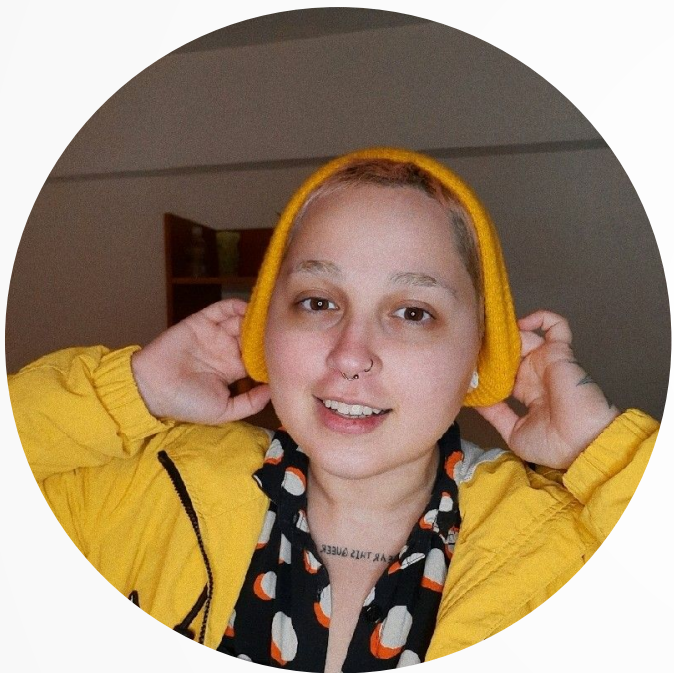




# Aprendendo CSS

Guia básico sobre CSS



# Kai Pimenta

Designer de formação e atualmente front end,  
participei do XP 34 como hacker.  
Trabalho com ecommerce, VTEX IO e React. :)



[linkedin.com/kai-pimenta](https://linkedin.com/kai-pimenta)



[github.com/kpmnta](https://github.com/kpmnta)

# O que vc vai ver nesse material



1. O que é CSS
2. Anatomia de um elemento CSS
3. Seletores CSS
4. Fontes e Textos
5. Unidades
6. Cor e Background
7. Display inline, block e inline-block
8. Display e Position
9. Espaçamento
10. Reset CSS
11. Layout Responsivo
12. Flexbox



# CSS



## O que é CSS

O CSS vem do inglês Cascading Style Sheet, ou seja, folha de estilos em cascada e é usado para estilizar páginas web. O nome de cascata se dá porque para trabalhar com o CSS é necessário seguir uma ordem de tags, classes ou IDs até chegar no componente que você quer estilizar.

O navegador lê seu arquivo de cima para baixo, ou seja a ordem que você coloca suas tags, classes e estilos são de extrema importância. Quando você tem duas regras que atingem o mesmo componente, o último a ser declarado vai ser o usado pelo navegador.



Para adicionar uma folha de de estilo no seu projeto é usado a tag <link> no <head> do HTML:

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

O atributo rel é usado para definir qual o relacionamento entre o documento atual e o documento linkado e o href indica o caminho para o documento.

Também é possível adicionar o CSS através da tag <style> ou adicionando um estilo inline, ou seja, direto na tag:

```
<h1 style="color: red;">Heading</h1>
```

O estilo inline deve ser usado com cuidado e pontualmente, já que ele tem mais prioridade do que o css exportado de um documento. Já a tag <style> muitas vezes é desaconselhada a ser usada para evitar poluir o seu html ou deixar ele muito mais pesado do que o necessário.

```
seletor  
p {  
    color: red;  
}  
propriedade valor
```

## Anatomia de um Elemento CSS

Para trabalhar com HTML devemos sempre usar as tags de marcação, abrindo e fechando a tag conforme necessário. Existem algumas tags que não precisam ser fechadas, abrem e fecham ao mesmo tempo, como por exemplo a tag `<img />`.



# Seletores CSS



## Por Tag

Para selecionar uma tag HTML só é necessário colocar o nome da tag, sem ponto, sem #, sem nada. por exemplo:

```
h1 {  
  
    color: red;  
  
}
```



## Por ID

Para selecionar um ID, você deve conferir no seu html o ID usado e colocar:

```
#seu-id {  
  
    color: red;  
  
}
```



## Por Classe

Para selecionar uma classe você deve conferir no seu html a classe e colocar:

```
.nome-da-classe {  
  
    color: red;  
  
}
```



## Atenção

Para selecionar um id sempre utilizamos o #, para as classes o ponto . e para tags html não usamos nada antes.



# Seletores CSS



## Combinando seletores

Imagine que nós temos o seguinte HTML:

```
<section class="minha-classe" >
  <div>
    <p> meu texto </p>
  </div>
  <p> meu texto </p>
</section>
```

Como poderíamos adicionar um estilo apenas para o <p> dentro da <div>?

Se a gente colocasse no nosso arquivo de css o seguinte:

```
p {
  color: red;
}
```

todos os <p> da página iriam ficar com esse estilo, que não é o que queremos, por isso podemos combinar seletores. Nesse exemplo poderíamos fazer o seguinte:

```
.minha-classe div p {
  color: red;
}
```





Existem muitas outras formas de concatenar seletores para especificarmos exatamente qual elemento queremos adicionar um estilo. Caso tenha interesse em se aprofundar ainda mais no assunto, sugiro dar uma olhada [nesse link](#)





# Fontes e textos

Definir as fontes e a formatação de textos em um projeto é uma das principais características que dão cara ao nosso website.

Algumas fontes já são nativas de navegadores, como por exemplo a Arial e a Verdana, mas muitas vezes queremos usar outras fontes mais diferenciadas e nesses casos sugiro procurar no [Google Fonts](#)

**Sobre a formatação do texto algumas propriedades são mais comuns:**

- font-family: onde você declara a fonte a ser usada.
- font-weight: qual o peso dessa fonte? Aqui colocarmos se é bold, light, normal, etc...
- font-size: qual o tamanho da fonte.

**Também podemos estilizar nosso texto de outras formas, como por exemplo:**

- text-transform: podemos colocar em caixa alta, caixa baixa...
- letter-spacing: dar espaçamento entre as letras
- line-height: altura de cada linha de texto



# Unidades

Você provavelmente já conhece o pixel, essa medida é muito comum quando falamos sobre a web mesmo quando não desenvolvemos, já no css a gente usa “px” quando estamos usando essa unidade, por exemplo:

```
h1 {  
    font-size: 12px;  
}
```

Também é comum usar o vh (viewport height) e o vw (viewport width), ajudam principalmente em layouts responsivos, uma vez que essas unidades calculam o tamanho do dispositivo e a partir daí coloca uma altura ou largura relativa, evitando que o seu layout quebre em determinados tamanhos.



## 01.

### EM



EM é mais uma unidade relativa, isso significa que por exemplo, se você coloca que todo o `<body>` tem uma `font-size: 14px;` e depois colocar que uma `<div>` que está dentro desse body tem uma `font-size: 1.5em`, ela vai ter 21px uma vez que  $1.5(em) \times 14(px) = 21$

#### Atenção:

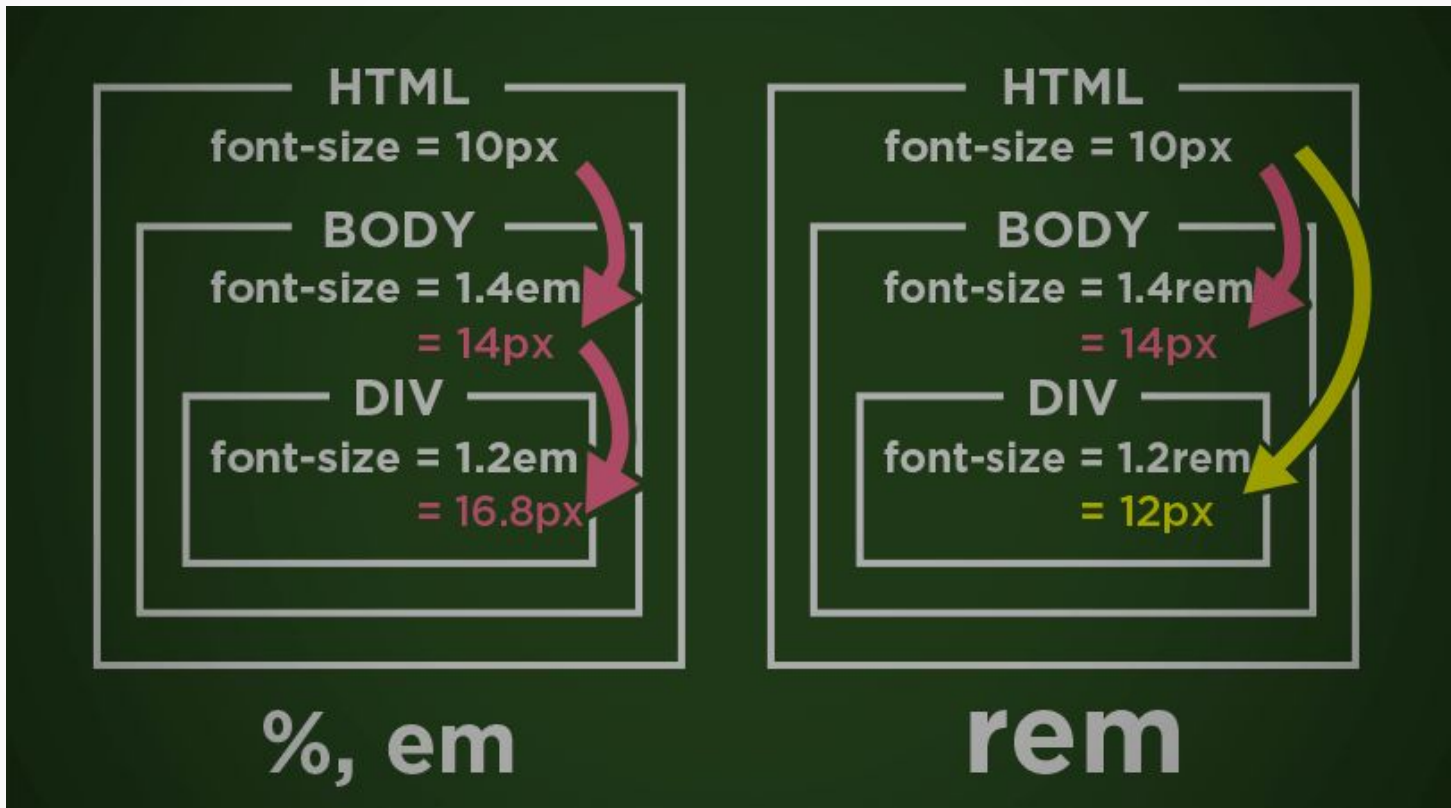
Quando você usa a unidade EM tome cuidado pois o CSS é um estilo em cascata então toda propriedade com tamanho relativo é relativo ao elemento pai dele, que é relativo ao pai e assim por diante.

## 02.

### REM



Para solucionar o problema que o `em` acarreta, também temos a unidade chamada de `rem`, que funciona muito similar ao `em` mas o R vem de root, ou seja, é relativo a raiz do projeto, aquele primeiro font-size que colocamos no `<body>` ou `<html>`.





# Adicionando cor ao seu site

Existem duas formas de adicionar cor a um site, dependendo do seu objetivo:

Para adicionar cor a uma fonte, usamos o atributo `color`, já para adicionar cor a um elemento inteiro, normalmente usamos o `background-color`.

Também é possível adicionar uma imagem a um background, nesse caso usamos o atributo `background-image`.

Para saber mais sobre o uso do atributo `background`, [clique aqui](#)

Por exemplo, se vamos fazer um botão podemos usar o seguinte:

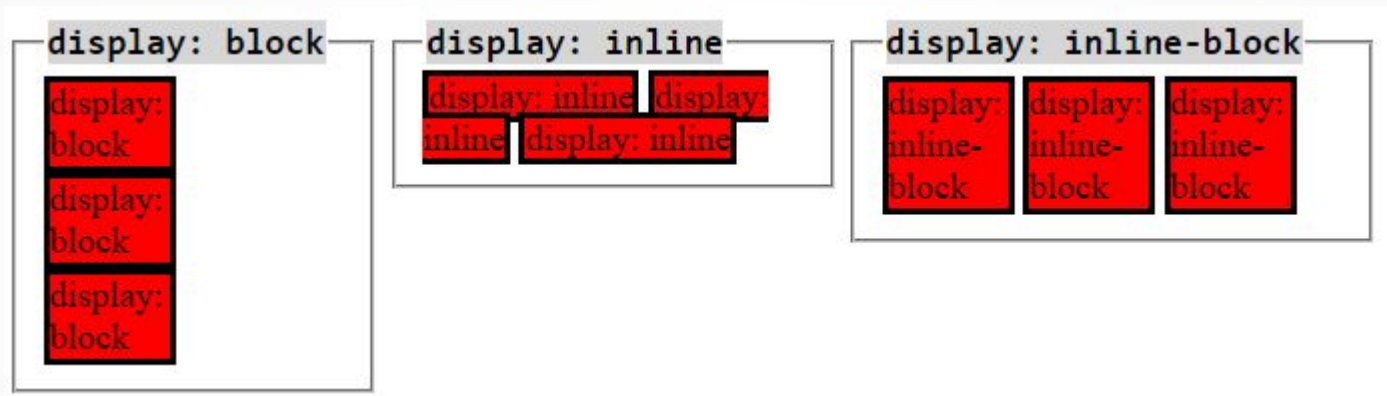
```
button {  
  color: #fff;  
  background-color: green;  
}
```

o resultado seria:



comprar

# O que são elementos block e inline?



A propriedade display especifica a exibição de um elemento. É muito importante entender esse funcionamento.

As tags HTML normalmente já vem com display específicos, por exemplo `<span>` é um display: inline e `<p>` é um display: block, mas você pode alterar isso de acordo com seu projeto.

## inline



Exibe um elemento na mesma linha, um atrás do outro.

Elementos inline só tomam o espaço necessário do conteúdo e colocar propriedades de altura e largura não vão surtir efeito

## block



Enquanto elementos inline são exibidos na mesma linha, elementos em bloco sempre começam uma nova linha (ou um novo bloco), ocupando toda a largura do elemento pai.

## inline-block



Exibe um elemento como um display inline mas possui a característica de um elemento em bloco, ou seja, você pode adicionar uma altura e uma largura a elementos inline-block



# Diferença entre display e position

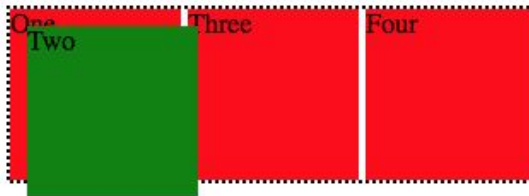
Enquanto a propriedade display especifica o comportamento de exibição dos elementos (inline, block, inline-block), como visto anteriormente, a propriedade de position especifica a posição do elemento dentro do documento, podendo ser absolute, relative, stick, static ou fixed.

Os mais comuns de serem usados são o absolute e relative, no absolute o elemento fica por cima de todos do documento e no relative ele é relativo ao elemento pai.

position: relative



position: absolute

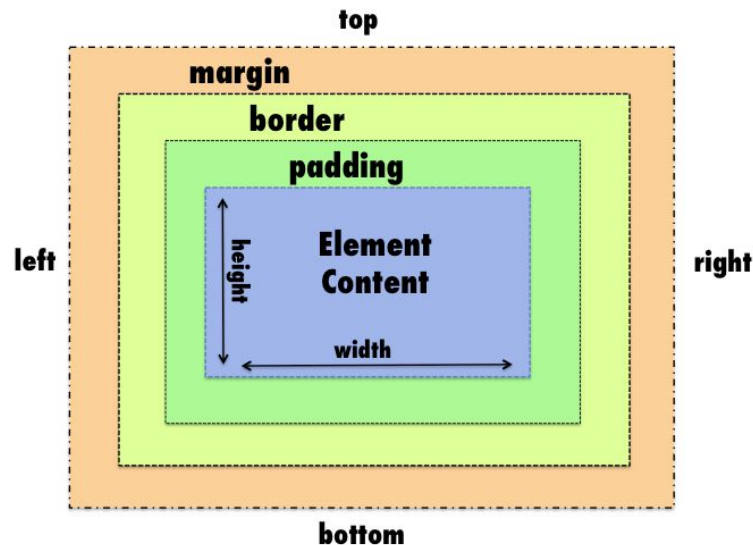


Para mais informações sobre a propriedade position, [clique aqui](#)

# Espaçamento

As formas mais comuns de você adicionar espaçamento aos seus elementos é através do uso de margens e paddings.

Apesar de funcionarem de uma forma muito semelhante, a principal forma de diferenciar essas duas propriedades é que o padding controla o espaçamento **dentro** do elemento e a margin controla o espaçamento **ao redor** do elemento.



## Pontos a serem lembrados



### Padding

O padding não vai funcionar em elementos inline, é provável que você precise adicionar um display: inline-block para resolver esse problema.



### Quando usar

Use margin para adicionar distância entre elementos.

Use padding para aumentar o tamanho de elementos ou dar uma distância entre o conteúdo e a borda do elemento pai.



### Margin

Margin tem a característica de overlap. Ou seja, se vc tem dois componentes um ao lado do outro ou um em cima do outro, ambos com margin, as margens vão colapsar e a maior vai ser usada.

Usando padding os dois paddings seriam usados porque padding é considerado parte do elemento, enquanto margin está do lado de fora do elemento.



# Reset CSS



```
h1 {                                user
  display: block;
  font-size: 2em;
  margin-block-start: 0.67em;
  margin-block-end: 0.67em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
  font-weight: bold;
}
```

Se você reparar quando inspecionamos um elemento HTML, por padrão já vem com alguns estilos predefinidos que não necessariamente queremos usar em nosso projeto.

Claro que dá pra fazer um site sem resetar o css mas quando não fazemos isso, o navegador vai adicionar suas propriedades padrão e como cada navegador tem suas próprias preferências, seu site pode tem grande chance de ficar diferente entre cada navegador.



Mas qual a melhor forma resolver isso?

Um dos Reset CSS mais famosos é o do [Eric Meyer](#), aonde ele zera todas as margins, paddings, remove estilo de listas, outlines... É bem completo!

Você pode copiar esse reset e criar um novo arquivo chamado reset.css e adicionar no seu html, lembre-se de adicioná-lo antes do seu estilo, para que você consiga sobrescrever o reset caso necessário.

Você também pode optar por fazer uma versão mais simples, colocando apenas os pontos que vão ser usados no seu projeto.



# Fazendo um layout responsivo

No desenvolvimento web falamos muito sobre layout responsivo, isso porque de alguns anos pra cá temos visto muito mais necessidade em pensar em como nossas páginas vão ficar em telas maiores ou menores.

Existem várias formas de você desenvolver já pensando na responsividade: você pode tanto usar a técnica do mobile-first, ou seja, desenvolver primeiro para telas de celular e depois escalar para telas maiores ou você pode começar desenvolvendo para telas grandes utilizando algumas propriedades que vão facilitar essa adaptação para celular.

Em ambos os casos é bem provável que você precise usar as media queries.



Uma media query é um pedaço de código que só vai ser renderizado a partir de um tamanho especificado ou até um tamanho especificado.

Você pode usar quantas **media queries** você precisar, por exemplo, na imagem ao lado temos um media query para a largura máxima de 768px e outro para larguras a partir (min-width) 769px

Algumas outras dicas que você pode utilizar para evitar que o site fique quebrado e ter a necessidade de ajustar com media queries é usar unidades relativas para tamanho de containers, imagens, textos e etc, como vh e vw, além de usar flexbox e grid.

```
/* Example */

/* Mobile and Tablet */
@media (max-width: 768px) {
  html {
    font-size: 16px + 2 * ((100vw - 360px) / 768px);
  }
}

/* Laptop and Desktops screens */
@media (min-width: 769px) {
  html {
    font-size: 14px + 10 * ((100vw - 769px) / 2048px);
  }
}

/* Excessively large screens */
html {
  font-size: 36px;
}
```

# FlexBox



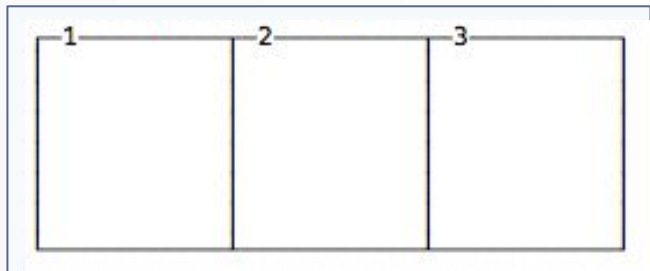
A principal ideia por trás do Flexbox é dar ao container a capacidade de alterar a largura e altura de seus itens para preencher melhor o espaço dele, isso significa que, independente das suas dimensões o conteúdo manterá um layout flexível dentro do seu elemento pai.

Antes da criação do flexbox, ajustes simples como fazer com que todos os elementos tenham o mesmo tamanho dentro de um container ou até mesmo centralizar um elemento ao centro vertical e horizontalmente, era muito complicado ou até mesmo impossível.

Para entender melhor sobre o funcionamento do flexbox, sugiro [essa documentação](#)



display: flex





Algumas coisas que devem ser levadas em consideração no flexbox é quais propriedades podemos usar no container pai e quais utilizamos nos itens dentro do container.

No container podemos alinhar tanto a direção em coluna quanto em linha usando o flex-direction. Por definição, sempre que usamos o display: flex, o conteúdo fica com o flex-direction: row, ou seja em linha, caso você queira diferente, deve mudar no seu CSS.

## Flexbox Properties

Parent

**flex-direction**

**flex-wrap**

**flex-flow**

**justify-content**

**align-items**

**align-content**

Child

**order**




**flex-grow**

**flex-shrink**

**flex-basis**

**flex**

**align-self**

 samanthaming
 samanthaming.com
 samantha\_ming

Uma das propriedades do flexbox mais usadas é a `justify-content`, que nos ajuda a alinhar o conteúdo dentro do container, no início, no fim, centralizado e com espaço entre ou ao redor. Isso vai depender claro do tamanho do seu container.

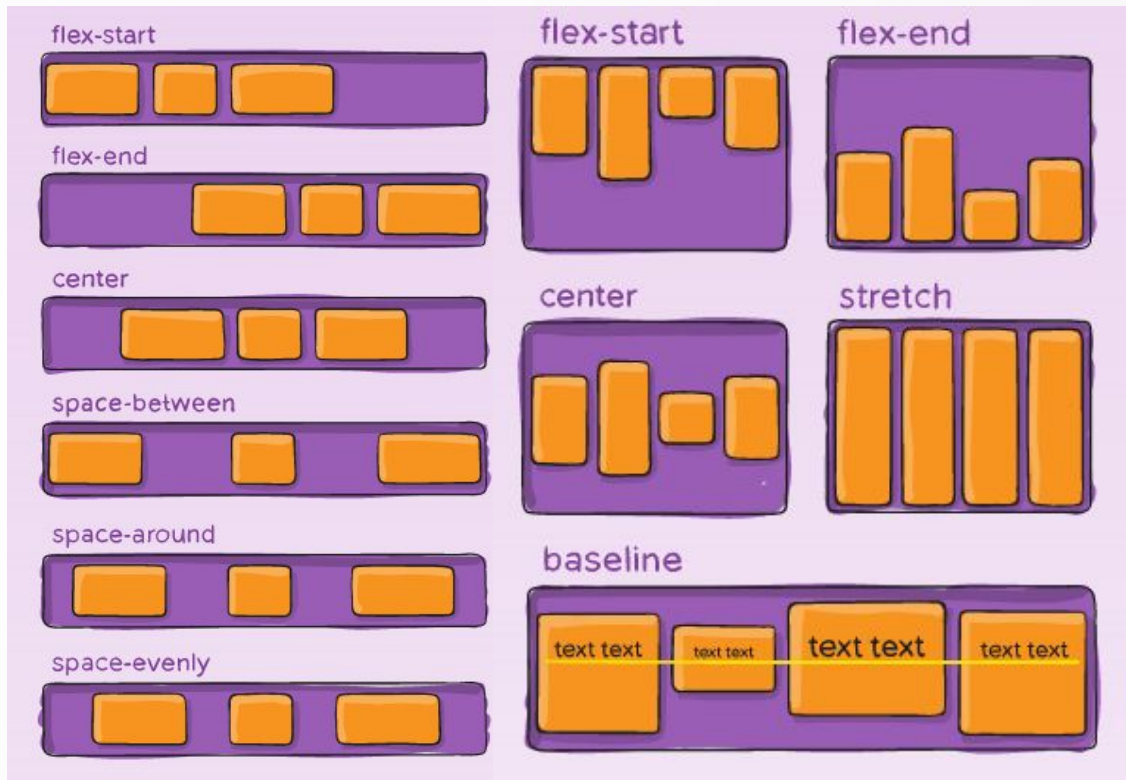
Também é muito usado o `align-items`, isso porque o conteúdo de um container flex é por definição `align-items: stretch`, por isso que os itens se esticam para ocupar toda a altura e largura do container, podemos mudar isso para centralizado e alinhado acima ou abaixo.

Quer aprender sobre flexbox jogando?

Conheça o [Flexbox Froggy](#)

`justify-content`

`align-items`



# Obrigade



