

Supports de cours Web

HTML & CSS

Adrien Cater
adrien.cater@eduvaud.ch

Web

Adrien Cater – adrien.cater@eduvaud.ch

Hello world.

Aperçu

- HTML
 - syntaxe, vocabulaire – balises, attributs
 - structure
- CSS
 - syntaxe, vocabulaire – propriétés, valeurs, unités
 - sélecteurs
 - axes inline et block, box model
 - mise en page – flexbox
 - responsive, CSS media queries
 - webfonts
 - couleur
- Web
 - fonctionnement du web – domaines, URL, serveur, protocoles, standards
 - site: structure, navigation, arborescence
 - navigateurs – moteurs de rendu
- Média
 - images, vidéo – formats, tailles et optimisation
- Process
 - esquisses, croquis, maquettes, wireframes, découpe
 - gestion des fichiers, organisation, arborescence
- Outils
 - VS Code – <https://code.visualstudio.com/> – éditeur, IDE
 - Web inspecteur   
 - Git & Github – <https://github.com/>

Help!

- MDN – <https://developer.mozilla.org/fr/>
- CSS Tricks – <https://css-tricks.com/>
- Cours web – <https://cours-web.ch/>
- Viscom-CIE1 – <https://inetis-ch.github.io/viscom-cie1/>

HTML & CSS

HTML *Hyper Text Markup Language*

Contenu, Description, Organisation, Sémantique, Structure, Arborescence

CSS *Cascading Style Sheets*

Design, Présentation, Visuel, Style

Start

- Texte brut / plain text / UTF-8 / précédemment: ASCII...
 - Open source / standards
- Éditeur texte / Text Editor
 - Text Edit (Apple) /Applications/
- IDE – Integrated development environment / Environnement de développement
 - Visual Studio Code code.visualstudio.com
 - BBEdit [bbedit.com](https://www.bbedit.com)
 - Espresso espressoapp.com
 - Nova nova.app
 - Sublime text sublimetext.com
- Préférences & configuration
 - Indentation (style d'indentation: "tabs -vs- spaces" / tabulations ou espaces)
 - Application d'ouverture par défaut pour HTML et CSS
 - Information → Ouvrir avec: → Tout modifier...
- Raccourcis clavier
 - command
 - option, alt
 - control, ctrl
 - shift, majuscule
 - tab
 - # 3
 - () 8 9
 - { } 8 9
 - [] 5 6
 - navigation
 - " 2
 - " 2 " 2
 - ' !" ' !"
 - « ; » ;
 -
 - tiret demi-cadratin (en) – tiret cadratin (em) –

Gestion des fichiers

Les dossiers et fichiers sont structurés dans une arborescence.
Gérez vos fichiers de manière logique et systématique.
La structure qui vous paraît logique et intuitive est la meilleure.

L'indication du chemin et les sous-dossiers sont habituellement écrits avec un slash /.

Retrouver votre dossier 'home' dans /Users/votrenom/

Créer un dossier ~/Documents/**Eracom**/

Créer des dossiers pour des cours, des projets, des clients...
.../Eracom/S2/coursweb/

Pro: Ajoutez des **raccourcis** aux dossiers actuels ou souvent utilisés
dans la **barre latérale** du Finder.

Essentiel pour le webdev, mais une bonne habitude générale: créer des noms de fichiers et
dossiers sans accents, ni espaces, ni ponctuation sauf tiret-au-milieu et sous_tiret.

a-z A-Z 0-9 _ -

Selon besoin, préfix des fichiers/dossiers: préfixe avec **AAAA-MM-JJ_**

Apprivoiser VSCode

- Créer d'abord le dossier de votre site/projet, au bon endroit.
- Dans VSCode, toujours créer une nouvelle fenêtre "New Window"
et ouvrez le dossier "Open Folder".
- Ne pas déplacer un projet ouvert dans VSCode

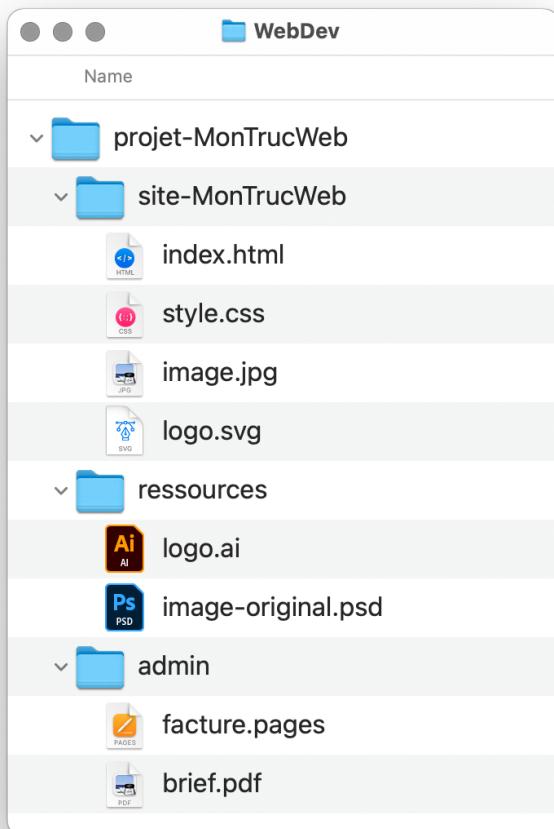
Projets web

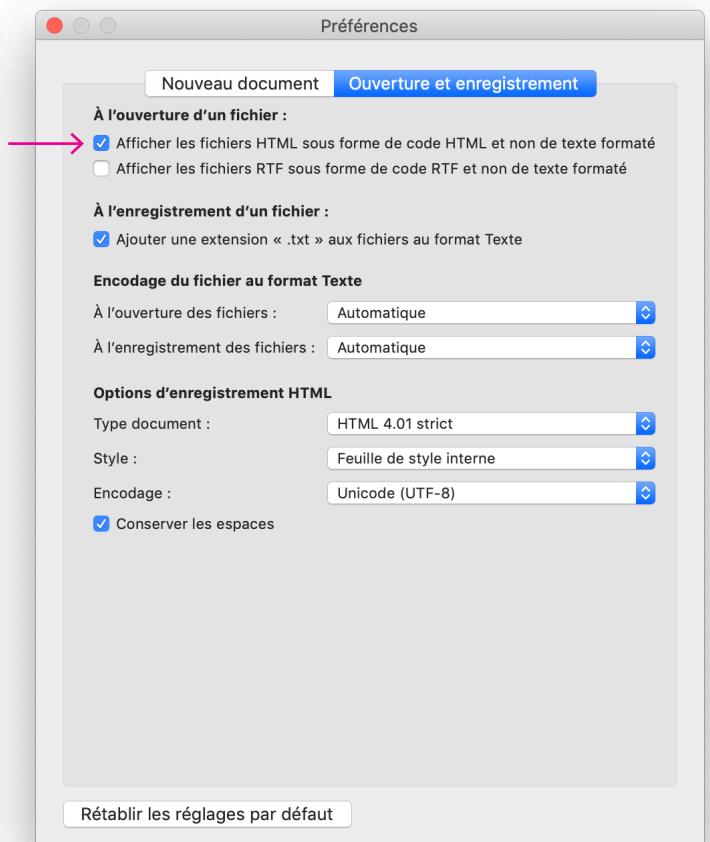
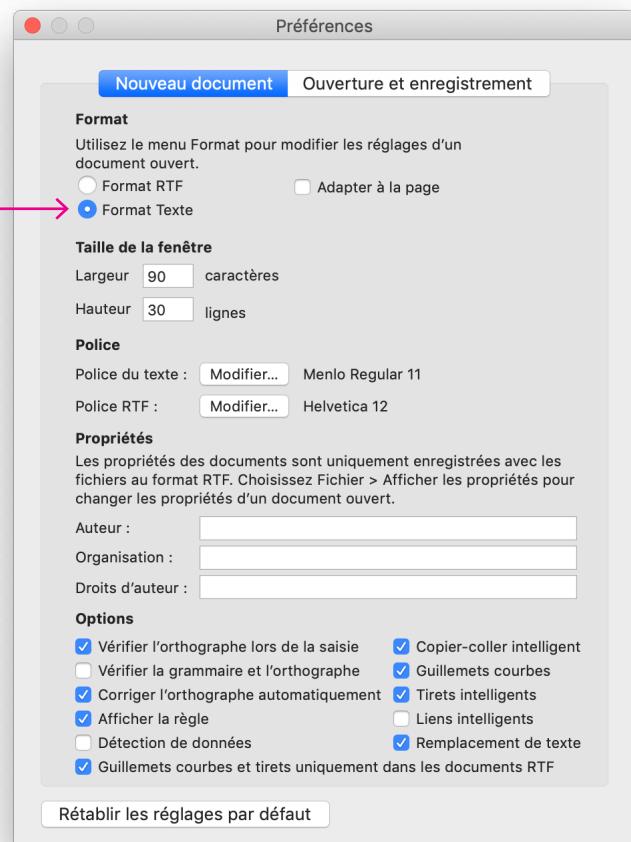
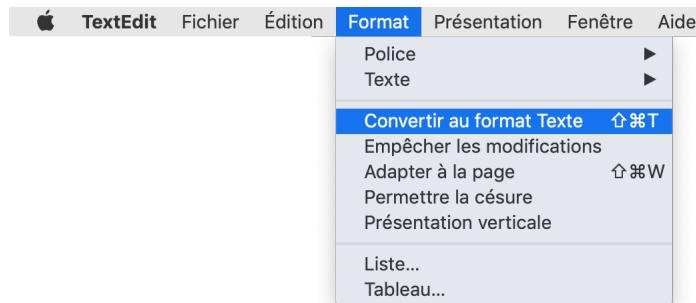
Les sites web (et toutes les ressources) doivent être regroupés dans un dossier, avec le fichier **index.html** à la racine.

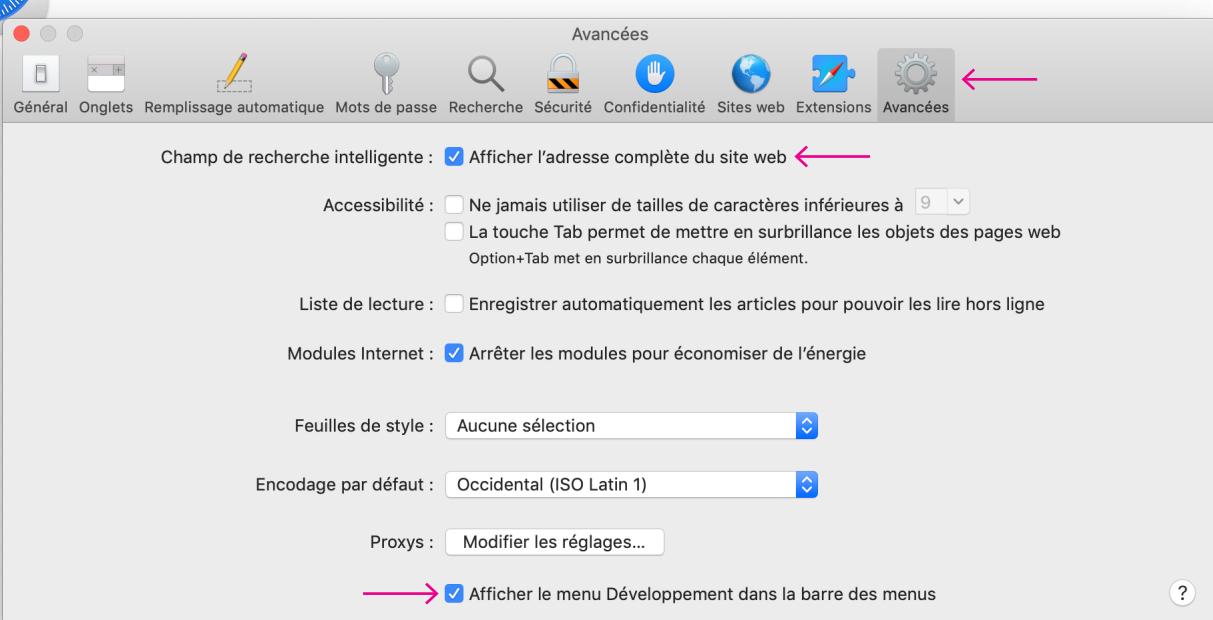
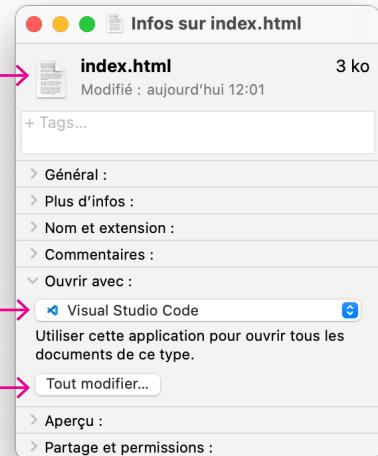
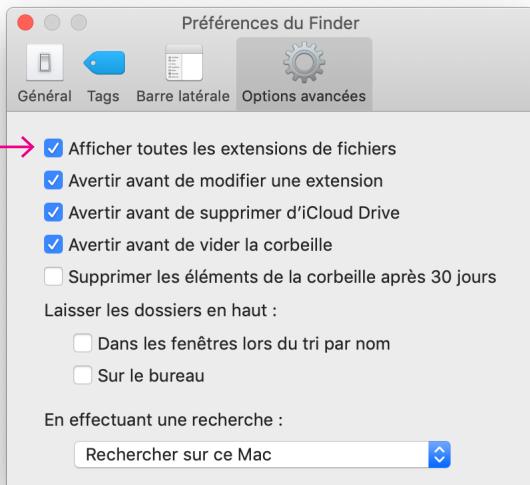
Le dossier d'un site web doit inclure *toutes* les ressources pour le site web (images, etc.), mais *que* les fichiers utilisés dans le site.

Les autres fichiers doivent se trouver dans un autre dossier ailleurs.

Un dossier projet 'parent' est recommandé pour des projets plus complexes.







Visual Studio Code – VSCode

code.visualstudio.com

Settings:  

Show all whitespace:

editor.renderWhitespace: *all*

Editor: Render Whitespace

Controls how the editor should render whitespace characters.



Tabs not spaces

editor.insertSpaces : *false*

Editor: Insert Spaces (Also modified elsewhere)

Insert spaces when pressing **Tab**. This setting is overridden based on the file contents when [Editor: Detect Indentation](#) is on.

editor.detectIndentation : *true*

Editor: Detect Indentation

Controls whether [Editor: Tab Size](#) and [Editor: Insert Spaces](#) will be automatically detected when a file is opened based on the file contents.

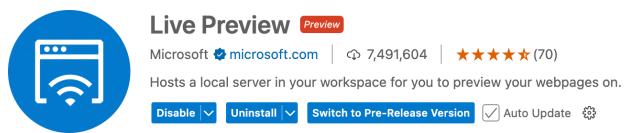
Disable popup hover tooltips

editor.hover.enabled : *false*

Editor > Hover: Enabled

Controls whether the hover is shown.

Extension: "Live Preview"



Aussi: "Live Server" (plus puissant, mais moins convenable)

Évitez la confusion et perte des fichiers:

- Créer d'abord le dossier de votre site/projet, au bon endroit.
- Dans VSCode, toujours créer une nouvelle fenêtre "New Window" et ouvrez le dossier "Open Folder".
- Ne pas déplacer un projet ouvert dans VSCode

Docs: code.visualstudio.com/docs/

Settings: afficher les réglages modifiés: recherche pour @modified

Autocomplete: pas besoin d'écrire <, simplement le tag. Aussi: div.machin

Raccourcis clavier:

Ajouter ou transformer en commentaire: ⌘ ⌘ ⌘ 7

Texte en ligne (compact) ou retour à ligne en bloc – Menu: View: Word Wrap: ⌘ Z

Sélection mots identiques – Menu: Selection: Add Next Occurrence: ⌘ D

Multiples sélections: ⌘ click

Sélection d'une ligne entière: ⌘ click ×3

Couper (= copier et supprimer) – Menu: Edit: Cut: ⌘ X

Navigation à travers le texte par le clavier:

Navigation: ⌞ ⌛ ⌈ ⌉ ⌈ ⌉ ⌈ ⌉

Passez entre les lignes, fin de la ligne, début de la ligne suivante...

Navigation par mot – option: ⌘ ⌞ ⌛

Navigation avec sélection - majuscule: ⌈ ⌉ ⌈ ⌉ ⌈ ⌉

Navigation et sélection par mot: ⌈ ⌞ ⌛ ⌉

HTML

Element
élément

```
<p class="intro"> Bonjour & hello! </p>
```

Attribute
attribut

Tag (opening)
balise d'ouverture

Value
valeur

Content
contenu

Entity
entité

(closing) Tag
balise de fermeture

<!-- Comment, commentaire -->

CSS

Rule
règle

```
p.intro { font-size: 16px; }
```

Selector
sélecteur

Property
propriété

Declaration
déclaration

Value
valeur

Unit
unité

Block
bloc

/* Comment, commentaire */

CSS

Séparation du contenu et de la forme.

Le HTML est un langage qui décrit des *contenus* et leur *structure*,
le CSS est un langage dédié au *graphisme*, à la *présentation*.

Le CSS contient des indications à propos d'un document HTML.

Le CSS est une langue déclarative; on exprime une volonté au navigateur de comment on souhaite afficher la page: "il faut que cet élément soit ainsi, et ces autres éléments soient comme ça".

Évolution constante: de nouvelles features sont adoptées par les navigateurs en continu. Votre page web d'hier fonctionnera demain (compatibilité en arrière), mais les techniques 'actuellement utilisées' ou recommandées ('best practices') sont en progression.

Chapitres importants

- Sélecteurs
 - Propriétés, Valeurs + Units
 - Inline / Block + Box model
 - Flexbox + mise en page + design process
 - Responsive
 - Webfonts
-

Notions de gestion et d'organisation

- /* utiliser des commentaires! */
 - noter des indications de l'organisation du code, des têtes de chapitre
 - notes pour fonctions 'avancées' ou 'bizarres' – pourquoi on a fait ça, comment ça marche, référence à la documentation ou source de la solution
 - des commentaires sont aussi utilisés pour désactiver du code 'provisoirement'
- hygiène du code: veiller en continu à que votre code soit bien formaté!
- organisation: structure ventilée par thème
 - 'voici le code qui gère la typographie, et par là on s'en occupe de la géométrie de mise en page.'
- organisation: regroupement pour tâche/élément particulier
 - 'voici tous les paramètres pour le menu de navigation'
- DRY ('Don't Repeat Yourself')
 - du code très répétitif signale qu'il y peut être un moyen plus logique ou efficient pour accomplir la tâche
 - du code répétitif (ou mal formaté) est pénible à entretenir, peut facilement cacher des erreurs
 - mais, certaines répétitions sont utiles: on répète des sélecteurs afin de ventiler des blocs de code à des tâches/aspects particuliers, ce qui facilitera la lecture et la modularité du code
 - utiliser (soigneusement) des contrindications avec conscience

Syntaxe

- **Règle** (*rule*) – l'ensemble des particules de syntaxe qui forment une 'phase complète' en CSS
 - un sélecteur et un bloc qui contient des déclarations
- **Sélecteur** (*selector*)
 - cibler le ou les éléments de la page HTML à modifier
- **Bloc** (*block*)
 - { en accolades }
 - tous les paramètres qui seront appliqués à la cible
 - peut contenir plusieurs déclarations
- **Déclaration** (*declaration*)
 - un paramètre à appliquer à la cible
 - la modification qu'on souhaite apporter
- **Propriété** (*property*)
 - l'aspect à modifier
 - la liste des propriétés est longue et constitue "l'épaisseur du dictionnaire" du vocabulaire CSS
- **Valeur** (*value*)
 - le résultat de la propriété souhaitée
 - pour chaque propriété, une série de valeurs possibles est définie
- **Unité** (*unit*)
 - si nécessaire, l'unité de mesure

Origine

Inline (*fortement déconseille*)

```
<p style="color: red;"> hello. </p>
```

Interne

```
<style>
    p { color: red; }
</style>
```

Externe

```
<link rel="stylesheet" href="styles.css">

@import url("style.css");
```

Cascade – en cas de contra-indication, les conflits sont résolus avec les principes de 'la cascade'; un système détaillé, mais qui se résume en deux 'principes' essentiels intuitifs:

- c'est le dernier qui parle qui gagne.
- le précis gagne sur le général.

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>
        <link href="style.css" rel="stylesheet"/>
        <style media="all" type="text/css"> /* code CSS */ </style>
        <script type="text/javascript"> /* code JavaScript */ </script>
        <script src="code.js"></script>
<body>
    <p>
        <h1>, <h2>, <h3>, <h4>, <h5>, <h6>
        <... class="..." id="...">
        <div>, <section>, <main>, <header>, <nav>, <footer>, <article>
        <span>
            <i>, <em>
            <b>, <strong>
            <strike>, <u>, <del>, <ins>, <sup>, <sub>, <small>, <big>
            
            <svg>
            <video autoplay muted controls loop> & <source src="..." type="...">
            <figure> & <figcaption>
            <a href="page2.html"> Lien vers page 2 </a>
            <blockquote>, <q>
            <pre>, <code>
            <ul> / <ol> & <li>
            <table> & <tr> & <td> & <th>
            <form action="..."> &
            <input
                type="radio | checkbox | text | button | ..."
                name="..." id="..." value="..." />
            <label for="input#id">
            <select>, <option>
            <textarea>
            <br/>
            <hr/>
            <!-- commentaire -->
&lt; &gt; &nbsp; &thinsp; &shy;
```


Document HTML5 minimum absolu

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <title>Titre du document</title>
  </head>
  <body>
    ***
  </body>
</html>
```

Gabarit ('boilerplate') des éléments structurels

```
<!DOCTYPE html>

<html lang="fr">

    <head>

        <meta charset="UTF-8">

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

        <title>Titre du document</title>

        <script src="script.js"></script>

        <link href="style.css" rel="stylesheet" />

        <style media="all"> ... </style>

    </head>

    <body>

        <header> ... </header>

        <nav> ... </nav>

        <main> ... </main>

            <section> ... </section>

        <footer> ... </footer>

    </body>

</html>
```

Éléments de structure HTML

```
<div class="machin truc" id="Chose" > ... </div>  
<header>  
<main>  
<footer>  
<nav>  
<section>  
  
  
  
<span class="machin truc" id="Chose" >
```

<div> est un élément **bloc neutre**, qui sert à regrouper ou partager des contenus, sans appliquer un sens ou style prédéfini. On rajoute presque toujours un `class` ou `ID` à des `div` pour les identifier, ou pour appliquer un style particulier.

 Sert le même rôle que `div`, mais en tant qu'élément **inline**

<header>, <main>, <nav>, <footer>, <section>, <article>, <figure> sont aussi des éléments blocs neutres, mais avec des noms qui ont un **sens**, qu'on appelle des blocs **sémantiques**. Ces éléments sont plus simples à lire dans le code source que <div class="header">

Travail de structure en HTML

- Regrouper des éléments associés en des blocs, et rassembler les contenus qui 'vont' ensemble.
- Emballer des éléments ou des blocs ensemble afin qu'ils restent ensemble. (*wrapper*)
- Partager un grand contenu en blocs, pour séparer des éléments. ('*div*' = *division*)

```
<div class="alpha">

    <div class="bravo">

        <div class="delta">
            </div> <!-- .delta -->

    </div> <!-- .bravo -->

    <div class="charlie">

        <div class="echo">

            <div class="foxtrot">
                </div> <!-- .foxtrot -->

            <div class="golf">
                </div> <!-- .golf -->

            </div> <!-- .echo -->

        </div> <!-- .charlie -->

    </div> <!-- .alpha -->
```

Il est essentiel de maintenir activement la mise en page de votre code, que ça reste lisible et la structure est évidente ou au moins compréhensible. Ceci permet plus facilement à identifier des erreurs.

(Ce travail de typographe doit idéalement faire partie de vos obsessions professionnelles!)

- des retours à la ligne, ou multiples retours à la ligne pour rajouter de l'espace
- l'indentation et alignement, avec tabulations et espaces
- des commentaires avec des mots clés utiles ou même des décos

L'emploi des <!-- commentaires --> peut aussi aider à clarifier.

Parfois plusieurs dizaines ou centaines de lignes séparent l'ouverture d'un div de sa fermeture. En bas de nos documents, nous trouvons souvent plusieurs tags de fermeture – un commentaire permet de savoir lequel appartient auquel.

CSS Propriétés (*Properties*)

```
font-family: + font stack, p.ex.: font-family: "Univers", "Helvetica Neue", Helvetica, Arial, sans-serif;
font-size:
font-weight: normal | bold | 100 ... 900 (400 = normal, 700 = bold) | lighter | bolder
font-style: normal | italic
line-height:
color:
text-align: left | right | center | ...
text-indent: letter-spacing: word-spacing:
white-space: normal | nowrap | pre | ...
text-rendering: optimizeSpeed | optimizeLegibility | geometricPrecision
text-decoration: none | line-through | underline | (line style color thickness)
text-transform: capitalize | uppercase | lowercase | none
font-variant: normal | small-caps

margin: x | auto | (top right bottom left) margin-top: / -right: / -bottom: / -left:
padding: x | (top right bottom left) padding-top: / -right: / -bottom: / -left:

border: (width style color)
border-width: x | (top right bottom left)
border-style: none | dotted | dashed | solid | double | ...
border-color:
border-top-width: / -right-style: / -bottom-color: / -left-width: / ...
outline: (width style color)
border-radius: x | (top right bottom left)

width: / min-width: / max-width:
height: / min-height: / max-height:
box-sizing: content-box | border-box
overflow: visible | hidden | scroll | auto
object-fit: fill | contain | cover | none | scale-down
aspect-ratio: w/h

display: inline | block | inline-block | flex | grid | none | contents | ...
flex container    flex-flow: flex-wrap: wrap | nowrap   flex-direction: column | row
  > flex items      flex:   flex-grow:   flex-shrink:   flex-basis:
grid container   grid-template-columns:   grid-template-rows:
  > grid items     grid-column-start / -end:   grid-row-start / -end:
gap:             grid container: row-gap: column-gap:
columns: / column-count: / -width:   > element column-span:

position: static | relative | absolute | fixed | sticky
top: / right: / bottom: / left:
float: left | right | none + clear: left | right | both | none
visibility: visible | hidden

background-color: / -image: / -position: / -repeat: / -size: / ...
background: ...
opacity:
transform: rotate() translate() scale() skew() ...

list... list-style: none | disc | circle | square | lower-alpha | decimal | "string"
list... list-style-image: / -position: / -type:
table... border-spacing: border-collapse: collapse | separate
```

Unités (Units)

px	12pt = 16px
pt	(px = 96dpi)
em	1 em = font-size
rem	root em = 12pt / 16px par défaut
%	(selon contexte...)
vw	% viewport width
vh	% viewport height
mm	
calc(x+y x-y x*y x/y)	
var(--foo)	à déclarer utilisant :root { --foo: bar; }
fr	"fraction" – unité pour grid

Couleurs (Colors)

Hexadécimal (3 ou 6 caractères 0-F)	00 = 0
#RRGGBB	11 = 17
#RGB	33 = 51
#RGB => #RRGGBB	66 = 102
	99 = 153
	CC = 204
#F1357A = rgb(241, 53, 122)	FF = 255
'F1' = 241 '35' = 53 '7A' = 122	

RGB / RGBA (RGB 0–255, Opacité 0–1)

rgb(241, 53, 122)

rgba(241, 53, 122, 0.5)

HSL / HSLA (Teinte 0–360, Saturation %, Luminosité %)

hsl(338, 87%, 58%)

hsla(338, 87%, 58%, 0.5)

Nommées

beige, fuchsia, darkseagreen, ...

Dégradés

background: linear-gradient(45deg, red, blue);

Fond image

background: url("file.jpg") no-repeat center center fixed;
background-size: cover;

/* commentaire */

Valeurs (Values)

inherit | auto | none | unset | initial | revert
... !important;

Selecteurs (Selectors)

tag
#id
.class
***** ("wildcard" / tout / n'importe quoi)

foo , bar (liste)
foo bar (parent → descendant)
foo > bar (parent → enfant directe)
foo + bar (élément suivant)

[attribute] (l'attribut existe)
[attribute="value"] (la valeur précise est...)
[attribute*="value"] (la valeur contient...)

a:link, a:active, a:visited, a:hover

:hover

:first-of-type :first-child
:last-of-type :last-child

:nth-of-type() :nth-child()
even | odd | an | an+b

:not(selector)

::before, ::after → {content: "foo";}

::selection
::first-line, ::first-letter

At-rules

@font-face { font-family...; src...; ... }

@media (requête-média) { ...{...} }
all | screen | print
(min-width) | (max-width)

@import "style.css";
@import url("style.css");

Sélecteurs CSS

Les sélecteurs CSS ciblent les éléments HTML

- balise/tag (nom) des éléments, attributs de class et ID
- position dans le "DOM" (Document Object Model):
structure imbriquée du document, hiérarchie & arborescence, rapport parent-enfant

Sélecteur CSS

- sélectionne un ou plusieurs éléments à modifier dans la page,
fonction à 'cibler' certains éléments
 - la partie d'une règle css qui précède le bloc en accolades: **sélecteur { }**
 - le bloc qui suit contient les paramètres qui seront appliqués aux éléments ciblés
-

Tag / Balise

```
p { color: red; }  
h1 { color: blue; }
```

Le sélecteur "type" est simplement la balise HTML (*sans les <chevrons> !!*), tous les éléments de ce type de balise sont ciblés.

```
.class <p class="important"> ... </p>  
.important { color: hotpink; }
```

Pour cibler un élément selon le *class*, on emploie un *point* "." suivi par le nom de class.

Ce sélecteur peut être précédé par une balise HTML (sans espace) pour être plus précis.

```
p.important { color: fuchsia; }
```

```
#ID <h1 id="LogoType"> ... </h1>  
#LogoType { color: steelblue; }
```

Pour cibler un élément selon un attribut d'ID, on emploi un *croisillon #* ou "hashtag" (raccourci clavier ↲ 3).

Les ID sont uniques. Les classes peuvent re-utilisés autant que nécessaire.

Liste, de, sélecteurs

Plusieurs sélecteurs séparés par des *virgules* ,
sont modifiées par le même bloc de déclarations.

```
h1, h2, h3 { font-family: sans-serif; }
```

Sélecteurs hiérarchiques

[espace]

```
header h1 { font-size: 48pt; }
```

Deux cibles séparées par un espace sélectionnent un élément **enfant**, ou descendant.
À lire/analyser de droite à gauche: on cible un élément enfant contenu dans un parent.

>

L'emploi d'un *chevron* (symbole 'plus grand que') est utilisé pour cibler qu'un descendant direct (plus stricte qu'un espace: un élément petit-enfant n'est pas sélectionné).

```
nav > ul { background-color: grey; }
```

+

Le symbole *plus* permet de cibler un élément suivant un autre, un voisin immédiat.
Le symbole *tilde* ~ sélectionné un voisin suivant "à proximité", plus loin dans le même parent.

```
h3 + p { margin-top: 0.25rem; }
```

*

L'*astérisque* (étoile) est le sélecteur universel: qui cible n'importe quel élément.

```
* { box-sizing: border-box; }
```

Références

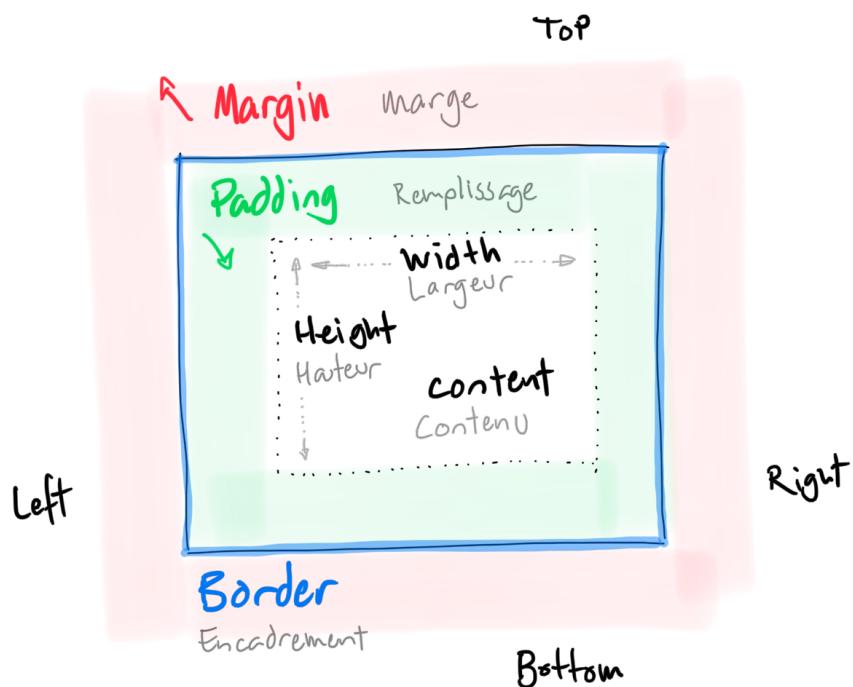
- <https://css-tricks.com/how-css-selectors-work/>
- https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Selectors
- [Jeu: CSS Diner \(flukeout.github.io\)](https://flukeout.github.io/)

Box model - modèle de boîte

Paramètres

- **width height** largeur & hauteur
 - min-width, max-width, min-height, max-height
- **margin** marge - extérieure
 - margin-top, margin-right, margin-bottom, margin-left
 - margin: top right bottom left ;
- **padding** remplissage - intérieure
 - ... idem margin
- **border:** width style color ;
- **box-sizing:** content-box | border-box ;
 - comment mesurer la taille d'un élément: avec ou sans le border et le padding
- **overflow:** visible | hidden | scroll ;
 - comment gérer un contenu qui est trop grand
- **display:** block | inline | inline-block ;
 - changer le comportement inline/block d'un élément
(les éléments n'ont pas toutes les propriétés de boîte telles qu'une largeur)
inline-block est un hybride des deux: un block qui se positionne inline
 - display peut aussi gérer les éléments enfants contenus: display: flex | grid ;
- **outline, outline-offset**

Box Model



Inline / Block

Composition de page

- Navigateurs web (*browsers*): essentiellement un **moteur de rendu**
 - interprétation de la source HTML & CSS, dessin et affichage du résultat à l'écran
- Web: un médium conçu et principalement orienté vers le **texte**
 - paradigme '**machine à écrire**' / composition typographique d'imprimerie / MS Word

The Flow

- Le **Flux** des éléments qui composent le page
 - le "cours" des contenus, liquide
 - les contenus définissent la mise en page
 - contraire d'un modèle *Illustrator* ou *PostScript* où le canevas ou page vient en premier
- **Inline** ("en ligne")
 - le texte est composé des caractères, en séquence
 - ce cours de texte en ligne remplit l'espace disponible de gauche à droite dans le sens de lecture
 - lorsque la limite de la zone texte est atteinte, le texte inline fait retour à la ligne: (**text wrap**)
- **Block:** les éléments sont empilés les uns après/sous les autres, de haut en bas
- **Axes:**
 - x = inline →
 - y = block ↓

Dimensions d'un bloc

- Un bloc se remplit avec le flux inline; la longueur des lignes est définie par l'espace disponible; puis le nombre de lignes est défini par la quantité de texte
- La **largeur** d'un bloc est définie par l'espace extérieur (*parent*)
- La **hauteur** est définie par la quantité des contenus intérieure (*enfants*)
- Les blocs occupent l'espace disponible en horizontale, et s'agrandissent en verticale pour accommoder leurs contenus

Références

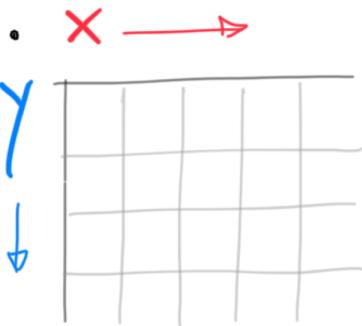
CSS Flow Layout

https://developer.mozilla.org/fr/docs/Learn/CSS/CSS_layout/Normal_Flow

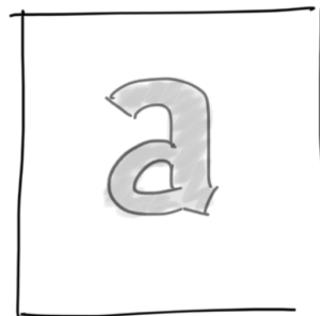
Disposition de bloc et en ligne avec le flux normal

https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Flow_Layout/Block_and_Inline_Layout_in_Normal_Flow

Postscript

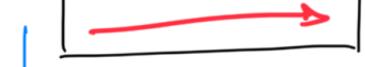


Illustrator

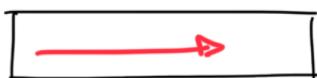
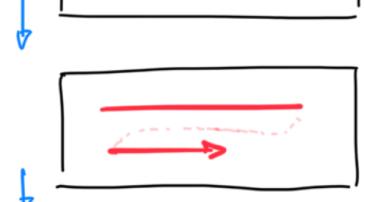
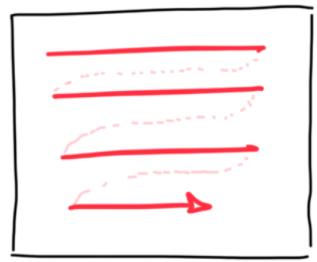


HTML

inline



block

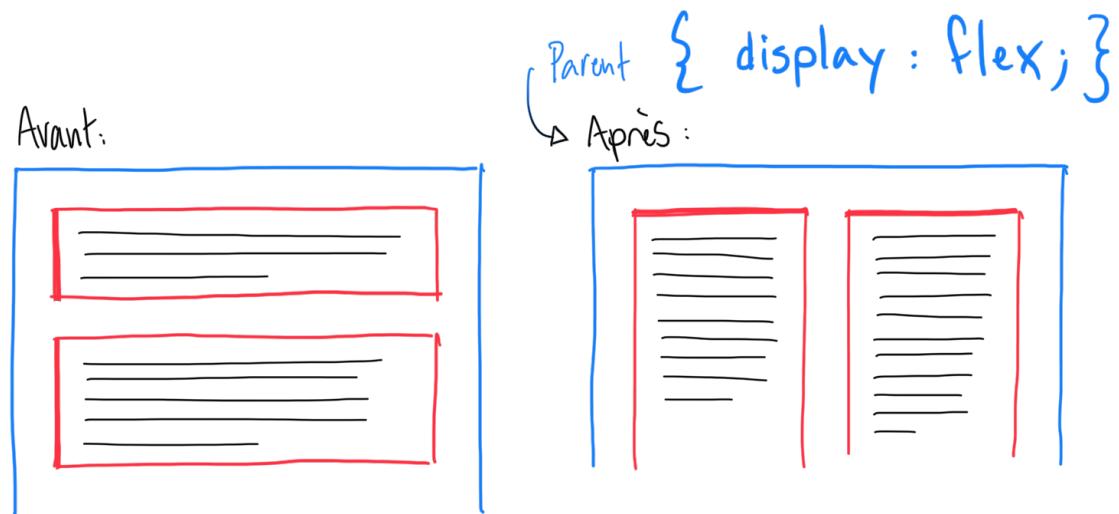


Word.doc



Flexbox

- Structure: parent → enfants
- Flex container → Flex items**
- Mot clé: un paramètre CSS pour 'déclencher' le mode flex sur un élément parent: **display: flex;**
 - peu importe les éléments HTML, balises / classes / ID



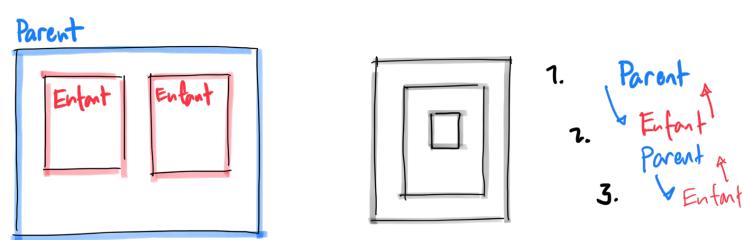
Flex: affichage en colonnes

```
.parent { display: flex; }  
.enfant { flex: 1; }
```

Système réutilisable avec un `class="row"`

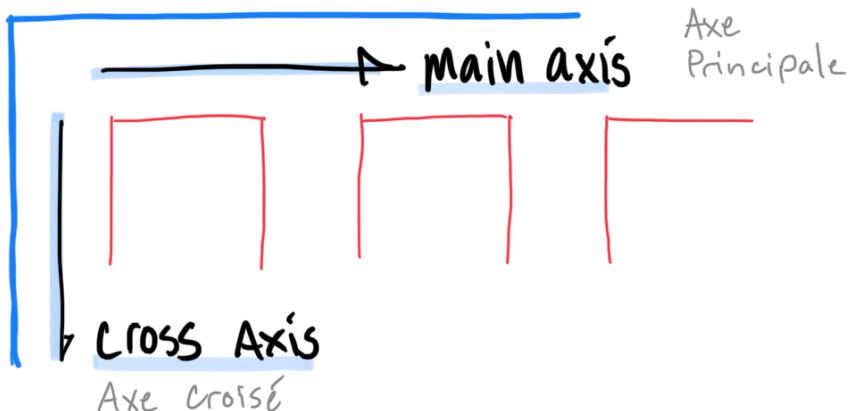
```
.row { display: flex; gap: 1rem; }  
.row > * { flex: 1; }
```

Utilisation du sélecteur descendant direct `>` et le sélecteur universel `*` pour gérer n'importe quels éléments enfants. (mais pas les petits-enfants!)
Le paramètre `gap` gère l'espace gouttière entre les colonnes.



Axes horizontal & vertical

Les axes appelés "x, y" dans un contexte Post-script (Illustrator...),
appelées "inline et block" dans le flux HTML standard,
sont appelées "**main**" & "**cross**" dans un mode flexbox.

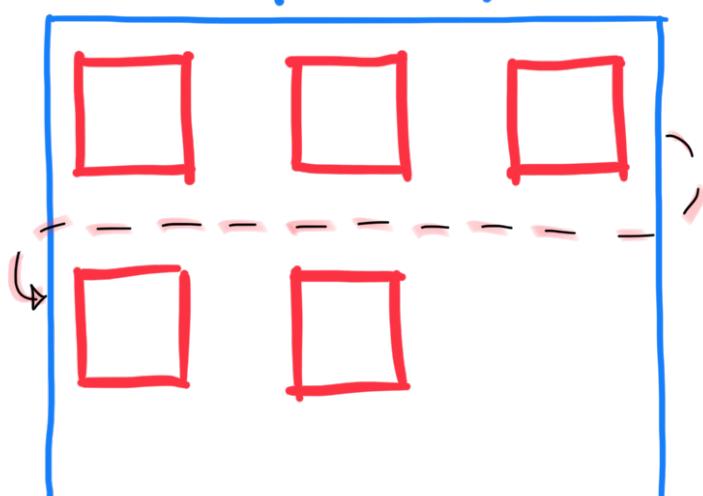


Flex-wrap

Crée un comportement "retour à la ligne" des éléments enfants, selon l'espace disponible (similaire à inline-block) – utile pour afficher les enfants en "grille":

```
.parent {  
    display: flex;  
    flex-wrap: wrap;  
}  
.enfant { flex: 0 0 12rem; }
```

flex-wrap: wrap;



Flexbox – syntaxe & paramètres

élément parent

- **display: flex;**
- **justify-content** ↔ disposition des éléments sur l'axe X horizontale – **axe principal – "main axis"**
justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly
- **align-items** ↑ disposition des éléments sur l'axe Y vertical – **axe croisé – "cross axis"**
align-items: flex-start | flex-end | center | baseline | stretch
- **flex-direction** – modifie le sens d'alignement des éléments
flex-direction: row | row-reverse | column | column-reverse;
- **gap** – espaces entre les éléments (gouttière) – différence h/v avec **row-gap** et **column-gap**
- **flex-wrap** – retour “à la ligne” des éléments, selon espace disponible (similaire à inline-block)
flex-wrap: nowrap | wrap | wrap-reverse;

éléments enfant:

- **flex: grow shrink basis** – comportement de taille des éléments, leur permettant d'agrandir ou rétrécir selon l'espace disponible, ainsi que leur taille de base.
 - **grow** facteur/proportion d'expansion ou agrandissement (normalement 0 ou 1)
 - **shrink** facteur/proportion de rétrécissement (normalement 0 ou 1)
 - **basis** dimension (largeur) taille de base souhaitée ou initiale
 - **flex: 1;** – raccourci pour **flex: 1 1 auto;**
 - **order** – permet de changer l'ordre d'un élément
-

Références

[CSS-tricks – A Complete Guide to Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

[CSS-tricks – Don't Overthink It](https://css-tricks.com/dont-overthink-it/)
<https://css-tricks.com/dont-overthink-it/>

[MDN – Guide de mise en page avec CSS – Flexbox \(en français\)](https://developer.mozilla.org/fr/docs/Learn/CSS/CSS_layout/Flexbox)
https://developer.mozilla.org/fr/docs/Learn/CSS/CSS_layout/Flexbox

web design process

croquis thumbnail – wireframe – maquette

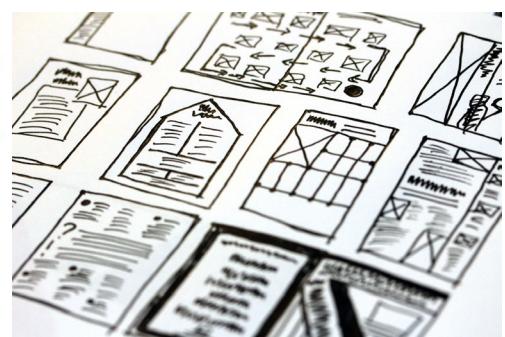
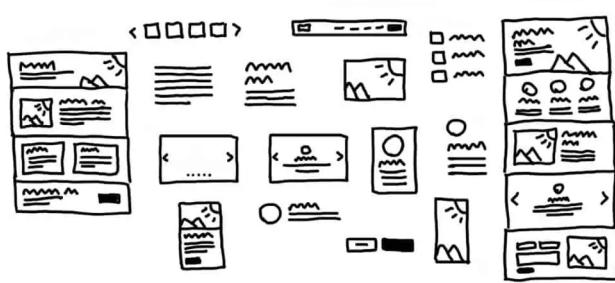
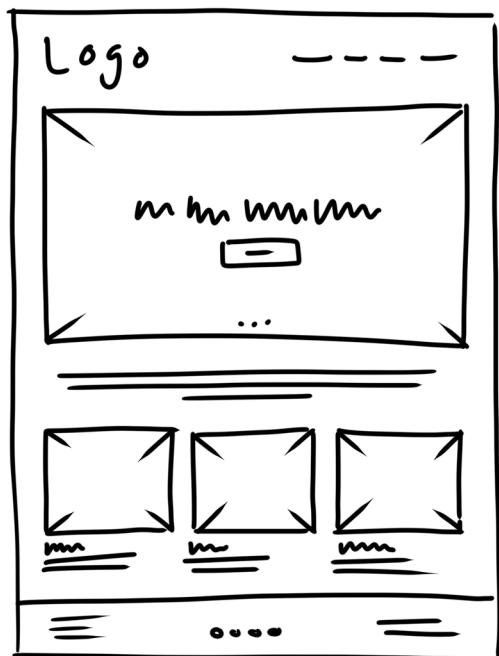
Croquis – Thumbnail

But: réfléchir rapidement aux structures visuelles, établir des stratégies de disposition générale, développer un concept

Médium: libre (crayon & papier)

Enjeu: brainstorming visuel, travailler rapidement, créer le plus grand nombre de variations

Astuce: travailler avec un gros feutre pour ne pas se perdre dans les détails

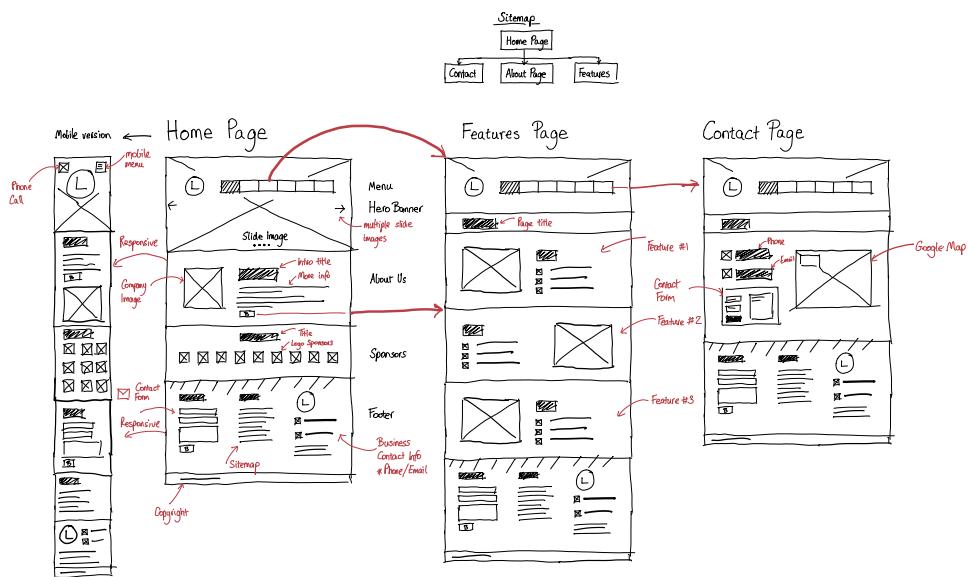
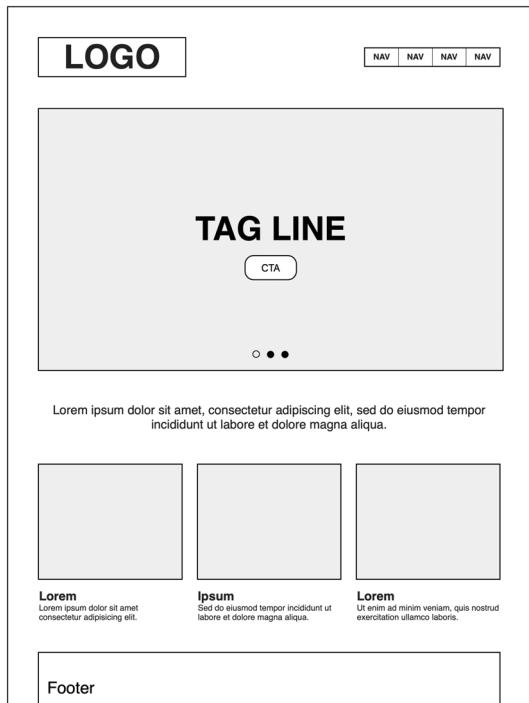


Wireframe

Buts: concevoir et développer la structure détaillée de la page; placer les éléments nécessaires dans l'espace, établir et détailler leur ordre et hiérarchie; avoir un plan.

Enjeu: travailler sur les contenus et leur structure en abstraction, sans se préoccuper de l'habillage visuel.

Médium: Crayon & papier, Illustrator, Indesign, Figma, ...



Maquette

But: détailler la présentation visuelle, établir le langage graphique, partager une vision avec clients & collaborateurs.

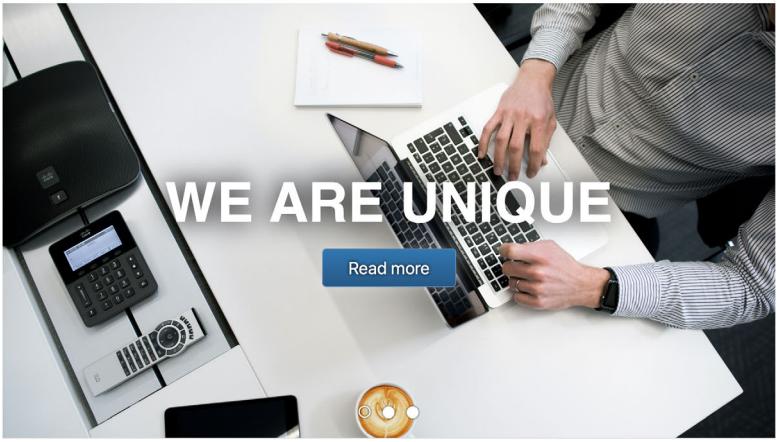
Enjeu: créer une maquette *réalisable* (la maquette est aussi une promesse).

Médium: Figma, Photoshop, Illustrator/InDesign, ...



YOUR WEBSITE

HOME | INFO | ABOUT | CONTACT



We are extremely unique and different from our competitors by having a website that looks exactly the same.



Always.
We could have four columns. But everyone else has three.



Three.
Feeling creative we added pic of smiling woman here.



Columns.
Yup, the website layout says we can offer only 3 services.

Generic Web Inc.
123 Main Avenue Road
Springfield 12345



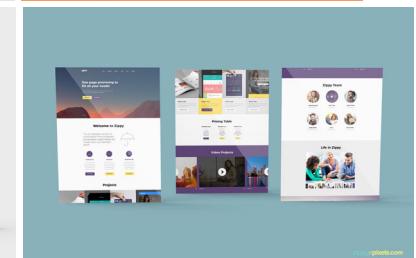
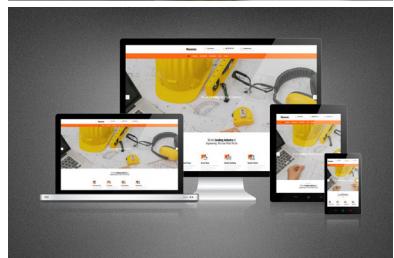
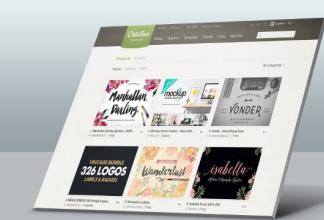
General Terms and Conditions
©2023 Generic Web Inc.

En architecture, une maquette est une représentation physique ou virtuelle réduite d'une structure construite dans le but d'étudier des aspects particuliers d'une conception architecturale ou de présenter un projet.

Des plans architecturaux incompréhensible, abstraite:



Des maquettes fascinante, intuitive:



web design process

Découpe

But: traduire un concept de mise en page visuel en code HTML & CSS

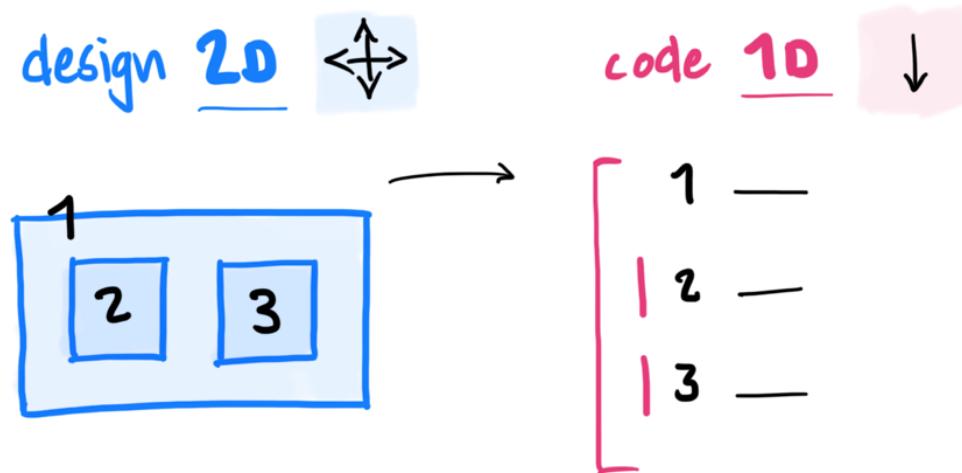
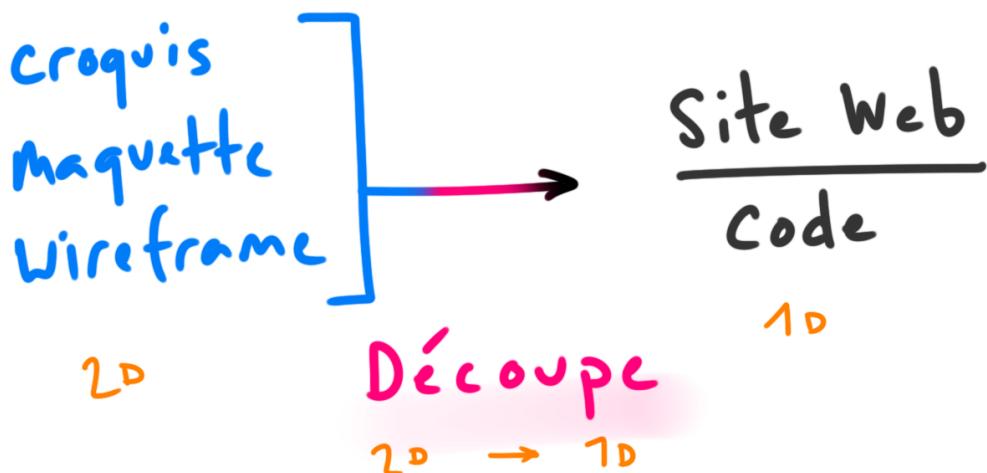
Médium: libre (crayon & papier, photocopie, stabilo boss, danse interprétative)

Enjeu: analyse 2D → 1D – visuel vers abstrait

Astuce: Analyse bloc par bloc.

Passage de l'extérieur vers l'intérieure, de haut en bas, et de gauche à droite.

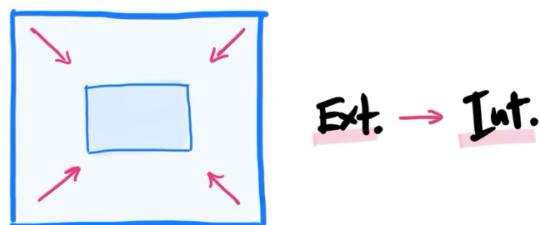
L'ordre de ces passages peut changer selon le résultat désiré.



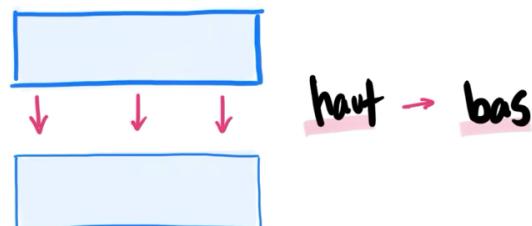
extérieur → intérieur

haut → bas

gauche → droit.



Ext. → Int.



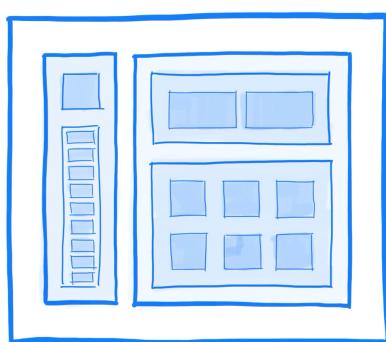
haut → bas

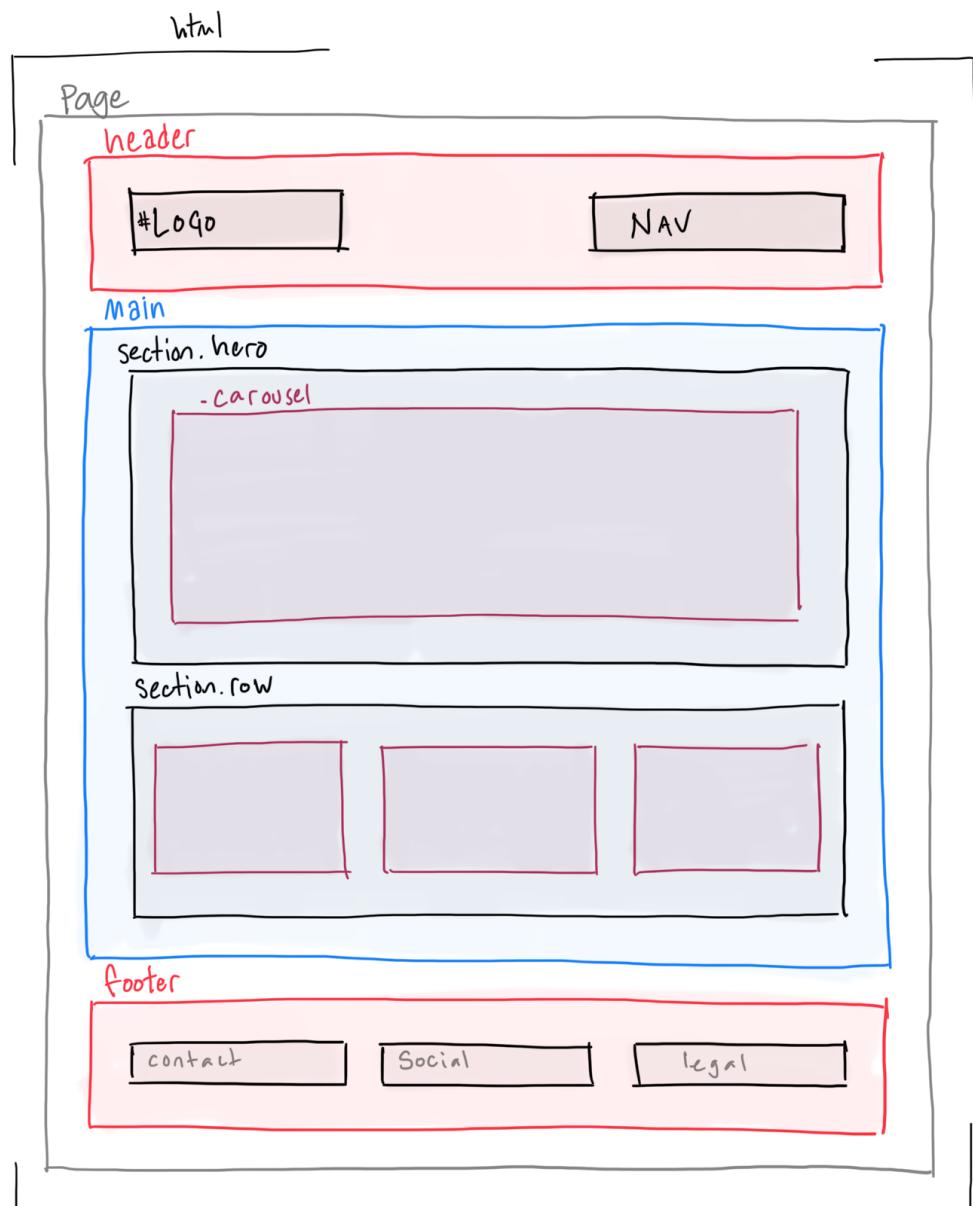
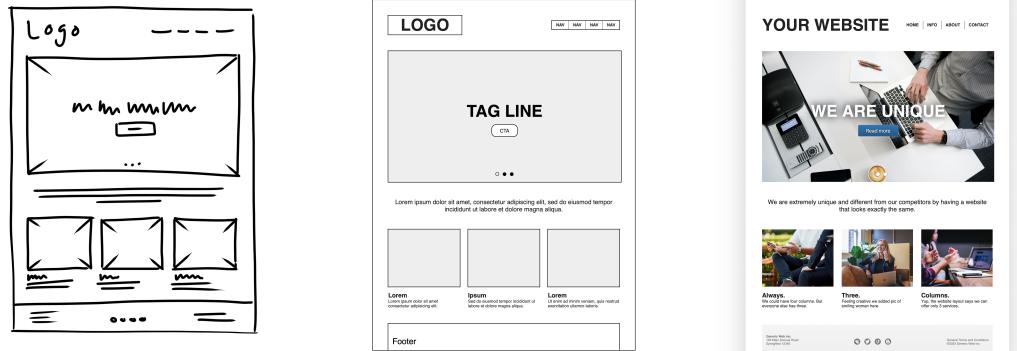


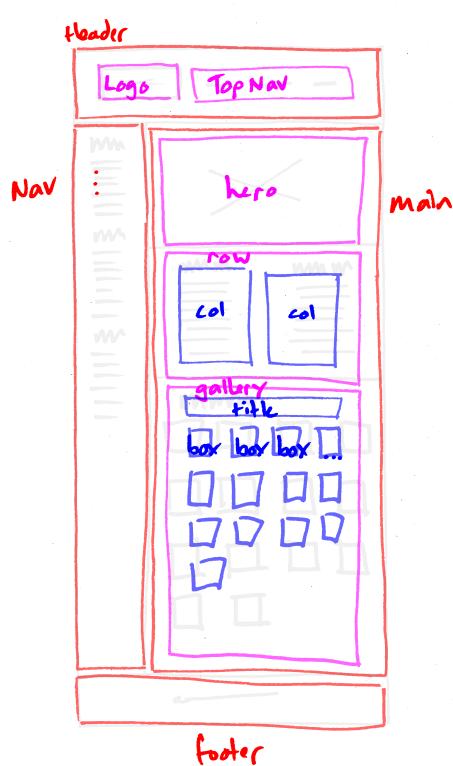
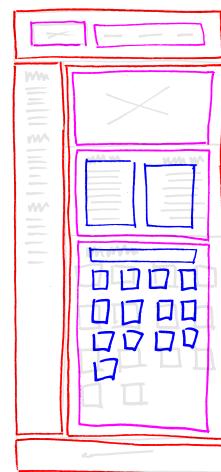
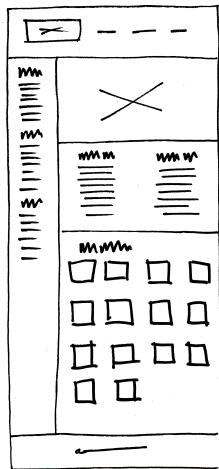
gauche → droite

Résultat: des blocs dans des blocs dans des blocs dans des blocs dans des blocs...

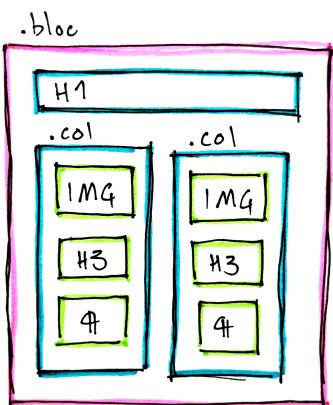
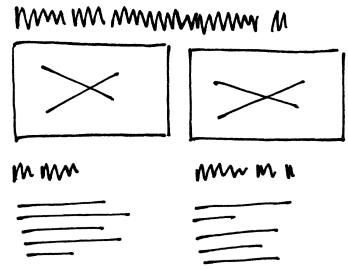
Avec le système Flex, chaque bloc est orienté dans un seul sens.
Des blocs intérieurs permet de changer l'orientation des blocs.







Croquis



Structure



HTML

```
<div class="bloc">  
  <h1> Bla bla chaton </h1>  
  
  <div class="col">  
      
    <h3> Lorem ipsum dolor </h3>  
    <p> Ut enim ad minim... </p>  
  </div> <!-- .col -->  
  
  <div class="col">  
      
    <h3> Duis aute irure... </h3>  
    <p> Velit esse cillum... </p>  
  </div> <!-- .col -->  
</div> <!-- .bloc -->
```

CSS

```
.bloc {  
  display: flex;  
  flex-wrap: wrap;  
}  
  
.block > h1 {  
  width: 100%;  
}  
  
.block > .col {  
  width: 50%;  
}
```


CSS Position

La propriété CSS **position** est un outil pour définir la façon dont un élément est placé dans la page. D'autres outils comme `display: flex` sont efficaces pour gérer le flux des éléments, mais pour certains effets on peut employer la position **absolute**, **fixed** ou **sticky**.

Sans intervention de notre part, un élément est par défaut positionné en mode **position: static;** qui est le comportement 'normale' d'un bloc.

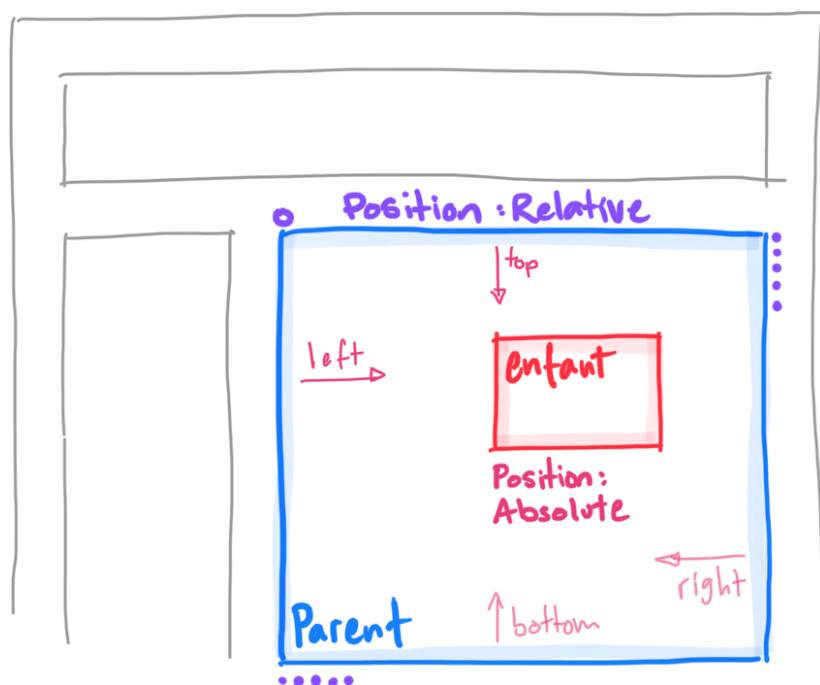
Absolute

On peut positionner un élément précisément sur les axes x et y avec la valeur `position: absolute;`

Le canevas ou zone dans laquelle on positionne l'élément est défini par un élément parent qui avec la valeur `position: relative;`

Parent (ou ancêtre, ou le body): `position: relative;`

Enfant à positionner: `position: absolute;`



Le position de l'élément sur les axes horizontal et vertical est défini avec les propriétés `top:` / `bottom:` / `left:` / `right:`

Position relatif à la fenêtre

Souvent employées pour les éléments de navigation ou UI, les valeurs **fixed** et **sticky** permettent de positionner un élément relative à la *fenêtre* du navigateur.

`position: fixed;` positionne l'élément par rapport au viewport (fenêtre)

`position: sticky;` un hybride de *static* et *fixed* selon le **scroll** du document

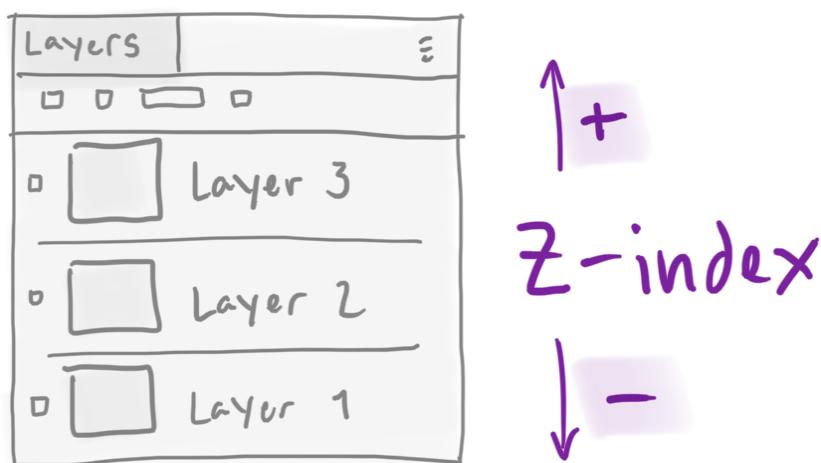
Z-index

Le positionnement des éléments hors du flux normal de la page implique des chevauchements. Le CSS prévoit un système de similaire à des calques pour gérer la superposition.

Les éléments avec une valeur **z-index** supérieur sont placées devant d'autres éléments avec une valeur **z-index** inférieur.

Les valeurs de z-index sont simplement des chiffres (positives ou négatives)

```
z-index: 999;  
z-index: 2;  
z-index: -1;
```



<https://developer.mozilla.org/fr/docs/Web/CSS/position>

<https://css-tricks.com/almanac/properties/p/position/>

Unités CSS

Dans le CSS, les propriétés sont suivies des **valeurs**.

Pour la mesure des dimensions ou distances des éléments, telle que la largeur d'un bloc ou la taille de texte, la valeur est chiffrée, et suivie d'une **unité**.

`font-size: 12pt;`

L'unité de mesure n'a pas besoin d'être singulière, constante, ou identique dans un projet ou même dans un bloc de code: on peut changer d'unité selon le besoin.

`font-size: 16px;`

Les unités **absolues** sont fixes et ne changent pas de valeur dans une page.

Les unités **relatives** s'adaptent leur valeur selon contexte.

`width: 80mm;`

`width: 50%;`

Les unités de **taille et distance** les plus courantes dans le web design sont

- **PX – pixel**
 - **PT – point**
 - **EM – em**
 - **REM – Root-EM**
 - **% – Pourcent**
 - **VH – Viewport Height & VW – Viewport Width**
-

https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Values_and_Units#dimensions

<https://css-tricks.com/the-lengths-of-css/>

Texte

La taille de texte par défaut dans une page web est **16px** ou **12pt**.

Font Size

12 pt = 16 px = 1rem

Le HTML étant un médium d'abord textuel, cette valeur est une référence axiomatique dans la conception web.

Une approche de web design rationnel, flexible, et ergonomique commence avec le texte courant comme mesure de départ. La géométrie des éléments plus grandes de la page, comme des colonnes de texte, puis la disposition générale des éléments, peuvent être calculées à partir de cette unité de base (REM-Based Layouts).



Points et Pixels

PX – Pixel

1px = 1/96 de pouce (inch) = "96 dpi".

1px = 1 pixel 'logique' d'écran.

Le pixel est une unité qui fait référence aux dimensions des pixels des écrans. Les écrans contemporains, étant de diverses résolutions et souvent d'une densité très élevée ('retina'), la matrice des **pixels physiques** qui composent le panneau d'affichage ne sont (plus) une mesure rationnelle.

Les interfaces graphiques (et les navigateurs) travaillent sur un canevas de **px logiques** ou virtuelles, souvent d'environ 96ppi (en imitation de l'espace de travail disponible sur d'anciens écrans, mais avec une qualité plus détaillé).

Les images bitmap/raster/matrice sont également composées de pixels, mais il n'y a pas de rapport direct ou fixe entre ceux-ci et l'unité des PX en CSS.

PT – Point

1pt = 1/72 de pouce = "72 ppi".

Le point est une unité typographique, l'unité traditionnelle pour mesurer la taille des caractères.

La taille par défaut de texte est 12pt.

Conversion PT / PX

ratio 3:4

1pt = 1.333px

1px = 0.75pt

16px = 12pt

96px = 72pt

IN – Pouce – Inch

CM – Centimètre

**1 pouce = "inch" – abréviation "in"
= 2.54 cm**

Points & Pixels



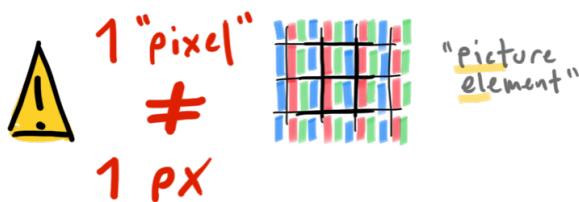
$$\begin{array}{c|c} \text{1 in} \\ \text{pouce} \\ \hline \end{array} = \begin{array}{c|c} \text{2.54 cm} \\ \hline \end{array}$$
$$\begin{array}{c|c} \text{72 pt} \\ \text{"72 ppi"} \\ \hline \end{array} = \begin{array}{c|c} \text{96 px} \\ \text{"96 ppi"} \\ \hline \end{array}$$

0.75 : 1

1 : 1.333̄

3/4 : 1

Écrans & Images



17" CRT 96ppi

1280 × 960 pixels

Pixels physiques



13" LCD 220ppi

2560 × 1920 pixels

$$1280 \times 960 \text{ px} = 1280 \times 960 \text{ px}$$

px : pixels logique
espace de travail

EM

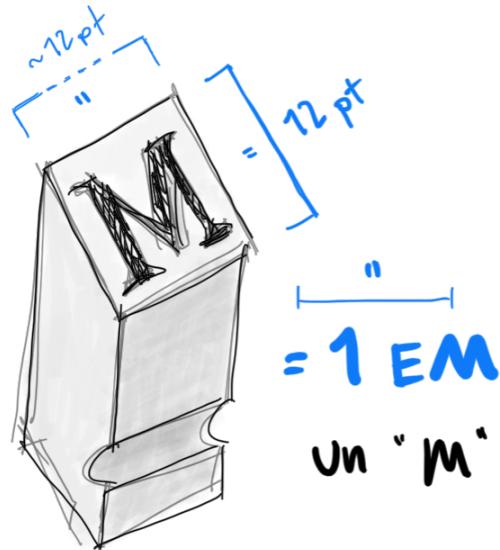
Unité qui correspond à la taille du texte actuel.

Cette unité est **relative et change selon le contexte**:

Dans un `<p>` paragraphe de texte normal un em peut être 12pt,
mais dans un titre `<h1>` un em peut valoir 36pt.

Son nom vient du caractère "M", étant la lettre le plus large (et son profil en plomb étant à peu près carré) cette dimension se prêtait aux mesures relatives à la taille de la typo utilisée.

Cette unité est très utile si on souhaite dimensionner un élément par rapport au texte qui l'entoure ou à proximité, ou surtout si on souhaite qu'un élément adapte sa taille avec le texte.



Paragraphe font-size 12 pt } **1em** = 12 pt = 16 px

H1 Font-Size 36 pt **1em** = 36 pt = 48 px

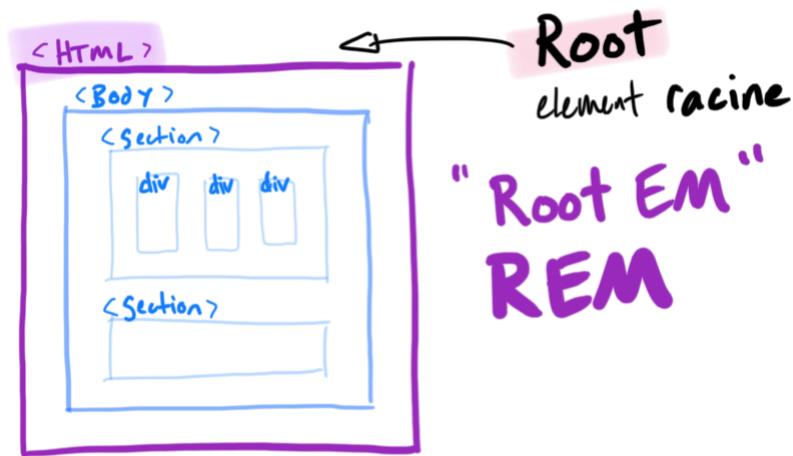
REM – Root EM

Le **rem** ("root em") est la valeur de la taille de texte (em) de l'élément **racine** du document, le tag `<html>`.

Cette unité est donc **semi-fixe** ou *pratiquement absolue*: garantit d'être pareil partout dans une page; le plus souvent inchangé de **16px/12pt**.

La valeur de cette unité *peut* changer avec un zoom de la taille d'affichage de la page, ou avec un réglage particulier, mais sera toujours pareil dans toute la page.

Ce comportement semi-fixe rend le REM utile pour la gestion de la mise en page responsive ou adaptative: une valeur stable, mais qui s'adapte selon besoin.



$$\text{HTML font-size} = 1\text{rem} = 12 \text{ pt} \\ = 16 \text{ px}$$



html { font-size: 10 pt; }
/* omg! 1REM = 10pt WTF? */

Pourcent

% – Pourcent de largeur

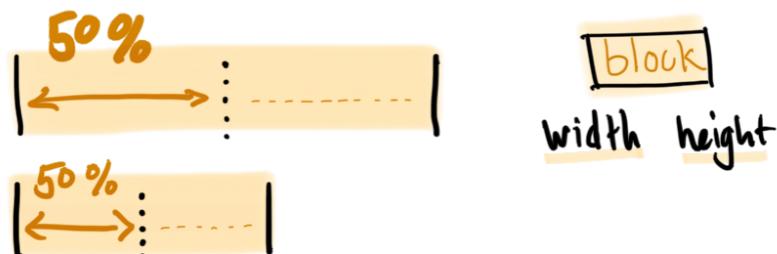
Cette unité n'a pas de valeur fixe, elle correspond (généralement) à l'espace disponible dans un bloc.

Un élément réglé à 50% occupera la moitié de l'espace disponible dans son bloc parent.

% – Pourcent de texte (font-size)

Si le % est utilisé pour des régales typographiques (notamment le line-height, ce pour-cent est relatif à un em).

% Pourcent



inline → font-size line-height

H1 Font-Size 36 pt 150% = 54 pt

Paragraphe font size 12 pt 150% = 18 pt

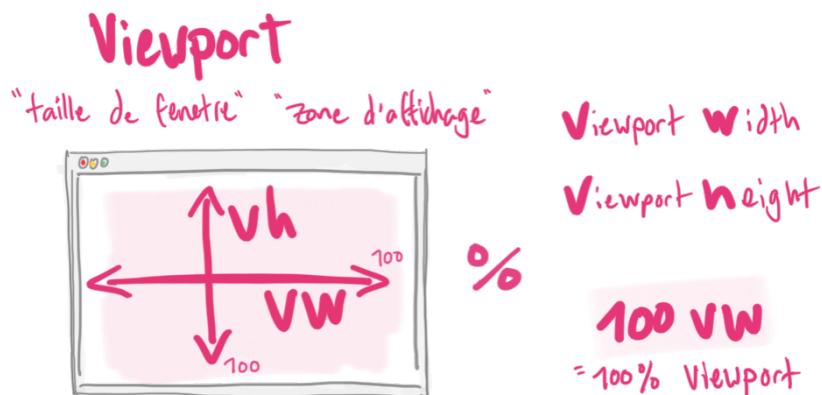
VH – Viewport height

Les unités VH et VW correspondant à la taille de la fenêtre du navigateur.

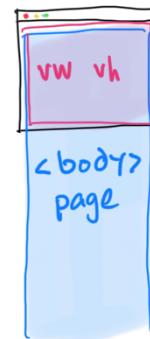
Les unités sont de 1% de la dimension, donc 100vh est l'hauteur de la fenêtre.
(100vw correspond à 100% de la largeur de fenêtre)

Ces valeurs changent avec le redimensionnement de la fenêtre du navigateur,
ou le selon la taille d'écran d'un smartphone.

- **vw** – 1% de la largeur de la fenêtre
- **vh** – 1% de la hauteur de la fenêtre
- **vmin & vmax** – 1% de la dimension le plus petite ou la plus grande de la fenêtre, selon l'orientation portrait ou paysage



⚠️ $Vh \neq$ hauteur de la page ou le `<body>`



⚠️ Attention: la valeur `100vw` (Viewport width) est rarement utilisée:
la largeur des 'ascenseurs' de scroll ne sont pas soustrait de cette valeur,
donc l'emploi du `width: 100vw` provoque un scroll horizontal peu desirable!
La simple valeur de `width: 100%` est plus utile!

Échelle

Design basée sur le REM – Le HTML étant un médium d'abord textuel, cette valeur est une référence axiomatique dans la conception web.

1rem = 12pt = 16px

échelle:

$$\boxed{12\text{ pt}} = \boxed{16\text{ px}} = \boxed{1\text{ rem}}$$

Texte & Éléments graphiques

- <p> – 12pt = 16px = 1rem
- <h1> – 30pt = 40px = 2.5rem

Hello. # paragraphe
Font-size : 12 pt
16 px
1 rem

Titre H1
Font-size : 30 pt
40 px
2.5 rem

- Emoji: – 12pt = 16px = 1rem
- Icône desktop: 24–48pt = 32–64px = 2–4rem

⌚ emoji 16px 1rem

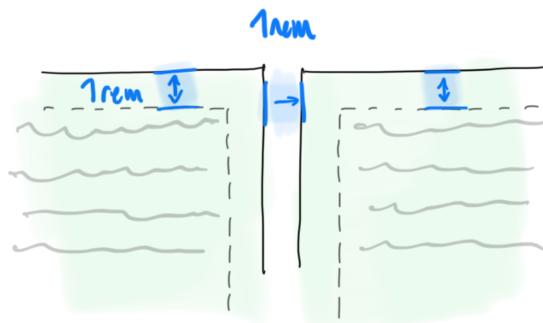
📁 icon 32–64px 2–4rem

🎯 touch target <48px <3 rem

Mise en page

Colonne de text:

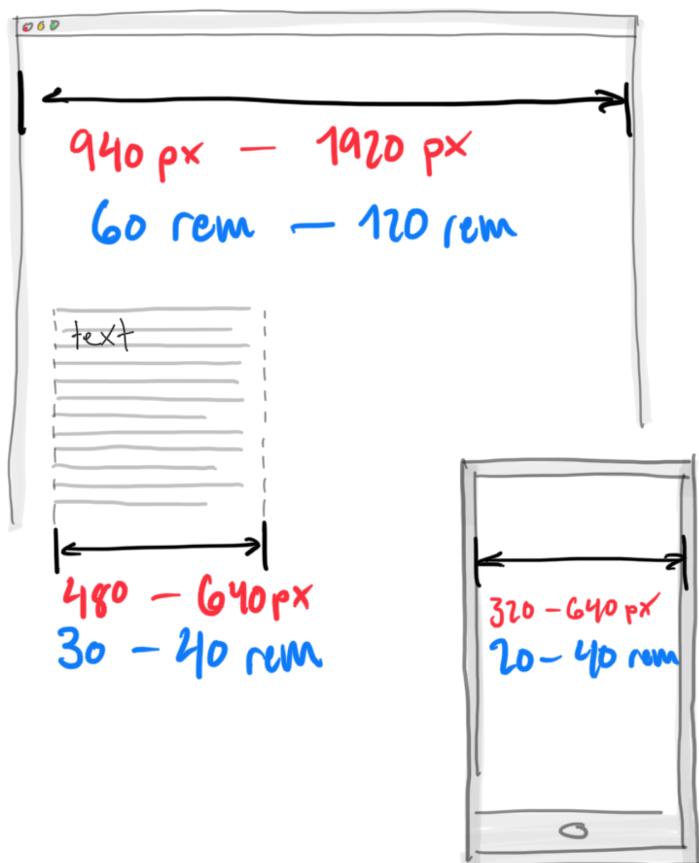
- 8 – 10 mots / 40 – 80 caractères
 $= 30 - 40 \text{ rem} = 480 - 640 \text{ px} = 360 - 480 \text{ pt}$
- margin entre éléments, gouttière entre colonnes, padding dans un bloc: 1–2 rem



Viewport – Fenêtre de navigateur – Écran

Taille des écrans: $= ? \times \infty$

- mobile: $20 < 40 \text{ rem} (320 < 640\text{px})$
- desktop: $> 60 \text{ rem} - 120 \text{ rem} (940\text{px} - 1920\text{px})$

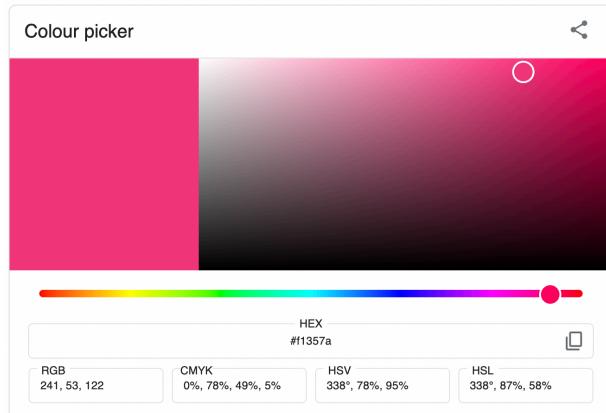


CSS Color

La couleur sur le web est additive en RVB.

L'espace / profil colorimétrique du web est (par défaut) le sRGB.

Un outil de choix et conversion de couleur est facilement accessible avec une recherche google pour "color picker"



https://developer.mozilla.org/fr/docs/Web/CSS/color_value

Couleur RGB()

L'annotation de couleur le plus facile à lire et à rapporter des valeurs d'autres logiciels de design est le format `rgb()`, qui accepte les valeurs standards de 0 à 255 pour le rouge, vert, et bleu, séparé par des virgules:

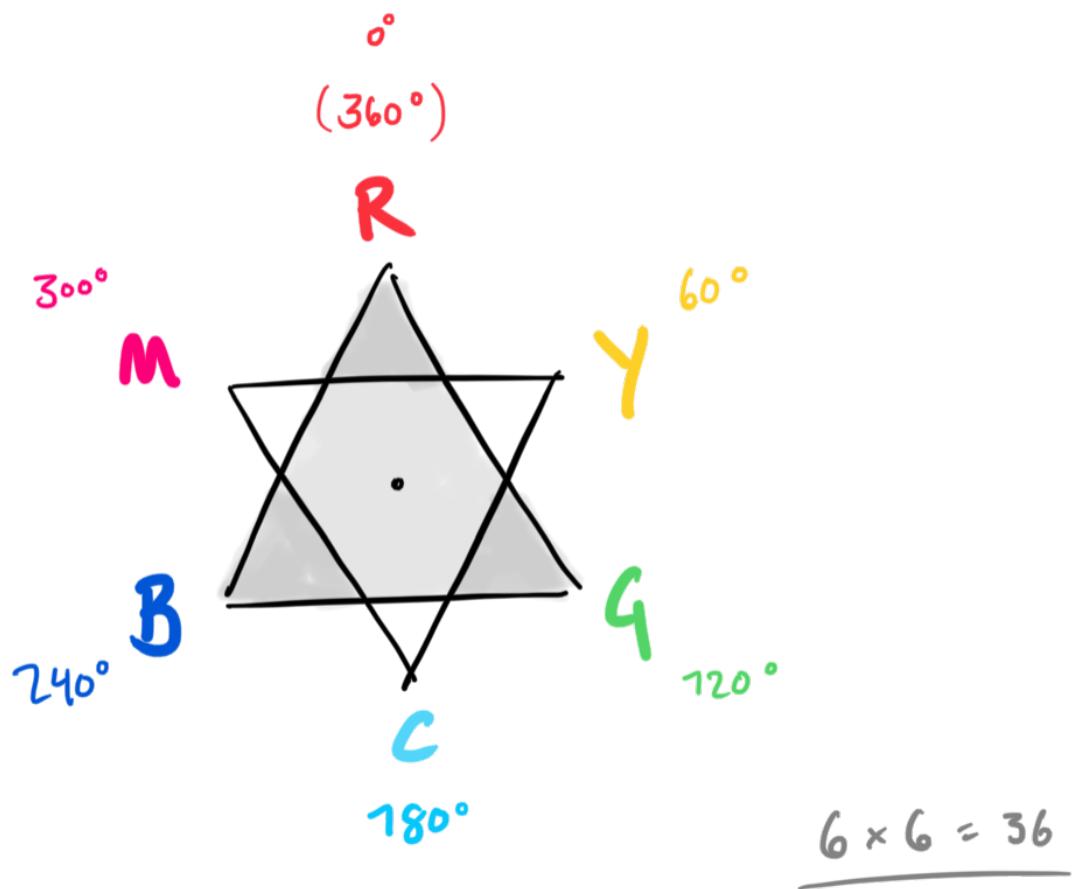
```
color: rgb(241, 53, 122);
```

Couleur RGBA()

Si on souhaite utiliser une couleur semi-transparente, on peut le spécifier en `rgba()`, le "a" pour alpha, l'opacité, qui sera une valeur entre **0** (entièrement transparent) et **1** (totalement opaque).

Attention, le séparateur décimal est le point. non pas la virgule,

```
background-color: rgba(241, 53, 122, 0.5);
```



$$\begin{array}{c}
 \text{R} \rightarrow \\
 \text{---} \\
 \text{Y} \\
 \text{---} \\
 \text{G}
 \end{array}
 \quad \text{Additif RGB:} \quad \frac{\text{R} + \text{G} = \text{Y}}{\text{R} + \text{G} = \text{Y}}$$

$$\begin{array}{c}
 \text{R} \rightarrow \\
 \text{---} \\
 \text{M} \cdot \text{---} \cdot \text{Y} \\
 \text{---} \\
 \text{---}
 \end{array}
 \quad \text{Soustractif CMY:} \quad \frac{\text{C} + \text{M} + \text{Y} = \text{R}}{\text{C} + \text{M} + \text{Y} = \text{R}}$$

#Hexadécimal ("hex")

Pour une notation un peu plus compacte avec moins de ponctuation, on annote traditionnellement les couleurs avec des valeurs de 0 à 255 en format hexadécimal.

Le système hexadécimal représente les nombres en base 16.

On n'a pas besoin de faire des conversions mathématiques, simplement se rappeler que les valeurs 10 à 15 on emploie les lettres A à F.

Chiffres hexadécimaux: **0 1 2 3 4 5 6 7 8 9 A B C D E F**

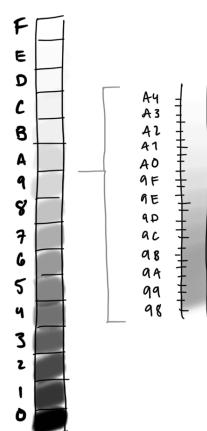
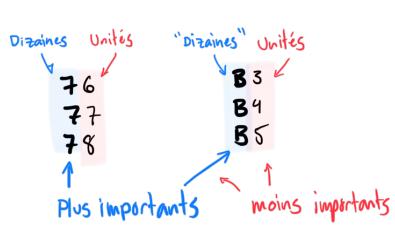
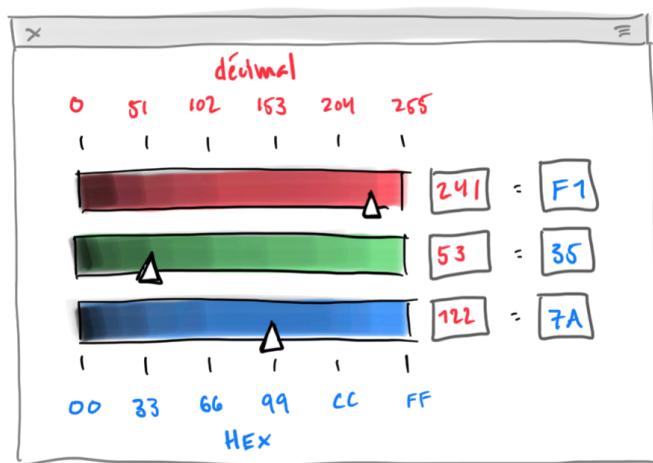
dans le système décimal, $10 \times 10 = 100$

dans le système hexadécimal, $16 \times 16 (\text{f} \times \text{f}) = 256$

Les couleurs hexadécimales sont précédées d'un croisillon ou hashtag #, en suite **6** valeurs écrites dans le format **RR VV BB** sans espaces:

color: #F1357A;

Couleur Hexadécimal



Hex court (3 chiffres)

La notation hexadécimale peut encore être raccourcie, en perdant un peu de précision, avec **3** valeurs RVB en lieu de 6.

Les 3 valeurs sont simplement 'répétées': #123 → #112233

```
background-color: #F37;  
/* équivalent à #FF3377 */
```

F1 35 7A



F37
FF 33 77



Hexadécimal avec opacité (4 ou 8 chiffres)

La couleur hexadécimale peut aussi être spécifiée avec transparence en utilisant l'annotation **#RRGGBBAA** ("AA" = alpha = transparence)

00 = transparent

FF = opaque

```
color: #F1357A7F;  
/* 7F = 50% opacité */
```

La couleur hex avec opacité raccourcie à 4 chiffres: background-color: #F377;

RR GG BB
#F1 35 7A

(3) # R G B # R G B A (4)

(6) # RR GG BB # RR GG BB AA (B)

"A" = alpha = transparence

HSL()

Le modèle HSL (“Hue, Saturation, Luminosity” – *Teinte, Saturation, Luminosité*) permet une manipulation des couleurs plus proche aux besoins humains et une logique unitive lors de la conception.



La teinte est spécifiée avec un chiffre entre 0 et 360, étant la position sur la roue chromatique, avec ou sans l'explicite unité “deg” pour degrés.

La saturation est notée en pourcent, 0% étant gris et 100% étant une couleur pure.

La luminosité est exprimée en pour cent, 0% étant noir, 100% étant blanche.

```
color: hsl(338, 87%, 58%);
```

```
color: hsl(338deg, 87%, 58%);
```

Ainsi des variantes claires ou obscures d'une couleur ‘theme’ peuvent être composées facilement avec une modification de la valeur de luminosité.

Attention, le sélecteur de couleurs “HSB” de la suite Adobe n'est pas compatible avec le système “HSL” du CSS – les valeurs de teinte sont les mêmes, mais le calcul de saturation et luminosité n'est pas pareil!

HSLA() – avec opacité

```
background-color: hsla(338, 87%, 58%, 0.5);
```

Couleurs nommées

PapayaWhip, DarkSlateGray, DarkOliveGreen, LightGoldenrodYellow, LightCoral, HotPink, MediumPurple, ...etc.

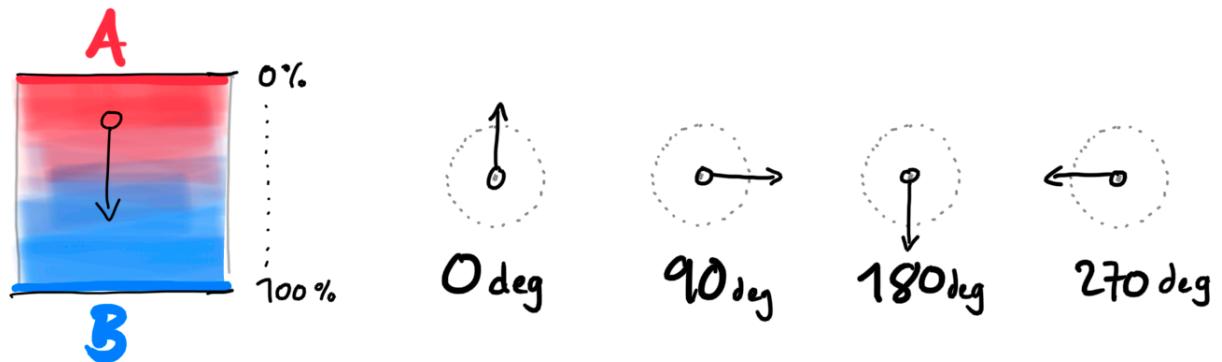
<https://developer.mozilla.org/en-US/docs/Web/CSS/named-color>

linear-gradient – Dégradé de couleurs

Le dégradées `linear-gradient` en CSS sont utilisés avec la propriété `background`.

Forme simple avec 3 valeurs: **orientation**, couleur **A**, couleur **B**:
`background: linear-gradient(angle, A, B);`

`background: linear-gradient(45deg, red, blue);`



Le syntaxe peut être enrichi avec plusieurs couleurs, la position des couleurs sur l'axe du dégradé, les positions de 'mi-chemin' des transitions, etc., etc...

- `background: linear-gradient(red, blue);`
- `background: linear-gradient(45deg, red, blue);`
- `background: linear-gradient(pink, green, yellow);`
- `background: linear-gradient(-45deg, khaki, lime, plum, salmon);`
- `background: linear-gradient(blue, 20%, red);`
- `background: linear-gradient(blue 40%, red 60%);`

<https://developer.mozilla.org/fr/docs/Web/CSS/gradient/linear-gradient>

https://developer.mozilla.org/fr/docs/Web/CSS/Images/Using_CSS_gradients

Bonus: Variables!

```
:root { --myColor: #f1357a; }
body { color: var(--myColor); }

html { --HotPink: 241, 53, 122; }
.chaton { background-color: rgba(var(--HotPink), 0.5); }
```

Typographie sur le web & Webfonts

developer.mozilla.org/fr/docs/Learn/CSS/Styling_text/Web_fonts

Fontes locales – “websafe”

Sans utilisation de webfonts, seules les polices de caractère *installées sur l'ordinateur du visiteur* (“côté client”) sont disponibles pour l'affichage d'une page. Ceci limite sévèrement le choix à l'intersection commune des polices installées par défaut et partout: ***Verdana, Georgia, Helvetica, Times ... Beurk.***

Font stack

Sans certitude de quelles polices sont disponibles, on spécifie une liste de préférence décroissante: “Utilisez le *Neue Haas Grotesk* de préférence, sinon *Helvetica*, sinon *Arial*, sinon un *sans-serif* quelconque”.

`font-family: "Neue Haas Grotesk", Helvetica, Arial, sans-serif;`

Les catégories génériques serif sans-serif monospace cursive fantasy sont le dernier recours à “n’importe quelle typographie de cette catégorie disponible”.
(Bonus: la fonte générique `system-ui` (sur Mac) appelle le *San Francisco*: un sans-serif très complet)

Webfonts

La technologie CSS de webfonts, disponible depuis 2011 environ, permet un site web d’inclure une police de caractère dans la page, sans qu'il soit installé par l’utilisateur.

Formats

Les technologies des webfonts ont été développées pour une optimisation d'affichage typographique dans le contexte web (poids de fichier, rapidité, optimisation pour écran, etc.)

Aujourd’hui les formats **Web Open Font Format**: `.WOFF` et `.WOFF2` sont aujourd’hui largement répandus comme standard.

Les formats webfont précédents de `.EOT` et `.SVG` ne sont plus d’actualité, mais sont parfois présents pour assurer une arrière-compatibilité avec les anciens navigateurs.

Les fontes “desktop” utilisable avec nos logiciels de PAO sont en format Open Type Format OTF, ou True-Type TTF qui peuvent fonctionner dans un projet web, mais ne sont peut-être pas optimisées pour l'affichage dans un navigateur.

Les fontes variables souvent (encore) déployées en format TTF avec (peut-être) une optimisation pour le web.

Hébergement

Self Hosted

Si vous êtes en possession des fichiers fontes (.woff), vous pouvez les utiliser directement sur un site web, il suffit d'inclure les fichiers dans votre projet web et les configurer correctement en CSS (@font-face...) – sans dépendance, abonnement, ou service externe.

Cloud Hosting

Des services de diffusion tels que Google Fonts proposent le déploiement des webfonts "préemballés" sans besoin de rajouter des fichiers dans votre site, il suffit de rajouter quelques lignes de code dans votre page pour charger ces ressources 'a distance'.

Les fonderies et créateurs de typographies commerciaux proposent souvent leurs propres services de cloud hosting (p.ex. typography.com/webfonts, fonts.adobe.com) avec des abonnements récurrents et souvent limités en le nombre de visites par mois, sur un seul site web par licence, etc.

Le fonctionnement *technique* des fichiers fontes, les réglages @font-face, etc. sont identiques au modèle self-hosted, simplement que les fichiers sont hébergées sur le serveur de quelqu'un d'autre.

L'avantage du 'Cloud Hosted' étant une facilité d'emploi, et parfois une optimisation technique qui dépasse les capacités *DIY*; les désavantages étant que vous n'avez pas la maîtrise des éléments, et le bon fonctionnement typographique de votre site dépend de disponibilité et bonne volonté d'un autre fournisseur.

Licences

Les fonderies séparent couramment leurs licences commerciales en "desktop" (pour la PAO) et "web" pour utilisation sur internet. Si les fontes OTF sont techniquement utilisables dans un projet web, les licences commerciaux l'interdisent dans la plupart de cas: une licence supplémentaire pour un déploiement web est souvent requise.

(On peut donc techniquement utiliser une police 'desktop' OTF pour une maquette/prototype en local, en attendant la validation du projet pour l'achat d'une licence web et fichiers WOFF...)

Utilisation webfonts en 3 étapes:

1. **Installation:** Un fichier webfont est inclus dans le dossier du projet:

myFont.woff

2. **Activation:** Une règle @font-face est déclarée dans le CSS:

```
@font-face {  
    font-family: "My Super Font";  
    src: url("myFont.woff");  
    font-weight: 400;  
    font-style: normal;  
}
```

3. **Utilisation:** Le typo est spécifié pour les éléments souhaités avec font-family:

```
body { font-family: "My Super Font"; }
```

① Installer



② Activer

@font-face {

```
    font-family : "myFont";  
    font-weight : 400 ;  
    font-style : normal ;  
    src : url ("font.woff"); ← 1.  
}
```

④ ... Répéter @font-face pour chaque .woff ...
 src:

③ Utiliser

```
body { font-family : "myFont" ; } ← 2.
```

@font-face

Une règle CSS @font-face donne un spécifie "quand le CSS indique l'utilisation de tel 'font-family' avec tels paramètres de graisse, etc., voici le fichier source à charger."

Les familles de typo sont souvent composées de nombreux fichiers fontes individuels: *normal, italic, light, light-italic*, etc...

Plusieurs règles @font-face doivent être écrites, une pour chaque fonte individuelle, avec les paramètres **style** et **weight** correspondant.

Chaque règle @font-face contient 4 déclarations importantes:

Family *font-family: "My Font";*

Indique le nom qu'on souhaite utiliser pour ce typo.

Ce nom est arbitraire, à votre libre choix, ce n'est pas lié à un paramètre technique – vous pouvez donner le nom que vous voulez, mais le *même* nom doit figurer dans l'utilisation du font, ailleurs dans le CSS.

Source *src: url("myFont.woff");*

Indique le chemin ou URL vers le fichier fonte.

On peut spécifier plusieurs **src**, séparé par des virgules pour gérer de différents formats du même fonte, si besoin y est pour compatibilité avec les différents (anciens) navigateurs.

```
src: url("MyFont.woff2") format('woff2'),  
     url("myFont.eot") format('EOT');
```

Style *font-style: normal;*

Spécifie si la fonte correspond au *italic* ou *normal*.

Weight *font-weight: 400;*

Spécifie la graisse de typo, soit avec les mots clés *normal* ou *bold*, soit avec les chiffres de 100 à 900. **400 = normal 700 = bold**

font-weight :

100	"Thin"
200	"Extra Light"
300	"Light"
→ 400	<u>normal</u>
500	"Regular"
600	"Semi - Bold"
→ 700	<u>Bold</u>
800	"Extra Bold"
900	"Black"

Variable fonts

Exemple:

```
@font-face {  
    font-family: 'MyVarFont';  
    src: url('JetBrainsMono-var.ttf') format('truetype');  
    font-weight: 100 800; /* variable weight... */  
}  
body {  
    font-family: 'MyVarFont', monospace;  
    font-weight: 345;  
}
```

Dans cet exemple, le font *JetBrainsMono-var.ttf* est variable sur l'axe de graisse, donc la paramètre `font-weight`: contient deux valeurs pour spécifier la plage disponible: de **100** à **800**.

Axes

Les variable fonts sont variables sur des axes. Des axes standards tels que la graisse ou la largeur sont courants, et des axes uniques (et parfois fantastiques) peuvent être inventés selon la volonté de la créatrice de typo.

Axes officiels

- Weight: **wght** | `font-weight: 123`
- Width: **wdth** | `font-stretch: 95%`
- Italics: **ital** | `font-variation-settings: 'ital' 1;`
- Slant: **slnt** | `font-variation-settings: 'slnt' 10;`
- Optical Size: **opsz** | `font-variation-settings: 'opsz' 8;`

Custom axes

`font-variation-settings: font-variation-settings: 'axis' value;`

References

Web Fonts

developer.mozilla.org/fr/docs/Learn/CSS/Styling_text/Web_fonts

Variable Fonts

developer.mozilla.org/fr/docs/Web/CSS/CSS_Fonts/Variable_Fonts_Guide

web.dev/variable-fonts

Catalogues / Collections

v-fonts.com Open source: v-fonts.com/licenses/open-source

uncut.wtf

theleagueofmoveabletype.com

fontlibrary.org

Guides

beautifulwebtype.com

typewolf.com

webtypography.net

betterwebtype.com

(PDF) [Stop Stealing Sheep, Erik Spiekermann](#)

CSS & Typographie avancé

- "Microtypographie"
- UTF
- HTML Entité (*entity*)
 - & ... ;
 - <
 - >
 - &
 - nbsp
 - emdash
 - <https://html.spec.whatwg.org/multipage/named-characters.html>

Web design is 95% typography

Propriétés CSS

- font-size
- line-height
- letter-spacing
 - "milli-em" - letter-spacing: -0.01em
- word-spacing
- text-indent
 - "hanging punctuation"
- text-align
 - ! césure (hyphenation) → text-align: justify; → lézarde
- text-rendering
 - optimizeSpeed
 - optimizeLegibility
 - <https://css-tricks.com/almanac/properties/t/text-rendering/>

OTF options

- font-feature-settings
 - font-feature-settings: "optn", "truc";
 - <https://rsms.me/inter/lab/>
 - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Fonts/OpenType_fonts_guide

Variable fonts

Variation axis – axe de variation

`font-variation-settings`

```
font-variation-settings: "wght" 375, "GRAD" 88;
```

- https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Fonts/Variable_Fonts_Guide
 - <https://variablefonts.io>
 - <https://variablefonts.io/about-variable-fonts>
 - <https://variablefonts.io/implementing-variable-fonts>
 - <https://variablefonts.io/variable-font-resources>
 - <https://variablefonts.typenetwork.com>
 - <https://www.axis-praxis.org/>
 - <https://v-fonts.com>
 - <https://v-fonts.com/licenses/open-source>
 - <https://play.typedetail.com>
 - <https://css-tricks.com/getting-the-most-out-of-variable-fonts-on-google-fonts/>
-

- <https://v-fonts.com/fonts/roboto-flex>
- <https://github.com/googlefonts/roboto-flex>

```
@font-face {  
    font-family: "robotnik";  
    src: url("RobotoFlex.ttf");  
}  
h1 {  
    font-family: "robotnik";  
    font-size: 10rem;  
    font-weight: 666;  
    letter-spacing: -0.02em;  
    font-variation-settings: "XTRA" 323, "YTLC" 570, "YTUC" 528, "YTAS" 649;  
}
```

Responsive – Media Query

- Format de page A4 = 210×297 mm
- Format de page Web = $\text{?} \times \infty$
(largeur inconnue, hauteur infinie)

Media query (“requête média”)

- “if” – bloc CSS “conditionnelle” – réglages à appliquer selon la taille de fenêtre.
 - “Si la fenêtre est plus grande que tel...” (... ou plus petit que...)
 - certaines règles s’appliquent dans certaines conditions
- Mot clé: **@media** suivi de **(condition)** et **{block}**
- Si le **(condition)** est vraie, le **{block}** est exécuté.
- 2 options principaux: **min-width** et **max-width**
- **min** “la fenêtre fait **au moins** taille N”:
 - **min-width** pour des fenêtres qui sont ‘au moins de cette largeur’
 - version **desktop**
- **max** “la fenêtre fais **jusqu'à** taille N”
 - **max-width** pour des fenêtres ‘jusqu’à cette largeur’
 - version **mobile**
- Le block du media query contient des block de CSS ‘normale’, une situation de “block dans un block” – attention aux accolades imbriquées!
 - query { block { block } }
- Le ou les “**breakpoint**” (*point d'arrêt*) sont les valeurs (min ou max) autour desquelles la mise en page va basculer.
- Approche “mobile first” ou “desktop first” (“Progressive Enhancement”/“Graceful Degradation”)
- Le media query n'est pas la **seule** approche pour réaliser une mise en page “responsive”; Les réglages **flex**, **flex-wrap**, et **max-width** sont toujours utiles.
- On peut déployer un nombre illimité de blocs **@media**, selon besoin. Si multiples réglages **desktop** et réglages mobiles sont utilisés, une stratégie de structure claire accompagnée de multiples commentaires utiles peut aider à apprivoiser le chaos!
- Les conditions peuvent être enchainées ou combinées avec **and** / **or** / **not**
 - **@media (min-width: 600px) and (max-width: 900px) { ... }**
- La fonction **@media** peut être utilisée pour d'autres fonctions d'adaptation selon le contexte.
 - version ‘Print’: **@media print { ... }**
 - version ‘Dark-mode’: **@media (prefers-color-scheme: dark) { ... }**

Code HTML meta viewport

```
<meta name="viewport" content="width=device-width, initial-scale=1">  
necessaire pour un fonctionnement CSS responsive correcte.
```

Exemple code CSS

Mobile first

```
/* paramètres généraux, version mobile */  
  
.body { font-family: sans-serif; }  
  
- - - - -  
  
/* réglages pour desktop */  
  
@media (min-width: 640px) {  
    .row { display: flex; }  
    .row > * { flex: 1; }  
}  


---


```

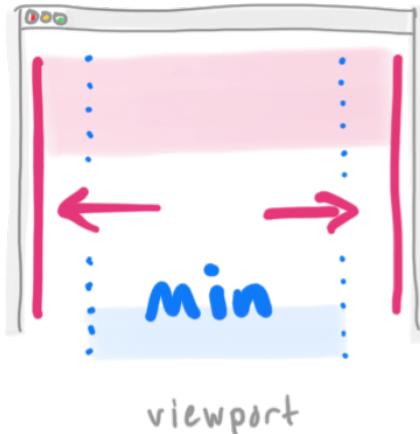
Desktop first

```
/* paramètres généraux, version desktop */  
  
body { font-family: sans-serif; }  
.row { display: flex; }  
.row > * { flex: 1; }  
  
- - - - -  
  
/* réglages pour mobile */  
  
@media (max-width: 640px) {  
    .row { flex-direction: column; }  
    /* .row { display: block; } */  
}
```

Référence

developer.mozilla.org/fr/docs/Learn/CSS/CSS_layout/Responsive_Design

| @media |

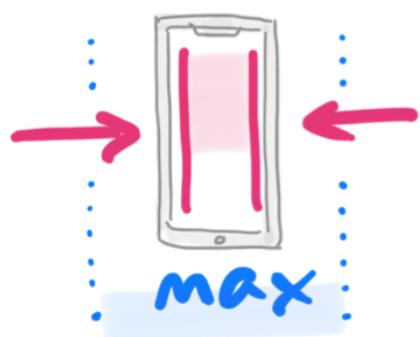


min-width :

"au moins..."

(desktop)

si la fenêtre fait au moins 640 px ...



max-width :

"jusqu'à..."

(mobile)

si la fenêtre fait jusqu'à 640 px ...

media query
= une question



Si oui :

@media (min-width : 640px) {

.row {

display: flex;

} /* affichage desktop */

}

si non,

...

Ressources / images / médias / résolution / optimisation

Images bitmap / matricielle / pixel

- "Résolution" = **nombre de pixels** (rangs et colonnes)

combien de pixels?

- mégapixels: nombre de pixels total ($w \times h$)

- "Résolution" = **densité des pixels** (72 ou 300 "dpi")

quelle taille sont les pixels?

- Pixel Par Pouce / Pixel Per Inch / Dots Per Inch

Affichage / Écrans

- Anciens écrans & navigateurs web

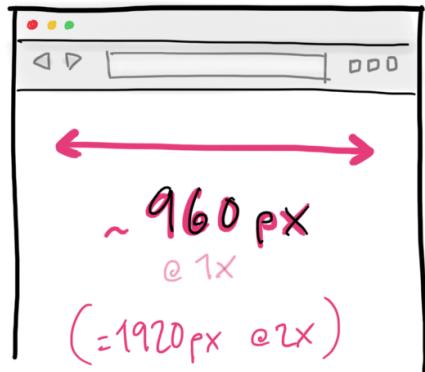
- écrans bas déf, gros pixels, environ 100ppi
- système analogique, jamais exact, réelle résolution exacte toujours inconnue
- peu importe la réelle résolution physique et taille de l'écran, les systèmes d'exploitation ("OS") *prétendaient* que tous les écrans étaient de 72dpi (Mac) ou 96dpi (Windows)
- affichage images sur le web: 1 pixel-image = 1pixel-écran
(les images n'étaient *jamais* ré-échantillonnées à l'affichage)
- unité CSS 1px = 1 pixel-écran (*peu importe la taille réelle*)
- tout était simple, un px = 1 pixel, tout le design-écran était fait dans la même matrice

- Nouveaux écrans & navigateurs web – **Retina ou HighDPI**

- écran "Retina" ≥ 200 dpi – pixels invisibles à l'œil nu
- 1 pixel écran n'est plus une mesure utile: variation de résolution entre les écrans trop importante (de ~100 dpi non-retina, à 460 dpi pour un téléphone)
- 1px \neq 1 pixel
- unité **CSS 1px = 1/96 inch**
(0.264583mm), 1cm = 37.795 px)
- les nouveaux ordinateurs plus puissants et les navigateurs web plus sophistiqués sont capables de redimensionner les images et garder une bonne qualité
néanmoins, les ordinateurs ne sont pas magiques, on ne peut toujours pas inventer des détails qui ne sont pas enregistrés – si on agrandit une image "trop", ça devient flou et/ou de mauvaise qualité!

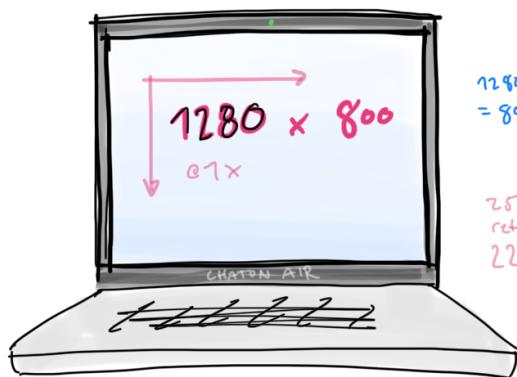
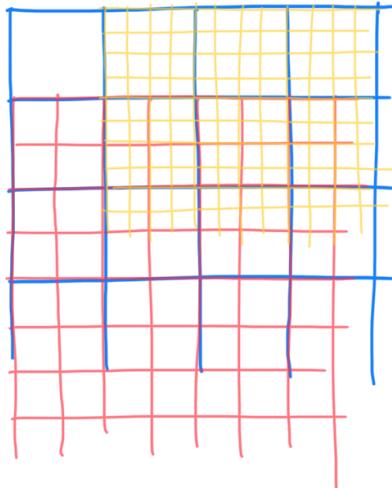
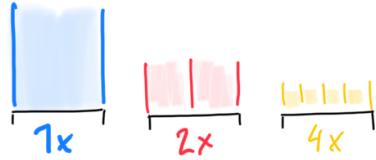
@2x

- premiers écrans "retina" étaient de "double" résolution → "@2x"
- "@1x" = nombre de pixels "logique", ou équivalent "ancien écran"



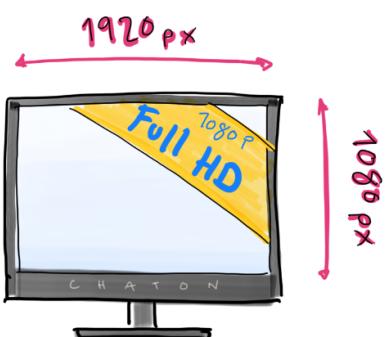
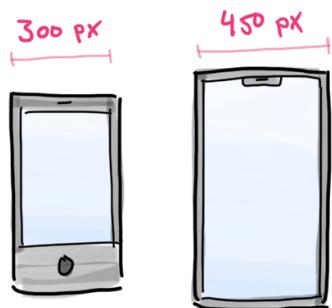
= 60 rem

$960 / 16$



1280 px
 $= 80 \text{ cm}$

2560×1600
retina 2x pixels
227 ppi



1080 px

Optimisation des fichiers

- but: réduction en "poids" de l'image (données en Bytes / KB, MB)
 - Bande passante: chargement de page plus rapide, plus efficient, etc.
 - Poids optimal? Au pif: **max 1 MB** (pour une image "grand")
- réduction du **nombre de pixels**
 - une image 1/2 dimensions pixel = 1/4 le nombre de pixels total = 25% de poids!
- **compression JPEG**
 - réduction facilement à 10% du poids "original" non compressé (par rapport à un TIFF par exemple)
 - compression avec perte: les informations sont perdues et échangées contre des pixels bizarres qui sont plus légers. Trop de compression entraîne des pertes de qualité – la recompression (ouvrir et re-enregistrer un JPEG) est particulièrement néfaste!

Taille des images

- utiliser les "responsive images"
(images élastiques qui redimensionnement selon l'espace disponible)
en CSS: `img { max-width: 100%; }`
- optimiser la taille de l'image selon sa mise en page finale, importance, etc.
- garder l'image originale (PSD), à la résolution originale,
dans un dossier à part (dossier "projet") hors du dossier "site";
puis exporter des JPEG réduits, de taille optimisée, selon besoin
 - exporter plusieurs tailles! p.ex.
 - `image01_big.jpg` (1920 pixels de large: ~800KB)
 - `image01_icn.jpg` (480 pixels de large: ~60KB, 16x plus léger!)

Quelle taille?

- HD vidéo: **1920 × 1080 px** ← mnémonique facile!
- Fenêtre navigateur desktop, largeur "normale" de **960 px à 1280 px** @1x
 - = 60 à 80 rem
- Écran MacBook Air Rétina 13" = **1280 × 800 px** @1x
 - pixels-écran = 2x
- Téléphone, orientation portrait
 - iPhone SE = **320 px** (pixels-écran = 2x)
 - iPhone 13 Pro Max = **428 px** (pixels-écran = 3x)
- Échelle simple "au pif":
 - 100% largeur fenêtre en qualité retina = environ 1920 pixels (qualité "HD")
 - "toute la largeur" de la fenêtre: environ 2000 pixels
 - "la moitié" de la fenêtre: environ 1000 pixels
 - "un quart" de la fenêtre: environ 500 pixels

Formats de ressources utilisables dans une page web

• SVG

- **vectoriel**
- éditable en texte
 - peut aussi être inclus dans le code source HTML
- Illustrator: Exporter → SVG (“responsive” oui, “minifier” non)
- dessin vectoriel toujours préférable aux images matricielles si possible:
 - faible poids de fichier, chargement rapide
 - échelle / qualité sans limite
- le format SVG peut contenir de texte, image bitmap et autres ressources complexes
(à l'encontre de l'utilité d'une ressource rapide et légère ...)
 - sauf cas exprès, utiliser la fonction “vectoriser le texte” avant export
 - vérifier le code et le poids de fichier exporté, selon effets ou fonctions utilisés dans l'original

• JPEG

- pour images **photographiques** ou images avec détails complexes
- compression avec perte
 - échelle de compression variable, aucune valeur standard (“85%” = ??)
 - rapport final qualité visuelle / poids dépend de l'image et ses détails, grain, etc.
- Photoshop: Exporter ...

• PNG

- compression sans perte, pour images avec *aplats de couleur*
- pixels peut avoir transparence variable
- pas optimale pour photo
- (SVG préférable si original vectoriel)

• GIF

- animation
- compression sans perte, pour images avec aplats de couleur
- pixels peut-être transparents (1bit)
- palette couleur très limitée, rendu visuel souvent assez mauvais, pas optimal pour photo

• Formats non compatibles

- PDF (oui, mais en “fichier à télécharger” pas comme ressource de page à afficher)
- PSD (garder les fichiers “source” dans un dossier projet à part!)
- Camera raw: DNG / CR2 / NEF / ...
- TIFF (→ PNG)
- EPS (→ SVG)

Le profile colorimétrique général pour le web est sRGB.

(Pour les images sans profil incorporé, les navigateurs présument un profil sRGB.)

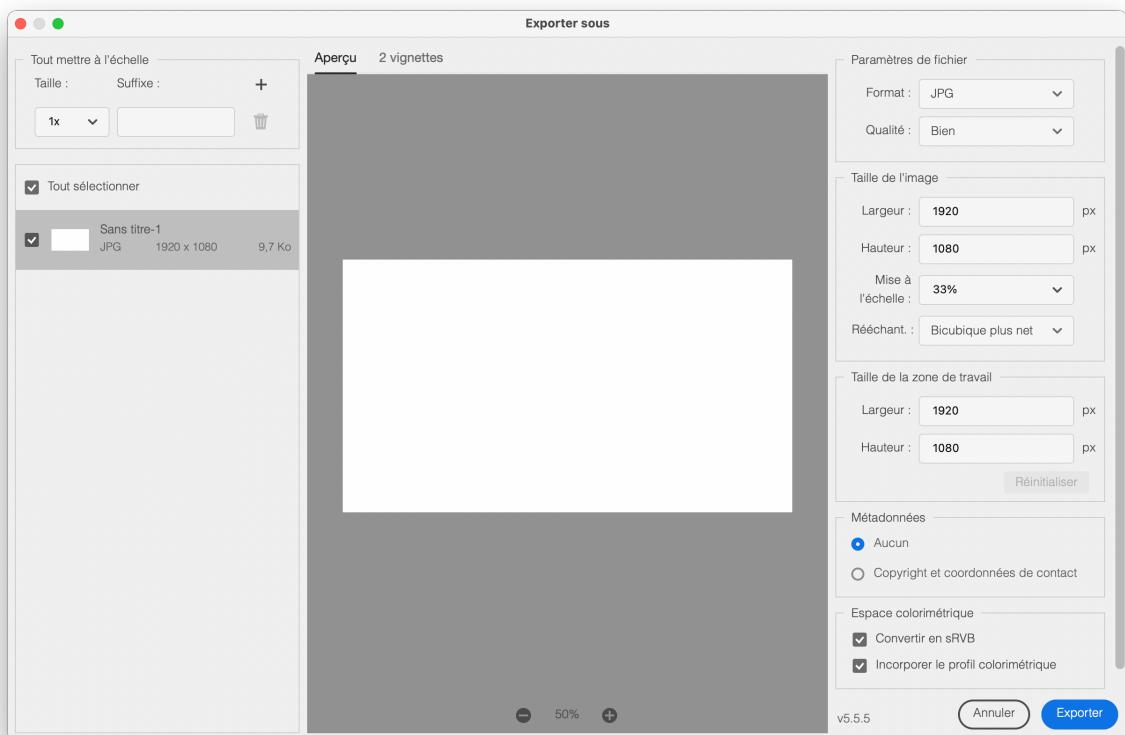
Vidéo

- compression h264
- format .mp4

Photoshop

Fichier → Exporter sous ...

- Format (jpeg)
 - Qualité compression ("nikel" ou ~90%)
- Taille / largeur (selon mise en page)
- Convertir en sRGB

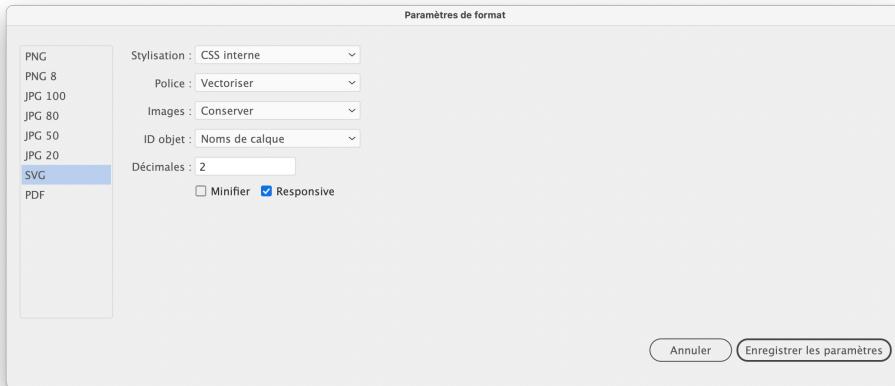
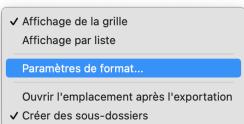
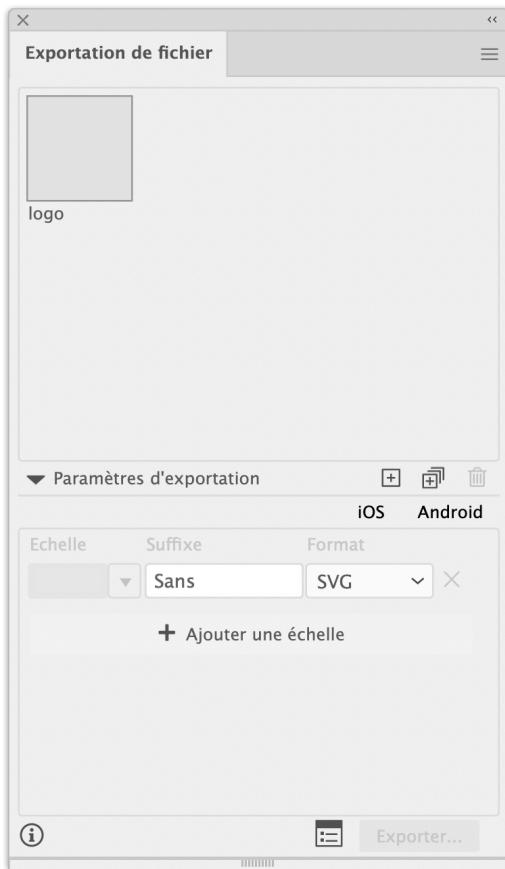


Illustrator

Panneau Exportation de fichier

Options: Paramètres de format...

SVG: Police: vectoriser Minifier: non, Responsive: oui



URL (Uniform Resource Locator)

`https://example.com`

Protocol
protocole

Domain
domaine

TLD (Top-Level Domain)
Domaine de premier niveau

`https://example.com/index.html`

Protocol
protocole

Domain
domaine

TLD

File
fichier

Path
chemin

`https://www.example.com/pages/autre/sous-dossier/page.html`

Protocol
protocole

Sub-Domain
sous-domaine

Domain
domaine

TLD

Path
chemin

File
fichier

`file:///Users/JeanNemar/Documents/Projet/site/test.html`

Protocol
protocole

Path
chemin

File
fichier

Local user software Internet DNS Server

`https://kittens.example.com/images/search.php?animal=chaton&color=roux`

Protocol
protocole

Sub-Domain
sous-domaine

Domain
domaine

TLD

Path
chemin

File
fichier

Query string
chaîne de requête

Transition & Animation

CSS Transitions

Transition entre les états d'un élément. (→ pseudo-class :hover, :active, ...)

```
a {  
    color: chartreuse;  
    background-color: papayawhip;  
    transition: all 0.5s ease;  
    /* transition: background-color 0.5s ease; */  
  
}  
  
a:hover {  
    color: honeydew;  
    background-color: darksalmon;  
}
```

transition:
transition-property:
transition-duration:
transition-timing-function:
transition-delay:
transition-behavior:

transition: [transition-property] [transition-duration] [transition-timing-function] [transition-delay];

<https://css-tricks.com/almanac/properties/t/transition/>

https://developer.mozilla.org/fr/docs/Web/CSS/CSS_transitions/Using_CSS_transitions

<https://developer.mozilla.org/fr/docs/Web/CSS/transition>

CSS Animation

@keyframes

Définition d'un scénario d'animation avec des keyframes.

```
element { animation: chaton 5s alternate infinite; }

@keyframes chaton {
    0% { color: hotpink; }
    100% { color: cornflowerblue; }
}
```

animation:

animation-name:
animation-duration:
animation-timing-function:
animation-delay:
animation-direction:
animation-iteration-count:
animation-fill-mode:
animation-play-state:

animation: [animation-name] [animation-duration] [animation-timing-function] [animation-delay] [animation-iteration-count] [animation-direction] [animation-fill-mode] [animation-play-state];

<https://css-tricks.com/almanac/properties/a/animation/>

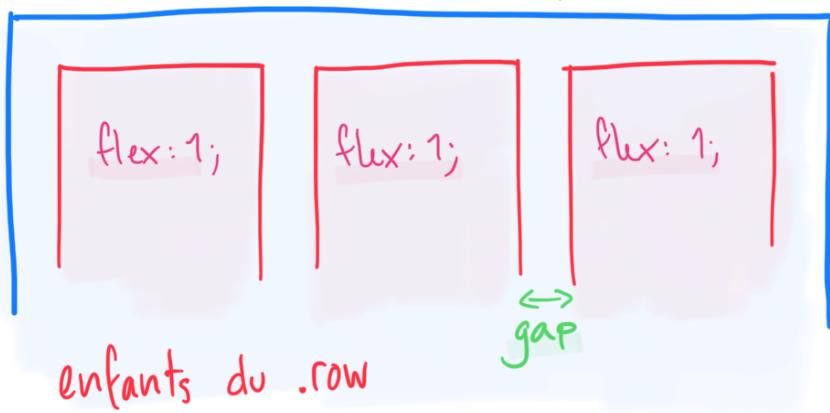
https://developer.mozilla.org/fr/docs/Web/CSS/CSS_animations

https://developer.mozilla.org/fr/docs/Web/CSS/CSS_animated_properties

Super easy columns with Flex

.row

display: flex;



.row {

display: flex;

gap: 1rem;

}

.row > * {

flex: 1;

}

/* ("row" ou "chaton" ou "machin",..) */

Super easy columns with Flex

```
.row {  
    display: flex;  
    gap: 1rem;  
}  
.row > * { flex: 1; }
```

```
<section class="row">  
    <div> bla bla 1er colonne </div>  
    <truc> bla bla 2e colonne </truc>  
    <div> bla bla 3e colonne </div>  
</section>
```

Utilisation

Appliquer le `class="row"` sur n'importe quel élément d'un document HTML afin que ses enfants s'affichent en colonnes.

- Vous pouvez employer un autre class & sélecteur que `.row`, selon votre logique
 - Regrouper des enfants dans des blocs pour les mettre ensemble dans la même colonne
-

Pour un affichage **responsive**/adapté mobile, ajouter la déclaration:

```
@media( min-width: 768px ){  
    .row { flex-direction: column; }  
}
```

Décryptage

.row un nom de classe arbitraire qui est appliqué sur l'élément **parent** des colonnes.

display: flex; le mot clé pour appliquer le système *Flexbox* sur un élément parent.

gap: 1rem; le paramètre qui gère la largeur des *gouttières* entre les colonnes

> * le sélecteur universel `*` qui sélectionne n'importe quel élément; combiné avec le sélecteur enfant direct `>` qui sélectionne les enfants directs d'un élément, mais pas plus loin. Ensemble, ils ciblent les enfants directs du `.row`, peu importe leur type.

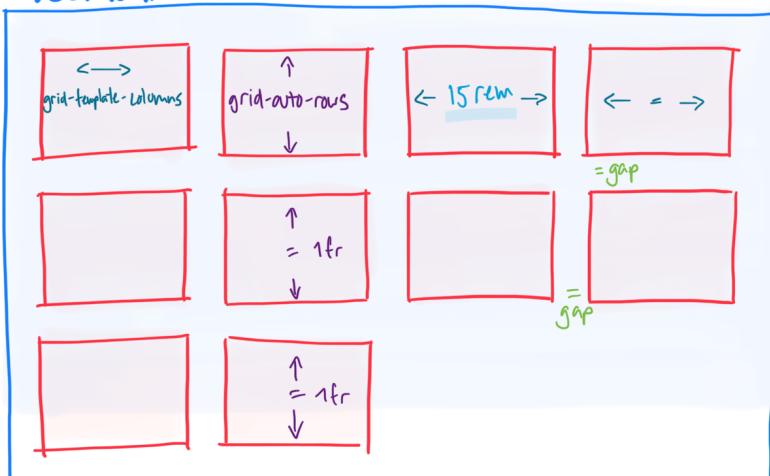
flex: 1; le paramètre de taille pour les enfants en mode flex.
Appliquée à tous les éléments enfants, ils seront tous de taille égale.

Cette déclaration est un raccourci pour la déclaration `flex: 1 1 auto;`

Cette déclaration est elle-même un raccourci pour `flex-grow:1; flex-shrink:1; flex-basis:auto;`

Super easy boxes with Grid

.container



.container {

display: grid;

grid-template-columns:

repeat(auto-fill,
minmax(
min(15rem, 100%), 1fr

);)

grid-auto-rows: 1fr;

gap: 1rem;

}

.container {

display: grid;

gap: 1rem;

grid-template-columns: repeat(auto-fill, minmax(min(15rem, 100%), 1fr));

grid-auto-rows: 1fr;

}

Utilisation

- Appliquer un le class `container` sur n'importe quel élément d'un document HTML afin que ses enfants s'affichera en grid adaptive.
 - Ou, modifier le sélecteur `.container` selon besoin
 - Modifier la valeur `15rem` pour adapter la taille des cases du grid
 - Le choix d'unité et valeur est libre selon besoin
 - Cette valeur sera la taille *minimum* des cases, qui grandiront (un peu) pour remplir équitablement l'espace disponible
 - La valeur `gap` gère la taille des gouttières entre les cases
 - La déclaration `grid-auto-rows: 1fr;` assure que toutes les rangées soient de la même hauteur: les rangées horizontales du grid s'adaptent à la hauteur du contenu de la case la plus remplie.
 - Pour régler manuellement la hauteur des cases, ajouter la déclaration:
`.container > * { min-height: 15rem; }`
 - Ou, remplacer la déclaration de min-height avec un rapport largeur/hauteur:
`aspect-ratio: 16/9;`
-

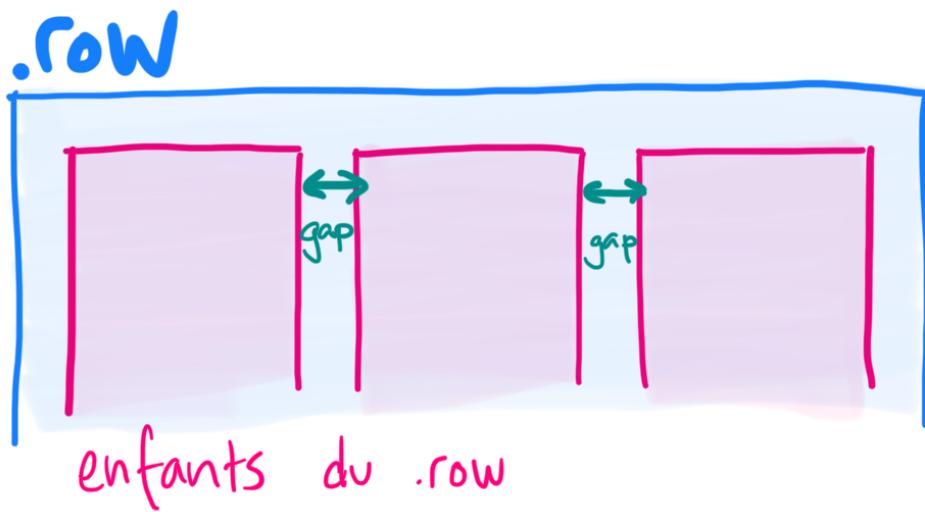
Décryptage:

- **.container** un nom de classe arbitraire qui est appliqué sur l'élément **parent** des boxes
 - **display: grid;** le mot clé pour appliquer le système *Grid* sur un élément parent
 - **grid-template-columns:** la disposition des colonnes (les pistes ou "tracks")
 - **repeat** toutes les pistes/tracks sont de la même taille
 - **autofill** créer autant de pistes/tracks que nécessaire
 - **minmax(A,B)** → = une valeur entre A and B
 - **minmax(15rem, 1fr)** → = une valeur entre 15rem et jusqu'à 1fr
 - **1fr** = une unité de part égale de l'espace disponible
 - *Si l'espace disponible est plus petit que 10rem, version mobile? Solution: min()*
 - **min(A,B)** → = une valeur A ou B, celui qui est plus petit
 - **minmax(min(A,B), C)** → = une valeur entre le plus petit entre A ou B, jusqu'au max. C
 - **gap: 1rem;** le paramètre qui gère la largeur des gouttières/espaces entre les blocs
 - **grid-auto-rows: 1fr;** toutes les rangées sont de même hauteur
 - **.container > *** les enfants directs du .container
 - **min-height:** hauteur minimum (hauteur max selon contenu)
 - **aspect-ratio: 16/9;** fixer un rapport largeur/hauteur
-

- Source: <https://evanminto.com/blog/intrinsically-responsive-css-grid-minmax-min/>

- https://developer.mozilla.org/fr/docs/Web/CSS/CSS_grid_layout/Basic_concepts_of_grid_layout

Super easy columns with grid



```
.row {  
    display: grid;  
    grid-auto-flow: column;  
    gap: 1rem;  
}
```

```
.row { display: grid; grid-auto-flow: column; gap: 1rem; }
```

