



FUNDAMENTOS DA PROGRAMAÇÃO WEB

AULA 4



Profª Margarete Klamas



CONVERSA INICIAL

Vamos nos aprofundar em HTML 5 e CSS 3.

O objetivo aqui é apresentar um aprofundamento em HTML 5, com os elementos estruturais semânticos. Em CSS, veremos as possibilidades em termos de desenvolvimento de layout e imagem responsiva. Ao concluir os cinco tópicos propostos, você será capaz de desenvolver a diagramação de uma aplicação, bem como poderá fazer alterações em páginas criadas por outros desenvolvedores.

No tópico 1, abordaremos elementos estruturais semânticos. No tópico 2, estudaremos grid. No tópico 3, veremos como utilizar flexbox. No tópico 4, veremos media queries e breakpoints. No tópico 5, veremos as soluções para aplicarmos responsividade em imagens.

Bons estudos!

TEMA 1 – ELEMENTOS ESTRUTURAIS (SEMÂNTICOS)

Já utilizamos, em alguns exemplos, o elemento `<div>`. Uma das características da `<div>` é ser do tipo bloco e ser um container genérico para outros elementos. Não possui uso específico, ou seja, diferentemente dos elementos `<h2>` e `<hr>`, por exemplo, que se destinam a marcar cabeçalhos de nível 2 e linha horizontal, esse elemento pode conter qualquer(qualquer) outro(s) elemento(s) e até mesmo elementos `<div>`. Trata-se de um elemento sem valor semântico. Semântica é um aspecto relevante em marcação HTML e orienta ao uso correto do elemento para marcar o conteúdo. O uso dos elementos HTML em conformidade com sua função é uma boa prática em HTML.

Ao diagramarmos páginas web, costumeiramente colocamos os conteúdos em regiões. A versão 5 do HTML criou regiões específicas para cada tipo de conteúdo. A seguir, veremos os elementos e as especificações do W3C¹.

1.1 Elemento `<header>`

O elemento `<header>` representa um grupo de ajudas introdutórias ou de navegação. Pode conter o cabeçalho da seção (um elemento `h1–h6`) ou um

¹ O Consórcio World Wide Web (W3C) é uma comunidade internacional que desenvolve padrões com o objetivo de garantir o crescimento da web. Disponível em: <https://www.w3c.br/>. Acesso em: 29 jan. 2023.



hgroup, mas isso não é obrigatório. Em geral, esses elementos contêm logotipos e caixas de busca.

1.2 Elemento `<section>`

O elemento `<section>` representa uma seção genérica de um documento ou aplicativo. Uma seção, neste contexto, é um agrupamento temático de conteúdo, normalmente com um título. A página inicial de um site pode ser dividida em seções para introdução, notícias e informações de contato. Não é um contêiner genérico. Quando um elemento é necessário para fins de estilo ou como uma conveniência para script, os autores são incentivados a usar o elemento `<div>`. Uma regra geral é que o `<section>` é apropriado apenas se o conteúdo do elemento for listado explicitamente na estrutura de tópicos do documento.

1.3 Elemento `<article>`

O elemento `<article>` representa uma composição independente em um documento, página, aplicativo ou site e que é, em princípio, distribuível independentemente ou reutilizável, por exemplo, em distribuição. Pode ser uma postagem no fórum, um artigo de revista ou jornal, uma entrada de blog, um comentário enviado por um usuário ou qualquer outro item de conteúdo independente.

1.4 Elemento `<nav>`

O elemento `<nav>` representa uma seção de uma página com links para outras páginas ou partes da página: uma seção com links de navegação. Nem todos os grupos de links em uma página precisam estar em um elemento `<nav>` — o elemento destina-se principalmente a seções que consistem em blocos de navegação principais.

1.5 Elemento `<aside>`

O elemento `<aside>` representa uma seção de uma página que consiste em conteúdo tangencialmente relacionado ao conteúdo ao redor do elemento `<aside>` e que pode ser considerado separado desse conteúdo. Essas seções são frequentemente representadas como barras laterais na tipografia impressa.



1.6 Elemento <footer>

O elemento <footer> representa um rodapé para seu conteúdo de corte ancestral mais próximo ou elemento raiz. Um rodapé normalmente contém informações sobre sua seção, como quem o escreveu, links para documentos relacionados, dados de direitos autorais e similares. Os rodapés não precisam necessariamente aparecer no final de uma seção, embora geralmente apareçam.

1.7 Elemento <hgroup>

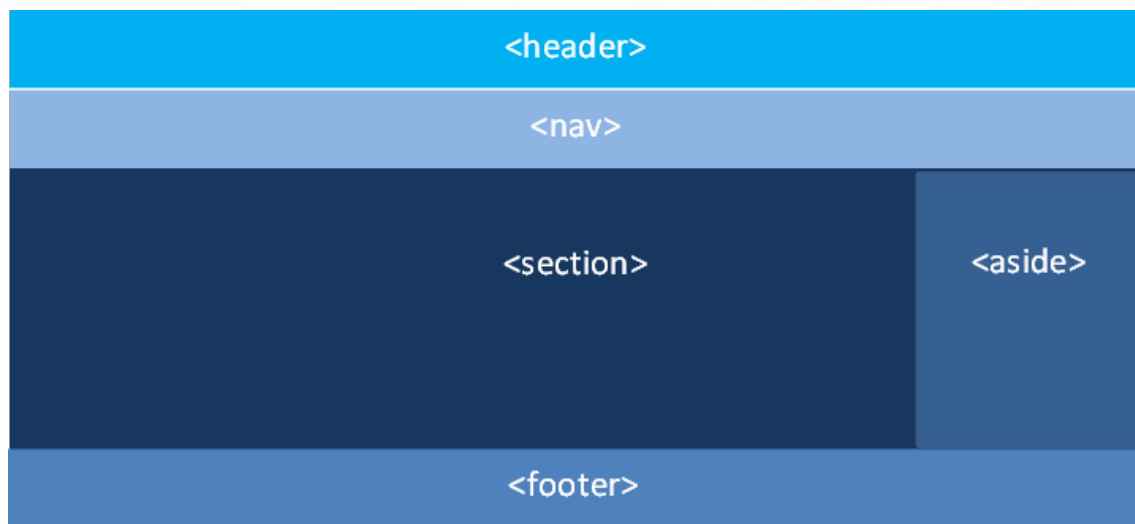
O elemento <hgroup> representa o título de uma seção. O elemento é usado para agrupar um conjunto de elementos h1– h6 quando o título tem vários níveis, como subtítulos, títulos alternativos ou slogans.

```
<hgrupo>  
  <h1> Título 1 </h1>  
  <h2> Título 2</h2>  
</hgroup>
```

1.8 Elemento <div>

O elemento <div> não tem nenhum significado especial. Representa seus filhos. Ele pode ser usado com os atributos class, id, para marcar a semântica comum a um grupo de elementos consecutivos. Os autores são fortemente encorajados a ver o elemento <div> como um elemento de último recurso, para quando nenhum outro elemento for adequado. O uso do elemento <div>, em vez de elementos mais apropriados, leva à baixa acessibilidade para os leitores e à baixa manutenção para os autores.

Considerando os elementos semânticos, podemos nomear as regiões:



TEMA 2 – GRID

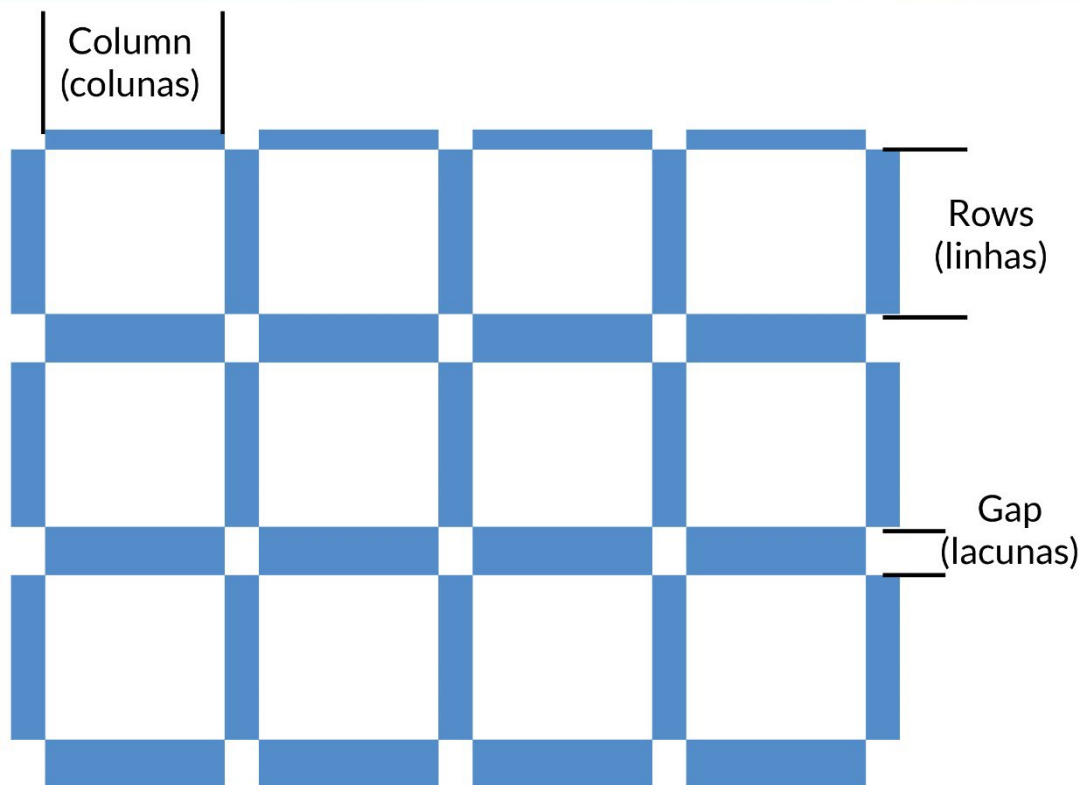
CSS apresenta várias propriedades para auxiliar no posicionamento de elementos, como floats, clear e display, elementos sem valor semântico em suas finalidades — e, pode-se dizer, um pouco confusos de se usar na hora de diagramar. Diagramar um layout pode se tornar uma tarefa que requer paciência.

Baseando-se na mídia impressa, utiliza grades como técnica de diagramação e, partindo dessa experiência, foram desenvolvidas funcionalidades de Grid Layout. Assim, pode-se dizer que surgiu uma possibilidade fantástica para criar layout CSS com grids.

O W3C define grid assim:

CSS Grid Layout é um sistema para criação de layout baseado em uma grade bidimensional e otimizado para design de interfaces de usuário. Nesse modelo de grade, os elementos-filhos do container que define a grade podem ser posicionados livremente em espaços criados na grade, quer ela tenha sido definida com suas dimensões flexíveis ou fixas.

O layout baseado em grid é baseado em colunas e linhas, que facilitam o design sem a necessidade de utilizar floats e posicionamento. O layout grid possui linhas, colunas e gap. Observem a representação do Grid:



Observe, na imagem a seguir, como podemos aplicar o grid em layouts. A tela foi dividida em 12 colunas com partes iguais, e foi definido que o `<header>` e o `<footer>` ocupariam as 12 colunas. O `<nav>` ocupa 2 colunas e o `<aside>` três colunas.

<header>											
1	2	3	4	5	6	7	8	9	10	11	12
<nav>										<aside>	
<footer>											



2.1 Elementos de grid

É possível definir um elemento pai, com um ou mais elementos filhos. Um elemento se torna um contêiner grid se sua propriedade `display` é definida como `grid` ou `inline-grid`.

grid container

Ao utilizar o layout com grid, deve-se declarar essa propriedade por primeiro.

```
.container {display: grid; }
```

grid lines

São os eixos verticais e horizontais que constituem o grid. Em CSS, por padrão, os eixos são nomeados por números sequenciais, iniciando em um, da esquerda para direita, horizontalmente, e de cima para baixo, verticalmente.

grid-template-columns

Utilizado para definir um nome para as grid lines verticais e a largura das colunas do grid.

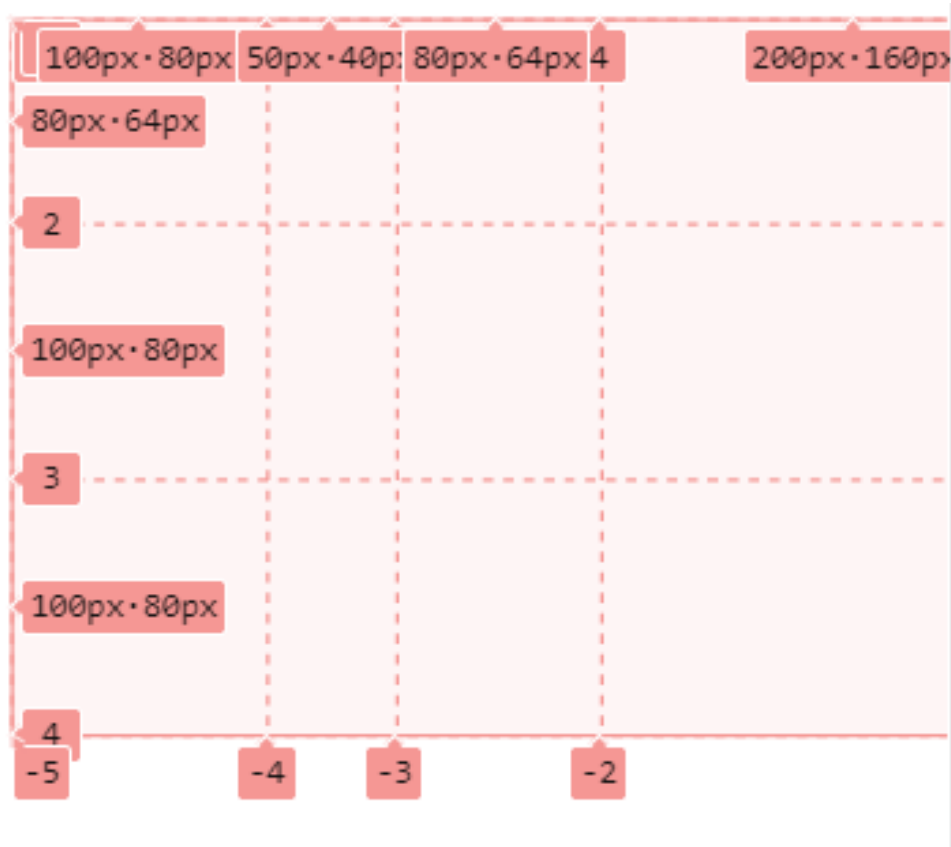
```
.container {  
    display: grid;  
    grid-template-columns: 100px 50px 80px 200px;  
    /* cria um grid com 4 colunas de largura fixa */  
}
```

grid-template-rows

Utilizado para definir um nome para as grid lines horizontais e a altura das linhas do grid. No exemplo a seguir, especificamos um grid com três linhas e altura fixas. O exemplo é literal.

```
.container {  
    display: grid;  
    grid-template-columns: 100px 50px 80px 200px;  
    grid-template-rows: 80px 100px 100px;  
}
```

Visualização do código acima (literal)



Gap

Cria um espaçamento (*gap*, em inglês) entre as colunas e/ou linhas do grid. Gap é a forma abreviada de se declararem as propriedades `grid-column-gap` e `grid-row-gap`. Exemplos:

```
grid-row-gap: 10px;
```

```
grid-column-gap: 20px;
```

```
gap: 10px 20px; /* equivalente às duas declarações anteriores */
```

```
gap: 8px; /* espaçamento igual tanto para colunas como para linhas */
```

```
gap: 0 8px; /* espaçamento somente para colunas */
```

Na declaração abreviada utilizando `gap`, o primeiro valor é para `grid-row-gap` e o segundo para `grid-column-gap`.

grid-item

São elementos filho de um grid container. No exemplo a seguir, as classes são os grids item.



CSS:

```
.box { background: #FFF; border: 4px solid rgb(16, 1, 1);  
border-color: #000; color: #000;}  
.container {  
    display: grid;  
    grid-template-columns: 100px 50px 80px 200px;  
    grid-template-rows: 80px 100px 100px;  
}
```

HTML:

```
<div class="container">  
    <div class="box">1</div>  
    <div class="box">2</div>  
    <div class="box">3</div>  
    <div class="box">4</div>  
    <div class="box">5</div>  
    <div class="box">6</div>  
    <div class="box">7</div>  
    <div class="box">8</div>  
    <div class="box">9</div>  
</div>
```

Visualização do grid (3 linhas x 4 colunas), sem gap:

1	2	3	4
5	6	7	8
9			



Visualização do grid (3 linhas x 4 colunas), com gap (8px):

CSS:

```
.box { background: #FFF; border: 4px solid rgb(16, 1, 1);  
border-color: #000; color: #000; }  
.container {  
  display: grid;  
  grid-template-columns: 100px 50px 80px 200px;  
  grid-template-rows: 80px 100px 100px;  
  gap: 8px; }
```

HTML:

```
<div class="container">  
  <div class="box">1</div>  
  <div class="box">2</div>  
  <div class="box">3</div>  
  <div class="box">4</div>  
  <div class="box">5</div>  
  <div class="box">6</div>  
  <div class="box">7</div>  
  <div class="box">8</div>  
  <div class="box">9</div>  
</div>
```

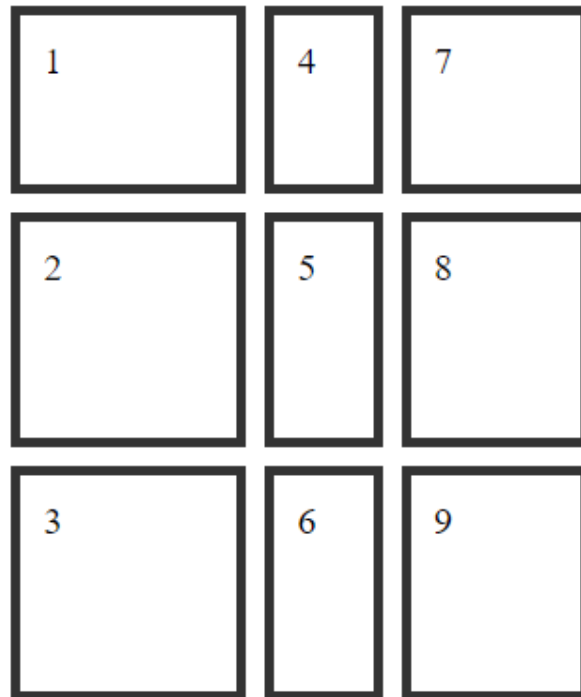


Podemos utilizar qualquer outro elemento estrutural como container e como grid item.

O preenchimento das colunas ocorre no fluxo de linhas, ou seja, preenche primeiro as linhas, conforme observamos nos exemplos acima. Caso queira que



a ordem dos grids itens seja por coluna, pode utilizar a declaração: `grid-auto-flow: column;`



grid-template

Essa propriedade é a forma abreviada de se declararem as propriedades `grid-template-rows` e `grid-template-columns`, nessa ordem.

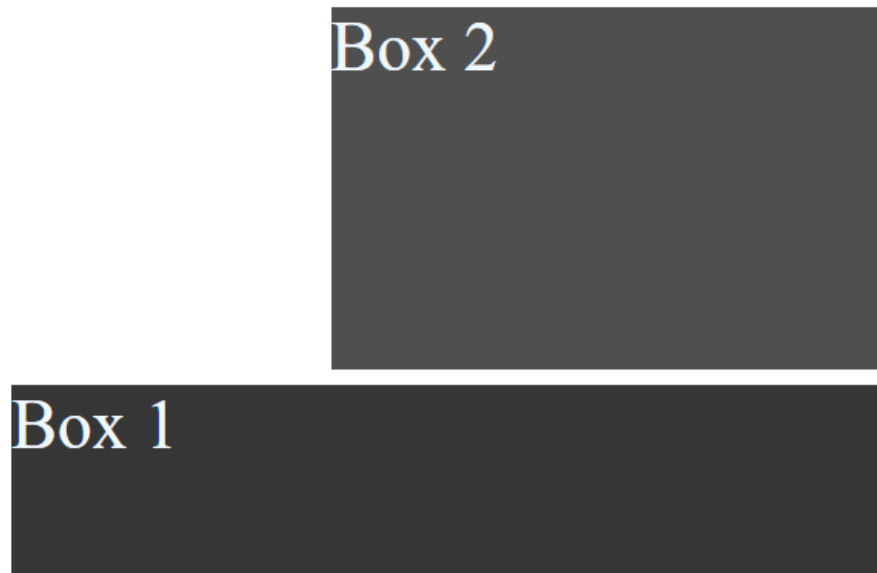
```
.container {  
  display: grid;  
  grid-template: 80px 100px 100px / 100px 50px 80px 200px;  
}
```

2.2 Posicionamento de grid-item

A seguintes propriedades permitem posicionar grid-item:

- `grid-column-start`: coluna que inicia a grid vertical;
- `grid-column-end`: coluna que finaliza a grid vertical;
- `grid-row-start`: linha que inicia a grid horizontal; e
- `grid-row-end`: linha que finaliza a grid horizontal.

Exemplo de posicionamento de grid item:



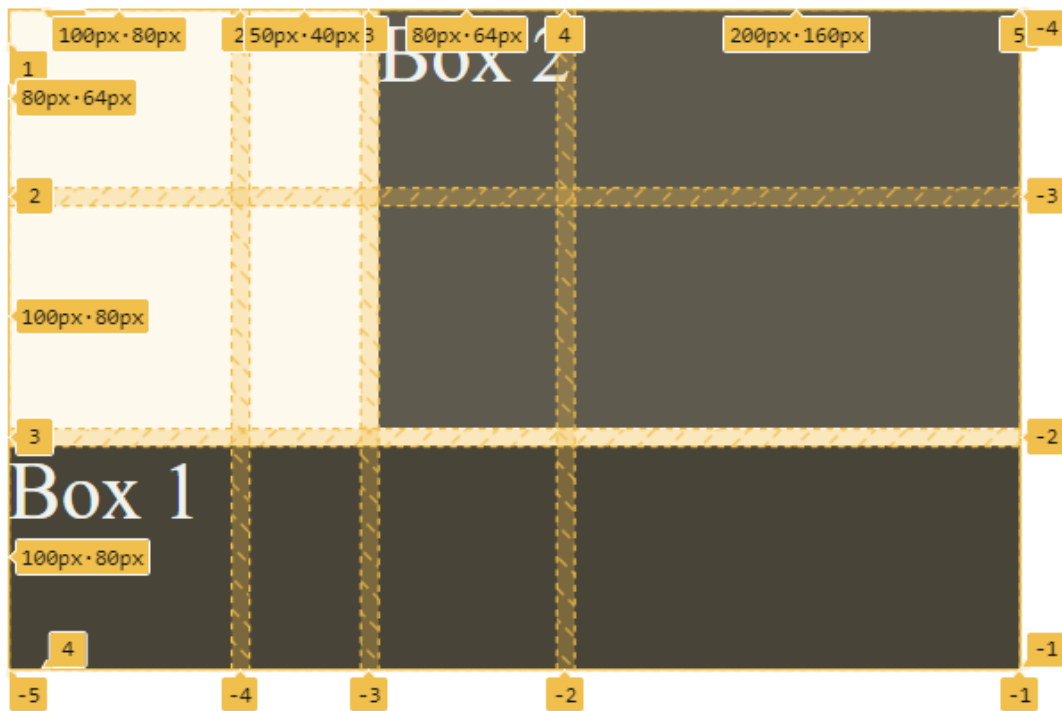
Para a diagramação acima, podemos fazer a seguinte declaração:

CSS	HTML
<pre>.container { display: grid; grid-template-columns: 100px 50px 80px 200px; grid-template-rows: 80px 100px 100px; gap: 8px; } .box1 { grid-column-start: 1; grid-column-end: 5; grid-row-start: 3; grid-row-end: 4; } .box2 { grid-column-start: 3; grid-column-end: 5; grid-row-start: 1; grid-row-end: 3; }</pre>	<pre><div class="container"> <div class="box box1">1</div> <div class="box box2">2</div> </div></pre>

Considerando as bordas externas, o box1 começa na coluna 1 e termina na coluna 5. Começa na linha 3 e termina na linha 4. O box 2 começa na coluna 3 e termina na coluna 5. Começa na linha 1 e termina na linha 3. Estão com gap de 8 px. Estamos utilizando o posicionamento numérico.



Visualização de grid



Podemos utilizar a marcação abreviada:

```
.box1 {  
  grid-column-start: 1;  
  grid-column-end: 5;  
  grid-row-start: 3;  
  grid-row-end: 4;  
}
```

```
.box1 {  
  grid-column: 1 / 5;  
  grid-row: 3 / 4;  
}
```

```
.box2 {  
  grid-column-start: 3;  
  grid-column-end: 5;  
  grid-row-start: 1;  
  grid-row-end: 3;  
}
```

```
.box2 {  
  grid-column: 3 / 5;  
  grid-row: 1 / 3;  
}
```

2.3 Posicionamento utilizando área

Para exemplificarmos o uso de áreas, vamos utilizar uma estrutura básica em HTML.

```
<div class="container">  
  <header class="topo">Topo</header>  
  <nav class="menu">Menu</nav>
```



```
<main class="principal">Principal</main>
<aside class="esquerdo">aside esquerdo</aside>
<aside class="direito">aside direito</aside>
<footer class="rodape">Rodapé</footer>
</div>
```

A seguir, as declarações CSS para compor o layout. Observe que, para fins didáticos, estamos usando medidas fixas.

```
.container{
display: grid;
grid-template-columns: 150px 700px 150px;
grid-template-rows: 60px 40px 200px 60px;
gap: 5px 10px; }
```

A largura total será 1000 pixels. Iremos diagramar 3 colunas e 4 linhas. Nessa etapa, estamos com este layout:

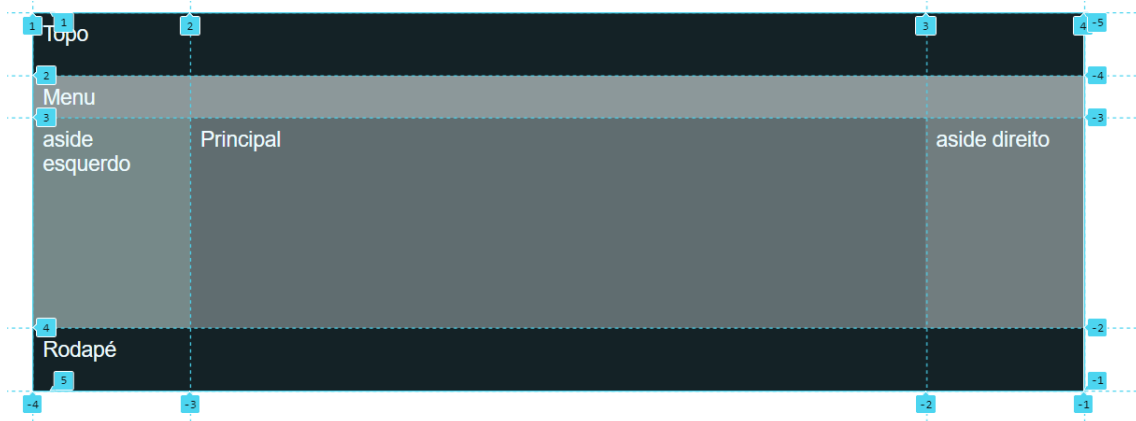
Topo	Menu	Principal
aside	aside direito	Rodapé

CSS

```
.topo {grid-area: 1 / 1 / 2 / 4;}
.menu {grid-area: 2 / 1 / 3 / 4;}
.principal {grid-area: 3 / 2 / 4 / 3;}
.esquerda {grid-area: 3 / 1 / 4 / 2;}
.direita {grid-area: 3 / 3 / 4 / 4;}
.rodape {grid-area: 4 / 1 / 5 / 4;}
```



Passamos a ter o seguinte layout (no browser não irá aparecer o número da grid):



2.4 Funções repeat() e calc()

A função `calc()` permite que se definam valores CSS com uso de expressões matemáticas. Possui a seguinte sintaxe:

```
seletor { propriedade: calc(expressão matemática); }
```

A função **repeat()**, destinada ao dimensionamento do grid, tem como parâmetros da função valores a serem definidos para as propriedades `grid-template-columns` e `grid-template-rows`. Possui dois parâmetros: o primeiro estabelece o número de repetições e o segundo uma medida ou faixa de medidas CSS de comprimento.

O exemplo mostrado a seguir ilustra o uso dessa função no dimensionamento de um grid fluido de 8 x 4.

CSS

```
. container {  
    display: grid;  
    grid-template-columns: repeat(8, 7%);  
    grid-template-rows: repeat(4, 4vh);  
    grid-gap: 10px calc( (100% - 7%* 12) / 11 );  
}
```



EXPLICAÇÃO

`grid-template-columns: repeat(8, 7%);`

Define 8 colunas com largura igual a 7%. Resulta em largura total do grid $8 \times 7\% = 56\%$.

`grid-template-rows: repeat(4, 4vh);`

Define 4 linhas com altura igual a 4vh (4% da altura da viewport²). Resulta em altura total do grid igual a: $4 \times 4vh = 16vh$ (16% da altura da viewport³)

`grid-gap: 8px calc((100% - 7%* 8) / 7);`

Define espaçamento vertical fixo de 8px entre as linhas do grid e espaçamento horizontal calculado pela função CSS `calc()` igual a $100\% - 56\% = 44\%$ dividido pelos 7 espaçamentos horizontais. Resulta em largura total do grid igual a: $56\% + 7 * 6.2857\% = 100\%$ (100% da largura da viewport “tela”).

2.5 Posicionamento com span

No item 2.3 posicionamos os grids itens utilizando a declaração `grid-area`. Podemos utilizar também a tag **span** para delimitarmos essas áreas, uma opção para a diagramação. Pode fazer uma analogia com mesclagem de células em uma tabela. É possível marcar o posicionamento utilizando **span N**. N se refere ao número de células horizontais e/ou verticais que irão compor a área.

Exemplo:

```
.container {
  display: grid;
  grid-template-columns: repeat(6, 100px);
  grid-template-rows: repeat(4, 60px);
  grid-gap: 5px;
}
.box1 {
  grid-column: 2 / span 4; //começa na coluna 2 e irá ocupar 4
  células.
}
.box2 {
  grid-row: 3 / span 2; //começa na linha 3 e irá ocupar 2 células.
}
.box3 {
  grid-column: 3 / span 4; //começa na coluna 3 e irá ocupar 4
```

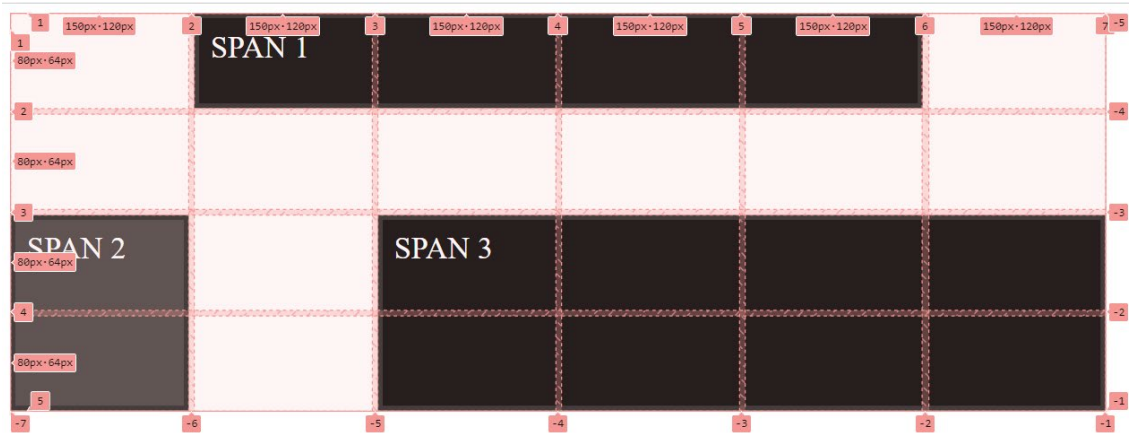
² Viewport: tela do dispositivo.



células.

```
grid-row: 3 / span 2; } //começa na linha 3 e irá ocupar 2 células.
```

Visualização do código acima:



2.6 Posicionamento com regiões nomeadas

É possível dar nomes às áreas. A propriedade para nomear do grid é `grid-template-areas` e os nomes escolhidos para cada grid area são usados como valor CSS para a propriedade `grid-area` definida para os grids items.

Exemplo:

```
<div class="container">
  <header class="topo">Topo</header>
  <nav class="menu">Menu</nav>
  <main class="principal">Principal</main>
  <aside class="esquerda">aside esquerdo</aside>
  <aside class="direita">aside direito</aside>
  <footer class="rodape">Rodapé</footer>
</div>
```

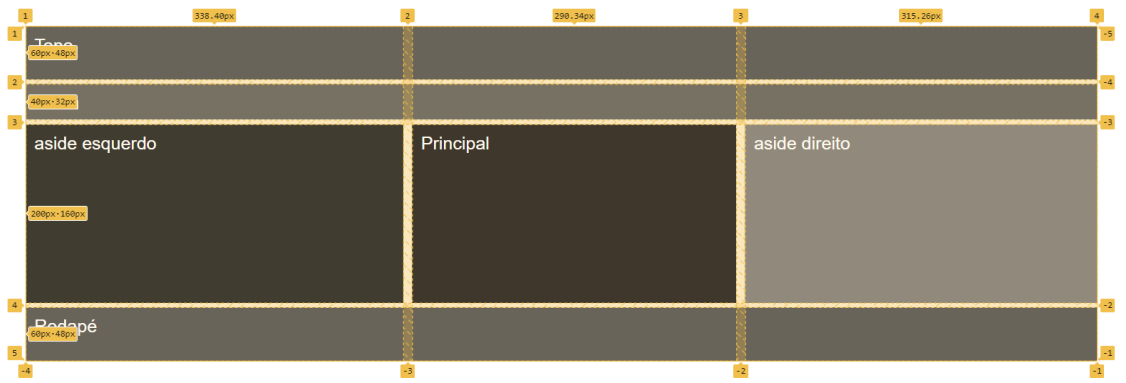
No CSS

```
.container {
  display: grid;
  grid-template-rows: 60px 40px 200px 60px;
  grid-template-areas:
    "topo      topo      topo"
    "menu      menu      menu"
    "esquerda principal direita"
    "rodape    rodape    rodape";
  grid-gap: 5px 10px;
}
.topo {grid-area: topo;}
```



```
.menu {grid-area: menu;}  
.principal {grid-area: principal;}  
.esquerda {grid-area: esquerda;}  
.direita {grid-area: direita;}  
.rodape {grid-area: rodape;}
```

Visualização do código acima:



Para especificar áreas com medidas, basta fazer a especificação de valores em `grid-template-rows` e `grid-template-columns`.

2.7 Unidade de Medida fr

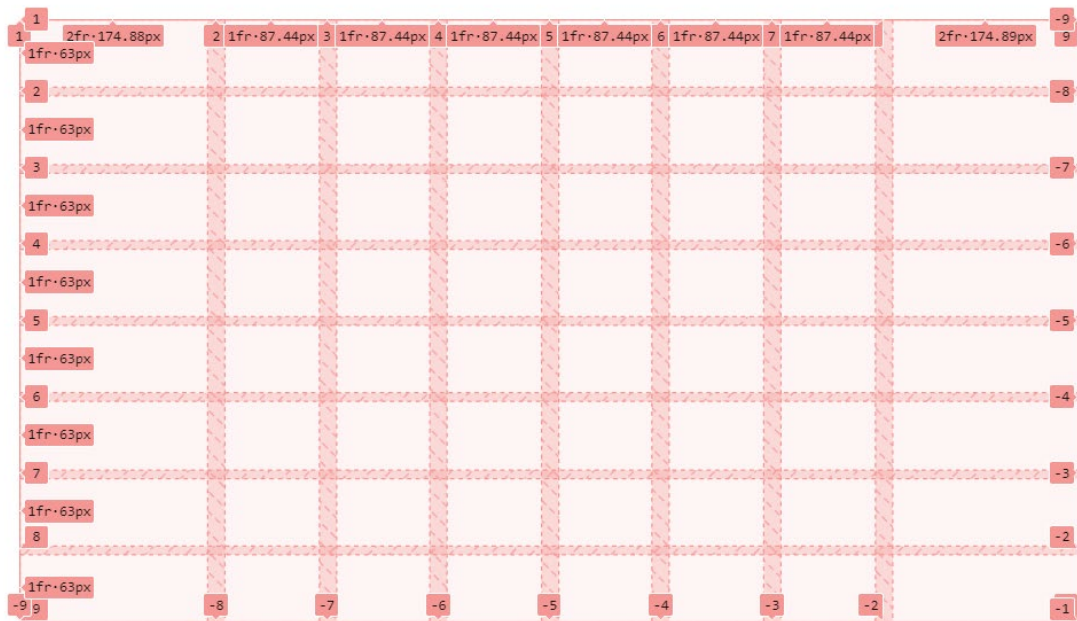
Foi criada uma unidade de medida relativa `fr`, que significa fração. Trata-se de uma unidade relativa cuja referência é a medida do espaço disponível, tanto na vertical quanto na horizontal.

Exemplo:

```
.container {  
  display: grid;  
  grid-template-columns: 2fr repeat(6, 1fr ) 2fr; //Cria 8  
  colunas, iniciando com 2 fr a esquerda e duas 2 fr a direita. No  
  centro 6 repetições de 1 fr.  
  grid-template-rows: repeat(8, 1fr ); //Cria 8 linhas de 1 f  
  cada  
  gap: 10px 20px;  
  height: 80vh;  
}
```



Visualização da especificação acima:



2.8 Função minmax()

A função `minmax()` permite estabelecer duas dimensões, tanto para largura quanto para altura. Podendo ser aplicada a `grid-template-columns` e `grid-template-rows`.

Exemplo:

```
.container {
  display: grid;
  grid-template-columns: 200px minmax(400px, 800px); // Define
  // duas colunas, sendo a primeira com largura fixa igual a 200px e
  // a segunda com largura mínima de 400px e máxima de 800px. Ou
  // seja, dentro dessa faixa de valores, o grid é fluido.
  grid-template-rows: 50px minmax(200px, auto) 50px; //
  // Define três linhas, sendo a segunda fluida com altura mínima de
  // 200px e máxima igual àquela a acomodar todo o conteúdo nela
  // inserido. As outras duas linhas são de altura fixa igual a
  // 50px.
  gap: 10px 20px;
}
```

TEMA 3 – FLEXBOX

O *Flexible Box Module*, geralmente chamado de **flexbox**, pode ser utilizado também para diagramação de páginas e apresenta possibilidades



avançadas de alinhamento. A diferença fundamental entre Flexbox e Grid Layout é que o primeiro tem o melhor uso para a criação de layout em uma dimensão, seja horizontal (posicionamento em linhas) ou vertical (posicionamento em colunas), e o segundo para a criação de layout em duas dimensões: linhas e colunas (Silva, 2022).

3.1 Elementos flexbox

container

Para se marcar utilizando flexbox, inicialmente é necessário definir um container.

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

flex-item

São os elementos filhos de um container modo flex. No exemplo a seguir, todos os elementos-filhos contidos dentro da div com class="container1" tornam-se elementos filhos.

HTML

```
<h2>Display flex</h2>
<div class="container1">
  <div class="box box1">DIV</div>
  <span class="box box2">SPAN</span>
  <em class="box box3">EM</em>
  <div class="box box4">DIV</div>
</div>
<br><br><br><br>
<h2>Display block</h2>
<div class="container2">
  <div class="box box1">DIV</div>
  <span class="box box2">SPAN</span>
  <em class="box box3">EM</em>
  <div class="box box4">DIV</div>
</div>
```

CSS

```
html { box-sizing: border-box; }
.container1 { display: flex; }
.container2 { display: block; }
```



```
.box {  
  color: white;  
  border: 4px solid #333;  
  font-size: 24px;  
  padding: 10px;  
}  
.box1 {background: #086bdd;}  
.box2 {background: #1c0989;}  
.box3 {background: #00028b;}  
.box4 {background: #7c58de;}
```

Visualização no browser:



Nessa imagem, apresentamos a visualização com o uso do display flex (1ª imagem) e com uso de display block (2ª imagem). Ao utilizar o display block (padrão), os elementos-filhos são exibidos conforme suas propriedades display inline (span e em) e block (div). Ao utilizar display flex, os elementos se apresentam em linha.

3.1 Elementos flexbox

display

O elemento display foi criado em 1996. Define se um elemento é exibido como block ou inline. O display pode ser usado para elementos filhos, como já vimos em grid. Agora vamos estudar o display flex.

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

Faz com que todos os elementos-filhos sejam exibidos no modo flex. É a primeira declaração a ser feita.


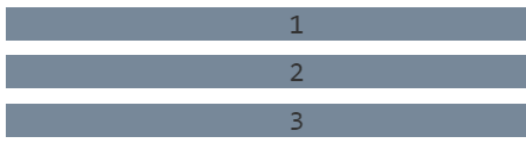

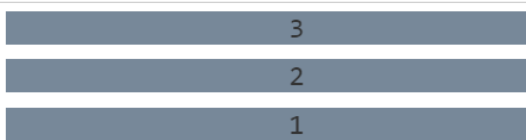


flex-direction

Essa propriedade admite os valores: row, column, row-reverse e column-reverse. Destina-se a definir o sentido e a direção do main axis³ respectivamente na horizontal para a direita (inicial), vertical para baixo, horizontal para a esquerda e vertical para cima. A sintaxe para se declarar essa propriedade é como a mostrada a seguir.

HTML	CSS
<pre><h1>display: flex;</h1> <section class="container flex"> <div class="item">Teste 1</div> <div class="item">Teste 2</div> <div class="item">Teste 3</div> </section></pre>	<pre>.flex { display: flex; flex-direction: row; } .item { margin: 5px; background: #228B22; text-align: center; font-size: 1.5em; } .container { max-width: 400px; margin: 0 auto; border: 1px solid #ccc; }</pre>

Exemplo: observe na imagem que o reverse inverte a ordem, tanto de linhas quanto de colunas.

<p>flex-direction: row;</p> 	<p>flex-direction: column;</p> 
<p>flex-direction: row-reverse;</p> 	<p>flex-direction: column-reverse;</p> 

³ Main axis por padrão é o eixo horizontal e o seu sentido é da esquerda para a direita.








justify-content

Essa propriedade admite os valores flex-start, center, flex-end, space-between e space-around. Com esses comandos, podemos estabelecer o alinhamento.

Sintaxe:

```
.container {  
display: flex;  
justify-content: space-around;  
}
```

Exemplos:

Justify-Content	
justify-content: flex-start;	justify-content: flex-center;
	
justify-content: flex-end;	justify-content: space-between ;
	
	justify-content: space-around;
	

flex-wrap

Essa propriedade possui como valores nowrap, wrap e wrap-reverse. Permite controlar se irá exibir o flex itens em uma linha contínua, ou com quebras de linha.

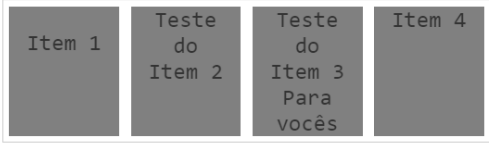
CSS:

```
.container {display: flex;  
flex-wrap: wrap; /* o valor padrão é nowrap */ }
```

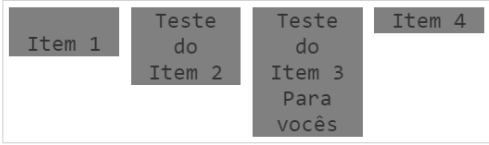


Exemplos:

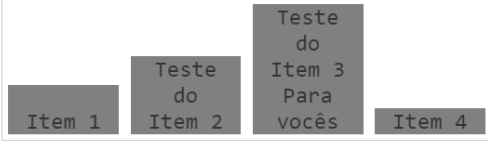
`align-items: stretch;`



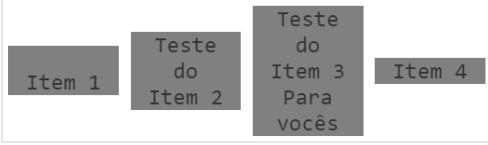
`align-items: flex-start;`



`align-items: flex-end;`



`align-items: center;`



align-items

Pode se utilizar para alinhar os flex items dentro do container, tendo como referência cross axis⁴. Essa propriedade possui como valores principais: flex-start, center, flex-end e stretch. O valor padrão é stretch, o que faz com que os flex items se expandam em altura até ocupar toda a altura do container.

CSS:

```
.container {  
  display: flex;  
  align-content: flex-start; /* o valor inicial é flex-start */  
}
```

Exemplos:

`align-content: flex-stretch;`



`align-content: flex-start;`



`align-content: flex-end;`



`align-content: center;`



`align-content: space-between;`



`align-content: space-around;`



⁴ Cross axis é o eixo vertical. Seu sentido é de cima para baixo.



align-content

Essa propriedade possui os valores `flex-start`, `center`, `flex-end`, `stretch`, `space-between` e `space-around`. Permite alinhar os flex items dentro do container, segundo a direção e o sentido do cross axis⁴ e aplica-se no caso em que os flex items são dispostos dentro do container em múltiplas linhas (`flex-wrap: row`). Ou seja, tem efeito semelhante ao da propriedade `align-items`, quando o container comporta múltiplas linhas.

CSS:

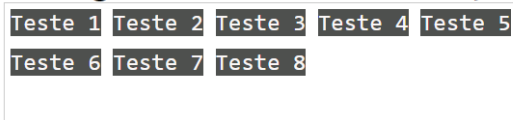
`align-content: flex-stretch;`



`align-content: center;`



`align-content: flex-start;`



`align-content: space-between;`



`align-content: flex-end;`



`align-content: space-around;`



TEMA 4 – MEDIA QUERIES

O W3C recomenda o uso de Media Queries. Com o uso de Media Queries, podemos criar regras CSS para serem utilizadas conforme o agente de usuário⁵.

Media Queries nos permite criarmos a responsividade, ou seja, diagramar páginas que se ajustem em smartphones, tablets e desktop.

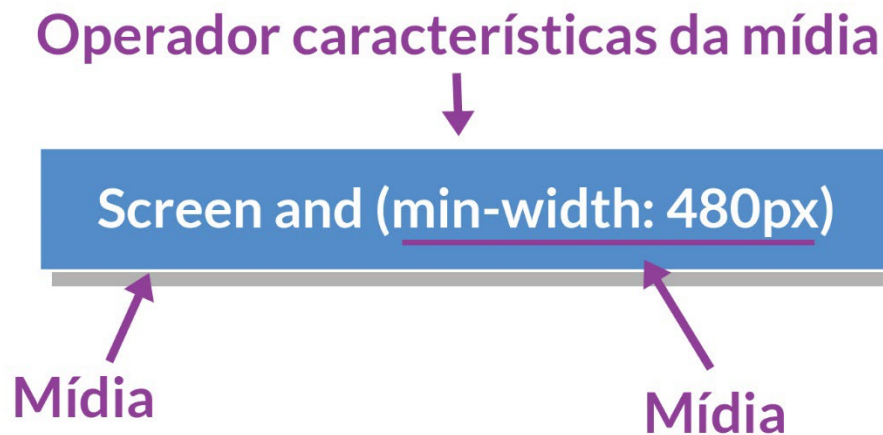
4.1 Anatomia Media Query

Media Query significa consulta de mídia. O uso de media query permite criar uma folha de estilo para uma determinada mídia, mediante consulta e identificação desta mídia. Na verdade, é possível descobrir as características da tela. Uma media query é uma expressão lógica que resulta em `true` ou `false`.

⁵ *Agente de usuário* é um termo utilizado para especificar navegadores e dispositivos.



Sintaxe:



A media query da figura acima vai resultar *true* se a mídia tiver uma tela com largura mínima de 480px.

Exemplo:

```
@media screen and (min-width: 600px) {  
  body {  
    background-color: lightblue;  
  }  
}
```

Nesse exemplo, se o dispositivo tiver a tela com largura mínima de 600px, a cor de fundo *lightblue* será exibida.

Operadores media query

Operador	Descrição
and	Resulta <i>true</i> se as condições à esquerda e à direita do operador forem verdadeiras.
only	Utilizada para esconder folhas de estilo dos navegadores antigos que não suportam <i>media query</i> .
not	Resulta <i>true</i> se a condição não for satisfeita.



4.2 Funcionalidades Media Query

Pode-se aplicar as seguintes funcionalidades: *width*, *height*, *device-width*, *device-height*, *orientation*, *aspect-ratio*, *device-aspect-ratio*, *color*, *color-index*, *monochrome*, *resolution*, *scan* e *grid*.

Vamos ver alguns exemplos de cada funcionalidade:

width: descreve a largura da área de saída do dispositivo do usuário. Exemplo:

```
@media screen and (min-width: 320px) and (max-width: 700px)
{....}
```

height: descreve a altura da área de saída do dispositivo do usuário. Exemplo:

```
@media screen and (min-height: 600px) and (max-height: 768px)
{....}
```

device-width: descreve a largura física da área de renderização do dispositivo do usuário. Exemplo:

```
@media screen and (min-device-width: 400px) and (max-device-
width: 700px) {....}
```

device-height: descreve a altura física da área de renderização do dispositivo do usuário. Exemplo:

```
@media screen and (min-device-height: 600px) and (max-device-
height: 768px) {....}
```

width e *height* são funcionalidades dinâmicas e *device-width* e *device-height* são funcionalidades fixas, pois tratam de medidas de dispositivos físicos.

orientation: os valores possíveis para *orientation* são *landscape* e *portrait*. Exemplo:

```
@media screen and (orientation: landscape) {....}
@media screen and (orientation: portrait) {....}
```



aspect-ratio: considera a razão entre a largura (*width*) e altura (*height*) da área de saída do dispositivo do usuário. Os valores possíveis são frações a/b. Exemplo:

```
@media screen and (aspect-ratio: 16/9) {....}  
@media screen and (min-aspect-ratio: 5/2) {....}
```

device-aspect-ratio: considera a razão entre a largura (*device-width*) e altura (*device-height*) da área de saída do dispositivo do usuário. Os valores possíveis são frações a/b. Exemplo:

```
@media screen and (device-aspect-ratio: 16/9) {....}  
@media screen and (min-device-aspect-ratio: 800/600) {....}
```

Color: descreve o número de bits por componente de cor no dispositivo do usuário. Oito bits permitem 256 cores. Exemplo:

```
@media screen and (color: 2) {....} /*dispositivos coloridos de  
8 bits*/  
@media screen and (color: 0) {....} /*dispositivos  
monocromáticos*/
```

color-index: descreve o número de entradas da paleta de cores do dispositivo do usuário. Exemplo:

```
<link rel="stylesheet" media=screen and (color-index)  
"href=..."> /* todos os dispositivos de paleta de cores*/
```

monochrome: descreve o número de bits por componente monocromático no dispositivo do usuário. Exemplo:

```
@media screen and (monochrome: 2) {...} /*dispositivos  
monocromáticos de 2 bits por pixel*/
```

resolution: descreve a resolução, densidade de pixel, no dispositivo do usuário. Exemplo:

```
@media screen and (min-resolution: 300dpi) {...} /*dispositivos  
com resolução maior que 300 pontos por polegadas*/
```



scan: descreve o processo de escaneamento de imagens em mídias do tipo TV.

Exemplo:

```
@media tv and (scan: progressive) {...}
@media tv and (scan: interlace) {...}
```

grid: descreve se a saída do dispositivo é em grid ou bitmap Exemplo:

```
@media handheld and (grid) and (max-width: 15em) {...}
```

4.3 Breakpoints

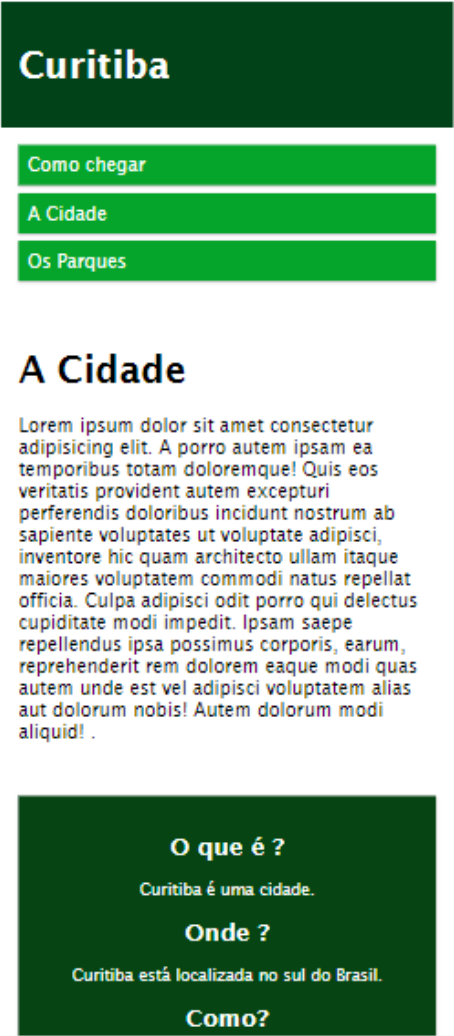
Breakpoints significam pontos de quebra. O termo é usado em design responsivo para designar a medida da largura da janela do navegador, ou o ponto em que há uma quebra do layout. Assim, breakpoints são os pontos nos quais o layout se readapta para se ajustar a uma nova largura da janela.

Observe a imagem de site visualizado numa tela de 1280x800:



O layout está diagramado com três colunas. Se visualizamos o mesmo layout num dispositivo com 393x851, o layout passa a ser exibido em uma coluna.



	<p>Essa adaptação do layout é possível com uso de media queries. Por exemplo:</p> <pre>@media only screen and (min-width: 768px) {...}</pre> <p>Quando a largura mínima da tela do dispositivo for de 768px, ele passa a exibir o layout com três colunas.</p> <p>Pode-se criar uma adequação para tablets:</p> <pre>@media only screen and (min-width: 600px) {...}</pre> <p>O layout passar a ser exibido em duas colunas.</p>
--	--

4.4 Mobile First

O uso de smartphones para navegação é atualmente mais comum do que o acesso em computadores pessoais. Por este motivo, a boa prática que temos hoje é a de desenvolvermos primeiramente para dispositivos móveis. O Google passou a indexar considerando primeiramente a versão dos conteúdos para esse dispositivo, desde o ano de 2019. Assim, a recomendação é de desenvolver primeiramente para dispositivos móveis. Ao fazer as declarações em CSS, a primeira que deve ser declarada é para smartphones, e assim para os demais dispositivos, utilizando o breakpoint.

TEMA 5 – RESPONSIVIDADE EM IMAGENS

O objetivo de tornar páginas e aplicações responsivas é que sejam exibidos de forma igual em dispositivos com diferentes tamanhos de tela e



resoluções. Vamos agora conhecer as marcações e declarações que podemos utilizar para esta finalidade, para imagens.

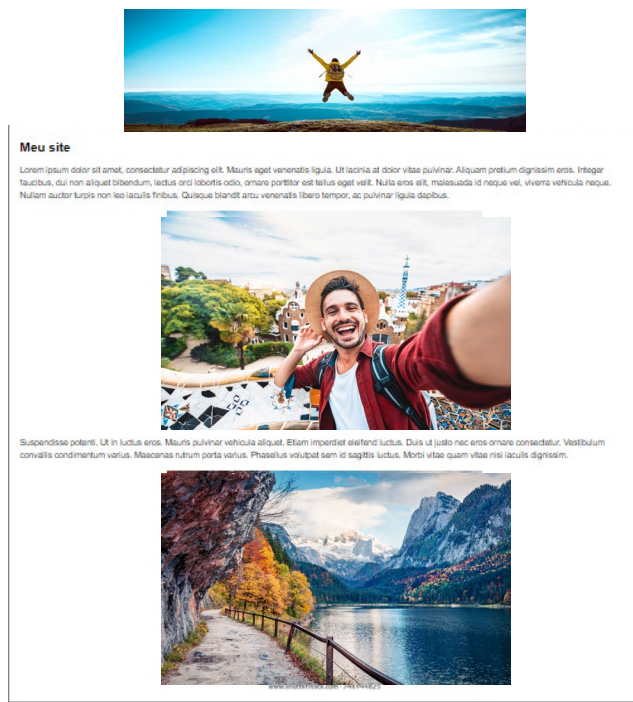
5.1 Imagens

Para exibir imagens, utilizamos a marcação ``. Podemos declarar no CSS da seguinte forma:

```
img {  
  display: block;  
  margin: 0 auto;  
  max-width: 100%;  
}
```

A declaração acima faz com que a imagem seja exibida centralizada (`margin: 0 auto;`) e com largura máxima de 100% (`max-width: 100%;`). Observe as imagens:

No desktop



No smartphone



No `<header>`, foi inserida uma imagem com uma pessoa saltando de paraquedas. Essa imagem foi configurada para estar centralizada; o foco da imagem é a pessoa. Quando visualizamos a imagem numa tela pequena, perdemos os seus lados esquerdo e direito. Outro aspecto, o conteúdo do `<body>`, foi configurado para o máximo de 1200 pixels de largura (`width`). Em



viewports⁶ maiores, o body continua a 1200 pixels e centralizado no espaço disponível. Em viewports⁷ menores, o body vai usar 100% da largura disponível.

5.2 Imagens flexíveis

Ainda não foram implementadas declarações consistentes para a utilização de imagens flexíveis. É um dos grandes problemas do desenvolvimento responsivo. Em 2014, o W3C recomendou o elemento *picture* e o atributo *srcset*.

Cada caso é um caso. Normalmente, necessitamos disponibilizar um conjunto de imagens, para que sejam exibidas de acordo com a largura da viewport (tela).

5.2.1 srcset

Para inserirmos um conjunto de imagens, utilizamos o atributo *srcset*. O browser escolherá dentre o conjunto de imagens, conforme a largura da viewport do usuário e baixará somente a imagem escolhida, otimizando a performance.

Sintaxe:

O atributo *srcset* deve conter um conjunto de imagens separadas por vírgulas, sendo obrigatório o URL apontando para o endereço de uma imagem. São opcionais (um deles deve ser declarado):

- descritivo de largura, expressa em pixel, constituído de um número inteiro positivo seguido da letra W;
- descritivo de altura, expressa em pixel, constituído de um número inteiro positivo seguido da letra H; e
- descritivo de densidade de pixel constituído por um número do tipo ponto flutuante seguido da letra X.

Exemplo:

```
<h1>
<img alt ="a imagem é o logo da empresa"
src =" logo.jpg"
srcset =" logo-HD.jpg 2x, logo-phone.jpg 150w, logo-phone-HD.jpg
```

⁶ Viewport: tela.



```
150w 2x" >  
</ h1 >
```

Explicação:

- **src:** para exibir a imagem em dispositivos com tela que não sejam de alta definição e como fallback⁷ para exibir a imagem nos navegadores que não entendem o atributo srcset;
- **logo-HD.jpg 2x:** imagem a ser usada em dispositivos de alta definição;
- **logo-phone.jpg 150w:** imagem a ser usada em dispositivos com largura de viewport até 150px e telas que não tenham alta definição;
- **logo-phone-HD.jpg 150w 2x:** imagem a ser usada em dispositivos com largura de viewport até 150px e telas em alta definição.

5.2.2 picture

A declaração CSS *picture* foi criada em 26 de fevereiro de 2013. O elemento *picture* é recomendado quando utilizar imagens com tamanhos diferentes de tela. Não deve ser utilizado quando for exibir somente uma imagem. Nesse caso, deve-se utilizar o elemento ``.

Sintaxe:

```
<picture>  
  <source media="(min-width:650px)" srcset="img.jpg">  
  <source media="(min-width:465px)" srcset="img_p.jpg">  
    
</picture>
```

O navegador procurará o primeiro elemento `<source>` em que a consulta de mídia corresponda à largura atual da tela do dispositivo do usuário e exibirá a imagem adequada (especificada no atributo *srcset*). O elemento `` é necessário como o último filho do elemento `<picture>`, como uma opção de *fallback* se nenhuma das imagens do atributo *srcset* não “servirem” ao navegador.

⁷ Fallback: uma opção a ser utilizada caso a opção preferida não esteja disponível.



5.2.3 Art direction

Art direction significa direção de arte. No contexto do design responsivo, significa criar imagens com tamanhos diferentes para diferentes dispositivos. Vamos considerar as seguintes dimensões: 320px x 240px, 640px x 480px, 700px x 525px, 1000px x 750px, 1600px x 1200px. (Observe que a proporção das medidas largura/altura é 4/3).

Exemplos de breakpoints e art direction:

HTML

```
< div class = " artdirection" alt ="exemplo " > </ div >
```

CSS

```
.artdirection {  
  max-width: 100%;  
  height: 240px;  
  border: 5px solid red;  
  background-image: url(imagens/ img1. jpg);  
  background-position: top;  
}  
@media screen and (min-width: 320px) {  
  .artdirection {  
    border: 5px solid blue;  
    background-image: url(imagens/ img2. jpg);  
    height: 525px;  
  }  
}  
@media screen and (min-width: 700px) {  
  .artdirection {  
    border: 5px solid green;  
    background-image: url(imagens/img2. jpg);  
    height: 750px;  
  }  
}  
@media screen and (min-width: 1600px) {  
  .artdirection {  
    border: 5px solid black;  
    background-image: url(imagens/ img3. jpg);  
    max-width: 1600px;  
    height: 1200px;  
  }  
}
```



Acima, vimos como estabelecer breakpoints com algumas larguras e as declarações com imagens de fundo para cada medida considerada.

FINALIZANDO

Chegamos ao final da nossa etapa, em que estudamos o HTML semântico para criar elementos da diagramação. No HTML 5, a semântica é importante e recomenda-se utilizá-la da maneira correta.

Estudamos também a teoria de: grids, flexbox, media queries, breakpoints e imagens responsivas.

Apresentaremos exemplos de utilização dessas declarações em outro momento. Bons estudos!



REFERÊNCIAS

GRID BY EXAMPLE. Disponível em: <<https://gridbyexample.com/>>. Acesso em: 29 jan. 2023.

MOZILLA DEVELOPER NETWORK. Disponível em: <<https://developer.mozilla.org/en-US/>>. Acesso em: 29 jan. 2023.

SILVA, M. S. **CSS Grid Layout**. Novatec, 2022.

_____. **CSS3**. Novatec, 2013.

W3C. Disponível em: <<http://www.w3.org>>. Acesso em: 29 jan. 2023.