Documents Numériques M1105

TD / TP n°10 *Focus sur le CSS : placement avec le module Flex*

Le positionnement des éléments d'une page web est sans aucun doute l'un des aspects les plus difficile du CSS. Il existe de nombreuses techniques permettant de mettre en page un site. Un petit survol historique :

Au temps des dinosaures, on utilisait des tableaux pour positionner les différents éléments d'une page (berk!). Puis le CSS est apparu. La propriété `float` [1] qui n'avait pourtant pas été conçue pour ça, a été largement utilisée par les webmaster pour faire de la mise en page (bof). Une autre technique consiste à utiliser la propriété `display` avec l'attribut `inline-block` par exemple. Aujourd'hui, l'utilisation de Flexbox me semble être la méthode recommandée, pour son efficacité, et parce que ce module est maintenant reconnu par tous les navigateurs récents.

A l'issue de ce TP, vous devrez savoir positionner dans une page web les éléments comme un en-tête, des menus sur le côté, etc.

Une petite sitographie à explorer quand vous aurez terminé le TP :

Quelques liens pour aller plus loin avec Flex :

`http://www.w3schools.com/css/css3_flexbox.asp`

`http://www.alsacreations.com/tuto/lire/1493-css3-flexbox-layout-module.html`

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

Quelques liens pour se documenter sur les autres techniques :

`https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-cs`
`le-positionnement-en-css`

`https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Positioning`

Attention

Regardez toujours la date de publication des articles que vous consultez sur le web. En ce qui concerne le html et le css, les informations sont très vite périmées.

1. La propriété `float` est très utile pour "habiller" une image avec le texte qui l'entoure par exemple.

Documents Numériques M1105 (TD / TP n°10)

Exercice 1. Approche de la notion de flux (A faire sans ordinateur) [2]

Le flux d'un document pourrait se définir comme étant le comportement naturel d'affichage des éléments d'une page web.

À retenir

Les éléments se succèdent dans l'ordre où ils sont déclarés dans le code HTML avec les règles suivantes par défaut :

- Les éléments de type *inline* (`a`, `img`, `strong`, `code`, etc.) s'affichent côte à côte et leur taille (largeur et hauteur) va être déterminée par le texte ou l'élément qu'ils contiennent.
- Les éléments type *block* (`h1`, `p`, `article`, `nav`, `header`, `section`, etc.) s'affichent les uns en-dessous des autres en occupant par défaut la totalité de la largeur de son conteneur (son *block parent*). Leur hauteur sera déterminée ce qu'ils contiennent.

On considère le fichier `TP6-flex.html` suivant :

```
<!DOCTYPE html>
<html lang="fr">
<head>  <meta charset="utf-8"/>
        <title>TP6</title>
        <link rel="stylesheet" href="TP6.css"/>
        <link rel="stylesheet" href="TP6-flex.css"/>
</head>
<body>
  <header id="gris">Mon entête en <em>gris</em></header>
  <section id="vert">Section en vert</section>
  <article id="jaune"><h1>Mon article <strong>jaune</strong></h1>
    <p>Un paragraphe</p>
    <p>Un autre paragraphe</p>
    <p>Et un dernier paragrapphe</p>
  </article>
  <section id="rose">Section en rose</section>
</body>
</html>
```

2. Cet exercice est une vue un peu simpliste du flux et de la catégorisation des éléments. Depuis HTML5, c'est un peu plus complexe que cela.

1.1 Dessiner l'arbre de cette page.

1.2 Quels sont les éléments de type *block*? Quelle règle ajouter dans le CSS pour que tous ces éléments soient encadrés?

1.3 Dessiner l'allure qu'aura alors la page dans un navigateur.

1.4 Dessiner l'allure qu'aura la page si on ajoute la règle `p{ width:50%; }`

1.5 Copiez dans votre *home* les fichiers qui se trouvent sur Célène, puis vérifier avec un navigateur vos réponses aux questions précédentes. [3]

Dans tout ce TP, vous ne toucherez pas au code des deux fichiers `TP6-flex.html` et `TP6.css`. Vous travaillerez uniquement sur le fichier `TP6-flex.css`.

1.6 Que se passe-t-il si on ajoute la règle `em, strong { width:50%; }`?

Positionnement en utilisant le module Flex-Box

Flexbox a été conçu pour étendre le modèle de boîte classique en CSS en introduisant *le Modèle de boîte flexible*. Le principe est simple : vous définissez un conteneur (*flex-container*), et à l'intérieur vous placez plusieurs éléments (*flex-items*). 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.

Exercice 2. Propriétés des *flex-container*

À retenir

La règle `#container { display : flex; }` transforme le bloc concerné en *flex-container* et ses enfants deviennent automatiquement des éléments de type *flex-item*. Par défaut, il sont "rangés" horizontalement dans le container sur une seule ligne.

2.1 J'ajoute au CSS la règle `#jaune{ display:flex; }`. Quel bloc sera transformé en *flex-container*? Quels blocs seront transformés en *flex-items*? Quelle sera alors l'allure de la page?

3. On pourra remarquer que l'allure de cette page n'est pas tout à fait la même suivant le navigateur qu'on utilise.

Vérifier ensuite avec le navigateur. Observer ce qu'il se passe si on change la taille de la fenêtre et/ou si on zoome/dezoome.

2.2 Je remplace la règle pécédente par `body{ display:flex; }`. Quel bloc sera transformé en *flex-container*? Quels blocs seront transformés en *flex-items*? Quelle sera alors l'allure de la page? Vérifier ensuite avec le navigateur. Observer ce qu'il se passe si on change la taille de la fenêtre et/ou si on zoome/dezoome.

À retenir

La règle `#container { flex-wrap: nowrap (default) | wrap }` définit si les *flex-items* du *flex-container* seront distribués sur une seule ligne (ou colonne selon l'axe principal) ou sur plusieurs lignes. En clair, si les *flex-items* ont le droit de passer à la ligne ou non.

2.3 Ajoutez dans votre fichier css la règle `body{ flex-wrap: wrap; }`. Observez le résultat et le comportement de la page quand on zoome et/ou change la taille de la fenêtre.

Exercice 3. Propriétés des blocs enfants *flex-items*

On peut vouloir des "flexibilités" différentes selon les blocs enfants. Il est alors possible d'ajouter des règles à certains (ou à tous) les *flex-items*.

À retenir

Il y a trois propriétés pour contrôler la largeur des *flex-items* :

flex-grow La propriété `flex-grow: <entier>` indique la capacité d'un flex-item à s'étirer pour couvrir lespace vide restant dans le flex-container

flex-shrink La propriété `flex-shrink: <entier>` indique la capacité pour un flex-item à se contracter si le flex-container manque de place pour tous les flex-items

flex-basis La propriété `flex-basis: <longueur>` indique la taille initiale de l'élément avant que l'espace restant/manquant ne soit distribué

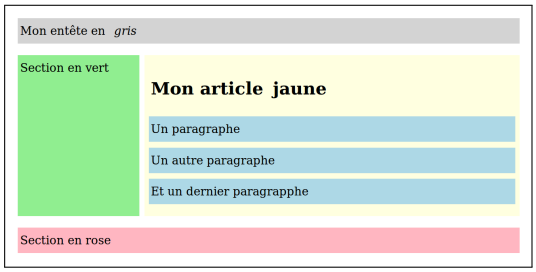
Les valeurs par défaut sont : `flex-grow: 0`, `flex-shrink: 1` et `flex-basis: auto`. En clair, les flex-items n'occupent initialement que la taille minimale de leur contenu; ils peuvent rétrécir si nécessaire, mais ils ne grandissent pas s'ils leur reste de la place.
Pour rendre un élément flexible, il suffit de lui attribuer une valeur de `flex-grow` supérieure à zéro. Cet élément occupera alors lespace restant au sein de son conteneur.
Par exemple : `flex-grow: 1;`

3.1 Ajoutez dans votre fichier css la règle `#vert{ flex-grow : 1; }`. Observez le résultat. Que se passe-t-il si vous transformez la valeur du `flex-wrap` du *flex-container* (transformez le `wrap` en `nowrap`) ?

3.2 Ajoutez au fichier css la règle `#gris{ flex-grow : 1; flex-shrink: 1; flex-basis: 60%;}` Observez le résultat. Et si vous transformez la valeur du `flex-wrap` du *flex-container*?

Remarque : dans ce TP, on se contentera de laisser la valeur de flex-shrink à 1, mais vous êtes tout à fait libres de faire des tests et/ou des recherches dessus

3.3 Modifiez votre fichier css pour obtenir le visuel ci-contre (vous n'avez toujours pas le droit de modifier le fichier html) :

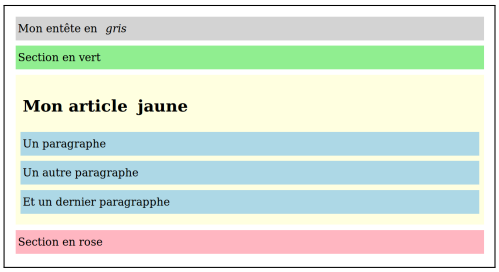


Exercice 4. Positionnement sans utiliser Flex mais en utilisant l'attribut CSS inline-block

Pour compléter ce TP sur le positionnement, il me semble important de vous montrer qu'il n'est pas toujours utile d'utiliser Flex. On peut modifier la distribution des boites (horizontale/verticale) en modifiant la valeur (`block` ou `inline`) de l'attribut `display` de chaque boîte "enfant".

Reprenons seulement les fichiers `TP6-flex.html` et `TP6.css` (sans le fichier `TP6-flex.css`)

4.1 Ajouter la propriété CSS :
`body>* { display: inline-block; }`
A quels éléments s'applique-t-elle? Quel résultat obtient-on?
4.2 Remplacer la propriété CSS précédente par `p { display: inline-block; }`.
A quels éléments s'applique-t-elle? Quel résultat obtient-on?



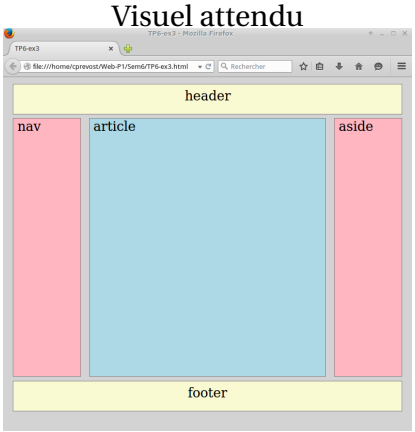
4.3 Faites un zoom (à l'aide de la molette de la souris) et/ou redimensionnez votre fenêtre. Quel(s) inconvénient(s) observez-vous?

Exercice 5. Une autre mise en page avec flex

TP6-ex7.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8"/>
    <title>TP6-Placement</title>
    <link rel="stylesheet" href="TP6-ex3.css"/>
 </head>
 <body>
    <header>header</header>
     <section>
            <article>article</article>
            <nav>nav</nav>
            <aside>aside</aside>
     <section>
     <footer>footer</footer>
</body>
</html>
```

Visuel attendu



5.1 Sans utiliser l'ordinateur, et sans fichier css, quelle apparence aura cette page?

5.2 Ecrire un fichier css pour obtenir le visuel attendu.

Exercice 6. A vous de jouer

Reprenez votre travail du TP4 (arbre + fichier html) puis concevez un fichier CSS qui vous donne un visuel qui ressemble à :