MODULE 03
FlexBox

420-C23-Web 1
Hiver 2026

Patrick Webster

FLEX

Introduction à FlexBox

FlexBox – « Flexible Box Layout »

Disposition avant FlexBox:

- Block
- Inline
- Table
- Float
- Positions Relatives/Absolues

FlexBox, pour Flexible Box Layout fait partie des modèles de disposition depuis CSS 3.

Permet de faire de la disposition:

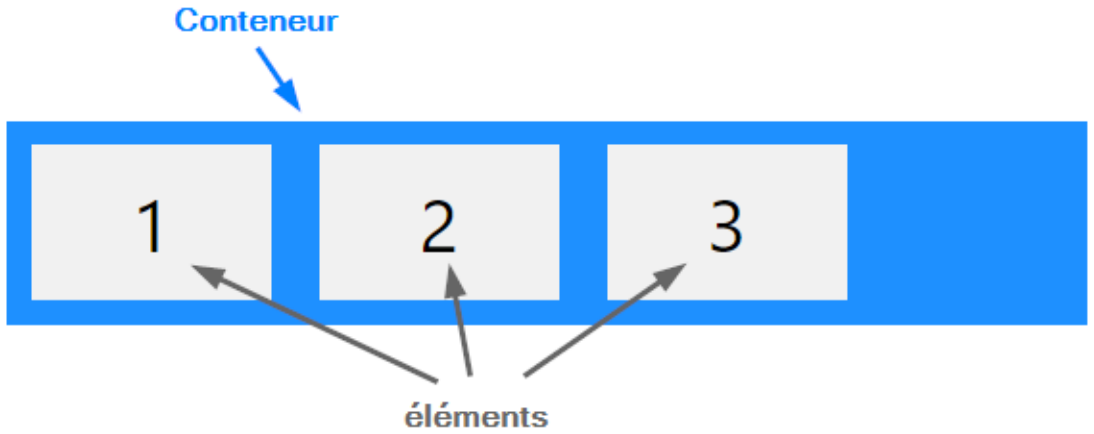
- Plus facilement/intuitivement
- Flexible (avec des éléments de taille variable ou dynamiques)
- Réactif
- Sans utiliser les positions ou float

FlexBox – Conteneur et Items

FlexBox n’est pas une propriété, mais un module de CSS qui comprend plusieurs propriétés.

Modèle de FlexBox:

- Un Conteneur
- Un à plusieurs Items



FlexBox – Fonctionnement général

Pour utiliser FlexBox, on définit un élément HTML comme conteneur de type FlexBox (le parent, souvent un div).

On définit ensuite comment les items à l'intérieur du FlexBox (les enfants) doivent être disposés.

› Il peut y a voir plusieurs conteneurs dans une page.

› Il peut y avoir des conteneurs dans des conteneurs.

FlexBox – Fonctionnement général

Propriétés pour conteneur

- display: flex
- justify-content
- align-items
- flex-direction
- flex-wrap
- align-content
- flex-flow

Propriétés pour item

- order
- align-self
- flex-grow
- flex-shrink
- flex-basis
- flex

FlexBox – démonstration – page de départ

1. Télécharger le document
   C23_M03_Documents_Étudiants_H26.zip (sur Léa)

2. Extraire le contenu dans un répertoire sur votre X:

3. Dans le dossier Demos\D03A_Demo-FlexBox-Intro,
   ouvrir le fichier d03a-demo-flex-intro.html
   avec VS Code.

FlexBox - démonstration

On introduit le fonctionnement de base du FlexBox à l'aide d'une page d'exemple sur laquelle on appliquera les attributs de FlexBox pour en voir les résultats.

On démarre avec un conteneur (un div) dans laquelle on insère 3 items (aussi des div), le tout avec des couleurs d'arrière-plan différentes pour visualiser les résultats.

Chacun des 3 éléments a une classe commune (sections) et une classe qui lui est unique (section1, section2 et section3)

Page de départ – code 1 de 2

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>
            .conteneur_page{
                background-color: whitesmoke; }
            .sections{
                width: 200px; }
            .section1{
                background-color: lightgreen; }
            .section2{
                background-color: aqua; }
            .section3{
                background-color: lightblue; }
        </style>
    </head>
```

Page de départ – code 2 de 2

```
 <body>
        <div class="conteneur_page">
            <div class="sections section1">
                Section 1
            </div>
            <div class="sections section2">
                Section 2
            </div>
            <div class="sections section3">
                Section 3
            </div>
        </div>
    </body>
</html>
```

Page de départ - résultat

On obtient la page suivante:



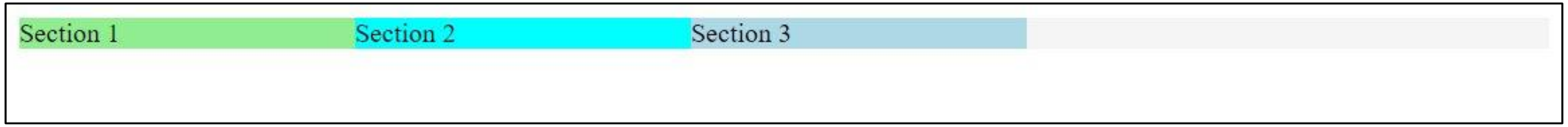
Définir un conteneur comme FlexBox

Utiliser display: flex pour définir le conteneur principal comme flex.

Modifiez le code:

```
.conteneur_page{
        background-color: whitesmoke;
        display: flex;
}
```

Résultat:



Définir un conteneur comme FlexBox

Par défaut:

- Les éléments sont disposés côte-à-côte.
- Les éléments sont disposés en partant de la gauche
- Il n’y a pas d’espaces entre les éléments

Comportement horizontal

Si les éléments sont plus larges que la page, ils restent en ligne par défaut.

Modifiez le code pour avoir des éléments dont la largeur totale est plus grande que la largeur de la page:

```
.sections{
        width: 600px;
}
```

Résultat:



Les éléments restent en ligne et dans leur rangée. On voit plus loin comment autoriser les items à changer de ligne.

L’attribut justify-content

On spécifie comment les éléments enfants sont justifiés à **l’horizontale** avec justify-content.

D’abord on change la largeur des éléments. Modifiez le code:

```
.sections{
	width: 300px;
}
```

Centrer horizontalement

Pour centrer les éléments avec justify-content, modifiez le code:

```
.conteneur_page{
	background-color: whitesmoke;
	display: flex;
	justify-content: center;
}
```

Résultat:



Espacer les éléments

FlexBox offre une valeur de justify-content pour ajouter un espacement entre les éléments et les extrémités.

Modifiez le code:

```
.conteneur_page{
        background-color: whitesmoke;
        display: flex;
        justify-content: space-around;
}
```

Résultat:



Note: un espacement égal est ajouté de chaque côté de chaque élément. Les espaces entre les éléments sont donc 2 fois la largeur des espaces aux extrémités. La valeur space-evenly ajoute un espace égal entre les éléments ET aux extrémités.

Justification à la verticale

L'attribut align-items permet de contrôler la justification à la **verticale** des éléments.

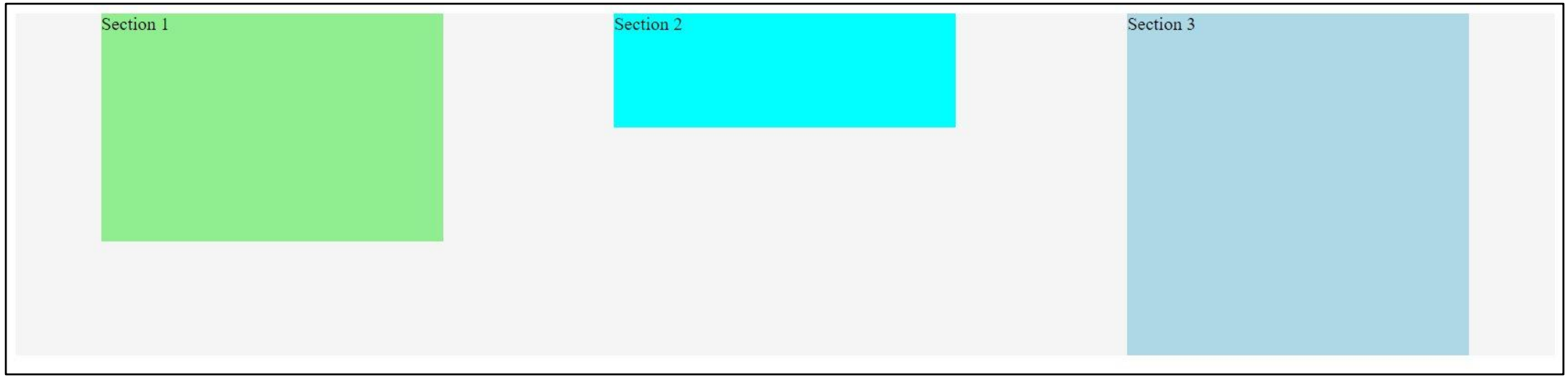
D'abord, on donne une hauteur différente aux éléments, modifiez le code:

```
.section1{
	background-color: lightgreen;
	height: 200px;
}
.section2{
	background-color: aqua;
	height: 100px;
}
.section3{
	background-color: lightblue;
	height: 300px;
}
```

Justification à la verticale – en haut

On obtient le résultat ci-dessous.

Par défaut les éléments sont justifiés en haut (défaut = flex-start)

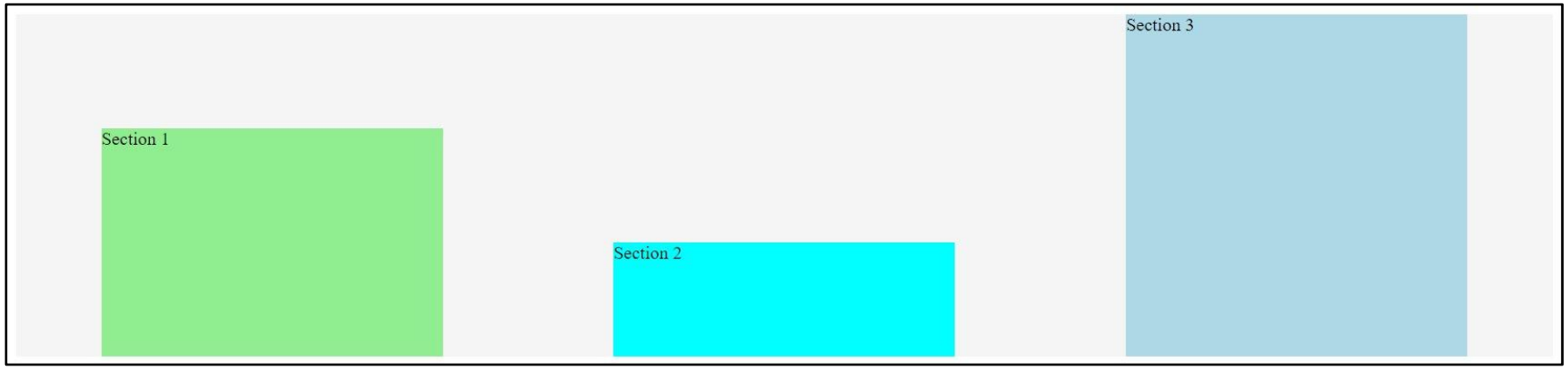


Justification à la verticale – en bas

Pour aligner les éléments en bas, modifiez le code:

```
.conteneur_page{
        background-color: whitesmoke;
        display: flex;
        justify-content: space-around;
        align-items: flex-end;
}
```

Résultat:

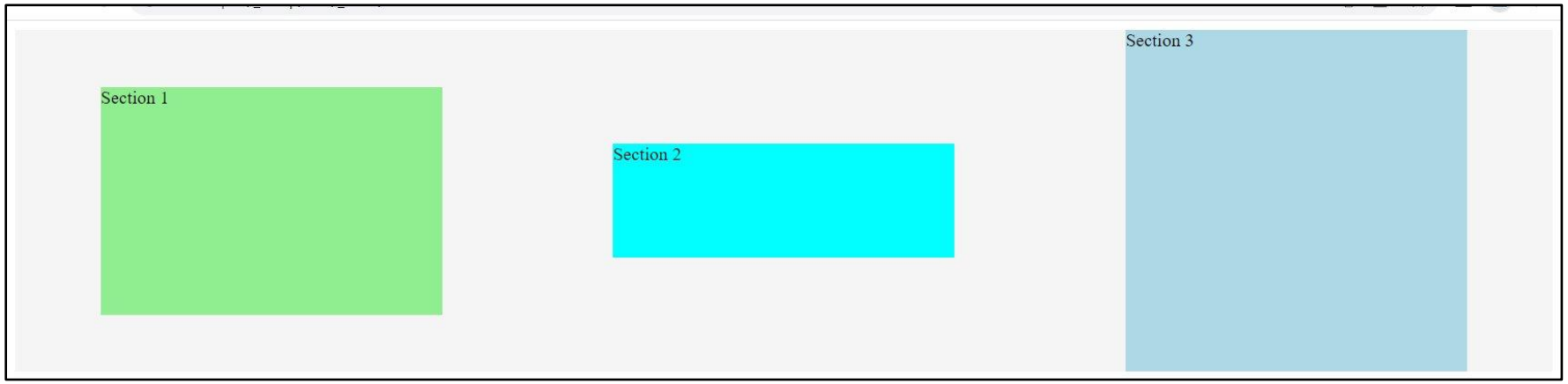


Justification à la verticale – centré

Pour centrer les éléments verticalement, modifiez le code:

```
.conteneur_page{
        background-color: whitesmoke;
        display: flex;
        justify-content: space-around;
        align-items: center;
}
```

Résultat:



Justification à la verticale - baseline

Avec l'autocomplete, on voit qu'il y a une valeur « baseline » qu'on peut attribuer au lieu de center ou flex-end, etc.

Faites une recherche par vous-même pour trouver ce que veut dire « baseline » comme valeur d'un attribut FlexBox.

Réponse :

Ordre des éléments

On peut contrôler **l'ordre** dans lequel apparaissent les éléments dans le conteneur.

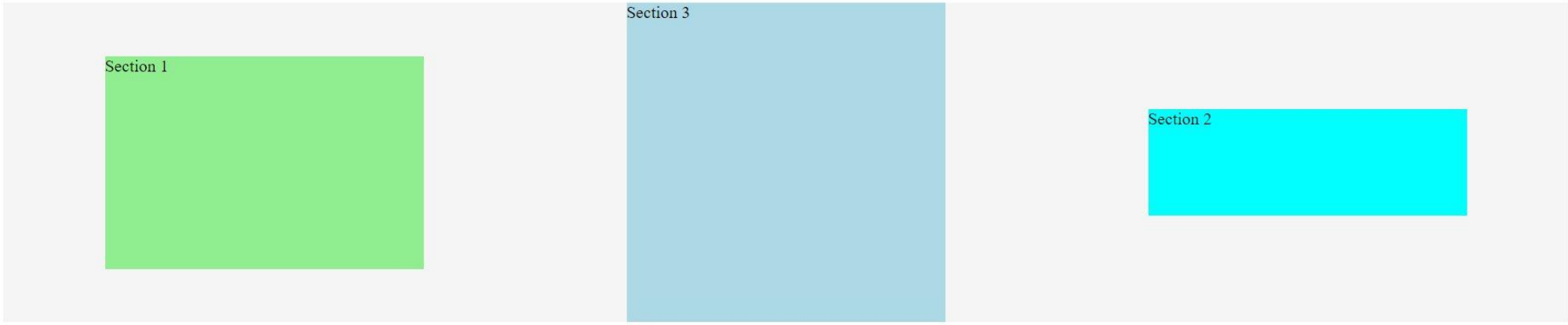
Par défaut ils sont affichés dans le même ordre où ils apparaissent dans le code.

On utilise l'attribut order sur l'élément enfant pour en forcer l'ordre d'affichage. Ici on force la section 2 à être affichée en position 1.

```
.section2{
        background-color: aqua;
        height: 100px;
        order: 1;
}
```

Ordre des éléments (suite)

Résultat:



Mais attendez.. la section 2 est à la fin? Par défaut, les éléments du conteneur ont un ordre de 0. On peut utiliser un ordre négatif pour déplacer un élément vers la gauche.

Justification d'un seul élément

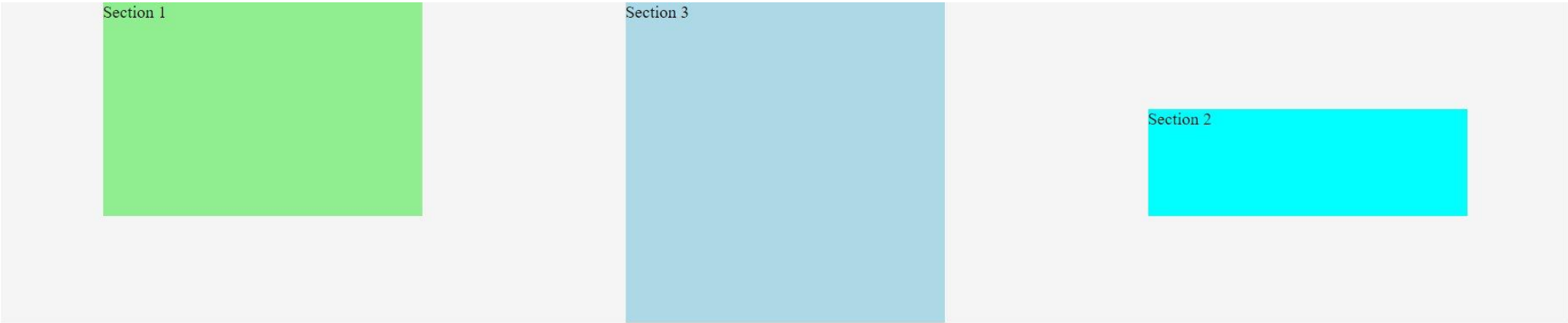
On peut contrôler l'alignement vertical d'un item indépendamment des autres. Ici, on utilise la propriété align-self sur l'élément enfant pour aligner la section 1 en haut alors que les autres éléments sont centrés.

D'abord, centrez verticalement les éléments, ensuite alignez seulement la section1 en haut:

```
.conteneur_page{
	background-color: whitesmoke;
	display: flex;
	flex-wrap: wrap;
	justify-content: space-around;
	align-items: center;
}
…
 .section1{
	background-color: lightgreen;
	height: 200px;
	align-self: start;
}
```

Justification d’un seul élément(suite)

Résultat:



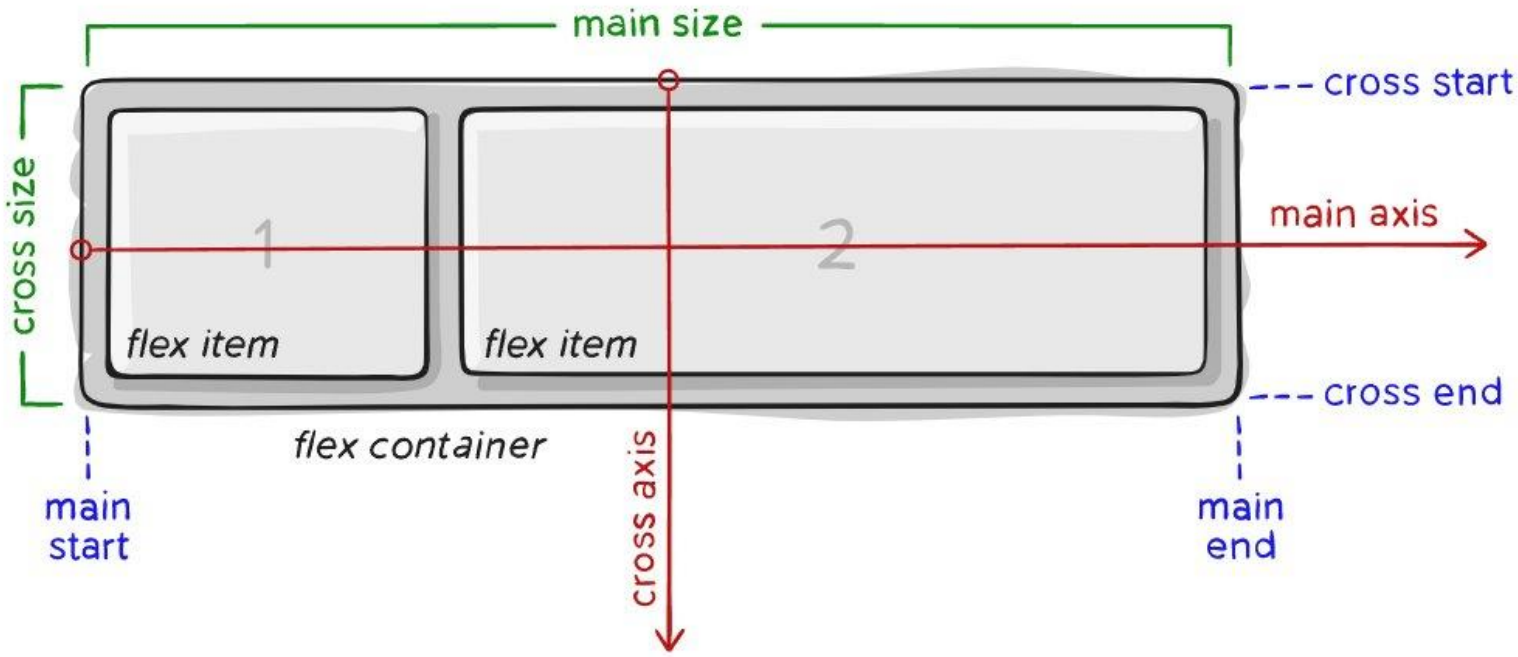
Dans un conteneur avec éléments centrés verticalement, la section 1 est centré en haut.

Orientation du FlexBox

FlexBox - direction

À ce point-ci, on a utilisé FlexBox comme un contenant horizontal avec orientation de gauche à droite et de haut en bas. On peut changer l'orientation par défaut d'un conteneur FlexBox.

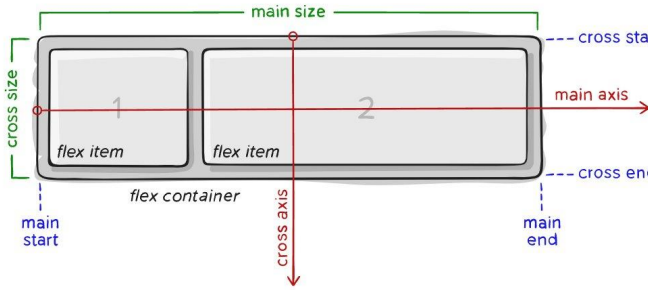
On ne décrit alors plus les directions comme gauche-droite et haut-bas, mais en termes d'axe principal (main axis) et d'axe secondaire (cross.axis).



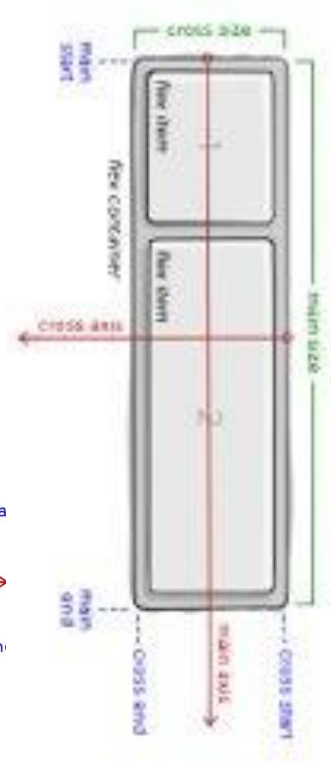
Justification à la verticale - centré

Quatre sens

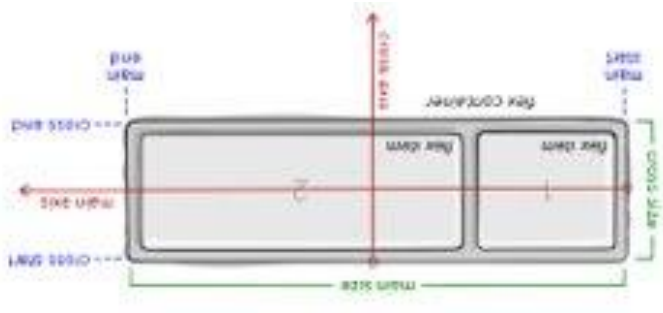
› Propriété:
  – flex-direction



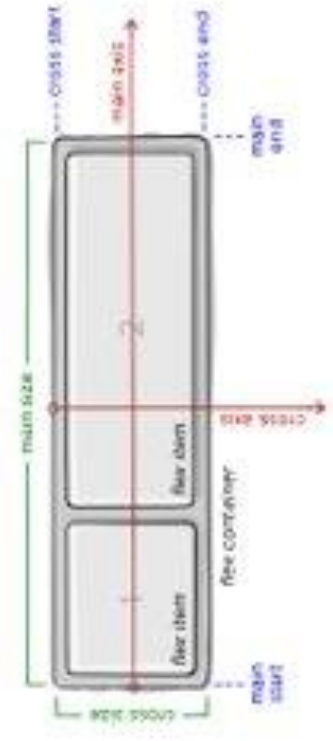
row



column



row-reverse



column-reverse

FlexBox - directions
Démo 03B

Démo 3B

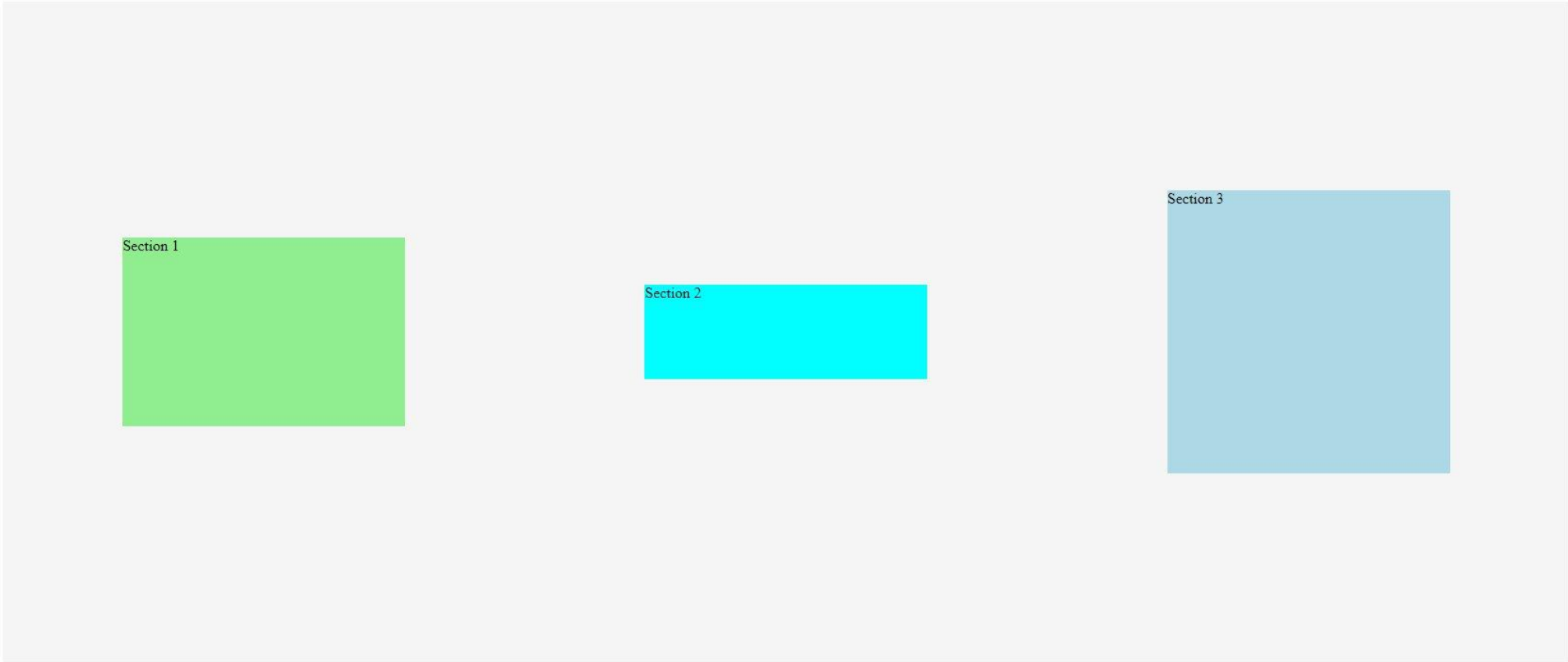
Dans le répertoire Demos\D03B_Demo-FlexBox-Orientation, ouvrez le fichier d03b-demo-flex-orientation.html dans VS Code

La disposition est basée sur le travail fait dans la démo précédente.

On change ensuite la direction du conteneur et ses propriétés pour constater l'effet d'en changer la direction.

Page de départ

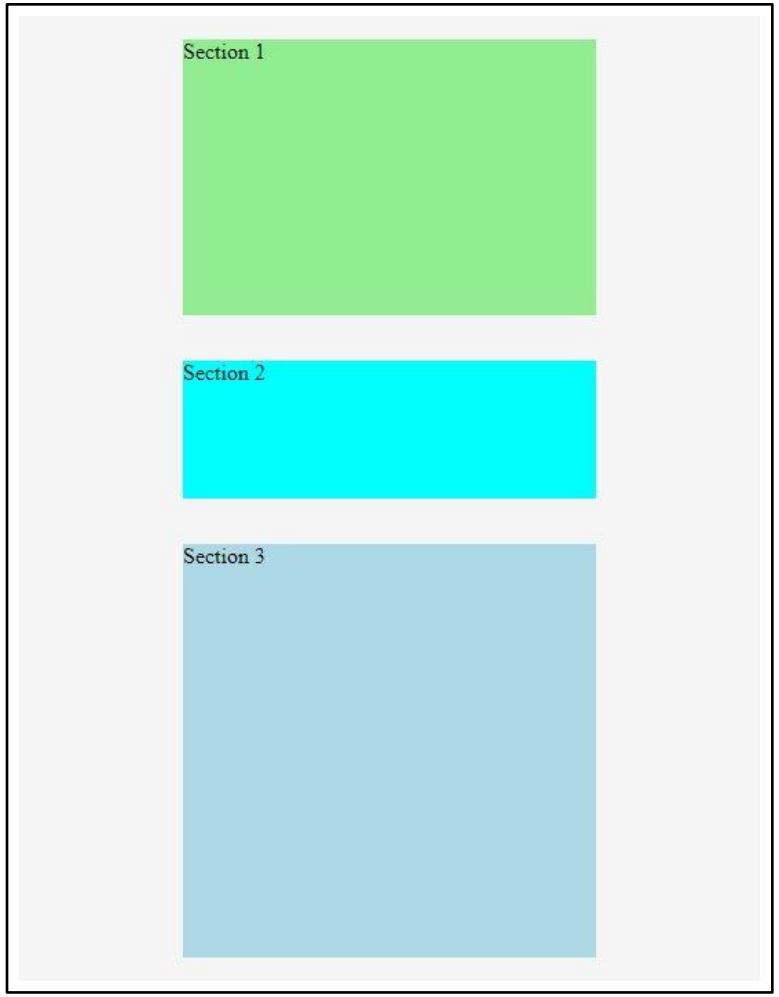
d03b-demo-flex-orientation.html



Changement de direction 1

Pour voir l'effet d'une autre direction au FlexBox, changez le code pour:

```
.conteneur_page{
        background-color: whitesmoke;
        display: flex;
        flex-direction: column;
        justify-content: space-around;
        align-items: center;
        height: 700px;
}
```



Le contenant aligne maintenant les items à la verticale.

Changement de direction - propriétés

Dans sa direction par défaut (flex-direction:row;) justify-content: flex-start; avait pour effet d'aligner les items à gauche.

L'axe principal du FlexBox est maintenant de haut en bas, quel sera l'effet de justify-content: flex-start; sur la page maintenant?

Pour la classe .conteneur_page, changez successivement les valeurs de propriétés suivantes et verifiez le résultat d'une fois à l'autre. Aviez-vous bien deviné?

1. justify-content: flex-start;
2. justify-content: flex-end;
3. justify-content: space-between;
4. justify-content: space-around; (retours à la configuration initiale)
5. align-items: flex-start;
6. align-items: flex-end;
7. Align-items: stretch;

FLEX

Laboratoire 03A

Réalisez le laboratoire L03A-FlexBox-Disposition.

Flex-grow, flex-shrink, flex-basis

Flex-grow, flex-shrink, flex-basis

Ces propriétés définissent comment un item peut changer de taille dans un conteneur Flex.

Elles s'appliquent à chacun des items.

Pour la suite démo, ouvrez le fichier d03c-demo-flex-reactif.html dans VS Code.

Flex-grow

flex-grow définit comment un item peut s'étirer dans un contenant flex, si il y a de l'espace libre.

Une valeur de 0 signifie qu'il garde sa taille par défaut.

Si un item est le seul item avec une valeur flex-grow > 0, il occupera tout l'espace disponible.

Si plusieurs items ont flex-grow > 0, ils se partageront l'espace libre proportionnellement à leur valeur.

Flex-grow - exemple

Au départ, la page n'a pas de propriété flex-grow de définie.

Tous les items ont donc une valeur par défaut de 0, conservent leur taille par défaut (la taille de leur contenu), et l'espace libre est répartit selon justify-content: space-around;

Ajoutez un flex-grow de 1 au deuxième item.

```
.section2{
        background-color: aqua;
        flex-grow: 1;
}
```



L'item occupe tout l'espace disponible.

Flex-grow – exemple 2

Ajoutez un flex-grow: 1
aux 2 autres items:

```
.section1{
      background-color: aqua;
      flex-grow: 1;     }
.section3{
      background-color: aqua;
      flex-grow: 1;     }
```

Les items se partagent l'espace libre à part égale – ils ont la même valeur de flex-grow.
Notez que les items n'ont pas la même largeur. C'est plus visible dans l'exemple suivant.



Flex-grow – exemple 3

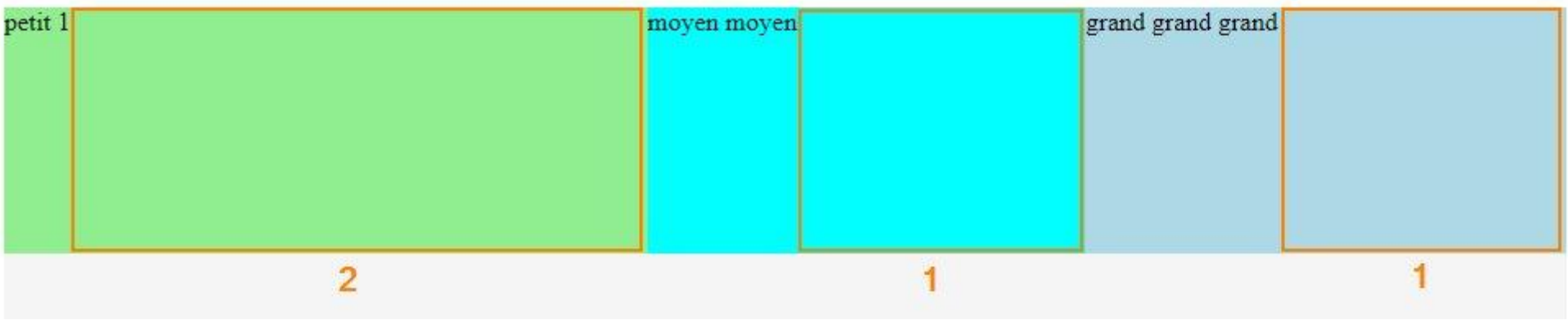
Changez le premier item avec un flex-grow: 2

```
.section1{
      background-color: aqua;
      flex-grow: 2;    }
```

Chaque item s'est étiré d'une taille proportionnelle à leur valeur de flex-grow.

L'item 1 a 2 parts des 4 parts disponibles.

C'est l'ajout à la taille de l'élément qui représente sa part d'espace libre, si les éléments n'ont pas la même taille au départ, ils seront de tailles différentes après.



Flex-grow – exemple 4

Si on veut que les items aient une taille égale, on doit leur donner une taille de départ égale. Changez le code pour donner une taille de 300px aux items et une valeur flex-grow de 1 à chacun.

```
.section1{
        background-color: lightgreen;
        width: 300px;
        flex-grow: 1;            }
 .section2{
        background-color: aqua;
        width: 300px;
        flex-grow: 1;            }
```

```
.section3{
        background-color: lightblue;
        width: 300px;
        flex-grow: 1;            }
```

Les items ont maintenant une taille égale.



Flex-shrink

flex-shrink définit comment les items peuvent se compresser si la taille du conteneur parent est insuffisant pour tous les contenir.

La valeur par défaut de flex-shrink est 1 => peut réduire de taille.

Modifiez le code pour donner une largeur de 600px à tous les items:

```
 .section1{
        background-color: lightgreen;
        width: 600px;
        flex-grow: 1;           }

.section2{
        background-color: aqua;
        width: 600px;
        flex-grow: 1;           }
.section3{
        background-color: lightblue;
        width: 600px;
        flex-grow: 1;           }
```

Flex-shrink – exemple 1 - defaut

Modifiez la largeur de votre navigateur (plus large et plus étroit) et constatez que:

les items apparaissent toujours sur toute la largeur

- Les items rapetissent au besoin
- La valeur par défaut, flex-grow: 1, autorise la compression.

Flex-shrink – exemple 2

Modifiez le code pour ajouter flex-shrink à 0 (compression non autorisée):

```
.section1{
	background-color: lightgreen;
	width: 600px;
	flex-grow: 1;
	flex-shrink: 0;		}
.section2{
	background-color: aqua;
	width: 600px;
	flex-grow: 1;
	flex-shrink: 0;		}
.section3{
	background-color: lightblue;
	width: 600px;
	flex-grow: 1;
	flex-shrink: 0;		}
```

Flex-shrink – exemple 2 - résultat

Trop c’est trop!

Le navigateur n’arrive plus à s’ajuster, les items font plus que la largeur de l’écran, ne sont pas autorisés à se compresser.

On a une barre défilement pour défiler de droite à gauche, mais le contenu dépasse même cette largeur, des 2 côtés.



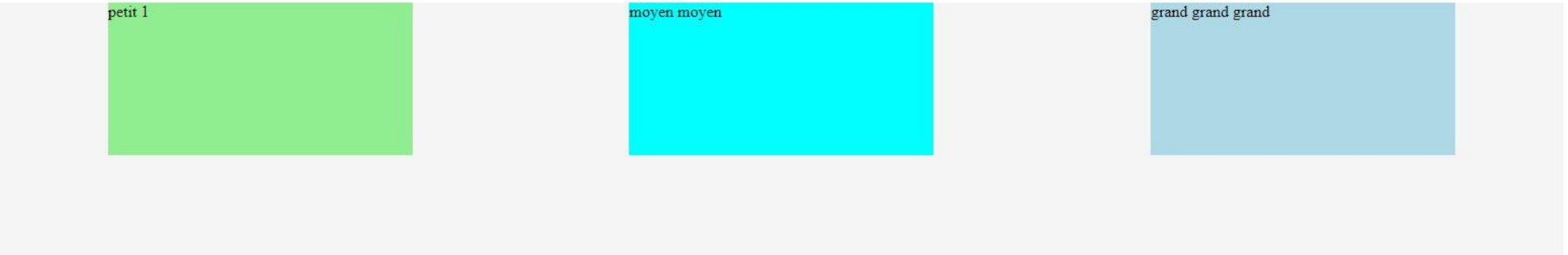
Flex-shrink – exemple 3

Modifiez le code pour interdire aux items de s'étirer, permettre aux items de se compresser et avoir une taille de 400px:

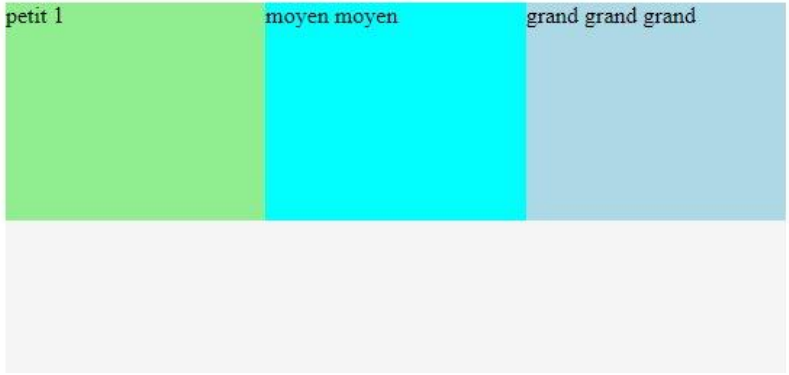
```
.section1{
        background-color: lightgreen;
        height: 150px;
        width: 400px;
        flex-grow: 0;
        flex-shrink: 1;          }
.section2{
        background-color: aqua;
        height: 150px;
        width: 400px;
        flex-grow: 0;
        flex-shrink: 1;          }
.section3{
        background-color: lightblue;
        height: 150px;
        width: 400px;
        flex-grow: 0;
        flex-shrink: 1;          }
```

Flex-shrink – exemple 3 - résultat

On retrouve l'effet du space-around appliqué et les items font exactement 400px.



Réduisez la taille du navigateur pour voir éventuellement l'espace libre disparaitre, puis réduisez encore pour constater que les items réduisent de taille, MOINS que leur 400px assigné pour occuper l'espace de façon égale.



Flex-basis

flex-basis définit la taille souhaitée, ou idéale qu'on désire donner à un item.

Sa valeur par défaut est auto, ou ajusté à la taille du contenu.

Le rôle de flex-basis est presque le même que que width, avec des différences:

- Flex-basis s'applique seulement aux éléments d'un contenant flex.
- Flex-basis est appliqué sur l'axe principal d'un contenant flex. Dans le cas de flex-direction: column; par exemple, width et flex-basis ne s'appliquent pas à la même orientation de l'élément et correspondrait plutôt à la propriété height.
- Il y a d'autres différences, par exemple lorsqu'un item est absolument positionné (pas couvert dans ce cours) et d'autres situation.

On utilise flex-basis lorsque le contrôle de la taille des items dans un conteneur flex, sur l'axe principal, est l'objectif à atteindre.

Flex-basis – exemple 1

Changer le code pour définir la largeur des items avec flex-basis plutôt que width:

```
.section1{
        background-color: lightgreen;
        height: 150px;
        width: 400px;
        flex-grow: 0;
        flex-shrink: 1;
        flex-basis: 400px;          }
.section2{
        background-color: aqua;
        height: 150px;
        width: 400px;
        flex-grow: 0;
        flex-shrink: 1;
        flex-basis: 400px;          }
.section3{
        background-color: lightblue;
        height: 150px;
        width: 400px;
        flex-grow: 0;
        flex-shrink: 1;
        flex-basis: 400px;          }
```

… et constatez simplement que le résultat est le même.

Flex – forme raccourcie

On a vu séparément les propriétés **flex-grow**, **flex-shrink** et **flex-basis**.

Mais il est recommandé, lorsqu'approprié, d'utiliser la syntaxe courte dont la forme est:

flex: valeur_grow valeur_shrink valeur_basis;

Donc, flex: 0 1 400px; représente l'état des propriétés de la page de démo actuelle.

La syntaxe du code sera:

```
.section1{
        background-color: lightgreen;
        height: 150px;
        flex: 0 1 400px;          }
.section2{
        background-color: aqua;
        height: 150px;
        flex: 0 1 400px;          }
.section3{
        background-color: lightblue;
        height: 150px;
        flex: 0 1 400px;          }
```

FLEXBOX FROGGY

FLEXBOX FROGGY est un petit jeu en ligne qui permet de pratiquer les propriétés de flex. Vous devez positionner des grenouilles (éléments flex) sur des nénuphars avec les propriétés appropriées.

Faites les niveaux 1 à 17 inclusivement.

Suivez le lien https://flexboxfroggy.com/#fr



Défis FlexBox

Les exercices de défis consistent à trouver une solution avec FlexBox pour résoudre un problème de disposition d'éléments dans une page web.

Des fichiers web et css sont fournis avec un début de code.

On fait chaque défi avec une limite de temps et on discute de la solution en groupe.

Flex-Wrap

Démo 3D – flex-wrap

Ouvrez le fichier d03d-demo-flex-wrap.html dans VS Code

On commence avec un contenant Flex avec 5 éléments carrés:

- 150px de haut
- flex-basis de 150px (largeur)
- justify-content est configuré en space-around
- flex-grow et flex-shrink sont laissés par défaut, peut rapetisser mais par grandir.

Essayez la page en modifiant la largeur du navigateur pour voir les éléments se rapprocher, puis rapetisser.

La propriété flex-wrap

L'attribut flex-wrap permet de définir si les éléments peuvent changer de ligne si la somme de leur largeur excède la largeur de l'espace affichable.

Les valeurs possibles sont:

- wrap — les éléments changent de ligne
- no-wrap — les éléments ne peuvent pas changer de ligne
- wrap-reverse — les éléments changent de ligne sur la ligne en haut

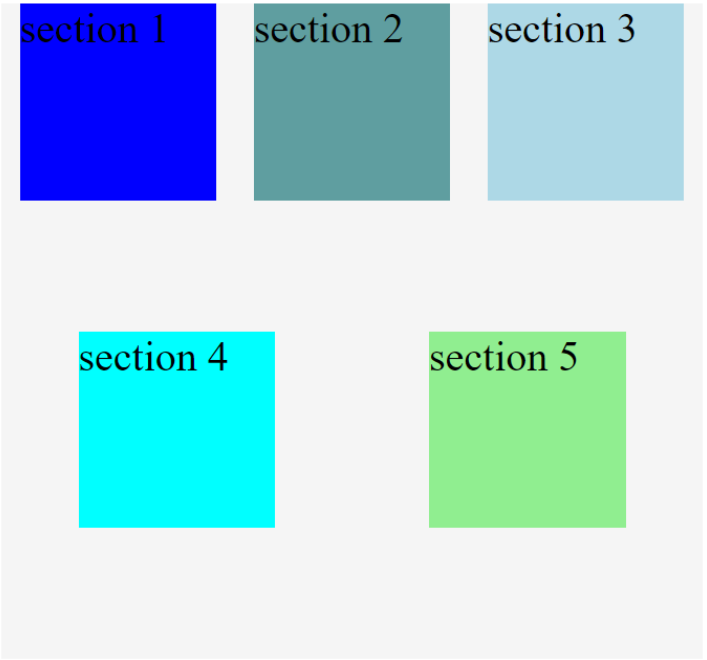
Changement de ligne automatique

L'attribut flex-wrap permet de définir comment un élément doit se comporter s'il dépasse le bout de la page.

Modifiez le code:

```
.conteneur_page{
    background-color: whitesmoke;
    height: 500px;
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}
```

Modifiez la largeur du navigateur. Résultat:



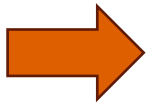
Notes:

1. la valeur par défaut de flex-wrap est: no-wrap
2. Le mécanisme wrap est appliqué avant le flex-shrink, flex-shrink n'aura un effet que lorsqu'il ne reste qu'un élément sur une rangée
3. À la verticale, la première rangée est alignée en haut et les autres de façon à partager l'espace libre également.

flex-wrap avec justify-content

Testez le comportement de flex-wrap avec justify-content. Modifiez le code:

```
.conteneur_page{
    background-color: whitesmoke;
    height: 500px;
    display: flex;
    justify-content: flex-start;
    flex-wrap: wrap;
}
```



Essayez les valeurs suivantes pour justify-content:

- flex-end
- center
- space-between
- space-evenly
- space-around (retours à la configuration initiale)

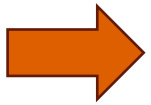
Note:

Avec flex-wrap: wrap, les valeurs de justify-content s'applique à chaque rangée indépendamment.

flex-wrap avec align-items

Testez le comportement de flex-wrap avec align-items. Modifiez le code:

```
.conteneur_page{
    background-color: whitesmoke;
    height: 500px;
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    align-items: flex-start;
}
```



Essayez les valeurs suivantes pour align-items:

- flex-end
- center

Note:

Avec flex-wrap: wrap, les valeurs de align-items préservent l'espace vertical entre les rangées d'items.

La propriété align-content

La propriété align-content définit le comportement de l'espace vertical entre les rangés d'un conteneur avec flex-wrap actif. Il n'y a aucun effet lorsque flex-wrap est configuré à no-wrap.

```
.conteneur_page{
    background-color: whitesmoke;
    height: 500px;
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    align-items: flex-start;
    align-content: normal;
}
```

Essayez les valeurs suivantes pour align-content:

- flex-start
- flex-end
- center
- space-around
- space-between
- space-evenly

Note:

Le comportement de align-items, est différent de align-content, entre autres il traite l'espace libre de façon différente.

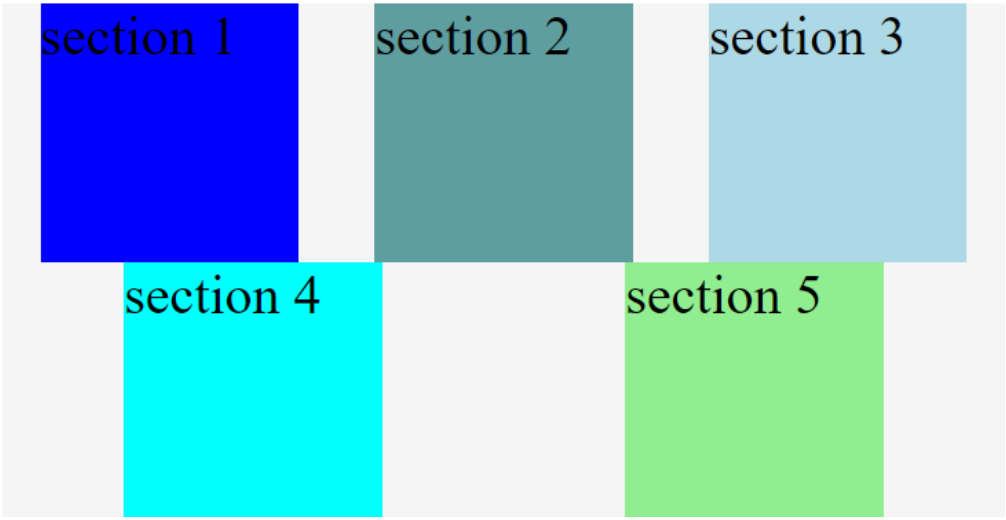
flex-wrap avec hauteur de conteneur variable

Jusqu'à maintenant, on a fonctionné avec un conteneur de hauteur fixe (500px) pour démontrer la gestion de l'espace à la verticale.

Pour les prochaines étapes on retire la hauteur fixe du conteneur pour que sa taille à la verticale s'ajuste au contenu (les rangées ont la hauteur des éléments, dans ce cas-ci, 150px.

Modifiez le code:

```
.conteneur_page{
    background-color: whitesmoke;
    height: 500px;
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    align-content: normal;
}
```



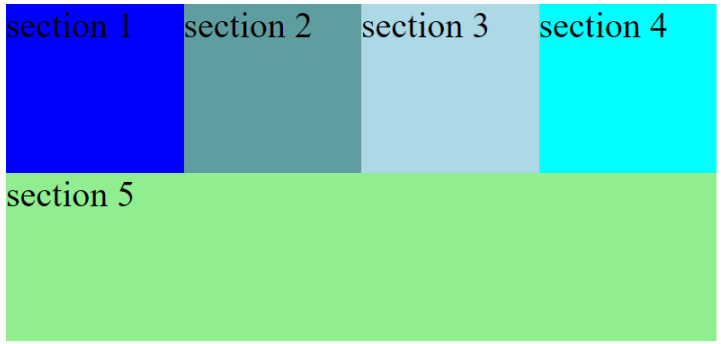
flex-wrap avec flex-grow

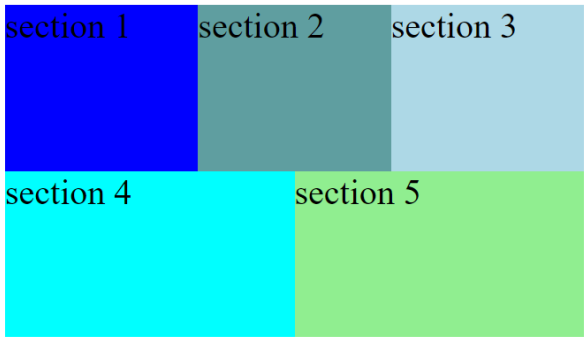
Le code actuel définit une valeur de flex-basis de 150px (largeur idéale 150px) et on n'a pas défini de flex-shrink ni de flex-grow, les valeurs par défaut sont donc utilisées: peut rapetisser, ne peut pas grandir.

On vérifie l'effet de permettre aux éléments de s'élargir, combiné avec flex-wrap: wrap.

Modifiez le code:

```
.sections{
    height: 150px;
    flex-basis: 150px;
    flex: 1 1 150px;
    font-size: 2em;
}
```





Il n'y a plus d'espace horizontal entre les éléments, on obtient un résultat parfaitement tuilé.

Les sections ne sont plus nécessairement carrées.

Gestion de l'espace avec padding/margin

On peut choisir de gérer l'espace entre et autours des éléments avec les propriétés margin et padding.

Dans l'exemple qui suit, on ajoute un padding au conteneur et des margin aux éléments.

```
.conteneur_page{
    background-color: whitesmoke;
    display: flex;
    justify-content:space-around;
    flex-wrap: wrap;
    padding: 0.5em;
}
```

```
.sections{
    height: 150px;
    flex: 1 1 150px;
    font-size: 2em;
    margin: 0.5em;
}
```

On a choisi une taille en em pour en démontrer l'utilisation.

Observez le résultat pour différentes largeurs de navigateur.

C'est maintenant l'espacement qui est géré et la taille des éléments s'y ajuste.

Taille minimum et maximum

flex-shrink et flex-grow permettent un comportement réactif pour les éléments d'un contenant flex.

Il arrive parfois qu'il y ait une limite à ne pas dépasser pour conserver une aspect visuel esthétique: pas trop petit et/ou pas trop grand.

Les propriétés min-width, max-width, min-height et max-height permettent de donner une taille minimum/maximum à respecter pour des éléments html. Incluant des éléments de conteneur flex.

On peut utiliser ces propriétés en combinaison avec height/width ainsi que flex-basis. Combinés avec flex-shrink et flex-grow, on obtient un comportement réactif avec une taille minimum et/ou maximum.

Gestion de taille idéale, min et max

On créé un exemple où l'objectif est le suivant: les éléments disposés 3 par rangée, pas plus petit que 300px, ni plus grand que 400px. De plus, on veut que les items soient alignés à gauche (pas centrés).

Modifiez le code:

```
.conteneur_page{
    background-color: whitesmoke;
    display: flex;
    justify-content: flex-start;
    flex-wrap: wrap;
    padding: 2%;
}
```

```
.sections{
    height: 150px;
    flex: 1 1 28%;
    font-size: 2em;
    margin: 2%;
}
```

Avec les tailles en %, la première rangée a toujours 3 éléments. Mais pour l'instant ils peuvent réduire et grandir sans limites.

Limiter la largeur

On limite maintenant les changements de largeur.

Modifiez le code:

```
.sections{
    height: 150px;
    flex: 1 1 28%;
    font-size: 2em;
    margin: 2%;
    min-width: 300px;
    max-width: 400px;
}
```

Modifiez la taille du navigateur pour voir les résultats. Il y a des avantages et des désavantages à cette combinaison spécifique de propriétés.

Rappel: les propriétés min-width, max-width, min-height et max-height peuvent être appliquées à toutes sorte d'éléments, pas seulement les éléments d'un contenant flex.

Les propriétés gap

On peut aussi gérer l'espacement entre les éléments et entre les rangées (si il y a wrap) avec les propriétés suivantes:

- column-gap
- row-gap
- gap (version raccourcie des 2 précédentes)

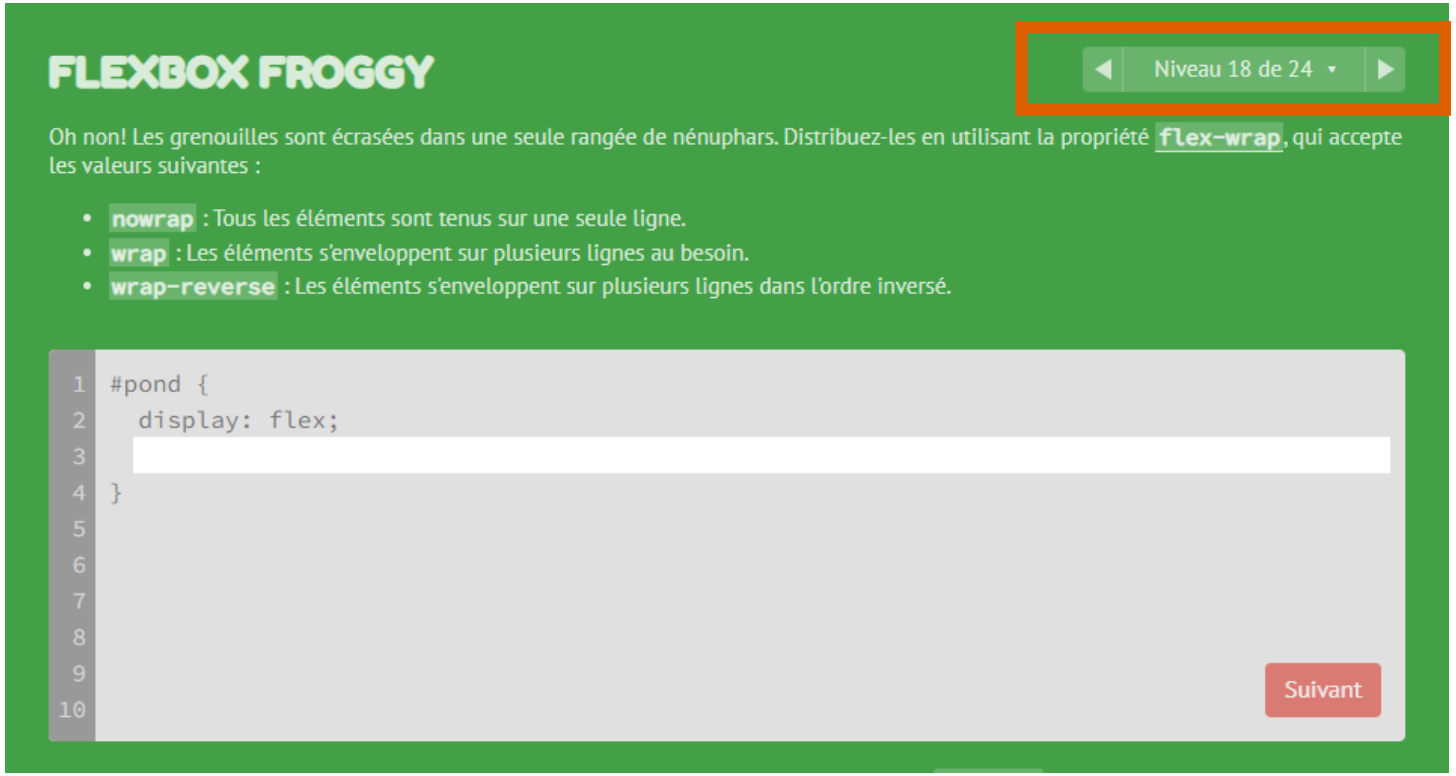
On explore un peu plus en détails ces propriétés avec les notions de GridBox.

FLEXBOX FROGGY – 2ième partie

Complétez les niveaux de FLEXBOX FROGGY.

Faites les niveaux 18 à 24 inclusivement.

Suivez le lien https://flexboxfroggy.com/#fr



Viewport

Viewport

Le **viewport** représente la taille affichable du navigateur.

En CSS, 4 unites sont disponibles pour mesurer les dimensions du viewport:

- **vw** : % de la largeur de l'écran affichable (Viewport Width)
- **vh** : % de la longueur de l'écran affichable (Viewport Height)
- **vmin** : % du plus petit entre : la largeur et la hauteur de l'écran affichable (Viewport minimum)
- **vmax** : % du plus grand entre : la largeur et la hauteur de l'écran affichable (Viewport maximum)

Est-ce qu'on afficher en format portrait ou paysage? vmin et vmax ne feront pas référence à la même dimension dépendant du cas.

Viewport - exemples

Chacune des 4 unites représente un pourcentage (%) de la mesure visée.

Exemples :

- 40vw = 40% de la largeur de la surface affichable
- 25vh = 25% de la hauteur de l'écran
- 0.5vw = un demi de 1 % de la largeur de la surface affichable
- 100vmin = 100% (tout) de la surface affichable de l'écran, de la plus petite dimension entre la hauteur et la largeur.

Viewport - utilisation

On utilise les unités de viewport aux mêmes endroits que les autres unités comme px, %, em etc.

Exemple :

```
.uneclasse{  width : 40vw;  height : 100vh;  border: 0.5vw;
                    padding: 5vw;  font-size: 3vw;  }
```

- Occupes toute la hauteur de l'affichage du navigateur et 40% de la largeur.
- Padding est 5% de la largeur affichable.
- Notez que même la taille de police peut ainsi être contrôlée.

Ces mesures s'adaptent au redimensionnement de la fenêtre du navigateur, un élément qui les utilise ajustera sa taille en fonction de celle du navigateur.

Viewport ou %?

Quelle est La différence entre l'unité (%) et vh/vw/vmin/vmax?

Le % est relatif au conteneur parent alors que les unités de viewport sont toujours calculées en fonction de la surface affichable du navigateur, indépendamment du conteneur parent.

Problème avec vw – le support de certains navigateurs (IE) ne tient pas compte du scroll bar.

On préfère utiliser % plutôt que vw pour un % de la largeur de l'écran (width : xx%).

Viewport ou %? (suite)

À quoi servent les unités viewport si % fonctionne?

Quelques exemples :

Pleine hauteur : 100% est calculé en fonction du contenu. Pas de contenu = 0 hauteur.

Ici, vh fonctionne comme un charme et occupera toute la hauteur quand-même

Formes (images) à proportion fixes en réactif : en %, un design fait d'élément à proportion fixe (un carré, un rectangle) perd son charme si on change la taille du navigateur.

Ici, width et height tous les 2 en vw fonctionne parfaitement

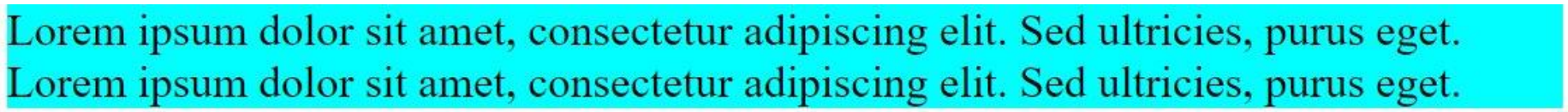
Du texte réactif : du texte qui doit occuper à peu près le même espace dans contenant donné est excellent en vw.

Viewport - démonstration

Démonstration D03E

Viewport - démonstration (démo 3e)

Ouvrez le fichier d03e-demo-viewport.html dans VS Code.



Viewport – exemple 1

On désire une page qui fait toute la hauteur du navigateur. On essaie avec height: 100%;.

Modifiez le code:

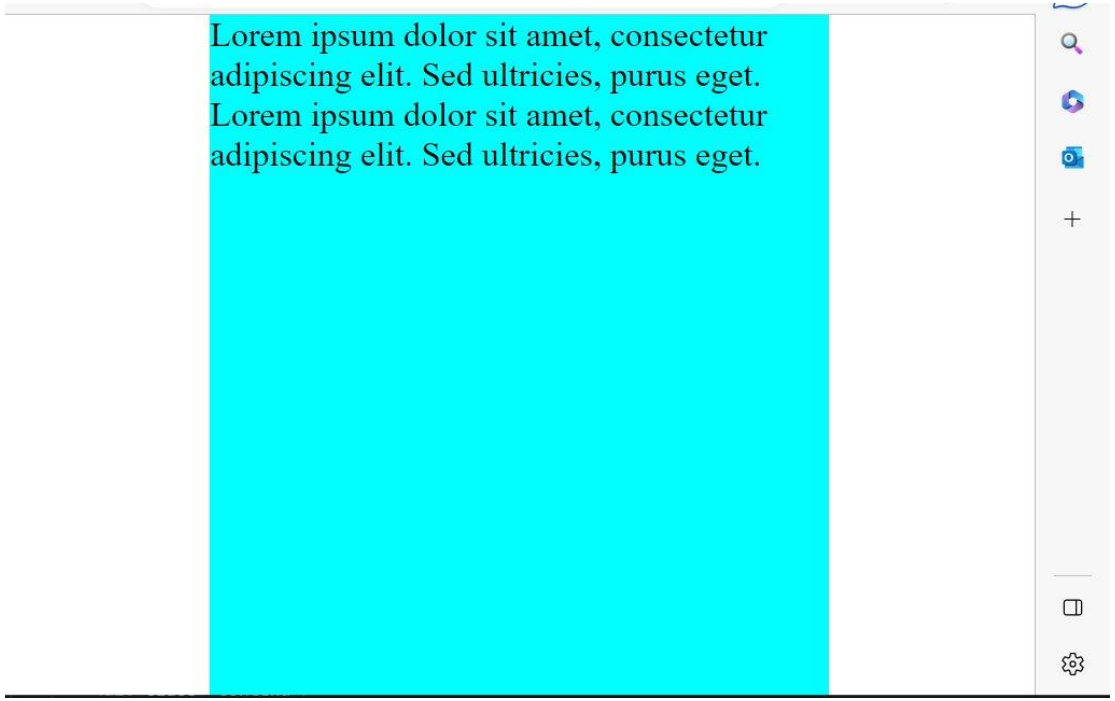
```
.contenu{
        width:60%;
        background-color: aqua;
        margin: 0px auto;
        font-size: 2em;
        height: 100%;      }
```

Pas de changement. En hauteur, le % est calculé à partir de la hauteur du CONTENU, pas du navigateur.

Viewport – exemple 1 (suite)

On essaie maintenant avec vh. Modifiez le code:

```
.contenu{
        width:60%;
        background-color: aqua;
        margin: 0px auto;
        font-size: 2em;
        height: 100vh;      }
```



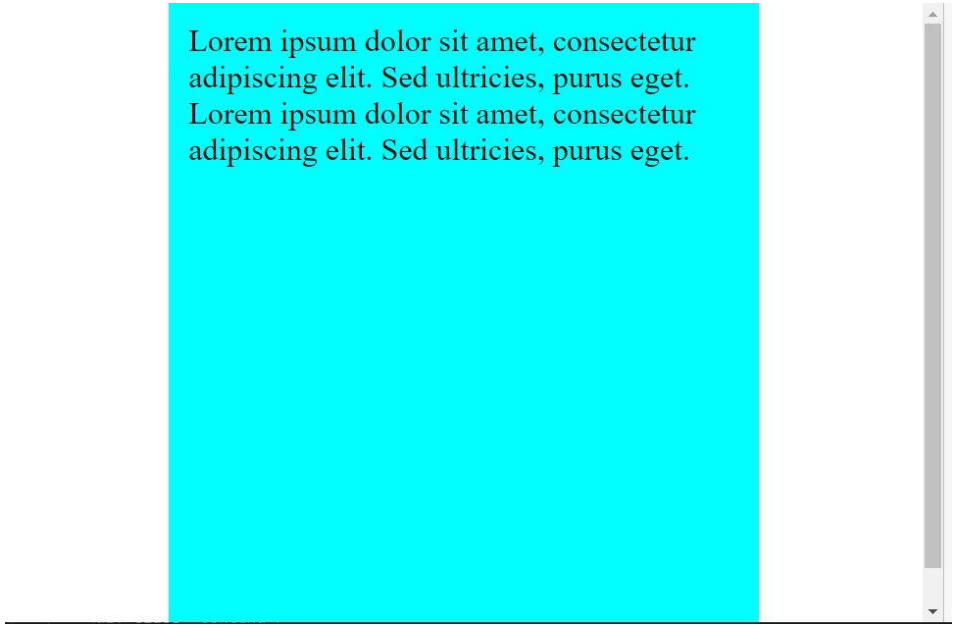
Super! Une page pleine hauteur.

Viewport – exemple 2

Un peu de mise en page, on ajoute du padding.

Modifiez le code:

```
.contenu{
        width:60%;
        background-color: aqua;
        margin: 0px auto;
        font-size: 2em;
        height: 100vh;
        padding: 20px;    }
```



Le padding ça va mais pourquoi la page dépasse vers le bas avec un scrollbar?

Le padding (et les bordures d'ailleurs) sont AJOUTÉS au contenu en width et en height.

La hauteur est maintenant « toute la hauteur du navigateur + 20px en haut + 20px en bas ».

Box-sizing

Pour régler ce problème typique, on doit parfois faire des calculs et soustraire les padding/margin/border de nos mesures.

Autre option typique: la propriété box-sizing: border-box; déclare que la taille (width, height) sont EXTÉRIEURES à la boite, les padding et border sont inclus dans les mesures.

Modifiez le code:

```
<style>
    html, body{
        margin:0px;
        padding:0px;
    }
    * {
        box-sizing: border-box;
    }
.contenu{
```

Et voilà! Plus de barre de défilement.

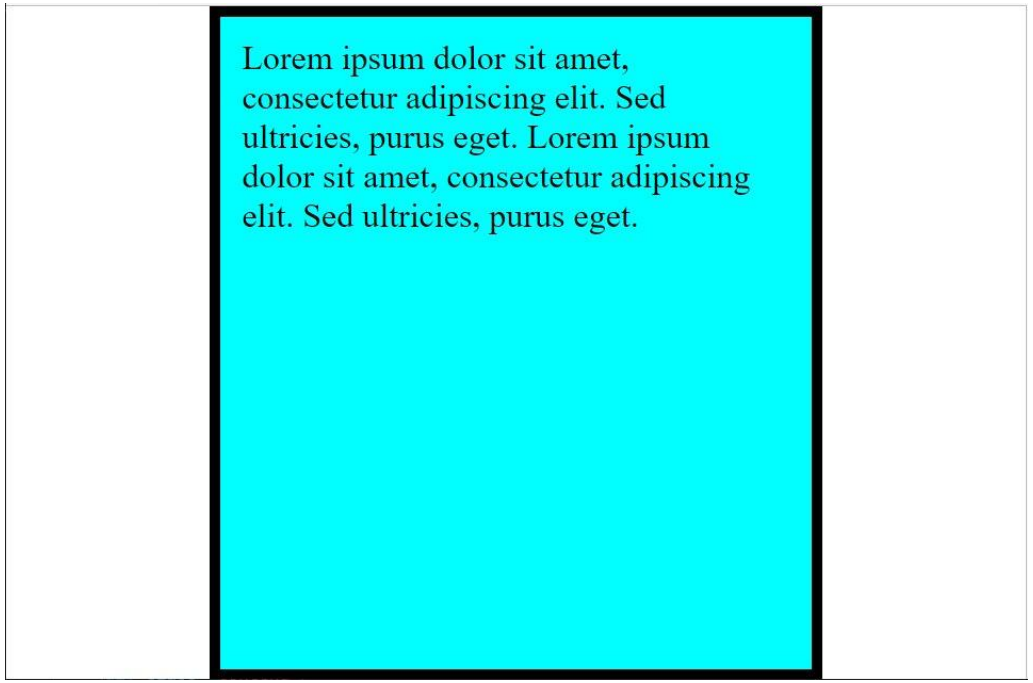
NOTE: on utilise ici le sélecteur *, qui signifie « pour tous les éléments ». L'utilisation de * n'est pas toujours recommandée, il faut utiliser au besoin seulement. Aurait-on pu obtenir le même résultat en appliquand **box-sizing** à un autre sélecteur?

Box-sizing – exemple 1

On ajoute une bordure.

Modifiez le code:

```
.contenu{
        width:60%;
        background-color: aqua;
        margin: 0px auto;
        font-size: 2em;
        height: 100vh;
        border: 10px solid black;
        padding: 20px;
}
```



Belle page plein écran.

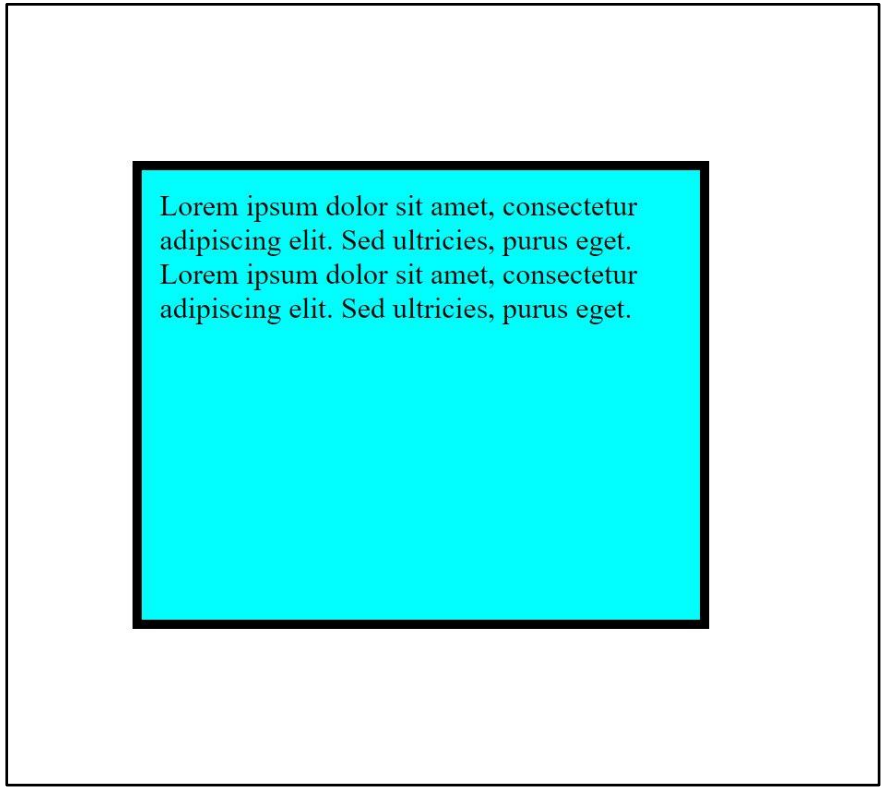
Notez que si le contenu prend plus d'espace que disponible dans l'écran, on aura à nouveau un scroll bar et la bordure du bas disparaitra vers le bas.

Viewport – page centrée verticalement

On peut aussi utiliser vh pour une page centrée.

Modifiez le code: :

```
.contenu{
        width:60%;
        background-color: aqua;
        margin: 20vh auto;
        font-size: 2em;
        height: 60vh;
        border: 10px solid black;
        padding: 20px;
}
```



Rapetissez le navigateur pour voir le texte dépasser de la boite.
On peut aussi rendre le texte réactif.

Viewport – gestion proportionnelle

Il arrive qu'on veuille respecter des proportions de padding, border, texte etc. lorsque la page change de taille.

Modifiez le code:

```
.contenu{
        width:60%;
        background-color: aqua;
        margin: 20vh auto;
        font-size: 4vw;
        height: 60vh;
        border: 1vw solid black;
        padding: 2vw;
}
```

Rapetissez le navigateur pour voir la bordure, le padding et le texte s'ajuster automatiquement. Le texte ne change plus de ligne lorsqu'on modifie la largeur.

Notez qu'on a choisi vw comme taille de référence et que dans le cas où le navigateur est large mais court, le texte peut encore sortir de la boite.

Viewport – gestion proportionnelle - 2

Pour faire des formes qui restent proportionnelles (comme un carré ou un rectangle qui garde ses proportions, modifiez le code:

```
.contenu{
        width:60vh;
        background-color: aqua;
        margin: 20vh auto;
        font-size: 3vh;
        height: 60vh;
        border: 1vh solid black;
        padding: 2vh;
}
```

On a choisi la même référence pour toutes les unités (vh).

Modifiez la taille du navigateur: on a un carré parfait et toutes les proportions sont conservées, même le texte, qui apparait exactement de la même façon indépendamment de la taille.

Note: il y a une limite de taille de texte qui est lisible, indépendamment de la taille du viewport. D'autres propriétés permettent de définir un minimum à la taille des éléments.

FLEX

Laboratoire 03B

Réalisez le laboratoire L03B_FlexBox-Reactif.

Requêtes média

Lorsque le réactif est progressif…

FlexBox offre une panoplie de solutions pour rendre une page web réactive.

Dans les solutions vues jusqu'à maintenant, le comportement n'est pas toujours idéal pour visionnement sur des appareils de tailles différentes (écran d'ordinateur, tablette, mobile etc.)

Le démo suivant montre un exemple de problème et comment il peut être résolu avec les requêtes média (media queries).

Média - démonstration

Démo D03F

Démo media 1 - présentation

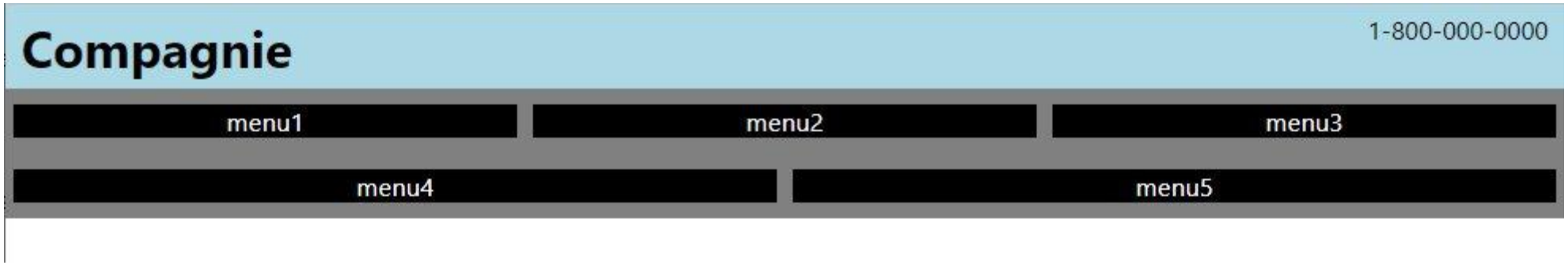
Ouvrez le fichier d03f-demo-media.html dans VS Code.



La page html montre un début de page avec une en-tête (header) qui inclut une section logo et un menu horizontal avec 5 éléments.

Démo media 1 - réactif

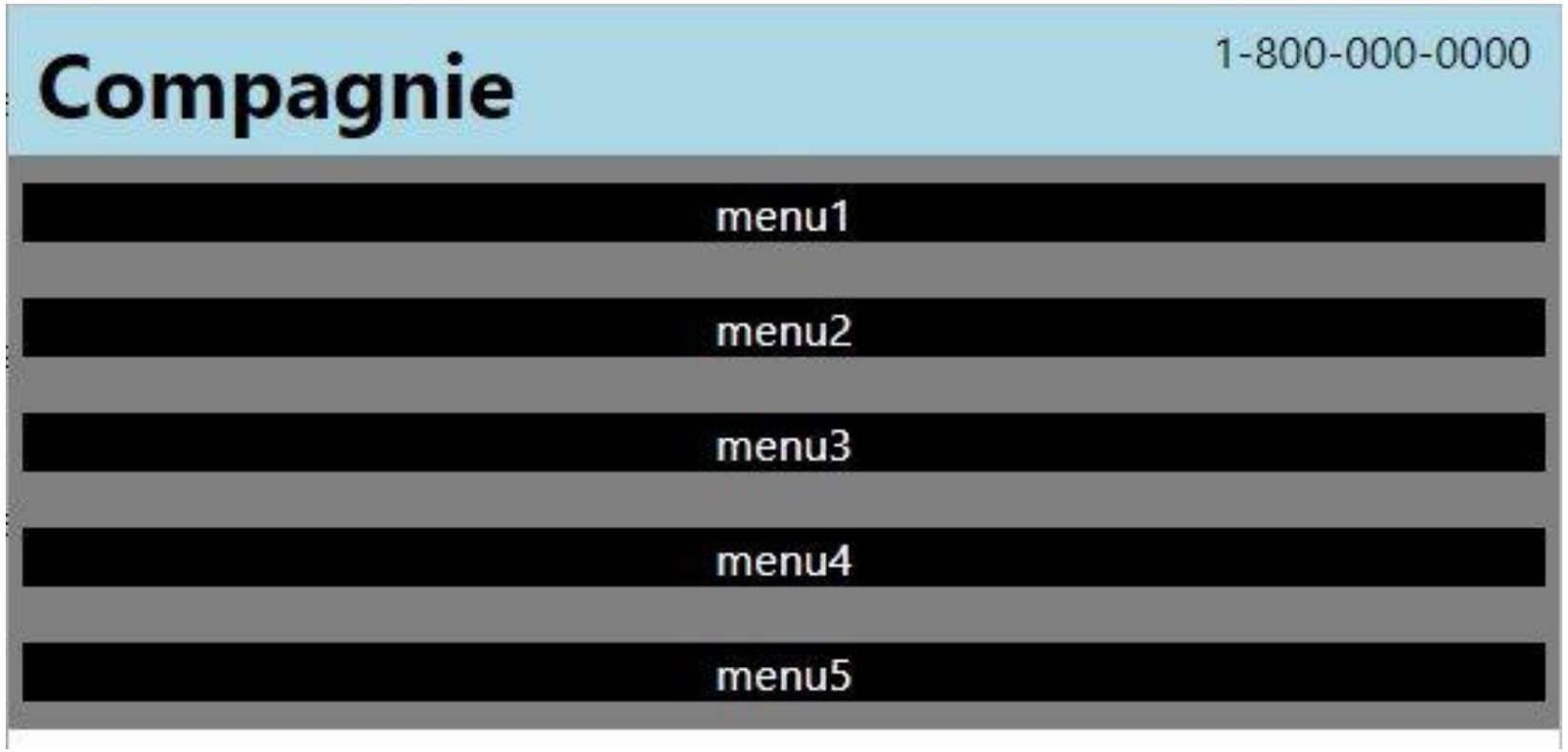
Réduisez graduellement la largeur du navigateur pour voir le menu s'adapter et faire un wrap, 1 bouton à la fois.



La page est bien réactive, mais si on travaille sur un écran plus petit, on voudrait plutôt que les boutons passent de tous horizontaux à tous verticaux, sans passer par un état intermédiaire.

Démo media 1 – réactif 2

Le résultat attendu serait plutôt:



Solution: requêtes média

Les requêtes médias (*medias queries*), introduites au standard de CSS 3, permettent d'interroger le navigateur à propos du matériel utilisé et de changer les styles en fonction de la réponse.

Entre autres, on peut modifier les styles en fonction de la taille de l'écran de l'appareil.

Requêtes média - syntaxe

Pour que les requêtes médias fonctionnent, il faut ajouter une balise meta dans l'en-tête html.

On utilise pour l'instant la syntaxe ci-dessous, on en explique les composants plus loin.

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

Cette balise indique que le viewport correspond à la taille de l'appareil et que le zoom de départ est 100%.

Requêtes média – exemple 1

On utilise les requêtes médias en vérifiant une condition de taille d'écran et en écrivant le css désiré pour cette taille d'écran.

Dans le document d03f-demo-media.html, ajoutez le code suivant, à la fin des autres styles:

```
    @media screen and (max-width: 600px) {
        .nav ul {
            flex-direction: column;
            align-items: stretch;
        }
        .nav ul li{
            margin: 5px;
        }
    }
</style>
```

Requête média – explication de syntaxe

Le code ajouté signifie que:

Pour les écrans (screen) sont la taille est inférieure à 600px (max-width: 600px), appliquer les styles suivants

- Afficher les éléments de la liste en colonne au lieu de rangée
- Chaque item occupe tout l'espace horizontal (stretch)
- Changer la marge autours de éléments pour 5px

Requêtes média – exemple 1 (suite)

Aussi, on ne désire plus de comportement wrap, c'est la requête média qui provoque le comportement réactif du menu. Changez le code:

```
.nav ul {
	display: flex;
	align-items: center;
	padding: 0px;
	margin: 0px;
	flex: 1 0 auto;
	flex-wrap: no-wrap;
}
```

```
.nav ul li{
	background-color: black;
	color: white;
	list-style-type: none;
	display:flex;
	justify-content:center;
	flex: 1 0 auto;
	margin: 10px 5px;
}
```

Requête média – exemple 1 - résultat

Affichez le résultat et changez la largeur du navigateur pour observer le comportement désiré.

Requête média – meta-viewport - explication

Syntaxe: balise meta, dans l'en-tête html

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

Certains appareils mobiles affichent le html sur une taille plus grande que l'écran puis rétrécissent le résultat pour qu'il entre dans l'écran.

Ce comportement interfère avec les requêtes média.

La balise meta -> viewport permet de spécifier l'utilisation d'un viewport adapté à la taille de l'écran dès le départ.

Requête média – meta-viewport – explication (suite)

`width=device-width`

Spécifie que width doit être celui de la largeur d'écran de l'appareil de navigation.

`initial-scale=1.0`

Spécifie que le zoom est à 100% lors du chargement initial de la page.

Requêtes média – @media

La requête média @media est placée APRÈS les autres styles.

Les styles ajoutés dans la requête remplaceront les styles précédents selon l'ordre normal de précédence.

Syntaxe: requête média dans le css:

```
@media not|only mediatype and (expressions) {
  CSS-Code;
}
```

Requêtes média – mediatype

Le mediatype s'applique sur les médias : screen, print, speech, mais nous n'utiliserons que screen dans ce cours.

Il existe plusieurs caractéristiques sur lesquelles les requêtes media peuvent s'appliquer, en voici seulement quelques-unes :

- HTML
  - attribut media pour les balises : a, area, link, source & style
- CSS
  - width, min-width, max-width, height, min-height, max-height
  - orientation
  - color
- On combine les conditions avec :
  - and, or & not

Requêtes média – exemple HTML

Voici un exemple HTML: charger une feuille de style différente dépendant de la taille de l'écran.

```
...
<head>
...

<link rel="stylesheet" href="style2.css media="screen and (max-width:1200px)/>
<link rel="stylesheet" href="style1.css media="screen and (max-width:600px)/>
...
</head>
...
```

Requêtes média – exemple CSS

Voici un exemple CSS:

```
.ma-classe { color : white; background-color : blue; }

/* lorsque l’écran est plus petit ou égal à 1200px */
@media screen and (max-width : 1200px) {
 .ma-classe { color : yellow; background-color : black; }
}



/* lorsque l’écran est plus petit ou égal à 800px */
@media screen and (max-width: 800px) { /* aussi appelé un "breakpoint" */
 .ma-classe { color : blue; background-color : white; }
}
```

Requête média – recommandations

Quelques recommandations à l'usage des requêtes médias :

- Comme pour tout le CSS, on tente de regrouper au maximum les propriétés communes, même à travers plusieurs requêtes média.
- Pour une même propriété, la dernière définition est celle retenue. Ainsi :
  - si on utilise une logique du style <= (max-width), il est nécessaire d'aller du plus grand au plus petit format
  - En contrepartie, si on utilise une logique du style >= (min-width), il est nécessaire d'aller du plus petit au plus grand format
  - De nos jours, on recommande de concevoir pour mobile en premier, puis de s'ajuster pour des écrans plus grands par la suite. Ceci implique l'utilisation de min-width au lieu de max-width (contrairement à notre premier démo média).
- On préfère utiliser autant que possible des unités relatives telles que : vh, vw, em, etc. à l'intérieur de la requête média.

Tester les Requêtes média avec Chrome

Essayez l'outil « Toggle device » dans les outils de développeur de Chrome.

Cet outil envoie des informations au navigateur concernant la taille de l'appareil pour les requêtes média, ce qui est différent de la taille du navigateur.



Requêtes média - exercices

Requêtes media – exercice 1

Dans le répertoire Laboratoires\L03C_Requête-Media récupérez le document ex03a-media-exercice-1.html. Ce document affiche des sections en format mobile. Modifiez le code pour que:

Sur des écrans plus larges que 600px le contenu soit à gauche (60% de la largeur) et la publicité soit à droite (40% de la largeur).

N'oubliez pas la balise requise dans l'en-tête html et ajoutez le code de la requête media à la fin des styles.

Voir l'exemple à la prochaine page.

Requêtes media – objectif

Petit



Grand



Requêtes media – exercice 2

Dans le répertoire Laboratoires\L03C_Requête-Media récupérez le document ex03b-media-exercice-2.html. Ce document affiche des sections en format mobile.

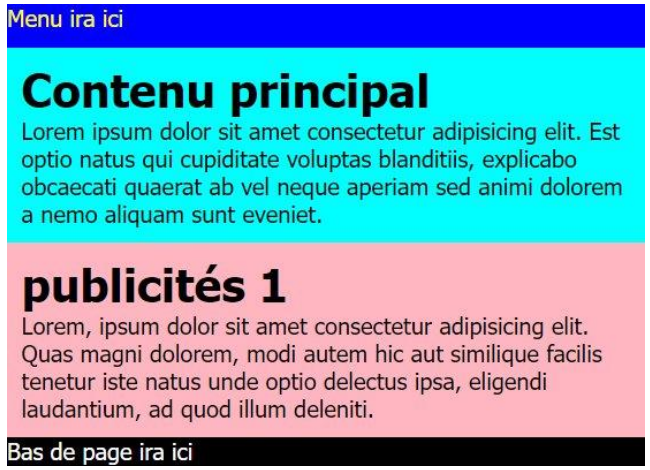
Notez que sur un grand écran on aura 3 sections au centre alors que sur un appareil mobile, la troisième section (pub2) n'apparaitra pas (contenu jugé pas assez important pour occuper du précieux espace sur petit écran.

De plus, on désire que sur grand écran, la section pub 1 soit à gauche et la section pub 2 à droite, alors que la section pub1 apparait sous le contenu en format mobile.

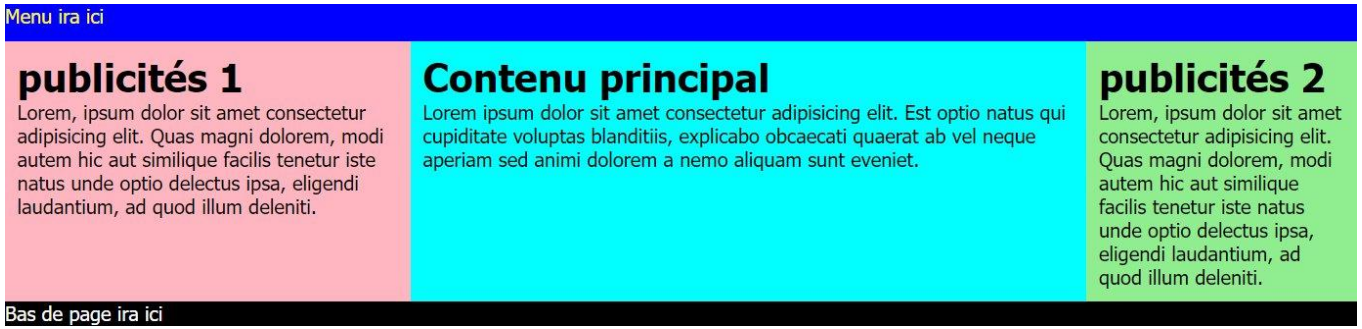
Modifiez le code pour obtenir les résultats montrés à la page suivante (proportions 30%/50%/20% pour la largeur de pub1/contenu/pub2 sur grand écran -> plus grand que 640px)

Requêtes media – objectif

Petit



Grand



FLEX

Laboratoire 03C

Réalisez le laboratoire L03C-Requêtes média.

Travail Pratique 3 (TP3)