Programação Web

1. Contextualização

> Como surgiu a Web?

Todos utilizamos a internet, mas e como ela surgiu? Como ela funciona?

Seres humanos vivem em sociedade, por isso necessitamos nos comunicar.

pedro.mateus@unicesumar.edu.br

A comunicação é a base de tudo, o rádio, telefone, televisão contribuiram muito para transformar a comunicação rápida e abrangente.

A internet é uma das formas de comunicação mais utilizadas atualmente.

Antes de falar mais sobre a internet, vamos voltar um pouquinho...

Comunicação a longa distância

> Telégrafo eletromagnético — Código Morse

Em **1835**, *Samuel Finley Breese Morse*, desenvolveu um sistema de codificação, que possibilitava passar mensagens a longa distância.

Esse sistema permitiu a criação de um telégrafo eletromagnético, o equipamento não possibilitava enviar e receber mensagens escritas ou faladas.

O sistema utilizava a codificação criada anteriormente, e através de um código de pontos e traços, era possível identificar letras e palavras.

Por meio de um sistema de transmição, recepcção e interpretação, para que as mensagens pudessem ser utilizadas em longas distâncias.

Também foi *Samuel Morse* que inventou o pedido de socorro **SOS**, representado por `...---...` no código morse.

Letra	Código Morse	Letra	Código Morse	Letra	Código Morse	Letra	Cód Mo
A	.-	I	..	Q	--.-	Y	-.--
B	-...	J	.---	R	.-.	Z	--..
C	-.-.	K	-.-	S	...		
D	-..	L	.-..	T	-		
E	.	M	--	U	..-		
F	..-.	N	-.	V	...-		
G	--.	O	---	W	.--		
H	....	P	.--.	X	-..-		

Por meio dos telégrafos iniciaram-se as primeiras redes de comunicação de longa distância, e junto com o crescimento das linhas férreas, possibilitavam mais facilidade no deslocamento das pessoas.

Vídeo - A mensagem de texto nasceu com o telégrafo

O próximo grande passo da comunicação foi com o telefone, criado por *Graham Bell* e utilizado pela primeira vez em 1876, possibilitando a comunicação por voz, em longas distâncias.

Redes de computadores e a internet

> A internet é uma rede de computadores

Muitos anos depois, na década de 60 do século XX e já com computadores eletrônicos, o Departamento de Defesa dos EUA (ARPA), espalhou em torno de 17 computadores em vários locais (para evitar perda de dados caso algum ataque nuclear ocorresse), e os conectou via uma rede telefônica, chamada **ARPANET**.

Anos depois, a rede foi aberta para algumas universidades e empresas, e em 1982 foi adotado o protocolo de comunicação TCP/IP, desenvolvido pela equipe do **UNIX**.

A partir daí, a rede começou a crescer e se tornou a internet que conhecemos hoje.

> Estrutura da internet

Existe uma certa estrutura para que a internet funcione.

O acesso só é possível mediante um intermediário, conhecida como ***provedor de acesso***, o qual conecta um usuário à grande rede.

Quando o usuário faz a conexão ao provedor de acesso, este registra alguns dados, como o endereço IP (quando não for fixo), nome, e data de entrada/saída do usuário.

O provedor de acesso é uma entidade que possuí acesso a outra entidade, denominada ***backbone***, responsável por conectar os provedores entre si. No Brasil quem faz esse papel é a **Embratel**.

Os provedores de acesso à Internet no Brasil

Navegação na internet

Através de um navegador (browser), o usuário pode acessar os mais diversos sites, armazenados em servidores, computadores configurados para disponibilizar conteúdo.

Esse conteúdo são códigos HTML, CSS e JavaScript, interpretados pelo navegador e exibidos ao usuário.

Assim como é possível acessar sites, também é possível que qualquer pessoa registre um site na grande rede, para isso é necessário contratar o serviço de um provedor de **hospedagem**, ex: registro br, que permite a compra de um **domínio**, e registro em um **servidor DNS**.

2. Ferramentas para desenvolvimento de sites

> HTML

É uma linguagem de marcação, que permite a criação de páginas web. A sigla HTML significa **HyperText Markup Language**, ou seja, linguagem de marcação de hipertexto.

Em 1994, a **W3C** (World Wide Web Consortium), fundada por **Tim Berners-Lee**, propôs uma padronização da linguagem HTML. Essa solicitação acabou gerando o lançamento do HML 3.0, em 1995.

Nada mais é que um arquivo de texto, que contém as marcações (**tags**) que serão interpretadas pelo navegador. É considerada uma linguagem de criação de documentos estáticos, por não possuir recursos de ações lógicas.

A versão mais atual do HTML é a 5, lançada em 2008, sendo vista como padrão para criação de páginas web.

O **HTML5** surgiu de uma junção entre o ***HTML4, XML e o XHTML***, com a proposta de ser uma linguagem mais simples, intuitiva, responsiva, que permitisse a criação de páginas mais complexas e acessíveis.

Também foi adicionado suporte melhor para áudio, vídeo e suporte para gráficos vetoriais.

CSS

É uma linguagem de estilização, que permite a criação de regras para estilizar elementos HTML. A sigla CSS significa **Cascading Style Sheets**, ou seja, folhas de estilo em cascata.

Também foi a W3C que apresentou a primeira versão do CSS, em 1996, e era recomendada para formatação de estilos em páginas web.

O **CSS3** foi apresentado em 2001, porém só em 2009 que passou a ser considerado para uso.

CSS permite que a página HTML seja estilizada de maneira eficiente e organizada. Também permite que as páginas se tornem mais responsivas e acessíveis.

> JavaScript

Criada em 1995, por **Brendam Eich** e apresentada pela primeira vez pela empresa Netscape (Atual Mozilla). O **JavaScript** (JS), inicialmente conhecido como *LiveScript*, é uma linguagem de programação baseada na linguagem **C**.

O primeiro navegador a usar códigos em JS foi o Netscape Navigator 2.0, seu objetivo era tornar as páginas web mais dinâmicas, executando ações lógicas por interações do usuário com a página HTML.

Com isso surgiram algumas variações do JS, que geravam divergências de funcionalidades. Para resolver esse problema, a entidade de normalização ECMA criou uma padronização para a linguagem: ECMA-262, com a linguagem ECMAScript que contempla a padronização que deve ser seguida pelos navegadores.

3. Estrutura básica de um documento HTML

> Tags

As tags são **elementos** que compõem um documento HTML, elas são responsáveis por definir a estrutura e o conteúdo da página.

Uma ***tag*** é uma palavra reservada delimitada pelos símbolos "**<**" e "**>**". As tags são geralmente usadas em duplas, uma de abertura e outra de fechamento, ex: `<tag> </tag>`, porém algumas tags não necessitam de fechamento (tags órfãs), ex: `<tag />`.

Grande parte das tags possuem ***atributos***, informações adicionais que podem ser passadas para a tag, ex: `<tag atributo="valor">`.

Estrutura hierárquica

O HTML possui uma estrutura mínima para a formatação de um documento, e deve conter algumas tags básicas para que o navegador possa interpretar o conteúdo.

Para mais informações, temos as docs W3C e MDN.

```html
<!-- Indica ao navegador como
  interpretar o código abaixo  -->
<!DOCTYPE html>
<!-- Bloco principal da página,
  idioma principal -->
<html lang="pt-BR">
  <!-- Agrupa as informações gerais
  (metadados) da página -->
  <head>
    <!-- Indica o tipo de codificação dos caracteres -->
    <meta charset="utf-8" />
    <!-- Texto quando favoritamos -->
    <title>Exibido na aba do navegador</title>
    <!-- Dentro do cabeçalho conseguimos
      carregar arquivos de scripts (JS) -->

    <!-- Também é onde carregamos arquivos,
      como por exemplo css -->
  </head>
  <!-- Agrupa as informações do corpo da página -->
  <body>
    <h1>Cabeçalho de nível 1, título da página</h1>
    <p>Conteúdo da página</p>
  </body>
</html>
```

4. Meta tags e Open graph

> Meta tags

São tags de metadados, com informações sobre o documento HTML, mais utilizadas pelo **navegador** e **mecanismos de busca** google, bing, etc.

Possuem informações sobre o conteúdo da página. No exemplo anterior, utilizamos para informar a codificação dos caracteres.

Open graph

O Open Graph é um protocolo que permite que as páginas HTML sejam interpretadas por redes sociais, como o Facebook, Twitter, LinkedIn, etc. Dessa maneira que redes sociais conseguem exibir prévias sobre o conteúdo da página, como título, descrição, imagem, etc.

5. HTML: Marcação com significado

> Semântica

Semântica é o estudo do significado das palavras, frases e símbolos.

No HTML, a semântica é utilizada para definir o significado dos elementos, ou seja, o que cada elemento representa.

A semântica do ***HTML*** é importante para que o **navegador** e os **mecanismos de busca** consigam interpretar o conteúdo da página, e também torna o conteúdo mais **acessível**

Possibilitando que usuários com deficiência visual, por exemplo, que utilizam **leitores de tela** (exemplo de tecnologia assistiva) para navegar na internet.

Como dito anteriormente, a Web sempre foi pensada para ser acessível a todos.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>UNICESUMAR - Programação WEB 2024</title>
  </head>
  <body>
    <header>
      <h1>Toda página deve possuir um, e apenas um, h1!</h1>
      <img src="https://i.imgur.com/D5lZGis.png" alt="Logo da UNICESUMAR" />
    </header>
    <main>
      <section>
        <h2>
          Semântica bem intuitiva, basta se familiarizar com os elementos html
          (tags)
        </h2>
        <h3>
          A maioria dos browsers já possuem alguns estilos próprios para certos
          elementos, como os títulos e subtítulos
        </h3>
        <p>
          Também existem elementos específicos para
          <em>Itálico (Emphasis element)</em> e também textos em
          <strong> Negrito (Strong element) </strong>
        </p>
      </section>
    </main>
    <footer>
      <p>
        &copy; 2024 Fulano
        <a
          href="https://github.com/satinp"
          title="Github do autor"
          rel="external"
        >
          Âncora que vincula essa página ao meu Github
        </a>
      </p>
    </footer>
  </body>
</html>
```

> Conteúdo de uma página

Quando um navegador renderiza o HTML, ele transforma espaços extras, tabs e quebras de linha em um único espaço.

O HTML é restrito aos caracteres ASCII— letras e símbolos comuns. O que torna diversos caracteres especiais inválidos, como alguns acentos, cedilhas, etc.

Parte desse problema pode ser resolvido utilizando a codificação de caracteres **UTF-8**.

O Unicode utf-8 é um super conjunto do ASCII, que permite a utilização de alguns caracteres especiais.

No exemplo anterior, utilizamos a tag `<meta charset="utf-8" />` para informar ao navegador que o documento está codificado em UTF-8.

Mesmo assim, é comum utilizarmos **entidades HTML** para representar caracteres especiais, por exemplo: `&copy;` para representar o símbolo de direitos autorais ©.

Mais informações podem ser encontradas na documentação da W3C.

Aqui também temos uma Tabela de entidades HTML.

> Elemento âncora

No exemplo do slide anterior também utilizamos a tag `<a>`, um elemento âncora, utilizado para criar links para outras páginas (externos), ou para outras partes da mesma página.

É o elemento responsável por fazer a web ser navegável, o que o torna um dos elementos mais importantes do HTML.

O elemento espera um atributo `href` que indica para onde a tag será direcionada. Podendo referenciar um arquivo local, um link externo, ou um elemento da mesma página.

Exemplos de âncoras.

ARIA

Os ARIA (Accessible Rich Internet Applications) são um conjunto de atributos em HTML que foram introduzidos pelo World Wide Web Consortium (W3C) para melhorar a acessibilidade de aplicativos web ricos e interativos para pessoas com deficiências.

Eles fornecem informações adicionais sobre a estrutura, função e comportamento de elementos HTML, permitindo que tecnologias assistivas, como leitores de tela, interpretem e apresentem corretamente o conteúdo para usuários com deficiências visuais, auditivas ou motoras.

Principais Atributos ARIA:

role (papel): Define o papel ou função semântica de um elemento HTML.

aria-* (propriedades ARIA): Define propriedades específicas relacionadas à acessibilidade de um elemento HTML, como estado, propriedade ou relação.

Por exemplo, o atributo `role="button"` pode ser aplicado a um `<div>` ou `<span>` para indicar que ele funciona como um botão, o que é útil para usuários de leitores de tela que precisam saber a função de um elemento interativo.

```
<div role="button" onclick="myFunction()">Clique aqui</div>
```

Por exemplo, o atributo aria-hidden="true" pode ser usado para elementos ocultados visualmente, mas ainda torná-lo acessível para leitores de tela.

```
<div aria-hidden="true">Indica que o conteúdo é visualmente oculto</div>
```

Mais informações podem ser encontradas na Referência de ARIA no MDN e também na Documentação oficial do W3C sobre ARIA

E alguns exemplos com Padrões ARIA

6. CSS: Estilizando a página

O CSS é uma linguagem de estilização, que permite a criação de regras para estilizar elementos HTML.

A estilização é feita através de **regras** aplicadas a **seletores**, esses indicam quais elementos serão formatados.

As regras são no formato chave e valor. Onde a chave é a propriedade a ser estilizada, e o valor é o estilo a ser aplicado.

> Construindo uma regra de estilo

```css
/* Comentário css */
h1 {
  color: red;
}
```

Onde o h1 é o **seletor**, e o conteúdo do bloco entre `{}` são os conjuntos de **regras**.

Os principais seletores utilizados serão os de **elementos**, **classes** e **ids**.

> Aplicando estilos com seletores de elementos

Conforme o exemplo acima, aplicamos um estilo para o elemento `h1`, onde definimos a cor do texto como vermelho.

Esse estilo será aplicado para todos os elementos `h1` da página.

Porque isso pode ser problemático?

> Aplicando estilos com seletores de classes

Para resolver o problema de estilização de todos os elementos `h1`, podemos utilizar **classes**.

As classes são utilizadas para agrupar elementos que possuem características em comum.

```
<!-- HTML -->
<h1>Título da página</h1>
<h1 class="titulo">Título da seção</h1>
```

Podemos, então, aplicar estilos para a classe `titulo`, que vai ser aplicada apenas para o segundo `h1` da página.

```
/* CSS */
h1 {
  color: red;
}

.titulo {
  color: green;
}
```

> Aplicando estilos com seletores de id

Os **ids** são utilizados para identificar um elemento de maneira única.

```
<!-- HTML -->
<h1>Título da página</h1>
<h1 class="titulo">Título da seção</h1>
<h1 id="outro-titulo">Outro título</h1>
```

```css
/* CSS */
h1 {
  color: red;
}

.titulo {
  color: green;
}

#outro-titulo {
  color: blue;
}
```

Maneiras de aplicar estilos

1. Atributo `style`:

```
<p style="color: red;">Texto do paragrafo vermelho</p>
```

2. Tag `<style>` dentro do `<head>`:

```
<head>
  <style>
    p {
      color: red;
    }
  </style>
</head>
```

3. Arquivo externo:

Podemos criar um arquivo com extensão `.css`, e referenciá-lo no documento HTML, também na tag `<head>`.

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

> Propriedades fundamentais

1. Cores

A versão mais moderna do css permite algumas maneiras de definir cores, como por exemplo: `rgb()`, `rgba()`, `hsl()`, `hsla()`, `hexadecimal`, `nome da cor`.

```
/* Define a cor do texto como um vermelho
    escuro usando um valor hexadecimal */

/* Indo de 00 (mais escuro) at´e FF (mais claro)
    para o valor de cada uma das cores. */
p {
  color: #8b0000;
}
```

```css
/* Define a cor de fundo como verde escuro
  usando valores RGB (Red, Green, Blue) */
div {
  background-color: rgb(0, 100, 0); /* Verde escuro */
}

/* Define a cor de fundo como azul com 50%
  de transparência usando valores RGBA (Alfa)*/
div {
  /* Azul com 50% de transparência */
  background-color: rgba(0, 0, 255, 0.5);
}

/* Define a cor de fundo como um azul claro usando
    valores HSL (Hue, Saturation, Lightness) */
div {
  background-color: hsl(210, 70%, 80%); /* Azul claro */
}
```

2. Margin e Padding

O atributo **margin** é a distância entre o elemento e os elementos vizinhos.

Já o **padding** é a distância entre o conteúdo do elemento e a borda do elemento.

Eles podem ser escritos de algumas maneiras.

- Uniforme:

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

- Individuais:

```
.box {
  margin-top: 10px;
  margin-right: 20px;
  margin-bottom: 15px;
  margin-left: 25px;
}
```

- Horizontais e Verticais:

```
.box {
  /* Margem superior e inferior de 10 pixels,
     margem direita e esquerda de 20 pixels */
  margin: 10px 20px;
}
```

e

```
.box {
  /* Margem superior de 10 pixels,
     margem direita e esquerda de 20 pixels,
     margem inferior de 15 pixels */
  margin: 10px 20px 15px;
}
```

- Alternadas:

```
.box {
  /* Margem superior de 10 pixels,
     margem direita de 20 pixels,
     margem inferior de 15 pixels,
     margem esquerda de 25 pixels */
  margin: 10px 20px 15px 25px;
}
```

Importante: A dimensão apropriada para a propriedade padding ou margin pode ser definida automaticamente pelo navegador mediante uso do valor auto.

Seletores de atributos

```
/* Seleciona todos os elementos <a> com o atributo
  "href" que contém "https://" */
a[href*="https://"]
{
  color: green;
}

/* Seleciona todos os elementos <a> com o atributo
  "title" */
a[title] {
  color: blue;
}
```

Documentação MDN sobre Seletores de atributos

Pseudo classes

```
/* Aplica o estilo quando elementos <a> estiverem
  em hover (mouse sobre o elemento) */
a:hover {
  color: green;
}

button:disabled {
  color: gray;
}
```

Documentação MDN sobre Pseudo Classes

Pseudo elementos

```css
/* Aplica estilo a primeira letra dos <p> */
p::first-letter {
  color: blue;
  text-transform: uppercase;
}

/* Aplica a cor laranja aos marcadores
  de cada list item */

li::marker {
  color: orange;
}
```

Documentação MDN sobre Pseudo Elementos

Combinadores

> Combinadores de filhos

Representado pelo sinal `>`, é colocado entre dois seletores. O estilo será aplicado apenas aos elementos que são filhos diretos do primeiro seletor.

```
/* Aplica estilo apenas aos elementos <span> que são
  filhos diretos de <p> */
p > span {
  color: red;
}
```

Combinadores de irmãos

Representado pelo sinal `+`, é colocado entre dois seletores. O estilo será aplicado apenas aos elementos que são irmãos imediatos do primeiro seletor.

```
/* Aplica estilo apenas aos elementos <span> que são
  irmãos imediatos de <p> */
p + span {
  background-color: lightgrey;
}
```

Combinadores de irmãos gerais

Representado pelo sinal `~`, é colocado entre dois seletores. O estilo será aplicado a todos os elementos que são irmãos do primeiro seletor.

```
/* Aplica estilo a todos os elementos <span> que são
  irmãos de <p> */

p ~ span {
  color: blue;
}
```

É possível utilizar todos os combinadores em conjunto.

```
/* Aplica estilo a todos os elementos <span> que são
  irmãos de <p> e filhos diretos de <div> */

div > p ~ span {
  color: blue;
}
```

Documentação MDN sobre Combinadores css

> Algumas das principais unidades de medidas css

Unidade	Descrição
`px`	Pixels
`em`	Relativo ao tamanho da fonte do elemento pai
`rem`	Relativo ao tamanho da fonte do elemento raiz (root)
`vw`	Relativo à largura da viewport (1vw = 1% view width)
`vh`	Relativo à altura da viewport (1vh = 1% view height)
`%`	Percentual

O modelo de caixas (box model)

Tudo no CSS possuí uma caixa ao seu redor, e cada caixa possui quatro áreas: **conteúdo**, **padding**, **border** e **margin**.

Temos vários tipos de caixas, sendo que essas geralmente sem encaixam na categoria de **caixas em bloco** (block boxes) e as **caixas em linha** (inline blocks). O tipo vai definir como a caixa se comporta na página e em relação as outras caixas.

> Caixas Block

As caixas em bloco são caixas que **ocupam toda a largura disponível** e **são empilhadas uma em cima da outra**.

As propriedades **width** e **height** são aplicadas a essas caixas, e elas **podem** ter **margin**, **padding** e **border**.

A menos que sejam alterados manualmente, alguns elementos HTML são caixas em bloco por padrão, como `<div>`, `<p>`, `<h*>`, `<main>`, `<nav>`, `<header>`, `<article>`, `<footer>`, etc.

Caixas Inline

As caixas em linha são caixas que não ocupam toda a largura disponível, são exibidas uma ao lado da outra.

As propriedades **width** e **height** não são aplicadas a essas caixas, **margin** e padding são aplicadas apenas horizontalmente.

Elementos HTML como `<span>`, `<a>`, `<strong>`, `<em>`, `<small>`, `<button>`, etc, são caixas **inline** por padrão.

Tipos de exibição outer (externos) e inner (internos)

As caixas **inline** e **block** são exemplos de caixas do tipo **outer**. Pois ditam como os outros elementos se comportam em relação a elas.

As caixas também possuem um tipo de exibição **inner**, indicando como os elementos na caixa se comportam. Por padrão os elementos dentro de uma caixa são posicionados em um fluxo normal (normal flow), significando que se comportam como caixas em bloco ou em linha.

É possível alterar o tipo de exibição inner utilizando a propriedade **display**.

Como por exemplo, a propriedade **display: flex** transforma a caixa em um **flex container**, o "outer display" continuará se comportando como os tipos "block", mas todos os filhos diretos da caixa se comportarão como **flex items**, seguindo as regras do **flexbox**.

Flexbox e Grid Layout

> Flexbox

Flexbox é um modelo de layout que permite o design de layout responsivo e previsível, sem depender de tamanhos específicos de tela, trabalha de uma maneira unidimensional, ou seja, ele lida com um eixo por vez, seja ele o eixo horizontal ou vertical.

Para adicionar o flex container a um elemento, basta adicionar a propriedade `display: flex`.

Algumas de suas principais propriedades são:

1. **flex-direction**: Define a direção principal do flex container, podendo ser `row` (padrão), `row-reverse`, `column` e `column-reverse`. Essa propriedade também **altera** qual será **o eixo principal**.

2. **flex-wrap**: Define se os itens devem ser quebrados em várias linhas, podendo ser `nowrap` (padrão), `wrap` e `wrap-reverse`.

3. **justify-content**: Define o alinhamento dos itens ao longo do **eixo principal**, podendo ser `flex-start` (padrão), `flex-end`, `center`, `space-between`, `space-around` e `space-evenly`.

4. **align-items**: Define o alinhamento dos itens ao longo do eixo transversal, podendo ser `stretch` (padrão), `flex-start`, `flex-end`, `center` e `baseline`.

5. **align-content**: Define o alinhamento dos itens ao longo do eixo transversal quando há mais de uma linha, podendo ser `stretch` (padrão), `flex-start`, `flex-end`, `center`, `space-between` e `space-around`.

Documentação MDN - Flexbox

CSS Tricks - Complete Guide to Flexbox

Todo container flex terá elementos filhos que se comportarão como **flex items**, e esses elementos também possuem propriedades específicas:

1. **flex-grow**: Define a capacidade de um item de crescer em relação aos outros itens do container. Ex: **Se existem 3 itens, um possui `flex-grow: 1`, e os outros 2 possuem `flex-grow: 2`, o primeiro item ocupará 1/3 do espaço disponível, e os outros 2 ocuparão 2/3.**

2. **flex-shrink**: Define a capacidade de um item de encolher em relação aos outros itens do container. Ex: **Se existem 3 itens, um possui `flex-shrink: 1`, e os outros 2 possuem `flex-shrink: 2`, o primeiro item encolherá 1/3 do espaço disponível, e os outros 2 encolherão 2/3.**

3. **flex-basis**: Define o tamanho inicial de um item.

4. **order**: Define a ordem de um item em relação aos outros itens do container. Ex: **Se existem 3 itens, e o primeiro possui order: 1, o primeiro item será exibido por último.**

5. **align-self**: Define o alinhamento de um item em relação aos outros itens do container. É igual o align-items, mas aplicado (sobrescreve) apenas a um item.

Treina WEB - Guia interativo Flex Items

> Grid

O Grid Layout é um sistema de layout bidimensional, ou seja, ele lida com dois eixos, o eixo horizontal e o eixo vertical.

Para adicionar o grid container a um elemento, basta adicionar a propriedade `display: grid`.

Algumas de suas principais propriedades são:

1. **grid-template-columns**: Define o número e o tamanho das colunas do grid, podendo ser definido em pixels, porcentagem, fr, etc.

Ex: `grid-template-columns: 100px 200px 300px;` que define 3 colunas de tamanhos diferentes.

A unidade `fr` segue a mesma lógica do `flex-grow`, onde o espaço disponível é dividido entre as colunas de acordo com o valor de `fr`.
`grid-template-columns: 1fr 2fr 3fr;` define 3 colunas de tamanhos diferentes, onde a primeira coluna ocupará 1/6 do espaço disponível, a segunda 2/6 e a terceira 3/6.

2. **grid-template-rows**: Define o número e o tamanho das linhas do grid, podendo ser definido em pixels, porcentagem, fr, etc.

Ambas as propriedades podem ser definidas de maneira mais dinâmica, utilizando a função `repeat()`. Também podem receber múltiplas unidades de medidas. Ex:

`grid-template-columns: repeat(2, 50px) 1fr 2fr 100px;` define 5 colunas, onde as duas primeiras possuem 50px, a terceira e quarta 1/3 e 2/3 respectivamente do espaço restante, e a quinta 100px.

3. **grid-template-areas**: Define um template de áreas para o grid, onde cada área é nomeada, e os elementos filhos do grid podem ser posicionados nessas áreas. Ex:

```
grid-template-areas:
  'header header header'
  'nav main main'
  'nav footer footer';
```

Onde o grid terá 3 linhas e 3 colunas, e os elementos filhos podem ser posicionados nas áreas definidas através da propriedade `grid-area`.

Codepen - Grid teplate areas

Transições e Animações

> Transições

As transições são utilizadas para criar efeitos/animações de transição entre dois estados de um elemento, por exemplo, mudar a cor de um botão quando o mouse passa por cima dele.

Ela pode ser aplicada a uma propriedade específica, ou a todas as propriedades. Podemos definir a duração, o tempo de espera, a função de tempo, e o atraso da transição.

```
div {
  transition: <property> <duration> <timing-function> <delay>;
}
```

MDN - CSS Grid Layout

Documentação MDN - Transições

Animações

As animações são utilizadas para criar efeitos mais complexos, onde é possível definir vários estados intermediários, e também é possível definir a duração, o tempo de espera, a função de tempo, e o atraso da animação.

São utilizadas através da propriedade `@keyframes`, onde definimos algo como uma função, que controla os estados intermediários da animação.

```css
@keyframes spin {
  0% {
    transform: rotate(0deg);
  }

  100% {
    transform: rotate(360deg);
  }
}
```

```
div {
  animation: <duration> <name> <iteration-count> <direction>;
}
```

Onde a propriedade `animation` recebe a duração, o nome da animação, a quantidade de vezes que a animação será executada, e a direção da animação.

Documentação MDN - Animações

Codepen spinner

Media queries

São responsáveis por aplicar estilos diferentes conforme as características do dispositivo que está exibindo a página, por exemplo, a largura da tela.

É um elemento chave para design responsivo, onde o layout da página se adapta conforme o dispositivo que está exibindo a página.

É muito comum utilizar media queries para definir estilos diferentes para dispositivos com telas menores, como smartphones e tablets, baseado na largura da viewport

```css
/* Define um estilo para telas menores que 600px */
@media (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

```css
/* Define um estilo para telas maiores que 600px */
@media (min-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

É possível utilizar várias media queries em conjunto, e também é possível utilizar operadores lógicos como `and`, `or` e `not`.

```css
/* Define um estilo para telas maiores que 600px e menores que 800px */
@media (min-width: 600px) and (max-width: 800px) {
  body {
    background-color: lightblue;
  }
}
```

Existem alguns breakpoints comuns que são utilizados para design responsivo, como por exemplo:

```
/* Extra small devices (phones, 600px and down) */
@media (max-width: 600px) {
}

/* Small devices (portrait tablets and large phones, 600px and up) */
@media (min-width: 600px) {
}

/* Medium devices (landscape tablets, 768px and up) */
@media (min-width: 768px) {
}

/* Large devices (laptops/desktops, 992px and up) */
@media (min-width: 992px) {
}

/* Extra large devices (large laptops and desktops, 1200px and up) */
@media (min-width: 1200px) {
}
```

Links úteis sobre media queries

W3Schools - Media query breakpoints

MDN - Guia de media queries

MDN - Usando media queries

Navegação interna de páginas

Utilizando a tag `<a>`, podemos criar links para páginas internas.

Por exemplo, criando outro arquivo html, e referenciando ele no atributo `href`.

```
<a href="pagina2.html">Ir para a página 2</a>
```

Ao clicar no link, o navegador irá carregar a página `pagina2.html` e todo seu conteúdo.

JavaScript

O JavaScript é como se fosse o cérebro de uma página web, ele é responsável por adicionar interatividade e dinamismo a uma página.

É a linguagem de programação mais utilizada na web, sendo suportada por todos os navegadores modernos.

Através do JavaScript é possível manipular o conteúdo da página, adicionar ou remover elementos, alterar estilos, criar animações, etc.

Para adicionar JavaScript a uma página, basta adicionar a tag `<script>` no cabeçalho ou no corpo da página.

Normalmente é utilizado no final do corpo da página, para que o JavaScript seja carregado após o conteúdo da página, pois carregar o JavaScript no cabeçalho pode bloquear o carregamento do site e impactar o desempenho.

É possível adicionar o código diretamente na tag `<script>`, ou referenciar um arquivo externo. Assim como o CSS.

MDN Script element

Utilizando com a tag script:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
    <script>
      alert('Olá, mundo!')
    </script>
  </head>
  <body>
    <h1>JavaScript</h1>
  </body>
</html>
```

Referencia a um arquivo externo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
    <script src="script.js"></script>
  </head>
  <body>
    <h1>JavaScript</h1>
  </body>
</html>
```

Carregando scripts externamente é uma boa prática, pois podemos reutilizar código, separar a lógica do design, facilita a manutenção e também podem ser cacheados pelo navegador. Um script externo também pode referenciar urls, como, por exemplo, a url de uma biblioteca, ex: jQuery.

Com o JavaScript conseguimos manipular o DOM, que é a estrutura de uma página web, e também podemos manipular estilos, eventos, etc.

Através do JavaScript temos acesso aos atributos globais do navegador, como o `window`, `document`, `navigator`, `screen`, `location`, `history`, `localStorage`, `sessionStorage`, etc.

Para acessar um elemento do DOM, podemos utilizar o método `document.querySelector()`, que retorna o primeiro elemento que corresponde ao seletor especificado.

```
// Seleciona o primeiro elemento <h1> da página
const h1 = document.querySelector('h1')

// Altera a cor do texto do h1 para vermelho
h1.style.color = 'red'
```

Também conseguimos acessar elementos através de classes e ids, utilizando `document.querySelector('.classe')` e `document.querySelector('#id')`.

Através da api `document` podemos criar elementos, adicionar ou remover elementos, alterar estilos, etc. Instance methods

```
// Cria um novo elemento <p>
const p = document.createElement('p')

// Adiciona um texto ao elemento <p>
p.textContent = 'Texto do parágrafo'

// Adiciona o elemento <p> ao final do body
document.body.appendChild(p)

// Remove o elemento <h1> da página
document.querySelector('h1').remove()

// Adiciona o p a uma div
document.querySelector('div').appendChild(p)

// Adiciona uma classe ao elemento <p>
p.classList.add('paragrafo')

// Adiciona um estilo ao elemento <p>
p.style.color = 'blue'
```

Com o dom também é possível adicionar eventos a elementos, por exemplo, um evento de clique, um evento de mouse sobre o elemento, um evento de teclado, etc.

Usando a DOM

```
// Adiciona um evento de clique ao elemento <p>
p.addEventListener('click', () => {
  alert('Clicou no parágrafo')
})

// Adiciona um evento ao passar o mouse sobre o elemento <p>
p.addEventListener('mouseover', () => {
  alert('Passou o mouse sobre o parágrafo')
})

// Adiciona um evento ao pressionar uma tecla
document.addEventListener('keydown', (event) => {
  console.log(event.key)
})
```

Para criação de lógica ou eventos que envolvam a manipulação do DOM, é importante que o JavaScript seja carregado após o conteúdo da página, para garantir que os elementos já estejam disponíveis.

Para isso, é comum adicionar o script no final do corpo da página, ou utilizar o evento `DOMContentLoaded`.

```
document.addEventListener('DOMContentLoaded', () => {
  // Código JavaScript
})
```

Também podemos acrescentar eventos js diretamente no html, utilizando atributos de funções que serão chamadas quando o evento ocorrer, que referenciam funções js ou que são códigos js.

```
function myFunction() {
  alert('Clicou no botão')
}
```

```
<button onclick="myFunction()">Clique aqui</button>

<button onclick="this.innerHTML = Date().toLocaleDateString()">
  Qual a data atual?
</button>
```

Tutorial JavaScript W3Schools

APIS:

Window, Document, Navigator, Location, History, Local storage, Session storage

Formulários

Os formulários são uma parte importante de uma página web, eles são utilizados para coletar informações dos usuários.

Permite que dados sejam inseridos, como texto, números, datas, arquivos, seleções, etc.

Temos diversos tipos de campos de formulários, como `<input>`, `<textarea>`, `<select>`, `<button>`, `<label>`, dentre outros.

Os inputs são os elementos mais comuns de um formulário, e possuem diversos tipos, como `text`, `password`, `email`, `number`, `date`, `file`, `checkbox`, `radio`, etc. Eles são definidos através do atributo `type`.

```
<!-- Para entrada de texto longo (descrição, comentário). -->
<input type="textarea" />
```

Além do type, também existem outros atributos importantes para os inputs, como `name`, `value`, `placeholder`, `required`, `disabled`, `readonly`, `min`, `max`, etc.

O atributo name é utilizado para identificar o campo, e é o nome que será enviado junto aos dados formulário for submetido.

```
<input type="textarea" name="descricao" />
```

O atributo value é um atributo opcional que define o valor inicial do campo. É obrigatório para inputs do tipo `checkbox` e `radio`. Pode ser utilizado via javascript para definir ou obter o valor do campo.

```
<input type="textarea" name="descricao" value="Sua descricao" />
```

```
document.querySelector('input[name="descricao"]').value = 'Nova descrição'
```

Todo formulário possui uma estrutura básica, que é composta por um elemento `<form>`, que é o container do formulário, e os elementos de input, botões, etc.

```
<!-- Formulário que envia os dados para /submit
     ao clicar no botão "Enviar" -->
<form name="meu-form" action="/rota/submit">
  <input type="text" name="nome" />
  <input type="email" name="email" />
  <button type="submit">Enviar</button>
</form>
```

O button com type `submit` indica que o formulário será submetido ao clicar no botão. O atributo `action` define a rota para onde os dados do formulário serão enviados.

Também é possível utilizar o atributo `method` para definir o método de envio dos dados, que pode ser `get` ou `post`.

```
<!-- Formulário que envia os dados para /submit
     ao clicar no botão "Enviar" -->
<form name="meu-form" action="/rota/submit" method="post">
  <input type="text" name="nome" />
  <input type="email" name="email" />
  <button type="submit">Enviar</button>
</form>
```

Agora será chamada a rota `/submit` com o método `post` ao submeter o formulário.

Para páginas estáticas, podemos optar por não utilizar o atributo `action`, e utilizar o atributo `onsubmit` para definir uma função que será chamada ao submeter o formulário.

```
<!-- Formulário que chama a função "enviar"
     ao clicar no botão "Enviar" -->
<form name="meu-form" onsubmit="enviar()">
  <input type="text" name="nome" />
  <input type="email" name="email" />
  <button type="submit">Enviar</button>
</form>
```

```
function enviar() {
  alert('Formulário enviado')
}
```

Conforme demonstrado nos exemplos acima, o elemento `button` tem um papel importante nos formulários e o atributo type define o comportamento do botão.

O type `submit` é utilizado para submeter o formulário, o type `reset` é utilizado para limpar os campos do formulário, e o type `button` é utilizado para botões que não submetem o formulário. **O type submit também é disparado ao pressionar a tecla `enter` em um campo de texto.**

Outro elemento importante para os formulários é o label. Nos exemplos acima podemos notar que os inputs não possuem um texto associado, o que pode ser ruim para a acessibilidade e usabilidade.

Podemos definir uma label para um input utilizando o atributo `for` do label, que recebe o id do input. Formando uma associação entre o label e o input. Dessa maneira temos uma página mais acessível e com uma melhor usabilidade. Vide que ao clicar no texto do label, o input é selecionado, e leitores de tela conseguem identificar a associação.

```
<!-- Formulário com labels associadas aos inputs -->
<form name="meu-form" onsubmit="enviar()">
  <label for="nome">Nome</label>
  <input type="text" name="nome" id="nome" />

  <label for="email">Email</label>
  <input type="email" name="email" id="email" />

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

Podemos também utilizar o atributo `placeholder` para adicionar um texto de exemplo ao input, que será exibido quando o input estiver vazio. Indicando ao usuário o que deve ser inserido no campo, ou até mesmo o formato que deve ser seguido, com um exemplo.

```
<!-- Formulário com placeholders nos inputs -->
<form name="meu-form" onsubmit="enviar()">
  <label for="nome">Nome</label>
  <input type="text" name="nome" id="nome" placeholder="Digite seu nome" />

  <label for="email">Email</label>
  <input type="email" name="email" id="email" placeholder="email@email.com" />

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

Outro atributo comum é o `required`, que é utilizado para definir que um campo é obrigatório, e o formulário não pode ser submetido se o campo estiver vazio. Ao tentar submeter o formulário com um campo obrigatório vazio, o navegador exibirá uma mensagem de erro, informando que o campo precisa ser preenchido.

```html
<!-- Formulário com campos obrigatórios -->
<form name="meu-form" onsubmit="enviar()">
  <label for="nome">Nome</label>
  <input
    type="text"
    name="nome"
    id="nome"
    placeholder="Digite seu nome"
    required
  />

  <label for="email">Email</label>
  <input
    type="email"
    name="email"
    id="email"
    placeholder="email@email.com"
    required
  />

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

Outro atributo comum é o `disabled`, que é utilizado para desabilitar um campo, impedindo que o usuário interaja com ele. O campo desabilitado não será enviado junto com os dados do formulário.

```
<!-- Formulário com campos desabilitados -->
<input
  type="text"
  name="nome"
  id="nome"
  placeholder="Digite seu nome"
  disabled
/>
```

Customizando formulários

```css
/* Estilo padrão para os inputs */
input {
  padding: 10px;
  margin: 10px;
  border: 1px solid #ccc;
}

/* Estilo para os inputs válidos */
input:valid {
  border-color: green;
}

/* Estilo para os inputs inválidos */
input:invalid {
  border-color: red;
}

/* Estilo para os inputs em foco */
input:focus {
  border-color: blue;
}

/* Estilo para os inputs desabilitados */
input:disabled {
  background-color: lightgrey;
}
```

Adicionando lógica ao formulário

Por padrão os formulários recarregam a página ao serem submetidos, para evitar esse comportamento, podemos utilizar o método `preventDefault()` do evento, que previne o comportamento padrão do evento. Primeiro temos que passar o evento como parâmetro da função.

```
<form name="meu-form" onsubmit="enviar(event)">[...]</form>
```

```
function enviar(e) {
  e.preventDefault()
}
```

Podemos acessar os dados do formulário através do objeto `FormData`, que é um objeto que permite a criação de pares chave/valor a partir de um formulário.

```
function enviar(e) {
  e.preventDefault()

  const form = e.target
  const formData = new FormData(form)

  console.log(formData.get('nome'))
  console.log(formData.get('email'))
  console.log(formData.get('email'))
}
```

Referências:

Livros:

- HTML5 e CSS3: guia prático e visual - Elizabeth Castro / Bruce Hyslop
- Guia de orientação e desenvolvimento de sites: HTML, XHTML, CSS e Javascript/Jscript - José A. Manzano / Suely Alves de Toledo

Sites/Documentações:

- W3C, MDN, W3Schools, Can I use, Open Graph.