escola britânica de artes criativas & tecnologia

Desenvolvedor Full Stack Python

Fundamentos do CSS Responsivo

A metatag viewport

No HTML temos a **metatag viewport**, localizada dentro da tag **head:**

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

Podemos entender o viewport como toda a área visível do navegador que é ocupada pela aplicação front-end, ou seja, todo o espaço de conteúdo, sem a barra de endereço, favoritos ou qualquer outro elemento do sistema.

Valores para metatag viewport

No atributo contente podemos inserir as regras:

Width: para definir a largura do viewport;
Height: para definir a altura do viewport;
Initial-scale: para definir a escala inicial do viewport;
User-scalable: para definir se será possível ou não aplicar zoom a página;
Minimum | maximum-scale: para definir a escala máxima e mínima da página.

Valores para metatag viewport

Os valores para width e height são especificados em valores numéricos ou utilizando a palavra device-width/device-height, que será referente a dimensão do dispositivo.

```
<metaname="viewport"content="width=device-width" />
<meta name="viewport" content="width=device-320" />
```

No último exemplo a largura do viewport foi limitada em 320 pixels, mesmo que a resolução do dispositivo seja maior, o recomendado é utilizar o valor device-width para aproveitar toda a resolução do dispositivo.

Valores para metatag viewport

User-scalable: utilizamos os valores 0 e 1, ou no e yes

```
<meta name="viewport" content="user-scalable=yes" />
```

Initial-scale, maximum-scalee minimum-scale: utilizamos os valores entre 0 e 1, podendo utilizar valores fracionados:

```
<meta name="viewport" content="initial-scale=1" />
```

Também é possível combinar as propriedade dentro da metatagviewport:

```
<meta name="viewport"
    content="initial-scale=1, user-scalable=no" />
```

Unidades de medida

No CSS temos dois tipos de unidades de medida, as fixas e flexíveis.

As unidades fixas são os pixels (px) e os pontos (pt), um ponto = 1/72 polegadas.

As unidades flexíveis são VW, VH, em, rem e porcentagem.

VW: unidade referente à largura do viewport (viewport width), exemplo:

```
{
width: 50vw;
height: 50vw;
}
```

No exemplo acima teremos um elemento que terá a largura e altura igual à metade da largura do viewport (50%).

VH: praticamente igual ao VW, porém se refere à altura do viewport (viewport height), caso queira que um elemento ocupe 100% da altura da tela, basta usar 100vh como valor para o height.

Unidades de medida

EM: a unidade em faz referência ao font-size do elemento-pai, existe uma convenção que o font-size padrão é de 16px, logo se o elemento estilizado conter a regra: font-size: 1em, e for filho direto da tag body, o valor 1em será igual à 16px. Portanto, 2em = 16 x 2 = 32px.

Caso o elemento esteja dentro de um div que possui a regra font-size: 32px, 2em seria igual à 64px, 32 * x = 64px;

REM: muito parecido com a unidade EM, porém se refere ao elemento raiz (root-em), que é a tagHTML;

Unidades de medida

Porcentagem: quando utilizada no font-size irá se referenciar ao font-size do elemento-pai, quando utilizada no width ou height, será referenciado ao width ou height do elemento-pai.

```
div{
   font-size: 64px;
}

div p {
    font-size: 200%; // 200% de 64px = 128px
     width: 100%; // ocupará a largura total da div
}
```

Media Queries e Breakpoints

Com as **media queries** podemos escrever regras CSS que serão aplicáveis a partir de determinada resolução de tela.

Para esta determinada resolução damos o nome de **breakpoint (ponto de quebra).**

Escrevemos uma media query no CSS da seguinte forma:

```
@media screenand(max-width: 640px) {
    body { background-color: red; }
}
```

O código CSS dentro do bloco @media será válido até uma resolução com largura de 640px, até esse ponto o elemento body terá a cor de fundo vermelha.

Media Queries e Breakpoints

Podemos adicionar outras condicionais à uma media query:

```
@media screenand(max-width: 640px) and(orientation: landspace) {
        body { background-color: red; }
}
```

No exemplo acima adicionamos mais uma condição, a orientação do dispositivo, que deverá estar na horizontal (landscape), para que a media query seja válida.

A condição de orientação pode receber os valores: **landscape (horizontal) e Portrait (retrato).**

Além do max-width podemos utilizar o min-width, max-height ou min-height:

```
@media screenand(min-width: 768px) and(max-height: 800px)
{
      ...
}
```

A tag Picture

Utilizando a tag **picture** podemos disponibilizar diferente arquivos de imagens:

```
<picture>
   <sourcesrcset="fachada-tablet.png" media="(max-width: 1024px)" />
   <sourcesrcset="fachada-mobile.png" media="(max-width: 640px)" />
    <imgsrc="fachada.png" alt="Fachada da loja Exemplo" />
</picture>
```

Na tag **source**, dentro de **picture**, especificamos os arquivos que serão utilizados quando determinadas media queries forem válidas.

Display Grid

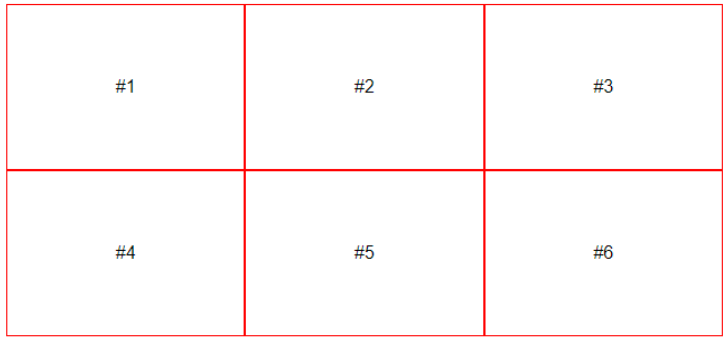
Com o valor **"grid"** para a propriedade CSS display podemos construir layouts de uma forma bem simples, até mesmo layouts responsivos.

O layout construindo com o display Grid é dividido em colunas e linhas.

Para usar display Grid é necessário estilizar o container da página com a regra **display: grid;** e além disso definir o layout que será criado, fazemos isso com a propriedade: **grid-template-columns.**

Exemplo:

```
.container{
    display: flex;
     grid-template-columns: 1fr 1fr1fr;
}
```



A propriedade grid-template-columns

A unidade de medida que utilizamos no exemplo anterior, fr, significa fração, se tivéssemos apenas um valor fr **(grid-template-columns: 1fr)** teríamos um layout com apenas uma coluna.

Além do **fr** podemos utilizar porcentagem, pixels e auto.

Exemplo:

10% 50% auto // auto seria igual à 40%
120pxauto50%// auto seria igual à 100% -(120px + 50%)

Espaçamento

Podemos adicionar espaçamento ao layout, chamamos isso de gap, para adicionar espaçamento à colunas utilizamos a propriedade **column-gap**, que recebe o valor em pixels ou percentual.

É importante levar em conta o espaçamento no momento em que se define o tamanho das colunas.

Exemplo:

```
.container {
    display: grid;
    grid-template-columns: 25% 25% 25% 25%;
     column-gap: 1%;
}
```

Teríamos 4 colunas com o espaçamento de 1%, o que faria com que o layout fosse maior que o container, neste exemplo o tamanho ideal para as colunas seria de 24,25%.

100% -3% (temos 3 divisões, entre as colunas) = 97
97/4=24,25 que é o tamanho ideal para cada coluna.

Espaçamento

Para espaçar as linhas usamos a propriedade row-gap, neste caso não precisamos nos preocupar com o container, afinal não limitamos a altura do layout.