

Nouveau chapitre, nouveau lot de propriétés CSS. Ici, nous allons nous intéresser aux bordures et aux effets d'ombrage que l'on peut appliquer, aussi bien sur le texte que sur les blocs qui constituent notre page.

Nous réutiliserons en particulier nos connaissances sur les couleurs pour choisir la couleur de nos bordures et de nos ombres.

Prêts à vous en mettre une nouvelle fois plein la vue ? 😎

Bordures standard

Le CSS vous offre un large choix de bordures pour décorer votre page. De nombreuses propriétés CSS vous permettent de modifier l'apparence de vos bordures : `border-width`, `border-color`, `border-style`...

Pour aller à l'essentiel, je vous propose ici d'utiliser directement la super-propriété `border` qui regroupe l'ensemble de ces propriétés. Vous vous souvenez de la super-propriété `background` ? Cela fonctionne sur le même principe : on va pouvoir combiner plusieurs valeurs.

Pour `border` on peut utiliser jusqu'à trois valeurs pour modifier l'apparence de la bordure :

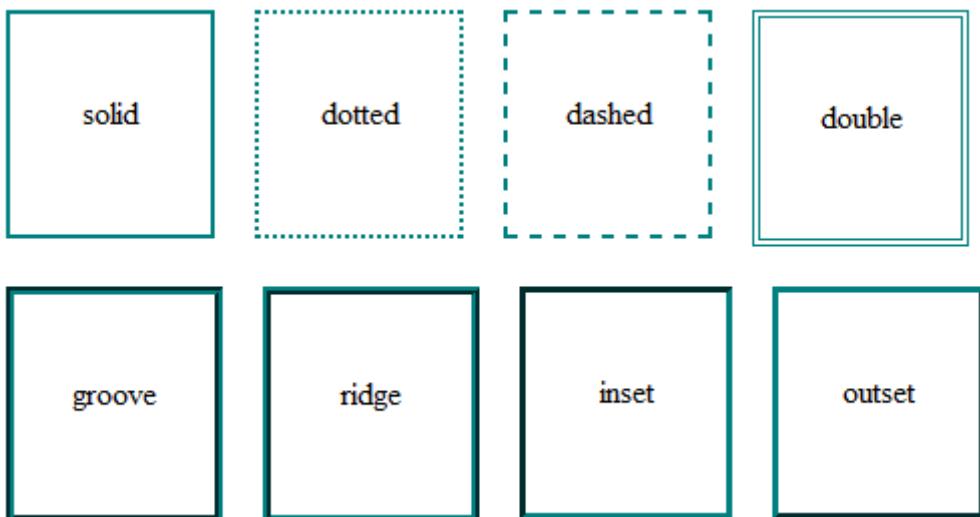
- **La largeur** : indiquez la largeur de votre bordure. Mettez une valeur en pixels (comme `2px`).
- **La couleur** : c'est la couleur de votre bordure. Utilisez, comme on l'a appris, soit un nom de couleur (`black`, `red`,...), soit une valeur hexadécimale (`#FF0000`), soit une valeur RGB (`rgb(198, 212, 37)`).
- **Le type de bordure** : là, vous avez le choix. Votre bordure peut être un simple trait, ou des pointillés, ou encore des tirets, etc. Voici les différentes valeurs disponibles :
 - `none` : pas de bordure (par défaut) ;
 - `solid` : un trait simple ;
 - `dotted` : pointillés ;
 - `dashed` : tirets ;
 - `double` : bordure double ;
 - `groove` : en relief ;
 - `ridge` : autre effet relief ;
 - `inset` : effet 3D global enfoncé ;

- outset : effet 3D global surélevé.

Ainsi, pour avoir une bordure bleue, en tirets, épaisse de 3 pixels autour de mes titres, je vais écrire :

```
h1
{
    border: 3px blue dashed;
```

La figure suivante vous présente les différents styles de bordures que vous pouvez utiliser.



Les différents types de bordures

En haut, à droite, à gauche, en bas...

Qui a dit que vous étiez obligés d'appliquer la même bordure aux quatre côtés de votre élément ?

Taratata, si vous voulez mettre des bordures différentes en fonction du côté (haut, bas, gauche ou droite), vous pouvez le faire sans problème. Dans ce cas, vous devrez utiliser ces quatre propriétés :

- border-top : bordure du haut ;
- border-bottom : bordure du bas ;
- border-left : bordure de gauche ;
- border-right : bordure de droite.

Il existe aussi des équivalents pour paramétriser chaque détail de la bordure si vous le désirez : border-top-width pour modifier l'épaisseur de la bordure du haut, border-top-color pour la couleur du haut, etc.

Ce sont aussi des super-propriétés, elles fonctionnent comme `border` mais ne s'appliquent donc qu'à un seul côté.

Pour ajouter une bordure uniquement à gauche et à droite des paragraphes, on écrira donc :

```
p  
{  
    border-left: 2px solid black;  
    border-right: 2px solid black;  
}
```

On peut modifier les bordures de n'importe quel type d'élément sur la page. Nous l'avons fait ici sur les paragraphes mais on peut aussi modifier la bordure des images, des textes importants comme ``, etc.

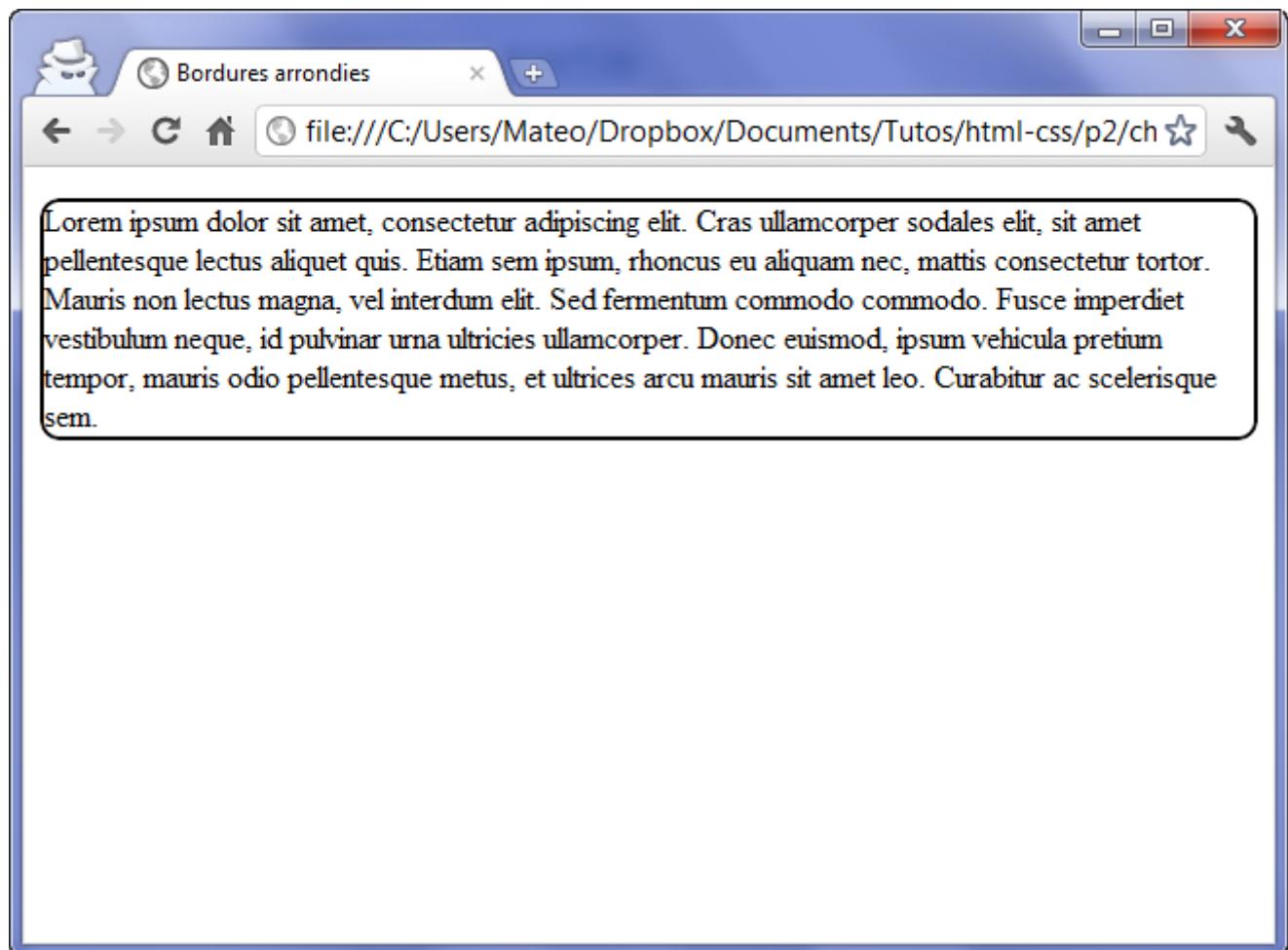
Bordures arrondies

Les bordures arrondies, c'est un peu le Saint Graal attendu par les webmasters depuis des millénaires (ou presque). Depuis que CSS3 est arrivé, il est enfin possible d'en créer facilement !

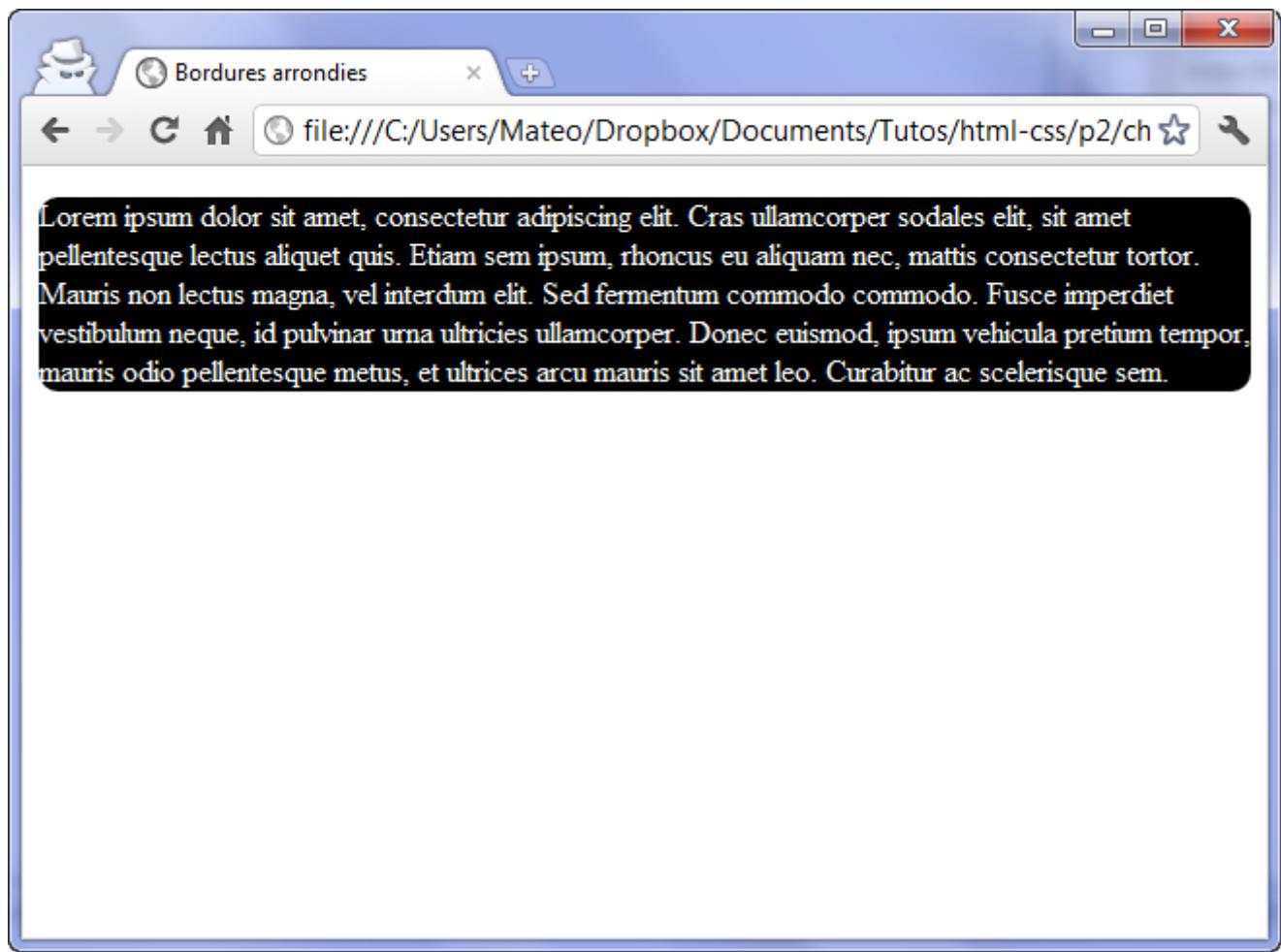
La propriété `border-radius` va nous permettre d'arrondir facilement les angles de n'importe quel élément. Il suffit d'indiquer la taille (« l'importance ») de l'arrondi en pixels :

```
p  
{  
    border-radius: 10px;  
}
```

L'arrondi se voit notamment si l'élément a des bordures, comme sur la figure suivante.



Des bordures arrondies
... ou s'il a une couleur de fond, comme sur la figure suivante.



Un fond aux coins arrondis

On peut aussi préciser la forme de l'arrondi pour chaque coin. Dans ce cas, indiquez quatre valeurs :

```
p  
{  
    border-radius: 10px 5px 10px 5px  
}
```

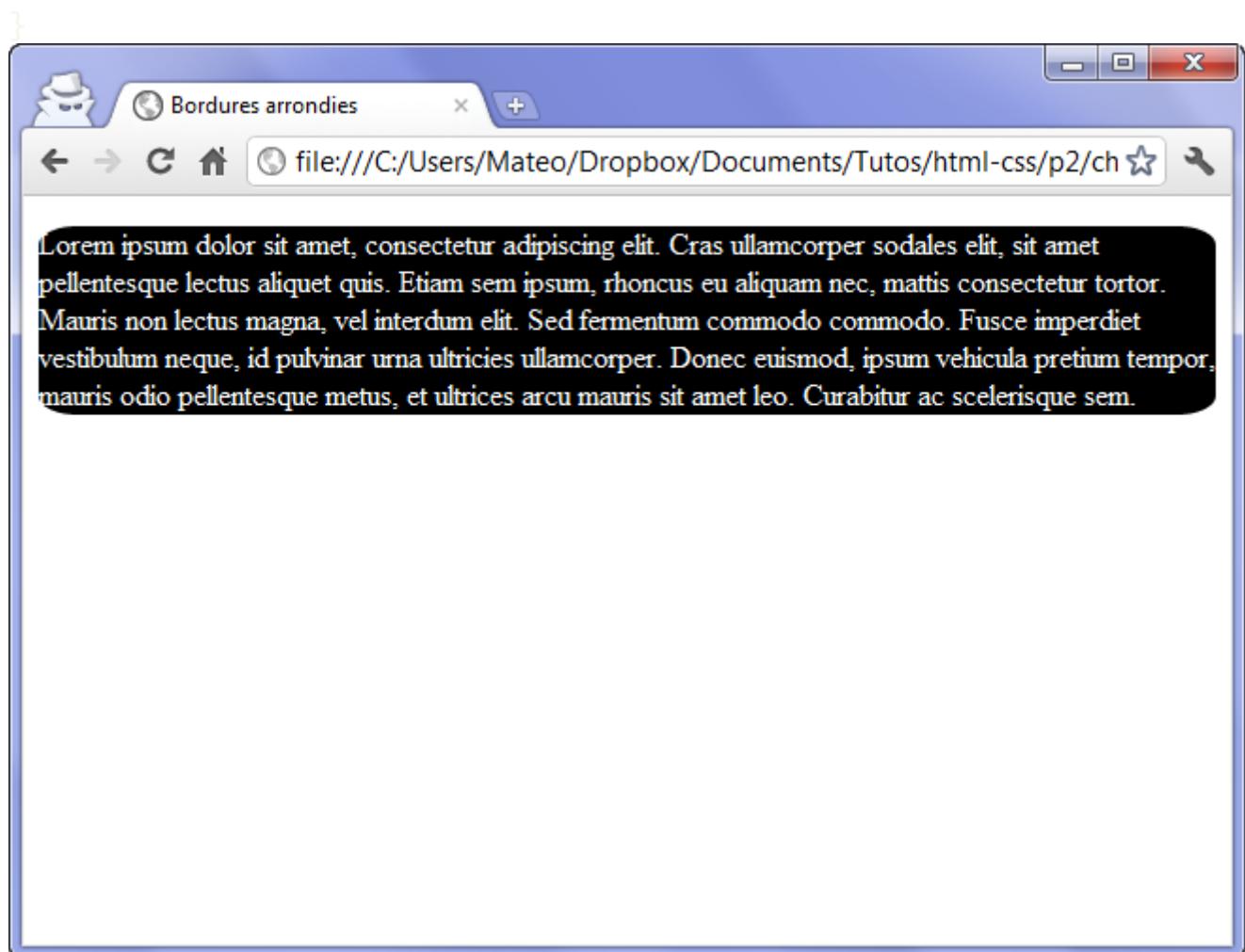
Les valeurs correspondent aux angles suivants dans cet ordre :

- 1 en haut à gauche ;
- 2 en haut à droite ;
- 3 en bas à droite ;
- 4 en bas à gauche.

Enfin, il est possible d'affiner l'arrondi de nos angles en créant des courbes elliptiques figure suivante). Dans ce cas, il faut indiquer deux valeurs séparées par une barre oblique (slash, caractère /). Le

mieux est certainement de tester pour voir l'effet :

```
p  
{  
    border-radius: 20px / 10px;  
}
```



Bordures arrondies elliptiques

Les ombres

Les ombres font partie des nouveautés récentes proposées par CSS3. Aujourd'hui, il suffit d'une seule ligne de CSS pour ajouter des ombres dans une page !

Nous allons ici découvrir deux types d'ombres :

- les ombres des boîtes ;
- les ombres du texte.

box-shadow : les ombres des boîtes

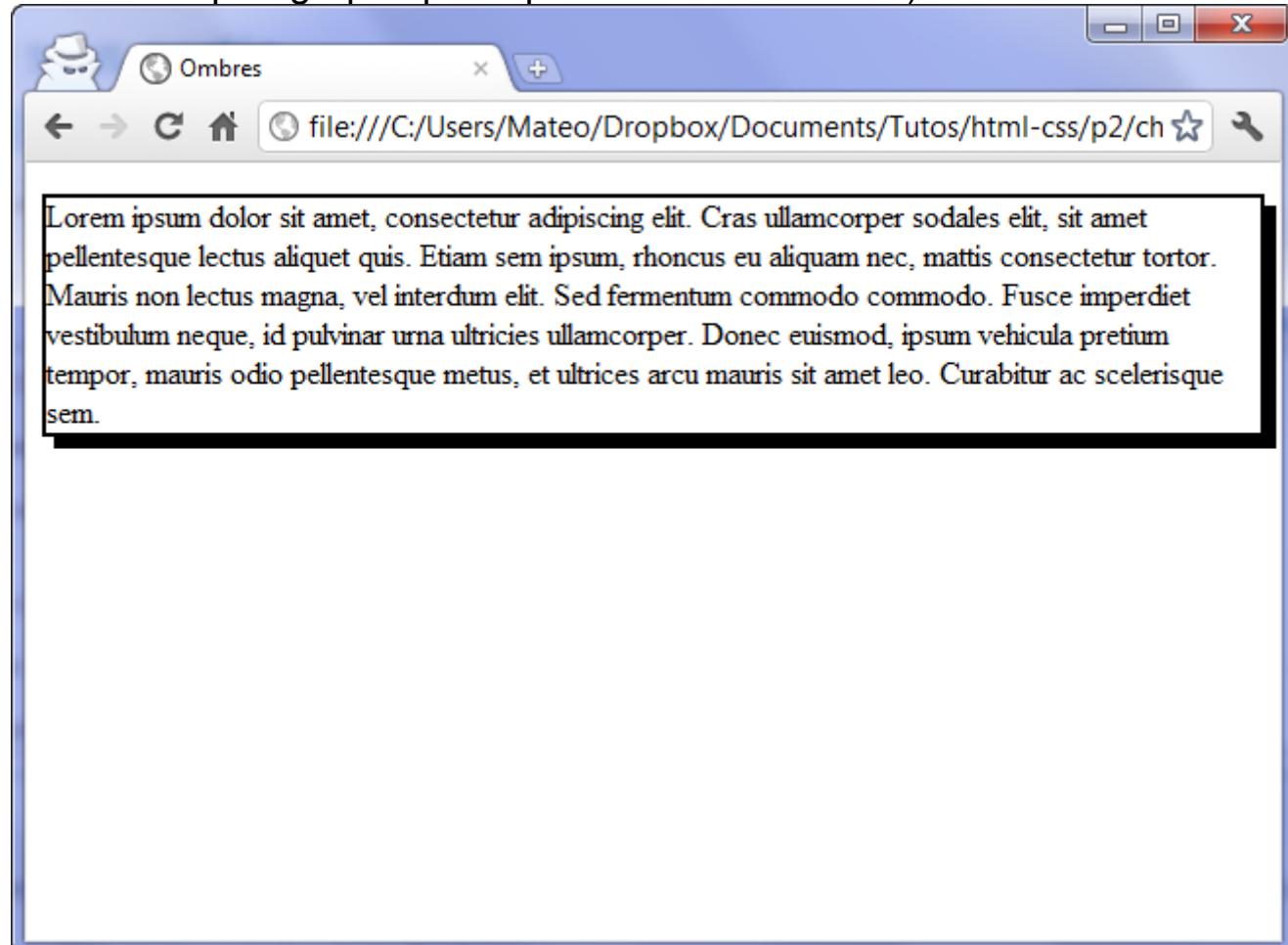
La propriété `box-shadow` s'applique à tout le bloc et prend quatre valeurs dans l'ordre suivant :

- 1 le décalage horizontal de l'ombre ;
- 2 le décalage vertical de l'ombre ;
- 3 l'adoucissement du dégradé ;
- 4 la couleur de l'ombre.

Par exemple, pour une ombre noire de 6 pixels, sans adoucissement, on écrira :

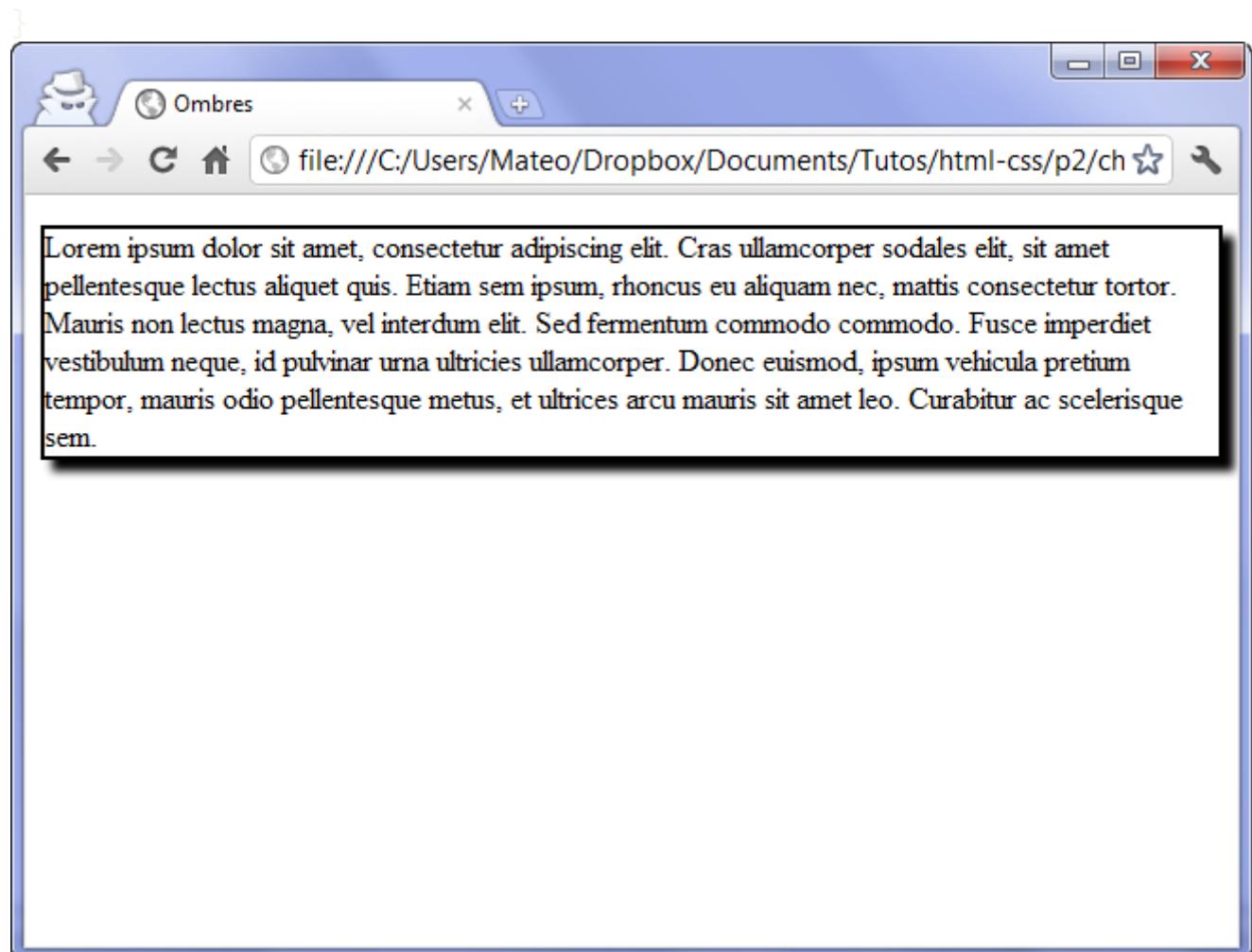
```
p  
{  
    box-shadow: 6px 6px 0px black;  
}
```

Cela donne le résultat illustré à la figure suivante (j'ai ajouté une bordure au paragraphe pour qu'on voie mieux l'effet).



Une ombre sous le paragraphe
Ajoutons un adoucissement grâce au troisième paramètre (figure suivante). L'adoucissement peut être faible (inférieur au décalage), normal (égal au décalage) ou élevé (supérieur au décalage).
Essayons un décalage normal :

```
p {  
    box-shadow: 6px 6px 6px black;  
}
```



Une ombre adoucie sous le paragraphe
On peut aussi rajouter une cinquième valeur facultative : `inset`.
Dans ce cas, l'ombre sera placée à l'intérieur du bloc, pour donner un effet enfoncé :

```
p {
```

```
{  
    box-shadow: 6px 6px 6px black inset;  
}
```

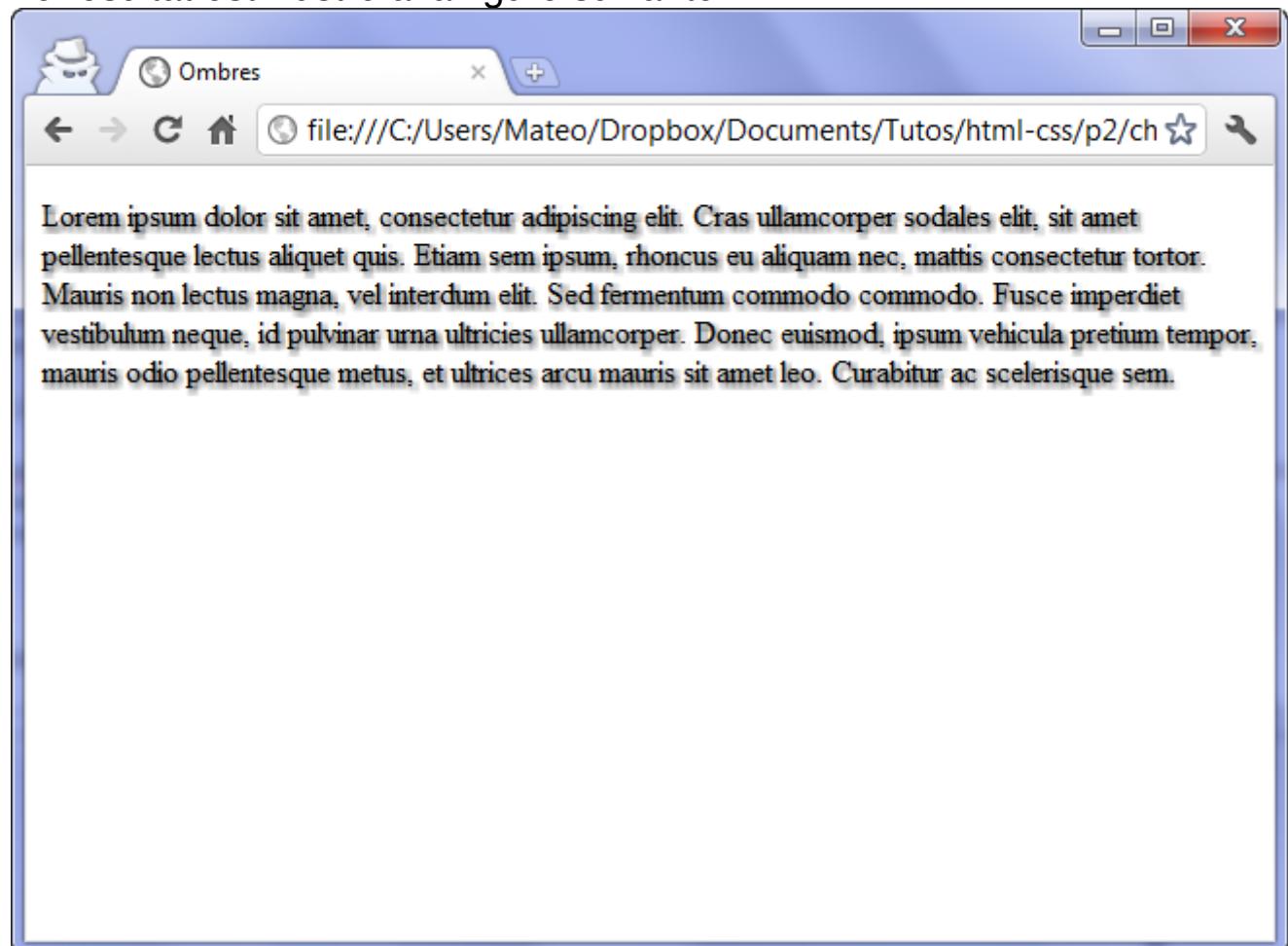
Je vous laisse essayer de voir le résultat.

text-shadow : l'ombre du texte

Avec `text-shadow`, vous pouvez ajouter une ombre directement sur les lettres de votre texte ! Les valeurs fonctionnent exactement de la même façon que `box-shadow` : décalage, adoucissement et couleur.

```
p  
{  
    text-shadow: 2px 2px 4px black;  
}
```

Le résultat est illustré à la figure suivante.



Texte ombré

En résumé

- On peut appliquer une bordure à un élément avec la propriété `border`. Il faut indiquer la largeur de la bordure, sa couleur et son type (trait continu, pointillés...).
- On peut arrondir les bordures avec `border-radius`.
- On peut ajouter une ombre aux blocs de texte avec `box-shadow`.
On doit indiquer le décalage vertical et horizontal de l'ombre, son niveau d'adoucissement et sa couleur.

Le texte peut lui aussi avoir une ombre avec `text-shadow`.

C'est une de ses forces : le CSS nous permet aussi de modifier l'apparence des éléments de façon dynamique, c'est-à-dire que des éléments peuvent changer de forme une fois que la page a été chargée. Nous allons faire appel à une fonctionnalité puissante du CSS : les pseudo-formats.

Nous verrons dans ce chapitre comment changer l'apparence :

- au survol ;
- lors du clic ;
- lors du focus (élément sélectionné) ;
- lorsqu'un lien a été consulté.

Vous allez voir que le langage CSS n'a pas fini de nous étonner !

Au survol

Nous allons découvrir dans ce chapitre plusieurs pseudo-formats CSS. Le premier que je vais vous montrer s'appelle `:hover`. Comme tous les autres pseudo-formats que nous allons voir, c'est une information que l'on rajoute après le nom de la balise (ou de la classe) dans le CSS, comme ceci :

`a:hover`

{

```
}
```

`:hover` signifie « survoler ». `a:hover` peut donc se traduire par : « Quand la souris est sur le lien » (quand on pointe dessus).

À partir de là, c'est à vous de définir l'apparence que doivent avoir les liens lorsqu'on pointe dessus. Laissez libre cours à votre imagination, il n'y a pas de limite.

Voici un exemple de présentation des liens, mais n'hésitez pas à inventer le vôtre :

```
a /* Liens par défaut (non survolés) */
```

```
{
```

```
  text-decoration: none;
```

```
  color: red;
```

```
  font-style: italic;
```

```
}
```

```
a:hover /* Apparence au survol des liens */
```

```
{
```

```
  text-decoration: underline;
```

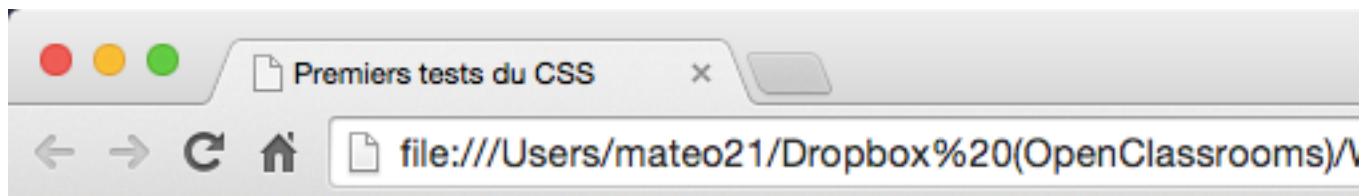
```
  color: green;
```

```
}
```

On a défini ici deux versions des styles pour les liens :

- pour les liens par défaut (non survolés) ;
- pour les liens au survol.

Le résultat se trouve à la figure suivante.



Quelques bonnes adresses

Vous connaissez **Google** ? C'est le moteur de recherche le plus utilisé au monde.

Vous connaissez **le W3C** ? Ce sont les personnes qui définissent HTML et CSS.

Vous connaissez **OpenClassrooms** ? Ah ben oui, quelle question stupide...



openclassrooms.com

Changement d'apparence au survol de la souris

Sympa, n'est-ce pas ?

Même si on l'utilise souvent sur les liens, vous pouvez modifier l'apparence de n'importe quel élément. Par exemple, vous pouvez modifier l'apparence des paragraphes lorsqu'on pointe dessus :

```
p:hover /* Quand on pointe sur un paragraphe */
```

```
{
```

```
}
```

Au clic et lors de la sélection

Vous pouvez interagir encore plus finement en CSS. Nous allons voir ici que nous pouvons changer l'apparence des éléments lorsque l'on clique dessus et lorsqu'ils sont sélectionnés !

:active : au moment du clic

Le pseudo-format `:active` permet d'appliquer un style particulier au moment du clic. En pratique, il n'est utilisé que sur les liens.

Le lien gardera cette apparence très peu de temps : en fait, le changement intervient lorsque le bouton de la souris est enfoncé. En clair, ce n'est pas forcément toujours bien visible.

On peut par exemple changer la couleur de fond du lien lorsque l'on clique dessus :

```
a:active /* Quand le visiteur clique sur le lien */
```

```
{
```

```
    background-color: #FFCC66;
```

```
}
```

:focus : lorsque l'élément est sélectionné

Là, c'est un peu différent. Le pseudo-format `:focus` applique un style lorsque l'élément est sélectionné.

C'est-à-dire ?

Une fois que vous avez cliqué, le lien reste « sélectionné » (il y a une petite bordure en pointillés autour). C'est cela, la sélection.

Ce pseudo-format pourra être appliqué à d'autres balises HTML que nous n'avons pas encore vues, comme les éléments de formulaires. Essayons pour l'instant sur les liens :

```
a:focus /* Quand le visiteur sélectionne le lien */
```

```
{
```

```
    background-color: #FFCC66;
```

```
}
```

Sous Google Chrome et Safari, l'effet ne se voit que si l'on appuie

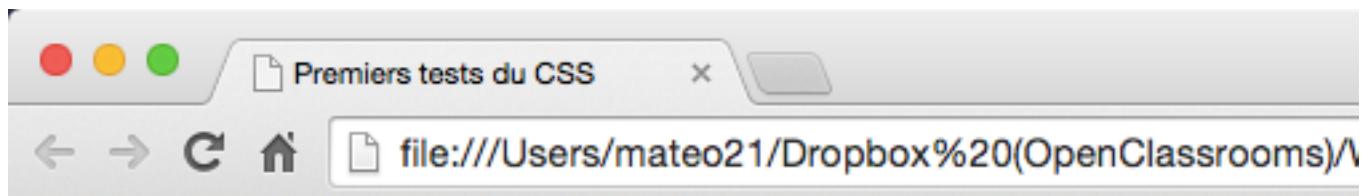
sur la touche **Tab**.

Lorsque le lien a déjà été consulté

Il est possible d'appliquer un style à un lien vers une page qui a déjà été vue. Par défaut, le navigateur colore le lien en un violet assez laid (de mon point de vue du moins !).

Vous pouvez changer cette apparence avec `:visited` (qui signifie « visité »). En pratique, sur les liens consultés, on ne peut pas changer beaucoup de choses à part la couleur (figure suivante).

```
a:visited /* Quand le visiteur a déjà vu la page concernée */  
{  
    color: #AAA; /* Appliquer une couleur grise */  
}
```



Quelques bonnes adresses

Vous connaissez *Google* ? C'est le moteur de recherche le plus utilisé au monde

Vous connaissez *le W3C* ? Ce sont les personnes qui définissent HTML et CSS

Vous connaissez *OpenClassrooms* ? Ah ben oui, quelle question stupide...

Liens visités en gris

Si vous ne souhaitez pas que les liens déjà visités soient colorés d'une façon différente, il vous faudra leur appliquer la même couleur qu'aux liens normaux. De nombreux sites web font cela (OpenClassrooms y compris !). Une exception notable : Google... ce qui est plutôt pratique, puisque l'on peut voir dans les résultats d'une recherche si on a déjà consulté ou non les sites que Google nous présente.

En résumé

5 En CSS, on peut modifier l'apparence de certaines sections

dynamiquement, après le chargement de la page, lorsque certains évènements se produisent. On utilise pour cela les pseudo-formats.

6 Le pseudo-format :`hover` permet de changer l'apparence au survol (par exemple : `a:hover` pour modifier l'apparence des liens lorsque la souris pointe dessus).

7 Le pseudo-format :`active` modifie l'apparence des liens au moment du clic, :`visited` lorsqu'un lien a déjà été visité.

Le pseudo-format :`focus` permet de modifier l'apparence d'un élément sélectionné.

Structurer sa page

Nous approchons de plus en plus du but. Si nos pages web ne ressemblent pas encore tout à fait aux sites web que nous connaissons, c'est qu'il nous manque les connaissances nécessaires pour faire la mise en page.

En général, une page web est constituée d'un en-tête (tout en haut), de menus de navigation (en haut ou sur les côtés), de différentes sections au centre... et d'un pied de page (tout en bas).

Dans ce chapitre, nous allons nous intéresser aux nouvelles balises HTML dédiées à la structuration du site. Ces balises ont été introduites par HTML5 (elles n'existaient pas avant) et vont nous permettre de dire : « Ceci est mon en-tête », « Ceci est mon menu de navigation », etc.

Pour le moment, nous n'allons pas encore faire de mise en page. Nous allons en fait préparer notre document HTML pour pouvoir découvrir la mise en page dans les prochains chapitres.

Les balises structurantes de HTML5

Je vais vous présenter ici les nouvelles balises introduites par HTML5 pour structurer nos pages. Vous allez voir, cela ne va pas beaucoup changer l'apparence de notre site pour le moment, mais il sera bien construit et prêt à être mis en forme ensuite !

<header> : l'en-tête

La plupart des sites web possèdent en général un en-tête, appelé header en anglais. On y trouve le plus souvent un logo, une

bannière, le slogan de votre site...

Vous devrez placer ces informations à l'intérieur de la balise

<header> :

header

<!-- Placez ici le contenu de l'en-tête de votre page -->

header

La figure suivante, par exemple, représente le site du W3C (qui se charge des nouvelles versions de HTML et CSS notamment). La partie encadrée en rouge correspondrait à l'en-tête :



L'en-tête du site du W3C

L'en-tête peut contenir tout ce que vous voulez : images, liens, textes...

Il peut y avoir plusieurs en-têtes dans votre page. Si celle-ci est découpée en plusieurs sections, chaque section peut en effet avoir son propre <header>.

<footer> : le pied de page

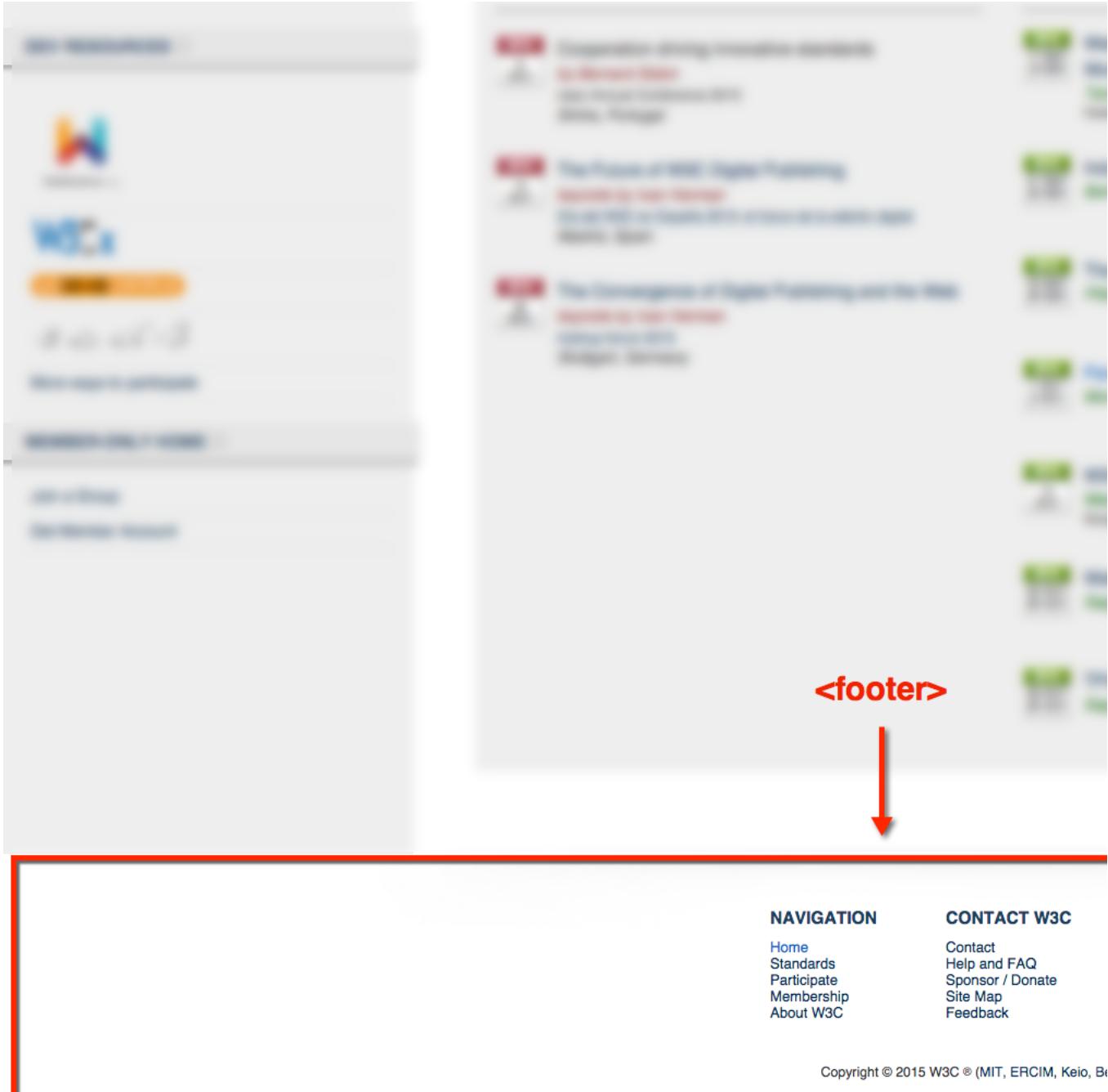
À l'inverse de l'en-tête, le pied de page se trouve en général tout en bas du document. On y trouve des informations comme des liens de contact, le nom de l'auteur, les mentions légales, etc.

footer

<!-- Placez ici le contenu du pied de page -->

footer

La figure suivante vous montre à quoi ressemble le pied de page du W3C.



Pied de page du W3C

<nav> : principaux liens de navigation

La balise `<nav>` doit regrouper tous les principaux liens de navigation du site. Vous y placerez par exemple le menu principal

de votre site.

Généralement, le menu est réalisé sous forme de liste à puces à l'intérieur de la balise <nav> :

```
nav
```

```
  ul
```

```
    li  a href="index.html" Accueil  a  li
```

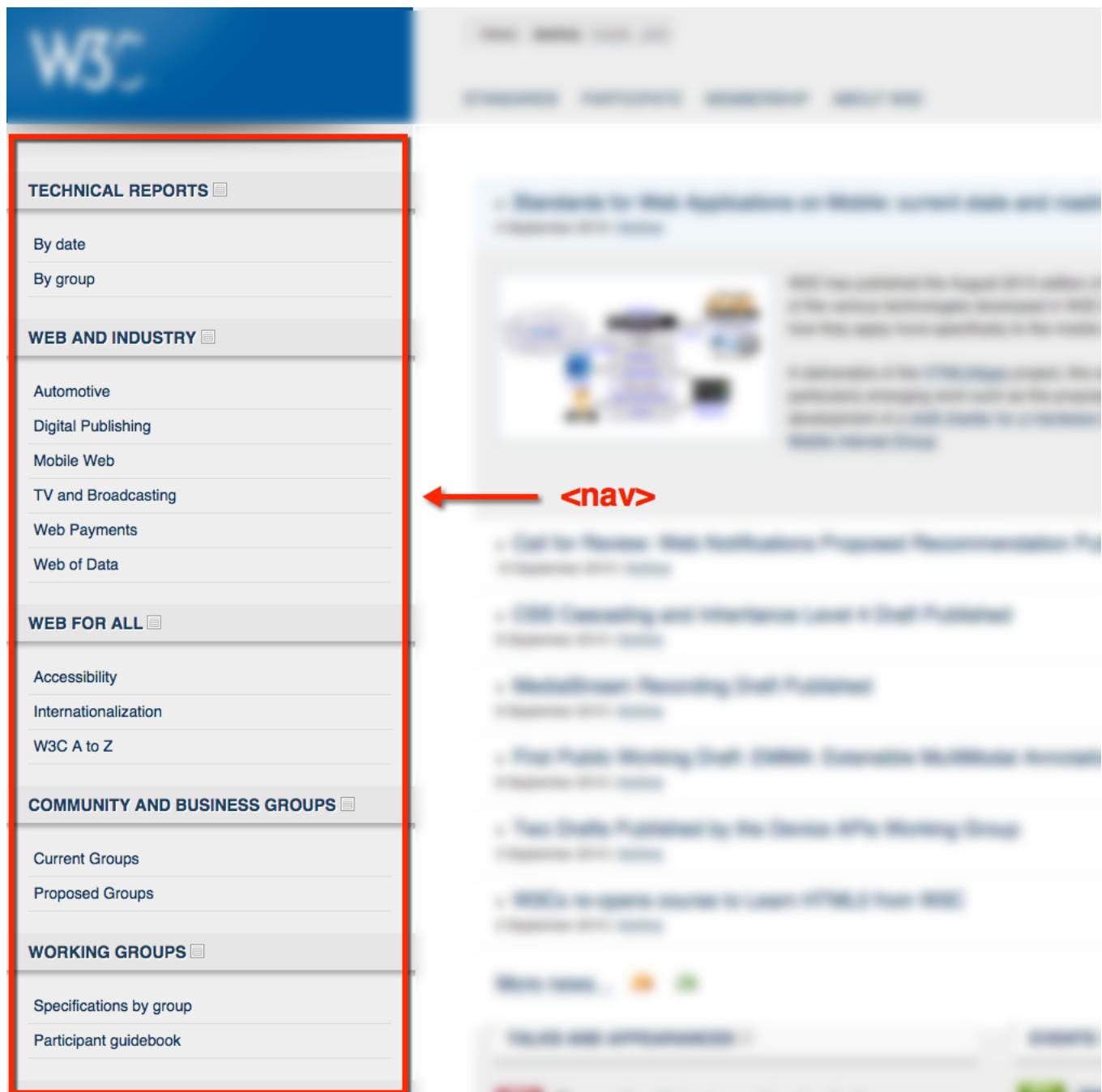
```
    li  a href="forum.html" Forum  a  li
```

```
    li  a href="contact.html" Contact  a  li
```

```
  ul
```

```
nav
```

Voici le menu sur le site du W3C : <nav>.



Le menu de navigation du W3C

<section> : une section de page

La balise `<section>` sert à regrouper des contenus en fonction de leur thématique. Elle englobe généralement une portion du contenu au centre de la page.

section

```
h1 Ma section de page h1
```

```
p Bla bla bla bla p
```

section

Sur la page d'accueil du portail Free.fr, on trouve plusieurs blocs qui pourraient être considérés comme des sections de page (figure suivante).

The screenshot shows the Free.fr homepage with several sections highlighted by red boxes:

- Actualités**: A large section at the top containing a thumbnail image of a rugby match, a list of news items, and a red label <section>.
- FHV**: A section featuring a thumbnail for a movie called "BEATDOWN", a promotional offer for "La vidéo à la demande en illimité à partir de 7,99€/mois", and a list of "Nouveautés de la semaine". It also contains a red label <section>.
- Assistance**: A section featuring a thumbnail of a Freebox device, a link to "Installer votre Freebox Révolution", and a list of troubleshooting topics. It also contains a red label <section>.

Des sections de page sur le portail de Free

Chaque section peut avoir son titre de niveau 1 (`<h1>`), de même que l'en-tête peut contenir un titre `<h1>` lui aussi. Chacun de ces blocs étant indépendant des autres, il n'est pas illogique de retrouver plusieurs titres `<h1>` dans le code de la page web. On a ainsi « Le titre `<h1>` du `<header>` », « Le titre `<h1>` de cette `<section>` », etc.

<aside> : informations complémentaires

La balise `<aside>` est conçue pour contenir des informations complémentaires au document que l'on visualise. Ces informations

sont généralement placées sur le côté (bien que ce ne soit pas une obligation).

aside

<!-- Placez ici des informations complémentaires -->

aside

Il peut y avoir plusieurs blocs `<aside>` dans la page.

Sur Wikipédia, par exemple, il est courant de voir à droite un bloc d'informations complémentaires à l'article que l'on visualise. Ainsi, sur la page présentant la planète Saturne (figure suivante), on trouve dans ce bloc les caractéristiques de la planète (dimensions, masse, etc.).



Bloc d'informations complémentaires sur Wikipédia

<article> : un article indépendant

La balise `<article>` sert à englober une portion généralement

autonome de la page. C'est une partie de la page qui pourrait ainsi être reprise sur un autre site. C'est le cas par exemple des actualités (articles de journaux ou de blogs).

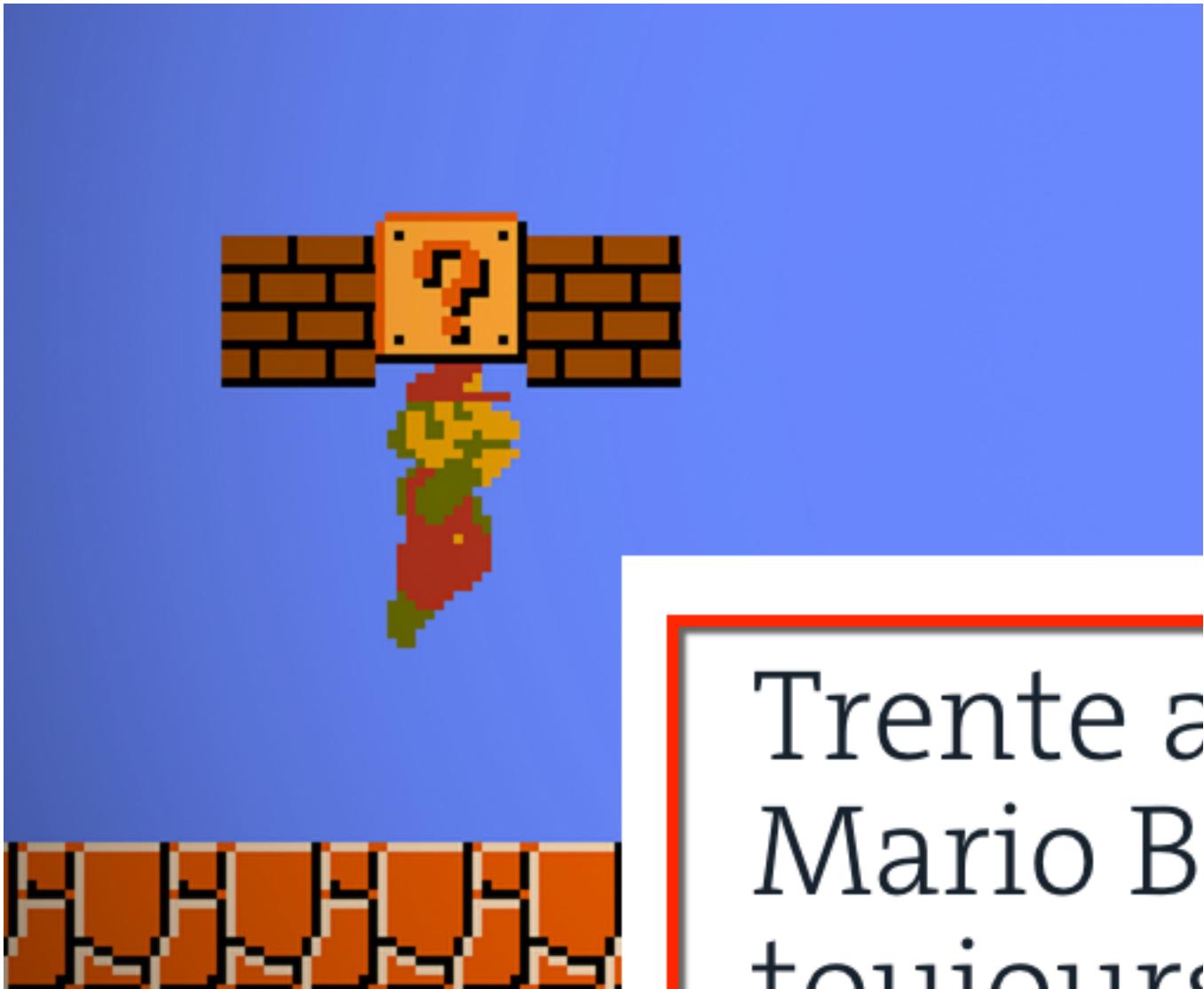
article

h1 Mon article **h1**

p Bla bla bla bla **p**

article

Par exemple, voici un article sur le Monde :



Trente ans à Mario B. toujours

Le jeu vidéo le plus alimenté d'alimenter active univers iconique et

Abonnez-vous
à partir de 1 €

Réagir

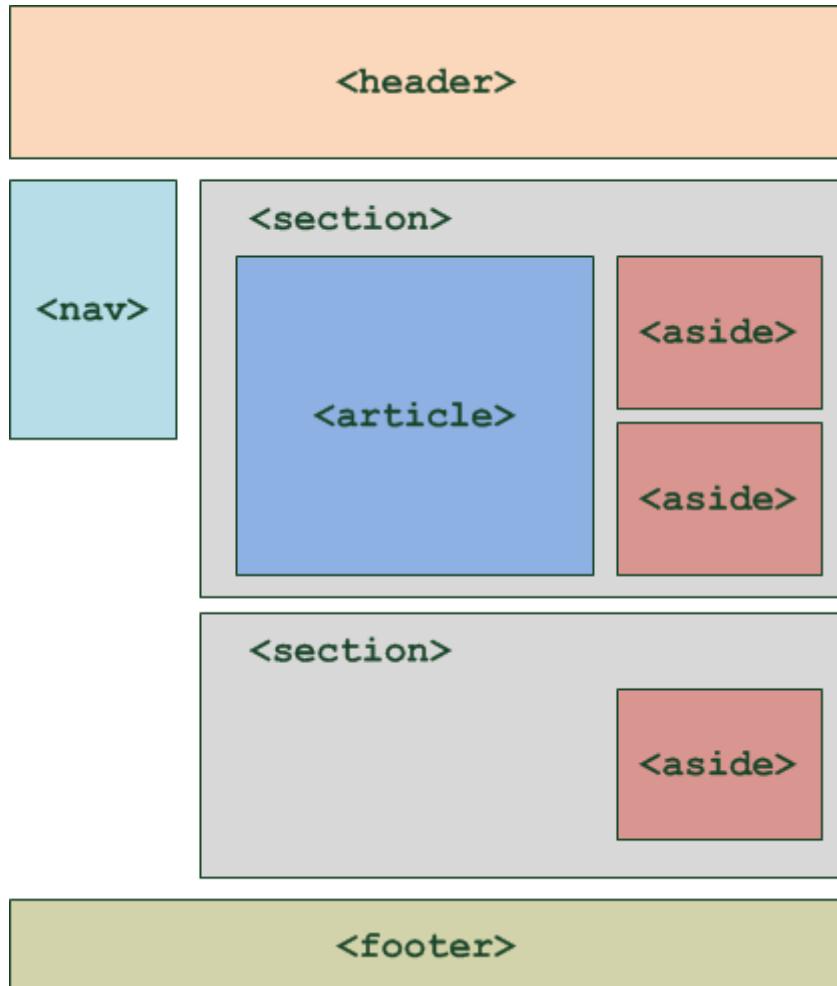
Recommander Partager 27 pc

Un article publié sur le Monde

Résumé

Ouf, cela fait beaucoup de nouvelles balises à retenir.

Heureusement, je vous ai fait un petit schéma (figure suivante) pour vous aider à retenir leur rôle !



Sections de la page identifiées par les balises

Ne vous y trompez pas : ce schéma propose un exemple d'organisation de la page. Rien ne vous empêche de décider que votre menu de navigation soit à droite, ou tout en haut, que vos balises **<aside>** soient au-dessus, etc.

On peut même imaginer une seconde balise **<header>**, placée cette fois à l'intérieur d'une **<section>**. Dans ce cas-là, elle sera considérée comme étant l'en-tête de la section.

Enfin, une section ne doit pas forcément contenir un **<article>** et des **<aside>**. Utilisez ces balises uniquement si vous en avez besoin. Rien ne vous interdit de créer des sections contenant seulement des paragraphes, par exemple.

Exemple concret d'utilisation des balises

Essayons d'utiliser les balises que nous venons de découvrir pour structurer notre page web. Le code ci-dessous reprend toutes les balises que nous venons de voir au sein d'une page web complète :

```
<!DOCTYPE html>

html
  head
    meta charset="utf-8"
    title Zozor - Le Site Web title
  head

body
  header
    h1 Zozor h1
    h2 Carnets de voyage h2
  header

  nav
    ul
      li a href="#" Accueil a li
      li a href="#" Blog a li
      li a href="#" CV a li
    ul
```

```
nav

section

    aside

        h1 À propos de l'auteur h1

        p C'est moi, Zozor ! Je suis né un 23
novembre 2005. p

    aside

    article

        h1 Je suis un grand voyageur h1

        p Bla bla bla bla (texte de l'article) p

    article

section

footer

    p Copyright Zozor - Tous droits réservés br

    a href="#" Me contacter ! a p

footer

body

html
Ce code peut vous aider à comprendre comment les balises doivent
être agencées. Vous y reconnaîtrez un en-tête, un menu de
navigation, un pied de page... et, au centre, une section avec un
```

article et un bloc `<aside>` donnant des informations sur l'auteur de l'article.

À quoi ressemble la page que nous venons de créer ?

À rien !

Si vous testez le résultat, vous verrez juste du texte noir sur fond blanc (figure suivante). C'est normal, il n'y a pas de CSS ! Par contre, la page est bien structurée, ce qui va nous être utile pour la suite.

The screenshot shows a Microsoft Internet Explorer window with the title bar "Zozor - Le Site Web". The address bar displays the local file path: "file:///C:/Users/Mateo/Dropbox/Documents/Tutos/html-css/p3/ch". The main content area contains:

- # Zozor
- ## Carnets de voyage
- [Accueil](#)
 - [Blog](#)
 - [CV](#)
- ### A propos de l'auteur
- C'est moi, Zozor ! Je suis né un 23 novembre 2005.
- ### Je suis un grand voyageur
- Bla bla bla bla (texte de l'article)
- Copyright Zozor - Tous droits réservés
[Me contacter !](#)

Une page bien structurée mais sans CSS

Les liens sont volontairement factices (d'où la présence d'un simple #), ils n'amènent donc nulle part (eh, c'est juste une page de démo) !

Je ne comprends pas l'intérêt de ces balises. On peut très bien obtenir le même résultat sans les utiliser !

C'est vrai. En fait, ces balises sont seulement là pour expliquer à l'ordinateur « Ceci est l'en-tête », « Ceci est mon pied de page », etc. Elles n'indiquent pas, contrairement à ce qu'on pourrait penser, où doit être placé le contenu. C'est le rôle du CSS, comme nous le

verrons dans peu de temps maintenant.

À l'heure actuelle, pour tout vous dire, ces balises ont encore assez peu d'utilité. On pourrait très bien utiliser des balises génériques `<div>` à la place pour englober les différentes portions de notre contenu. D'ailleurs, c'est comme cela qu'on faisait avant l'arrivée de ces nouvelles balises HTML5.

Néanmoins, il est assez probable que, dans un futur proche, les ordinateurs commencent à tirer parti intelligemment de ces nouvelles balises. On peut imaginer par exemple un navigateur qui choisisse d'afficher les liens de navigation `<nav>` de manière toujours visible ! Quand l'ordinateur « comprend » la structure de la page, tout devient possible.

En résumé

- Plusieurs balises ont été introduites avec HTML5 pour délimiter les différentes zones qui constituent la page web :
 - `<header>` : en-tête ;
 - `<footer>` : pied de page ;
 - `<nav>` : liens principaux de navigation ;
 - `<section>` : section de page ;
 - `<aside>` : informations complémentaires ;
 - `<article>` : article indépendant.
- Ces balises peuvent être imbriquées les unes dans les autres.
Ainsi, une section peut avoir son propre en-tête.

Ces balises ne s'occupent pas de la mise en page. Elles servent seulement à indiquer à l'ordinateur le sens du texte qu'elles contiennent. On pourrait très bien placer l'en-tête en bas de la page si on le souhaite.

Le modèle des boîtes

Une page web peut être vue comme une succession et un empilement de boîtes, qu'on appelle « blocs ». La plupart des éléments vus au chapitre précédent sont des blocs

:`<header>`,`<article>`,`<nav>`... Mais nous connaissons déjà d'autres blocs : les paragraphes`<p>`, les titres`<h1>`...

Dans ce chapitre, nous allons apprendre à manipuler ces blocs comme de véritables boîtes. Nous allons leur donner des dimensions, les agencer en jouant sur leurs marges, mais aussi apprendre à gérer leur contenu... pour éviter que le texte ne dépasse de ces blocs !

Ce sont des notions fondamentales dont nous allons avoir besoin pour mettre en page notre site web... Soyez attentifs !

Les balises de type block et inline

En HTML, la plupart des balises peuvent se ranger dans l'une ou l'autre de deux catégories :

- Les balises **inline** : c'est le cas par exemple des liens `<a>`.
- Les balises **block** : c'est le cas par exemple des paragraphes `<p></p>`.

Il existe en fait plusieurs autres catégories très spécifiques, par exemple pour les cellules de tableau (`type=table-cell`) ou les puces (`type=list-item`). Nous n'allons pas nous y intéresser pour le moment car ces balises sont minoritaires.

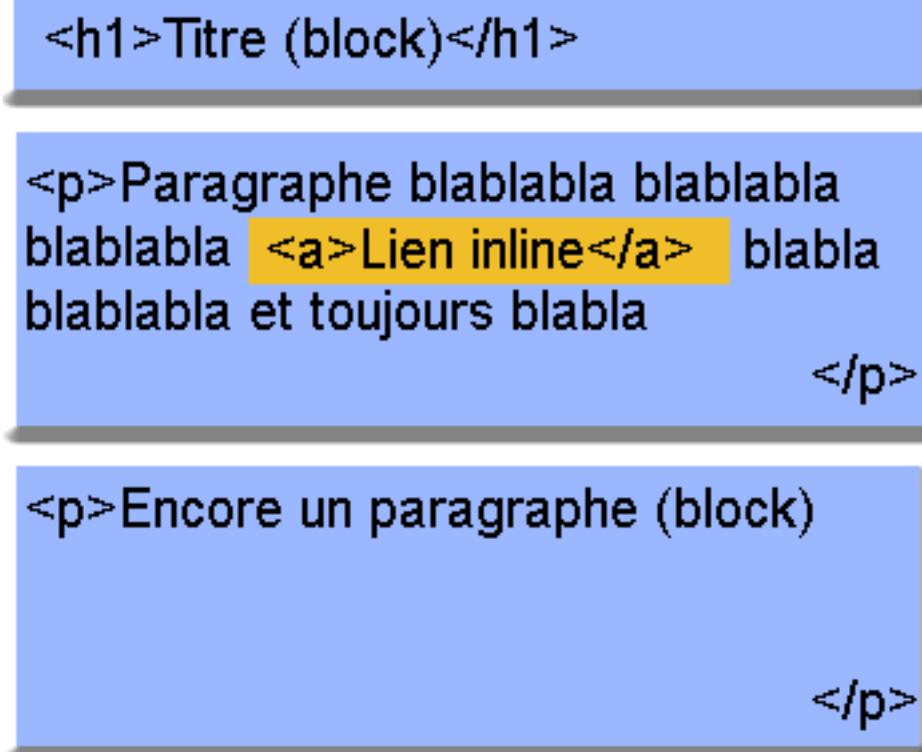
Mais comment je reconnaiss une balise inline d'une balise block ?

C'est en fait assez facile :

- **block** : une balise de type block sur votre page web crée automatiquement un retour à la ligne avant et après. Il suffit d'imaginer tout simplement un bloc. Votre page web sera en fait constituée d'une série de blocs les uns à la suite des autres. Mais vous verrez qu'en plus, il est possible de mettre un bloc à l'intérieur d'un autre, ce qui va augmenter considérablement nos possibilités pour créer le design de notre site !
- **inline** : une balise de type inline se trouve obligatoirement à l'intérieur d'une balise block. Une balise inline ne crée pas de retour à la ligne, le texte qui se trouve à l'intérieur s'écrit donc à la suite du texte précédent, sur la même ligne (c'est pour cela que l'on parle de balise « en ligne »).

Depuis HTML5, la catégorisation des différents éléments est un peu plus complexe que cela. Cependant, cette petite simplification va vous permettre de bien comprendre la différence entre le concept de "bloc" et le concept de "en ligne". 

Pour bien visualiser le concept, voici en figure suivante un petit schéma que je vous ai concocté.



Différence entre une balise inline et une balise block

8 Sur fond bleu, vous avez tout ce qui est de type block.

9 Sur fond jaune, vous avez tout ce qui est de type inline.

Comme vous pouvez le voir, les blocs sont les uns en-dessous des autres. On peut aussi les imbriquer les uns à l'intérieur des autres (souvenez-vous, nos blocs<section>contiennent par exemple des blocs<aside>!).

La balise inline<a>, elle, se trouve à l'intérieur d'une balise block et le texte vient s'insérer sur la même ligne.

Quelques exemples

Afin de mieux vous aider à assimiler quelles balises sont inline et quelles balises sont block, voici un petit tableau dressant la liste de quelques balises courantes.

<p>	
<footer>	
<h1>	<mark>
<h2>	<a>
<article>	

... ...

Ce tableau n'est pas complet, loin de là. Si vous voulez avoir la liste complète des balises qui existent et savoir si elles sont de type inline ou block, reportez-vous à l'[annexe donnant la liste des balises HTML](#).

Les balises universelles

Vous les connaissez déjà car je vous les ai présentées il y a quelques chapitres. Ce sont des balises qui n'ont aucun sens particulier (contrairement à `<p>` qui veut dire « paragraphe », `` « important », etc.).

Le principal intérêt de ces balises est que l'on peut leur appliquer une `class` (ou `un_id`) pour le CSS quand aucune autre balise ne convient.

Il existe deux balises génériques et, comme par hasard, la seule différence entre les deux est que l'une d'elle est inline et l'autre est block :

- ``(**inline**) ;
- `<div></div>`(**block**).

Respectez la sémantique !

Les balises universelles sont « pratiques » dans certains cas, certes, mais attention à ne pas en abuser. Je tiens à vous avertir de suite : beaucoup de webmasters mettent des `<div>` et des `` trop souvent et oublient que d'autres balises plus adaptées existent.

Voici deux exemples :

5 **Exemple d'un span inutile** :``. Je ne devrais jamais voir ceci dans un de vos codes alors qu'il existe la balise `` qui sert à indiquer l'importance !

6 **Exemple d'un div inutile** :`<div class="titre">`. Ceci est complètement absurde puisqu'il existe des balises faites spécialement pour les titres (`<h1>`, `<h2>`...).

Oui, vous allez me dire qu'au final le résultat (visuel) est le même. Je suis tout à fait d'accord. Mais les balises génériques n'apportent aucun sens à la page et ne peuvent pas être comprises par l'ordinateur. Utilisez toujours d'autres balises plus adaptées quand c'est possible. Google lui-même le conseille pour vous aider à améliorer la position de vos pages au sein de ses résultats de recherche !

Les dimensions

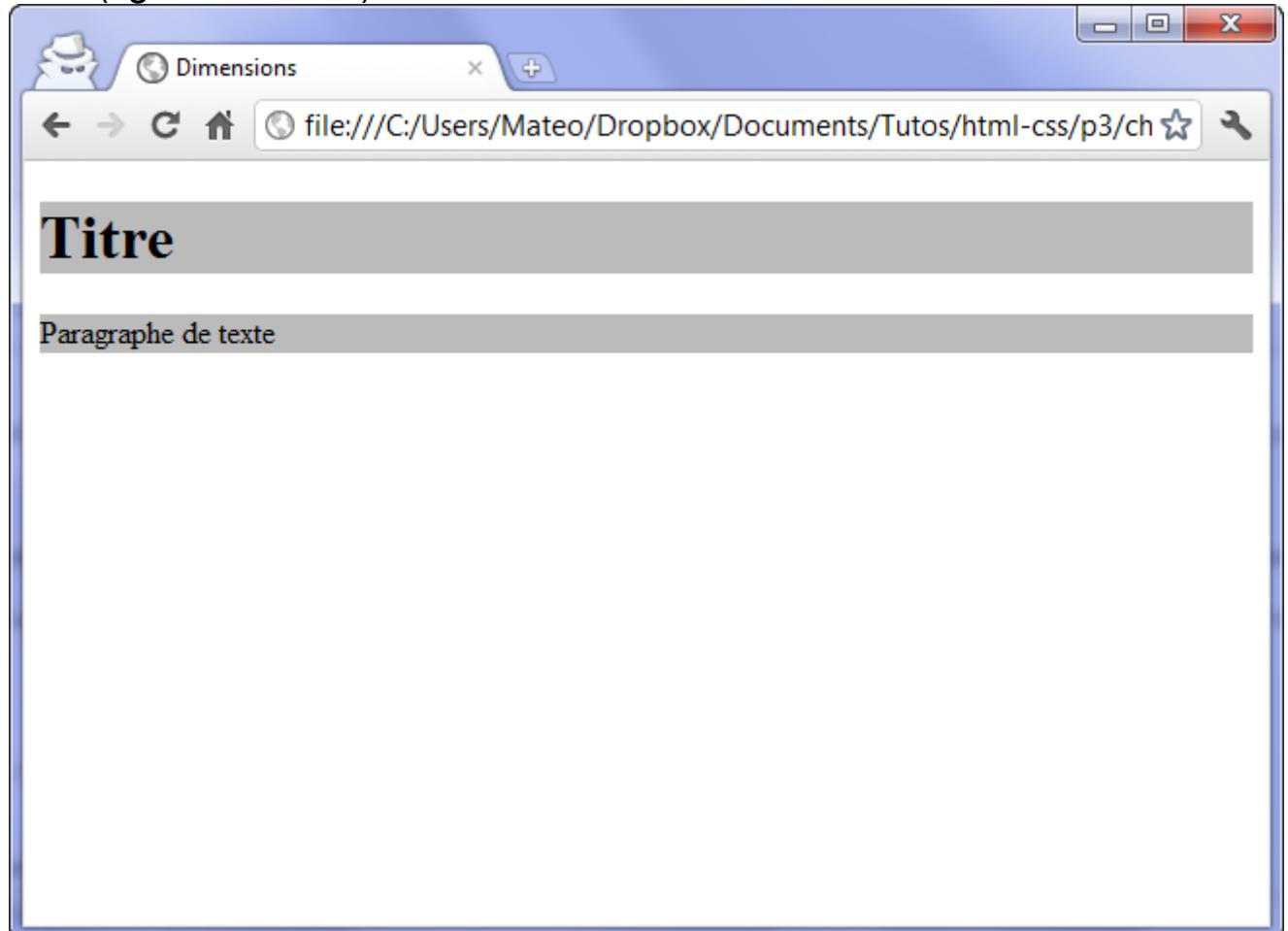
Nous allons ici travailler uniquement sur des balises de type block.
Pour commencer, intéressons-nous à la taille des blocs.

Contrairement à un inline, un bloc a des dimensions précises. Il possède une largeur et une hauteur. Ce qui fait, ô surprise, qu'on dispose de deux propriétés CSS :

- `width`: c'est la largeur du bloc. À exprimer en pixels (px) ou en pourcentage (%).
- `height`: c'est la hauteur du bloc. Là encore, on l'exprime soit en pixels (px), soit en pourcentage (%).

Pour être exact, `width` et `height` représentent la largeur et la hauteur du contenu des blocs. Si le bloc a des marges (on va découvrir ce principe un peu plus loin), celles-ci s'ajouteront à la largeur et la hauteur.

Par défaut, un bloc prend 100% de la largeur disponible. On peut le vérifier en appliquant à nos blocs des bordures ou une couleur de fond (figure suivante).

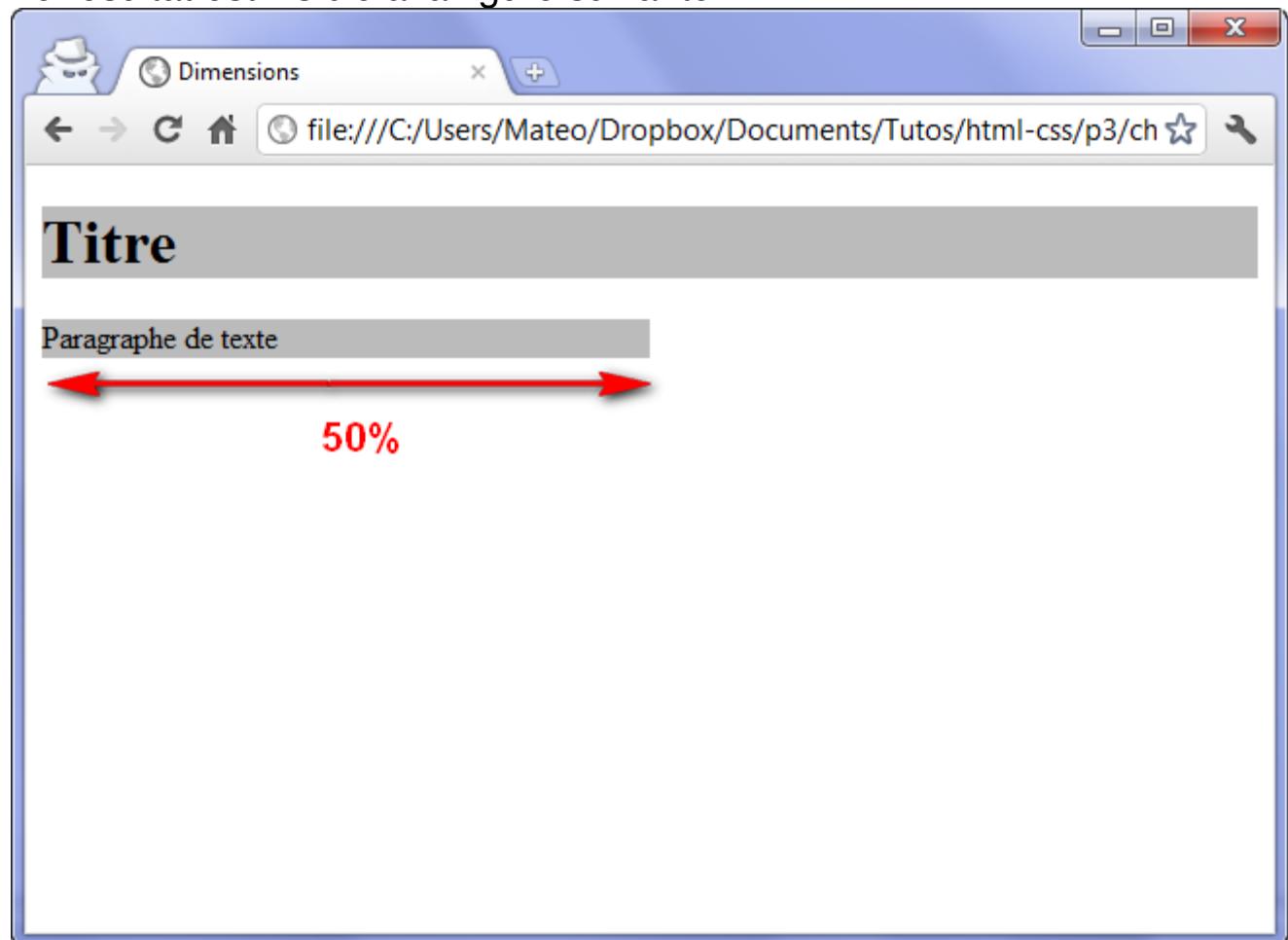


Les blocs prennent toute la largeur disponible

Maintenant, rajoutons un peu de CSS afin de modifier la largeur des paragraphes. Le CSS suivant dit : « Je veux que tous mes paragraphes aient une largeur de 50% ».

```
p  
{  
    width: 50%;  
}
```

Le résultat est visible à la figure suivante.



Un paragraphe de 50% de largeur

Les pourcentages seront utiles pour créer un design qui s'adapte automatiquement à la résolution d'écran du visiteur.

Toutefois, il se peut que vous ayez besoin de créer des blocs ayant une dimension précise en pixels :

```
p
```

```
{  
    width: 250px;  
}
```

Minimum et maximum

On peut demander à ce qu'un bloc ait des dimensions minimales et maximales. C'est très pratique car cela nous permet de définir des dimensions « limites » pour que notre site s'adapte aux différentes résolutions d'écran de nos visiteurs :

- `min-width`: largeur minimale ;
- `min-height`: hauteur minimale ;
- `max-width`: largeur maximale ;
- `max-height`: hauteur maximale.

Par exemple, on peut demander à ce que les paragraphes occupent 50% de la largeur et exiger qu'il fassent au moins 400 pixels de large dans tous les cas :

```
p {  
    width: 50%;  
    min-width: 400px;  
}
```

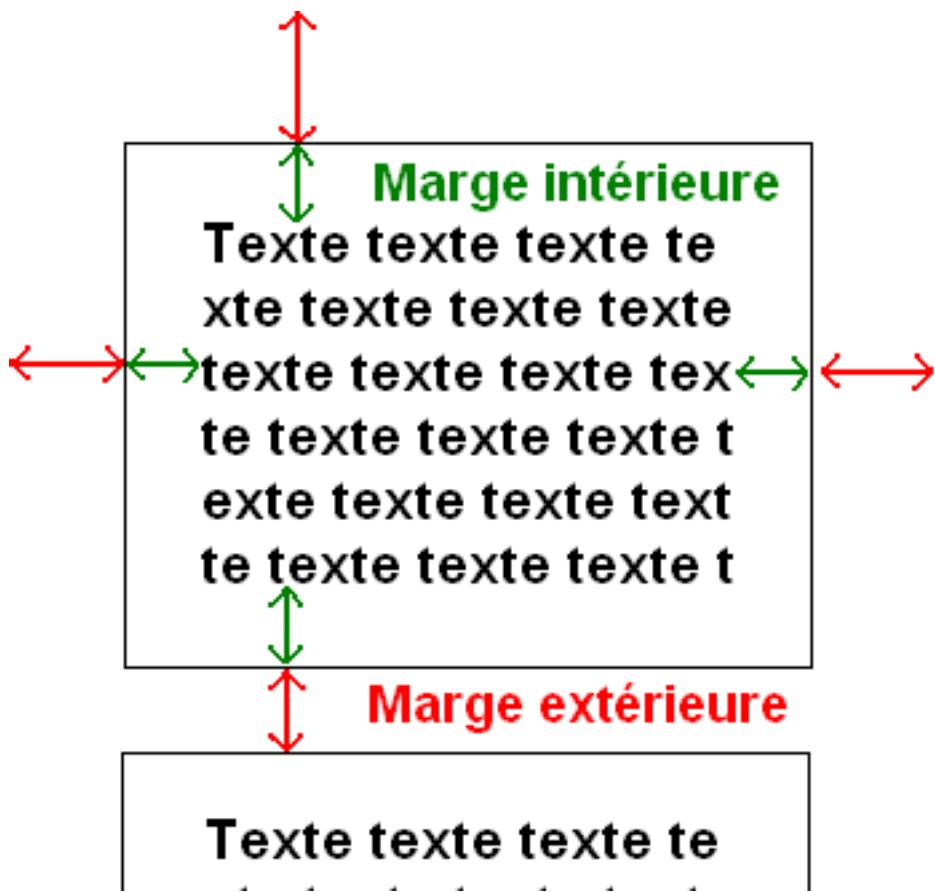
Observez le résultat en modifiant la largeur de la fenêtre de votre navigateur. Vous allez voir que, si celle-ci est trop petite, le paragraphe se force à occuper au moins 400 pixels de largeur.

Les marges

Il faut savoir que tous les blocs possèdent des marges. Il existe deux types de marges :

- les marges intérieures ;
- les marges extérieures.

Regardez bien le schéma qui se trouve à la figure suivante.



Sur ce bloc, j'ai mis une bordure pour qu'on repère mieux ses frontières.

- L'espace entre le texte et la bordure est la marge intérieure (en vert).
- L'espace entre la bordure et le bloc suivant est la marge extérieure (en rouge).

En CSS, on peut modifier la taille des marges avec les deux propriétés suivantes :

- `padding`: indique la taille de la marge intérieure. À exprimer en général en pixels (px).
- `margin`: indique la taille de la marge extérieure. Là encore, on utilise le plus souvent des pixels.

Les balises de type inline possèdent également des marges. Vous pouvez donc aussi essayer ces manipulations sur ce type de balises.

Pour bien voir les marges, prenons deux paragraphes auxquels j'applique simplement une petite bordure (figure suivante) :

```
{  
width: 350px;  
  
border: 1px solid black;  
  
text-align: justify;
```



Marges par défaut sur les paragraphes

Comme vous pouvez le constater, il n'y a par défaut pas de marge intérieure (padding). En revanche, il y a une marge extérieure (margin). C'est cette marge qui fait que deux paragraphes ne sont pas collés et qu'on a l'impression de « sauter une ligne ».

Les marges par défaut ne sont pas les mêmes pour toutes les balises de type block. Essayez d'appliquer ce CSS à des balises <div> qui contiennent du texte, par exemple : vous verrez que, dans ce cas, il n'y a par défaut ni marge intérieure, ni marge

extérieure !

Supposons que je veuille rajouter une marge intérieure de 12 px aux paragraphes (figure suivante) :

```
p  
{  
    width: 350px;  
    border: 1px solid black;  
    text-align: justify;  
    padding: 12px; /* Marge intérieure de 12px */
```

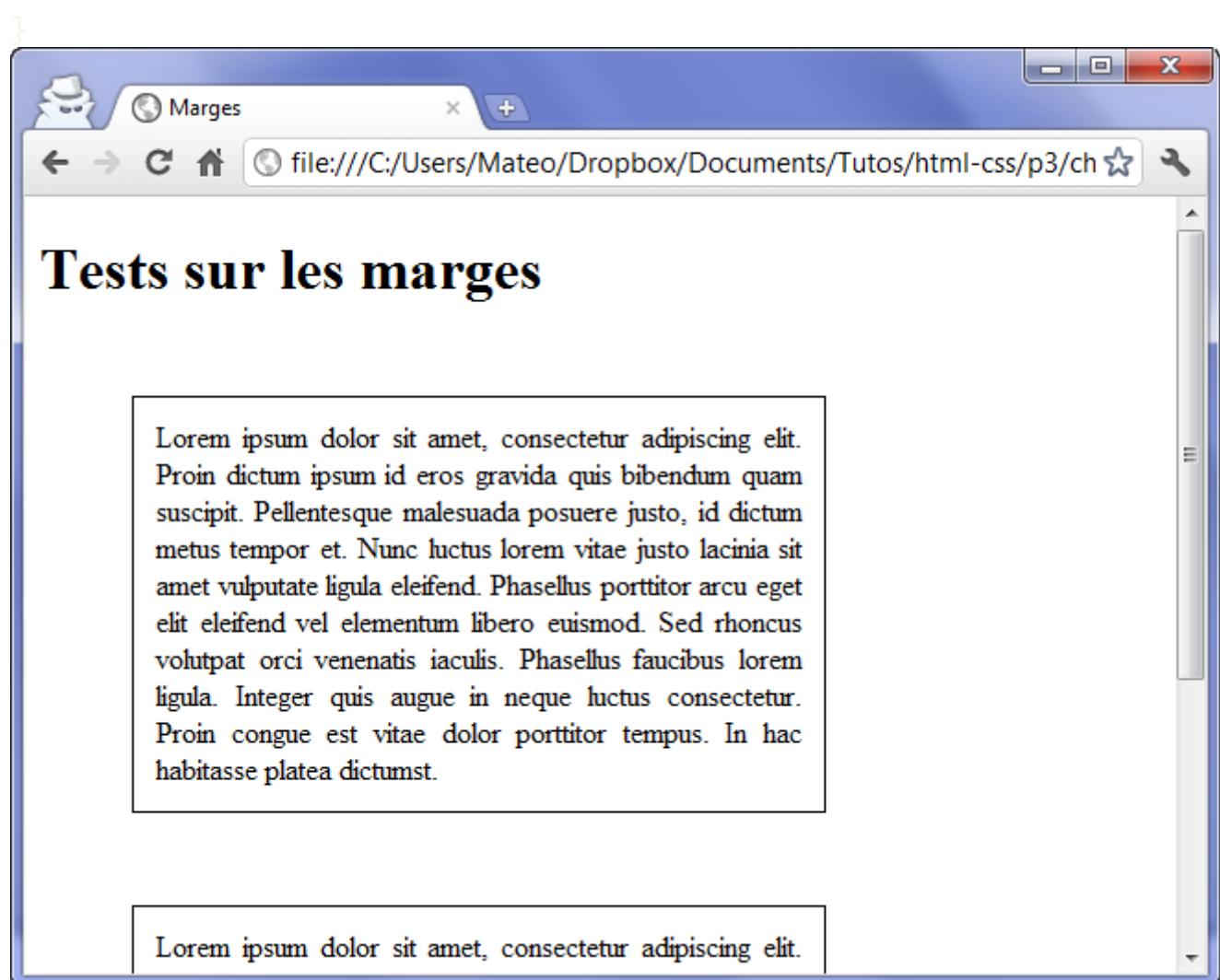


Une marge intérieure ajoutée aux paragraphes

Maintenant, je veux que mes paragraphes soient plus espacés

entre eux. Je rajoute la propriété `margin` pour demander à ce qu'il y ait 50 px de marge entre deux paragraphes (figure suivante) :

```
p  
{  
    width: 350px;  
  
    border: 1px solid black;  
  
    text-align: justify;  
  
    padding: 12px;  
  
    margin: 50px; /* Marge extérieure de 50px */  
}
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin dictum ipsum id eros gravida quis bibendum quam suscipit. Pellentesque malesuada posuere justo, id dictum metus tempor et. Nunc luctus lorem vitae justo lacinia sit amet vulputate ligula eleifend. Phasellus porttitor arcu eget elit eleifend vel elementum libero euismod. Sed rhoncus volutpat orci venenatis iaculis. Phasellus faucibus lorem ligula. Integer quis augue in neque luctus consectetur. Proin congue est vitae dolor porttitor tempus. In hac habitasse platea dictumst.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

[Une marge extérieure ajoutée aux paragraphes](#)

Mais ??? Une marge s'est rajoutée à gauche aussi !

Eh oui, `margin`(comme `padding`d'ailleurs) s'applique aux quatre côtés du bloc.

Si vous voulez spécifier des marges différentes en haut, en bas, à gauche et à droite, il va falloir utiliser des propriétés plus précises... Le principe est le même que pour la propriété `border`, vous allez voir !

En haut, à droite, à gauche, en bas... Et on recommence !

L'idéal serait que vous reteniez les termes suivants en anglais :

- top : haut ;
- bottom : bas ;
- left : gauche ;
- right : droite.

Ainsi, vous pouvez retrouver toutes les propriétés de tête.

Je vais quand même vous faire la liste des propriétés pour `margin` et `padding`, histoire que vous soyez sûrs que vous avez compris le principe.

Voici la liste pour `margin`:

- `margin-top`: marge extérieure en haut ;
- `margin-bottom`: marge extérieure en bas ;
- `margin-left`: marge extérieure à gauche ;
- `margin-right`: marge extérieure à droite.

Et la liste pour `padding`:

- `padding-top`: marge intérieure en haut ;
- `padding-bottom`: marge intérieure en bas ;
- `padding-left`: marge intérieure à gauche ;
- `padding-right`: marge intérieure à droite.

Il y a d'autres façons de spécifier les marges avec les propriétés `margin` et `padding`. Par exemple :

`margin: 2px 0 3px 1px;` signifie « 2 px de marge en haut, 0 px à droite (le px est facultatif dans ce cas), 3 px en bas, 1 px à gauche ».

Autre notation raccourcie : `margin: 2px 1px;` signifie « 2 px de marge en haut et en bas, 1 px de marge à gauche et à droite ».

Centrer des blocs

Il est tout à fait possible de centrer des blocs. C'est même très pratique pour réaliser un design centré quand on ne connaît pas la résolution du visiteur.

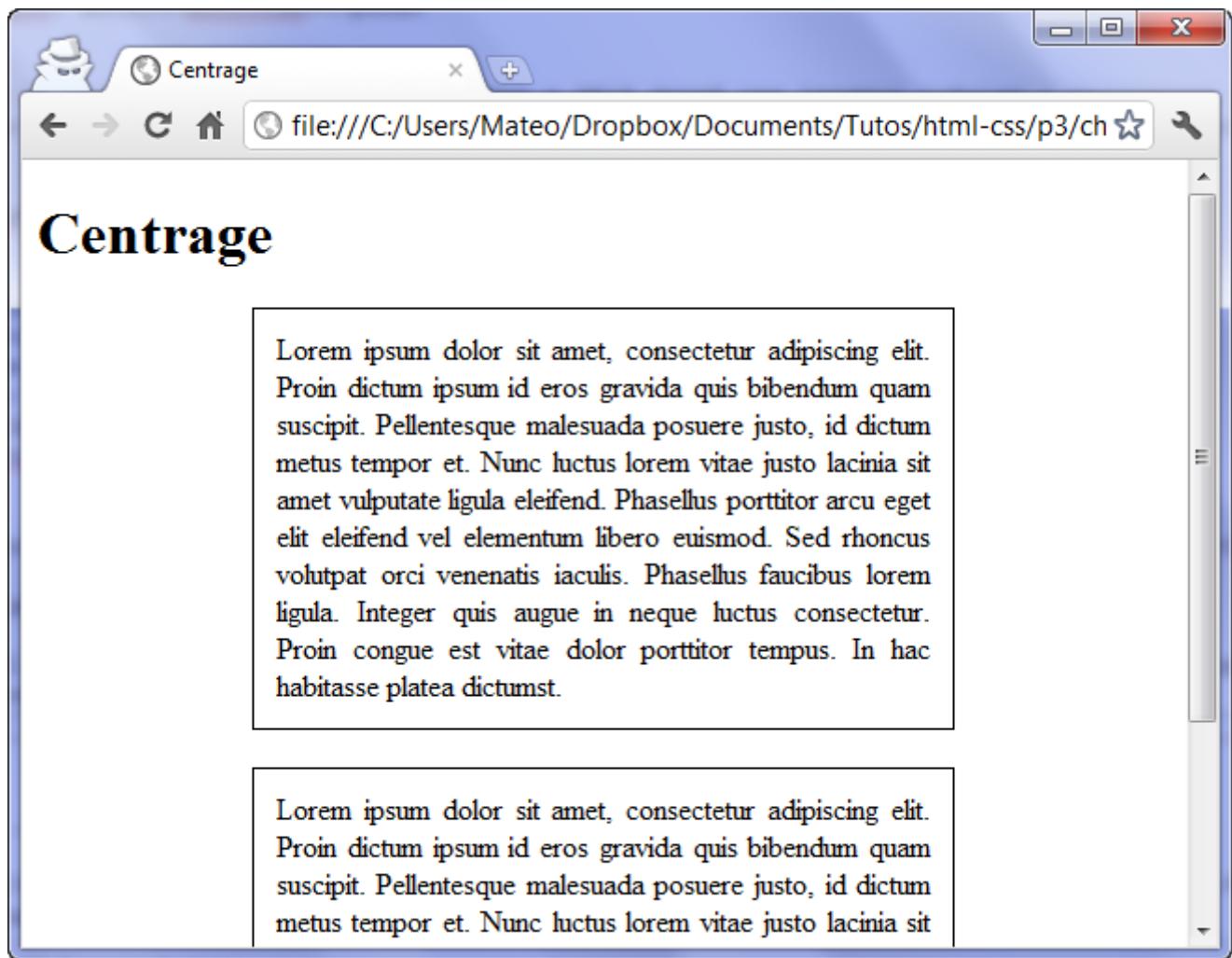
Pour centrer, il faut respecter les règles suivantes :

- donnez une largeur au bloc (avec la propriété `width`) ;
- indiquez que vous voulez des marges extérieures automatiques, comme ceci :`margin: auto;`.

Essayons cette technique sur nos petits paragraphes (lignes 3 et 4)

```
:  
  
p  
{  
    width: 350px; /* On a indiqué une largeur (obligatoire) */  
    margin: auto; /* On peut donc demander à ce que le bloc  
soit centré avec auto */  
    border: 1px solid black;  
    text-align: justify;  
    padding: 12px;  
    margin-bottom: 20px;  
}
```

Et voici le résultat à la figure suivante.



Centrage des paragraphes

Ainsi, le navigateur centre automatiquement nos paragraphes ! Il n'est cependant pas possible de centrer verticalement un bloc avec cette technique. Seul le centrage horizontal est permis.

Quand ça dépasse...

Lorsqu'on commence à définir des dimensions précises pour nos blocs, comme on vient de le faire, il arrive qu'ils deviennent trop petits pour le texte qu'ils contiennent.

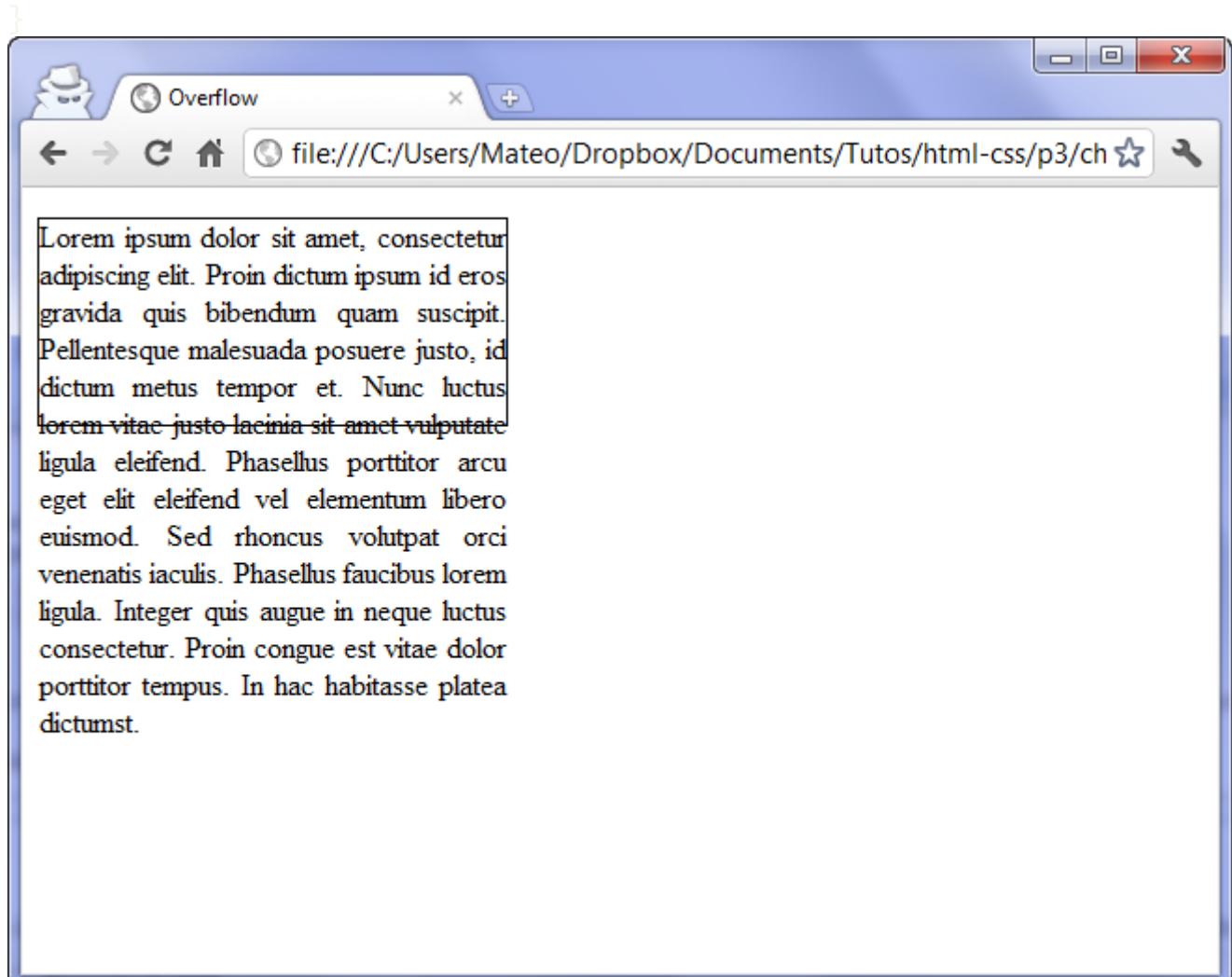
Les propriétés CSS que nous allons voir ici ont justement été créées pour contrôler les dépassements... et décider quoi faire si jamais cela devait arriver.

overflow: couper un bloc

Supposons que vous ayez un long paragraphe et que vous vouliez (pour une raison qui ne regarde que vous) qu'il fasse 250 px de large et 110 px de haut. Ajoutons-lui une bordure et remplissons-le

de texte... à ras-bord (figure suivante) :

```
p {  
    width: 250px;  
    height: 110px;  
    text-align: justify;  
    border: 1px solid black;  
}
```



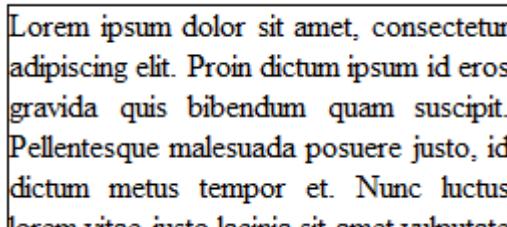
Le texte dépasse du bloc de paragraphe
Horreur ! Le texte dépasse des limites du paragraphe !
Eh oui ! Vous avez demandé des dimensions précises, vous les avez eues ! Mais... le texte ne tient pas à l'intérieur d'un si petit

bloc.

Si vous voulez que le texte ne dépasse pas des limites du paragraphe, il va falloir utiliser la propriété `overflow`. Voici les valeurs qu'elle peut accepter :

- `visible`(par défaut) : si le texte dépasse les limites de taille, il reste visible et sort volontairement du bloc.
- `hidden`: si le texte dépasse les limites, il sera tout simplement coupé. On ne pourra pas voir tout le texte.
- `scroll`: là encore, le texte sera coupé s'il dépasse les limites. Sauf que cette fois, le navigateur mettra en place des barres de défilement pour qu'on puisse lire l'ensemble du texte. C'est un peu comme un cadre à l'intérieur de la page.
- `auto`: c'est le mode « pilote automatique ». En gros, c'est le navigateur qui décide de mettre ou non des barres de défilement (il n'en mettra que si c'est nécessaire). C'est la valeur que je conseille d'utiliser le plus souvent.

Avec `overflow: hidden`, le texte est donc coupé (on ne peut pas voir la suite), comme sur la figure suivante.

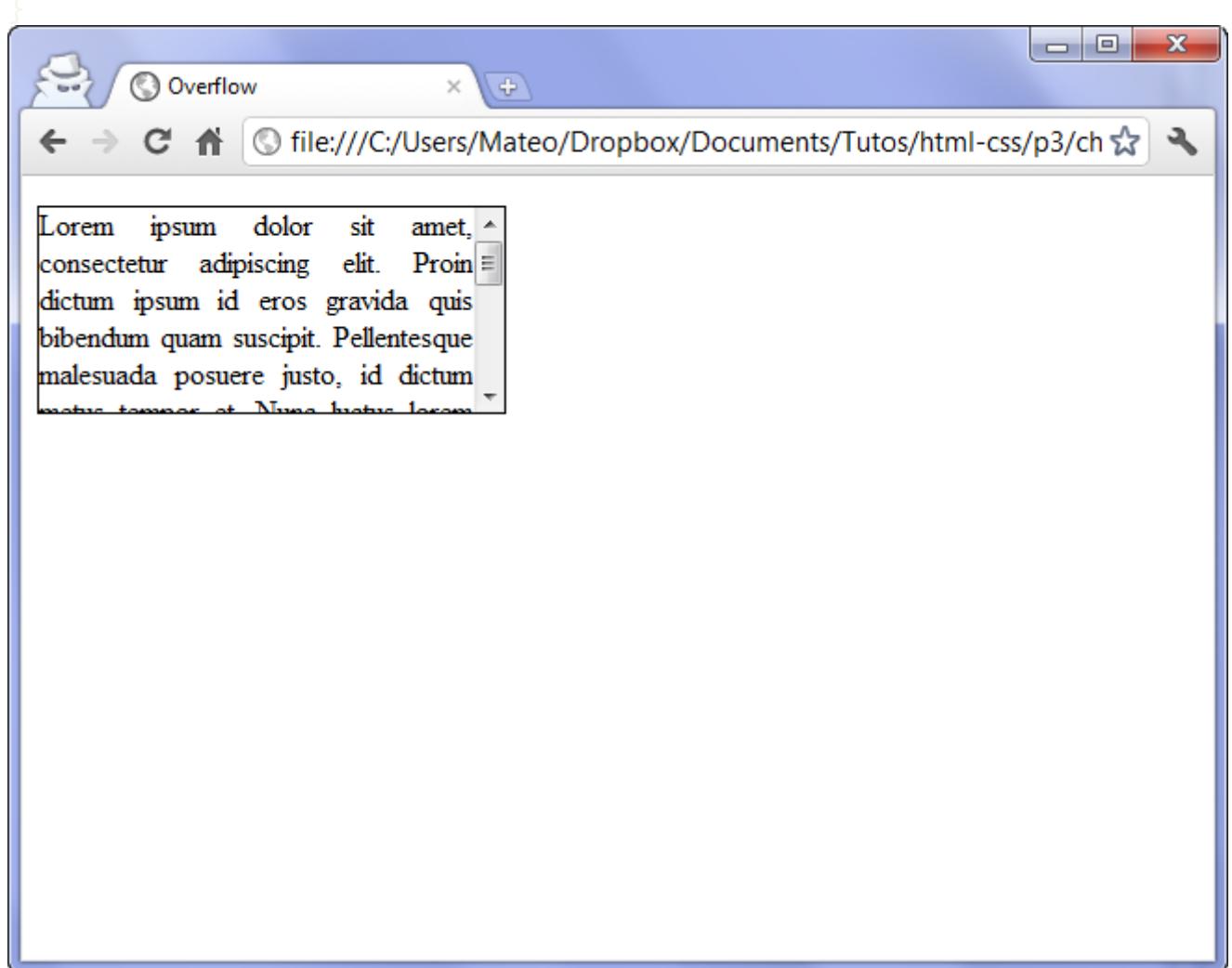


Le texte est coupé aux limites du paragraphe

Essayons maintenant `overflow: auto`; avec le code CSS suivant (résultat à la figure suivante) :

```
p {  
    width: 250px;  
    height: 110px;  
    text-align: justify;  
    border: 1px solid black;
```

```
overflow: auto;
```



Des barres de défilement sont ajoutées au paragraphe

Eurêka ! Des barres de défilement nous permettent maintenant de consulter le contenu qui n'était pas visible.

Il existe une ancienne balise HTML, <iframe>, qui donne à peu près le même résultat. Cependant, l'usage de cette balise est déconseillé aujourd'hui. Elle permet de charger tout le contenu d'une autre page HTML au sein de votre page.

word-wrap: couper les textes trop larges

Si vous devez placer un mot très long dans un bloc, qui ne tient pas dans la largeur, vous allez adorer word-wrap. Cette propriété permet de forcer la césure des très longs mots (généralement des adresses un peu longues).

La figure suivante représente ce que l'on peut avoir quand on écrit une URL un peu longue dans un bloc.

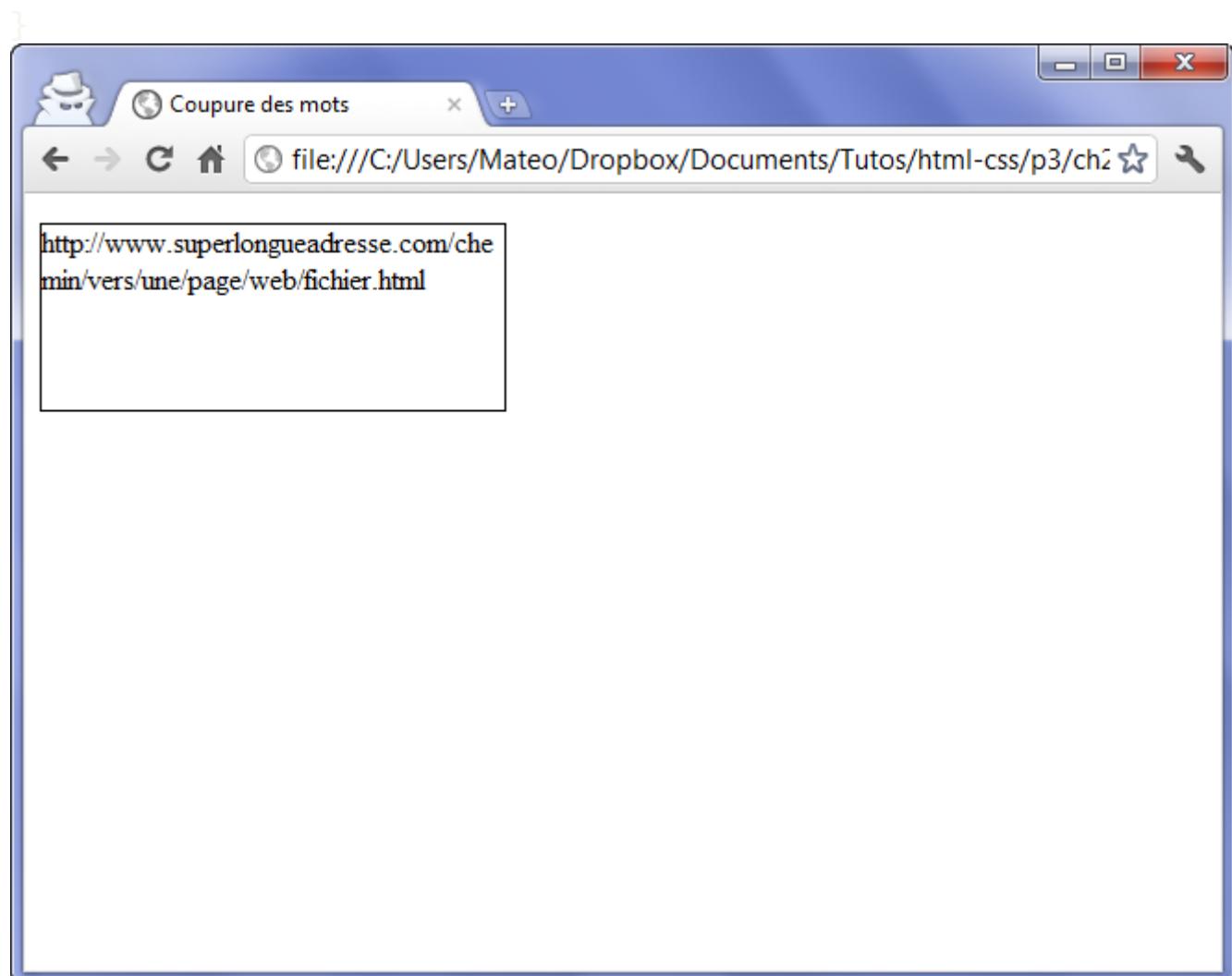
`http://www.superlongueadresse.com/chemin/vers/une/page/web/fichier.html`

Le texte déborde en largeur

L'ordinateur ne sait pas « couper » l'adresse car il n'y a ni espace, ni tiret. Il ne sait pas faire la césure.

Avec le code suivant, la césure sera forcée dès que le texte risque de dépasser (figure suivante).

```
p {  
    word-wrap: break-word;  
}
```



Le texte est coupé pour ne pas déborder
Je conseille d'utiliser cette fonctionnalité dès qu'un bloc est susceptible de contenir du texte saisi par des utilisateurs (par exemple sur les forums de votre futur site). Sans cette astuce, on peut « casser » facilement le design d'un site (en écrivant par exemple une longue suite de «aaaaaaaaaaa »).

En résumé

- On distingue deux principaux types de balises en HTML :
 - Le type block (`<p>`,`<h1>`...) : ces balises créent un retour à la ligne et occupent par défaut toute la largeur disponible. Elles se suivent de haut en bas.
 - Le type inline (`<a>`,``...) : ces balises délimitent du texte au milieu d'une ligne. Elles se suivent de gauche à droite.
- On peut modifier la taille d'une balise de type block avec les propriétés CSS `width(largeur)` et `height(hauteur)`.
- On peut définir des minima et maxima autorisés pour la largeur et la hauteur `:min-width,max-width,min-height,max-height`.
- Les éléments de la page disposent chacun de marges intérieures (`padding`) et extérieures (`margin`).

S'il y a trop de texte à l'intérieur d'un bloc de dimensions fixes, il y a un risque de débordement. Dans ce cas, il peut être judicieux de rajouter des barres de défilement avec la propriété `overflow` ou de forcer la césure avec `word-wrap`.

La mise en page avec Flexbox

Allez, il est temps d'apprendre à mettre en page notre site. Vous savez ? Placer un en-tête, des menus sur le côté, choisir où apparaît une information, etc. C'est la pièce manquante du puzzle pour que nous puissions enfin créer notre site ! 😊

Il y a plusieurs façons de mettre en page un site. Au fil du temps, plusieurs techniques ont existé :

- Au début, les webmasters utilisaient des tableaux HTML pour faire la mise en page (berk)
- Puis, CSS est apparu et on a commencé à faire une mise en page à l'aide de la propriété `float` (bof)

- Cette technique avait des inconvénients. Une autre, plus pratique, a consisté à créer des éléments de type `inline-block` sur la page (mouais)
- Aujourd'hui, une bien meilleure technique encore existe : **Flexbox** ! Elle permet toutes les folies (ou presque 😊) et c'est celle que je vous recommande d'utiliser si vous en avez la possibilité, lorsque vous créez un nouveau site. [Flexbox est désormais reconnu par tous les navigateurs récents !](#)

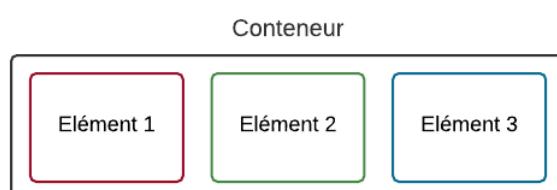
Nous découvrirons donc le fonctionnement de Flexbox dans ce chapitre. Pour ceux qui ont besoin d'informations sur les autres techniques de mise en page plus anciennes, je vous invite à consulter le chapitre "Quelques autres techniques de mise en page". Cela peut toujours vous être utile.

Un conteneur, des éléments

Le principe de la mise en page avec Flexbox est simple : vous définissez un conteneur, et à l'intérieur vous placez plusieurs éléments. Imaginez un carton dans lequel vous rangez plusieurs objets : c'est le principe !

Sur une même page web, vous pouvez sans problème avoir plusieurs conteneurs (plusieurs cartons si vous préférez 😊). Ce sera à vous d'en créer autant que nécessaire pour obtenir la mise en page que vous voulez.

Commençons par étudier le fonctionnement d'un carton (ehh pardon, d'un conteneur).



Un conteneur et ses éléments

Le conteneur est une balise HTML, et les éléments sont d'autres balises HTML à l'intérieur :

```
div id="conteneur"
```

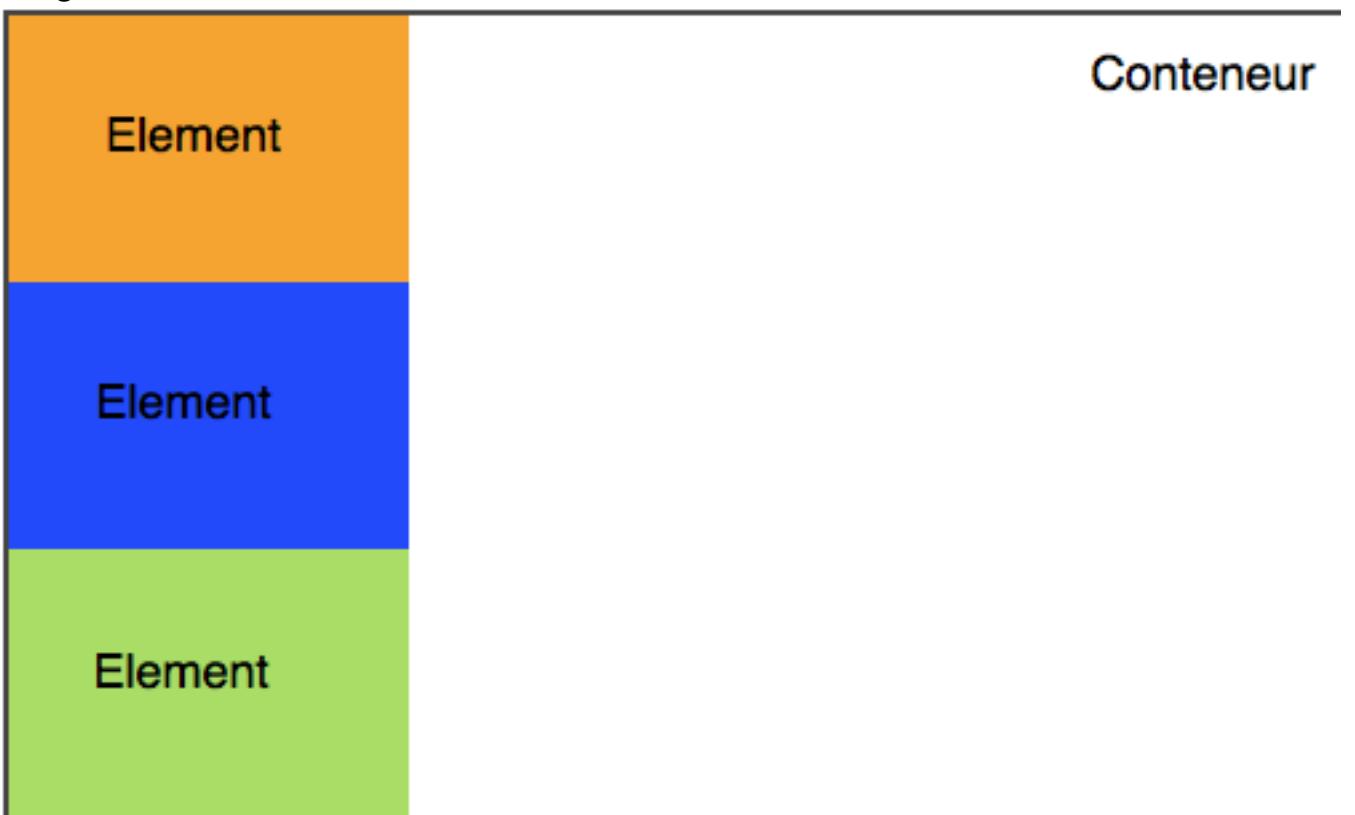
```
div class="element">Elément 1
```

```
div class="element" Élément 2 div  
div class="element" Élément 3 div  
div
```

Ok, jusque-là, vous devriez suivre. 😊

Mais si je fais ça, par défaut, mes éléments vont se mettre les uns en-dessous des autres non ? Ce sont des blocs après tout !

Oui tout à fait. Si je mets une bordure au conteneur, une taille et une couleur de fond aux éléments, on va vite voir comment ils s'organisent :



Par défaut, les blocs se placent les uns en-dessous des autres. Rien de bien nouveau, c'est le comportement normal dont nous avons l'habitude.

Soyez flex !

Découvrons maintenant Flexbox. Si je mets une (une seule !) propriété CSS, tout change. Cette propriété, c'est `flex`, et je l'applique au conteneur :

```
#conteneur
```

```
{  
  display: flex;  
}
```

... alors les blocs se placent par défaut côté à côté. Magique !



Un coup de flex et les blocs se positionnent côté à côté !

La direction

Flexbox nous permet d'agencer ces éléments dans le sens que l'on veut. Avec `flex-direction`, on peut les positionner verticalement ou encore les inverser. Il peut prendre les valeurs suivantes :

- `row` : organisés sur une ligne (par défaut)
- `column` : organisés sur une colonne
- `row-reverse` : organisés sur une ligne, mais en ordre inversé
- `column-reverse` : organisés sur une colonne, mais en ordre inversé

Exemple :

```
#conteneur
```

```
{  
  display: flex;  
  
  flex-direction: column;  
}
```



Les éléments sont disposés en colonne

Mais mais... c'est pareil qu'au début non ? On avait ce résultat sans Flexbox après tout !

C'est vrai. Mais maintenant que nos éléments sont flex, ils ont tout un tas d'autres propriétés utiles que nous allons voir juste après, on va y revenir.

Essayez aussi de tester l'ordre inversé pour voir :

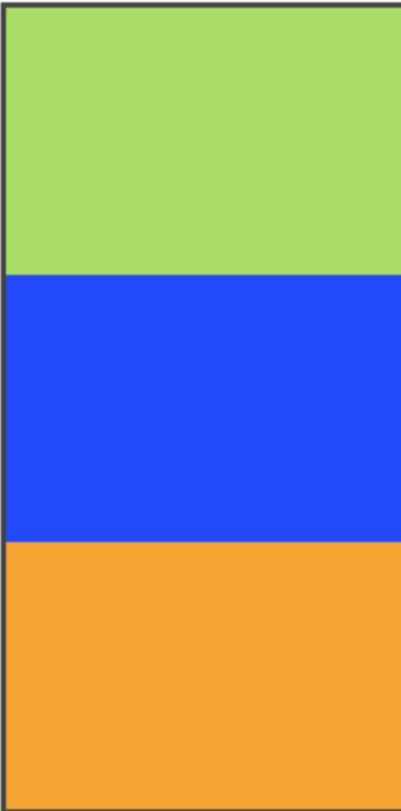
```
#conteneur
```

```
{
```

```
  display: flex;
```

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

```
}
```



Les éléments sont en colonne... dans l'ordre inverse !

Regardez bien la différence : les blocs sont maintenant dans l'ordre inverse ! Je n'ai pas du tout changé le code HTML qui reste le même depuis le début.

Le retour à la ligne

Par défaut, les blocs essaient de rester sur la même ligne s'ils n'ont pas la place (ce qui peut provoquer des bugs de design parfois). Si vous voulez, vous pouvez demander à ce que les blocs aillent à la ligne lorsqu'ils n'ont plus la place avec `flex-wrap` qui peut prendre ces valeurs :

- 10 `nowrap` : pas de retour à la ligne (par défaut)
- 11 `wrap` : les éléments vont à la ligne lorsqu'il n'y a plus la place
- 12 `wrap-reverse` : les éléments vont à la ligne lorsqu'il n'y a plus la place en sens inverse

Exemple :

```
#conteneur
```

```
{
```

```
  display: flex;
```

```
flex-wrap: wrap;
```

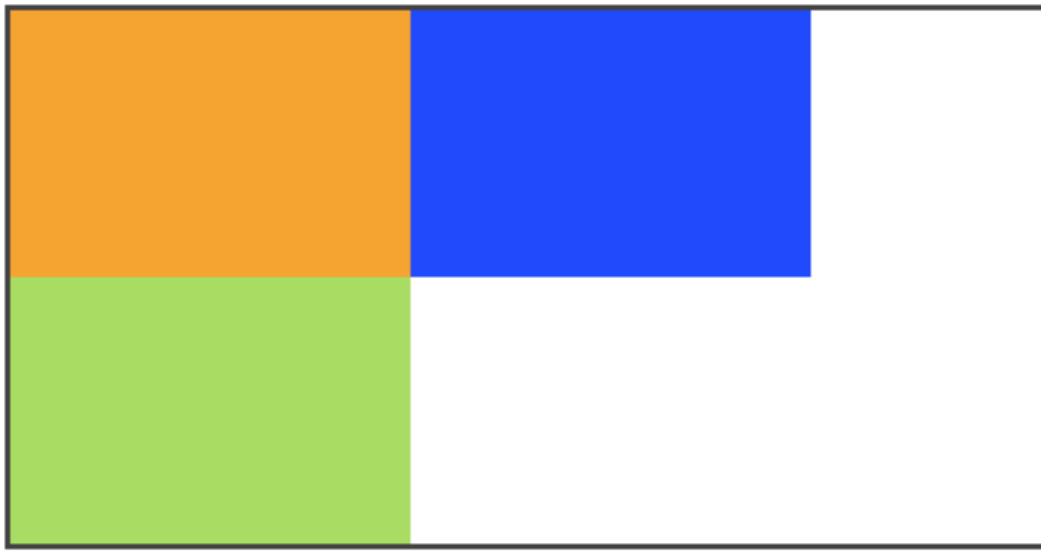
```
}
```

Voici l'effet que prennent les différentes valeurs sur une même illustration :

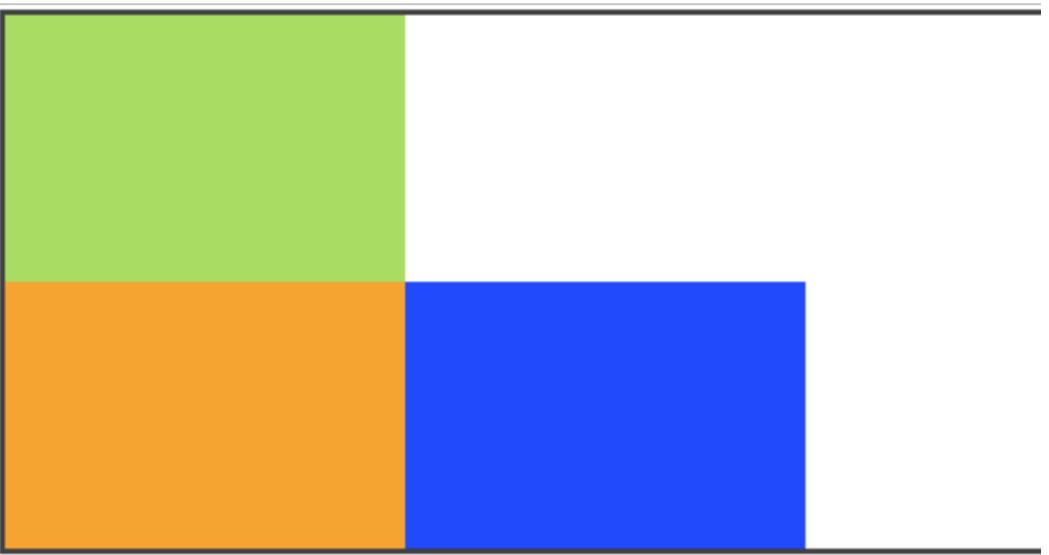
`nowrap`



`wrap`



`wrap-reverse`



Gestion du retour à la ligne avec `flex-wrap`

Alignez-les !

Reprenons. Les éléments sont organisés soit horizontalement (par défaut), soit verticalement. Cela définit ce qu'on appelle **l'axe principal**. Il y a aussi un axe secondaire (cross axis) :

- Si vos éléments sont organisés horizontalement, l'axe secondaire est l'axe vertical.
- Si vos éléments sont organisés verticalement, l'axe secondaire est l'axe horizontal.

Pourquoi je vous raconte ça ? Parce que nous allons découvrir comment aligner nos éléments sur l'axe principal et sur l'axe secondaire.

Aligner sur l'axe principal

Pour faire simple, partons sur des éléments organisés horizontalement (c'est le cas par défaut).

Pour changer leur alignement, on va utiliser `justify-content`, qui peut prendre ces valeurs :

7 `flex-start` : alignés au début (par défaut)

8 `flex-end` : alignés à la fin

9 `center` : alignés au centre

10 `space-between` : les éléments sont étirés sur tout l'axe (il y a de l'espace entre eux)

11 `space-around` : idem, les éléments sont étirés sur tout l'axe, mais ils laissent aussi de l'espace sur les extrémités

Par exemple :

```
#conteneur
```

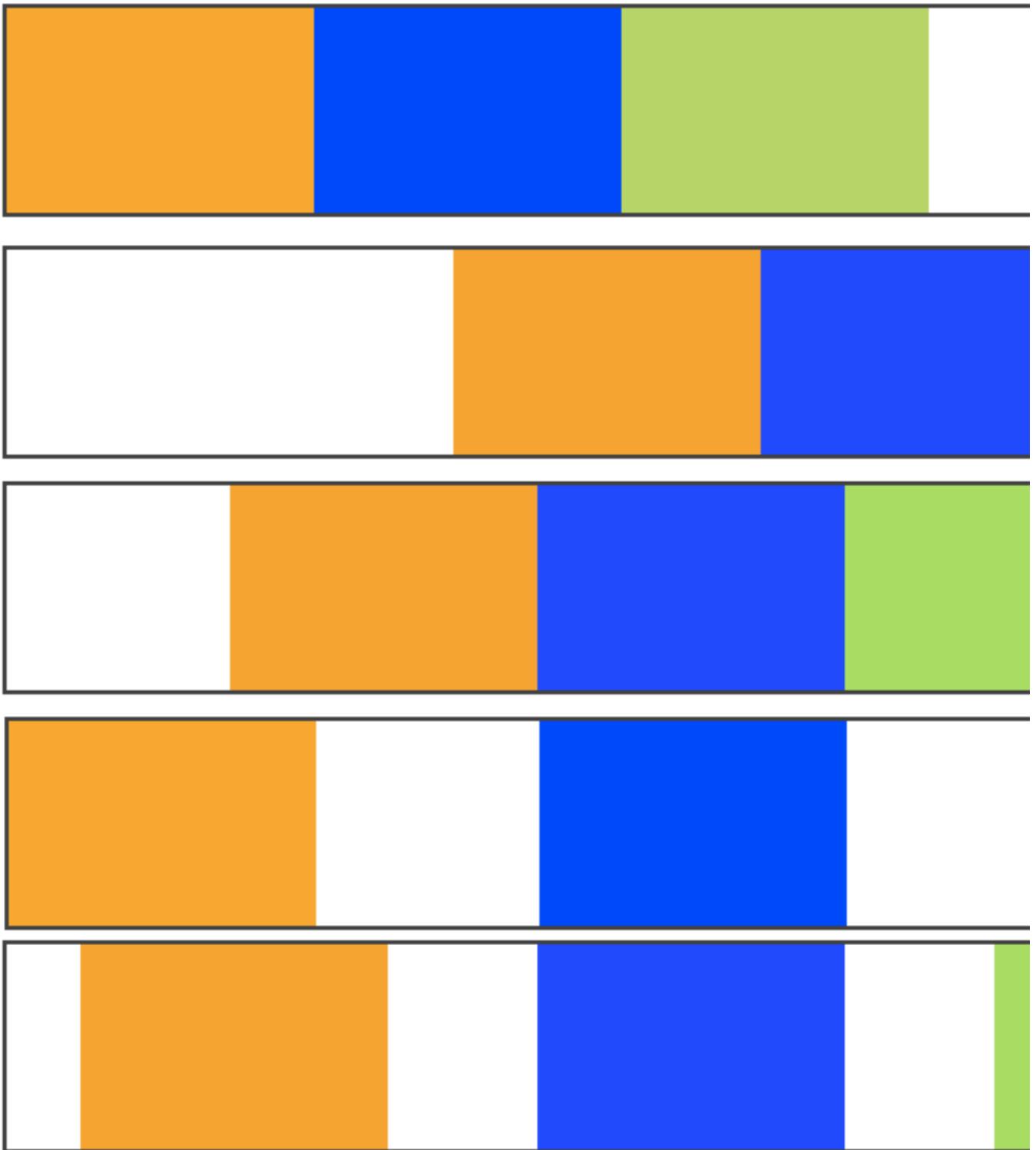
```
{
```

```
  display: flex;
```

```
  justify-content: space-around;
```

```
}
```

Le mieux est encore de tester toutes les valeurs possibles pour voir ce que ça donne, vous pensez pas ? 😊



Les différentes valeurs possibles pour l'alignement avec `justify-content` Vous voyez comment les éléments s'alignent différemment selon les cas ? Avec une simple propriété, on peut intelligemment agencer nos éléments comme on veut ! 😊 Maintenant, voici ce qu'il faut bien comprendre : **ça marche aussi si vos éléments sont dans une direction verticale.** Dans ce cas, l'axe vertical devient l'axe principal, et `justify-content` s'applique

aussi :

```
#conteneur
```

```
{
```

```
  display: flex;
```

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

```
  justify-content: center;
```

```
  height: 350px; /* Un peu de hauteur pour que les éléments  
aient la place de bouger */
```

```
}
```



Avec une direction verticale (column), le centrage fonctionne de la même façon cette fois en hauteur !

Essayez, à vous de jouer ! 😊
Aligner sur l'axe secondaire

Comme je vous disais, si nos éléments sont placés dans une direction horizontale (ligne), l'axe secondaire est... vertical. Et inversement, si nos éléments sont dans une direction verticale (colonne), l'axe secondaire est horizontal.

Avec `align-items`, nous pouvons changer leur alignement sur l'axe secondaire. Il peut prendre ces valeurs :

- `stretch` : les éléments sont étirés sur tout l'axe (valeur par défaut)
- `flex-start` : alignés au début
- `flex-end` : alignés à la fin
- `center` : alignés au centre
- `baseline` : alignés sur la ligne de base (semblable à `flex-start`)

Pour ces exemples, nous allons partir du principe que nos éléments sont dans une direction horizontale (mais n'hésitez pas à tester aussi dans la direction verticale !).

#conteneur

```
{  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}
```



Un alignement sur l'axe secondaire avec align-items nous permet de centrer complètement l'élément dans le conteneur !

St Graal du développeur web, le centrage vertical et horizontal peut d'ailleurs être obtenu encore plus facilement. Dites que votre conteneur est une flexbox et établissez des marges automatiques sur les éléments à l'intérieur. C'est tout ! Essayez !

#conteneur

{

 display: flex;

}

.element

{

```
margin: auto;
```

```
}
```

Aligner un seul élément

Il est possible de faire une exception pour un seul des éléments sur l'axe secondaire avec `align-self` :

```
#conteneur
```

```
{
```

```
  display: flex;
```

```
  flex-direction: row;
```

```
  justify-content: center;
```

```
  align-items: center;
```

```
}
```

```
.element:nth-child(2) /* On prend le deuxième bloc élément */
```

```
{
```

```
  background-color: blue;
```

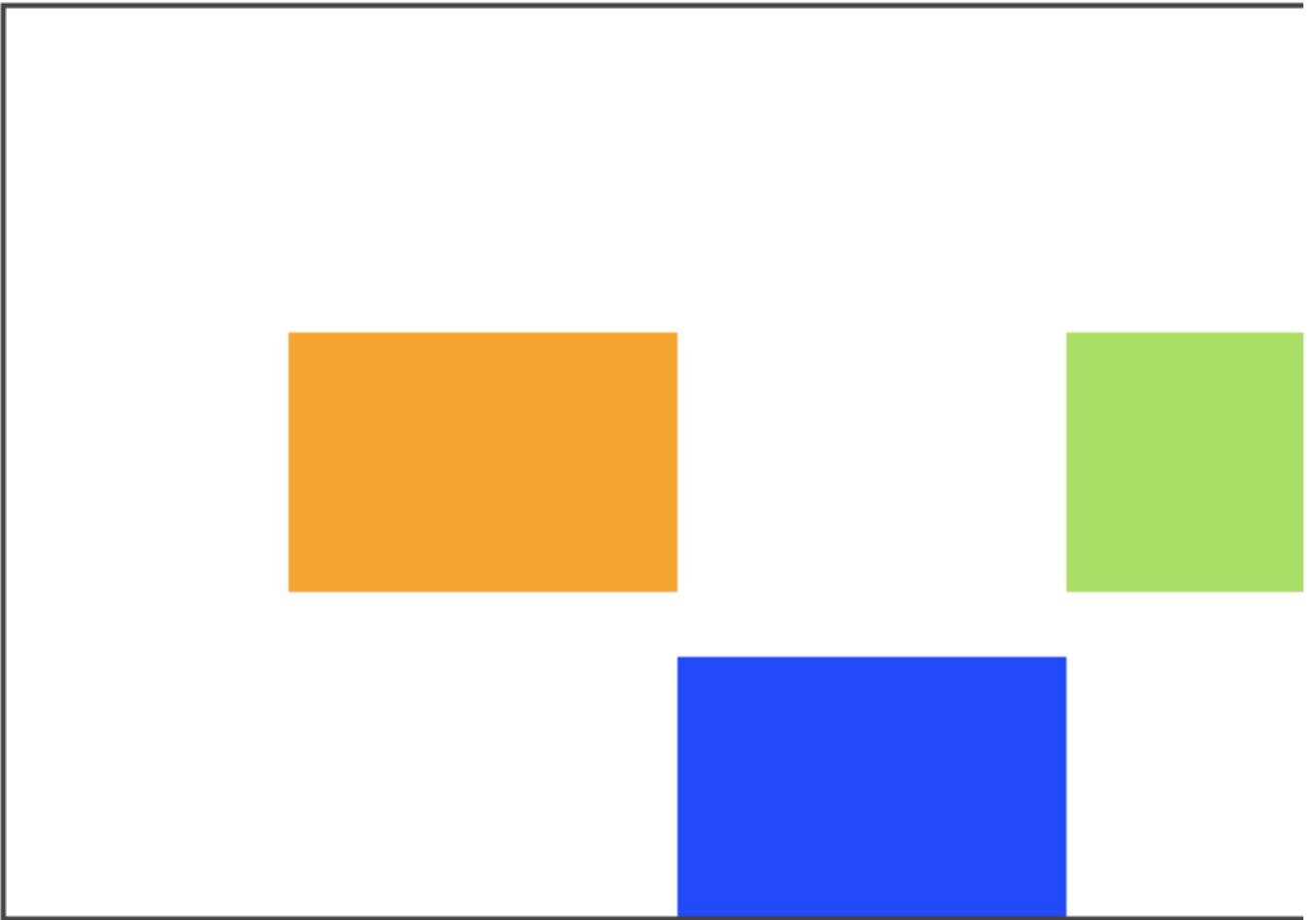
```
  align-self: flex-end; /* Seul ce bloc sera aligné à la fin
```

```
*/
```

```
}
```

```
/* ... */
```

Résultat :



Un élément aligné différemment des autres avec align-self. Tiens je crois que j'ai dessiné une tête en pixel art !

Répartir plusieurs lignes

Si vous avez plusieurs lignes dans votre Flexbox, vous pouvez choisir comment celles-ci seront réparties avec align-content . Cette propriété n'a aucun effet s'il n'y a qu'une seule ligne dans la Flexbox.

Prenons donc un cas de figure où nous avons plusieurs lignes. Je vais rajouter des éléments :

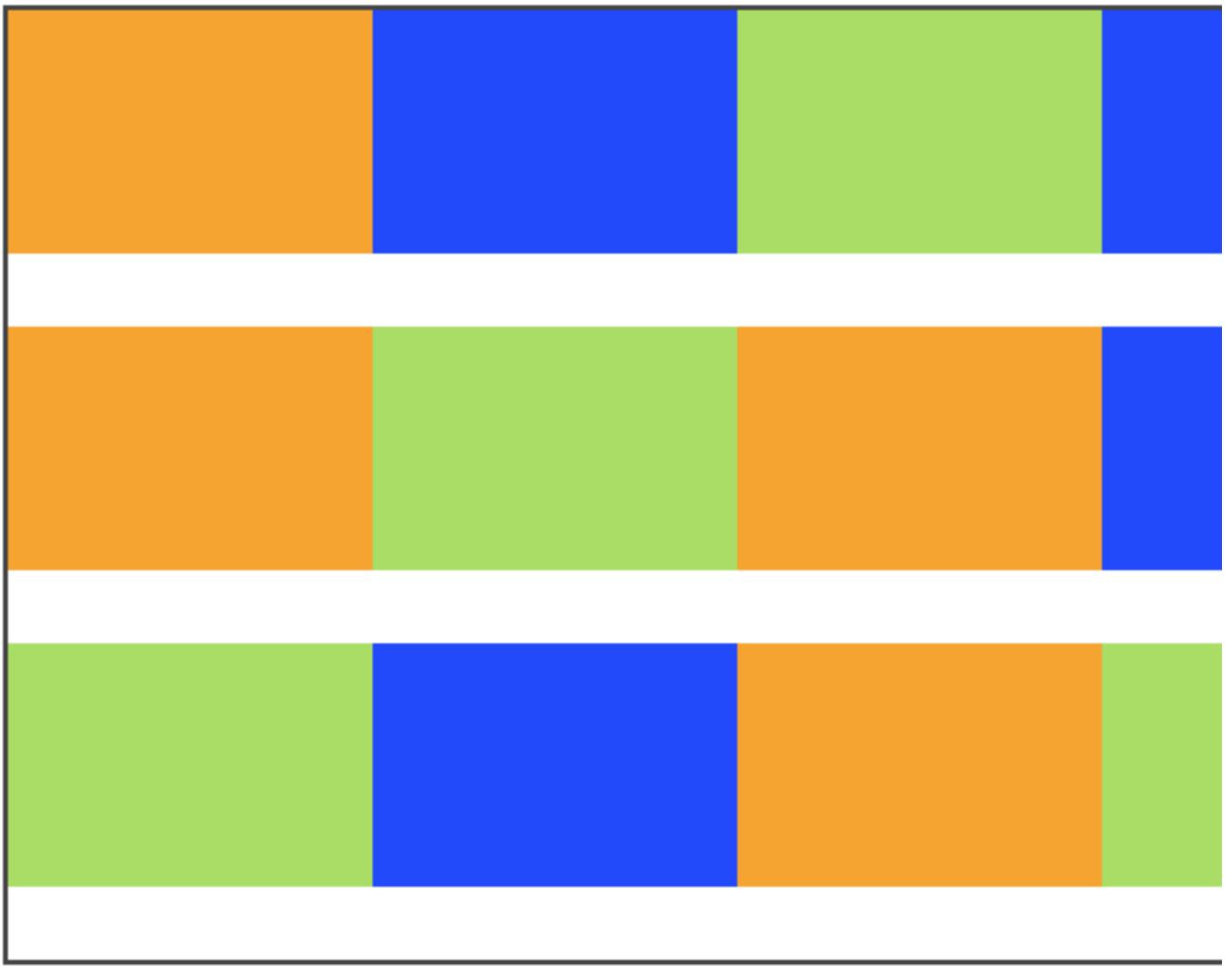
```
div id="conteneur"  
      div class="element"    div  
      div class="element"    div  
      div class="element"    div
```

J'autorise mes éléments à aller à la ligne avec `flex-wrap` :

#conteneur

display: flex;

`flex-wrap: wrap;`

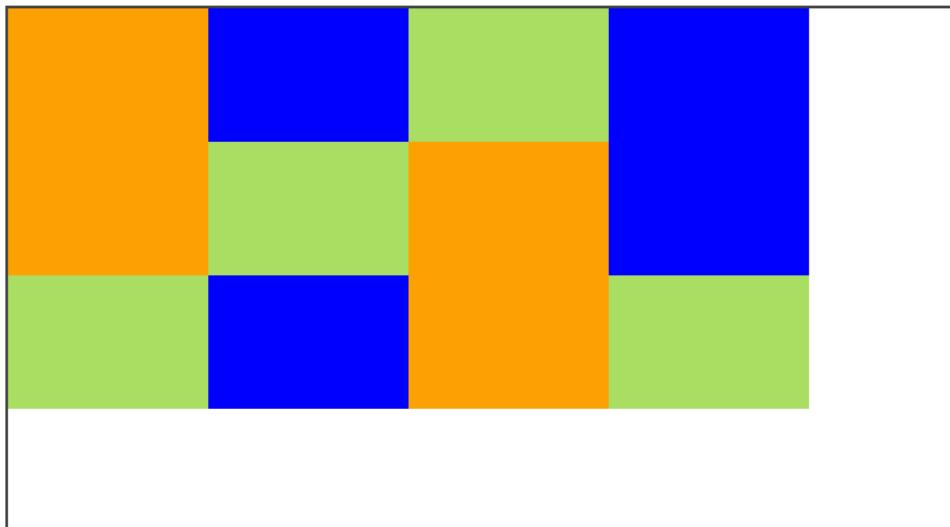


Plusieurs lignes dans une Flexbox

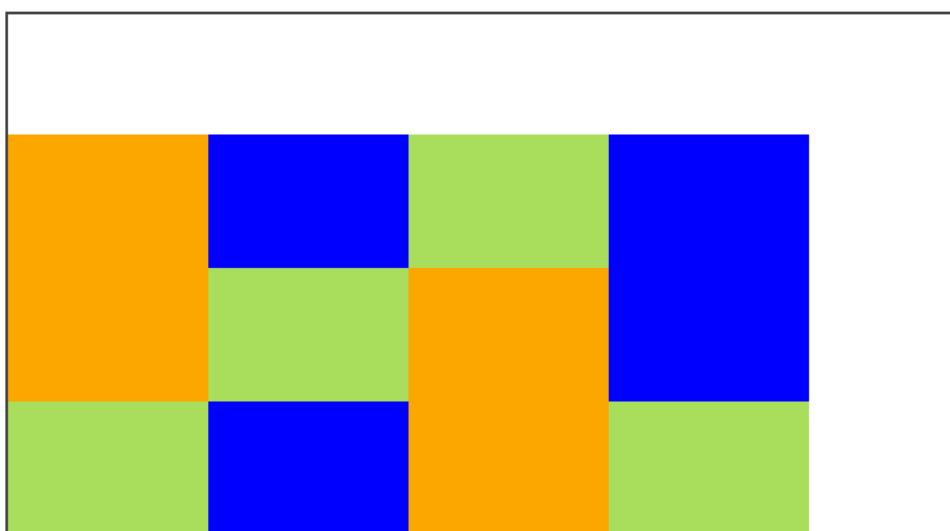
Jusque-là, rien de vraiment nouveau. Voyons voir comment les lignes se répartissent différemment avec la nouvelle propriété align-content que je voulais vous présenter. Elle peut prendre ces valeurs :

- flex-start : les éléments sont placés au début
- flex-end : les éléments sont placés à la fin
- center : les éléments sont placés au centre
- space-between : les éléments sont séparés avec de l'espace entre eux
- space-around : idem, mais il y a aussi de l'espace au début et à la fin
- stretch (par défaut) : les éléments s'étirent pour occuper tout l'espace

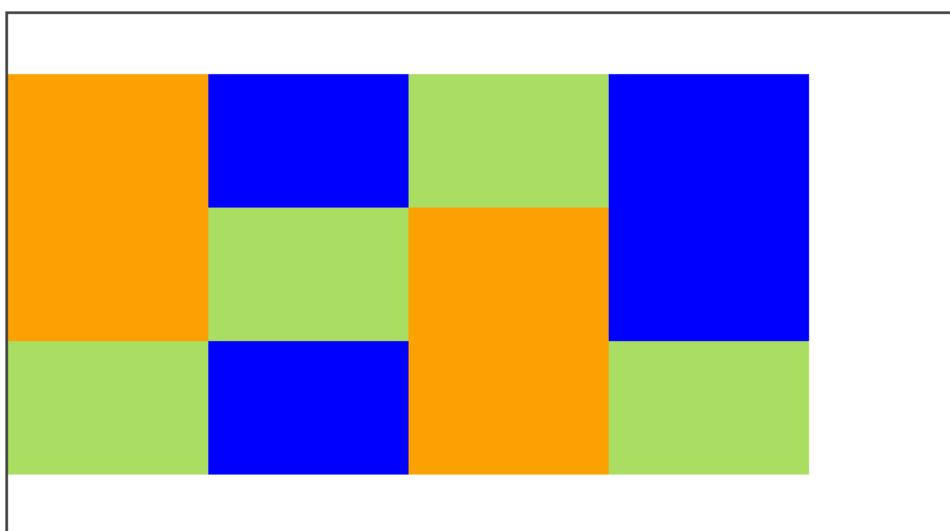
Voici ce que donnent les différentes valeurs :



flex-start



flex-end



center

Les lignes sont placées différemment avec align-content

Rappel à l'ordre

Sans changer le code HTML, nous pouvons modifier l'ordre des éléments en CSS grâce à la propriété `order`. Indiquez simplement un nombre, et les éléments seront triés du plus petit au plus grand nombre.

Reprenez une simple ligne de 3 éléments :

```
#conteneur
```

```
{
```

```
    display: flex;
```

```
}
```



Une ligne de 3 éléments

Si je dis que le premier élément sera placé en 3e position, le second en 1ère position et le troisième en 2nde position, l'ordre à l'écran change !

```
.element:nth-child(1)
```

```
{
```

```
    order: 3;
```

```
}
```

```
.element:nth-child(2)
```

```
{
```

```
    order: 1;
```

```
}
```

```
.element:nth-child(3)
```

```
{  
    order: 2;  
}
```



Avec order, nous pouvons réordonner les éléments en CSS

Encore plus flex : faire grossir ou maigrir les éléments

Allez encore une dernière technique, après on passe à la pratique.



Avec la propriété `flex`, nous pouvons permettre à un élément de grossir pour occuper tout l'espace restant.

```
.element:nth-child(2){  
    flex: 1;  
}
```



Le second élément s'étire pour prendre tout l'espace
Le nombre que vous indiquez à la propriété `flex` indique dans quelle mesure il peut grossir par rapport aux autres.

```
.element:nth-child(1)
```

```
{
```

```
  flex: 2;
```

```
}
```

```
.element:nth-child(2)
```

```
{
```

```
  flex: 1;
```

```
}
```

Ici, le premier élément peut grossir 2 fois plus que le second élément :



Le premier élément peut grossir deux fois plus que le second élément. La propriété `flex` est en fait une super-propriété qui combine `flex-grow` (capacité à grossir), `flex-shrink` (capacité à maigrir) et `flex-basis` (taille par défaut). J'utilise simplement `flex` comme je vous l'ai montré ici, mais si vous voulez en savoir plus, je vous invite à vous renseigner sur ces autres propriétés.

En résumé

- Il existe plusieurs techniques pour positionner les blocs sur la page. Flexbox est la technique la plus récente et de loin la plus puissante, que je vous recommande d'utiliser.
- Le principe de Flexbox est d'avoir un conteneur, avec plusieurs éléments à l'intérieur. Avec `display: flex;` sur le conteneur, les éléments à l'intérieur sont agencés en mode Flexbox (horizontalement par défaut).
- Flexbox peut gérer toutes les directions. Avec `flex-direction`, on

peut indiquer si les éléments sont agencés horizontalement (par défaut) ou verticalement. Cela définit ce qu'on appelle l'axe principal.

- L'alignement des éléments se fait sur l'axe principal avec `justify-content`, et sur l'axe secondaire avec `align-items`.
- Avec `flex-wrap`, on peut autoriser les éléments à revenir à la ligne s'ils n'ont plus d'espace.
- S'il y a plusieurs lignes, on peut indiquer comment les lignes doivent se répartir entre elles avec `align-content`.
- Chaque élément peut être réagencé en CSS avec `order` (pas besoin de toucher au code HTML !).
- Avec la super-propriété `flex`, on peut autoriser nos éléments à occuper plus ou moins d'espace restant.

Flexbox, c'est cool.