Leçon 3 : Le responsive design avec les Media Queries de CSS3 et Flexbox

Cours_ UVCI
PRW2103-2

Avril 2018
1.0

Table des matières

Objectifs

À la fin de cette leçon, vous serez capable de :

- *Comprendre* la notion de responsive design ;
- *Mettre* en œuvre le responsive design avec les Media Queries.
- *Comprendre* la technique flexbox et ses avantages ;
- *Structurer* une page web avec « Flexbox »

Section 1 - Le Responsive Design

1. 1. Qu'est ce que le Responsive Design ?

Le responsive design, aussi appelé *RWD* (abréviation de Responsive Web Design) est une nouvelle façon de concevoir les sites web afin qu'ils puissent être consultés de manière optimale quel que soit le support utilisé (Ordinateur, tablette ou smartphone).

Avec la mode des tablettes et des smartphones, les habitudes des internautes changent. Les sites Internet doivent donc s'adapter à ces nouveaux supports dont les définitions d'écran sont quasiment toutes différentes.

Les premières réponses données a cette problématique a été de créer des version mobiles pour le sites web existant (*exemple* :de *http://abidjan.net*, on crée une version mobile à l'adresse *http://mobile.abidjan.net*) cela fait donc deux sites différents à gérer avec pratiquement les mêmes contenus ou pas. De plus, quand apparaissent les tablettes qui ne sont ni smartphone ni ordinateurs avec des tailles d'écran, le problème aggrave et devra encore s'aggraver avec l'apparition d'autre format d'écran comme les télés connectées. Il est alors question de trouver une solution adaptative et évolutive quelque soit le type de terminal utilisé par les visiteurs.

C'est là tout l'intérêt de la technologie du "*responsive web design*" qui signifie littéralement " *conception web adaptative*" qui est donc une technique de réalisation de site web, qui permet de faire des sites qui s'adaptent automatiquement à l'espace disponible sur l'écran.

En responsive web design, on ne dit pas "o*n va faire un site mobile, un site tablette et un site ordinateur*". On fait un seul site, conçu pour exploiter au mieux la taille de l'écran. L'avantage, c'est qu'on peut ainsi gérer toutes les tailles d'écran d'un coup.

Mais comment cela peut-il se faire ?

Le principe du responsive design est simple : une seule interface accompagnée d'une URL est prévue pour fonctionner avec une ergonomie adaptée quel que soit le terminal de lecture (ordinateur, tablette ou smartphone).

Cette technologie relativement récente utilise les *media queries* des feuilles de style. Cette fonction en *CSS3* permet de cibler la largeur, la hauteur ou encore l'orientation d'une page dans une seule et unique feuille de style.

Au final et pour une même adresse web, les pages du site s'adaptent automatiquement au format de lecture utilisé. Sur les plus petits écrans, seuls les contenus les plus intéressants restent affichés sur la page et il n'est plus nécessaire de zoomer pour lire le texte.

Si vous voulez en faire l'expérience, il vous suffit de vous rendre sur un site utilisant cette technologie (par exemple http://www.barackobama.com/) et de réduire la taille de la fenêtre avec votre souris. Vous verrez alors, comme par magie, le contenu du site s'adapter à la taille de la fenêtre.

Source : https://user.oc-static.com/files/420001_421000/420712.gif

En général, *on se base sur la largeur de l'écran pour déterminer comment les éléments du site doivent s'afficher :*

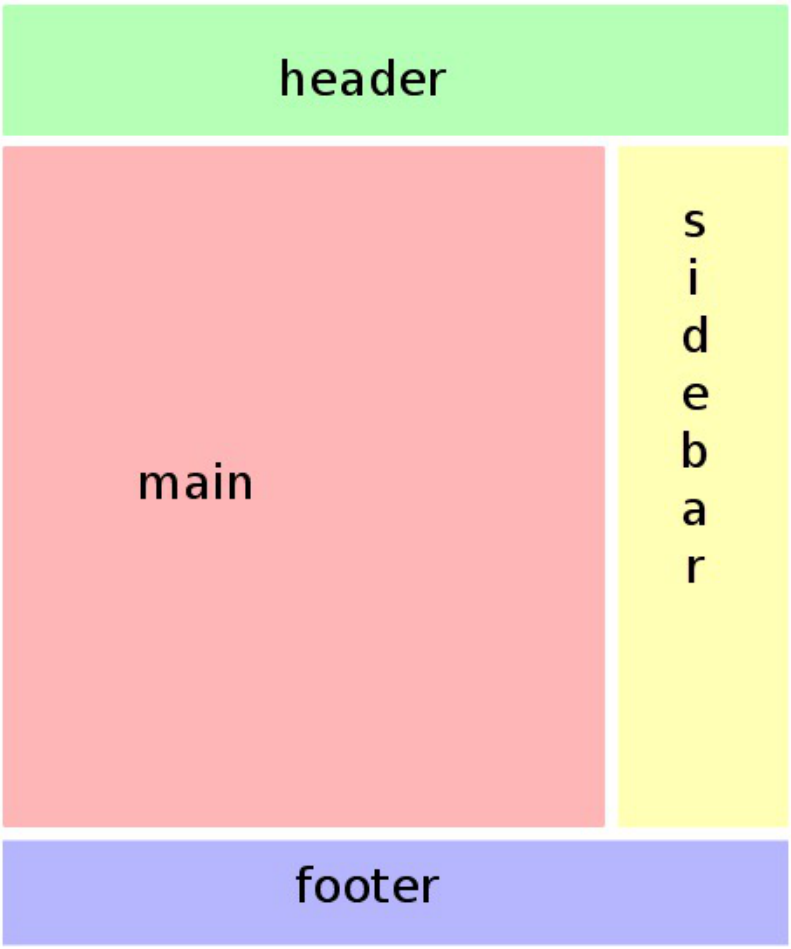
Quand on dispose de moins de place en largeur, on peut mettre moins de blocs côte à côte. Dans ce cas, on les positionne plutôt de haut en bas.

On peut aussi considérer que certains blocs moins importants sont inutiles sur des mobiles et les faire disparaître. On peut aussi, à l'inverse, faire apparaître de nouveaux blocs et outils de navigation sur mobile si on le souhaite.

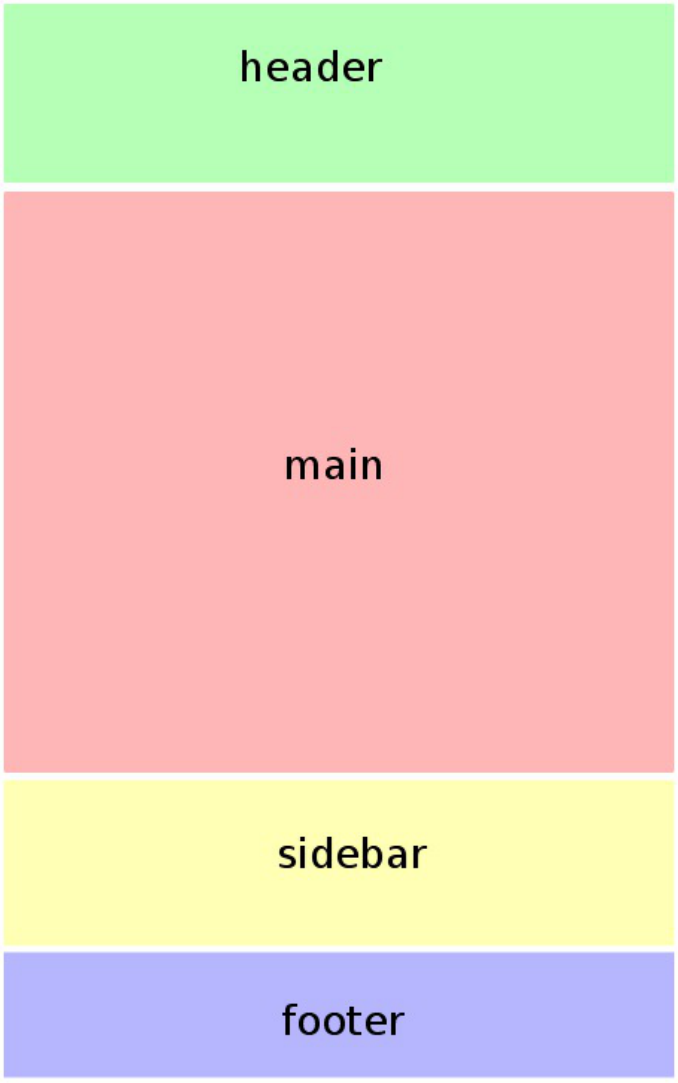
En général, un design d'une page web se présente sommairement de quatre éléments :

- Une entête
- Un pied de page
- Une partie principale
- Une colonne latérale

Comme ceci :



Avec le responsive design, on peut le styler pour les petits écrans pour donner le design suivant en conservant ici tous les élément de la page précédente mais adaptés à la taille de l'écran.



2. 2. Mise en œuvre du responsive design avec media queries

Maintenant que nous savons comment un design doit se comporter en fonction de la largeur , il reste à donner vie à notre design responsive.

Pour le faire, il faut connaître le fonctionnement technique des Media Queries CSS3, qui permettent d'adapter le design en fonction de certains critères comme la largeur de l'écran en permettant d'écrire des règles qui disent à l'ordinateur par exemple : "*Si la largeur de l'écran est supérieure à X et inférieure à Y, alors placer ce bloc en-dessous du précédent*".

Les *media queries* font partie des nouveautés de CSS3. Il ne s'agit pas de nouvelles propriétés mais de règles que l'on peut appliquer dans certaines conditions.

Les *media queries* sont donc des règles qui indiquent quand on doit appliquer des propriétés CSS3. Il y a deux façons de les utiliser :

- en chargeant une feuille de style.css différente en fonction de la règle (ex : « Si la résolution est inférieure à 1280px de large, charge le fichier *mobile_resolution.css*») ;
- en écrivant la règle directement dans le fichier *.css* habituel (ex : « Si la résolution est inférieure à 1280px de large, charge les propriétés CSS ci-dessous »).

a) Chargement d'une feuille de style différente

Souvenez-vous de la balise<link />qui permet, dans un code HTML, de charger un fichier.css ; on

peut lui ajouter un attribut *media*, dans lequel on va écrire la règle qui doit s'appliquer pour que le fichier soit chargé. On dit qu'on fait une « requête de media » (*media query en anglais*). Voici un exemple :

<link rel="stylesheet" *media="screen and (max-width: 1280px)"* href="petite_resolution.css" />

Au final, le code HTML pourrait proposer plusieurs fichiers CSS : un par défaut et un ou deux autres qui seront chargés en supplément uniquement si une règle correspondante s'applique.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <link rel="stylesheet" href="style.css" /> <!-- Pour tout le monde -->
6 <link rel="stylesheet" media="screen and (max-width: 1280px)" href=
  "petite_resolution.css" /> <!-- Pour ceux qui ont une résolution inférieure à
  1280px -->
7 <title>Media queries</title>
8 </head>
9
```

Pour que le navigateur décide quelle feuille de style faut-il appliquer, il se réfère à l'attribut *media*.

L'attribut *media* est déclaré sur la balise <link> pour indiquer au navigateur dans quel cas il faut appliquer la feuille de style associée. Les valeurs qu'elle peut accueillir sont nombreuses, mais voici les plus courantes:

- *screen*: signifie les écrans.
- *print*: pour les imprimantes.
- *handheld*: désigne les appareils mobiles ou écrans de petite taille.
- *projection*: signifie les projecteurs.
- *tv:* pour les téléviseurs.
- *tty*: signifie terminal (on peut visualiser une page Web sur un terminal comme l'invite de commande).
- *all*: qui désigne tous les périphériques.

Vous avez donc compris qu'il faut créer une feuille de style pour chaque type de périphérique si nous voulons utiliser cette méthode.

b) Chargement des règles directement dans la feuille de style

Une autre technique, que nous allons utiliser dans ce cours pour des raisons pratiques, consiste à écrire ces règles dans le même fichier CSS que d'habitude.

Dans ce cas, on écrit la règle dans le fichier *.css* comme ceci :

```
1
2 @media screen and (max-width: 1280px)
3 {
4 /* Rédigez vos propriétés CSS ici */
5 }
6
```

Au lieu de créer plusieurs feuilles de styles, chacune pour un type de périphérique, on peut donc créer une seule qui renferme tous les styles groupés par les *conditions Media Queries* commençant par *@media*.

La règle *@media* est suivie du type de périphérique puis deux accolades ({}) qui contiennent l'ensemble des styles CSS à appliquer.

Cependant, le fait de savoir que le périphérique est un écran ne suffit pas à définir les styles adéquats, car il y a des écrans larges et des petits écrans. Heureusement, les Media Queries proposent des opérateurs logiques qui permettent de mieux affiner la cible du style.

c) Opérateurs logiques

Les opérateurs logiques disponibles pour les Media Queries sont:

- *and*: qui signifie un ET logique. Par exemple, la règle A and B est vraie si A et vrai et B est vrai aussi.
- *only*: qui signifie UNIQUEMENT. Par exemple, la règle only A est vraie si A tout seul est vrai.
- *not*: qui signifie NON. Par exemple, la règle not A est vraie si A est fausse.

```
1
2 @media screen and (min-width:1280px){
3    .pub{
4       diaplay:inline-block;
5    }
6 }
```

Ce code signifie qu'on peut afficher la publicité (désignée par la classe pub) si le périphérique est un écran et la largeur de la fenêtre (du navigateur) est supérieure ou égale à 1280 pixels.

d) Critère des Media Queries

Les critères sont placés entre des parenthèses pour constituer un élément de la condition à vérifier. Il existe plusieurs critères sur lesquels reposent les Media Queries. Mais je vais vous lister ceux dont vous aurez le plus besoin.

- *width*: signifie la largeur de la fenêtre d'affichage (est pas celle de l'écran). Si par exemple nous spécifions (*width*:800px) cela veut dire que cet élément de la condition est vrai si la largeur de la fenêtre est exactement égale à 800 pixels. On peut préfixer ce critère par *min-* (pour min-width) ou max- (pour max-width) pour désigner respectivement la largeur minimale et la largeur maximale.
- *height*: signifie la hauteur de la fenêtre d'affichage. On peut préfixer ce critère par *min-* (pour *min-height*) ou *max-* (pour *max-height*) pour désigner respectivement la hauteur minimale et la hauteur maximale.
- *device-width*: signifie la largeur de l'écran. On peut préfixer ce critère par *min-* (pour *min-device-width*) ou *max-* (pour *max-device-width*) pour désigner respectivement la largeur minimale de l'écran et la largeur maximale de l'écran.
- *device-height*: signifie la hauteur de l'écran. On peut préfixer ce critère par *min-* ou *max-*.
- *aspect-ratio*: signifie le ratio (rapport entre la largeur et la hauteur) de la fenêtre d'affichage.
- *device-aspect-ratio*: signifie le ratio de l'écran (par exemple: 16/9 ou 4/3).
- *orientation*: signifie l'orientation de la zone d'affichage et peut prendre deux valeurs: *portrait* pour l'affichage vertical ou *landscape* pour l'affichage horizontale (ou paysage).

3. 3. Cas pratiques avec media queries

a) changement du fond d'écran en fonction de la taille de l'écran

---

Soit le code HTML minimal suivant :

```html
<doctype html>
<html lang="fr">
    <head>
        <title>Un site responsive design</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
        <h1>Le Responsive Web Design</h1>
        <p>La couleur de fond de la page change
        en fonction de la taille de l'écran.</p>
    </body>
</html>
```

Nous voulons changer la couleur de fond de l'écran en fonction de sa taille selon les règles suivantes :

- Si la taille de l'écran d'un visiteur est *inférieure à 480px*, la couleur de fond de notre élément body (donc de notre page) sera *orange*
- Pour les écrans ayant une taille *entre 481px et 1179px*, la couleur de fond de notre page sera *blanche*
- Pour les écrans d'une taille *supérieure à 1280px*, le fond de la page sera de couleur *verte*

Nous allons alors créer notre fichier style.css lié à notre page HTML précédente dans laquelle nous allons appliquer les media queries pour gerer l'affichage en fonction des taille de l'écran du visiteur. le code CSS minimal est le suivant :

rwd_1.html | style.css

```css
/*Si la taille de l'écran d'un visiteur est inférieure à 480px*/
@media screen and (max-width: 680px){
    body{
        background-color: orange;
    }
}
/*Pour les écrans ayant une taille entre 481px et 1179px*/
@media screen and (min-width: 681px) and (max-width: 1279px){
    body{
        background-color: #ffffff;
    }
}
/*Pour les écrans d'une taille supérieure à 1280px*/
@media screen and (min-width: 1280px){
    body{
        background-color: green;
    }
}
```

Contenu du fichier style.css

Méthode : Travaux Pratiques N°1

Ceci représente votre *TP N°1* du Responsive Web Design. Pour réaliser ce TP, vous devez :

- Créer le fichier HTML avec le code ci-dessus ;

- Créer le fichier *style.css* avec le code CSS ci-dessus et le placer dans le même repertoire que le fichier HTML

- Exécuter le fichier HTML dans votre navigateur, réduisez ou agrandissez la taille du navigateur pour voir les effets de votre code CSS en fonction de la teille de l'écran d'affichage.

b) Changer l'agencement des éléments HTML en fonction de la taille de l'écran

---

Les media queries intéressants pour masquer certains éléments ou pour modifier leur agencement selon la taille de l'écran des visiteurs.

Pour faire cela, nous pouvons utiliser les propriétés *display*, *position* et les *valeurs relatives en pourcentages* de façon générale.

Supposons la code HTML suivant :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>TP 2 : Responsive web design</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="style2.css">
</head>

    <body>

        <h1>TP 2 : Responsive web design</h1>
         <div class="menu">
            <span style="font-size:28px">Ici, le menu du site.</span>
        </div>
        <div class="div1">
            <p>Paragraphe de la barre latérale gauche</p>
        </div>
        <div class="div2">
            <p>Ici la partie centrale du site avec assez de contenu</p>
        </div>
        <div class="div3">
            <p>Voila aussi une barre latérale droite</p>
        </div>
        <div class="footer">
            Ici, le pied de page avec mon copyright
        </div>
    </body>
</html>
```

en lui associant le code CSS dans le fichier référence *style.css* ci-dessous :

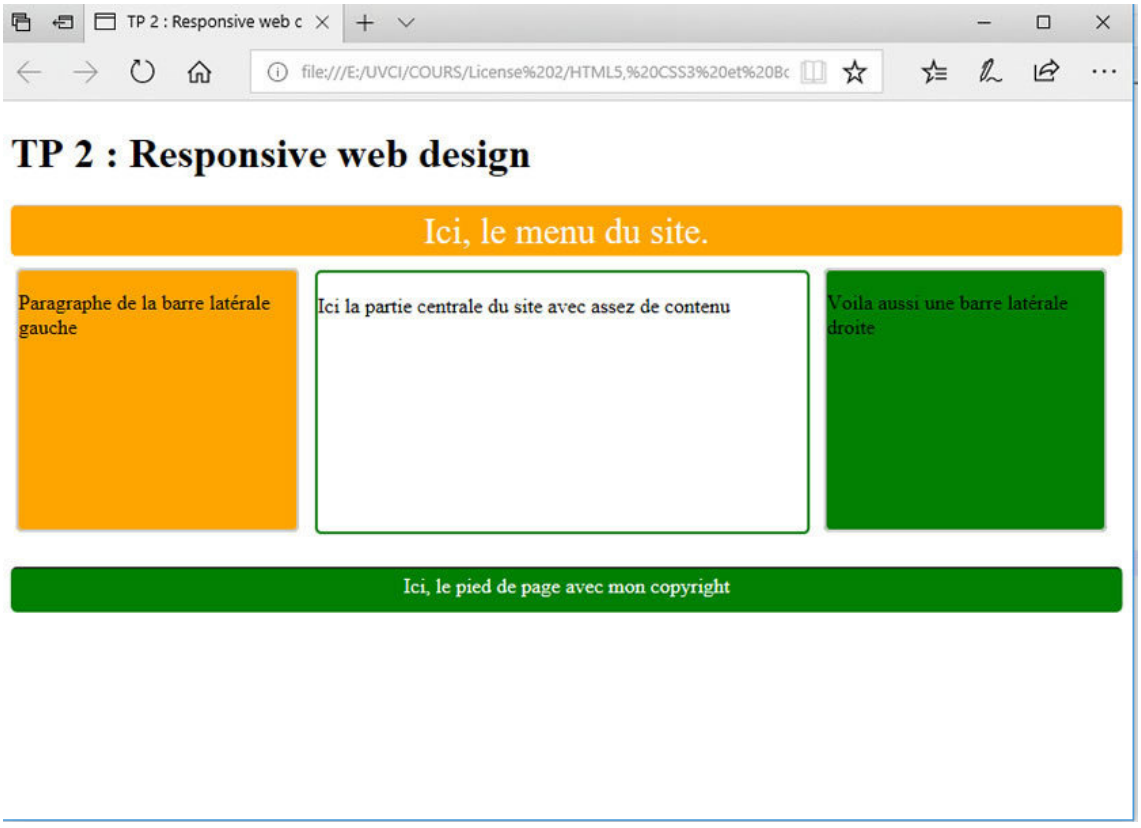
```css
.menu{
  padding-top:4px;
    width: 100%;
    height: 35px;
    margin-top: 5px;
    border-top: 1px solid #ccc;
    text-align: left;
    background-color: orange;
  border-radius:5px;
  margin-bottom:5px;
  color:#fff;
  text-align: center;
}
.div1,.div3{
    display: inline-block;
    width: 25%;
    min-height:200px;
    border: 2px solid #ccc;
    margin: 4px;
  border-radius:5px;
}
.div2{
    display: inline-block;
    width: 44%;
```

```
    min-height:200px;
    border: 2px solid green;
    margin: 4px;
  border-radius:5px;
}

.div1{
    background-color: orange;
}
.div2{
    background-color: #fff;
}
.div3{
    background-color: green;
}

.footer{
  padding-top:4px;
    width: 100%;
    height: 30px;
    margin-top: 5px;
    border-top: 1px solid black;
    text-align: center;
    background-color: green;
  color:#fff;
  border-radius:5px;
}
```

le rendu visuel de la page sur un écran de PC donne la figure suivante :



Pour l'affichage sur mobile, nous désirons maintenant afficher la page sur une seule colonne et enlever le pied de page et le menu afin de faciliter la lisibilité des visiteurs.

Admettons qu'on considère tous les écrans de mobiles d'une taille intérieure à 780px.

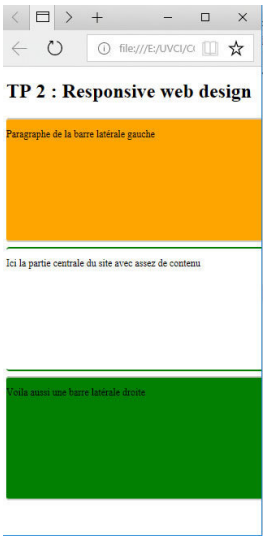
Le CSS peut donc être modifié pour donner le code suivant, en utilisant les media Queries :

```
.menu{
  padding-top:4px;
    width: 100%;
    height: 35px;
    margin-top: 5px;
    border-top: 1px solid #ccc;
    text-align: left;
    background-color: orange;
  border-radius:5px;
  margin-bottom:5px;
  color:#fff;
  text-align: center;
}
.div1,.div3{
    display: inline-block;
    width: 25%;
    min-height:200px;
    border: 2px solid #ccc;
    margin: 4px;
  border-radius:5px;
}
.div2{
    display: inline-block;
    width: 44%;
    min-height:200px;
    border: 2px solid green;
    margin: 4px;
  border-radius:5px;
}

.div1{
    background-color: orange;
}
.div2{
    background-color: #fff;
}
.div3{
    background-color: green;
}

.footer{
  padding-top:4px;
    width: 100%;
    height: 30px;
    margin-top: 5px;
    border-top: 1px solid black;
    text-align: center;
    background-color: green;
  color:#fff;
  border-radius:5px;
}
/*---------- media Query pour les petits ecrans ---*/
@media screen and (max-width: 780px){
```

```
54      .div1, .div2, .div3{
55          width: 100%;
56          border-right: none;
57          border-left: none;
58          margin-left: 0px;
59          margin-right: 0px;
60      }
61      .menu{
62          display: none;
63      }
64    .footer{
65          display: none;
66      }
67  }
68
```



Pour le même code HTML, en modifiant le CSS, on a resultant ci-contre sur un écran plus petit.

Remarquez que l'on avait déjà commencé à préparer le terrain : nous avons utilisé des valeurs en pourcentages pour nos éléments *div* afin que ceux-ci grandissent ou rapetissent en même temps que la page web.

Et, comme vous pouvez le constater, nous avons pu modifier complètement notre page grâce à *@media*. En effet, les règles définies dans *@media vont être prioritaires sur les règles « générales »* dès que la taille de l'écran va passer sous les 780px.

Méthode : Travaux Pratiques N° 2

Ceci représente votre *TP N°2* du Responsive Web Design. Pour réaliser ce TP, vous devez :

- Créer le fichier HTML avec le code ci-dessus ;

- Créer le fichier *style2.css* avec le code CSS ci-dessus sans les media queries et le placer dans le même répertoire que le fichier HTML

- Exécuter le fichier HTML dans votre navigateur, réduisez ou agrandissez la taille du navigateur pour voir les effets de votre code CSS en fonction de la taille de l'écran d'affichage.

- Modifiez le fichier *style2.css* avec le code CSS en y ajoutant les media queries

- Exécuter le fichier HTML dans votre navigateur, réduisez ou agrandissez la taille du navigateur pour voir les effets de votre code CSS en fonction de la taille de l'écran d'affichage.

4. 4. Quelques astuces particulières

- Pour les images

Les images doivent avoir une hauteur et une largeur pour un rendu plus rapide de la page. Cependant, il est recommandé de mettre ces valeurs dans les attributs « *height* » et « *width* » de l'élément :

<img src="image.jpg" alt="lorem ipsum" height="320" width="640">

Ensuite, ajouter ceci au code CSS destiné aux petits écrans :

img {

max-width: 100%;

height: auto;

}

De cette façon, les images ne dépasseront jamais de la largeur de l'écran : si l'écran est plus petit que l'image, alors c'est l'image qui se redimensionne en conservant ses proportions.

- Éviter le débordement des images de fond

Quand on utilise la propriété CSS *background-image* il se peut que vous utilisiez une image de la taille exacte du bloc auquel il est appliqué. Dans ce cas, si vous réduisez ce bloc pour qu'il soit affiché correctement sur un téléphone, l'image de fond est en partie masquée.

Pour remédier à cela, vous pouvez utiliser la propriété CSS3 *background-size* : celle-ci va alors décider de la façon dont l'image de fond occupe l'espace de son conteneur.

• Avec aucun *background-size* (l'image est placée telle qu'elle dans le bloc et on découpe ce qui dépasse) :

• Avec *background-size: contain* (on redimensionne l'image pour qu'elle soit entièrement visible) :

• Avec *background-size: cover* (on agrandit l'image en conservant les proportions pour qu'elle recouvre tout, puis on découpe ce qui dépasse en bas) :

• Avec *background-size: 50px 70px* (l'image fait 50px par 70px) :

• Avec *background-size: 100% 100%* (l'image fait la taille du bloc, quite à perdre ses proportions) :

Notez que l'on peut également jouer avec *background-position* et *background-repeat* pour obtenir d'autres effets.

5. 5. le viewport HTML

Le *viewport* correspond à la surface sur laquelle une page Web est visible (ou la surface de la fenêtre utilisée par le navigateur). Cette notion a été introduite suite à l'avènement des appareils mobiles (smartphones et tablettes) sur lesquelles on ne peut pas redimensionner la fenêtre du navigateur (comme on peut le faire sur un écran classique).

Avec *viewport*, on va demander à ce que nos pages web s'adaptent à la taille de la fenêtre de chacun de nos visiteurs, afin que notre contenu la remplisse.

Pour déclarer la nouvelle valeur du viewport, on fait appel à la balise *<meta>*comme vous avez dû le constater dans les TP précédents :

```
1 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Ce code signifie que la valeur du *viewport* initiale correspond à la largeur en pixel CSS de l'appareil (la largeur qu'il pense avoir) et le zoom initial est égal à 1x (ce qui signifie: aucun zoom).

On peut aussi ajouter les sous-propriétés: *height, maximum-scale et minimum-scale* qui signifient respectivement hauteur, zoom maximal, et zoom minimal.

Le code *width=device-width* va faire en sorte que la page s'adapte à la taille de l'écran de votre visiteur avec les éléments HTML prenant la place disponible.

Le code *initial-scale=1.0* définit le niveau de zoom initial lorsque la page est chargée par le navigateur.

Voici un exemple de résultat avec et sans utilisation de `meta viewport` de notre code HTML pour un affichage sur mobile :

Source : http://www.pierre-giraud.com/html-css/cours-complet/imgs/utilisation-meta-viewport.png

6. Exercice : Quiz d'entrainement

Exercice

1) Si on veut adapter l'affichage d'un contenu selon la largeur de l'écran, quelle propriété doit-on déclarer ans la règle @media?

- ◯ width
- ◯ screen-width
- ◯ device-width

Exercice

2) Que signifie le code suivant ?

```
@media print{
    *{
        font-family:serif;
    }
}
```

- ◯ Tous les textes seront écrits en police de la famille générique Sérif si c'est un équipement, autre qu'une imprimante, qui affiche le document.
- ◯ Tous les textes seront écrits en police de la famille générique Sérif si c'est une imprimante qui imprime le document.

Exercice

3) Pourquoi les mobiles considèrent-ils la définition en pixel CSS (Device Width) alors qu'ils ont déjà une définition réelle?

- ◯ La définition réelle n'a aucun sens sans définition en pixels CSS.
- ◯ La définition réelle renferme beaucoup de pixles sur un espace réduit, ce qui entraîne l'affichage en petits caractères du contenu hypertexte.
- ◯ La définition réelle sert à afficher le contenu hypertexte alors que la définition en pixel CSS sert à afficher les photos de la Caméra.

Exercice

4) Que signifie le code suivant ?

```
@media screen and (min-width:960px){
    #conteneur{
        width:920px;
    }
}
```

- ◯ La balise identifiée par "conteneur" aura une largeur de 920px si la fenêtre du navigateur fait au minimum 960px de largeur.
- ◯ La balise identifiée par "conteneur" aura une largeur de 920px si la fenêtre du navigateur fait au maximum 960px de largeur.
- ◯ La balise identifiée par "conteneur" aura une largeur de 960px si la fenêtre du navigateur fait au minimum 920px de largeur.
- ◯ La balise identifiée par "conteneur" aura une largeur de 960px si la fenêtre du navigateur fait au maximum 920px de largeur.

Exercice

5) Si nous n'appliquons pas la balise META VIEWPORT comme ceci:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

- ◯ Le texte de la page Web sera trop petit sur un appareil mobile.
- ◯ Le texte de la page Web sera trop grand sur un appareil mobile.
- ◯ Le texte s'affichera normalement sur un appareil mobile.

Section 2 - Présentation de Flexbox

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

L'utilisation des tableaux HTML pour faire la mise en page ;

L'utilisation de CSS `float` CSS pour aider à faire la mise en page, avec beaucoup d'inconvénients ;

La création d' éléments de type `inline-block` dans le documents pour faire la mise en page ;

Aujourd'hui, une bien meilleure technique encore existe : *Flexbox* !

La suite de ce cours est basé sur un tutoriel du site Web *Alsacreations.com* où l'auteur *Raphaël*, propose un apprentissage par la pratique de flexbox.

1. 1. le principe de Flexbox

Le module des boîtes flexibles, aussi appelé « *flexbox* », a été conçu comme un modèle de disposition unidimensionnel et comme une méthode permettant de distribuer l'espace entre des objets d'une interface ainsi que de les aligner.

Flexbox se fonde schématiquement sur une architecture de ce type :

- Un conteneur "`flex-container`" permettant de créer un contexte général d'affichage,
- Un ou plusieurs "`flex-item`" qui ne sont rien d'autre que les enfants directs du conteneur, quels qu'ils soient.

Le "`flex-container`", qui définit le contexte global de modèle de boîte flexible, est tout simplement n'importe quel élément HTML doté de la déclaration `display: flex;` ou `display: inline-flex;`.

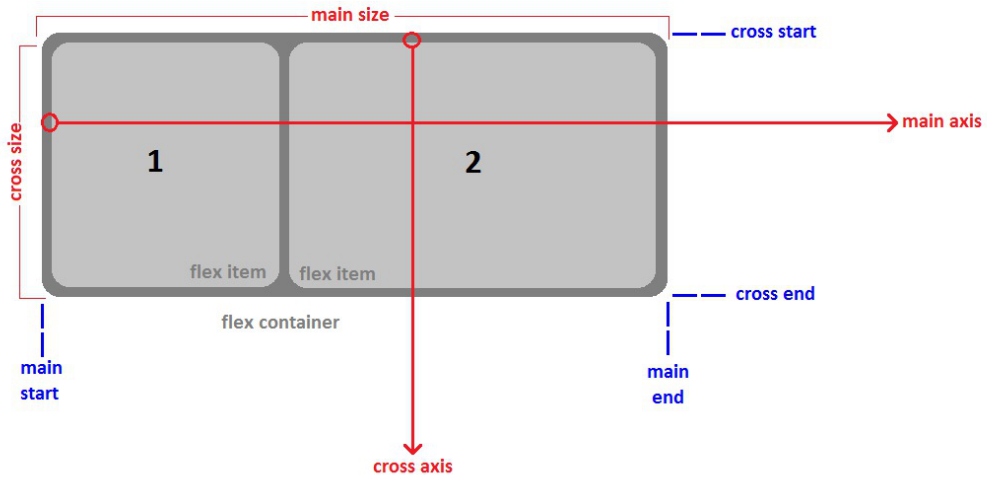
Ses enfants deviennent alors automatiquement (inutile de leur déclarer quoi que ce soit) des éléments de type "`flex-item`" :

```
1 .container {
2   display: flex;
3 }
```

Un élément "`flex-item`" n'est plus considéré comme un "`bloc`" ou un "`inline`" classique (d'ailleurs les valeurs de `display` autre que none, et même certaines propriétés telles que `float` n'ont plus d'effet sur lui).

Lorsqu'on travaille avec les boîtes flexibles, deux axes interviennent : *l'axe principal* (main axis en anglais) et *l'axe secondaire* (cross axis en anglais).

L'axe principal est défini par la propriété `flex-direction` et l'axe secondaire est l'axe qui lui est perpendiculaire. Tout ce que nous manipulons avec les boîtes flexibles fera référence à ces axes.



Source : https://www.supinfo.com/articles/resources/205357/813/0.png

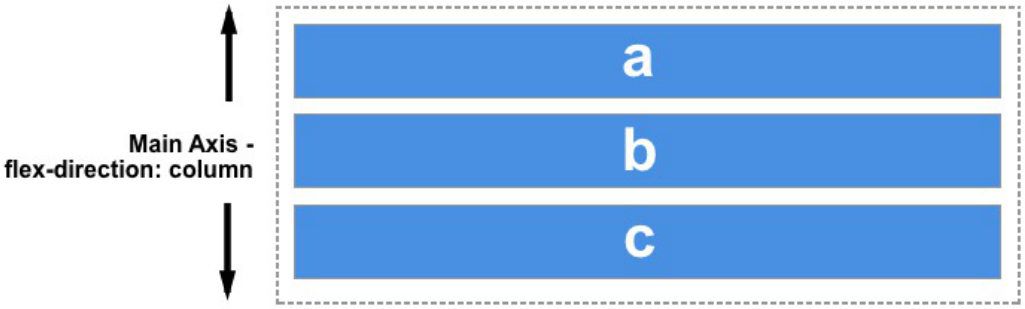
- *L'axe principal*

L'axe principal est défini par la propriété `flex-direction` qui peut prendre quatre valeurs :

- `row`
- `row-reverse`
- `column`
- `column-reverse`

Si on choisit la valeur `row` ou `row-reverse`, l'axe principal sera aligné avec la direction « en ligne » (inline direction) (c'est la direction logique qui suit le sens d'écriture du document).

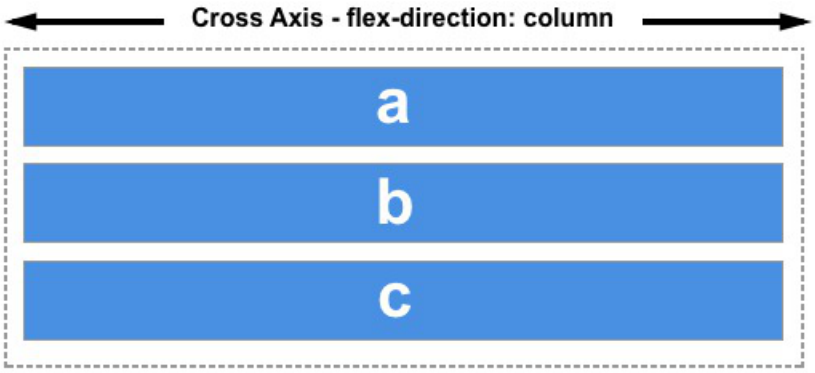
Si on choisit la valeur `column` ou `column-reverse`, l'axe principal suivra la direction de bloc (*block direction*) et progressera le long de l'axe perpendiculaire au sens d'écriture.



- L'axe secondaire (cross axis)

L'axe secondaire est perpendiculaire à l'axe principal. Ainsi, si `flex-direction` vaut `row` ou `row-reverse`, l'axe secondaire suivra l'axe des colonnes.

Si l'axe principale est `column` ou `column-reverse`, l'axe secondaire suivra celui des lignes (horizontales).



Comprendre les liens entre les différents axes est crucial lorsqu'on commence à aligner/justifier des éléments flexibles sur un axe ou l'autre grâce aux fonctionnalités et propriétés des boîtes flexibles.

2. 2. Flexbox en pratique

a) Distribution et axe principal

La distribution, c'est à dire le sens d'affichage horizontal ou vertical des éléments "`flex-items`" est définie par la propriété `flex-direction` dont les valeurs peuvent être :

- *row* (distribution horizontale, valeur par défaut)
- *row-reverse* (distribution horizontale inversée)
- *column* (distribution verticale)
- *column-reverse* (distribution verticale inversée)

Cette propriété s'applique au "`flex-container`" et détermine l'axe principal du modèle de boîte flexible.

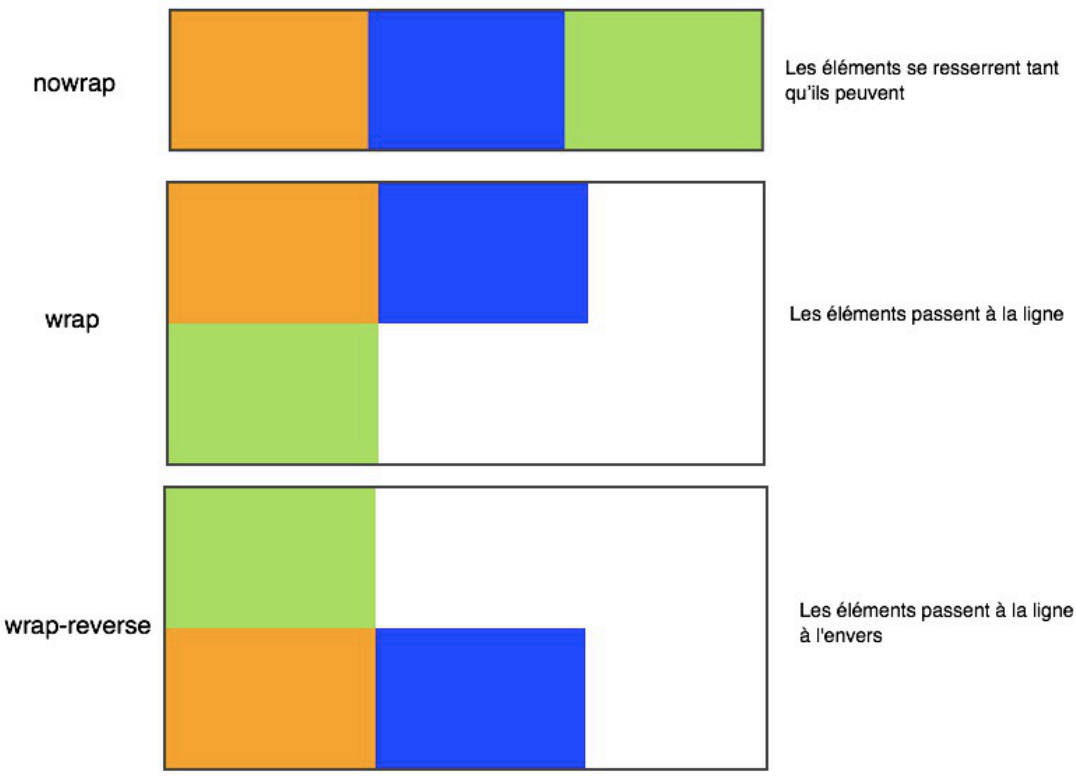
```
1 .container {
2   display: flex;
3   flex-direction: column;
4 }
```

La propriété `flex-wrap` définit si le contenu sera distribué 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.

Les valeurs de *flex-wrap* sont :

- `nowrap` (les éléments ne passent pas à la ligne, valeur par défaut)
- `wrap` (les éléments passent à la ligne dans le sens de lecture)
- `wrap-reverse` (les éléments passent à la ligne dans le sens inverse)

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`.



Source : OpenClassroom
(https://s3-eu-west-1.amazonaws.com/sdz-upload/prod/upload/flex_ wrap.png)

À noter qu'il existe une propriété raccourcie *flex-flow* qui regroupe `flex-direction` et `flex-wrap` .

```
/* affichage en ligne et passage à la ligne autorisé */
.container {
  flex-flow: row wrap;
}
```

b) Les alignements

Flexbox propose de gérer très finement les alignements et centrages, en différenciant les deux axes d'affichage de cette manière :

- *L'alignement dans l'axe principal* est traité via la propriété `justify-content`
- *L'alignement dans l'axe secondaire* est géré avec `align-items`

Ces deux propriétés s'appliquent au "`flex-container`".

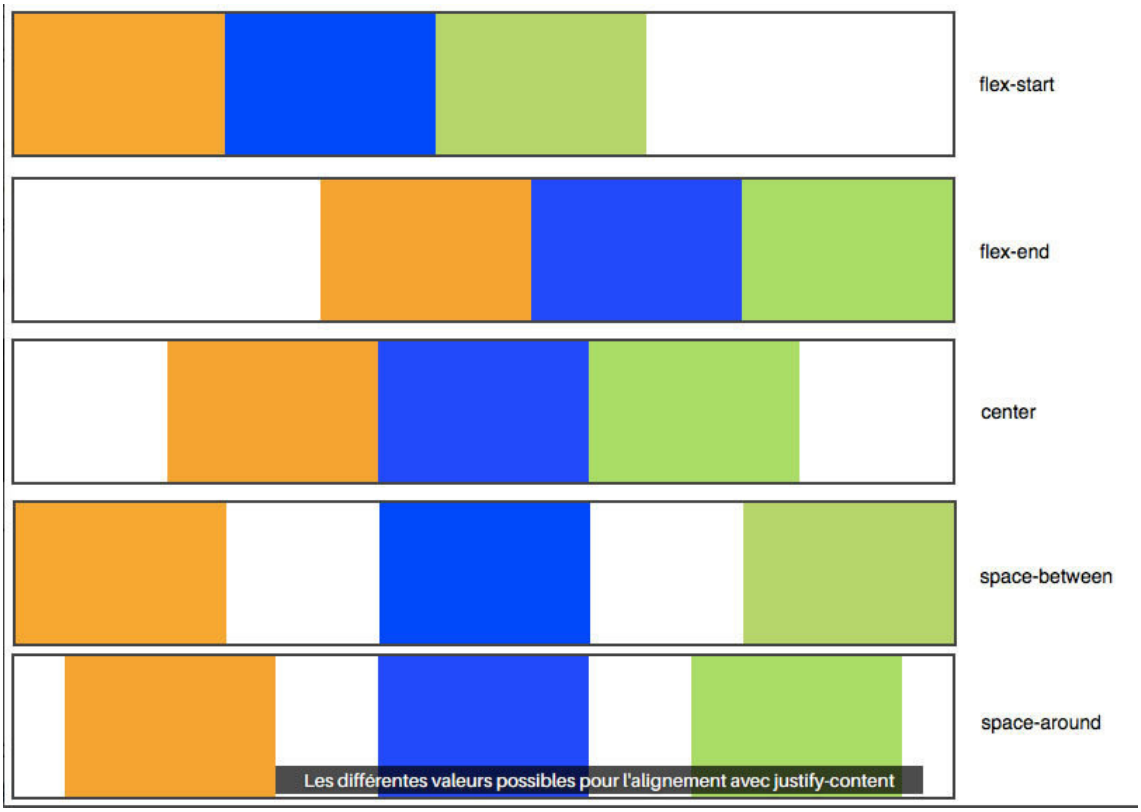
Les alignements dans l'axe de lecture principal sont définis à l'aide de la propriété `justify-content`, dont les valeurs possibles sont :

- `flex-start` (éléments positionnés au début du sens de lecture, valeur par défaut)
- `flex-end` (éléments positionnés à la fin)
- `center` (position centrale)
- `space-between` (répartition “justifiée”)
- `space-around` (variante de répartition “justifiée”

On peut écrire par exemple :

```
/* éléments positionnés en bas du conteneur */
.container{
  display: flex;
  flex-direction: column;
  justify-content: flex-end;
}

/* ou encore */
#conteneur{
    display: flex;
    justify-content: space-around;
}
```



Les différentes valeurs possibles pour l'alignement avec justify-content

Ca 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 */
}
```

Dans l'axe secondaire, les alignements sont régis via la propriété *`align-items`*, dont les valeurs sont :

- *`flex-start`* (au début)
- *`flex-end`* (à la fin)
- *`center`* (au centre)
- *`baseline`* (généralement identique à flex-start)
- *`stretch`* (étirés dans l'espace disponible, valeur par défaut)

Rappelons-nous que 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.

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;
}
```

La propriété *`align-self`* , permet de distinguer l'alignement d'un "*`flex-item`*" de ses frères. Les valeurs de cette propriété sont identiques à celles de *`align-items`*.

```
/* seul le paragraphe sera à droite */
.container {
  align-items: stretch;
}
p {
  align-self: flex-end;
}
```


```css
.parent {
   display: flex;
   flex-direction: column;
   align-items: stretch;
}
.oignon {
   align-self: flex-end;
}
```

La propriété *`margin`* lorsqu'elle est affectée à un "*`flex-item`*" ouvre de nouvelles perspectives, notamment dans l'axe vertical puisque *Flexbox* n'est plus lié à un sens de lecture en particulier.

En clair, il devient possible de positionner un élément en bas de son conteneur à l'aide d'un *`margin-top: auto`*, ou mieux : *centrer à la fois horizontalement et verticalement via un simple `margin: auto`.*

```css
/* paragraphe centré horizontalement et verticalement */
.container {
  display: flex;
}
.container > p {
  margin: auto;
}
```

c) L'ordonnancement avec Flexbox

L'une des fonctionnalités les plus avant-gardistes du modèle d'affichage Flexbox est de pouvoir réordonner à sa guise chacun des éléments indépendamment grâce à la propriété *`order`*.

Les valeurs de *`order`* agissent telles des pondérations : *les éléments dont la valeur est la plus forte se trouveront à la fin de la pile*. C'est-à-dire que les éléments sont triés du plus petit au plus grand nombre.

La propriété *`order`* s'applique aux "*`flex-items`*" et sa valeur initiale est *0*.

d) La flexibilité

Cela ne devrait étonner personne, la notion de *flexibilité* constitue le fondement du module de positionnement Flexbox, et c'est là qu'intervient l'indispensable propriété `flex`.

La propriété `flex` est un raccourci de trois propriétés, `flex-grow`, `flex-shrink` et `flex-basis`, qui s'appliquent aux "`flex-items`" et dont les fonctionnalités sont:

- *flex-grow :* capacité pour un élément à s'étirer dans l'espace restant,
- *flex-shrink :* capacité pour un élément à se contracter si nécessaire,
- *flex-basis :* taille initiale de l'élément avant que l'espace restant ne soit distribué.

Par défaut, les valeurs de ces propriétés 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.*

Pour rendre un élément flexible, il suffit de lui attribuer une valeur de `flex-grow` *(ou* `flex` *en raccourci) supérieure à zéro.*

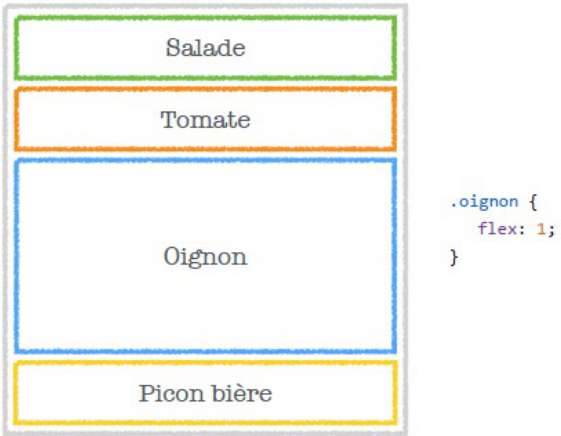
Cet élément occupera alors l'espace restant au sein de son conteneur :

/* .salade occupera l'espace restant */

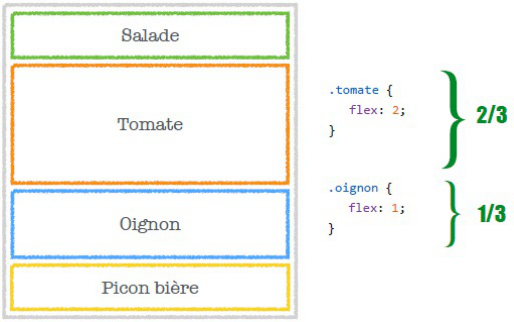
.oignon{

flex: 1;

}



Plusieurs éléments peuvent être rendus flexibles et se répartir l'espace restant. L'espace disponible est alors tout simplement distribué entre les éléments flexibles.



Rappel : 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.

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 ré-agencé 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.

Source : OpenClassroom , La mise en page avec Flexbox (Tutoriel)

3. Exercice : Quiz d'entrainement

Exercice

1) Quelle propriété de Flexbox existe vraiment ?

- ◯ re-flex
- ◯ corn-flex
- ◯ flex-wrap
- ◯ tarti-flex

Exercice

2) Quelle est l'orientation par défaut au sein d'un flex container ?

- ◯ verticale
- ◯ diagonale
- ◯ horizontale
- ◯ perpendiculaire
- ◯ contondante

Exercice

3) Quelle(s) déclaration(s) de Flexbox n'existe pas ?

- ☐ align-items: space-between;
- ☐ justify-content: space-between;
- ☐ align-content: space-between;
- ☐ display: inline-flex;
- ☐ align-self: auto;

Exercice

4) Si ces deux déclarations sont appliquées sur un flex item, quelle sera sa largeur ?

```
width:500px; flex-basis:250px;
```

- ◯ 750px
- ◯ 500px
- ◯ 250px
- ◯ 1250px
- ◯ auto

Exercice

5) Quelle est la valeur par défaut de flex-flow ?

- ◯ auto
- ◯ normal
- ◯ row nowrap
- ◯ row wrap
- ◯ row no-wrap