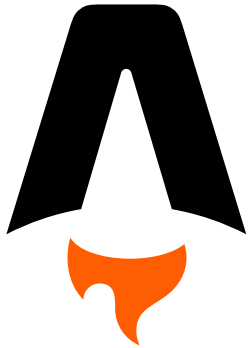
# FORMATION



astro

**Par ValT**

**1ère formation inversée**

# C'est quoi ?


- Astro est un framework de développement WEB

- Dans un seul fichier en .astro, on peut gérer le HTML, le CSS et le javascript

# Particularités

## Il a comme particularité d'être très modulaire. On peut créer :

- des layouts : des templates de page qu'on peut réutiliser à l'infini ==> Pour éviter de copier-coller du code pour des pages qui ont une mise en page similaire.
    - Exemple : Sur le site conference.minet.net, on a un unique layout qui est utilisé pour les pages des 10 conférences
- des components : du code en astro qu'on peut insérer à pleins d'endroits différents dans n'importe quelle page. On peut leur faire passer des arguments, ce qui permet de modifier certains paramètres en fonction du contexte
    - Exemple : on peut créer un component "Button.astro" qui est un bouton stylisé, et qui prend en argument une chaine de caractère pour modifier le texte affiché sur le bouton


# Avantage de la modularité

L'avantage d'un framework modulaire, c'est qu'il suffit de modifier une fois un layout ou un component pour modifier toutes les pages qui les utilisent. C'est très utile si on veut rapidement mettre à jour graphiquement un site.

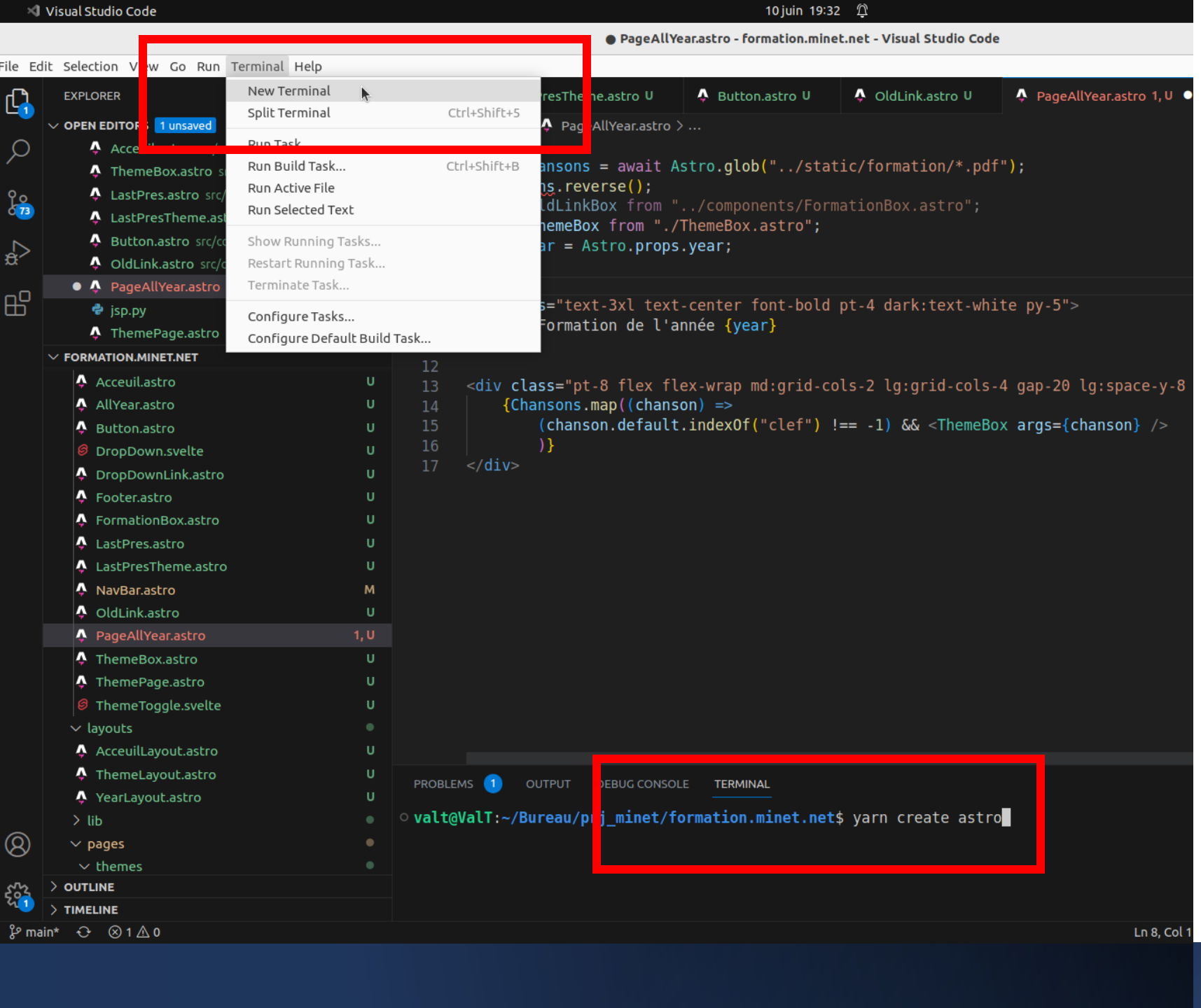
# Setup un projet en Astro

**1ère étape : Avoir un bon environnement de travail**

**Je recommande bien évidemment VSCode qui est de base très pratique pour ce type de projet, mais qui permet en plus de télécharger des extensions très utiles pour travailler en astro**

**Les 4 extensions VSCode à avoir dans l'idéal sont :**

- **L'extension Astro**
- **L'extension Svelte**
- **L'extension Tailwind CSS Intellisense**
- **L'extension MDX**




# Setup un projet en Astro

2ème étape : Créer votre projet

Dans VSCode, ouvrez un terminal Bash et tapez la commande :

**Yarn create astro**

Puis sélectionnez le dossier où créer votre projet.

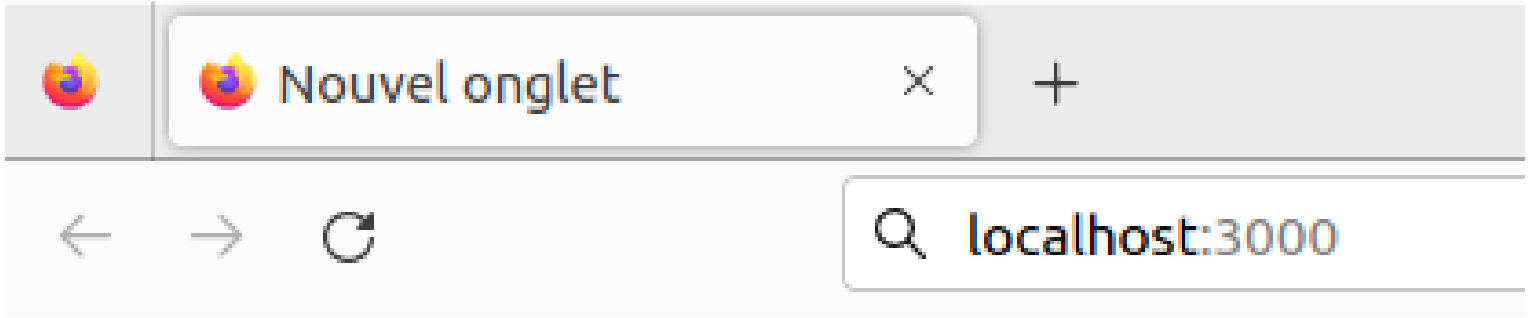
Ensuite... C'est tout ! Votre projet est déjà fonctionnel.

# Setup un projet en Astro

Pour le consulter en local, tapez dans le terminal :

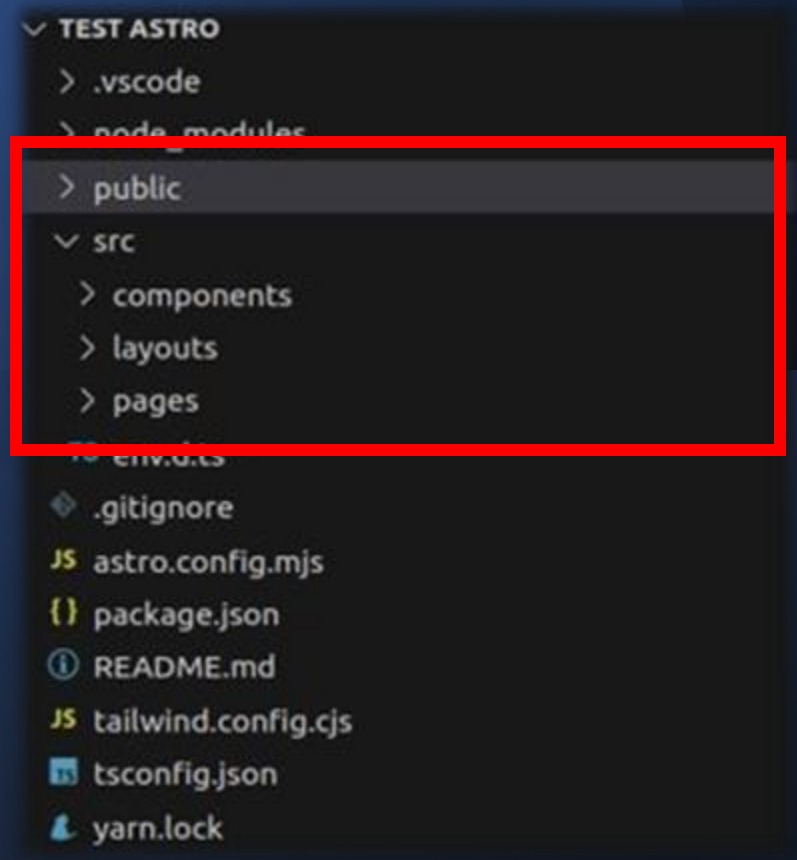
Yarn dev (à refaire à chaque ouverture de VSCode)

La page racine de votre projet est consultable sur votre moteur de recherche à l'adresse : http://localhost:3000/
(le port peut varier, il est indiqué dans la console)




Chose intéressante, à chaque ctrl + S, votre projet est automatiquement mis à jour sur votre moteur de recherche. On peut voir ses modifications en direct.

# Les principaux dossiers du projets




A la création du projet, différents dossiers et fichiers sont déjà apparus. Je vous présente les plus importants :

- Dossier public : c'est là qu'il faut mettre les fichiers que vous voulez afficher sur votre site (photo, vidéo, document à télécharger

- Dossier Src : dedans se trouve :
  - un dossier pages : Chaque fichier .astro à l'intérieur correspond à une page. En particulier, la page index.astro correspond à la racine de votre site (la page d'acceuil)

  - Un dossier layout : Chaque fichier .astro à l'intérieur représente un template de page réutilisable

  - Un dossier components : Chaque fichier .astro à l'intérieur représente un component que vous pouvez insérer dans n'importe quelle page/layout

# Comment sont rootés les pages ?

**Le routage en web, c'est savoir quelle URL vous devez taper pour accéder à une page de votre site. Cela dépend entièrement de la position de la page dans les dossiers de votre projet.**

**Voici quelques exemples :**

| Emplacement dans les dossiers | Adresse URL |
|---|---|
| src/pages/index.astro | http://monsite.com |
| src/pages/about.astro | http://monsite.com/about |
| src/pages/about/index.astro | http://monsite.com/about |
| src/pages/about/me.astro | http://monsite.com/about/me |
| src/pages/posts/1.md | http://monsite.com/posts/1 |

# Comment fonctionne le HTML ?

Exactement pareil que du HTML classique

Pour le coup ça bouge pas

Là ça change.
Le CSS est directement intégré dans le HTML. Pas de fichier externe supplémentaire

Pour donner un style à un bloc, il faut faire comme ceci :

"<bloc class="Tailwind CSS"> Texte stylisé affiché sur le site </bloc>"

Chaque bloc possède donc ses propres caractéristiques

```
<div class="px-2 py-5">
    <h3 class="text-4xl text-center font-bold pt-4 dark:text-white">Dernière Formation en Date</h3>
        <img src={vraiAffiche.default} class=" mx-auto rounded-lg mt-5 mb-5" style="width:50%"/>


        <div class="text-center mt-4 py-4 text-gray-900 dark:text-white">
            <a class="inline-flex items-center gap-2 bg-gray-100 border border-gray-300 focus:outline-

                <img src={downloadDark} class="w-10 h-10 hidden dark:block">
                <img src={download} class="w-10 h-10 dark:hidden">
                <span class="ml-2 font-bold text-2xl ">Accès à la Formation</span>
            </a>
        </div>
</div>
```

Un bloc hérite du CSS des blocs qui le contiennent

# Comment fonctionne le CSS ?

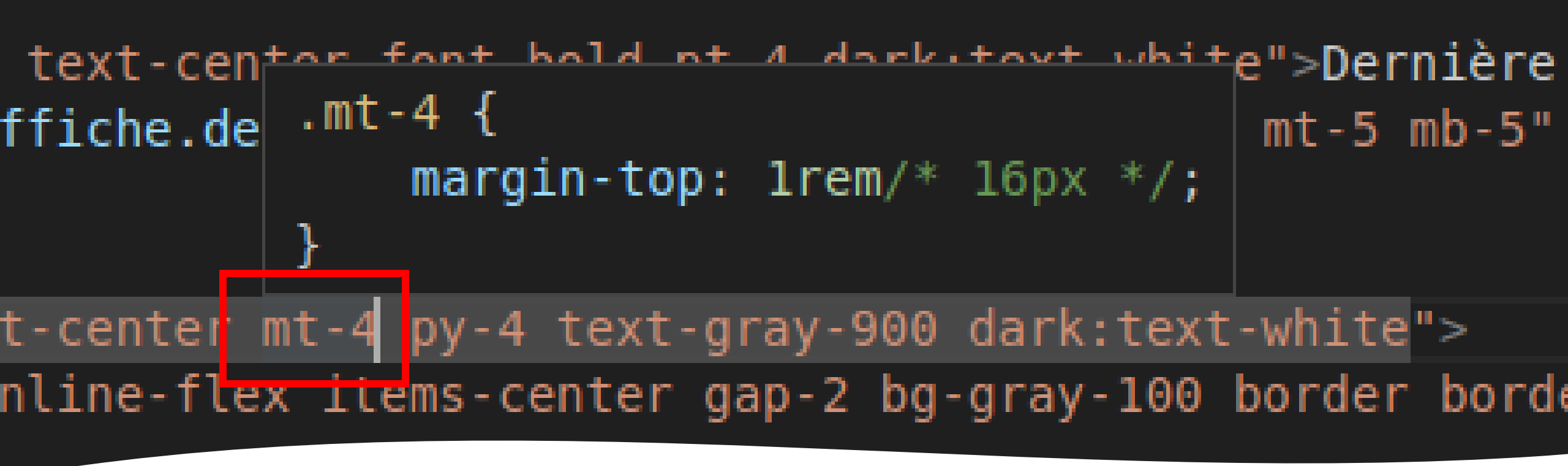
# Comment fonctionne le CSS

En astro, il ne s'agit pas de CSS classique, mais de


Cela permet de faire pareil que le CSS, mais sous une forme plus synthétique

Exemple :

- "width : 40px" devient "w-16"
- "text-align : center" devient "text-center"
- "px-5" = "pr-5" + "pl-5" = ajoute 20px de padding à droite et à gauche
- "my-2.5" = "mb-2.5" + "mt-2.5" = ajoute 10px de margin en haut et en bas




# Comment fonctionne le CSS

Avec l'extension VSCode "Tailwind CSS intellisense", vous n'avez pas besoin d'apprendre par coeur les changements. Elle autocomplète votre argument CSS. Et en passant votre souris sur un argument CSS, ça vous affiche l'équivalent en CSS classique

Mais en cas de doute, la liste de tous les changements CSS <--> Tailwind sont sur : https://tailwindcss.com/docs/installation

ChatGPT pourra également vous aider, il connait Tailwind CSS.

# Problème avec Tailwind CSS

A noter : en tailwind, des valeurs pour certains arguments ne sont pas disponibles et vous donneront des erreurs (ou bien votre code fera comme si cet argument n'existait pas)

Par exemple, vous pouvez marquer "w-16", mais pas "w-17", car cette valeur précise n'est pas disponible.

Pour ne pas se tromper en utilisant des valeurs qui n'existent pas, utilisez l'autocomplétion ou vérifiez que quelque chose apparait quand vous passez votre souris dessus.

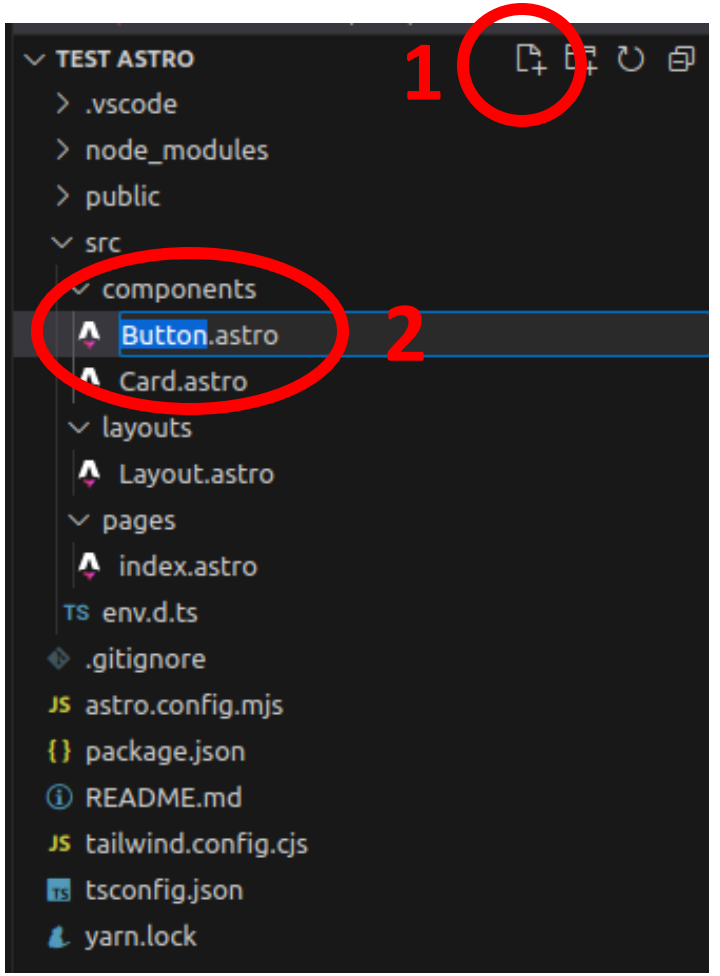
# Comment créer un component

Comme je l'ai expliqué, le but du component est d'écrire une fois du code que l'on peut réinsérer autant de fois que l'on veut

Pour cela, il faur créer un nouveau fichier dans le dossier "component" renommé sous la forme :

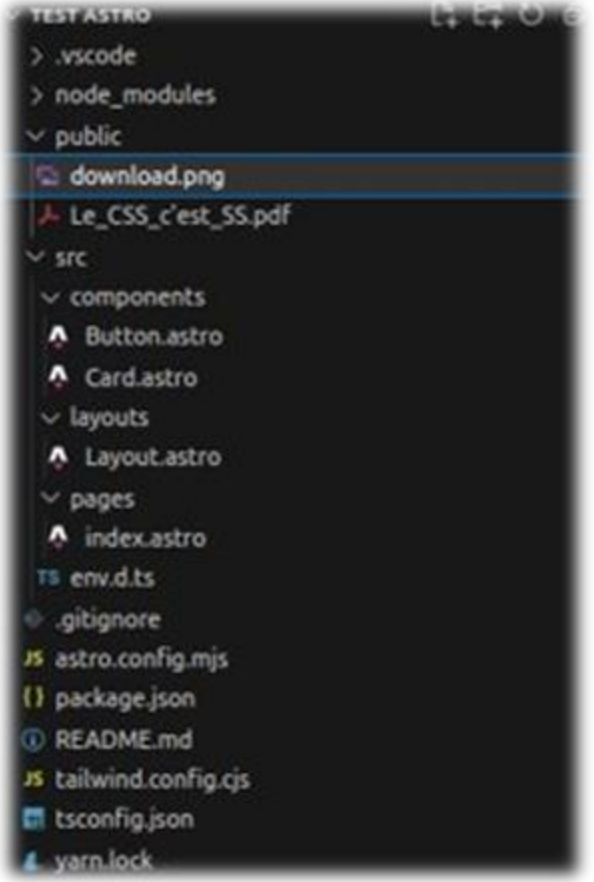
"Nom_Component.astro"

Attention ! Le nom de vos components et layouts doivent toujours commencer par une MAJUSCULE !



# Importer des documents dans votre code

On en profite pour voir comment importer des fichiers du dossier "public", pour les utiliser dans le code. Par exemple un document pdf ou une image


Là j'ai créé un component Button.astro pour faire un joli bouton cliquable avec une icone de téléchargement et un texte :

Pour importer un component, tapez la commande suivante entre des triples tirets au début de votre document :


Notez que pour aller chercher l'emplacement du component, on utilise la même logique qu'en bash. On est initialement dans le dossier où l'on veut importer le component. Les .. servent à revenir en arrière dans la hiérarchie des fichiers pour entrer dans le dossier components où se trouve "Button.astro"

Il est également possible d'importer des components dans des Layouts ou dans d'autres components. Les possibilités sont uniquement limitées par votre imagination.

# Comment utiliser un component

# Les triples --- ---

**Vous avez remarqué qu'on utilisait déjà les triples tirets au début du document pour aller chercher des fichiers dans public**

**En réalité, on peut faire encore bien plus de chose à l'intérieur. C'est là que vous allez placer tous ce qui n'est pas du HTML/CSS : le code en rapport avec astro, vos variables, vos fichiers, et même des fonctions en javascript**

src > components > LastPresTheme.astro > ...

```
1   ---
2   import download from '/download.png';
3   import downloadDark from '/downloadDark.png';
4
5   const {frontmatter} = Astro.props;
6   const formations = await Astro.glob("../static/formation/*.pdf");
7   const affiches = await Astro.glob("../static/affiche/*");
8
9   var vraiAffiche = affiches[0].default;
10
11  formations.reverse();
12
13  {affiches.map((affiche) => {
14      if(affiche.default.indexOf(frontmatter.identifiant) !== -1) {vraiAffiche = affiche.default}})}
15
16  ---
```

# Revenons en à "Comment utiliser un component"

**Enfait il reste plus grand chose à faire**

```
37
38    <Button/>
39
```

**Voilà votre Button inséré au milieu du code HTML**

**Notez qu'un component inséré dans un bloc hérite de son CSS :**

```
36
37    <div class="mt-16 pb-10">
38        <Button/>
39    </div>
40
```

# Passer des arguments dans vos components

Avoir du code réutilisable c'est bien !

Avoir du code personnalisable, c'est mieux !

J'aimerai bien réutiliser du code mais en changeant juste un petit paramètre

Etape 1 : Créer un argument dans votre component

src > components > Button.astro > ...

```
1   ---
2   import download  from "/download.png"
3
4   //Si un seul argument
5   const {image} = Astro.props;
6
7   //Si plusieurs arguments
8   const {text,file} = Astro.props;
9   ---
```

# Passer des arguments dans vos components

**Etape 2 :** Utiliser vos arguments dans votre component (je reprends mon bouton)

src > components > Button.astro > ...

```
1   ---
2   import download  from "/download.png"
3
4   //Si un seul argument
5   const {image} = Astro.props;
6
7   //Si plusieurs arguments
8   const {text,file} = Astro.props;
9   ---
10
11
12  <div class="text-center mt-4 py-4 text-gray-900 dark:text-white">
13      <a class="inline-flex items-center gap-2 bg-gray-100 border border-gray-300 rounded-full px-5 py-2.5 mr-2 mb-2" href={file} target="_blank">
14
15          <img src={download} class="w-10 h-10 dark:hidden">
16          <span class="ml-2 font-bold text-2xl ">{text}</span>
17      </a>
18  </div>
```

**Toutes les variables définies dans l'en-tête du fichier s'utilisent avec des accolades { } dans le code HTML**

src > pages > index.astro > ...

```astro
---
import Button from '../components/Button.astro';
---

<div class="mt-16 pb-10">
    <Button text="Astro Décollage !" file="/Le_CSS_c'est_SS.pdf" />
</div>

```

# Passer des arguments dans vos components

**Etape 3 :** Utiliser votre component avec des arguments

# Important : { } = Javascript

Depuis tout à l'heure, j'utilise des variables en les plaçant entre accolade { }

En réalité, TOUS ce qui se trouve entre accolade est interprété comme du Javascript dans un fichier .astro

C'est assez intéressant d'en être conscient car ça permet de faire des choses plus complexe directement dans votre code HTML

Vous avez accès à tout les outils qu'offre le langage javascript très rapidement et directement intégré à votre HTML

# A quoi servent les layouts ?

Le component, c'est un contenu que l'on peut ajouter n'importe où.

Le Layout, c'est plutôt la boite que l'on va utiliser pour contenir des choses.

En théorie, il n'y a pas vraiment de différence entre un component et un Layout : ce sont tous les 2 des fichiers en .astro. Mais en pratique ça rend les choses beaucoup plus clairs.

Le but du Layout, c'est de contenir la mise en page globale, la barre de navigation, le pied de page, et tout autre élément qui pourrait servir sur plusieurs pages différentes.

L'intérêt du Layout apparait clairement quand vous créer un site internet avec de nombreuses pages : vous créez un Layout une fois, et il pourra resservir pour la plupart des pages.

Vous pouvez même utiliser un Layout dans un autre Layout. Cela permet d'avoir un Layout de base qui définit le style global du site. A l'intérieur vous rajoutez un Layout plus spécifique pour un certain type de page. Et dedans vous ajoutez vos components.

Avec ce système, sauf exception, chaque page doit littéralement uniquement contenir, un ou plusieurs Layouts, avec à l'intérieur des components.

# Comment utiliser un layout ?

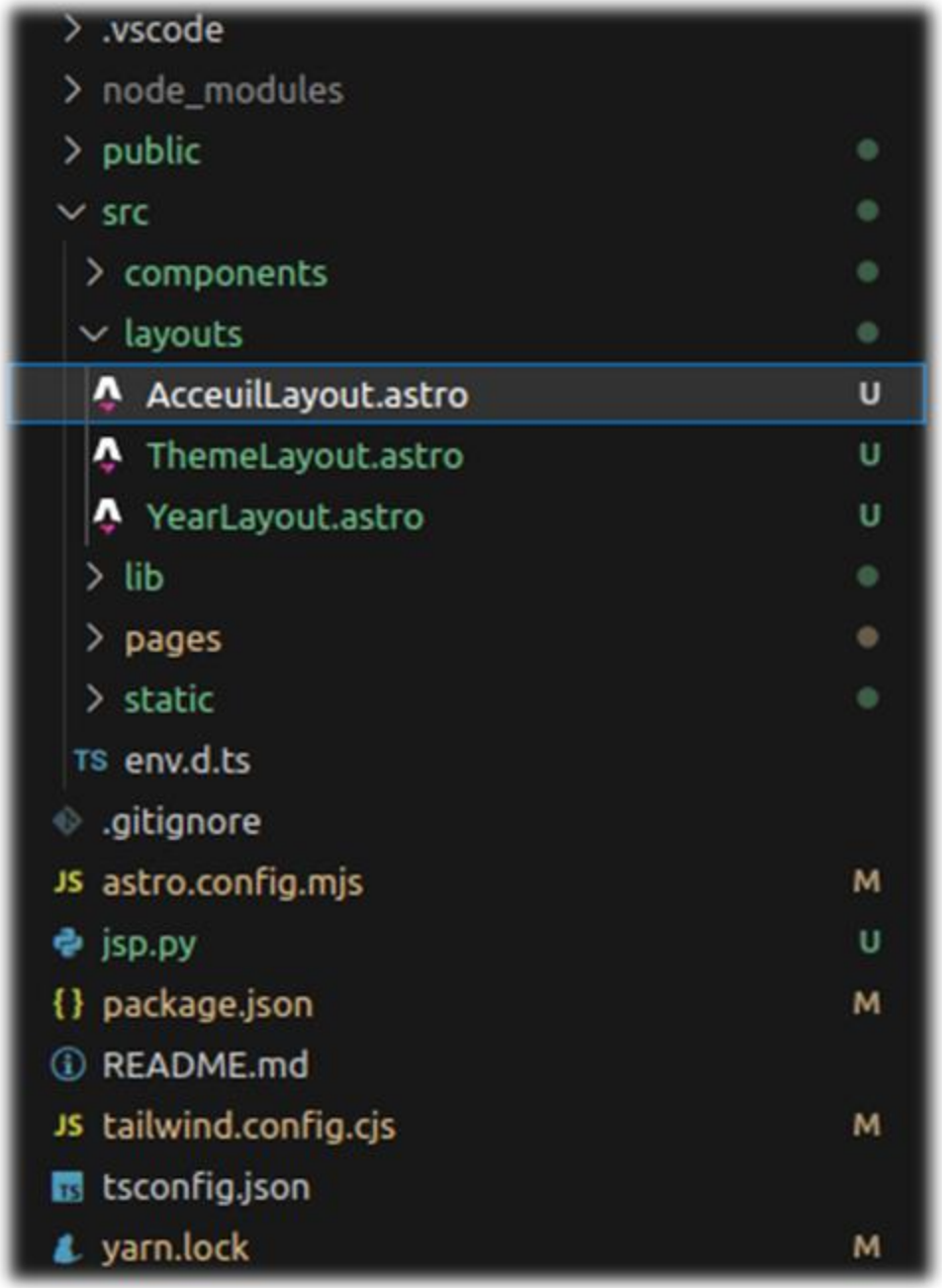
1ère étape : il faut créer le Layout.

Tout comme un component, mais dans le dossier "Layout", vous allez créer un fichier renommé de la sorte :

"Nom_Du_Layout.astro"

Attention : votre nom de layout DOIT commencer par une MAJUSCULE

Ici j'ai créé 3 layouts : un Layout principal "AcceuilLayout" que j'utiliserai pour chaque page, et 2 Layouts secondaire "ThemeLayout" et "YearLayout" que je vais rajouter au premier pour des pages spécifiques



src > layouts > AcceuilLayout.astro > ...

```astro
---
import NavBar from "../components/NavBar.astro";
import Footer from "../components/Footer.astro";
---

<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,initial-scale=1">
        <title>Formation MiNET</title>
        <meta name="description" content="Stockage des formations de MiNET">
        <meta property="og:title" content="Accueil">
        <link rel="stylesheet">
        <link rel="shortcut icon" href="/favicon.png">
        <meta property="og:description" content="Accueil du site des formations de MiNET">
        <script is:inline>
            // On page load or when changing themes, best to add inline in `head` to avoid FOUC
            if (localStorage.getItem('color-theme') === 'dark' || (!('color-theme' in localStorage) && w
                document.documentElement.classList.add('dark');
                localStorage.setItem('color-theme', 'dark');
            } else {
                document.documentElement.classList.remove('dark');
                localStorage.setItem('color-theme', 'light');
            }
        </script>

    </head>
    <body class="flex flex-col min-h-screen bg-white dark:bg-gray-900">
        <div class="pb-40">
            <NavBar title="Navbar" />
        </div>
        <slot />
        <div class="grow invisible"></div>
        <Footer />
    </body>
</html>
```

# Comment utiliser un layout

**2ème étape :** coder votre Layout

Ça, c'est le Layout "AcceuilLayout" du site formation.minet.net. C'est mon Layout principal avec :

les données générales du site, un script pour le mode jour/nuit, une NavBar et un Footer.

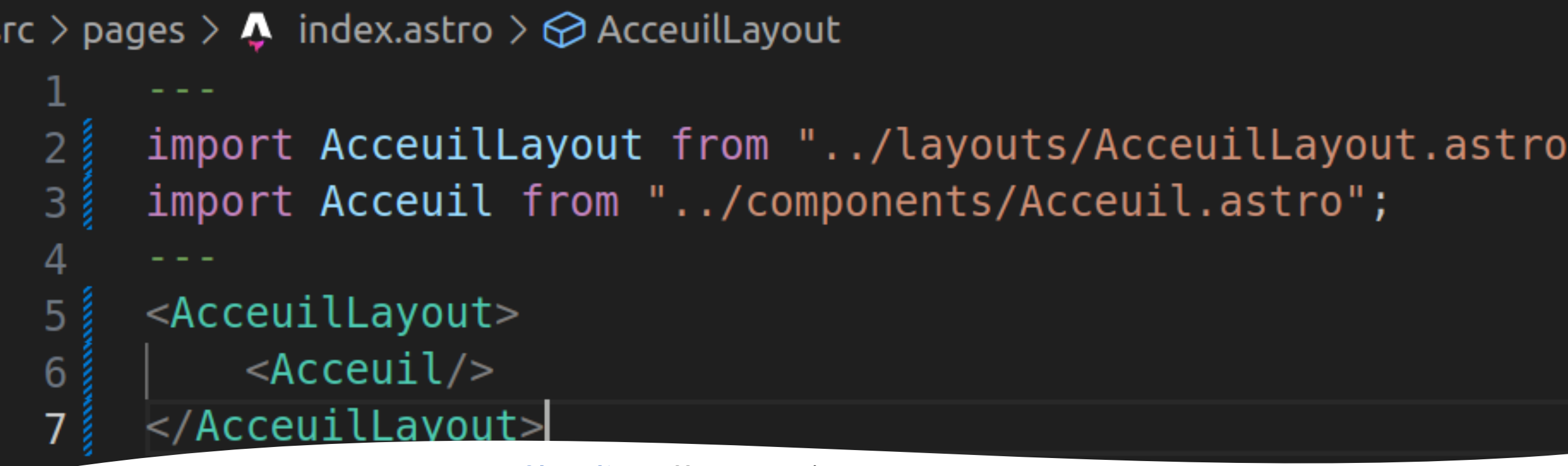
# Comment utiliser un layout

```
29      <body class="flex flex-col min-h-screen
30          <div class="pb-40">
31              <NavBar title="Navbar" />
32          </div>
33          <slot />
34          <div class="grow invisible"></div>
35          <Footer />
36      </body>
37  </html>
```

- Ce qu'il est important de remarquer, c'est la balise <Slot/>

- Comme je l'ai dis, le Layout est un contenant, un template réutilisable avec une mise en page globale. Dedans on va pouvoir mettre du contenu spécifique à chaque page qui l'utilise.

- La balise <Slot /> correspond à l'endroit où on va insérer le contenu




# Comment utiliser un layout

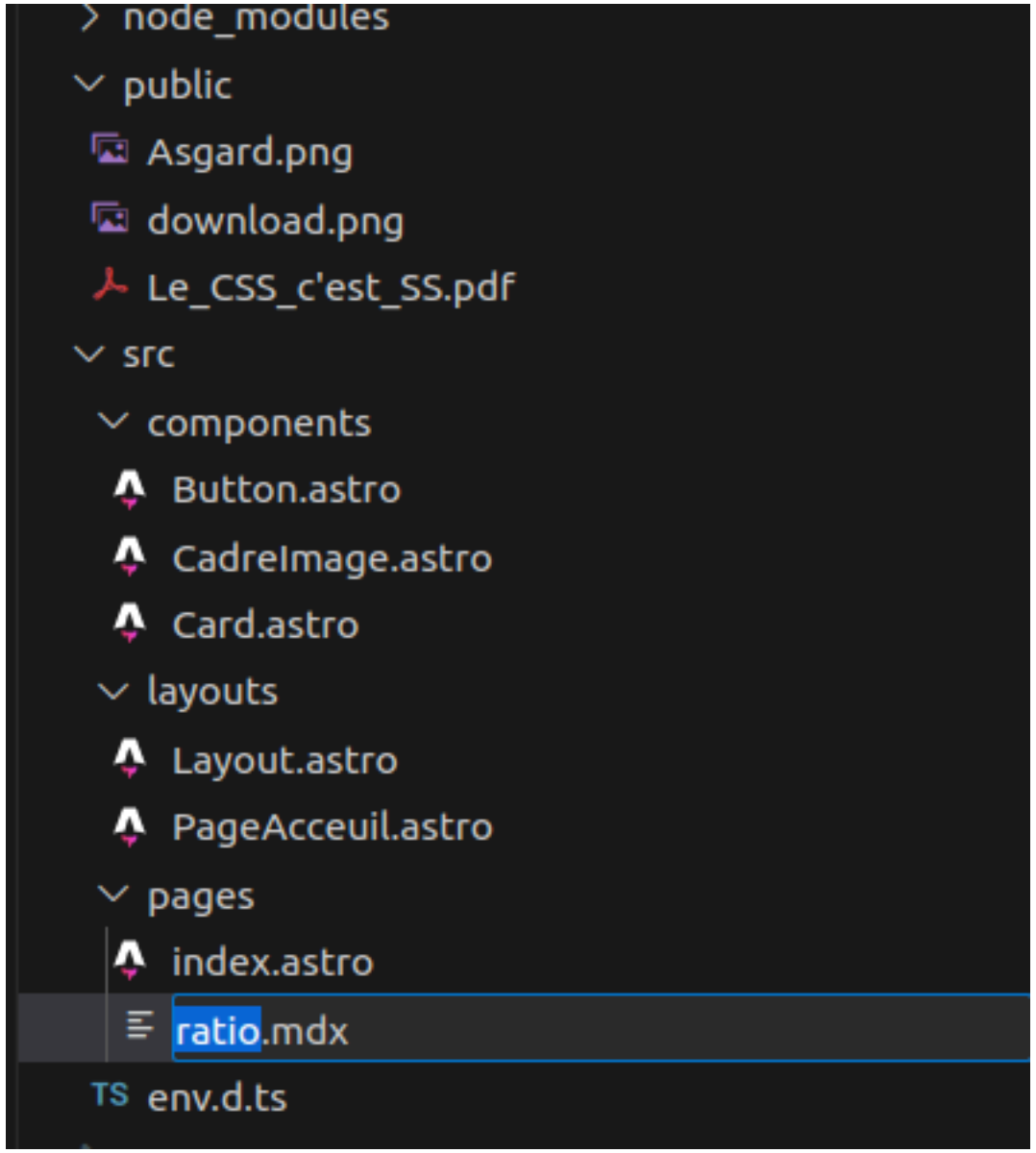
**3ème étape :** Mettre votre Layout sur une page

Vous importez votre Layout comme vous importeriez un component (en allant ce coup-ci le chercher dans le dossier Layout).

Puis vous créer simplement une balise ouvrante et fermante avec le nom d'importation de votre layout (Ici AcceuilLayout)

Entre les 2 Vous mettez le contenu à insérer dans le layout, qui apparaitra à la place de la balise <Slot />

Perso ce que j'aime bien faire c'est avoir un component qui correspond à chaque page. Donc forcément, à la fin, il ne reste pas grand-chose dans les pages en elles-mêmes




# Des fichiers markdown en astro

Chose intéressante sur votre site, en plus d'avoir des pages en .astro, vous pouvez faire des pages en .mdx (en Markdown Extended)

On y accède pareil que les .astro :

http://NomSite/NomFichier

Le markdown c'est un langage web extrêmement simplifié pour créer des pages internet avec que du texte. Ca ne permet pas de faire des trucs complexes, mais c'est un gain de temps considérable si vous n'avez pas besoin de plus.

Vous pouvez regarder la page wikipedia si vous voulez comprendre comment ça fonctionne :

https://fr.wikipedia.org/wiki/Markdown

rc > pages > ratio.mdx

```
1   ---
2   layout: ../../layouts/Layout.astro
3   ---
4
5   import Image from '../components/CadreImage
6   import RagInt  from "/Asgard.png"
7
8   ## PAN PAN PAN
9   on ratio tous les astros
10
11  ## PAN PAN PAN
12  les astros puent la gastro
13
14  <Image src={RagInt} />
15
```

# Markdown Extended x Astro

Grâce au Mardown Extended, vous pouvez combiner markdown et astro

En haut de votre page, dans les triples ---, vous pouvez rajouter un de vos Layouts. Votre page .mdx utilisera ce Layout, et son contenu sera inséré à la place de la balise <Slot/>.

Il est même possible d'importer des components et des fichiers pour les ajouter dans votre page.

Par exemple ici, j'ai créé un component qui a l'apparence d'un cadre photo et qui prend en argument une photo dans mon dossier public.

src > pages > ratio.mdx

```
1  ---
2  layout: ../../layouts/Layout.astro
3  year: 2023
4  auteur: Nicolas R. Asgard MiNET Nukkaï
5  ---
```

# Des arguments en markdown

Si vous avez plusieurs fichiers markdown, vous pouvez créer des arguments.

Pour cela, écrivez dans chacun de vos fichiers markdown des variables dans les triples --- sous la forme :

"nomVariable: valeur"

Pour vous montrer comment utilisez ces arguments, il faut que je vous parle rapidement des boucles for en astro

src > components > Acceuil.astro > ...

```
1  ---
2  const Chansons = await Astro.glob('../pages/chansons/*.mdx');
3  ---
4
```

# Des boucles for sur des pages internet

Supposons que j'ai dans mon dossier "Page", j'ai un autre dossier "chansons" avec dedans pleins de fichiers .mdx qui correspondent tous à une page

J'ai donné à tous ces fichiers 2 variables "auteur" et "year".

Astro fourni alors une commande **await Astro.glob** qui permet d'**importer tous les fichiers contenus dans un dossier** sous la forme d'une unique variable (ça sera un tableau du coup) :

```
<div class="pt-8 flex flex-wrap md:grid-cols-2 lg:grid-cols-4 gap-20
    {Chansons.map((chanson) =>
        <ThemeBox args={chanson} />
    )}
</div>
```

Je peux désormais itérer sur mon tableau avec la fonction "map".

"chanson" est ici la variable de ma boucle for. Je vais placer, pour chaque fichier contenu dans "Chansons" , un component "ThemeBox" qui prend en argument un fichier .mdx.

A noter, la fonction map fonctionne à l'intérieur et en dehors des triples ---. Pensez juste à la mettre entre accolades { } quand vous êtes dans le code HTML.

# Des boucles for sur des pages internet

src > components > ThemeBox.astro > chanson

```
1   ---
2   const chanson = Astro.props;
3   ---
4
5   <div class=" w-56 hover:scale-125 ease-in-out duration-150 hover:drop-shadow-lg hover:bg-gray-5
6       <a href={chanson.url}>
7           <img class="w-48 my-3 mx-auto text-sky-600 dark:text-sky-300 rounded-2xl items-center"
8                   src={chanson.frontmatter.icon}
9                   alt={chanson.name}/>
10          <h3 class="mb-2 text-xl font-bold dark:text-white">{chanson.frontmatter.title}</h3>
11      </a>
12  </div>
```

# Utiliser les arguments des fichiers Markdown

On y arrive enfin ! Là je suis dans le component ThemeBox. Pour rappel, chanson contient ce que j'ai fait passer en argument dans mon component, ici une page .mdx

Pour utiliser mes variables "auteur" et "year" dans mon code HTML, j'ai juste à mettre entre accolade { } : "chanson.frontmatter.NomVariable"

A noter : il y a plusieurs variables par défaut dans ma page .mdx telles que :

- Son URL qui s'appelle : "chanson.url"
- Le nom du fichier qui s'appelle : "chanson.name"

# Comment obtenir l'URL d'un fichier quelconque

On a vu juste avant que les fichiers .mdx possèdent une variable par défaut qui contient son URL

Je le mets là parce que j'ai mis du temps à le comprendre : si vous avez un fichier qui n'est pas un fichier .mdx (par exemple une image ou un pdf situé dans le dossier "public"), la valeur par défaut pour son URL c'est :

"NomFichier.default"

Lorsque j'ai utilisé la fonction "map" sur "Chansons" 2 slides en arrière, j'ai itéré sur chacun des fichiers .mdx qu'elle contenait.

Dans certains cas, je voudrais peut être sélectionner seulement les fichiers qui respectent une certaine condition

Pour une raison obscure, les boucles if ne s'écrivent pas de la même manière à l'intérieur ou en dehors des triples ---. Voici la version dans les triples --- :

```
7   const affiches = await Astro.glob("../static/affiche/*");
8   var vraiAffiche = affiches[0];
9
10  affiches.map((affiche) => {
11      if(affiche.default.indexOf("clef") !== -1) {vraiAffiche = affiche.default}
12  })
```

Je vais ici chercher à récupérer UNE affiche qui contient le mot "clef" dans son URL.

IndexOf cherche si un string contient un autre string, et renvoie "–1" si ce n'est pas le cas

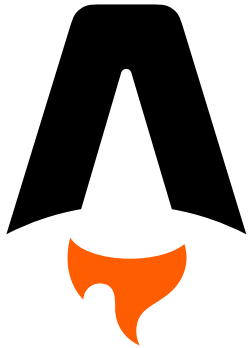
# Les boucles if en astro

```
12
13  <div class="pt-8 flex flex-wrap md:grid-cols-2 lg:grid-cols-4 gap-20 lg:space-y-8
14      {Chansons.map((chanson) =>
15          (chanson.default.indexOf("clef") !== -1) && <ThemeBox args={chanson} />
16          )}
17  </div>
```

**Version dans le code HTML (pour une raison obscure, j'ai l'impression qu'on ne peut pas mettre autre chose qu'un component si la condition est validée) :**

# Les boucles if en astro

# FORMATION



astro

**Par ValT**

**1ère formation inversée**