FORMATION INFORMATIQUE

HTML et CSS

Les bases pour construire une page web.
Explique simplement, sans blabla inutile.

DanielCraft - Livre debutant

Sommaire

Clique un chapitre ou un sous-chapitre pour y aller.

CHAPITRE 1

Chapitre 1 - Salut, c'est quoi une page web ?

A gauche la structure. A droite le style. HTML puis CSS.

Une page web, ce n'est pas un document magique tombe du ciel. Ce n'est pas non plus "Internet" en entier. C'est un fichier (ou plusieurs) que ton **navigateur** lit, puis affiche. Tu ecris des instructions. Chrome, Firefox, Edge ou Safari les interpretent. Toi, tu vois un titre, un texte, une image, un bouton. Derriere, il y a du code. Chez DanielCraft, on aime commencer simple : avant de devenir "developpeur", tu apprends a parler deux langues de base du web - **HTML** et **CSS**.

Le HTML, c'est la structure. Les murs, les pieces, les portes. Il dit : ici un titre, ici un paragraphe, ici une image, ici un formulaire. Le CSS, c'est l'apparence. La peinture, les rideaux, l'espace entre les meubles. Il dit : ce titre est grand et vert, ce fond est doux, ce bouton a des coins ronds. Sans HTML, il n'y a rien a regarder. Sans CSS, ca marche, mais c'est souvent moche - texte noir sur fond blanc, zero atmosphere. Les deux marchent ensemble. Toujours. Tu restes le pilote. Le navigateur affiche.

En 2026, quand quelqu'un dit "j'ai fait une page web", il parle le plus souvent de ca : un fichier HTML, un peu de CSS, parfois une image. Derriere, il y a des builders, des CMS, des frameworks. Pour toi, le geste reste le meme : ecrire, sauvegarder, ouvrir, corriger. Tu comprends ce que tu touches. Tu n'es plus spectateur d'un outil opaque.

> **★ A RETENIR**
>
> HTML = structure. CSS = apparence. Navigateur = guide. Toi = auteur.

Ce que ce n'est pas

Ce n'est pas un logiciel a installer pour "faire un site" en cliquant partout. Ce n'est pas non plus **JavaScript** : JS viendra plus tard pour rendre la page interactive. Ce n'est pas "mettre du bleu dans le HTML" : la couleur, c'est le job du CSS. Et ce n'est surtout pas un truc reserve aux genies. Lea, freelance web, reconstruit des pages clients tous les jours avec ces bases. Max, artisan plombier, a fait sa page vitrine avec un titre, trois photos, et un formulaire contact. Sam, enseignant, montre a ses eleves que le web, c'est d'abord de l'ordre et de la clarte.

Ce n'est pas non plus "Internet" magique. Une page locale sur ton disque, c'est deja une page web. Pour la mettre en ligne, il faudra un hebergement - plus loin. D'abord, tu apprends a ecrire. Ensuite, tu partages.

Imagine une maison. Le HTML pose les pieces. Le CSS decide si c'est cosy ou clinique. Le navigateur, c'est le guide qui te fait entrer et te montre le salon. Quand tu ouvres un site, tu lances ce guide. Il lit le code. Il te montre la page. Tu ecris du texte special. Lui, il transforme. Plus loin dans ce livre, on ajoutera balises, liens, images, formulaires, couleurs, boites, Flexbox, responsive. Pas pour te noyer. Pour que tu saches ce que tu manipules.

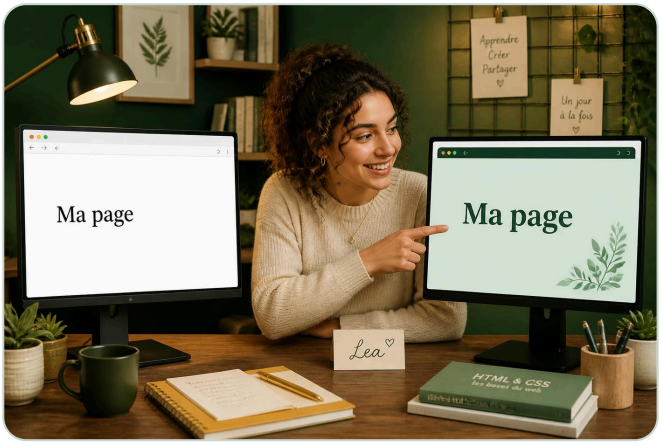
Lea dit : "structure d'abord, style ensuite". Max a grave ca apres avoir melange les deux trop tot. Sam dessine la maison au tableau : murs HTML, peinture CSS, visiteurs = navigateur. Les eleves comprennent en dix minutes ce que des heures de jargon n'expliquent pas.

Ce que tu vas savoir faire

A la fin, tu sauras creer une vraie page web. Tu y mettras du texte, des images, des liens, des listes, un formulaire. Tu changeras couleurs et polices. Tu rangeras les blocs. Tu penseras telephone. Puis un mini-projet, un recap, des ateliers, l'accessibilite, un quiz, et un bravo. Niveau debutant solide. Pas besoin d'etre "tech". Besoin de curiosite et de patience : tu modifies, tu sauvegardes, tu rafraichis. Cinq minutes actives valent mieux qu'une heure passive.

Comment lire ce livre

Lis dans l'ordre au debut. Les premiers chapitres posent le sol. Les ateliers font faire. Le quiz verifie. A chaque fin, il y a un "A toi". Fais-le. Chez DanielCraft, on forme des gens qui livrent petit, souvent, proprement - pas des collectionneurs de tutos oublies. Tu peux revenir ensuite a un chapitre precis (Flexbox, responsive, formulaires) comme a une fiche. Le livre est un atelier, pas une encyclopedie a lire d'une traite sans les mains.



Exemple : sans CSS vs avec CSS, meme contenu.

Petite histoire

Lea devait livrer une page "A propos" pour un fleuriste. Avant, elle ouvrait un builder, cliquait partout, et ne savait plus ou etait le vrai contenu. Maintenant, elle ecrit d'abord le HTML : titre, trois paragraphes, photo, lien contact. Ensuite seulement le CSS : fond creme, titre vert, image arrondie. Quarante minutes, page nette. Le client comprend. Lea assume.

Max, lui, voulait "juste une page avec mon numero". Il a commence par coller du style partout dans le HTML. Ca cassait. Un ami DanielCraft lui a dit : structure d'abord, style ensuite. Il a recommence. Trois soirs. Sa page tient sur telephone. Les clients appellent. Ce n'est pas de la magie. C'est de la methode.

Erreur classique

Croire que HTML sert a rendre une page jolie. Beaucoup cherchent "comment mettre du bleu" dans le HTML. Autre piege : croire que "c'est trop technique pour moi" apres dix minutes. Souvent, le probleme n'est pas toi. C'est le melange trop tot : style, structure, et trop d'outils d'un coup. Commence nu. Habiller ensuite. Autre piege : vouloir un site "pro" des le soir 1. Une page claire avec un titre et deux paragraphes bat une maquette jamais finie.

> ⚠ **ATTENTION**
>
> Ne cherche pas la couleur dans le HTML. Structure d'abord. Le CSS habille apres.

En vrai

Ouvre un site que tu visites souvent. Clic droit, Inspecter (ou F12). Clique une ligne HTML. Repere une regle CSS a droite. Note une chose structuree et une chose stylee. Tu viens de lire du vrai code, sans installer quoi que ce soit. Puis imagine ta propre page en trois mots : sujet, contenu, ambiance. Tu poses le terrain pour le chapitre suivant.

A toi

Ecris en trois phrases : (1) une page que tu aimerais avoir (perso, artisan, classe, hobby), (2) ce que tu veux y mettre comme contenu, (3) ce que tu aimerais rendre joli ensuite. Garde ce papier. On y revient au mini-projet. Chez DanielCraft, ce petit brief vaut plus qu'une heure de tutorials flous.

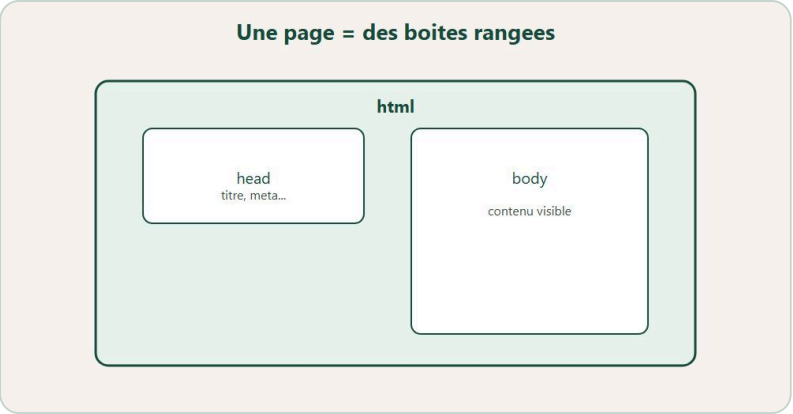
Exemple pour voir la difference

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Structure vs style</title>
  <style>
    body { font-family: sans-serif; background: #f0f0f0; margin: 0; padding: 20px; }
    h1 { color: #1a5f4a; }
    .carte { background: white; padding: 16px; border-radius: 8px; }
  </style>
</head>
<body>
  <div class="carte">
    <h1>Ma page</h1>
    <p>Le HTML dit : c'est un titre, c'est un paragraphe.</p>
    <p>Le CSS dit : fond gris clair, titre vert, carte blanche.</p>
  </div>
</body>
</html>
```

Copie, sauvegarde, ouvre. Tu vois deja la difference entre structure et style. Dans ce livre, on demonte ca piece par piece, avec Lea, Max et Sam comme compagnons de route - et DanielCraft comme fil : petit, clair, testable.

CHAPITRE 2

Chapitre 2 - Ton premier fichier HTML



La page, c'est des boites rangees les unes dans les autres.

On va creer un fichier. Rien de complique. Pas de compte. Pas de carte bleue. Pas de "plateforme magique". Juste un editeur, un navigateur, et toi. Chez DanielCraft, on insiste la-dessus : le premier geste web, c'est sauvegarder un `.html` et l'ouvrir. Quand tu vois ton texte a l'ecran, quelque chose bascule. Tu n'es plus spectateur. Tu es auteur. Lea le vit encore a chaque nouveau stagiaire. Max l'a vecu sur sa page plomberie. Sam le recree en classe chaque annee.

Tu as besoin de deux choses. Un editeur de texte : le Bloc-notes marche, VS Code (gratuit) est plus confortable. Un navigateur : celui que tu utilises deja. C'est tout. Lea travaille dans VS Code parce que la coloration aide a voir les balises. Max a commence avec le Bloc-notes et ca a suffi pour sa premiere page artisan. Sam montre les deux a ses eleves : l'outil change, le fichier reste le meme. L'important, c'est le geste : ecrire, sauvegarder, ouvrir, corriger, rafraichir.

En 2026, quand quelqu'un dit "j'ai cree ma premiere page", il parle souvent de ce cycle. Derriere, il y a des generateurs et des CMS. Pour toi, le geste fondateur reste nu : un fichier, un navigateur, un F5. Tu restes le pilote. Le fichier obeit.

> **★ A RETENIR**
>
> Sauvegarde en `.html`, ouvre dans le navigateur, modifie, F5. C'est le cycle de base.

Ce que ce n'est pas

Ce n'est pas encore un site en ligne. Ton fichier vit sur ton ordinateur. Pour le mettre sur Internet, il faudra un hebergement - plus tard. Ce n'est pas un framework. Ce n'est pas "du HTML avance". C'est un squelette : **doctype**, html, head, body, un titre, un paragraphe. Si tu cherches deja les animations et les menus deroulants, tu te disperses. D'abord le fichier. Ensuite le reste.

Ce n'est pas non plus un fichier Word renomme. HTML, c'est du texte brut avec des balises. Pas de bouton "gras" cache. Chaque instruction est visible. C'est une force : tu vois ce que tu fais. Lea dit : "si tu ne vois pas la balise, tu ne controles pas la page".

Pense a une lettre. Le `<head>`, c'est l'enveloppe et les infos pour le facteur (titre de l'onglet, encodage des accents). Le `<body>`, c'est le message que le destinataire lit. Le navigateur ouvre la lettre et l'affiche. Toi, tu ecris. Lui, il montre. Lea dit : "head pour la machine, body pour l'humain". Max retient : "ce que je vois dans l'onglet, c'est le head ; ce que je vois sur la page, c'est le body". Sam dessine l'enveloppe au tableau. Les eleves ne melangent plus les deux.

Cree le fichier

Ouvre ton editeur. Copie le code ci-dessous. Enregistre en `index.html` (attention a l'extension `.html`, pas `.txt`). Double-clique dessus. Ca s'ouvre dans le navigateur.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Ma premiere page</title>
</head>
<body>
  <h1>Salut !</h1>
  <p>Ca y est. J'ai fait une page web.</p>
</body>
</html>
```

Si tu vois "Salut !" en gros, bravo. C'est bon. Si tu vois le code brut au lieu de la page, regarde l'extension du fichier : souvent Windows a ajoute `.txt` en douce. Renomme proprement. Sauvegarde. Rouvre. Le contraste enseigne mieux qu'un paragraphe.

> ⚠ **ATTENTION**
>
> Sur Windows, verifie que le fichier s'appelle bien `index.html` et pas `index.html.txt`. Sinon le navigateur affiche le code brut.

On explique ligne par ligne (sans stress)

`<!DOCTYPE html>` dit au navigateur : c'est du HTML moderne. Toujours en haut. `<html lang="fr">` ouvre la page et precise le francais. `<head> ... </head>` contient les infos pour le navigateur, pas le gros du contenu visible. Exemple : le titre de l'onglet avec `<title>`. `<meta charset="UTF-8">` fait marcher les accents. Crucial en francais. Sans ca, tu peux voir des caracteres bizarres. `<body> ... </body>` est le corps : tout ce que tu vois va ici. `<h1>` est un gros titre. `<p>` est un paragraphe.

Tu n'as pas a reciter ca par coeur. Tu as a reconnaitre le squelette. Lea le tape une fois par projet. Max le copie depuis son premier fichier. Sam le fait reconstruire de memoire en fin de seance.

Les balises ouvrantes et fermantes

Regarde : `<p>` ouvre. `</p>` ferme. Le slash `/` veut dire "c'est fini pour ce truc". Comme des parentheses. Tu ouvres, tu fermes. Si tu oublies une fermeture, la page peut avoir l'air cassee. Respire. Corrige. Rafraichis (**F5**). Chez DanielCraft, le cycle "sauvegarder puis F5" est sacre. Sans lui, tu debogues un fantome.

> ✦ **ASTUCE**
>
> Apres chaque modif : sauvegarde, puis F5. Si tu oublies de rafraichir, tu crois que "ca n'a rien change".

Petite histoire

Lea a perdu vingt minutes la premiere fois parce que son fichier s'appelait `index.html.txt`. Le navigateur affichait le code comme du texte. Elle a demande a un collegue. Il a rit doucement, a montre l'extension, et elle n'a plus jamais fait l'erreur. Max, lui, avait mis le titre dans le body et se demandait pourquoi l'onglet disait encore "sans titre". Il a deplace le `<title>` dans le head. Clique. Compris.

Sam fait parfois l'exercice en classe : deux fichiers identiques, un `.html` et un `.txt`. Les eleves comprennent que le navigateur ne devine pas : il lit ce que tu lui donnes, avec le bon nom. Trois scenes, une lecon : le nom du fichier compte autant que son contenu.

Erreur classique

Enregistrer en `.txt` sans le voir. Oublier le charset et croire que "le francais casse HTML". Mettre tout dans le head. Ou oublier de rafraichir apres une modif et croire que "ca n'a rien change". Toujours : sauvegarder, puis F5. Autre piege : modifier le mauvais fichier (copie sur le bureau vs copie dans le dossier projet). Verifie le chemin avant de paniquer. Lea garde une regle : une seule source de verite, un seul dossier projet.

En vrai

Cree le fichier. Change le texte du `h1` avec ton prenom. Sauvegarde. Rafraichis. Tu viens de modifier un site. Serieux. Puis change le titre de l'onglet. Observe la difference entre ce qui est dans le head et ce qui est dans le body. Ajoute une deuxieme phrase dans un nouveau `<p>`. Tu construis deja une vraie page, meme minuscule. Si le code brut s'affiche, regarde l'extension. Corrige. Relance.

A toi

Ecris une page "Ma premiere page" avec ton prenom dans le h1, une phrase sur toi dans un paragraphe, et un titre d'onglet clair. Note en une ligne ce qui t'a bloque, s'il y a eu un blocage. On debugge plus tard - la, on valide le geste. Garde ce fichier : tu t'en serviras dans tout le livre. Style DanielCraft : petit, clair, testable, un fichier qui existe vraiment.

CHAPITRE 3

Chapitre 3 - Les balises, c'est des etiquettes

Une balise, c'est une etiquette posee sur un contenu.

Une **balise**, c'est une etiquette. Elle dit au navigateur : ce bout de texte, c'est un titre ; celui-la, c'est un lien ; celui-ci, c'est une image. Sans etiquettes, le navigateur voit une soupe de mots. Avec des etiquettes, il construit une page. Chez DanielCraft, on repete souvent : HTML, ce n'est pas "ecrire joli". C'est nommer correctement ce que tu montres. Lea etiquette avant de styler. Max etiquette avant de chercher la couleur. Sam demande le sens avant la taille.

La forme generale est simple. Tu ouvres, tu mets un contenu, tu fermes : `<nom>contenu</nom>`. Exemple : `<strong>Important</strong>` met le mot en gras parce que tu as dit "important", pas seulement parce que c'est plus gros. En 2026, quand quelqu'un dit "j'ai mis des balises", il parle souvent de cette hygiene. Derriere, il y a des centaines de balises HTML. Pour toi, une poignee suffit. Tu restes le pilote. L'etiquette oriente.

> **★ A RETENIR**
>
> Balise = sens. Attribut = detail. Ouvre, contenu, ferme - sauf les balises seules.

Ce que ce n'est pas

Ce n'est pas du CSS. Une balise ne "fait pas joli" par mission premiere : elle donne du **sens**. Ce n'est pas non plus une liste infinie a apprendre par coeur ce soir. Une poignee de balises te mene deja loin. Et ce n'est pas "plus de balises = meilleure page". Trop d'etiquettes inutiles, c'est du bruit. Un `div` partout "parce que ca marche" cache souvent un manque de reflexion.

Ce n'est pas non plus du JavaScript. Les balises decrivent. Elles ne calculent pas, ne verifient pas un mot de passe, ne chargent pas des donnees toutes seules. Structure d'abord. Comportement, plus tard. Lea le rappelle aux clients impatients de "faire bouger" avant d'avoir un plan clair.

Tu ranges une cuisine. Tu colles des etiquettes : farine, sucre, sel. Le navigateur est le cuisinier. S'il lit "farine" sur le sucre, le gateau rate. Pareil sur une page : si tu mets un titre dans un `div` vague juste pour la taille, tu mens un peu a la machine - et aux lecteurs d'ecran, et a Google. Dis la verite structurelle. Habiller, ce sera le CSS. Max a compris le jour ou un lecteur d'ecran a lu sa page "n'importe comment" parce que tout etait en `div`. Sam fait coller des post-its sur des bouts de papier avant d'ecrire une ligne de code.

Quelques balises utiles tout de suite

Tu vas croiser `h1` a `h6` pour les titres (`h1` le plus important), `p` pour un paragraphe, `a` pour un lien, `img` pour une image, `ul` / `ol` / `li` pour les listes, `div` pour une boite generique, `span` pour un petit bout dans une phrase, `br` pour un retour a la ligne. Tu n'as pas a tout maitriser d'un coup. Tu as a reconnaitre le motif : ouvrir, contenu, fermer - sauf pour quelques balises seules.

```
<h1>Mon titre</h1>
<p>Un paragraphe avec un mot <strong>important</strong>.</p>
<a href="contact.html">Contact</a>
```

Balises seules (pas de fermeture)

Certaines n'ont pas de contenu a enfermer. `<br>` va a la ligne. `<img src="photo.jpg" alt="Une photo">` affiche une image. Tu ne fermes pas comme un paragraphe. Tu donnes des **attributs**, et c'est tout. `<meta charset="UTF-8">` dans le head est pareil : une instruction, pas un bloc de texte. Lea dit : "pas de contenu a enfermer, pas de balise fermante". Sam fait lever la main : "qui fermerait un `img` ?" Personne, apres deux semaines.

Attributs : des infos en plus

Dans une balise, tu ajoutes des details. `<a href="https://exemple.com">Clique ici</a>` : `href` est l'adresse. Sur une image, `src` est le chemin, `alt` est la description si l'image ne charge pas - utile aussi pour l'accessibilite. Les attributs ne sont pas de la deco. Ce sont des instructions precises. Max a gagne des appels avec un simple `href="tel:..."`. Lea refuse les `alt=""` vides sur des images importantes.

Exemple concret : `<a href="contact.html">Contact</a>` - la balise dit "lien", `href` dit ou ca mene, le texte dit ce que le visiteur lit. Trois infos, une seule etiquette.

HTML, c'est de l'ordre

Le navigateur lit de haut en bas. Ce que tu mets en premier apparait en premier (sauf si le CSS change l'ordre plus tard). Range ton code. **Indente**. Lea dit que l'indentation lui a fait gagner des heures de debug. Max a appris apres avoir cherche une balise non fermee pendant une soiree entiere. Sam refuse les fichiers "en ligne" sans retours : l'oeil ne voit plus rien. Chez DanielCraft, un fichier indente n'est pas du luxe. C'est de la survie.

> **✦ ASTUCE**
> Indente d'un cran a chaque ouverture de balise. Tes yeux trouvent les erreurs plus vite.

Petite histoire

Sam a donne un exo : "page avec un titre, deux paragraphes, un mot important". Un eleve a tout mis dans des `div` et a force la taille en CSS colle n'importe ou. Ca marchait "un peu". Puis le lecteur d'ecran a lu n'importe comment. Sam a fait recommencer avec `h1`, `p`, `strong`. Meme contenu. Meilleure page. Lea, sur un audit client, retrouve souvent ce piege : joli a l'oeil, flou pour la machine.

Max avait mis son numero dans un `p` sans lien. Les visiteurs copiaient-coliaient mal. Un `<a href="tel:0612345678">` plus tard, les appels ont monte. Bonne balise, bon attribut, bon resultat. Trois scenes, une lecon : l'etiquette juste change la vie de la page.

Erreur classique

Oublier de fermer une balise. Ecrire `</p>` sans avoir ouvert `<p>`. Mettre un `h1` parce que "c'est plus gros" alors que tu voulais juste un sous-titre. Si la page a l'air cassee, regarde d'abord les ouvertures et fermetures. Respire. Corrige. Rafraichis. Autre piege : inventer des noms de balises (`<titre>`, `<texte>`). Le navigateur ne reconnait que le vocabulaire HTML. Lea dit : "tu ne inventes pas le francais ; tu ne inventes pas le HTML".

> ⚠ **ATTENTION**
>
> Ne choisis pas une balise pour sa taille par defaut. Choisis-la pour son sens. La taille, c'est le CSS.

En vrai

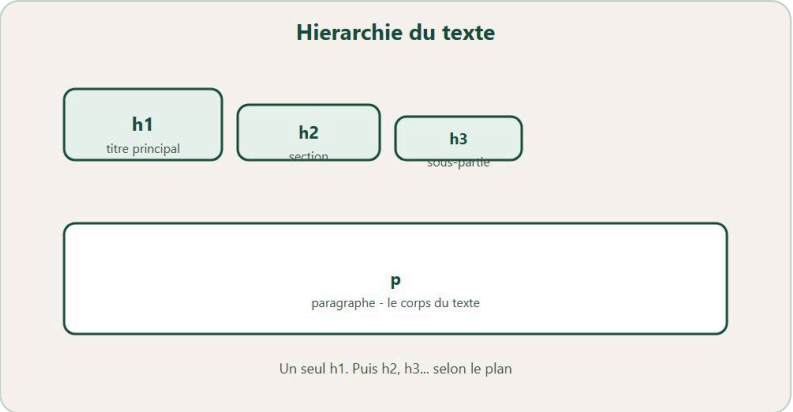
Ecris une page avec un `h1`, deux `p`, et un mot en `<strong>`. Ouvre-la. Puis ajoute un `span` autour d'un mot sans style : tu verras que sans CSS, `span` ne change presque rien - c'est normal. C'est une poignee pour plus tard. Ajoute un commentaire `<!-- section contact -->` : il ne s'affiche pas, il t'aide. Relis ton fichier a voix haute en nommant chaque balise. Le reflexe s'ancre.

A toi

Fais la meme page "Ma journee en etiquettes" : un titre, trois phrases, un mot important, et une note en commentaire HTML `<!-- ... -->` pour toi. Le commentaire ne s'affiche pas. Il te parle. Comme un post-it invisible. Relis ton code a voix haute en nommant chaque balise : "titre, paragraphe, important". Ca ancre le reflexe DanielCraft. Garde ce fichier pour le chapitre texte.

CHAPITRE 4

Chapitre 4 - Titres, textes, paragraphes



h1, h2, h3, p : la hierarchie du texte.

Le texte, c'est le coeur de presque toutes les pages. Boutique, blog, page artisan, fiche de cours : on lit. Si le texte est un mur, on fuit. Si les titres sont dans le desordre, on se perd. Chez DanielCraft, on traite le texte HTML comme un plan de redaction : un sujet clair, des parties, des sous-parties, des phrases qui respirent. Lea coupe les murs chez ses clients. Max a appris a faire trois petits blocs au lieu d'un roman. Sam interdit les sauts de niveaux "parce que c'est joli".

Les titres vont de `h1` a `h6`. Regle simple : un seul `h1` par page (le sujet principal). Ensuite des `h2` pour les parties, des `h3` pour les sous-parties. C'est comme un sommaire. Ca aide le lecteur. Ca aide aussi Google. Mais surtout, ca aide toi a penser clair avant de styler. En 2026, quand quelqu'un dit "ma page est bien structuree", il parle souvent de ce plan. Derriere, il y a du SEO et de l'accessibilite. Pour toi, le geste reste humain : ecrire pour qu'on comprenne. Tu restes le pilote. Le plan guide.

> **★ A RETENIR**
>
> Un seul `h1` par page. Ensuite `h2`, puis `h3`. Le plan avant la taille.

Ce que ce n'est pas

Ce n'est pas "choisir h1 parce que c'est plus gros". La taille, c'est le job du CSS. Ce n'est pas non plus un seul paragraphe geant de quarante lignes. Ce n'est pas du gras partout : si tout est important, plus rien ne l'est. Ce n'est pas non plus du texte copie-colle depuis Word avec des styles caches : tu repars propre en HTML, phrase par phrase.

Ce n'est pas "personne ne voit la structure". Les lecteurs d'ecran la lisent. Google la lit. Toi dans six mois, tu la liras aussi quand tu devras corriger une page client. Lea dit : "le plan HTML, c'est ta table des matieres invisible".

Imagine un livre. Le titre du livre, c'est le `h1`. Les chapitres, ce sont les `h2`. Les sous-parties, les `h3`. Les paragraphes portent les idees. Le gras et l'italique soulignent, ils ne remplacent pas le plan. Si tu sautes des niveaux juste pour la taille, tu ecris un livre dont la table des matieres ment. Le lecteur humain peut s'en sortir. La machine, elle, perd le fil. Sam dessine un sommaire au tableau avant chaque exo de texte. Max a arrete les `<br><br><br>` le jour ou sa page mobile est devenue un desert blanc.

Les titres en pratique

```
<h1>Le plus grand</h1>
<h2>Un cran en dessous</h2>
<h3>Encore un peu plus petit</h3>
```

Un `h1` = le sujet de la page. Des `h2` = les grandes sections. Des `h3` = les details sous une section. Tu peux descendre jusqu'a `h6`, mais rarement besoin au debut. Si tu hesites entre `h2` et `h3`, demande-toi : est-ce une partie ou un detail de la partie ? Lea pose cette question a voix haute avant de coder. Sam l'exige en copie.

Les paragraphes

```
<p>Une idee par paragraphe, c'est souvent plus clair.</p>
<p>Tu peux en mettre plusieurs a la suite.</p>
```

Coupe. Respire. Lis a voix haute si besoin. Une page web n'est pas un SMS interminable. Elle n'est pas non plus un memoire universitaire. Elle est un chemin pour un humain presse. Trois phrases par paragraphe, souvent, c'est deja bien. Max a divise sa page "services" en six petits blocs : les clients ont enfin lu jusqu'au bout. Chez DanielCraft, on aime les pages qui respirent.

Gras, italique, et cie

```
<p>Voici un mot <strong>important</strong>.</p>
<p>Voici un mot <em>en emphase</em> (souvent en italique).</p>
<p>Un <mark>surlignage</mark> pour attirer l'oeil.</p>
```

`strong` = vraiment important. `em` = on insiste un peu. Utilise-les avec parcimonie. Lea dit a ses clients : "trois mots en strong max par paragraphe, sinon on crie partout". Le CSS pourra grossir un titre ; le HTML dit pourquoi il est un titre. Sam raye le gras excessif au stylo rouge sur les copies papier.

Citations et code

```
<blockquote>
  Une citation un peu longue.
</blockquote>
<p>En ligne, on peut citer <code>du code</code>.</p>
```

`code` est pratique quand tu expliques une balise dans une phrase. `blockquote` dit : ceci est cite, pas ton propos principal. Sam les utilise pour des definitions courtes en cours. Lea pour un avis client. Max pour une phrase du fabricant sur ses materiaux. Trois usages, une meme idee : marquer la difference entre ta voix et une autre voix.

Commentaires (pour toi, pas pour le visiteur)

```
<!-- Ceci ne s'affiche pas sur la page -->
```

Utile pour te laisser des notes, ou desactiver un bout sans l'effacer. Max commente les sections "a revoir" avant d'envoyer a un neveu qui l'aide. Sam demande aux eleves de commenter leur intention : `<!-- section horaires -->`. Lea commente les zones fragiles avant une livraison. Le commentaire est un post-it invisible. Utilise-le.

Petite histoire

Lea a repris une page "services" ou chaque sous-titre etait un `h1` parce que le client voulait "que ca claque". Resultat : bruyant pour les lecteurs d'ecran, confus pour le SEO, et moche a corriger. Elle a remis un `h1`, des `h2`, et grossi en CSS. Meme look voulu. Meilleure structure. Le client n'a rien perdu. La page a gagne.

Max avait empile des `<br><br><br>` pour "faire des paragraphes". Sur mobile, c'etait un desert blanc. Sam lui a montre les vrais `<p>`. Meme contenu, meilleure respiration. Lea dit souvent : "si tu mets trois br de suite, tu cherches probablement un paragraphe". Trois scenes, une hygiene.

> **⚠ ATTENTION**
> Ne saute pas de `h1` a `h4` "parce que c'est plus petit". Utilise le bon niveau, puis stylise en CSS.

Erreur classique

Sauter de `h1` a `h4` sans raison. Tout mettre en `strong`. Utiliser `<br><br><br>` pour "faire des paragraphes" au lieu de vrais `<p>`. Ou croire que le plan HTML, "personne ne le voit". Si, beaucoup le voient autrement : clavier, lecteur d'ecran, moteur de recherche, toi dans six mois. Autre piege : un mur de texte sans titres. Coupe. Titre. Respire.

En vrai

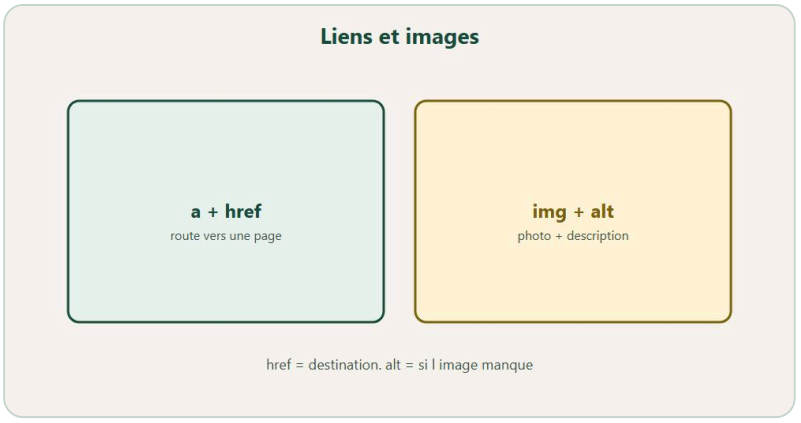
Ouvre ta page. Ecris "Ma journee" avec un `h1`, trois `h2` (matin, apres-midi, soir), un petit paragraphe sous chaque. Relis. Si tu peines a respirer, coupe encore. Puis ajoute un `blockquote` avec une phrase que quelqu'un t'a dit aujourd'hui. Tu vois la difference entre ton texte et une citation. Sauvegarde. F5. Montre a quelqu'un si tu peux.

A toi

Ajoute sous chaque partie un mot en `strong` et une phrase avec `em`. Puis enleve le gras partout sauf un seul mot par section. Observe comme la page devient plus calme. Note ce que tu preferes. C'est ton gout qui se forme - DanielCraft adore ca. Garde cette page : tu la styliseras bientot, et tu seras content d'avoir un plan propre a habiller.

CHAPITRE 5

Chapitre 5 - Liens et images



Lien = route. Image = preuve visuelle avec alt.

Sans liens, le web serait des pages isolees. Sans images, ce serait souvent triste - ou du moins plus abstrait. Les liens connectent. Les images montrent. Ensemble, ils font passer une page de "texte sur fond" a "petit monde navigable". Chez DanielCraft, on apprend les deux avec la meme exigence : un lien clair, une image decrite, des chemins qui existent vraiment. Lea verifie chaque lien avant de livrer. Max a appris les `alt` utiles apres un retour client. Sam note des points si l'`alt` dit juste "image".

Un lien, c'est la balise `a` (anchor, ancre). L'attribut `href` dit ou ca mene. Une image, c'est `img`, avec `src` pour le fichier et `alt` pour la description. En 2026, quand quelqu'un dit "ma page a des liens et des photos", il parle souvent de ca. Derriere, il y a des galeries et des menus complexes. Pour toi, le geste reste simple : promesse claire, chemin reel, description utile. Tu restes le pilote. Le visiteur suit.

> **★ A RETENIR**
>
> Lien = `a` + `href` + texte clair. Image = `src` + `alt` utile. Chemins reels, noms coherents.

Ce que ce n'est pas

Ce n'est pas "clique ici" partout. Le texte du lien doit dire la destination. Ce n'est pas une image sans `alt` "parce que c'est evident". Ce n'est pas non plus coller vingt photos enormes sans dossier `images/`. Et ce n'est pas encore du CSS : on peut mettre une largeur debutant, mais le vrai controle viendra plus tard.

Ce n'est pas non plus une image "dans" le HTML. Le HTML pointe vers un fichier a cote. Si le fichier manque, tu vois une icone cassee. C'est normal. Ca veut dire : verifie le chemin, pas panique sur le code. Lea dit : "le HTML montre le chemin ; le fichier doit exister au bout".

Pense a un plan de ville. Les liens sont les routes entre les lieux. Les images sont les photos sur le plan. Si une route mene nulle part (`href` casse), le visiteur se perd. Si une photo n'a pas de legende (`alt` vide inutile), une partie des gens ne comprend pas. Tu construis des chemins et des preuves visuelles. Lea dit : "un lien, c'est une promesse ; une image, c'est une preuve". Max compare a un devis : "si le chemin photo est faux, le client croit que je n'ai pas de realisations". Sam dessine des fleches au tableau entre pages.

Les liens

```
<a href="https://danielcraft.fr">Aller sur DanielCraft</a>
```

Vers une autre page de ton site : `<a href="contact.html">Contact</a>`. Vers un nouvel onglet : ajoute `target="_blank"` et `rel="noopener"` (petite secu, prends le reflexe). Pour ouvrir le mail : `<a href="mailto:salut@exemple.com">M'ecrire</a>` - ca marche si le visiteur a un client mail configure. Pour appeler : `<a href="tel:0612345678">Appeler</a>`. Max utilise les deux sur sa page artisan. Lea refuse les "clique ici". Sam raye ce texte sur les copies.

> ✦ **ASTUCE**
>
> Texte du lien = destination claire. "Contact" bat "clique ici". Toujours.

Les images

```
<img src="chat.jpg" alt="Un chat orange sur un canape">
```

`src` = ou est le fichier. `alt` = description. Toujours. Pour ceux qui ne voient pas l'image, pour le referencement, pour toi quand le chemin est faux. Organise ton projet simplement :

```
mon-site/
  index.html
  images/
    chat.jpg
```

Alors : `<img src="images/chat.jpg" alt="Un chat orange">`. Largeur debutant possible avec `width="300"` ; plus tard, le CSS fera mieux. Attention a la **casse** : `Photo.JPG` et `photo.jpg` ne sont pas pareils sur tous les serveurs. Max a appris ca sur le telephone de son neveu. Lea normalise tout en minuscules des le premier jour.

Figure + legende

```
<figure>
  <img src="images/chat.jpg" alt="Un chat orange">
  <figcaption>Mon chat, Roi du canape.</figcaption>
</figure>
```

Plus propre quand tu veux une legende visible, pas seulement un `alt`. Le `alt` decrit pour l'accessibilite ; le `figcaption` commente pour tout le monde. Lea utilise `figure` sur les portfolios. Sam le demande des qu'une image porte un message pedagogique. Max l'utilise pour une photo de chantier avec une legende honnete.

Petite histoire

Lea livrait une page portfolio. Deux liens menaient a d'anciennes URL. Le client a clique devant elle. Silence. Elle a rougi, corrige en cinq minutes, et ajoute une checklist "cliquer chaque lien". Max avait mis `src="Photo.JPG"` alors que le fichier etait `photo.jpg`. Sur son PC Windows, parfois ca passait. Sur le telephone du neveu, non. Il a normalise les noms en minuscules. Sam fait casser les liens volontairement en cours pour forcer le debug calme. Trois scenes, une hygiene : teste avant de montrer.

> ⚠ **ATTENTION**
>
> `alt="image"` ne sert a rien. Decris ce qu'on voit : sujet, contexte, utile.

Erreur classique

Oublier le dossier `images/`. Ecrire `alt="image"`. Utiliser "clique ici". Ouvrir un nouvel onglet sans `rel="noopener"`. Croire que l'image "est dans le HTML" alors qu'elle est un fichier a cote. HTML pointe. Il n'embarque pas magiquement la photo. Autre piege : lien vers `http://` sur un site en `https://` - parfois ca bloque. Garde les chemins simples et coherents. Chez DanielCraft, un chemin clair bat une galerie fancy cassee.

En vrai

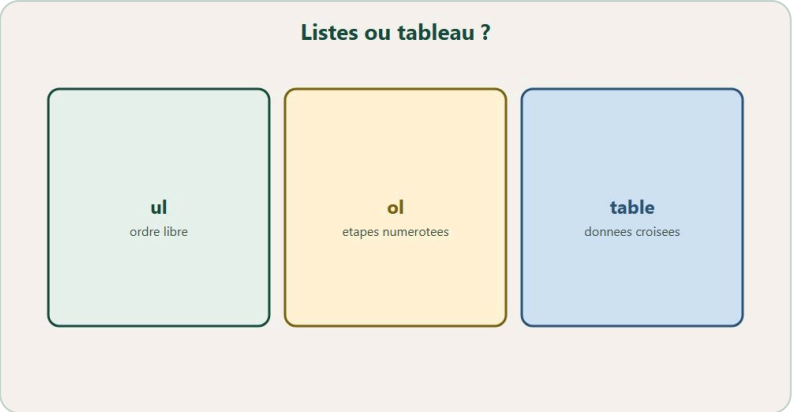
Ajoute un lien vers une page que tu aimes. Ajoute une image (photo, dessin, peu importe). Ecris un vrai `alt`. Clique le lien. Si l'image ne s'affiche pas, lis le `src` a voix haute et verifie le dossier. Ouvre les outils developpeur : parfois le navigateur te dit exactement quel fichier il cherche. Corrige. Relance. Le geste compte plus que la theorie.

A toi

Fais une mini page "Mes trois liens utiles" avec trois ancres explicites, une image avec `figure` / `figcaption`, et un `mailto` vers une adresse fictive. Note le chemin exact de ton image. Ce papier te servira au debug. Puis clique chaque lien : tu entraines le reflexe pro que DanielCraft attend avant livraison. Garde cette page pour le mini-projet.

CHAPITRE 6

Chapitre 6 - Listes et tableaux



ul libre, ol ordonne, table pour les croisements.

Des listes partout : courses, etapes, menus, avantages produit. Des tableaux quand tu compares vraiment des infos en lignes et colonnes. Les deux structurent. Ils ne "decorent" pas. Chez DanielCraft, on les enseigne tot parce qu'ils sauvent des pages confuses : au lieu d'un paragraphe qui enumere vingt choses, tu ranges. Lea utilise des listes pour les livrables client. Max met ses horaires en tableau simple. Sam force les eleves a choisir liste ou tableau avant d'ecrire.

Une liste a puces (`ul` + `li`) dit : voici des elements sans ordre strict. Une liste numerotee (`ol` + `li`) dit : l'ordre compte - des etapes. Un **tableau** (`table` , `tr` , `th` , `td`) dit : ces donnees se croisent. En 2026, quand quelqu'un dit "j'ai structure mon contenu", il parle souvent de ca. Derriere, il y a des composants complexes. Pour toi, la bonne question reste : "quelle forme a l'info ?" Tu restes le pilote. La forme suit le fond.

> **★ A RETENIR**
>
> Ordre libre -> `ul` . Etapes -> `ol` . Donnees croisees -> `table` . Pas de tableau pour la mise en page.

Ce que ce n'est pas

Ce n'est pas un outil de mise en page generale. Ancienne mauvaise habitude : faire tout le layout avec des tableaux. Aujourd'hui, non - c'est le job du CSS (Flexbox plus loin). Ce n'est pas non plus des listes imbriquees a cinq niveaux "parce que ca a l'air pro". Illisible. Et ce n'est pas un tableau pour trois mots : une liste suffit souvent.

Ce n'est pas non plus des tirets dans un paragraphe a la place d'une vraie liste. Visuellement, ca peut ressembler. Structurellement, le navigateur ne voit pas une liste. Et les lecteurs d'ecran non plus. Lea dit : "si c'est une liste, ecris une liste".

La liste, c'est un carnet a puces. Le tableau, c'est une grille d'emplois du temps. Si tu mets l'emploi du temps en puces, tu perds les croisements. Si tu mets la liste de courses en tableau a dix colonnes, tu te fatigues pour rien. Choisis l'outil selon la forme de l'info. Max dit : "ma liste de materiel, c'est des puces ; mes creneaux dispo, c'est un tableau". Sam dessine les deux formes au tableau avant chaque exo. Les eleves votent. Puis ils codent.

Liste a puces

```
<ul>
  <li>Pain</li>
  <li>Lait</li>
  <li>Beurre</li>
</ul>
```

`ul` = unordered list. `li` = list item. Simple, clair, reutilisable. Parfait pour des avantages, des ingredients, des outils sans ordre impose. Lea les utilise pour les "inclus dans le devis". Max pour son materiel de base. Sam pour les objectifs de seance.

Liste numerotee

```
<ol>
  <li>Ouvrir l'editeur</li>
  <li>Ecrire le HTML</li>
  <li>Sauvegarder</li>
  <li>Ouvrir dans le navigateur</li>
</ol>
```

Parfait pour des procedures. Lea numerote ses checklists de mise en ligne. Max numerote "devis, chantier, facture". Sam numerote les etapes d'un exercice : l'ordre compte, le `ol` le dit clairement. Si tu peux changer l'ordre sans perdre le sens, c'etait probablement un `ul`.

> **✦ ASTUCE**
>
> Avant d'ecrire : "l'ordre compte-t-il ?" Oui -> `ol` . Non -> `ul` . Donnees croisees -> `table` .

Liste dans une liste

Ca marche. Tu peux imbriquer. Mais n'en abuse pas : au-dela de deux niveaux, le lecteur se noie. Prefere plusieurs listes courtes avec des titres `h2` / `h3` . Lea decoupe les gros projets client en sections avec listes courtes plutot qu'une tour de Pise a puces. Chez DanielCraft, lisible bat impressionnant.

Les tableaux

```
<table>
  <thead>
    <tr>
      <th>Langage</th>
      <th>Role</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>HTML</td>
      <td>Structure</td>
    </tr>
    <tr>
      <td>CSS</td>
      <td>Style</td>
    </tr>
  </tbody>
</table>
```

`table` enveloppe. `tr` est une ligne. `th` une cellule d'en-tete. `td` une cellule normale. Tu empiles comme des briques. `thead` / `tbody` clarifient pour toi et pour l'accessibilite. Les `th` disent "c'est un titre de colonne", pas juste du texte en gras par defaut. Max met ses horaires ainsi. Lea ses specs produit. Sam ses notes d'eleves en demo.

Petite histoire

Max voulait "un joli menu" avec un tableau a une cellule. Ca tenait a peu pres. Puis sur telephone, c'etait l'enfer. Lea lui a dit : liste ou Flexbox, pas tableau. Il a refait. Sam, en classe, montre un tableau HTML vs un layout CSS : meme look possible, intentions differentes. Les eleves retiennent surtout : tableau = donnees, pas deco.

Lea a herite d'un site e-commerce ou les specs produit etaient des paragraphes avec des virgules. Elle a mis un vrai tableau : poids, dimensions, garantie. Les clients ont arrete de poser les memes questions. Structure claire, moins de mails. Trois scenes, une lecon.

> **⚠ ATTENTION**
>
> Ne construis pas ta mise en page avec des tableaux. Tableau = donnees. Layout = CSS.

Erreur classique

Utiliser un tableau pour centrer un logo. Oublier les `th` et tout mettre en `td` : on perd le sens des en-tetes. Mettre une liste sans `ul` / `ol`, juste des tirets dans un `p` : le navigateur ne voit pas une liste. Ecrire la structure. Croire qu'un tableau "c'est plus pro" pour trois mots : une `ul` suffit. Autre piege : imbriquer cinq niveaux de listes. Coupe. Titre. Respire.

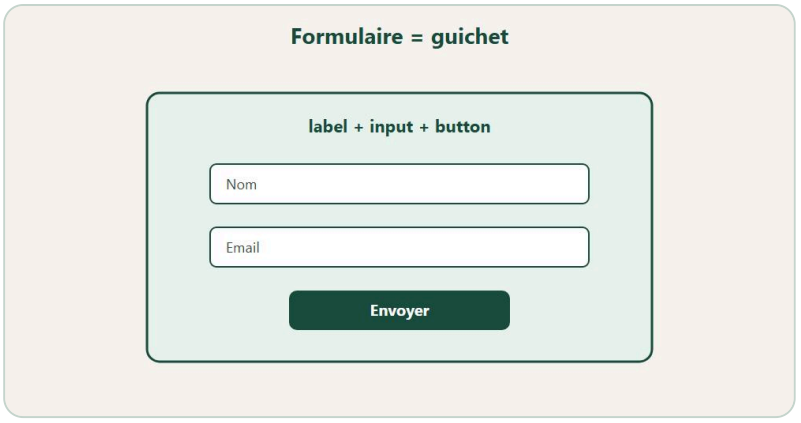
En vrai

Fais une liste de cinq choses que tu veux apprendre. Puis un petit tableau jour / activite (Lundi / HTML, etc.). Ouvre la page. Si le tableau parait trop large, c'est normal : on stylera plus tard. La, valide la structure. Relis a voix haute : est-ce que l'ordre de ta liste numerotee a du sens ? Si non, passe en `ul`.

A toi

Cree une page "Ma semaine" avec un `h1`, une liste d'objectifs, et un tableau de trois jours. Ajoute une phrase sous le tableau qui explique pourquoi tu as choisi tableau et pas liste. Cette phrase force le jugement - competence DanielCraft. Garde-la : tu la reliras au mini-projet quand tu choisiras quoi mettre ou.

Chapitre 7 - Les formulaires



Label, champ, bouton : un guichet clair.

Un **formulaire**, c'est quand tu demandes des infos a quelqu'un : nom, email, message, choix. Tu vois le genre partout - contact, inscription, devis, quiz. En HTML, tu construis les champs. Tu n'as pas encore le serveur qui recoit. Et c'est ok. Chez DanielCraft, on separe : d'abord une forme claire et etiquetee, ensuite le branchement technique. Lea refuse de livrer un contact sans labels. Max a ajoute les siens apres qu'un client age ait bloque. Sam note "placeholder seul" comme erreur grave.

La base enveloppe tout dans `<form>`. Chaque champ a un `<label>` clair. Un `<input>` ou un `<textarea>` recoit la saisie. Un `<button type="submit">` envoie. Le `for` du label doit matcher l'`id` de l'input : cliquer le texte focus le champ. Pratique. Accessible. Professionnel. En 2026, quand quelqu'un dit "j'ai un formulaire contact", il parle souvent de ca. Derriere, il y a des backends et des validations avancees. Pour toi, le geste reste : labels, types, clarte. Tu restes le pilote. Le visiteur comprend.

> **★ A RETENIR**
>
> Label + id/for + type correct. Structure claire d'abord ; envoi reel, plus tard.

Ce que ce n'est pas

Ce n'est pas encore une vraie collecte securisee. `action="#"` veut dire "on n'envoie nulle part de special" pour l'apprentissage. Ce n'est pas non plus `required` = securite totale : le navigateur aide, un serveur devra verifier plus tard. Ce n'est pas un formulaire sans labels "parce que le placeholder suffit" : le placeholder disparait, le label reste.

Ce n'est pas non plus "joli = utilisable". Un formulaire peut etre esthetique et illisible. Les pros commencent par la clarte. Le CSS viendra habiller des champs deja comprehensibles. Lea dit : "d'abord on comprend, ensuite on habille".

Pense a un guichet. Le label est la pancarte au-dessus du guichet. Le champ est la fenetre ou tu glisses le papier. Le bouton est "valider". Si la pancarte manque, les gens (et les lecteurs d'ecran) devinent. Max compare souvent : "sans label, c'est comme un guichet sans numero - tu fais la queue au hasard". Sam fait remplir un mauvais formulaire en classe pour sentir la douleur, puis le corriger. Lea projette les deux versions cote a cote chez le client. Le vote est unanime.

```
<form action="#" method="post">
  <label for="prenom">Prenom</label>
  <input id="prenom" name="prenom" type="text">
  <button type="submit">Envoyer</button>
</form>
```

Types d'input utiles

```
<input type="text" placeholder="Ton prenom">
<input type="email" placeholder="toi@exemple.com">
<input type="password">
<input type="number" min="1" max="120">
<input type="checkbox"> J'accepte
<input type="radio" name="choix" value="a"> Option A
<input type="radio" name="choix" value="b"> Option B
```

`placeholder` = texte fantome. Utile en complement, pas en remplacement du label. Les radios partagent le meme `name` pour ne choisir qu'une option. Deux radios avec des `name` differents, et tu peux tout cocher : piege classique. Sam le piege volontairement. Max est tombe une fois. Lea le verifie sur chaque livraison.

> ✦ **ASTUCE**
>
> Le `for` du label = l'`id` du champ. Clique le texte : le champ se focus. Teste toujours ce geste.

Zone de texte et liste deroulante

```
<label for="msg">Message</label>
<textarea id="msg" name="msg" rows="5"></textarea>

<label for="ville">Ville</label>
<select id="ville" name="ville">
  <option value="paris">Paris</option>
  <option value="lyon">Lyon</option>
</select>
```

`textarea` pour les longs messages. `select` + `option` pour une liste fermee. Choisis selon la liberte que tu veux laisser. Lea met un `select` pour le sujet (Devis, Support, Autre) : moins de messages hors sujet. Max ajoute une checkbox "rappel le matin" : simple, humain. Chez DanielCraft, un champ de moins bien place bat trois champs flous.

Requis

```
<input type="email" required>
```

Le navigateur refuse d'envoyer si c'est vide. Utile pour debuter. Rappel : ce n'est pas une vraie securite cote serveur. Pour plus tard. Mais ca t'entraine a penser "quels champs sont obligatoires ?" avant de styler. Lea coche `required` sur email et message. Max sur telephone. Sam exige au moins un champ requis par formulaire d'exercice.

Exemple : label visible au-dessus du champ.

Petite histoire

Lea a herite d'un formulaire "joli" sans labels, avec des placeholders gris. Sur telephone, un client a tape, a perdu le contexte, a envoye n'importe quoi. Elle a remis labels + `required` + types corrects (`email`). Les messages sont devenus lisibles. Max a fait un devis contact avec checkbox "j'accepte d'etre rappele" : simple, clair, humain. Sam fait comparer deux versions : placeholder seul vs labels visibles. Vote unanime pour les labels. Trois scenes, une hygiene.

> ⚠ **ATTENTION**
> Le placeholder ne remplace pas le label. Quand on tape, le placeholder disparait. Le label reste.

Erreur classique

Oublier le lien `for` / `id` . Mettre deux radios avec des `name` differents (du coup on peut tout cocher). Croire que le formulaire "envoie un mail tout seul" sans backend. Styler avant d'avoir des labels. Structure d'abord. Autre piege : oublier `name` sur les champs - sans `name` , rien n'est identifie cote serveur plus tard. Lea le rappelle aux stagiaires avant chaque livraison.

En vrai

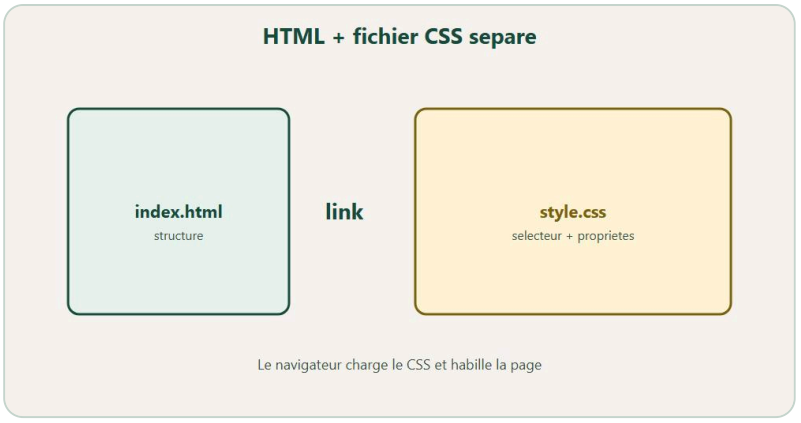
Fais un formulaire Contact : nom, email, message, bouton Envoyer. Clique le label : le champ doit se focus. Laisse l'email vide avec `required` et tente d'envoyer : le navigateur doit raler. Puis remplis avec un faux email : le type `email` doit aussi raler. Tu vois que le HTML aide deja un peu. Note ce qui manque encore (le serveur). Tu restes lucide.

A toi

Ajoute une case a cocher "Je prefere etre contacte le matin" et une liste deroulante de sujet (Devis, Question, Autre). Ecris en deux phrases ce que tu feras plus tard cote serveur, meme si tu ne sais pas encore comment. Le but : savoir ce qui manque - posture de pilote DanielCraft. Garde ce formulaire : tu le styliseras au chapitre CSS, et tu le reprendras au mini-projet.

CHAPITRE 8

Chapitre 8 - CSS : on habille la page



HTML + link + fichier CSS separe.

La, le HTML est nu. On va lui mettre des habits. Le **CSS** ne cree pas le contenu : il decide comment le contenu apparait - couleurs, tailles, espacements, alignements. Chez DanielCraft, on branche le CSS le plus tot possible dans un fichier separe, parce que melanger tout dans le HTML devient vite le bordel. Lea travaille presque toujours en fichier separe. Max a commence en `<style>`, puis a migre. Sam interdit le style en ligne des la deuxieme seance.

Il y a trois facons d'ajouter du CSS. Dans une balise `<style>` dans le `head` : pratique pour tester. Dans un fichier `style.css` relie par `<link>` : le mieux pour un vrai projet. En ligne avec `style="..."` sur une balise : a eviter souvent. En 2026, quand quelqu'un dit "j'ai du CSS", il parle souvent du fichier separe. Derriere, il y a des preprocesseurs et des frameworks. Pour toi, le geste reste : selecteur, propriete, valeur, fichier branche. Tu restes le pilote. La page s'habille.

> **★ A RETENIR**
>
> Fichier `style.css` + `<link>` dans le head. Selecteur + propriete + valeur. Verifie le branchement avant de paniquer.

Ce que ce n'est pas

Ce n'est pas un remplacement du HTML. Sans structure, le plus beau CSS habille du vide. Ce n'est pas non plus "une seule facon absolue" : les trois methodes existent, mais la discipline compte. Ce n'est pas magique : une typo dans un **selecteur** et "rien ne change". Tu verifieras. Ce n'est pas non plus JavaScript : le CSS ne calcule pas, ne verifie pas, ne charge pas des donnees.

Ce n'est pas "joli = bon". Un CSS peut rendre une page illisible. La lisibilite reste la base. On habille, on n'etouffe pas. Lea dit : "si tu ne lis plus ton texte, ce n'est pas du design, c'est du bruit".

Le HTML pose les meubles. Le CSS choisit peinture, luminaires, distances. Le selecteur CSS dit "qui" (tous les `p`, le `h1`, la classe `.carte`). Les **proprietes** disent "quoi" (`color`, `font-size`...). Tu parles a des etiquettes. Si l'etiquette HTML n'existe pas, le CSS parle dans le vide. Lea dit : "selecteur = adresse ; propriete = instruction". Max retient : "si rien ne change, je regarde le link d'abord". Sam dessine la chaine au tableau : HTML, link, CSS, navigateur.

Fichier CSS separe (le mieux)

`style.css` :

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

Dans le HTML, dans le `<head>` :

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

Cree le fichier a cote de `index.html`. Relie. Rafraichis. Si les paragraphes deviennent bleus, c'est branche. Si rien ne bouge, ne reecris pas dix regles : verifie le chemin du `<link>` en premier. Une lettre suffit a tout casser (`styles.css` vs `style.css`). Lea a perdu une heure la-dessus. Max aussi. Sam le piege volontairement.

> ✦ **ASTUCE**
>
> Prefere un fichier `style.css` des le debut. Tu gagnes en clarte, en reutilisation, en calme.

Anatomie d'une regle

```
selecteur {
  propriete: valeur;
}

h1 {
  color: tomato;
  font-size: 40px;
}
```

Commentaire CSS : `/ note pour toi /`. Utile comme les commentaires HTML, dans un autre dialecte. Chaque ligne se termine par un point-virgule `;`. Les deux-points `:` separent propriete et valeur. Une faute la, et toute la regle peut etre ignoree. Chez DanielCraft, on lit la syntaxe avant de blamer le navigateur.

Petite histoire

Lea a passe une heure a "deboguer le bleu" : le `<link>` pointait vers `styles.css` alors que le fichier s'appelait `style.css`. Une lettre. Max avait mis le link apres `</html>` : ignore. Sam projette volontairement un mauvais chemin et fait chercher la classe. Le geste devient un reflexe : verifier le branchement avant de reecrire dix regles.

Max avait aussi ecrit `color blue` sans deux-points. Rien ne marchait. Il a cru que "CSS etait casse". Une correction de syntaxe, tout est parti. Lea garde cette phrase sur un post-it : "d'abord le link, ensuite la syntaxe, ensuite le selecteur". Trois scenes, une methode.

> ⚠ **ATTENTION**
>
> Si "rien ne change" : verifie le `<link>`, le nom du fichier, puis la syntaxe (`:` et `;`). Dans cet ordre.

Erreur classique

Oublier le `<link>`. Se tromper de nom de fichier (casse comprise). Ecrire `color blue;` sans deux-points. Oublier le point-virgule et croire que "CSS est casse" alors qu'une seule ligne derape. Mettre du style en ligne partout puis ne plus oser toucher. Prefere un fichier. Autre piege : modifier le CSS sans rafraichir le navigateur - meme reflexe que pour le HTML : F5. Lea chronometre parfois le debug "rien ne change" : sous deux minutes si on suit l'ordre.

En vrai

Cree `style.css` a cote de `index.html`. Relie-les avec `<link>`. Mets tous les `p` en bleu. Rafraichis. Puis change `h1 { color: teal; }`. Si ca bouge, tu pilotes. Deplace une regle dans un `<style>` temporaire pour tester, puis remets-la dans le fichier : tu sens pourquoi DanielCraft pousse le fichier separe. Casse volontairement le chemin. Lis. Repare. Le contraste enseigne.

A toi

Ecris trois regles : couleur de fond du `body`, couleur du `h1`, taille des `p`. Ensuite deplace une regle du fichier vers un `<style>` temporaire, teste, remets dans le fichier. Tu dois sentir pourquoi le fichier separe gagne : clarte, reutilisation, moins de panique. Note le chemin exact de ton `style.css` - tu en auras besoin jusqu'au mini-projet.

CHAPITRE 9

Chapitre 9 - Couleurs, polices, tailles

Fond, texte, titres : contraste utile.

Maintenant on rend ca vivant. Couleurs, polices, tailles : c'est souvent ce que les gens appellent "le design". En vrai, c'est surtout de la **lisibilite**. Un beau site illisible n'est pas beau longtemps. Chez DanielCraft, on commence doux : fond clair, texte sombre, titre qui ressort, paragraphes qui respirent. Lea garde une petite palette de trois couleurs max. Max a choisi un vert chantier et s'y tient. Sam interdit le jaune fluo sur blanc.

Tu peux ecrire une couleur par nom (`teal` , `navy`), en **hex** (`#ff6600`), ou en rgb. Hex est le plus courant sur le web. `#000000` est noir. `#ffffff` est blanc. Tu n'as pas a retenir les codes par coeur : un color picker suffit. En 2026, quand quelqu'un dit "j'ai choisi ma charte", il parle souvent de ca. Derriere, il y a des design systems. Pour toi, le geste reste : contraste, hierarchie, respiration. Tu restes le pilote. L'oeil suit.

> **★ A RETENIR**
>
> Fond clair, texte sombre, titre qui ressort, line-height qui respire. Lisibilite avant spectacle.

Ce que ce n'est pas

Ce n'est pas quinze polices differentes. Ce n'est pas du texte gris clair sur blanc. Ce n'est pas "plus gros = mieux" sans hierarchie. Ce n'est pas non plus obligatoire d'importer Google Fonts le premier jour : des polices systeme bien choisies suffisent largement pour apprendre.

Ce n'est pas non plus "design = decoration". Tu choisis des couleurs pour guider l'oeil, pas pour impressionner ton neveu. Un accent sur le bouton contact. Un titre qui ressort. Le reste calme. Lea dit : "moins de couleurs, plus de clarte". Sam raye les palettes arc-en-ciel sur les copies.

Pense a une vitrine. La couleur du mur (fond), la couleur des etiquettes (texte), la taille des pancartes (titres vs paragraphes), l'espace entre les lignes (**line-height**). Si tout crie, personne n'entre. Si tout chuchote pareil, personne ne sait ou regarder. Tu crees un contraste utile, pas un spectacle. Max dit : "ma vitrine artisan, c'est fond clair, texte fonce, vert sur le bouton - point". Lea projette deux versions chez le client : contraste faible vs contraste. Le vote est rare.

```
body {
  color: #222222;
  background: #f7f3ee;
}
h1 {
  color: teal;
}
```

Polices

```
body {
  font-family: Georgia, "Times New Roman", serif;
  font-size: 18px;
  line-height: 1.6;
}
```

`font-family` propose une police, puis des secours. `line-height: 1.6` fait respirer. Pour un look moderne sans empattement : `Arial, sans-serif`. Pour un ton un peu plus litteraire : `Georgia, serif`. Deux familles, ca suffit au debut. Lea n'importe des polices exotiques que quand le client insiste - et elle teste toujours la vitesse de chargement. Chez DanielCraft, une police lisible bat une police "originale" illisible.

Graisse, style, taille, alignement

```
h1 { font-weight: 700; font-size: 2.5rem; text-align: center; }
p  { font-size: 1rem; }
em { font-style: italic; }
```

`px` est simple. `rem` est relatif a la taille de base, plus souple pour l'accessibilite. Au debut, `px` est ok. Tu passeras a `rem` quand tu seras a l'aise. `text-align` accepte `left`, `right`, `center`, `justify`. Centrer tout un long article fatigue : reserve le center aux titres courts ou aux citations. Sam interdit le justify sur mobile en debut d'apprentissage : trop de trous bizarre.

Petite histoire

Lea a livre une page "elegante" en gris `#bbbbbb` sur blanc. Sur le telephone du client en exterieur, illisible. Elle a fonce le texte, augmente le line-height, et la page est devenue "moins design" selon le client - puis il a recu plus de messages. Max a mis un titre a 60px sur mobile : ca cassait tout. Il a appris a tester petit ecran. Sam fait comparer deux versions en classe : contraste faible vs contraste. Vote unanime pour le texte fonce. Trois scenes, une lecon : lisibilite d'abord.

> **⚠ ATTENTION**
>
> Texte gris clair sur blanc = piege classique. Fonce. Augmente le line-height. Relis dehors si tu peux.

Erreur classique

Trop de couleurs. Police minuscule "parce que ca fait pro". Centrer tout le texte d'un long article. Oublier le line-height. Changer dix proprietes d'un coup et ne plus savoir ce qui a aide. Une modification, un regard. Autre piege : copier une palette trendy sans tester sur ton contenu reel. Ce qui marche sur un site de mode peut tuer une page artisan sobre. Lea teste toujours sur le vrai texte client, pas sur du "Lorem ipsum".

En vrai

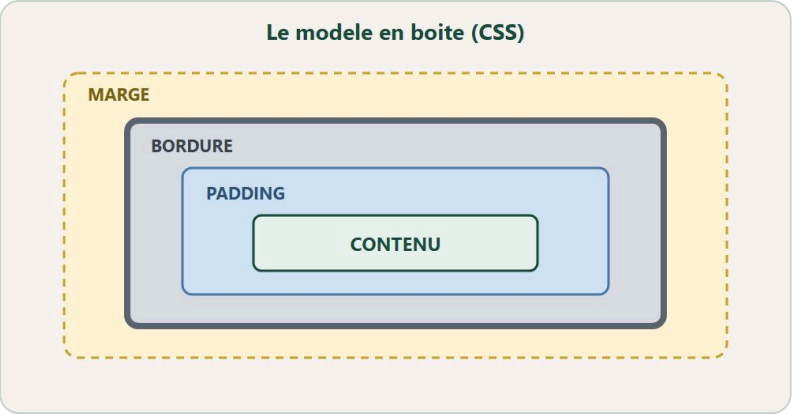
Choisis un fond doux, un texte sombre, un titre colore, des paragraphes a 16-18px minimum. Lis trois phrases a l'ecran. Si tu plisses les yeux, corrige. Puis change seulement la couleur d'accent et observe : parfois un seul changement suffit a rendre la page plus "toi". Reduis la fenetre. Relis. Le telephone revele beaucoup.

A toi

Cree une mini palette ecrite : fond, texte, accent. Applique-la. Puis change seulement l'accent. Note quelle version tu garderais pour ta page perso. Le gout se travaille comme le code - progressivement. DanielCraft conseille de garder cette palette sur un post-it jusqu'au mini-projet : coherence avant fantaisie. Tu sauras ou tu vas.

CHAPITRE 10

Chapitre 10 - Les boites (margin, padding, border)



Margin dehors, padding dedans, border entre les deux.

En CSS, presque tout est une **boite**. Un paragraphe est une boite. Une image est une boite. Un bouton est une boite. Comprendre le modele en boite, c'est arreter de se battre contre "l'espace bizarre". Chez DanielCraft, c'est souvent le chapitre ou les debutants passent de "je bricole" a "je controle". Lea change padding quand le texte colle au bord. Max change margin quand deux blocs se touchent trop. Sam fait dessiner le modele sur papier avant le CSS.

De l'exterieur vers l'interieur : **margin** (espace autour, hors de la boite), **border** (la bordure), **padding** (air dans la boite, entre bord et contenu), puis le contenu. Tu te tromperas entre margin et padding. Tout le monde se trompe. Tu corrigeras. En 2026, quand quelqu'un dit "je maitrise un peu le CSS", il parle souvent de ca. Derriere, il y a Grid et des layouts complexes. Pour toi, le geste reste : padding dedans, margin dehors, inspecter. Tu restes le pilote. La boite obeit.

> **★ A RETENIR**
>
> Padding = air dedans. Margin = air dehors. `border-box` des le depart. Inspecte avec F12.

Ce que ce n'est pas

Ce n'est pas "mettre des `<br>` pour faire de l'air". Ce n'est pas non plus ignorer `box-sizing` et subir des largeurs mysterieuses. Ce n'est pas croire que margin et padding sont synonymes. Et ce n'est pas encore Flexbox : la on parle d'une boite seule et de son enveloppe.

Ce n'est pas non plus "je mets de la margin partout jusqu'a ce que ca marche". C'est tentant. Ca devient vite un puzzle. Une propriete, un test, un regard. Lea dit : "si tu ajoutes la cinquieme margin sans comprendre, arrete-toi et inspecte". Sam chronometre le debug boite : sous cinq minutes avec F12.

Une photo encadree. Le padding, c'est le passe-partout blanc entre photo et cadre. La border, c'est le cadre. La margin, c'est l'espace jusqu'a la photo voisine sur le mur. Si tu elargis le passe-partout, la photo respire dans son cadre. Si tu elargis l'espace mur, tu eloignes les cadres entre eux. Max retient : "padding dedans, margin dehors". Simple. Durable. Lea projette le schema colore des outils developpeur : margin orange, border jaune, padding vert. Les clients comprennent d'un coup.

```css
.carte {
  background: white;
  border: 2px solid #333;
  padding: 20px;
  margin: 16px;
  border-radius: 12px;
}
```

> ✦ **ASTUCE**
>
> Texte colle au bord -> augmente le padding. Deux cartes trop proches -> augmente la margin (ou utilise `gap` en Flexbox).

Largeur, hauteur, box-sizing

```css
.boite {
  width: 300px;
  max-width: 100%;
  height: auto;
}

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

`max-width: 100%` evite les debordements telephone. Sans `border-box`, padding + border peuvent agrandir la boite de facon contre-intuitive. Avec `border-box`, `width` inclut padding et border. Plus clair. Mets ca presque toujours en haut de ton CSS. Lea le met en ligne 1. Max a arrete les scrolls horizontaux honteux le jour ou il l'a compris. Chez DanielCraft, c'est un reflexe moderne, pas une option.

> ✦ **ASTUCE**
>
> Ajoute `* { box-sizing: border-box; }` en haut de ton CSS. Reflexe moderne, moins de surprises.

Display : block ou inline

`block` prend toute la largeur (`p` , `div` , `h1`). `inline` reste dans le flux du texte (`span` , `a` sans style). `inline-block` mixe : dans la ligne, mais padding propre possible.

```css
span.badge {
  display: inline-block;
  padding: 4px 8px;
  background: #eee;
}
```

Utile pour des etiquettes, des badges, des petits boutons texte. Sam les utilise pour marquer "nouveau" ou "gratuit" dans les exercices. Lea pour des tags clients. Max pour "urgence" sur un devis en ligne. Trois usages, une meme boite inline-block.

Petite histoire

Lea avait une "carte produit" ou le texte collait a la bordure. Elle ajoutait des margins partout. Pire. Un collegue a dit : padding. Une ligne. Sourire. Max mettait `width: 400px` sans `max-width`, et sur telephone ca debordait avec un scroll horizontal honteux. `max-width: 100%` l'a sauve. Sam fait colorier le modele en boite dans les outils developpeur : margin orange, border jaune, padding vert. Les eleves voient enfin. Trois scenes, un cran : tu controles l'espace au lieu de le subir.

Erreur classique

Utiliser margin quand tu veux padding (ou l'inverse) dix fois de suite sans tester. Oublier `box-sizing`. Mettre des largeurs fixes partout. Empiler des margins qui "collapsent" de facon surprenante entre deux blocs. Quand c'est flou : inspecte (F12) et regarde la boite coloree. Ne devine pas : regarde. Autre piege : des `<br>` pour "faire de l'air". Prefere padding et margin. Lea raye les br de decoration sur les audits.

En vrai

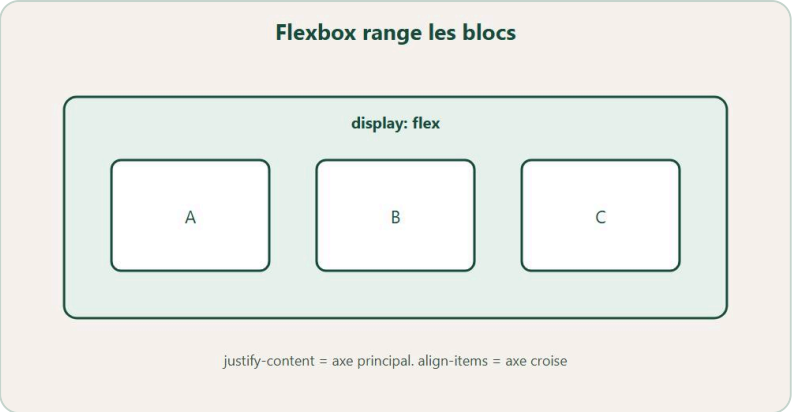
Cree une classe `.carte` avec fond, padding, bordure, coins ronds. Applique-la a une `div` avec titre + paragraphe. Enleve le padding. Remets. Remplace par de la margin. Sens la difference. Reduis la fenetre du navigateur : si rien ne debord, tu progresses. Ouvre F12 et regarde la boite coloree. Tu vois enfin ce que tu manipules.

A toi

Ajoute `* { box-sizing: border-box; }`, une carte a `width: 300px; max-width: 100%;`, et un badge `inline-block`. Reduis la fenetre. Si rien ne debord, tu as gagne un reflexe web moderne. Note-le pour le mini-projet DanielCraft : boites maitrisees, page credible sur telephone. Tu prepares Flexbox avec un sol solide.

CHAPITRE 11

Chapitre 11 - Flexbox : ranger les blocs



Flexbox range les blocs en ligne ou en colonne.

Flexbox, c'est l'outil qui range les blocs sur une page. Menu en ligne, colonnes qui s'alignent, logo a gauche et liens a droite, centrage vertical sans pleurer : ca sauve des soirees entieres. Avant Flexbox, les developpeurs bricolaient avec des floats, des clearfix et des coleres silencieuses. Aujourd'hui, tu dis a un parent : "range tes enfants en flex". Ils obeissent. Chez DanielCraft, on l'apprend des qu'on a compris les boites, parce que le web moderne aligne beaucoup de choses cote a cote ou les unes sous les autres.

Tu as deux roles a retenir. Le **conteneur flex** (le parent) recoit `display: flex`. Les **items flex** (les enfants) se placent selon les regles que tu choisis. Tu peux les mettre en ligne, en colonne, les centrer, les repartir, leur donner de l'espace. La propriete `gap` ajoute de l'air entre eux proprement, sans empiler des margins hasardeuses. Flexbox ne remplace pas le HTML semantique : il range ce qui existe deja.

Lea, freelance web, utilise Flexbox sur presque chaque livraison client. Max, artisan, l'a decouvert quand il voulait aligner son logo et son numero sur la meme barre. Sam, enseignant, fait un atelier ou les eleves doivent reproduire une barre de navigation : ils finissent toujours par tomber sur flex. Trois metiers, un meme outil. En 2026, quand quelqu'un dit "j'aligne avec flex", il parle souvent de ca. Derriere, il y a Grid. Pour toi, le geste reste : parent flex, enfants ranges. Tu restes le pilote.

> **★ A RETENIR**
> Flexbox = le parent range les enfants. `display: flex` sur le conteneur, pas sur chaque enfant.

Ce que ce n'est pas

Ce n'est pas **CSS Grid**, l'autre grand outil de mise en page (on le verra plus tard dans ta formation). Ce n'est pas obligatoire sur chaque `<div>` de la page : si un bloc est seul, inutile de le flexer. Ce n'est pas magique sans reflechir a l'axe : tu dois savoir si tu ranges en ligne (`row`) ou en colonne (`column`). Et ce n'est surtout pas une excuse pour oublier la structure HTML : Flexbox aligne, il ne remplace pas `<header>`, `<nav>`, `<main>` et le sens du contenu.

Ce n'est pas non plus "je mets flex partout et ca marche". Sans `gap`, sans axe clair, tu recrees le bordel d'avant avec un autre outil. Lea dit : "flex avec intention, pas flex par panique".

Imagine une etagere. Le meuble est le conteneur flex. Les boites posees dessus sont les items. Tu decides si elles se mettent cote a cote ou empilees. Tu decides si elles collent a gauche, se centrent, ou se repartissent. `justify-content` travaille sur l'axe principal. `align-items` travaille sur l'axe croise. Tu experimentes, tu changes une valeur, tu regardes. C'est normal de confondre les deux au debut. Max les confond encore parfois. Sam fait changer une valeur a la fois en classe. Lea chronometre : "cinq minutes pour une barre propre".

```
<div class="rangee">
  <div>A</div>
  <div>B</div>
  <div>C</div>
</div>
```

```
.rangee {
  display: flex;
  gap: 16px;
}
```

Trois blocs cote a cote, espaces proprement. Simple. Efficace. Chez DanielCraft, ce mini exemple se montre en trente secondes.

Direction et alignement

La direction par defaut est `row` : les enfants se placent en ligne. Passe en `column` et ils s'empilent. Pour une barre de navigation classique :

```
.rangee {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
}

.colonne {
  display: flex;
  flex-direction: column;
}
```

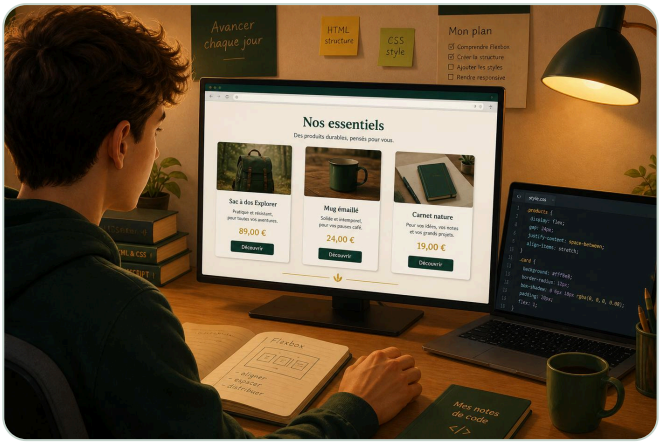
`space-between` pousse le premier element a gauche et le dernier a droite. `align-items: center` aligne verticalement au milieu. Pour centrer un bloc au coeur d'une zone :

```
.centre {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 200px;
}
```

Les enfants peuvent aussi partager l'espace avec `flex: 1`. Pratique pour trois cartes de meme largeur :

```
.rangee > div {
  flex: 1;
}
```

> **✦ ASTUCE**
>
> Logo a gauche, liens a droite : flex + `justify-content: space-between` + `align-items: center`. Cinq lignes de CSS, barre propre. Lea livre ca en dix minutes.



Exemple : trois cartes alignees avec flex.

Petite histoire

Lea devait aligner un logo a gauche et deux liens a droite pour un fleuriste. Avant Flexbox, elle bricolait des floats et des largeurs fixes qui cassaient au premier redimensionnement. Maintenant : un conteneur flex, `space-between`, `align-items: center`, `gap` si besoin. Cinq lignes. Livraison propre. Max voulait empiler ses boutons d'appel sur telephone : meme conteneur, mais `flex-direction: column` dans une media query. Sam fait ses eleves chercher seuls : quand ils trouvent flex, ils comprennent pourquoi le web moderne ne souffre plus autant qu'avant. Trois scenes, un outil.

Erreur classique

Mettre `display: flex` sur l'enfant au lieu du parent. C'est l'erreur numero un. Si rien ne range, verifie ou tu as ecrit flex. Confondre `justify-content` et `align-items` pendant trois jours, c'est normal aussi : change une valeur, regarde, recommence. Oublier `gap` et compenser avec des margins inegales sur chaque enfant, ca desaligne vite. Forcer des largeurs fixes partout et perdre la souplesse de flex, c'est retomber dans les vieilles habitudes.

> **⚠ ATTENTION**
>
> `display: flex` se met sur le parent, pas sur chaque enfant. Si ca ne range pas, c'est souvent la premiere chose a verifier.

En vrai

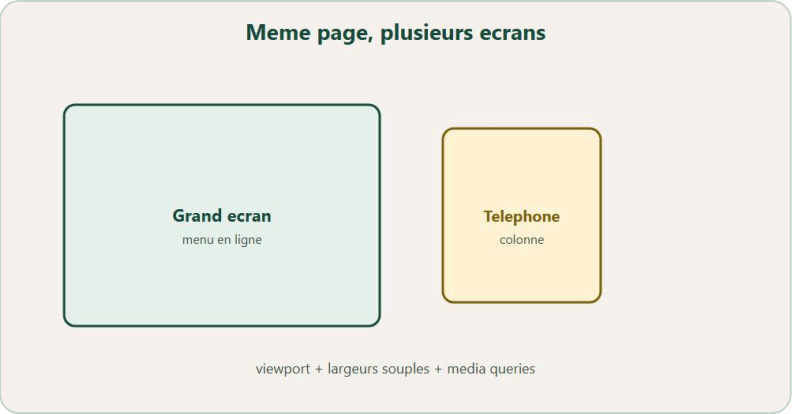
Ouvre ta page perso ou une page vierge. Cree une barre avec un "logo" (un mot en gras) a gauche et deux liens a droite. Indice : flex + `space-between` + `align-items: center`. Puis passe la barre en colonne avec `flex-direction: column`. Observe la difference. Cinq minutes actives valent mieux qu'une lecture passive. Si ca ne range pas, regarde le parent. Toujours le parent.

A toi

Cree trois cartes cote a cote avec `display: flex` et `gap`, chacune avec un titre et un paragraphe. Ajoute `flex: 1` sur les enfants pour qu'elles partagent l'espace. Reduis la fenetre du navigateur : si ca se serre bizarrement, note-le pour le chapitre responsive. Tu prepares le terrain pour la suite. Style DanielCraft : petit layout clair, testable, montrable.

CHAPITRE 12

Chapitre 12 - Un site qui marche sur telephone



Meme site, grand ecran ou telephone : ca s'adapte.

La plupart des gens regardent le web sur mobile. En 2026, ignorer le telephone, c'est ignorer la moitie de tes visiteurs - souvent plus. Si ta page est illisible sur un ecran etroit, c'est rate, meme si elle est parfaite sur ton grand moniteur de bureau. Le **responsive**, ce n'est pas un bonus qu'on ajoute a la fin quand on a le temps. C'est une politesse envers celui qui lit dans le metro, dans sa cuisine, ou sur son canape avec un pouce epais.

Chez DanielCraft, Lea livre toujours en testant une largeur etroite avant de dire "c'est fini". Max a perdu un client parce que son numero de telephone etait coupe hors ecran : le visiteur a abandonne sans appeler. Sam commence ses demos en mode telephone pour montrer que le web, ce n'est pas seulement un grand ecran devant une classe. Trois metiers, une meme lecon : pense doigt, pense petit ecran, pense des le debut. Tu restes le pilote. L'ecran etroit revele tes erreurs.

La base technique, c'est la balise **viewport** dans le `<head>`. Sans elle, le telephone croit souvent que ta page fait la largeur d'un grand ecran et zoom de facon horrible. Avec elle, le navigateur adapte la largeur a l'appareil. Mets-la. Toujours. Avant le reste.

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

> **★ A RETENIR**
>
> Viewport dans le head. Images en `max-width: 100%`. Menu qui s'empile. Teste a 360px aussi.

Ce que ce n'est pas

Ce n'est pas "faire un deuxieme site" separe pour mobile. Tu ajustes le meme HTML avec du CSS adapte. Ce n'est pas seulement une **media query** copiee-collee : les largeurs souples, les images fluides et les boutons assez grands comptent autant. Ce n'est pas ignorer le doigt : une zone cliquable minuscule, c'est une punition quotidienne pour tes visiteurs. Et ce n'est surtout pas tester uniquement sur ton PC en declarant victoire sans jamais reduire la fenetre.

Ce n'est pas non plus "responsive = media query uniquement". Sans viewport, sans images fluides, la media query ne sauve rien. Lea le verifie dans cet ordre : viewport, images, menu, boutons, scroll horizontal.

Pense a une valise. Sur grand ecran, tu deplies tout : menu en ligne, image large, texte aere. Sur telephone, tu plies sans jeter le contenu. Le menu passe peut-etre en colonne. L'image ne debord pas sur le cote. Le texte reste lisible sans zoom permanent avec deux doigts. Tu adaptes le contenant, tu gardes le message. Le responsive, c'est de la diplomatie visuelle. Max dit : "ma page doit tenir dans une poche". Sam fait sortir les eleves avec leur telephone. Lea projette le mode F12 avant/apres.

```
img {
  max-width: 100%;
  height: auto;
}

.conteneur {
  width: min(100% - 2rem, 700px);
  margin-inline: auto;
}

.menu {
  display: flex;
  gap: 1rem;
}

@media (max-width: 600px) {
  .menu {
    flex-direction: column;
  }
}
```

Traduction : les images ne debordent jamais. La boite centrale ne depasse pas 700px. En dessous de 600px, le menu s'empile. Tu peux changer le seuil selon ton design. Le principe reste le meme. Chez DanielCraft, on valide avec les doigts, pas seulement avec les yeux.

Tailles de doigt et vrais tests

Vise une zone cliquable confortable : environ 40 pixels de haut minimum pour un bouton ou un lien important. Reduis la fenetre du navigateur. Ouvre F12 et le mode telephone. Teste a 360px de large, pas seulement a 800px. Si tu peux, regarde sur un vrai telephone plus tard : rien ne remplace le doigt reel. Lea garde une checklist : viewport, images fluides, texte lisible, boutons assez grands, pas de scroll horizontal.

> **⚠ ATTENTION**
> Sans viewport, le telephone "zoome" comme si ta page etait un grand ecran. Ajoute la meta. Toujours. Avant le reste du CSS responsive.

Petite histoire

Max avait une belle page sur son ordinateur. Sur mobile, il fallait scroller lateralement pour lire une seule phrase. Honte douce quand un client lui a envoye une capture avec "je vois rien". Viewport + `max-width: 100%` + menu en colonne sous 600px : repare en une soiree. Lea montre aux clients le mode responsive avant/apres : ca convainc plus qu'un discours. Sam note "oublie viewport" comme erreur classique numero un. Les eleves qui l'oublient reviennent toujours la corriger en premier. Trois scenes, une checklist.

Erreur classique

Oublier le viewport. Ensuite : images en largeur fixe qui debordent, boutons trop petits pour un doigt, media query avec le mauvais selecteur. Tester seulement a 1200px et declarer victoire. Le scroll horizontal, c'est presque toujours un element coupable : une image, une largeur fixe, un padding oublie. Autre piege : texte trop petit "parce que ca fait moderne". Sur telephone, ca fait "je ne lis pas".

> ✦ **ASTUCE**
>
> Ouvre ta page en mode telephone F12. Scrolle horizontalement. Si ca bouge sur le cote, tu as un probleme a trouver avant de continuer.

En vrai

Ajoute le viewport si ce n'est pas fait. Passe ton menu en colonne sous 600px avec une media query. Redimensionne la fenetre. Cherche le scroll horizontal. S'il existe, inspecte element par element jusqu'a trouver le coupable. Souvent c'est une image ou une `width` fixe. Corrige. Reteste a 360px. Tu valides vraiment.

A toi

Ecris ta checklist mobile perso en cinq lignes (viewport, images, texte, boutons, scroll horizontal) et applique-la a ta page en cours. Coche chaque point. Ce papier vaut plus qu'une theorie relue dix fois. Chez DanielCraft, on valide avec les doigts, pas seulement avec les yeux sur un ecran large. Tu prepares le mini-projet : une page qui tient partout.

CHAPITRE 13

Chapitre 13 - Mini-projet : ta page perso

Le but : une petite page a toi, claire et perso.

On assemble tout ce que tu as appris. Une page sur toi - ou un personnage invente si tu preferes rester discret. L'objectif est concret : une page unique avec `index.html` et `style.css`, un en-tete et un menu, une section "A propos", une section "Ce que j'aime" en liste, une image, un formulaire contact, un pied de page. Une seule page, mais complete. Pas un examen piege. Une preuve que les pieces collent entre elles.

Chez DanielCraft, le mini-projet n'est pas la ou tu dois impressionner avec des effets fous. C'est la ou tu livres une version 1 honnete : structure vraie, style separe, telephone inclus, contenu a toi. Lea fait ce genre de page en version client chaque mois. Max a fait la sienne pour son activite d'artisan et l'a montree a sa famille. Sam demande la meme structure a toute la classe pour comparer les gouts. En 2026, quand quelqu'un dit "j'ai fait ma page perso", il parle souvent de ca. Tu livres aujourd'hui. Tu ameliores demain. Tu restes le pilote.

> **★ A RETENIR**
>
> Version 1 livrable bat maquette eternelle. Solide d'abord. Wow ensuite.

Ce que ce n'est pas

Ce n'est pas un site de vingt pages avec blog, boutique et espace membre. Ce n'est pas du JavaScript pour l'instant. Ce n'est pas "parfait ou rien" : si tu attends la perfection, tu ne publies jamais, meme en local. C'est livrable, lisible, a toi. Une vitrine d'une piece, pas un chateau entier. Lea dit : "version 1 qui existe bat version 12 dans ta tete".

Imagine une vitrine avec une seule piece visible. On entre par le **header**. On lit qui tu es dans "A propos". On decouvre ce que tu aimes. On peut t'ecrire via le formulaire. On sort par le **footer**. Le menu saute aux sections avec des ancres `#apropos`, `#likes`, `#contact`. Le CSS tient sur telephone. Tes couleurs, pas celles du voisin. Max compare a un meuble : "imparfait, mais a moi". Sam vote pour la page la plus claire, pas la plus chargee.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ma page perso</title>
```

```
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header class="entete">
    <strong>Ton Prenom</strong>
    <nav class="menu">
      <a href="#apropos">A propos</a>
      <a href="#likes">J'aime</a>
      <a href="#contact">Contact</a>
    </nav>
  </header>
  <main class="conteneur">
    <section id="apropos">
      <h1>Salut, moi c'est ...</h1>
      <p>Deux ou trois phrases sur toi.</p>
      <img src="images/moi.jpg" alt="Photo de ...">
    </section>
    <section id="likes">
      <h2>Ce que j'aime</h2>
      <ul>
        <li>...</li>
        <li>...</li>
        <li>...</li>
      </ul>
    </section>
    <section id="contact">
      <h2>Me contacter</h2>
      <form action="#" method="post">
        <label for="nom">Nom</label>
        <input id="nom" name="nom" type="text" required>
        <label for="email">Email</label>
        <input id="email" name="email" type="email" required>
        <label for="msg">Message</label>
        <textarea id="msg" name="msg" rows="4"></textarea>
        <button type="submit">Envoyer</button>
      </form>
    </section>
  </main>
  <footer class="pied">
    <p>Fait avec HTML et CSS. Et un peu de cafe.</p>
  </footer>
</body>
</html>
```

```
* { box-sizing: border-box; }
body {
  margin: 0;
  font-family: Georgia, serif;
  font-size: 18px;
  line-height: 1.6;
  color: #222;
  background: #f4f1ec;
}
.entete {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 1.5rem;
  background: #1f6f5b;
  color: white;
}
```

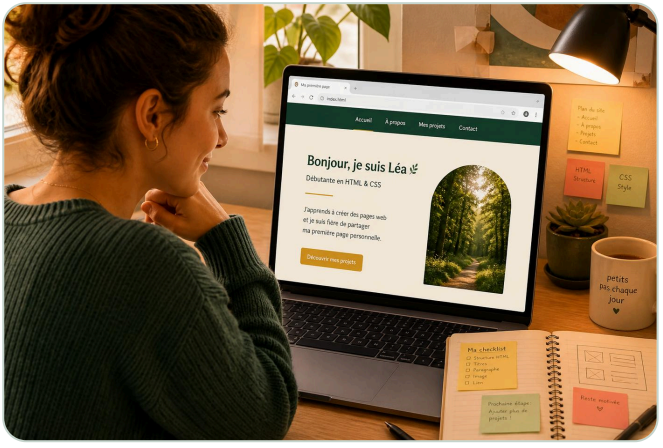
```
.menu { display: flex; gap: 1rem; }
.menu a { color: white; }
.conteneur { width: min(100% - 2rem, 720px); margin: 2rem auto; }
img { max-width: 100%; height: auto; border-radius: 12px; }
form { display: flex; flex-direction: column; gap: 0.5rem; }
input, textarea, button { font: inherit; padding: 0.6rem 0.8rem; }
button {
  background: #1f6f5b; color: white; border: 0;
  border-radius: 8px; cursor: pointer;
}
.pied { text-align: center; padding: 2rem; color: #555; }
@media (max-width: 600px) {
  .entete { flex-direction: column; gap: 0.75rem; }
  .menu { flex-direction: column; align-items: center; }
}
```

Adapte les couleurs. Remplace les textes. Mets ta photo. C'est ton chantier, pas une copie morte du livre.

Criteres de reussite

Ta page s'ouvre sans erreur visible. Le menu saute aux bonnes sections. C'est lisible sur petite largeur. Tes couleurs sont les tiennes. Les labels du formulaire sont presents. L'image a un vrai `alt`. Le **viewport** est dans le head. Le CSS est dans un fichier separe. Chez DanielCraft, ces criteres battent n'importe quel effet wow.

> ✦ **ASTUCE**
>
> Construis en une soiree. Pas dix. Livre une version 1. Ameliore demain. C'est le rythme DanielCraft : petit, souvent, proprement.



Exemple : ta page ouverte dans le navigateur.

Petite histoire

Lea demande toujours "montre-moi sur telephone" avant "montre-moi le code". Max a copie le squelette, change les textes, mis sa photo de chantier, et a envoye le dossier a son neveu avec fierte. Sam affiche trois pages perso anonymisees et fait voter pour la plus claire, pas la plus chargee. La clarte gagne presque toujours. Trois scenes, une lecon : livre, montre, ameliore.

Erreur classique

Tout coller dans un seul fichier HTML avec les styles en ligne. Oublier les `id` des ancres. Image introuvable. Formulaire sans labels. Ne pas tester sous 600px. Viser le wow avant le socle. Autre piege : dix soirees de peaufinage sans jamais dire "c'est une v1". Lea chronometre parfois : "ce soir, tu livres".

En vrai

Construis la page en une soiree. Pas dix soirees espacees. Livre une version 1 complete. Ameliore demain avec un seul point (contraste, menu mobile, ou `alt` plus precis). Une version honnete bat une maquette eternelle jamais montree. Ouvre-la sur telephone si tu peux. Souris. C'est a toi.

A toi

Construis ta page perso maintenant. Bonus si tu veux : ajoute un second fichier `projets.html`, relie les deux pages, mets trois projets inventes en liste. Puis ecris trois ameliorations futures. Chez DanielCraft, on aime les versions 1 honnetes plus que les maquettes eternelles. Tu as les outils. Il ne reste que le geste.

CHAPITRE 14

Chapitre 14 - Ce qu'il faut retenir (+ la suite)

Carte des bases : structure, style, boites, flex, telephone.

Presque au bout du parcours principal. Encore des ateliers pratiques, un peu d'accessibilite, un quiz, puis on cloture. Ce chapitre sert de carte mentale. Pas une liste seche a recracher : une vue d'ensemble pour ranger ce que tu as deja fait de tes mains, avec tes propres erreurs et tes propres victoires. Chez DanielCraft, retenir, c'est savoir ou regarder - pas decorer sa memoire.

HTML, c'est la structure : les pieces de la maison, le sens du contenu. **CSS**, c'est le style : peinture, espacement, polices, couleurs. Les balises sont des etiquettes de sens. En CSS, tu ecris un selecteur puis des proprietes. Les boites, c'est margin, border, padding, avec `box-sizing: border-box`. **Flexbox** range les blocs. Viewport + largeurs souples + media queries font tenir la page sur telephone. Quatre piliers : structure vraie, style separe, test frequent, telephone inclus.

Lea dit qu'elle reapprend encore des details CSS chaque annee. Max est fier de sa page vitrine et sait qu'elle evoluera. Sam rappelle : retenir, c'est refaire, pas surligner passivement. Trois metiers, une meme verite. En 2026, quand quelqu'un dit "je connais les bases HTML/CSS", il parle souvent de cette boite. Derriere, il y a Grid, animations, frameworks. Pour toi, le socle porte tout. Tu restes le pilote.

> **★ A RETENIR**
> Structure vraie. Style separe. Test frequent. Telephone inclus. Quatre piliers, un socle solide.

Ce que ce n'est pas

Ce n'est pas "tu sais tout le web". Loin de la. Ce n'est pas JavaScript encore (sauf si tu as deja pique par curiosite). Ce n'est pas la fin de l'apprentissage : c'est la fin d'un premier socle solide. Beaucoup de developpeurs pros ont commence exactement la ou tu es. La difference, c'est qu'ils ont continue a livrer petit, souvent, proprement. Lea le repete aux juniors impatients de "passer a React".

Tu as une boite a outils. Dedans : un fichier HTML propre, un fichier CSS separe, des balises qui ont du sens, des liens et images honnetes, un formulaire etiquete, des couleurs lisibles, les boites comprises, flex pour aligner, le responsive pour le doigt. Le quiz va secouer la boite. Les ateliers vont t'obliger a chercher sans tuto sous les yeux. Le bravo viendra apres l'effort. Tu n'as pas besoin de tout tenir en memoire. Tu as besoin de savoir ou chercher et comment tester.

Petites habitudes qui changent tout

Indente ton code. Ferme tes balises. Mets toujours un `alt` utile. Relie un vrai fichier `.css`. Teste souvent : modifie, sauvegarde, F5. Une erreur ? Regarde la derniere chose changee. Noms de fichiers simples, minuscules, sans espaces. Viewport present. Contraste lisible. Ces habitudes battent n'importe quelle astuce isolee. Max a sa checklist sur un post-it. Lea la sienne imprimee. Sam la fait recopier a la main.

> ✦ **ASTUCE**
>
> Ecris ta checklist perso en six lignes max, maintenant. Ce qui manque dans ta tete = ce qu'il faut revoir avant le quiz.

Petite histoire

Lea garde une checklist imprimee : viewport, link CSS, alt, test mobile, labels formulaire. Max a la meme en version post-it sur son ecran de garage. Sam la fait recopier aux eleves a la main. Quand un eleve oublie le viewport pour la troisieme fois, Sam ne gronde pas : il renvoie vers le chapitre 12 et demande de corriger. Trois scenes, une methode : checklist avant panique.

Erreur classique

Croire que "retenir" veut dire "tout savoir par coeur". Non. Retenir, c'est savoir ou regarder et quoi verifier en premier. Oublier le `<link>`. Mauvais chemin d'image. Typo de classe. Oublier le viewport. Confondre margin et padding. Ca arrive a tout le monde. Le debug, c'est le vrai boulot du web. Les gens qui avancent supportent de chercher sans se raconter qu'ils sont nuls.

La suite (quand tu seras chaud)

Pousser le CSS plus loin (Grid, animations legeres). JavaScript pour rendre la page interactive. Accessibilite plus profonde. Mettre un site en ligne. Mais d'abord : les ateliers debug et produit, l'accessibilite de base, le quiz. Puis tu pourras dire "c'est fait" sans mentir. Chez DanielCraft, on empile les etages sur un sol solide - pas sur des tutos flous.

En vrai

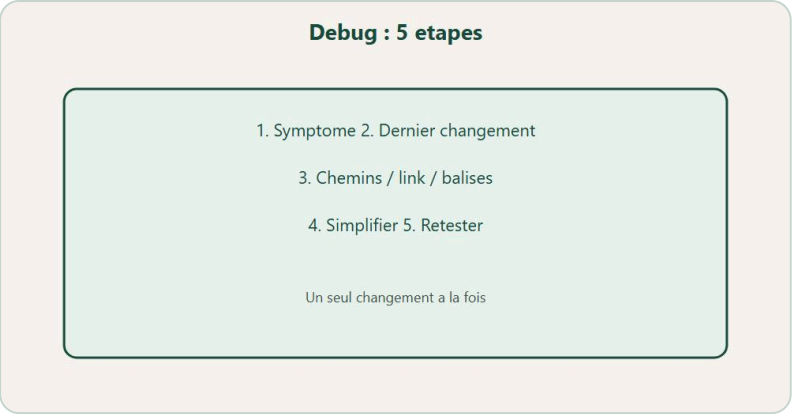
Rouvre ta page perso du chapitre 13. Sans relire le livre, ameliore un seul point : le contraste, le menu mobile, ou un `alt` plus precis. Une amelioration consciente bat dix lectures passives de ce chapitre recap. Chronometre-toi : quinze minutes max. Puis stop. Tu as avance.

A toi

Ecris sur papier : trois forces de ton parcours, deux faiblesses honnetes, une prochaine etape (atelier debug, page produit, ou quiz). Garde ce papier. C'est ton plan post-livre - style DanielCraft : petit, clair, actionnable. Date-le si tu peux. Tu sauras ou reprendre.

CHAPITRE 15

Chapitre 17 - Atelier debug : la page est cassee



Symptome, dernier changement, chemins, simplifier, retester.

Ca arrive. Souvent. Et c'est normal. Ici, on apprend a chercher sans paniquer. Le **debug**, ce n'est pas la preuve que tu es nul. C'est le metier. Une page qui casse, c'est une page qui te parle : quelque chose ne va pas, et tu vas trouver ou. Chez DanielCraft, on dit clairement : les gens qui avancent supportent de chercher. Ceux qui abandonnent au premier bug ne decouvrent jamais la satisfaction de reparer seuls.

Lea passe une part non negligeable de sa semaine a lire des chemins faux et des classes mal orthographiees. Ce n'est pas glamour, mais c'est paye. Max a gagne en confiance le jour ou il a reussi a reparer sa page vitrine seul. Sam transforme les paniques en methode. Objectif : casser volontairement, puis reparer avec une checklist. Duree : 25 a 40 minutes. En 2026, quand quelqu'un dit "je debug", il parle souvent de cette hygiene. Tu restes le pilote. L'erreur est un signal.

> **★ A RETENIR**
> Une hypothese. Un test. Puis la suivante. Jamais dix changements d'un coup.

Ce que ce n'est pas

Ce n'est pas "lire la console JavaScript" en premier (on est surtout HTML/CSS ici). Ce n'est pas changer dix choses a la fois. Ce n'est pas recopier tout internet. Une hypothese, un test, une conclusion. Puis la suivante. Lea appelle ca "chirurgie", pas "demolition". Sam chronometre la panique puis la methode : la deuxieme gagne toujours.

Tu es medecin de page web. Tu observes le symptome : style absent, image cassee, menu folle, scroll horizontal. Tu demandes : qu'est-ce qui a change en dernier ? Tu verifies les organes vitaux : doctype, balises fermees, chemins, link CSS, classes qui matchent. Tu simplifies : tu commentes un gros bloc, tu retestes. Tu ne remplaces pas le patient entier au premier doute. Max prefere ca aux soirees a tout recommencer. Lea aussi.

Methode en 5 etapes

1. 1. Relis la derniere chose que tu as changee.
2. 2. Observe le resultat. Ouvre F12 si besoin.
3. 3. Verifie les balises ouvertes et fermees.
4. 4. Verifie les chemins : `href`, `src`, `<link>` CSS.

5. 5. Simplifie : commente un gros bloc, reteste, isole le coupable.

> ⚠ **ATTENTION**
>
> Casse volontairement, puis repars. Tu entraines le calme sans la pression d'un vrai client qui attend.

Checklist HTML

`<!DOCTYPE html>` present. `lang="fr"`. `charset` UTF-8 et **viewport** dans le head. Balises fermees. Un seul `h1`. Ancres du menu qui matchent des `id` reels. Labels de formulaire lies aux champs. Chez DanielCraft, cette checklist sauve des heures.

Checklist CSS

`<link rel="stylesheet" href="style.css">` present, nom exact. Classe HTML = classe CSS (casse comprise). Point-virgules. `box-sizing: border-box` si les largeurs debordent. Piege classique : `Style.css` vs `style.css`. Sur Windows ca peut marcher. Ailleurs, ca casse. Noms simples, minuscules, partout.

Exercices

Exercice 1 (10 min) : enleve une balise fermante `</section>`. Observe. Repare.

Exercice 2 (10 min) : renomme ton CSS sans mettre a jour le `<link>`. Observe la page nue. Repare.

Exercice 3 (10 min) : casse le `src` d'une image. Lis le `alt`. Corrige.

Exercice 4 (optionnel) : `.menu` en HTML et `.Menu` en CSS. Comprends. Uniformise.

Petite histoire

Lea raconte qu'un junior a reecrit tout un fichier CSS pour un point-virgule manquant. Elle lui a appris : "c'est une ligne, pas un fichier entier". Max a cherche deux heures un style absent : le `<link>` etait commente. Sam chronometre panique puis methode. Trois scenes, une posture : calme, checklist, petit fix.

Erreur classique

Changer dix choses a la fois. Tu ne sauras plus ce qui a marche. Un changement, un test, une note. Ecris ce que tu as appris apres chaque bug : symptome, cause, correction, lecon. Sans livrable, le cerveau classe ca comme "galere". Avec livrable, comme "competence". Autre piege : tout effacer pour recommencer. Rarement necessaire. Souvent panique.

En vrai

Refais l'atelier sur une copie de ta page perso. Casse trois choses differentes. Repare avec la checklist. Chronometre-toi : tu dois aller plus vite qu'au premier essai. Le calme s'installe avec la repetition. Pas magie. Habitude.

A toi

Cree un fichier notes `atelier-debug.md` (ou papier) avec : symptome, cause, correction, lecon pour chaque exercice. Refais l'atelier une semaine plus tard sans relire tes notes d'abord. Si tu vas plus vite, tu as integre. Sinon, recommence. DanielCraft prefere la repetition active au binge de tutos oublies le lendemain.

CHAPITRE 16

Chapitre 18 - Atelier : une mini page produit

Nom, prix, preuves, action : hierarchie visuelle.

On fait une page "produit" simple. Comme une fiche boutique en ligne, sans la boutique entiere derriere. Objectif concret : nom du produit, prix visible, courte description, liste d'avantages, bouton "Ajouter au panier" (meme s'il ne fait rien encore sans JavaScript), CSS mobile-friendly. Une fiche claire, pas une usine a gaz. Chez DanielCraft, cet atelier entraine la **hierarchie visuelle** et la retenue : savoir ce qui compte en premier pour le lecteur presse.

Lea livre des fiches produit pour des artisans et des petites marques. Elle sait qu'un prix cache en petit gris, personne ne le voit. Max a fait la sienne pour un kit d'entretien : nom, prix, trois avantages, bouton d'appel. Sam utilise l'exercice pour parler clarte, pas "design circus". En 2026, quand quelqu'un dit "j'ai une fiche produit", il parle souvent de cette hierarchie. Tu restes le pilote. L'oeil suit ton ordre.

> **★ A RETENIR**
>
> Le regard tombe sur le nom, puis le prix, puis pourquoi c'est bien, puis l'action. Respecte cet ordre.

Ce que ce n'est pas

Ce n'est pas une vraie boutique avec panier, paiement et stock. Ce n'est pas quinze polices et six popups qui crient "PROMO". Ce n'est pas une image de stock illegible sans `alt`. Un seul objectif : que quelqu'un comprenne le produit en quelques secondes, sans zoomer, sans deviner. Lea dit : "si tu dois expliquer ou est le prix, tu as rate".

Une etiquette de magasin bien faite. Le regard tombe d'abord sur le nom, puis sur le **prix**, puis sur pourquoi c'est bien, puis sur l'action. Si tu inverses (blabla d'abord, prix perdu en bas), tu perds le lecteur presse. La hierarchie visuelle, c'est guider l'oeil, pas le noyer. Max compare a son etal de marche : "le prix se voit avant le roman". Sam chronometre : "trois secondes pour comprendre ?"

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Casque SoftBeat</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
```

```
  <main class="fiche">
    <p class="tag">Nouveau</p>
    <h1>Casque SoftBeat</h1>
    <p class="prix">79 €</p>
    <p>Un casque confortable pour la maison, les cours, et les trains un peu longs.</p>
    <h2>Pourquoi il est cool</h2>
    <ul>
      <li>Leger</li>
      <li>Bonne autonomie</li>
      <li>Son clair</li>
    </ul>
    <a class="btn" href="#">Ajouter au panier</a>
  </main>
</body>
</html>
```

```
* { box-sizing: border-box; }
body {
  margin: 0;
  font-family: Georgia, serif;
  background: #f6f3ee;
  color: #222;
}
.fiche {
  width: min(100% - 2rem, 420px);
  margin: 2rem auto;
  padding: 1.5rem;
  background: white;
  border-radius: 12px;
  box-shadow: 0 8px 24px rgba(0,0,0,0.08);
}
.tag { color: #1f6f5b; font-weight: 700; margin: 0; }
.prix { font-size: 2rem; font-weight: 700; margin: 0.5rem 0; }
.btn {
  display: inline-block;
  margin-top: 1rem;
  padding: 0.8rem 1.2rem;
  background: #1f6f5b;
  color: white;
  text-decoration: none;
  border-radius: 8px;
}
@media (max-width: 480px) {
  .prix { font-size: 1.6rem; }
}
```

Adapte le produit. Change les couleurs. Mets une vraie image avec `alt` si tu veux. Garde la hierarchie.

Criteres de reussite

Le nom et le prix se voient tout de suite. Les avantages sont en liste. Le bouton est assez grand sur telephone. La fiche tient a 360px sans scroll horizontal. Le viewport est present. Chez DanielCraft, ces criteres battent dix badges "promo".

Petite histoire

Lea a herite d'une fiche ou le prix etait en bas, en gris, apres trois paragraphes marketing. Elle l'a remonte sous le titre. Les clics ont monte. Max a fait sa fiche kit d'entretien pour le marche : trois avantages, gros prix, bouton "Appeler". Les gens ont compris sans lui demander. Sam fait comparer deux versions en classe : prix visible vs prix cache. Vote unanime. Trois scenes, une lecon.

Erreur classique

Prix trop petit. Bouton trop petit. Trop de texte avant l'essentiel. Image sans `alt`. Oublier le mobile. Autre piege : cinq polices et six couleurs "parce que ca fait boutique". Prefere une fiche calme et nette. Lea raye le bruit visuel sur les audits.

> ⚠ **ATTENTION**
>
> Si le prix n'est pas lisible en trois secondes, remonte-le et grossis-le. Le reste peut attendre.

En vrai

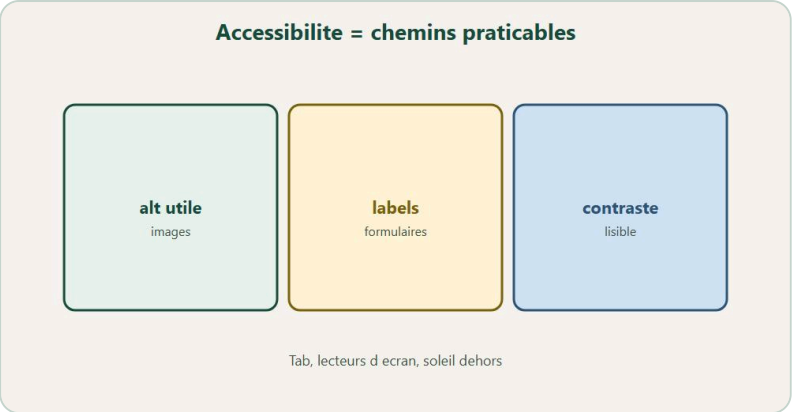
Construis la fiche avec ton produit (reel ou invente). Montre-la a quelqu'un pendant cinq secondes. Demande : "c'est quoi, ca coute combien ?" Si la reponse est juste, tu as gagne. Sinon, corrige la hierarchie. Reteste. Cinq minutes actives valent mieux qu'une heure de peaufinage flou.

A toi

Fais ta fiche produit maintenant. Bonus : ajoute une image avec `figure` / `figcaption`, et un second bouton "En savoir plus" vers une ancre. Ecris trois lignes sur ce que tu as choisi de mettre en premier et pourquoi. Posture DanielCraft : hierarchie consciente, pas deco au hasard.

CHAPITRE 17

Chapitre 19 - Accessibilite et bonnes manieres



alt, labels, contraste : chemins praticables.

Accessibilite, ca veut dire : le plus de gens possible peuvent utiliser ta page, quel que soit leur corps, leur materiel ou leur fatigue. Y compris avec un clavier seul, un lecteur d'ecran, un contraste faible, un doigt epais, une connexion lente, ou une fin de journee ou tu plisses les yeux. Pas besoin d'etre expert certifie. Juste quelques reflexes honnetes des le debut.

Chez DanielCraft, ce n'est pas un chapitre "moralisateur" qu'on survole. C'est du professionnalisme concret. Lea le vend comme de la qualite : "votre site sera lisible par plus de monde". Max l'a compris quand un client age a dit "enfin je peux lire sans loupe". Sam le note dans ses grilles avec autant de serieux que le HTML. Tu as deja des bases : `alt`, labels, titres ranges, viewport. Ici, on solidifie. En 2026, quand quelqu'un dit "c'est accessible", il parle souvent de ces reflexes. Tu restes le pilote. Le soin se voit.

> **★ A RETENIR**
>
> `alt` utile. Contraste ok. Liens explicites. Titres ranges. Zones cliquables confortables. Labels presents.

Ce que ce n'est pas

Ce n'est pas "faire moche au nom de l'accessibilite". Souvent, accessible = plus clair, donc plus beau utile. Ce n'est pas une certification WCAG complete en un chapitre. Ce n'est pas non plus mettre `alt=""` partout sans reflechir : decoratif vs informatif, tu choisis consciemment. Une image purement decorative peut avoir un alt vide. Une photo de produit, non. Lea refuse les `alt="image"`. Sam raye les "clique ici".

Tu invites du monde chez toi. Certains voient bien, d'autres moins. Certains utilisent la souris, d'autres la touche Tab. Certains lisent dehors au soleil. Tu n'eteins pas la lumiere "parce que c'est design sombre et premium". Tu gardes des chemins praticables. Ta page est une maison hospitaliere, pas un club prive. Max dit : "si mon oncle ne peut pas lire, j'ai rate". Sam fait Tab les yeux detournes. Lea projette un mauvais contraste : personne ne lit du fond de la salle.

```
<img src="chat.jpg" alt="Chat orange assis sur un canape gris">
```

Pas `alt="image"`. Pas `alt="photo1"`. Lea ecrit des `alt` comme des legendes courtes. Max decrit "camion de depannage devant la maison a Lyon". Sam demande de lire les alt a voix haute : si ca ne decrit rien, c'est a refaire.

Contraste, liens, boutons, titres

Texte gris tres clair sur fond blanc = fatigue et exclusion. Texte sombre sur fond clair = mieux. Si tu plisses les yeux, change. Liens : evite "clique ici", prefere "Voir les tarifs" ou "Appeler Max". Boutons : assez grands sur telephone (environ 40px de haut). Titres : `h1` puis `h2` puis `h3`, sans sauter des niveaux juste pour la taille. Chez DanielCraft, ces regles sont du soin, pas du luxe.

> ✦ **ASTUCE**
>
> Teste Tab sur ta page avant de dire "c'est fini". Si tu te perds au clavier, un lecteur d'ecran se perdra aussi.

Clavier et checklist rapide

Essaie Tab : tu dois atteindre tous les liens et boutons importants dans un ordre logique. Si tu te perds, simplifie. Mini checklist : `alt` utiles, contraste ok, liens explicites, titres ranges, zones cliquables confortables, viewport, labels. Corriger trois points sur ta page perso, c'est deja un vrai plus professionnel.

Petite histoire

Lea a livre un site "premium" en gris tres pale. Audit : contraste insuffisant. Correction : texte plus sombre, boutons plus nets, liens soulignes. Le client a dit "moins luxe". Puis ses utilisateurs ages ont repondu plus vite. Il a garde la version accessible. Max a remplace "clique ici" par "Appeler Max au 06..." : plus d'appels. Sam projette un mauvais contraste : personne ne lit. Trois scenes, une lecon.

Erreur classique

Aller trop loin dans le "design pale". Oublier le clavier. Mettre du texte important uniquement dans une image. Croire que l'accessibilite, "c'est pour plus tard". Plus tard, c'est plus cher, et tu as deja perdu des visiteurs. Autre piege : `alt=""` sur une image informative pour "faire plaisir au validateur". Choisis consciemment.

En vrai

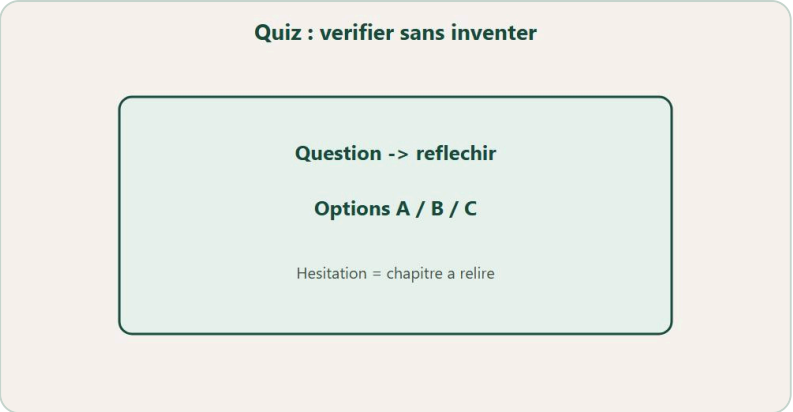
Reprends ta page perso. Corrige trois points : un `alt` plus precis, un meilleur contraste, un lien plus explicite. Teste Tab du debut a la fin. Lis tes `alt` a voix haute. Si ca decrit vraiment, tu as gagne. Sinon, reecris. Cinq minutes de soin valent une heure de deco inutile.

A toi

Ecris ta checklist accessibilite en six lignes et colle-la pres de ton ecran (ou en commentaire HTML en tete de page). Prochaine page que tu construis, tu l'utilises avant de dire "c'est fini". Reflexe DanielCraft : accessible n'est pas optionnel, c'est du soin apporte au travail. Tu en es capable des maintenant.

QUIZ

Quiz final - Teste-toi !



Verifier ce que tu as compris, sans inventer.

Avant de dire "j'ai fini", on joue un peu. Lis chaque question. Cherche la reponse dans ta tete avant de regarder les options. Les corriges sont juste apres. Pas de triche... enfin, si tu veux relire un chapitre entre deux questions, c'est ton livre. L'idee n'est pas de te punir. C'est de verifier que **HTML** et **CSS** ne sont plus du brouillard dans ta tete.

Chez DanielCraft, un quiz sert a reperer les chapitres a relire, pas a humilier. Lea revoit parfois ces bases avant un audit client. Max a refait le quiz deux fois : mieux la seconde. Sam note les hesitations, puis renvoie vers deux ou trois chapitres cibles. Toi aussi : le score mesure un instant, pas ta valeur. Respire. Douze questions. Douze checkpoints. Pas de piege volontaire. Juste le socle.

> ✦ **ASTUCE**
>
> Note tes hesitations pendant le quiz. Pour chacune, cinq minutes dans le chapitre lie. Puis reteste ces questions seules, sans regarder les reponses.

Questions

Q1. HTML, ca sert surtout a quoi ?

- A) Mettre des couleurs et des polices
- B) Structurer le contenu de la page
- C) Remplacer le navigateur

Q2. CSS, ca sert surtout a quoi ?

- A) Donner le style (couleurs, tailles, rangement...)
- B) Creer la base de donnees du site
- C) Envoyer des emails aux visiteurs

Q3. Ou places-tu le vrai contenu visible de la page ?

- A) Dans `<head>`
- B) Dans `<body>`
- C) Dans `<!DOCTYPE html>`

Q4. Quelle balise donne le titre principal (le plus important) de la page ?

- A) `<p>`
- B) `<h1>`
- C) `<br>`

Q5. Pour un lien cliquable, quel attribut donne l'adresse de destination ?

- A) `src`
- B) `alt`
- C) `href`

Les cinq premieres couvrent le socle : roles HTML/CSS, body, titres, liens. Si tu hesites ici, revois les chapitres 1 a 5 avant de continuer.

Q6. Sur une image, `alt` sert surtout a :

- A) Compresser le fichier
- B) Decrire l'image (accessibilite, fallback)
- C) Changer la couleur du bord

Q7. Une liste a puces se construit avec :

- A) `<ul>` et `<li>`
- B) `<table>` et `<td>`
- C) `<form>` et `<input>`

Q8. Dans un formulaire, le `label` doit surtout :

- A) Remplacer le bouton
- B) Etre lie au champ (`for` / `id`)
- C) Etre cache pour faire joli

Q9. Le mieux pour du CSS de projet, c'est souvent :

- A) Tout en `style="..."` sur chaque balise
- B) Un fichier `.css` relie avec `<link>`
- C) Mettre le CSS apres `</html>`

Q10. Padding, c'est surtout :

- A) L'espace a l'exterieur de la boite
- B) L'espace a l'interieur de la boite
- C) Une balise HTML

La moitie du quiz. Images, listes, formulaires, CSS, boites. C'est la que Lea revoit le plus souvent avant une livraison.

Q11. Pour ranger des blocs cote a cote en CSS moderne, on utilise souvent :

- A) Des tableaux de mise en page
- B) Flexbox (`display: flex`)

- C) Dix `<br>`

Q12. Pour que la page s'adapte au telephone, tu as besoin notamment de :

- A) La meta viewport
- B) Un deuxieme site separe obligatoire
- C) Supprimer toutes les images

Les deux dernieres verifient Flexbox et responsive. Si tu hesites, les chapitres 11 et 12 meritent dix minutes.

Reponses

1. 1. **B** - HTML structure le contenu. Le CSS habille.
2. 2. **A** - CSS = style, couleurs, tailles, rangement.
3. 3. **B** - Le contenu visible vit dans le `<body>`.
4. 4. **B** - Un seul `h1` principal par page en general.
5. 5. **C** - `href` donne la destination du lien.
6. 6. **B** - `alt` decrit l'image pour l'accessibilite et le fallback.
7. 7. **A** - `ul` + `li` pour une liste a puces.
8. 8. **B** - Label lie au champ : cliquer le texte focus le champ.
9. 9. **B** - Fichier CSS separe + `<link>` : clair et maintenable.
10. 10. **B** - Padding = air a l'interieur. Margin = air a l'exterieur.
11. 11. **B** - Flexbox range les enfants du conteneur.
12. 12. **A** - Viewport d'abord, puis largeurs souples et media queries.

Score

- 10-12 : propre, le socle tient. Passe au bravo et montre ta page.
- 7-9 : bien, relis boites + Flexbox + responsive, puis reteste les floues.
- moins de 7 : refais les chapitres 1, 8, 10, 12 avec les "A toi", puis reviens.

Petite histoire

Sam rappelle : le score mesure un instant, pas ta valeur. Lea a rate des QCM en formation et livre proprement aujourd'hui. Max a eu 8 au premier passage, a relu le viewport, a refait a 11. Il a souri. Toi aussi, tu peux. Le chiffre est une carte, pas un jugement. Ce qui compte, c'est ce que tu sais refaire avec les doigts.

En vrai

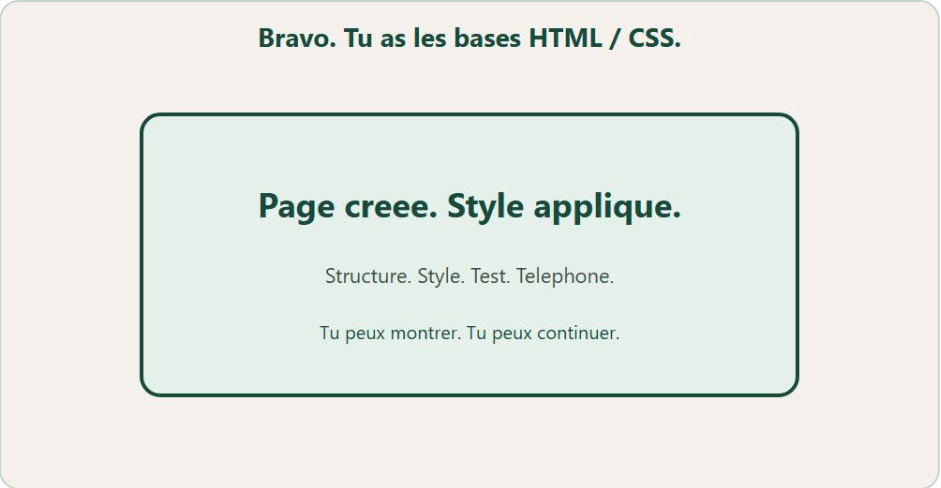
Sans regarder les corriges, liste tes 3 questions les plus floues. Cinq minutes de relecture par chapitre lie. Puis reteste ces 3 seulement. L'important n'est pas d'etre parfait du premier coup. C'est de progresser a chaque passage. Ensuite, lis le bravo. Tu l'as merite.

A toi

Note ton score, tes 3 hesitations, et la date. Dans une semaine, reteste les questions ou tu as doute. Si tu passes de 7 a 11, tu verras la progression concrete. Style DanielCraft : petit, mesurable, honnete.

FINAL

Bravo.



Bravo. Tu es alle au bout. C'est deja une vraie victoire.

Serieusement : bravo.

Tu es alle au bout d'un vrai premier livre sur le web. Pas juste "j'ai regarde une video de quarante minutes en mangeant des chips". Tu as lu. Tu as essaye. Tu as (surement) plante au moins une fois. Puis tu as continue. C'est exactement comme ca que ca marche. Chez DanielCraft, on celebre ce geste plus que la perfection du premier jet : tu as tenu le parcours du debut a la fin, avec les mains dans le code. Tu as cree. Tu as debugge. Tu as pense telephone. Tu as pense accessibilite un peu. Serieux : bravo.

Lea se souvient encore de sa premiere page moche et fiere, avec des couleurs qui piquaient les yeux et un menu casse sur mobile. Max a montre la sienne a sa famille comme un meuble qu'il aurait fabrique au garage : imparfait, mais a lui. Sam applaudit les eleves qui ont debugge sans abandonner, meme ceux qui ont fini le quiz a 6/12 et sont revenus la semaine suivante avec une page plus propre. Tu es dans cette famille-la maintenant. Pas "aspirant developpeur". Quelqu'un qui a deja livre du vrai HTML et du vrai CSS.

En 2026, quand quelqu'un dit "je sais faire une page web", il parle souvent de ce que tu viens de faire. Derriere, il y a JavaScript, Grid, la mise en ligne. Pour toi, le geste fondateur est la : structure, style, test, telephone. Tu restes le pilote. La page existe parce que tu l'as ecrite. Garde ce reflexe.

> **★ A RETENIR**
>
> Tu sais creer une page HTML propre et l'habiller en CSS. C'est deja enorme pour un debut. Garde le geste.

Ce que ce n'est pas

Ce n'est pas la fin de l'apprentissage web. Loin de la : JavaScript, Grid, mise en ligne, accessibilite avancee, tout ca t'attend si tu veux. Ce n'est pas un diplome magique qui ouvre des portes toutes seules. Ce n'est pas non plus "tu dois enchainer JavaScript ce soir avant de dormir". Si tu es fatigue, tu te reposes. Tu l'as merite. La suite viendra quand tu seras chaud, pas quand tu te sentiras coupable.

Tu as pose les murs et peint une premiere piece. La maison peut grandir : plus de pieces, plus d'electricite (**JavaScript**), plus de visiteurs (mise en ligne). Mais la piece existe deja. Tu peux y entrer. Tu peux l'ameliorer ce week-end. Tu peux la montrer a quelqu'un. Ce n'est pas une maquette dans ta tete : c'est du concret sur ton disque dur. Max a imprime une capture. Lea envoie ses vieilles pages aux stagiaires. Sam dit : le bravo solidifie le courage.

Ce que tu sais faire maintenant

Tu sais creer une page HTML propre avec du sens : titres, paragraphes, liens, images, listes, formulaires. Tu habilles avec du CSS dans un fichier separe. Tu comprends les boites. Tu ranges avec Flexbox. Tu penses telephone avec viewport et media queries. Tu as des reflexes d'accessibilite de base. Tu as une methode de debug. C'est deja enorme. Beaucoup de gens "interesses par le web" n'ont jamais pousse jusqu'ici.

Petite histoire

Lea envoie parfois ses anciennes pages debutantes a des stagiaires stresses : "regarde d'ou je viens". Ca detend plus qu'un discours. Max a imprime une capture de sa page vitrine et l'a mise au garage. Quand un client doute, il montre : "je l'ai faite moi-meme". Sam dit en fin d'annee : le bravo n'annule pas les erreurs futures ; il solidifie le courage de les affronter. Trois scenes, une famille. Tu en fais partie.

Petite mission (si tu veux)

Reprends ta page perso du chapitre 13. Ameliore juste trois trucs : une meilleure couleur de contraste, une image avec un vrai `alt` descriptif, un menu qui marche bien sur petite largeur. Montre-la a quelqu'un. Meme a ton chat. Un humain te dira ce qui est clair. C'est de l'or gratuit. Chez DanielCraft, montrer solidifie autant que coder.

La suite, quand tu seras chaud

JavaScript pour rendre la page un peu vivante : menu qui s'ouvre, bouton qui reagit, formulaire qui valide. Puis peut-etre publier en ligne. Mais pas ce soir si tu es vide. Repose-toi. On forme des gens qui livrent petit, souvent, proprement - pas des collectionneurs de tutos oublies au fond d'un onglet.

En vrai

Ecris trois lignes sur papier : ce dont tu es fier apres ce livre, ce que tu veux revoir, la date a laquelle tu ouvriras le livre JavaScript (ou tu amelioreras encore ta page perso). Colle ce papier quelque part visible. Encore bravo. Et a bientot pour la suite. Tu l'as merite pour de vrai.

A toi

Prends une photo de ta page perso ouverte dans le navigateur. Garde-la. Dans six mois, refais la meme photo apres ameliorations. Tu verras ta progression mieux qu'avec n'importe quel score de quiz. Style DanielCraft jusqu'au bout : petit, clair, visible, honnete.

Tu as les briques. Maintenant, construis.