CSS 3

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Plan

CSS

CSS : **C**ascading **S**tyle **S**heets

- Langage permettant de compléter le langage **HTML**
- Permettant la mise en forme d'un document **HTML**
- Standardisé par **W3C**
- Trois inclusions différentes d'un code **CSS**
  - en utilisant l'attribut `style` dans des balises **HTML** classiques
  - en ajoutant une balise `<style>` dans `<head>`
  - en plaçant le contenu de style dans un fichier d'extension `.css` puis en référençant ce dernier dans un fichier d'extension `.html` par la balise `<link>`

CSS

Évolution du **CSS**

- **CSS 1** : Introduit en 1996. Les débuts ont été difficiles en raison de la guerre des navigateurs (Internet Explorer, Netscape, etc.).
- **CSS 2** : Apparue en 1998, cette version a introduit environ 70 nouvelles propriétés par rapport à CSS 1.
- **CSS 3** : Bien que cette version ne soit pas encore officiellement finalisée, de nombreuses nouvelles propriétés sont déjà prises en charge par les navigateurs.
- En parallèle à **CSS 3**, le développement de **CSS 4** a commencé en 2010.

CSS

HTML en 91 et **CSS** en 96 : comment fait-on avant le **CSS** ?

Avec des balises **HTML** comme

- `<b>...</b>, <u>...</u>,...`
- `<font color="#000000">` (**dépréciée**)
- `<center>` (**dépréciée**)
- ...

CSS

Première méthode :

```
<tag style="property: value;">
```

CSS

Première méthode :

```
<tag style="property: value;">
```

Exemple :

```
<p style="color:red;">
  Bonjour
</p>
```


Le paragraphe `Bonjour` sera affiché en rouge.

CSS

Deuxième méthode (syntaxe)

```
<style type="text/css">
    selector {
        property: value;
    }
</style>
```

CSS

Deuxième méthode (syntaxe)

```
<style type="text/css">
    selector {
        property: value;
    }
</style>
```

Exemple : le head

```
<style type="text/css">
    p {
        color: red;
    }
</style>
```

Exemple : le body

```
<p>
  Bonjour
</p>
```

Tous les paragraphes de mon document seront affichés en rouge.

CSS

Avec HTML 5

```
<style>
    p {
        color: red;
    }
</style>
```


Plus besoin de préciser le type pour la balise `style`.

CSS

Troisième méthode : on place le contenu CSS dans un fichier (main.css)

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

Dans le fichier HTML :

```
<head>
    <link rel="stylesheet" type="text/css" href="
       main.css">
</head>
```

Tous les paragraphes de mon document seront affichés en rouge.

```
<!doctype HTML>
<html>
    <head>
        <title>Premiere page web </title>
        <meta charset='utf-8'/>
        <style>
            p{
                color: blue;
            }
        </style>
    </head>
    <body>
        <p> je m'affiche en bleu </p>
        <p style="color:red;"> je m'affiche en rouge </p>
        <p> je m'affiche en bleu </p>
    </body>
</html>
```

je m'affiche en bleu
je m'affiche en rouge
je m'affiche en bleu

CSS

Sélecteur **CSS**

- Une balise
- Une classe
- Un identifiant
- Une combinaison de sélecteurs

CSS

Dans le fichier HTML

```
<p> bonjour </p>
```

Dans le fichier CSS

```
p {
    color: blue;
}
```

CSS

Une classe (`class`)

regroupe plusieurs éléments (pas forcément de même type) dans une même famille.

CSS

Une classe (`class`)

regroupe plusieurs éléments (pas forcément de même type) dans une même famille.

Dans le fichier HTML

```
<h1 class=bleu> titre </h1>
<p class=rouge> bonjour </p>
<p class=bleu> bonsoir </p>
```

Dans le fichier CSS

```
.bleu {
    color: blue;
}

.rouge {
    color: red;
}
```

CSS

Un identifiant (`id`)

concerne un seul élément de la page

CSS

Un identifiant (`id`)

concerne un seul élément de la page

Dans le fichier HTML

```
<p id=gras> bonsoir </p>
```

Dans le fichier CSS

```
#gras
{
    font-weight:
       bold;
}
```

CSS

Les sélecteurs

`*`	tous les éléments	
`h1, p`	les `h1` et `p`	`<h1>...<p>...<p>...<h1>`
`div p`	les `p` situés dans `div`	`<div>oui<p>...<span>non<p>`
`p#gras`	les `p` ayant un `id` gras	`<p id='gras'>oui<p>non`
`#gras`	tout élément ayant un `id` gras	`<p id='gras'>oui<p>non`
`p.bleu`	les `p` de la classe bleu	`<p class='bleu'>oui<p>non`
`h1 + p`	les `p` directement après `h1`	`<h1>...</h1><p>oui</p><p>non`
`div > p`	les `p` enfant direct de `div`	`<div>...<p>oui</p>` `<div>...<span>...<p>non`
`div ~ p`	les `p` précédés par `div` avec `p` et `div` ont le même parent	`<ul>...</ul><p>non</p>` `<div>...</div><p>oui</p>`

CSS

Ordre de priorité de sélecteurs

Règle générale : **CSS** donne la priorité au sélecteur le plus spécifique.

- Entre classe et balise : classe
- Entre identifiant et balise : identifiant
- Entre identifiant et classe : identifiant
- Entre deux sélecteurs (qui sont ni identifiant, ni classe) : le dernier dans le fichier **CSS** si aucun des deux n'est plus spécifique que l'autre.

CSS

Le code HTML

```
<p id=mon-id class=ma-classe>
    mon texte
</p>
```

Le code CSS

```
p {
    color: blue;
}

.ma-classe {
    color: green;
}
```


Le paragraphe `mon texte` sera affiché en vert quel que soit l'ordre des sélecteurs.

CSS

Le code HTML

```
<p id=mon-id class=ma-classe>
    mon texte
</p>
```

Le code CSS

```
p {
    color: blue;
}

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


Le paragraphe `mon texte` sera affiché en rouge quel que soit l'ordre des sélecteurs.

CSS

Le code HTML

```
<p id=mon-id class=ma-classe>
    mon texte
</p>
```

Le code CSS

```
p {
    color: blue;
}

#mon-id {
    color: red;
}

.ma-classe {
    color: green;
}
```


Le paragraphe `mon texte` sera affiché en rouge quel que soit l'ordre des sélecteurs.

CSS

Remarque

Pour forcer une valeur, on utilise la propriété `!important`. Le paragraphe `mon texte` sera donc affiché en bleu.

Le code HTML

```
<p id=mon-id class=ma-classe>
    mon texte
</p>
```

Le code CSS

```
p {
    color: blue!important;
}

#mon-id {
    color: red;
}

.ma-classe {
    color: green;
}
```

CSS

Remarque

Utiliser le moins possible la constante `important`.

CSS

Exercice 1

Sans utiliser ni classes, ni identifiants, écrire le code **CSS** qui permet de colorier la liste suivante.

- Langages de programmation
  - Java
  - C++
  - PHP
- Éditeurs de texte
  - Sublime text
  - Atom
  - Notepad++

La construction de la liste doit être W3C Valid.

CSS

Solution

```
li > ul
{
    color: blue;
}

ul > li > ul > li ~ li
{
    color: red;
}
```

Exercice 2

En utilisant seulement deux classes (aucun identifiant), écrire le code **CSS** qui permet de colorier la liste suivante.

- France
  - Bleu
  - Blanc
  - Rouge
- Tunisie
  - Blanc
  - Rouge
- Uruguay
  - Blanc
  - Bleu

La construction de la liste doit être W3C Valid.

CSS

Exercice 3

Refaire l'exercice précédent en utilisant une seule classe (aucun identifiant).

CSS

Les sélecteurs d'attribut

`img[width]`	les images ayant un attribut width	`<img width=200>`
`input[type=text]`	les inputs de type text	`<input type='text'>`
`img[title^=red]`	les images avec un titre commençant par red	`<img title='red bull'>oui` `<img title='white red'>non`
`img[title*=red]`	les images avec un titre contenant le mot red	`<img title='red bull'>oui`
`[lang|=en]`	les éléments avec un attribut lang commençant par en	`<div lang="en-us">oui` `<div lang="fr">non`
`img[src$=.jpg]`	les images ayant un nom se terminant par .jpg	`<img src='1.jpg'>oui` `<img src='jpg.png'>non`
`td[colspan='3']`	les cases d'un tableau qui sont sur 3 colonnes	`<td colspan=3>oui</td>` `<td>non</td>`

CSS

Pseudo-sélecteur : syntaxe

```
sélecteur(s):pseudo-sélecteur
{
    propriété: valeur;
}
```

CSS

Pseudo-sélecteurs (liens)

`a:hover`	les liens survolés
`a:visited`	les liens visités
`a:active`	les liens actifs
`a:link`	les liens non visités

CSS

Pseudo-sélecteurs (formulaires)

`input:optional`	les inputs sans la propriété `required`
`input:required`	les inputs avec la propriété `required`
`input:read-only`	les inputs avec la propriété `readonly`
`input:read-write`	les inputs sans la propriété `readonly`
`input:valid`	les inputs avec une valeur valide
`input:invalid`	les inputs avec une valeur invalide
`input:out-of-range`	les inputs avec une valeur en dehors de l'intervalle
`input:input:in-range`	les inputs avec une valeur dans l'intervalle
`input:focus`	les inputs ayant le focus
`input:checked`	les inputs cochés
`input:enabled`	les inputs activés
`input:disabled`	les inputs désactivés
`div:focus-within`	la div contenant un descendant ayant le focus

Le contenu HTML

```
<div class="container">
    <input type="text" placeholder="Votre nom"><br>
    <input type="text" placeholder="Votre prénom">
</div>
```

Le contenu HTML

```
<div class="container">
    <input type="text" placeholder="Votre nom"><br>
    <input type="text" placeholder="Votre prénom">
</div>
```

Le style CSS

```
.container {
    border: 2px solid gray;
}
.container:focus-within {
    background-color: skyblue;
}
```

Le contenu HTML

```
<div class="container">
    <input type="text" placeholder="Votre nom"><br>
    <input type="text" placeholder="Votre prénom">
</div>
```

Le style CSS

```
.container {
    border: 2px solid gray;
}
.container:focus-within {
    background-color: skyblue;
}
```


Résultat

En cliquant sur l'un des deux `input`, toute la `div` s'affiche en bleu.

CSS

Pseudo-sélecteurs (enfants)

`li:first-child`	tous les éléments `li` qui sont les premiers
`li:not(:first-child)`	tous les éléments `li` qui ne sont pas les premiers
`p:nth-child(2)`	les `p` qui sont le second enfant de leur parent
`p:nth-child(even)`	les `p` d'indice pair
`p:nth-child(odd)`	les `p` d'indice impair
`p:nth-child(2n)`	≡ `p:nth-child(even)`
`p:nth-child(2n+1)`	≡ `p:nth-child(odd)`
`p:nth-child(3n)`	les `p` d'indice 0, 3, 6...
`p:nth-child(3n+2)`	les `p` d'indice 2, 5, 8...
`p:nth-last-of-type(2)`	les seconds enfants de type `p` de leur parent en commençant par le dernier
`p:only-child`	les `p` qui sont unique enfant de leur parent

`:first-child` **vs** `:first-of-type`

`:first-child` **vs** `:first-of-type`

Le style CSS

```
p:first-child{
  background-color: red;
}
p:first-of-type{
  color: blue;
}
```

Le contenu HTML

```
<div>
  <p>bonjour</p>
  <p>bonsoir</p>
</div>
<div>
  <h1>bonjour</h1>
  <p>bonsoir</p>
</div>
```

`:first-child` **vs** `:first-of-type`

Le style CSS

```
p:first-child{
  background-color: red;
}
p:first-of-type{
  color: blue;
}
```

Le contenu HTML

```
<div>
  <p>bonjour</p>
  <p>bonsoir</p>
</div>
<div>
  <h1>bonjour</h1>
  <p>bonsoir</p>
</div>
```


Le résultat est

bonjour

bonsoir

bonjour

bonsoir

CSS

Pseudo-sélecteurs (autres)

`p:lang(en)`	les `p` avec un attribut `lang` dont la valeur = 'en'
`main :where(h1, h2, h3)`	≡ `main h1, main h2, main h3`
`:is(header, main) p:hover`	≡ `header p:hover, main p:hover`

CSS

Pseudo-élément : syntaxe

```
sélecteur(s)::pseudo-élément
{
    propriété: valeur;
}
```

CSS

Pseudo-éléments

`p::first-letter`	la première lettre de chaque élément `p`
`p::first-line`	la première ligne de chaque élément `p`
`p::selection`	les portions sélectionnées d'un paragraphe
`p::before` (**CSS 3**)	le contenu avant chaque élément `p`
`p:before` (**CSS 2**)	
`p::after` (**CSS 3**)	le contenu après chaque élément `p`
`p:after` (**CSS 2**)	

CSS

Les pseudo-éléments `::after` et `before` s'utilisent souvent avec la propriété `content`.

Le code HTML

```
<p>
  John Wick.
</p>
```

Le code CSS

```
p::before {
  content : "Hello,";
}
p::after {
  content : "Have a nice day.";
}
```


Le résultat

```
Hello, John Wick. Have a nice day.
```

CSS

Pour bien maîtriser les sélecteurs

`https://flukeout.github.io/`

CSS

Quelques unités de mesure

- `px` (pour **pixel** : **picture element**)
- `em` (pour **emphemeral unit**)
- `mm` et `cm` : millimètre et centimètre
- `in` (pour **inch** : pouce)
- `%` : pourcentage
- `vh` pour `view height` et `vw` pour `view width`
- ...

CSS

Quelques unités de mesure

- `px` (pour **pixel** : **picture element**)
- `em` (pour **emphemeral unit**)
- `mm` et `cm` : millimètre et centimètre
- `in` (pour **inch** : pouce)
- `%` : pourcentage
- `vh` pour `view height` et `vw` pour `view width`
- ...

Remarque

W3C recommande l'utilisation de `px`, `em` et `%` pour les écrans.

CSS

La propriété `font-size`

- permet de modifier la taille du texte.
- a une valeur par défaut égale à `16px` pour la plupart des navigateurs.
- accepte toutes les unités de mesure.

CSS

La propriété `font-size` accepte aussi les constantes absolues suivantes

- `xx-small` (généralement `9px`)
- `x-small` (généralement `10px`)
- `small` (généralement `13px`)
- `medium` (généralement `16px`) : valeur par défaut
- `large` (généralement `18px`)
- `x-large` (généralement `24px`)
- `xx-large` (généralement `32px`)

CSS

La propriété `font-size` accepte aussi les constantes relatives suivantes

- `smaller` pour avoir une taille inférieure à celle du parent.
- `larger` pour avoir une taille supérieure à celle du parent.

CSS

Le code HTML

```
<div>
    div
    <p>paragraphe</p>
</div>
```

Le code CSS

```
div {
    font-size: 16px;
}

p {
    font-size: 2em;
}
```


Le paragraphe aura la taille 32 pixels.

CSS

Le code HTML

```
<div>
    <ul>
        <li>
            élément
            <ul>
                <li>
                    sous-élément
                </li>
            </ul>
        </li>
    </ul>
</div>
```

Le code CSS

```
div {
    font-size: 16px;
}

li {
    font-size: 2em;
}
```

`élément` aura la taille 32 pixels et `sous-élément` la taille 64 pixels.

Le code HTML

```
<div>
    <ul>
        <li>
            élément
            <ul>
                <li>
                    sous-élément
                </li>
            </ul>
        </li>
    </ul>
</div>
```

Le code CSS

```
div {
    font-size: 16px;
}

li {
    font-size: 2rem;
}
```

Pour que `élément` et `sous-élément` aient la même taille, on utilise `rem` (**root empheral unit**).

CSS

Les propriétés de fond

- `color` : couleur de texte
- `background-color` : couleur de fond

HTML

Les formats acceptés pour les couleurs

- Le nom en anglais : `red`, `blue`... (140 noms acceptés : `https://www.w3schools.com/colors/colors_names.asp`)
- Le code hexadécimal : `https://www.w3schools.com/colors/colors_hexadecimal.asp`
- Le code RGB : `https://www.w3schools.com/colors/colors_rgb.asp`
- Le code HSL : `https://www.w3schools.com/colors/colors_hsl.asp`

HTML

Quelques éditeurs de couleur

- https://www.w3schools.com/colors/colors_picker.asp
- https://colorhunt.co/
- https://color.adobe.com/fr/create/color-wheel

CSS

Autres propriétés de fond

- `background-color` : couleur de fond
- `background-image` : url(image de fond)
- `background-repeat` : permet de répéter l'image
  - horizontalement et verticalement avec la valeur `repeat`
  - verticalement avec la valeur `repeat-y`
  - horizontalement avec la valeur `repeat-x`
  - aucune répétition avec la valeur `no-repeat`
- `background-position` : permet d'indiquer la position de l'image (`left top`, `right top`, `center`...)
- ...

CSS

La propriété `background`

- un raccourci de toutes les autres propriétés sur le `background`
- Exemple : `background: pink url("mario.jpg") no-repeat center;`

CSS

La propriété `background`

- un raccourci de toutes les autres propriétés sur le `background`
- Exemple : `background: pink url("mario.jpg") no-repeat center;`

Autres propriétés

- `color` : couleur du texte
- `opacity` : transparence (valeur de 0 à 1)
- ...

CSS

Exemple : code HTML

```
<div class="gris">
    <p class="blanc">
        Bonjour
    </p>
</div>
```

Code CSS

```
.gris {
    background-color: grey;
    opacity: 0.5;
}

.blanc {
    background-color: white;
    opacity: 1;
}
```

CSS

Exemple : code HTML

```
<div class="gris">
    <p class="blanc">
        Bonjour
    </p>
</div>
```

Code CSS

```
.gris {
    background-color: grey;
    opacity: 0.5;
}

.blanc {
    background-color: white;
    opacity: 1;
}
```


Remarque

À cause de l'opacité héritée de la `div`, le paragraphe `Bonjour` est affiché en gris.

CSS

Solution

Utiliser des couleur `rgba` : le dernier paramètre correspond à l'opacité.

CSS

Solution

Utiliser des couleur `rgba` : le dernier paramètre correspond à l'opacité.

Exemple : code HTML

```
<div class="gris">
    <p class="blanc">
        Bonjour
    </p>
</div>
```

Code CSS

```
.gris {
    background-color: rgba(128, 128, 128, 0.5);
}

.blanc {
    background-color: rgba(255, 255, 255, 1);
}
```

CSS

Les valeurs de la propriété `font-weight`

- `normal`
- `bold`
- `bolder`
- `lighter`

Les valeurs de la propriété `font-style`

- `normal`
- `italic` (version spéciale de la fonte)
- `oblique` (version inclinée de la fonte)
- `oblique angle`

Les valeurs de la propriété `writing-mode`

- `horizontal-tb` (par défaut)
- `vertical-rl`
- `vertical-lr`
- ...

CSS

Les valeurs de la propriété `text-decoration`

- `none`
- `underline`
- `overline`
- `line-through`

Les valeurs de la propriété `font-family`

- `"Times New Roman"`
- `Arial`
- `Arial, Helvetica, sans-serif`
- ...

CSS

Les valeurs de la propriété `text-decoration`

- `none`
- `underline`
- `overline`
- `line-through`

Les valeurs de la propriété `font-family`

- `"Times New Roman"`
- `Arial`
- `Arial, Helvetica, sans-serif`
- ...

La propriété `text-decoration` peut accepter 3 valeurs

1. position de la ligne
2. style de la ligne : `solid`, `dashed`, `dotted`, `wavy`, `double`...
3. couleur de la ligne

La propriété `@font-face` permet de définir sa propre police.

La propriété `@font-face` permet de définir sa propre police.

Exemple

```
@font-face {
  font-family: maFonte;
  src: url(fichier.woff);
  font-weight: bold;
}
div {
  font-family: maFonte;
}
```

La propriété `@font-face` permet de définir sa propre police.

Exemple

```
@font-face {
  font-family: maFonte;
  src: url(fichier.woff);
  font-weight: bold;
}
div {
  font-family: maFonte;
}
```


Fontes disponibles de **Google**

- `https://fonts.google.com/`
- sélectionner un et l'ajouter dans la partie `head` de votre document **HTML**

CSS

Autres fontes disponibles

- `https://www.dafont.com/fr/`
- `https://www.fonts.com/font/monotype/arial`

CSS

Exercice

Étant donnée une balise `<div class=container>` contenant plusieurs balises `<p>` (au moins trois), écrire le code **CSS** qui permet d'afficher

- en gras le paragraphe survolé,
- en italic les paragraphes situés avant le paragraphe survolé et
- en souligné les paragraphes situés après le paragraphe survolé.

CSS

Exemple

Je fais parti des paragraphes situés avant le paragraphe survolé.

Je fais parti des paragraphes situés avant le paragraphe survolé.

Je suis le paragraphe survolé.

Je fais parti des paragraphes situés après le paragraphe survolé.

Je fais parti des paragraphes situés après le paragraphe survolé.

CSS

Correction : HTML

```
<div class="container">
  <p>paragraph 1</p>
  <p>paragraph 2</p>
  <p>paragraph 3</p>
  <p>paragraph 4</p>
  <p>paragraph 5</p>
</div>
```

Correction : CSS

```
.container:hover p {
  text-decoration: underline;
}
.container p:hover {
  text-decoration: none;
  font-weight: bold;
}
.container p:hover ~ p {
  text-decoration: none;
  font-style: italic;
}
```

CSS

La propriété `line-height`

- permet de définir la hauteur de la ligne.
- accepte la constante (`normal` : valeur par défaut) ou un nombre, un pourcentage...

CSS

Exemple : code HTML

```
<div class="a">
    line-height: normal <br>
    valeur par défaut
</div>

<div class="b">
    line-height: 1.6<br>
    valeur recommandée<br>
    le nombre 1.6 sera multiplié par le
       font-size
</div>

<div class="c">
    line-height: 80%<br>
    difficile à lire.
</div>
```

Code CSS

```
div.a {
    line-height: normal;
    border: 1px black solid;
}

div.b {
    line-height: 1.6;
    border: 1px black solid;
}

div.c {
    line-height: 80%;
    border: 1px black solid;
}
```

La propriété `text-indent`

- permet d'indenter la première ligne d'un texte.
- accepte les unités `px`, `em` et `%`.

La propriété `text-indent`

- permet d'indenter la première ligne d'un texte.
- accepte les unités `px`, `em` et `%`.

Valeurs de la propriété `direction`

- `rtl` : pour écrire un texte de droite à gauche.
- `ltr` : pour écrire un texte de gauche à droite.

La propriété `text-indent`

- permet d'indenter la première ligne d'un texte.
- accepte les unités `px`, `em` et `%`.

Valeurs de la propriété `direction`

- `rtl` : pour écrire un texte de droite à gauche.
- `ltr` : pour écrire un texte de gauche à droite.

La propriété `letter-spacing`

- permet de définir l'espace entre les lettres.
- accepte les unités `px` et `em` (valeur par défaut `normal`).

CSS

Exemple (HTML)

```
<p>De droite à gauche.</p>
<p>De gauche à droite.</p>
```


CSS

```
p:first-of-type {
    direction: rtl;
}

p:last-of-type {
    text-indent: 200px;
    direction: ltr;
    letter-spacing: 3PX;
}
```

CSS

Pour ajouter une bordure

- `border-width` : qui accepte
  - une valeur en pixels, `em`...
  - une constante `thin`, `medium` et `thick`
- `border-style` : plusieurs valeurs possibles : `none`, `solid`, `dashed`, `dotted`, `double`...
- `border-color` : couleur
- ...

CSS

Pour appliquer une bordure à un paragraphe

```
p
{
    border-width: 1px;
    border-color: red;
    border-style: dotted;
}
```

Remarque

La propriété `border-width` peut accepter plusieurs valeurs.

Remarque

La propriété `border-width` peut accepter plusieurs valeurs.

Exemple : code HTML

```
<p id=one-value>
    4 sides
</p>

<p id=two-values>
    top/bottom and right/left
</p>

<p id=three-values>
    top and right/left and bottom
</p>

<p id=four-values>
    top and right and bottom and left
</p>
```

Code CSS

```
#one-value {
    border-style: solid;
    border-width: 1px;
}
#two-values {
    border-style: solid;
    border-width: 1px 3px;
}
#three-values {
    border-style: solid;
    border-width: 1px 3px 5px;
}
#four-values {
    border-style: solid;
    border-width: 1px 3px 5px 7
       px;
}
```

CSS

Ou utiliser une seule propriété `border` (l'ordre des valeurs n'a pas d'importance)

```
p
{
    border: 1px red dotted;
}
```

CSS

Ou utiliser une seule propriété `border` (l'ordre des valeurs n'a pas d'importance)

```
p
{
    border: 1px red dotted;
}
```


Remarques

- L'ordre des valeurs de `border` n'a pas d'importance.
- La seule la valeur obligatoire de `border` est celle de `border-style`.
- En l'absence d'une valeur, `border-color` prend par défaut la couleur du texte.
- En l'absence d'une valeur, `border-style` prend par défaut la valeur `medium`.

CSS

Autres propriétés de bordure

- `border-top` : haut
- `border-bottom` : bas
- `border-left` : gauche
- `border-right` : droite
- `border-radius` : pour arrondir une bordure, peut prendre 4 valeurs différentes pour en haut à gauche, en haut à droite, en bas à droite, en bas à gauche.
- `box-shadow` : pour l'ombre. prend quatre valeurs, le décalage horizontal de l'ombre (en px), le décalage vertical de l'ombre (en px), l'adoucissement du dégradé (en px) et la couleur de l'ombre.
- `text-shadow` : pour l'ombre d'un texte.
- ...

CSS

Exemple

```
p
{
    border-bottom: 1px red dotted;
    border-top: 1px red dashed;
    border-right: 1px blue solid;
    border-left: 1px blue double;
    box-shadow: 3px 3px 1px gray;
}
```

CSS

Autres propriétés de tableau

- `border-collapse` : pour préciser si les bordures de cellules d'un tableau sont
  - fusionnées (`collapse`),
  - séparées (`separate`, par défaut). On peut utiliser la propriété `border-spacing` pour définir l'espace entre les bordures.
- `caption-side` : pour indiquer l'emplacement de la légende (`caption`)
  - en bas (`bottom`),
  - en haut (`top`, par défaut).
- `empty-cells` : pour indiquer comment faire avec la bordure de cellules vides
  - pour les cacher (`hide`),
  - pour les afficher (`show`, par défaut).
- ...

CSS

Exemple : code HTML

```
<table>
    <caption>légende</caption>
    <tr>
        <td>1</td>
        <td>2</td>
    </tr>
    <tr>
        <td>3</td>
        <td></td>
    </tr>
</table>
```

Code CSS

```
table {
    border: 1px solid black;
    caption-side: bottom;
    border-collapse: separate;
    empty-cells: hide;
}

td {
    border: 1px solid black;
}
```

CSS

Les valeurs de la propriété `text-align` (horizontal)

- `center`
- `left` (par défaut)
- `right`
- `justify`

CSS

Exemple : code HTML

```
<div>
    premier texte
</div>
```

Code CSS

```
div {
    width: 300px;
    height: 100px;
    border: 1px solid black;
    text-align: center;
}
```

CSS

Les valeurs de la propriété `vertical-align` (pour les balises `inline` et les cellules de tableau)

- `top`
- `baseline` (par défaut)
- `middle`
- `bottom`
- `sub`
- `super`
- ...

CSS

Exemple : code HTML

```
<table>
    <tr>
        <td>1</td>
        <td>2</td>
    </tr>
    <tr>
        <td>3</td>
        <td>4</td>
    </tr>
</table>
```

Code CSS

```
table {
    width: 300px;
    height: 100px;
    border: 1px solid black;
    text-align: center;
    border-collapse: collapse;
}

td {
    border: 1px solid black;
}

td:first-child {
    vertical-align: text-top;
}

td:last-child {
    vertical-align: bottom;
}
```

CSS

Considérons les codes HTML et CSS suivants :

```html
<!-- HTML -->
<div>
    texte
</div>
```

```css
/* CSS */
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
}
```

CSS

Considérons les codes HTML et CSS suivants :

```html
<!-- HTML -->
<div>
    texte
</div>
```

```css
/* CSS */
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
}
```


Exercice

Ajouter le code **CSS** (sans modifier le précédent) permettant de centrer verticalement et horizontalement le contenu de la balise `div` précédente (de type **block**).

CSS

Première solution

```
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
    text-align: center;
    line-height: 100px;
}
```

CSS

Deuxième solution

```
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
    text-align: center;
    display: table-cell;
    vertical-align: middle;
}
```

CSS

Les valeurs de la propriété `transform`

- `translate()`
- `rotate()`
- `skew()`, `skewX()` et `skewY()` (obliquer)
- ...

CSS

Exemple avec `translate` : code HTML

```
<div id=translate>
     translation
</div>
```

Code CSS

```
#translate {
    width: 300px;
    height: 100px;
    background-color: red;
    border: 1px solid black;
    transform: translate(50px, 100px);
}
```

CSS

Exemple avec `rotate` : code HTML

```
<div>
    normale
</div>
<div id=rotate>
    rotation
</div>
```

Code CSS

```
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
}

#rotate {
    transform: rotate(20deg);
}
```

CSS

Exemple avec `skew` : code HTML

```
<div>
    normale
</div>
<div id=skew>
    skewing
</div>
```

Code CSS

```
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
}

#skew {
    transform: skewX(20deg);
}
```

CSS

Comment changer les puces ou les numéros ?

- La propriété : `list-style-type`
- Les valeurs possibles pour les listes non-ordonnées : `circle`, `square`, `none`...
- Les valeurs possibles pour les listes ordonnées : `upper-roman`, `lower-alpha`, `none`...

CSS

Comment changer les puces ou les numéros ?

- La propriété : `list-style-type`
- Les valeurs possibles pour les listes non-ordonnées : `circle`, `square`, `none`...
- Les valeurs possibles pour les listes ordonnées : `upper-roman`, `lower-alpha`, `none`...

On peut aussi utiliser une image comme puce pour les `<ul>`

```
list-style-image: url('image.gif');
```

CSS

Supprimer les valeurs par défaut ? (pour un menu par exemple)

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}
```

CSS

Supprimer les valeurs par défaut ? (pour un menu par exemple)

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}
```


Les puces à l'intérieur ou à l'extérieur de la liste

- `list-style-position`
- Les valeurs possibles : `inside` ou `outside` (par défaut)

CSS

Exemple avec `outside`

- Tu as regardé l'égaliseur de Denzel Washington ? ou John Wick de Keanu Charles Reeves ?
- Tu aimes bien le Brésil ? Tu te souviens de Romario, Bebeto, Ronaldo et Dunga ?

CSS

Exemple avec `outside`

- Tu as regardé l'égaliseur de Denzel Washington ? ou John Wick de Keanu Charles Reeves ?
- Tu aimes bien le Brésil ? Tu te souviens de Romario, Bebeto, Ronaldo et Dunga ?

Exemple avec `inside`

- Tu as regardé l'égaliseur de Denzel Washington ? ou John Wick de Keanu Charles Reeves ?
- Tu aimes bien le Brésil ? Tu te souviens de Romario, Bebeto, Ronaldo et Dunga ?

CSS

Exemple avec `outside`

- Tu as regardé l'égaliseur de Denzel Washington ? ou John Wick de Keanu Charles Reeves ?
- Tu aimes bien le Brésil ? Tu te souviens de Romario, Bebeto, Ronaldo et Dunga ?

Exemple avec `inside`

- Tu as regardé l'égaliseur de Denzel Washington ? ou John Wick de Keanu Charles Reeves ?
- Tu aimes bien le Brésil ? Tu te souviens de Romario, Bebeto, Ronaldo et Dunga ?

Autre exemple sur https://www.w3schools.com/css/tryit.asp?filename=trycss_list-style-position

CSS

Un raccourci ?

```
ul {
    list-style: square inside url("image.gif");
}
```

CSS

Un raccourci ?

```
ul {
    list-style: square inside url("image.gif");
}
```


Les valeurs, dans l'ordre, correspondent à :

- `list-style-type` : si un style-image est spécifié, ceci sera affiché quand l'image ne peut être affiché
- `list-style-position` : pour indiquer la position des puces (à l'intérieur ou à l'extérieur)
- `list-style-image` : pour spécifier l'image à utiliser comme puce

CSS

Deux propriétés pour utiliser un compteur **CSS**

- `counter-reset` : pour initialiser le compteur.
- `counter-increment` : pour incrémenter ou décrémenter le compteur.

CSS

Le code HTML

```
<p>Paragraphe</p>
<p>Paragraphe</p>
<div>Div</div>
<div>Div</div>
```

Le code CSS

```
p, div {
    counter-increment: myIndex 1;
}
p::before, div::before {
    content: counter(myIndex) "- ";
}
div:first-of-type {
    counter-reset: myIndex;
}
```


Le résultat

```
1- Paragraphe
2- Paragraphe
1- Div
2- Div
```

CSS

Plusieurs propriétés pour manipuler les colonnes en **CSS**

- `column-count` : prend comme valeur le nombre de colonne à utiliser pour afficher le contenu d'une balise.
- `column-width` : définit la largeur d'une colonne.
- `columns` : prend un couple de valeurs, la première pour la largeur de la colonne et la deuxième pour le nombre de colonnes.
- `column-gap` : définit la distance entre les colonnes
- `column-rule-color` : définit la couleur de la bordure verticale entre les colonnes.
- `column-rule-width` : définit la largeur de la bordure verticale entre les colonnes.
- `column-rule-style` : définit le style de la bordure verticale entre les colonnes.
- `column-rule` = `column-rule-width` + `column-rule-style` (obligatoire) + `column-rule-color`
- ...

CSS

Le code HTML

```
<p class=colonne>
    générer lorem ipsum
</p>
```

Le code CSS

```
.colonne {
  column-count: 3;
  column-gap: 40px;
  column-rule: 4px double red;
}
```


Résultat

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.

CSS

Plusieurs valeurs pour la propriété `cursor`

- `pointer`
- `grab`
- `help`
- `move`
- `progress`
- `wait`
- `zoom-in`
- `zoom-out`
- `n-resize`
- `not-allowed`
- ...

CSS

Pour modifier la couleur du curseur dans une zone de saisie, on peut utiliser la propriété `caret-color`

```
input {
  caret-color: skyblue;
  color: red;
  cursor: pointer;
}
```


Code HTML

```
<input type=text placeholder="Votre nom">
```

CSS

Pour modifier la couleur de la bordure autour de la zone de saisie, on peut utiliser la propriété `outline-color`

```
input {
  outline-color: teal;
}
```

CSS

Pour modifier la couleur de la bordure autour de la zone de saisie, on peut utiliser la propriété `outline-color`

```
input {
  outline-color: teal;
}
```

Code HTML

```
<input type=text placeholder="Votre nom">
```

CSS

Pour modifier la couleur de la bordure autour de la zone de saisie, on peut utiliser la propriété `outline-color`

```
input {
  outline-color: teal;
}
```

Code HTML

```
<input type=text placeholder="Votre nom">
```


Nous pouvons également utiliser

- `outline-style` pour modifier le style (`double`, `dashed`...) de la bordure autour de la zone de saisie
- `outline-width` pour modifier l'épaisseur de la bordure autour de la zone de saisie

CSS

Pour modifier la couleur d'une zone de choix si la case est cochée, on peut utiliser la propriété `accent-color`

```
input {
  accent-color: hotpink;
}
```


Code HTML

```
<input type=checkbox> Sportif ?
```

CSS

`position : static`

- La position, par défaut, naturelle sur la page.
- La position de l'élément ne peut jamais changer même si les propriétés `top`, `right`, `bottom` et `left` sont spécifiées.

CSS

`position : static`

- La position, par défaut, naturelle sur la page.
- La position de l'élément ne peut jamais changer même si les propriétés `top`, `right`, `bottom` et `left` sont spécifiées.

`position : relative`

- La position qui peut changer par rapport à la position statique de l'élément.
- La position de l'élément peut changer si une valeur a été spécifiée pour au moins une des ces propriétés `top`, `right`, `bottom` et `left`.

CSS

Pour tester la différence entre `static` et `relative`

```
div.static {
  position: static;
  left: 30px;
  border: 1px solid red;
  width: 400px;
  height: 200px;
}
div.relative {
  position: relative;
  left: 30px;
  border: 1px solid red;
  width: 400px;
  height: 200px;
}
```

HTML

```
<div class="static"> static </div>

<div class="relative"> relative </div>
```

CSS

`position : absolute`

- Position définie par rapport au block englobant
- Se réfère à son premier ancêtre positionné (ayant une position non `static`) qu'il rencontre.

CSS

Pour tester la différence entre `absolute`, `relative` et `static`, on ajoute la classe suivante dans le fichier CSS

```
div.absolute {
  position: absolute;
  top: 80px;
  left: 30px;
  width: 200px;
  height: 100px;
  border: 1px solid red;
}
```


HTML

```
<div class="static"> static </div>

<div class="relative"> relative </div>

<div class="absolute"> absolute </div>
```

CSS

position : fixed

position qui reste fixe, l'élément est toujours positionné par rapport à la fenêtre du navigateur même si on descend dans la page (avec un scroll par exemple).

CSS

position : fixed

position qui reste fixe, l'élément est toujours positionné par rapport à la fenêtre du navigateur même si on descend dans la page (avec un scroll par exemple).

Pour tester `fixed`

```
div.fixed {
  position: fixed;
  border: 1px solid red;
}
```

HTML

```
<div class ="fixed"> fixed </div>

<img src="http://www.lsis.org/elmouelhia/images/Bruxelles
   .jpg" alt="">
```

CSS

`position : sticky` (collant)

position dépendante de défilement de l'utilisateur

CSS

`position : sticky` (collant)

position dépendante de défilement de l'utilisateur

Exemple : https://www.w3schools.com/css/tryit.asp?filename=trycss_position_sticky

CSS

La propriété `z-index`

- `z-index` prend une valeur numérique. L'élément avec un plus grand `z-index` sera placé au dessus des autres.
- La propriété ne s'applique que sur des éléments ayant une position `absolute`, `relative` ou `fixed`.

CSS

Propriété et valeur

- `width` (largeur) : valeur en pixel, pourcentage relatif
- `height` (hauteur) : valeur en pixel, pourcentage relatif
- `min-width` (largeur minimale) : valeur en pixel, pourcentage relatif
- `min-height` (hauteur minimale) : valeur en pixel, pourcentage relatif
- `max-width` (largeur minimale) : valeur en pixel, pourcentage relatif
- `max-height` (hauteur minimale) : valeur en pixel, pourcentage relatif

CSS

Deux types de marges

- `margin` (marge extérieure)
- `padding` (marge intérieure)

CSS

marge extérieure : espace entre les boites

- `margin-top` (marge extérieure en haut)
- `margin-bottom` (marge extérieure en bas)
- `margin-right` (marge extérieure à droite)
- `margin-left` (marge extérieure à gauche)
- `margin` (marge extérieure haute, basse, droite et gauche)

CSS

marge intérieure : espace entre le contenu et la limite de la boite

- `padding-top` (marge intérieure en haut)
- `padding-bottom` (marge intérieure en bas)
- `padding-right` (marge intérieure à droite)
- `padding-left` (marge intérieure à gauche)
- `padding` (marge intérieure haute, basse, droite et gauche)

CSS

`margin` (ou `padding`) peut prendre :

- une seule valeur : appliquée aux quatre cotés de la boîte.
- deux valeurs : la première pour `top`, la deuxième pour `right` et `left` et la troisième pour `bottom`.
- trois valeurs : la première pour `top` et `bottom` et la deuxième pour `right` et `left`.
- quatre valeurs respectivement pour `top`, `right`, `bottom` et `left`.

CSS

`margin` accepte comme valeur :

- une longueur en `px`, `em`...
- un pourcentage.
- `auto` permet de centrer un élément dans son conteneur.

CSS

`margin` accepte comme valeur :

- une longueur en `px`, `em`...
- un pourcentage.
- `auto` permet de centrer un élément dans son conteneur.

`padding` accepte comme valeur :

- une longueur en `px`, `em`...
- un pourcentage.

CSS

La propriété `overflow`

Et si notre texte ne rentre pas dans la boite, on peut utiliser la propriété `overflow` pour indiquer ce qu'il faut faire

- `hidden` : cacher le texte qui dépasse
- `visible` : le texte reste visible à l'extérieur de la boite
- `scroll` : ajouter une barre de défilement pour parcourir le texte qui dépasse
- `auto` : le navigateur décide d'ajouter une barre de défilement si nécessaire

CSS

La propriété `float`

permet de déclarer et choisir l'emplacement d'un objet flottant.

CSS

La propriété float

permet de déclarer et choisir l'emplacement d'un objet flottant.

Les valeurs de la propriété float

- left
- right

Les valeurs de la propriété clear : pour annuler float

- left
- right
- both

CSS

Considérons l'exemple suivant :

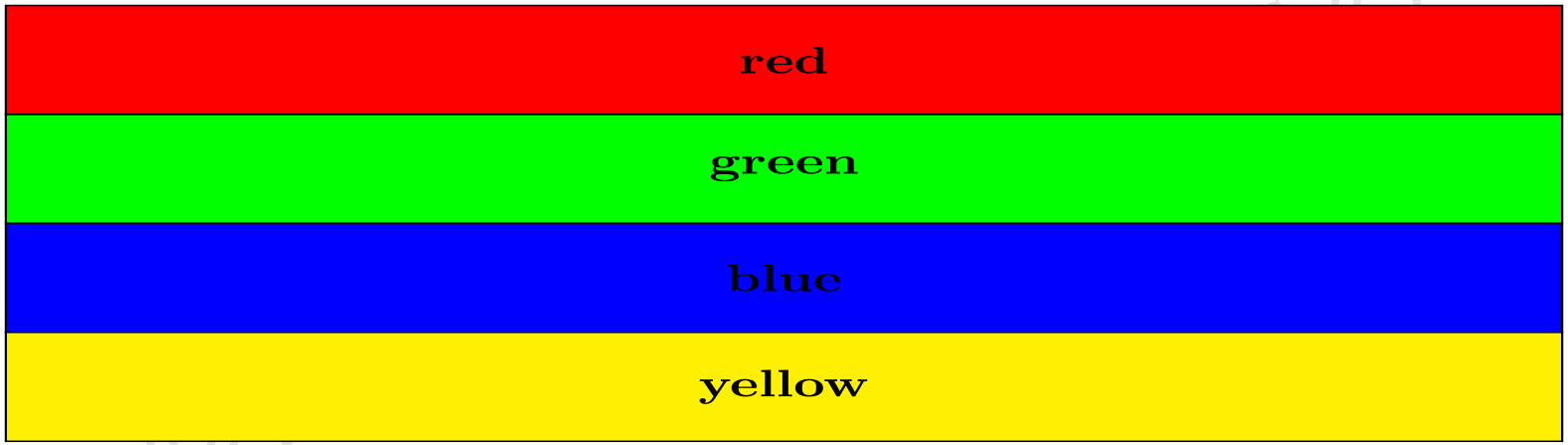
Le fichier `index.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title> float et clear </title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="container" >
      <div class=component1 >red</div>
      <div class=component2 >green</div>
      <div class=component3 >blue</div>
      <div class=component4 >yellow</div>
    </div>
  </body>
</html>
```

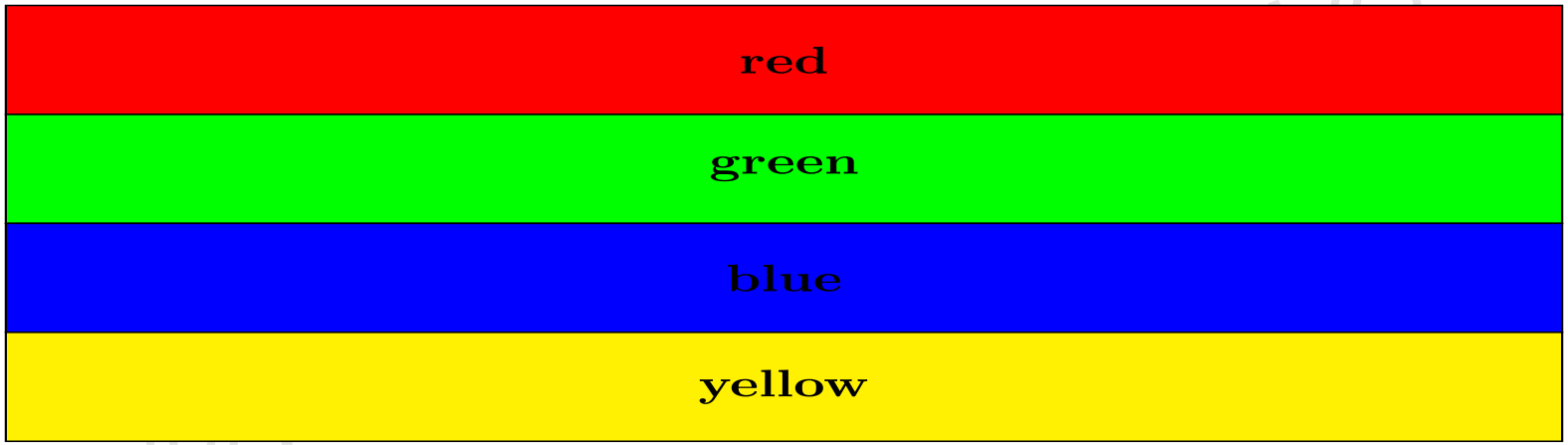
Le fichier `style.css`

```
.component1 {
  background-color: red;
}
.component2 {
  background-color: green;
}
.component3 {
  background-color: blue;
}
.component4 {
  background-color: yellow;
}
.container > div {
  text-align: center;
}
```

Le résultat du code précédent :



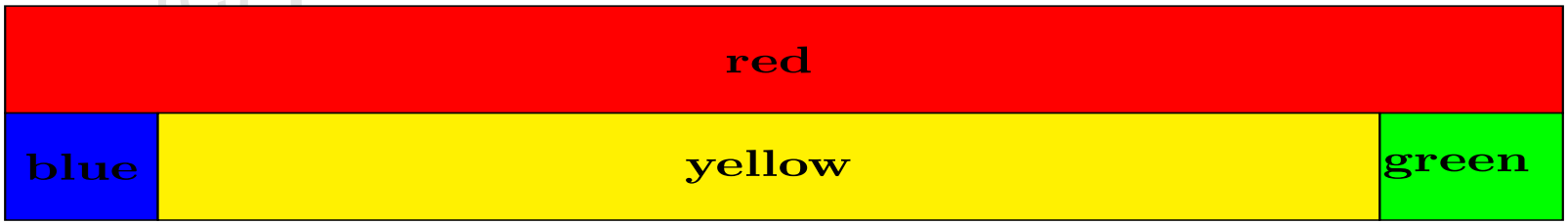
Le résultat du code précédent :



Et si on veut placer le bleu à gauche du vert ?

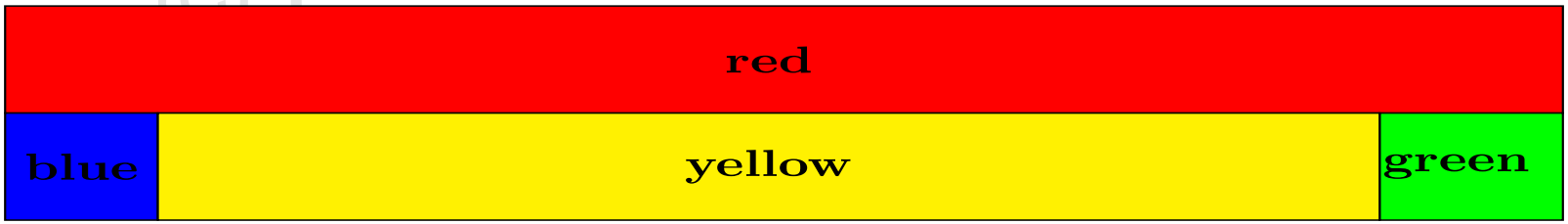
CSS

```
.component2 {
  float: right;
  background-color: green;
}
.component3 {
  float: left;
  background-color: blue;
}
```



CSS

```
.component2 {
  float: right;
  background-color: green;
}
.component3 {
  float: left;
  background-color: blue;
}
```

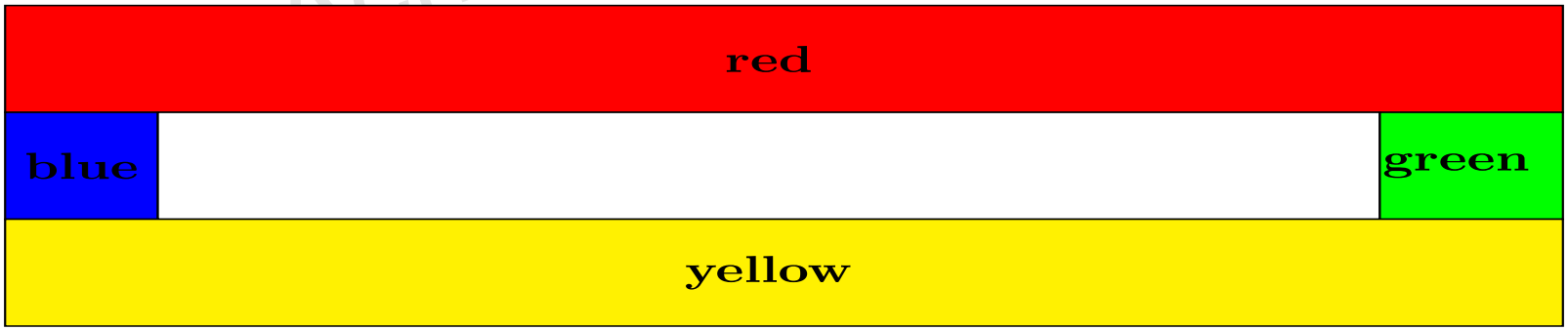


Oui, mais le jaune, on voulait qu'il reste à la ligne

CSS

Dans ce cas, il faut ajouter

```
.component4 {
  clear: both;
  background-color: yellow;
}
```



CSS

La propriété visibility

Plusieurs valeurs possibles pour visibility :

- hidden : pour rendre un élément invisible (sa place est conservée)
- visible : pour rendre un élément visible
- collapse : pour rendre un élément du tableau invisible, mais la place qu'il occupe est perdue (propriété compatible seulement avec IE et Firefox)
- inherit : pour avoir la même visibilité de l'élément parent

CSS

collapse vs hidden : CSS

```
#hiddenRow {
        visibility: hidden;
}
#collapseRow {
        visibility: collapse;
}
```

collapse vs hidden : HTML

```
<table>
        <tr><td> 1 </td><td> 2 </td></tr>
        <tr id=hiddenRow><td> 3 </td><td> 4 </td></tr>
        <tr><td> 5 </td><td > 6 </td></tr>
        <tr id=collapseRow><td> 7 </td><td > 8 </td></tr>
        <tr><td > 9 </td><td> 10 </td></tr>
</table>
```

CSS

collapse vs hidden : CSS

```
#hiddenRow {
        visibility: hidden;
}
#collapseRow {
        visibility: collapse;
}
```

collapse vs hidden : HTML

```
<table>
        <tr><td> 1 </td><td> 2 </td></tr>
        <tr id=hiddenRow><td> 3 </td><td> 4 </td></tr>
        <tr><td> 5 </td><td > 6 </td></tr>
        <tr id=collapseRow><td> 7 </td><td > 8 </td></tr>
        <tr><td > 9 </td><td> 10 </td></tr>
</table>
```

Résultat :

```
1       2

5       6
9       10
```

CSS

La propriété display

Chaque élément **HTML** a une propriété `display` qui est par défaut soit `inline` soit `block`.

CSS

La propriété display

Chaque élément **HTML** a une propriété `display` qui est par défaut soit `inline` soit `block`.

Autres valeurs possibles ?

Oui : `none`, `inline-block`, `table-cell`, `flex`, `grid`...

CSS

La propriété display

Chaque élément **HTML** a une propriété display qui est par défaut soit inline soit block.

Autres valeurs possibles ?

Oui : none, inline-block, table-cell, flex, grid...

La valeur none pour la propriété display

Un élément **HTML** ayant la valeur none pour la propriété display ne sera pas affiché, sans avoir à le supprimer réellement de la page (DOM).

CSS

`visibility:hidden` VS `display: none`

- `display: none` masque totalement l'élément et annule des propriétés telles que `margin, padding, width...`
- `visibility: hidden` masque seulement l'élément, ce qui peut laisser des espaces vides.

CSS

Pour tester : CSS

```
.propDisplay {
  display: none;
}
.propHidden {
  visibility: hidden;
}
```

HTML

```
Bonjour. Après les deux points, il y a un mot dont  display = none : <
   span class="propDisplay"> ICI </span>. Entre parentheses, il y a un
   mot dont visibility = hidden (<span class="propHidden"> ICI </span>)
   .
```

CSS

Pour tester : CSS

```
.propDisplay {
  display: none;
}
.propHidden {
  visibility: hidden;
}
```

HTML

```
Bonjour. Après les deux points, il y a un mot dont  display = none : <
   span class="propDisplay"> ICI </span>. Entre parentheses, il y a un
   mot dont visibility = hidden (<span class="propHidden"> ICI </span>)
   .
```

Le résultat

```
Bonjour. Après les deux points, il y a un mot dont display = none : .
   Entre parenthèses, il y a un mot dont visibility = hidden (     ).
```

CSS

Pour tester : CSS

```
.propDisplay {
  display: none;
}
.propHidden {
  visibility: hidden;
}
```

HTML

```
Bonjour. Après les deux points, il y a un mot dont  display = none : <
   span class="propDisplay"> ICI </span>. Entre parentheses, il y a un
   mot dont visibility = hidden (<span class="propHidden"> ICI </span>)
   .
```

Le résultat

```
Bonjour. Après les deux points, il y a un mot dont display = none : .
   Entre parenthèses, il y a un mot dont visibility = hidden (     ).
```

Aller vérifier le DOM dans le navigateur

CSS

La valeur inline-block de la propriété display

inline-block : permet de placer des éléments inline tout en préservant leurs capacités d'éléments block, tels que la possibilité de définir une largeur et une hauteur, des marges et padding top et bottom,...

Exemple avec block : CSS

```
.propBlock {
        display: block;
        height: 100px;
        width: 100px;
}
```

Exemple avec block : HTML

```
<p> Bonjour, ceci est une balise <span class=propBlock> block </span>,
   aurevoir </p>
<p> CSS3 </p>
```

Exemple avec block : CSS

```
.propBlock {
        display: block;
        height: 100px;
        width: 100px;
}
```

Exemple avec block : HTML

```
<p> Bonjour, ceci est une balise <span class=propBlock> block </span>,
   aurevoir </p>
<p> CSS3 </p>
```


Le résultat

```
Bonjour, ceci est une balise
block



, aurevoir
```

Exemple avec inline : CSS

```
.propInline {
        display: inline;
        height: 100px;
        width: 100px;
}
```

Exemple avec inline : HTML

```
<p> Bonjour, ceci est une balise <span class=propInline> inline</span>,
    aurevoir </p>
<p> CSS3 </p>
```

Exemple avec inline : CSS

```
.propInline {
        display: inline;
        height: 100px;
        width: 100px;
}
```

Exemple avec inline : HTML

```
<p> Bonjour, ceci est une balise <span class=propInline> inline</span>,
    aurevoir </p>
<p> CSS3 </p>
```


Le résultat

```
Bonjour, ceci est une balise inline, aurevoir

CSS3
```

Exemple avec inline-block : CSS

```
.propInlineBlock {
        display: inline-block;
        height: 100px;
        width: 100px;
}
```

Exemple avec inline-block : HTML

```
<p> Bonjour, ceci est une balise <span class=propInlineBlock> inline-
  block </span>, aurevoir </p>
<p> CSS3 </p>
```

Exemple avec inline-block : CSS

```
.propInlineBlock {
        display: inline-block;
        height: 100px;
        width: 100px;
}
```

Exemple avec inline-block : HTML

```
<p> Bonjour, ceci est une balise <span class=propInlineBlock> inline-
  block </span>, aurevoir </p>
<p> CSS3 </p>
```


Le résultat

```
Bonjour, ceci est une balise inline-block      , aurevoir



CSS3
```

CSS

Remarque

La propriété `display: inline-block` ajoute un léger espace blanc entre les éléments.

CSS

Remarque

La propriété `display: inline-block` ajoute un léger espace blanc entre les éléments.

Plusieurs solutions pour supprimer l'espace blanc ajouté par `display: inline-block`

- Attribuer une valeur négative à `margin-right`
- Attribuer la valeur zéro à la propriété `font-size` du parent
- Utiliser `flex` ou `grid`

CSS

Autres valeurs de display

- list-item : pour afficher sous forme de liste, avec retour à la ligne après chaque élément.
- inherit : pour avoir les propriétés du block conteneur.
- flex
- grid
- table
- ...

CSS

Exercice

Avant de continuer avec **Flexbox** et **Grid**, faisons un exercice sur les menus.

CSS

Exercice

Avant de continuer avec **Flexbox** et **Grid**, faisons un exercice sur les menus.

Comment créer un menu ?

- Éléments `<ul>` et `<li>` + `<a>` pour les actions (liens)
- Styles CSS pour l'affichage des différents menus et sous-menus
- Exploitation des pseudo-classes (ex `:hover`) pour faire apparaître/disparaître les sous-menus

CSS

```
<nav>
  <ul id="menu">
    <li><a href="page1.html">menu1</a>
      <ul class="sous-menu">
        <li><a href="page11.html">menu11</a></li>
        <li><a href="page12.html">menu12</a></li>
      </ul>
    </li>
    <li><a href="page2.html">menu2</a></li>
    <li><a href="page3.html">menu3</a></li>
    <li><a href="page4.html">menu4</a>
      <ul class="sous-menu">
        <li><a href="page41.html">menu41</a></li>
        <li><a href="page42.html">menu42</a></li>
      </ul>
    </li>
  </ul>
</nav>
```

CSS

Exercice, en se basant sur le code **HTML** précédent

- écrire un premier code **CSS** permettant de créer un menu vertical
- écrire un deuxième code **CSS** permettant de créer un menu horizontal

CSS

Solution pour le menu vertical

```
ul {
    list-style-type: none;
}

.sous-menu {
    display: none;
}

li:hover > ul {
    display: list-item;

}
```

CSS

Une deuxième solution pour le menu vertical consistant à afficher les sous-menu à coté du menu principal

```
ul {
    list-style-type: none;
}

.sous-menu {
    display: none;
}

li:hover > ul {
    display: inline-block;
    position: absolute;
    padding-left: 0px;
    padding-top: 0px;
}
```

CSS

Solution pour le menu horizontal

```
ul {
    list-style-type: none;
}

li > ul {
    display: none;
}

#menu > li {
    display: inline-block;
    margin-left: 100px;
}

li:hover > ul {
    display: block;
    position: absolute;
    padding: 0px;
}
```

CSS

Considérons l'exemple suivant :

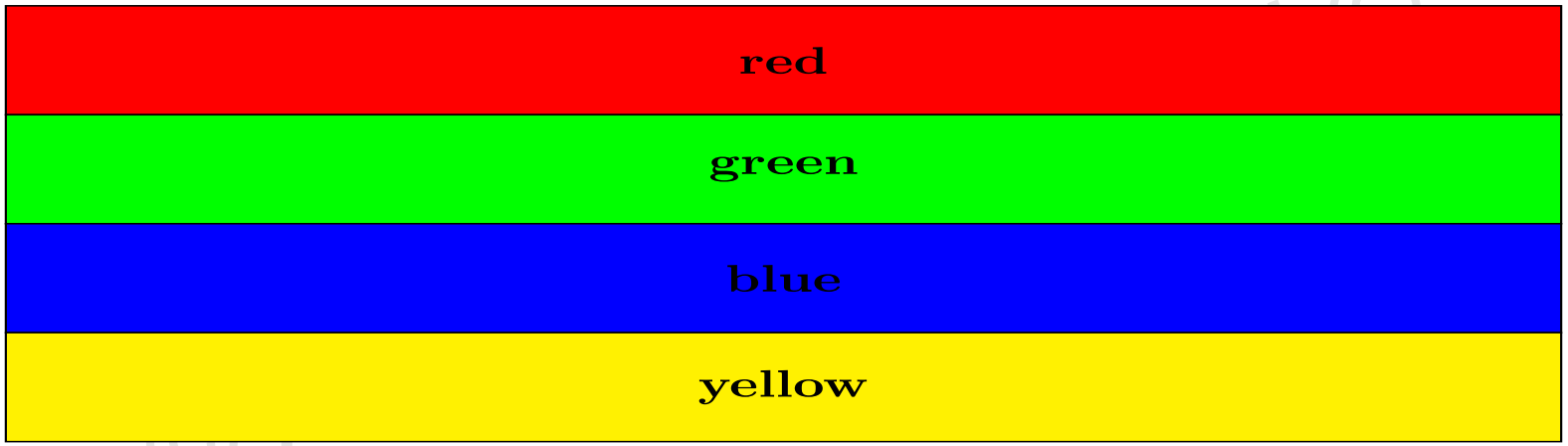
Le fichier `flex.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title> flex </title>
    <link rel="stylesheet" href="flex.css">
  </head>
  <body>
    <div class="container">
      <div class=component1 >red</div>
      <div class=component2 >green</div>
      <div class=component3 >blue</div>
      <div class=component4 >yellow</div>
    </div>
  </body>
</html>
```

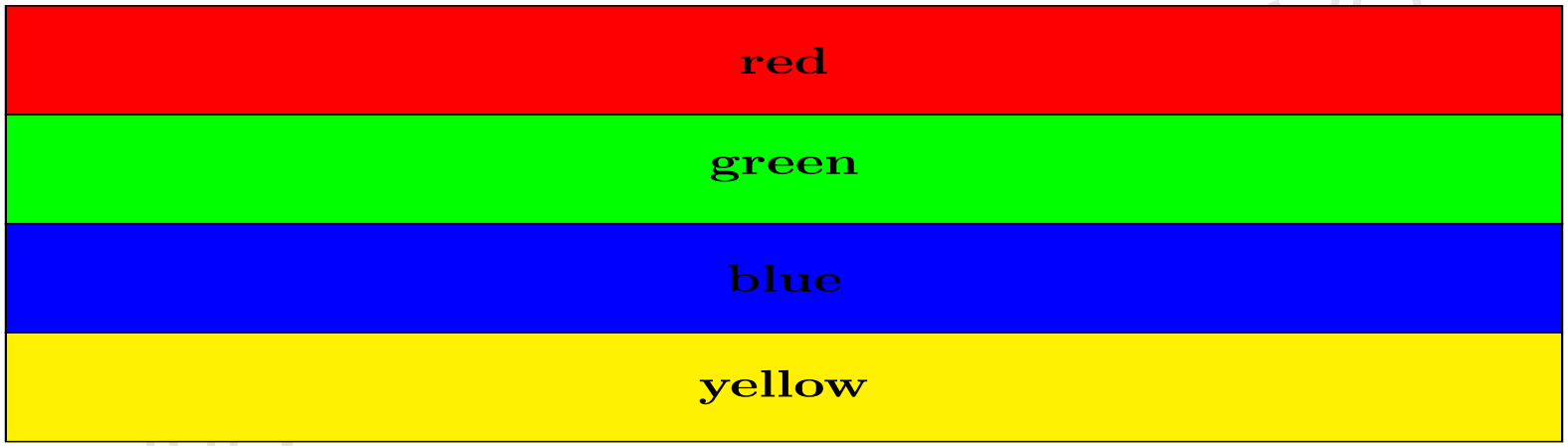
Le fichier `flex.css`

```
.component1 {
  background-color: red;
}
.component2 {
  background-color: green;
}
.component3 {
  background-color: blue;
}
.component4 {
  background-color: yellow;
}
.container {
  background-color: black;
}
```

Le résultat du code précédent :



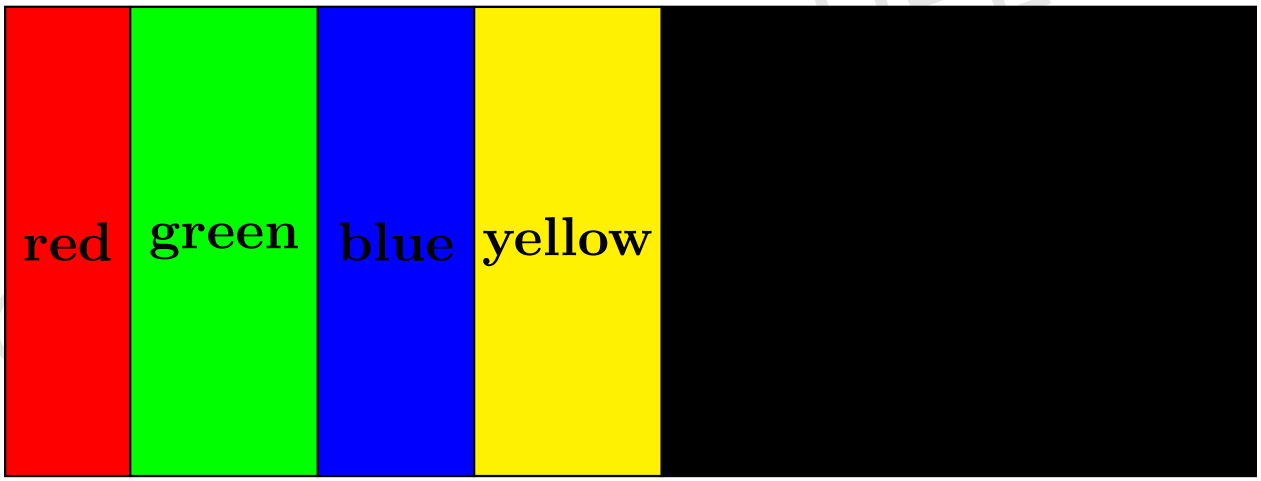
Le résultat du code précédent :



Et si on veut placer les balises `div` les unes à coté des autres ?

CSS

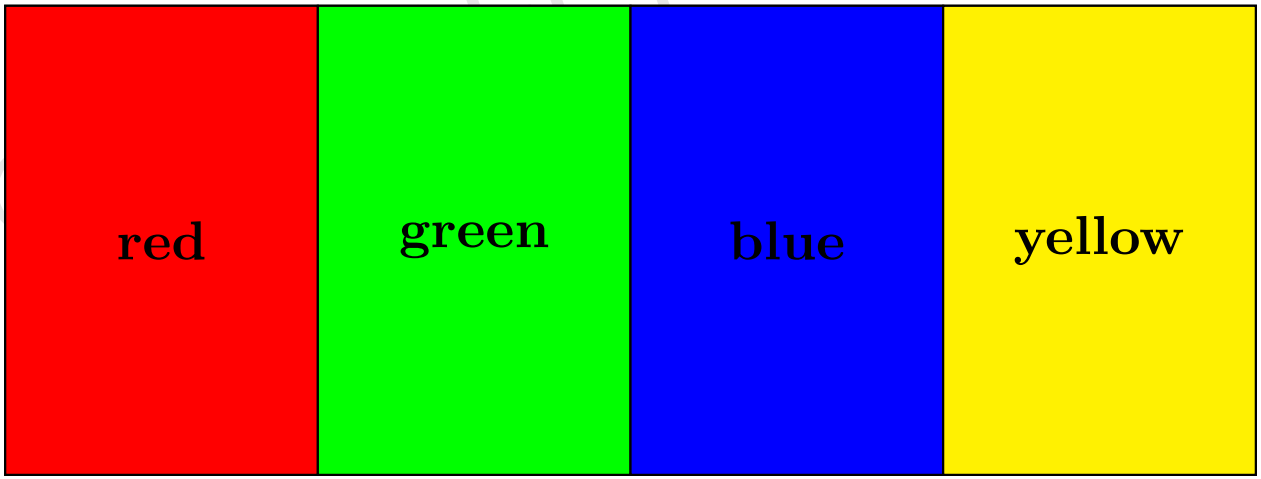
```
.container {
  display: flex;
}
```



Et si on veut donner la même largeur à toutes les `div` ?

CSS

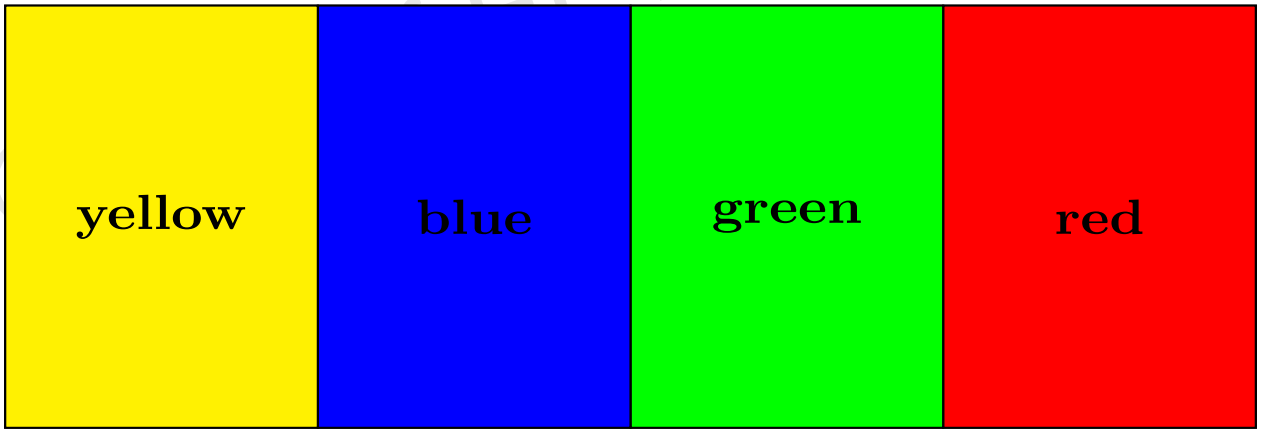
```
.container {
  display: flex;
}
.container > div {
  width: 30%;
}
```



Et si on veut afficher les `div` dans le sens opposé ?

CSS

```
.container {
  background-color: black;
  display: flex;
  flex-direction: row-reverse;
}
```



CSS

Autres valeurs de `flex-direction`

- `row` (par défaut) : organiser les boites sur une ligne.
- `row-reverse` : organiser les boites sur une ligne dans le sens inverse.
- `column` : organiser les boites sur une colonne.
- `column-reverse` : organiser les boites sur une colonne dans le sens inverse.

CSS

Autres valeurs de `flex-direction`

- `row` (par défaut) : organiser les boites sur une ligne.
- `row-reverse` : organiser les boites sur une ligne dans le sens inverse.
- `column` : organiser les boites sur une colonne.
- `column-reverse` : organiser les boites sur une colonne dans le sens inverse.

Remarque

- Les 4 `div` de l'exemple précédent avait chacune une largeur de 30%. Mais **Flexbox** les place, par défaut, sur une seule ligne.
- Et si on veut le forcer à retourner à la ligne si espace insuffisant ? $\Rightarrow$ il faut utiliser la propriété `flex-wrap`.

CSS

La propriété `flex-wrap`

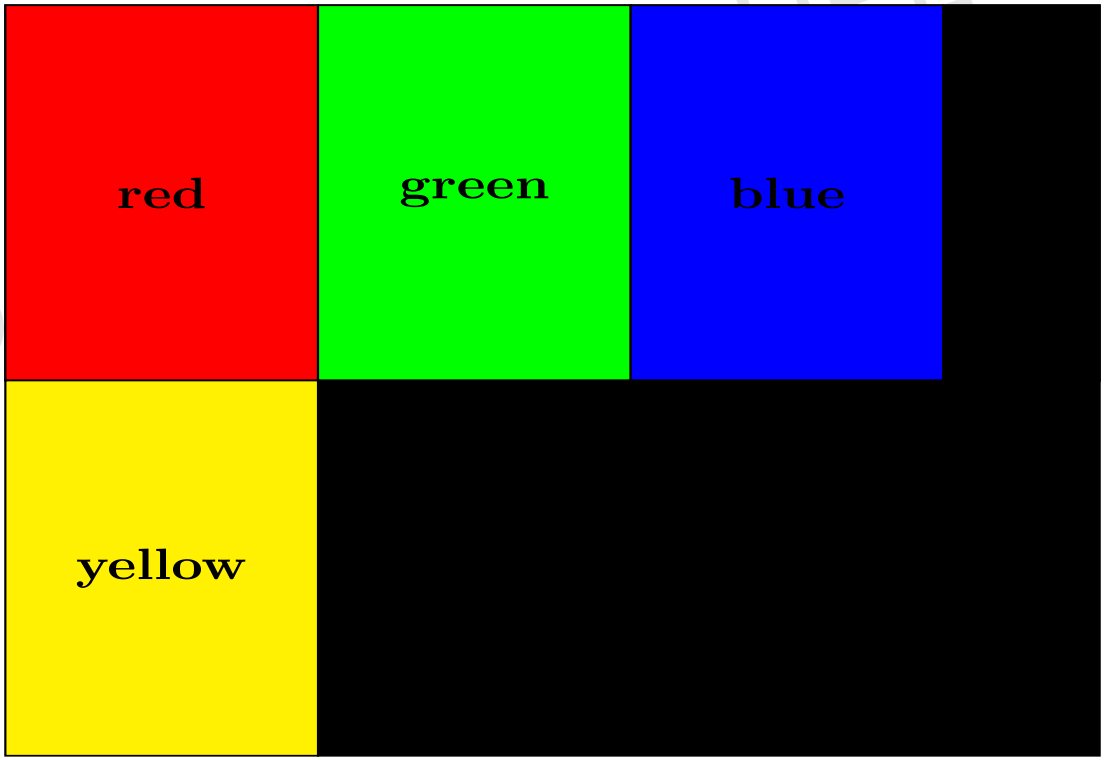
Pour indiquer s'il faut retourner à la ligne si la boite container ne suffit pas

- `nowrap` (par défaut) : organiser les boites sans retour à la ligne
- `wrap` : retourner à la ligne si place insuffisante
- `wrap-reverse` : retourner à la ligne si place insuffisante dans le sens inverse

CSS

Revenons à l'exemple précédent

```
.container {
  background-color: black;
  display: flex;
  flex-wrap: wrap;
}
```



CSS

flex-flow

- raccourci de `flex-direction` et `flex-wrap`
- exemple : `flex-flow : row wrap;`

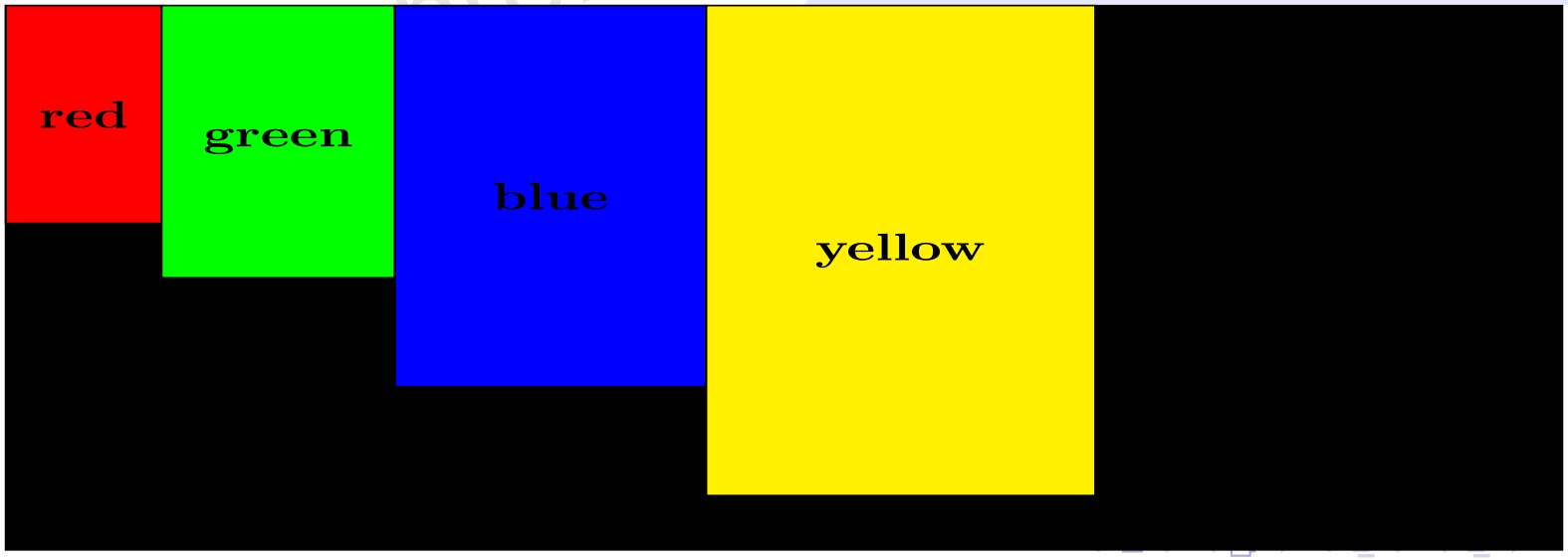
CSS

Trois propriétés pour l'alignement des boites

- `justify-content` : permet d'aligner horizontalement les boites d'une même ligne (si elles n'utilisent pas tout l'espace valable).
- `align-items` : permet de définir la disposition verticale des boites d'une même ligne (si elles n'utilisent pas tout l'espace valable).
- `align-content` : permet d'aligner verticalement les lignes de boites d'un conteneur (sans effet si toutes les boites sont sur une seule ligne)

Pour la suite, on considère le code CSS suivant

```
.component1 {
  background-color: red;
  width : 10%;
  height: 80px;
}
.component2 {
  background-color: green;
  width : 15%;
  height: 100px;
}
.component3 {
  background-color: blue;
  width : 20%;
  height: 140px;
}
.component4 {
  background-color: yellow;
  width : 25%;
  height: 180px;
}
.container {
  background-color: black;
  display: flex;
  height: 250px;
}
```



CSS

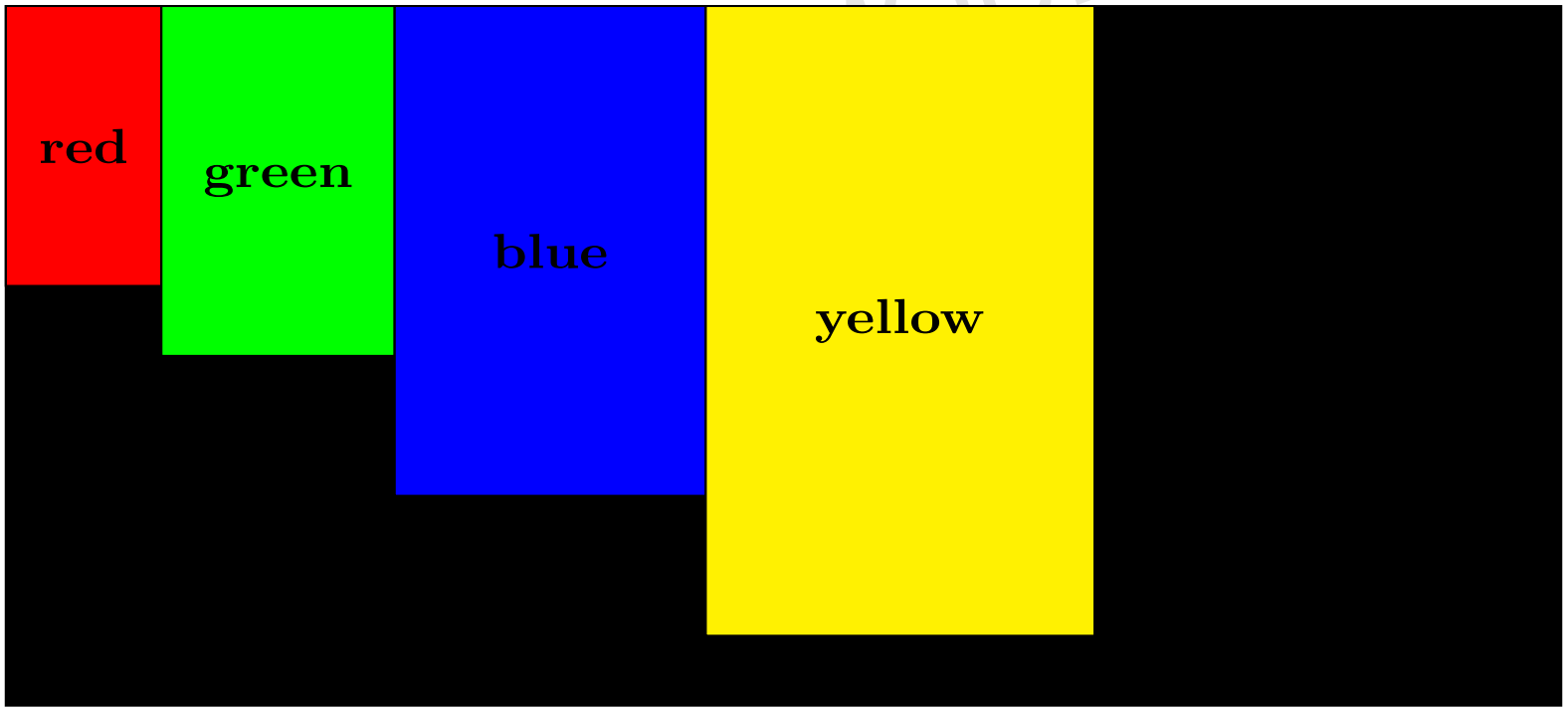
justify-content : pour l'alignement des boites (horizontal)

- center : pour centrer les éléments dans le conteneur
- flex-start : pour aligner les éléments au début du conteneur
- flex-end : pour aligner les éléments à l'extrémité gauche du conteneur
- space-around : pour ajouter de l'espace autour de chaque élément
- space-between : pour ajouter de l'espace entre les éléments

CSS

`flex-start` est la valeur par défaut de `justify-content` et ne change donc rien de la configuration précédente

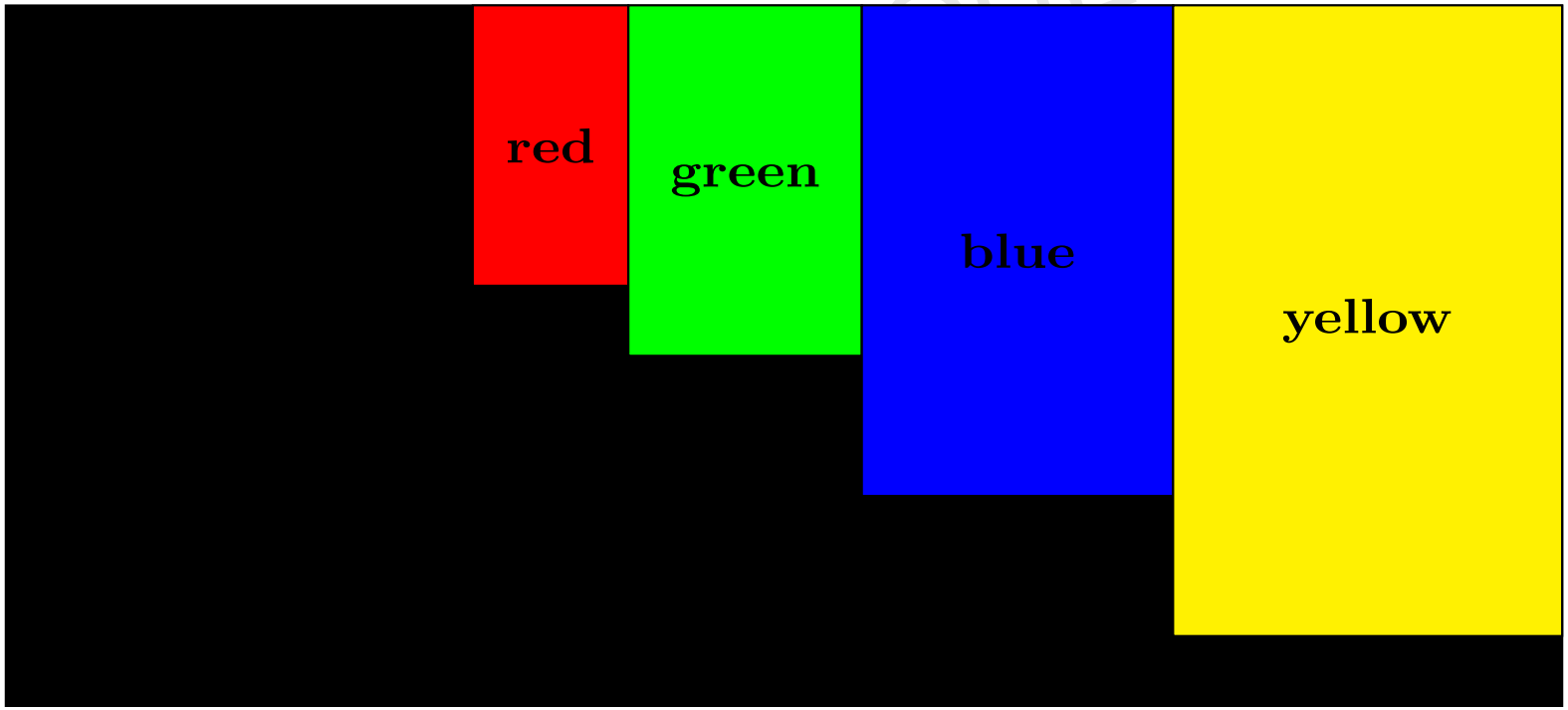
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  justify-content: flex-start;
}
```



CSS

Résultat de `justify-content: flex-end`

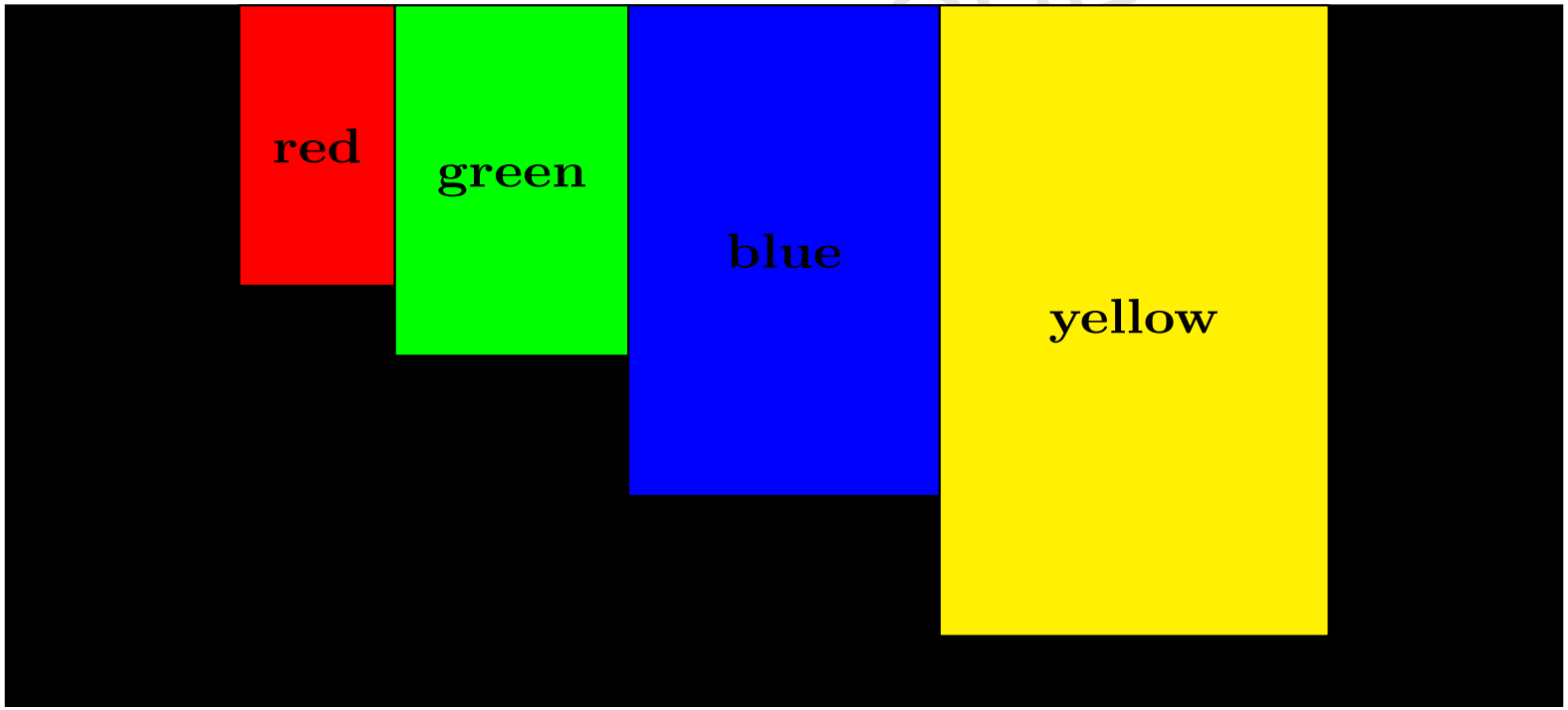
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  justify-content: flex-end;
}
```



CSS

Résultat de `justify-content: center`

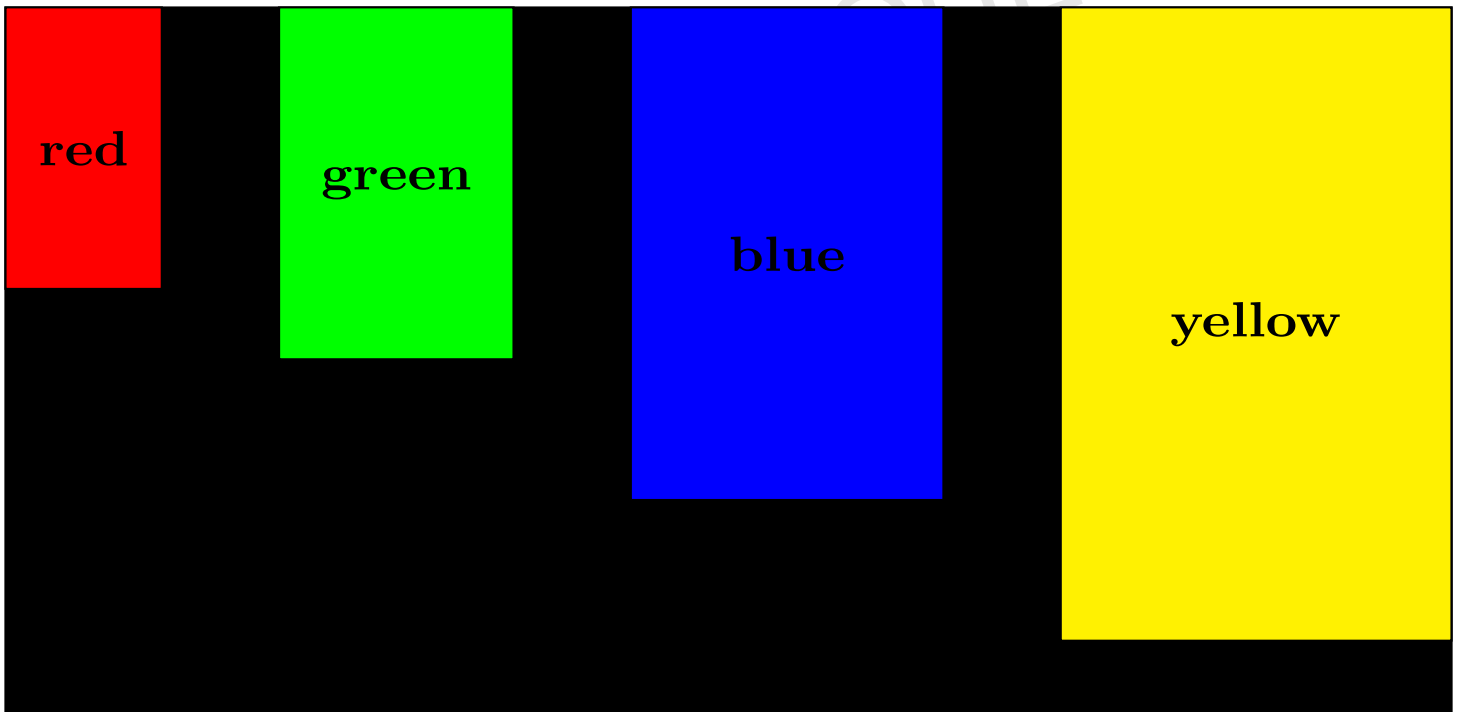
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  justify-content: center;
}
```



CSS

Résultat de `justify-content: space-between`

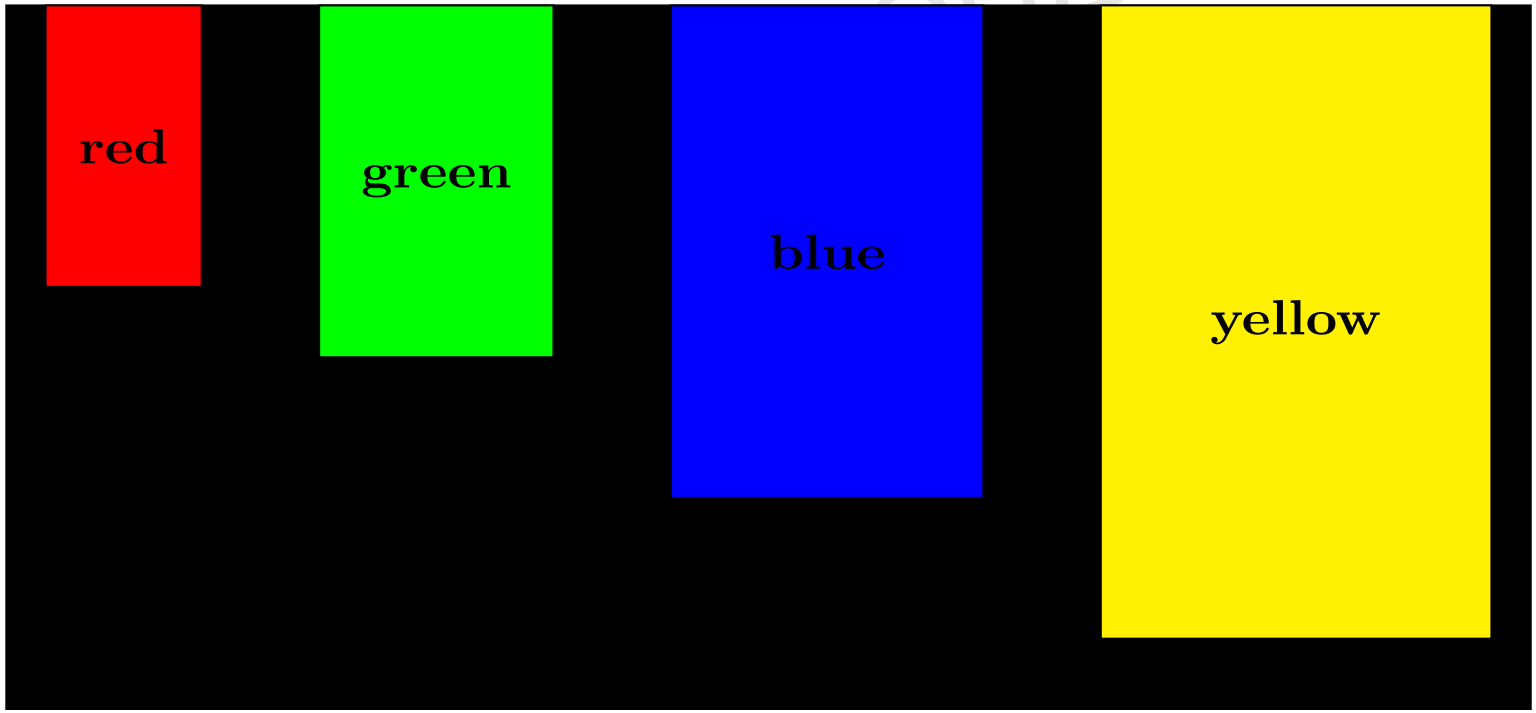
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  justify-content: space-between;
}
```



CSS

Résultat de `justify-content: space-around`

```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  justify-content: space-around;
}
```



CSS

`align-items` : pour l'alignement (vertical)

- peut prendre les valeurs `center`, `flex-start` et `flex-end` tout comme `justify-content`
- ou aussi `stretch` (valeur par défaut, rempli le conteneur en hauteur avec les éléments) ou `baseline` (qui aligne les éléments flexibles d'une façon que leurs lignes de base soient alignées)

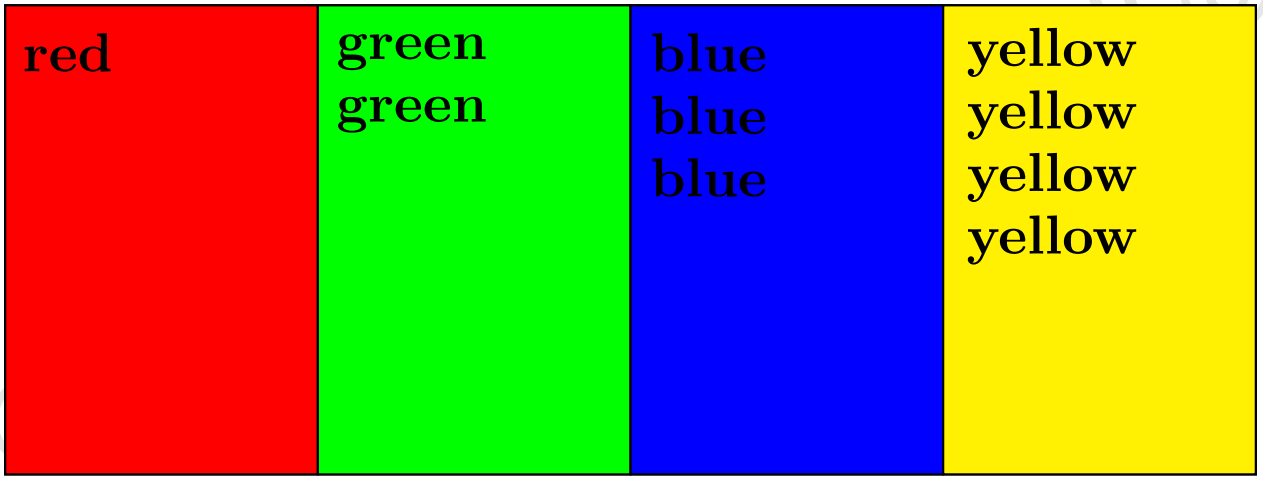
CSS

Pour la suite, on considère les codes HTML et CSS suivants

```
<div class="container" >
  <div class=component1>
    red
  </div>
  <div class=component2>
    green <br> green
  </div>
  <div class=component3>
    blue <br> blue <br> blue
  </div>
  <div class=component4>
    yellow <br> yellow <br>
    yellow <br> yellow
  </div>
</div>
```

```
.component1 {
  background-color: red;
}
.component2 {
  background-color: green;
}
.component3 {
  background-color: blue;
}
.component4 {
  background-color: yellow;
}
.container > div {
  width: 30%;
}
.container {
  background-color: black;
  display: flex;
  height: 200px;
}
```

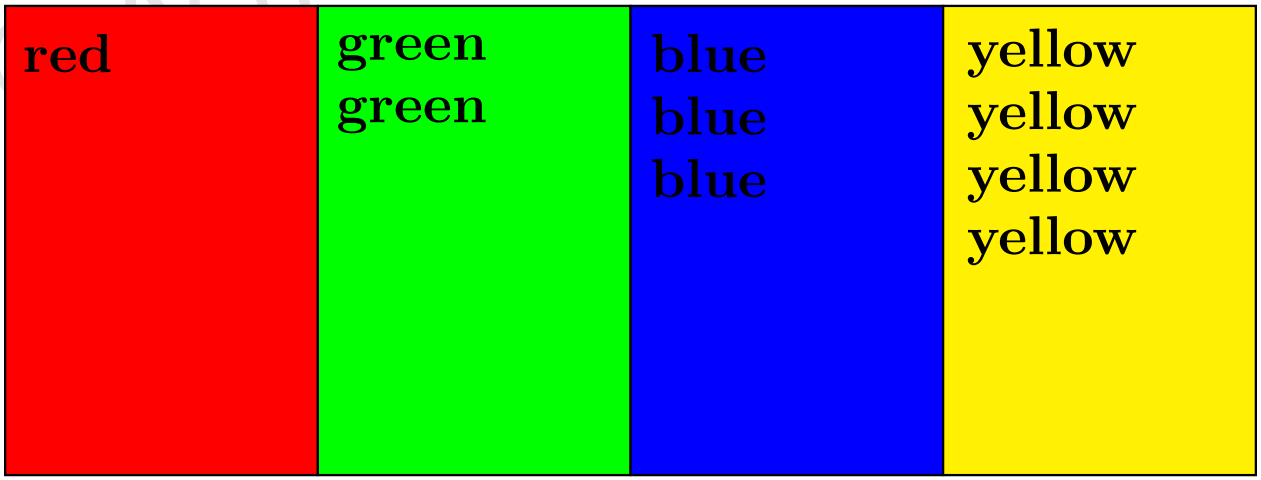
Le code précédent permet l'emplacement suivant



CSS

`stretch` est la valeur par défaut de `align-items` et ne change donc rien de la configuration de départ

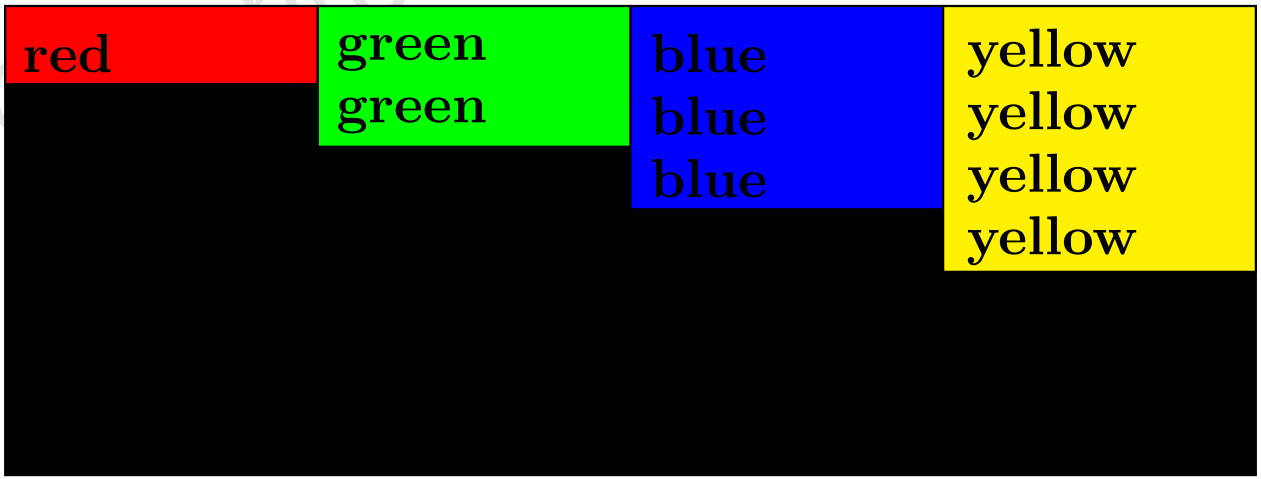
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  align-items: stretch;
}
```



CSS

Résultat de la valeur `flex-start` affectée à `align-items`

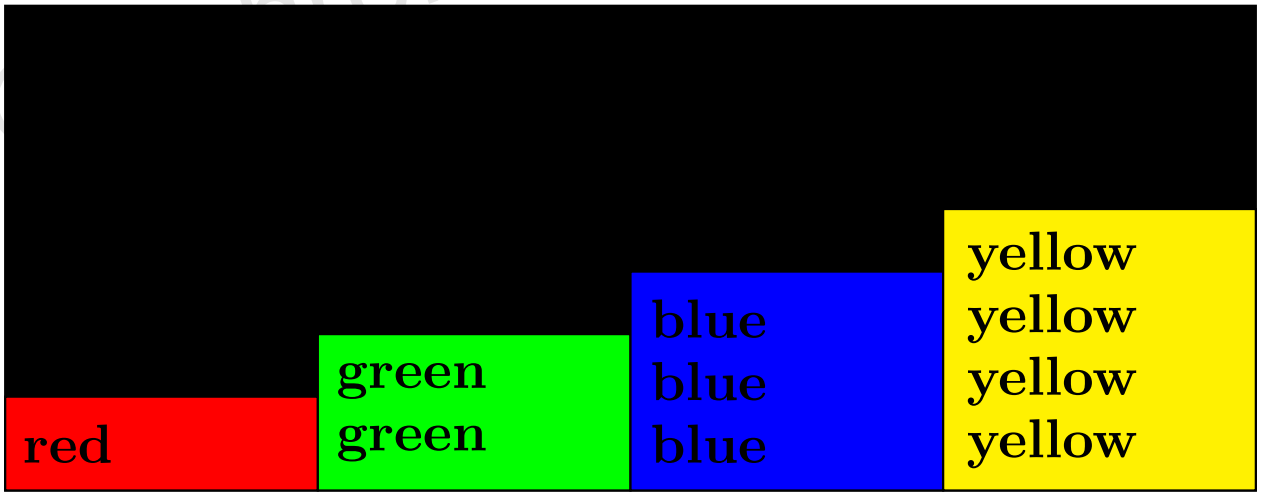
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  align-items: flex-start;
}
```



CSS

Résultat de la valeur `flex-end` affectée à `align-items`

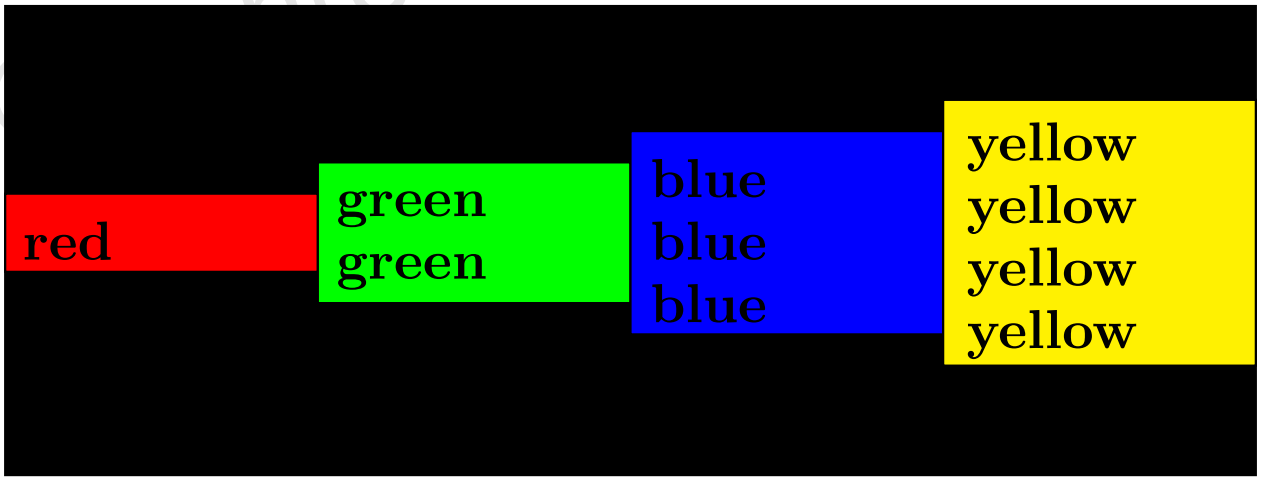
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  align-items: flex-end;
}
```



CSS

Résultat de la valeur `center` affectée à `align-items`

```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  align-items: center;
}
```



CSS

Pour pouvoir tester la valeur `baseline`, on ajoute un `padding-top` pour quelques components

```
.component1 {
  background-color: red;
}
.component2 {
  background-color: green;
  padding-top: 20px;
}
.component3 {
  background-color: blue;
  padding-top: 10px;
}
.component4 {
  background-color: yellow;
  padding-top: 30px;
}
```

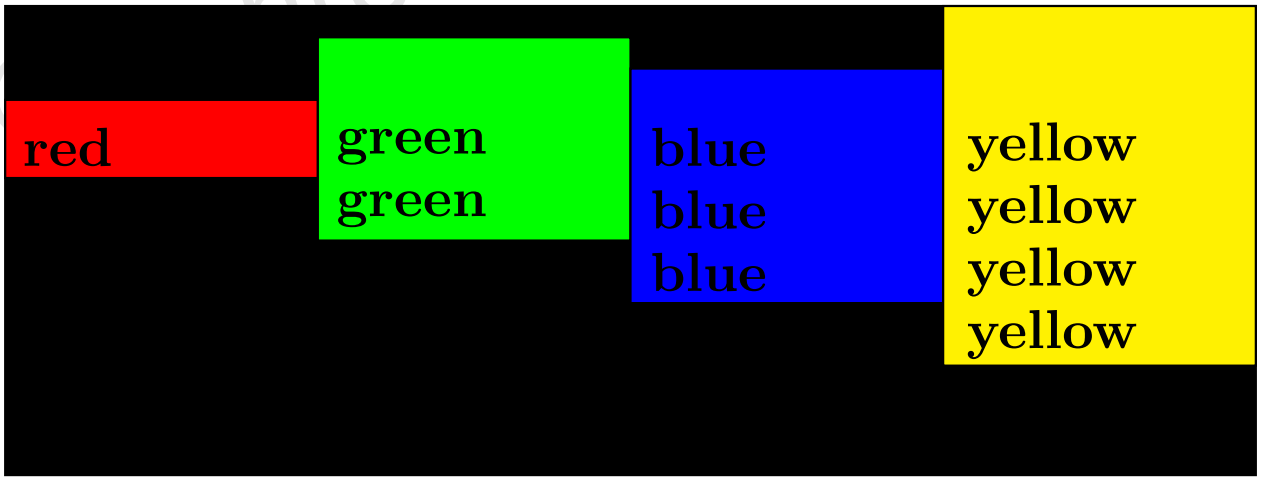
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  align-items: baseline;
}
.container > div {
  width: 30%;
}
```

Le HTML ne change pas.

CSS

Résultat de la valeur `baseline` affectée à `align-items`

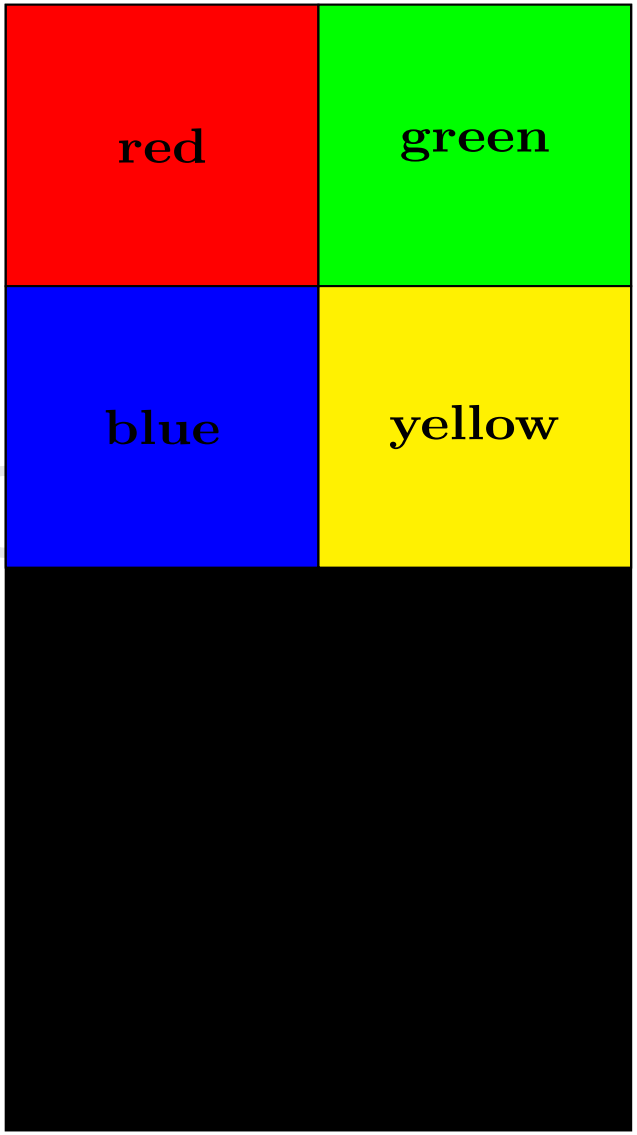
```
.container {
  background-color: black;
  display: flex;
  height: 200px;
  align-items: baseline;
}
```



Le fichier flex.css

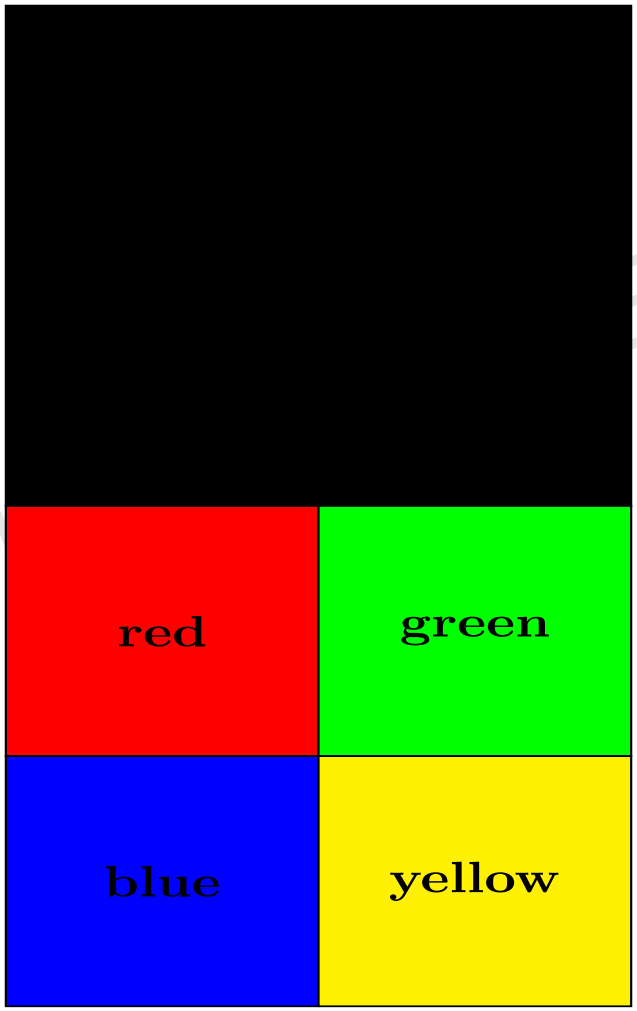
```
.component1 {
  background-color: red;
}
.component2 {
  background-color: green;
}
.component3 {
  background-color: blue;
}
.component4 {
  background-color: yellow;
}
.container {
  background-color: black;
  display: flex;
  flex-wrap: wrap;
  align-content: flex-start;
  height: 150px;
}
.container > div {
  width : 50%;
  height: 50px;
}
```

Le résultat



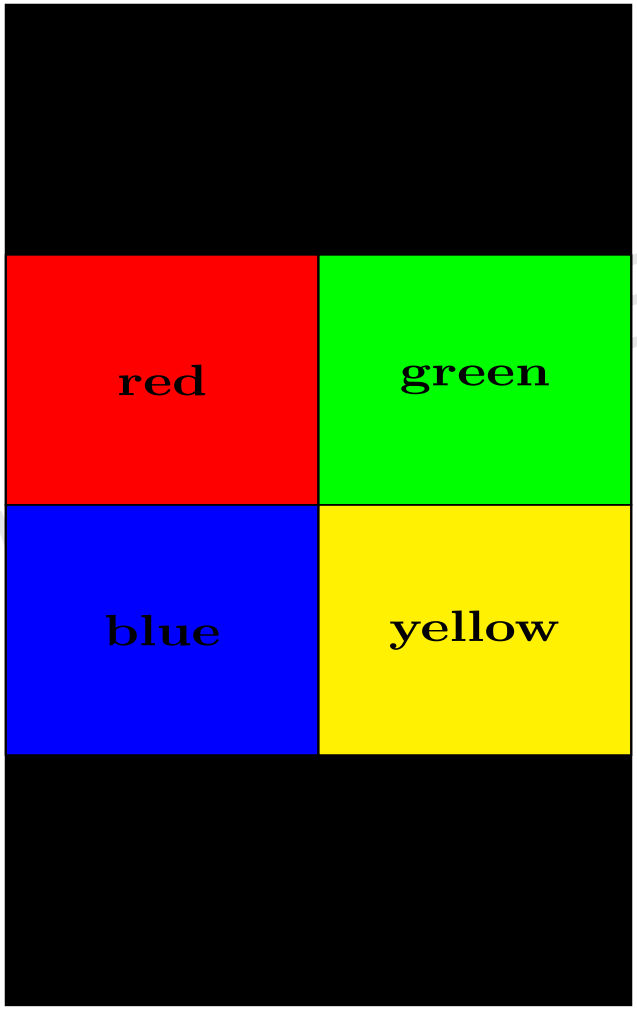
CSS

Exemple avec `align-content: flex-end;`



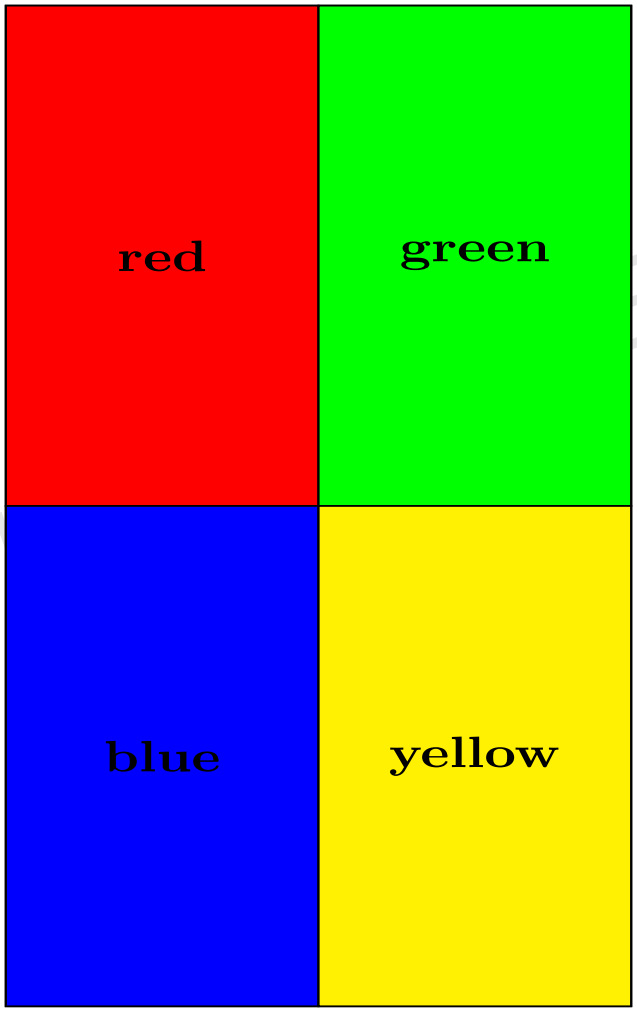
CSS

Exemple avec `align-content: center;`



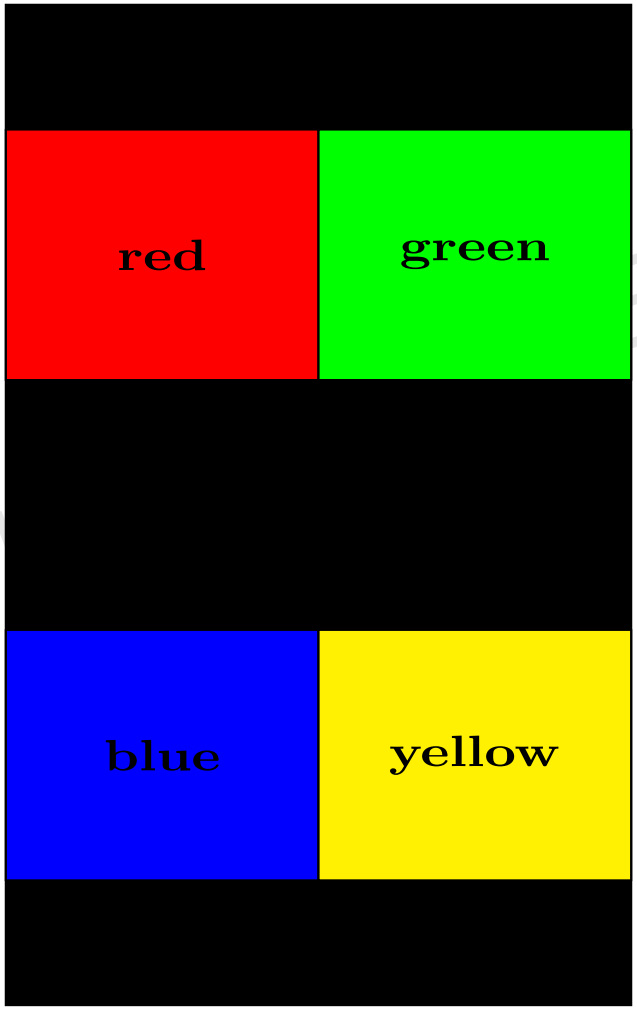
CSS

Exemple avec `align-content: stretch;`



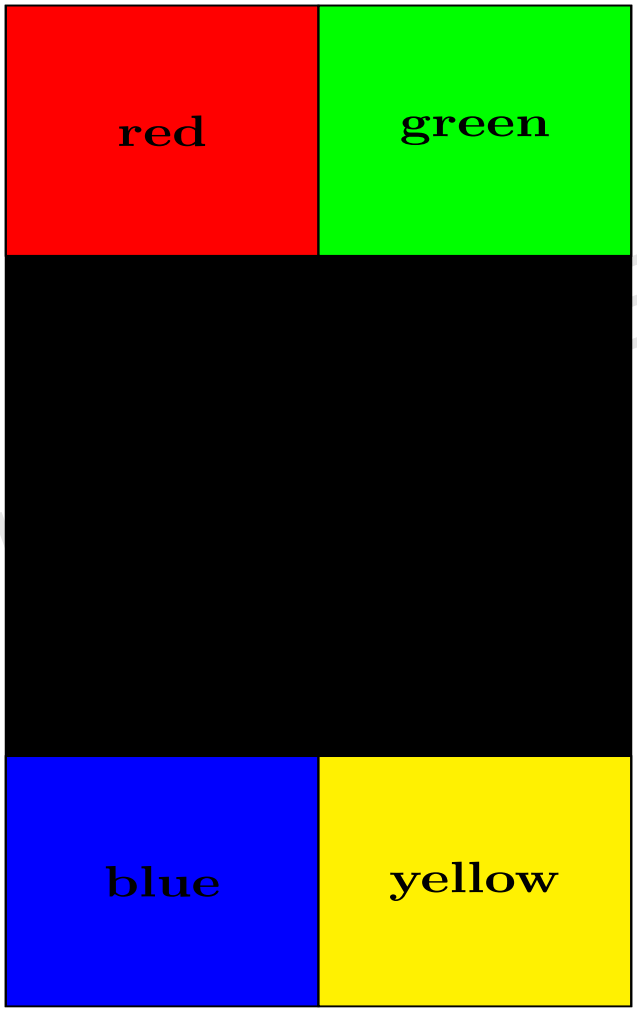
CSS

Exemple avec `align-content: space-around;`



CSS

Exemple avec `align-content: space-between;`



CSS

Propriétés pour les éléments enfants

Les propriétés précédentes sont à mettre dans le style du container. Mais il existe des propriétés qu'on peut préciser dans le style de chaque enfant

CSS

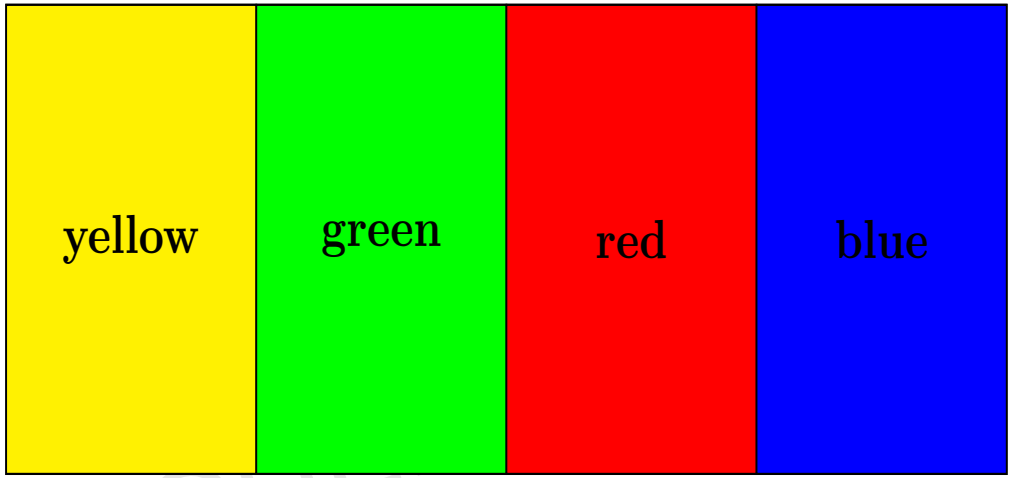
order

- permet de changer l'ordre d'affichage des composants d'un conteneur

Le fichier flex.css

```
.component1 {
  background-color: red;
  order:3;
}
.component2 {
  background-color: green;
  order: 2;
}
.component3 {
  background-color: blue;
  order:4;
}
.component4 {
  background-color: yellow;
  order:1;
}
.container {
  background-color: black;
  display: flex;
}
.container > div {
        width : 30%;
}
```

Le résultat



CSS

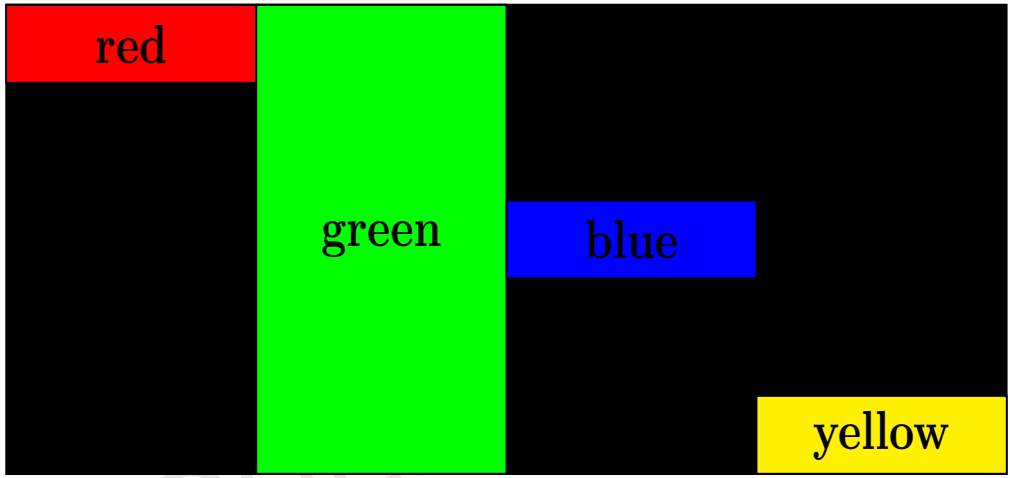
align-self

permet de changer la valeur d'`align-items` pour un block donné.

Le fichier `flex.css`

```
.component1 {
  background-color: red;
  align-self: flex-start;
}
.component2 {
  background-color: green;
  align-self: stretch;
}
.component3 {
  background-color: blue;
  align-self: center;
}
.component4 {
  background-color: yellow;
  align-self: flex-end;
}
.container {
  background-color: black;
  display: flex;
  height: 150px;
}
.container > div {
        width : 30%;
}
```

Le résultat



CSS

Propriétés pour modifier la largeur

- `flex-grow` : permet d'agrandir certains blocks par rapport à la taille des autres.
- `flex-shrink` : permet de rétrécir certains blocks si nécessaire lorsqu'il n'y a pas assez d'espace dans une rangée.
- `flex-basis` : permet de spécifier la largeur initiale d'un composant, avant qu'un éventuel espace disponible soit distribué selon les facteurs flex.
- ...

CSS

flex-basis

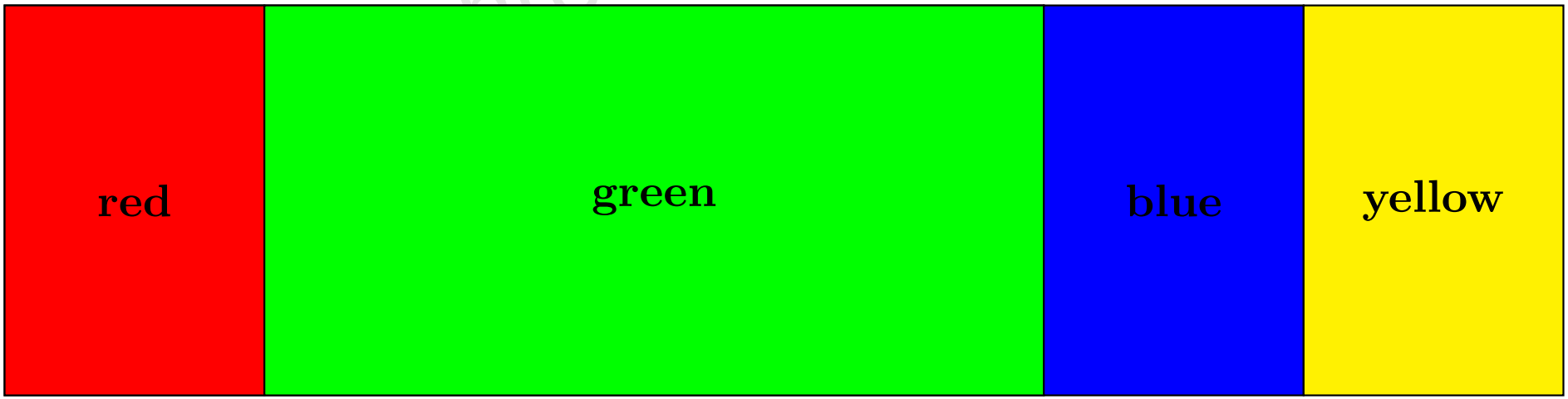
précise la largeur d'un enfant indépendamment de tous les autres.

Exemple avec `flex-basis`

```
.component1 {
    background-color: red;
}
.component2 {
    background-color: green;
    flex-basis: 300px;
}
.component3 {
    background-color: blue;
}
```

```
.component4 {
    background-color: yellow;
}
.container {
    background-color: black;
    display: flex;
}
.container>div {
    width: 100px;
}
```

Le résultat



CSS

flex-grow

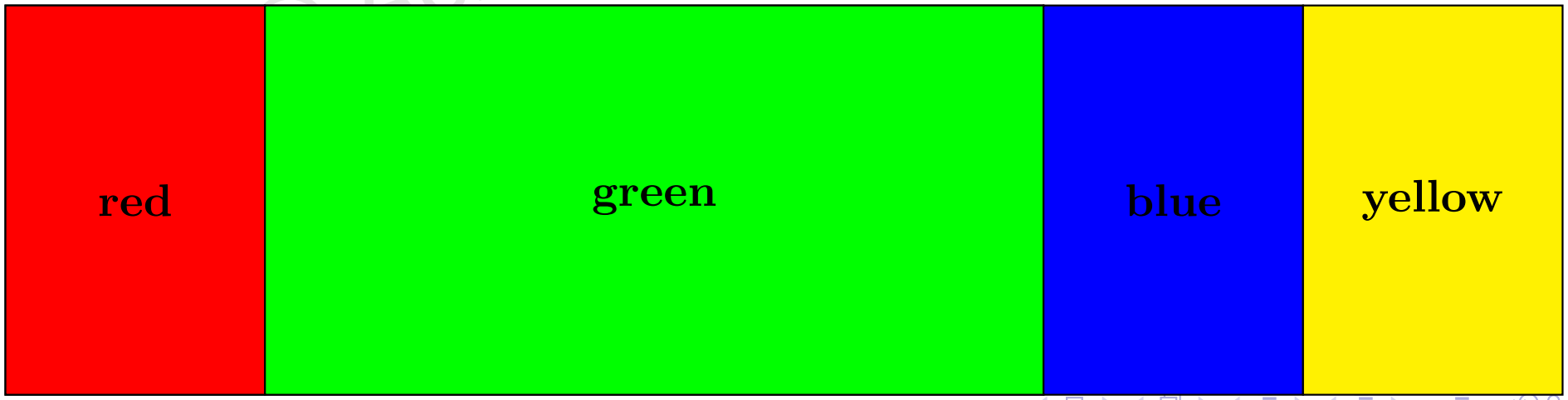
précise de combien croître la taille des enfants.

Exemple avec `flex-grow`

```
.component1 {
  background-color: red;
  flex-grow: 1;
}
.component2 {
  background-color: green;
  flex-grow: 3;
}
.component3 {
  background-color: blue;
  flex-grow: 1;
}
```

```
.component4 {
  background-color: yellow;
  flex-grow: 1;
}
.container {
  background-color: black;
  display: flex;
}
```

Le résultat



CSS

Pour mieux comprendre Flexbox

- http://flexboxfroggy.com/#fr
- http://www.flexboxdefense.com/

CSS

Considérons l'exemple suivant :

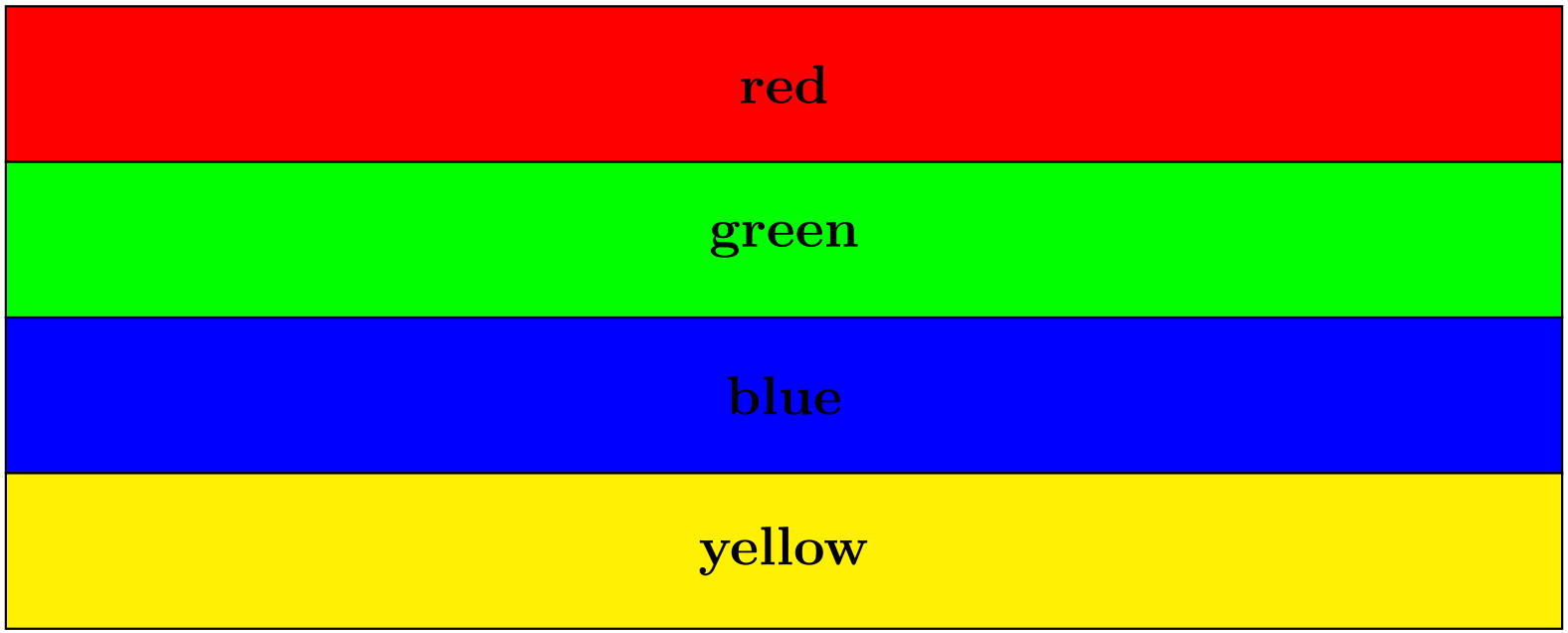
Le fichier `grid.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title> grid </title>
    <link rel="stylesheet" href="grid.css">
  </head>
  <body>
    <div class="container" >
      <div class=component1 >red</div>
      <div class=component2 >green</div>
      <div class=component3 >blue</div>
      <div class=component4 >yellow</div>
    </div>
  </body>
</html>
```

Le fichier `grid.css`

```
.component1 {
  background-color: red;
}
.component2 {
  background-color: green;
}
.component3 {
  background-color: blue;
}
.component4 {
  background-color: yellow;
}
.container {
  background-color: black;
  display: grid;
}
.container > div {
  height: 100px;
}
```

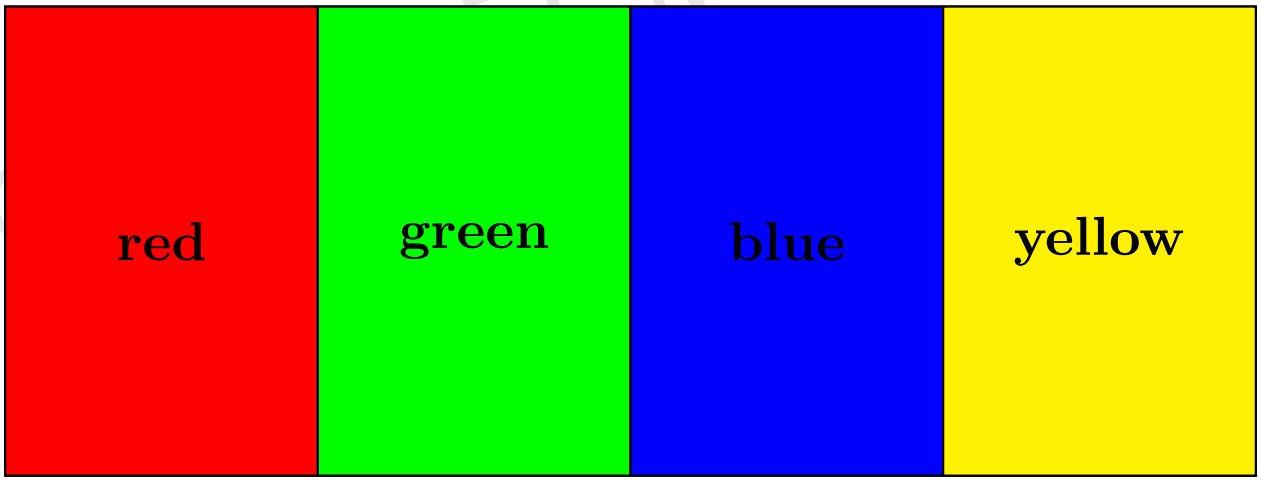
CSS



CSS

Définissons une grille composée de 4 colonnes

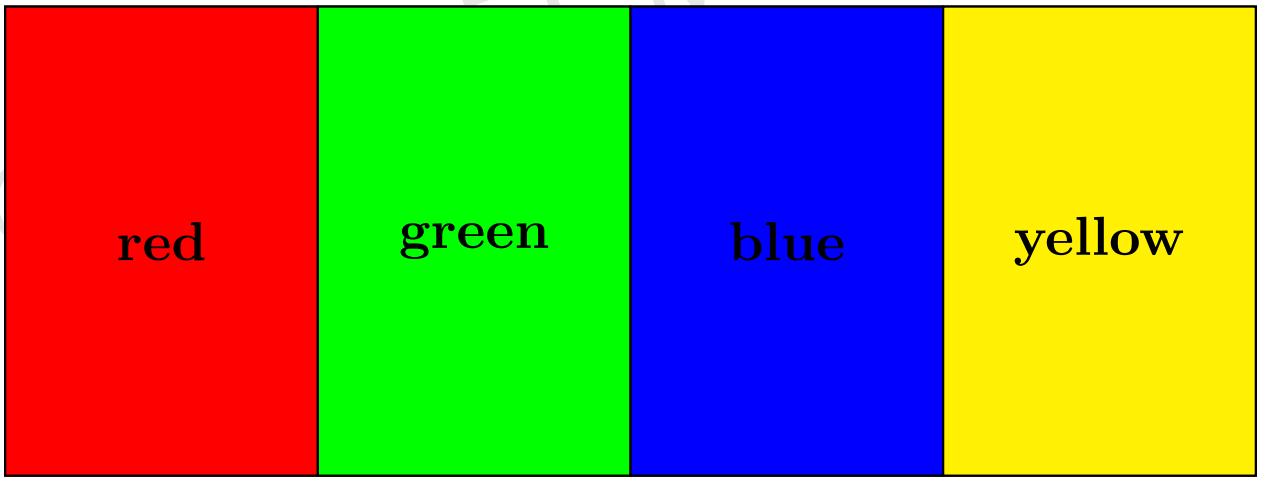
```
.container {
  background-color: black;
  display: grid;
  grid-template-columns: auto auto auto auto;
}
.container > div {
  height: 100px;
}
```



CSS

Définissons une grille composée de 4 colonnes

```
.container {
  background-color: black;
  display: grid;
  grid-template-columns: auto auto auto auto;
}
.container > div {
  height: 100px;
}
```

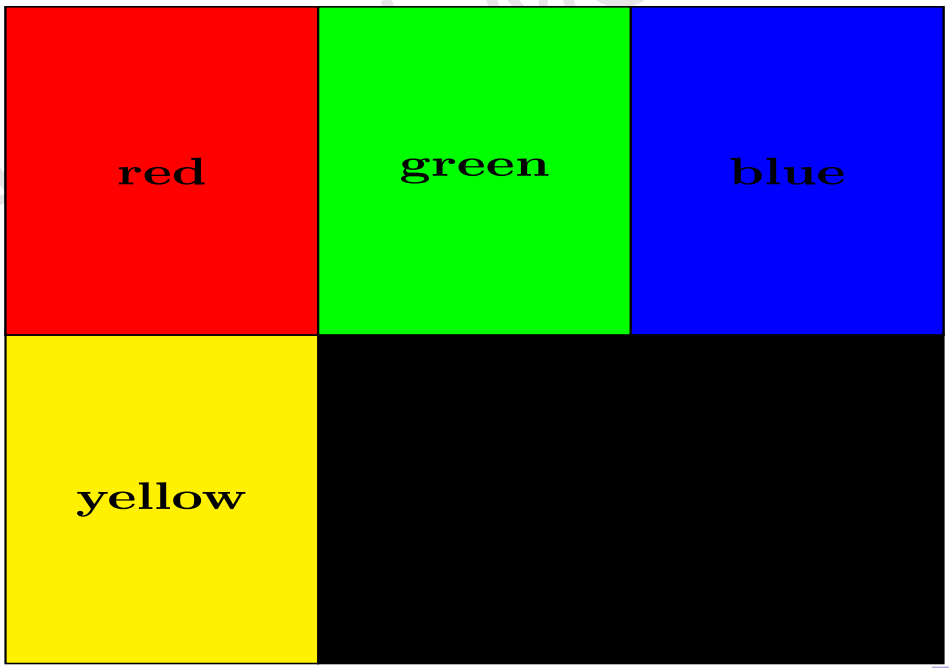


Si le nombre de composants dépasse le nombre de colonnes, alors la grille ajoutera automatiquement une ligne pour placer les composants manquants.

CSS

Exemple avec une grille composée de 3 colonnes

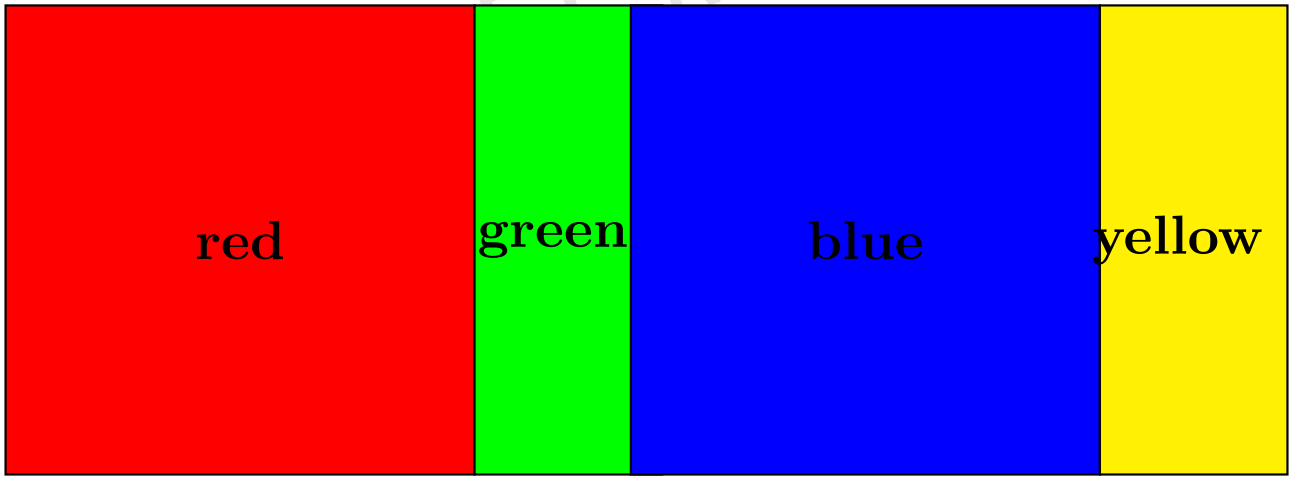
```
.container {
  background-color: black;
  display: grid;
  grid-template-columns: auto auto auto;
}
.container > div{
  height: 100px;
}
```



CSS

On peut aussi utiliser le nombre de fractions : nombre de part de l'espace disponible

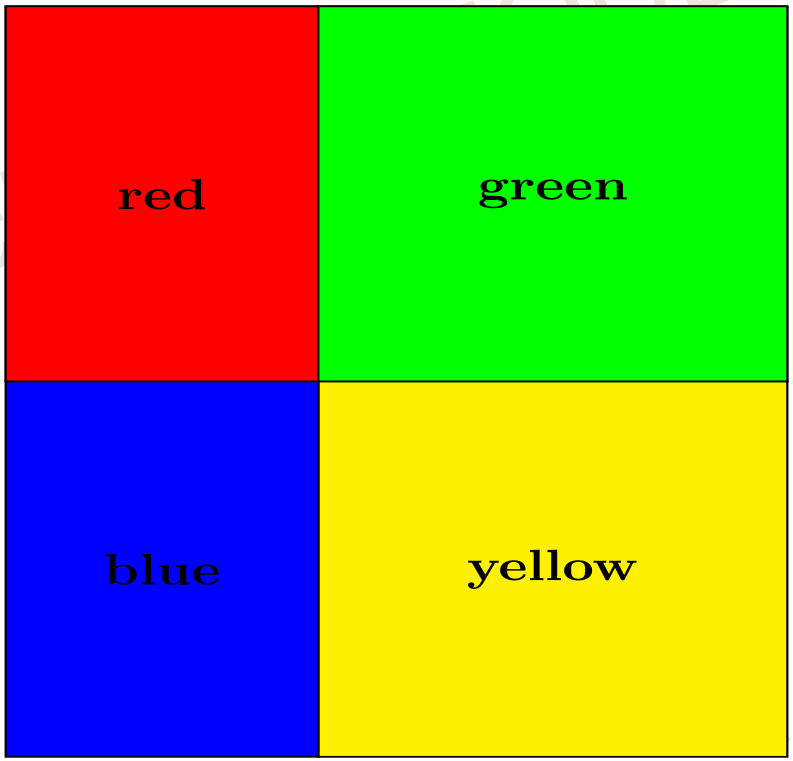
```
.container {
  background-color: black;
  display: grid;
  grid-template-columns: 1.5fr 0.5fr 1.5fr 0.5fr;
}
.container > div{
  height: 100px;
}
```



Cette écriture `grid-template-columns: 1.5fr 0.5fr 1.5fr 0.5fr;` à `grid-template-columns: repeat(2, 1.5fr 0.5fr);`.

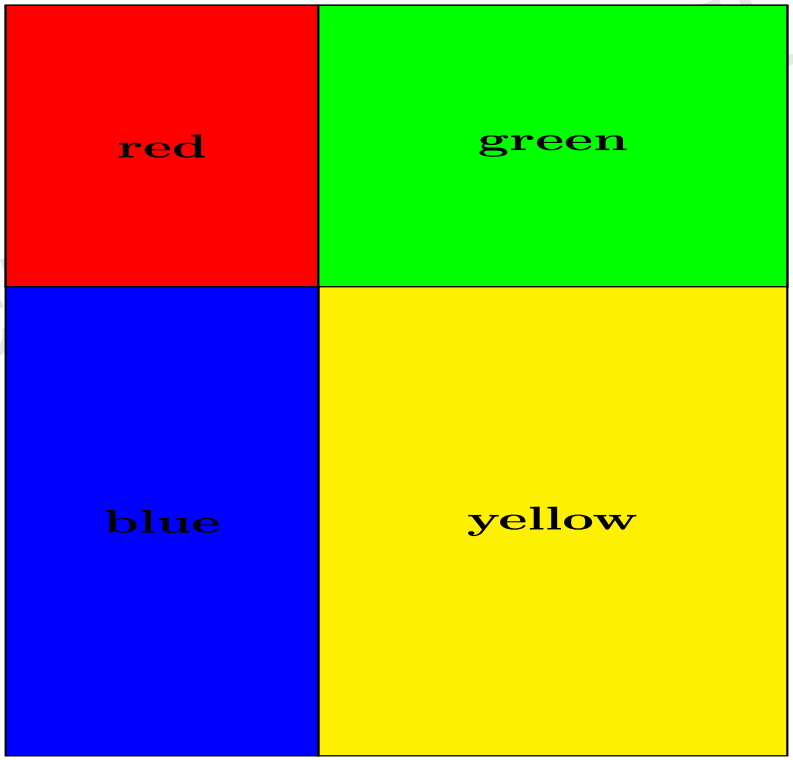
Exemple avec une grille composée de 2 colonnes avec deux largeurs différentes

```
.container {
  background-color: black;
  display: grid;
  grid-template-columns: 40% 60%; // on peut aussi utiliser px : pixel
     ou fr : fraction : nombre de part de l'espace disponible
}
.container > div {
  height: 100px;
}
```



On peut aussi définir la hauteur de chaque rangée (ligne)

```
.container {
  background-color: black;
  display: grid;
  grid-template-columns: 40% 60%;
  grid-template-rows: 100px 200px;
}
```



CSS

On peut aussi remplacer les deux propriétés `grid-template-columns` **et** `grid-template-rows` **par** `grid-template`

```
.container {
  background-color: black;
  display: grid;
  grid-template: 100px 200px / 40% 60%;
  /*
  grid-template-columns: 40% 60%;

  grid-template-rows: 100px 200px;
  */
}
```

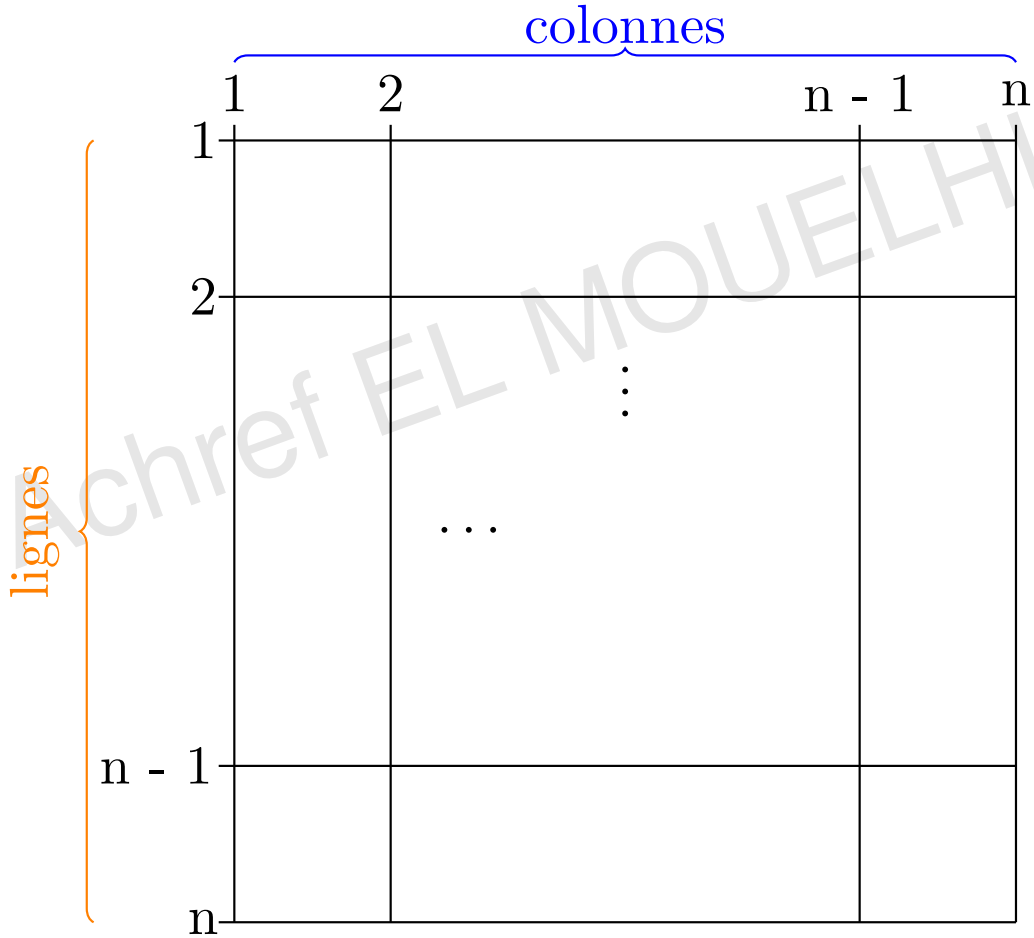
CSS

Grid : autres propriétés

- `grid-column-gap` : permet de définir un espace entre les colonnes
- `grid-row-gap` : permet de définir un espace entre les lignes
- `justify-content` : permet de définir la façon dont l'espace doit être réparti entre et autour des composants par rapport à un axe vertical
- `align-content` : permet de définir la façon dont l'espace doit être réparti entre et autour des composants par rapport à un axe horizontal

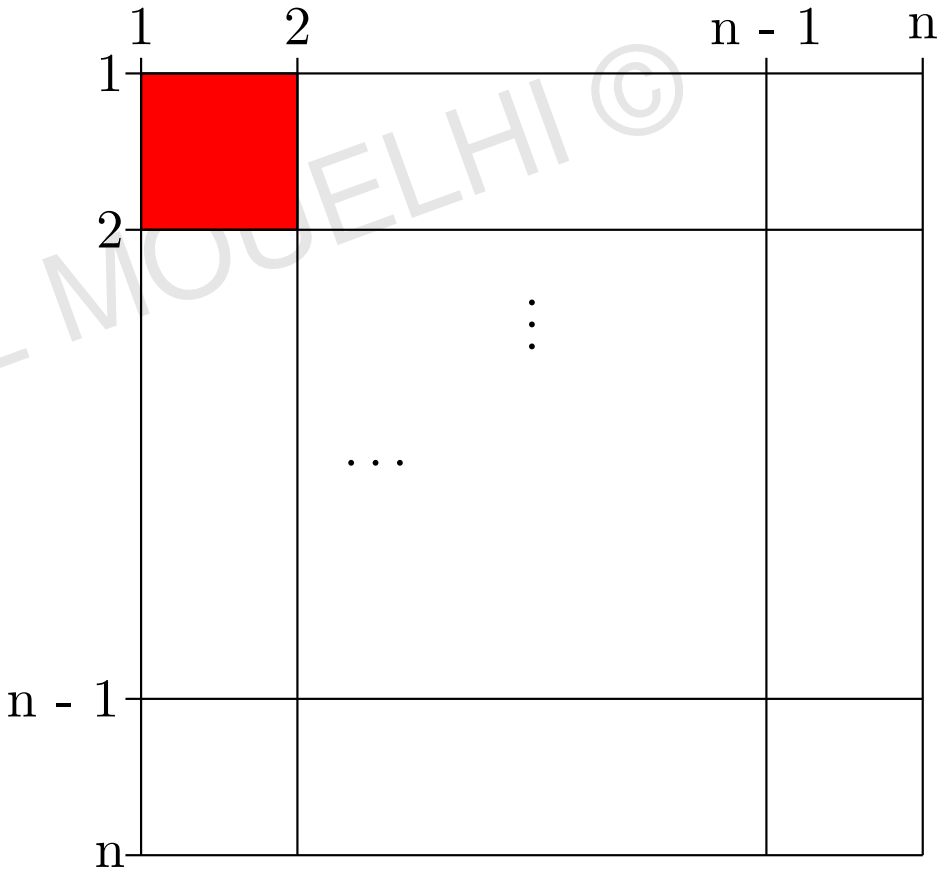
CSS

On peut aussi placer les composants en précisant pour chacun entre quelles colonnes et lignes il faut le placer



Exemple

```
grid-column-start: 1;
grid-column-end: 2;
grid-row-start: 1;
grid-row-end: 2;
```



On peut aussi modifier les propriétés des composants

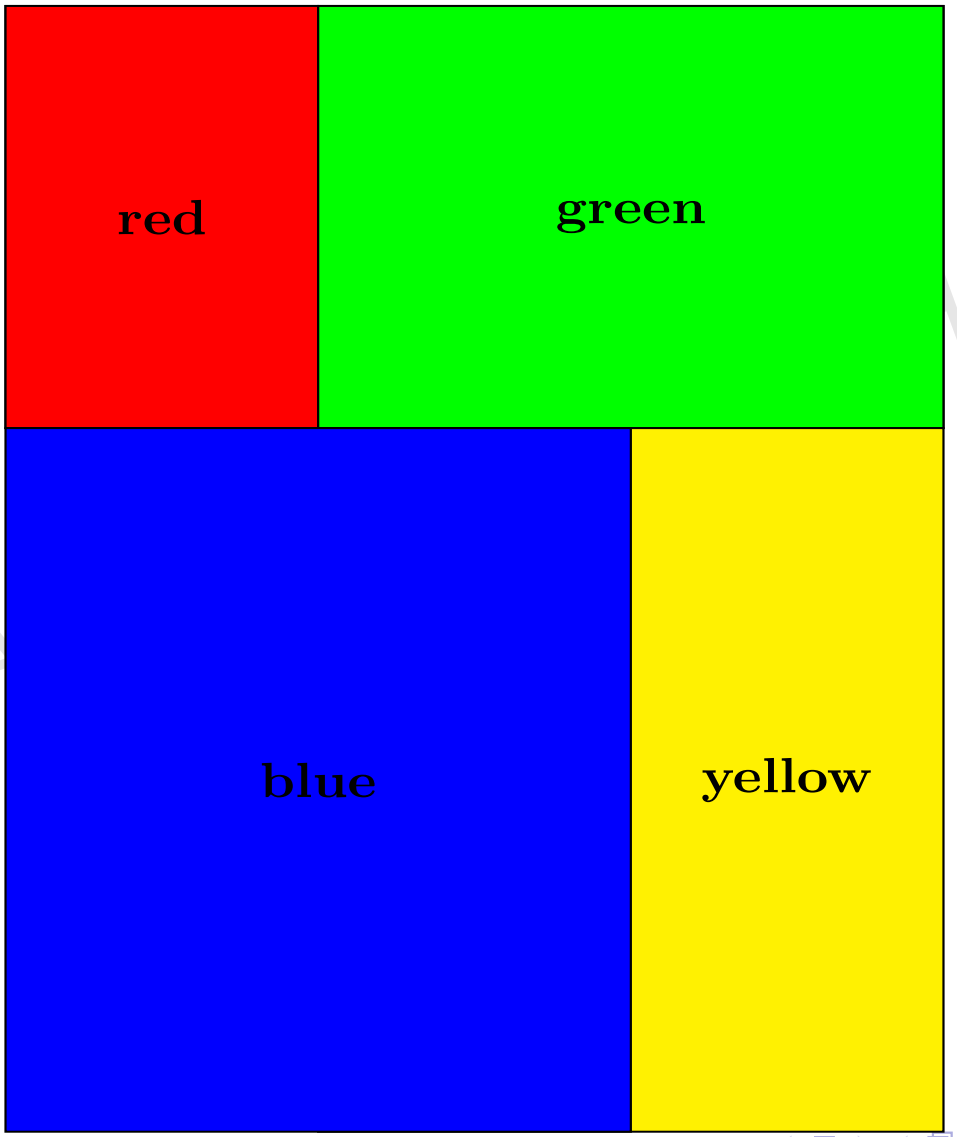
Le fichier `grid.css`

```
.component1 {
  background-color: red;
  grid-column-start: 1;
  grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 3;
}
.component2 {
  background-color: green;
  grid-column-start: 2;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 2;
}
.component3 {
  background-color: blue;
  grid-column-start: 1;
  grid-column-end: 3;
  grid-row-start: 2;
  grid-row-end: 4;
}
```

```
.component4 {
  background-color: yellow;
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row-start: 2;
  grid-row-end: 4;
}
.container {
  background-color: black;
  display : grid;
  height: 500px;
  grid-template-columns: repeat(3, 1fr);
}
```

CSS

Le résultat est :



On peut aussi utiliser la propriété `span` qui indique le nombre de colonnes (lignes)

Le fichier `grid.css`

```
.component1 {
  background-color: red;
  grid-column-start: 1;
  grid-column-end: span 2;
  grid-row-start: 1;
  grid-row-end: span 2;
}
.component2 {
  background-color: green;
  grid-column-start: 2;
  grid-column-end: span 2;
  grid-row-start: 1;
  grid-row-end: 2;
}
.component3 {
  background-color: blue;
  grid-column-start: 1;
  grid-column-end: span 2;
  grid-row-start: 2;
  grid-row-end: span 2;
}
```

```
.component4 {
  background-color: yellow;
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row-start: 2;
  grid-row-end: span 2;
}
.container {
  background-color: black;
  height: 500px;
  display : grid;
  grid-template-columns: repeat(3, 1fr);
}
```

CSS

On peut encore le simplifier

```
.component1 {
  background-color: red;
  grid-column: 1 / span 2;
  grid-row: 1 / span 2;
}
.component2 {
  background-color: green;
  grid-column: 2 / span 2;
  grid-row: 1  / span 1;
}
.component3 {
  background-color: blue;
  grid-column: 1 / span 2;
  grid-row: 2 / span 2;
}
.component4 {
  background-color: yellow;
  grid-column: 3 / span 1;
  grid-row: 2 / span 2;
}
```

CSS

On peut encore le simplifier

```
.component1 {
  background-color: red;
  grid-column: 1 / span 2;
  grid-row: 1 / span 2;
}
.component2 {
  background-color: green;
  grid-column: 2 / span 2;
  grid-row: 1  / span 1;
}
.component3 {
  background-color: blue;
  grid-column: 1 / span 2;
  grid-row: 2 / span 2;
}
.component4 {
  background-color: yellow;
  grid-column: 3 / span 1;
  grid-row: 2 / span 2;
}
```

On peut utiliser la propriété `z-index` pour définir l'ordre dans lequel les composants s'empilent.

On peut aussi utiliser `grid-area: row-start / col-start / row-number / col-number`

```
.container {
  background-color: black;
  height: 500px;
  display : grid;
  grid-template-columns: repeat(3, 1fr);
}
.component1 {
  background-color: red;
  grid-area: 1 / 1 / span 2 / span 2;
}
.component2 {
  background-color: green;
  grid-area: 1 / 2 / span 1 / span 2;
}
.component3 {
  background-color: blue;
  grid-area: 2 / 1 / span 2 / span 2;
}
.component4 {
  background-color: yellow;
  grid-area: 2 / 3 / span 2 / span 1;
}
```

On peut aussi définir un grid-template-areas

```
.component1 {
  background-color: red;
  grid-area: rouge;
}
.component2 {
  background-color: green;
  grid-area: vert;
}
.component3 {
  background-color: blue;
  grid-area: bleu;
}
.component4 {
  background-color: yellow;
  grid-area: jaune;
}
.container {
  height: 500px;
  display : grid;
  grid-template-areas:
  'rouge vert vert'
  'bleu bleu jaune'
  'bleu bleu jaune';
}
```

CSS

Pour mieux comprendre **Grid**

- http://cssgridgarden.com/#fr

CSS

Problématique

- Une application (ou un site) Web est composée de plusieurs éléments **HTML**.
- Plusieurs éléments **HTML** peuvent avoir les mêmes valeurs pour certaines propriétés **CSS**(par exemple, couleur du texte, couleur de bouton...).
- Cette couleur sera utilisée plusieurs fois dans le fichier **CSS**.
- Si on décide un jour de modifier cette valeur, il faudra remplacer toutes ces occurrences.

CSS

Problématique

- Une application (ou un site) Web est composée de plusieurs éléments **HTML**.
- Plusieurs éléments **HTML** peuvent avoir les mêmes valeurs pour certaines propriétés **CSS**(par exemple, couleur du texte, couleur de bouton...).
- Cette couleur sera utilisée plusieurs fois dans le fichier **CSS**.
- Si on décide un jour de modifier cette valeur, il faudra remplacer toutes ces occurrences.

Solution

Utiliser les variables **CSS**.

CSS

Variables **CSS**

- Appelées aussi les propriétés personnalisées (Custom Properties en anglais)
- Permettant d'éviter la répétition de valeurs
- Permettant d'avoir une feuille de style dynamique
- Facilitant ainsi l'évolution de l'application

CSS

Exemple sans les variables CSS

```
body {
  background-color: skyblue;
}

h2 {
  border: 2px solid skyblue;
}

.container {
  color: skyblue;
  background-color: red;
  padding: 15px;
}

button {
  background-color: red;
  color: skyblue;
  border: 1px solid skyblue;
  padding: 5px;
}
```

Exemple avec les variables CSS

```
:root {
  --blue: #6495ed;
  --red: #ff0000;
}

body {
  background-color: var(--blue);
}

h2 {
  border-bottom: 2px solid var(--blue);
}

.container {
  color: var(--blue);
  background-color: var(--red);
  padding: 15px;
}

button {
  background-color: var(--red);
  color: var(--blue);
  border: 1px solid var(--blue);
  padding: 5px;
}
```

CSS

Remarques

- `:root` permet de déclarer des variables globales exploitables dans tout le fichier **CSS**.
- Le nom d'une variable **CSS** doit commencer par deux tirets.

CSS

Le code HTML

```
<div>
    <p>bonjour</p>
</div>
```

Le code CSS

```
:root {
    --bg-color: skyblue;
}
div {
    color: red
}
p {
    color: var(--bg-color);
}
```


Le paragraphe `bonjour` sera affiché en bleu.

CSS

Le code HTML

```
<div>
    <p>bonjour</p>
</div>
```

Le code CSS

```
:root {
   --bg-color: skyblue;
}
div {
   color: red
}
p {
   color: var(--bg-colour);
}
```


Si la variable **CSS** n'existe pas, le paragraphe récupère la couleur de son parent (rouge).

CSS

Le code HTML

```
<div>
    <p>bonjour</p>
</div>
```

Le code CSS

```
:root {
   --bg-color: skyblue;
}
div {
   color: red
}
p {
   color: var(--bg-colour, blue);
}
```


La fonction `var()` accepte un deuxième paramètre correspondant à la valeur par défaut si la variable **CSS** n'existe pas.

CSS

Les Media Queries

définit les techniques CSS d'adaptation du contenu de notre page web en fonction de la résolution d'écran de nos visiteurs

CSS

Les Media Queries

définit les techniques CSS d'adaptation du contenu de notre page web en fonction de la résolution d'écran de nos visiteurs

```
<head>
    <meta charset="utf-8">
    <title>Affichage selon resolution</title>
    <link rel="stylesheet" media="screen" href="main.css">
    <link rel="stylesheet" media="print" href="print.css">
</head>
```

CSS

Les Media Queries

définit les techniques CSS d'adaptation du contenu de notre page web en fonction de la résolution d'écran de nos visiteurs

```
<head>
    <meta charset="utf-8">
    <title>Affichage selon resolution</title>
    <link rel="stylesheet" media="screen" href="main.css">
    <link rel="stylesheet" media="print" href="print.css">
</head>
```

Depuis CSS2

On avait la possibilité de définir une version :

- `screen` : pour les écrans
- `print` : pour impression

CSS

Dans le fichier print.css

```
#menu, #footer, #header {
    display:none;
}
body {
    font-size:80%;
}
```

CSS

Tout mettre dans un fichier ?

Il faut juste préciser chaque fois le type de périphérique concerné par le style.


```
@media print {
    #menu, #footer, #header {
        display:none;
    }
    body {
        font-size:80%;
    }
}

@media screen {
    ...
}
```

CSS

Depuis CSS2, autres types de média disponibles

- `handheld` : Périphériques mobiles ou de petite taille
- `projection` : Projecteurs (ou présentations avec slides)
- `tv` : Téléviseur
- `all` : Toutes les résolutions
- ...

Plusieurs navigateurs mobiles ignorent le media `handheld` (par exemple Safari Mobile) et se considèrent comme un média screen.

CSS

Depuis CSS3, on peut définir des règles

- `height` : hauteur de la zone d'affichage
- `width` : largeur de la zone d'affichage
- `device-height` : hauteur du périphérique
- `resolution` résolution du périphérique
- `max-height` : hauteur maximale de la zone d'affichage
- `max-width` : largeur maximale de la zone d'affichage
- `min-height` : hauteur minimale de la zone d'affichage
- `min-width` : largeur minimale de la zone d'affichage
- `orientation` : orientation du périphérique (portrait ou paysage)
- ...

CSS

```
<link rel="stylesheet" media="(orientation:portrait)
   " href="file1.css">

<link rel="stylesheet" media="(orientation:landscape
   )" href="file2.css">
...
```

CSS

Depuis CSS3, il est possible de combiner ces règles

- `and` : et
- `only` : seulement
- `not` : non

CSS

```
<link rel="stylesheet" media="screen and (max-width: 640px)" href="
   file1.css">

<link rel="stylesheet" media="only screen and print" href="file2.css">
...
```

CSS

```
<link rel="stylesheet" media="screen and (max-width: 640px)" href="
   file1.css">

<link rel="stylesheet" media="only screen and print" href="file2.css">
...
```

ou dans un seul fichier CSS :

```
@media screen and (max-width: 1024px) {

}
@media all and (min-width: 1024px) and (max-width: 1280px) {

}
@media projection and tv {

}
@media all and (orientation: portrait) {

}
```

CSS

Liste de valeurs par défaut pour les balises **HTML**

`https://www.w3schools.com/cssref/css_default_values.asp`

CSS

Pour tester la compatibilité d'une propriété **CSS 3** avec les navigateurs

`https://caniuse.com/`

CSS

Pour mieux apprendre **HTML**, **CSS**, **JavaScript**...

- `https://htmlcheatsheet.com/`
- `https://css-tricks.com/snippets/`

CSS

Quelques templates gratuits

- https://www.gettemplate.com/
- https://www.free-css.com/