalhar com a *tag* **<footer>**, que será colocada após a *tag* **</body>**, que define o seu fechamento. Sabendo disso, faça a construção das linhas de códigos abaixo no Visual Studio:

```
</body>
  <footer>
    <p>  Copyright 2024 - by Aluno Curso Técnico  Instagram | Twitter</p>

</footer>
```

Fonte: Elaborado pelo autor (2024)

Para uma melhor estilização desses elementos, nós podemos usar o CSS. Assim, teremos uma melhor formatação do rodapé. Veja como ficou nossa página na imagem a seguir:



Fonte: Elaborado pelo autor (2024)

Pronto! Finalizamos a construção da nossa primeira página utilizando somente HTML! Não esqueça de salvar o seu código em um repositório.

DESAFIO PRÁTICO

Tags HTML essenciais

Desafio: Trabalhando com formulário.

Descrição

Você recebeu a tarefa de criar um formulário para uma das páginas em desenvolvimento na empresa em que trabalha. Para realizar isso, será necessário aplicar os conhecimentos adquiridos ao longo das aulas e desenvolver uma página contendo exclusivamente um formulário de pesquisa.

Objetivos

- Demonstrar conhecimento na construção de formulário;
- Trabalhar com as *tags* para construção de formulário, principalmente os diferentes tipos da *tag* **<input>**;
- Salvar o projeto em um repositório como formulário de pesquisa.

Orientações

- Crie um formulário com o título "**FORMULÁRIO DE PESQUISA GERAL**" e inclua no formulário os seguintes campos em caixa de texto: nome, *e-mail* e endereço. Adicione também um campo para inserção da data de nascimento;
- Inclua a pergunta: "**Como você avalia seu conhecimento em HTML?".** Use uma *tag* para uma quebra de linha;
- Insira uma *tag* **<input>** do tipo *radio* com três opções: **Muito Bom, Bom** e **Preciso estudar mais**;
- Coloque a pergunta: "**Assinale a *tag* que melhor assimilou na construção de uma página**". Use uma *tag* para uma quebra de linha;
- Abaixo, adicione opções que permitam seleção múltipla: **Tabela, Formulário, Imagens, *Favicon*, *Links*;**
- Finalize incluindo uma caixa de texto para que o usuário possa deixar a sua opinião geral;
- Após completar o formulário, salve o projeto e direcione-o para um repositório.

DESAFIO PRÁTICO II

Criação de uma página HTML5 simples

Desafio: Estruturando um projeto..

Descrição

Agora que possui conhecimento suficiente para trabalhar com o desenvolvimento *web* em HTML, você recebeu a responsabilidade de criar uma página simples. Essa página deve permitir que o usuário acesse outras páginas por meio de *links* incorporados nela.

Objetivos

- Demonstrar conhecimento na utilização de *links* e as suas aplicações;
- Criar a estrutura de uma página com recursos aprendidos;

Orientações

- Crie uma página com o título "**Acesso direto**", utilizando formatação para título;
- Em seguida, adicione um texto explicativo sobre a finalidade dessa página, informando ao usuário que se trata de uma página para acesso direto a vídeos de aprimoramento em HTML;
- Logo abaixo do texto, insira as informações dos links:
    - **YouTube:** "DESENVOLVENDO MEU PRIMEIRO SITE COM HTML". Acesso em: 06/01/24. - (https://youtu.be/wjDteOd996I). Esse *link* deve abrir em uma nova aba;
    - **YouTube:** "5 MINUTOS DE HTML PARA INICIAR EM PROGRAMAÇÃO!". Acesso em: 06/01/24. - (https://youtu.be/3oSIqIqzN3M). Esse *link* deve abrir na mesma página.

- **YouTube:** "HTML // Dicionário do Programador". Acesso em: 06/01/24. - (https://youtu.be/4dQtz1PpY9A). Esse *link* deve abrir em uma nova aba.

- Certifique-se de criar os *links* conforme as especificações, abrindo o primeiro item em uma nova aba, o segundo na mesma página e o terceiro em uma nova aba.

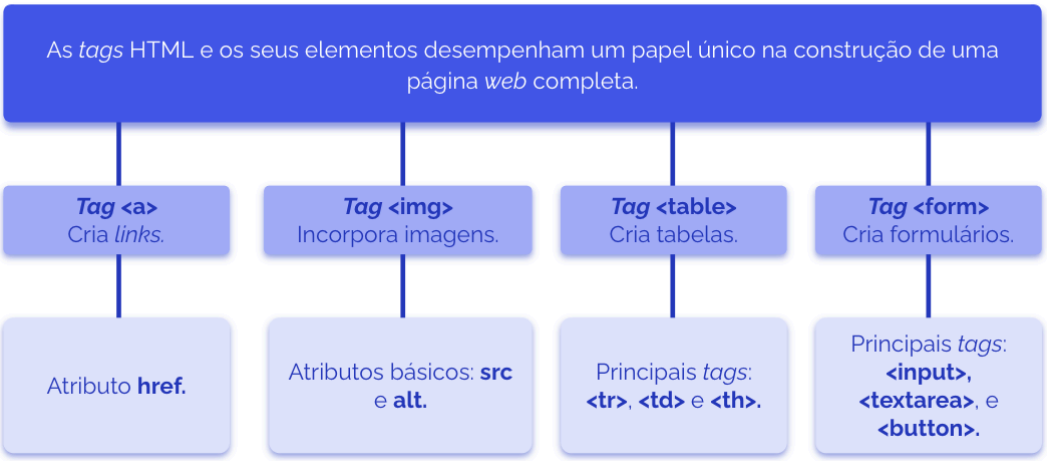
Estudamos que as *tags* HTML podem ser consideradas os elementos que dão vida à sua página na *web*. Para facilitar o seu entendimento sobre o assunto, podemos dizer que a *tag* **<a>** é como um GPS, conectando os seus visitantes a diferentes partes do ciberespaço. Com a *tag* **<img>,** é possível adicionar cor e personalidade à nossa página, fazendo com que o visual tenha mais propriedade digital, permitindo que o usuário visualize melhor a ideia do *site*.

Também aprendemos que os formulários, marcados pela *tag* **<form>**, são os convites interativos para que os visitantes de um *site* deixem seus registros *on-line*. Imagine criar um mural de mensagens ou uma lista de convidados: é isso que os formulários fazem por você.

Abordamos que a *tag* **<table>** é como um maestro da organização, colocando informações em fileiras e colunas. Imagine uma lista de convidados organizada ou uma tabela de horários fácil de entender: tudo isso é possível com essa *tag*.

Por fim, vimos que essas *tags* criam uma experiência única na *web*, transformando a sua página em um lugar acolhedor, bonito e super funcional. Abaixo, veja um mapa mental que vai te ajudar nos estudos.

HTML e as *tags* de *layout*



ATIVIDADE DE FIXAÇÃO

1. Para qual atributo da *tag* **<input>** você atribuiria um tipo de campo de senha?
   a. **type**
   b. **password**
   c. **secure**
   d. input-type

2. Qual atributo da *tag* **<img>** é usado para fornecer um texto alternativo para a imagem?
   a. **alt**
   b. **caption**
   c. **description**
   d. **text**

3. Como você define um *link* para abrir em uma nova guia ou janela?
   a. ****
   b. ****
   c. ****
   d. ****

4. O que a *tag* **<table>** representa em HTML?
    a. *Links.*
    b. Texto destacado.
    c. Tabelas de dados.
    d. Elementos de formulário.

5. Qual *tag* é utilizada para criar formulários em HTML?
    a. **<input>**
    b. **<form>**
    c. **<submit>**
    d. **<data>**

6. Qual é a finalidade da *tag* **<img>** em HTML?
    a. Criar *links*.
    b. Adicionar imagens.
    c. Definir estilos de texto.
    d. Inserir vídeos.

7. Qual *tag* é usada para criar *links* em HTML?
    a. **<link>**
    b. **<a>**
    c. **<url>**
    d. **<linkto>**

8. Você está criando uma página para um *blog* e deseja adicionar um *link* para uma outra postagem. Além disso, você quer incluir uma imagem que sirva como um botão de "Leia mais". Qual das alternativas a seguir apresenta a opção correta para criar o *link* e incluir a imagem?
    a. **<img src="leia-mais.jpg" alt="Leia mais">**
    b. **<link to="outra-postagem.html"><img link="leia-mais.jpg" alt="Leia mais"></img>**
    c. **<img src="leia-mais.jpg" alt="Leia mais">**
    d. **<link href="outra-postagem.html"><img src="leia-mais.jpg" alt="Leia mais"></img>**

9. Você está desenvolvendo um formulário de registro para um *site*. Este formulário precisa coletar o nome, o *e-mail* e a senha do usuário. Como você configuraria os campos de entrada no HTML?

   a. **<form>**
   Nome: **<input type="text" name="nome">
**
   E-mail: **<input type="text" name="email">
**
   Senha: **<input type="text" name="senha">
**
   </form>

   b. **<form>**
   Nome: **<input type="text" id="nome" name="nome">
**
   E-mail: **<input type="email" id="email" name="email">
**
   Senha: **<input type="password" id="senha" name="senha">
**
   </form>

   c. **<form>**
   Nome: **<input type="name" name="nome">
**
   E-mail: **<input type="email" name="email">
**
   Senha: **<input type="password" name="senha">
**
   </form>

   d. **<form>**
   Nome: **<input type="text" id="nome" name="nome">
**
   E-mail: **<input type="text" id="email" name="email">
**
   Senha: **<input type="text" id="senha" name="senha">
**
   </form>

10. Você está desenvolvendo o *site* para uma loja *on-line* e deseja criar uma seção promocional com um *banner* que, ao ser clicado, leve os usuários para uma página de descontos. Além disso, você quer adicionar um texto alternativo à imagem, para promover acessibilidade. Qual seria a forma correta de implementar isso em HTML?

    a. **<img src="banner-promocional.jpg" alt="Descontos Incríveis">**

    b. **<img src="banner-promocional.jpg" alt="Descontos Incríveis">**

    c. **<link to="pagina-descontos.html"><img src="banner-promocional.jpg" alt="Descontos Incríveis"></link>**

    d. **<img link="banner-promocional.jpg" alt="Descontos Incríveis"></img>**

CAPÍTULO 04

Introdução ao *Cascading Style Sheets* (CSS)

O que esperar deste capítulo:

- Entender sobre o CSS e a sua importância;
- Compreender as diferenças entre as versões as anteriores e o CSS3;
- Aprender sobre os seletores e as suas aplicações em HTML.

4.1 Entendendo sobre o CSS

Até agora, nós estamos trabalhando com desenvolvimento *web.* Porém, só estudamos o HTML, sendo essa uma **linguagem de marcação de hipertexto,** assim como foi visto anteriormente. No entanto, em alguns momentos, falamos sobre a possível utilização de CSS nas nossas páginas. Mas, afinal... O que é o CSS e quais são as diferenças e as mudanças que ele pode causar no desenvolvimento *web* como um todo?

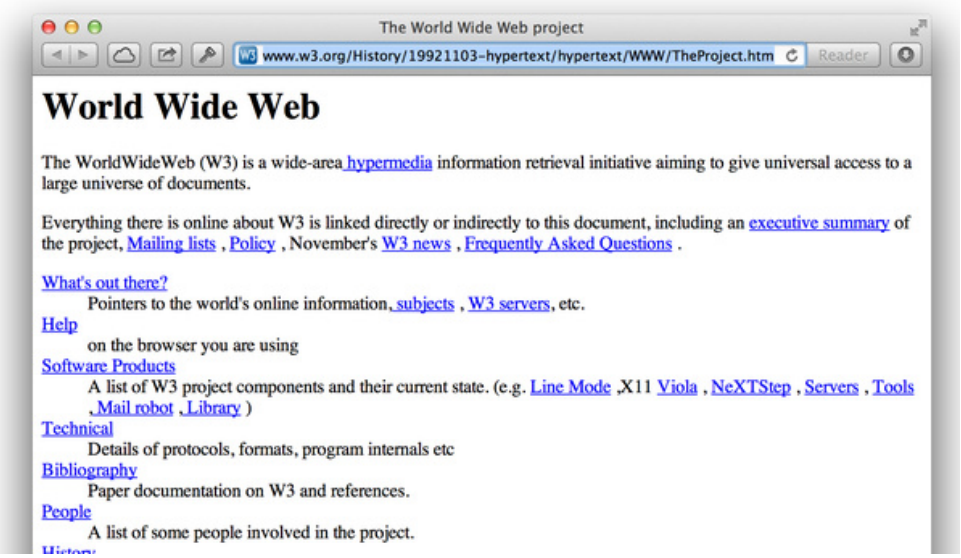
Mais conhecido como CSS, o ***Cascading Style Sheets*** ou, traduzindo para o português, a **Folha de Estilo em Cascatas**, é uma **linguagem de estilos** que define o *layout* dos documentos HTML. Usando uma analogia, podemos dizer que o CSS é como o estilista da internet, dando um toque de *glamour* às páginas *web*. Para entender melhor a diferença entre o HTML e o CSS, imagine que o HTML seja a estrutura de uma casa e que o CSS é a pintura, os móveis e a decoração que transformam essa casa em um lar acolhedor. Com ele, nós podemos mudar as cores, as fontes, os tamanhos e os *layouts*, criando um visual único e atraente.

Sendo assim, com o CSS, é possível estilizar todos os elementos da página, desde os cabeçalhos até os botões. Além disso, o "*cascading*" em seu nome remete ao efeito cascata, ou seja, as regras estipuladas podem se acumular, permitindo um controle refinado sobre o *design* da página. Então, se você quer deixar seu *site* com a sua cara, o CSS é o aliado perfeito para dar um toque de estilo e personalidade aos seus desenvolvimentos.

Vale lembrar que o CSS, assim como o HTML, **não é uma linguagem de programação**, mas uma **linguagem de folhas de estilos** que é utilizada para aprimorar os documentos escritos em uma linguagem de marcação.

4.2 Conhecendo a história do CSS e do HTML

Antigamente, no início da internet, trabalhava-se apenas com o HTML, que é uma forma muito simples de se apresentar as páginas *web*.



Disponível em: https://gizmodo.uol.com.br/wp-content/blogs.dir/8/files/2012/08/web-tbl.jpg. Acesso em: 18 jan. 2024.

Com o passar do tempo, novas necessidades de alternativas de formatação foram surgindo e, com isso, novas *tags* e atributos foram criados no HTML para que ele pudesse atender às demandas de *layout*, passando a cumprir as necessidades de **estruturação** e de **apresentação**. No entanto, os desenvolvedores começaram a esbarrar em um problema: não havia como definir um padrão que se aplicasse a várias páginas. Isso fazia com que qualquer alteração tivesse que ser aplicada página a página, algo que se tornou inviável quando se tratava de *sites* grandes.

Dessa forma, o CSS foi criado como solução para esses problemas e passou a ser utilizado especificamente para definir a formatação e ajustar as características de um documento com cores, fontes, alinhamentos, entre outros aspectos de estilos. Além disso, ele permitiu a possibilidade de compartilhamento de formatos. Ou seja, tornou-se possível que uma mesma definição de *layout* fosse aplicada a várias páginas, aumentando, assim, a flexibilidade e diminuindo a repetição e a extensão dos códigos em HTML.

Disponível em: https://www.shutterstock.com/pt/image-vector/vector-set-flat-website-templates-338747663. Acesso em: 18 jan. 2024.

Nos anos 1970, bem antes da criação do CSS, alguns desenvolvedores já trabalhavam com linguagens similares, com o propósito de formatar e estruturar documentos. No entanto, isso só ganhou mais força depois do lançamento do HTML.

Em 1995, os desenvolvedores Håkon Wium Lie e Bert Bos apresentaram uma proposta de linguagem para o **W3C** (*Word Wide Web Consortium*). Já no ano seguinte, em 1996, o CSS foi desenvolvido, visando complementar a linguagem HTML. Com a tarefa de separar o conteúdo do *site* da sua apresentação visual, essa linguagem trouxe como proposta a finalidade de alterações de elementos como cores de texto, fonte, espaçamento em blocos, entre outros aspectos estéticos das páginas.

O CSS3 é a sua versão mais recente, voltada para o conceito de *Web* 2.0, agregando recursos de transições, imagens e efeitos para criação de animações.

Disponível em: https://www.shutterstock.com/pt/image-vector/man-programmer-gui-designer-character-working-1932923438. Acesso em: 18 jan. 2024.

Função do CSS

A principal função do CSS é **separar a formatação do conteúdo do HTML**. Dessa forma, o navegador solicita as informações ao servidor, depois a estrutura do HTML é carregada e, automaticamente, o *link* da folha de estilo também é. Mas... o que isso quer dizer? Vamos entender melhor juntos!

Quando você cria uma página da *web* usando o **HTML**, você define a **estrutura** e o conteúdo da página. No entanto, para tornar essa página visualmente atraente, você precisa **estilizar a aparência** dela, utilizando cores, fontes e *layout*. É aqui entra o **CSS**.

Disponível em: https://www.shutterstock.com/pt/image-vector/html-css-javascript-layers-website-coding-2067900194. Acesso em: 18 jan. 2024.

Sendo assim, a principal finalidade do CSS consiste em separar a formatação visual do conteúdo presente no HTML. Ou seja, em vez de misturar as instruções de estilo diretamente no HTML, **você cria um arquivo CSS separado**, o que facilita a organização e a manutenção do código.

Quando alguém acessa a sua página da *web*, o navegador solicita as informações ao servidor em que o *site* está hospedado, que, por sua vez, envia a estrutura HTML da página para o navegador, mas também inclui um *link* para o arquivo CSS. Dessa forma, **o navegador carrega não apenas a estrutura da página, mas, também, as instruções de estilo contidas no arquivo CSS**, permitindo que a página seja exibida conforme você a projetou, com todas as formatações visuais aplicadas automaticamente. Isso proporciona uma experiência visual consistente e agradável aos visitantes do seu *site*.

Para que serve o CSS?

Criação de animações	Criação de efeitos visuais	*Sites* dinâmicos	*Landings pages*
Proporciona uma experiência mais atrativa em comparação às páginas de antigamente, também conhecidas como estáticas. No entanto, requer mais atenção na execução de suas estruturas.	Efeitos sobrepostos permitem *layouts* diferenciados, com imagens que possuem profundidade, tornando a experiência visual mais agradável. Exemplo: efeito *parallax*.	Possibilidades de páginas responsivas, transições, contas personalizadas e textos ou caixas que mudam de cor conforme a navegação.	Páginas para captação de *leads* ou dados de visitantes que contam com componentes gráficos em destaque e formulários.

Além dessas aplicações, o CSS permite diversos benefícios no desenvolvimento *web*. Vamos conhecer alguns dos principais?

- Possibilidade do controle do *layout* de vários documentos a partir de um único arquivo CSS, gerando economia de tempo em desenvolvimento;
- Aplicação de *layouts* responsivos de acordo com o dispositivo utilizado. Ou seja, as páginas terão a visualização correta de acordo com o tamanho da tela e do dispositivo que o usuário estiver utilizando para acessar a página;
- Possibilidade de manter a mesma formatação em diferentes navegadores;
- Aplicação de técnicas mais sofisticadas de desenvolvimento;
- Menor consumo de banda e melhor desempenho devido ao reuso do mesmo código de formatação em várias páginas, diminuindo o tamanho do código.

4.3 Diferenças entre versões CSS

De modo geral, pode-se dizer que o que diferencia uma versão CSS da outra são os **pontos de melhoria nos recursos**, que têm como finalidade melhorar a experiência de *design* e a estilização na *web*. Vamos analisar as versões CSS existentes e conhecer os recursos oferecidos em cada uma delas?

4.4 Métodos de aplicação do CSS no documento HTML

Os métodos de aplicação, também chamados de regras de aplicação, referem-se às diferentes maneiras de incorporar estilos e formatações visuais a uma página *web* usando a linguagem de estilo em cascata (CSS). Existem três métodos principais para aplicar CSS em um documento HTML. Eles são:

Método *in-line*

Os estilos são **aplicados diretamente dentro das *tags* HTML**, usando o atributo **style**, como no exemplo ao lado. Este método é **útil para aplicar estilos específicos a elementos individuais**, mas pode tornar o código HTML mais complexo e menos fácil de manter.

HTML

```
<p style="color: blue; font-size: 16px;">Este é um parágrafo com estilo inline.</p>
```

Método interno

Os estilos são definidos dentro da *tag* **<style>** no cabeçalho **<head>** do documento HTML. Veja um exemplo na imagem ao lado. Este método **permite centralizar os estilos em um único local**, facilitando a manutenção, especialmente em páginas com vários elementos.

HTML

```
<head>
  <style>
    p {
      color: green;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <p>Este é um parágrafo com estilo interno.</p>
</body>
```

Método externo

Os estilos são definidos em um arquivo CSS separado e, em seguida, vinculados ao documento HTML usando a *tag* **<link>**. O arquivo "**styles.css**" conteria as definições de estilo, como no exemplo ao lado. Este método é recomendado para projetos maiores, pois facilita a reutilização de estilos em várias páginas e mantém uma clara separação entre a estrutura HTML e os estilos CSS.

HTML

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

CSS

```
p {
  color: red;
  font-size: 20px;
}
```

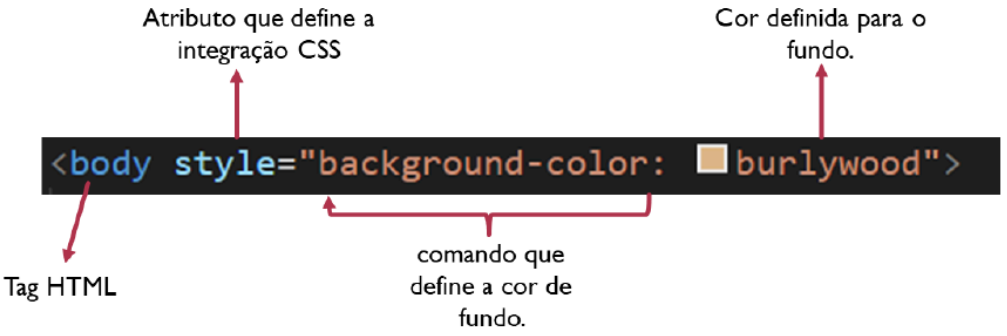
Vamos conhecer melhor cada método?

In-line

Esse método é trabalhado dentro das *tags* do HTML, conforme a necessidade de estilizar um determinado elemento.

Por exemplo: vamos considerar que é preciso colocar uma cor de fundo em uma página. Usando esse método, basta realizar a declaração dentro da *tag* **<body>** para dar cor à página como um todo.

Veja a sintaxe abaixo e repita o processo no seu Visual Code. Para isso, utilize a página criada anteriormente somente com HTML.



Fonte: Elaborado pelo autor (2024)

Fonte: Elaborado pelo autor (2024)

Interno

Esse método já foi muito usado pelos desenvolvedores, mas não é o mais eficiente. Com este método, é possível aplicar estilo a uma única página, declarando as regras normalmente dentro do cabeçalho, usando a *tag* **<style>**.

Veja a sintaxe abaixo e faça o processo no seu Visual Code. Para isso, utilize a página criada anteriormente somente com o HTML.

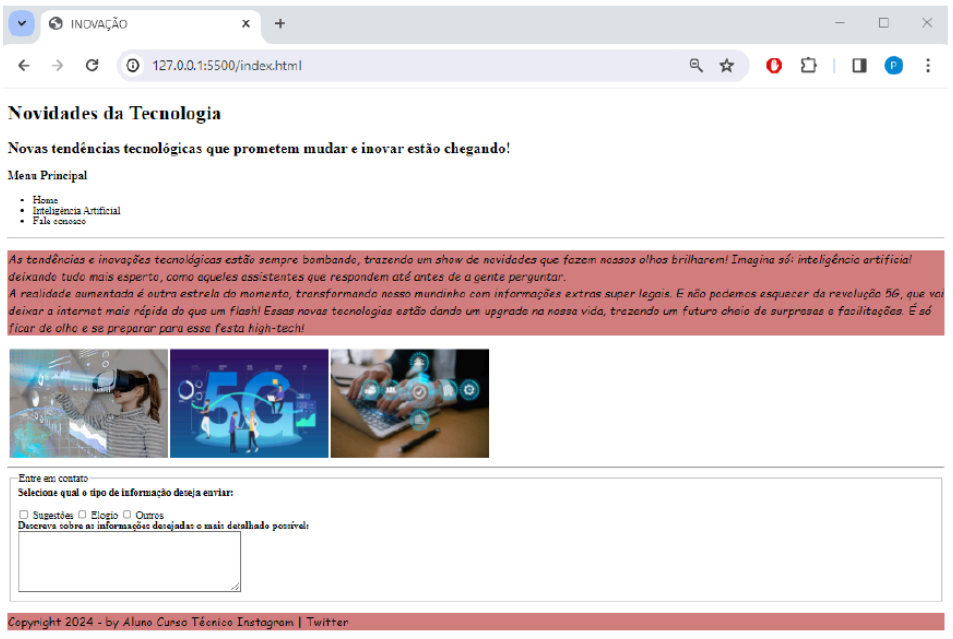
Tag <style></style>	Utilizada para definir a aplica elem
Atributo type	Define o tipo de arquivo a se **text**
P (referência à *tag* <p>)	Significa que será aplicada a for parág
background-color	Cor de
font-family	Referente ao tipo de f
font-size	Tamanho da fonte

Tag <style> fica dentro do cabeçalho

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>INOVAÇÃO</title>
    <STYLE TYPE="text/css">
        P {background-color: #d38080;
        font-family:'Comic Sans MS';
        font-size: 14pt}
        </STYLE>
</head>
```

Fonte: Elaborado pelo autor (2024)

Resultado



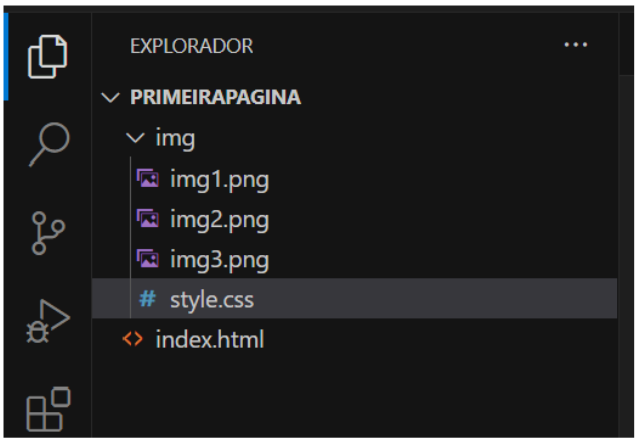
Fonte: Elaborado pelo autor (2024)

Externo

Esse é o método mais utilizado pelos desenvolvedores. Uma de suas vantagens é a reutilização de uma folha de estilo usada para várias páginas. Qualquer necessidade de alteração é feita somente no arquivo da folha de estilo, alterando todas as páginas ligadas a ela.

Para trabalhar com esse método, é necessário criar um arquivo na pasta do projeto. Esse arquivo de folha de estilo deve ser salvo com a extensão **.css**, sendo declarado nele toda a formatação das páginas em questão.

Por padrão, a maioria dos desenvolvedores nomeiam a folha de estilo como **style.css**. Porém, nada impede de ter mais de uma folha de estilo com nomes diferenciados, desde que se utilize a extensão padrão.

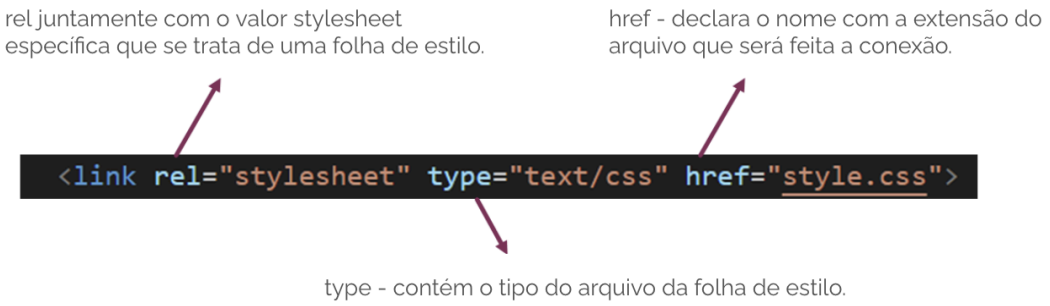


Fonte: Elaborado pelo autor (2024)

A conexão do arquivo **style.css** com a página HTML é feita por meio da *tag* **<link>**, sempre declarada dentro do cabeçalho. Veja as imagens abaixo e siga os exemplos no seu Visual Code.

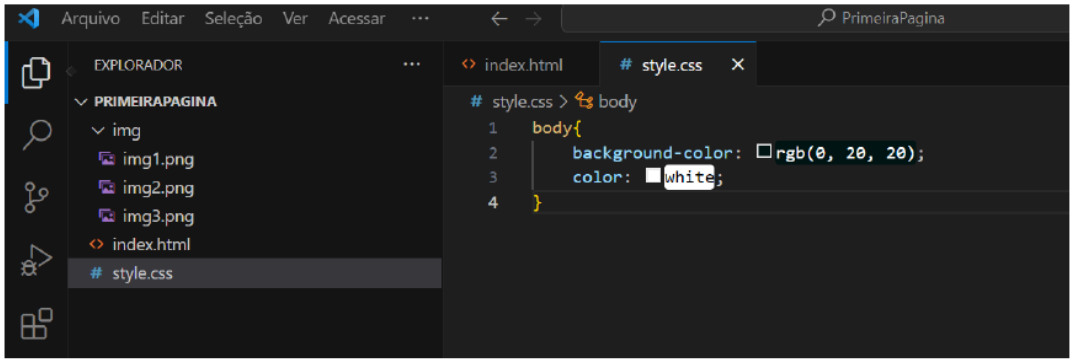
```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>INOVAÇÃO</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
```

Fonte: Elaborado pelo autor (2024)

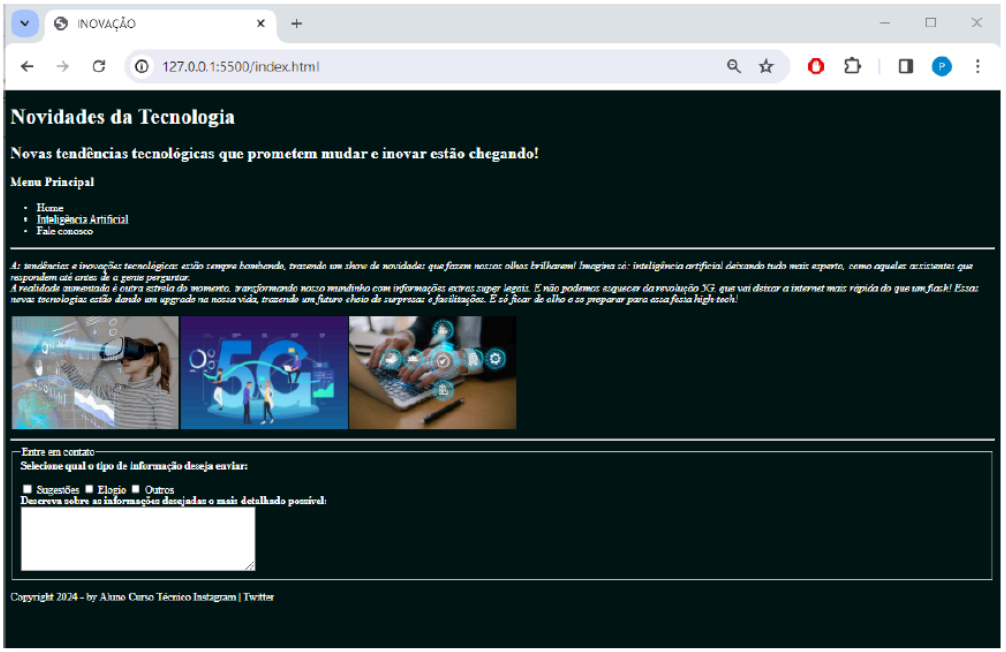


Fonte: Elaborado pelo autor (2024)

Já no arquivo **style.css**, faça as declarações de formatação que desejar. Veja o exemplo a seguir e realize o mesmo processo no seu Visual Code.



Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

Agora, a pergunta que fica é: qual é o método de aplicação mais indicado para utilizar junto aos documentos HTML?

Na maioria das vezes, essa é uma questão muito particular de cada desenvolvedor. Porém, por via de regra, o ideal é trabalhar com o método que dará menos trabalho no decorrer do desenvolvimento e nas atualizações a serem feitas ou no aproveitamento de estilização. Nós devemos sempre ter em mente que o tempo em desenvolvimento é muito precioso.

4.5 Seletores CSS e as suas aplicações

Agora que já entendemos os possíveis métodos para realizar a conexão entre o HTML e o CSS, podemos nos aprofundar em outras informações que farão grande diferença ao trabalhar com a linguagem CSS. Nós vamos falar sobre dois componentes importantes: o seletor e a declaração.

Seletor

Seletores CSS são como palavras-chave que **ajudam a dizer ao computador quais partes do *site* você quer estilizar**. Entre esses seletores, existem dois tipos especiais: as **pseudoclasses** e os **pseudoelementos**, que vamos conhecer mais adiante.

Com os seletores, é possível atribuir qualquer propriedade para os elementos, representados por *tags* e referenciados pelos seletores.

Por exemplo, ao formatar todas as tags **<p>** em CSS, você cria o seletor "**p**" e, dentro das chaves **{ }**, especifica quais propriedades serão aplicadas ao seletor.

```
p {
    color: blueviolet;
}
```

Fonte: Elaborado pelo autor (2024)

Nesse caso, o estilo definido para o seletor **p** foi a cor **blueviolet**.

Declaração

A declaração em CSS é como a ordem que você dá ao computador sobre como estilizar algo específico. Cada declaração tem duas partes: **uma propriedade e um valor**. A propriedade é o que você quer mudar (por exemplo, a cor do texto) e o valor é como você quer que essa propriedade seja (por exemplo, vermelho). Acompanhe no exemplo a seguir.

```
/* Declaração para mudar a cor do texto para azul */
p {
  color: blue;
}

/* Declaração para deixar o texto em negrito */
.destaque {
  font-weight: bold;
}
```

O seletor diz onde aplicar as mudanças e a declaração diz quais mudanças fazer. Juntos, eles permitem que você controle a aparência do seu *site*, tornando-o bonito e personalizado.

O componente seletor possui alguns tipos e nós vamos conhecê-los agora! Você poderá aplicar os seus conhecimentos, pois, após a apresentação de cada tipo, é importante que você **abra o Visual Code para seguir os exemplos que serão apresentados.** Dessa forma, vamos praticar tudo o que aprendemos passo a passo. Você está pronto? Então, vamos nessa!

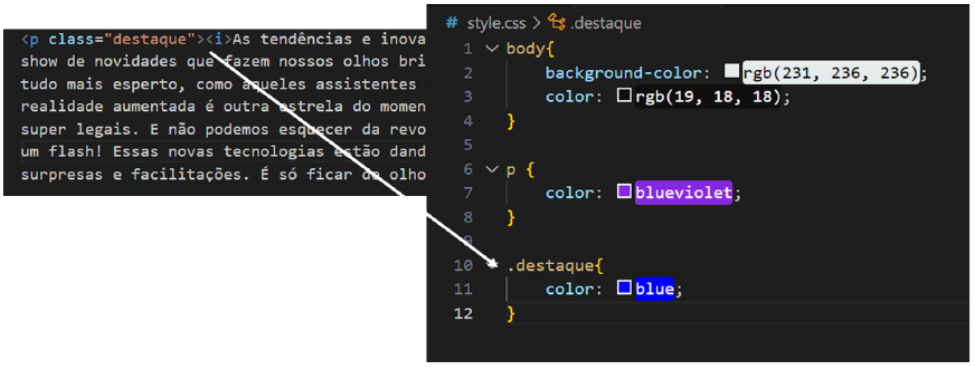
Seletor class

O seletor **class** permite identificar e estilizar um grupo de elementos HTML que compartilham a mesma classe. Para isso, deve-se criar uma classe junto a um determinado elemento ou determinadas *tags*. No arquivo CSS, ela terá sempre um ponto antes do nome da classe como seletor. Por exemplo: **.class**.

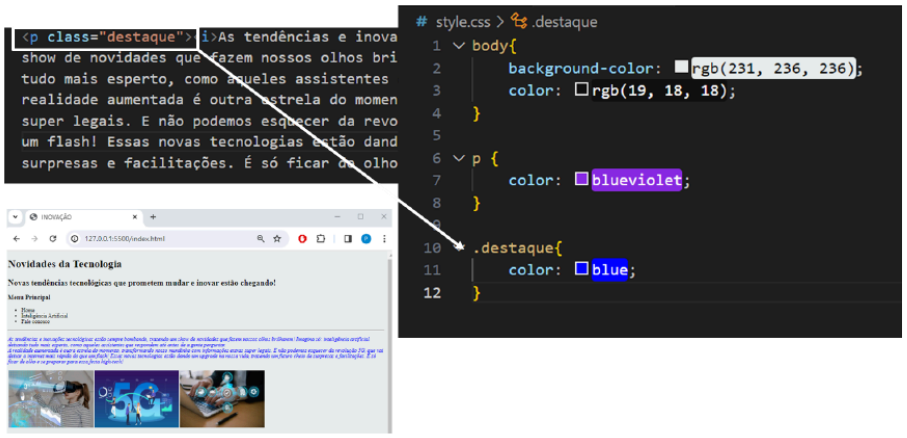
Agora, vamos para o momento de colocar a mão na massa! Veja os exemplos abaixo e, depois, aplique no Visual Code.

```
<p class="destaque"><i>As tendências e inovações tecnológicas estão sempre bombando, trazendo um
show de novidades que fazem nossos olhos brilharem! Imagina só: inteligência artificial deixando
tudo mais esperto, como aqueles assistentes que respondem até antes de a gente perguntar. <br>A
realidade aumentada é outra estrela do momento, transformando nosso mundinho com informações extras
super legais. E não podemos esquecer da revolução 5G, que vai deixar a internet mais rápida do que
um flash! Essas novas tecnologias estão dando um upgrade na nossa vida, trazendo um futuro cheio de
surpresas e facilitações. É só ficar de olho e se preparar para essa festa high-tech! </i></p>
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

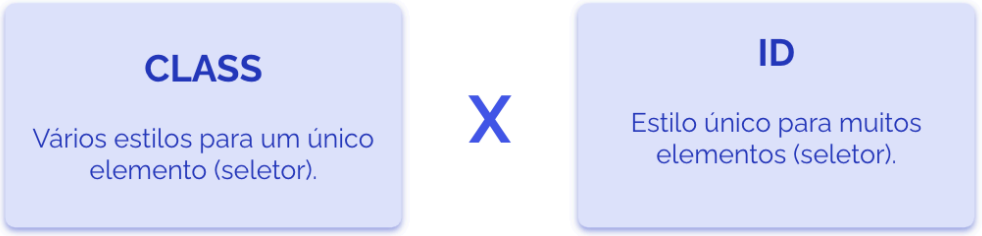
Seletor ID

Um seletor de **ID** é semelhante ao seletor de classe. Porém, ele identifica, exclusivamente, um elemento HTML na página. No CSS, você usa um *hash* (#) seguido do nome do **ID** como seletor. Por exemplo, se você quiser estilizar um elemento específico com o ID "**cabecalho**", você usaria **#cabecalho** como seletor.

Veja os exemplos abaixo e, depois, aplique no Visual Code.

Fonte: Elaborado pelo autor (2024)

Diferenças entre class e ID



Disponível em:<http://tinyurl.com/yrtss3v6>. Acesso em 06 jan. 2024.

Vale ressaltar que os dois seletores (**class** e **ID**) são trabalhados a todo momento com a *tag* ** **. Essa, por sua vez, define uma divisão ou uma seção em um documento HTML. Essa *tag* foi criada com a finalidade de permitir um mecanismo genérico para agrupar estruturas de documentos.

** : na prática**

```
<div id='demonstracao'>
  <h1>Devmedia</h1>
  <img src='https://www.devmedia.com.br/' alt='logomarca'/>
  <ul>
     <li>JavaScript</li>
     <li>Python</li>
     <li>PHP</li>
     <li>Java</li>
  </ul>
</div>
```

Disponível em:<http://tinyurl.com/j94943yb>. Acesso em 06 jan. 2024.

Agora, você se lembra que nós mencionamos que existem dois tipos especiais de seletores, que são as **pseudoclasses** e os **pseudoelementos**? Com eles, nós podemos trabalhar algumas palavras-chave que agregam maior valor aos seletores. Vamos conhecer melhor cada um deles a seguir.

Pseudoclasses

Uma pseudoclasse é como uma palavra-chave adicionada a um seletor. O seu uso é feito sempre com a aplicação do sinal de dois pontos (:) e especifica um estado especial ao elemento. Por exemplo, imagine que você queira dizer para o seu *site* "quando alguém passar o *mouse* sobre um *link*, mude a cor dele". Assim, quando o usuário passar o *mouse* sobre o menu da página, ele mudará de cor. Veja no exemplo:

```
nav#menu a:hover{
  color: #f0f0f0;
  text-decoration: underline;
}
```

Fonte: Elaborado pelo autor (2024)

Alguns tipos de pseudoclasses são:

- **:hover** – efeito usado para quando se passa o *mouse* em cima do elemento;
- **:active** – efeito para ativar o elemento;
- **:visited** – efeito que apresenta quando o *link* é visitado;
- **:focus** – efeito para quando um elemento recebe foco.

Pseudoelementos

Os pseudoelementos são como adicionar algo extra a uma parte específica de uma palavra ou de uma frase. Vamos dizer que você queira estilizar apenas a primeira letra de um parágrafo de uma maneira diferente. Você usaria um pseudoelemento chamado **::first-letter**. Assim, o computador entende que você está falando sobre a primeira letra e não sobre o parágrafo inteiro.

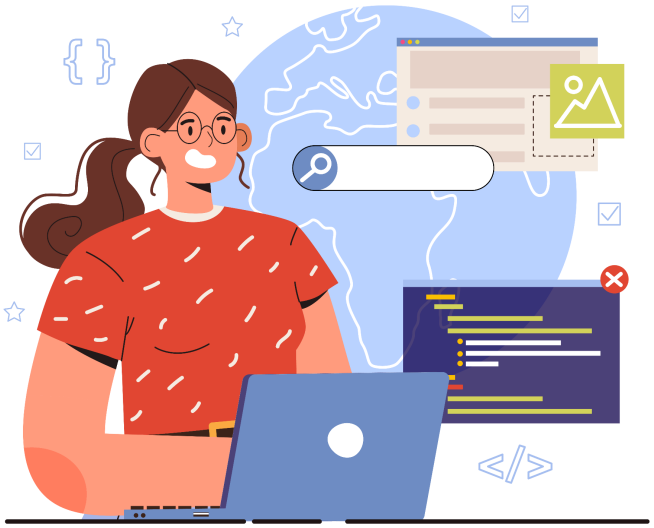
HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Exemplo Pseudo-Elemento</title>
<head>
<body>
  <p class="destaque">Lorem ipsum dolor sit amet, consectetur adipiscing elit.
</p>
</body>
</html>
```

CSS:

```
.destaque::first-letter {
  font-size: 1.5em; /* Tamanho da primeira letra */
  color: blue; /* Cor da primeira letra */
  font-weight: bold; /* Deixa a primeira letra em negrito */
}
```

Neste exemplo, a classe **.destaque** é aplicada ao parágrafo. O seletor **::first-letter** é usado para estilizar a primeira letra do conteúdo desse parágrafo de uma maneira diferente do restante do texto. No CSS, definimos que a primeira letra terá um tamanho maior, será azul e estará em negrito. Experimente ajustar esses valores para ver como eles afetam a aparência da primeira letra no Visual Code!



Disponível em: https://www.shutterstock.com/pt/image-vector/diverse-women-ai-stem-concept-female-2150679419. Acesso em: 18 jan. 2024.

RESUMO

Estudamos que o CSS (*Cascading Style Sheets*) é como a maquiagem do mundo digital, transformando as páginas HTML em verdadeiras obras de arte. Isso acontece porque essa linguagem define o estilo e a aparência de uma página *web*, permitindo que o desenvolvedor escolha as cores, as fontes, os *layouts* e outros elementos para dar personalidade ao seu *site*.

Também abordamos que, ao longo do tempo, o CSS evoluiu com diferentes versões, cada uma trazendo melhorias e novos recursos. Desde o CSS1 até o CSS3, as versões mais recentes abriram portas para animações, sombras e outros detalhes visuais incríveis.

Exploramos a importância de interligar o CSS com o HTML, que é como ter a combinação perfeita de música e dança, pois o HTML cria a estrutura da página, enquanto o CSS entra em cena para estilizar e torná-la visualmente atraente. Juntos, eles formam uma parceria imbatível para construir *websites* cativantes.

Vamos imaginar: se o HTML é a estrutura da casa, o CSS é a pintura, o papel de parede e os móveis que fazem desse espaço um lar encantador. É a interligação dessas linguagens que permite que seu *site* brilhe e deixe uma impressão duradoura nos visitantes.

ATIVIDADE DE FIXAÇÃO

1. Você está iniciando um curso *on-line* e se depara com a pergunta: "Como podemos definir CSS?". Assinale a alternativa que a responde corretamente.
    a. É uma linguagem de programação focada em funcionalidades lógicas.
    b. É um estilo de música muito popular entre desenvolvedores.
    c. É uma tecnologia que define a aparência visual de páginas HTML.
    d. É um formato de arquivo para armazenar dados de planilhas.

2. Você está participando de um grupo de estudo e surge a pergunta: "Qual versão do CSS introduziu recursos avançados como transições e animações?". Como você responderia?
    a. CSS1.
    b. CSS2.
    c. CSS3.
    d. CSS4.

3. Você está construindo seu próprio *site* e precisa estilizá-lo. Qual método você usaria para aplicar o CSS ao HTML?
    a. Incorporando o CSS diretamente nas *tags* HTML.
    b. Enviando o código CSS por *e-mail*.
    c. Utilizando um arquivo externo CSS vinculado ao HTML.
    d. Aplicando as tags de CSS em uma planilha de Excel.

4. Ao criar um menu de navegação, você quer estilizar todos os itens do menu de uma maneira única. Como você aplicaria uma classe CSS aos itens do menu no HTML?
    a. **<menu class="estilizado">**
    b. ** **
    c. **<ul class="menu">**
    d. **<li style="estilizado">**

5. Você está estilizando um cabeçalho especial na sua página e quer garantir que apenas um elemento tenha esse estilo único. Como você aplicaria um ID CSS ao elemento no HTML?
    a. ** **
    b. **<header id="especial" class="cabecalho">**
    c. **<h1 id="especial">Cabeçalho Especial</h1>**
    d. ** **

6. Ao criar uma lista de *links*, você deseja que eles mudem de cor quando o *mouse* passa sobre eles. Qual pseudoclasse CSS você usaria?
    a. **:hover**
    b. **:active**
    c. **:visited**
    d. **:link**

7. Você tem um formulário complexo e deseja estilizar apenas um campo específico de texto dentro dele. Como você faria isso usando **class** e **id**?
    a. **<input class="campo" id="texto">**
    b. **<input id="campo" class="texto">**
    c. **<input class="campo-texto" id="texto">**
    d. **<input id="campo-texto" class="texto">**

8. Explique o que é o CSS e como ele se relaciona com o HTML. Para isso, busque uma explicação abrangente sobre a definição do CSS e o seu papel na apresentação visual de páginas HTML. Apresente, de forma objetiva, como o CSS complementa o HTML, destacando a importância dessa combinação.

9. Descreva um exemplo prático de como o CSS pode ser aplicado para estilizar um elemento HTML específico. Aqui, o objetivo é que você forneça um exemplo concreto de como aplicar estilos a um elemento HTML usando CSS. Isso irá permitir avaliar a compreensão prática do uso do CSS em conjunto com o HTML.

10. Como as classes e IDs são usados no CSS em associação com o HTML para aplicar estilos específicos? Procure responder com uma explicação sobre o papel das classes e IDs na aplicação de estilos usando CSS. Como esses seletores são utilizados para direcionar estilos a elementos específicos em uma página HTML?

CAPÍTULO 05

Trabalhando com propriedades e estilização final (CSS)

O que esperar deste capítulo:

- Aplicar propriedades e estilos comuns do CSS;
- Compreender o *mode* de camadas;
- Desenvolver as folhas de estilo em geral.

5.1 Propriedades de estilos comuns

O CSS (*Cascading Style Sheet*) é utilizado para **estilizar elementos** escritos em uma linguagem de marcação como o HTML. Para entender como essas duas linguagens trabalham juntas, pense em uma página da *web* como uma casa. Nela, o HTML funciona como a estrutura básica, como as paredes e o teto. Já o CSS funcionaria como a pintura, os móveis e a decoração que tornam a casa bonita e agradável.

Mas, antes de explorarmos todas as práticas e possibilidades que o CSS pode oferecer, precisamos entender o que são as suas propriedades.

Em CSS, uma **propriedade** nada mais é do que uma **característica a ser estilizada.** Isso pode ser definir cores, alterar tamanhos ou ajustar as dimensões dos elementos. Dessa forma, podemos entender que existem diversos tipos de propriedades e cada uma poderá aceitar certos valores.

No mundo de desenvolvimento *web*, as propriedades de estilos são os ingredientes essenciais para dar vida e personalidade às páginas. Elas funcionam como comandos mágicos que **moldam cores, tamanhos, espaçamentos** e **muito mais**. De **color** a **font-size**, essas propriedades desempenham um papel vital, permitindo que os desenvolvedores **transformem códigos em *designs*** visuais cativantes. Conhecer e dominar essas propriedades é fundamental para criar experiências *on-line* envolventes e visualmente atraentes.

Para entendê-las melhor, vamos dividir as propriedades em três pontos importantes:

Tipos de medidas

Todas as propriedades podem, de alguma forma, ser trabalhadas de forma qualificadas e quantificadas. Trabalharemos de forma **qualificada** quando o valor da propriedade se referir a uma informação textual e de forma **quantificada** quando o seu valor se referir a algo mensurável (valores), como a espessura de uma borda ou o tamanho de uma fonte.

O CSS permite utilizar diferentes **unidades de medidas**, sendo extremamente importante conhecê-las para saber o momento adequado em utilizá-las. Essas unidades de medida se dividem em **absolutas** e **relativas**.

As **unidades absolutas** são aquelas que não dependem de nenhum outro valor de referência ou contexto da página. Ou seja, uma propriedade definida em unidades absolutas terá sempre o mesmo valor, independentemente das configurações ou do ambiente da página. Alguns exemplos são:

Unidade	Descriçã
in	Polegadas
cm	Centímetro
mm	Milímetro.

pt	Ponto (1 pt = 1/72 polegadas), nor impressão.
pc	Pica (1 pc = 12 pt), normalmente u
px	*Pixel*, unidade mais utilizada. Dete diretamente na tela. No entanto, cor utilizam diferentes resoluções de te pode variar de dispositivo

As **unidades relativas** são aquelas que utilizam um valor de referência definido anteriormente. Elas são mais flexíveis, pois se adaptam ao ambiente da página, sendo úteis para criar *layouts* responsivos, em que os elementos se ajustam proporcionalmente ao tamanho da tela ou a outros elementos na página. Alguns exemplos dessas unidades são:

Unidade	Descrição	U
em	Tamanho da fonte no elemento utilizado como base de referência.	Utilizand que serã uma letr seja, dimensã
rem	Tamanho da fonte na raiz do HTML como base de referência.	Utilizand que serã uma letr
%	Fração da dimensão do elemento pai.	50% é element
vw	1% do tamanho horizontal do navegador.	50vw
vh	1% do tamanho vertical do navegador.	33vh
vmin	1% da menor dimensão do navegador (horizontal ou vertical).	50vm dimens
vmax	1% da maior dimensão do navegador (horizontal ou vertical).	33vm dimens

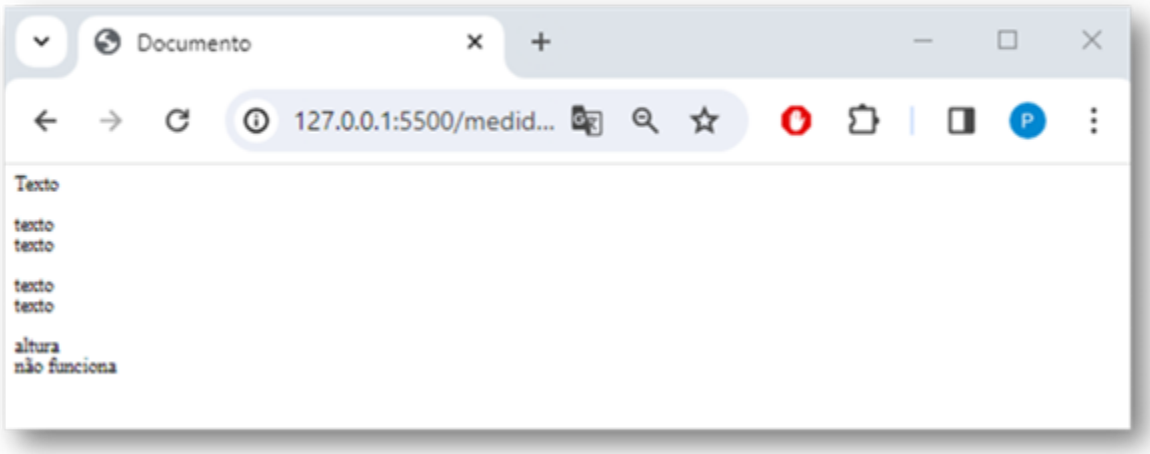
Considerando todas essas informações, vamos entender algumas dessas unidades na prática. Para isso, abra o seu **Visual Code** e crie dois arquivos chamados medidas, sendo um **medida.html** e outro **medida.css.** Depois disso, execute os exemplos abaixo.

Arquivo medida.html

```
    <link rel="stylesheet" type="text/css" href="medida.css">
</head>
<body>
    Texto<br><br> <!--primeira linha na visualização-->
    <div class="m1">
      texto<br> <!--segundo grupo de texto na visualização-->
      <div class="m1">
      texto<br><!--segundo grupo de texto na visualização-->
      </div>
    </div>  <br>

    <div class="m2">
      texto<br><!--terceiro grupo de texto na visualização-->
      <div class="m2">
      texto<br><!--terceiro grupo de texto na visualização-->
      </div>
    </div>   <br>
    <div class="m3">
    height <!--texto referente a altura-->
      <div class="m3">
       não funciona
      </div>
    </div>
</body>
</html>
```

Fonte: Elaborado pelo autor (2024)

Visualização da página

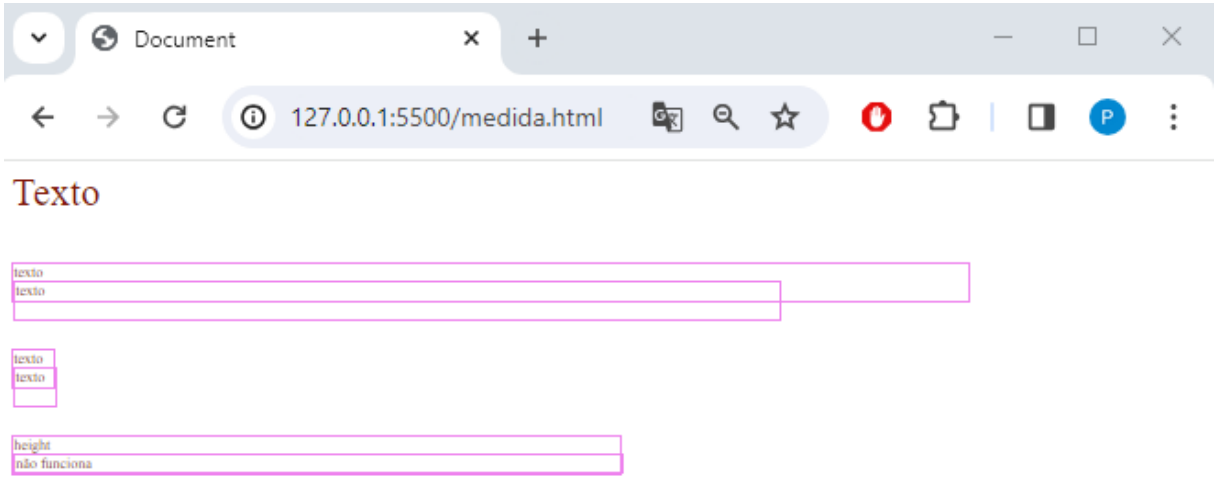


Fonte: Elaborado pelo autor (2024)

Aplicação de unidades de medidas com CSS

```
# medida.css > .m3
1   div {
2       border: 1px solid violet;
3       font-size: 12px;
4   }
5   html {
6       font-size: 32px;
7       color: rgb(126, 21, 7);
8   }
9   .m1 {
10      width: 80%;
11      height: 30px;
12  }
13  .m2 {
14      width: 1rem;
15      height: 30px;
16  }
17  .m3 {
18      width: 50vw;
19      height: 30%;
20  }
21
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

Alinhamento de parágrafos

Para definir o alinhamento de parágrafos, nós podemos usar as propriedades a seguir:

Text-align	Text-indent	Line-height
Alinhamento do texto: **left** (à esquerda), **right** (à direita), **center** (centralizado) e **justify** (justificado).	Recuo de primeira linha.	Espaçamento entre linhas.

Fonte: Elaborado pelo autor (2024)

Veja o exemplo abaixo e, depois, execute passo a passo no Visual Code.

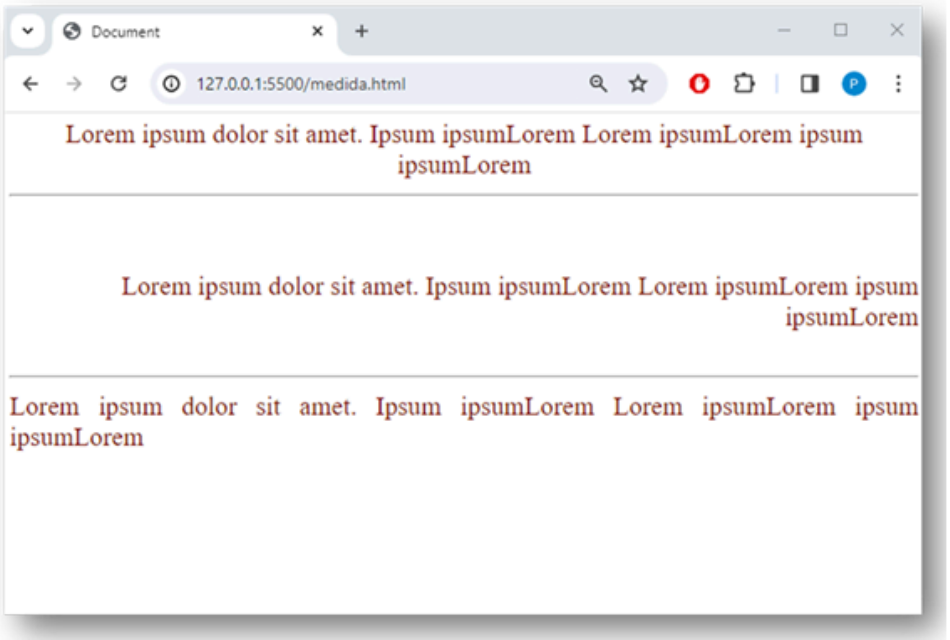
```
     <link rel="stylesheet" type="text/css" href="medida.css">
    </head>
    <body>
          <div  class="m">  Lorem  ipsum  dolor  sit  amet.  Ipsum
ipsumLorem Lorem ipsumLorem ipsum ipsumLorem <hr><br><br>
     <div class="m1">
          Lorem  ipsum  dolor  sit  amet.  Ipsum  ipsumLorem  Lorem
ipsumLorem ipsum ipsumLorem <br><br> <hr>
          <div class="m2">
          Lorem  ipsum  dolor  sit  amet.  Ipsum  ipsumLorem  Lorem
ipsumLorem ipsum ipsumLorem <br>
               </div>
     </div> </div>  <br>
       </body>
    </html>
```

Fonte: Elaborado pelo autor (2024)

```css
.m {
    font-size: 32px;
    color: rgb(126, 21, 7);
    text-align: center;
}
.m1 {
    text-align: right;
}
.m2 {
  text-align: justify;
}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



Fonte: Elaborado pelo autor (2024)

Definição de cores e fundos

Abaixo, você pode ver uma tabela que nos dá uma ideia de como aplicar propriedades que:

- alteram as cores do texto;
- alteram a cor de fundo da página;
- inserem imagens de fundo.

Propriedade	Descrição
color	Cor do texto.
background-color	Cor de fundo.
background-image	Está relacionada à imagem de fundo. Nesse caso, é preciso especificar o local em que o arquivo está armazenado. Por exemplo: **url(img/img4.png).**
background-repeat	Define que a imagem de fundo se repete, podendo ser: - **repeat -** repetir a imagem tanto para horizontal quanto vertical; - **no-repeat -** exibir a imagem apenas uma vez; - **repeat-x -** repetir a imagem somente na horizontal; - **repeat-y -** repetir a imagem somente na vertical.
background-attachment	Define o comportamento da imagem referente à rolagem da tela, podendo ser: - **fixed** - quando a imagem não acompanha a rolagem da tela; **scroll** - quando a imagem se movimenta de acordo com a rolagem da tela.
background-position	Refere-se ao posicionamento da imagem. Ou seja, ela identifica o posicionamento vertical (**top, bottom** *e* **center**) e o horizontal (**left**, **right** e **center**).

Com relação às cores, é preciso entender que elas podem ser especificadas de formas diferenciadas, assim como está destacado na tabela abaixo:

	RGB hexadecimal	RGB decimal	RGB percentual	Nome da cor em inglês
Definição	Ele usa uma combinação de seis caracteres alfanuméricos (0-9, A-F) precedidos por "#" para representar a intensidade de vermelho/*red* (RR), verde/*green* (GG) e azul/*blue* (BB).	Neste método, você especifica a cor usando os valores decimais da intensidade de vermelho, verde e azul.	É similar ao método decimal, mas, aqui, os valores são representados em porcentagem.	Essa abordagem é mais simples. Ela especifica a cor através do seu nome em inglês. No entanto, nem todas as cores têm nomes associados.
Exemplo	Por exemplo, **"#FF00CC"** representa uma cor com intensidade total de vermelho (#FF), nenhum verde (#00), e metade da intensidade de azul (#CC).	Por exemplo, **"RGB [255, 0, 100]"** indica intensidade total de vermelho (255), nenhum verde (0), e uma intensidade média de azul (100).	**"RGB [100%, 0%, 50%]"** representa intensidade total de vermelho (100%), nenhum verde (0%), e metade da intensidade de azul (50%).	Ao escrever "*red*", por exemplo, você terá a cor vermelha "pura".

Utilizando o Visual Code, execute os exemplos abaixo, aplicando:

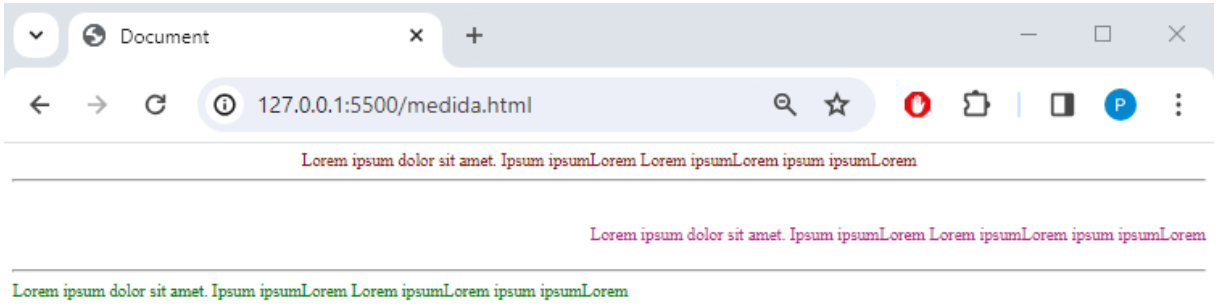
- cor de texto;
- cor de fundo;
- imagem de fundo.

Cor de texto

```css
.m {
  color: rgb(126, 21, 7);
   text-align: center;
}
.m1 {
   text-align: right;
   color: #b5258a;
}
.m2 {
  text-align: justify;
  color: green;
}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



Fonte: Elaborado pelo autor (2024)

Cor de fundo

```css
body{
   background-color: cornflowerblue;
}
```

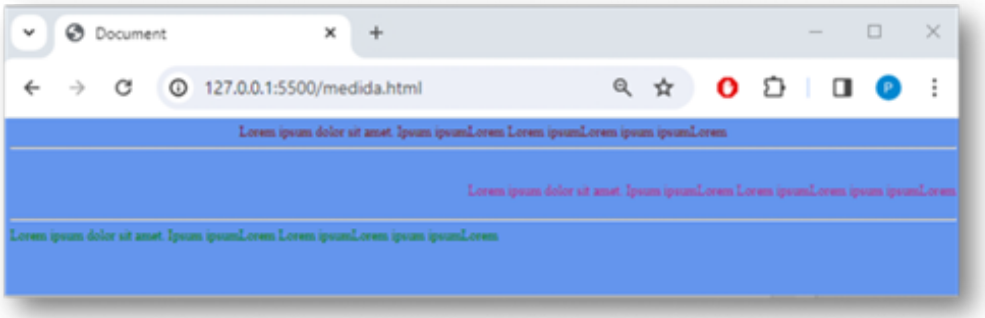
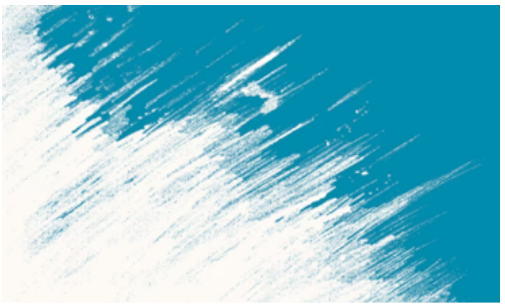


Imagem de fundo

Nesse caso, é fundamental que você tenha uma imagem salva na pasta imagens do projeto.

Exemplo de imagem de fundo:



Disponível em:<vector_corp-http://tinyurl.com/4svz7w4r>. Acesso em 06 jan. 2024.

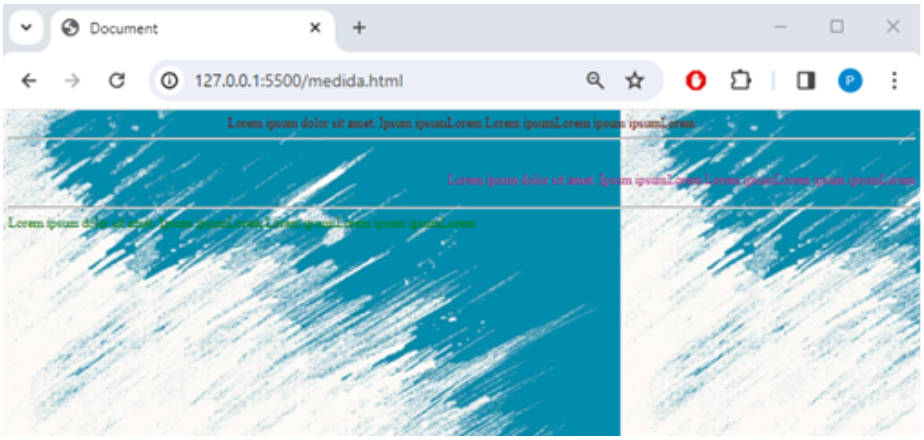
Aplicação com CSS

Aplique os códigos abaixo:

```css
body{
    background-image: url(img/img4.png);
    background-repeat:repeat-x;
    background-attachment: fixed;
    background-position: center;
}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



Fonte: Elaborado pelo autor (2024)

Font (fontes)

Ao trabalhar com a propriedade das fontes, existem algumas definições importantes. Para entender melhor, observe as propriedades e as descrições abaixo:

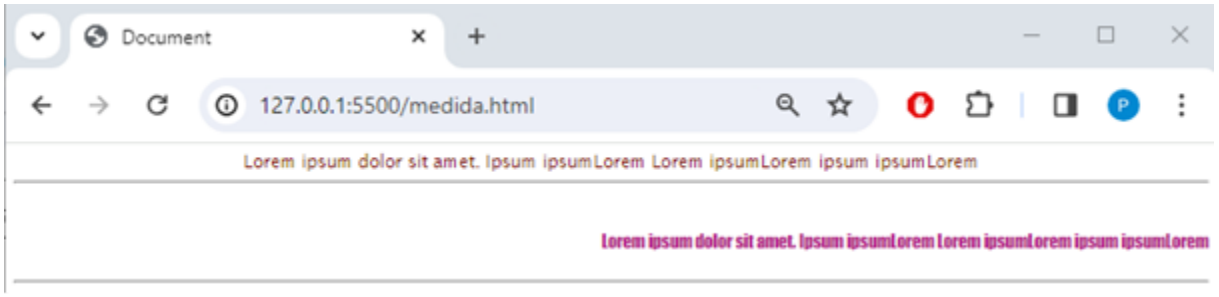
Propriedade	Descrição
font-family	Refere-se ao nome da fonte, ou seja, ao tipo da letra. Nesse caso, ao escolher o tipo de fonte, você verá, automaticamente, toda família a qual a fonte pertence.
font-style	Representa o estilo itálico (**italic**) ou oblíquo (**oblique**).
font-weight	Determina a largura da fonte (ex.: negrito ou *bold*).
text-decoration	Especifica uma decoração para o texto, como sublinhado (**underline**), riscado (**line-through**), sublinhado invertido (**overline**).
text-transform	Define se o texto será apresentado em letras maiúsculas (**uppercase**), minúsculas (**lowercase**) ou com as iniciais em maiúsculas (**capitalize**).
font-size	Especifica o tamanho da fonte.
letter-spacing	Refere-se ao espaçamento entre as letras.
Font-variant	Unidade utilizada para fontes maiúsculas de menor altura.
word-spacing	Indica o espaçamento entre palavras.

Para entender melhor a aplicação, utilize o Visual Code, seguindo os exemplos abaixo.

```
.m {
  color: rgb(126, 21, 7);
   text-align: center;
   font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode',
   Geneva, Verdana, sans-serif;
}
.m1 {
   text-align: right;
   color: #b5258a;
   font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



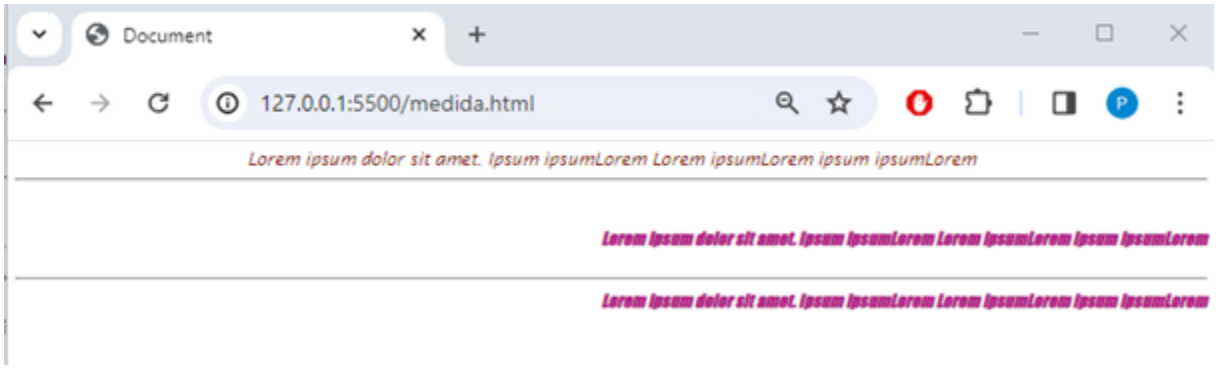
Fonte: Elaborado pelo autor (2024)

Pratique a aplicação de itálico e negrito seguindo os códigos abaixo:

```
.m {
  color: rgb(126, 21, 7);
   text-align: center;
   font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode',
   Geneva, Verdana, sans-serif;
   font-style: italic;
}
.m1 {
   text-align: right;
   color: #b5258a;
   font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
   font-weight: bold;
}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



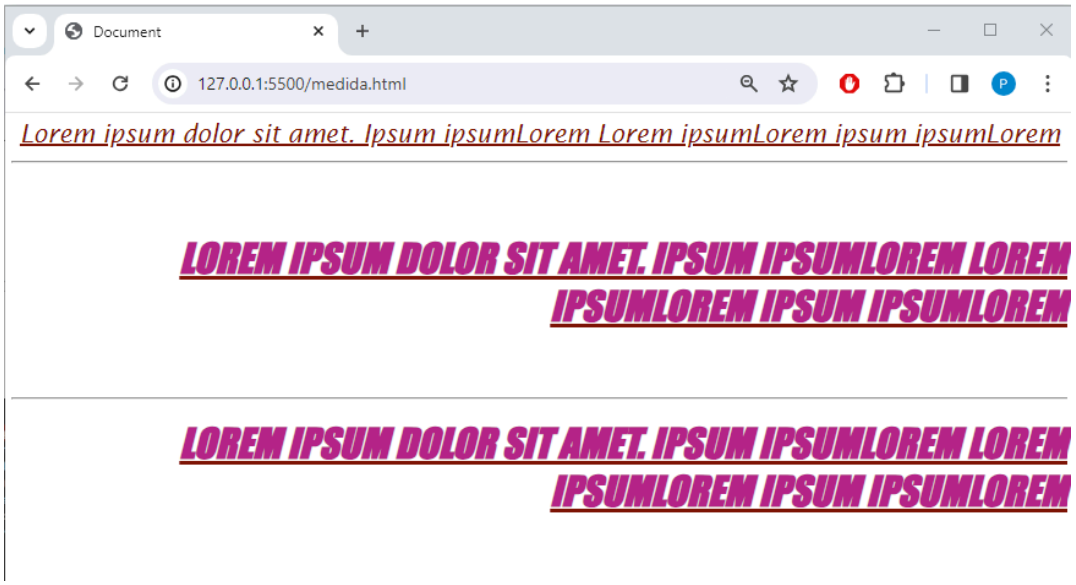
Fonte: Elaborado pelo autor (2024)

Pratique a aplicação de tamanho de fonte e sublinhado seguindo os códigos abaixo:

```css
.m {
  color: rgb(126, 21, 7);
   text-align: center;
   font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode',
   Geneva, Verdana, sans-serif;
   font-style: italic;
   font-size: 30px;
   text-decoration: underline;

}
.m1 {
   text-align: right;
   color: #b5258a;
   font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
   font-weight: bold;
   font-size: 3rem;
   text-transform: uppercase;
}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado

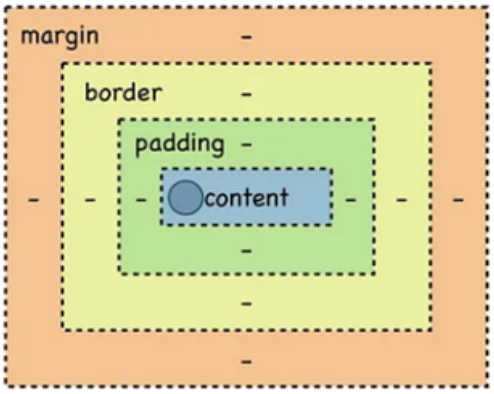


Fonte: Elaborado pelo autor (2024)

5.2 Modelo de camadas - *box model*

No CSS, o b*ox model* é como uma gaveta com divisórias que define o espaço ocupado por cada elemento na página. Imagine que cada elemento HTML está envolvido em uma caixa imaginária, composta por conteúdo, preenchimento, borda e margem. O conteúdo é como o coração da caixa, o preenchimento é a almofada que o envolve, a

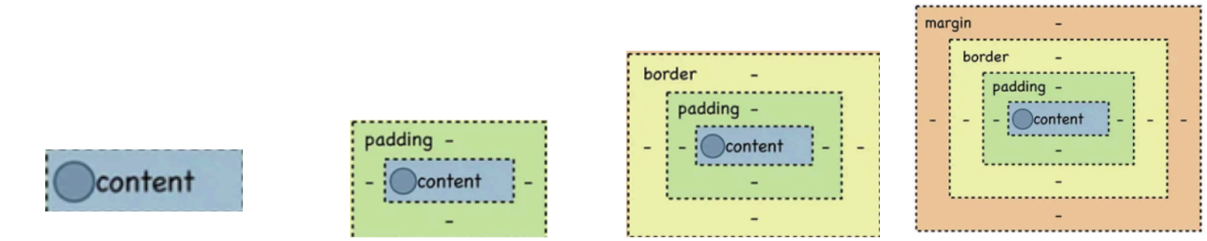
borda é o contorno estilizado e a margem é o espaço que mantém as coisas arrumadas.



Disponível em:<http://tinyurl.com/mvxnut3p>. Acesso em 06 jan. 2024.

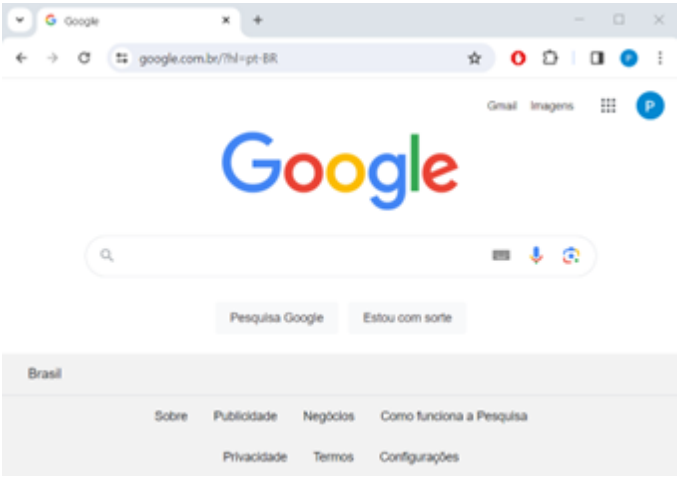
Quando aplicamos estilos, como **width**, **height**, **padding**, ou **border**, nós estamos moldando essa caixa. Compreender o *box model* é como dominar a arte de organizar e estilizar elementos, criando *layouts* visualmente harmoniosos.

Vamos conhecer cada um desses componentes e entender como eles permitem uma melhor organização ao aplicarmos a estilização na página.



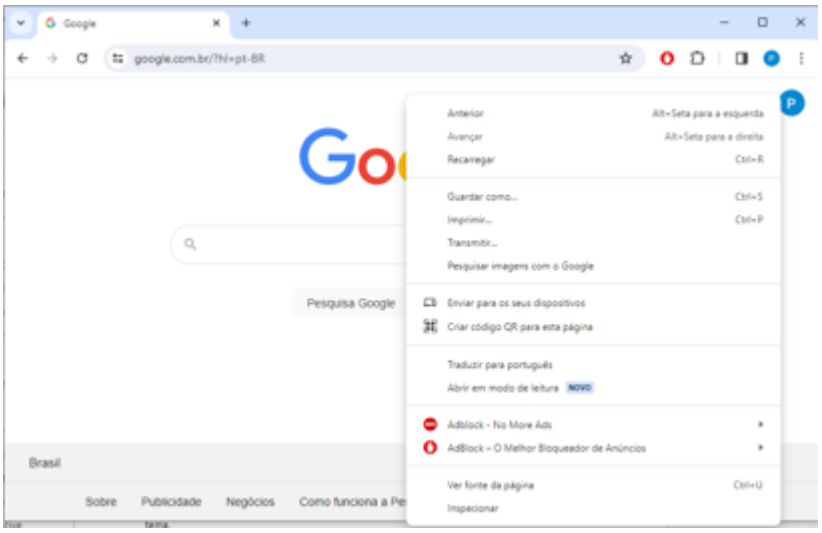
***Content* (conteúdo)**	***Padding* (preenchimento)**	***Border* (borda)**	***Margin* (margem)**
Refere-se ao espaço ocupado pelo conteúdo real do elemento. Por exemplo, no caso de um elemento de texto, o conteúdo seria o próprio texto.	Diz respeito à área de preenchimento ao redor do conteúdo e à borda do elemento. Serve para criar espaço entre o conteúdo do elemento e a sua borda.	As bordas são usadas para definir limites visuais ao redor do conteúdo e do preenchimento.	A margem é o espaço em branco entre o elemento e seus vizinhos, contribuindo para o espaçamento e o *layout* geral da página.

Uma das maneiras de verificar as dimensões do *box model* é pela própria página *web*. Para isso, acesse a página da Google como modelo.



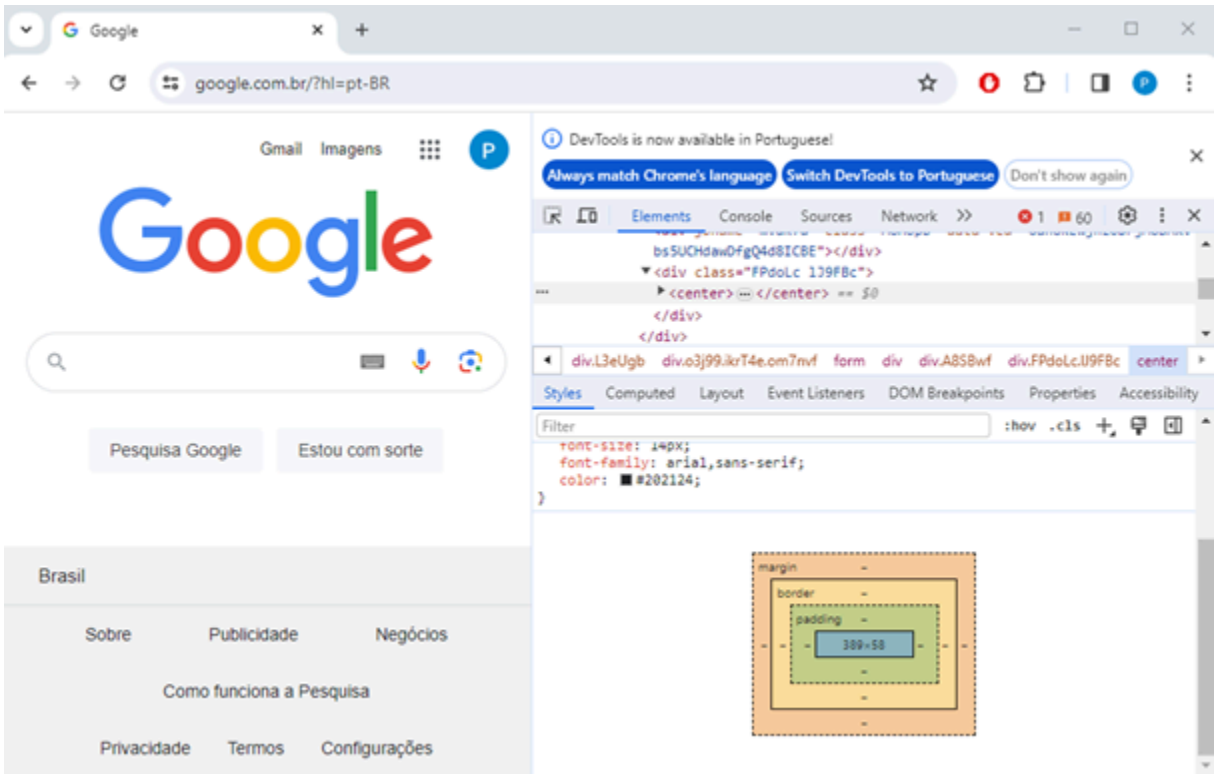
Fonte: Elaborado pelo autor (2024)

Na sequência, em qualquer área da página clique com o **botão direito do *mouse*** e, em seguida, escolha a opção **inspecionar.**



Fonte: Elaborado pelo autor (2024)

Em seguida, será apresentada uma página lateral. Utilize a rolagem para verificar as especificações do *box model.*



Fonte: Elaborado pelo autor (2024)

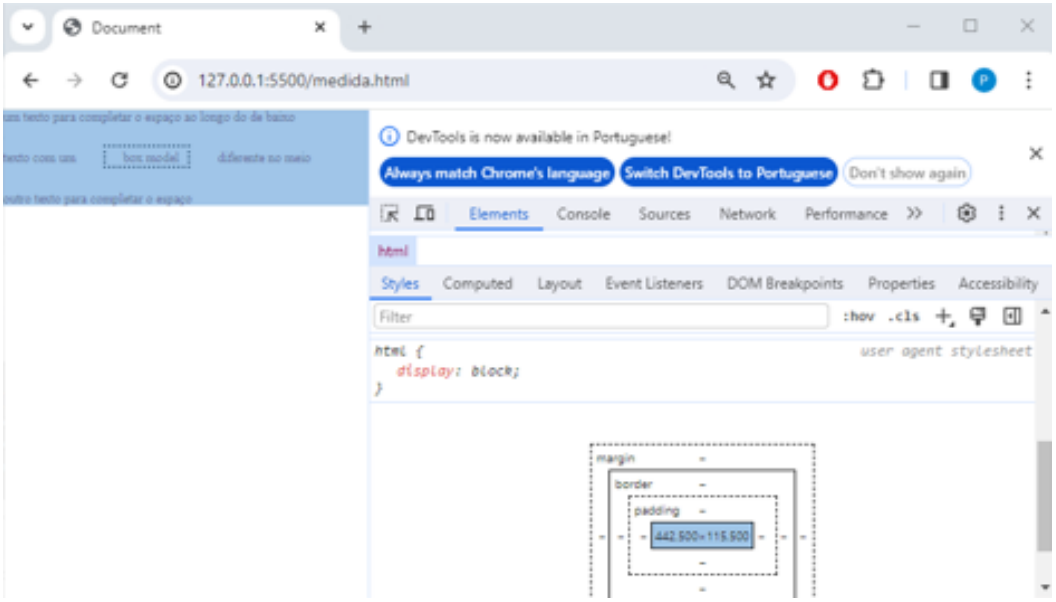
Ao passar o *mouse* sobre cada um dos elementos do *box model*, as informações de dimensionamento serão apresentadas na página principal.

IMPORTANTE!

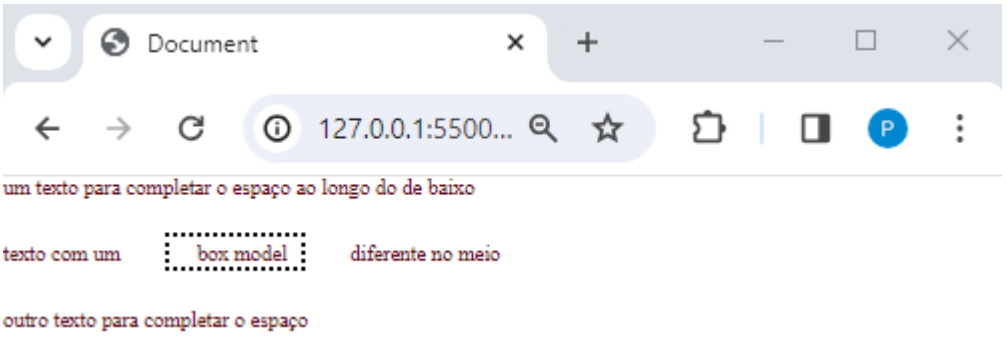
Os valores dos componentes são representados em ***pixels*** (**px**).

```
* { color: rgb(91, 9, 18); }
    span {
        border: 3px dotted rgb(0, 6, 6);
        padding: 5px 10px 2px 20px;
        margin: 30px;
    }
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024)

Vamos entender cada um dos elementos? Para isso, abra o seu Visual Code e faça os processos apresentados a seguir.

Aplicando o CSS Reset

Normalmente, cada navegador possui as propriedades de **margin**, **border** e **padding** já definidas**.** Sendo assim, uma técnica conhecida para zerar essas propriedades, redefinindo-as de acordo com o seu desenvolvimento, é o **CSS Reset**. Essa é uma prática recomendada para que o resultado visual esteja de acordo com qualquer navegador.

Siga o exemplo abaixo:

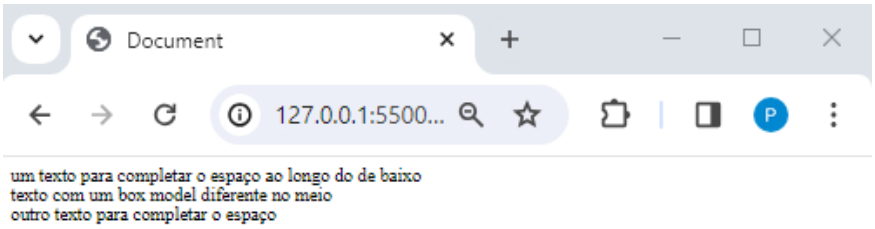
```
    <link rel="stylesheet" type="text/css" href="medida.css">
</head>
<body>
    um texto para completar o espaço ao longo do de baixo<br>
    texto com um <span>box model</span> diferente no meio<br>
    outro texto para completar o espaço

</body>
</html>
```

Fonte: Elaborado pelo autor (2024)

```
# medida.css > *
1
2   * {
3       padding: 0;
4       margin: 0;
5       border: 0;
6   }
```

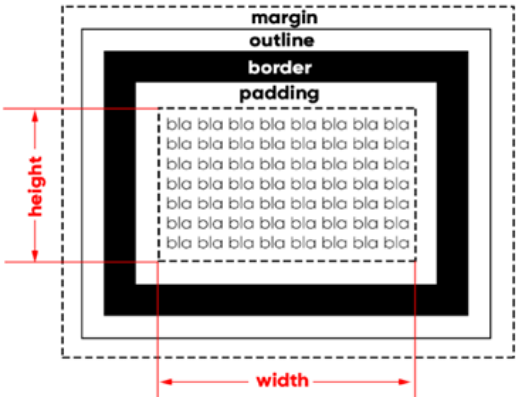
Fonte: Elaborado pelo autor (2024)

Resultado esperado



Fonte: Elaborado pelo autor (2024)

Entendendo os elementos da caixa



Disponível em:<http://tinyurl.com/cza34fbx>. Acesso em: 08 jan. 2024.

Assim como já estudamos, tudo começa no **conteúdo** (**content**), que, no nosso

exemplo, está representado pelo texto em geral (**bla bla bla bla..**). Normalmente, toda caixa começa apenas com o conteúdo, sem as demais propriedades.

No entanto, todo conteúdo tem a possibilidade de ter largura (**width**) e altura (**height**) aplicadas, esse conjunto de propriedades sendo chamado de **box-size (tamanho da caixa).**

Fique de olho!

Vale reforçar que o tamanho da caixa não inclui as medidas do **padding**, **border** ou **margin**.

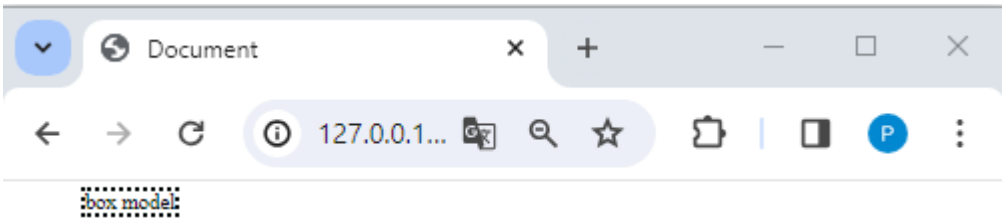
Antes de começarmos a realizar os exemplos, entenda as características de cada uma dessas propriedades:

Propriedade	Características
margin	**margin-top** (margem superior), **margin-right** (margem da direita), **margin-bottom** (margem inferior) e **margin-left** (margem da esquerda).
padding	**padding-top** (enchimento superior), **padding-right** (enchimento da direita), **padding-bottom** (enchimento inferior) e **padding-left** (enchimento da esquerda).
border	**border-top** (borda superior), **border-right** (borda da direita), **border-bottom** (borda inferior) e **border-left** (borda da esquerda).

Vamos entender melhor com os exemplos abaixo. Lembre-se de executar cada

um deles no Visual Code para facilitar seu entendimento!

- **Passo 1 - Observe o estado da página antes de executar as aplicações:**



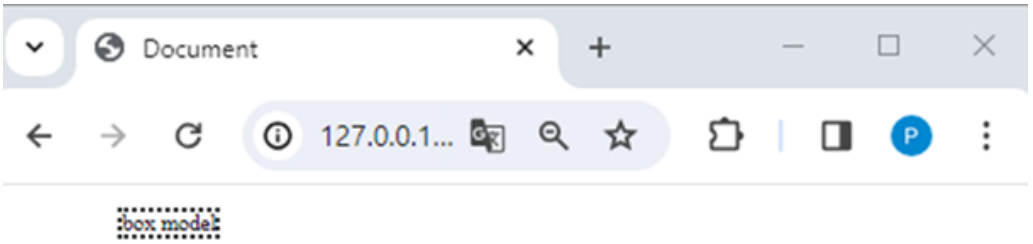
Fonte: Elaborado pelo autor (2024)

- **Passo 2 - Agora, nós vamos aplicar margem na página seguindo a orientação abaixo:**

```
*{
    margin-top: 10px;
    margin-left: 20px;
}
```

Fonte: Elaborado pelo autor (2024)

- **Resultado esperado:**



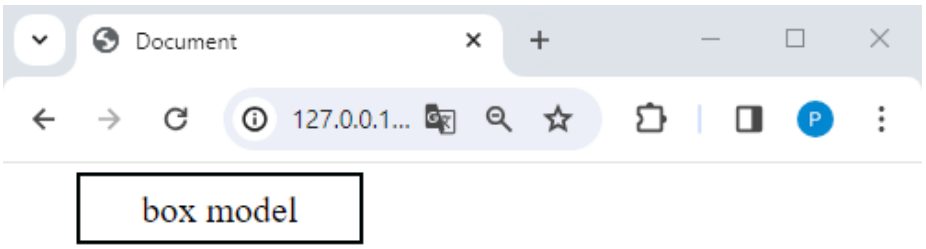
Fonte: Elaborado pelo autor (2024)

- **Passo 3 – Em seguida, aplique o padding (espaçamento entre a borda e o texto).**
    - Para isso é importante, considerar os espaçamentos da altura superior e inferior, bem como dos lados direito e esquerdo. Por isso, deve-se definir essas quatro medidas.

```
span {
        font-size: 30px;
        border: 3px solid rgb(0, 6, 6);
        padding: 10px 50px 10px 50px;

    }
```

Fonte: Elaborado pelo autor (2024)

- **Resultado esperado**



Fonte: Elaborado pelo autor (2024)

Border (bordas)

As bordas, assim como o próprio nome já indica, permitem adicionar **contornos** em volta dos elementos do HTML. Elas possuem características diferentes e apropriadas para definir o **tamanho**, a **cor** e os **estilos**.

Veja algumas das propriedades na tabela abaixo.

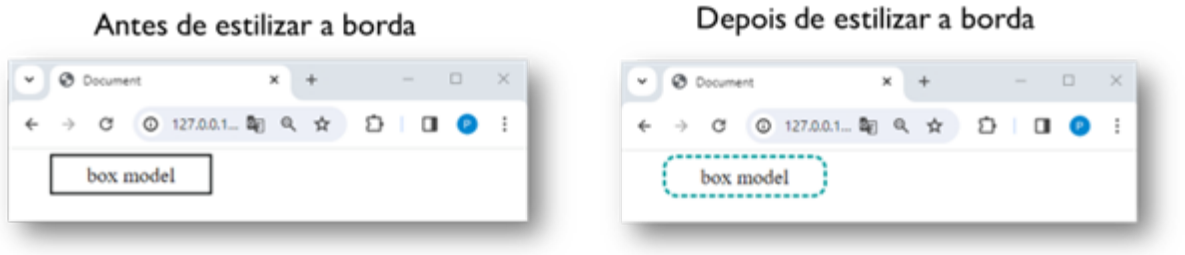
Propriedade das bordas	Características
border-color	Determina a cor da borda que, por sua vez, poderá ser definida pelos diferentes sistemas de cores.
border-style	Permite alterar os estilos da borda, que podem ser: • **none**: sem borda; • **hidden**: é semelhante ao estilo anterior, isto é, apesar de existir uma borda no contorno do elemento, ela não aparecerá na página; • **dotted**: borda pontilhada; • **dashed**: borda tracejada; • **solid**: borda contínua.
border-radius	Permite adicionar cantos arredondados à borda.

Vamos entender melhor com os exemplos a seguir. Lembre-se de executar cada um deles no seu Visual Code para um melhor entendimento!

```css
span {
    font-size: 30px;
    border: 5px dashed #099f9f;
    border-radius: 20px;
    padding: 10px 50px 10px 50px;

}
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



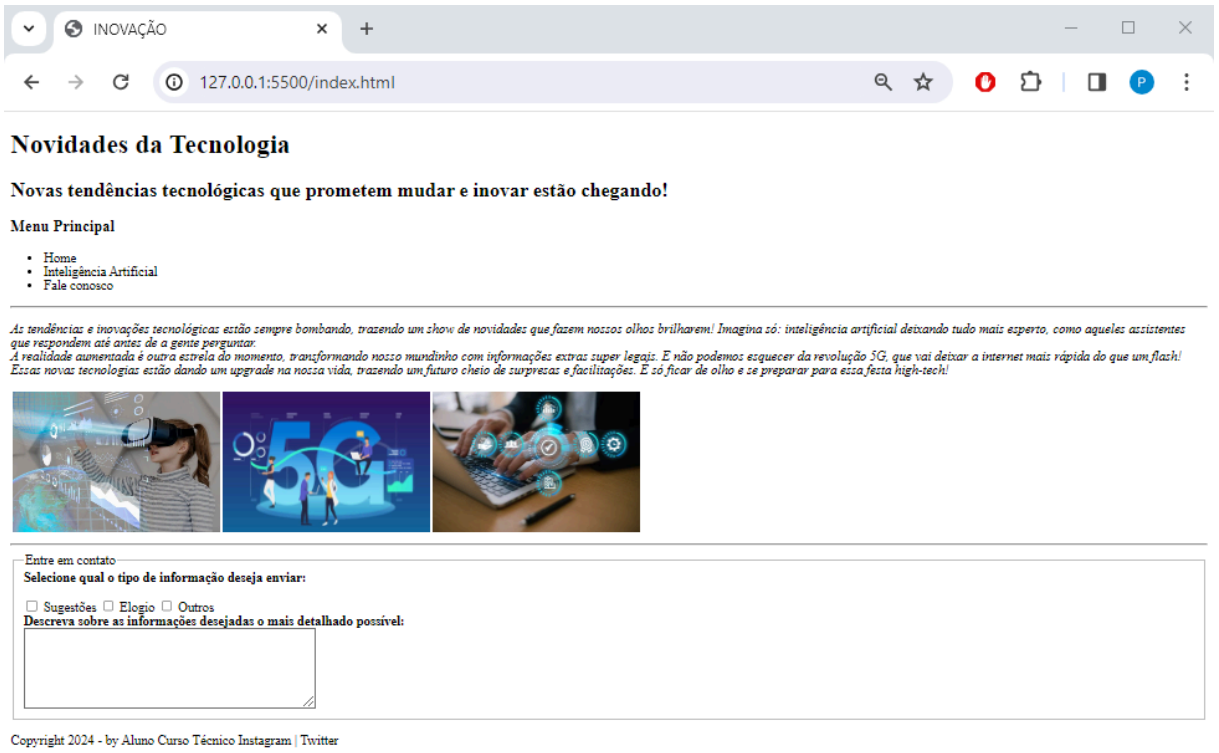
Fonte: Elaborado pelo autor (2024)

5.3 Elaborando uma página com HTML e CSS3

Após colher experiências valiosas com relação ao CSS, é hora de você embarcar em uma jornada criativa renovada. Assim como um artista diante de uma tela em branco, nós temos a oportunidade de aplicar aprendizados, experimentar novas ideias e transformar desafios passados em oportunidades para brilhar.

Com a bagagem do conhecimento adquirido, cada projeto é uma tela novinha em folha, esperando para ser preenchida com inovação e criatividade. Esse é o momento em que colocaremos todo aprendizado em prática. Para isso, daremos continuidade à primeira página trabalhada somente com as *tags* HTML. Porém, vamos usar o CSS para estilizar cada ponto dela.

Lembre-se que nossa **primeira página** se encontra totalmente "crua" em relação à sua formatação, assim como se pode ver na imagem abaixo.



Fonte: Elaborado pelo autor (2024)

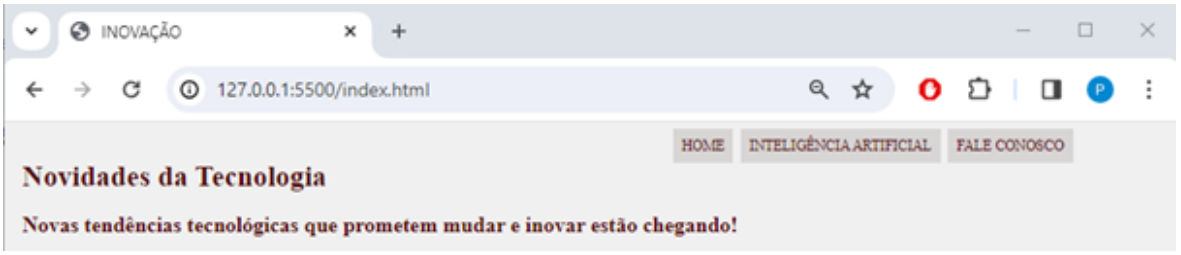
Agora, vamos colocar a mão na massa e deixar a página um pouco mais atraente? Para isso, abra o seu projeto **PrimeiraPagina**, crie um arquivo **style.css** e faça o *link* com sua Primeira Página.

Estilizando o cabeçalho

No caso do cabeçalho, nós vamos:

- ajustar a cor do texto utilizando o **ID=topo**;
- ajustar o menu para que ele fique no topo da página e que tenha um efeito ao passar o *mouse* sobre ele.

Com essas alterações, o cabeçalho deverá ficar assim:



Fonte: Elaborado pelo autor (2024)

Agora, veja como foram trabalhadas as propriedades do CSS em conjunto com as *tags* e os atributos do HTML.

Reproduza cada um dos exemplos abaixo no seu Visual Code para uma melhor compreensão.

```
body{
    font-family:'Times New Roman', Times, serif;
    background-color: #f0f0f0;
    margin: 10px;
    padding: 10px;
    }
     /*formatação cabeçalho - dando cor ao texto*/
    #topo {
    color: #510606;
  }
```

Fonte: Elaborado pelo autor (2024)

```
  /*formatação do menu - usando a Tag nav em conjunto com o ID menu para formatar a lista*/
nav#menu ul{
    list-style: none;
    text-transform: uppercase;
    position: absolute;
    top: -10px;
    left: 700px ;
    }
```

Fonte: Elaborado pelo autor (2024)

```
/*colocando cada item da lista um ao lado do outro*/
nav#menu li{
display:inline-block ;
background-color: #d8d5d5;
padding: 10px;
margin: 2px;
transition: background-color 1s;
}
```

Fonte: Elaborado pelo autor (2024)

```css
/*bloqueando os itens do menu*/
nav#menu {
  display:block;
}
 /*para não aparecer o título MENU PRINCIPAL*/
nav#menu h3 {
  display:none;
}
 /*estilizando o item da lista para ter um efeito quando passar o mouse*/
nav#menu li:hover{
  background-color: #606060;
}
```

Fonte: Elaborado pelo autor (2024)

Estilizando o conteúdo da página

Para começar a estilizar nosso conteúdo, nós vamos começar pelo texto. Como sabemos, em nosso conteúdo nós temos o uso de *tags* importantes e, além do texto, podemos lidar com imagens, tabelas e formulários.

Para iniciar, vamos definir o **estilo da fonte**, o seu **tipo** e o seu **tamanho**. Observe abaixo:

```css
.destaque{
font-style: normal;
font-size: 20px;
font-family: Arial, Helvetica, sans-serif;

}
```

Fonte: Elaborado pelo autor (2024)

Agora, vamos trabalhar com as imagens, sem haver a necessidade de sempre manipular tabelas. Para isso, no arquivo **index.html**, crie uma ** ** com uma classe interna denominada "**tabela**", conforme exemplificado abaixo:

```html
<div class="tabela">
<table>
    <tr>
    <td><img width="250" height="170" src="img/img1.png"></td>
    <td><img width="250" height="170" src="img/img2.png"></td>
    <td><img width="250" height="170" src="img/img3.png"></td>
    </tr>

  </table>
</div>
```

Fonte: Elaborado pelo autor (2024)

Na sequência, nós vamos aplicar a seguinte estilização nas imagens:

```css
.tabela img{
    width: 90%;
    height: 70%;
    border-radius: 15px;

  border: 4px solid transparent;
  background:linear-gradient(#e7cb16eb, #bb0505, #081280) padding-box,
  linear-gradient(260deg, #e7cb16eb, #bb0505, #081280)border-box;
  transition:  transform 0.3s;
  }
```

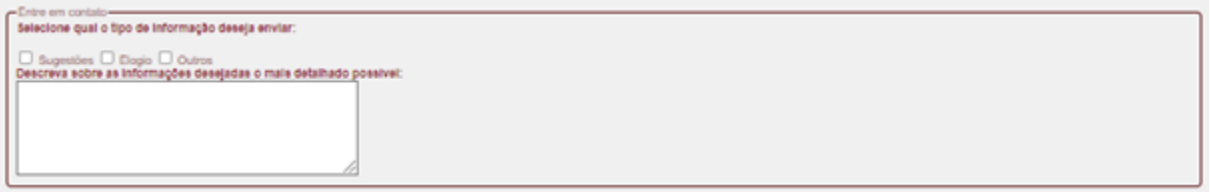
Fonte: Elaborado pelo autor (2024)

Fonte: Elaborado pelo autor (2024)

Para finalizar a estilização do conteúdo da página, vamos trabalhar com alguns aspectos do formulário. Observe a imagem abaixo e reproduza no seu Visual Code.

```css
/*tag <fieldset>*/
  fieldset {
   border-color: #510606;
  }
   /* formatação de texto do formulário */
   .grupo{
       font-family: sans-serif;
       font-size: 12px;
       color: #901e31;
       border-radius: 5px;
   }
```

Fonte: Elaborado pelo autor (2024)



Fonte: Elaborado pelo autor (2024).

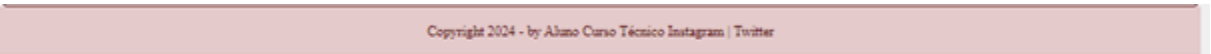
Estilizando o rodapé

Agora, para estilizar o rodapé, siga o exemplo da imagem a seguir.

```css
footer {
    text-align: center;
    padding: 5px;
    background-color: #e4caca;
    color: #510606;
    font-size: 16px;
     }
```

Fonte: Elaborado pelo autor (2024)

Resultado esperado



Fonte: Elaborado pelo autor (2024)

Veja, na imagem abaixo, o resultado final de uma aplicação de CSS básica.

Fonte: Elaborado pelo autor (2024)

DESAFIO PRÁTICO

Estilização de elementos HTML com CSS3

Desafio: Trabalhando com formulários.

Descrição

Agora que adquiriu conhecimento suficiente para trabalhar com a estilização de elementos, você recebeu a tarefa de aprimorar o formulário criado na sua última demanda de trabalho. Utilize os recursos aprendidos para estilizar o formulário de pesquisa, tornando-o mais atraente e convidativo para o usuário.

Objetivos

- Demonstrar conhecimento com propriedades de formatação em CSS;
- Trabalhar com a estilização do máximo de elementos que conseguir, tendo cuidado com os exageros;
- Salvar o projeto em um repositório como formulário de pesquisa estilizado.

Orientações

- Verifique no seu repositório o formulário salvo com o nome **FORMULÁRIO DE PESQUISA GERAL;**
- Abra o arquivo e crie um outro para trabalhar a folha de estilo separadamente do documento HTML;
- Formate o **Fieldset** para que tenha cor na linha;
- Garanta que todo o texto do formulário tenha uma única cor.;
- Deixe o **textarea** com os cantos arredondados;
- Não se esqueça de adicionar um título apropriado e estilizado, assim como definir o rodapé.

Estilização de elementos HTML com CSS3

Desafio: Estruturando um projeto.

Descrição

A página de *links* precisa de aprimoramentos! Portanto, você está encarregado de melhorar toda a página, incluindo o cabeçalho e o rodapé. Considere que todo o conteúdo já foi desenvolvido anteriormente ao trabalhar com os *links* para o YouTube. Além disso, será necessário adicionar imagens relacionadas aos *links* existentes.

Objetivos

- Demonstrar conhecimento no aperfeiçoamento de páginas pré-desenvolvidas;
- Criar a estrutura de uma página com os recursos aprendidos;
- Trabalhar toda estilização do *site*, ou seja, cabeçalho, rodapé, conteúdo e imagens.

Orientações

- Aprimore a estilização da página com o título "**Acesso direto**", utilizando a formatação apropriada para títulos;
- Melhore a apresentação do texto explicativo sobre a finalidade dessa página;
- Utilize os recursos aprendidos para apresentar as informações sobre os *links*, incluindo as imagens correspondentes a eles, juntamente com seus respectivos *links;*
- Adicione efeitos às imagens para que sejam ativados ao passar o *mouse* sobre elas.

ATIVIDADE DE FIXAÇÃO

1. Descreva o papel das propriedades **margin** e **padding** no *box model* do CSS, destacando como essas propriedades influenciam o *layout* de um elemento na página.

2. Explique como o *box model* é aplicado na construção de um componente específico em um *site*. Destaque a relação entre conteúdo, *padding*, borda e margem e como isso afeta a apresentação visual.

3. Em sua visão, fale sobre a importância de dominar o CSS ao trabalhar em projetos *web*. Como as propriedades de estilo contribuem para a usabilidade e a estética? De que forma isso impacta a experiência do usuário final?
4. Você está estilizando um cabeçalho para um *site* de viagens. Qual propriedade

CSS você usaria para definir a cor de fundo do cabeçalho?

a. **background-color**
b. **text-color**
c. **color-background**
d. **bg-color**

5. Ao criar uma seção de destaques em um *blog*, como você aplicaria uma margem de 10 *pixels* ao redor de cada caixa usando o CSS?

a. **margin: 10px;**
b. **padding: 10px;**
c. **spacing: 10px;**
d. **margin-box: 10px;**

6. Em um *site* de comércio eletrônico, você deseja que os botões de "**Adicionar ao Carrinho**" tenham uma borda e que o texto fique em negrito. Como você aplicaria isso no CSS?

a. **button-style: bold, border;**
b. **font-weight: bold; border: 1px solid;**
c. **border-style: bold; font: bold;**
d. **button: bold, border;**

7. Ao criar um formulário de contato, você percebe que os campos estão muito próximos uns dos outros. Como as propriedades do *box model*, como **padding** e **margin**, podem ser aplicadas para melhorar o espaçamento e a organização dos elementos?

8. Você está projetando o estilo de um *blog* e precisa destacar os títulos. Como você utilizaria as propriedades **color** e **font-size** do CSS para tornar os títulos visualmente atraentes?

9. Você está desenvolvendo um *site* e deseja estilizar o título principal da página. Qual propriedade do CSS você usaria para definir a cor do texto desse título?

a. **text-color**
b. **color**
c. **font-color**
d. **title-color**

10. Ao criar um *layout* para uma seção de notícias, você percebe que os títulos estão

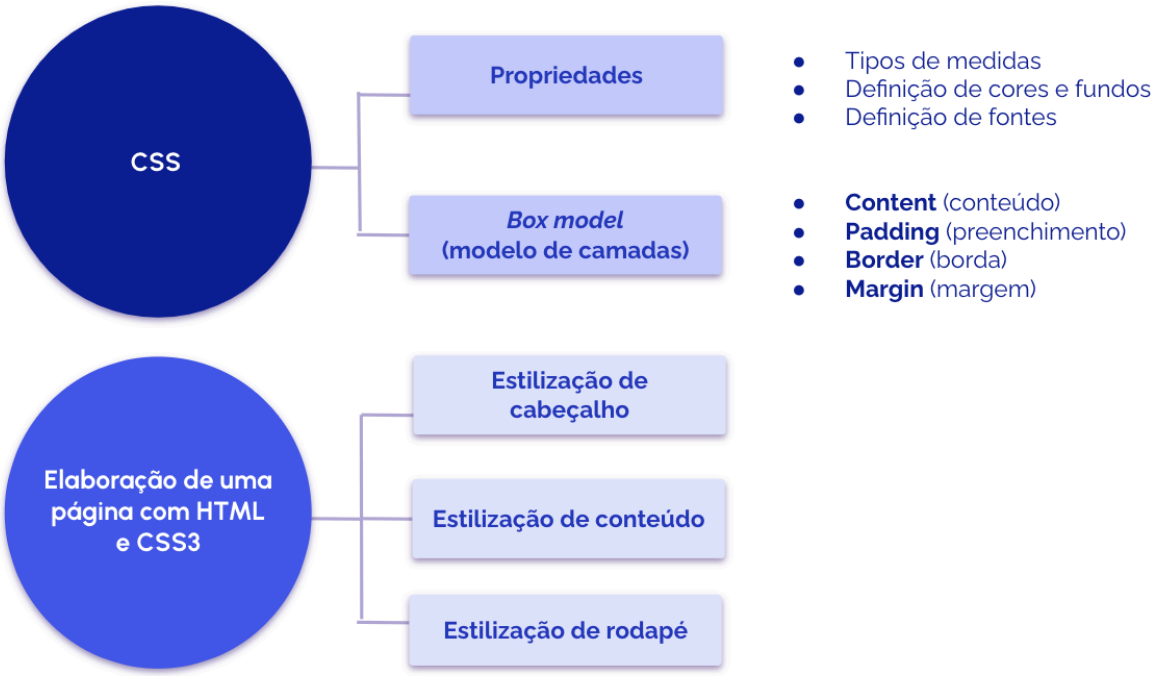
muito próximos uns dos outros. Qual propriedade do CSS você usaria para adicionar espaço entre os títulos?

a. **padding**
b. **margin**
c. **spacing**
d. **line-height**

Nesse tema, aprendemos sobre a importância de dominar propriedades e a estilização final usando CSS para criar páginas *web* visualmente atraentes e funcionais. Inicialmente, mergulhamos nas propriedades de estilos comuns, explorando como diferentes tipos de medidas e unidades de medidas com CSS são aplicados para alinhar parágrafos e definir o *layout* dos elementos na página. A partir da definição de cores e fundos, abordamos como personalizar a cor do texto, cor de fundo, e como utilizar imagens de fundo para enriquecer o *design* visual das páginas.

Avançamos para o estudo das fontes, enfatizando a escolha e aplicação de fontes para melhorar a legibilidade e estética do conteúdo. Também exploramos o modelo de camadas, ou *box model*, essencial para o design *responsivo*, incluindo a aplicação do CSS Reset para garantir a consistência entre diferentes navegadores. Discutimos os elementos da caixa, como bordas, e como esses elementos influenciam o espaçamento e a estrutura da página.

Finalmente, aplicamos todo o conhecimento adquirido na elaboração de uma página com HTML e CSS3, focando em estilizar cabeçalhos, conteúdos e rodapés. Este processo culminou na criação de uma página *web* completa, em que as orientações detalhadas ajudaram a alcançar os resultados esperados, demonstrando a transformação de um *layout* básico em uma página sofisticada e bem estruturada. Este capítulo fornece uma base sólida para o desenvolvimento de habilidades em CSS, essenciais para qualquer aspirante a desenvolvedor *web*.


CSS
Propriedades
Box model
(modelo de camadas)
Elaboração de uma página com HTML e CSS3
Estilização de cabeçalho
Estilização de conteúdo
Estilização de rodapé
Tipos de medidas
Definição de cores e fundos
Definição de fontes
Content (conteúdo)
Padding (preenchimento)
Border (borda)
Margin (margem)

CAPÍTULO 06

Explorando técnicas de *layout* e posicionamento com CSS3

O que esperar deste capítulo:

- Posicionar elementos com **position: relative** e **position: absolute**;
- Fazer l*ayouts* flexíveis com **display: flex**;
- Criar *designs* responsivos com **@media queries**.

6.1 A importância do uso dos posicionamentos

O posicionamento em CSS desempenha papel crucial na criação de *layouts* responsivos e visualmente agradáveis. Ele permite controlar a disposição e o empilhamento de elementos na página, garantindo que o *design* seja flexível e se ajuste a diferentes tamanhos de tela e dispositivos.

Para começar, vamos explorar alguns pontos-chave. Eles são:

- controle preciso – o posicionamento em CSS oferece um controle preciso sobre a localização de elementos na página. Isso é essencial para garantir que os elementos sejam colocados exatamente onde são necessários, evitando sobreposições indesejadas;
- Responsividade – ao utilizar técnicas de posicionamento, os desenvolvedores podem criar *layouts* responsivos e que se adaptam a diferentes dispositivos, tamanhos de tela e orientações. Isso é essencial para proporcionar uma experiência consistente e amigável em uma variedade de contextos, como *desktops*, *tablets* e *smartphones*, assim como podemos ver na imagem abaixo.

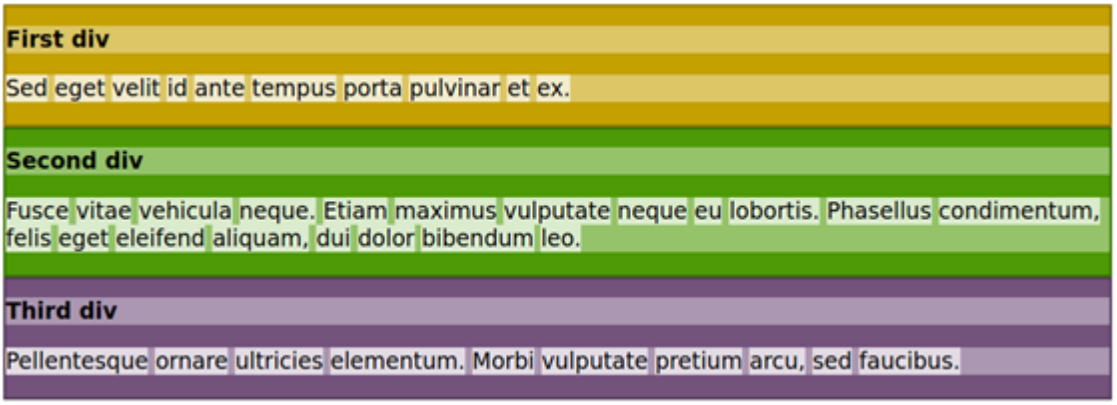
Disponível em: <http://tinyurl.com/muszkafz>. Acesso em 30 jan. 2024.

6.1.2 Modelo de posicionamento em CSS

O modelo de posicionamento em CSS define como os elementos HTML são exibidos na página, determinando como eles se relacionam uns com os outros e com o fluxo normal do documento. Existem dois principais tipos de posicionamento: o posicionamento padrão e o posicionamento específico. Vamos conhecer melhor cada um?

Posicionamento padrão

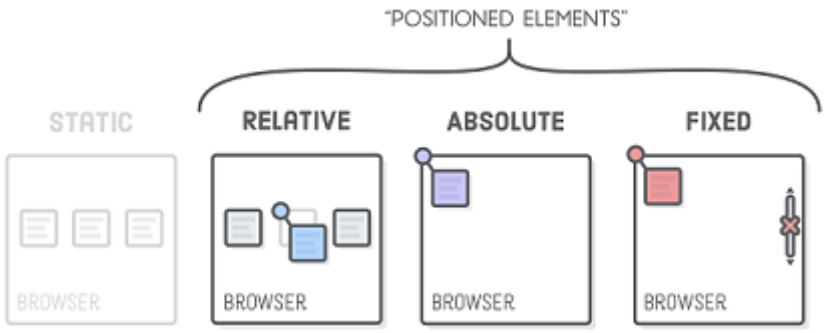
No posicionamento padrão, os elementos são exibidos conforme o fluxo normal do documento. Eles são empilhados de acordo com a ordem em que aparecem no código. O posicionamento é influenciado pelo tipo de elemento e pelas propriedades de estilo aplicadas. Elementos em posicionamento padrão respeitam o *layout* normal da página, ocupando espaço conforme o seu conteúdo e as suas propriedades de estilo.



Disponível em: <http://tinyurl.com/mr3spnsf>. Acesso em 30 jan. 2024.

Posicionamento específico

No posicionamento específico, as pessoas desenvolvedoras têm mais controle sobre a localização e o empilhamento dos elementos. Propriedades como **position, top, right, bottom** e **left** são utilizadas para ajustar a posição dos elementos em relação ao seu contexto, que pode ser o documento, o elemento pai ou outro elemento específico. O posicionamento específico inclui técnicas como **position: relative, position: absolute, position: fixed**, entre outras.



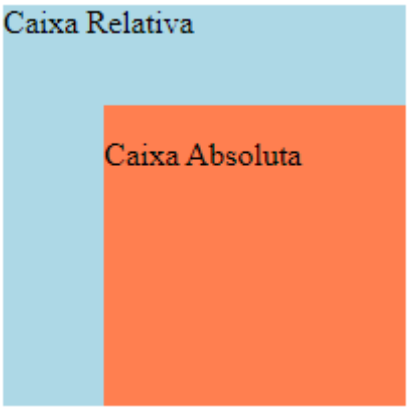
Disponível em: <http://tinyurl.com/rxzct2yj>. Acesso em 30 jan. 2024.

6.1.3 Posicionamento de elementos com position: relative, position: absolute

No mundo do desenvolvimento *web*, saber posicionar elementos de forma precisa é essencial para criar *layouts* bonitos e interativos. Duas ferramentas importantes para isso são as propriedades CSS **position: relative** e **position: absolute**. Essas propriedades dão aos desenvolvedores o poder de **organizar a aparência dos elementos em uma página**, indo além do *layout* básico.

Imagine essas propriedades como ferramentas para organizar os móveis em um quarto. Nesse contexto, a propriedade **position: relative** seria o equivalente a movimentar um sofá dentro do cômodo, ajustando a sua posição em relação aos demais móveis. Por outro lado, **position: absolute** seria comparável a pegar uma cadeira e posicioná-la em um local específico, independentemente da disposição dos outros móveis no quarto, podendo, por exemplo, colocá-la sobre a cama.

Vejamos, agora, outro exemplo mais prático. Se você quiser criar uma caixa de texto que fique dentro de outra caixa colorida na sua página, é possível usar **position: relative** na caixa maior, ajustando a posição da caixa de texto dentro dela. Por outro lado, para posicionar a caixa de texto em um canto específico, mesmo que haja outros elementos, você usa **position: absolute**. Veja, na imagem a seguir, como isso seria feito.



Fonte: elaborado pelo autor (2024).

Essas propriedades ajudam os desenvolvedores a criarem *layouts web* personalizados e visualmente atraentes.

6.1.4 Position: absolute

Em CSS, é como se a propriedade **position: absolute** fornecesse superpoderes a um elemento na página. Imagine que esse elemento pode sair do lugar comum em que todos os outros estão e ser posicionado em qualquer lugar, como se estivesse "flutuando" sobre o restante da página.

A magia acontece quando você diz para esse elemento posicionar-se em relação ao seu "pai" mais próximo que está usando **position: relative**. Assim sendo, é como se houvesse um ponto de referência especial no elemento e você utilizasse os comandos **top**, **right**, **bottom** e **left** para dizer exatamente onde esse ponto de referência está em relação ao elemento pai.

Por exemplo, se você tem um bloco de texto em uma página e quer que uma imagem fique ao lado desse bloco, mas um pouco mais para cima, você pode usar **position: absolute.** É como dizer à imagem: "posicione-se em relação ao bloco de texto, mas fique um pouco mais para cima".

Essa propriedade é uma maneira poderosa de personalizar a posição dos elementos em uma página. Para que você compreenda melhor, considere o seguinte exemplo:

- HTML:

```html
<body>
 <div class="mae">
   <div class="filha">
   </div>
 </div>
</body>
```

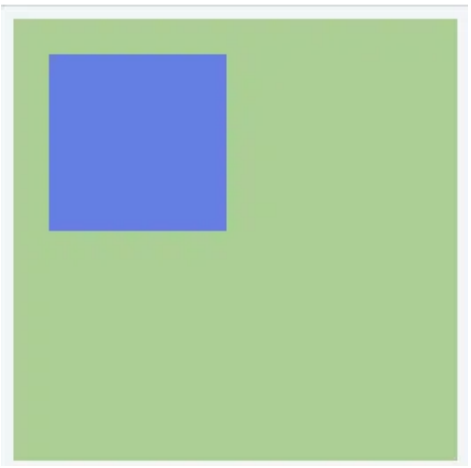
Fonte: elaborado pelo autor (2024).

- CSS:

```css
.mae{
  position: relative;
  width: 250px;
  height: 250px;
  background-color: #A1D490;
}

.filha{
  position: absolute;
  top: 20px;
  left: 20px;
  width: 100px;
  height: 100px;
  background-color: #6475E8;
}
```

Fonte: elaborado pelo autor (2023).

A partir desse código, teremos o seguinte resultado:



Fonte: elaborado pelo autor (2024).

6.1.5 Position: relative

Ao utilizar **position: relative**, temos a capacidade de ajustar o posicionamento de um elemento em relação à sua posição original. Esse método é valioso quando desejamos fazer pequenos ajustes ao mover um elemento dentro do seu próprio espaço na página. Vamos considerar o exemplo prático a seguir:

- HTML:

```
<div class="container">
  <div class="texto">
    <p>Este é um exemplo prático de position: relative.</p>
  </div>
</div>
```

Fonte: elaborado pelo autor (2023).

- CSS:

```
.container {
  width: 300px;
  height: 200px;
  border: 2px solid #ccc;
}

.texto {
  position: relative;
  left: 20px;
  top: 10px;
  background-color: #f0f0f0;
  padding: 10px;
}
```

Fonte: elaborado pelo autor (2023).

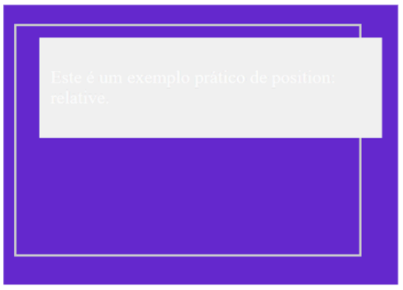
🤔 Vamos entender melhor esse código?

Seletor de *Container*

- **width: 300px**: define a largura do elemento com a classe *container* como 300 *pixels*;
- **height: 200px**: define a altura do elemento com a classe *container* como 200 *pixels*;
- **border: 2px solid #ccc**: adiciona uma borda de dois *pixels*, sólida e de cor cinza (#ccc) ao elemento com a classe **.container**.

Texto seletor

- **position: relative**: embora isso seja redundante, já que **.texto** é um elemento filho de **.container** que já tem **position: relative**, essa propriedade pode ser incluída para maior clareza;
- **left: 20px**: move o elemento **.texto** 20 *pixels* para a direita em relação à sua posição original dentro do elemento **.container**;
- **top: 10px**: move o elemento .texto dez *pixels* para baixo em relação à sua posição original dentro do elemento **.container**;
- **background-color: #f0f0f0**: define a cor de fundo do elemento **.texto** como um tom de cinza claro (#f0f0f0);
- **padding: 10px**: adiciona um preenchimento interno de dez *pixels* ao redor do conteúdo do elemento "**.texto**".



Resultado do código na página.

Fonte: elaborado pelo autor (2024).

6.1.6 Elaborando um case

Agora que você já se familiarizou com as propriedades de posicionamento, vamos criar um caso prático para trabalhar o que aprendemos. O objetivo é criar um componente HTML, mas utilizando as propriedades **position: absolute** e **position: relative**. No exemplo, usaremos a **criação de um cartão de perfil de usuário** que pode ser posicionado em uma página, conforme for necessário. Para isso, siga os seguintes passos:

- crie a estrutura HTML;
- implemente o posicionamento, utilizando **position: absolute** e **position: relative**.

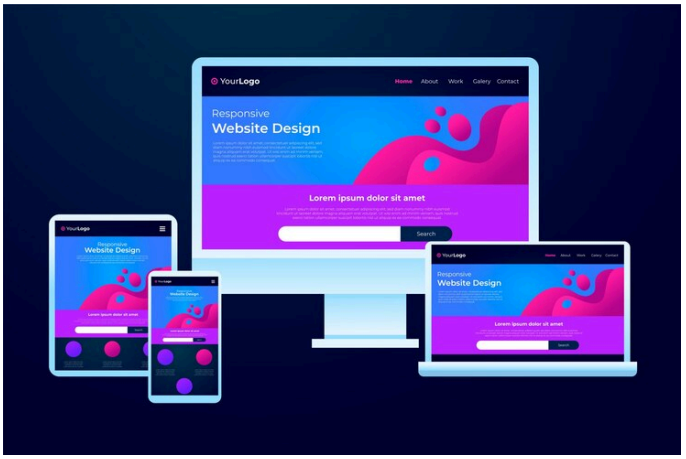
A ideia é construir um *card* de um perfil que se pareça com o seguinte modelo:

Fonte: elaborado pelo autor (2024).

6.2 *Layouts* flexíveis com display: flex

Atualmente, uma variedade de dispositivos, como *smartphones*, *tablets* e *laptops*, são amplamente utilizados para acessar a internet. Por conta dessa diversidade, a adaptação eficiente dos conteúdos para cada tela torna-se crucial. É aqui que entram os *layouts* flexíveis, pois são uma abordagem essencial para garantir uma experiência consistente em diferentes dispositivos e tamanhos de tela, variando de *smartphones* até monitores de alta resolução. Esses *layouts* dinâmicos se ajustam de maneira inteligente, assegurando que o conteúdo seja apresentado de forma harmoniosa, independentemente do dispositivo utilizado, e proporcionando aos usuários uma navegação fluida e acessível em qualquer dispositivo.

Nesta aula, vamos mergulhar nos conceitos básicos que formam a espinha dorsal dos *layouts* flexíveis, com foco na propriedade CSS **display: flex**.

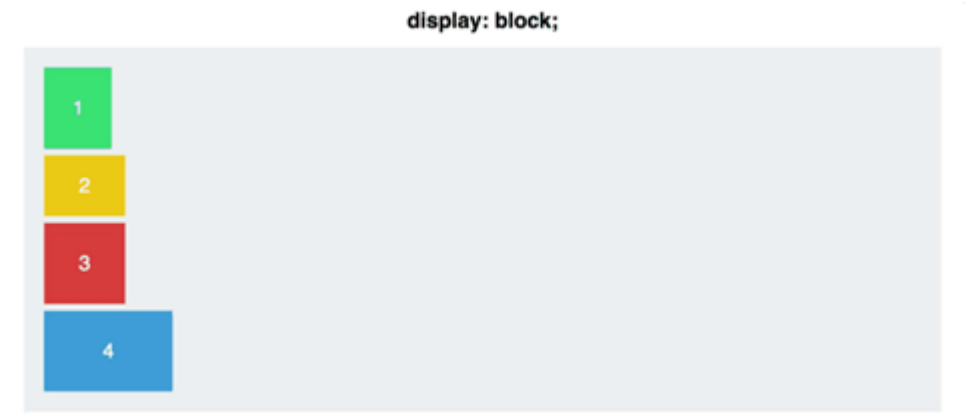


Disponível em: <freepik-http://tinyurl.com/k3a8tzd5>. Acesso em: 30 jan. 2024.

A principal vantagem dos *layouts* com **display: flex** é a facilidade com que se adaptam a diferentes tamanhos de tela. Agora, você deve estar se perguntando como podemos colocar tudo isso em prática. A resposta é simples: com o Flexbox.

O Flexbox, ou Flexible Box, é uma técnica de desenvolvimento *web* que facilita o *design* de *layouts* mais flexíveis e responsivos. Ele permite alinhar, distribuir e espaçar os elementos de forma mais eficiente, garantindo que eles se ajustem automaticamente a diferentes tamanhos de tela e dispositivos. Dessa forma, é possível criar uma maneira eficaz de distribuir o espaço disponível entre os itens flexíveis.

Imagine que você está organizando elementos em uma linha ou coluna, como itens em uma prateleira. No desenvolvimento *web*, o Flexbox proporciona essa organização, possibilitando criar *layouts* em que os elementos ocupam proporções específicas do espaço, promovendo uma distribuição equitativa e proporcional.



Disponível em: <http://tinyurl.com/2ww22csm>. Acesso em 30 jan. 2024.

A seguir, vejamos exemplos de aplicação de **display: flex**.

- HTML:

```
<body>
    <div class="container">
        <div class="box"><p>1</p></div>
        <div class="box"><p>2</p></div>
        <div class="box"><p>3</p></div>
        <div class="box"><p>4</p></div>
    </div>
</body>
</html>
```

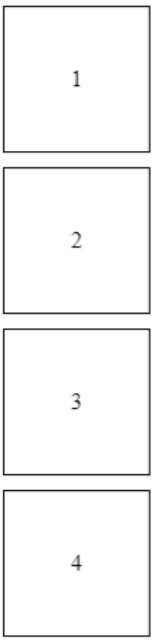
Fonte: elaborado pelo autor (2024).

- CSS:

```css
<style>
    .container {
        display: flex;
    }

    .box {
        width: 100px;
        height: 100px;
        border: 1px solid #000;
        margin: 10px;
        display: flex;
        justify-content: center;
        align-items: center;
    }
</style>
```

Fonte: elaborado pelo autor (2024).

Nos exemplos, está sendo criada uma regra para a classe **.container**, que é aplicada onde há quatro **div** filhas. A propriedade **display: flex** é usada para definir que os elementos filhos diretos do *container* fiquem com os itens flexíveis, permitindo que os filhos se organizem de acordo com as propriedades do modelo de *layout* flexível. Dessa forma, o resultado da nossa página é o seguinte:



Fonte: elaborado pelo autor (2024).

💻🚀 #CodingChallenge

Coloque em prática o que aprendemos até aqui! Acesse o CodePen para experimentar e testar os códigos que exploramos. Modifique o tamanho das caixas, teste mudar as cores de cada box e brinque com o código para personalizar a página do seu jeito. Essa é a oportunidade perfeita para consolidar o conhecimento e aprimorar as suas habilidades de codificação.

6.2.1 Propriedades do display: flex

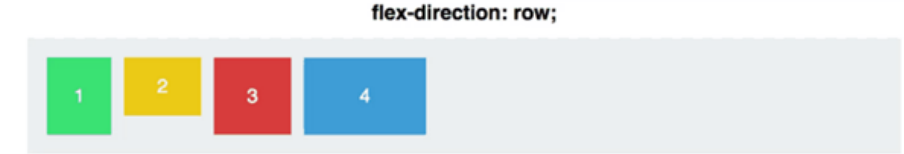
A discussão sobre as propriedades que afetam o **display: flex** é fundamental para entender como o Flexbox opera e como podemos controlar o *layout* de forma eficaz. Vamos analisar três propriedades principais – **flex-direction**, **justify-content** e **align-items** – para, então, explorar como elas impactam o *container* flexível.

Flex-direction

A propriedade **flex-direction** determina a direção principal dos itens flexíveis dentro de um *container*. Em outras palavras, ela determina se os itens devem ser dispostos horizontalmente (da esquerda para a direita) ou verticalmente (de cima para baixo) dentro do *container*. As suas opções incluem:

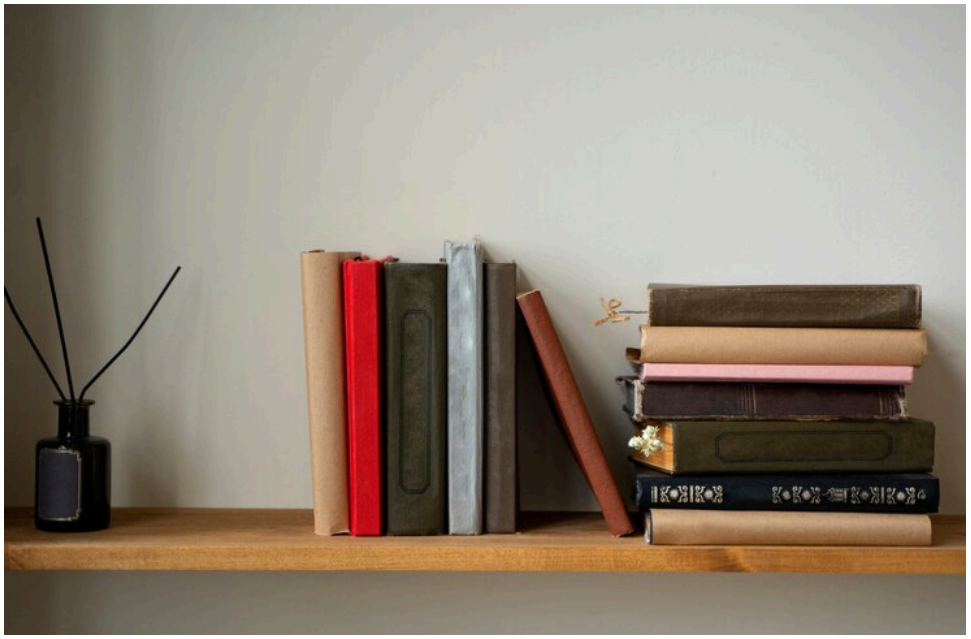
- **row** – itens são posicionados na mesma linha (da esquerda para a direita);
- **column** – itens são posicionados em uma coluna (de cima para baixo);
- **row-reverse** – funciona da mesma forma que a opção **row**, mas na ordem inversa, ou seja, dispondo os itens na mesma linha, porém da direita para esquerda;
- **column-reverse** – funciona da mesma forma que a opção **column**, mas na ordem inversa, isto é, dispondo os itens de baixo para cima.

Veja só o exemplo abaixo. Nele, está sendo aplicada a função **row**:



Disponível em: <http://tinyurl.com/2ww22csm>. Acesso em 30 jan. 2024.

Para que você entenda melhor a propriedade **flex-direction** e as suas opções, imagine uma prateleira de livros. Se você organizasse os livros da esquerda para a direita, isso seria um exemplo de **flex-direction: row** (direção horizontal). Se, no entanto, sua escolha fosse organizar os títulos de cima para baixo, isso seria um exemplo de **flex-direction: column** (direção vertical).



Disponível em: <freepik-http://tinyurl.com/566caxj7>. Acesso em 30 jan. 2024.

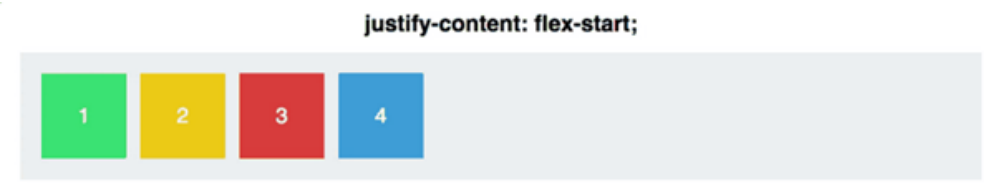
Viu como é simples? Agora, vamos conhecer a próxima propriedade!

Justify-content

A propriedade **justify-content** controla o alinhamento dos itens flexíveis ao longo do eixo principal (o eixo definido por **flex-direction**). As suas opções incluem:

- **flex-start** – os itens são alinhados no início do *container*;
- **flex-end** – os itens são alinhados no final do *container*;
- **center** – os itens são centralizados no *container*;
- **space-between** – distribui os itens uniformemente, com espaçamento **entre** eles;
- **space-around** – distribui os itens uniformemente, com espaçamento **ao redor** deles.

Veja, na imagem abaixo, um exemplo que ilustra como a propriedade **justify-content** funciona.



Disponível em: <http://tinyurl.com/2ww22csm>. Acesso em 30 jan. 2024.

Align-items

A propriedade **align-items** controla o alinhamento dos itens flexíveis ao longo do eixo transversal (o eixo perpendicular ao eixo principal). As suas opções incluem:

- **flex-start** – os itens são alinhados no início do *container*;
- **flex-end** – os itens são alinhados no final do *container*;
- **center** – os itens são centralizados no *container*;
- **baseline** – os itens são alinhados pela linha de base. Essa opção é muito útil quando há diferentes tamanhos de texto;
- **stretch** – os itens são esticados para preencher o *container*.

Veja o exemplo abaixo:

Disponível em: <http://tinyurl.com/2ww22csm>. Acesso em 30 jan. 2024.

Agora, analise os códigos CSS abaixo e tente identificar onde as propriedades que acabamos de estudar estão localizadas no código. Para aumentar o nível de desafio, acesse o CodePen e escreva os códigos para ver o que aparece na página.

```
    <style>
        .container {
            display: flex;
            border: 2px solid #333;
            margin: 20px;
            padding: 10px;
        }

        .box {
            width: 50px;
            height: 50px;
            background-color: lightblue;
            margin: 5px;
            display: flex;
            justify-content: center;
            align-items: center;
        }
    </style>
```

Fonte: elaborado pelo autor (2024).

```
<h3>flex-direction</h3>
<div class="container" style="flex-direction: row;">
    <div class="box">1</div>
    <div class="box">2</div>
    <div class="box">3</div>
    <div class="box">4</div>
</div>
```

Fonte: elaborado pelo autor (2024).

```
<h3>justify-content</h3>
<div class="container" style="justify-content: flex-end;">
    <div class="box">1</div>
    <div class="box">2</div>
    <div class="box">3</div>
    <div class="box">4</div>
</div>
```

Fonte: elaborado pelo autor (2024).

```
<h3>align-items</h3>
<div class="container" style="align-items: flex-start;">
    <div class="box">1</div>
    <div class="box">2</div>
    <div class="box">3</div>
    <div class="box">4</div>
</div>
```

Fonte: elaborado pelo autor (2024).

6.2.2 Elaborando um *case*

Layout flexível: construindo uma galeria de fotos responsiva

Objetivo: criar um *layout* flexível para uma galeria de fotos responsiva, usando as propriedades **display: flex, flex-direction, justify-content** e **align-items**. A galeria deve se ajustar de maneira harmoniosa em diferentes tamanhos de tela, proporcionando uma experiência visualmente agradável.

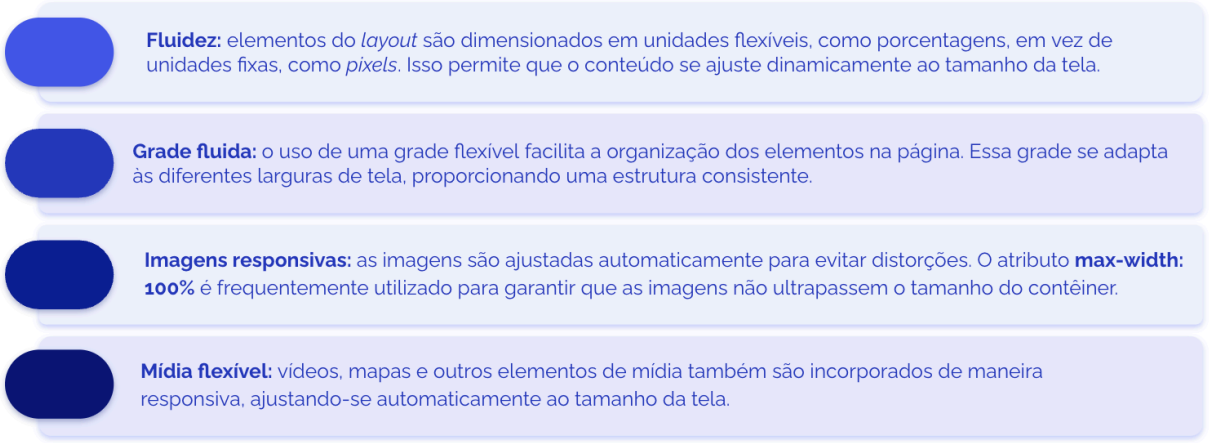
Requisitos:

1. Utilizar HTML e CSS para construir a estrutura e o estilo da galeria;
2. A galeria deve conter, pelo menos, seis fotos fictícias;
3. Implementar um *layout* flexível que se adapte a diferentes dispositivos;
4. Experimentar diferentes valores para as propriedades **flex-direction**, **justify-content** e **align-items**, visando alcançar um *design* visualmente equilibrado.

6.3 *Design* responsivo com @media queries

Conceitos básicos de *design* responsivo

O *design* responsivo é uma abordagem de *design* que busca proporcionar uma experiência consistente e agradável para o usuário em uma variedade de dispositivos e tamanhos de tela. Em vez de criar *layouts* fixos para diferentes tipos de dispositivos, o *design* responsivo utiliza técnicas flexíveis para adaptar o conteúdo e o *layout*, garantindo uma visualização otimizada em qualquer tela.



Fonte: elaborado pelo autor (2024).

Entender os conceitos básicos de *design* responsivo fará com que você esteja melhor preparado para criar *sites* e aplicações *web* que atendam às demandas de acessibilidade e usabilidade, facilitando a construção de projetos mais dinâmicos e eficientes.

Breakpoints e o papel das @medias queries em responsividade

As **@media queries** são parte essencial do *design* responsivo. Elas permitem aplicar estilos específicos com base nas características da mídia, como largura da tela, orientação do dispositivo, resolução, entre outros. Isso significa que podemos personalizar o *design* com base nas necessidades de diferentes dispositivos.

Para entender mais desse assunto, vamos conhecer algumas das funções das **@media queries.**

Consulta de mídia	Sintaxe básica	Pontos de quebra (*breakpoints*)
É uma **@media query** que é uma regra condicional. Ela verifica se determinadas condições de mídia são atendidas.	É uma **@media query** de sintaxe simples. Por exemplo, **@media (min-width: 768px)** aplicará estilos apenas quando a largura da tela for igual ou superior a 768 *pixels*.	Os pontos de quebra são valores específicos de largura de tela em que o *layout* ou os estilos podem ser ajustados para melhor atender às características da mídia.

Fonte: elaborado pelo autor (2024).

Veja um exemplo da aplicação dos *breakpoints* no *design* responsivo:

Com *breakpoints* Sem *breakpoints*

Disponível em: <http://tinyurl.com/37wdpztb>. Acesso em 30 jan. 2024.

Agora, para testar os nossos conhecimentos, vamos realizar a criação de um *layout* básico responsivo, aplicando esses conceitos na prática. Acesse o CodePen e vamos começar com um exemplo de código HTML e CSS.

- HTML:

```
<body>
    <div class="container">
        <header>
            <h1>Meu Site Responsivo</h1>
        </header>
        <main>
            <p>Bem-vindo ao meu site!</p>
        </main>
        <footer>
            <p>&copy; 2024 Meu Site Responsivo</p>
        </footer>
    </div>
</body>
```

Fonte: elaborado pelo autor (2024).

- CSS:

```css
body {
    margin: 0;
    font-family: 'Arial', sans-serif;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
}

header, main, footer {
    text-align: center;
    margin-bottom: 20px;
}

@media (min-width: 768px) {
    header, main, footer {
        text-align: left;
    }
}
```

Fonte: elaborado pelo autor (2024).

🤓 Entendendo melhor

Ao analisar o código, percebemos que o **@media (min-width: 768px)** está sendo aplicado apenas quando a largura da tela é igual ou superior a 768 *pixels*. Também podemos ver a seguinte linha: **header, main, footer { text-align: left; }**:, o que quer dizer que, dentro dessa consulta de mídia, o texto dentro dos elementos **<header>**, **<main>** e **<footer>** é alinhado à esquerda, substituindo a configuração de **text-align: center;** anterior. Isso é feito para telas maiores, nas quais um alinhamento à esquerda pode ser mais apropriado.

E aí, o que você está achando da nossa jornada *dev* até aqui? Se estiver com o cérebro fervilhando depois de ler tantas linhas de código, faça uma pausa e volte já. Mas, se o seu cérebro estiver fervilhando de empolgação, vamos continuar com a nossa aventura no mundo do desenvolvimento *web*!

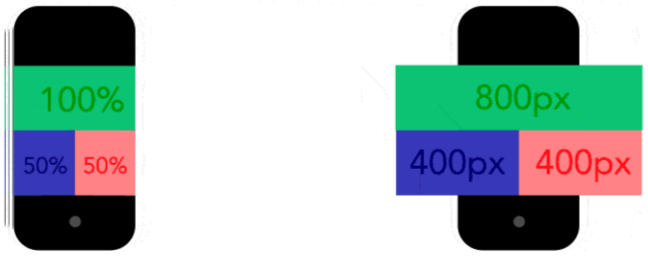
Trabalhando com unidades relativas

Agora, vamos explorar um conceito fundamental que influencia diretamente o *layout* e a adaptação de páginas *on-line*: as **unidades relativas**. Elas são essenciais para criar *designs* flexíveis e responsivos, permitindo que os elementos se ajustem dinamicamente com base em fatores como o tamanho da tela do dispositivo. Esse tópico é crucial para a construção de *layouts web* modernos e adaptáveis.

A tela pode ser um *desktop*, um dispositivo móvel ou qualquer coisa intermediária.

Com a diversidade de tamanhos de tela, a densidade de *pixels* pode variar, então, torna-se necessário que as unidades sejam flexíveis e funcionem em qualquer lugar. É aí que entram as unidades relativas, como a porcentagem, por exemplo.

Imagine que um cliente pediu para você fazer um *box* na página do *site* dele com 50% de largura. Isso significa que esse box sempre ocupará metade do tamanho da tela. Acompanhe como essa situação aconteceria na prática, vendo o exemplo abaixo.



Unidades relativas Unidades estáticas

Disponível em: <http://tinyurl.com/37wdpztb>. Acesso em 30 jan. 2024.

6.3.1 Elaborando um case

Agora, vamos ao desafio. Depois de aprender tudo sobre *design* responsivo, você deve criar um *layout* simples e responsivo para um *site* pessoal, usando **@media queries** e unidades relativas. Aqui, o foco é praticar a adaptação do *design* a diferentes tamanhos de tela, garantindo uma experiência consistente em dispositivos variados.

Requisitos

- Utilizar HTML e CSS para construir a estrutura e o estilo do *site*;
- Implementar, pelo menos, três **@media queries** para ajustar o *layout* em diferentes *breakpoints*;
- Utilizar unidades relativas, como porcentagens, para tornar o *design* flexível.

Desafios adicionais

- Implementar. pelo menos. três **@media queries** para ajustar o *layout* em diferentes *breakpoints* (móvel, *tablet*, *desktop*);
- Utilizar unidades relativas, como porcentagens, para definir larguras, alturas e margens, tornando o *layout* flexível.

6.3.2 @media queries para componentes responsivos: alteração de estilos com base na orientação do dispositivo

Ao criar *layouts* responsivos, é crucial considerar não apenas a largura da tela, mas, também, a orientação do dispositivo (paisagem ou retrato). As **@media queries** podem ser usadas para ajustar estilos com base nesse fator. Seguindo com a nossa aula sobre técnicas de *layout* e posicionamento, vamos começar analisando um exemplo de consulta de mídia para dispositivos em modo paisagem.

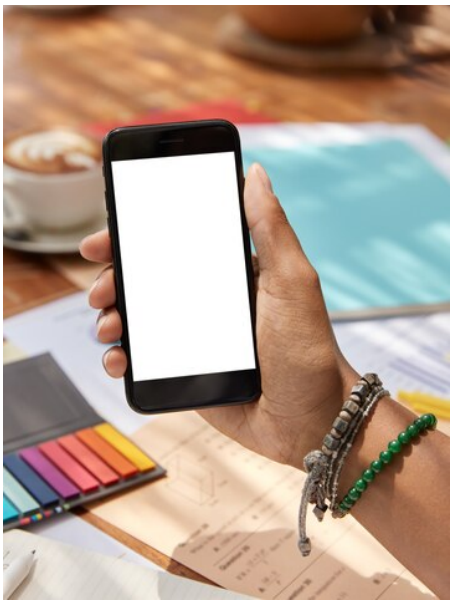
```
@media screen and (orientation: landscape) {
    /* Estilos específicos para dispositivos em modo paisagem */
    body {
        background-color: #a0d8f1;
    }

    /* Outras regras de estilo específicas... */
}
```

Fonte: elaborado pelo autor (2024).

Quando a orientação da tela é considerada em **modo paisagem**, significa que a largura da tela é maior que a altura. Isso ocorre, geralmente, quando o dispositivo está sendo mantido de lado, como acontece com muitos *tablets* e *laptops*, pois, frequentemente, o usuário utiliza o dispositivo posicionado na horizontal. Esse recurso serve muito para a apresentação de imagens panorâmicas, em que a exibição horizontal precisa ser mais eficiente.

Disponível em: <freepik-http://tinyurl.com/2kcnu5xj>. Acesso em: 30 jan. 2024.

A orientação de tela em **modo retrato** significa que a altura da tela é maior que a largura. Isso ocorre quando o dispositivo é mantido em pé, como os celulares.



Disponível em: <freepik-http://tinyurl.com/yucy7wyf>. Acesso em: 30 jan. 2024.

Nessas condições de uso do dispositivo, é importante que o desenvolvedor trabalhe com o *design* vertical e imagens verticais. Além disso, nesse contexto são priorizados os conteúdos de visualização em telas altas.

Vamos ver como isso deve ser escrito no código. Na imagem abaixo, o código mostra uma consulta de mídia para dispositivos em modo retrato:

```
@media screen and (orientation: portrait) {
    /* Estilos específicos para dispositivos em modo retrato */
    body {
        background-color: #f4e2d8;
    }

    /* Outras regras de estilo específicas... */
}
```

Fonte: elaborado pelo autor (2024)

Observação: em algumas situações, é possível combinar orientações em uma única consulta de mídia para estilos mais específicos em casos particulares. Nesse caso, os estilos específicos para dispositivos em modo retrato possuem a estilização de largura menor que 600px.

```
@media screen and (orientation: portrait) and (max-width: 600px) {
    /* Estilos específicos para dispositivos em modo retrato com largura menor que 600px */
}
```

Fonte: elaborado pelo autor (2024)

A importância da acessibilidade em *designs* responsivos

A criação de *layouts* flexíveis e responsivos não apenas se adapta a diferentes dispositivos, mas, também, desempenha papel fundamental na garantia de uma experiência inclusiva para todos os usuários, independentemente de suas necessidades específicas. Vamos entender melhor como o *design* responsivo pode ser uma ferramenta poderosa para tornar a *web* acessível a todos.

Disponível em: <freepik-http://tinyurl.com/4r8v393e>. Acesso em: 30 jan. 2024.

Antes de iniciarmos essa parte da aula, é importante entender as vantagens e a importância da existência de projetos que apliquem a acessibilidade em *designs* responsivos:

1. **Equidade e inclusão:** a acessibilidade promove a equidade ao garantir que todas as pessoas, incluindo aquelas com deficiências, tenham acesso igual às informações e funcionalidades *on-line*.
2. **Ampla variedade de dispositivos:** *designs* responsivos são essenciais, considerando a diversidade de dispositivos usados para acessar a *web*. A acessibilidade garante que os usuários com deficiência possam interagir de forma eficiente em qualquer dispositivo.
3. **Leis e normas:** muitas regiões e organizações têm leis e normas que exigem a conformidade com padrões de acessibilidade na *web* e o *design* responsivo deve ser inclusivo para atender a todos esses requisitos.
4. **Ampliação do público-alvo:** ao tornar um *site* acessível, expande-se o seu público-alvo, incluindo pessoas com deficiências visuais, auditivas, motoras ou cognitivas.
5. **Melhoria na experiência do usuário:** recursos acessíveis, como descrições de texto em imagens ou navegação por teclado, melhoram a experiência não apenas para os usuários com algum tipo de deficiência, mas para todos.

Fonte: elaborado pelo autor (2024).

Um bom exemplo de aplicação de recurso de acessibilidade muito presente nas páginas *web* é a **alteração de cores do *site***. Usando um contraste adequado, utilizamos o **@media queries** para ajustar as cores em elementos de texto e fundo para garantir a legibilidade para usuários com deficiência visual. Veja um exemplo de código com definições de acessibilidade:

```
@media (prefers-contrast: high) {
    body {
        background-color: #fff;
        color: #000;
    }
}
```

Fonte: elaborado pelo autor (2024)

🤓 Analisando o código

- **@media (prefers-contrast: high):** é a declaração da consulta de mídia. Ela especifica que as regras de estilo dentro das chaves {} serão aplicadas apenas se a preferência de contraste do usuário estiver configurada como "**high**" (alto contraste);
- **body:** refere-se ao elemento **<body>** do HTML, que é a parte principal da página;
- **background-color: #fff:** define a cor de fundo do elemento **<body>**, como branco (#fff) quando a preferência de contraste do usuário for configurada como "**high**";

- **color: #000:** define a cor do texto do elemento **<body>**, como preto (#000) quando a preferência de contraste do usuário for configurada como "**high**".

Esse código é um exemplo de como é possível ajustar o esquema de cores de um *site* para garantir um alto contraste quando o usuário tiver essa preferência configurada em seu dispositivo. Isso é uma prática de *design* inclusivo, proporcionando uma experiência mais acessível para usuários que precisam ou preferem um maior contraste entre as cores em uma página.

ATIVIDADE DE FIXAÇÃO

1. Qual é a diferença entre **position: relative** e **position: absolute** em CSS?
2. Como o posicionamento relativo (**position: relative**) afeta o elemento em relação ao seu próprio fluxo no documento?
3. Como você pode centralizar um elemento usando **position: absolute**?
4. Quais são as principais propriedades do modelo de *layout* flexível (**display: flex**)?
5. Como você alinha itens horizontalmente usando **display: flex**?
6. Descreva como é possível alinhar os itens verticalmente com o **display: flex**.
7. Explique como ocorre a aplicação do eixo principal e do eixo transversal em um *layout* flexível.
8. Como é possível inverter a ordem dos itens, usando **flex-direction**?
9. O que são **@media queries** e qual é o seu papel no *design* responsivo?
10. Como você pode aplicar estilos diferentes com base na largura da tela usando **@media queries**?
11. Explique o motivo de ser importante considerar o *design* responsivo ao desenvolver um *site*.

DESAFIO PRÁTICO

Design responsivo com @media queries

Desafio: modernizando a página inicial.

Descrição

A empresa ABC optou por atualizar a sua página inicial para torná-la mais atraente e responsiva. A partir disso, foi decidido que seria mais adequado incorporar elementos visuais modernos, tais como navegação flexível, seções dinâmicas e um rodapé informativo. O departamento de *marketing* reconhece que, para alcançar esse objetivo, será necessário utilizar técnicas como **position: absolute** e **relative**, **display: flex** e **@media queries**, visando garantir um *design* responsivo.

Objetivos

- Implementar um menu de navegação horizontal, explorando as técnicas de **display: flex**;
- Adicionar uma seção de destaque na página inicial, utilizando **position: absolute** e **position: relative** para um posicionamento preciso;
- Incorporar um rodapé responsivo com informações úteis e ajustes, utilizando **@media queries**.

Orientações

- Inicie o projeto criando um arquivo base e seguindo o padrão de desenvolvimento estabelecido;
- Certifique-se de que a página criada pelo grupo seja responsiva, adaptando-se a diferentes tamanhos de tela e empregando **@media queries** para ajustes específicos.

Nesta aula, aprendemos sobre três elementos fundamentais no CSS3 para criar *layouts* flexíveis e responsivos. Primeiro, exploramos diferentes técnicas de posicionamento, discutindo as propriedades ***relative*** e ***absolute***. Com isso, conseguimos entender que isso é crucial para dispor os elementos na página de maneira precisa.

***Layouts* flexíveis**	Ao transformar *containers* em ambientes flexíveis, podemos organizar dinamicamente elementos, garantindo uma adaptação eficiente a diferentes dispositivos. A flexibilidade desses *layouts* é vital para o *design* responsivo, permitindo que as páginas se ajustem de forma harmoniosa a variados tamanhos de tela.
@media queries	Integradas para criar *designs* responsivos, ajustando automaticamente o *layout* com base nas características do dispositivo. Essas técnicas combinadas capacitam os desenvolvedores a criar páginas *web* coesas e otimizadas para diversas experiências de usuário.

CAPÍTULO 07

Arquitetura *web* moderna com CSS3

O que esperar deste capítulo:

- Criando menus de navegação horizontal e vertical;
- Criação de um *layout* responsivo simples.

Layout e posicionamento com CSS3

Vamos turbinar as nossas habilidades de *design web*?

Chegou o momento de desbravar o mundo de ***layout* e posicionamento com CSS3**. Nessa aventura, vamos transformar os seus *sites* com um toque de estilo que vai fazer todo mundo ficar surpreso.

Primeiro, vamos dominar a arte de **criar menus**, tanto na horizontal quanto na vertical. Afinal, quem não quer um menu que seja não só funcional, mas, também, um espetáculo visual? E não é só isso: vamos aprender a fazer *layouts* que se ajustem como mágica a qualquer tamanho de tela.

Durante essa jornada, vamos explorar alguns conceitos, como *grids*, Flexbox e *media queries*, mas não se preocupe, porque não é nada complicado. São só ferramentas poderosas que vão transformar o seu *site*. Então, se prepare, pois estamos prestes a mergulhar fundo no universo do *web design* com o incrível CSS3! 🚀

Desvendando o mundo do *layout web*: *grid* e Flexbox em ação

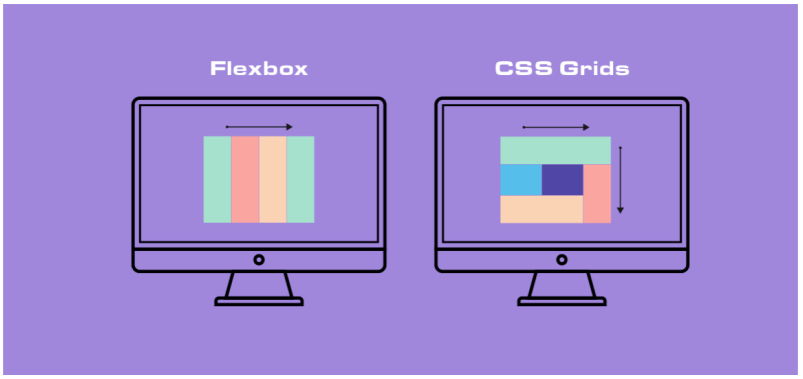
Flexbox e o *grid layout* são duas tecnologias fundamentais para o *design* de *layout* em páginas da *web*, oferecendo abordagens distintas para organizar elementos de maneira eficiente e responsiva. Para compreender a diferença entre eles, observe o esquema abaixo:

Comparando o Flexbox e o *grid layout*

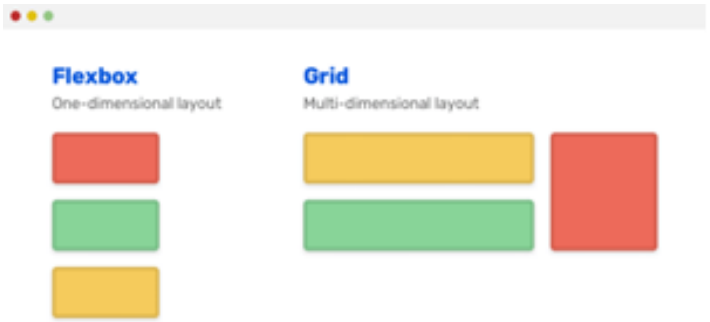
Flexbox ou *box* flexível	*Grid layout*
• É um modelo de *layout* **unidimensional,** projetado para distribuir espaços ao longo de um eixo, seja ele horizontal ou vertical; • Permite organizar elementos em uma única linha ou coluna, ajustando automaticamente o tamanho e a posição dos itens com base no espaço disponível; • Adequado para quem busca estruturas mais **adaptáveis**.	• É uma tecnologia que possibilita a criação de *layouts* **bidimensionais;** • Permite que os elementos sejam organizados em linhas e colunas, proporcionando uma estrutura clara e **flexível;** • Adequado para estruturas de página mais **complexas**.

Fonte: elaborado pelo autor (2024).

Assim, cada uma dessas tecnologias apresenta um *design* próprio, como é possível ver nas imagens a seguir.



Disponível em: <http://tinyurl.com/yztyeama>. Acesso em: 31 jan. 2024.



Disponível em: <http://tinyurl.com/y3z2aj59>. Acesso em: 31 jan. 2024.

Muitas vezes, eles são usados em conjunto para aproveitar o melhor dos dois mundos, dependendo das necessidades de *layout* de uma página da *web*.

Na prática, como isso funciona?

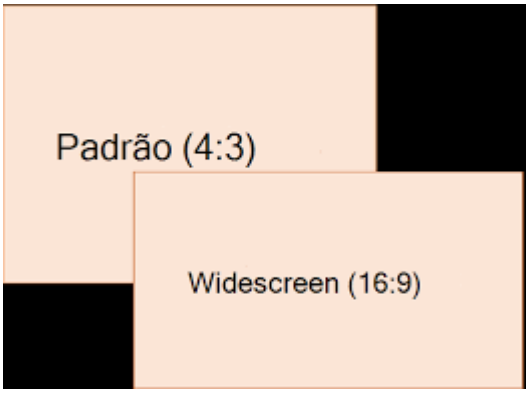
Grids e Flexbox são muito úteis para criar barras de navegação horizontais, verticais e, até mesmo, estruturar páginas mais complexas com um cabeçalho, uma seção principal e um rodapé.

Menus navegacionais e efeitos de transição em CSS3

A navegação horizontal é a preferida para quem quer criar uma apresentação **mais ampla e fluida**. Ela se destaca em *layouts widescreen*, proporcionando uma sensação de continuidade e facilitando a leitura.

Você sabe o que é o *widescreen*?

Ele nada mais é do que um *design* de página que é mais largo do que o padrão comum. É como ter uma **tela panorâmica**, o que pode ser ótimo para mostrar mais conteúdo lado a lado, especialmente em monitores mais largos.



Disponível em: <http://tinyurl.com/2ubkyvz4>. Acesso em 31 jan. 2024.

Assim, entendendo como criar menus horizontais, você pode oferecer aos usuários uma experiência de navegação intuitiva, melhorando a acessibilidade e a compreensão do conteúdo.

Pense na seguinte situação...

Quando nosso guarda-roupa está bagunçado é bem mais difícil achar a roupa que queremos, não é?

Da mesma forma, se a disposição visual de um *site* é desorganizada, dificilmente o usuário vai conseguir acessá-lo de maneira fluida e intuitiva.
Por isso, a diagramação eficaz dos elementos na interface de usuário desempenha um papel crucial na experiência do usuário. Uma disposição organizada e atraente **facilita a navegação, reduz a carga cognitiva e torna a interação mais agradável.**

Fonte: elaborado pelo autor (2024).

Então, nossa missão é explorar como a utilização de *grids* e do Flexbox pode ser empregada para criar *layouts* flexíveis, garantindo que a navegação horizontal não apenas sirva a um propósito funcional, mas, também, contribua para uma experiência visualmente agradável.

DESAFIO PRÁTICO

Criação de um layout responsivo simples

Desafio: criação de um *layout* responsivo para um *blog* de viagens.

Descrição

Você foi contratado para desenvolver o *layout* responsivo de um *blog* de viagens. O cliente busca proporcionar uma experiência agradável aos usuários em dispositivos móveis, *tablets* e computadores.

O *blog* de viagens contém diversas seções, como postagens recentes, categorias de destinos e uma área para fotos destacadas. O objetivo é assegurar que o conteúdo seja apresentado de maneira legível e atrativa em diferentes tamanhos de tela.

Objetivos

- Criar um *design* responsivo que se adapte a dispositivos móveis, *tablets* e *desktops*;
- Implementar uma navegação intuitiva para facilitar o acesso às diferentes seções do *blog*;
- Garantir que as imagens sejam carregadas de forma eficiente, considerando o desempenho em dispositivos móveis;
- Utilizar técnicas de CSS para garantir uma experiência de usuário consistente em diferentes resoluções.

Orientações

- Empregue *media queries* para ajustar o estilo de acordo com o tamanho da tela;
- Considere a usabilidade, dando prioridade ao conteúdo mais relevante para dispositivos móveis;
- Realize testes de compatibilidade em diversos navegadores e dispositivos.

Criando um menu de navegação horizontal

Para compreender melhor, reproduza, depois da leitura, o exemplo abaixo:

index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <title>Navegação Horizontal</title>
</head>
<body>

  <nav class="horizontal-menu">
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">Sobre</a></li>
      <li><a href="#">Serviços</a></li>
      <li><a href="#">Contato</a></li>
    </ul>
  </nav>
</body>
</html>
```

Fonte: elaborado pelo autor (2024).

CSS explicativo

```css
/* Define a formatação padrão do corpo do documento */
body {
    font-family: Arial, sans-serif;
}

/* Estiliza o contêiner da navegação horizontal */
.horizontal-menu {
    background-color: #333;
}

/* Remove o espaçamento padrão e a formatação de listas */
.horizontal-menu ul {
    padding: 0;
    margin: 0;
    list-style: none;
    overflow: hidden;
}

/* Configuração para os itens da lista flutuarem à esquerda, criando o layout horizontal */
.horizontal-menu li {
    float: left;
}

/* Estiliza os links na navegação */
.horizontal-menu a {
    display: block; /* Converte o link em um bloco para ocupar a largura total do contêiner */
    padding: 14px 16px; /* Adiciona espaçamento interno */
    color: white; /* Cor do texto */
    text-align: center; /* Centraliza o texto */
    text-decoration: none; /* Remove a sublinhado padrão dos links */
}

/* Estiliza os links quando o cursor está sobre eles */
.horizontal-menu a:hover {
    background-color: #ddd; /* Cor de fundo ao passar o cursor sobre o link */
    color: black; /* Cor do texto ao passar o cursor sobre o link */
}
```

Fonte: elaborado pelo autor (2024).

Seguindo as orientações presentes nas imagens, o resultado será:

Fonte: elaborado pelo autor (2024).

Criando um menu de navegação vertical

Para compreender melhor, reproduza, após a leitura, o exemplo abaixo:

```html
<div class="vertical-menu">
  <ul>
    <li><a href="#home">Home</a></li>
    <li><a href="#about">About</a></li>
    <li><a href="#services">Services</a></li>
    <li><a href="#contact">Contact</a></li>
  </ul>
</div>
```

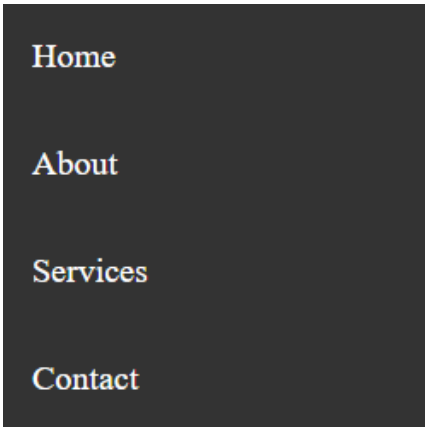
Fonte: elaborado pelo autor (2024).

CSS explicativo

```css
/* Estiliza o contêiner da navegação vertical */
.vertical-menu {
    width: 200px; /* Largura do menu */
}

/* Remove o espaçamento padrão e a formatação de listas */
.vertical-menu ul {
  padding: 0;
  margin: 0;
  list-style: none;
}

/* Estiliza os itens da lista */
.vertical-menu li {
  background-color: #333; /* Cor de fundo do item */
  margin: 0; /* Remove margem padrão */
}

/* Estiliza os links na navegação vertical */
.vertical-menu a {
  display: block;
  padding: 16px; /* Espaçamento interno */
  color: white; /* Cor do texto */
  text-decoration: none;
}

/* Estiliza os links quando o cursor está sobre eles */
.vertical-menu a:hover {
  background-color: #ddd; /* Cor de fundo ao passar o cursor sobre o link */
  color: black; /* Cor do texto ao passar o cursor sobre o link */
}
```

Fonte: elaborado pelo autor (2024).

Seguindo as orientações presentes nas imagens, o resultado será:



Fonte: elaborado pelo autor (2024).

Conceito de *layout* responsivo

O *layout* responsivo é uma abordagem essencial no *web design* moderno que visa criar interfaces que se ajustem, de maneira eficiente, a diferentes dispositivos e tamanhos de tela.

Você já passou pela situação de abrir um *site* no celular e ele parecer estar cortado ou apresentar ícones em formatação esquisita? Para que isso não ocorra, é importante saber utilizar um *layout* responsivo, pois a sua ideia fundamental é proporcionar **uma boa experiência ao usuário em dispositivos variados, adaptando o *site* aos diferentes tamanhos de tela.** Isso é alcançado através de técnicas flexíveis de *design* e programação que respondem dinamicamente às características específicas de cada tela.

Para implementar um *layout* responsivo, utiliza-se uma combinação de práticas, tais quais:

- *design* fluido;
- *media queries*, uma ferramenta de ajuste de tela;
- adaptação de imagens e fontes.

Unidades flexíveis, como as porcentagens, garantem que os elementos se ajustem proporcionalmente ao tamanho da tela, enquanto as ***media queries*** permitem utilizar estilos específicos com base em características da tela, como largura e altura. Imagens flexíveis e fontes adaptáveis asseguram que o conteúdo visual e textual seja exibido de maneira adequada.

Abaixo, observe alguns comandos práticos que podemos utilizar com *media queries:*

```
style.css > ...
/* Estilos para telas menores que 600 pixels */
@media only screen and (max-width: 600px) {
    body {
        font-size: 16px; /* Ajusta o tamanho da fonte para telas menores */
    }
    .menu {
        display: none; /* Oculta o menu em telas menores para economizar espaço */
    }
}
```

Fonte: elaborado pelo autor (2024).

Testes e impacto positivo

Testar o *design* responsivo em diversos dispositivos é crucial para garantir uma experiência consistente. Além de proporcionar uma navegação mais amigável em dispositivos móveis, o *design* responsivo contribui para melhores classificações em mecanismos de busca, aprimorando a visibilidade *on-line*. Ao priorizar a adaptação a diferentes contextos, o *layout* responsivo não apenas reflete o compromisso com uma experiência do usuário acessível e eficaz, mas, também, contribui para uma melhor classificação nos motores de busca, já que o Google favorece *sites* otimizados para dispositivos móveis em seus resultados de pesquisa.

Algumas formas de testar o *design* responsivo são:

Utilizar dispositivos físicos	Permite testar seu *site* em dispositivos reais, como *smartphones* ou *tablets*. Assim, você saberá exatamente como será a experiência do usuário ao acessá-lo.
Ferramentas *on-line*	Ferramentas como **BrowserStack** ou **CrossBrowserTesting** permitem testar seu *site* em vários dispositivos e navegadores, sem precisar possuir todos esses dispositivos físicos.
Testes manuais	Para testar manualmente, você pode, simplesmente, redimensionar a janela do navegador durante o desenvolvimento do *site*. Isso ajuda a identificar problemas de *layout* em diferentes tamanhos de tela.

Fonte: elaborado pelo autor (2024).

DESAFIO PRÁTICO

Layouts flexíveis com display: flex

Desafio: a revolução da experiência do usuário.

Descrição

Suponha que você faça parte da equipe de desenvolvimento da TechInnovate, uma empresa pioneira em tecnologia que está prestes a lançar uma aplicação revolucionária de gerenciamento de tarefas. A sua missão é apresentar duas opções de layouts de menus para um novo cliente interessado nesta aplicação inovadora. Nós estamos falando especificamente de menus direcionados ao mercado imobiliário com funcionalidades de gerenciamento de tarefas.

Objetivos

- Criar dois modelos de menus, um vertical e um horizontal, para que o cliente faça a sua escolha;
- Definir os elementos que estarão presentes nos dois menus. Lembre-se de trabalhar com as mesmas informações.

Orientações

- Em grupo, entrem em consenso sobre os elementos dos menus;
- Criem os modelos de menu;
- Expliquem o motivo das escolhas e avaliem se elas correspondem ao mercado imobiliário.

DESAFIO PRÁTICO II

Criação de um layout responsivo simples

Desafio: desenvolvimento de um layout responsivo para uma loja virtual.

Descrição

Uma loja virtual necessita de um *layout* que garanta uma experiência de compra fluída em dispositivos diversos, abrangendo desde *smartphones* até *desktops*. O principal objetivo é garantir acessibilidade e usabilidade em todas as etapas do processo de compra.

A loja possui categorias de produtos, páginas detalhadas dos produtos, carrinho de compras e páginas de finalização de compra. O cliente deseja que os usuários possam navegar e efetuar compras de forma eficiente em qualquer dispositivo.

Objetivos

- Desenvolver um *layout* responsivo que se adapte a diferentes tamanhos de tela;
- Garantir uma experiência de usuário suave durante a navegação pelos produtos e processo de compra;
- Testar a eficácia do *layout* em dispositivos com diferentes resoluções e propor soluções para melhorar a usabilidade.

Orientações

- Utilize frameworks responsivos, se necessário, para agilizar o desenvolvimento;
- Certifique-se de que os elementos da página, como botões de compra e formulários, são acessíveis e fáceis de usar em dispositivos móveis;
- Realize testes de desempenho para garantir uma experiência de carregamento rápido em conexões mais lentas.

ATIVIDADE DE FIXAÇÃO

1. **Menu horizontal:** crie um menu de navegação horizontal, utilizando HTML e CSS3. Aplique estilos para destacar os itens do menu.

2. **Menu vertical estilizado:** desenvolva um menu de navegação vertical com elementos estilizados. Utilize cores e fontes para tornar a apresentação mais atraente.

3. ***Layout* responsivo simples:** construa um *layout* responsivo básico. Certifique-se de que os elementos se ajustem a diferentes tamanhos de tela.

4. **Botões de transição:** adicione efeitos de transição aos botões do seu menu. Ao passar o *mouse* sobre eles, experimente transições suaves de cor ou tamanho, por exemplo.

5. ***Media queries*:** aplique *media queries* para otimizar a exibição do *layout* em dispositivos móveis. Teste a adaptação em diferentes larguras de tela.

6. **Teste de *layout* responsivo:** utilize algumas das ferramentas estudadas para testar o *layout* criado por você.

7. Explique, de forma simples, o conceito de um menu vertical em *web design*. Quais são as vantagens de usar um menu vertical em um *site*? Dê um exemplo de uma situação em que um menu vertical seria mais adequado do que um menu horizontal.

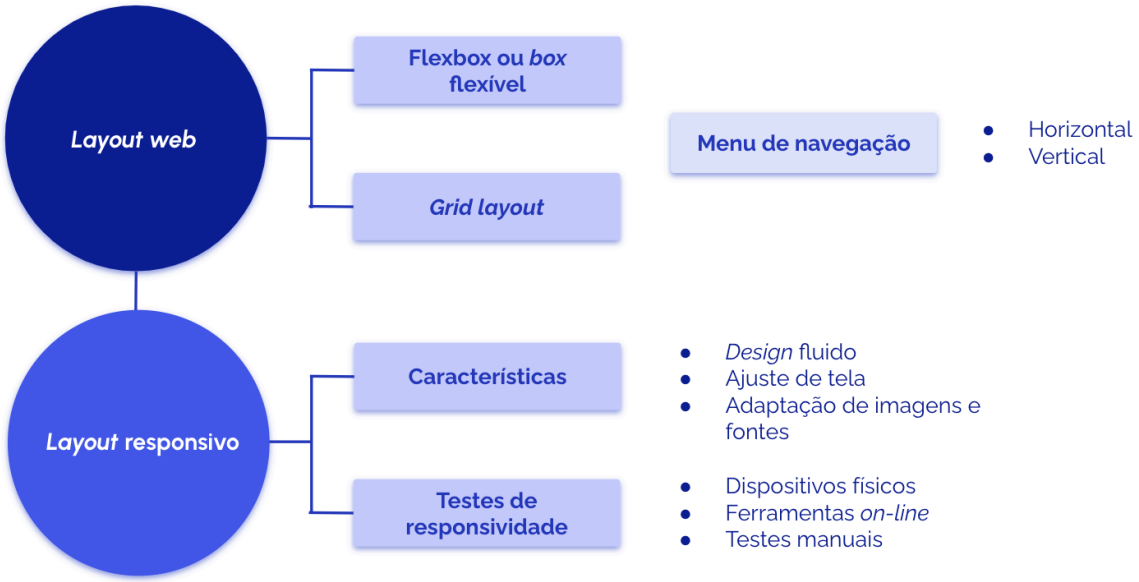
8. O que é um *layout* responsivo em *web design* e por que ele é importante para a experiência do usuário? Explique, brevemente, como um *layout* responsivo se adapta a diferentes dispositivos.

9. Quais são os principais desafios que os *designers* e desenvolvedores enfrentam ao desenvolver um *site* com um *layout* responsivo? Discuta como eles podem equilibrar a estética do *site* com a otimização para diferentes tamanhos de tela e como a velocidade de carregamento pode ser mantida em dispositivos móveis.

10. Quais são os principais fatores que os *designers* devem considerar para garantir uma navegação eficiente e esteticamente agradável ao criar um menu horizontal para um *site*? Descreva como a organização da informação e a legibilidade podem ser otimizadas em um menu horizontal para melhorar a experiência do usuário.

RESUMO

Estudamos que o CSS3 é uma ferramenta poderosa para aprimorar o *web design*, especialmente no *layout* e posicionamento de elementos. Durante nossa jornada, aprendemos a criar menus horizontais e verticais usando tecnologias, como *grid* e Flexbox, ferramentas que ajudam a organizar elementos de forma eficiente e responsiva.



Fonte: elaborado pelo autor (2024).

Abordamos que, ao criar menus horizontais, melhoramos a experiência de navegação do usuário, tornando-a intuitiva e acessível. Nesse contexto, criar um *layout* responsivo é crucial, pois, assim, a sua página poderá se adaptar a diferentes dispositivos e tamanhos de tela. Para isso, é importante se atentar a algumas características, como *design* fluido, **@media queries** e adaptação de imagens, garantindo uma experiência consistente.

Vimos, também, que testar o *design* responsivo em diversos dispositivos é essencial não apenas para uma navegação amigável, mas, também, para obter melhores classificações nos motores de busca, refletindo o compromisso com uma experiência do usuário eficaz.

CAPÍTULO 08

Explorando técnicas de *layout* e posicionamento com CSS3

O que esperar deste capítulo:

- Elementos HTML5 avançados, como **<video>**, **<audio>** e **<canvas>**;
- Transições e animações CSS3.

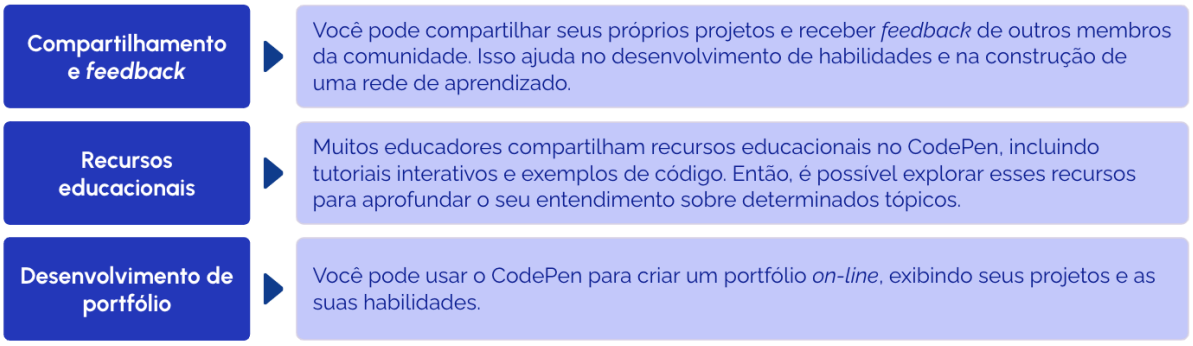
CodePen: o *playground* moderno dos desenvolvedores

Disponível em: <http://tinyurl.com/2xeznyr>. Acesso em 31 jan. 2024.

Para consolidar e experimentar as técnicas do HTML5 e do CSS3 que conhecemos anteriormente, o CodePen é o seu aliado. Essa é uma plataforma *on-line* que permite que os desenvolvedores e entusiastas da programação criem, experimentem e compartilhem códigos, especialmente em HTML, CSS e JavaScript. Ele oferece um **ambiente de desenvolvimento simplificado e interativo**, no qual os usuários podem criar pequenos projetos, protótipos ou trabalhar em projetos mais complexos, visualizando o resultado no navegador instantaneamente.

Para quem está começando a aprender programação, o CodePen pode ser uma ferramenta valiosa. A seguir, estão algumas formas de usufruir dos recursos dessa plataforma:

Experimentação Interativa	Ao testar e experimentar códigos de HTML, CSS e JavaScript de forma interativa, vendo imediatamente o resultado na mesma página, você entende como as diferentes linguagens de marcação e programação interagem para criar páginas *web* dinâmicas.
Projetos pequenos	Ao aplicar os conceitos aprendidos, você pode criar páginas *web* simples, galerias de fotos ou animações básicas, que são exemplos de projetos iniciais.
Visualização de códigos	A comunidade do CodePen é ativa e compartilha uma variedade de projeto. Assim, é possível explorar códigos criados por outros usuários para aprender novas técnicas e boas práticas.



Fonte: elaborado pelo autor (2024).

Como vimos anteriormente, o CodePen é uma ferramenta prática e pode ser usado de maneira flexível, dependendo dos objetivos do usuário. Ele oferece uma forma amigável e divertida de explorar a programação, encorajando a criatividade e o aprendizado prático dos conceitos de desenvolvimento *web*.

Dica: busque por *templates* relacionados a animações CSS3 e *layouts* responsivos no CodePen com inspirações prontas para usar em seus projetos.

Explorando o poder do HTML5 e do CSS3

Se você chegou até aqui nos seus estudos, está na hora de ir além com o nosso aprendizado. Por isso, vamos explorar o HTML5 e o CSS3, versões mais recentes dessas linguagens, para desenvolver habilidades interessantes. Porém, antes de começarmos a nos aventurar nesse assunto, vamos relembrar alguns pontos sobre HTML e CSS.

Fonte: elaborado pelo autor (2024).

HTML5

Nessa primeira parte, vamos nos aprofundar nos elementos avançados do HTML5, como **<video>**, **<audio>** e **<canvas>**. Juntos, vamos aprender a dar um *upgrade* na experiência multimídia dos *sites*, utilizando todos os recursos e possibilidades que a versão HTML5 tem para oferecer.

Vamos começar com os vídeos e músicas, já que, afinal, todos gostamos de conteúdo multimídia.

Tag <video>

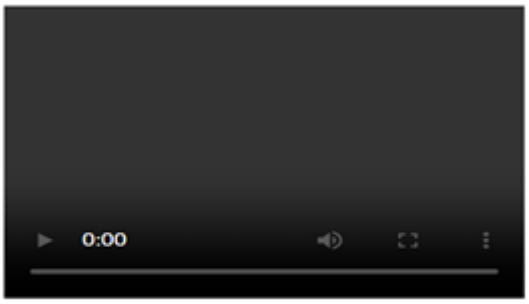
Para incorporar um vídeo em uma página *web,* utilizamos a *tag* **<video>**. Além disso, para tornar a experiência do usuário mais interativa e envolvente, é possível adicionar controles de reprodução, como *play*, *pause* e barra de progresso. Para isso, usamos o atributo **controls**. Existe, também, o elemento **<source>** dentro da *tag* **<video>**, que especifica a fonte do vídeo e o tipo de formato suportado. Além desses recursos, ainda é possível ajustar as dimensões do vídeo incorporado, usando os atributos **width** e **height**. Vamos ver um exemplo para que seja possível entender como tudo isso é feito.

Entrada:

```
3   <video width="320" height="240" controls>
4       <source src="seu_video.mp4" type="video/mp4">
5   </video>
```

Fonte: elaborado pelo autor (2024).

Saída (resultado na página):



Fonte: elaborado pelo autor (2024).

Tag <audio>

A *tag* **<audio>** permite incorporar arquivos de áudio diretamente na página que está sendo desenvolvida. Aqui, também podemos adicionar controles de áudio, o que é feito com o atributo **controls**. Assim como no caso da *tag* **<video>**, o elemento **<source>** é usado para especificar a fonte do áudio e o tipo de formato suportado. Vamos ver um exemplo.

Entrada:

```
<audio controls>
  <source src="🤟🎸♫.mp3" type="audio/mp3">
</audio>
```

Fonte: elaborado pelo autor (2024).

Saída (resultado na página):

Fonte: elaborado pelo autor (2024).

Tag <canvas>

Além de áudio e vídeo, também é interessante haver uma parte artística na nossa página. Para isso, temos a *tag* **<canvas>**, um elemento poderoso que nos permite desenhar gráficos, criar animações, renderizar visualizações interativas e, também, disponibilizar jogos direto na página.

Usando JavaScript com o contexto 2D (**getContext("2d")**), é possível desenhar formas, imagens e aplicar estilos. No exemplo abaixo, criamos um retângulo vermelho no *canvas*, mas as possibilidades são infinitas.

Entrada:

```
15    <canvas id="meuCanvas" width="200" height="100" style="border:1px solid ■#000;"></canvas>
16
17  <script>
18    var canvas = document.getElementById("meuCanvas");
19    var ctx = canvas.getContext("2d");
20    ctx.fillStyle = "#FF0000";
21    ctx.fillRect(10, 10, 150, 80);
22  </script>
```

Fonte: elaborado pelo autor (2024).

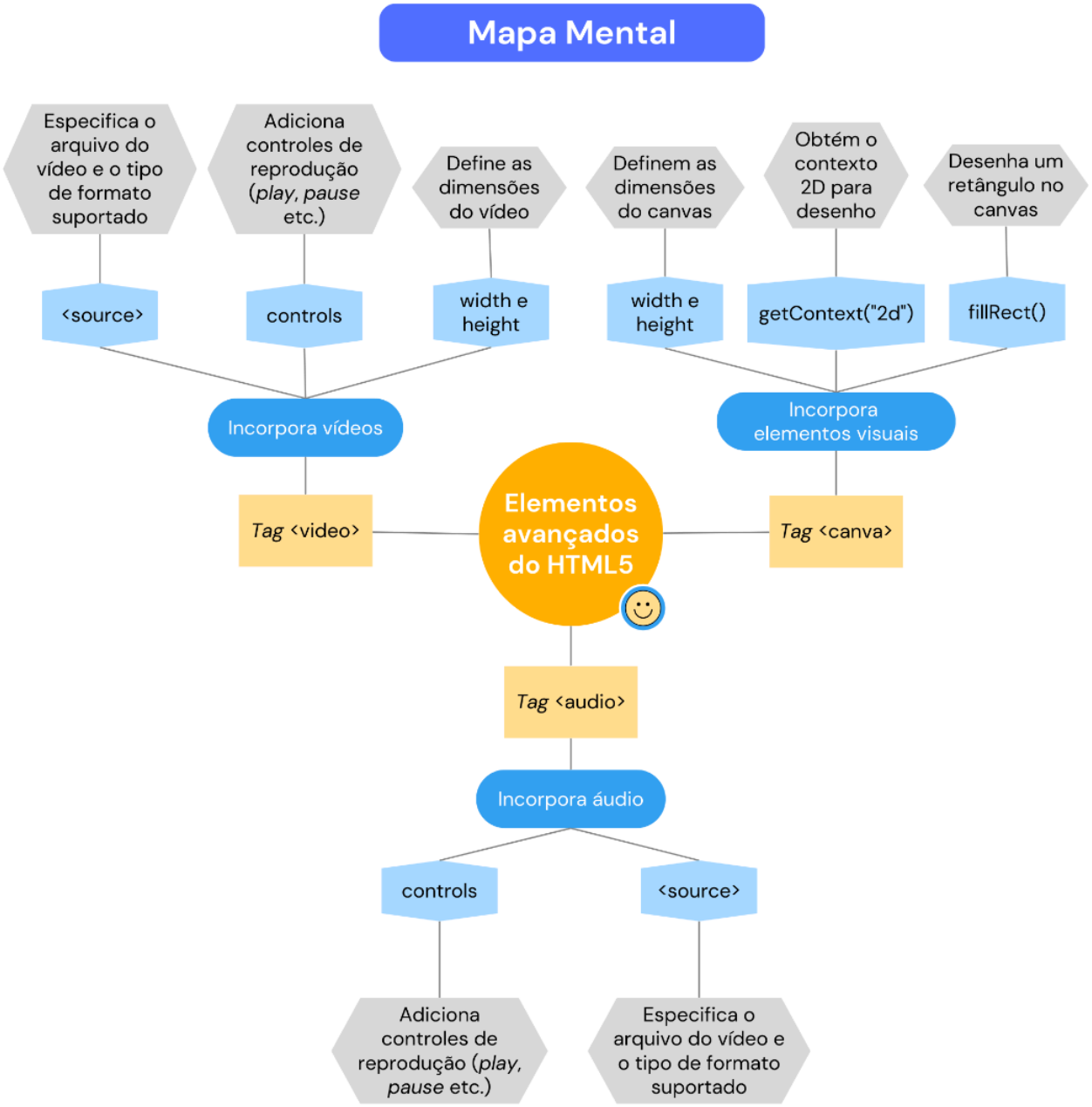
Saída (resultado na página):

Fonte: elaborado pelo autor (2024).

Hora do desafio!

Utilizando o CodePen, desenvolva uma página *web* que contenha um *player* de áudio e vídeo. Lembre-se de adicionar um menu de controles para o usuário.

Antes de partirmos para o CSS3, observe um mapa mental contendo o que aprendemos até aqui sobre os recursos do HTML5:



Cores e movimentos: conhecendo os recursos do CSS3

Agora, vamos mergulhar nas transições e animações do CSS3. Juntos, vamos descobrir como deixar as mudanças de estado dos elementos suaves e como criar animações incríveis com *keyframes*.

Com o domínio desses recursos, você conseguirá deixar o seu *site* com um visual moderno e interativo.

Transições e animações CSS3: elevando a experiência visual 🚀

Em um mundo cada vez mais digital, a estética e a interatividade das páginas *web* desempenham papel crucial na atração e engajamento do usuário. Nesse contexto, as transições e animações CSS3 se destacam como ferramentas poderosas para transformar uma interface estática em algo dinâmico e visualmente atraente. Vamos explorar a magia por trás desses recursos e descobrir como eles podem ser incorporados para elevar a experiência do usuário.

As transições são como pinceladas suaves que guiam o olhar do usuário através da página. Elas permitem a mudança gradual de propriedades, como cor, tamanho e posição, criando uma experiência mais fluida. Utilizando a propriedade **transition**, podemos definir o tempo e o tipo de efeito desejado. Por exemplo, ao passar o *mouse* sobre um botão, podemos alterar suavemente a sua cor, proporcionando um toque de elegância sutil.

Exemplo:

```
# style.css > ...
1
2   button {
3     background-color: #3498db;
4     transition: background-color 0.5s ease;
5   }
6
7   button:hover {
8     background-color: #e74c3c;
9   }
```

Fonte: elaborado pelo autor (2024).

As animações CSS3, por sua vez, são como pequenos espetáculos que transformam a página em um palco dinâmico. Com a propriedade **@keyframes**, podemos criar sequências de *frames* que definem como um elemento evolui ao longo do tempo, seja um *banner* deslizante, um ícone saltitante ou qualquer outra ideia criativa, como animações que proporcionam um toque personalizado ao *site*.

Exemplo:

```
style.css > ...
1
2   @keyframes bounce {  0%, 20%, 50%, 80%, 100% {
3       transform: translateY(0);
4     }
5     40% {
6       transform: translateY(-30px);
7     }
8     60% {
9       transform: translateY(-15px);
10    }
11  }
12
13  div {
14    animation: bounce 2s infinite;
15  }
16
```

Fonte: elaborado pelo autor (2024).

Ao explorar transições e animações CSS3, embarcamos em uma jornada de modernidade e inovação. Esses recursos não apenas tornam as páginas visualmente impressionantes, mas, também, proporcionam uma experiência envolvente. Este capítulo desvendará os mistérios por trás dessas técnicas, te capacitando a transformar *designs* estáticos em experiências dinâmicas.

DESAFIO PRÁTICO

Elementos HTML5 avançados, como <video>, <audio>, e <canvas>

Desafio: explorando a criatividade digital.

Descrição

Como aspirante a desenvolvedor *web*, você foi contratado para aprimorar a presença *on-line* de uma empresa de entretenimento que busca se destacar na internet. A empresa deseja uma página interativa que destaque os seus eventos, oferecendo uma experiência envolvente aos visitantes. A sua missão é incorporar elementos HTML5 avançados ao *site*, como vídeos e animações, para proporcionar uma experiência única e memorável.

Objetivos

- Utilizar as *tags* **<video>** e **<audio>** para integrar conteúdo multimídia relacionado aos eventos da empresa;
- Implementar transições e animações CSS3 para destacar elementos visuais e proporcionar uma experiência mais dinâmica.

Orientações

- **Elementos multimídia**: utilize a *tag* **<video>** para exibir um *trailer* dos eventos e **<audio>** para ambientar a página com uma trilha sonora temática;
- **Animações CSS3**: integre transições e animações CSS3 para tornar os elementos visuais mais atrativos. Considere aplicar efeitos ao passar o *mouse* sobre elementos-chave.

DESAFIO PRÁTICO II

Transições e animações CSS3

Desafio: desenvolvimento de um *site* pessoal com *layout* responsivo.

Descrição

Imagine que você deseja criar um *site* pessoal para compartilhar informações sobre você, seus interesses e os seus projetos. O objetivo é desenvolver um *layout* responsivo que se adapte a diferentes tamanhos de tela, proporcionando uma experiência de qualidade tanto em computadores quanto em dispositivos móveis.

O *site* pessoal deve conter algumas seções, como "Sobre Mim", "Projetos", "Habilidades" e "Contato". O *layout* precisa ser amigável e de fácil navegação, ajustando-se automaticamente para garantir uma apresentação atrativa em *smartphones*, *tablets* e computadores.

Objetivos

- Utilizar estrutura semântica HTML5 para organizar o conteúdo;
- Implementar um *design* responsivo usando *media queries* em CSS3;
- Adicionar uma barra de navegação que se ajuste conforme a largura da tela;
- Incluir imagens, ícones ou outros elementos gráficos para tornar o *site* visualmente atraente;
- Testar o *layout* em diferentes dispositivos para garantir a adaptabilidade.

Orientações

- Utilize *tags* HTML5 apropriadas para estruturar o conteúdo da página;
- Utilize *media queries* em CSS para ajustar estilos, de acordo a largura da tela;
- Crie uma barra de navegação simples que seja intuitiva para o usuário;
- Incorpore imagens ou ícones para ilustrar suas habilidades e projetos;
- Teste o *site* em dispositivos móveis, *tablets* e computadores para garantir uma experiência consistente.

Juntos, aprendemos que o HTML5, a quinta versão da linguagem de marcação HTML, trouxe melhorias significativas para o desenvolvimento *web*. Uma das suas inovações mais notáveis é a incorporação nativa de áudio e vídeo. A *tag* **<audio>** permite a fácil inclusão de conteúdo de áudio, enquanto a *tag* **<video>** possibilita a incorporação de vídeos, proporcionando uma experiência multimídia mais rica. Vimos que as duas *tags* suportam atributos, como **controls,** para oferecer opções de reprodução ao usuário.

Além disso, estudamos que o HTML5 introduziu a *tag* **<canvas>**, oferecendo uma área de desenho no navegador, permitindo a criação de gráficos, animações e imagens

interativas através de JavaScript. Essa funcionalidade expandiu as possibilidades de *design* e interatividade na *web*.

Também abordamos que o CSS3, a terceira versão das Folhas de Estilo em Cascata, representa um avanço substancial na estilização *web*, proporcionando maior flexibilidade, controle e expressividade às páginas. Vimos que, com recursos como o *design* responsivo, as transições, as animações e os estilos mais avançados, o CSS3 desempenha papel crucial na criação de interfaces *web* modernas e visualmente atraentes, resultando em *layouts* mais flexíveis que se ajustam a diferentes tamanhos de tela, melhorando a experiência do usuário em dispositivos variados.

Exploramos que uma outra inovação significativa do CSS3 é a capacidade de aplicar efeitos de transição e animações diretamente no navegador, sem a necessidade de *scripts* adicionais. Isso proporciona uma experiência visual mais dinâmica e interativa, com transições suaves entre estados de estilo.

Finalmente, vimos que, ao abraçar os elementos avançados e técnicas modernas, você não estará apenas construindo páginas *web*, mas, também, moldando experiências digitais. Avance, experimente e encante os seus usuários! Após essa viagem pelo mundo do HTML5 e CSS3, você está pronto para criar páginas que vão além do básico. Agora, é hora de botar a mão na massa e fazer seus *sites* brilharem na *web.*

ATIVIDADE DE FIXAÇÃO

1. **Criando página simples:** crie uma página simples em HTML, na qual serão utilizadas aplicações com vídeos, áudio e *canvas*.

2. **Botões de transição:** crie um menu (horizontal ou vertical) e adicione efeitos de transição aos botões do seu menu. Ao passar o *mouse* sobre eles, experimente utilizar transições suaves de cor ou tamanho.

3. **Inserindo vídeo:** insira um vídeo em sua página. O ideal é que seja um vídeo curto e que tenha relação com o conteúdo de desenvolvimento *web*.
4. **Inserindo áudio:** insira um áudio em sua página. O ideal é que seja um áudio que tenha relação com o conteúdo de desenvolvimento *web*.

5. **Inserindo canvas:** insira um *canvas* em sua página. O ideal é que você elabore um *canvas* que tenha relação com o conteúdo de desenvolvimento *web*.

6. Você está desenvolvendo um *site* de treinamento *on-line* e deseja incorporar um vídeo instrucional. Qual *tag* HTML5 é apropriada para incorporar um vídeo no seu *site*?
    a. **<media>**
    b. **<video>**
    c. **<multimedia>**
    d. **<playback>**

7. Ao criar uma página para um *podcast*, qual *tag* HTML5 você deve utilizar para incorporar arquivos de áudio?
    a. **<sound>**
    b. **<audio>**
    c. **<music>**
    d. **<playback>**

8. Você deseja criar uma aplicação *web* interativa que envolve desenhos e gráficos dinâmicos. Qual elemento HTML5 é mais adequado para essa finalidade?
    a. **<draw>**
    b. **<graphics>**
    c. **<canvas>**
    d. **<paint>**

9. Para adicionar um efeito de transição suave entre diferentes estados de estilo em um elemento HTML, qual propriedade do CSS3 você deve usar?
    a. **transition-effect**
    b. **animate**
    c. **style-transition**
    d. **transition**

10. Você está criando uma *landing page* e deseja adicionar uma animação sutil para chamar a atenção para um elemento específico. Qual propriedade do CSS3 é

utilizada para criar animações?

a. **animate**
b. **transition**
c. **keyframes**
d. **transform**

CAPÍTULO 09

Pré-processadores e *frameworks* CSS: desbravando o mundo do estilo!

O que esperar deste capítulo:

- Aprender a usar pré-processadores CSS, como SASS ou LESS;
- Introduzir o Bootstrap como um *framework* CSS.

Se você já se aventurou pelo universo do desenvolvimento *web*, deve ter se deparado com termos como **SASS**, **LESS** e **Bootstrap**, mas o que essas palavras significam? Agora, vamos explorá-las juntos, entendendo, também, os **pré-processadores CSS e os *frameworks*. Vamos lá?**

Pré-processadores e *frameworks*

Os pré-processadores CSS e os *frameworks* são ferramentas que facilitam o desenvolvimento e a manutenção de estilos em folhas de estilo para páginas *web*. Eles, no entanto, desempenham funções distintas.

Pré-processadores	*Frameworks*
• Os pré-processadores ajudam a criar estilos para páginas *web* de forma simples, flexível. • Os pré-processadores permitem que você crie blocos reutilizáveis, em vez de repetir as mesmas linhas de código, por exemplo.	• Os *frameworks* CSS são conjuntos predefinidos de estilos e padrões de *layout* que podem ser utilizados para agilizar o desenvolvimento *web*.
Exemplos de pré-processadores são o **SASS** e o **LESS**.	O **Bootstrap** é um exemplo de *framework* e, com ele, você não precisa começar do zero, pois ele oferece um conjunto de estilos pré-definidos, facilitando a criação de páginas bonitas e consistentes.

Fonte: elaborado pelo autor (2024).

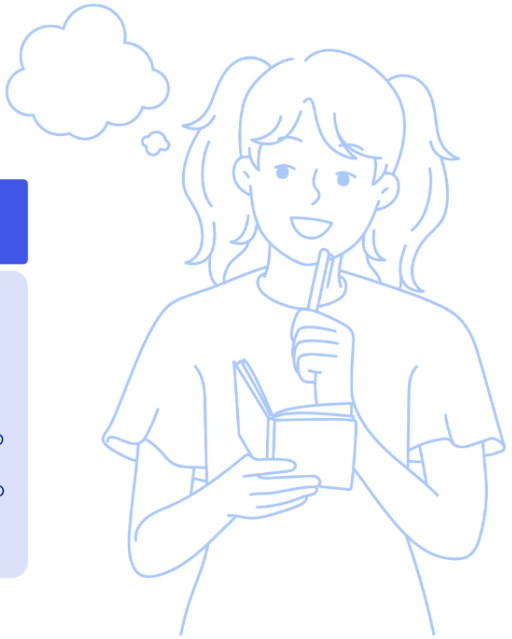
Como isso pode nos ajudar? 💻

Agora, você deve estar se perguntando: por que eu deveria usar isso?

Pense na seguinte situação...

Imagine que você está construindo uma casa!

Os **pré-processadores** são como uma caixa de ferramentas avançada, repleta de instrumentos que facilitam seu trabalho. Os ***frameworks*** são como um plano de construção sólido, dando a você uma estrutura robusta para começar.
Em resumo, essas ferramentas não só tornam o desenvolvimento mais rápido e eficiente, mas também mais divertido!



Fonte: elaborado pelo autor (2024).

Ao mergulhar nos pré-processadores e nos *frameworks*, você estará preparando seu arsenal para enfrentar os desafios do desenvolvimento *web* moderno. Prepare-se para explorar um novo mundo de estilo e produtividade! 🚀

O uso de pré-processadores CSS, como **SASS (*Syntactically Awesome Stylesheets*)** ou **LESS (*Leaner Style Sheets*)**, representa um salto qualitativo no desenvolvimento *web* moderno. Essas ferramentas poderosas oferecem uma abordagem mais avançada e eficiente para lidar com estilos, superando as limitações do CSS tradicional. Assim, é importante conhecer os benefícios e as funcionalidades desses pré-processadores.

Veja só um aperitivo! 😋

Recursos dos pré-processadores

Variáveis e Aninhamento

As **variáveis** permitem definir valores reutilizáveis. Além disso, o **aninhamento** de seletores, ou seja, a prática de incluir um seletor dentro de outro, simplifica a estrutura do código, tornando-o mais legível e fácil de manter.

Mixins e Funções	A capacidade de criar ***mixins*** (fragmentos de código reutilizáveis) e **funções** oferece uma modularidade significativa, permitindo que você consiga reutilizar estilos e código já definidos, simplificando a atualização global do *design*.
Herança de Estilos	Os pré-processadores suportam **herança de estilos**, um recurso fundamental para garantir consistência visual em várias partes do projeto, reduzindo redundâncias e aumentando a coesão.

Fonte: elaborado pelo autor (2024).

Essas são apenas algumas funcionalidades desses pré-processadores. Exploraremos outras conforme avançarmos no estudo de cada um deles.

Fonte: elaborado pelo autor (2024).

Ao compreender os benefícios e a aplicação prática dos pré-processadores CSS, os desenvolvedores estarão aptos a aprimorar seus fluxos de trabalho, acelerar o desenvolvimento e manter folhas de estilo mais flexíveis e fáceis de manter. Este capítulo representa uma etapa essencial na jornada de qualquer desenvolvedor *web* em busca de eficiência e profissionalismo.

Vamos começar conhecendo melhor o SASS!

SASS (*Syntactically Awesome Stylesheets*)

Disponível em:<http://tinyurl.com/3xn9mw7c>. Acesso em: 31 jan. 2024.

O SASS é um pré-processador CSS que fornece recursos poderosos e uma abordagem mais programática para o desenvolvimento de estilos. Alguns dos principais recursos do SASS incluem:

- **Variáveis**, pois a capacidade de declarar variáveis permite armazenar valores reutilizáveis, facilitando a manutenção e a consistência no *design*. No exemplo abaixo, as variáveis **$primary-color** e **$font-size** são declaradas para armazenar esses valores.

```
// Exemplo com variáveis

@primary-color: #3498db;
@font-size: 16px;

body {
  font-size: @font-size;
  color: @primary-color;
}

a {
  color: @primary-color;

  &:hover {
    text-decoration: underline;
  }
}
```

Fonte: elaborado pelo autor (2024).

- ***Nesting* (aninhamento)**, uma vez que a sintaxe de aninhamento de seletores no SASS reflete a estrutura hierárquica do HTML, tornando o código mais claro e conciso. Veja o exemplo abaixo:

```
// Exemplo com aninhamento

nav {
  background-color: #333;

  ul {
    list-style: none;
    padding: 0;

    li {
      display: inline-block;
      margin-right: 10px;

      a {
        color: white;
        text-decoration: none;

        &:hover {
          text-decoration: underline;
        }
      }
    }
  }
}
```

Fonte: elaborado pelo autor (2024).

- ***Mixins***, que são blocos de código reutilizáveis, facilitando a aplicação de estilos

complexos de maneira eficiente. Neste exemplo, o *mixin* **border-radius** é criado para fornecer uma maneira eficiente de aplicar estilos de borda arredondada. Ele é, então, utilizado nos seletores **.box** e **.button**;

```scss
// Exemplo com mixins

@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
  border-radius: $radius;
}

.box {
  @include border-radius(10px);
  background-color: #eee;
}

.button {
  @include border-radius(5px);
  background-color: #3498db;
  color: white;
  padding: 10px;
}
```

Fonte: elaborado pelo autor (2024).

- **Herança**, pois o SASS suporta herança de estilos, permitindo que seletores compartilhem estilos comuns. No exemplo, a herança é usada para compartilhar estilos comuns entre os botões primários e secundários, evitando a repetição de código.

```scss
// Exemplo com herança

%base-button {
  padding: 10px;
  font-size: 16px;
  text-align: center;
}

.button-primary {
  @extend %base-button;
  background-color: #3498db;
  color: white;
}

.button-secondary {
  @extend %base-button;
  background-color: #ddd;
  color: #333;
}
```

Fonte: elaborado pelo autor (2024).

Saiba mais no vídeo abaixo:

Fonte: Vídeo *#ScriptFE: CSS Sass - Aprenda em 15 minutos*, do canal do YouTube Násser Youssef Ali. Disponível em: <https://youtu.be/KnsNYOPHyTc>. Acesso em 31 de jan. de 2024.

Após ver o vídeo, responda:

a. Qual é a sua percepção em relação ao uso do SASS e do CSS3?
b. Qual é a dificuldade que você encontra para aplicar o SASS?

LESS (*Leaner Style Sheets*)

Disponível em: <http://tinyurl.com/2h84d5e7>. Acesso em 31 jan. 2024.

O LESS oferece funcionalidades semelhantes ao SASS, proporcionando uma maneira mais dinâmica e flexível de escrever estilos. Abaixo, veja alguns recursos-chave do LESS.

- **Variáveis**: da mesma forma que o SASS, o LESS permite a definição de variáveis para facilitar a manutenção e a consistência;

- ***Mixins***: no LESS, os *mixins* são semelhantes aos SASS, oferecendo uma maneira de reutilizar blocos de código;
- **Aninhamento**: o aninhamento de seletores no LESS simplifica a estrutura do código, melhorando a legibilidade;
- **Funções**: o LESS introduz funções que podem ser aplicadas para realizar operações em valores. Na imagem abaixo, a função **.calculate-line-height** é criada para calcular a altura da linha com base no tamanho da fonte. Isso ilustra o uso de funções no LESS para realizar operações em valores.

```
// Exemplo com funções

@base-font-size: 16px;

.calculate-line-height(@font-size) {
  line-height: @font-size * 1.5;
}

body {
  font-size: @base-font-size;
  .calculate-line-height(@base-font-size);
}
```

Fonte: elaborado pelo autor (2024).

SASS vs LESS 🥊

Disponível em: <http://tinyurl.com/mtbas4tb>. Acesso em: 31 jan. 2024.

Recursos	SASS	LESS
Variáveis	As variáveis no SASS são declaradas com o prefixo **`$`** (por exemplo, `$primary-color`).	Utiliza o prefixo **`@`** para variáveis, como em `@primary-color`.
	Exemplo: $primary-color: #3498db	**Exemplo**: @primary-color: #3498db
Aninhamento	Permite aninhar seletores, facilitando a leitura e a organização do código.	Também suporta aninhamento de seletores, proporcionando uma estrutura mais clara e legível.
	Exemplo: nav { ul { margin: 0; } }	**Exemplo**: `.nav { & ul { margin: 0; } }`
Mixins	Os *mixins* são definidos usando `@mixin` e aplicados com `@include`.	Declara *mixins* com `.mixin()` e os utiliza com `.mixin()`.
	Exemplo: `@mixin border-radius($radius) { ... }`	**Exemplo**: `.border-radius(@radius) { ...}`
Herança	Permite que um seletor herde estilos de outro usando `@extend`.	Alcança herança de estilos por meio da reutilização de *mixins*, como `.success { .message(); }`.
	Exemplo: error { @extend .message; }	**Exemplo**: `.success { .message(); }`

Fonte: elaborado pelo autor (2024).

Tanto o SASS quanto o LESS oferecem funcionalidades poderosas, mas as diferenças sutis podem influenciar a escolha entre eles, dependendo das preferências da equipe e das necessidades específicas do projeto. Ambos são ótimas opções para aprimorar a eficiência e a manutenção de estilos em projetos *web* modernos.

Introdução ao Bootstrap: a revolução do desenvolvimento *front-end*

Se você está ansioso para aprimorar suas habilidades em desenvolvimento *web*, o Bootstrap é a chave para abrir portas incríveis!

Você sabia?

Em 2010, o Twitter, enfrentando desafios de consistência e velocidade no desenvolvimento *web*, lançou internamente um conjunto de ferramentas para acelerar o processo. Essas ferramentas se tornaram a espinha dorsal do que seria conhecido como Bootstrap.

Fonte: elaborado pelo autor (2024).

Impulsionados pelo espírito do código aberto, a equipe do X (antigo Twitter) decidiu compartilhar o Bootstrap com a comunidade. A simplicidade, a consistência visual e a facilidade de uso conquistaram os corações e as mentes dos desenvolvedores, resultando em uma adoção global. Empresas, pequenas equipes e desenvolvedores individuais encontraram no Bootstrap uma ferramenta essencial.

O Bootstrap não se acomodou com o sucesso. A equipe de desenvolvimento continuou aprimorando o *framework*, lançando novas versões e incorporando as últimas tendências de *design* e tecnologia. Hoje, ele é mais do que um *framework:* é um ícone na história do desenvolvimento *web*. Sua influência é evidente em incontáveis *sites* e projetos, deixando um legado duradouro na forma como concebemos e construímos experiências *on-line.*

Imagine se você pudesse criar páginas *web* incríveis sem passar horas e horas codificando cada detalhe. O Bootstrap é exatamente isso: um conjunto mágico de estilos CSS, *scripts* JavaScript e componentes prontos para o uso, transformando a criação de *websites* em uma jornada empolgante e sem complicações.

Saiba mais no vídeo abaixo:

Fonte: Vídeo *Bootstrap 📃 (Guia para Iniciantes) 📃 2024 - Hostinger Brasil*, do canal Hostinger Brasil. Disponível em: https://youtu.be/jsTJL6Da5wc. Acesso em: 31 jan. 2024.

Após ver o vídeo, responda:

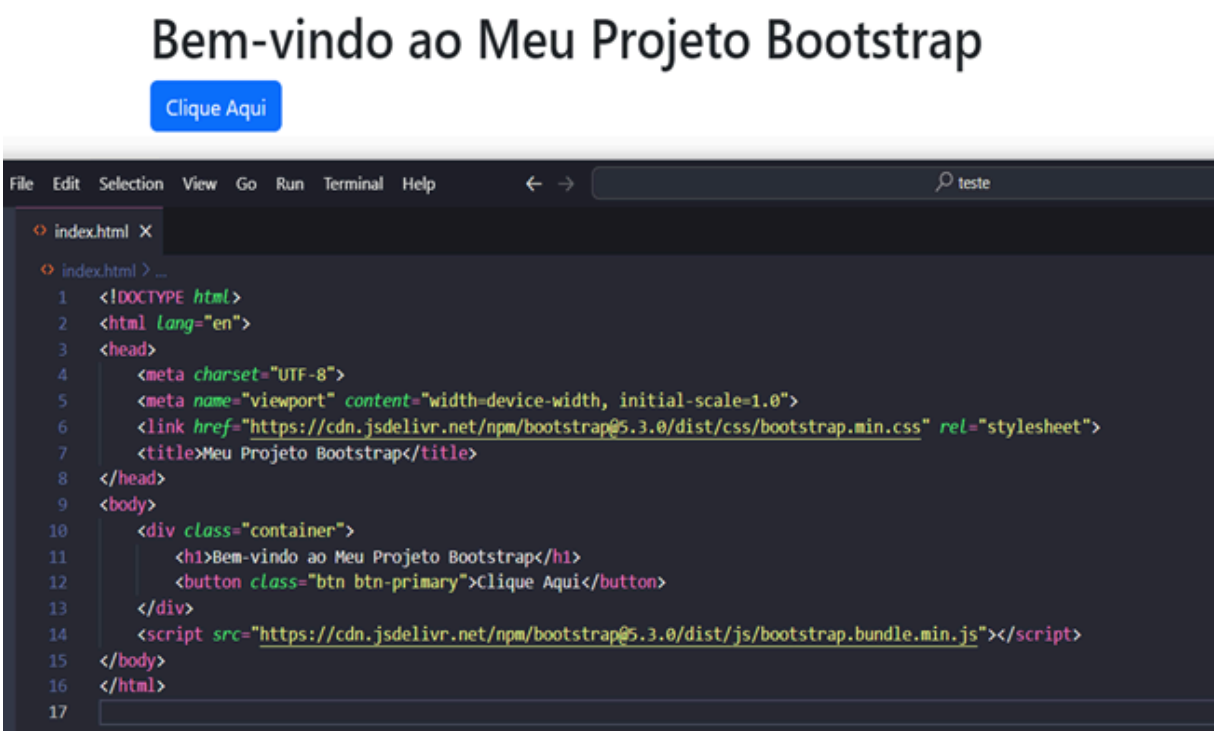
a. Você acredita que só trabalhar com o Bootstrap é suficiente para desenvolvimento *web*? Explique.
b. Quais são as vantagens e as desvantagens do Boostrap?

Fonte: elaborado pelo autor (2024).

Principais características e vantagens

- **Grade responsiva**: o Bootstrap incorpora uma grade flexível e responsiva que se ajusta automaticamente a diferentes tamanhos de tela, proporcionando uma experiência consistente em dispositivos variados;
- **Componentes prontos para uso**: com uma variedade de componentes pré-estilizados, o Bootstrap permite que os desenvolvedores construam rapidamente páginas *web* atraentes, sem a necessidade de começar do zero;
- **Facilidade de personalização**: embora forneça estilos padrão atraentes, o Bootstrap é altamente personalizável. Os desenvolvedores podem ajustar o tema, as cores e os estilos para atender às necessidades específicas do projeto;
- **Compatibilidade com navegadores**: o Bootstrap é compatível com todos os principais navegadores, garantindo uma experiência consistente para os usuários, independentemente do navegador que estão usando.

Exemplos de utilização:



Fonte: elaborado pelo autor (2024).

Como o Bootstrap torna as coisas mais produtivas

- **Rápido desenvolvimento**: com componentes prontos para uso, o Bootstrap acelera significativamente o desenvolvimento, permitindo que os desenvolvedores se concentrem em aspectos mais complexos;
- **Consistência visual**: ao adotar a estética visual do Bootstrap, os projetos ganham uma consistência que agrada aos olhos dos usuários;
- **Adaptação responsiva**: a grade responsiva facilita a criação de *layouts* que se ajustam perfeitamente a dispositivos de todos os tamanhos, economizando tempo em ajustes manuais;
- **Documentação abundante**: o Bootstrap oferece uma documentação extensa e clara, fornecendo informações detalhadas sobre como usar cada componente;
- **Funcionalidade**: com sua abordagem pragmática e eficiente, o Bootstrap torna-se uma escolha valiosa para desenvolvedores que buscam criar interfaces *web* atraentes e altamente funcionais.

Site responsivo com Bootstrap: navegação, conteúdo e rodapé

Fonte: elaborado pelo autor (2024).

Barra de navegação (<nav>)

Fonte: elaborado pelo autor (2024).

- A barra de navegação é criada usando a classe **.navbar** do Bootstrap;
- O botão de alternância (**<button class="navbar-toggler">**) é usado para telas menores;
- Os itens de navegação (**<ul class="navbar-nav">**) são organizados em uma lista.

Conteúdo principal ( )

```
<!-- Conteúdo Principal -->
<div class="container mt-5">
  <h1>Bem-vindo ao Meu Site</h1>
  <p>Explore nosso conteúdo incrível e descubra as maravilhas que temos para oferecer.</p>
</div>
```

Fonte: elaborado pelo autor (2024).

- Todo o conteúdo principal é envolvido em um *container* para centralização e ajustes em telas diferentes;
- Inclui um título (**<h1>**) e um parágrafo descritivo (**<p>**).

Rodapé (<footer>)

```
<!-- Rodapé -->
<footer class="footer mt-auto py-3 bg-light">
  <div class="container">
    <span class="text-muted">© 2024 Meu Site. Todos os direitos reservados.</span>
  </div>
</footer>
```

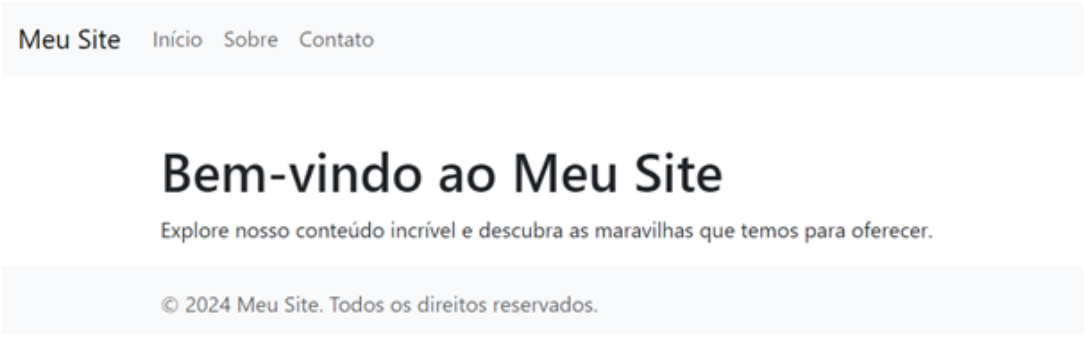
Fonte: elaborado pelo autor (2024).

- O rodapé é estilizado usando a classe **.footer** do Bootstrap;
- Contém um texto simples indicando os direitos autorais.

Links para Bootstrap (<link> e <script>):

- Os *links* para os arquivos CSS e JS do Bootstrap são fornecidos diretamente do CDN.

Esse código é a base para criar um *site* responsivo e esteticamente agradável usando Bootstrap. Os comentários fornecem uma explicação detalhada de cada parte do código.



Fonte: elaborado pelo autor (2024).

DESAFIO PRÁTICO

Introdução ao Bootstrap como um framework CSS

Desafio: melhoria do *layout* de uma página de receitas, utilizando o Bootstrap.

Descrição

A proposta é aprimorar a página de receitas de um *site*, tornando-a mais atrativa e funcional em dispositivos variados. O Bootstrap será utilizado para simplificar o processo de criação de um *layout* responsivo.

A página atual apresenta uma lista de receitas, porém carece de melhorias visuais e de usabilidade. O objetivo é desenvolver um *design* mais moderno, organizando as receitas de maneira clara e assegurando que a página seja amigável em dispositivos móveis.

Objetivos

- Implementar um menu de navegação simples utilizando o Bootstrap;
- Organizar as receitas em *cards* ou colunas para facilitar a visualização;
- Utilizar classes de responsividade do Bootstrap para adaptar o *layout* em diferentes tamanhos de tela;
- Adicionar botões de "Detalhes" em cada receita para futuras expansões;
- Garantir que as imagens e o texto estejam bem distribuídos e se ajustem a

diferentes dispositivos.

Orientações

- Utilize a documentação do Bootstrap para inserir um menu de navegação;
- Explore classes, como **container**, **row** e **col**, para organizar o conteúdo;
- Adicione classes de responsividade, como **d-flex** e **flex-wrap**, para otimizar o *layout*;
- Experimente utilizar componentes Bootstrap, como modais, para os detalhes das receitas, mesmo que não estejam implementados inicialmente.

DESAFIO PRÁTICO II

Introdução ao Bootstrap como um framework CSS

Desafio: modernizando a página inicial.

Descrição

A empresa ABC decidiu atualizar a sua página inicial para torná-la mais atraente e responsiva. Eles contam com a sua ajuda para incorporar elementos visuais modernos, tais como navegação flexível, seções dinâmicas e um rodapé informativo.

Objetivos

- Implementar um menu de navegação horizontal utilizando Bootstrap;
- Adicionar uma seção de destaque na página inicial usando componentes visuais modernos oferecidos pelo Bootstrap;
- Incorporar um rodapé responsivo com informações úteis.

Orientações

- Utilize o Bootstrap para criar uma estrutura sólida e visualmente atraente;
- Explore diferentes componentes e classes do Bootstrap para melhorar a aparência da página;
- Certifique-se de que a página seja responsiva, adaptando-se a diferentes tamanhos de tela.

ATIVIDADE DE FIXAÇÃO

1. ***Mixins* mágicos**: utilize um pré-processador CSS (SASS ou LESS) para criar um *mixin* que aplique um gradiente de cores em diferentes elementos da página.

2. **Estilo personalizado**: escolha uma página existente e implemente estilos personalizados, usando SASS ou LESS. Destaque a facilidade de manutenção e a clareza no código.

3. ***Grid* com Bootstrap**: crie um *layout* responsivo usando a grade do Bootstrap e explore diferentes classes para colunas em dispositivos de diferentes tamanhos.

4. **Componentes do Bootstrap**: utilize componentes do Bootstrap, como *navbars* e carrosséis, em um projeto, visando demonstrar como eles podem agilizar o desenvolvimento.

5. **Adapte estilos com Bootstrap**: modifique o estilo padrão de um componente do Bootstrap, usando classes adicionais ou personalizando variáveis de tema.

6. **Transformações com SASS**: implemente transições suaves em um elemento HTML, usando SASS para tanto. Mostre como a linguagem simplifica a criação de efeitos visuais.

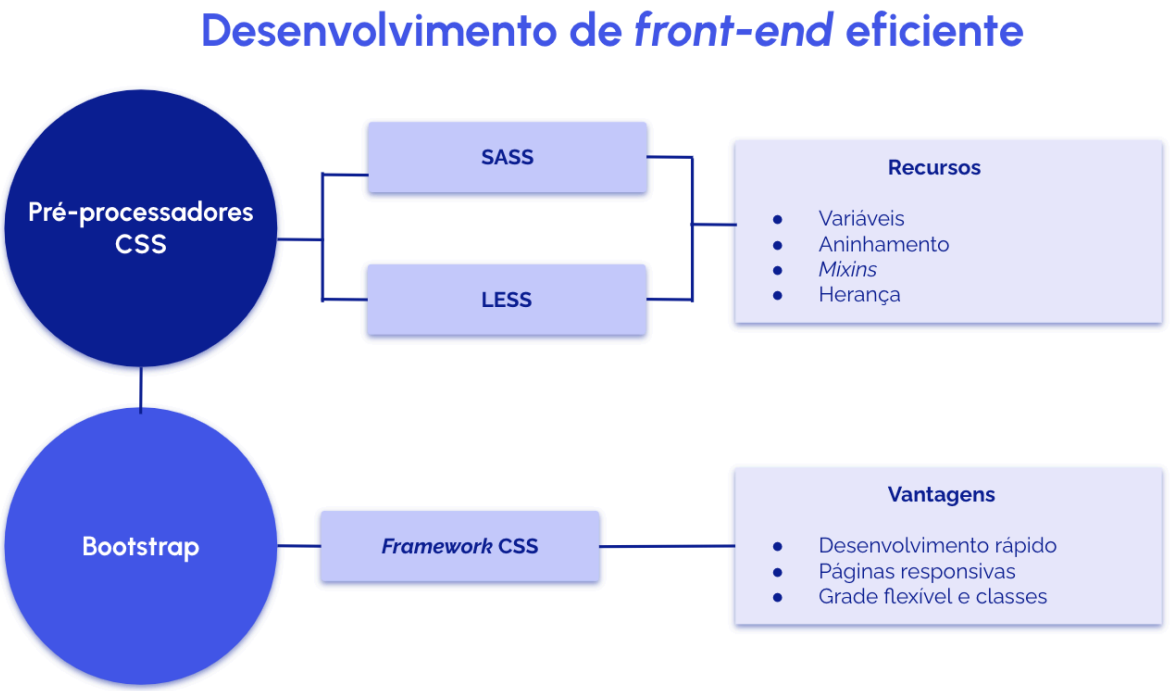
7. **Refatoração LESS**: pegue um arquivo CSS existente e converta-o para LESS, destacando as melhorias na organização e legibilidade do código.

8. **Personalização avançada com Bootstrap**: explore recursos avançados do Bootstrap, como criação de temas personalizados, para adaptar o visual do *framework* de acordo com as necessidades do projeto.

9. **Animações com LESS**: usando LESS, crie uma animação envolvente em um elemento e ressalte como a sintaxe simplificada facilita o desenvolvimento de animações CSS.

10. **Integração SASS e Bootstrap**: construa uma página *web* integrando SASS para estilos personalizados e Bootstrap para *layout* responsivo, evidenciando a sinergia dessas tecnologias.

Nós mergulhamos nas águas dos pré-processadores CSS, explorando ferramentas como SASS e LESS, e aprendemos que, com variáveis, *mixins* e aninhamento, essas tecnologias nos proporcionam mais controle, reusabilidade e organização em nossos estilos, tornando a escrita de código CSS mais eficiente.



Fonte: elaborado pelo autor (2024).

Em seguida, adentramos o universo do Bootstrap, um poderoso *framework* CSS que simplifica e acelera o desenvolvimento *web*. Com seus estilos predefinidos, componentes e utilitários, o Bootstrap oferece uma abordagem eficaz para criar páginas responsivas. Sua grade flexível e classes facilitam a obtenção de *designs* modernos e consistentes.

Em resumo, exploramos ferramentas que impulsionam nosso desenvolvimento *front-end*, tornando-o mais eficiente, organizado e alinhado com as melhores práticas. Agora, estamos mais preparados para criar experiências incríveis na *web*!

Referências

DIAZ, Léo. *Curso de CSS3 unidade 1 - introdução ao CSS*. SlideShare, 25 maio 2016. Disponível em: https://pt.slideshare.net/wilborn7/curso-de-css3-unidade-1-introduo-ao-css. Acesso em: 5 fev. 2024.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. *Lógica de Programação*: A Construção de Algoritmos e Estruturas de Dados. 4. ed. São Paulo: Pearson Prentice Hall, 2019.

FLATSCHART, Fábio. *HTML 5*: embarque imediato. 1. ed. Rio de Janeiro: Brasport, 2011.

GOLDBERG, Josh. *Aprendendo TypeScript*. São Paulo: Novatec, 2022.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. *Algoritmos*: Lógica para Desenvolvimento de Programação de Computadores. 28. ed. São Paulo: Érica, 2018.

MARTINS, Elaine. *O que é World Wide Web?*. TechMundo, 17 out. 2008. Disponível em: https://www.tecmundo.com.br/web/759-o-que-e-world-wide-web-.htm. Acesso em: 5 fev. 2024.

PAZ, Mônica. *Webdesign*. Curitiba: Intersaberes, 2021.

RAMALHO, Luciano. *Python Fluente*: Programação Clara, Concisa e Eficaz. São Paulo: Novatec, 2016.

BORGES, Luiz Eduardo. *Python para Desenvolvedores*. 2. ed. Rio de Janeiro: Novatec, 2010.

ZIVIANI, Nivio. *Projeto de Algoritmos com Implementações em Pascal e C*. 4. ed. São Paulo: Cengage Learning, 2015.