Web : principes de base (HTML/CSS)

DA1

Henallux

Module 6
Layout et CSS avancé

Au programme

- **Le flux naturel des éléments HTML**
  - *Modes d'affichage standards (inline et block)*
  - *Le box model*
- **Rompre avec flux naturel**
  - *Modifier le mode d'affichage d'un élément*
  - *Éléments flottants*
- **Intermède : la spécificité en CSS**
- **Techniques de layout avancées**
  - *Positionnement*
  - *Flexboxes et grid layout*

Flux naturel des éléments HTML

Au programme de ce chapitre…

- **Le flux naturel en HTML**
  - *Agencement standard des éléments HTML*
- **Modes d'affichage standards**
  - *Inline, block (et inline-block)*
- **Le box model**
  - *Rembourrage, bordures et marges autour des éléments*

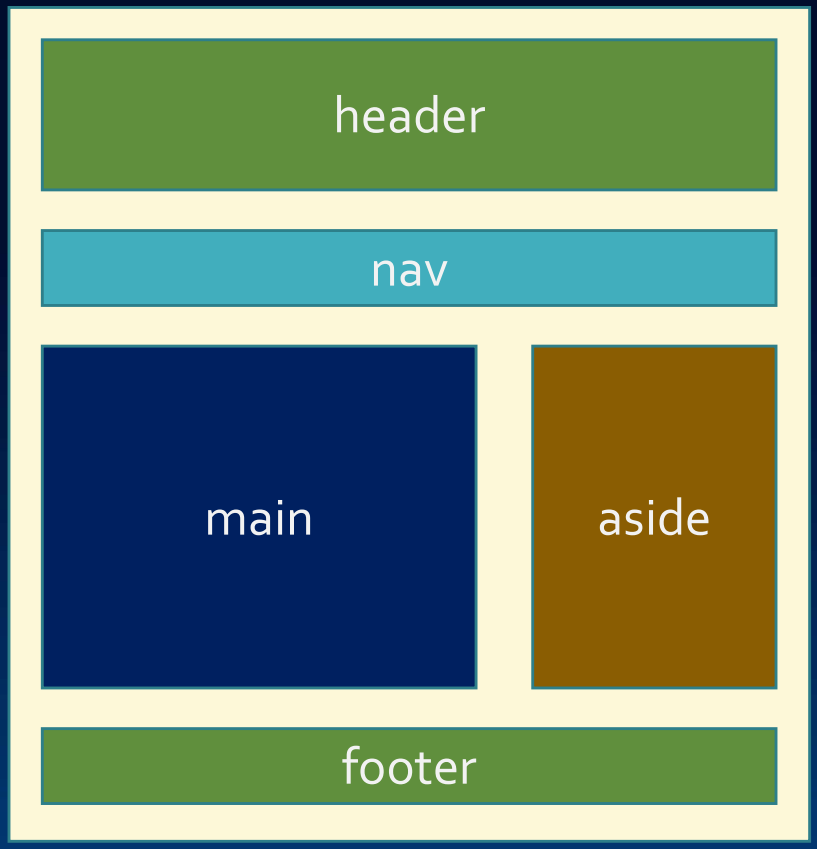
Ensuite : *Rompre avec le flux naturel*

Le flux naturel en HTML

- **Layout** = agencement des éléments d'une page

```
<header> … </header>
<nav> … </nav>
<main> … </main>
<aside> … </aside>
<footer> … </footer>
```

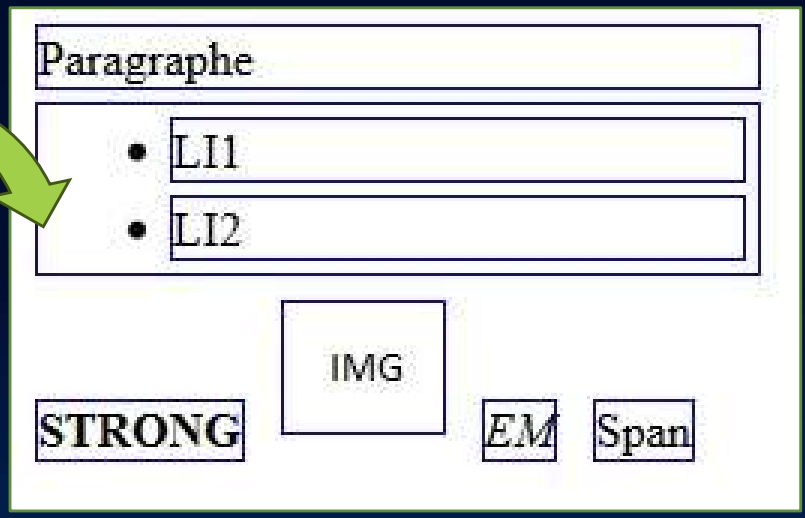
Agencement désiré



Le flux naturel en HTML

- **Flux naturel** = agencement par défaut

```
<p>Paragraphe</p>
<ul>
  <li>LI1</li>
  <li>LI2</li>
</ul>
<strong>STRONG</strong>
<img src="img.jpg" />
<em>EM</em>
<span>Span</span>
```



- Observations :
  - Chaque élément occupe un espace rectangulaire.
  - Certains forcent un passage à la ligne ; d'autres pas.

Modes d'affichage standards

- Les deux principaux **modes d'affichage** :

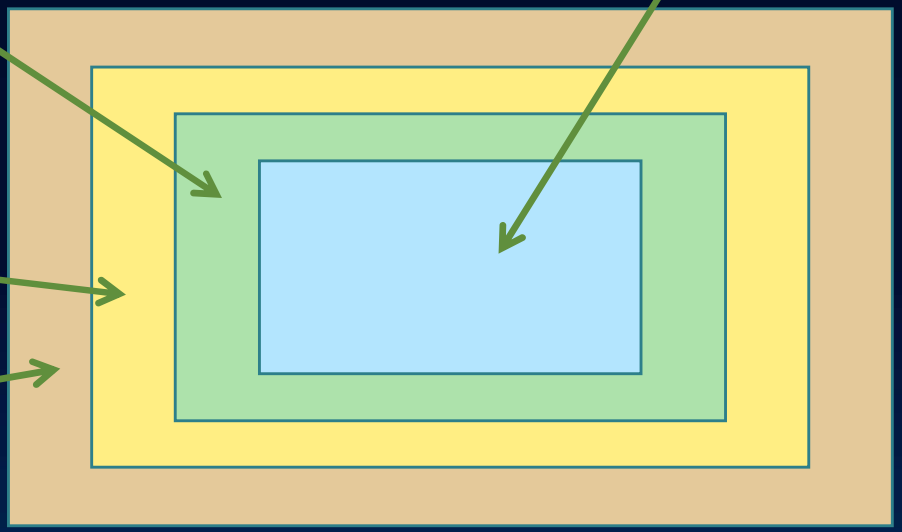
Éléments block	Éléments inline
Exemples : , <p>, <header>	*Exemples : , , <img/>*
empilés les uns sur les autres	affichés les uns à la suite des autres
précédés d'un passage à la ligne	retour à la ligne si plus assez de place
occupe un espace rectangulaire	
largeur = toute la place disponible	largeur = selon le contenu

- Remarques :
  - Il existe d'autres modes d'affichage plus ou moins complexes.
  - Quel que soit le mode d'affichage d'un élément HTML, celui-ci occupe un espace rectangulaire décrit par le box model !

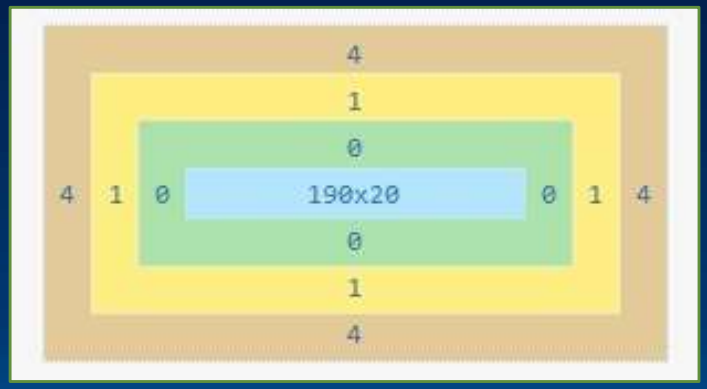
Le box model

= modèle en « oignon » à 3 couches autour du contenu :

- Rembourrage/padding
  Utilise la couleur de fond (`background-color`)
- Bordure/border
- Marge/margin
  transparent



Vue dans l'inspecteur Firefox précisant les dimensions :



Le box model

Fixer les dimensions en CSS

Contenu	Rembourrage	Bordure	Marge
`width` `height`	`padding`	`border-width` `border`	`margin`

- Une seule valeur `border-width: 2px`
- Dans une direction `margin-top: 8px`
  suffixes : -top, -bottom, -left et -right
- Valeurs différentes `padding: 1px 2px 3px 4px`
  toujours dans l'ordre : top, right, bottom, left
- Deux valeurs `margin: 8px 4px`
  1re valeur pour top/bottom, 2e valeur pour left/right

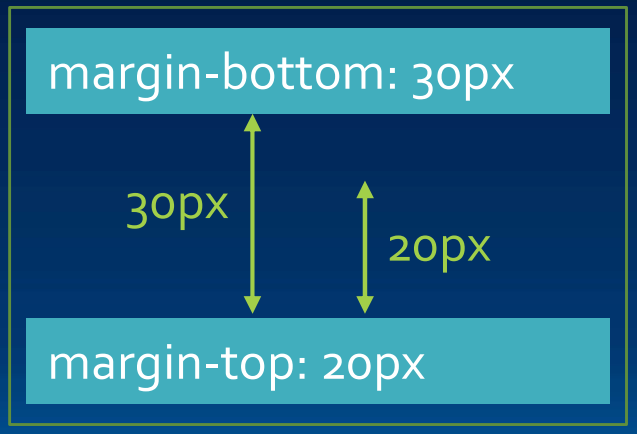
Le box model

Dimensions et modes d'affichage

- Sur les éléments block dont la largeur est fixée :
  - Coller à droite : `margin-left: auto`
  - Centrer : `margin-left: auto; margin-right: auto`
- Les éléments **inline** ignorent les propriétés

  `width, height, padding-top/bottom, margin-top/bottom`

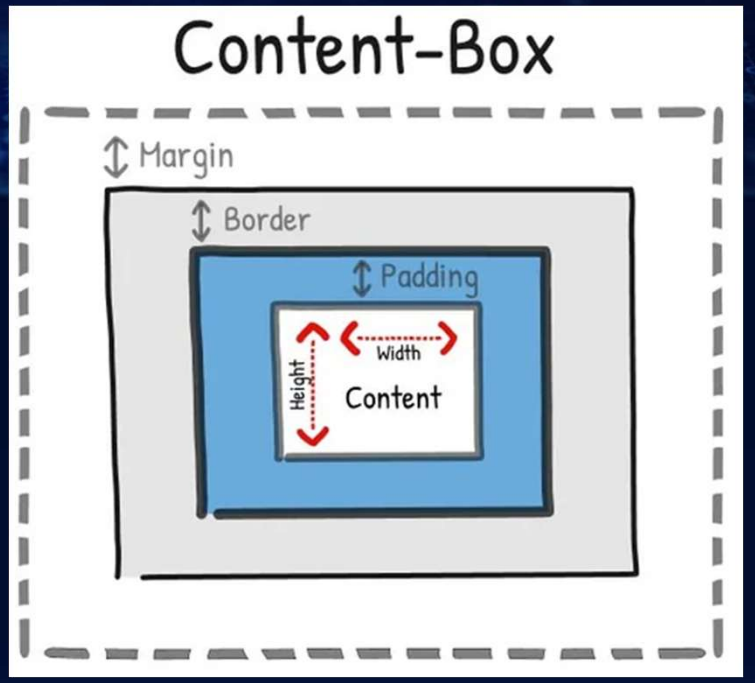
  Pour les utiliser : mode **inline-block**
- Les marges des éléments qui se suivent se chevauchent. Seule la plus grande est prise en compte.

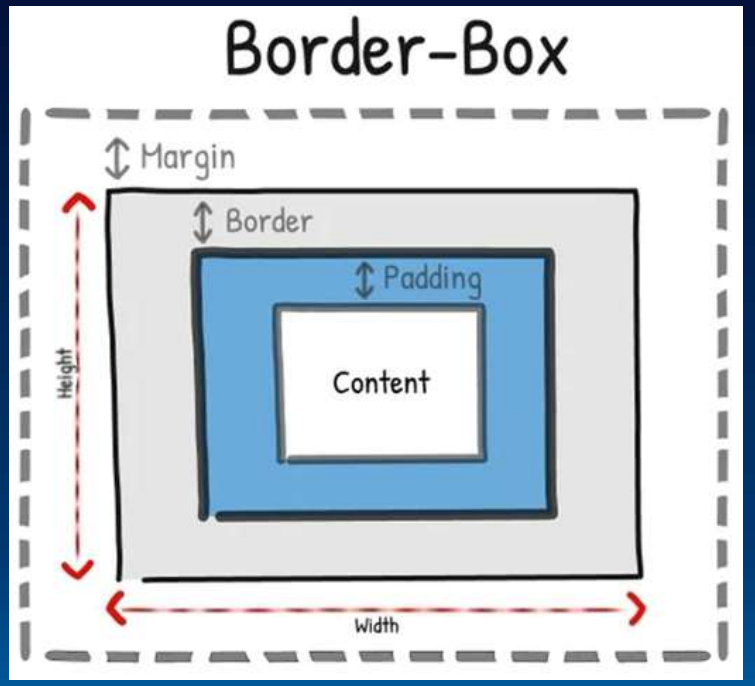


Le box model

Box-sizing

- Par défaut, `width` et `height` se rapportent à la taille du contenu.
- La propriété `box-sizing` modifie ce que ces propriétés mesurent :
  - `box-sizing: content-box` (contenu)
  - `box-sizing: padding-box` (contenu + padding)
  - `box-sizing: border-box` (contenu + padding + border)





Rompre avec le flux naturel

Au programme de ce chapitre…

- **Modifier le mode d'affichage**
  - *Via CSS*
- **Éléments flottants**
  - *Image autour de laquelle s'étend un texte par exemple*

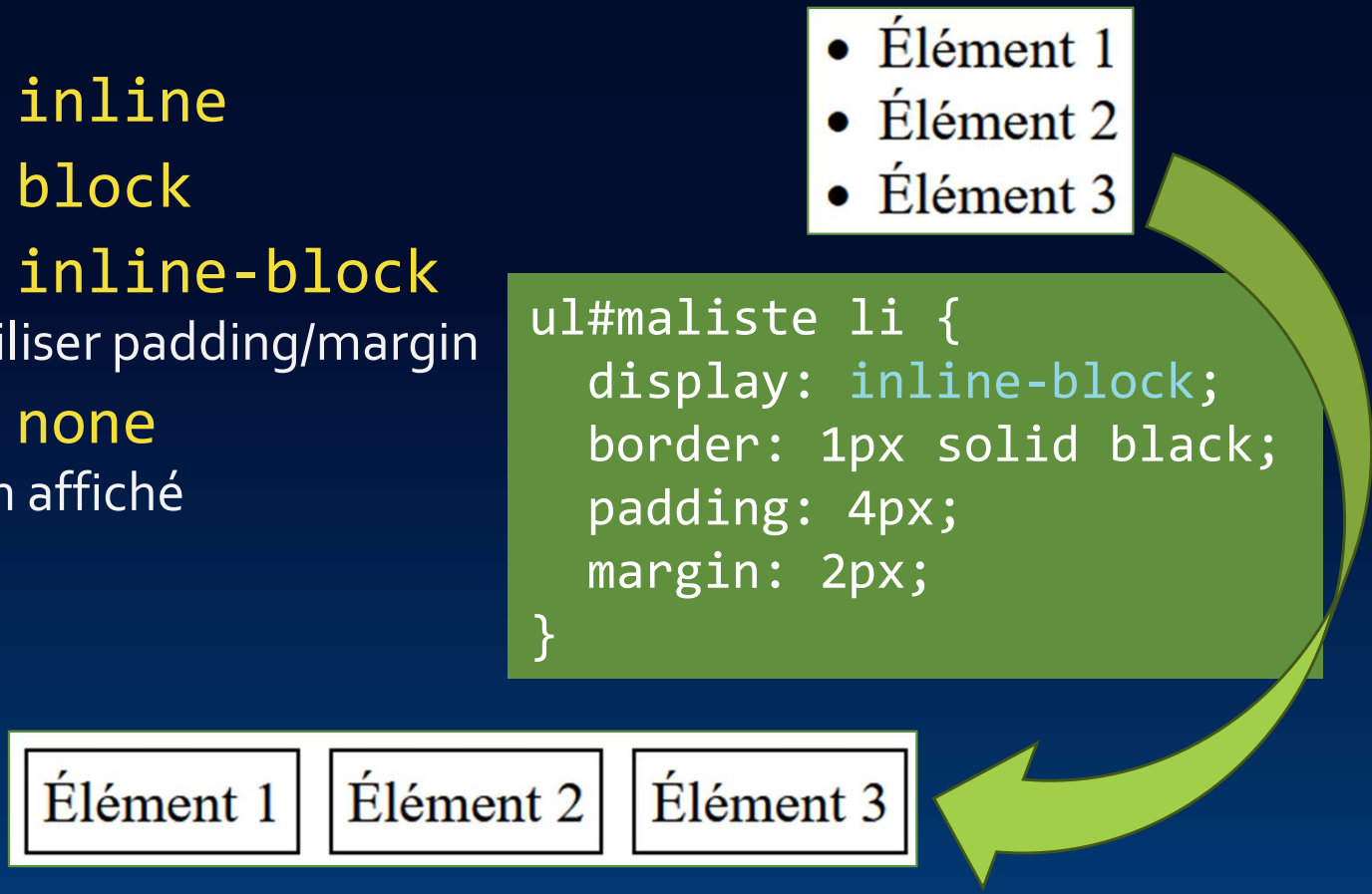
D'autres techniques plus avancées pour rompre avec le flux naturel seront abordées plus tard

Ensuite : *Spécificité en CSS*

Modifier le mode d'affichage

La propriété **`display`** modifie le mode d'affichage :

- `display: inline`
- `display: block`
- `display: inline-block`
  Permet d'utiliser padding/margin
- `display: none`
  Élément non affiché



Élément flottant

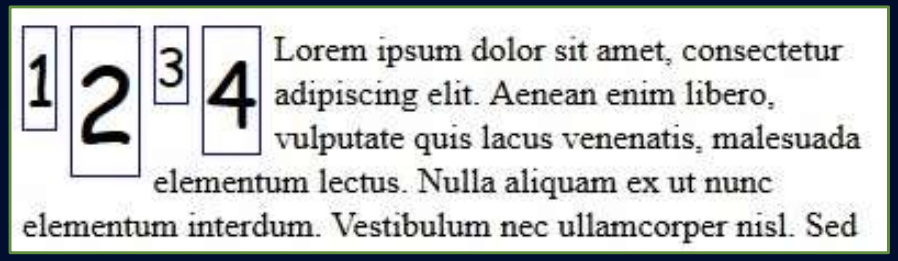
La propriété **`float`** permet d'extraire un élément du flux naturel. L'élément est considéré en mode block et placé contre le bord droit ou gauche de son parent.

- `float: left`
- `float: right`
- `float: none`
  Élément non flotté

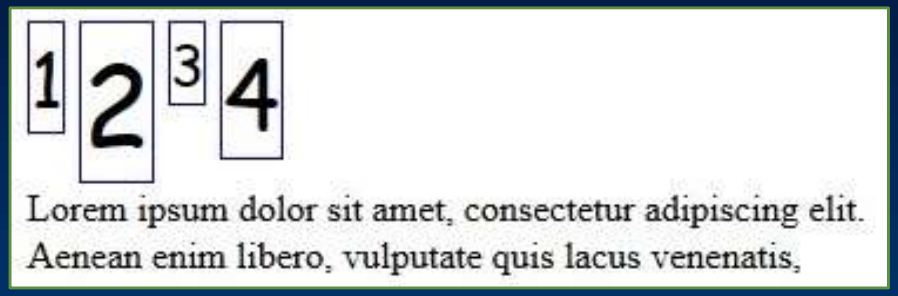
```
<img src="schtroumpf.jpg"
style="float:right;" />
<p>Lorem ipsum dolor sit
amet, consectetur
adipiscing elit. …</p>
```

Élément flottant

- Si on fait flotter plusieurs éléments du même côté, ils se placent côte à côte.
  - *Exemple : 4 blocs en float: left*



- Si on veut placer un élément en-dessous des flottants, on peut utiliser la propriété CSS **clear**.
  - `clear: left`
    sous les flottants à gauche
  - `clear: right`
    sous les flottants à droite
  - `clear: both`
    sous tous les flottants



Spécificité en CSS

Au programme de ce chapitre…

- **… pour résoudre les conflits entre règles CSS**
  - *Quand plusieurs règles s'appliquent, laquelle utiliser ?*

Ensuite : *Techniques de layout avancées*

Spécificité en CSS

Pourquoi se poser la question ?

```
<p id="intro">
    Ce <em>paragraphe</em> contient plusieurs
    <em>mots-clefs</em>, certains sont
    <em id="important">importants</em> et
    d'autres <em>moins</em>.
</p>
```

```
#intro em { background-color: yellow }
```

Ce *paragraphe* contient plusieurs *mots-clefs*, certains sont *importants* et d'autres *moins*.

Et si on ajoute la règle suivante ?

```
#important {background-color: blue}
```

Spécificité en CSS

- La règle CSS dont le **sélecteur** a la plus grande **spécificité** prend le pas.
- La **spécificité** d'un sélecteur se forme à partir de quatre valeurs/chiffres :

Inline	Identificateur	Classes	Balises
1 si style inline 0 sinon	nombre d'identificateurs	nombre de classes, attributs, pseudo-classes	nombre de balises

- Exemple : `#form5 input[type="radio"] + label.nombre`

0	1	2	2

Spécificité en CSS

Retour à l'exemple introductif :

```
#intro em { background-color: yellow }
#important {background-color: blue}
```

Inline	ID	Classes	Balises
0	1	0	1
0	1	0	0

```
#intro #important {background-col…}
```

0	2	0	0

Remarques

- Un style inline prend le pas sur toutes les règles.
- Les pseudo-classes fonctionnelles `:is`, `:not` et `:has` n'ajoutent pas d'elles-mêmes la spécificité mais leur paramètre est pris en compte.

```
div:not(.exemple) p
```

0	0	1	2

- La pseudo-classe `:where` fonctionne exactement comme `:is` mais son paramètre n'est pas pris en compte pour la spécificité.

Techniques de layout avancées

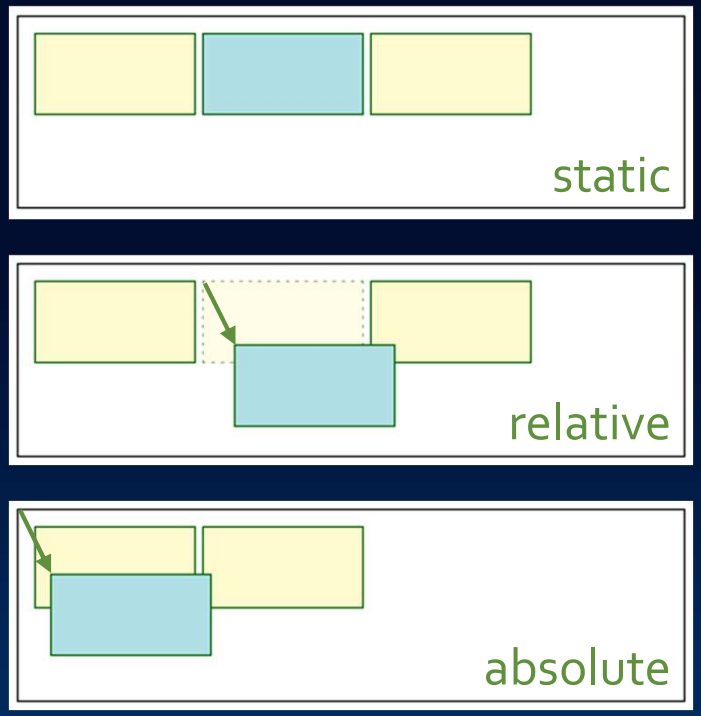
Au programme de ce chapitre…

- **Positionnement en CSS**
  - *Positionner de manière précise un élément*
- **Flexboxes**
  - *Pour une mise en page automatisée et réactive*
- **Grid layout**
  - *(pas couvert dans le cadre du cours)*
  - *https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout*

Positionnement en CSS

La propriété **position** permet de gérer le positionnement précis.

- `position: static`
  positionnement standard
- `position: relative`
  déplacement par rapport à la position standard
- `position: absolute`
  élément sorti du flux HTML et
  position par rapport à un ancêtre
- `position: fixed`
  élément sorti du flux HTML et
  position par rapport à la fenêtre d'affichage
- `position: sticky`
  positionnement semi-fixe (pas traité ici)



Pour les déplacements/positions : propriétés `top`, `left`, `right` et `bottom`.

Positionnement en CSS

Exemples de **positionnement relatif** (`position: relative`)

```
<p>Voici un <em>test</em> de position.</p>
```

- Position normale

- `bottom: 20px`
  éloignement du bas de 20px

- `left: 20px; top: 20px`
  éloignement de 20px de la gauche / du bas

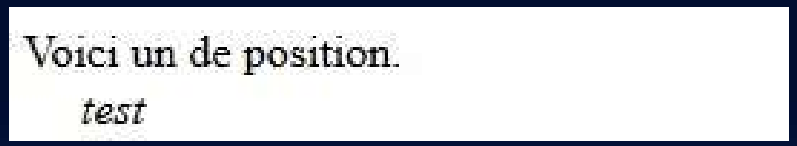
- `bottom: 10px; right: -20px`
  équivalent à `bottom: 10px; left: 20px`

Positionnement en CSS

Exemples de **positionnement absolu** (`position: absolute`)

```
<p>Voici un <em>test</em> de position.</p>
```

Référence = le plus proche ancêtre positionné (ici <p>, en position: relative)



- `left: 20px; top: 20px`
  à 20px des bords gauche/haut de <p>

- Sans précision de position
  position standard (mais retiré du flux)

- `bottom: 20px`
  à 20px du bas de <p>

- `bottom: 10px; right: 20px`

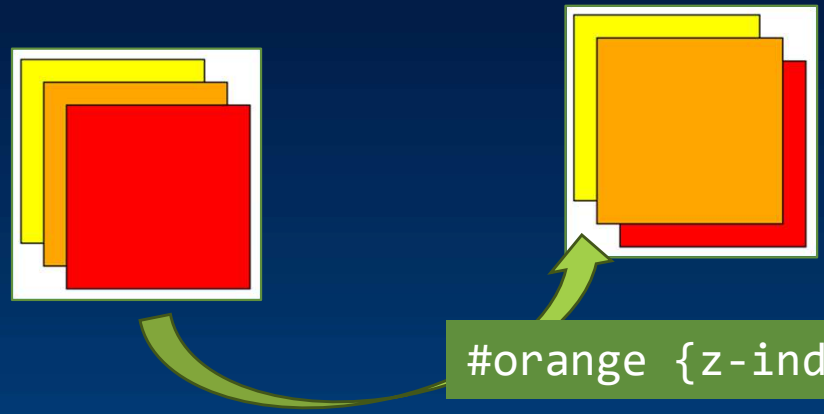
Positionnement en CSS

La propriété **z-index** permet de préciser la « hauteur » des couches/calques des éléments HTML.

- Valeur grande = élément au premier plan
  Valeur 0 = calque standard
  Valeur négative = sous le calque standard

- Par défaut, les calques sont empilés dans l'ordre de leur déclaration.

```
<div id="jaune"></div>
<div id="orange"></div>
<div id="rouge"></div>
```



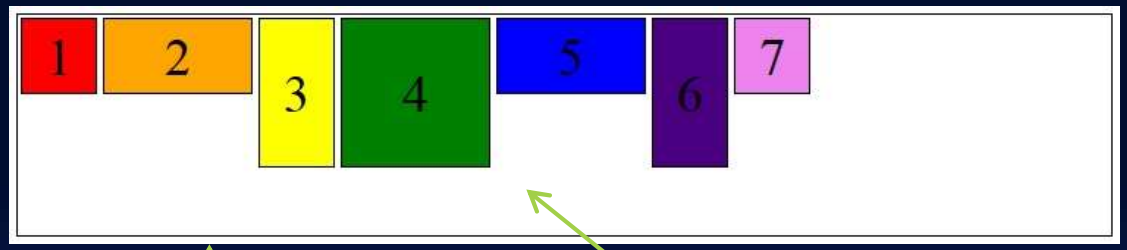
```
#orange {z-index: 5}
```

Mise en place d'une flexbox

- Flexbox = rendre un conteneur (= **flexbox**) responsable de la mise en page de ses fils (= **flex items**).

- Exemple :

```
<div class="cadre">
  <div>…1…</div>
  <div>…2…</div>
  …
</div>
```



Flexbox

7 flex items
- affichés de gauche à droite
- alignés sur le haut
- justifiés vers la gauche

Pour le transformer en « flexbox » :

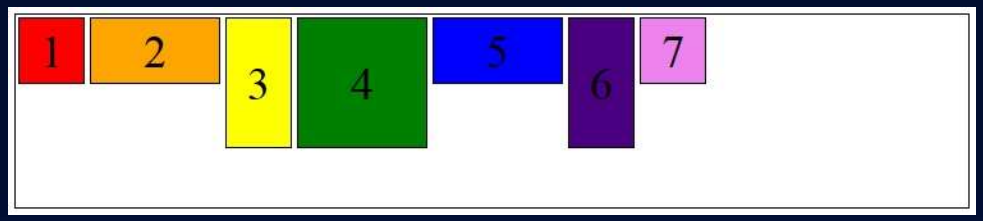
```
.cadre { display : flex }
```

Uniquement sur le père

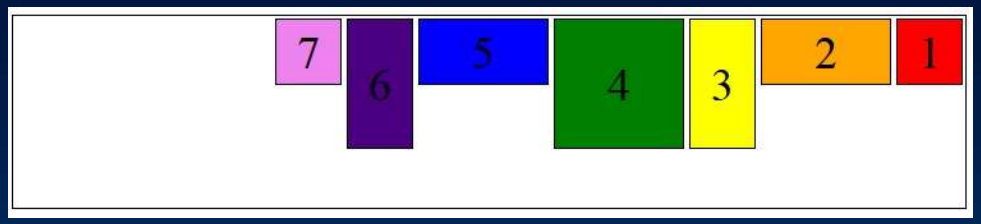
Propriété flex-direction

Quatre modes (`flex-direction`) d'affichage :

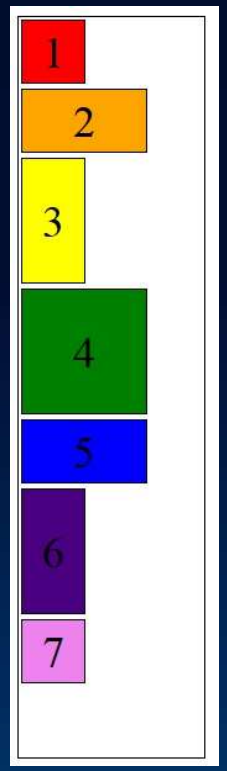
- **`row`** (défaut) : ligne, de gauche à droite

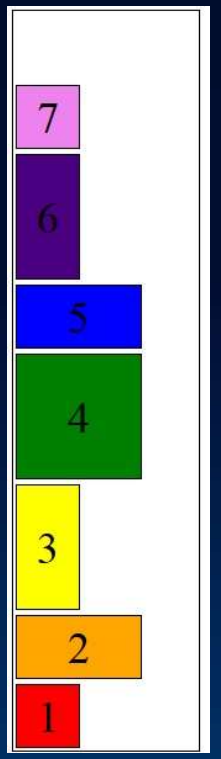


- **`row-reverse`** : ligne, de droite à gauche



- **`column`** : colonne, de haut en bas
- **`column-reverse`** : colonne, de bas en haut



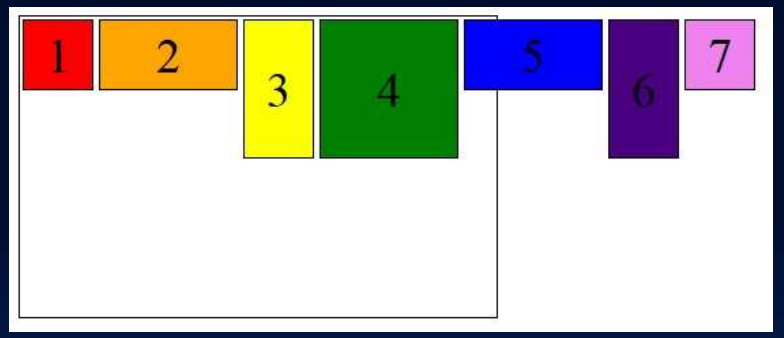


La direction est très importante car toutes les autres propriétés sont interprétées en fonction d'elle !

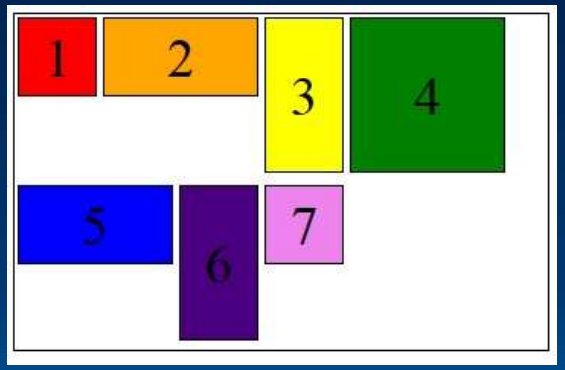
Propriété flex-wrap

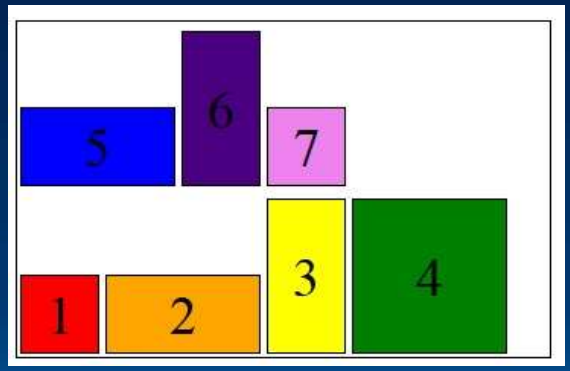
= Que faire lorsque les éléments à afficher sont trop nombreux pour tenir sur une ligne/colonne (`flex-wrap`) ?

- `nowrap` (défaut) : débordement



- `wrap` : passage à la ligne/colonne suivante
- `wrap-reverse` : passage à la ligne/colonne précédente



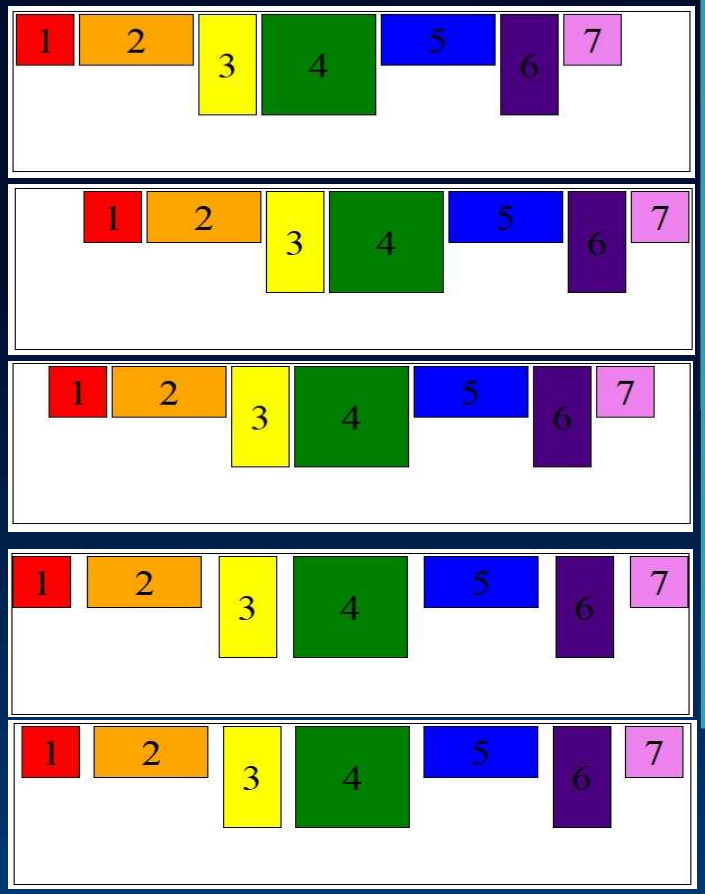


Propriété justify-content

= Placement dans la direction principale (`justify-content`) :

- **`flex-start`** (défaut) : vers le début
- **`flex-end`** : vers la fin
- **`center`** : au centre

Signification selon la direction :
= gauche/droite pour row/row-reverse
= haut/base pour column/column-reverse

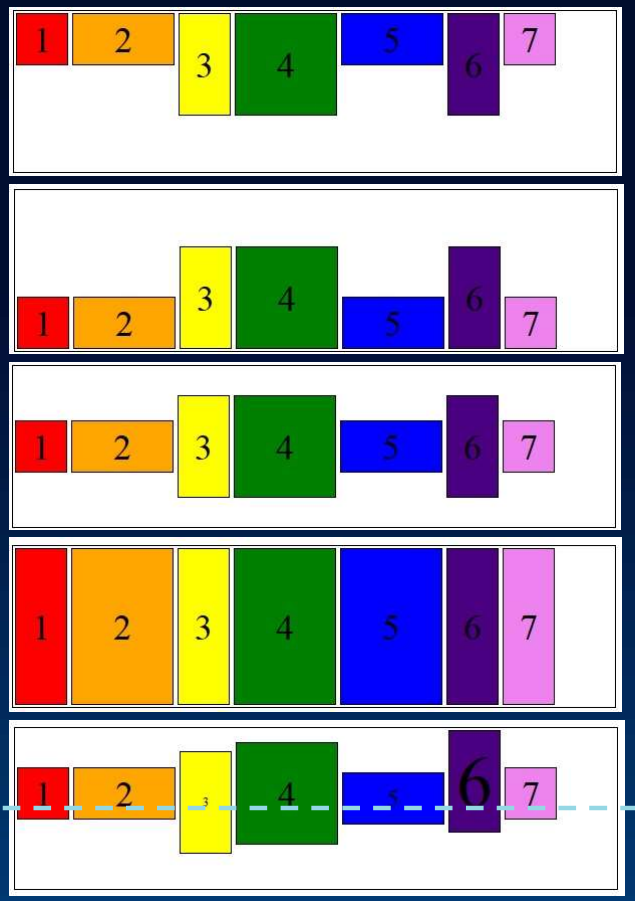


- **`space-between`** : espacements entre les éléments
- **`space-around`** : entre, avant et après les éléments

Propriété align-items

= Comment aligner les éléments entre eux (`align-items`) ?

- `flex-start` : vers le « début »
- `flex-end` : vers la « fin »
- `center` : selon le milieu
- `stretch` (défaut) : les étendre pour occuper toute la « largeur » disponible
- `baseline` : aligner selon la ligne d'écriture



Propriété align-content

= Si le contenu s'étend sur plusieurs rangées (lignes/colonnes), comment positionner celles-ci (`align-content`) ?

- `flex-start` (défaut) : toutes les rangées rassemblées vers le début
- `flex-end` : toutes les rangées rassemblées vers la fin
- `center` : toutes les rangées centrées au milieu du conteneur
- `stretch` : rangées étirées pour occuper tout l'espace disponible
- `space-between` : espacements entre les rangées
- `space-around` : espacements entre, avant et après les rangées

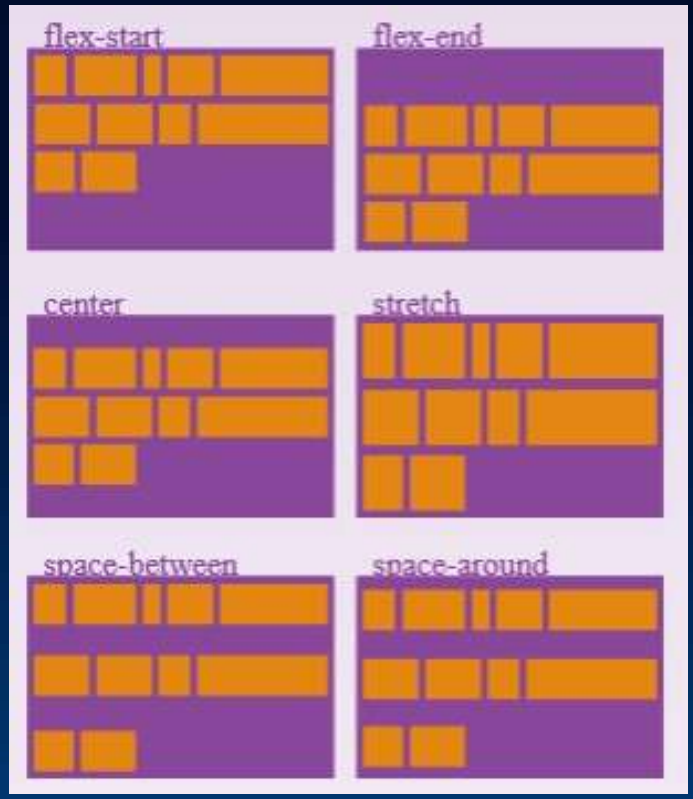


Image : css-tricks.com

Propriété flex-grow (élément)

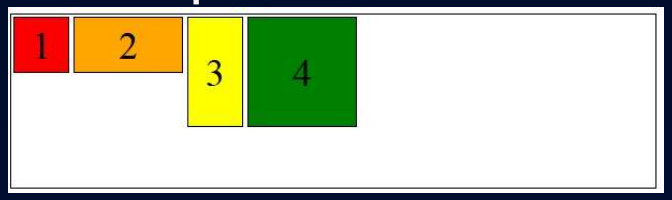
- Par défaut, dans la direction principale (gauche-droite pour une ligne), les éléments sont affichés avec leur taille naturelle et il peut rester de l'espace libre en fin de rangée.

- On peut utiliser la propriété **`flex-grow`** sur les éléments pour indiquer comment répartir l'espace libre.

  `flex-grow:1` signifie « je reçois 1 part de l'espace libre »
  `flex-grow:2` signifie « je reçois 2 parts de l'espace libre »
  etc.
  `flex-grow:0` signifie « je ne m'étends pas » (valeur par défaut)

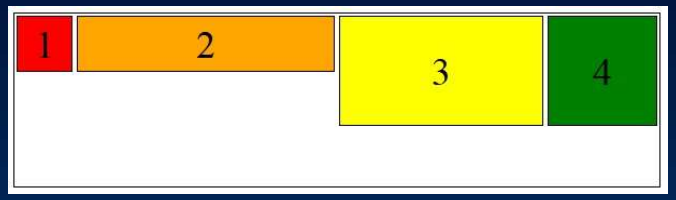
Propriété flex-grow (élément)

- Exemples :

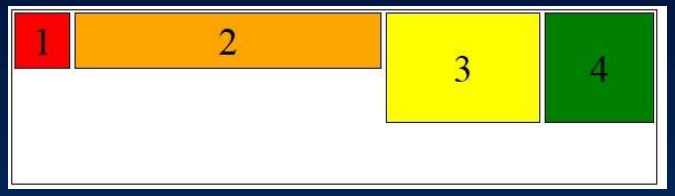


Par défaut
il reste de l'espace libre à droite

`flex-grow:1` sur l'élément 2
100% de l'espace libre est donné à l'élément 2



`flex-grow:1` sur les éléments 2 et 3
l'espace libre sera réparti équitablement (50%) entre les éléments 2 et 3



`flex-grow:2` sur l'élément 2 ;
`flex-grow:1` sur l'élément 3
l'élément 2 prend 2/3 de l'espace libre et l'élément 3, le reste (donc 1/3)