


5 : DISPOSITION ÉLÉMENTS

I. Types de disposition des éléments (display)

Explications :

Une des choses essentielles en CSS c'est la disposition des éléments ou bien le mode d'affichage de ces éléments sur la page. Pour ça on va pouvoir utiliser la propriété **display** qui va nous permettre de changer ce mode d'affichage ou la façon dont les éléments ou les blocs seront gérés sur la page et on va voir que cette propriété peut prendre pas mal de valeurs.


La première chose sur laquelle j'aimerais revenir c'est la différence entre **div** et **span**. Une div va prendre le plus de place possible sur la page c'est pour ça qu'on ne peut pas mettre deux div l'une à côté de l'autre alors qu'une balise span va prendre le moins de place possible.



Une div...
Un span...

This visual representation shows two elements. The first element, 'Une div...', is a green rectangle that takes up the full width of the container. The second element, 'Un span...', is a light blue rectangle that also takes up the full width, appearing directly below the first one. This illustrates the default 'display: block' behavior where elements stack vertically and take up as much space as possible.

Si on les mets à la suite dans le code :



Une div...
Une autre div...
Un span... Un autre span...

This visual representation shows four elements. The first two are green rectangles ('Une div...' and 'Une autre div...') stacked vertically, each taking up the full width. The next two are light blue rectangles ('Un span...' and 'Un autre span...') stacked vertically, each taking up the full width. This illustrates the default 'display: block' behavior for all elements.

On peut donc dire que le display de ces éléments, c'est à dire la façon dont ils sont affichés, est différent.


Sur une div par défaut on retrouve un **display block**, qui s'affiche comme un block en prenant le plus de place possible, alors que dans le cas d'un élément span, cela va s'afficher sous forme **inline** c'est-à-dire dans la ligne, et prendre le moins de place possible.

Si je vous présente cela de cette façon, vous vous doutez bien que nous allons pouvoir modifier le type d'affichage de ces éléments et pour ça on va pouvoir utiliser la propriété **display**.

style.css

```
div{  
  background: green;  
  display: inline; //Va se comporter comme un span  
}  
span {  
  background: lightblue;  
  display: block; //Va se comporter comme une div  
}
```

Résultat :



Une div... Une autre div...
Un span...
Un autre span...

This visual representation shows the result of applying the CSS rules. The first two elements are green rectangles ('Une div...' and 'Une autre div...') stacked vertically, each taking up the full width. The next two are light blue rectangles ('Un span...' and 'Un autre span...') stacked vertically, each taking up the full width. This illustrates the result of applying the CSS rules where the display property is swapped for div and span elements.

L'exemple précédent est à faire en théorie car cela revient à changer le comportement par défaut de nos balises. Si vous avez besoin des caractéristiques d'une div, prenez une div et vice versa pour le span.

Il existe par contre une propriété qui est entre **inline** et **block** et qui se nomme **inline-block**.

Dans l'absolu, si on voulait définir ce qu'est **inline-block**, ce serait un élément dont l'extérieur (la partie externe) réagirait comme un display inline, c'est-à-dire qu'on pourrait en afficher plusieurs sur la même ligne, mais, réagirait quand même à l'intérieur comme un block.

style.css

```
div{
  background: green;
  display: inline-block;
}
span {
  background: lightblue;
}
```

Résultat (en ajoutant des `
` dans index.html) pour ne pas avoir span et div côte à côte :



Le span a repris son display de base en inline et la div est en inline-block. La seule différence que vous pouvez voir ici c'est l'espace qu'il y a entre les deux blocs de div, c'est simplement une petite marge qui est liée à l'utilisation d'inline-block.

Le dernier display que je vais vous présenter sera le **display:none;** qui permet de masquer un élément. Dans l'exemple, nous le mettrons dans une balise que nous nommerons «caché».

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href='css/style.css'>
  </head>
  <body>
    <div class="cache"> ←
      Une div...
    </div>
    <div>
      Une autre div...
    </div>
    <br>
    <br>
    <span>
      Un span...
    </span>
    <span>
      Un autre span...
    </span>
  </body>
</html>
```

index.html

style.css

```
div{
  background: green;
  display: inline-block;
}
span {
  background: lightblue;
}

.cache {
  display:none;
}
```

Résultat :

Une autre div...

Un span... Un autre span...

On voit bien que la première div où il y avait dedans le texte «une div...» est maintenant cachée. Il existe bien d'autres display en CSS mais ils ont des usages bien spécifiques, les 4 présentés ci-dessus sont les plus utilisés et sont à connaître pour faire du CSS. Il existe un autre type de display qui est de plus en plus utilisé et que nous allons voir à la suite, le display flex.

II. Le display flex

Explications :

Le **display flex** est un moyen de gérer la façon dont va être affiché une div de manière un peu plus simplifiée. Au lieu d'avoir à utiliser des **inline-block** par ci par là, on va pouvoir utiliser ce **flex**.

Pour l'expliquer, nous allons créer une div "mère" ou nous mettrons des div "filles" avec chacune un background-color :

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Accueil</title>
    <link rel="stylesheet" type="text/css" href='css/style.css'>
  </head>
  <body>
    <div class="flexible">
      <div style="background: lightgreen">1</div>
      <div style="background: pink">2</div>
      <div style="background: lightblue">3</div>
      <div style="background: lightgrey">4</div>
      <div style="background: yellow">5</div>
    </div>
  </body>
</html>
```

index.html

Résultat :



Nous allons maintenant ajouter un display flex à notre class flexible dans style.css.

style.css

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

Résultat :



On voit que **nos différentes div qui étaient à la base des block** qui prennent le plus de largeur possibles **ont été converties et sont maintenant toutes petites**. Leur display a été modifié en un display propre au display flex. La petite chose où il faut faire attention c'est que le display flex n'a pas été mis sur chaque div mais uniquement sur la div mère qui contient toutes les autres div. Il existe d'autres propriétés liées au display flex que nous ne pouvons pas utiliser sans display flex et que nous allons essayer. Nous allons commencer avec le **flex-direction** :

style.css

```
.flexible{  
    display: flex;  
    flex-direction: row;  
}
```

Résultat :



Cela n'a rien changé car la direction est de base sur row, c'est-à-dire en ligne. Nous allons maintenant essayer avec column :

```
flex-direction: column;
```

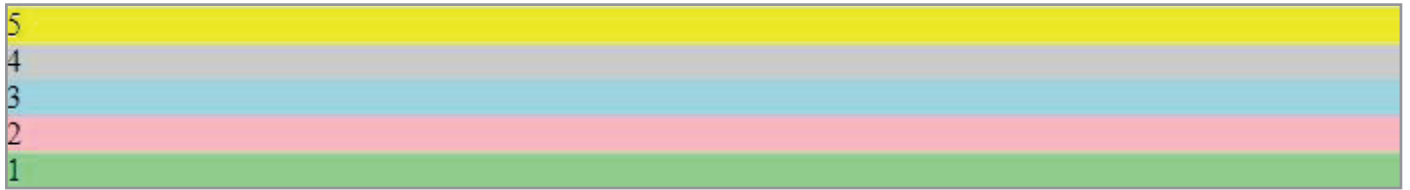
Résultat :



On retrouve quelque chose de similaire à notre `display: block;`, en colonne donc.

```
flex-direction: column-reverse;
```

Résultat :



On voit que ça renverse l'ordre des éléments. On peut également le faire avec `row` en mettant `row-reverse` :



Mis à part l'ordre des éléments et leurs affichages en ligne ou en colonne on va également pouvoir choisir le **justify-content**, c'est-à-dire la façon dont les éléments sont disposés entre eux.

Par défaut on aura **justify-content: flex-start** pour que les éléments soient alignés au début de notre balise flex.

style.css

```
.flexible{
  display: flex;
  flex-direction: row;
  justify-content: flex-start;
}
```

Résultat :



On retrouve notre affichage initial car comme dit ci-dessus le `justify-content` qui n'est pas initialisé est par défaut en `flex-start`. Essayons maintenant en écrivant `flex-end` dans `justify-content` :



Les éléments se retrouvent de l'autre côté de la page sans être inversés car pas de `row-reverse`. Si on l'utilise avec un `row-reverse` :

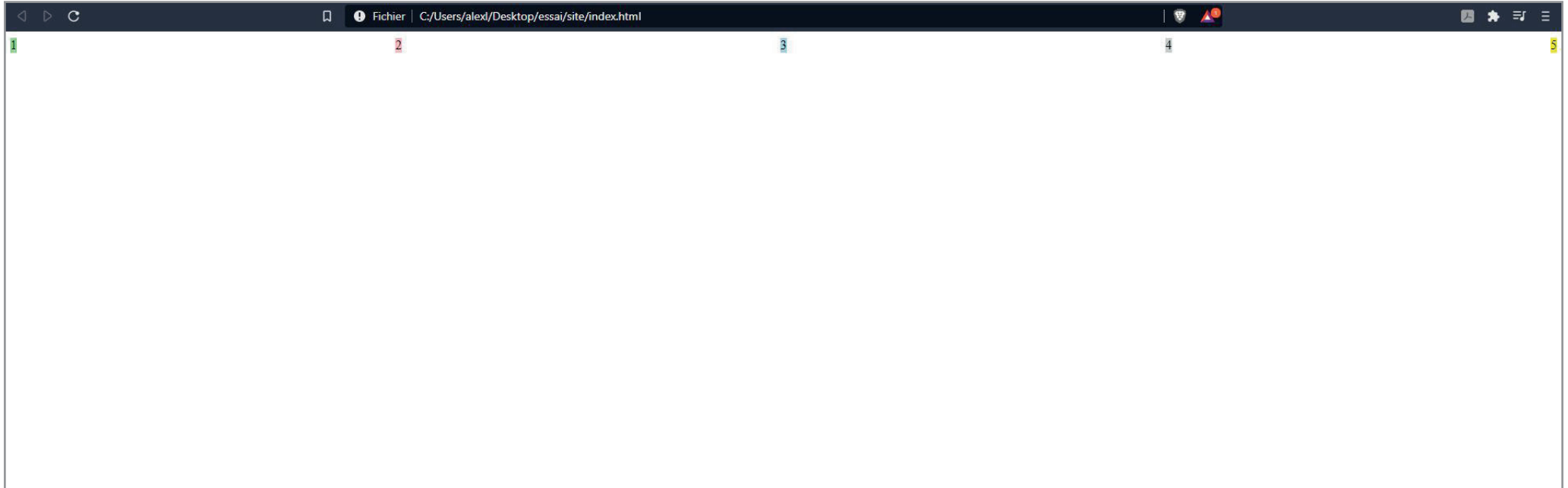


Les éléments se retrouvent de l'autre côté de la page en étant inversés.

On va retourner à notre exemple de base (`style.css` ci-dessus) en mettant `justify-content: center`; à la place de `justify-content: flex-start`;



On peut encore choisir deux autres formes de disposition des éléments, l'une d'entre elles s'appelle *space-between* :



On peut voir que le même espace a été fait entre chaque éléments. Essayons maintenant avec *space-around* :



Avec *space-around*, c'est l'espace autour de nos différents items qui va être le même.

La principale différence entre *space-between* et *space-around*, c'est que dans *space-between*, le premier élément va se coller au flex-start et le dernier élément au flex-end comme sur la capture d'écran.

Avec un *space-around*, on aura un écart au flex-start et au flex-end comme sur la capture d'écran.

On va maintenant parler d'une nouvelle propriété qui est la propriété **align-items**. Cette propriété va nous permettre de gérer nos items sur le plan vertical.

La valeur par défaut d'align-items est stretch (align-items:stretch;) ce qui veut dire que tous nos éléments vont avoir la même hauteur (comme dans la série d'exemples précédents donc).

Pour voir comment stretch marche, nous allons prendre un exemple où nous allons rajouter des valeurs à notre `<div>2</div>` avec des `
` (des sauts de ligne) :

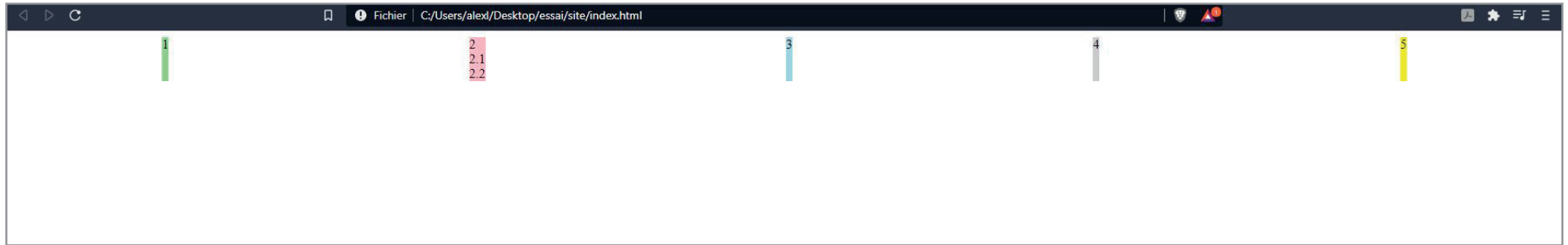
```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Accueil</title>
    <link rel="stylesheet" type="text/css" href='css/style.css'>
  </head>
  <body>
    <div class="flexible">
      <div style="background: lightgreen">1</div>
      <div style="background: pink">
        2<br>
        2.1<br>
        2.2<br>
      </div>
      <div style="background: lightblue">3</div>
      <div style="background: lightgrey">4</div>
      <div style="background: yellow">5</div>
    </div>
  </body>
</html>
```

index.html

style.css

```
.flexible{
  display: flex;
  flex-direction: row;
  justify-content: space-around;
  align-items: stretch; //Optionnel et pour l'exemple car valeur de
                          //base de align-items déjà sur stretch
}
```

Résultat :



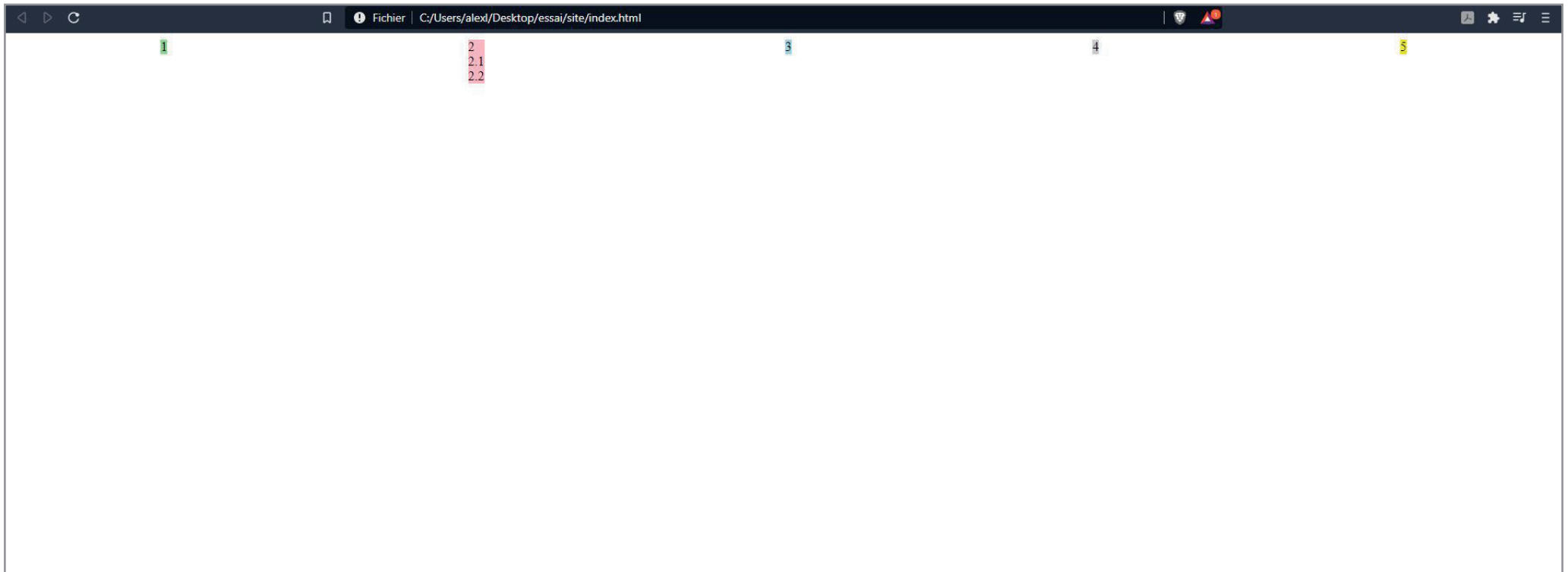
On peut voir que la div avec le contenu «2»,«2.1»,«2.2» a été agrandie au niveau de la hauteur, mais toutes les autres div qui sont autour ont également été alignés automatiquement sur la hauteur.

En gros tous les items d'un même container (à savoir notre div mère) et le align-items en stretch vont prendre la hauteur de l'élément le plus haut.

Si on essaie avec align-items : flex-end; ce sera le même principe que nous avons pu utiliser sur notre justify-content mais sur le plan vertical :



Si on essaie en mettant align-items: flex-start; ce sera le même principe que l'exemple précédent mais en partant du début :



Si on essaie en mettant align-items: center; :



Jusqu'à présent on a ajouté des propriétés uniquement sur le container (sur le .flexible qu'on a créé) mais sachez qu'on peut également les ajouter sur les items un par un, au cas par cas. On va faire un exemple avec flex-grow, qui définit le facteur de largeur de notre bloc :

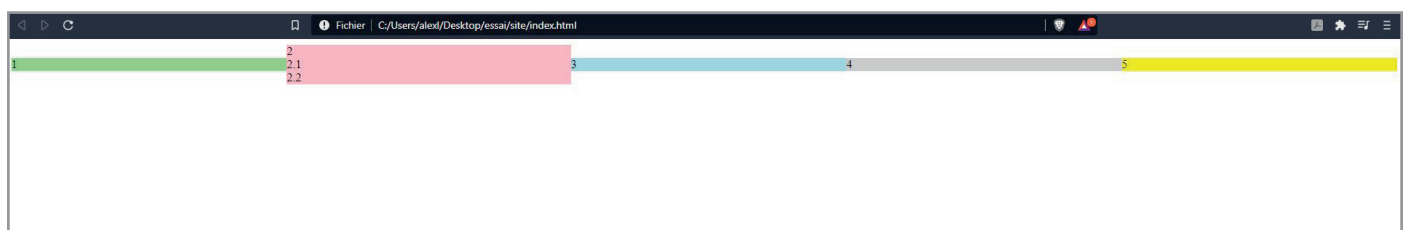
```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Accueil</title>
    <link rel="stylesheet" type="text/css" href='css/style.css'>
  </head>
  <body>
    <div class="flexible">
      <div style="background: lightgreen;flex-grow: 1">1</div>
      <div style="background: pink;flex-grow: 1">
        2<br>
        2.1<br>
        2.2<br>
      </div>
      <div style="background: lightblue;flex-grow: 1">3</div>
      <div style="background: lightgrey;flex-grow: 1">4</div>
      <div style="background: yellow;flex-grow: 1">5</div>
    </div>
  </body>
</html>
```

index.html

```
style.css

.flexible{
  display: flex;
  flex-direction: row;
  justify-content: space-around;
  align-items: center;
}
```

Résultat :



On voit donc bien que chacun à pris une fois la même largeur en prenant le plus de place possible.

Dans cette partie, je vous ai donné un gros aperçu du display flex, de ce à quoi il pouvait nous servir, mais sachez qu'il existe encore quelques autres propriétés qu'on va pouvoir utiliser avec display flex.