

CSS 3, rappels



Le modèle de boîte



Le modèle de boîte

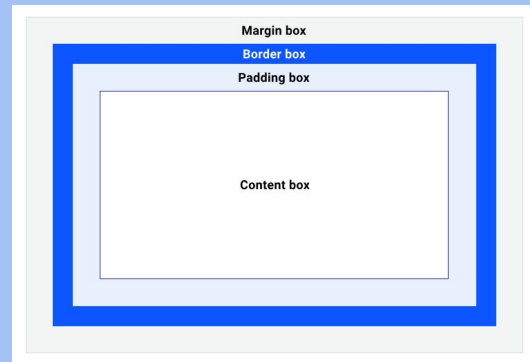
Les boîtes sont constituées de zones de distinctes qui effectuent toutes un travail spécifique.

Vous commencez par la zone de contenu, qui correspond à la zone dans laquelle se trouve le contenu. Comme vous l'avez appris précédemment, ce contenu peut contrôler la taille de son élément parent. Il s'agit donc généralement de la zone la plus variable.

La zone de marge intérieure entoure la zone de contenu et correspond à l'espace créé par la propriété **padding**. Étant donné que la marge intérieure se trouve à l'intérieur de la zone, son arrière-plan sera visible dans l'espace qu'elle crée.

Si des règles de dépassement sont définies pour notre zone, telles que **overflow: auto** ou **overflow: scroll**, les barres de défilement occupent également cet espace.

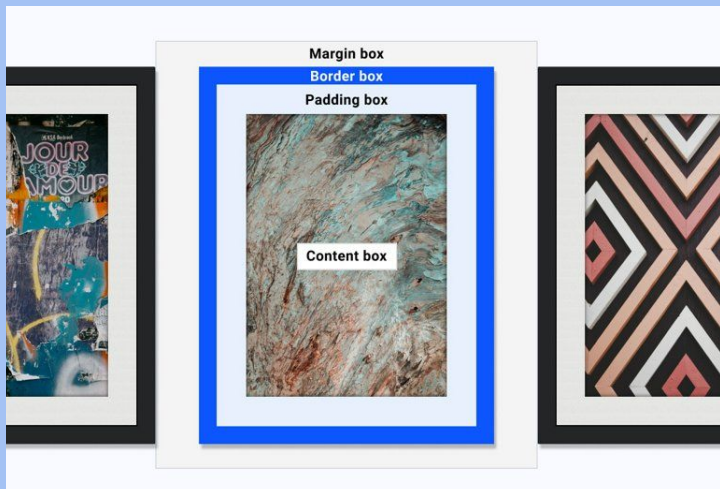
La zone de bordure entoure la zone de remplissage, et son espace est occupé par la valeur **border**. La zone de bordure correspond aux limites de votre zone, tandis que le bord de bordure représente la limite de ce que vous pouvez voir visuellement. La propriété **border** permet d'encadrer visuellement un élément.



Le modèle de boîte

La dernière zone, la zone de marge, correspond à l'espace autour de la zone, défini par la règle **margin** de votre zone. Les propriétés telles que **outline** et **box-shadow** occupent également cet espace, car elles sont peintes sur le dessus. Elles n'affectent donc pas la taille de notre boîte. La boîte peut avoir un **outline-width** de 200px, et tous les éléments à l'intérieur, y compris la zone de bordure, auront exactement la même taille.

Une analogie



Dans ce schéma, vous avez trois cadres photo, montés à un mur, l'un à côté de l'autre. Le diagramme comporte des étiquettes qui associent des éléments du cadre au modèle de boîte.

Pour décomposer cette analogie:

- La zone de contenu correspond à l'illustration.
- La boîte de marge intérieure est le panneau de montage blanc situé entre le cadre et l'œuvre.
- La zone de bordure est le cadre, qui sert de bordure littérale à l'œuvre.
- La zone de marge correspond à l'espace entre chaque image.
- L'ombre occupe le même espace que la zone de marge.

Le modèle de boîte

Contrôler le modèle de boîte

Pour comprendre comment contrôler le modèle de boîte, vous devez d'abord comprendre ce qui se passe dans votre navigateur.

Chaque navigateur applique une feuille de style user-agent aux documents HTML. Le CSS utilisé varie d'un navigateur à l'autre, mais ils offrent des valeurs par défaut raisonnables pour faciliter la lecture du contenu. Elles définissent l'apparence et le comportement des éléments si aucun CSS n'est défini. C'est également dans les styles de user-agent que le `display` par défaut d'une zone est défini. Par exemple, dans un flux normal, la valeur `display` par défaut d'un élément `<div>` est `block`, la valeur `display` par défaut d'un élément `` est `list-item` et la valeur `display` par défaut de `` est `inline`.

Un élément `inline` possède une marge de bloc, mais les autres éléments ne la respectent pas. Utilisez `inline-block` pour que ces éléments respectent la marge de bloc, tout en conservant la plupart des comportements qu'ils avaient en tant qu'éléments `inline`. Par défaut, un élément `block` remplit l'espace intégré disponible, tandis que les éléments `inline` et `inline-block` n'ont que la taille de leur contenu.

Le modèle de boîte

En plus de comprendre comment les styles du user-agent affectent chaque zone, vous devez également comprendre **box-sizing**, qui indique à notre boîte comment calculer sa taille.

Par défaut, tous les éléments ont le style user-agent suivant: **box-sizing: content-box**;

Lorsque vous définissez **content-box** comme valeur de **box-sizing**, cela signifie que lorsque vous définissez des dimensions, telles que **width** et **height**, elles sont appliquées à la zone de contenu.

Si vous définissez ensuite padding et border, ces valeurs seront ajoutées à la taille de la zone de contenu.

Les sélecteurs



Les sélecteurs

Comment faire si vous souhaitez afficher du texte en plus grand et en rouge uniquement s'il s'agit du premier paragraphe d'un article ?

```
<article>  
  <p>Je veux être rouge et plus grand que l'autre texte.</p>  
  <p>Je veux être de taille normal et de la couleur par défaut.</p>  
</article>
```

Utilisez un sélecteur CSS pour trouver cet élément spécifique et appliquer une règle CSS, comme celle-ci.

```
article p:first-of-type {  
  color: red;  
  font-size: 1.5rem;  
}
```

Afin de vous aider à résoudre de telles situations, le CSS vous propose de nombreuses options pour sélectionner des éléments et leur appliquer des règles, allant de très simples à très complexes.

Composants d'une règle CSS

Pour comprendre le fonctionnement des sélecteurs et leur rôle dans CSS, il est important de connaître les parties d'une règle CSS. Une règle CSS est un bloc de code contenant un ou plusieurs sélecteurs et une ou plusieurs déclarations.

Les sélecteurs



Dans cette règle CSS, le sélecteur est **.ma-regle-css**. Il recherche tous les éléments de classe **.ma-regle-css** sur la page. Il y a trois déclarations entre les accolades. Une déclaration est une paire propriété/valeur qui applique des styles aux éléments mis en correspondance par les sélecteurs.

Une règle CSS peut comporter autant de déclarations et de sélecteurs que vous le souhaitez.

Les sélecteurs

Sélecteurs simples

Le groupe de sélecteurs le plus simple cible les éléments HTML, ainsi que les classes, les ID et d'autres attributs qui peuvent être ajoutés à une balise HTML.

Sélecteur universel

Un sélecteur universel, également appelé caractère générique, correspond à n'importe quel élément. Avec cette règle, chaque élément HTML de la page affichera un texte de "rouge".

```
*{  
  color: red;  
}
```

Sélecteur de type

Un sélecteur de type correspond directement à un élément HTML. Avec cette règle, chaque élément <section> comporte 2em de padding sur tous les côtés.

```
section{  
  padding: 2em;  
}
```

Sélecteur de classe

Un élément HTML peut comporter un ou plusieurs éléments définis dans leur attribut class. Le sélecteur de classe correspond à tous les éléments auxquels cette classe est appliquée. Tous les éléments auxquels la classe est appliquée sont colorés en rouge:

```
<div class="red"></div>  
<p class="red"></p>  
<button class="red"></button>
```

```
.my-class{  
  color: red;  
}
```

Les sélecteurs

Sélecteurs d'ID

Le seul élément de la page ayant un attribut id doit être associé à cet ID.

Vous sélectionnez les éléments avec un sélecteur d'ID comme suit.

Ce CSS applique une bordure bleue à l'élément HTML dont l'id est rad:

```
#rad{  
  border: 1px solid blue;  
}
```



Remarque : Si le navigateur rencontre plusieurs instances d'un élément id, il applique tout de même les règles CSS qui correspondent à son sélecteur. Cependant, tout élément qui possède un attribut id est censé avoir une valeur unique. Par conséquent, à moins que vous n'écriviez du code CSS très spécifique pour un seul élément, évitez d'appliquer des styles avec le sélecteur id, car vous ne pourrez pas les réutiliser ailleurs.

Sélecteur d'attribut

Vous pouvez rechercher les éléments ayant un attribut HTML donné ou une valeur donnée pour un attribut HTML à l'aide du sélecteur d'attribut.

Pour demander au CSS de rechercher les attributs, entourez le sélecteur avec des crochets [].

```
[data-type="primary"]{  
  color: red;  
}
```

```
<div data-type="primary"></div>
```

Les sélecteurs

Au lieu de rechercher une valeur spécifique de data-type, vous pouvez également rechercher des éléments avec l'attribut présent, quelle que soit sa valeur.

```
<div data-type="primary"></div>
<div data-type="secondary"></div>
```

Ces deux éléments <div> s'affichent en rouge.

Vous pouvez utiliser des sélecteurs d'attributs sensibles à la casse en ajoutant un opérateur s à votre sélecteur d'attributs.

```
[data-type]{
  color: red;
}
```

```
[data-type="primary" s]{
  color: red;
}
```

Cela signifie que si un élément HTML avait la valeur ***data-type Primary*** au lieu de **primary**, il ne recevrait pas de texte en rouge. Vous pouvez faire le contraire (insensibilité à la casse) en utilisant un opérateur i.

En plus des opérateurs "case", vous avez accès à des opérateurs qui correspondent à des parties de chaînes dans les valeurs d'attribut.

Les sélecteurs

Sélecteurs de regroupement

Un sélecteur ne doit pas nécessairement correspondre à un seul élément. Vous pouvez regrouper plusieurs sélecteurs en les séparant par des virgules:

```
/* href contenant "exemple.com" */  
[href*='exemple.com'] {  
    color: red;  
}  
  
/* href qui commence par https */  
[href^='https'] {  
    color: green;  
}  
  
/* href qui finit par .com */  
[href$='.com'] {  
    color: blue;  
}
```

Les sélecteurs

Cet exemple étend le changement de couleur aux éléments `` et ``. Il est également étendu à une classe nommée `.my-class` et à un élément doté d'un attribut `lang`.

```
strong, em, .my-class, [lang]{  
  color: red;  
}
```

Pseudo-classes et pseudo-éléments

CSS fournit des types de sélecteurs utiles qui se concentrent sur un état de plate-forme spécifique, comme le survol d'un élément, des structures à l'intérieur ou des parties d'un élément.

Pseudo-classes

Les éléments HTML se retrouvent dans différents états, soit parce qu'ils interagissent, soit parce que l'un de leurs éléments enfants est dans un certain état.

Par exemple, un utilisateur peut pointer sur un élément HTML avec le pointeur de la souris ou un élément enfant. Dans ce cas, utilisez la pseudo-classe `:hover`.

Les sélecteurs

Pseudo-élément

Les pseudo-éléments diffèrent des pseudo-classes, car au lieu de répondre à l'état de la plate-forme, ils agissent comme s'ils inséraient un nouvel élément avec CSS.

Les pseudo-éléments sont également différents des pseudo-classes d'un point de vue syntaxique, car au lieu d'un seul deux-points “:”, nous utilisons un double deux-points “::”.

Remarque : Le double deux-points “::” permet de distinguer un pseudo-élément d'une pseudo-classe. Cependant, comme cette distinction



*n'était pas présente dans les anciennes versions des spécifications CSS, les navigateurs acceptent un seul deux-points pour les pseudo-éléments d'origine, tels que **:before** et **:after**, pour assurer la rétrocompatibilité avec les anciens navigateurs, comme IE11.*

```
/* Our link is hovered */
a:hover {
  outline: 1px dotted green;
}
/* Sets all even paragraphs
to have a different background */
p:nth-child(even) {
  background: white;
}
```

```
.my-element::before {
  content: 'Prefix - ';
}
```

Les sélecteurs

Comme dans la démonstration ci-dessus, où vous avez préfixé le libellé d'un lien avec le type de fichier dont il s'agit, vous pouvez utiliser le pseudo-élément **::before** pour insérer du contenu au début d'un élément ou le pseudo-élément **::after** pour insérer du contenu à la fin d'un élément.

Les pseudo-éléments ne se limitent pas à l'insertion de contenu. Vous pouvez également les utiliser pour cibler des parties spécifiques d'un élément.

Par exemple, supposons que vous ayez une liste vous allez utiliser **::marker** pour appliquer un style à chaque puce (ou numéro) de la liste.

```
li::marker{  
  color: red;  
}
```

Vous pouvez également utiliser **::selection** pour appliquer un style au contenu mis en surbrillance par un utilisateur.

```
::selection {  
  background: black;  
  color: white;  
}
```


Les sélecteurs

Sélecteurs complexes

Vous avez déjà vu un large éventail de sélecteurs, mais vous aurez parfois besoin d'un contrôle plus précis avec votre CSS. C'est là que les sélecteurs complexes interviennent.

Notez que même si les sélecteurs suivants nous offrent plus de puissance, nous ne pouvons en cascade que vers le bas, en sélectionnant des éléments enfants. Nous ne pouvons pas cibler le haut et sélectionner un élément parent. Nous aborderons la notion de cascade et son fonctionnement dans une leçon ultérieure.

Combinateurs

Un combinateur est ce qui se trouve entre deux sélecteurs. Par exemple, si le sélecteur était **p > strong**, le combinateur est le caractère **>**. Les sélecteurs qui utilisent ces combinaisons vous permettent de sélectionner des éléments en fonction de leur position dans le document.

Combinateur descendant

Pour comprendre les combinateurs de descendants, vous devez d'abord comprendre les éléments parents et enfants.

Les sélecteurs

```
<p>A paragraph of text with some <strong>bold text for emphasis</strong>.</p>
```

L'élément parent est l'élément **<p>** qui contient du texte.

Cet élément **<p>** contient un élément ****, qui met son contenu en gras.

*Comme il se trouve dans **<p>**, il s'agit d'un élément enfant.*

```
p strong{  
  color: blue;  
}
```

Un combinateur descendant nous permet de cibler un élément enfant.

Cette commande utilise un **espace** pour indiquer au navigateur de rechercher les éléments enfants:

Le code suivant sélectionne tous les éléments **** qui sont uniquement des éléments enfants d'éléments **<p>**. Ils sont ainsi bleus de manière récursive.

Les sélecteurs

Combinateur “suivant”

Vous pouvez rechercher un élément qui suit immédiatement un autre élément en utilisant un caractère `+` dans votre sélecteur.

Pour ajouter de l'espace entre des éléments “empilés”, utilisez le combinateur “suivant”, `*`, pour ajouter de l'espace uniquement si un élément suit un élément enfant de la classe `.top`. Vous pouvez ajouter une marge à tous les éléments enfants de `.top` à l'aide du sélecteur suivant:

```
.top * {  
  margin-top: 1em;  
}
```

Le problème est que, comme vous sélectionnez chaque élément enfant de `.top`, cette règle peut créer un espace supplémentaire inutile. Le combinateur “suivant”, associé à un sélecteur universel, vous permet non seulement de contrôler les éléments qui obtiennent de l'espace, mais également de l'appliquer à n'importe quel élément.

Parcontre vous bénéficiez ainsi d'une certaine flexibilité à long terme, quels que soient les éléments HTML affichés dans `.top`.

Les sélecteurs

Combinateur "ultérieur"

Un combinateur ultérieur est très semblable à un sélecteur frère suivant. Cependant, au lieu d'un caractère **+**, utilisez un caractère **~**. La différence est qu'un élément doit simplement suivre un autre élément ayant le même parent, plutôt que d'être l'élément suivant avec le même parent.

Ce combinateur ultérieur offre un peu moins de rigidité, ce qui est utile dans des contextes tels que l'exemple ci-dessus, où nous changeons la couleur d'un contacteur personnalisé lorsque la case à cocher associée présente l'état **:checked**.

Combinateur enfant

Un combinateur enfant (également appelé descendant direct) vous permet de mieux contrôler la récursion associée aux sélecteurs de combinatoire. En utilisant le caractère **>**, vous limitez le sélecteur de combinatoire afin qu'il s'applique uniquement aux enfants directs.

Prenons l'exemple de sélecteur de frères sœurs précédent. L'espace est ajouté à chaque élément suivant, mais si l'un de ces éléments a pareillement des éléments frères suivants en tant qu'enfants, cela peut entraîner un espacement supplémentaire indésirable. Pour résoudre ce problème, modifiez le sélecteur de frères et sœurs suivant pour intégrer un combinateur enfant

Les sélecteurs

Sélecteurs composés

Vous pouvez combiner des sélecteurs pour améliorer la spécificité et la lisibilité.

Par exemple, pour cibler les éléments `<a>` qui ont également une classe `.my-class`, écrivez le code suivant:

Cela n'applique pas de couleur rouge à tous les liens. La couleur rouge n'est appliquée qu'à `.my-class` quand il se trouve sur un élément `<a>`.

```
a.my-class {  
  color: red;  
}
```

La cascade



La cascade

CSS est l'abréviation de "**Cascading Style Sheets**".

La cascade est un algorithme permettant de résoudre les conflits dans lesquels plusieurs règles CSS s'appliquent à un élément HTML.

C'est la raison pour laquelle le texte du bouton stylisé avec le CSS suivant sera bleu.

```
button {  
  color: red;  
}  
button {  
  color: blue;  
}
```

Comprendre l'algorithme de cascade vous permet de comprendre comment le navigateur résout les conflits de ce type. L'algorithme de la cascade est divisé en quatre étapes distinctes.

1. Position et ordre d'apparence: ordre d'affichage de vos règles CSS.
2. Spécificité: algorithme qui détermine le sélecteur CSS qui présente la correspondance la plus forte.
3. Origine: indique l'ordre d'affichage du CSS et sa provenance. Il peut s'agir d'un style de navigateur, du code CSS d'une extension de navigateur ou du CSS que vous avez créé.
4. Importance: certaines règles CSS sont pondérées plus que d'autres, en particulier avec le type de règle important.

La cascade

Position et ordre d'apparition

L'ordre dans lequel vos règles CSS apparaissent et la façon dont elles apparaissent sont prises en compte par la cascade lors du calcul de la résolution des conflits.

La démonstration au début de cette leçon est l'exemple le plus simple de position. Deux règles ont des sélecteurs de spécificité identique. La dernière à déclarer l'emporte.

Les styles peuvent provenir de diverses sources sur une page HTML, telles qu'une balise **<link>**, une balise **<style>** intégrée et le code CSS intégré tel que défini dans l'attribut `style` d'un élément.

Si vous avez une balise **<link>** qui inclut du code CSS en haut de votre page HTML, une autre **<link>** incluant du code CSS en bas de votre page: la balise **<link>** inférieure est la plus spécifique. Il en va de même avec les éléments **<style>** intégrés. Plus ils sont situés plus bas sur la page, plus ils sont précis.

Cet ordre s'applique également aux éléments **<style>** intégrés.

S'ils sont déclarés avant un **<link>**, le CSS de la feuille de style associée sera le plus spécifique.

La cascade

Un attribut style intégré dans lequel le code CSS est déclaré remplace tous les autres CSS, quelle que soit sa position, sauf si une déclaration est définie à “**!important**”.

La position s'applique également dans l'ordre de votre règle CSS.

Dans cet exemple, l'élément aura un arrière-plan violet, car **background: purple** a été déclaré en dernier. Comme le fond vert a été déclaré avant l'arrière-plan violet, il est désormais ignoré par le navigateur.

```
.my-element {  
  background: green;  
  background: purple;  
}
```

Être capable de spécifier deux valeurs pour la même propriété peut être un moyen simple de créer des solutions de remplacement pour les navigateurs qui n'acceptent pas une valeur particulière.

La cascade

Dans l'exemple suivant, **font-size** est déclaré deux fois. Si **clamp()** est compatible avec le navigateur, la déclaration **font-size** précédente sera supprimée. Si **clamp()** n'est pas compatible avec le navigateur, la déclaration initiale sera respectée et la taille de la police sera de **1.5 rem**.

```
.my-element {  
  font-size: 1.5rem;  
  font-size: clamp(1.5rem, 1rem + 3vw, 2rem);  
}
```



Remarque : Utiliser cette approche qui consiste à déclarer deux fois la même propriété fonctionne, car les navigateurs ignorent les valeurs qu'ils ne comprennent pas. Contrairement à d'autres langages de programmation, CSS ne génère pas d'erreur ni n'interrompt votre programme lorsqu'il détecte une ligne qu'il ne peut pas analyser. La valeur qu'il ne peut pas analyser n'est pas valide et est donc ignorée. Le navigateur continue ensuite à traiter le reste du code CSS sans altérer les éléments qu'il comprend déjà.

La spécificité



La spécificité

Supposons que vous utilisiez les codes HTML et CSS suivants:

```
<button class="hello">Hello, Spécificité !</button>
```

```
button {  
  color: red;  
}  
.hello {  
  color: blue;  
}
```

Il y a deux règles concurrentes.

L'une colorera le bouton en rouge et l'autre en bleu.

Quelle règle s'applique à l'élément ?

Il est essentiel de comprendre l'algorithme de la spécification CSS concernant la spécificité pour comprendre comment le CSS décide entre des règles concurrentes.

La spécificité est l'une des quatre étapes distinctes de la cascade, que nous avons abordée avec la cascade.

La spécificité

Évaluation de la spécificité

Chaque règle de sélecteur reçoit un score. Vous pouvez considérer la spécificité comme un score total, et chaque type de sélecteur gagne des points pour ce score. Le sélecteur ayant le score le plus élevé l'emporte.

Avec la spécificité d'un projet réel, l'équilibre consiste à s'assurer que les règles CSS que vous prévoyez d'appliquer sont effectivement appliquées, tout en limitant généralement les scores pour éviter la complexité.

Le score doit être aussi élevé que nécessaire, au lieu de viser le score le plus élevé possible.

À l'avenir, vous devrez peut-être appliquer de nouvelles règles CSS plus importantes.

Si vous visez le score le plus élevé, vous rendra votre travail difficile.

Noter chaque type de sélecteur

Chaque type de sélecteur permet de gagner des points. Vous additionnez tous ces points pour calculer la spécificité globale d'un sélecteur.

La spécificité

Sélecteur universel

Un sélecteur universel (*) n'a aucune spécificité et n'obtient 0 point.
Cela signifie que toute règle avec un ou plusieurs points la remplacera

```
*{  
  color: red;  
}
```

Sélecteur d'éléments ou de pseudo-éléments

Un sélecteur d'élément (type) ou de pseudo-élément obtient 1 point de spécificité .

Sélecteur de type

```
div{  
  color: red;  
}
```

Sélecteur de pseudo-éléments

```
::selection{  
  color: red;  
}
```

Sélecteur de classe, de pseudo-classe ou d'attribut

Un sélecteur de classe, de pseudo-classe ou d'attribut obtient 10 points de spécificité.

Sélecteur de classe

```
.my-class{  
  color: red;  
}
```

La spécificité

Sélecteur de pseudo-classe

```
:hover{  
  color: red;  
}
```

Sélecteur d'attribut

```
[href='#']{  
  color: red;  
}
```

La pseudo-classe **:not()** n'ajoute rien au calcul de spécificité. Toutefois, les sélecteurs transmis en tant qu'arguments sont ajoutés au calcul de la spécificité.

Cet exemple présente 11 points de spécificité, car il comporte un sélecteur de type **div** et une classe à l'intérieur de **:not()**.

```
div:not(.my-class){  
  color: red;  
}
```

Sélecteur d'ID

Un sélecteur d'ID obtient 100 points de spécificité, à condition que vous utilisiez un sélecteur d'ID **#myID** et non un sélecteur d'attributs **[id="myID"]**.

```
#myID{  
  color: red;  
}
```

La spécificité

Attribut de style intégré

Le code CSS appliqué directement à l'attribut style de l'élément HTML obtient un score de spécificité de 1 000 points. Autrement dit, pour le remplacer en CSS, vous devez écrire un sélecteur extrêmement spécifique.

```
<div style="color: red"><div>
```

Ajout de la règle **important**.

Enfin, un **important** à la fin d'une valeur CSS obtient un score de spécificité de 10 000 points.

Il s'agit du niveau de spécificité le plus élevé qu'un élément individuel peut obtenir.

Une règle **important** est appliquée à une propriété CSS. Par conséquent, tout ce qui se trouve dans la règle globale (sélecteur et propriétés) n'obtient pas le même score de spécificité.

```
.my-class{  
  color: red !important; /* 10,000 points */  
  background: white; /* 10 points */  
}
```


La spécificité

Spécificité en contexte

La spécificité de chaque sélecteur correspondant à un élément est additionnée. Prenons l'exemple de code HTML suivant:

```
<a class="first-class second-class" href="#">Lien</a>
```

Ce lien comporte deux classes. Ajoutez le CSS suivant pour qu'il obtienne **un point de spécificité**:

```
a{ color: red; }
```

Référence l'une des classes de cette règle, qui comporte désormais **11 points de spécificité**:

```
a.first-class{ color: green; }
```

Ajoutez l'autre classe au sélecteur. Elle dispose maintenant de **21 points de spécificité**:

```
a.first-class.second-class{ color: purple; }
```

Référence l'une des classes de cette règle, qui comporte désormais **31 points de spécificité**:

La spécificité

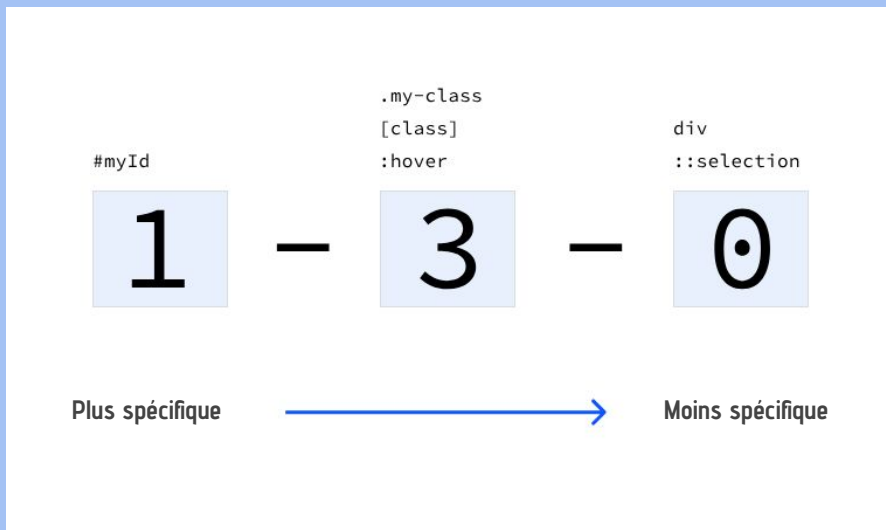
```
a.first-class.second-class[href]{ color: blue; }
```

Enfin, ajoutez une pseudo-classe :hover à tout cela. Le sélecteur se termine par **41 points de spécificité**:

```
a.first-class.second-class[href]:hover{ color: pink; }
```

Calcul de la spécificité

Dans les diagrammes et les calculateurs de spécificité, la spécificité est souvent représentée comme suit:



Le groupe de gauche se compose de sélecteurs **id**. Le second groupe comprend les sélecteurs de **classe**, **d'attribut** et de **pseudo-classe**.

Le dernier groupe est constitué des sélecteurs d'**éléments** et de **pseudo-éléments**.

La spécificité

Augmenter pragmatiquement la spécificité

Imaginons que notre code CSS se présente comme suit:

```
.my-button {  
  background: blue;  
}  
button[onclick] {  
  background: grey;  
}
```

Avec du code HTML qui ressemble à ceci:

```
<button class="my-button" onclick="alert('hello')">Cliquer moi</button>
```

L'arrière-plan du bouton est **gris**, car le deuxième sélecteur gagne 11 points de spécificité (0-1-1). En effet, il possède un sélecteur de type **button**, à savoir 1 point, et un sélecteur d'attribut **[onclick]**, qui équivaut à 10 points.

La règle précédente **.my-button** obtient 10 points (0-1-0), car elle comporte un sélecteur de classe.

Si vous souhaitez booster cette règle, répétez le sélecteur de classe comme suit:

```
button.my-button {  
  background: blue;  
}  
button[onclick] {  
  background: grey;  
}
```

Le bouton présente désormais un arrière-plan **bleu**, car le nouveau sélecteur obtient un score de spécificité de 20 points (0-2-0).



Attention : *Si vous avez fréquemment besoin d'améliorer la spécificité de cette méthode, cela peut signifier que vous rédigez des sélecteurs trop spécifiques. Déterminez si vous pouvez factoriser votre CSS pour réduire la spécificité d'autres sélecteurs et éviter ce problème.*

CLICK ME

CLICK ME

La spécificité

Avec un score de spécificité égal, la dernière règle établie l'emporte.

Poursuivons l'exemple du bouton pour l'instant et changeons le CSS de la manière suivante:

Le bouton a un fond gris, car **les deux sélecteurs ont le même score de spécificité** (0-1-0)

Si vous changez les règles dans l'ordre des sources, le bouton devient bleu, car cette règle correspond à la spécificité d'un autre sélecteur (**.my-button**) qui cible le même élément.

```
.my-button{  
  background:blue;  
}  
[onclick]{  
  background: grey;  
}
```

```
[onclick]{  
  background: grey;  
}  
.my-button{  
  background:blue;  
}
```

L'héritage



L'héritage

Imaginons que vous veniez d'écrire du code CSS pour que les éléments ressemblent à des boutons.

```
<article>
  <a class="my-button" href="http://exemple.fr">
    Bouton Lien
  </a>
</article>
```

Vous ajoutez ensuite un élément **link** à un article de contenu, avec la valeur class de **.my-button**.

Cependant, il y a un problème, le texte ne correspond pas à la couleur que vous attendiez.

Que s'est-il passé ?

Si vous n'indiquez aucune valeur, certaines propriétés CSS héritent des paramètres.

Dans le cas de ce bouton, il a hérité de la propriété **color** de ce CSS:

Le mécanisme d'**héritage** est une fonctionnalité puissante qui vous aide à écrire moins de code CSS.

```
.my-button {
  background: pink;
  display: inline-block;
  padding: 1rem 2rem;
  text-decoration: none;
  font: inherit;
  text-align: center;
}
```

```
article a {
  color: marron;
}
```

L'héritage

Flux d'héritage

Examinons le fonctionnement de l'héritage, à l'aide de l'extrait de code HTML suivant:

```
<html>
  <body>
    <article>
      <p>Lorem ipsum dolor sit amet</p>
    </article>
  </body>
</html>
```

L'élément racine **<html>** n'hérite de rien, car il s'agit du premier élément du document. Ajoutez du code CSS à l'élément HTML, et il commencera à se répercuter sur le document.

```
html {
  color: gray;
}
```

L'héritage

La propriété **color** est héritée par défaut par d'autres éléments.

L'élément **html** comporte la déclaration : **color**: “gray”.

Par conséquent, tous les éléments pouvant hériter d'une couleur auront désormais une couleur “gray”.

```
body {  
  font-size: 1.2rem;  
}
```



Remarque : Comme cette démonstration définit la taille de la police sur l'élément `body`, la taille de police initiale de l'élément `html` sera toujours définie par le navigateur (feuille de style “user-agent”), mais `article` et `p` hériteront de la taille de police déclarée par le `body`. En effet, l'héritage n'est répercuté que vers le bas.

```
p {  
  font-style: italic;  
}
```

Seul **<p>** présentera du texte en italique, car il s'agit de l'élément imbriqué le plus profond. L'héritage ne revient que vers le bas, et non vers les éléments parents.

L'héritage

Quelles propriétés sont héritées par défaut ?

Les propriétés CSS ne sont pas toutes héritées par défaut, mais il y en a beaucoup. Pour référence, voici la liste complète des propriétés héritées par défaut, tirées de la référence W3 de toutes les propriétés CSS:

- azimuth
- border-collapse
- border-spacing
- caption-side
- color
- cursor
- direction
- empty-cells
- font-family
- font-size
- font-style
- font-variant
- font-weight
- font
- letter-spacing
- line-height
- list-style-image
- list-style-position
- list-style-type
- list-style
- orphans
- quotes
- text-align
- text-indent
- text-transform
- visibility
- white-space
- widows
- word-spacing

L'héritage

Fonctionnement de l'héritage

Chaque élément HTML a chaque propriété CSS définie par défaut avec une valeur initiale. Une valeur initiale est une propriété qui n'est pas héritée et apparaît par défaut si la cascade ne parvient pas à calculer une valeur pour cet élément.

Les propriétés pouvant être héritées sont diffusées en cascade vers le bas, et les éléments enfants obtiennent une valeur calculée qui représente la valeur de leur parent. Cela signifie que si font-weight est défini sur bold pour un parent, tous les éléments enfants seront en gras, sauf si leur font-weight est définie sur une valeur différente ou si la feuille de style user-agent comporte une valeur font-weight pour cet élément.

Comment hériter et contrôler explicitement l'héritage

L'héritage peut affecter les éléments de manière inattendue. Le CSS dispose donc d'outils pour vous aider.

Le mot clé inherit

Vous pouvez configurer une propriété pour qu'elle hérite de la valeur calculée de son parent à l'aide du mot clé **inherit**.

L'héritage

```
strong{  
  font-weight: 900;  
}
```

Cet extrait CSS définit tous les éléments **** comme ayant une **font-weight** de **900**, au lieu de la valeur **bold** par défaut, qui serait l'équivalent de *font-weight: 700*.

```
.my-component{  
  font-weight: 500;  
}
```

La classe **.my-component** définit **font-weight** sur **500** à la place. Pour rendre les éléments **** dans **.my-component** également, **font-weight: 500** ajoutez:

```
.my-component strong{  
  font-weight: inherit;  
}
```

Désormais, les éléments **** dans **.my-component** auront une **font-weight** de **500**.

Vous pouvez définir explicitement cette valeur, mais si vous utilisez **inherit** et que le CSS de **.my-component** change à l'avenir, vous pouvez garantir que votre **** sera automatiquement à jour.

Le mot clé initial

L'héritage peut causer des problèmes avec vos éléments, et **initial** vous offre une option de réinitialisation performante. Vous avez appris précédemment que chaque propriété possède une valeur par défaut en CSS. Le mot clé **initial** rétablit la valeur initiale par défaut pour une propriété.

L'héritage

```
aside strong{
  font-weight: initial;
}
```

Cet extrait supprimera l'épaisseur en gras de tous les éléments `` d'un élément `<aside>` et le rendra normal, qui correspond à la valeur initiale.

Le mot clé unset

La propriété **unset** se comporte différemment selon qu'une propriété est héritée par défaut ou non.

Si une propriété est héritée par défaut, le mot clé **unset** sera identique à **inherit**.

Si la propriété n'est pas héritée par défaut, le mot clé **unset** est égal à **initial**.

Il peut être difficile de se souvenir des propriétés CSS héritées par défaut. Dans ce contexte, **unset** peut s'avérer utile.

Par exemple, **color** est hérité par défaut, mais pas **margin**. Vous pouvez donc écrire ceci:

```
p{
  margin-top: 2rem;
  color: blue;
}
aside p{
  margin: unset;
  color: unset;
}
```

Désormais, **margin** est supprimé, et **color** revient à la valeur calculée héritée.

Vous pouvez également utiliser la valeur unset avec la propriété all.

Pour en revenir à l'exemple ci-dessus, que se passe-t-il si l'on ajoute quelques propriétés supplémentaires aux styles de **p** ?

Seule la règle définie pour margin et color s'appliquera.

L'héritage

Désormais, **margin** est supprimé, et **color** revient à la valeur calculée héritée.

Vous pouvez également utiliser la valeur **unset** avec la propriété **all**.

Pour en revenir à l'exemple ci-dessus, que se passe-t-il si l'on ajoute quelques propriétés supplémentaires aux styles de **p** ?
Seule la règle définie pour **margin** et **color** s'appliquera.

```
p{
  margin-top: 2rem;
  color: blue;
}
aside p{
  margin: unset;
  color: unset;
}
```

```
p{
  margin-top: 2rem;
  color: blue;
  padding: 2rem;
  border: 1px solid;
}
aside p{
  margin: unset;
  color: unset;
}
```

Si vous remplacez la règle **aside p** par **all:unset**, les styles globaux qui seront appliqués à **p** à l'avenir n'auront pas d'importance, ils seront toujours non définis.

```
aside p{
  margin: unset;
  color: unset;
  all: unset;
}
```

Les couleurs



Les couleurs

La couleur est une partie importante de tout site Web. En CSS, il existe de nombreuses options pour les types de couleurs, les fonctions et les traitements.

Comment décidez-vous quel type de couleur utiliser ? Comment rendre vos couleurs semi-transparentes ? Dans cette leçon, vous allez apprendre les options dont vous disposez pour vous aider à prendre les bonnes décisions pour votre projet et votre équipe.

Le CSS dispose de différents types de données, tels que des chaînes et des nombres. La couleur est l'un de ces types et utilise d'autres types, tels que les nombres pour ses propres définitions.

Couleurs numériques

Il est très probable que votre première exposition aux couleurs dans CSS ait lieu via des couleurs numériques. Nous pouvons travailler avec des valeurs de couleur numériques sous différentes formes.

Les couleurs

Couleurs hexadécimales

```
h1 {  
  color: #b71540;  
}
```

La notation hexadécimale (souvent abrégée en hexadécimal) est une syntaxe abrégée pour le RVB, qui attribue une valeur numérique au rouge, au vert et au bleu, qui sont les trois couleurs primaires.

Les plages hexadécimales sont 0-9 et A-F.

Lorsqu'elles sont utilisées dans une séquence de six chiffres, elles sont traduites en plages numériques RVB, qui sont comprises entre 0 et 255, correspondant respectivement aux canaux rouge, vert et bleu.

Vous pouvez également définir une valeur alpha avec n'importe quelle couleur numérique. Une valeur alpha est un pourcentage de transparence. En code hexadécimal, vous ajoutez deux autres chiffres à la séquence de six chiffres, ce qui donne une séquence de huit chiffres. Par exemple, pour définir le noir en code hexadécimal, écrivez #000000. Pour ajouter une transparence de 50 %, remplacez-la par #00000080.

Étant donné que l'échelle hexadécimale est 0-9 et A-F, les valeurs de transparence ne sont probablement pas celles que vous attendiez. Voici quelques valeurs clés courantes ajoutées au code hexadécimal noir, #000000:

Les couleurs

- La valeur alpha de 0 % (qui est entièrement transparente) correspond à 00: #00000000
- La valeur alpha 50% correspond à 80: #00000080
- 75% alpha correspond à BF: #000000BF

Pour convertir un hexadécimal à deux chiffres en décimal, prenez le premier chiffre et multipliez-le par 16 (car l'hexadécimal est la base 16), puis ajoutez le deuxième chiffre.

En utilisant l'exemple BF comme exemple pour une valeur alpha à 75 % :

1. B est égal à 11, qui équivaut à 176 lorsque 16 est multiplié par 16
2. F est égal à 15
3. $176 + 15 = 191$
4. La valeur alpha est comprise entre 191 et 75% de 255.



Remarque : Vous pouvez également écrire des codes hexadécimaux en utilisant un raccourci à trois chiffres. Un code hexadécimal à trois chiffres est un raccourci vers une séquence équivalente à six chiffres. Par exemple, #a4e est identique à #aa44ee. Pour ajouter la valeur alpha, #a4e8 est étendu à #aa44ee88.

Les couleurs

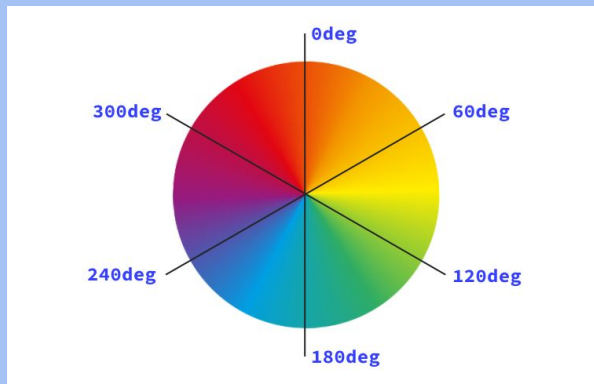


Remarque : Les virgules ont été supprimées de la notation `rgb()` et `hsl()`, car les fonctions de couleur plus récentes, telles que `lab()` et `lch()`, utilisent des espaces au lieu de virgules comme délimiteurs. Cette modification offre plus de cohérence non seulement avec les nouvelles fonctions de couleur, mais aussi avec CSS en général. Pour améliorer la rétrocompatibilité, vous pouvez toujours utiliser des virgules pour définir `rgb()` et `hsl()`.

Couleurs TSL (teinte, saturation, luminosité)

```
h1 {  
  color: hsl(344, 79%, 40%);  
}
```

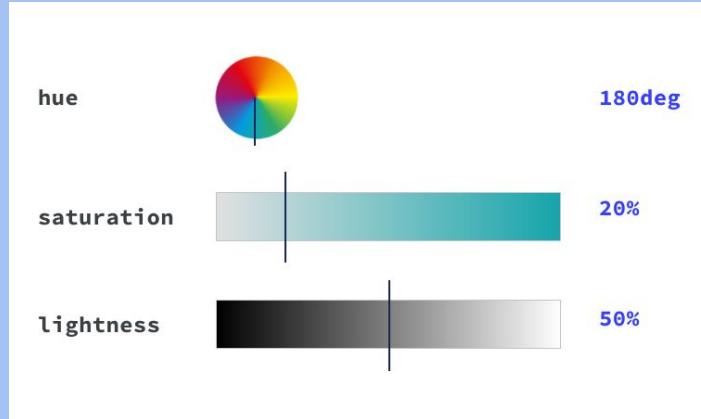
HSL signifie teinte, saturation et luminosité. La teinte décrit la valeur indiquée sur la roue chromatique, de 0 à 360 degrés, en commençant par le rouge (0 et 360). Une teinte de 180 ou 50% correspondrait à la plage bleue. C'est l'origine de la couleur que nous voyons.



Les couleurs


La saturation indique l'éclat de la teinte sélectionnée. Une couleur entièrement désaturée (avec une saturation de 0%) apparaîtra en nuances de gris. Enfin, la luminosité est le paramètre qui décrit l'échelle du blanc au noir de la lumière supplémentaire. Une luminosité de 100% donne toujours du blanc.

À l'aide de la fonction de couleur `hsl()`, vous définissez un vrai noir en écrivant `hsl(0 0% 0%)`, voire `hsl(0deg 0% 0%)`. En effet, le paramètre de teinte définit le degré dans la roue chromatique, qui, si vous utilisez le type numérique, est 0-360. Vous pouvez également utiliser le type d'angle, à savoir (0deg) ou (0turn). La saturation et la luminosité sont définies à l'aide de pourcentages.



Les couleurs

Alpha est défini dans `hsl()`, de la même manière que `rgb()` en ajoutant une `/` après les paramètres de teinte, saturation et luminosité, ou à l'aide de la fonction `hsla()`. La valeur alpha peut être définie avec un pourcentage ou un nombre décimal compris entre 0 et 1. Par exemple, pour définir un noir alpha à 50 %, utilisez `hsl(0 0% 0% / 50%)` ou `hsl(0 0% 0% / 0.5)`. À l'aide de la fonction `hsla()`, écrivez `hsla(0, 0%, 0%, 50%)` ou `hsla(0, 0%, 0%, 0.5)`.


 **Remarque** : De nouveaux types de couleurs sont disponibles dans CSS. Ces méthodes incluent **`lab()`** et **`lch()`**, qui permettent de spécifier une plage de couleurs beaucoup plus large qu'en RVB.

Couleurs nommées

Il existe 148 couleurs nommées en CSS. Il s'agit de noms anglais simples tels que violet, tomato et goldenrod. Outre les couleurs standards, des mots clés spécifiques sont également disponibles :

- **transparent** est une couleur totalement transparente. C'est aussi la valeur initiale de `background-color`.
- **currentColor** est la valeur dynamique contextuelle calculée de la propriété **color**. Si vous utilisez une couleur de texte `red` et que vous définissez `border-color` sur **currentColor**, il sera également rouge. Si l'élément pour lequel vous définissez `currentColor` n'a pas de valeur pour `color` définie, `currentColor` sera calculé par la cascade à la place.

Les couleurs

 **Remarque** : Les mots clés système sont des couleurs définies par le thème de votre système d'exploitation. Voici quelques exemples de ces couleurs : *Background*, qui est la couleur d'arrière-plan, ou *Highlight*, qui est la couleur de mise en surbrillance des éléments sélectionnés. Il ne s'agit là que de deux des nombreuses options disponibles. Les mots clés de couleur ne sont pas sensibles à la casse. Toutefois, des couleurs système avec majuscules s'affichent souvent afin de les différencier des mots clés de couleur standards.

Où utiliser la couleur dans les règles CSS ?

Si une propriété CSS accepte le type de données `<color>` en tant que valeur, elle accepte toutes les méthodes d'expression de couleur ci-dessus. Pour styliser le texte, utilisez les propriétés `color`, `text-shadow` et `text-decoration-color`, qui acceptent toutes la couleur comme valeur ou comme couleur dans la valeur. Pour les arrière-plans, vous pouvez définir une couleur comme valeur pour `background` ou `background-color`. Vous pouvez également utiliser des couleurs dans les dégradés, comme `linear-gradient`. Les dégradés sont un type d'image qui peut être défini de façon programmatique en CSS. Les dégradés acceptent au moins deux couleurs dans n'importe quelle combinaison de formats de couleurs (hex, rvg ou hsl, par exemple). Enfin, `border-color` et `outline-color` définissent la couleur des bordures et des contours de vos zones. La propriété `box-shadow` accepte également la couleur.

Les couleurs

Couleur et contraste

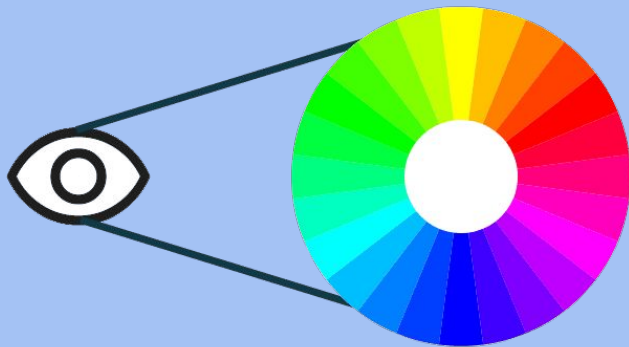
Avez-vous déjà essayé de lire du texte sur un écran et l'avez trouvé difficile à lire en raison du jeu de couleurs ou avez-vous eu du mal à voir l'écran dans un environnement très lumineux ou faiblement éclairé ? Ou peut-être avez-vous un problème de vision des couleurs plus permanent, comme 300 millions de personnes daltoniennes ou 253 millions de personnes malvoyantes ?

En tant que concepteur ou développeur, vous devez comprendre comment les gens perçoivent les couleurs et les contrastes, que ce soit de manière temporaire, en fonction de la situation ou de manière permanente. Cela vous aidera à mieux répondre à ses besoins visuels.

Perception des couleurs

Saviez-vous que les objets ne possèdent pas de couleur, mais qu'ils reflètent les longueurs d'onde de la lumière ? Lorsque vous voyez des couleurs, vos yeux reçoivent et traitent ces longueurs d'onde et les convertissent en couleurs.

Les couleurs



En ce qui concerne l'accessibilité numérique, nous parlons de ces longueurs d'onde en termes de teinte, de saturation et de luminosité (HSL). Le modèle HSL a été créé comme alternative au modèle de couleurs RVB et s'aligne plus précisément sur la façon dont un humain perçoit la couleur.



Remarque : En CSS, vous pouvez spécifier une couleur de différentes manières, par exemple à l'aide de noms de couleur, RVB, RVBA, HEX, HSL, HSLa, HSV ou HSVa. HSLa est similaire à HSL, seule une valeur alpha a été ajoutée. Alpha est une mesure de l'opacité, définie comme un nombre compris entre 0 (totalement transparent) et 1,0 (totalement opaque).

La teinte est une manière qualitative de décrire une couleur, telle que le rouge, le vert ou le bleu, où chaque teinte a un point spécifique sur le spectre de couleurs avec des valeurs comprises entre 0 et 360, le rouge à 0, le vert à 120 et le bleu à 240.

Les couleurs

Mesurer le contraste des couleurs

Pour aider les personnes souffrant de divers handicaps visuels, le groupe WAI a créé une formule de contraste des couleurs pour s'assurer qu'un contraste suffisant existe entre le texte et son arrière-plan. Lorsque ces rapports de contraste des couleurs sont suivis, les personnes modérément malvoyantes peuvent lire le texte en arrière-plan sans avoir besoin d'une technologie d'assistance qui améliore le contraste.

Deutéranopie

La deutéranopie (communément appelée daltonisme) affecte 1 % à 5 % des hommes, et 0,35 à 0,1 % des femmes. Les personnes atteintes de deutéranopie sont moins sensibles à la lumière verte. Ce filtre de daltonisme simule ce type de daltonisme.

Protanopie

La protanopie (communément appelée daltonisme) affecte 1,01% à 1,08% des hommes et 0,02% de 0,03% des femmes. Les personnes atteintes de protanopie ont une sensibilité réduite à la lumière rouge. Ce filtre de daltonisme simule ce type de daltonisme.

Achromatopsie ou monochromatisme

L'achromatopsie ou le monochrome (ou daltonisme complet) n'apparaît que très, très rarement. Les personnes souffrant d'achromatopsie ou de monochrome n'ont pratiquement aucune perception de la lumière rouge, verte ou bleue. Ce filtre de daltonisme simule ce à quoi pourrait ressembler ce type de daltonisme.

Les couleurs

Calculer le contraste des couleurs

La formule du contraste des couleurs utilise la luminance relative des couleurs pour aider à déterminer le contraste, qui peut varier de 1 à 21. Cette formule est souvent abrégée en [color value]:1. Par exemple, le noir pur sur le blanc pur a le rapport de contraste des couleurs le plus élevé : 21:1.

Le texte de taille standard, y compris les images de texte, doit avoir un rapport de contraste des couleurs de 4.5:1 pour respecter les exigences minimales des WCAG concernant la couleur. Le texte de grande taille et les icônes essentielles doivent présenter un rapport de contraste des couleurs de 3:1. Les textes de grande taille sont caractérisés par une mise en gras d'au moins 18 pt / 24 pixels ou 14 pt/18,5 pixels. Les logos et les éléments décoratifs ne sont pas soumis à ces exigences de contraste des couleurs.

Heureusement, aucune opération mathématique avancée n'est requise, car de nombreux outils effectuent le calcul du contraste des couleurs à votre place. Des outils tels qu'Adobe Color, l'analyseur de contraste des couleurs, Leonardo et le sélecteur de couleur des outils pour les développeurs Chrome peuvent vous indiquer rapidement les rapports de contraste des couleurs et proposer des suggestions pour vous aider à créer les paires de couleurs et les palettes les plus inclusives.

Les couleurs

Utiliser la couleur

En l'absence de niveaux de contraste des couleurs appropriés, les mots, les icônes et les autres éléments graphiques sont difficiles à découvrir, et la conception peut rapidement devenir inaccessible. Mais il est également important de prêter attention à la façon dont la couleur est utilisée à l'écran, car vous ne pouvez pas l'utiliser seule pour transmettre des informations ou des actions, ni distinguer un élément visuel.

Par exemple, si vous dites cliquez sur le bouton vert pour continuer, mais omettez tout contenu ou identifiant supplémentaire au bouton, il sera difficile pour les personnes atteintes de certains types de daltonisme de savoir sur quel bouton cliquer. De même, de nombreux graphiques, diagrammes et tableaux utilisent uniquement la couleur pour transmettre des informations. L'ajout d'un autre identifiant, comme un motif, du texte ou une icône, est essentiel pour aider les utilisateurs à comprendre le contenu.

L'examen de vos produits numériques en nuances de gris est un bon moyen de détecter rapidement les problèmes de couleur potentiels.

Les couleurs

Requêtes média axées sur les couleurs

En plus de vérifier les rapports de contraste des couleurs et l'utilisation des couleurs à l'écran, vous devriez envisager d'appliquer les requêtes média de plus en plus populaires et compatibles, qui offrent aux utilisateurs plus de contrôle sur ce qui s'affiche à l'écran.

Par exemple, à l'aide de la requête média `@prefers-color-scheme`, vous pouvez créer un thème sombre, qui peut être utile aux personnes souffrant de photophobie ou de sensibilité à la lumière. Vous pouvez également créer un thème à contraste élevé avec `@prefers-contrast`, qui répond aux besoins des personnes présentant des déficiences de couleurs et une sensibilité au contraste.



Remarque : Des requêtes média et des paramètres d'OS supplémentaires sont à prendre en compte pour l'accessibilité des couleurs, mais ils sont beaucoup moins pris en charge que les deux listés dans ce module. Pour en savoir plus sur les différents paramètres d'accessibilité de l'OS, consultez l'article [Système d'exploitation et modes d'affichage de l'accessibilité du navigateur](#).

Les couleurs

Jeu de couleurs préféré

La requête média @prefers-color-scheme permet aux utilisateurs de choisir une version claire ou sombre du site Web ou de l'application qu'ils consultent. Vous pouvez observer ce changement de thème en modifiant vos paramètres de préférence clair/sombre et en accédant à un navigateur compatible avec cette requête multimédia. Consultez les instructions pour Mac et Windows concernant le mode sombre.

Contraste préféré

La requête média @prefers-contrast vérifie les paramètres de l'OS de l'utilisateur pour voir si le contraste élevé est activé ou désactivé. Pour voir ce changement de thème en action, modifiez vos paramètres de préférences de contraste et accédez à un navigateur compatible avec cette requête multimédia (paramètres du mode de contraste Mac et Windows).

Utiliser toutes ces requêtes média

Vous pouvez utiliser ensemble toutes ces requêtes média axées sur les couleurs pour offrir encore plus de choix à vos utilisateurs.

Les unités




Les unités

Les nombres

Des nombres sont utilisés pour définir les champs `opacity` et `line-height`, et même les valeurs de canal de couleur dans `rgb`. Les nombres sont des entiers sans unité (1, 2, 3, 100) et des nombres décimaux (0,1, 0,2, 0,3). Les nombres ont une signification selon leur contexte. Par exemple, lorsque vous définissez `line-height`, un nombre est représentatif d'un ratio si vous le définissez sans unité secondaire:

```
p {  
  font-size: 24px;  
  line-height: 1.5;  
}
```

Dans cet exemple, 1.5 est égal à 150% de la taille de police calculée en pixels de l'élément `p`. Cela signifie que si `p` a une `font-size` de 24px, la hauteur de la ligne sera calculée comme 36px.

 **Remarque :** Nous vous recommandons d'utiliser une valeur sans unité pour `line-height`, plutôt que d'en spécifier une. Comme vous l'avez appris dans le module sur l'héritage, `font-size` peut être hérité. Définir un `line-height` sans unité permet de conserver la hauteur de ligne par rapport à la taille de la police. Cette méthode offre une meilleure expérience que `line-height: 15px`, par exemple, qui ne change pas et peut paraître étrange avec certaines tailles de police.

Les unités

Les nombres peuvent également être utilisés dans les cas suivants::

- Lorsque vous définissez des valeurs de filtres: **filter: sepia(0.5)** applique un filtre sépia 50% à l'élément.
- Lorsque vous définissez l'opacité, **opacity: 0.5** applique une opacité de 50%.
- Dans les canaux de couleur: **rgb(50, 50, 50)**, où les valeurs comprises entre 0 et 255 sont acceptables pour définir une valeur de couleur.
- Pour transformer un élément, **transform: scale(1.2)** redimensionne l'élément de 120% par rapport à sa taille initiale.

Pourcentages

Lorsque vous utilisez un pourcentage dans CSS, vous devez savoir comment il est calculé.

Par exemple, width est calculé en tant que pourcentage de la largeur disponible dans l'élément parent.

```
div {  
  width: 300px;  
  height: 100px;  
}  
div p {  
  width: 50%;  
}
```

Les unités

Dans l'exemple ci-dessus, la largeur de `div p` est 150px, en supposant que la mise en page utilise la `box-sizing: content-box` par défaut.

Si vous définissez `margin` ou `padding` en pourcentage, elles prendront partie de la largeur de l'élément parent, quelle que soit la direction.

```
div {  
  width: 300px;  
  height: 100px;  
}  
div p {  
  margin-top: 50%; /* calculé: 150px */  
  padding-left: 50%; /* calculé: 150px */  
  transform: translateX(10%); /* calculé: 15px */  
}
```

Si vous définissez une valeur `transform` en pourcentage, elle est basée sur l'élément avec l'ensemble de transformation. Dans cet exemple, `p` a la valeur `translateX` de 10% et la valeur `width` de 50%. Tout d'abord, calculez la largeur: 150px, car elle correspond à 50% de la largeur de son parent. Prenez ensuite 10% sur 150px, soit 15px.

Les unités

Dans l'exemple ci-dessus, la largeur de `div p` est 150px, en supposant que la mise en page utilise la `box-sizing: content-box` par défaut.

Si vous définissez `margin` ou `padding` en pourcentage, elles prendront partie de la largeur de l'élément parent, quelle que soit la direction.

```
div {  
  width: 300px;  
  height: 100px;  
}  
div p {  
  margin-top: 50%; /* calculé: 150px */  
  padding-left: 50%; /* calculé: 150px */  
  transform: translateX(10%); /* calculé: 15px */  
}
```

Si vous définissez une valeur `transform` en pourcentage, elle est basée sur l'élément avec l'ensemble de transformation. Dans cet exemple, `p` a la valeur `translateX` de 10% et la valeur `width` de 50%. Tout d'abord, calculez la largeur: 150px, car elle correspond à 50% de la largeur de son parent. Prenez ensuite 10% sur 150px, soit 15px.

Les unités

Dimensions et longueurs

Si vous associez une unité à un nombre, elle devient une dimension. Par exemple, 1rem est une dimension. Dans ce contexte, l'unité associée à un nombre est appelée "jeton de dimension" dans les spécifications. Les longueurs sont des dimensions faisant référence à la distance. Elles peuvent être absolues ou relatives.

Longueurs absolues

Toutes les longueurs absolues sont résolues sur la même base, ce qui les rend prévisibles partout où elles sont utilisées dans votre CSS. Par exemple, si vous utilisez cm pour dimensionner votre élément, puis l'imprimer, le résultat devrait être précis si vous le comparez à une règle.

```
div {  
  width: 10cm;  
  height: 5cm;  
  background: black;  
}
```

Si vous imprimiez cette page, cette **div** sera imprimée sous la forme d'un rectangle noir de 10 x 5 cm.

N'oubliez pas que le CSS est utilisé non seulement pour le contenu numérique, mais aussi pour le style du contenu imprimé.

Les longueurs absolues peuvent s'avérer très utiles lors de la conception pour impression.

Les unités

Longueurs relatives

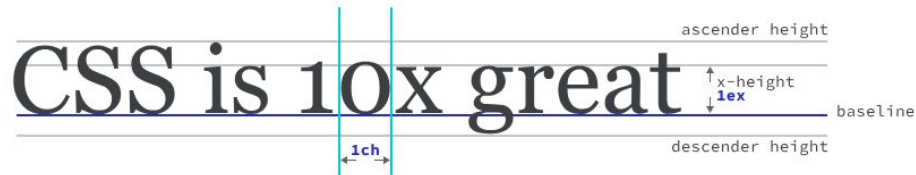
Toutes les longueurs absolues sont résolues sur la même base, ce qui les rend Une longueur relative est calculée par rapport à une valeur de base, comme un pourcentage. La différence entre ces valeurs et les pourcentages est que vous pouvez dimensionner les éléments en fonction du contexte. Cela signifie que CSS comporte des unités telles que ch qui utilisent la taille du texte comme base, et vw qui est basée sur la largeur de la fenêtre d'affichage (la fenêtre de votre navigateur). Les longueurs relatives sont particulièrement utiles sur le Web en raison de sa réactivité.

Unités de taille de police relative

Le code CSS fournit des unités utiles qui dépendent de la taille des éléments de la typographie affichée, comme la taille du texte lui-même (unité em) ou la largeur des caractères de police (unité ch).

Les unités

| unité | par rapport à: |
|------------|---|
| <u>em</u> | Par rapport à la taille de police, par exemple, 1,5 em sera 50% plus grand que la taille de police calculée de base de son parent. (Historiquement, la hauteur de la lettre majuscule "M"). |
| <u>ex</u> | Heuristique pour déterminer si la hauteur x, une lettre "x" ou ".5em" doit être utilisée dans la taille de police actuelle calculée de l'élément. |
| <u>cap</u> | Hauteur des lettres majuscules dans la taille de police actuellement calculée de l'élément. |
| <u>ch</u> | <u>Avancement des caractères moyen d'un glyphe étroit dans la police de l'élément (représenté par le glyphe "0").</u> |
| <u>ic</u> | <u>Avancement des caractères moyen d'un glyphe pleine largeur dans la police de l'élément, comme représenté par le glyphe "水" (idéogramme de l'eau CJK, U+6C34).</u> |
| <u>rem</u> | Taille de police de l'élément racine (16 pixels par défaut). |
| <u>lh</u> | Hauteur de ligne de l'élément. |
| <u>rlh</u> | Hauteur de ligne de l'élément racine. |



Les unités

Unités relatives à la fenêtre d'affichage

Vous pouvez utiliser les dimensions de la fenêtre d'affichage (fenêtre du navigateur) comme base relative. Ces unités correspondent à l'espace disponible de la fenêtre d'affichage.

| unité | par rapport à |
|-------------|---|
| <u>vw</u> | 1% de la largeur de la fenêtre d'affichage Les utilisateurs se servent de ce module pour effectuer des astuces sur les polices (par exemple, pour redimensionner la police d'en-tête en fonction de la largeur de la page, de sorte que la police se redimensionne à mesure que l'utilisateur redimensionne l'écran). |
| <u>vh</u> | 1% de la hauteur de la fenêtre d'affichage Vous pouvez l'utiliser pour organiser les éléments dans une interface utilisateur, si vous disposez d'une barre d'outils en pied de page, par exemple. |
| <u>vi</u> | 1% de la taille de la fenêtre d'affichage dans l'axe intégré de l'élément racine L'axe fait référence aux modes d'écriture. Dans les modes d'écriture horizontal tels que l'anglais, l'axe intégré est horizontal. Dans les modes d'écriture verticaux tels que certaines polices de caractères japonais, l'axe intégré s'exécute de haut en bas. |
| <u>vb</u> | 1% de la taille de la fenêtre d'affichage dans l'axe de bloc de l'élément racine Pour l'axe du bloc, il s'agit du sens de la langue. Les langues de gauche à droite, comme l'anglais, ont un axe de bloc vertical, car les lecteurs anglophones analysent la page de haut en bas. Un mode d'écriture vertical présente un axe de bloc horizontal. |
| <u>vmin</u> | 1% de la plus petite dimension de la fenêtre d'affichage |
| <u>vmax</u> | 1% de la plus grande dimension de la fenêtre d'affichage. |
| <u>rlh</u> | Hauteur de ligne de l'élément racine. |

Les unités

Unités diverses

D'autres unités ont été spécifiées pour traiter des types de valeurs spécifiques.

Unités d'angle

Dans le module de couleur, nous avons examiné les unités d'angle, qui sont utiles pour définir des valeurs en degrés, telles que la teinte dans hsl. Elles sont également utiles pour faire pivoter des éléments dans des fonctions de transformation.

```
div {  
  width: 150px;  
  height: 150px;  
  transform: rotate(60deg);  
}
```

Unités de résolution

Dans l'exemple précédent, la valeur de min-resolution est 192dpi. L'unité dpi correspond aux points par pouce (points par pouce). Pour cela, il est utile de détecter les écrans à très haute résolution, tels que les écrans Retina, dans une requête média, et de diffuser une image de résolution supérieure.

Mise en page



Mise en page

Mise en page: présent et futur

Le CSS moderne dispose d'outils de mise en page exceptionnellement puissants. Nous disposons de systèmes dédiés pour la mise en page. Nous allons voir dans les grandes lignes ce que nous avons à notre disposition, avant d'étudier plus en détail Flexbox et Grid dans les prochains modules.

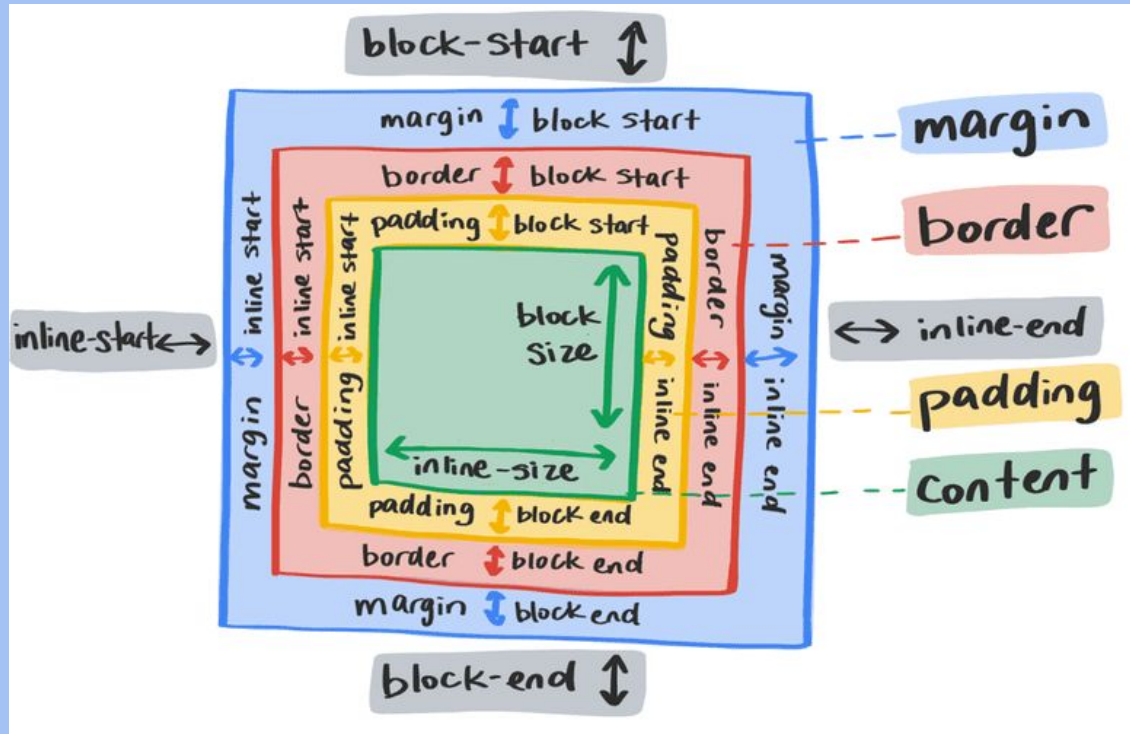
Comprendre la propriété display

La propriété display a deux fonctions. La première chose qu'il fait consiste à déterminer si la zone à laquelle il est appliqué agit en tant que zone intégrée ou en tant que bloc.

```
.my-element{  
  display: inline;  
}
```

Les éléments inline se comportent comme les mots d'une phrase. Ils sont posés l'un à côté de l'autre, dans le sens de l'alignement. Les éléments tels que `` et ``, qui sont généralement utilisés pour styliser des éléments de texte dans des éléments contenant tels qu'un `<p>` (paragraphe), sont intégrés par défaut. Elles conservent également les espaces blancs environnants.

Mise en page



Vous ne pouvez pas définir de largeur et de hauteur explicites pour les éléments inline. Les marges et marges intérieures de niveau bloc sont ignorées par les éléments environnants.

Mise en page

```
.my-element{  
  display: block;  
}
```

Les éléments de bloc ne sont pas côte à côte. Il crée une nouvelle ligne pour lui-même. À moins qu'il ne soit modifié par un autre code CSS, un élément de bloc s'agrandit jusqu'à la dimension intégrée, et s'étend donc sur toute la largeur en mode d'écriture horizontal. La marge de tous les côtés d'un élément de bloc sera respectée.

```
.my-element{  
  display: flex;  
}
```

La propriété `display` détermine également le comportement des enfants d'un élément. Par exemple, si vous définissez la propriété `display` sur `display: flex`, la zone devient une zone au niveau du bloc et convertit également ses enfants en éléments Flex. Cela permet d'activer les propriétés flexibles qui contrôlent l'alignement, l'ordre et le flux.

Mise en page

Flexbox et Grid

Il existe deux mécanismes de mise en page principaux qui créent des règles de mise en page pour plusieurs éléments : flexbox et grid. Elles partagent des similitudes, mais sont conçues pour résoudre différents problèmes de mise en page.

Flexbox

Flexbox est un mécanisme de mise en page conçu pour les mises en page unidimensionnelles. Mise en page sur un seul axe, horizontalement ou verticalement. Par défaut, Flexbox alignera les éléments enfants de l'élément les uns à côté des autres, dans le sens de l'alignement, et les étirera dans la direction du bloc. Ils auront ainsi tous la même hauteur.

```
.my-element{  
  display: flex;  
}
```

```
<main class="wrapper">  
  <div class="my-element">  
    <div class="box">Elément A</div>  
    <div class="box">Elément B</div>  
  </div>  
</main>
```

Elément A

Elément B

Mise en page

Grid

La grille est semblable à flexbox à bien des égards, mais elle est conçue pour contrôler les mises en page multi-axes plutôt que les mises en page à un seul axe (espace vertical ou horizontal).

La grille vous permet d'écrire des règles de mise en page sur un élément doté de `display: grid` et introduit de nouvelles primitives de style de mise en page, telles que les fonctions `repeat()` et `minmax()`. Une unité de grille utile est l'unité `fr`, qui correspond à une fraction de l'espace restant. Vous pouvez créer des grilles traditionnelles à 12 colonnes, avec un écart entre chaque élément, avec 3 propriétés CSS:

```
.my-element{  
  display: grid;  
}
```

```
<main class="wrapper">  
  <div class="my-element">  
    <div class="box">Elément A</div>  
    <div class="box">Elément B</div>  
    ...  
    <div class="box">Elément B</div>  
  </div>  
</main>
```

```
.my-element{  
  display: grid;  
  grid-template-columns: repeat(12, 1fr);  
  gap: 1rem;  
}
```

Elément
A

Elément
B

Elément
C

Elément
D

Elément
E

Elément
F

Elément
G

Elément
H

Mise en page

L'exemple ci-dessus présente une mise en page à un seul axe.

Alors que Flexbox traite principalement les éléments comme un groupe, la grille vous permet de contrôler précisément leur emplacement en deux dimensions.

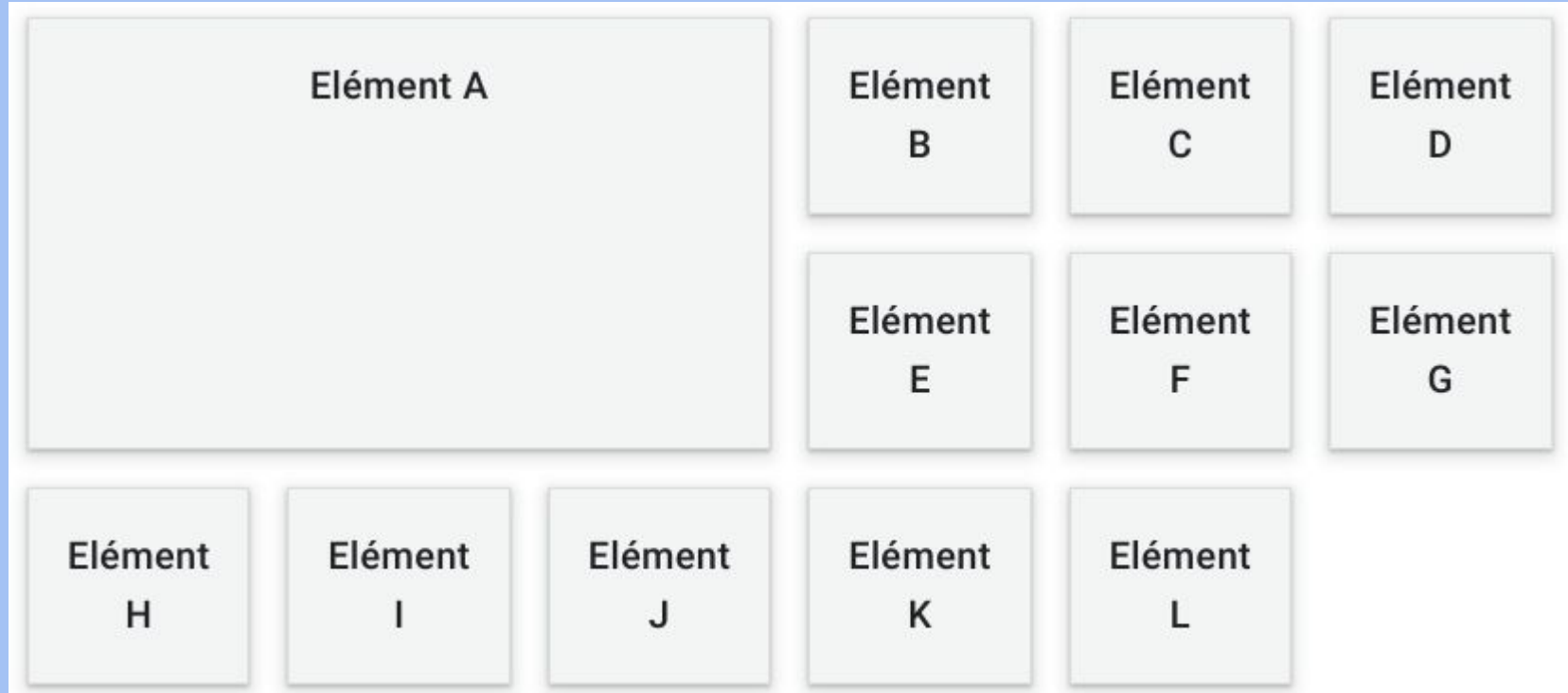
Nous pourrions définir que le premier élément de cette grille occupe deux lignes et trois colonnes:

```
<main class="wrapper">
  <div class="my-element">
    <div class="box">Elément A</div>
    <div class="box">Elément B</div>
    <div class="box">Elément C</div>
    ...
    ...
    <div class="box">Elément K</div>
    <div class="box">Elément L</div>
  </div>
</main>
```

```
.my-element{
  display: grid;
  grid-template-columns: repeat(6, 1fr);
  gap: 1rem;
  max-width: 500px
}
.my-element:first-child{
  display: grid;
  grid-row: 1/3;
  grid-column: 1/4;
}
```

Mise en page

Les propriétés `grid-row` et `grid-column` indiquent au premier élément de la grille de s'étendre au début de la quatrième colonne, à partir de la première colonne, puis de s'étendre à la troisième ligne à partir de la première ligne.



Mise en page

Mise en page du flux

Si vous n'utilisez pas de grille ni de Flexbox, vos éléments s'affichent selon un flux normal. Il existe un certain nombre de méthodes de mise en page que vous pouvez utiliser pour ajuster le comportement et la position des éléments dans un flux normal.

Bloc en ligne

Vous vous souvenez que les éléments environnants ne respectent pas la marge de bloc et la marge intérieure d'un élément "inline" ? Avec inline-block, vous pouvez résoudre ce problème.

```
p span{  
  display: inline-block;  
}
```

L'utilisation de inline-block vous donne une zone qui présente certaines des caractéristiques d'un élément au niveau du bloc, mais qui s'affiche toujours conformément au texte.

```
span{  
  margin-top: .5rem;  
}
```


Mise en page

```
<div>  
<span>Ces deux éléments</span> <span>sont des span normaux,  
qui sont <code>display: inline</code> par défaut</span>.  
<p>Cet <span>élément span</span>, à l'intérieur d'un élément  
paragraphe est <code>display: inline-block</code>, donc sa  
marge supérieure ne sera plus ignorée.  
</p>  
</div>
```

```
p span{  
  display: inline-block;  
}  
  
span{  
  margin-top: .5rem;  
  outline: 1px solid gray;  
}
```

Ces deux éléments sont des span normaux, qui sont `display: inline` par défaut.

Cet élément span, à l'intérieur d'un élément paragraphe est

`display: inline-block`, donc sa marge supérieure ne sera plus ignorée.

Mise en page

Float

Si vous avez une image dans un paragraphe de texte, ne serait-il pas pratique que ce texte entoure cette image comme vous pourriez le voir dans un journal ? Vous pouvez le faire avec “**float**”.

```
img{
  float: left;
  margin-right: 1rem;
}
```

! Attention : Lorsque vous utilisez l'option **float**, gardez à l'esprit que tous les éléments qui suivent l'élément flottant peuvent voir leur mise en page modifiée. Pour éviter cela, vous pouvez effacer le flottement, soit en utilisant **clear :both** sur un élément qui suit votre élément flottant, soit en utilisant **display:flow-root** sur le parent de vos éléments flottants.

La propriété **float** indique à un élément de "flotter" dans la direction spécifiée. L'image de l'exemple ci-dessous est invitée à flotter vers la gauche, ce qui permet ensuite aux éléments suivant de "s'enrouler" autour. Vous pouvez demander à un élément de se superposer selon ces valeurs **left**, **right** ou **inherit**.

```
<article>

<p>Cette image flotte à gauche, ce qui signifie que ces paragraphes
s'enroulent autour d'elle.</p>
<p>Lorem ipsum... Ajouter suffisamment de texte par rapport à la taille de
l'image</p>.
</article>
```

```
img{
  float: left;
  width: 150px;
  height: 150px;
  margin-right: 1rem;
}
p + p{
  margin-top: 1rem;
}
```

Mise en page

Mise en page à plusieurs colonnes

Si vous avez une très longue liste d'éléments, comme une liste de tous les pays du monde, cela peut entraîner beaucoup de défilement et une perte de temps pour l'utilisateur. Cela peut également créer des espaces blancs superflus sur la page. Avec le CSS multicolonne, vous pouvez le diviser en plusieurs colonnes pour résoudre ces deux problèmes.

Cela divise automatiquement cette longue liste en deux colonnes et ajoute un écart entre les deux colonnes.

Au lieu de définir le nombre de colonnes dans lesquelles le contenu est divisé, vous pouvez également définir une largeur minimale souhaitée en utilisant la fonction `column-width` (largeur de colonne). Lorsque l'espace disponible dans la fenêtre de visualisation augmente, des colonnes supplémentaires sont automatiquement créées et lorsque l'espace diminue, les colonnes sont également réduites. Cette fonction est très utile dans les contextes de conception de sites web réactifs.

```
<h1>Les pays</h1>
<ul class="countries">
  <li>Argentine</li>
  <li>Albanie</li>
  <li>Algérie</li>
  <li>Andorre</li>
  ...
  <li>Zimbabwe</li>
</ul>
```

```
.countries{
  column-count: 2;
  column-gap: 1rem;
}
```

```
.countries{
  width: 100%;
  column-width: 260px;
  column-gap: 1rem;
}
```

Mise en page

Position

Dans les mécanismes de mise en page, la propriété **position** modifie le comportement d'un élément dans le flux normal du document et sa relation avec les autres éléments. Les options disponibles sont **relative**, **absolute**, **fixed** et **sticky**. La valeur par défaut est **static**.

```
.my-element{  
  position: relative;  
  top: 15px;  
}
```

Cet élément est déplacé de *15px* vers le bas en fonction de sa position actuelle dans le document, car il est positionné par rapport à lui-même. L'ajout de **position: relative** à un élément en fait également le bloc conteneur de tous les éléments enfants avec **position: absolute**.

Cela signifie que son enfant sera désormais repositionné vers cet élément particulier, plutôt que vers le parent relatif le plus élevé, lorsque **position: absolute** est appliqué sur une élément.

Mise en page

Lorsque vous définissez **position** sur **absolute**, l'élément sort du flux de document actuel. Cela implique deux choses:

1. Vous pouvez positionner cet élément où vous le souhaitez à l'aide de **top**, **right**, **bottom** et **left** dans son parent relatif le plus proche.
2. Tout le contenu entourant un élément absolu s'ajuste pour remplir l'espace restant laissé par cet élément.

Un élément dont la valeur **position** est définie sur **fixed** se comporte de la même manière que **absolute**, son parent étant l'élément `<html>` racine.

Les éléments fixes sont ancrés depuis le haut à gauche en fonction des valeurs **top**, **right**, **bottom** et **left** que vous définissez.

Vous pouvez obtenir à la fois qu'un élément soit absolu et fixe à partir d'un certain endroit de la page en utilisant **sticky**.

Avec cette valeur, lorsque la fenêtre d'affichage défile au-delà de l'élément, elle reste ancrée aux valeurs **top**, **right**, **bottom** et **left** que vous avez définies.

```
.my-element{  
  position: relative;  
  width: 100px;  
  height: 100px;  
}  
.second-element{  
  position: absolute;  
  top: 15px;  
  right: 0;  
  width: 50px;  
  height: 50px;  
}
```

Z-index et contexte d'empilement



Z-index et contexte d'empilement

Supposons que vous ayez quelques éléments qui sont absolument positionnés et qui sont censés être superposés les uns aux autres. Vous pouvez écrire un peu de code HTML comme ceci:

```
<div class="items">
  <div class="item-1">Élément 1</div>
  <div class="item-2">Élément 2</div>
  <div class="item-3">Élément 3</div>
</div>
```

Mais laquelle est superposée par défaut ? Pour savoir quel élément effectuera cette opération, vous devez comprendre le **z-index** et les contextes d'empilement.

Z-index

La propriété **z-index** définit explicitement l'ordre des calques pour le code HTML en fonction de l'espace 3D du navigateur, c'est-à-dire l'axe **Z**. La propriété z-index accepte une valeur numérique qui peut être un nombre positif ou négatif. Les éléments s'afficheront au-dessus d'un autre élément s'ils ont une valeur z-index plus élevée. *Si aucun z-index n'est défini sur vos éléments, le comportement par défaut est que l'ordre des sources des documents détermine l'axe Z.* Cela signifie que les éléments plus bas dans le document se trouvent au-dessus des éléments qui apparaissent avant eux.

Z-index et contexte d'empilement

```
.items > * {  
  width: 250px;  
  height: 200px;  
}  
.items > * + * {  
  margin-top: -150px;  
  opacity:.8;  
  box-shadow: 0 -1px 10px rgba(0 0 0 / 60%);  
}  
.items > :first-child {  
  background: lightblue;  
  border: 2px solid blue;  
}  
.items > :nth-child(2) {  
  background: pink;  
  border: 2px solid hotpink;  
}  
.items > :last-child {  
  background: yellow;  
  border: 2px solid gold;  
}
```

Dans le flux normal, si vous définissez une valeur spécifique pour z-index et que cela ne fonctionne pas, vous devez définir la valeur position de l'élément sur une valeur autre que **static**.

```
.items > * {  
  position: relative;  
}  
.items > :first-child {  
  ...  
  z-index: 3;  
}  
.items > :nth-child(2) {  
  ...  
  z-index: 1;  
}  
.items > :last-child {  
  ...  
  z-index: 2;  
}
```


Z-index et contexte d'empilement

Ce n'est toutefois pas le cas si vous vous trouvez dans un contexte Flexbox ou Grid, car vous pouvez modifier le z-index des éléments Flex ou Grid sans ajouter position: relative.

```
.items > * {  
  display: grid;  
  grid-template-columns: repeat(5, 100px);  
  max-width: 500px;  
}  
.items > * {  
  display: flex;  
  flex-direction: column;  
  align-items: flex-end;  
  justify-content: flex-end;  
  width: 250px;  
  height: 200px;  
}  
.items > :first-child {  
  ...  
  z-index: 3;  
  grid-column: 1/2;  
}
```

```
.items > :nth-child(2) {  
  ...  
  z-index: 1;  
  grid-column: 3/4;  
}  
.items > :last-child {  
  ...  
  z-index: 2;  
  grid-column: 4/5;  
}
```

Z-index et contexte d'empilement

Z-index négatif

Pour définir un élément derrière un autre élément, ajoutez une valeur négative pour z-index.

```
<div class="parent">
  <div class="child">
    Je suis derrière mon parent
  </div>
</div>
```

```
.parent {
  background: rgb(232 240 254 /.4);
}
.child{
  z-index: -1;
  position: relative;
}
```

Tant que **.parent** possède la valeur initiale de **z-index** à **auto**, l'élément **.child** se trouve en arrière-plan. Ajoutez le code CSS suivant à **.parent**, l'élément **.child** ne s'affichera plus derrière.

Étant donné que **.parent** comporte désormais une valeur **position** autre que **static** et une valeur **z-index** autre que **auto**, il a créé un contexte d'empilement.

Cela signifie que même si vous définissez **.child** sur une **z-index** de **-999**, il ne se trouvera pas derrière **.parent**.

```
.parent {
  z-index: 0;
  position: relative;
  ...
}
```

Z-index et contexte d'empilement

Contexte d'empilement

Un contexte d'empilement est un groupe d'éléments qui ont un parent commun et qui se déplacent ensemble vers le haut et le bas de l'axe Z.

```
<div class="top">
  <code>z-index : 1;</code>
  <div class="inner">z-index : 999;</div>
</div>
<div class="bottom">
  <code>z-index : 1;</code>
  <div class="inner">z-index : 999;</div>
</div>
```

Dans cet exemple, le premier élément parent a une z-index de 1. Il crée donc un contexte d'empilement. Son élément enfant a une z-index de 999.

À côté de ce parent, il y a un autre élément parent avec un enfant.

Le parent a une z-index de 2 et l'élément enfant a également une z-index de 2. Étant donné que les deux parents créent un contexte d'empilement, le z-index de tous les enfants est basé sur celui de leur parent

.Les z-index des éléments d'un contexte d'empilement sont toujours relatifs à l'ordre actuel du parent dans son propre contexte d'empilement.

```
.top, .bottom {
  width: 250px;
  padding: 80px 20px;
  position: relative;
}
.inner{
  padding: 20px;
  background: yellow;
  border: 2px solid gold;
}
.bottom {
  z-index: 2;
  margin-top: -90px;
  opacity: .8;
  background: pink;
  border: 2px solid hotpink;
}
.top {
  z-index: 1;
  background: lightblue;
  border: 2px solid blue;
}
.top > inner{
  z-index: 999;
}
.bottom > inner{
  z-index: 2;
}
```

Z-index et contexte d'empilement

Créer un contexte d'empilement

Vous n'avez pas besoin d'appliquer **z-index** et **position** pour créer un contexte d'empilement.

Vous pouvez créer un contexte d'empilement en ajoutant une valeur pour les propriétés qui créent une nouvelle couche composite telles que **opacity**, **will-change** et **transform**. Vous pouvez voir la liste complète des propriétés sur [MDN](#).

Pour expliquer ce qu'est une couche composite, imaginez qu'une page Web est un canevas.

Un navigateur utilise votre code HTML et CSS pour déterminer la taille de la toile. La page est ensuite peinte sur cette toile. Si un élément doit changer (par exemple, s'il change de position), le navigateur doit alors revenir en arrière et retravailler ce qu'il faut peindre.

Pour améliorer les performances, le navigateur crée des couches composites qui sont superposées au canevas. C'est un peu comme des notes adhésives : le déplacer et le modifier n'a pas un impact énorme sur la toile globale. Une nouvelle couche composite est créée pour les éléments avec certains éléments car ils sont très susceptibles de changer. Par conséquent, le navigateur s'assure que ces modifications sont performantes en utilisant le GPU pour appliquer les ajustements de style.

Flexbox



Flexbox

Un modèle de conception qui peut être délicat dans le responsive design est la barre latérale qui s'intègre à certains contenus. Là où il y a de la fenêtre d'affichage, ce modèle fonctionne très bien, mais lorsque l'espace est condensé, cette mise en page rigide peut devenir problématique.

```
<div class="page">
  <main class="content">
    <article>
      <p>Contenu principal, qui se place à côté.</p>
    </article>
  </main>
  <aside class="sidebar">
    <p>Contenu associé.</p>
  </aside>
</div>
```

```
.page{
  display: flex;
}
.page > *{
  padding: 1rem;
  min-height: 50vh;
}
.content{
  border: 1px solid gray;
}
.sidebar{
  background: gray;
  width: 15rem;
}
```

Flexbox

Le modèle de mise en page flexible (Flexbox) est un modèle de mise en page conçu pour du contenu unidimensionnel. Il excelle à prendre un ensemble d'éléments de différentes tailles et à renvoyer la meilleure mise en page pour ces éléments.

Il s'agit du modèle de mise en page idéal pour ce modèle de barre latérale. Flexbox permet non seulement d'aligner la barre latérale et le contenu de manière intégrée, mais également d'intégrer la barre latérale si l'espace disponible est insuffisant. Au lieu de définir des dimensions rigides que le navigateur doit respecter, avec Flexbox, vous pouvez indiquer des limites flexibles pour indiquer comment le contenu peut s'afficher.

```
<div class="page">
  <main class="content">
    <article>
      <p> Il s'agit d'un article de contenu qui se trouve en ligne
      avec une barre latérale. Redimensionnez le navigateur pour voir
      comment, lorsqu'il n'y a pas assez de place, la barre latérale se
      retrouve sur une nouvelle ligne.</p>
    </article>
  </main>
  <aside class="sidebar">
    <p>Contenu associé.</p>
  </aside>
</div>
```

```
.page{
  display: flex;
  flex-wrap: wrap;
}
.page > *{
  padding: 1rem;
  min-height: 50vh;
}
.content{
  flex-basis: 0;
  flex-grow: 2;
  min-width: 70%;
  border: 1px solid gray;
}
```

```
.sidebar{
  background: gray;
  flex-grow: 1;
  flex-basis: 12rem;
}
```

Flexbox

Que pouvez-vous faire avec une mise en page flexible ?

Les mises en page Flex offrent les fonctionnalités suivantes :

- Elles peuvent s'afficher sous la forme d'une ligne ou d'une colonne.
- Ils respectent le mode d'écriture du document.
- Il s'agit d'une seule ligne par défaut, mais il peut être demandé de s'encapsuler –“wrapper”–, sur plusieurs lignes.
- Les éléments de la mise en page peuvent être réorganisés visuellement, loin de leur ordre dans le DOM.
- L'espace peut être réparti à l'intérieur des éléments. Ceux-ci deviennent plus grands en fonction de l'espace disponible dans l'élément parent.
- L'espace peut être réparti autour des éléments et des lignes flexibles dans une mise en page encapsulée à l'aide des propriétés **flex-wrap**.
- Les éléments eux-mêmes peuvent être alignés sur l'axe transversal.

Flexbox

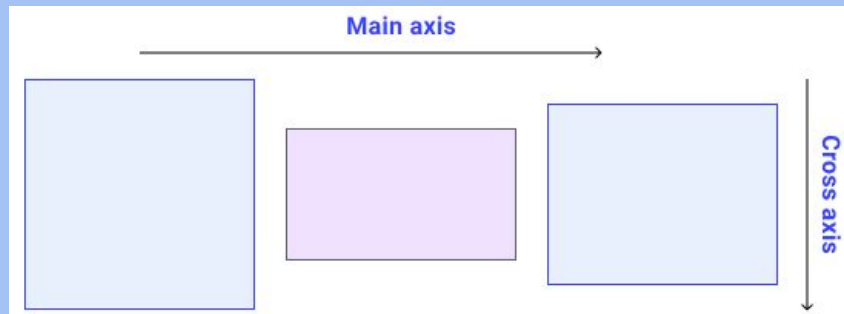
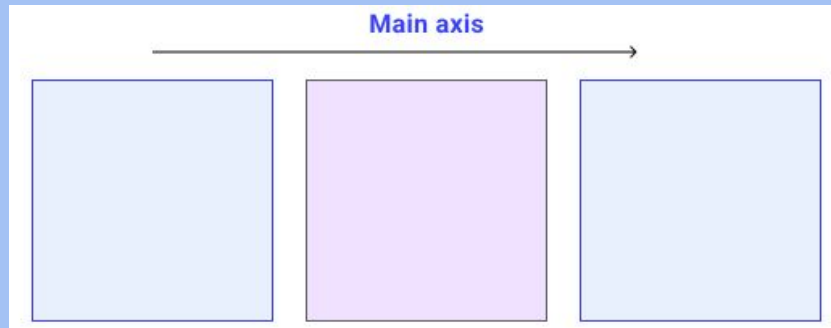
L'axe principal et l'axe transversal

Pour comprendre flexbox, il est essentiel de comprendre le concept d'axe principal et d'axe transversal.

L'axe principal est celui défini par votre propriété **flex-direction**. Si la valeur est **row**, votre axe principal se trouve le long de la ligne, et si la valeur est **column**, votre axe principal se trouve le long de la colonne.

Les éléments flex se déplacent en tant que groupe sur l'axe principal. N'oubliez pas: nous avons un tas d'éléments et nous essayons d'obtenir la meilleure mise en page pour ceux-ci en tant que groupe.

L'axe transversal s'étend dans l'autre sens par rapport à l'axe principal. Par conséquent, si **flex-direction** est défini sur **row**, l'axe transverse s'étend le long de la colonne.



Flexbox

Vous pouvez faire deux choses sur l'axe transversal. Vous pouvez déplacer les éléments individuellement ou de manière groupée afin qu'ils s'alignent les uns par rapport aux autres et au conteneur flexible. De plus, si vous avez des lignes flexibles encapsulées –"wrappées"–, vous pouvez les traiter comme un groupe pour contrôler la façon dont l'espace leur est attribué. Vous verrez comment tout cela fonctionne en pratique tout au long de ce guide. Pour l'instant, n'oubliez pas que l'axe principal suit votre flex-direction.

Créer un conteneur flexible

Voyons le comportement de Flexbox en prenant un groupe d'éléments de différentes tailles et en utilisant Flexbox pour les disposer.

```
<div class="container" id="container">  
  <div>Un</div>  
  <div>Deux</div>  
  <div>Trois</div>  
</div>
```

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

Flexbox

Les éléments enfants ont un comportement appliqué par défaut, du fait de leurs valeurs initiales :

- Les éléments s'affichent sur une ligne;
- Ils ne s'encapsulent pas, pas d'effet "wrap";
- Elles ne poussent pas pour remplir le conteneur;
- Ils sont alignés au début du conteneur.

Contrôler l'orientation des éléments

La propriété **flex-direction** fait que les éléments s'affichent sous forme de ligne, car la valeur initiale de **flex-direction** est **row**. Si vous voulez une ligne, vous n'avez pas besoin d'ajouter la propriété.

Pour changer de direction, ajoutez la propriété et l'une des quatre valeurs suivantes:

- **row**: les éléments sont disposés en ligne.
- **row-reverse**: les éléments sont disposés sur une ligne en partant de la fin du conteneur flexible.
- **column**: les éléments sont disposés sous forme de colonne.
- **column-reverse**: les éléments sont disposés sous la forme d'une colonne à partir de la fin du conteneur flexible.

Flexbox

Inverser l'affichage des éléments et l'accessibilité

Soyez prudent lorsque vous utilisez des propriétés qui éloignent l'affichage visuel de l'ordre des éléments dans le document HTML, car cela peut avoir un impact négatif sur l'accessibilité. Les valeurs **row-reverse** et **column-reverse** en sont un bon exemple. La réorganisation ne se produit que pour l'ordre visuel, et non l'ordre logique. Il est important de le comprendre, car l'ordre logique correspond à l'ordre dans lequel un lecteur d'écran lira le contenu à voix haute, et toute personne naviguant à l'aide du clavier suivra.

Modes et direction d'écriture

Par défaut, les éléments Flex sont présentés sous forme de ligne. Une ligne s'exécute dans le sens où les phrases circulent dans votre mode d'écriture et dans la direction du script. Cela signifie que si vous travaillez en arabe, qui se lit de droite à gauche (rtl), les éléments s'aligneront sur la droite. L'ordre de tabulation commencera également sur la droite, car c'est ainsi que les phrases sont lues en arabe.

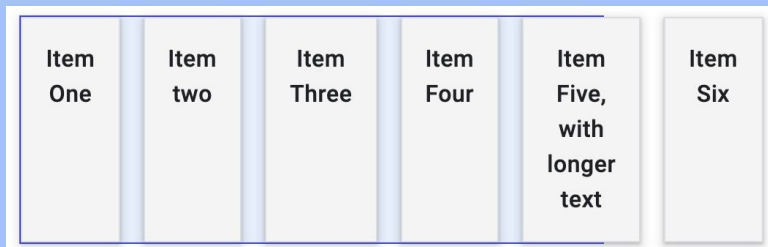
Si vous utilisez un mode d'écriture vertical, comme certaines polices de caractères japonaises, une ligne s'affichera verticalement, de haut en bas.

Par conséquent, le comportement par défaut des éléments Flex est lié au mode d'écriture du document. La plupart des tutoriels sont écrits en anglais ou dans un autre mode d'écriture horizontal, de gauche à droite. Ainsi, il est facile de supposer que les éléments Flex s'alignent sur la gauche et s'exécutent horizontalement.

Flexbox

Wrap et des éléments flexibles

La valeur initiale de la propriété **flex-wrap** est **nowrap**. Cela signifie que si l'espace dans le conteneur est insuffisant, les éléments dépasseront.



```
.container{  
  display: flex;  
  flex-wrap: wrap;  
}
```

Les éléments qui s'affichent à l'aide des valeurs initiales seront réduits le plus possible, jusqu'à atteindre la taille **min-content** avant un débordement.

Pour wrapper les éléments, ajoutez **flex-wrap: wrap** au conteneur flexible (voir ci-dessus).

Lorsqu'un conteneur flexible est wrappé, il crée plusieurs lignes flexibles. En termes de distribution de l'espace, chaque ligne agit comme un nouveau conteneur flexible. Par conséquent, si vous wrappez des lignes, il n'est pas possible d'aligner un élément de la ligne 2 sur un élément de la ligne 1 au-dessus. C'est ce que signifie "Flexbox" qui est unidimensionnel. Vous pouvez contrôler l'alignement sur un axe, une ligne ou une colonne, mais pas les deux ensemble, comme c'est le cas avec **grid**.

Flexbox

Le raccourci Flex-flow

Vous pouvez définir les propriétés flex-direction et flex-wrap à l'aide du raccourci flex-flow. Par exemple, pour définir flex-direction sur column et autoriser l'encapsulation des éléments:

```
.container{  
  display: flex;  
  flex-flow: column wrap;  
}
```

Contrôler l'espace à l'intérieur des éléments Flex

En supposant que notre conteneur dispose de plus d'espace que nécessaire pour afficher les éléments, ceux-ci s'alignent au début et ne s'agrandissent pas pour occuper l'espace. Elles cessent de croître au maximum de leur taille de contenu.

En effet, la valeur initiale des propriétés **flex-** est la suivante:

- **flex-grow: 0** : les éléments ne poussent pas.
- **flex-shrink: 1** : la taille des éléments peut être inférieure à celle de leur **flex-basis**.
- **flex-basis: auto** : la taille de base des éléments est auto.

Elle peut être représentée par la valeur du mot clé **flex: initial**. La propriété abrégée **flex** ou les termes longs **flex-grow**, **flex-shrink** et **flex-basis** sont appliqués aux enfants du conteneur flexible.

Flexbox

Pour faire augmenter les éléments, tout en permettant aux éléments volumineux de disposer de plus d'espace que les petits, utilisez **flex:auto**. Vous pouvez essayer en utilisant la démo ci-dessus. Les propriétés sont alors définies sur:

- **flex-grow: 1** : les éléments peuvent dépasser **flex-basis**.
- **flex-shrink: 1** : la taille des éléments peut être inférieure à leur valeur **flex-basis**.
- **flex-basis:auto** : la taille de base des éléments est **auto**.

L'utilisation de **flex: auto** signifie que les éléments ont des tailles différentes, car l'espace partagé entre les éléments est partagé après la disposition de chaque élément en tant que taille de contenu maximale. Ainsi, un grand élément occupera plus d'espace. Pour forcer tous les éléments à avoir une taille cohérente et ignorer la taille du contenu, remplacez **flex:auto** par **flex: 1** dans l'exemple.

Cela se décompose comme suit:

- **flex-grow: 1** : les éléments peuvent dépasser **flex-basis**.
- **flex-shrink: 1** : la taille des éléments peut être inférieure à leur valeur **flex-basis**.
- **flex-basis: 0** : la taille de base des éléments est **0**.

L'utilisation de **flex: 1** indique que tous les éléments ont une taille nulle. Par conséquent, tout l'espace du conteneur flexible peut être distribué. Comme tous les éléments ont un facteur **flex-grow** de **1**, ils croissent tous de la même manière et l'espace est partagé de manière égale.

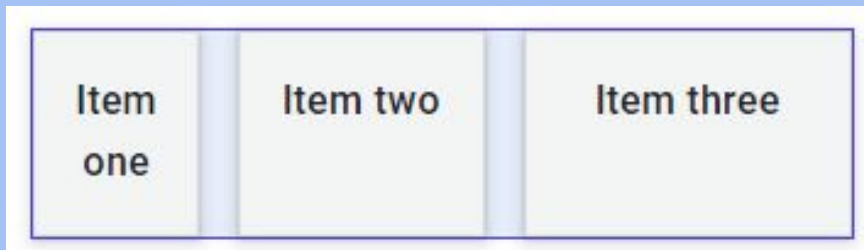
Flexbox

Permettre aux articles de se développer à des rythmes différents

Vous n'avez pas besoin d'attribuer un facteur **flex-grow** de **1** à tous les éléments. Vous pouvez attribuer différents facteurs **flex-grow** à vos éléments flex. Dans la démonstration ci-dessous, le premier élément contient **flex: 1**, le deuxième **flex: 2** et le troisième **flex: 3**.

A mesure que ces éléments passent de **0**, l'espace disponible dans le conteneur flexible est partagé en six éléments.

Une partie est attribuée au premier élément, deux parties au deuxième et trois parties au troisième.



```
<div class="container" id="container">
  <div class="box">Un</div>
  <div class="box">Deux</div>
  <div class="box">Trois</div>
</div>
```

```
.container{
  display: flex;
}
.box:nth-child(1){
  flex: 1
}
.box:nth-child(2){
  flex: 2
}
.box:nth-child(3){
  flex: 3
}
```


Flexbox

Réorganisation des éléments Flex

Vous pouvez réorganiser les éléments de votre conteneur Flex à l'aide de la propriété `order`. Cette propriété permet d'organiser les éléments dans des groupes ordinaux. Les éléments sont disposés dans la direction indiquée par `flex-direction`, en commençant par les valeurs les plus basses.

Si plusieurs éléments ont la même valeur, il sera affiché avec les autres éléments associés à cette valeur.

Présentation de l'alignement de Flexbox

Flexbox s'accompagne d'un ensemble de propriétés permettant d'aligner des éléments et de répartir l'espace entre eux.

Ces propriétés étaient si utiles qu'elles ont depuis été déplacées dans leur propre spécification que vous les rencontrerez également dans la mise en page de **grid**.

L'ensemble de propriétés peut être réparti dans deux groupes. Les propriétés de distribution de l'espace et les propriétés d'alignement.

```
.container{  
  display: flex;  
}  
.box:nth-child(1){  
  order: 1  
}  
.box:nth-child(2){  
  order: 2  
}  
.box:nth-child(3){  
  order: 3  
}
```

Flexbox

Les propriétés qui répartissent l'espace sont les suivantes:

- **justify-content** : répartition de l'espace sur l'axe principal.
- **align-content** : répartition de l'espace sur l'axe transversal.
- **place-content** : raccourci permettant de définir les deux propriétés ci-dessus.
-

Les propriétés utilisées pour l'alignement dans flexbox sont les suivantes:

- **align-self** : aligne un seul élément sur l'axe transversal.
- **align-items** : aligne tous les éléments en tant que groupe sur l'axe transversal.

Si vous travaillez sur l'axe principal, les propriétés commencent par justify-. L'axe transversal commence par align-.

Distribuer de l'espace sur l'axe principal

Avec le code HTML utilisé précédemment, les éléments flex disposés sous forme de ligne, il y a de l'espace sur l'axe principal. Les éléments ne sont pas assez grands pour remplir complètement le conteneur flexible. Les éléments sont alignés au début du conteneur flexible, car la valeur initiale de **justify-content** est **flex-start**. Les éléments sont alignés au début et tout espace supplémentaire se trouve à la fin.

Flexbox

Ajoutez la propriété **justify-content** au conteneur flex et attribuez-lui la valeur **flex-end**. Les éléments s'affichent à la fin du conteneur et l'espace libre est placé au début.

Vous pouvez également répartir l'espace entre les éléments avec **justify-content: space-between**. Il y'en a d'autre à consulter [MDN](#)..



Remarque : Pour que la propriété **justify-content** puisse agir, vous devez disposer d'espace libre dans votre conteneur sur l'axe principal.
Si vos éléments occupent l'axe, il n'y a pas d'espace à partager.
La propriété n'a donc aucun effet.

flex-direction: column

Si vous remplacez **flex-direction** par **column**, **justify-content** fonctionnera sur la colonne. Pour disposer d'espace libre dans votre conteneur lorsque vous travaillez en tant que colonne, vous devez attribuer à votre conteneur un **height** ou un **block-size**, sinon vous n'aurez pas d'espace libre pour le distribuer.

```
.container{  
  display: flex;  
  justify-content: end;  
}
```

Flexbox

Répartir l'espace entre les lignes flexibles

Avec un conteneur flexible, vous pouvez avoir de l'espace à répartir sur l'axe transversal. Dans ce cas, vous pouvez utiliser la propriété **align-content** avec les mêmes valeurs que **justify-content**. Contrairement à **justify-content**, qui aligne les éléments sur **flex-start** par défaut, la valeur initiale de **align-content** est **stretch**. Ajoutez la propriété **align-content** au conteneur flexible pour modifier ce comportement par défaut.

```
.container{  
  display: flex;  
  justify-content: end;  
  align-content: center;  
}
```

Le raccourci place-content

Pour définir à la fois **justify-content** et **align-content**, vous pouvez utiliser **place-content** avec une ou deux valeurs. Une seule valeur sera utilisée pour les deux axes, si vous spécifiez que la première sera utilisée pour **align-content** et la seconde pour **justify-content**.

```
.container{  
  display: flex;  
  place-content: end;  
}  
  
.container{  
  display: flex;  
  place-content: center end;  
}
```

Flexbox

Aligner des éléments sur l'axe transversal

Sur l'axe transversal, vous pouvez également aligner vos éléments dans la ligne flexible à l'aide de **align-items** et **align-self**.

L'espace disponible pour cet alignement dépend de la hauteur du conteneur flexible.

La valeur initiale de **align-self** est **stretch**. C'est pourquoi les éléments flexibles d'une ligne s'étirent par défaut jusqu'à la hauteur de l'élément le plus grand. Pour modifier cela, ajoutez la propriété **align-self** à tous vos éléments flex. Utilisez l'une des valeurs suivantes pour aligner l'élément:

- **flex-start**
- **flex-end**
- **center**
- **stretch**
- **baseline**

Consultez la liste complète des valeurs sur [MDN](https://developer.mozilla.org/fr/docs/Web/CSS/align-self).

```
.container{  
  display: flex;  
  justify-content: end;  
  align-content: center;  
}
```

```
.container{  
  display: flex;  
  place-content: end;  
}  
.container{  
  display: flex;  
  place-content: center end;  
}
```

Flexbox

Pourquoi n'y a-t-il pas de “justify-self” dans Flexbox ?

Les éléments Flex agissent comme un groupe sur l'axe principal. Il n'y a donc pas de concept de division d'un article individuel de ce groupe.

Dans la mise en page en grille, les propriétés **justify-self** et **justify-items** fonctionnent sur l'axe intégré pour aligner les éléments sur cet axe dans leur zone de grille. En raison de la façon dont les mises en page flex traitent les éléments comme un groupe, ces propriétés ne sont pas implémentées dans ce contexte.

Il est intéressant de noter que flexbox fonctionne très bien avec les marges automatiques. Si vous avez besoin de séparer un élément d'un groupe ou de séparer le groupe en deux groupes, vous pouvez appliquer une marge “auto”. La marge automatique absorbe tout l'espace dans la direction où elle est appliquée. Cela signifie qu'il pousse l'élément vers la droite, ce qui divise les groupes.

Centrer un élément verticalement et horizontalement

Les propriétés d'alignement peuvent être utilisées pour centrer un élément dans une autre zone. La propriété **justify-content** aligne l'élément sur l'axe principal, à savoir la ligne, la propriété **align-items** sur l'axe transversal.

Grid



Grid

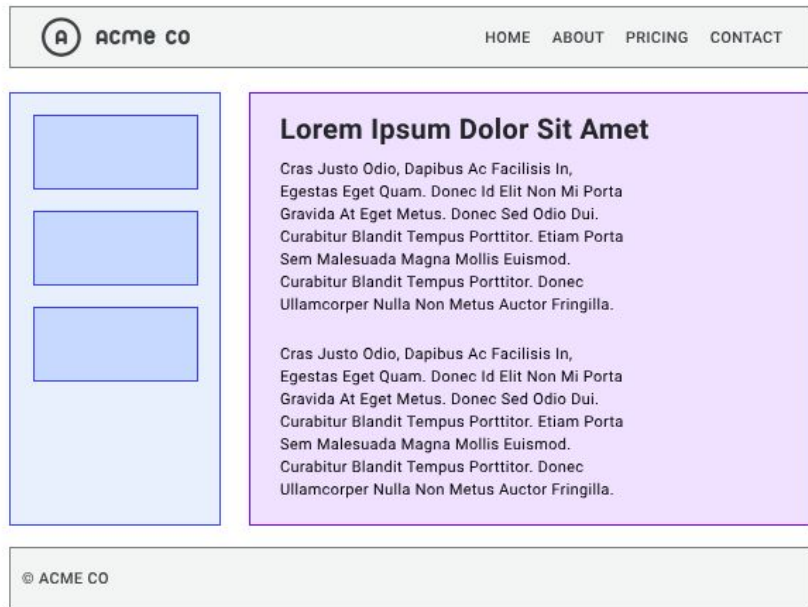
Une mise en page très courante en conception web est une mise en page d'en-tête, de barre latérale, de corps et de pied de page.

Avec la grille CSS, il est relativement simple de faire une mise page, tout en disposant de nombreuses options.

En effet, la grille est une méthode de mise en page conçue pour le contenu en deux dimensions. c'est-à-dire disposer les éléments en lignes et en colonnes en même temps.

Lors de la création d'une mise en page avec grid, vous définissez une grille avec des lignes et des colonnes. Vous placez ensuite les éléments sur cette grille, ou laissez le navigateur les placer automatiquement dans les cellules que vous avez créées.

Il y a beaucoup de choses à gérer avec grid, mais avec un aperçu de ce qui est disponible, vous créerez des mises en page en grille en un rien de temps.



Grid

Présentation

Alors, que pouvez-vous faire avec la grille ? Les mises en page en grille présentent les caractéristiques suivantes. Ce guide vous présente tous ces éléments.

- Une grille peut être définie avec des lignes et des colonnes. Vous pouvez choisir la taille de ces pistes de ligne et de colonne ou elles peuvent réagir à la taille du contenu.
- Les enfants directs du conteneur de la grille seront automatiquement placés sur cette grille.
- Vous pouvez également les placer à l'endroit qui vous convient le mieux.
- Les lignes et les zones de la grille peuvent être nommées pour faciliter le placement.
- L'espace libre dans le conteneur de la grille peut être réparti entre les pistes.
- Les éléments de la grille peuvent être alignés dans leur zone.

Terminologie de la grille

La grille comprend un tas de nouvelles terminologies, car c'est la première fois que CSS dispose d'un véritable système de mise en page.

Grid

Grille

Une grille est composée de lignes qui s'étendent horizontalement et verticalement. Si votre grille a quatre colonnes, elle comportera cinq lignes de colonnes, dont celle qui suit la dernière colonne.

Les lignes sont numérotées à partir de 1. La numérotation suit le mode d'écriture et l'orientation du script du composant. Cela signifie que la ligne de colonne 1 s'affichera à gauche dans une langue qui se lit de gauche à droite et à droite dans une langue (comme l'arabe) qui se lit de droite à gauche.

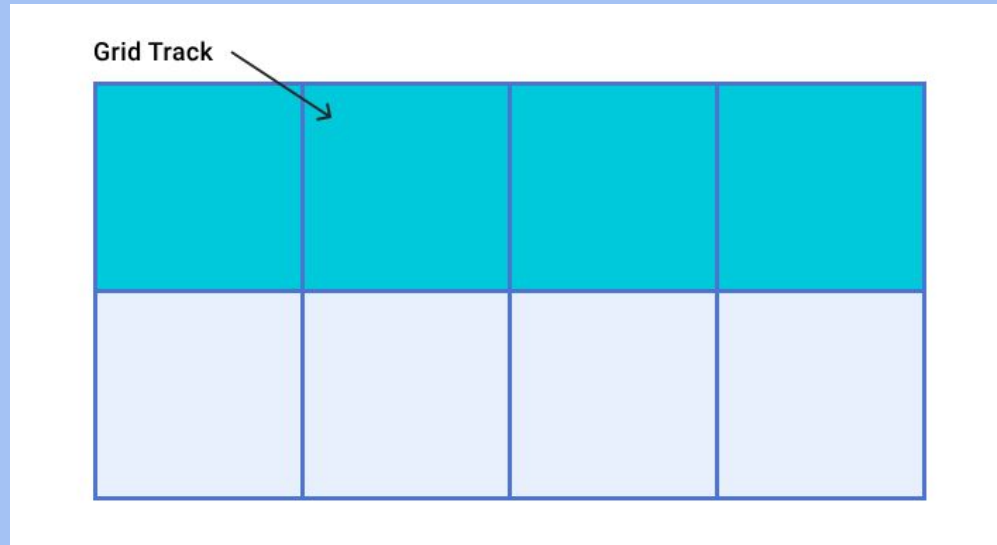
Grid Line



Grid

Pistes de grille

Une piste est l'espace entre deux lignes de la grille. Une piste de ligne est entre deux lignes de ligne et une piste de colonne entre deux lignes de colonne. Lorsque nous créons notre grille, nous créons ces pistes en leur attribuant une taille.



Grid

Cellule de grille

Une cellule de grille est le plus petit espace d'une grille défini par l'intersection des pistes de lignes et de colonnes. C'est comme une cellule de tableau ou une cellule d'une feuille de calcul. Si vous définissez une grille et ne placez aucun des éléments, ils seront automatiquement disposés en un élément dans chaque cellule de grille définie.

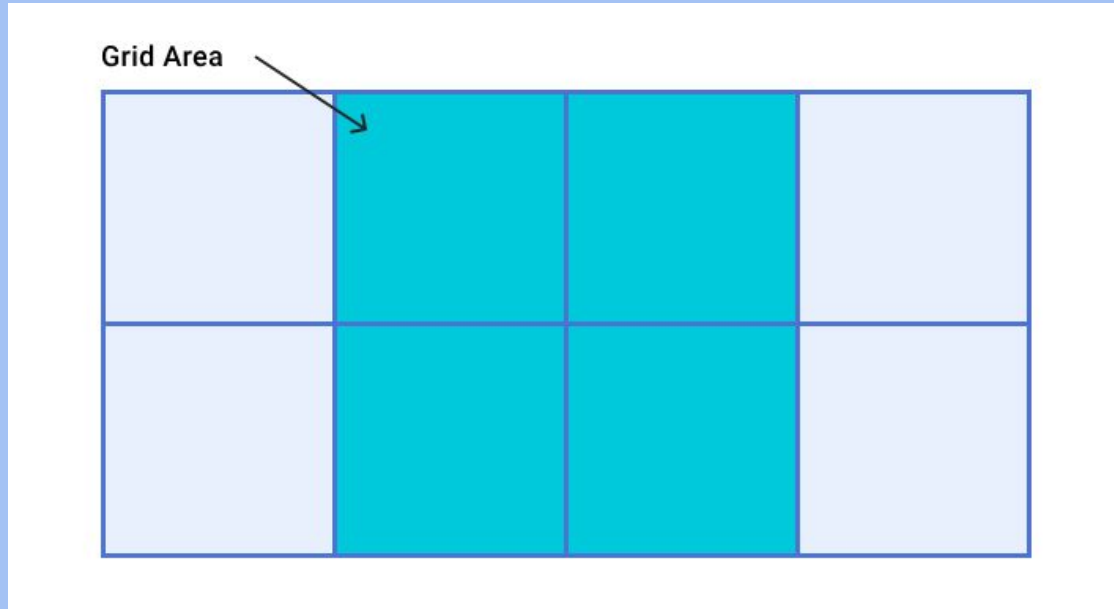
Grid Cell



Grid

Zone de la grille

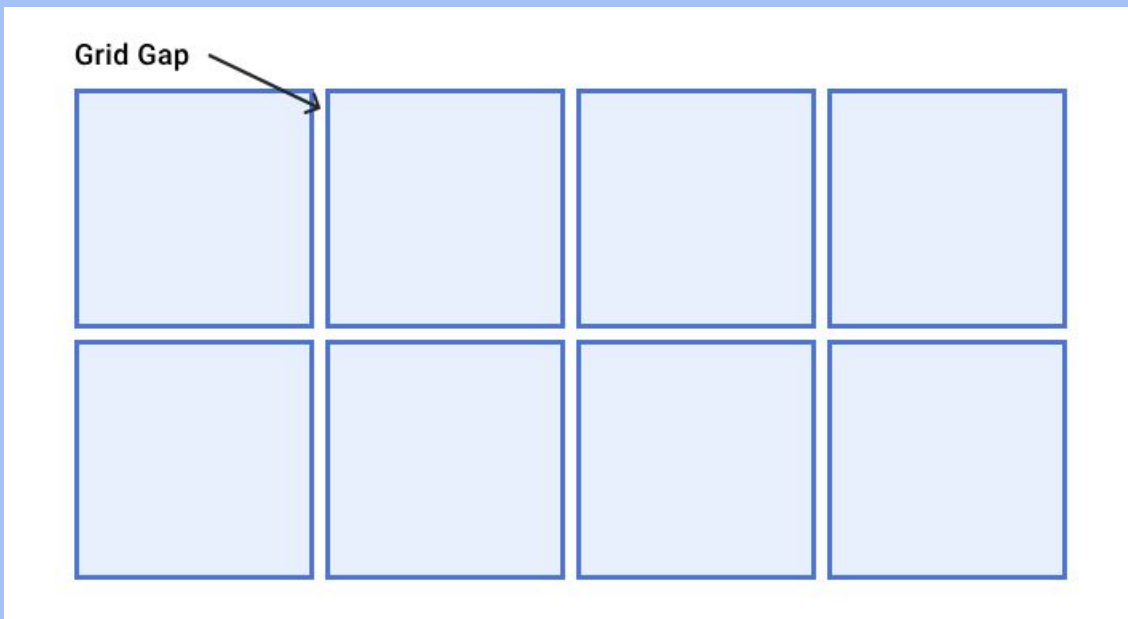
Plusieurs cellules de la grille. Les zones de grille sont créées en faisant en sorte qu'un élément s'étend sur plusieurs pistes.



Grid

Gouttière de la grille -gap-

Une gouttière est un espace entre les pistes. Pour des raisons de dimensionnement, ces espaces se comportent comme des pistes normales. Vous ne pouvez pas placer de contenu dans un espace, mais vous pouvez y faire passer des éléments de la grille.



Grid

Le conteneur “grid”

l'élément HTML auquel **display: grid** est appliqué, crée un contexte de mise en forme de grille pour les enfants directs.

Élément de grille

Un élément de grille est un élément enfant direct du conteneur de la grille.

Lignes et colonnes

Pour créer une grille de base, vous pouvez définir une grille avec trois pistes à colonnes, deux pistes de lignes et un espace de 10 pixels entre les pistes, comme suit.

Cette grille illustre bon nombre des éléments décrits dans la section Terminologie. Il a trois pistes à colonnes. Chaque piste utilise une unité de longueur différente. Elle comporte deux pistes de ligne, l'une utilisant une unité de longueur et l'autre en auto. Lorsqu'elle est utilisée comme un dimensionnement de piste automatique, on peut considérer qu'elle est aussi grande que le contenu. Par défaut, les pistes sont redimensionnées automatiquement.

```
.container{  
  display: grid;  
}
```

```
<div class="container">  
  <div class="item"></div>  
  <div class="item">Deux</div>  
  <div class="item">Trois</div>  
</div>
```

```
.container{  
  display: grid;  
  grid-template-columns: 5em 100px 30%;  
  grid-template-rows: 200px auto;  
  gap: 10px;  
}
```

Grid

La superposition de la grille dans les outils pour les développeurs Chrome peut vous aider à comprendre les différentes parties de la grille.

Dans Chrome, utilisez l'inspecteur et inspectez l'élément sur fond gris, avec l'ID container. Mettez la grille en surbrillance en sélectionnant l'élément correspondant dans le DOM, à côté de l'élément **.container**.

Dans l'onglet "Mise en page", sélectionnez Afficher les numéros des lignes dans le menu déroulant pour afficher les numéros de ligne sur votre grille.

Learn CSS - A simple grid

A simple grid

In this example the grid container creates a grid with three column tracks and two row tracks.

Item one, Item two, Item three, Item four, Item five

Elements, Console, Sources, Network

```
<html>
  <head>
  </head>
  <body translate="no">
    <div class="wrapper">
      <div class="flow">
        <h1>A simple grid</h1>
        <figure class="callout">
          <div class="container" id="container">
            <div class="box">Item one</div>
            <div class="box">Item two</div>
            <div class="box">Item three</div>
            <div class="box">Item four</div>
            <div class="box">Item five</div>
          </div>
        </figure>
      </div>
    </div>
  </body>
</html>
```

result-iframe, html, body, main, div.wrapper, article.flow, div#container.container

Styles, Computed, Layout, Event Listeners, DOM Breakpoints, Properties

Grid

Overlay display settings

Show line numbers

Show track sizes

Show area names

Extend grid lines

Grid overlays

button.UserMenu_userMenuButton-2eiYO

div#container.container

Flexbox

Flexbox overlays

Grid

Mots clés associés à la taille intrinsèque

Outre les dimensions de longueur et de pourcentage décrites dans la section sur les unités de dimensionnement, les pistes de grille peuvent utiliser des mots clés de dimensionnement intrinsèque. Ces mots clés sont définis dans la spécification "box sizing" et ajoutent d'autres méthodes de dimensionnement en CSS, et pas seulement les pistes de la grille.

- **min-content**
- **max-content**
- **fit-content()**

Le mot clé **min-content** permet de réduire au maximum la taille d'une piste afin que son contenu ne dépasse pas. Si vous modifiez l'exemple de mise en page de la grille pour qu'elle comporte trois pistes à trois colonnes toutes à la taille de **min-content**, elles deviendront aussi étroites que le mot le plus long de la piste.

Le mot clé **max-content** a l'effet inverse. La piste devient suffisamment large pour que tout le contenu puisse s'afficher en une seule longue chaîne ininterrompue. Cela peut entraîner des dépassements, car la chaîne ne sera pas encapsulée (pas de "wrap").

Grid

La fonction **fit-content()** agit tout d'abord comme **max-content**. Une fois que la piste atteint la taille que vous transmettez à la fonction, ici **10em**, le contenu commence à être encapsulé, "wrap". Ainsi, **fit-content(10em)** créera une piste inférieure à **10em**, si la taille maximale du contenu est inférieure à **10em**, mais jamais supérieure à 10em.

Dans la démonstration suivante, testez les différents mots clés de dimensionnement intrinsèque en modifiant le dimensionnement des pistes de la grille en modifiant la valeur auto par **min-content**.

```
<div class="container">
  <div class="item"></div>
  <div class="item">Deux</div>
  <div class="item">
    Trois avec un peu plus de contenu
  </div>
  <div class="item">Quatre</div>
  <div class="item">Cinq</div>
</div>
```

```
.container{
  display: grid;
  grid-template-columns: repeat( 3, auto);
  grid-template-rows: 200px auto;
  gap: 10px;
  padding: 1rem;
  max-width: 30rem;
}
```

Grid

L'unité fr

Nous utilisons les variables de longueur, les pourcentages, ainsi que ces nouveaux mots clés. Il existe également une méthode de dimensionnement spéciale qui ne fonctionne qu'avec une mise en page en grille. Il s'agit de l'unité **fr**, une longueur flexible décrivant une partie de l'espace disponible dans le conteneur de la grille.

L'unité **fr** fonctionne de la même manière que l'unité **flex: auto** dans flexbox. Elle répartit de l'espace après la disposition des éléments. Par conséquent, nous avons trois colonnes qui reçoivent toutes la même part d'espace disponible:

```
.container{  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr;  
}
```

Comme l'unité **fr** partage l'espace disponible, elle peut être combinée à un écart de taille fixe ou à des pistes de taille fixe. Pour avoir un composant avec un élément de taille fixe et que la deuxième piste occupe l'espace restant, vous pouvez l'utiliser comme liste de suivi de **grid-template-columns: 200px 1fr**.

Utiliser différentes valeurs pour l'unité **fr** partagera l'espace en proportion. Des valeurs plus élevées obtiennent plus d'espace libre. Dans l'exemple, modifiez la valeur de la troisième piste.

Grid

La fonction `minmax()`

Cette fonction vous permet de définir des tailles minimale et maximale pour les éléments. Cela peut être très utile. Si nous prenons l'exemple de l'unité **fr** ci-dessus qui répartit l'espace restant, il pourrait être écrit en utilisant **`minmax()`** sous la forme **`minmax(auto, 1fr)`**. La grille examine la taille intrinsèque du contenu, puis distribue l'espace disponible après avoir laissé suffisamment d'espace au contenu. Cela signifie que vous n'obtiendrez peut-être pas de pistes qui ont chacune une part égale de l'espace disponible dans le conteneur de la grille.

Pour forcer une piste à occuper une part égale de l'espace dans le conteneur de la grille moins les espaces, utilisez **`minmax`**. Remplacez **`1fr`** comme taille de piste par **`minmax(0, 1fr)`**.

La taille minimale de la piste est alors définie sur 0, et non la taille minimale du contenu.

La grille prend ensuite toute la taille disponible dans le conteneur, déduit la taille nécessaire pour les écarts et partage le reste en fonction de vos unités **fr**.

Notation `repeat()`

Vous pouvez utiliser la fonction **`repeat()`** pour répéter n'importe quelle section de la liste des pistes. Vous pouvez, par exemple, répéter un certain nombre de pistes.

Grid

Si vous souhaitez créer une grille de suivi de 12 colonnes avec des colonnes égales, vous pouvez utiliser le CSS ci-contre à l'aide de **repeat()**, plutôt que de répéter 12 fois **minmax(0,1fr)**

```
.container{  
  display: grid;  
  grid-template-columns: repeat(12, minmax(0,1fr));  
}
```

Vous pouvez également avoir des pistes normales et une section qui se répète.

```
.container{  
  display: grid;  
  grid-template-columns: 200px repeat(2, 1fr 2fr) 200px; /* Soit 6 pistes */  
}
```

auto-fill et auto-fit

Vous pouvez combiner tout ce que vous avez appris sur le dimensionnement des pistes, **minmax()** et les répétitions pour créer un modèle utile avec une mise en page en grille.

Peut-être ne souhaitez-vous pas spécifier le nombre de pistes de colonnes, mais plutôt en créer autant que possible pour votre conteneur. Pour ce faire, utilisez **repeat()** et les mots clés **auto-fill** ou **auto-fit**.

Grid

```
.container{  
  display: grid;  
  grid-template-columns: repeat( auto-fill, 200px );  
}
```

L'exemple inclut autant de pistes que nécessaire.

Toutefois, les canaux ne sont pas flexibles. Vous obtiendrez un blanc à la fin jusqu'à ce que vous ayez assez d'espace pour une autre piste de 200 pixels.

Si vous ajoutez la fonction **minmax()**, vous pouvez demander autant de pistes que possible, avec une taille minimale de **200 pixels** et une taille maximale de **1fr**. La grille présente ensuite les pistes de **200 pixels** et l'espace restant leur est réparti de manière égale.

Vous obtenez ainsi un format responsif en deux dimensions sans avoir à effectuer de requêtes média.

Il existe une légère différence entre **auto-fill** et **auto-fit**. Dans la démonstration suivante, vous allez utiliser une mise en page en grille en utilisant la syntaxe expliquée ci-dessus, mais avec seulement deux éléments de grille dans le conteneur de la grille. À l'aide du mot clé **auto-fill**, vous pouvez constater que des canaux vides ont été créés.

Remplacez le mot clé par **auto-fit** pour que la taille des canaux soit réduite à **0**. Cela signifie que les canaux flexibles s'agrandissent pour occuper plus d'espace.

Grid

Emplacement automatique

Jusqu'à présent, vous avez vu le placement automatique sur la grille à l'œuvre dans les exemples. Les éléments sont placés sur la grille un par cellule, dans l'ordre dans lequel ils apparaissent dans la source. C'est tout ce dont vous avez besoin pour de nombreuses mises en page. Si vous avez besoin de plus de contrôle, plusieurs options s'offrent à vous. La première consiste à ajuster la mise en page de l'emplacement automatique.

Placer des éléments dans des colonnes

Le comportement par défaut de la mise en page en grille consiste à placer les éléments le long des lignes. Vous pouvez placer les éléments dans des colonnes à l'aide de **grid-auto-flow: column**. Vous devez définir des pistes de ligne, sinon les éléments créeront des pistes de colonnes intrinsèques et les disposeront sur une longue ligne.

Ces valeurs se rapportent au mode d'écriture du document. Une ligne s'exécute toujours dans la direction d'exécution d'une phrase en mode d'écriture du document ou du composant. Dans la démonstration suivante, vous pouvez modifier le mode de la valeur de **grid-auto-flow** et de la propriété **writing-mode**.

Grid

```
.container{
  writing-mode: horizontal-tb;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: repeat( 2, minmax(100px,auto));
  grid-auto-flow: row;
  gap: 10px;
}
```

Vous pouvez utiliser les valeur suivantes pour **writing-mode** :

- **horizontal-tb** : défaut, pour les scripts **RTL**, le contenu circule sera de droite à gauche. La ligne horizontale suivante est positionnée sous la ligne précédente.
- **vertical-rl** : le contenu se positionne de haut en bas et la ligne verticale suivante est positionnée à gauche de la ligne précédente. Pour les scripts RTL, le contenu circule verticalement de bas en haut et la ligne verticale suivante est positionnée à droite de la ligne précédente.
- **vertical-lr** : le contenu s'affiche de haut en bas et la ligne verticale suivante est positionnée à droite de la ligne précédente.

Vous pouvez utiliser les valeur suivantes pour **grid-auto-flow** :

- **row** : le contenu s'affiche en ligne les uns derrières les autres,
- **column** : Le contenu s'affiche en colonne d'abord

Grid

Sur plusieurs pistes

Vous pouvez faire en sorte que tout ou partie des éléments d'une mise en page placée automatiquement s'étendent sur plusieurs pistes. Pour **grid-column-end** ou **grid-row-end**, utilisez le mot clé **span** et le nombre de lignes où le contenu peut s'étendre.

```
.item{  
  grid-column-end: span 2; /* S'étendra sur deux lignes et couvrira donc deux pistes */  
}
```

Comme **grid-column-start** n'est pas spécifié, cette propriété utilise la valeur initiale de **auto** et est placée selon les règles de placement automatique. Vous pouvez également spécifier la même chose à l'aide du raccourci **grid-column**:

```
.item{  
  grid-column: auto / span 2;  
}
```

Grid

Comblent les “espaces”

Une mise en page placée automatiquement avec certains éléments couvrant plusieurs pistes peut entraîner l'apparition d'une grille avec quelques cellules non remplies. Le comportement par défaut de la mise en page sous forme de grille avec une mise en page entièrement placée automatiquement consiste à toujours progresser vers l'avant. Les éléments seront placés dans l'ordre dans lequel ils se trouvent dans la source ou en fonction de toute modification effectuée avec la propriété **order**. S'il n'y a pas assez d'espace pour accueillir un élément, la grille laisse un espace vide et passe à la piste suivante.

L'exemple suivant illustre ce comportement. Ajoutez la valeur **dense** à **grid-auto-flow**. Une fois cette valeur en place, la grille récupère les éléments suivant dans la mise en page et les utilise pour combler les espaces vides. Cela peut signifier que l'affichage se déconnecte de l'ordre logique.

```
.container{
  writing-mode: horizontal-tb;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: repeat(2,minmax(100px,auto));
  grid-auto-flow: row dense;
  gap: 10px;
}
```

Grid

Place les éléments

Vous disposez déjà de nombreuses fonctionnalités de la grille CSS. Voyons maintenant comment positionner les éléments sur la grille que nous avons créée.

La mise en page sous forme de grille CSS est basée sur une grille de lignes numérotées. Le moyen le plus simple de placer des éléments sur la grille consiste à les placer d'une ligne à une autre. Dans ce guide, vous découvrirez d'autres façons de placer des éléments, mais vous aurez toujours accès à ces lignes numérotées.

Les propriétés que vous pouvez utiliser pour placer les articles par numéro de ligne sont les suivantes:

- **grid-column-start**
- **grid-column-end**
- **grid-row-start**
- **grid-row-end**

Ils comportent des raccourcis qui vous permettent de définir simultanément les lignes de début et de fin:

- **grid-column**
- **grid-row**

Grid

Pour placer votre élément, définissez les lignes de début et de fin de la zone de la grille dans laquelle il doit être placé.

```
.container{
  display: grid;
  grid-template-columns: repeat(4,1fr);
  grid-template-rows: repeat(2, 200px 100px));
  grid-auto-flow: row dense;
  gap: 10px;
}
.item{
  grid-column-start: 1; /* Commence à la colonne ligne 1 */
  grid-column-end: 4; /* Fini à la colonne ligne 4 */
  grid-row-start: 2; /* Commence à la colonne ligne 2 */
  grid-row-end: 4; /* Fini à la colonne ligne 4 */
}
```

Grid

Empiler les éléments

Le positionnement basé sur la ligne vous permet de placer des éléments dans la même cellule de la grille. Cela signifie que vous pouvez empiler des éléments ou faire en sorte qu'un élément en chevauche partiellement un autre. Les éléments venant plus tard dans la source s'affichent au-dessus des éléments placés plus tard dans la source. Vous pouvez modifier cet ordre d'empilement en utilisant **z-index**, comme pour les éléments positionnés.

Numéros de ligne négatifs

Lorsque vous créez une grille à l'aide de **grid-template-rows** et **grid-template-columns**, vous créez ce qu'on appelle la grille explicite. Il s'agit d'une grille que vous avez définie et que vous avez donnée aux pistes. Parfois, vous aurez des éléments qui s'affichent en dehors de cette grille explicite. Par exemple, vous pouvez définir des pistes de colonnes, puis ajouter plusieurs lignes d'éléments de grille sans jamais définir de pistes de lignes. Par défaut, les pistes sont redimensionnées automatiquement. Vous pouvez également placer un élément à l'aide de **grid-column-end** qui se trouve en dehors de la grille explicite définie.

Dans les deux cas, la grille crée des pistes pour que la mise en page fonctionne.

Ces pistes sont appelées grille implicite.

Grid

La plupart du temps, cela ne fera aucune différence si vous travaillez avec une grille implicite ou explicite. Toutefois, les emplacements basés sur les lignes peuvent faire la principale différence entre les deux.

En utilisant des numéros de ligne négatifs, vous pouvez placer des éléments à partir de la ligne de fin de la grille explicite. Cela peut être utile si vous souhaitez qu'un élément s'étend de la première à la dernière ligne de colonne. Dans ce cas, vous pouvez utiliser **grid-column: 1 / -1**. L'élément s'étire directement sur la grille explicite.

Toutefois, cela ne fonctionne que pour la grille explicite. Utilisez une mise en page de trois lignes d'éléments placés automatiquement, où vous souhaitez que le tout premier élément s'étend jusqu'à la ligne de fin de la grille. Cela ne fonctionnera pas car les pistes sont créées dans la grille implicite. Il n'y a aucun moyen d'atteindre la fin de la grille implicite avec -1



Grid

Dimensionnement des pistes implicites

Les pistes créées dans la grille implicite sont redimensionnées automatiquement par défaut. Toutefois, si vous souhaitez contrôler le dimensionnement des lignes, utilisez la propriété **grid-auto-rows** et, pour les colonnes, **grid-auto-columns**.

Pour créer toutes les lignes implicites avec une taille minimale de 10em et une taille maximale de auto:

```
.container{  
  display: grid;  
  grid-auto-rows: min(10em,auto);  
}
```

Pour créer des colonnes implicites avec un format de pistes de 100 et 200 pixels. Dans ce cas, la première colonne implicite sera de 100 pixels, la deuxième de 200 pixels, la troisième colonne de 100 pixels, etc.

```
.container{  
  display: grid;  
  grid-auto-columns:100px 200px;  
}
```

Grid

Lignes de la grille nommées

Il peut être plus facile de placer des éléments dans une mise en page si les lignes ont un nom plutôt qu'un nombre. Vous pouvez nommer n'importe quelle ligne de votre grille en ajoutant le nom de votre choix entre crochets. Plusieurs noms peuvent être ajoutés, séparés par un espace entre les mêmes crochets. Une fois que vous avez nommé des lignes, vous pouvez les utiliser à la place des nombres.

```
.container{
  display: grid;
  grid-template-columns: [main-start aside-start] 1fr
                        [aside-end content-start] 2fr
                        [content-end main-end];
}
.sidebar{
  grid-column: aside-start / aside-end /* Entre les lignes 1 et 2 */
}
.footer{
  grid-column: main-start / main-end /* De la ligne 1 à la ligne 3 */
}
```


Grid

Zones du modèle de grille

Vous pouvez également nommer des zones de la grille et placer des éléments sur ces zones nommées. Cette technique est très pratique, car elle vous permet de voir à quoi ressemble votre composant directement dans le code CSS.

Pour commencer, attribuez un nom aux enfants directs de votre conteneur de grille à l'aide de la propriété `grid-area`:

```
header{  
  grid-area : header;  
}  
.sidebar{  
  grid-area : sidebar;  
}  
.content{  
  grid-area : content;  
}  
footer{  
  grid-area : footer;  
}
```

Grid

Le nom peut être autre que les mots clés **auto** et **span**. Une fois tous vos éléments nommés, utilisez la propriété **grid-template-areas** pour définir les cellules de grille que chaque élément couvrira. Chaque ligne est définie entre guillemets.

```
.container{
  display: grid;
  grid-template-columns: repeat(4, 1fr );
  grid-template-areas: "header header header header"
                      "sidebar content content content"
                      "sidebar footer footer footer"
}
```

Voici quelques règles lorsque vous utilisez **grid-template-areas**.

- La valeur doit correspondre à une grille complète sans cellules vides.
- Pour s'étendre sur plusieurs pistes, répétez le nom.
- Les zones créées en répétant le nom doivent être rectangulaires et ne peuvent pas être déconnectées.

Si vous enfreignez l'une des règles ci-dessus, la valeur est considérée comme non valide et supprimée. Pour laisser des espaces blancs sur la grille, utilisez un ou plusieurs `.` sans espace blanc entre eux.

Par exemple, pour laisser la toute première cellule de la grille vide, je pourrais ajouter une série de `“.”`:

Grid

Comme l'intégralité de votre mise en page est définie à un seul endroit, vous pouvez la redéfinir facilement à l'aide de requêtes média.

Dans l'exemple suivant, j'ai créé une mise en page à deux colonnes qui passe à trois colonnes en redéfinissant la valeur de **grid-template-columns** et **grid-template-areas**.

Ouvrez l'exemple dans une nouvelle fenêtre pour tester la taille de la fenêtre d'affichage et voir le changement de mise en page.

Grid

Propriétés abrégées

La propriété **grid-template** est un raccourci pour **grid-template-rows**, **grid-template-columns** et **grid-template-areas**. Les lignes sont définies en premier, avec la valeur de **grid-template-areas**. Le dimensionnement des colonnes est ajouté après /.

```
.container{  
  display: grid;  
  grid-template:  
    "header header header" minmax(150px, auto )  
    "sidebar content content" auto  
    "sidebar footer footer" auto / 1fr;  
}
```

Propriété grid

Le raccourci **grid** peut être utilisé exactement de la même manière que le raccourci **grid-template**. Lorsqu'elle est utilisée de cette manière, elle rétablit les valeurs initiales des autres propriétés de grille acceptées. L'ensemble complet étant:

- grid-template-rows
- grid-template-columns
- grid-template-areas
- grid-auto-rows
- grid-auto-columns
- grid-auto-flow

Vous pouvez également utiliser ce raccourci pour définir le comportement de la grille implicite, par exemple :

```
.container{  
  display: grid;  
  grid: repeat(2, 80px ) / auto-flow 120px  
}
```

Grid

Alignement

La mise en page sous forme de grille utilise les mêmes propriétés d'alignement que celles que vous avez apprises dans le guide de flexbox. Dans la grille, les propriétés qui commencent par `justify-` sont toujours utilisées sur l'axe intégré, c'est-à-dire la direction dans laquelle les phrases s'exécutent dans votre mode d'écriture.

Les propriétés commençant par `align-` sont utilisées sur l'axe de bloc, c'est-à-dire la direction dans laquelle les blocs sont disposés en mode d'écriture.

- **`justify-content`** et **`align-content`** : distribuent de l'espace supplémentaire dans le conteneur de la grille autour ou entre les pistes.
- **`justify-self`** et **`align-self`** : sont appliqués à un élément de grille pour le déplacer à l'intérieur de la zone de grille dans laquelle il est placé.
- **`justify-items`** et **`align-items`** : sont appliqués au conteneur de la grille pour définir toutes les propriétés **`justify-self`** des éléments.

Grid

Distribuer de l'espace supplémentaire

Dans ce exemple, la grille est plus grande que l'espace nécessaire pour disposer les pistes à largeur fixe. Cela signifie que nous disposons d'un espace à la fois pour les dimensions inline et les dimensions de bloc de la grille. Essayez différentes valeurs pour **align-content** et **justify-content** pour observer le comportement des titres.

```
.container{
  display: grid;
  align-content: stretch;
  justify-content: stretch;
  grid-template-column: auto auto auto
  grid-template-rows: 90px 100px
}
.item:nth-child(1){
  grid-row: 1 / 3;
  grid-column: 1;
}
.item:nth-child(3){
  grid-row: 2;
  grid-column: 2 / -1;
}
```

Grid

Notez que les écarts deviennent plus importants lorsque vous utilisez des valeurs telles que **space-between**, et que tout élément de grille couvrant deux pistes s'agrandit également pour absorber l'espace supplémentaire ajouté à l'écart.



Remarque : Comme pour Flexbox, ces propriétés ne fonctionnent que s'il existe de l'espace supplémentaire à répartir. Si vos pistes de la grille remplissent soigneusement le conteneur, il n'y aura pas d'espace à partager.

Déplacer du contenu

Les éléments avec une couleur d'arrière-plan semblent remplir entièrement la zone de la grille dans laquelle ils sont placés, car la valeur initiale de **justify-self** et **align-self** est **stretch**.



Remarque : Si votre élément est une image ou un autre élément présentant un format intrinsèque, la valeur initiale sera **start** au lieu de **stretch** afin d'éviter d'étirer l'élément.

Dans l'exemple, modifiez les valeurs de **justify-items** et **align-items** pour voir comment cela modifie la mise en page. La taille de la zone de la grille ne change pas, mais les éléments sont déplacés dans la zone définie.

Propriétés logiques



Propriétés logiques

Introduction

Dans ce exemple, nous avons un libellé de texte avec une icône intégrée.

```
<p>
  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"
    aria-hidden="true" focusable="false">
    <path
      d="M19,5h-2V3H7v2H5C3.9,5,3,5.9,3,7v1c0,2.55,1.92,4.63,4.39,4.94c0.63,1.
      5,1.98,2.63,3.61,2.96V19H7v2h10v-2h-4v-3.1
      c1.63-0.33,2.98-1.46,3.61-2.96C19.08,12.63,21,10.55,21,8V7C21,5.9,20.1,5
      ,19,5z M5,8V7h2v3.82C5.84,10.4,5,9.3,5,8z M12,14
      c-1.65,0-3-1.35-3-3V5h6v6C15,12.65,13.65,14,12,14z
      M19,8c0,1.3-0.84,2.4-2,2.82V7h2V8z" />
    </svg>
    <span>CSS c'est chouette</span>
  </p>
```

```
p{
  display: inline-flex;
  align-items: center;
  font-weight: bold;
  font-size: 1.5em;
  padding: 0.5em;
  background: #f3f3f3;
  border: 1px solid #DADCE0;
}

p svg{
  width: 1.2em;
  height: 1.2em;
  margin-right: 0.5em;
  flex: none;
  fill: #3740FF;
}
```



CSS c'est chouette

Propriétés logiques

L'icône se trouve à gauche du texte, avec un petit espace entre les deux, fourni par `margin-right` sur l'icône. Il y a toutefois un problème, car cela ne fonctionne que lorsque l'orientation du texte est de gauche à droite. Si le texte est lu de droite à gauche (comme c'est le cas pour l'arabe), l'icône sera positionnée contre le texte.

```
<p dir="rtl">
  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"
  aria-hidden="true" focusable="false">
    <path
d="M19,5h-2V3H7v2H5C3.9,5,3,5.9,3,7v1c0,2.55,1.92,4.63,4.39,4.94c0.63,1.5,1.98
,2.63,3.61,2.96V19H7v2h10v-2h-4v-3.1
c1.63-0.33,2.98-1.46,3.61-2.96C19.08,12.63,21,10.55,21,8V7C21,5.9,20.1,5,19,5z
M5,8V7h2v3.82C5.84,10.4,5,9.3,5,8z M12,14
c-1.65,0-3-1.35-3-3V5h6v6C15,12.65,13.65,14,12,14z
M19,8c0,1.3-0.84,2.4-2,2.82V7h2V8z" />
    </svg>
    <span>CSS رائے!</span>
  </p>
```

```
p{
  display: inline-flex;
  align-items: center;
  font-weight: bold;
  font-size: 1.5em;
  padding: 0.5em;
  background: #f3f3f3;
  border: 1px solid #DADCE0;
}

p svg{
  width: 1.2em;
  height: 1.2em;
  margin-right: 0.5em;
  flex: none;
  fill: #3740FF;
  border: 1px solid red;
}
```

CSS جيد حقًا 

Propriétés logiques

Comment en tenir compte dans les CSS ? Les propriétés logiques vous permettent de résoudre ces situations. Ils offrent, entre autres avantages, une assistance automatique et sans frais pour l'internationalisation. Elles vous aident à créer un frontal plus résilient et inclusif.

Terminologie

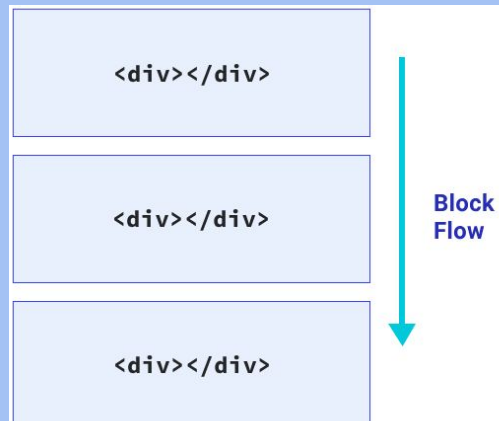
Les propriétés physiques en haut, à droite, en bas et à gauche font référence aux dimensions physiques de la fenêtre d'affichage. Vous pouvez les considérer comme une rose des vents sur une carte.

En revanche, les propriétés logiques font référence aux bords d'une boîte en relation avec le flux de contenu. Par conséquent, ils peuvent changer si l'orientation du texte ou le mode d'écriture change. 1

Il s'agit d'un grand changement par rapport aux styles directionnels, et cela nous offre beaucoup plus de flexibilité pour habiller nos interfaces.

Flux "block"

Le flux "block" correspond à la direction dans laquelle les blocs de contenu sont placés. Par exemple, s'il y a deux paragraphes, le flux "block" est l'endroit où le deuxième paragraphe ira. Dans un document en français, le flux "block" est de haut en bas. Pensez-y dans le contexte de paragraphes de texte qui se suivent, de haut en bas.



Propriétés logiques

Flux “inline”

Le flux “inline” correspond à la circulation du texte dans une phrase. Dans un document en français, le flux inline s’affiche de gauche à droite. Si vous modifiez la langue du document de votre page Web en arabe (<html lang="ar">), le flux inline s’affichera de droite à gauche.



Le texte se positionne dans la direction déterminée par le mode d’écriture du document. Vous pouvez modifier l’orientation du texte à l’aide de la propriété **writing-mode**. Il peut s’appliquer à l’ensemble du document ou à des éléments individuels.

Propriétés logiques

```
<article>
<p>Cette ligne à la valeur
<code>horizontal-tb</code> qui est le valeur
par défaut de <code>writing-mode</code>
</p>
<p>Cette ligne à la valeur
<code>vertical-rl</code> se lit de droite à
gauche et du haut vers le bas.
</p>
<p>Cette ligne à la valeur
<code>vertical-lr</code> se lit de gauche à
droite et dE haut en bas
</p>
</article>
```

Cette ligne à la
valeur
`horizontal-tb`
qui est le valeur
par défaut de
`writing-mode`

Cette ligne à la valeur
`vertical-rl` — se lit
de droite à gauche et
du haut vers le bas.

Cette ligne à la valeur
`vertical-lr` se lit de
gauche à droite et dE
haut en bas

```
p:nth-child(2){
  writing-mode: vertical-rl;
}
p:nth-child(3){
  writing-mode: vertical-lr;
  padding-block: 1em;
}
p {
  max-inline-size: 20ch;
}
article {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 1rem;
}
article > * {
  padding: 0.5em;
  border: 1px solid #DADCE0;
}
```

Propriétés logiques

Flux relatif

Auparavant, dans les CSS, nous ne pouvions appliquer que des propriétés telles que la marge en fonction de la direction de leurs côtés. Par exemple,

margin-top est appliqué à la partie supérieure physique de l'élément. Avec les propriétés logiques, **margin-top** devient **margin-block-start**.

Cela signifie que, quelle que soit la langue et l'orientation du texte, le flux de blocs dispose de règles de marge appropriées.

```
<div class="sample">
  <div>
    <p>J'ai comme valeur <code>margin-top</code>.</p>
  </div>
  <div>
    <p>J'ai comme valeur <code>margin-block-start</code>.</p>
  </div>
</div>
```

J'ai comme valeur `margin-top`.

J'ai comme valeur `margin-block-start`.

```
.sample {
  writing-mode: horizontal-tb;
}
.sample > * {
  border: 1px solid blue;
  background: lightblue;
  max-width: 20rem;
}
.sample > * + * {
  margin-block-start: 1em;
}
.sample p {
  padding: 1em;
  background: white;
}
.sample div:nth-child(1) p {
  margin-top: 1em;
}
.sample div:nth-child(2) p {
  margin-block-start: 1em;
}
```

Propriétés logiques

Taille

Pour éviter qu'un élément ne dépasse une certaine largeur ou hauteur, écrivez une règle comme celle-ci:

Les équivalents relatifs au flux sont **max-inline-size** et **max-block-size**.

Vous pouvez également utiliser **min-block-size** et **min-inline-size** au lieu de **min-height** et **min-width**.

Avec les propriétés logiques, cette règle de largeur et de hauteur maximales ressemblerait à ceci:

```
.my-element {  
  max-width: 150px;  
  max-height: 100px;  
}
```

```
.my-element {  
  max-inline-size: 150px;  
  max-block-size: 100px;  
}
```

Début et fin

Au lieu d'utiliser des directions comme le haut, la droite, le bas et la gauche, utilisez le début et la fin. Vous obtenez ainsi un bloc **"block-start"**, **"inline-end"**, **"block-end"** et **"inline-start"**. Elles vous permettent d'appliquer des propriétés CSS qui réagissent aux changements du mode d'écriture.

Par exemple, pour aligner le texte à droite, vous pouvez utiliser le code CSS suivant:

Avec les valeurs logiques, il existe des valeurs **start** et **end** qui correspondent à l'orientation du texte.

```
p{  
  text-align: right;  
}
```

```
p{  
  text-align: end;  
}
```

Propriétés logiques

Espacement et positionnement

Les propriétés logiques de `margin`, `padding` et `inset` facilitent le positionnement des éléments, et permettent de déterminer la manière dont ils interagissent les uns avec les autres sur les différents modes d'écriture. Les propriétés liées à la marge et à la marge intérieure sont toujours des mappages directs avec les directions, mais la principale différence est que lorsqu'un mode d'écriture change, ils changent en même temps.

Cela permet d'ajouter un espace intérieur vertical avec `padding` et de le repousser de la gauche avec `margin`. La propriété `top` la déplace également vers le bas. Avec des équivalents de propriété logiques, cela ressemblerait plutôt à ceci :

Cela permet d'intégrer l'espace dans l'espace avec `padding` et de le repousser à partir du début de la ligne avec `margin`. La propriété `inset-block` la déplace vers l'intérieur à partir du début du bloc.

La propriété `inset-block` n'est pas la seule option abrégée avec des propriétés logiques. Cette règle peut être condensée à l'aide de versions abrégées des propriétés de marge et de marge intérieure

```
.my-element {  
  padding-top: 2em;  
  padding-bottom: 2em;  
  margin-left: 2em;  
  position: relative;  
  top: 0.2em;  
}
```

```
.my-element {  
  padding-block-start: 2em;  
  padding-block-end: 2em;  
  margin-inline-start: 2em;  
  position: relative;  
  inset-block-start: 0.2em;  
}
```

```
.my-element {  
  padding-block: 2em;  
  margin-inline: 2em 0;  
  position: relative;  
  inset-block: 0.2em;  
}
```


Propriétés logiques

Bordures

Vous pouvez également ajouter `border` et `border-radius` à l'aide de propriétés logiques. Pour ajouter une bordure en bas et à droite, avec un rayon vers la droite, vous pouvez écrire une règle comme celle-ci:

Vous pouvez également utiliser des propriétés logiques comme celles-ci:
Le **end-end** dans **`border-end-end-radius`** est la fin du block et la fin inline.

```
.my-element {  
  border-bottom: 1px solid red;  
  border-right: 1px solid red;  
  border-bottom-right-radius: 1em;  
}
```

Unités

Les propriétés logiques apportent deux nouvelles unités: **vi** et **vb**.

Une unité **vi** correspond à 1% de la taille de la fenêtre d'affichage dans le sens de l'intégration. L'équivalent de propriété non logique est **vw**.

L'unité **vb** correspond à 1% de la fenêtre d'affichage dans le sens du bloc. L'équivalent de propriété non logique est **vh**.

Ces unités seront toujours mappées avec la direction de lecture.

Par exemple, si vous souhaitez qu'un élément occupe 80% de l'espace intégré disponible d'une fenêtre d'affichage, l'utilisation de l'unité **vi** définit automatiquement cette taille de haut en bas si le mode d'écriture est vertical.

```
.my-element {  
  border-block-end: 1px solid red;  
  border-inline-end: 1px solid red;  
  border-end-end-radius: 1em;  
}
```

Propriétés logiques

Utiliser les propriétés logiques de manière pragmatique

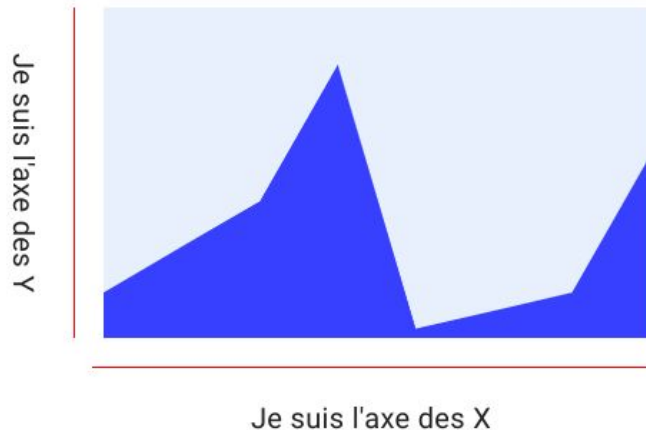
Les propriétés logiques et les modes d'écriture ne sont pas réservés à l'internationalisation. Vous pouvez les utiliser pour créer une interface utilisateur plus polyvalente.

Si vous disposez d'un graphique comportant des étiquettes sur l'axe X et l'axe Y, vous pouvez souhaiter que le texte de l'étiquette Y soit lu verticalement.

Étant donné que le **writing-mode** de l'étiquette de l'axe Y de la démonstration est **vertical-rl**, vous pouvez utiliser les mêmes valeurs **margin** sur les deux étiquettes.

La valeur **margin-block-start** s'applique aux deux étiquettes, car le début du bloc se trouve à droite pour l'axe Y et en haut pour l'axe X. Les côtés de début de bloc ont une bordure rouge pour vous permettre de les voir.

Mon graphique



Propriétés logiques

```
<h1>Mon graphique</h1>
<div class="container">
  <p class="label">Je suis l'axe des Y</p>
  <div class="chart">
    <svg width="100%" height="100%" viewBox="0 0 100 50">
      <polygon points="0,50 0.00,41.67 14.29,33.33
28.57,25.00 42.86,0.00 57.14,48.33 71.43,45.00 85.71,41.67
100.00,16.67 100,50" class="line"></polygon>
    </svg>
  </div>
  <p class="label">Je suis l'axe des X</p>
</div>
```

```
.container {
  display: flex;
  flex-wrap: wrap;
  max-width: 30rem;}
h1 {
  border-bottom: 1px solid gray;
  margin-bottom: 2rem;
  min-width: 20rem;}
```

```
p.label {
  margin-block-start: 1em;
  padding-block-start: 1em;
  margin-block-end: 0;
  text-align: center;
  border-block-start: 1px solid red;
}
p:first-of-type {
  writing-mode: vertical-rl;
}
p:last-of-type {
  flex-basis: calc(100% - 3.2em);
  margin-inline-start: auto;
}
.chart {
  flex: auto;
  background: lightblue;
  padding-block-start: 2em;
}
svg {
  display: block;
  fill: blue;
}
```

Propriétés logiques

Résoudre le problème d'icône

Maintenant que nous avons abordé les propriétés logiques, ces connaissances peuvent être appliquées au problème de conception avec lequel nous avons commencé. La marge a été appliquée à droite de l'élément de l'icône. Pour que l'espacement entre l'icône et le texte soit pris en charge dans tous les sens de lecture, la propriété **margin-inline-end** doit être utilisée à la place.

Maintenant, quelle que soit la direction de lecture, l'icône se positionnera et s'espacera de manière appropriée.

```
.sample{  
  direction: ltr  
}  
p svg{  
  width: 1.2em;  
  height: 1.2em;  
  margin-right: 0.5em;  
  flex: none;  
  fill: #3740FF;  
  border: 1px solid red;  
}
```



CSS c'est chouette

```
.sample{  
  direction: rtl  
}  
p svg{  
  width: 1.2em;  
  height: 1.2em;  
  margin-inline-end: 0.5em;  
  flex: none;  
  fill: #3740FF;  
  border: 1px solid red;  
}
```



CSS c'est chouette