CSS

Media et varia

Utilisation de plusieurs fichiers CSS

- Il est possible d'utiliser plusieurs fichiers CSS.
- La balise HTML **`link`** peut être utilisée à plusieurs reprises pour importer plusieurs fichiers externes.
- Un fichier CSS peut en importer un autre grâce à **`@import ...`**
- N'oubliez pas que la dernière définition d'une même propriété est celle utilisée.

Attributs : HTML et CSS

- CSS :
  - Il existe une gamme de sélecteurs utilisant spécifiquement les attributs d'éléments HTML. Ils utilisent les "[...]" avec plusieurs variantes pour effectuer des sélections flexibles selon l'attribut *attribut* :

balise[*attribut*]	sélectionne les balises identifiées ayant défini l'attribut indiqué
balise[*attribut*="*valeur*"]	balise dont la valeur de l'attribut est égale à *valeur*
balise[*attribut*~="*valeur*"]	contient *valeur* en mot entier, avec ou sans espaces
balise[*attribut*|="*valeur*"]	valeur d'*attribut* est *valeur* ou bien *valeur* suivie d'un -
balise[*attribut*^="*valeur*"]	valeur d'*attribut* débutant par *valeur*
balise[*attribut*$="*valeur*"]	valeur d'*attribut* se terminant par *valeur*
balise[*attribut**="*valeur*"]	valeur d'*attribut* contient *valeur*

    - Le nom de la balise n'est pas nécessaire.
      - Le comportement s'appliquera à toutes les balises dont les attributs correspondent.

Variable

- Il est possible de déclarer et utiliser des variables en CSS.
  - On déclare les variables en CSS directement dans un bloc déclaratif en préfixant la variable par deux tirets :

```
:root {
     --ma-couleur : red;
}
```

  - Comme dans un langage de programmation conventionnel, la variable possède une portée. Cette dernière suit le DOM (hiérarchie des styles). Par exemple, une variable déclarée dans le sélecteur :root (le plus haut sélecteur, avant même html) sera considérée globale alors qu'une variable déclarée dans une autre balise pourra être locale avec accès seulement par les enfants.

Variable

- La fonction **`var(...)`** donne accès aux valeurs de variables déclarées :
  `p { color : var(--ma-couleur); }`
- Une variable peut être redéfinie dans la cascade des styles (incluant les requêtes média)
- Il existe plusieurs limitations liées aux variables. Voici les plus importantes :
  - Les variables sont utilisées pour déterminer les valeurs de propriétés et non pas pour définir sur laquelle des propriétés appliquer une valeur :
    `:root { --prop : padding-right }`
    `p { var(--prop) : 0px; } /* ERREUR */`
  - Il est impossible de créer des fonctions personnalisées ou d'effectuer directement des calculs à partir des variables.
    - Toutefois, la fonction **`calc(...)`** permet de faire des calculs.

Calcul

- Il est possible de réaliser des calculs avec la fonction **`calc(...)`**.
  - Les calculs s'appliquent sur les nombres seulement (pas d'opération sur les chaînes de caractères).
  - Les opérations supportées sont : **`+ - * /`**
  - Les espaces sont obligatoires autours des opérateurs **+** et **–**. Même s'ils ne sont pas requis pour les ***** et **/**, il est recommandé de les mettre par soucis d'uniformité.
  - L'usage d'une variable dans un calcul nécessite tout de même la fonction **`var(...)`**.
  - Il est possible de mélanger les unités compatibles dans les opérateurs **+** et **–** alors que les opérateurs ***** et **/** requièrent au moins un scalaire sans unité.

Calcul

- Un exemple :

```
:root {
    --taille : 1.5em;
    --bordure : 5px;
    --largeur : calc(50vw - 10% + 2 * var(--bordure));
    --hauteur : var( --largeur);
}

p {
    --var-locale-pour-couleur : calc(8 * 30);
    width : var(--largeur);
    color : rgb(var(--var-locale-pour-couleur), 64, 255);
    padding : calc(20vw - var(--bordure));
}
```

Requêtes média

- Les requêtes média (media query) permettent de cibler certaines caractéristiques du navigateur et d'appliquer des propriétés CSS spécifiques et adaptées selon le contexte.
- Elles s'appliquent sur les medias : screen, print, speech, mais nous n'utiliserons que screen dans ce cours.
- Il existe plusieurs caractéristiques sur lesquelles elles 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

- Voici un exemple HTML:

```
...
<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

- 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; }
}
```

Requêtes média

- 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 contre partie, si on utilise une logique du style >= (min-width), il est nécessaire d'aller du plus petit au plus grand format
  - On préfère utiliser autant que possible des unités relatives telles que : vh, vw, em, etc. à l'intérieur de la requête média.

Requêtes média

- Lorsqu'on travaille avec des émulateurs et/ou appareils de tailles différentes, il est nécessaire de s'ajuster.
  - Dans le <head> du document, on doit ajouter:

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

  - width=device-width
    - On spécifie que *width* doit être celui de la largeur d'écran de l'appareil de navigation.
  - initial-scale=1.0
    - On veut que le zoom soit à 100% lors du chargement initial de la page.

Flexbox

- Standard CSS de disposition des éléments.
  - Il permet d'avoir un design adaptatif à l'écran plus facilement qu’avec les méthodes traditionnelles.

- Ce n’est pas une seule propriété, mais plutôt un module de propriétés communicantes.

Flexbox

- Étant donné les caractéristiques innées des éléments HTML, le fureteur essaie de les organiser tel qu'il peut:
  - Verticalement pour les éléments de type block
  - Horizontalement pour les éléments de type inline

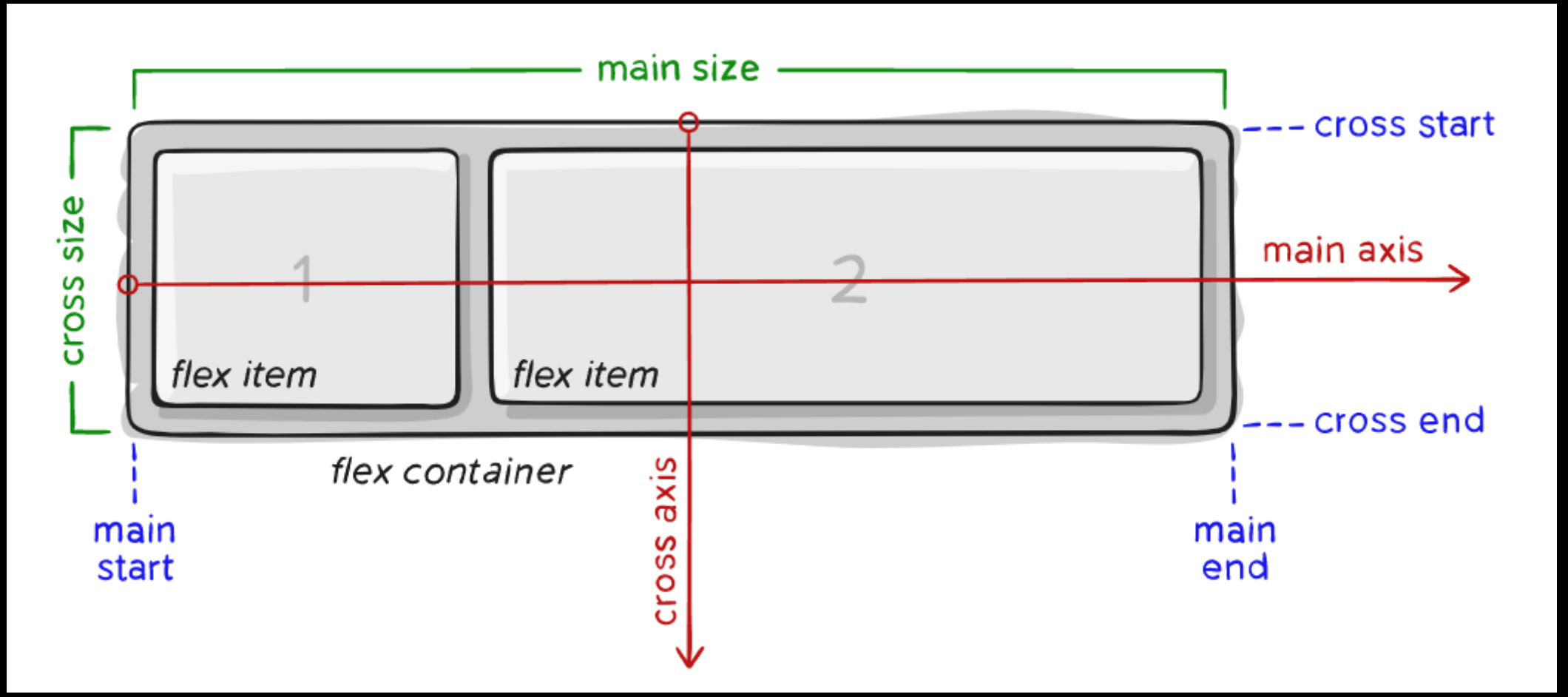
- Flexbox permet une plus grande *flexibilité* à cet égard.

- Tout commence en définissant une règle CSS bien particulière à l'élément contenant:

```
.contenant {display: flex;}
```

Flexbox

- En voici la logique visuelle



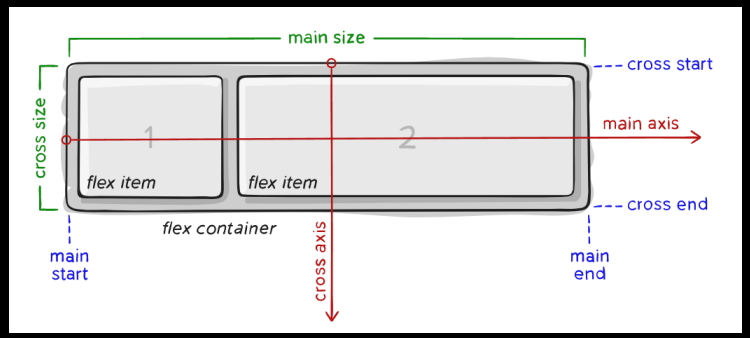
Flexbox

- *main axis*, c'est l'axe principal selon lequel seront disposés les éléments.
  - Cet axe peut être vertical ou horizontal, de gauche à droite ou bien de droite à gauche.
    - Ça dépend des valeurs qu'on donne aux propriétés du flexbox.

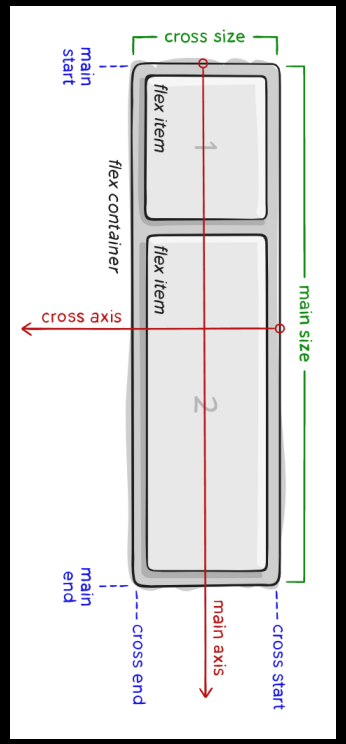
- *cross axis* est perpendiculaire à *main axis.*
  - La direction de disposition des éléments dépend également des valeurs qu'on donne au propriétés du flexbox.

Flexbox

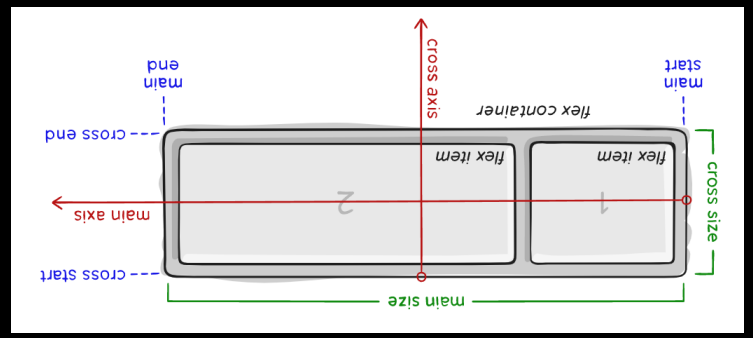
- Quatre sens
  - Propriété
    - flex-direction



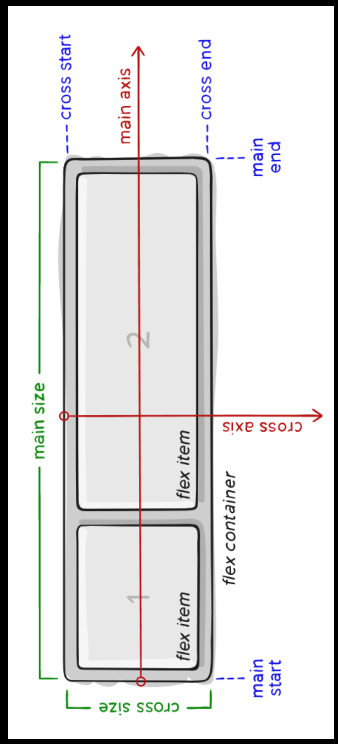
row



column



row-reverse



column-reverse

Flexbox

- Exemple 1

```
.contenant {
  display: flex;
  height: 300px; /* peu importe */
}

.top-nav {
  width: 100px;  /* peu importe */
  height: 100px; /* peu importe */
  margin: auto;  /* Magie magie */
}
```

Flexbox

- Exemple 2

```
.contenant {
  display: flex;

  /* direction et sens du main axis */
  flex-direction: row;

  /* lorsque le contenu dépasse la largeur du contenant */
  flex-wrap: wrap;

  /* comment on distribute l’espace restant */
  justify-content: space-around;
}
```

Flexbox

- Exemple 3 (2++)

```
.contenant {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: flex-end; /* Gros, items alignés sur la fin du main axis */
}

@media screen and (max-width: 800px) {
  .contenant {
    justify-content: space-around; /* Moyen, items centrés, distribution égale de l’espace */
  }
}

@media screen and (max-width: 400px) {
  .contenant {
    flex-direction: column; /* Petit, on dispose verticalement, en colonnes */
    /* on GARDE justify-content: space-around; */
  }
}
```

CSS

- Ceci n’est qu’un survol des possibilités que nous offre le CSS.
  - À vous de jouer.

Bibliographie

- https://www.w3schools.com/
- https://css-tricks.com/