

INTRODUÇÃO AO DESENVOLVIMENTO WEB

---

**WEB DEV 4 NOOBIES**



# GUSTAVO VIEGAS

[www.gfviegas.com](http://www.gfviegas.com)

- ▶ Mais de 5 anos de experiência profissional em desenvolvimento;
- ▶ Trabalhou a maior parte da carreira com desenvolvimento web;
- ▶ Atualmente é prestador de serviços em soluções de TI, em diversas áreas;
- ▶ Graduando em Ciência da Computação na Universidade Federal de Viçosa - Campus Florestal;
- ▶ Entusiasta, evangelizador e ativista (chato!) da comunidade dev;

## CARACTERÍSTICAS DO CURSO

- ▶ Aulas com apresentação do conteúdo;
- ▶ Live-coding com demonstrações práticas do conteúdo estudado;
- ▶ Exercícios e revisões;
- ▶ Projeto final de um layout responsivo simulando um MVP;

# CONTEÚDO DO CURSO

1. Como funciona a web;
2. Sintaxe do HTML;
3. Tags HTML;
4. Sintaxe do CSS;
5. Seletores CSS;
6. Responsividade;
7. SASS;
8. HTML semântico;
9. Introdução à JavaScript;
10. Frameworks;
11. Projeto final;

## AS CAMADAS DA WEB – FRONT E BACK

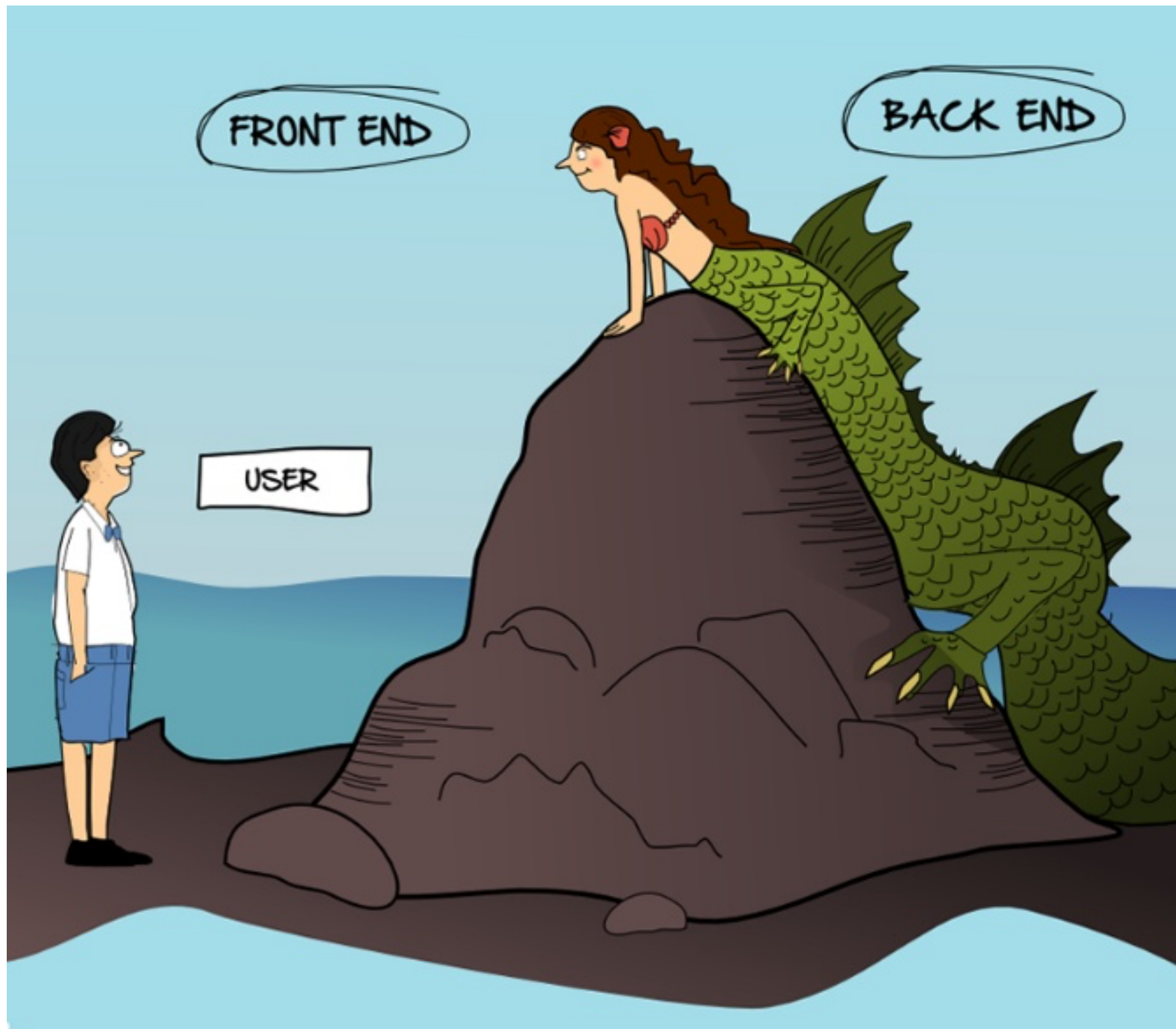
- ▶ A interação de usuários com um sistema web passa por algumas camadas.
- ▶ Apesar de ter responsabilidades diferentes, devem ter uma harmonia entre elas para o funcionamento correto de uma aplicação.
- ▶ Se comunicam através de requisições HTTP.

## O FRONTEND

- ▶ "Interface" que possui todos os elementos que o usuário interage diretamente.
- ▶ Pode ser uma página web, um formulário, um aplicativo, etc.
- ▶ Roda diretamente no dispositivo do usuário, como os browsers. Conhece apenas a linguagem de marcação HTML, de estilo CSS e de programação JavaScript (com muitas limitações).

## O BACKEND

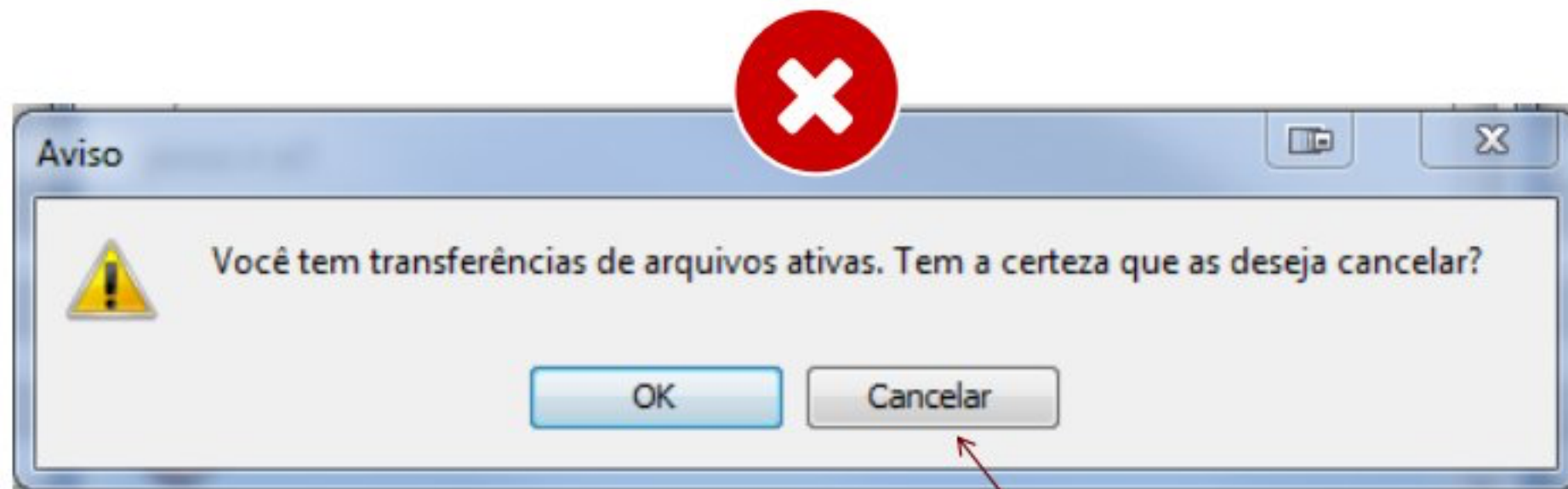
- ▶ "Interface" que possui todas as regras de negócio, modelagem de dados e comunicações da aplicação.
- ▶ Recebe e retorna dados através de protocolos bem definidos
- ▶ Faz o trabalho sujo, que o usuário não tem conhecimento ou visualização.
- ▶ Rodam em servidores dedicados com linguagens de programação como JavaScript, Java, PHP, Python, Ruby, C#, C++..... anything!



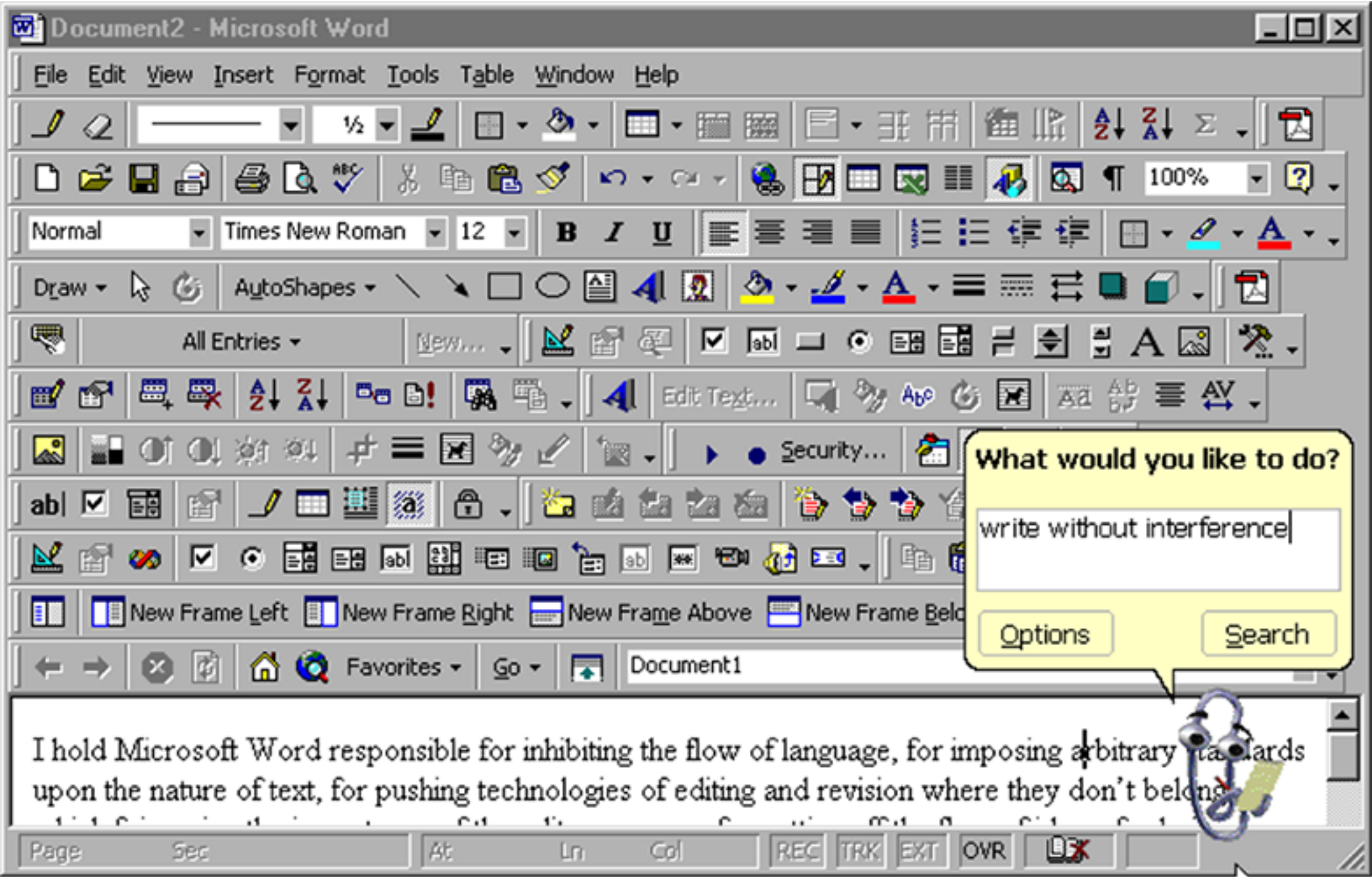


# INTERFACE HOMEM-MÁQUINA

- ▶ Parece contraintuitivo, mas... muitas vezes o mais importante de um sistema é o seu **FRONTEND!!!**
- ▶ Um layout funcional, intuitivo e convidativo é essencial para o sucesso de uma aplicação.
- ▶ Se o usuário não conseguir, ou não gostar de utilizar um sistema, por mais útil que seja, ele se torna instantaneamente descartável para ele!
- ▶ Pense nos sistemas e aplicativos que você usa. Quais deles tem uma interface difícil de ser utilizada?

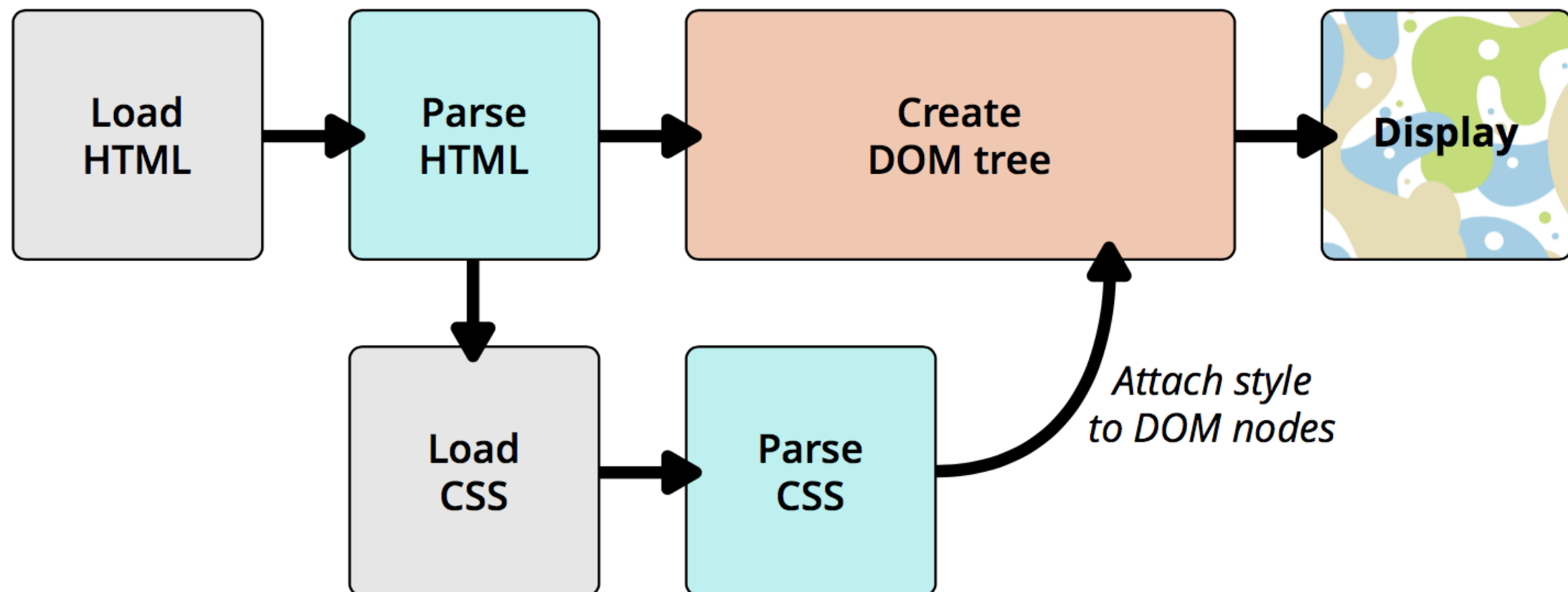


Estou cancelando a  
transferência ou cancelando o  
cancelamento da transferência?



## O DOM

- ▶ O navegador converte HTML e CSS para dentro do **DOM** (**D**omain **O**bject **M**odel). O DOM representa o documento na memória do computador (navegador).



# INTRODUÇÃO A HTML

- ▶ É a linguagem que o navegador interpreta para a exibição de conteúdo.
- ▶ HTML é uma sigla para **H**yper**T**ext **M**arkup **L**anguage.
- ▶ Cada elemento visual de uma página web consiste de uma ou mais tags.
- ▶ Normalmente, websites possuem centenas de tags, na maioria das vezes aninhadas.
- ▶ Exemplo: a página inicial do g1 possui mais de 4200 tags!

# INTRODUÇÃO A HTML

- ▶ Existem diversas tags, com os mais diversos propósitos.
- ▶ Exemplos: parágrafos, tabelas, links, imagens, rodapés, etc.
- ▶ Um elemento HTML consiste basicamente de um conteúdo encapsulado por uma tag de abertura e uma tag de fechamento.

## A ANATOMIA DE UM ELEMENTO HTML

```
<p> 0 time mais legal da NBA é o Los Angeles Clippers! </p>
```

Tag de abertura

Conteúdo

Tag de fechamento

- ▶ Tag de abertura: nome do elemento (p) envolvido entre colchetes angulares de abertura (<) e fechamento (>). Indica onde o elemento começa ou inicia o efeito.
- ▶ Conteúdo: pode ser um texto, ou outro elemento.
- ▶ Tag de fechamento: O mesmo de abertura, mas tem um / antes do nome do elemento. Indica o fim do elemento.

## ANINHANDO ELEMENTOS

```
<p> 0 time <strong> mais </strong> legal da NBA é o Los Angeles Clippers! </p>
```

Quando colocamos elementos dentro de outros elementos, chamamos isto de **aninhamento**. Para por exemplo, enfatizar uma palavra em negrito, podemos usar a tag strong.



```
<p> 0 time <strong> mais legal da NBA é o Los Angeles Clippers! </p> </strong>
```

Deve-se, entretanto, garantir que quando abrimos um elemento dentro de outro, o último elemento aberto **deve** ser o primeiro a ser fechado. O código acima, portanto, está errado.



## ERROS DE SINTAXE OU ABERTURA

No slide anterior, um elemento sobrepôs outro, o que é essencialmente *errado*. Entretanto, o navegador não *compila* código HTML e deixa de interpretá-lo caso algo esteja errado.

Ao invés disso, o navegador vai tentar *adivinhar* o que você quis dizer e exibir o conteúdo assim mesmo.

Isso gera resultados inesperados! **Nunca** deixe o seu código errado mesmo que ele apareça corretamente no navegador!

## TAGS SEM CONTEÚDO

Existem tags que são especiais ou já exibem algum elemento visual e não possuem conteúdo dentro delas. Neste caso, não há necessidade de se criar uma tag de fechamento.

Exemplos: imagens, inputs, meta tags, source de CSS, etc..

```
1 
2 <meta name="charset" content="utf-8">
3 <link rel="stylesheet" href="./css/master.css">
```

# ATRIBUTOS

Além dos elementos mostrados anteriormente, as tags também podem possuir atributos.

Os atributos adicionam peculiaridades e nos permite identificar e customizar cada tag.

Por exemplo: um campo de entrada (input) pode ser do tipo numérico, texto, senha ou de data. Não há necessidade de ter uma tag pra cada tipo de input, basta alterarmos um atributo.

# ATRIBUTOS

```
<input type="text" name="username" value="">  
<input type="password" name="senha" value="">
```

Os atributos seguem a sintaxe **nome="valor"**. As tags podem ter inúmeros atributos e o valor deve estar em torno de aspas.

```

```



Deve-se garantir o fechamento das aspas e o fechamento de um atributo antes de começar outro. Também é importante verificar por atributos duplicados na mesma tag.

## COMENTÁRIOS

A linguagem HTML permite a inserção de comentários delimitando-os com `<!--` e `-->`

Qualquer elemento dentro das tags delimitadoras de comentários será ignorada pelo browser e não será interpretada de qualquer maneira.

```
<!-- Aqui eu insiro um comentário -->  
  
<!-- <input type="password" name="senha" value="">  
A tag acima não será exibida, pois está delimitada pelos comentários  
-->
```

## CASE-INSENSITIVE

A linguagem HTML é "case-insensitive", ou seja, não distingue letras minúsculas e maiúsculas.

Porém é considerado boa prática manter a marcação HTML em letras minúsculas.

```
<!-- Todas as tags abaixo são válidas. -->  
<H1> Aqui tem um título </h1>  
<p> Aqui tem um <sTrOnG> parágrafo </STRONG> </p>  
<IMG SRC= "./IMAGENS/LOGO.PNG">  
<!-- Porém os valores são case-sensitive... -->
```

## O ESQUELETO DE UMA PÁGINA HTML

- ▶ Uma página HTML contém algumas tags ~~obrigatórias~~ que define aspectos importantes.
- ▶ Podemos dizer que uma página HTML é delimitada pela tag `<html>` e é dividida em duas importantes seções: o cabeçalho (**head**) e o corpo (**body**).
- ▶ Além dessas seções, é comum declarar a tag **!DOCTYPE**, que é a primeira tag do arquivo, especificando qual versão do HTML está sendo usada. No contexto do curso, usaremos o HTML 5.

# O ESQUELETO DE UMA PÁGINA HTML

```
<!DOCTYPE html>
<html>
  <!-- Cabeçalho -->
  <head>
    <title>Título da página</title>
    <meta charset="utf-8">
  </head>

  <!-- Conteúdo da página -->
  <body>
    <h1>Hello world!</h1>
    <p>Página simples que só possui um título e um parágrafo.</p>
  </body>
</html>
```



## CABEÇALHO (HEAD)

- ▶ A seção head é somente de interesse do navegador e é onde se configura elementos como o título da página (obrigatório), codificação, dimensões suportadas, idioma, atributos para compartilhamento, etc.
- ▶ É o primeiro elemento a ser carregado pelo navegador. Dessa forma é comum utilizar essa seção também para carregar arquivos de estilização.

## CORPO (BODY)

- ▶ É a seção onde o navegador efetivamente renderiza seus elementos. É obrigatório ter ao menos um elemento dentro dele.
- ▶ O conteúdo é renderizado seguindo a ordem de declaração no arquivo. Então se você colocou um título e logo após um parágrafo, o parágrafo será renderizado após o título. Posteriormente veremos que é possível alterar a ordem de exibição com CSS.

## DESAFIO #0

- ▶ Crie um arquivo "index.html" de acordo com as especificações:
  - ▶ Defina o título da página com o seu nome
  - ▶ Insira um texto na tag body
- ▶ Abra o arquivo no browser.

# TÍTULOS E SUBTÍTULOS

- ▶ Títulos e subtítulos são elementos importantes e comuns.
- ▶ Em HTML existem 6 tipos de títulos, definidos do mais importante para o menos importante (de **<h1>** à **<h6>**).
- ▶ A tag **<h1>** é a mais importante, além do tamanho do título, também é mais relevantes para *crawlers* e outras ferramentas automatizadas.
- ▶ A tag **<h6>** é a menos relevante, e possui o menor tamanho comparado a **<h5>**, **<h4>**, **<h3>**, etc...

## PARÁGRAFOS, QUEBRA DE LINHA E DIVISORES

- ▶ Um parágrafo é delimitado pela tag **<p>**. O navegador já formata como um parágrafo devidamente.
- ▶ Para fazer uma quebra de linha (equivalente a um shift+enter nos editores de documentos), é utilizada a tag self-closed **<br>**.
- ▶ A tag **<hr>** cria uma linha que ocupa todo o seu campo delimitador horizontalmente, útil para dividir seções de textos.

## ÊNFASES

- ▶ A tag **<strong>** coloca o texto em negrito. Há uma tag correspondente (**<b>**) mas é mais fraca semanticamente. Além do seu estilo, ela é utilizada para dar uma ênfase forte ao seu conteúdo.
- ▶ A tag **<em>** coloca o texto em itálico. A tag **<i>** é a sua correspondente mais fraca. Além de colocar o texto em itálico, ela é utilizada para dar uma ênfase simples.
- ▶ A tag **<small>** coloca o texto em uma fonte menor, útil para descrições de imagens ou textos de ajuda

# LISTAS

- ▶ Existem dois tipos de listas: ordenadas e não-ordenadas.
- ▶ Para definir uma lista ordenada, se usa a tag **<ol>** e não-ordenada a tag **<ul>**. Cada elemento da lista é delimitado pela tag **<li>**.

```
<ol>
  <li>Lista</li>
  <li>Ordenada</li>
  <li>
    <ul>
      <li>Lista</li>
      <li>Não</li>
      <li>Ordenada</li>
    </ul>
  </li>
</ol>
```

## DIVISORES

- ▶ Existe tags com o simples intuito de dividir o conteúdo em blocos. Esses blocos, que podem ou não ter diferenças visuais, são importantes para depois podermos delimitarmos e estilizar blocos de elementos.
- ▶ Existem tags semanticamente corretas para cada situação, mas por enquanto basta usar a tag **<div>**.
- ▶ Alguns blocos que podem ser interessante ser delimitados em tags são: cabeçalhos, menus laterais, rodapés, etc.



## LINKS

- ▶ Para criar um link, basta usar a tag `<a>` e colocar no atributo `href` o destino.
- ▶ Pode-se usar o atributo `target` para definir onde esse link deve ser aberto ( `_self`, `_blank`, `_top`).

```
<!-- Abre em uma nova aba/janela -->
```

```
<a href="https://www.twitter.com" target="_blank">Twitter</a>
```

```
<!-- Abre na aba/janela atual. Equivalente a target="_self" -->
```

```
<a href="https://www.reddit.com">Reddit</a>
```

# IMAGENS

- ▶ Para inserir uma imagem, basta usar a tag `<img>` e colocar no atributo `src` o caminho da imagem.
- ▶ Esse caminho pode ser uma URL externa ou interna, com qualquer extensão válida.

```

```

## DESAFIO #1

- ▶ Crie um arquivo "index.html" de acordo com as especificações:
  - ▶ O título deve ter o nome de uma série que você goste.
  - ▶ Deve possuir um título com o nome da série e um subtítulo com um slogan que você desejar.
  - ▶ Deve possuir uma lista ordenada dos seus 5 personagens favoritos.
  - ▶ Deve possuir dois parágrafos, separados por uma linha horizontal, contendo a descrição da série, um link para a série na wikipedia, ao menos um elemento com ênfase forte e um com ênfase simples.
- ▶ Abra o arquivo no browser e inspeccione os elementos no modo desenvolvedor.

## DECLARANDO ESTILOS

- ▶ A tag HTML **<style>** define estilos para os elementos HTML.
- ▶ Estes estilos são definidos pela linguagem de estilização CSS.
- ▶ CSS é uma sigla para **C**ascading **S**tyle **S**heets (folha de estilo em cascata)
- ▶ A maioria das tags HTML já possuem estilos definidos por padrão. Eles podem ser substituídos, em cascata.

# INTRODUÇÃO A CSS

- ▶ A tag **<style>** pode declarar estilos diretamente no HTML.
- ▶ Você também pode declarar estilos com o atributo `style` em qualquer elemento HTML.
- ▶ O mais comum é carregar os estilos de um arquivo externo, com extensão `.css`, com a tag **<link>**.
- ▶ Pode ser definidos inúmeras tags de estilização, entretanto o último estilo definido para um elemento X é o que tem maior prioridade e sobrescreverá qualquer estilo em conflito declarado anteriormente.

# INTRODUÇÃO A CSS

- ▶ O mais comum é definir os estilos CSS na **<head>**.

```
<head>
  <title>Título da página</title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: red;
    }
  </style>
  <link rel="stylesheet" href="./css/master.css">
</head>
```

- ▶ Caso o arquivo *master.css* possuir alguma definição de cor do background para o elemento body, ele que será considerado. Regras que não derem conflito são "misturados".

# INTRODUÇÃO A CSS

- ▶ Uma regra de estilo é definida do tipo:  
**seletor** {  
    **propriedade**: *valor*;  
}
- ▶ Uma mesma regra pode ser reescrita diversas vezes para um mesmo seletor.
- ▶ Regras são aplicadas a um elemento com diversos seletores diferentes.

## UNIDADES DE MEDIDAS

- ▶ Para estilos de alturas, tamanhos de fontes, larguras, espaçamentos, margens, entre outras, é necessário usar uma unidade de medida.
- ▶ Pode-se usar unidades relativas ao “*container*” que o elemento está contido. Neste caso pode-se usar %.
- ▶ Existem também unidades relativas ao **tamanho da fonte**. Elas são *em*, *rem* e *ex*.
- ▶ E há as tradicionais unidades absolutas: *px*, *cm*, *in*, etc.



## UNIDADES DE MEDIDAS

- ▶ O ideal é utilizar **px** para tamanhos de fontes (propriedade size).
- ▶ E para o restante utilizar **rem**, que é a unidade relativa mas ao root. Ou seja, sempre será relativo ao body!
- ▶ Isso faz com que quando algum usuário com baixa visão ou que simplesmente quer dar um "zoom" na página, o layout todo corresponda.
- ▶ E é uma convenção de boa prática.

## CORES

- ▶ A forma mais simples de utilizar uma cor em CSS é usar o nome dela. Existem mais de 140 cores já definidas com seus nomes. Ex: blue, red, mediumaquamarine, gainsboro.
- ▶ Cada cor nomeada é associado a um valor hexadecimal. Este valor tem 6 caracteres e cada unidade de 2 caracteres define a força da sua cor RGB.

Ex: **#FF00AA** = máximo de cor vermelha, nada de verde e um pouco de azul.



## CORES

- ▶ Também se pode usar a sintaxe `rgb(x,y,z)` e `rgba(x,y,z,o)` onde `o` é a opacidade (canal alpha).
- ▶ Há também o formato HSL, HSV e VEC3 (pouco utilizado).

# SELETORES

- ▶ Os seletores são, basicamente, o caminho para um ou mais elementos.
- ▶ Para facilitar a identificação de elementos, pode se definir classes e identificadores únicos nos seus elementos (**class** e **id**).
- ▶ A class pode ser reutilizada em inúmeros elementos e é identificado no css com o prefixo "."
- ▶ A id é identificada no css com o prefixo "#".

# SELETORES

- ▶ Para o CSS não importa se o ID é único ou não.
- ▶ Mas para o JavaScript importa... e importa muito!
- ▶ Além disso existem validadores de HTML que acusam caso você utilize um mesmo ID para mais de um elemento.
- ▶ Os seletores podem ter todo o caminho até um elemento, onde seus filhos são separados por espaço.
- ▶ Existem também separadores para irmãos, filhos diretos, etc.

# SELETORES

## CSS Selectors

| <u>Selector</u> | <u>Role</u>                            |
|-----------------|--|
| p{ }            | Tag selector, <b>all p tags</b>        |
| #para{ }        | Id para ( <b>unique</b> )              |
| .para1{ }       | Class para1 ( <b>multiple</b> )        |
| p.para{ }       | P tag with class para                  |
| P .para{ }      | P with child having class para         |
| div p{ }        | p tag having parent div.               |
| *{ }            | All tags{ <b>Universal Selector</b> }  |
| h1, h3, h5{ }   | Only h1, h3 and h5 ( <b>grouping</b> ) |
| .para a{ }      | A with parent para class               |
| body{ }         | Parent of all tags                     |

## PSEUDO-CLASSES E PSEUDO-ELEMENTOS

- ▶ Pseudo-classes são seletores que seleciona um elemento baseado no seu elemento que não está presente no DOM. Exemplo: `a:visited` vai capturar todos os links visitados.
- ▶ Pseudo-elementos por sua vez são seletores que seleciona entidades que não estão presentes no HTML. Ex: `p::first-line` captura a primeira linha de um parágrafo.

## LIVE-CODING

- ▶ Aprendendo algumas propriedades CSS, posicionamento, largura, altura, bordas, espaçamento, margens, floats e etc.
- ▶ Utilize o navegador para brincar com as regras de alguma página!



## DESAFIO #2

- ▶ Modifique o HTML do desafio anterior, dividindo as seções devidas e adicionando as devidas classes e ids necessários.
- ▶ Defina o tipo e tamanho da fonte e um background na raiz do documento.
- ▶ Estilize a lista para que o primeiro elemento seja exibido em negrito. Os elementos de índice ímpar deve ter a cor vermelha.
- ▶ Defina bordas coloridas para todas os divisores definidos. Garanta que cada um tenha um espaçamento grande do outro.
- ▶ Estilize o título para que tenha uma fonte diferente da usada no resto do corpo.

## DESAFIO #3

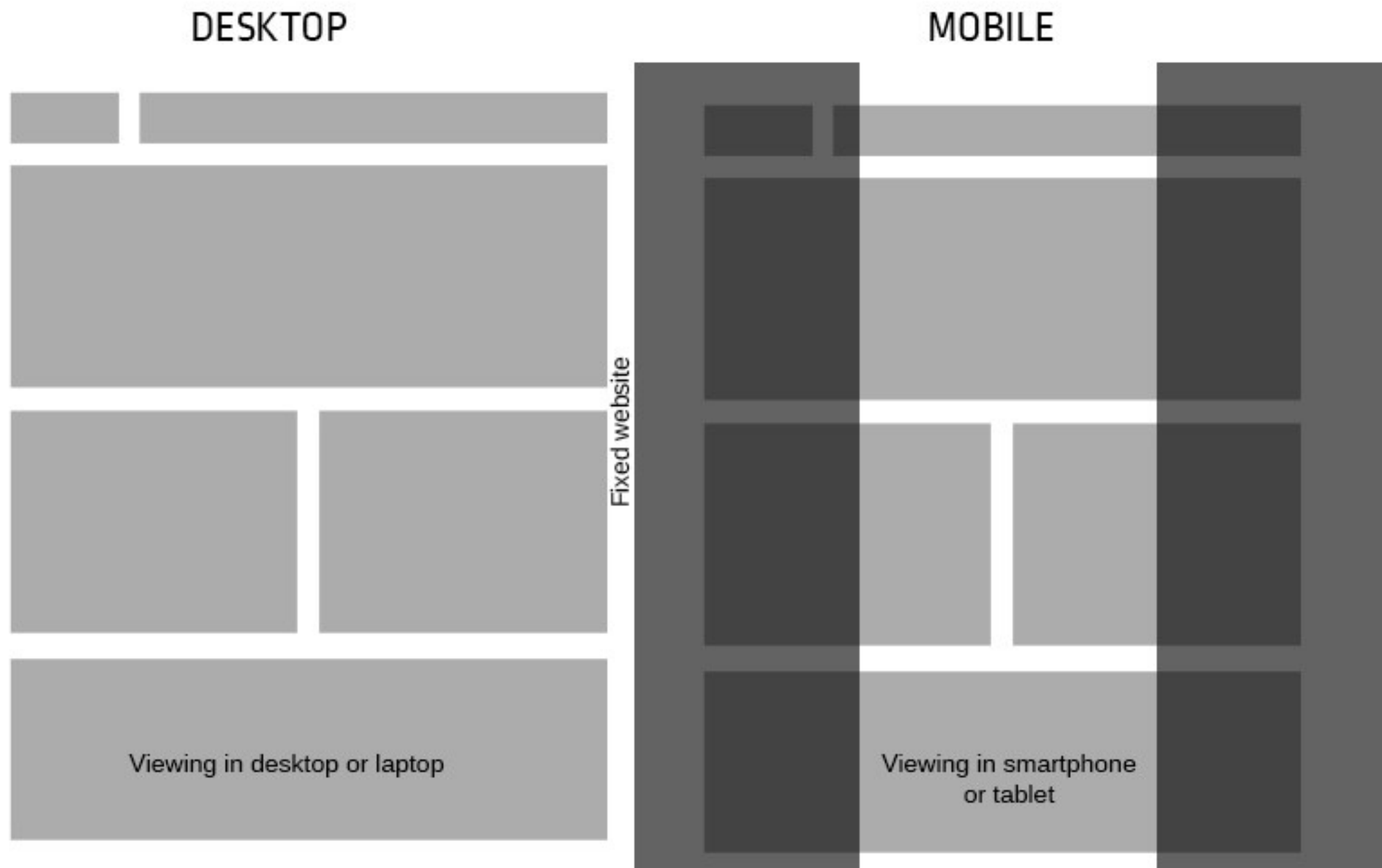
- ▶ Implemente o layout ao lado, para um jornal de Fake News.
- ▶ Os blocos verdes e amarelo devem ter parágrafos de texto.
- ▶ O bloco azul deve ser um rodapé com dados de contato.
- ▶ O bloco azul deve ter uma logo e um menu.



## LAYOUT FIXO, FLUIDO, ADAPTÁVEL E RESPONSIVO

- ▶ Abra o desafio anterior e simule ele no modo responsivo do navegador. Como ele se comportou?
- ▶ Os layouts podem se comportar de 4 maneiras em dispositivos mobile.
- ▶ Se deve prestar bastante atenção à isso hoje, já que a maioria massiva de usuários vem de dispositivos móveis.

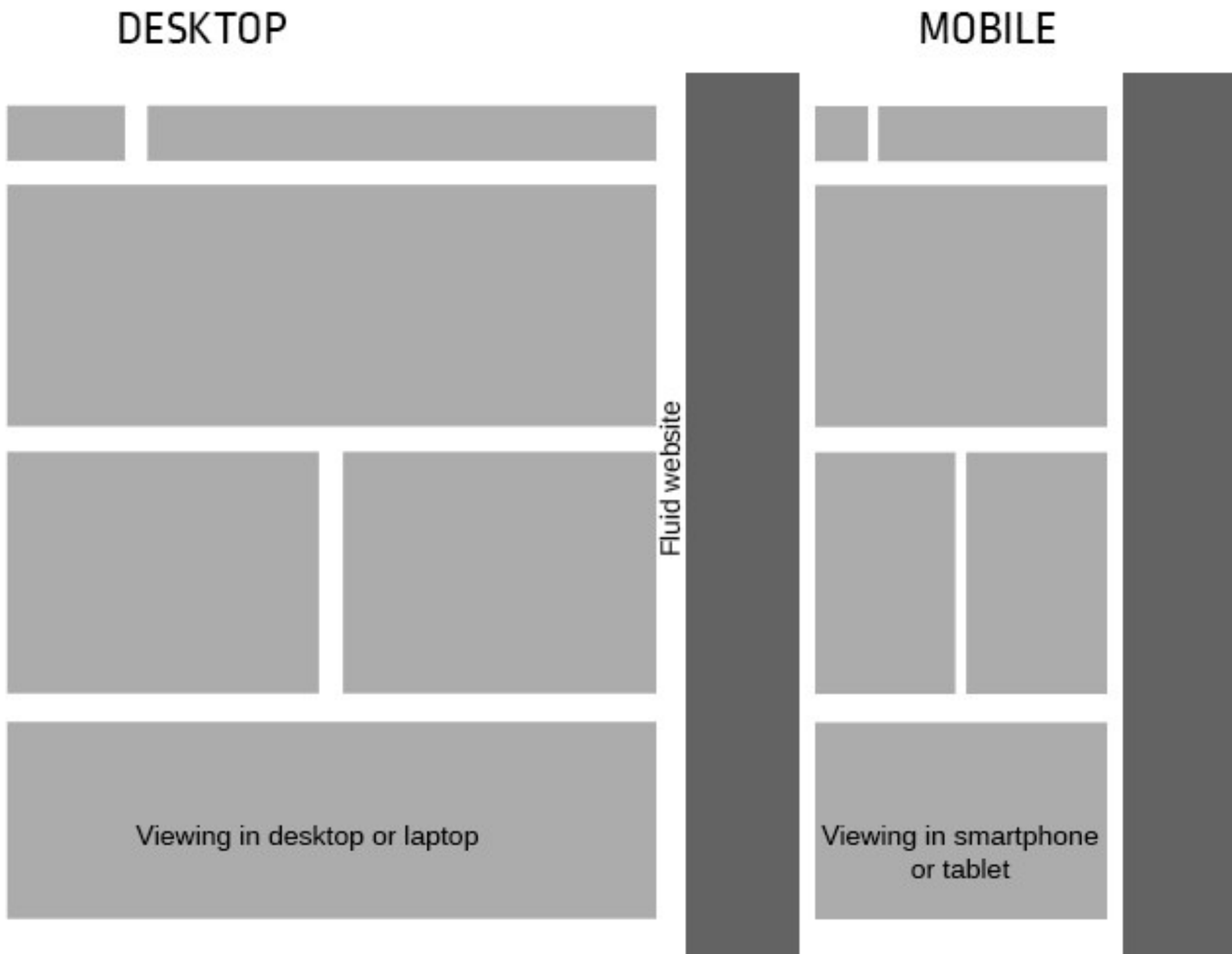
# LAYOUT FIXO



## LAYOUT FIXO

- ▶ Fáceis de se implementar. Não se importa com dimensões de imagens, fontes, etc.
- ▶ Não há definições de regras específicas pra o mobile.
- ▶ Um espaço gigantesco para se dar scroll no mobile.
- ▶ Usabilidade terrível!

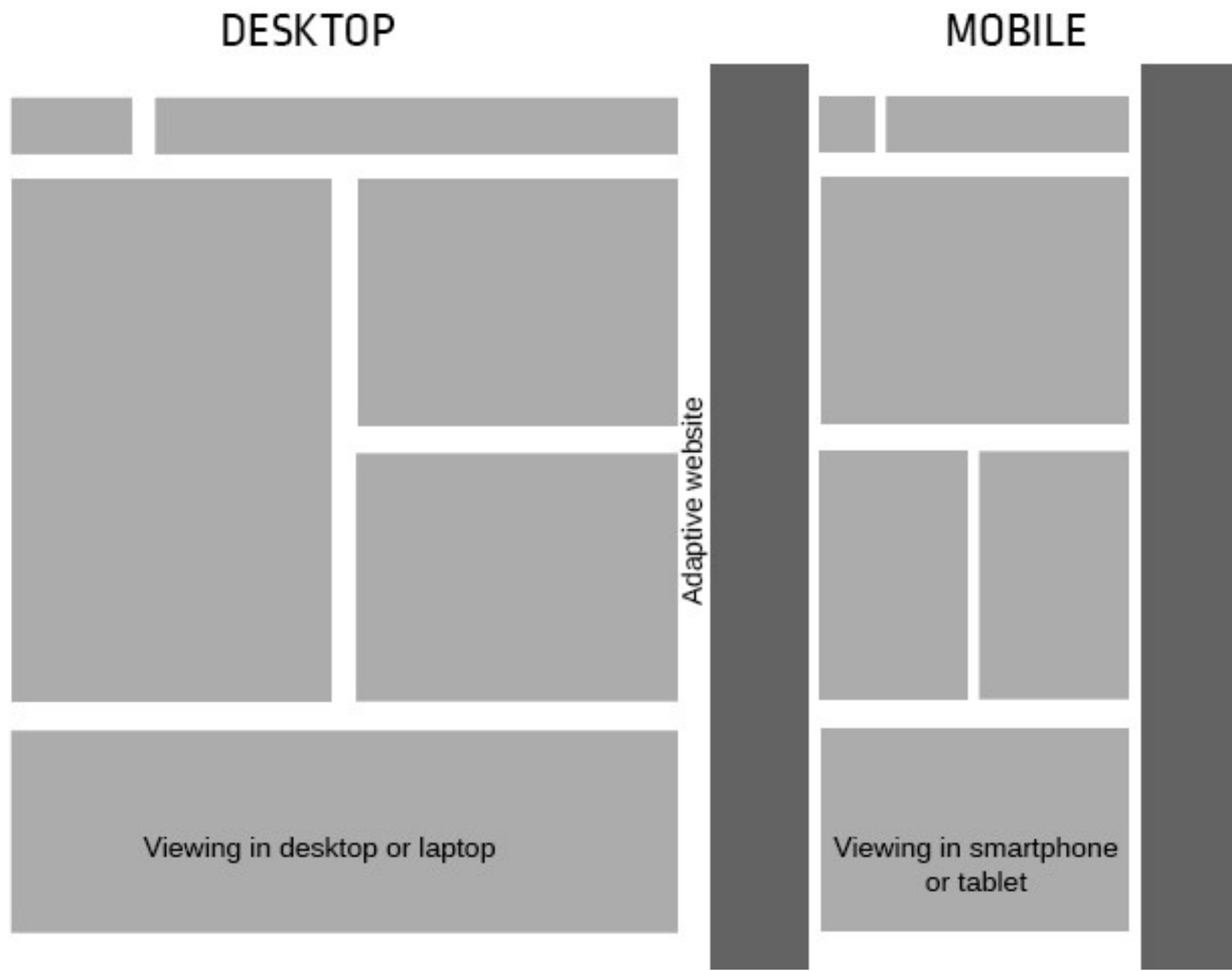
# LAYOUT FLUIDO



## LAYOUT FLUIDO

- ▶ Bem mais amigável do que o fixo.
- ▶ Há definições de regras específicas pra o mobile.
- ▶ O layout é idêntico em qualquer dispositivo, podendo ele então ficar muito "esticado" ou muito "encolhido".
- ▶ Usabilidade razoável, dependendo da situação.

# LAYOUT ADAPTÁVEL

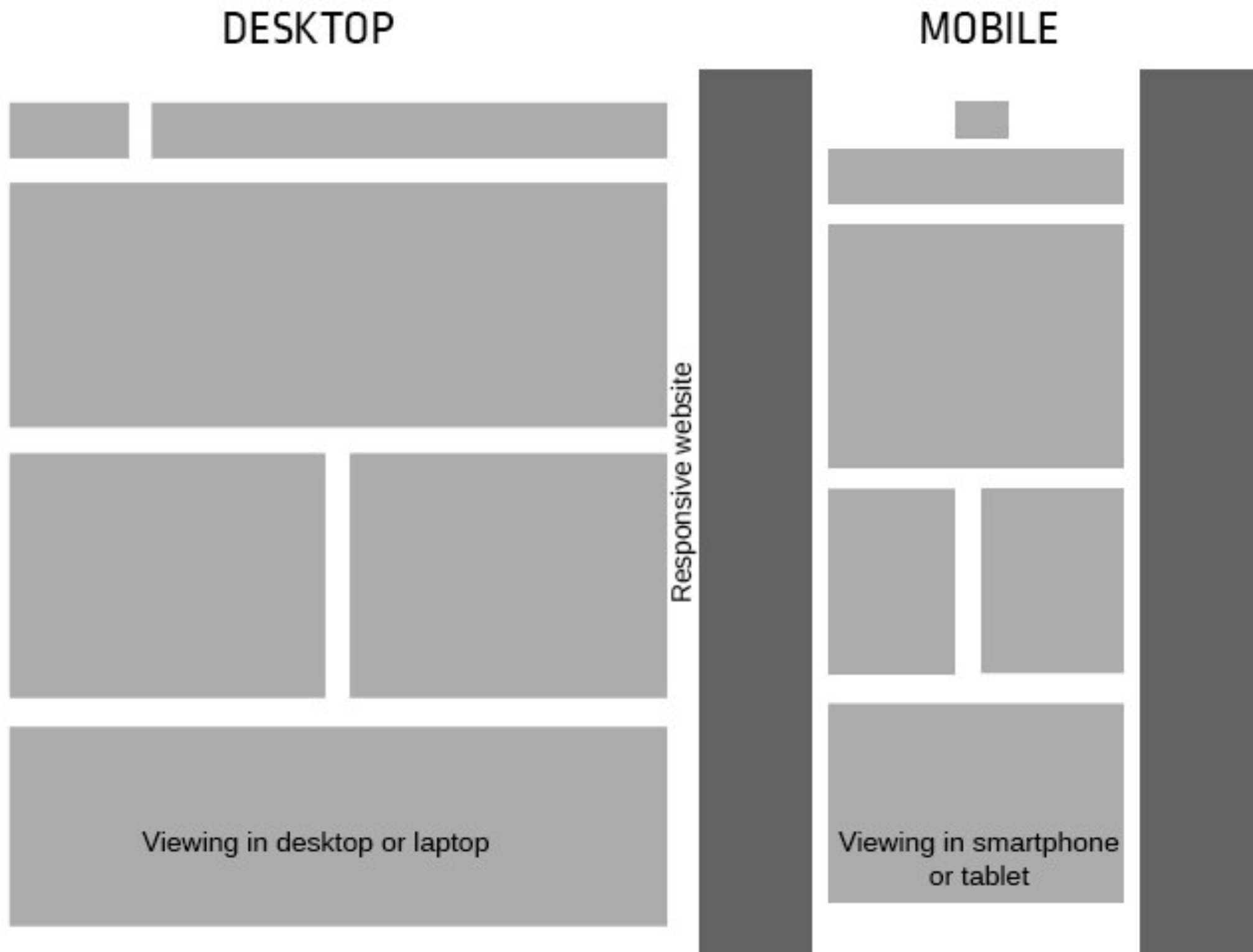




## LAYOUT ADAPTÁVEL

- ▶ Trabalho em dobro (ou mais) para o desenvolvedor.
- ▶ Tem que se ajustar ao "trend" de tamanhos de dispositivos.
- ▶ Múltiplos layouts devem ser carregados mesmo se não for utilizado.
- ▶ Ausência de reusabilidade e atomicidade.
- ▶ Apesar disso, é completamente amigável para os dispositivos móveis.

# LAYOUT RESPONSIVO



## LAYOUT RESPONSIVO

- ▶ Um mesmo layout se ajusta para **qualquer** tamanho e tipo de dispositivo.
- ▶ Tem desempenho muito melhor, já que carrega apenas um layout.
- ▶ Experiência de usuária ideal.
- ▶ Mais trabalhoso e difícil de se implementar.

# ADAPTIVE AND RESPONSIVE WEB DESIGN

## INFOGRAPHICS

### ADAPTIVE

Server use HTML, which is pre-selected for different devices with different screen size.



Information is pre-selected and only specific device based information will be displayed



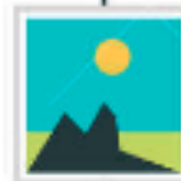
Templates are optimized for each device



### DEVICE DETECTION



### CONTENT OPTIMIZATION



### DEVICE OPTIMIZATION



### RESPONSIVE

Devices are detecting by "media queries". Flexible grid and images are sized correctly to screen of devices.



All content is downloading whether it is uses or not



One template for all devices



BILLIONS



Desktop Internet Users




Mobile Internet Users

UM EXEMPLO...

New York, NY

Tuesday, April 15th

Overcast



58°F


Precipitation: 100%

Humidity: 97%

Wind: 4 mph SW

Pollen Count: 36


Today



68°  
36°

Pollen  
36


Wednesday



50°  
39°

Pollen  
36


Thursday



55°  
39°

Pollen  
36

Friday



54°  
43°

Pollen  
36

## MEDIA QUERIES

- ▶ Media Queries são a resposta para o desenvolvimento de layouts para dispositivos móveis!
- ▶ Não só para dispositivos móveis, mas para diferentes tamanhos de dispositivos, como os 4K, televisões, etc.
- ▶ Permitem identificar o tamanho do viewport do usuário, se está em landscape ou não, se possui suporte a cores, qual tipo de dispositivo, razão de pixels, etc..
- ▶ Estilos definidos para cada tipo de dispositivo que são carregados, mas não aplicados.

# MEDIA QUERIES

- ▶ **@media** [(expressões booleanas)] {  
    regras...  
}

```
<!-- CSS media query em um elemento de link -->
```

```
<link rel="stylesheet" media="(max-width: 800px)" href="example.css" />
```

```
<!-- CSS media query dentro de um stylesheet -->
```

```
<style>
```

```
  @media (max-width: 600px) {
```

```
    .logo {
```

```
      display: none;
```

```
    }
```

```
  }
```

```
</style>
```

## MEDIA QUERIES

- ▶ As expressões booleanas para media queries são variadas. As mais importantes são:
  - ▶ max-width: valor, min-width: valor
  - ▶ orientation: landscape / orientation: portrait
- ▶ Além disso tem palavras chaves como **not**, **only**, **all**, **and**, etc..
- ▶ **@media** (min-width: 700px), handheld **and** (orientation: landscape) { ... }



## MOBILE-FIRST

- ▶ Um dos maiores desafios do desenvolvimento responsivo.
- ▶ A ideia é declarar as regras CSS primeiro para dispositivos mobiles e tratar o desktop como exceção (nas media queries).
- ▶ Desta forma, desenvolve o layout todo pensando em sua otimização pra mobile!

## META VIEWPORT

- ▶ Para definir a janela de visualização, existe uma meta tag (viewport) a ser inserida no cabeçalho. Use-a para controlar a largura e o dimensionamento da janela de visualização dos navegadores.
- ▶ Inclua **width=device-width** para corresponder à largura da tela em pixels independentes de dispositivo.
- ▶ Inclua **initial-scale=1** para estabelecer uma relação 1:1 entre pixels CSS e pixels independentes de dispositivo.
- ▶ Garanta que sua página seja acessível não desativando o dimensionamento do usuário.

## viewport not set

This page does **NOT** have the meta viewport set.

This is a test to show how the meta viewport tag affects a page. To simplify this example as much as possible, font boosting has been disabled.

**Note:** each box on the background is 50px wide.

## viewport set

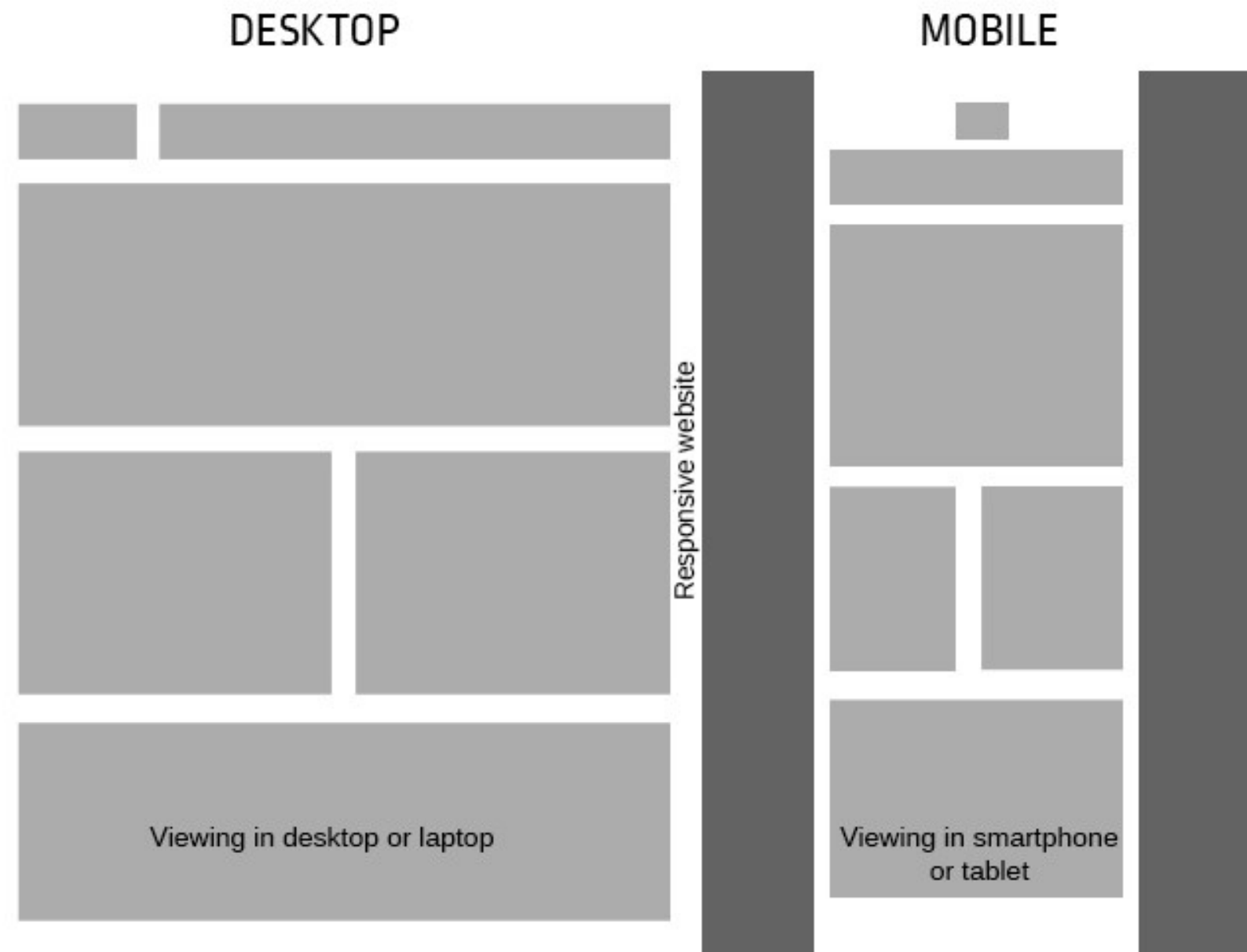
This page **DOES** have the meta viewport set.

This is a test to show how the meta viewport tag affects a page. To simplify this example as much as possible, font boosting has been disabled.

**Note:** each box on the background is 50px wide.

## DESAFIO #4

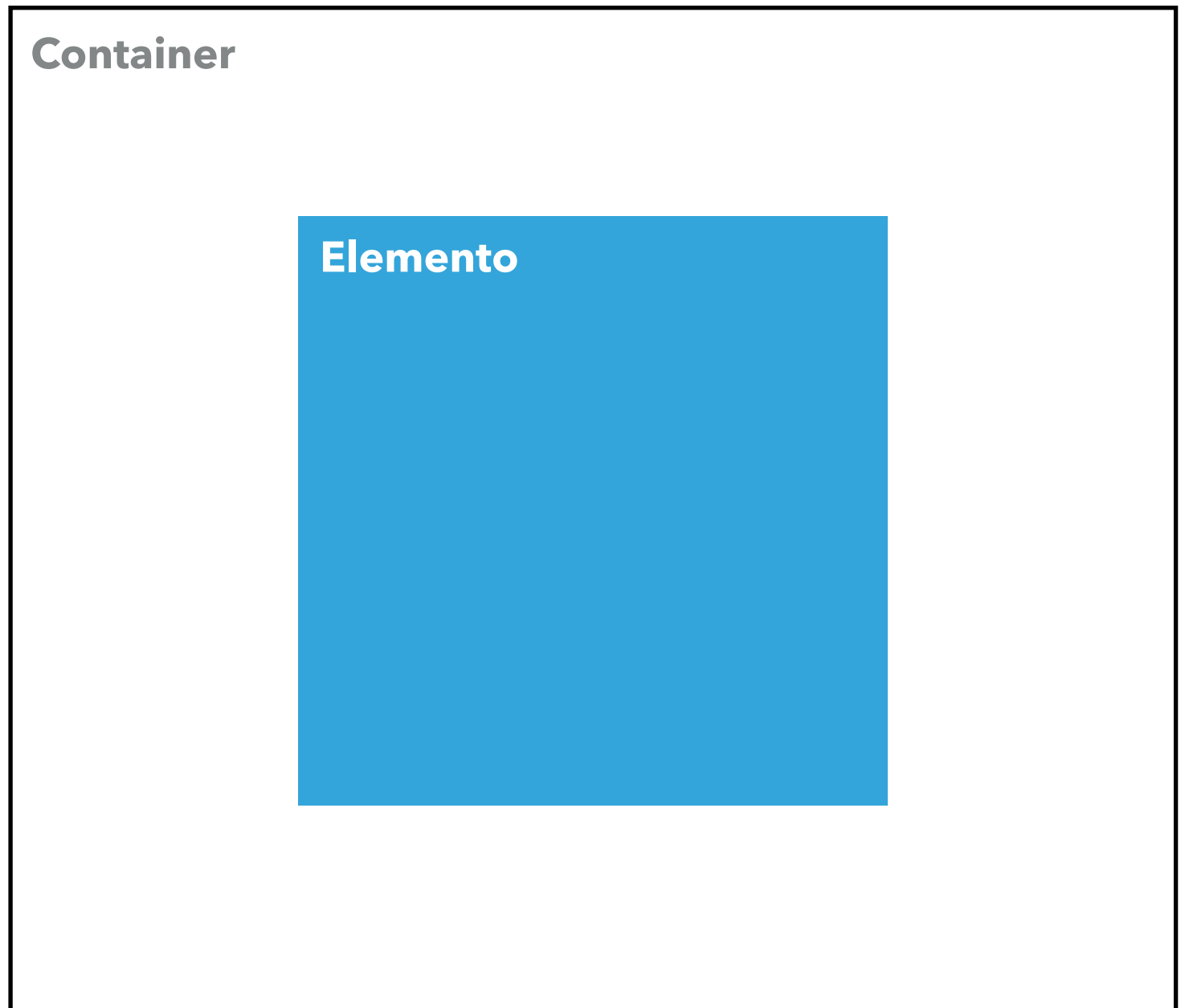
- ▶ Altere o layout do desafio #3 para ficar da forma ao lado.
- ▶ Utilize media queries para ficar fluido também em dispositivos 4k.
- ▶ Para um desafio maior: refaça o layout mobile-first.



## DESAFIO #5

- ▶ Desafio muito simples:

Faça uma reprodução do seguinte layout, com divs de larguras e alturas fixas:

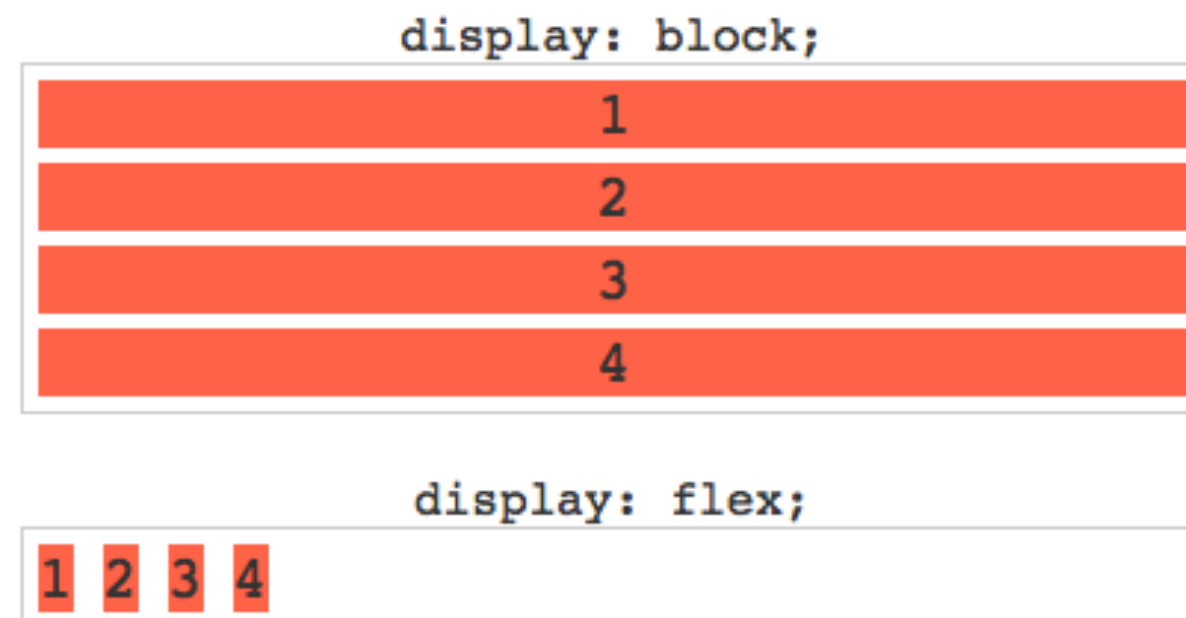


## FLEXBOX!

- ▶ O CSS 3 introduziu o maravilhoso *flexbox*, seu novo melhor amigo no desenvolvimento web.
- ▶ Modelo de layout unidimensional: um método que oferece distribuição de espaço entre itens numa interface e capacidades poderosas de alinhamento.
- ▶ É constituído de dois principais elementos: **Flex Container** e **Flex Item**.

## FLEX CONTAINER

- ▶ O Flex Container é a tag que envolve os itens flex, ao indicar **display: flex**, essa tag passa a ser um Flex Container.
- ▶ Todos os filhos diretos de um *Flex Container*, são por padrão, um *Flex Item*, que será estudado em sequência.



## CONTAINER – DIRECTION

- ▶ Define a direção dos Flex Items. Por padrão ele é row (linha), por isso quando o **display: flex**; é adicionado, os elementos ficam em linha, um do lado do outro.
- ▶ A mudança de **row** para **column** geralmente acontece quando estamos definindo os estilos para o mobile. Assim você garante que o conteúdo seja apresentado em coluna única.



## CONTAINER – DIRECTION

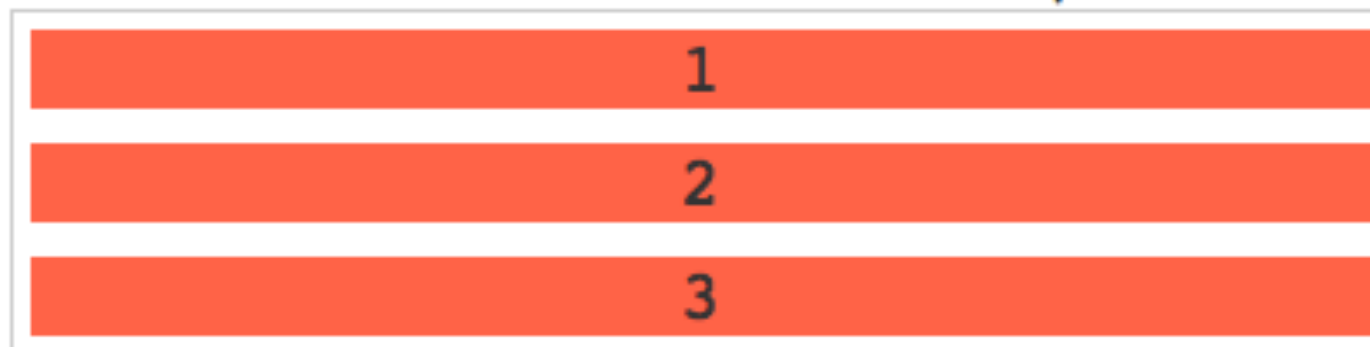
`flex-direction: row;`



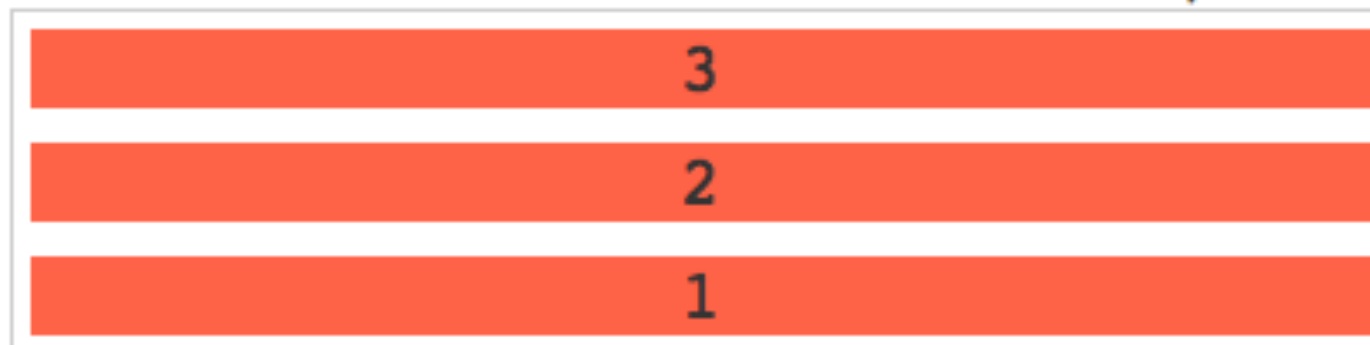
`flex-direction: row-reverse;`



`flex-direction: column;`



`flex-direction: column-reverse;`



## CONTAINER – WRAP

- ▶ Define se os itens devem quebrar ou não a linha. Por padrão eles não quebram linha, isso gera overflow de conteúdo.
- ▶ Essa é geralmente uma propriedade que é quase sempre definida como **flex-wrap: wrap**; Pois assim quando um dos flex itens atinge o limite do conteúdo, o último item passa para a coluna debaixo e assim por diante.

`flex-wrap: nowrap;`

TesteDoItem1 TesteDoItem2 TesteDoItem3

`flex-wrap: wrap;`

TesteDoItem1 TesteDoItem2

TesteDoItem3

`flex-wrap: wrap-reverse;`

TesteDoItem3

TesteDoItem1 TesteDoItem2

## CONTAINER – JUSTIFY CONTENT

- ▶ Alinha os itens de acordo com a direção.
- ▶ Excelente propriedade para ser usada em casos que você deseja alinhar um item na ponta esquerda e outro na direita, como em um simples header com marca e navegação.

# CONTAINER – JUSTIFY CONTENT

`justify-content: flex-start;`

1   Teste 2   3   4

`justify-content: flex-end;`

1   Teste 2   3   4

`justify-content: center;`

1   Teste 2   3   4

`justify-content: space-between;`

1   Teste 2   3   4

`justify-content: space-around;`

1   Teste 2   3   4

`justify-content: flex-start; // column`

1

Teste 2

3

`justify-content: flex-end; // column`

1

Teste 2

3

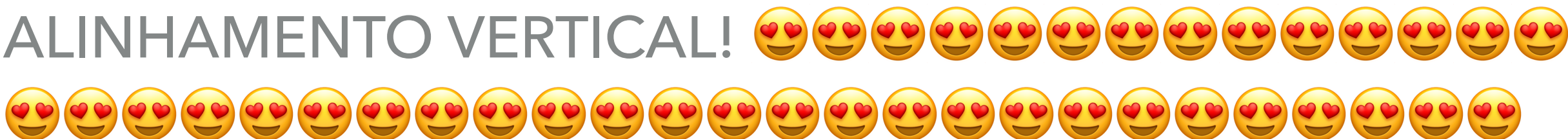
`justify-content: center; // column`

1

Teste 2

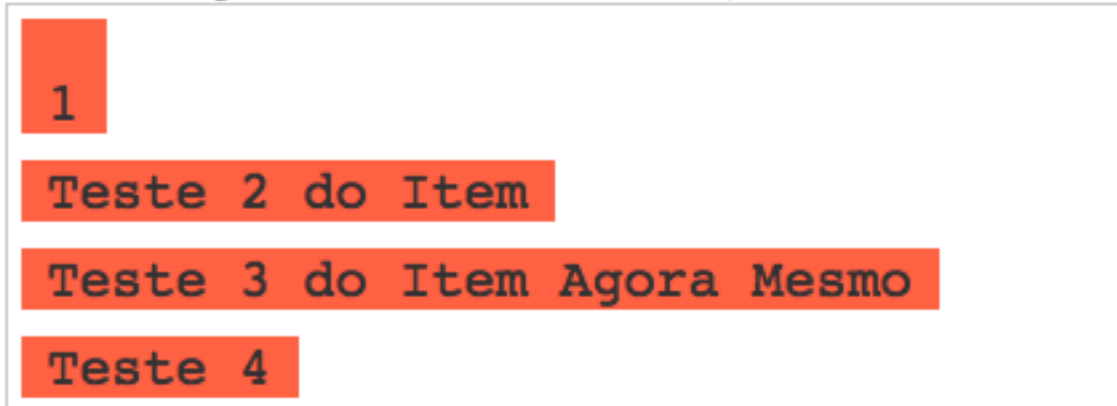
3

## CONTAINER – ALIGN ITEMS

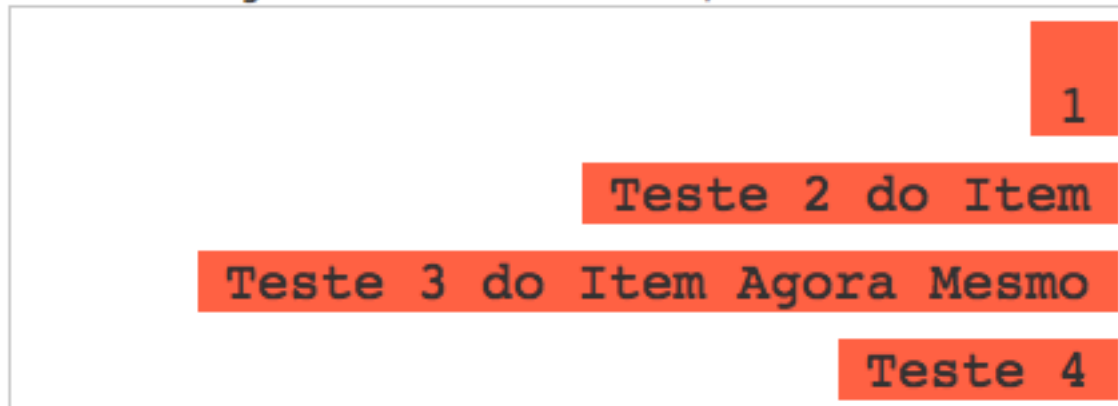
- ▶ O align-items alinha os itens de acordo com o eixo do container. O alinhamento é diferente para quando os itens estão em colunas ou linhas.
- ▶ E aqui está a resposta da pergunta de **R\$1.000.000,00!!!**
- ▶ ALINHAMENTO VERTICAL! 
- ▶ Valor padrão: *stretch*. Faz com que os elementos cresçam igualmente.

# CONTAINER – ALIGN ITEMS

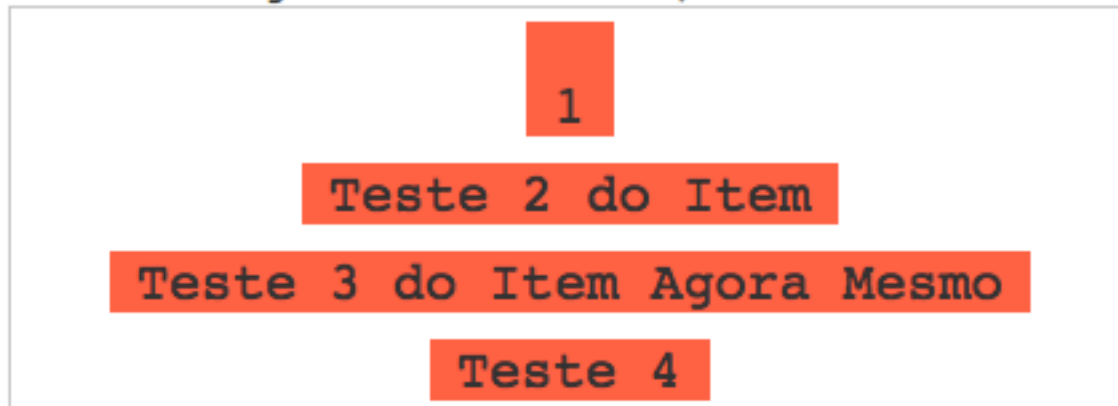
`align-items: flex-start; // column`



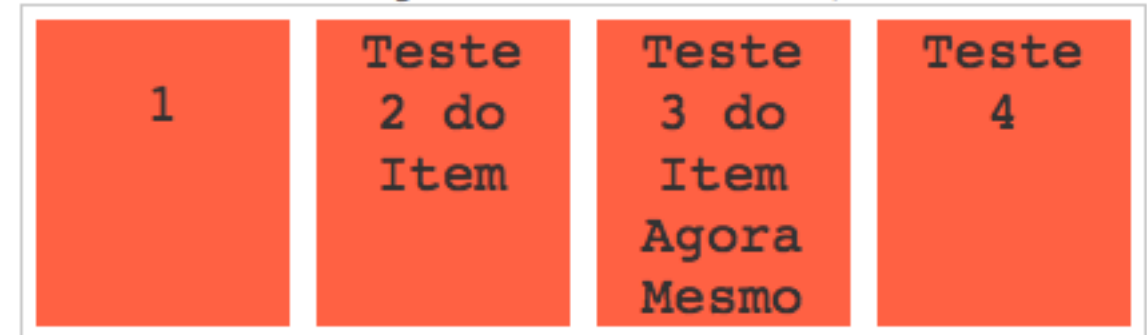
`align-items: flex-end; // column`



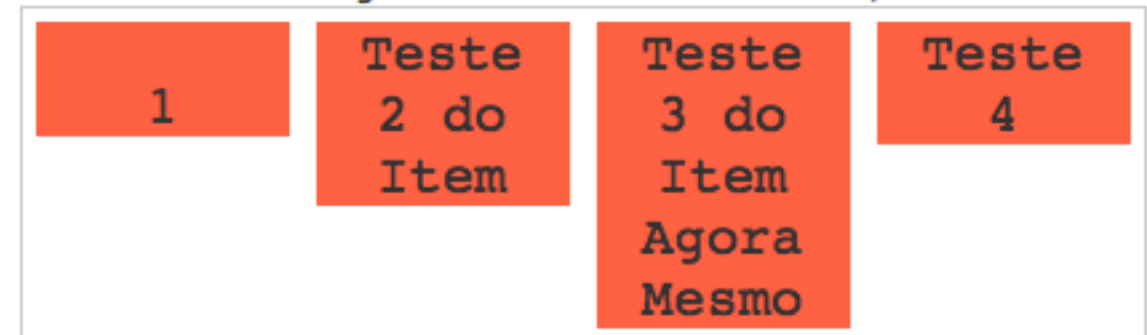
`align-items: center; // column`



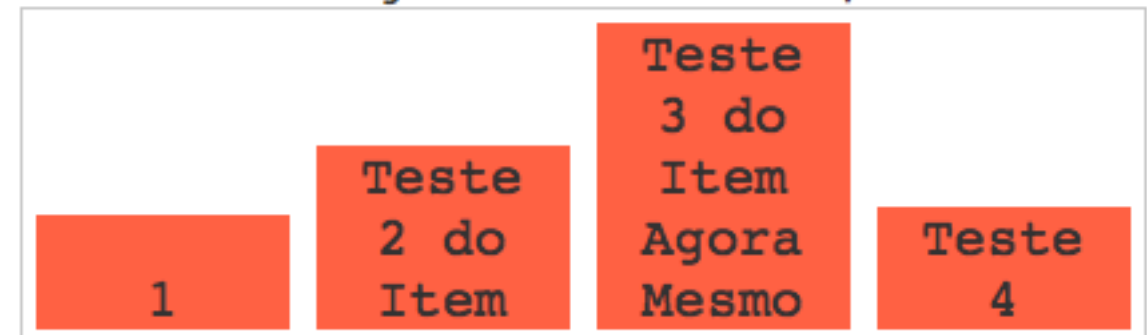
`align-items: stretch;`



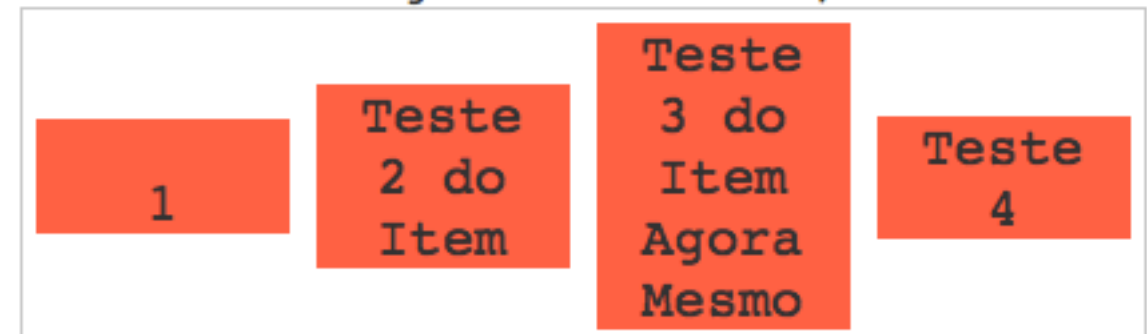
`align-items: flex-start;`



`align-items: flex-end;`



`align-items: center;`




## CONTAINER – ALIGN ITEMS

- ▶ O que será que acontece se misturarmos **align-items:** **center** e **justify-content: center** ?

## CONTAINER – ALIGN ITEMS

```
align-items: center; e justify-content: center;
```



Esse item  
está  
verticalmente  
alinhado ao  
centro.



## CONTAINER – ALIGN CONTENT

- ▶ Alinha as linhas do container em relação ao eixo vertical. A propriedade só funciona se existir mais de uma linha de itens.
- ▶ Além disso o efeito dela apenas será visualizado caso o container seja maior que a soma das linhas dos itens.
- ▶ Portanto, o height deve ser definido.

# CONTAINER – ALIGN CONTENT

`align-content: stretch;`



`align-content: flex-start;`



`align-content: center;`



`align-content: space-between;`



## FLEX-ITEM

- ▶ Todo elemento filho de um *Flex Container* é um *Flex Item*.
- ▶ Ele também pode ser um container, podendo-se criar cascatas de *Flex Containers*.
- ▶ Principais regras: **grow**, **shrink**, **order** e **align-self**.
- ▶ Consulte o [Cheatsheet](#)!
- ▶ Desafio adicional: conclua o [Flexbox Froggy](#).

## DESAFIO #6

- ▶ Use canhões, explosões e sua honra para defender a sua torre em um jogo de Tower Defense!
- ▶ Para ganhar, você deve usar Flexbox.
- ▶ Objetivo final: Finalize a onda final (12) do [Flexbox Defense](#).

## HTML SEMÂNTICO

- ▶ A responsabilidade do HTML **NÃO** é a apresentação final e detalhes de design.
- ▶ O HTML precisa ser **claro e limpo**, focado em marcar o conteúdo.
- ▶ Ao estilizarmos as nossas *divs* é bem claro o que cada elemento representa.
- ▶ E se desabilitarmos o CSS? Como distinguir?

## HTML SEMÂNTICO

- ▶ A tag **<p>** marca um parágrafo e possui tanto valor semântico como visual, já que o browser tem regras de estilo para o parágrafo.
- ▶ Lembra da tag **<b>** e **<i>**? Eles possuem valor puramente visual, algo demarcado como B ou I não tem valor semântico.
- ▶ Algumas tags são erroneamente utilizadas apenas para o benefício visual, como a tag **<blockquote>** que indenta o texto. Outros exemplos: **<h1 ~ h6>**.

# HTML SEMÂNTICO

- ▶ E a tag **<div>**?
- ▶ Existem diversas alternativas semânticas. **<header>**, **<section>**, **<article>**, **<aside>**, **<nav>**, **<footer>**, **<main>**, etc...
- ▶ As imagens também ganham destaque com as tags **<figure>** e **<figcaption>**.

# HTML SEMÂNTICO

```
<body>

  <header>
    <h1>Samba Nacional</h1>
  </header>

  <main>
    <section>...
    <section>...
  </main>

  <footer>
    <p>musicos.com.br. Todos os direitos reservados.</p>
  </footer>
</body>
```



# HTML SEMÂNTICO

```
<main>
  <section>
    <h2>Alcione</h2>
    <figure>
      
      <figcaption>Alcione se apresentando em Campo Grande, 2009.</figcaption>
    </figure>
    <article>
      <h3>Os maiores hits</h3>
      <p>Conheça as maiores obras de uma das maiores artistas do Brasil.</p>
    </article>
    <article>
      <h3>Prestigiada</h3>
      <p>Conheça os prêmios que a Marrom venceu em sua longa carreira internacional.</p>
    </article>
  </section>
  <section>
    <h2>Baden-Powell</h2>
    <figure>
      
      <figcaption>Baden-Powell e Vinicius de Moraes apresentando o Canto de Ossanha.</figcaption>
    </figure>
    <article>
      <h3>Histórico</h3>
      <p>Um dos maiores instrumentistas da história, conhecido por sua harmonia impecável no violão</p>
    </article>
    <article>
      <h3>Vanguarda da Bossa e AfroSamba</h3>
      <p>Junto com Vinicius de Moraes, Tom Jobim e outros grandes nomes compôs a vanguarda da música nacional na origem do rádio nacional.</p>
    </article>
  </section>
</main>
```

# INTRODUÇÃO A JAVASCRIPT

- ▶ Linguagem de script multiparadigmas, multiplataformas.
- ▶ Possui bibliotecas padrão globais como **Date**, **Math**, **Array**, etc.
- ▶ Além destas, ela se estende para diversos propósitos como o Frontend e o Backend.
- ▶ O lado Frontend fornece objetos para controlar um navegador e o DOM.
- ▶ Exemplo: permitem que uma aplicação coloque elementos em um formulário HTML e responda a eventos do usuário, como cliques do mouse, entrada de dados e de navegação da página.

# INTRODUÇÃO A JAVASCRIPT

- ▶ Para utilizar JavaScript no frontend, basta declarar o código dentro das tags **<script>** **</script>**.
- ▶ Também se pode carregar um arquivo externo com extensão **.js** pelo atributo **src**.  
Exemplo: **<script src="./index.js"></script>**
- ▶ Normalmente declarado no fim do body.
- ▶ Output no console do modo desenvolvedor.

# INTRODUÇÃO A JAVASCRIPT

- ▶ Linguagem interpretada, com tipagem dinâmica, não podem acessar o disco rígido e é altamente dinâmica.
- ▶ Sintaxe muito similar com a de C++ ou Java.
- ▶ Execução **assíncrona!!**
- ▶ Variáveis podem ser declaradas com **var**, **let** e **const**.
- ▶ Variáveis podem ser utilizadas antes de ser declaradas. Neste caso são valores interpretados como **undefined**.
- ▶ Existem escopos de variáveis, entre elas, o escopo global.

# INTRODUÇÃO A JAVASCRIPT

- ▶ Tipos de dados:
  - ▶ Boolean: **true/false**.
  - ▶ Nulo: **null**.
  - ▶ Números: **Number**.
  - ▶ Strings: **String**.
  - ▶ Objetos: **Object**.
- ▶ Lembrete: dinamicamente tipada!

# OBJETOS JAVASCRIPT

- ▶ Linguagem *essencialmente* orientada a objetos.
- ▶ Existem diversos objetos úteis com propriedades e métodos convenientes. Exemplos: qualquer array.
- ▶ Acesso às propriedades com a mesma sintaxe de C++.  
**objeto.propriedade**
- ▶ Aninhamento de objetos e propriedades:  
**objeto.metodo().propriedade.metodo()**
- ▶ Mas Arrays podem ser acessados com seu índice. **array[90]**

```
const x = 90

function fazAlgo() {
  console.log(y) // undefined
  let y = "Teste"
  x = {legal: true} // Gera exceção
}

let fazAlgo = parseFloat("900.233") // Também gera exceção

const getId = function (n) {
  return Math.sqrt(n ** 3)
}

const animais = ["Leão", , "Lobo", "Peixe", , "Macaco"]

// Misturando tudo!
const dados = {
  animais: animais,
  id: getId(0),
  nome: "Teste"
}
```

# JSON

- ▶ Com estes conhecimentos de tipos de dados se introduz um formato de troca de dados, similar ao XML.
- ▶ Entretanto, é **muito** mais leve e legível.
- ▶ Vem da sigla **JavaScript Object Notation**.
- ▶ É basicamente a sintaxe de declaração de objetos JavaScript, porém com a obrigatoriedade de uso de aspas duplas em strings e sem acesso a variáveis globais da linguagem.



```
{
  "dados": [
    {
      "id": 1,
      "name": "Leanne Graham",
      "username": "Bret",
      "email": "Sincere@april.biz",
      "phone": "1-770-736-8031 x56442",
      "website": "hildegard.org",
      "company": {
        "name": "Romaguera-Crona",
        "catchPhrase": "Multi-layered client-server neural-net",
        "bs": "harness real-time e-markets"
      }
    },
    {
      "id": 2,
      "name": "Ervin Howell",
      "username": "Antonette",
      "email": "Shanna@melissa.tv",
      "phone": "010-692-6593 x09125",
      "website": "anastasia.net",
      "company": {
        "name": "Deckow-Crist",
        "catchPhrase": "Proactive didactic contingency",
        "bs": "synergize scalable supply-chains"
      }
    }
  ]
}
```

## DESAFIO #7

- ▶ Baixe o conteúdo do desafio no GitHub. Leia o código.
- ▶ Estilize a página como preferir.
- ▶ Adicione 2 músicas que você goste no JSON.
- ▶ Ao clicar em uma imagem, mostre também o título da música logo acima do vídeo com uma h2.
- ▶ Exiba também o autor da música do lado do título que você acabou de inserir

# PROMISES

- ▶ Uma Promise é um objeto que representa a eventual conclusão ou falha de uma operação assíncrona.
- ▶ Basicamente diz: "Faça essa operação e quando você concluir, faça esta (then). Se der algum erro, faça isto (catch)".
- ▶ Essencialmente, uma promise é um objeto retornado para o qual você adiciona callbacks, em vez de passar callbacks para uma função.

# PROMISES

## ► Com callbacks...

```
function doSomething (success, failure) {  
  const operationSuccessful = doSomethingElse()  
  // ....  
  
  if (!operationSuccessful) return failure()  
  return success()  
}  
  
function successCallback (result) {  
  console.log('Sucesso! Função retornou os seguintes dados: ' + result)  
}  
  
function failureCallback (error) {  
  console.log('Erro! Função retornou o seguinte erro: ' + error)  
}  
  
doSomething(successCallback, failureCallback)
```

# PROMISES

## ► Com promises...

```
function doSomething () {  
  return new Promise(function (resolve, reject) {  
    const operationSuccessful = doSomethingElse()  
    // ....  
  
    if (!operationSuccessful) return resolve()  
    return reject(new Error())  
  })  
}  
  
function successCallback (result) {  
  console.log('Sucesso! Função retornou os seguintes dados: ' + result)  
}  
  
function failureCallback (error) {  
  console.log('Erro! Função retornou o seguinte erro: ' + error)  
}  
  
doSomething()  
  .then(successCallback)  
  .catch(failureCallback)
```

## FETCH DE DADOS

- ▶ API disponível para manipular e gerar requisições HTTP.
- ▶ Retorna uma promise ao realizar um request.

```
function buscarDados (numero) {  
  window.fetch('https://numbersapi.com/' + numero + '/math')  
    .then(dados => {  
      console.log(dados)  
    })  
    .catch(error => {  
      console.error('Aconteceu o erro ' + error.message)  
    })  
}  
  
buscarDados(200)
```

## FRAMEWORKS

- ▶ Os frameworks existem pra deixar a vida do desenvolvedor mais fácil e produtiva.
- ▶ Entretanto, deve-se tomar cuidado com a quantidade de recursos externos e desnecessários sendo consumidos.
- ▶ Além disso, é essencial ter conhecimentos de desenvolvimento por si só e não apenas “montar bloquinhos” de códigos encontrados na internet.

## FRAMEWORKS

- ▶ Os frameworks CSS fornece um arsenal de componentes e estilizações prontas para aumentar a reusabilidade e padronização de seus componentes.
- ▶ Um dos principais perks dos frameworks CSS é o Grid, que nada mais é que a divisão da largura da página em blocos de mesma proporção.
- ▶ Os frameworks JS são diversos e tem diversos propósitos: desde ajudar a compor layouts até a manipular todos os dados do browser e até aplicativos híbridos.



## FRAMEWORKS

- ▶ Lembre-se cada kb inútil carregado conta. Embora pareça bobo, um framework comprimido tem em média **90kb**, o que não é muita coisa, certo?
- ▶ Mas se você tiver 100.000 usuários, este único framework consumiria mais de **8,5 GIGABYTES** de tráfego.
- ▶ Lembrando que cada kb além de custo de tempo, também custa \$\$ . Bastante \$ \$ .

## FRAMEWORKS

- ▶ Para utilizar um framework, basta carregar os seus arquivos necessários (CSS ou JS).
- ▶ Você pode baixar estes arquivos e importar localmente, ou colocar uma URL externa nas tags **<link>** e **<script>**.
- ▶ Dica: a maioria dos frameworks CSS dão suporte completo a SASS, já que foram construídos com ele. Dessa forma é extremamente trivial customizar o framework e utilizar apenas o que se necessita dele.

## DESAFIO #8

- ▶ Importe o framework CSS [Bulma](#) para um novo projeto. Ele deverá ser 100% responsivo.
- ▶ Utilizando os seus recursos faça o layout de uma página com: header com navigator, footer, um título e 12 cards de tamanho idêntico, organizados em 4 linhas.
- ▶ No navigator, coloque 4 links, pra 4 álbuns diferentes.
- ▶ Cada card deve mostrar uma imagem e uma descrição. Dica: procure componentes na documentação.

## DESAFIO #8

- ▶ Os links do nav deve mostrar a primeira palavra do nome do álbum que é buscado em uma requisição HTTP à API: [Endpoint de Albuns](#).
- ▶ Ao clicar em cada um dos links do nav, deve-se carregar as fotos referentes a este álbum em uma requisição HTTP à API: [Endpoint de Fotos](#) , passando o id na requisição. 1º link: id = 1, 2º link: id = 2, etc...
- ▶ Cada foto deve ser exibida em um card na página, limitando-se a 12 fotos.