**Meu caderno de anotações para a formação front-end da Alura**

**HTML:**

Toda estrutura HTML segue uma padronização para seu funcionamento, com o body dividido em três elementos principais:

Header - Cabeçalho

Main – Corpo e conteúdo

Footer – Rodapé

Section – Uma tag semântica utilizada para definir uma seção de conteúdo, muito utilizada para agrupar conteúdos e tornar o conteúdo do website semântico.

**CSS**

Reset-CSS:

O reset CSS é um recurso importante a ser utilizado para limpar as configurações padrão de CSS que já são importadas junto ao navegador, para utilizar o reset CSS pode-se usar a documentação padrão, ou um reset definido manualmente, utilizando o seletor \*

Altura: 100vh – Utilizamos a View High para demonstrar ao navegador que aquele elemento tem de ocupar toda a altura da minha página, não deixando nada fora da sua altura (o mesmo serve para o vw (width))

Como a altura dos dispositivos é diferente, e o conteúdo do site pode ser mostrado em diferentes dispositivos, a vh é importante de ser utilizada para padronizar a altura de leitura em n dispositivos.

Box-Sizing: Border-Box

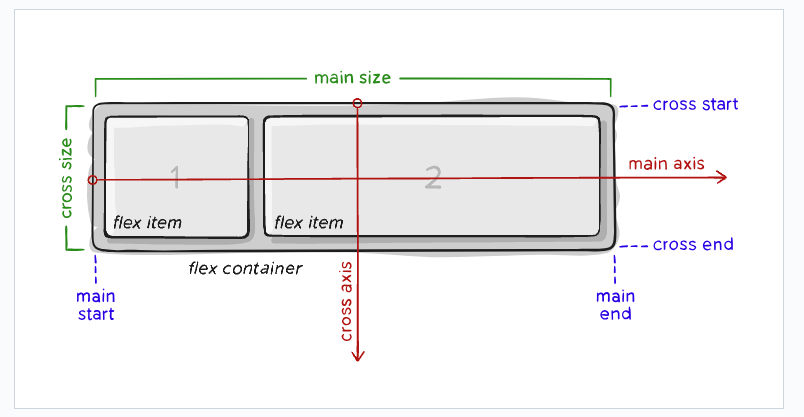
Propriedade padrão do front-end para definir que uma caixa/conteúdo não saia do seu pai, com essa propriedade o conteúdo sempre ficará dentro das suas respectivas caixas pai (evitando problemas de saída de elementos)

Todo elemento raiz tem 16px como padrão do navegador de tamanho de fonte

Flexbox:

É a tecnologia utilizada para alinhar as caixas e posicioná-las conforme o dev deseja, assim, utilizamos no elemento pai dos elementos que queremos ajustar na página utilizando o parâmetro display:flex para dizer a máquina que utilizaremos flexbox.

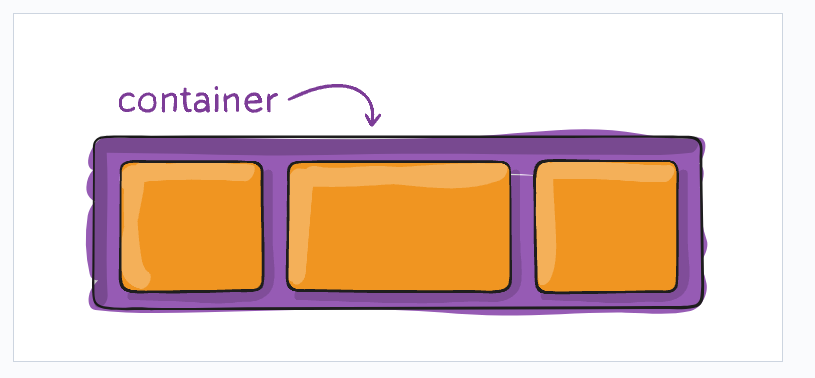
Se o leiaute "padrão" é baseado nas direções block e inline, o leiaute Flex é baseado em direções "flex flow". Veja abaixo um diagrama da especificação, explicando a ideia central por trás do leiaute Flex.



**Flexbox: Seletores e parâmetros para container**

Display: Flex

Esta propriedade define um flex container; inline ou block dependendo dos valores passados. Coloca todos os elementos-filhos diretos num contexto Flex. Sempre será utilizada no seletor do elemento pai, para definir o flexbox dos elementos filhos.



Flex-Direction:

Estabelece o eixo principal, definindo assim a direção em que os flex items são alinhados no flex container. O padrão é em linha, ou seja na horizontal da esquerda para a direita, mas é possível transformar em linha-reverso, coluna, ou coluna-reversa

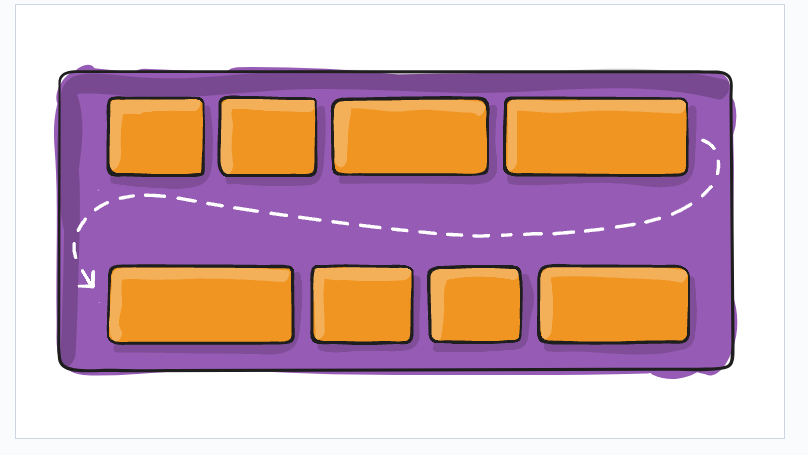
.flex-container {

flex-direction: row | row-reverse | column | column-reverse;

}

Flex-wrap:

Por padrão, os flex items vão todos tentar se encaixar em uma só linha. Com esta propriedade você pode modificar esse comportamento e permitir que os ítens quebrem para uma linha seguinte conforme for necessário.



.flex-container {

flex-wrap: nowrap | wrap | wrap-reverse;

}

É possível juntar as propriedades do flex-direction e do flex-wrap em uma só propriedade, o flex-flow

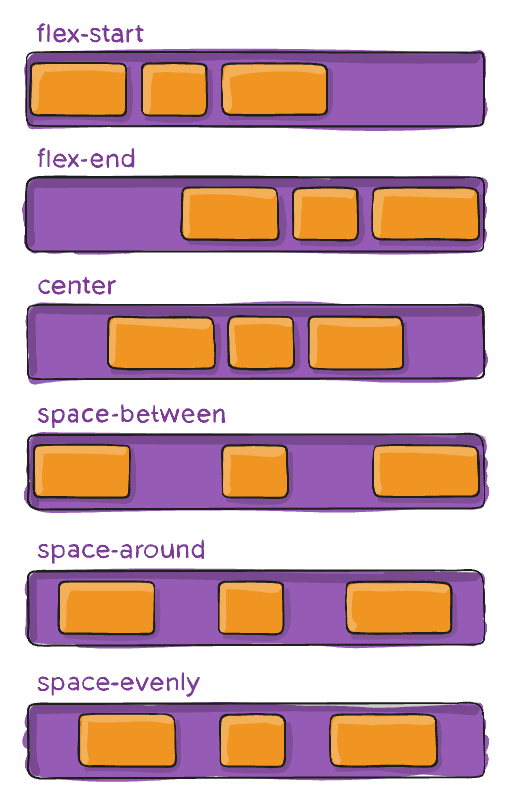
Assim:

.flex-container {

flex-flow: row nowrap | row wrap | column nowrap | column wrap;

}

Justify-content:



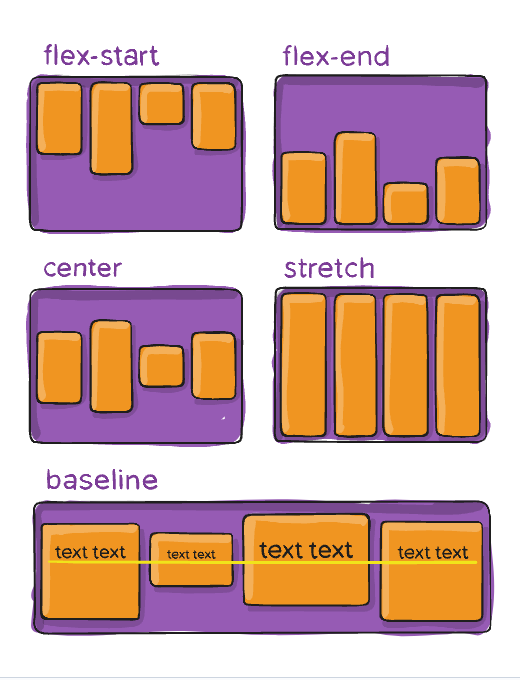
.flex-container {

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;

}

* flex-start (padrão): os ítens são alinhados junto à borda de início (start) de acordo com qual for a flex-direction do container.
* flex-end: os ítens são alinhados junto à borda final (end) de acordo com qual for a flex-direction do container.
* start: os ítens são alinhados junto à borda de início da direção do writing-mode (modo de escrita).
* end: os ítens são alinhados junto à borda final da direção do writing-mode (modo de escrita).
* left: os ítens são alinhados junto à borda esquerda do container, a não ser que isso não faça sentido com o flex-direction que estiver sendo utilizado. Nesse caso, se comporta como start.
* right: os ítens são alinhados junto à borda direita do container, a não ser que isso não faça sentido com o flex-direction que estiver sendo utilizado. Nesse caso, se comporta como start.
* center: os ítens são centralizados na linha.
* space-between: os ítens são distribuídos de forma igual ao longo da linha; o primeiro ítem junto à borda inicial da linha, o último junto à borda final da linha.
* space-around: os ítens são distribuídos na linha com o mesmo espaçamento entre eles. Note que, visualmente, o espaço pode não ser igual, uma vez que todos os ítens tem a mesma quantidade de espaço dos dois lados: o primeiro item vai ter somente uma unidade de espaço junto à borda do container, mas duas unidades de espaço entre ele e o próximo ítem, pois o próximo ítem também tem seu próprio espaçamento que está sendo aplicado.
* space-evenly: os ítens são distribuídos de forma que o espaçamento entre quaisquer dois itens da linha (incluindo entre os ítens e as bordas) seja igual.

Align-items:



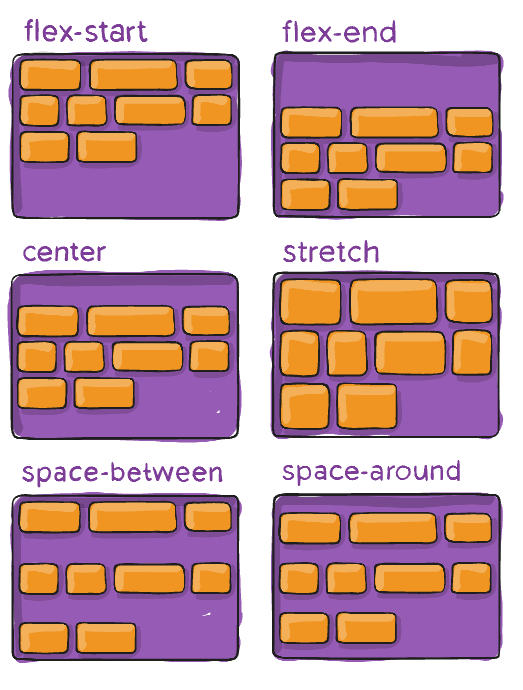
define o comportamento padrão de como flex items são alinhados de acordo com o eixo transversal (cross axis). De certa forma, funciona de forma similar ao justify-content, porém no eixo transversal (perpendicular ao eixo principal).

.flex-container {

align-items: stretch | flex-start | flex-end | center | baseline;

}

Align-content:



Organiza as linhas dentro de um flex container quando há espaço extra no eixo transversal, similar ao modo como justify-content alinha ítens individuais dentro do eixo principal.

**Importante:** Esta propriedade não tem efeito quando há somente uma linha de flex items no container.

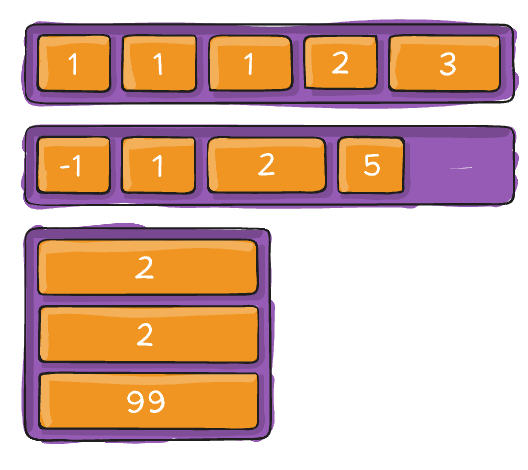
.flex-container {

align-content: flex-start | flex-end | center | space-between | space-around | stretch;

}

* flex-start / start: ítens alinhados com o início do container. O valor (com maior suporte dos navegadores) flex-start se guia pela flex-direction, enquanto start se guia pela direção do writing-mode.
* flex-end / end: ítens alinhados com o final do container. O valor (com maior suporte dos navegadores) flex-end se guia pela flex-direction, enquanto end se guia pela direção do writing-mode.
* center: ítens centralizados no container.
* space-between: ítens distribuídos igualmente; a primeira linha junto ao início do container e a última linha junto ao final do container.
* space-around: ítens distribuídos igualmente com o mesmo espaçamento entre cada linha.
* space-evenly: ítens distribuídos igualmente com o mesmo espaçamento entre eles.
* stretch (padrão): ítens em cada linha esticam para ocupar o espaço remanescente entre elas.

**Propriedades para elementos filhos:**



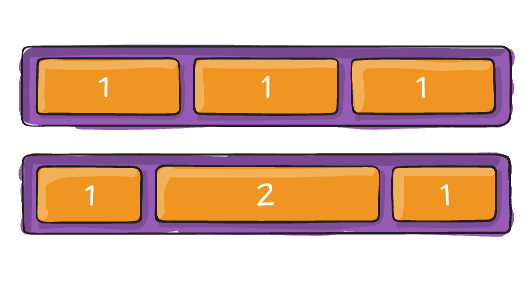
Determina a ordem em que os elementos aparecerão.

.flex-item {

order: <número>; */\* o valor padrão é 0 \*/*

}

Flex-grow



Define a habilidade de um flex item de crescer, caso necessário. O valor dessa propriedade é um valor numérico sem indicação de unidade, que serve para cálculo de proporção. Este valor dita a quantidade de espaço disponível no container que será ocupado pelo item.

Se todos os ítens tiverem flex-grow definido em 1, o espaço remanescente no container será distribuído de forma igual entre todos. Se um dos ítens tem o valor de 2, vai ocupar o dobro de espaço no container com relação aos outros (ou pelo menos vai tentar fazer isso).

.flex-item {

flex-grow: <numero>; */\* o valor default(padrão) é 0 \*/*

}

**flex-shrink**

Define a habilidade de um flex item de encolher, caso necessário.

.flex-item {

flex-shrink: <número>; */\* o valor padrão é 0 \*/*

}

Flex-basis:

Define o tamanho padrão para um elemento antes que o espaço remanescente do container seja distribuído. Pode ser um comprimento (por exemplo, 20%, 5rem, etc) ou uma palavra-chave. A palavra-chave auto significa "observe minhas propriedades de altura ou largura" (o que era feito pela palavra-chave main-size, que foi depreciada). A palavra-chave content significa "estabeleça o tamanho com base no conteúdo interno do ítem" - essa palavra-chave ainda não tem muito suporte, então não é fácil de ser testada, assim como suas relacionadas: max-content, min-content e fit-content.

.flex-item {

flex-basis: flex-basis: | auto; */\* o valor padrão é auto \*/*

}

Com o valor de 0, o espaço extra ao redor do conteúdo não é considerado. Com o valor de auto, o espaço extra é distribuído com base no valor de flex-grox do ítem.

### ****flex****

Esta é a propriedade shorthand para flex-grow, flex-shrink e flex-basis, combinadas. O segundo e terceiro parâmetros (flex-shrink e flex-basis) são opcionais. O padrão é 0 1 auto, mas se você definir com apenas um número, é equivalente a 0 1.

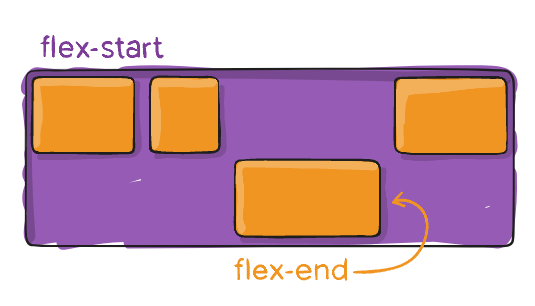
.item {

flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]

}

**É recomendado que você utilize esta propriedade shorthand** ao invés de definir cada uma das propriedades em separado. O shorthand define os outros valores de forma inteligente.

Align-self



Permite que o alinhamento padrão (ou o que estiver definido por align-items) seja sobrescrito para ítens individuais.

Por favor veja a explicação da propriedade align-items para entender quais são os possíveis valores.

.item {

align-self: auto | flex-start | flex-end | center | baseline | stretch;

}

Atalho para fazer comentário selecionando no CSS: CTRL + K + C

Propriedade importante para o flexbox: GAP

Tipos de medidas CSS:

Existem dois tipos de medidas no CSS, as medidas absolutas e relativas

As medidas absolutas dizem respeito a unidades de medida que nunca se alteram, ou seja, sempre mantem seu tamanho fixo de acordo com o valor que o programador definir, assim, 16px sempre serão 16px independente do dispositivo utilizado para acessar aquela página web.

Já as medidas relativas, são as medidas que acompanham o contexto do dispositivo utilizado, ou seja, variam de acordo com o tamanho da tela ou do tamanho desejado da fonte, trazendo responsividade ao website criado.

Sobre fontes com tamanho relativo:

É muito importante utilizarmos as unidades de medidas relativas nos textos e títulos ou até mesmo nos elementos do nosso site, como por exemplo, as imagens.

Essa medida ajuda as pessoas com baixa visão que precisam alterar a aparência do navegador ou aplicar mais zoom para aumentar o tamanho dos elementos.

Outra medida relativa de se utilizar, é a porcentagem, com ela conseguimos alterar o tamanho dos itens que estamos estilizando de acordo com o elemento pai, ou seja, o elemento filho terá uma % referenciada ao tamanho do elemento pai, mas que se altera de acordo com o tamanho da tela, tornando o site responsivo.

Devtools:

Para abrir o devtools no navegador, basta utilizar o atalho CTRL + SHIFT + C

Com o devtools aberto, você pode:

* Escolher o tipo de dispositivo (celular, tablet etc);
* Inspecionar os elementos HTML;
* Alterar o tamanho da tela;
* Alterar os estilos CSS da página temporariamente;
* Apertar as teclas **CTRL + SHIFT + M** para abrir a visualização responsiva.

Sobre media queries:

Na media queries é importante a noção de um parâmetro que pode ser utilizado amplamente quando um valor de width ou height não está atendendo ou não atende o mínimo do elemento pai se encontra na página, este parâmetro é o auto, ele serve para informar que o width ou heigth ocupe o tamanho necessário para adequar o conteúdo a tela automaticamente, tornando o site responsivo.

### ****CSS Grid:****

### ****O que é o CSS Grid?****

### **Basicamente o CSS Grid divide a tela em uma grade de linhas e colunas, oferecendo um conjunto de propriedades para manipular elementos das formas mais variáveis possíveis.**

### **Display: Grid**

### **Assim como no Flexbox, o CSS grid funciona via containers, utilizando um container pai agregar os elementos, assim a propriedade que deve ser utilizada para tornar a página um grid, é display:grid**

### **Grid-Template-Columns**

### **É a propriedade que define o número e o tamanho de colunas que serão dispostas no CSS grid.**

### **Ex:**

grid-template-columns: 200px 200px 200px 200px 200px 200px;

}

### **(Foram 6 divs inseridas no HTML, cada valor de pixel corresponde a uma coluna, relacionada na ordem que foram inseridas)**

### **grid-template-rows**

### **A propriedade grid-template-rows funciona de maneira semelhante à grid-template-columns, mas em vez de definir as colunas, você define o número e o tamanho das linhas em um contêiner de grade CSS**

### **Ex:**

grid-template-rows: 100px 100px 100px 100px 100px 100px;

### **(Foram 6 divs inseridas no HTML, cada valor de pixel corresponde a uma linha, relacionada na ordem que foram inseridas)**

### grid-template-areas:

### Essa propriedade permite que você defina áreas nomeadas no grid, facilitando a organização de elementos filhos em regiões específicas do layout.

Uma propriedade muito importante, pois permite com que você consiga organizar o conteúdo de acordo com o que foi definido previamente nas linhas e colunas

Ex:

grid-template-columns: 1fr 3fr;

grid-template-rows: 200px 500px 200px;

grid-template-areas:

"cabecalho cabecalho cabecalho"

"menu-lateral principal principal"

"rodape rodape rodape";

### grid-auto-rows

A propriedade grid-auto-rows é usada para definir o tamanho padrão das linhas que são criadas automaticamente em um contêiner de grade CSS quando não há uma especificação explícita para essas linhas. Isso é útil quando você possui um layout em grade onde algumas colunas ou linhas são explicitamente definidas, mas outras são criadas dinamicamente com base no conteúdo.

.container {

padding: 10px;

border: 1px solid #2bdefd;

display: grid;

grid-template-columns: 100px 50px 100px;

/\* grid-template-rows: 100px; \*/

grid-auto-rows: 100px;

}

### grid-auto-columns

A propriedade grid-auto-columns é semelhante à grid-auto-rows, mas ela controla o tamanho padrão das colunas criadas automaticamente quando não há uma especificação explícita para essas colunas.

Ex:

.container {

padding: 10px;

border: 1px solid #2bdefd;

display: grid;

grid-template-areas: "1 2 3 4 5";

grid-auto-columns: 200px;

}

### grid-auto-flow

Esta propriedade determina como os itens de grade são automaticamente colocados no grid. Você pode definir se eles fluem em linhas ou colunas automaticamente.

Ex:

.flow-row {

display: grid;

grid-auto-flow: column;

}

.flow-column {

display: grid;

grid-auto-flow: row;

}

.flow-dense {

display: grid;

grid-auto-flow: column;

}

### justify-items

A propriedade justify-items é usada para alinhar horizontalmente os itens (elementos filhos) dentro das células de uma grade (grid). Ela permite definir como os itens se comportam em relação ao eixo horizontal da célula da grade.

Ex:

Justify-items: start, end, center, stretch

### align-items

A propriedade align-items é usada para alinhar verticalmente os itens (elementos filhos) dentro das células de uma grade (grid). Ela permite definir como os itens se comportam em relação ao eixo vertical da célula da grade.

Align-items: start, end, center, stretch

### justify-content

A propriedade justify-content é usada para alinhar horizontalmente as células da grade (grid cells) dentro do contêiner da grade (grid container). Ela controla o alinhamento dos grupos de células da grade em relação ao eixo horizontal do contêiner da grade.

Funciona parecido como no flexbox:

Justify-content: start, end, center, space-around, space-between, space-evenly, stretch

### align-content

A propriedade align-content é usada para alinhar verticalmente os grupos de células da grade (grid cells) dentro do contêiner da grade (grid container). Ela controla o alinhamento dos grupos de células da grade em relação ao eixo vertical do contêiner da grade.

Também funciona como no flexbox:

Align content: start, end, center, space-around, space-between, space-evenly, stretch

### gap / row-gap / column-gap

Essas propriedades controlam o espaçamento entre as células do grid, permitindo criar espaçamentos uniformes entre as linhas e colunas.

**Documentações**

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

<https://developer.mozilla.org/pt-BR/docs/Learn/CSS/CSS_layout/Flexbox>

<https://www.w3schools.com/css/css3_box-sizing.asp>

<https://www.figma.com/file/fLQYKVNuB4kqnvO1mnNMSp/Portfolio---Curso-2?type=design&node-id=1-11&mode=design&t=bijK3dOs4aXyuTI0-0>

[https://www.alura.com.br/artigos/guia-de-unidades-no-css?\_gl=1\*ztrqi0\*\_ga\*NjY2OTY4MjIyLjE2OTQzMDUwNjI.\*\_ga\_1EPWSW3PCS\*MTcwNzgzMzQ4NC4xNC4xLjE3MDc4MzYzNTIuMC4wLjA.\*\_fplc\*eUs1MEVDY0Q1NjRIN2d2TGdxdyUyQndKUllhbnRhTW53R0swJTJGWHZjRzZ2WTdZdVVEbUtVSjRyTHdXMWVxbkU0OWVEWVpsU0FqMUlzbWdnRlUyTnJJdEhZem5tMVFzeGhVcVR5NThtcXZqeVcyY2haYWZSNTVXYmlwMzE5Z0IlMkJRJTNEJTNE](https://www.alura.com.br/artigos/guia-de-unidades-no-css?_gl=1*ztrqi0*_ga*NjY2OTY4MjIyLjE2OTQzMDUwNjI.*_ga_1EPWSW3PCS*MTcwNzgzMzQ4NC4xNC4xLjE3MDc4MzYzNTIuMC4wLjA.*_fplc*eUs1MEVDY0Q1NjRIN2d2TGdxdyUyQndKUllhbnRhTW53R0swJTJGWHZjRzZ2WTdZdVVEbUtVSjRyTHdXMWVxbkU0OWVEWVpsU0FqMUlzbWdnRlUyTnJJdEhZem5tMVFzeGhVcVR5NThtcXZqeVcyY2haYWZSNTVXYmlwMzE5Z0IlMkJRJTNEJTNE)

<https://cursos.alura.com.br/course/html-css-classes-posicionamento-flexbox/task/147896>

[https://www.alura.com.br/artigos/css-border-estilizando-bordas-elementos-css?\_gl=1\*g2vsy8\*\_ga\*NjY2OTY4MjIyLjE2OTQzMDUwNjI.\*\_ga\_1EPWSW3PCS\*MTcwNzg3MTc1OS4xNi4xLjE3MDc4NzI5NDkuMC4wLjA.\*\_fplc\*eUs1MEVDY0Q1NjRIN2d2TGdxdyUyQndKUllhbnRhTW53R0swJTJGWHZjRzZ2WTdZdVVEbUtVSjRyTHdXMWVxbkU0OWVEWVpsU0FqMUlzbWdnRlUyTnJJdEhZem5tMVFzeGhVcVR5NThtcXZqeVcyY2haYWZSNTVXYmlwMzE5Z0IlMkJRJTNEJTNE](https://www.alura.com.br/artigos/css-border-estilizando-bordas-elementos-css?_gl=1*g2vsy8*_ga*NjY2OTY4MjIyLjE2OTQzMDUwNjI.*_ga_1EPWSW3PCS*MTcwNzg3MTc1OS4xNi4xLjE3MDc4NzI5NDkuMC4wLjA.*_fplc*eUs1MEVDY0Q1NjRIN2d2TGdxdyUyQndKUllhbnRhTW53R0swJTJGWHZjRzZ2WTdZdVVEbUtVSjRyTHdXMWVxbkU0OWVEWVpsU0FqMUlzbWdnRlUyTnJJdEhZem5tMVFzeGhVcVR5NThtcXZqeVcyY2haYWZSNTVXYmlwMzE5Z0IlMkJRJTNEJTNE)

<https://css-tricks.com/almanac/properties/g/gap/>

[https://www.alura.com.br/artigos/css-guia-do-flexbox?\_gl=1\*zdaofb\*\_ga\*NjY2OTY4MjIyLjE2OTQzMDUwNjI.\*\_ga\_1EPWSW3PCS\*MTcwNzg3MTc1OS4xNi4xLjE3MDc4NzM0OTYuMC4wLjA.\*\_fplc\*eUs1MEVDY0Q1NjRIN2d2TGdxdyUyQndKUllhbnRhTW53R0swJTJGWHZjRzZ2WTdZdVVEbUtVSjRyTHdXMWVxbkU0OWVEWVpsU0FqMUlzbWdnRlUyTnJJdEhZem5tMVFzeGhVcVR5NThtcXZqeVcyY2haYWZSNTVXYmlwMzE5Z0IlMkJRJTNEJTNE](https://www.alura.com.br/artigos/css-guia-do-flexbox?_gl=1*zdaofb*_ga*NjY2OTY4MjIyLjE2OTQzMDUwNjI.*_ga_1EPWSW3PCS*MTcwNzg3MTc1OS4xNi4xLjE3MDc4NzM0OTYuMC4wLjA.*_fplc*eUs1MEVDY0Q1NjRIN2d2TGdxdyUyQndKUllhbnRhTW53R0swJTJGWHZjRzZ2WTdZdVVEbUtVSjRyTHdXMWVxbkU0OWVEWVpsU0FqMUlzbWdnRlUyTnJJdEhZem5tMVFzeGhVcVR5NThtcXZqeVcyY2haYWZSNTVXYmlwMzE5Z0IlMkJRJTNEJTNE)

<https://cursos.alura.com.br/course/html-css-classes-posicionamento-flexbox/task/147896>

<https://cursos.alura.com.br/course/html-css-responsividade-publicacao-projetos/task/147968>

[https://www.alura.com.br/artigos/criando-layouts-com-css-grid-layout?\_gl=1\*eojbg7\*\_ga\*NjY2OTY4MjIyLjE2OTQzMDUwNjI.\*\_ga\_1EPWSW3PCS\*MTcxMDExMzQzMC41MS4xLjE3MTAxMTM4OTQuMC4wLjA.\*\_fplc\*RGNtT3JvM3VEZDB4YzBHU1UlMkJrQzUxbTQ5a0IzcjJkNml3aXFDQTRPR1BaNGs1ZkRzQmNwbEV6VTZEbUk2bk1vUSUyQkY5ODV4T3VjTEx0UEdrQTNvaGRiTDhDcXNiblE2RGI4JTJCU1BmeHdpYWNxT0RsSkc3RnZ5VTNzcjhCMkJRJTNEJTNE](https://www.alura.com.br/artigos/criando-layouts-com-css-grid-layout?_gl=1*eojbg7*_ga*NjY2OTY4MjIyLjE2OTQzMDUwNjI.*_ga_1EPWSW3PCS*MTcxMDExMzQzMC41MS4xLjE3MTAxMTM4OTQuMC4wLjA.*_fplc*RGNtT3JvM3VEZDB4YzBHU1UlMkJrQzUxbTQ5a0IzcjJkNml3aXFDQTRPR1BaNGs1ZkRzQmNwbEV6VTZEbUk2bk1vUSUyQkY5ODV4T3VjTEx0UEdrQTNvaGRiTDhDcXNiblE2RGI4JTJCU1BmeHdpYWNxT0RsSkc3RnZ5VTNzcjhCMkJRJTNEJTNE)

[https://www.alura.com.br/artigos/nomes-de-classes-no-css?\_gl=1\*ofiw4u\*\_ga\*NjY2OTY4MjIyLjE2OTQzMDUwNjI.\*\_ga\_1EPWSW3PCS\*MTcxMDExMzQzMC41MS4xLjE3MTAxMTM4NjUuMC4wLjA.\*\_fplc\*RGNtT3JvM3VEZDB4YzBHU1UlMkJrQzUxbTQ5a0IzcjJkNml3aXFDQTRPR1BaNGs1ZkRzQmNwbEV6VTZEbUk2bk1vUSUyQkY5ODV4T3VjTEx0UEdrQTNvaGRiTDhDcXNiblE2RGI4JTJCU1BmeHdpYWNxT0RsSkc3RnZ5VTNzcjhCMkJRJTNEJTNE](https://www.alura.com.br/artigos/nomes-de-classes-no-css?_gl=1*ofiw4u*_ga*NjY2OTY4MjIyLjE2OTQzMDUwNjI.*_ga_1EPWSW3PCS*MTcxMDExMzQzMC41MS4xLjE3MTAxMTM4NjUuMC4wLjA.*_fplc*RGNtT3JvM3VEZDB4YzBHU1UlMkJrQzUxbTQ5a0IzcjJkNml3aXFDQTRPR1BaNGs1ZkRzQmNwbEV6VTZEbUk2bk1vUSUyQkY5ODV4T3VjTEx0UEdrQTNvaGRiTDhDcXNiblE2RGI4JTJCU1BmeHdpYWNxT0RsSkc3RnZ5VTNzcjhCMkJRJTNEJTNE)

https://www.alura.com.br/artigos/seu-codigo-css-pode-ser-mais-limpo-flexivel-e-reaproveitavel