Table des matières

PROGRAMMATION EN PYTHON

ELECTRONIQUE

INTELLIGENCE ARTIFICIELLE

SYSTÈMES D'EXPLOITATION

CANSAT

TECHNOLOGIES

GESTION DE PROJETS

PROGRAMMATION EN PYTHON

Python

> *Python est un langage de programmation interprété, polyvalent et convivial qui a gagné en popularité au fil des ans. Créé par Guido van Rossum et publié pour la première fois en 1991, Python est devenu l'un des langages les plus utilisés dans le monde de la programmation en raison de sa simplicité, de sa lisibilité et de sa grande communauté de développeurs.*

Les avantages de Python

Simplicité et lisibilité

L'une des principales caractéristiques de Python est sa syntaxe simple et concise, qui le rend facile à apprendre et à utiliser même pour les débutants. La lisibilité du code Python est également un avantage majeur, permettant aux développeurs de comprendre rapidement le fonctionnement d'un programme.

Polyvalence

Python est un langage polyvalent qui peut être utilisé dans une grande variété de domaines, notamment le développement web, l'analyse de données, l'intelligence artificielle, l'automatisation, la robotique et bien plus encore. Sa polyvalence en fait un outil indispensable pour de nombreux développeurs et entreprises.

Grande bibliothèque standard

Python dispose d'une vaste bibliothèque standard qui offre des fonctionnalités prêtes à l'emploi pour diverses tâches de programmation. Cela permet aux développeurs d'économiser du temps en utilisant des modules et des packages déjà développés plutôt que de réinventer la roue.

Communauté active

La communauté Python est l'une des plus grandes et des plus actives dans le monde de la programmation. Cette communauté dynamique contribue au développement continu de Python en créant de nouveaux packages, en fournissant des ressources d'apprentissage et en soutenant les nouveaux arrivants.

Qui utilise Python et pourquoi ?

Développeurs web

Python est largement utilisé dans le développement web pour la création de sites web, d'applications web et de services web. Des frameworks populaires comme Django et Flask permettent de développer rapidement des applications web robustes et évolutives.

Scientifiques des données

Python est devenu le langage de choix pour de nombreux scientifiques des données en raison de ses puissantes bibliothèques d'analyse de données telles que NumPy, Pandas et Matplotlib. Ces outils facilitent l'exploration, la manipulation et la visualisation des données.

Ingénieurs en intelligence artificielle et en machine learning

Python est largement utilisé dans le domaine de l'intelligence artificielle et de l'apprentissage automatique en raison de sa facilité d'utilisation et de ses bibliothèques spécialisées telles que TensorFlow, Keras et scikit-learn. Ces bibliothèques permettent de développer et de déployer des modèles d'apprentissage automatique de manière efficace.

Développeurs de logiciels et d'applications

De nombreuses entreprises utilisent Python pour le développement de logiciels et d'applications en raison de sa polyvalence, de sa productivité et de sa capacité à s'intégrer facilement avec d'autres langages et technologies.

La popularité croissante de Python

Python connaît une popularité croissante grâce à ses nombreux avantages et à sa polyvalence. Selon divers indices de popularité, tels que l'indice TIOBE et l'enquête Stack Overflow Developer Survey, Python figure parmi les langages de programmation les plus populaires et les plus demandés dans l'industrie.

Sa popularité continue de croître en raison de son adoption par de grandes entreprises comme Google, Facebook, Amazon et Netflix, qui utilisent Python pour diverses applications et services.

En conclusion, Python est bien plus qu'un simple langage de programmation. C'est un outil puissant et polyvalent qui continue de gagner en popularité grâce à sa simplicité, sa lisibilité, sa grande communauté et sa polyvalence. Que vous soyez un débutant en programmation ou un développeur expérimenté, Python offre une multitude d'opportunités pour créer des applications innovantes et résoudre des problèmes complexes dans divers domaines.

Notes de cours

- Télécharger les notes de cours "officielles"

EXERCICES DU JOUR

- Exercices sur les variables
- Pour les courageux
- Exercices sur les conditions
- Exercices sur les boucles

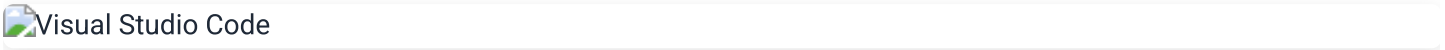
Vidéos "Python au lycée"

- Premiers pas: console, calculs, fichiers .py et print (6')
- Premiers pas: variables, commentaires (8'30)
- Les 4 types de variables de base
- Si...alors...(sinon) (partie 1) structure et condition (8'50)
- Entrée au clavier: input(), int() et float() (9'44)
- Si...alors...(sinon) (partie 2) (5'20)
- Booléens, comparaisons, random() (8')
- Nombres au hasard ("aléatoires") (5'40)
- Les fonctions (Partie 1) (5'38)
- Les fonctions (Partie 2) (5'38)
- Les boucles **for** (5'38)
- Les boucles **for... in range()** (5'38)

Python: Les outils pour développer

> *Lorsque vous vous lancez dans le développement en Python, choisir les bons outils peut faire toute la différence dans votre productivité et votre expérience de développement. Dans cet article, nous allons passer en revue certains des outils les plus populaires utilisés par les développeurs Python et discuter de leurs principales caractéristiques.*

1. Visual Studio Code (VS Code)


Visual Studio Code

Visual Studio Code est un éditeur de code léger et puissant développé par Microsoft. Il offre une multitude de fonctionnalités pour les développeurs Python, notamment :

- **Prise en charge de Python intégrée :** VS Code propose une prise en charge intégrée de Python avec des fonctionnalités telles que la coloration syntaxique, la saisie semi-automatique, le débogage et la gestion des environnements virtuels.
- **Extensions Python :** Vous pouvez étendre les fonctionnalités de VS Code en installant des extensions Python, telles que Python, Pylance, ou encore Anaconda Extension Pack, qui ajoutent des fonctionnalités avancées.
- **Personnalisable :** VS Code est hautement personnalisable avec des thèmes, des raccourcis clavier et des extensions pour répondre à vos besoins spécifiques de développement Python.

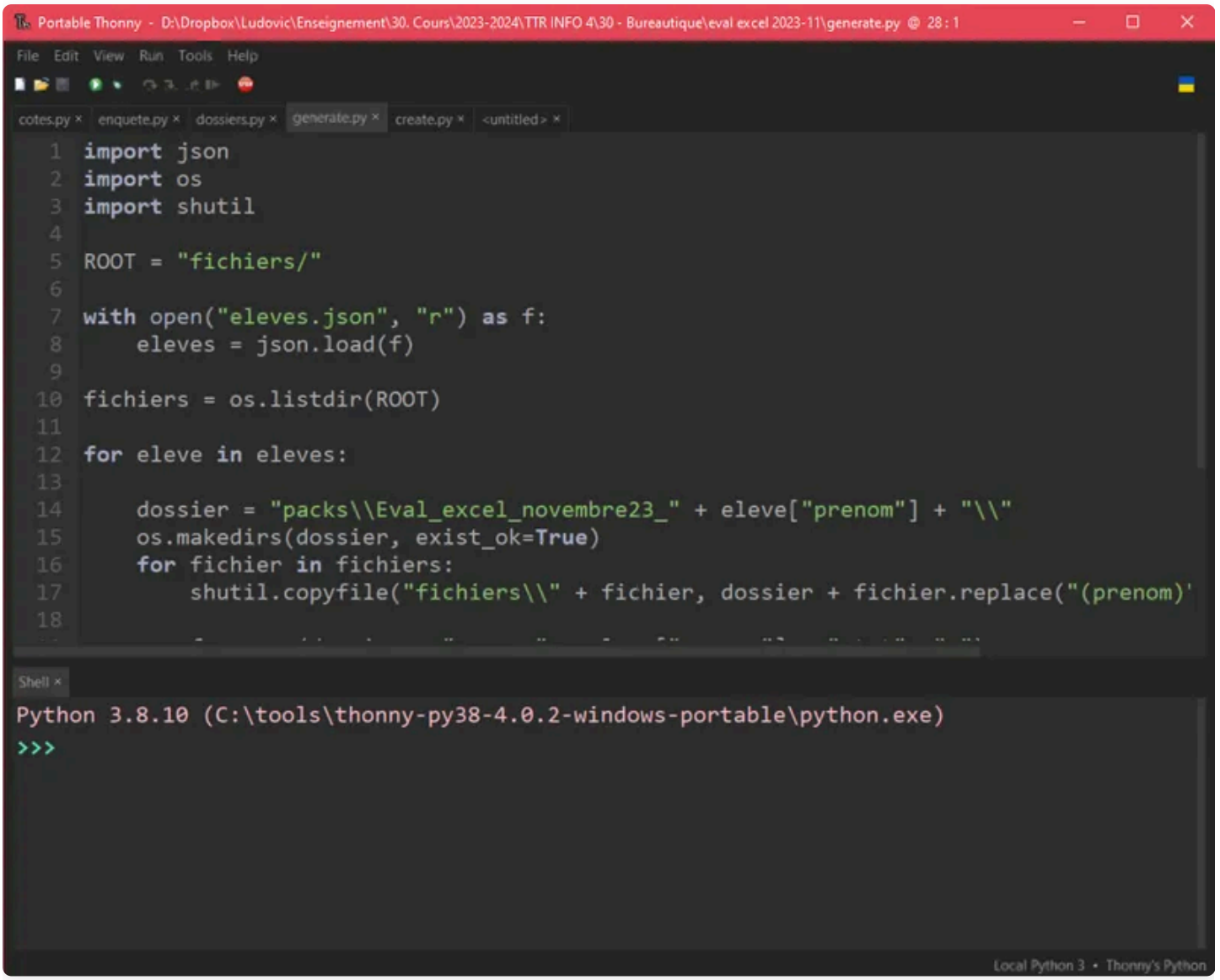
Configuration de Visual Studio Code pour Python

Pour configurer VS Code pour le développement Python, suivez ces étapes simples :

1. Installez l'**extension Python** depuis le marché des extensions de VS Code.
2. Configurez votre environnement virtuel Python en sélectionnant l'interpréteur Python souhaité dans les paramètres de l'extension.
3. Utilisez les fonctionnalités intégrées de débogage et de gestion des packages pour développer efficacement vos projets Python.

Avant de pouvoir utiliser VS Code pour développer en Python, **vous devez d'abord avoir installé Python sur votre ordinateur**.

2. Thonny Python



Thonny est un environnement de développement intégré (IDE) spécialement conçu pour les débutants en programmation Python. Ses caractéristiques principales incluent :

- **Interface utilisateur conviviale :** Thonny offre une interface utilisateur simple et intuitive qui facilite l'apprentissage de la programmation Python pour les débutants.
- **Débogage pas à pas :** Thonny propose une fonctionnalité de débogage pas à pas qui permet aux débutants de comprendre le comportement de leur code en le suivant ligne par ligne.
- **Explorateur de variables :** Thonny dispose d'un explorateur de variables qui affiche les valeurs des variables en temps réel pendant l'exécution du code.
- **Installation facile :** Thonny est facile à installer et est disponible pour Windows, macOS et Linux.

Un des avantages principaux de **Thonny**, c'est qu'il **est livré avec sa propre version de Python**. Il n'est donc pas nécessaire d'installer Python manuellement!

🔔 Pour tout savoir sur Thonny IDE et le télécharger, rendez-vous sur la page Thonny Python

3. www.online-python.com

Online Python est une plateforme en ligne qui offre un environnement de développement Python entièrement fonctionnel dans votre navigateur. Ses fonctionnalités comprennent :

- **Accès instantané :** Pas besoin d'installer quoi que ce soit. Il vous suffit d'ouvrir votre navigateur et de vous rendre sur le site pour commencer à coder en Python.
- **Prise en charge de la saisie semi-automatique :** Online Python propose une saisie semi-automatique qui facilite l'écriture de code Python.
- **Exécution de code en temps réel :** Vous pouvez exécuter votre code Python en temps réel et voir les résultats instantanément dans la sortie.
- **Partage de code :** Online Python vous permet de partager votre code avec d'autres personnes en générant simplement un lien.

Autres outils populaires pour le développement Python

En plus de Visual Studio Code, Thonny Python et www.online-python.com, il existe d'autres outils populaires utilisés par les développeurs Python pour améliorer leur flux de travail de développement. Parmi ces outils, on trouve :

PyCharm

PyCharm est un environnement de développement intégré (IDE) développé par JetBrains, offrant une gamme complète de fonctionnalités pour le développement Python, y compris un débogueur intégré, la gestion des packages, la refonte de code et bien plus encore.

Il existe 2 versions de PyCharm: la version Pro (payante) et la version Community (gratuite), donc **fais attention à quelle version tu installes**.

Jupyter Notebook

Jupyter Notebook est une application Web open source qui vous permet de créer et de partager des documents interactifs contenant du code Python, des visualisations et des explications textuelles. Il est largement utilisé pour l'analyse de données, l'apprentissage machine et la recherche scientifique.

Comment installer Python sur son ordinateur

Installer Python sur votre ordinateur est un processus simple et direct. Voici les étapes générales pour installer Python :

1. Rendez-vous sur le site officiel de Python à l'adresse https://www.python.org/downloads/ et téléchargez la dernière version de Python pour votre système d'exploitation (Windows, macOS, ou Linux).
2. Lancez le programme d'installation téléchargé et suivez les instructions à l'écran pour installer Python sur votre ordinateur. Assurez-vous de cocher l'option "Ajouter Python à PATH" lors de l'installation pour pouvoir exécuter Python depuis n'importe quel répertoire de votre système.
3. Une fois l'installation terminée, vous pouvez vérifier si Python a été correctement installé en ouvrant une fenêtre de terminal (ou d'invite de commande sur Windows) et en tapant la commande suivante :

```
python --version
```

Cette commande affichera la version de Python installée sur votre système. Si vous voyez le numéro de version de Python, cela signifie que l'installation s'est déroulée avec succès.

En suivant ces étapes simples, vous pouvez installer Python sur votre ordinateur et commencer à développer des applications en Python en un rien de temps.

Conclusion

Que vous soyez débutant ou développeur expérimenté, avoir les bons outils est essentiel pour développer en Python. Visual Studio Code, Thonny Python et www.online-python.com sont quelques-uns des outils les plus populaires et les plus utilisés dans la communauté Python. En choisissant l'outil qui convient le mieux à vos besoins et à votre style de développement, vous pouvez améliorer votre productivité et votre expérience de développement Python.

Thonny Python

> *Lorsqu'il s'agit d'apprendre à programmer en Python, choisir le bon environnement de développement peut faire toute la différence. Thonny IDE se présente comme un outil puissant et convivial, spécialement conçu pour les débutants en Python. Dans cet article, nous explorerons les avantages de Thonny, son interface utilisateur intuitive, et comment l'utiliser efficacement pour démarrer votre voyage dans le monde de la programmation Python.*

Les Avantages de Thonny IDE

Convivialité

Thonny se distingue par sa **simplicité d'utilisation**. Contrairement à certains IDE plus complexes, Thonny est conçu pour être **intuitif**, ce qui le rend **idéal pour les débutants** en programmation.

Interface Unifiée

L'interface utilisateur de Thonny est soigneusement organisée, ce qui facilite la navigation et l'accès aux différentes fonctionnalités de l'IDE. Les outils essentiels tels que l'éditeur de code, la console interactive et le débogueur sont tous intégrés de manière transparente dans **une seule fenêtre**.

Débogage Pas à Pas

Thonny offre un **débogueur intégré** qui permet aux utilisateurs d'exécuter leur code pas à pas, d'observer les valeurs des variables en cours d'exécution et de détecter les erreurs plus facilement.

Gestionnaire de Packages Intégré

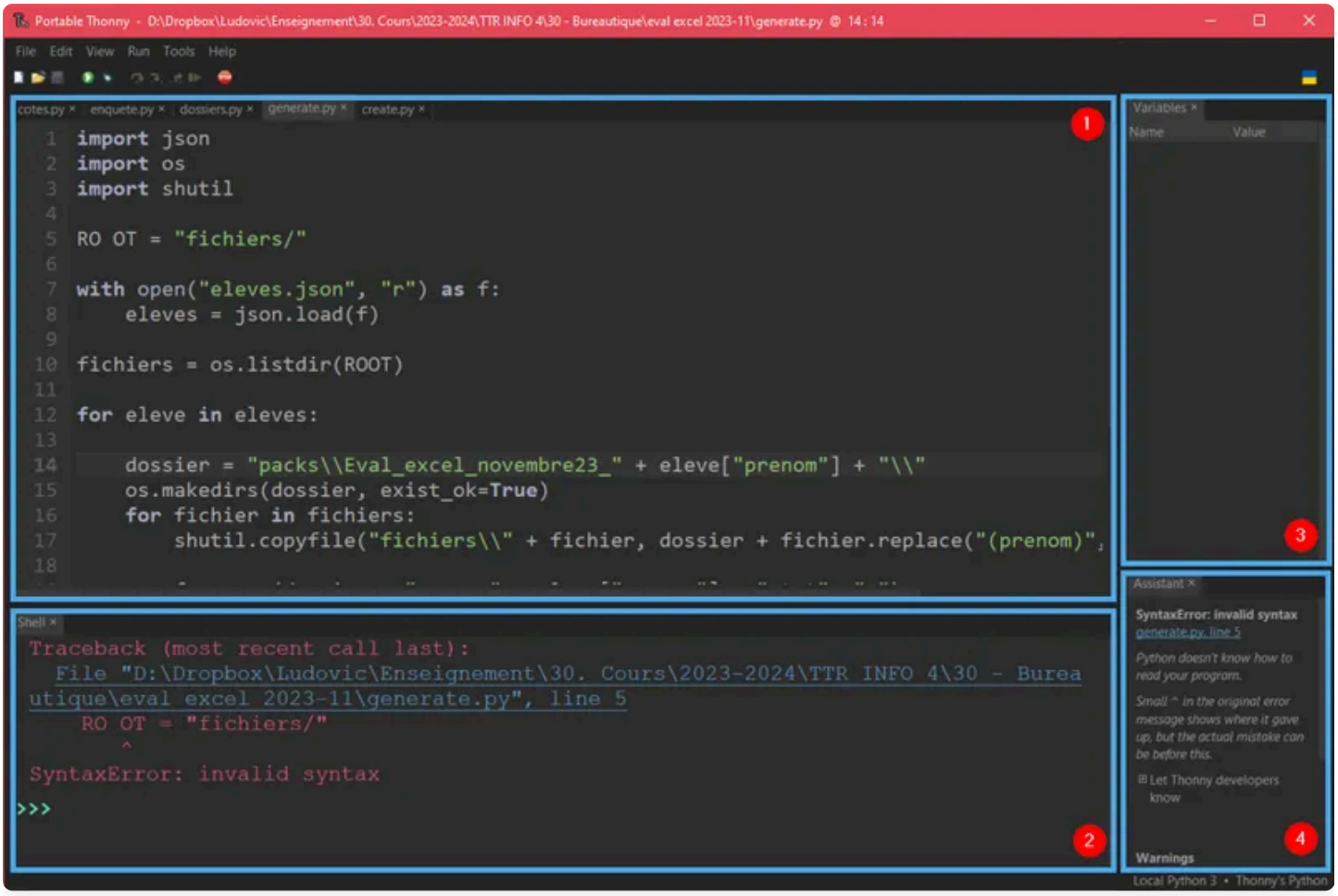
Thonny est livré avec un gestionnaire de packages/modules intégré qui facilite l'installation de modules Python supplémentaires directement depuis l'IDE, ce qui vous permet d'étendre les fonctionnalités de Python selon vos besoins.

Documentation Intégrée

Une documentation complète est disponible directement dans Thonny, ce qui vous permet d'accéder rapidement aux informations sur les fonctions et les modules Python sans avoir à quitter l'IDE.

Interface de Thonny

L'interface utilisateur de Thonny est conçue pour être simple et intuitive, ce qui permet aux utilisateurs de se concentrer sur l'essentiel : écrire du code. Voici un aperçu des principaux composants de l'interface de Thonny :



1. Éditeur de Code

C'est là où vous écrivez et modifiez votre code Python. L'éditeur de code de Thonny offre des fonctionnalités telles que la **coloration syntaxique**, l'**indentation automatique** et la suggestion de code pour améliorer votre flux de travail de programmation.

💡 La **coloration syntaxique** est une fonctionnalité qui consiste à **attribuer des couleurs différentes** aux **différents éléments d'un langage de programmation**. Cette technique vise à rendre le code source **plus lisible et compréhensible** en mettant en évidence les différents éléments syntaxiques tels que les **mots-clés**, les **variables**, les chaînes de caractères, les commentaires, etc. La coloration syntaxique **aide les programmeurs à identifier rapidement la structure et la logique du code**, ce qui facilite la détection des erreurs et améliore la productivité lors de l'écriture et de la maintenance du code.

2. Console Interactive

La console interactive vous permet d'exécuter du code Python en temps réel et d'afficher les résultats instantanément. Cela facilite le processus de test et de débogage de petits morceaux de code.

3. Explorateur de Variables

L'explorateur de variables affiche les valeurs des variables en cours d'exécution **pendant le débogage**, ce qui vous permet de surveiller l'état de votre programme à chaque étape.

4. Assistant

Thonny IDE poss§de un assistant intégré, conçu pour **aider les débutants** à programmer en Python. L'assistant propose des suggestions contextuelles, des corrections automatiques et des explications pour guider les utilisateurs tout au long de leur processus d'apprentissage. Par exemple, lorsque vous commencez à taper du code Python, l'assistant peut suggérer des complétions de code, vous permettant ainsi d'explorer les différentes options disponibles et d'apprendre de nouvelles fonctionnalités du langage. De plus, l'assistant fournit des **informations utiles sur les erreurs de syntaxe**, ce qui peut être particulièrement bénéfique pour les étudiants en programmation qui cherchent à comprendre et à corriger leurs erreurs.

5. Fenêtre de Gestion des Packages

Cette fenêtre vous permet d'installer, de mettre à jour et de supprimer des packages Python supplémentaires directement depuis l'IDE.

🔔 Pour plus d'informations sur l'installation de packages, rendez-vous sur la page "installer des modules"

Comment Utiliser Thonny pour Programmer en Python

Maintenant que nous avons exploré les avantages et l'interface de Thonny, voyons comment l'utiliser pour programmer en Python :

1. Installation de Thonny

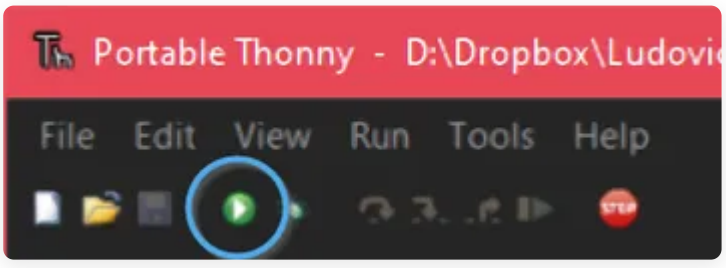
Téléchargez et installez Thonny à partir du site officiel de Thonny. Une fois l'installation terminée, lancez Thonny depuis votre menu d'applications.

2. Écrire du Code

Utilisez l'éditeur de code de Thonny pour écrire votre code Python. Vous pouvez commencer par des exemples simples et progresser à votre propre rythme.

3. Exécution du Code

Après avoir écrit votre code, vous pouvez l'exécuter en appuyant sur le bouton "Exécuter" ou en utilisant le raccourci clavier correspondant (`F5`). Les résultats s'afficheront dans la console interactive.



4. Débogage

Si vous rencontrez des erreurs, utilisez le débogueur intégré pour parcourir votre code pas à pas et identifier les problèmes éventuels.

5. Installation de Packages Additionnels

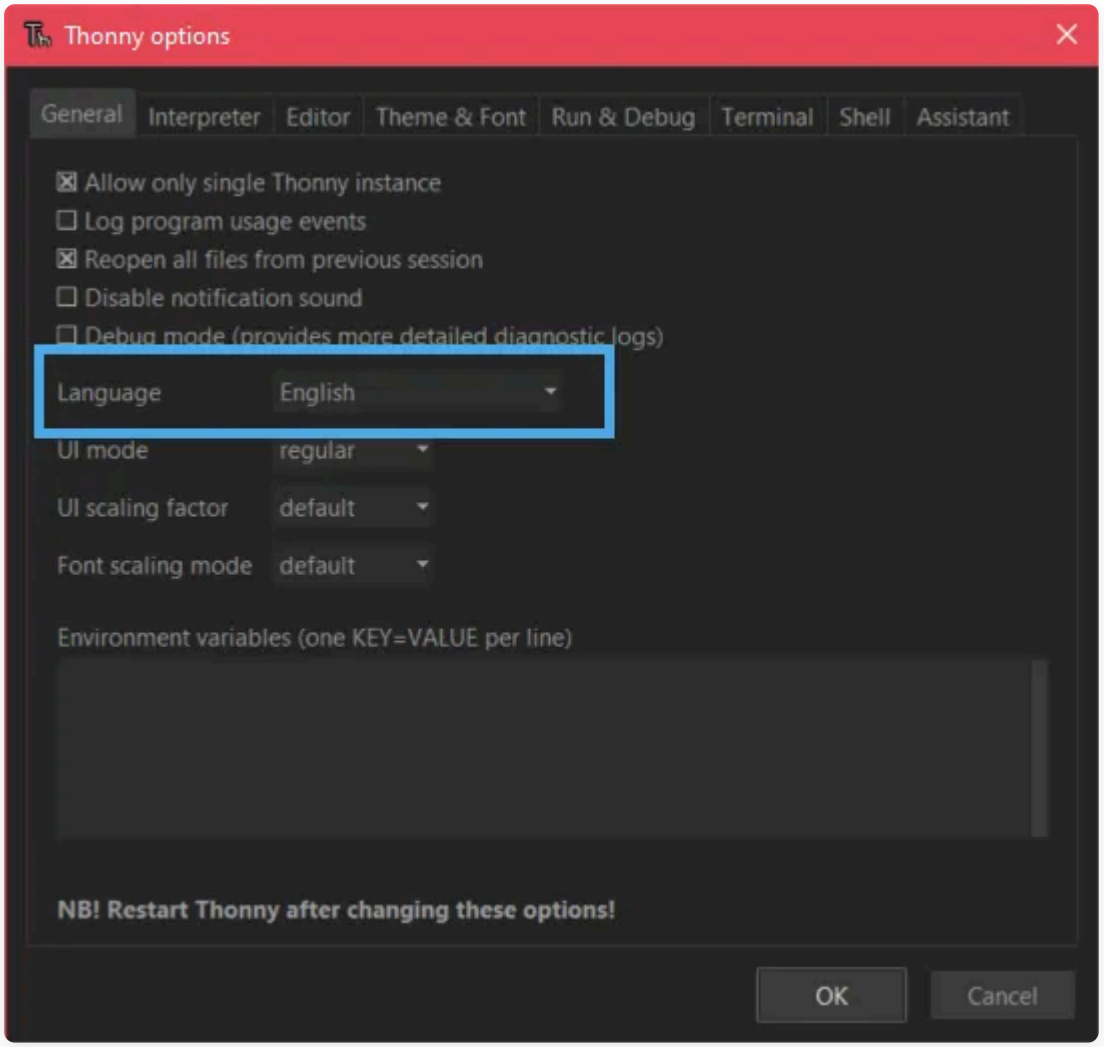
Si vous avez besoin de fonctionnalités supplémentaires, utilisez le gestionnaire de packages intégré pour installer les modules Python nécessaires à votre projet.

Configurer Thonny IDE

il est possible de configurer Thonny IDE pour qu'il corresponde à votre utilisation. Pour se faire, il faut passer par le menu "Outils (GB ***Tools***)" / "Préférences". Voici les options principales.

Changer la langue d'affichage

Pour changer de langue, rendez-vous dans l'onglet "Général" et choisissez votre langue dans la liste déroulante.



Vous devrez redémarrer Thonny pour que ce changement soit pris en compte.

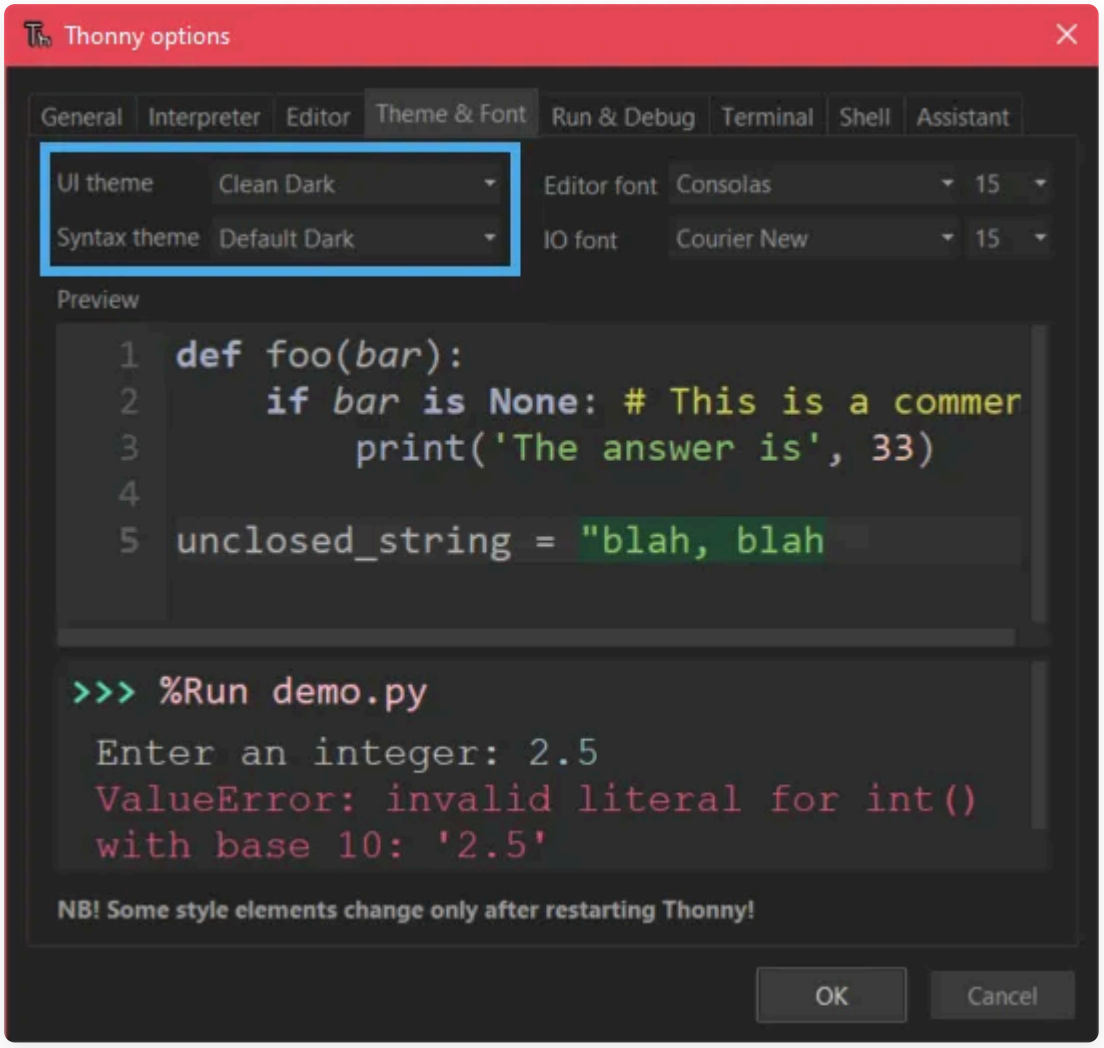
Change le thème d'affichage

Thonny possède un **mode sombre**.

Le mode sombre, également connu sous le nom de "**dark mode**", offre un fond d'écran foncé avec un texte clair, ce qui **réduit la luminosité de l'écran** et **crée un contraste visuel agréable**. Les développeurs apprécient souvent le mode sombre pour son **aspect esthétique**, mais aussi pour ses avantages ergonomiques, notamment la **réduction**

de la fatigue oculaire lors de longues sessions de programmation, surtout dans des environnements peu éclairés.

Pour configurer le mode sombre, rendez-vous dans l'onglet "Theme & Font", et choisissez un thème pour l'interface ET pour la syntaxe (le code écrit en python):



Bien sûr, vous pouvez choisir le thème que vous convient 😉

Conclusion

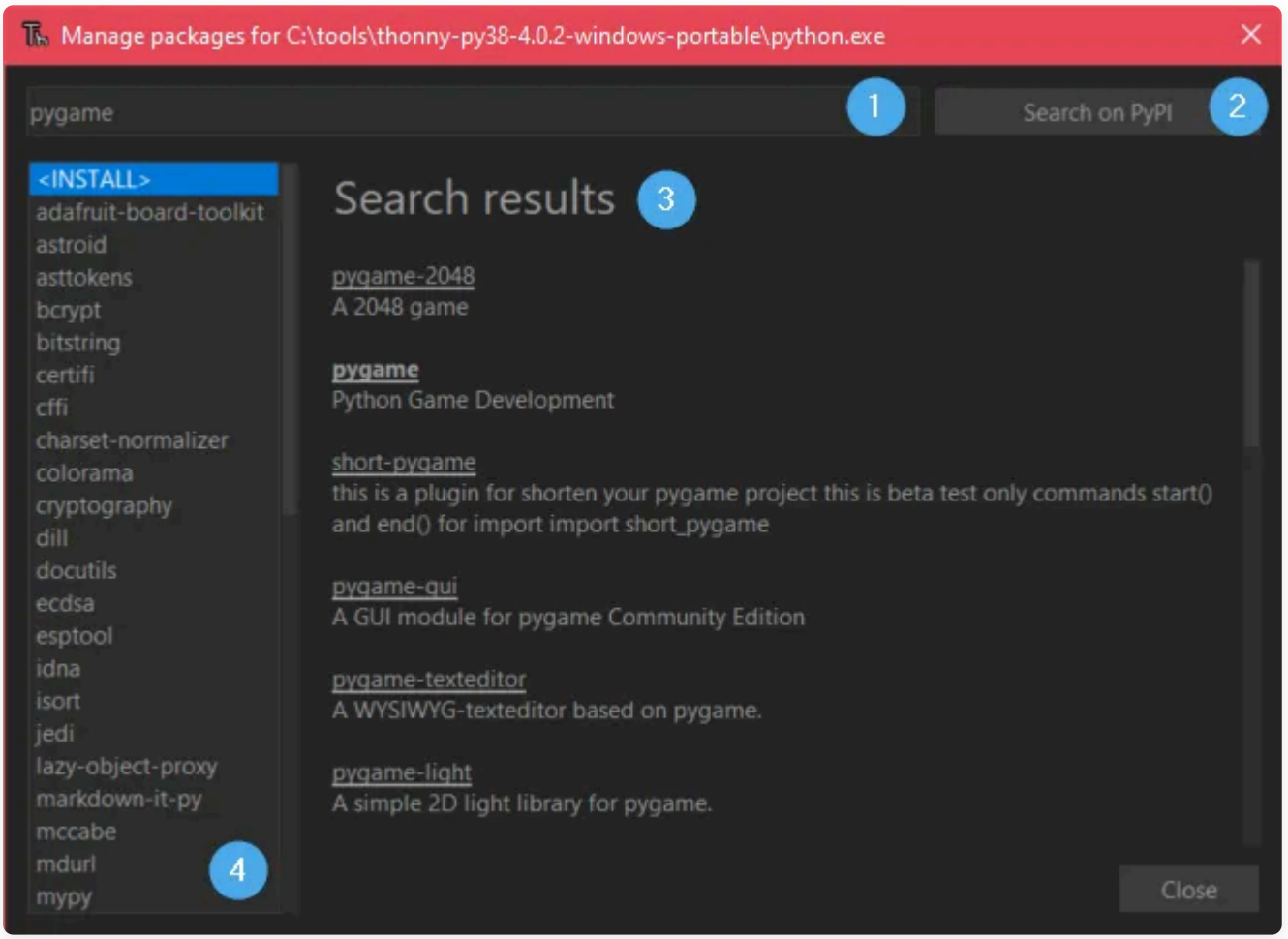
Thonny IDE est un outil exceptionnel pour les débutants en Python, offrant une expérience de développement conviviale, une interface intuitive et des fonctionnalités puissantes pour écrire, exécuter et déboguer du code Python. Que vous soyez novice en programmation ou un développeur expérimenté, Thonny peut vous accompagner tout au long de votre parcours d'apprentissage et de développement en Python. N'hésitez pas à télécharger Thonny et à commencer à explorer le monde passionnant de la programmation Python dès aujourd'hui !

Thonny Python - Installer des modules

> *L'une des fonctionnalités essentielles de Thonny est la capacité d'installer des modules Python supplémentaires pour étendre ses fonctionnalités. Dans ce guide, nous allons vous montrer étape par étape comment installer des modules Python dans Thonny, en incluant des captures d'écran pour une meilleure compréhension.*

Accéder à la gestion des packages

Pour accéder à la gestion des packages, cliquez sur le menu `Outils` (GB ***Tools***), puis sélectionnez `Gérer les packages ...` (GB ***Manage packages***). Cela ouvrira une nouvelle fenêtre où vous pourrez rechercher, installer et supprimer des packages Python.

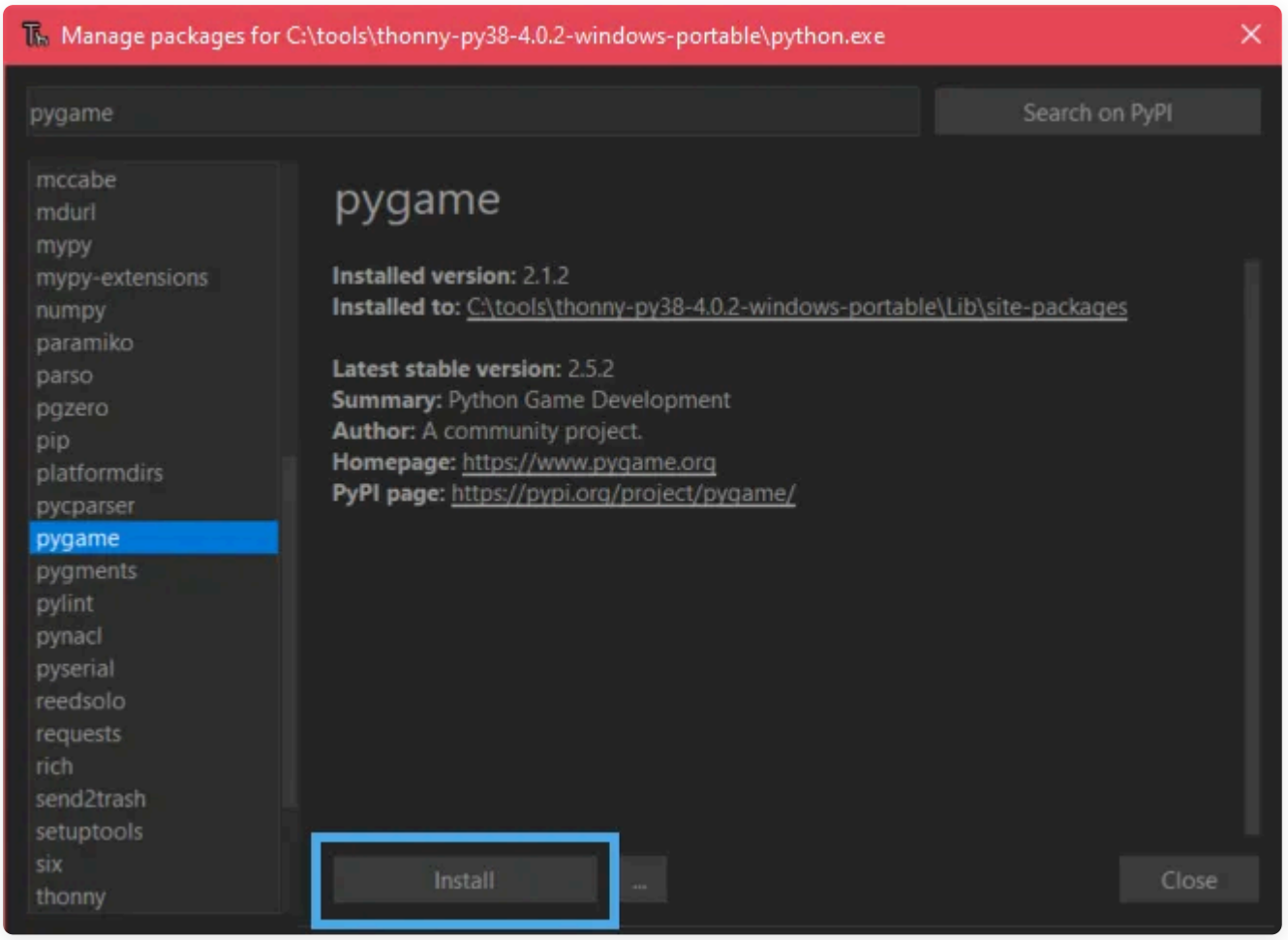


Étape 3: Rechercher et installer un module Python

Dans la fenêtre de gestion des packages, vous verrez une barre de recherche en haut `[1]`. Utilisez cette barre de recherche pour trouver le module Python que vous souhaitez installer. Par exemple, recherchons le module `requests`, souvent utilisé pour effectuer des requêtes HTTP dans Python. Appuyer sur `[Entrée]` ou cliquez sur

`[2]` pour valider votre recherche

Par exemple, tapez "pygame" dans la barre de recherche, puis appuyez sur la touche `Entrée` pour lancer la recherche. Une fois que le module `pygame` apparaît dans la liste des résultats, cliquez sur le nom du module `[3]` pour afficher la fenêtre d'installation.



Cliquez sur le bouton `Installer` pour commencer le processus d'installation.

Étape 4: Vérifier l'installation du module

Une fois l'installation terminée, vous verrez un message indiquant que le package a été installé avec succès. Vous pouvez maintenant fermer la fenêtre de gestion des packages.

Pour vérifier que le module a été correctement installé, vous pouvez l'importer dans votre script Python. Par exemple, créez un nouveau fichier Python dans Thonny et ajoutez la ligne suivante en haut du fichier :

```
import pygame
```

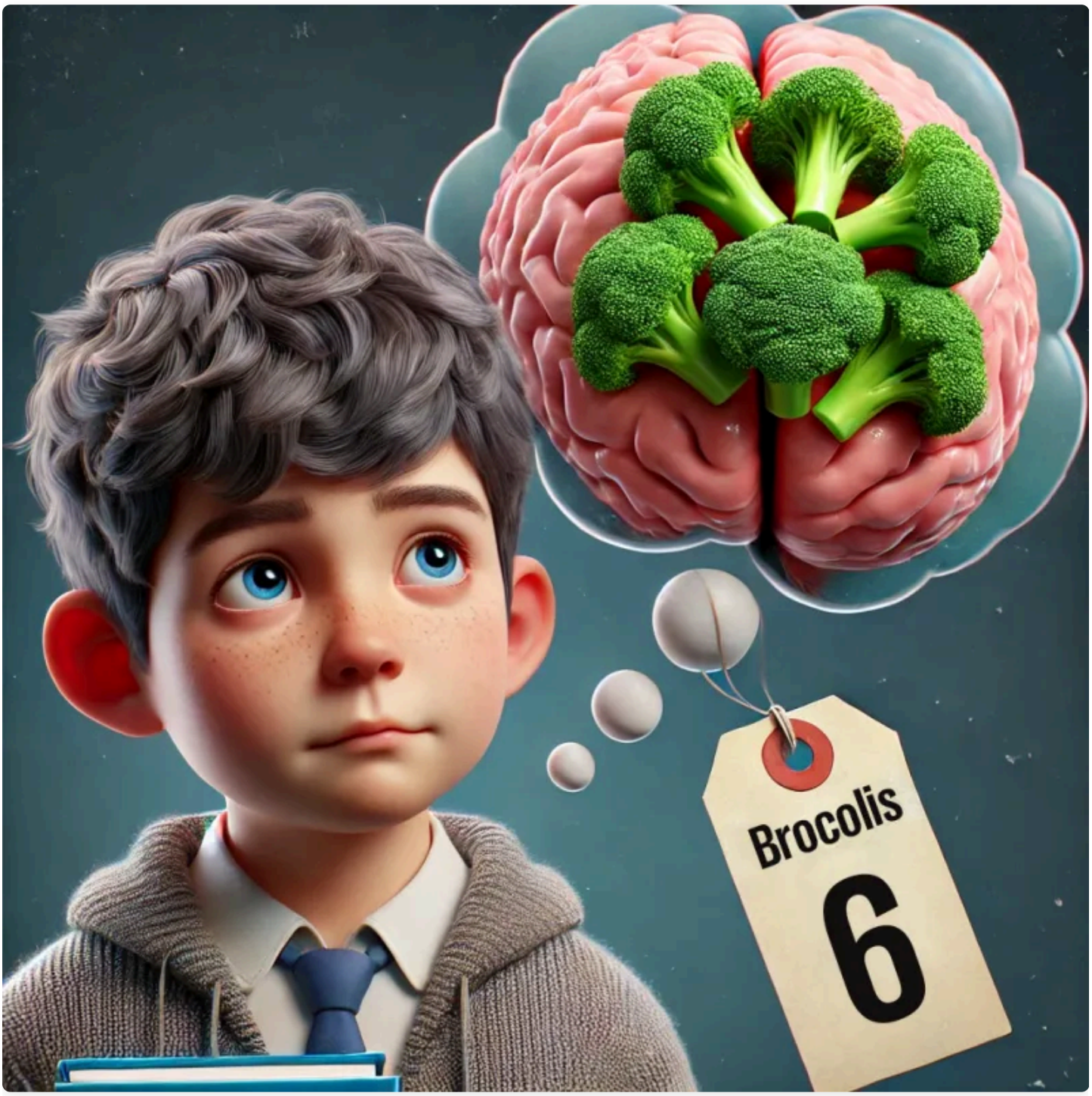
Si vous ne voyez pas d'erreurs, cela signifie que le module a été installé avec succès et que vous pouvez commencer à l'utiliser dans vos projets Python.

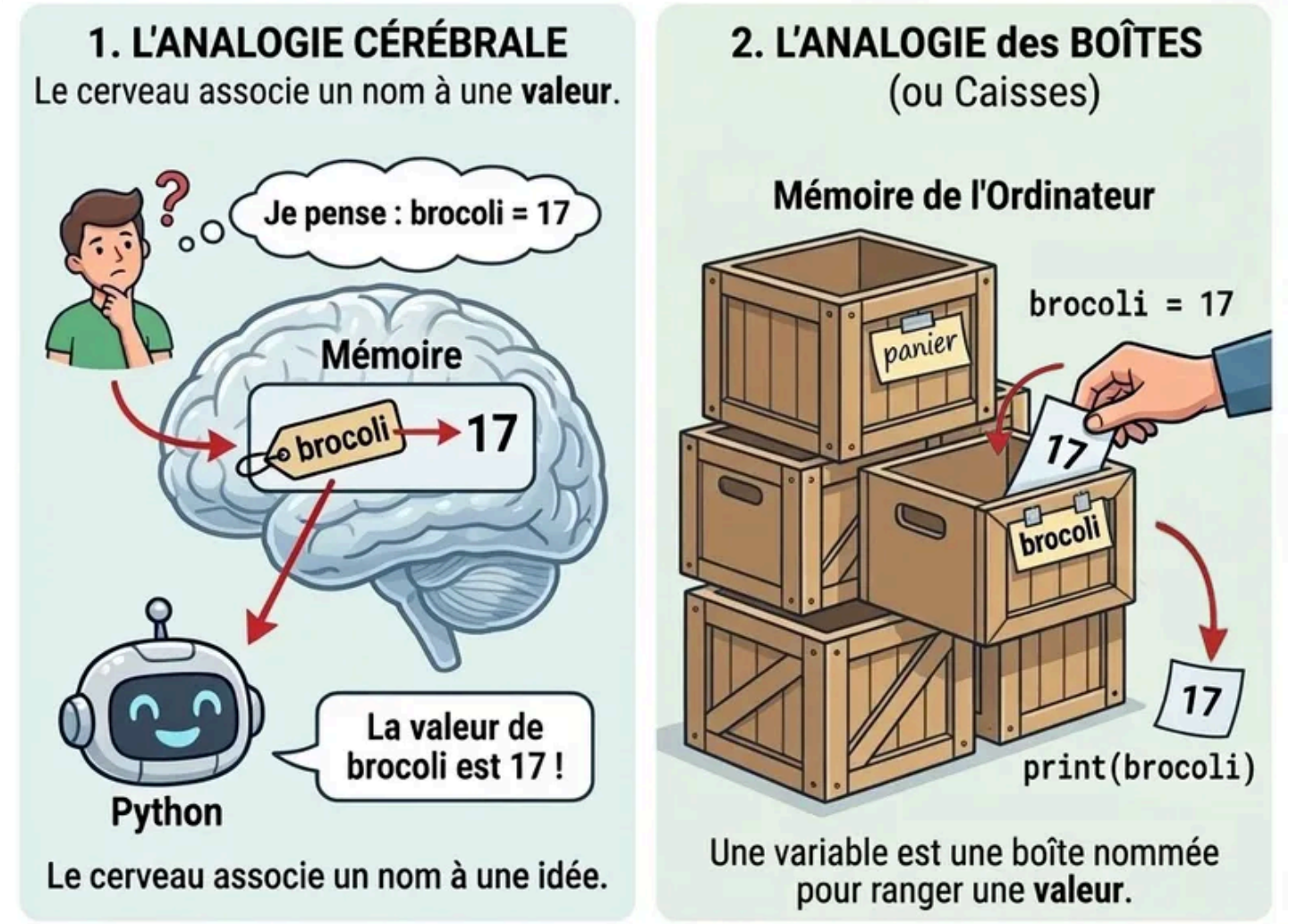
Conclusion

Dans ce guide, nous avons exploré comment installer des modules Python dans Thonny Python. En suivant ces étapes simples, vous pourrez étendre les fonctionnalités de Thonny et développer des projets Python plus avancés. N'hésitez pas à explorer d'autres modules disponibles et à expérimenter avec eux dans vos propres projets.

Les variables en Python

> *Les variables sont l'un des concepts fondamentaux en programmation. Elles permettent de **stocker des informations et de les utiliser dans un programme**.*





Une variable est **un espace de mémoire temporaire où on stocke une valeur**.

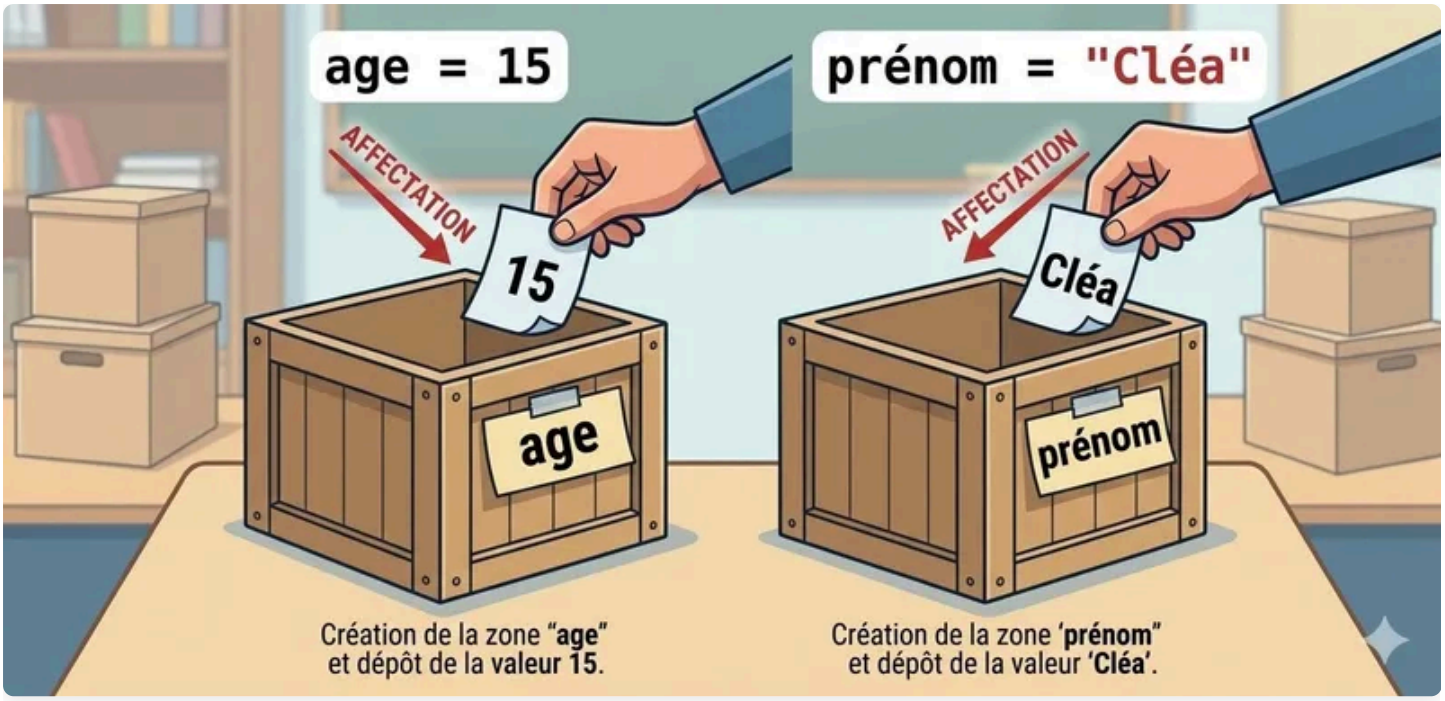
▸ Premier exemple

Prenons un exemple pour illustrer ce concept en Python :

```
nom = "Cléa"
age = 15
```

Dans cet exemple :

- `nom` est une variable qui contient le texte `Cléa`,
- `age` est une variable qui contient le nombre `15`,

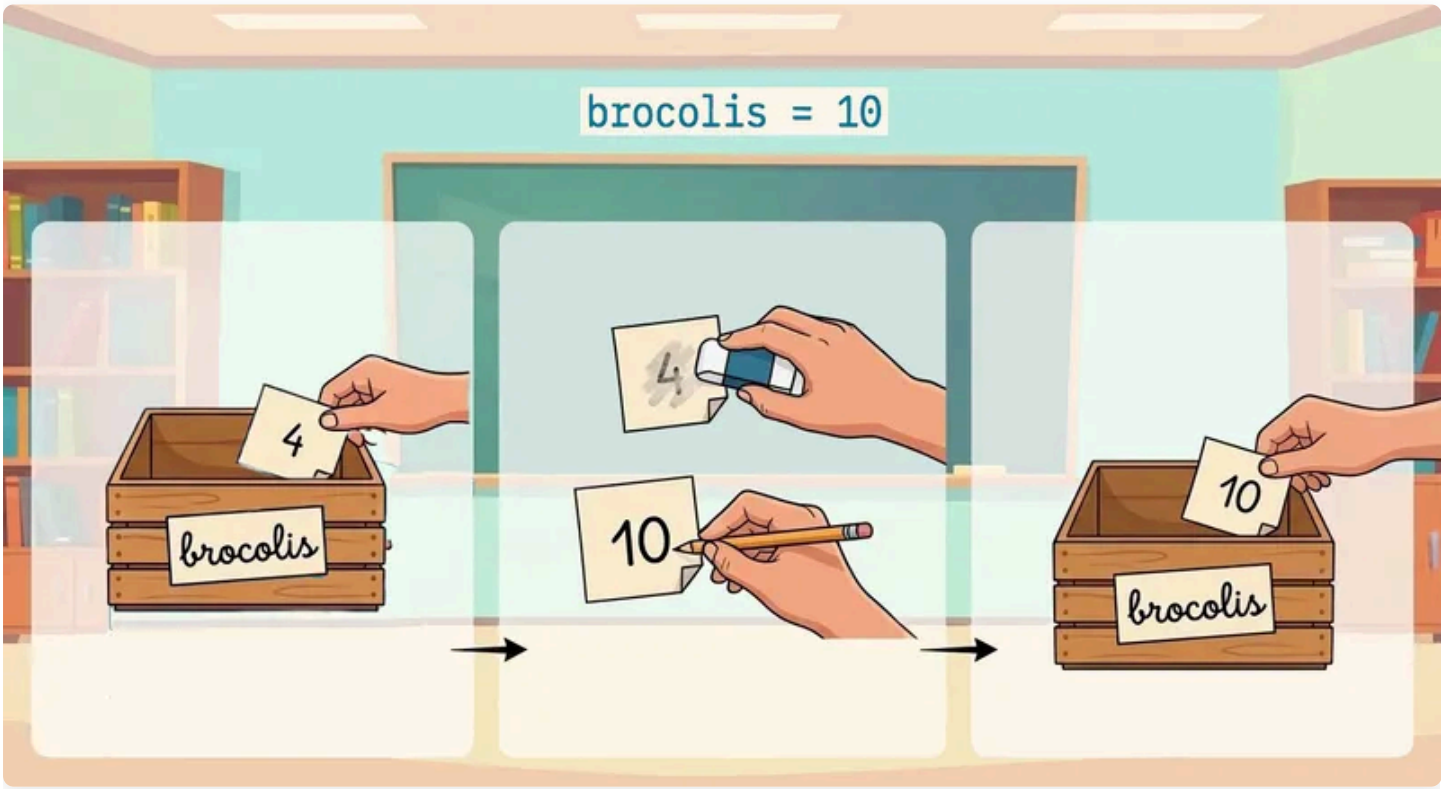


Ensuite, on pourra récupérer ces valeurs pour les afficher ou les utiliser dans un calcul par exemple.

Une variable est une zone **temporaire** dans la **mémoire** de l'ordinateur, à laquelle on donne un **nom** et dans laquelle on stocke une **valeur**.

On peut doncc changer la valeur d'une variable... autant de fois qu'on veut. Imagine une variable `brocolis` qui contient `4`. On peut lui affecter une nouvelle valeur:

```
brocolis = 10
```



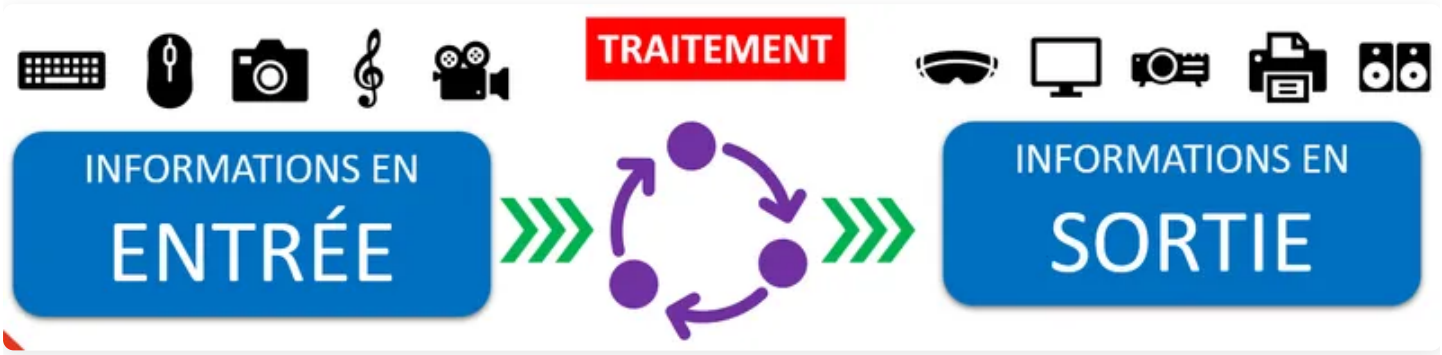
Importance des variables

Les variables servent à **stocker des informations** que le programme peut **utiliser** ou **modifier** au cours de son exécution. Elles agissent comme des "**boîtes**" dans lesquelles on peut ranger une valeur, par exemple un nombre, une chaîne de caractères, ou encore le résultat d'un calcul. **On utilise les variables lorsque l'on souhaite conserver une donnée pour s'en servir plus tard**, par exemple pour effectuer des **opérations**, **afficher un résultat**, ou gérer des interactions avec l'utilisateur.

Les variables permettent également de **rendre le code plus clair** et réutilisable, car elles **remplacent des valeurs fixes par des noms compréhensibles**.

> *En résumé, **on utilise des variables pour stocker les valeur qui seront affichées, utilisées ou modifiées plus tard dans le programme**, souvent plusieurs fois.*

Souviens-toi de ce schéma:



Un programme reçoit des données/informations en entrée, leur applique un traitement et fournit des données/informations en sortie. **Dans un programme informatique, il y a donc des données et des traitements**. Retiens ceci:

Les données sont stockées dans des variables.

Déclarer ses Variables

En Python, les variables sont **déclarées** en leur **affectant** une valeur.

En Python, **déclarer une variable** signifie **créer une variable** en lui donnant un **nom** et une **valeur**.

Par exemple, pour déclarer une variable nommée "`age`" avec une valeur de 25, nous pouvons utiliser le code suivant :

```
age = 25
```

Cela veut dire qu'on va **stocker** la valeur `25` dans la variable `nom`. Cela se fait **de droite à gauche**.

💡 Il faut **toujours** déclarer une variable avant de l'utiliser. Si tu utilises une variable avant de l'avoir déclarée, Python va afficher l'erreur **NameError**:

```
>>> %Run -c $EDITOR_CONTENT
  Traceback (most recent call last):
    File "<string>", line 1, in <module>
  NameError: name 'age' is not defined

>>>
```

`"name 'age' is not defined"` signifie que la variable `age` n'a pas été déclarée au moment où elle est utilisée.

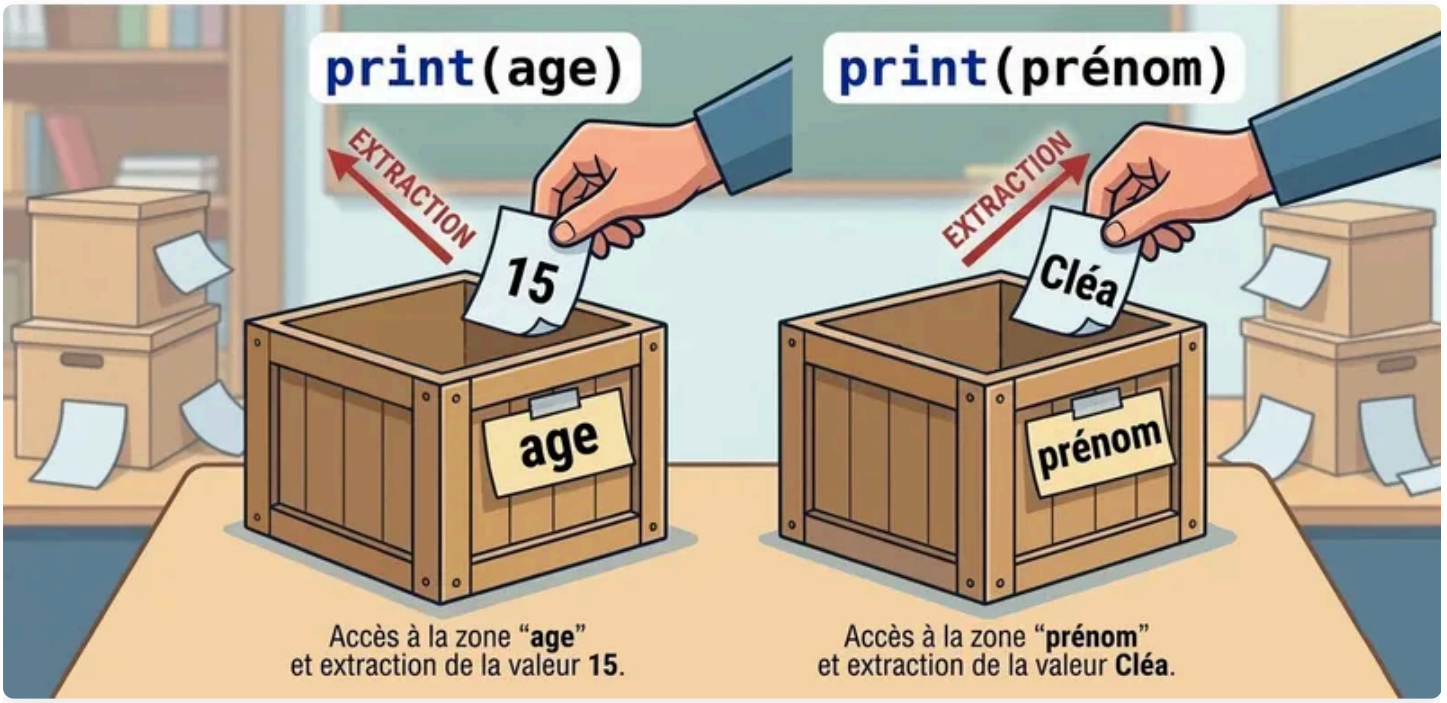
Cela peut peut être dû au fait que:

- tu essaies d'utiliser une variable avant de l'avoir déclarée
- il y a une faute dans le nom (ex: `prenon` au lieu de `prenom` ou une différence minuscules/majuscules)
- tu as oublié les guillemets autours d'un texte (par exemple: `print(Sara)`)

Utiliser les Variables

Une fois la variable déclarée, on peut l'afficher ou l'utiliser dans des opérations et autres instructions.

```
age = 15
prenom = "Cléa"
print (age) # Affichera 15
print (prenom) # Affichera Cléa
```

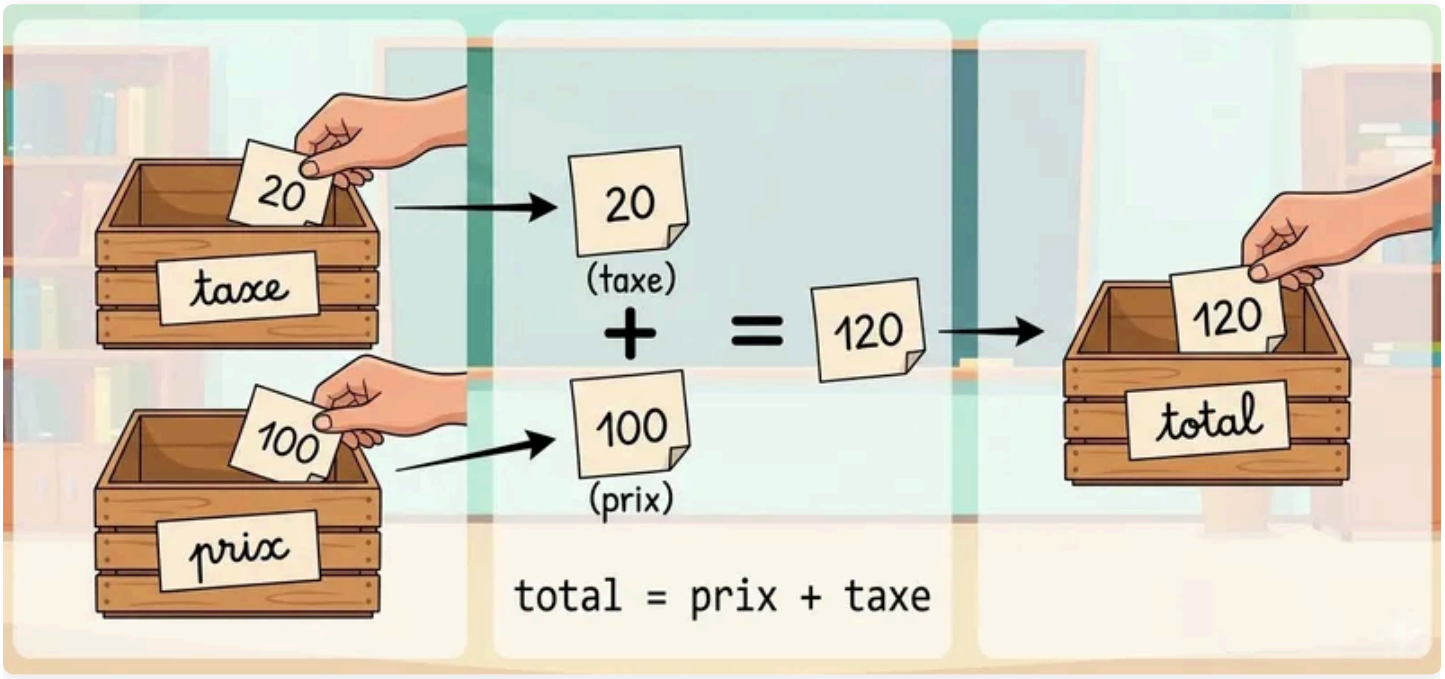


Explications

- `print` permet d'afficher quelque chose à l'écran.
- ligne 3: On affiche le **contenu** de la variable `age` : 15
- ligne 4: On affiche le **contenu** de la variable `prenom` : Cléa

```
prix = 100
taxe = 20
total = prix + taxe # total vaut maintenant 120

print (total) # Affiche 120
```



Explications

- ligne 3: On calcul le total en additionnant les **valeurs** contenues dans les variables `prix` et `taxe` et on stocke le résultat dans la variable ´total`.
- ligne 5: On affiche le **contenu** de la variable `total`, c'est à dire `120`

Ce code affiche:

```
120
```

Il est aussi possible, en une instruction, de mettre à jour la valeur d'une variable:

```
brocolis = 18

brocolis = brocolis + 4 # maintenant, brocolis vaut 22 (18+4)
```

Ligne 4: Python exécute d'abord les instructions à droite du `=`. Il récupère la valeur de `brocolis` (=18), et y ajoute 4 (=22) . Python stocke ensuite le résultat (22) dans la variable `brocolis`.

Conclusion

Les variables sont une partie essentielle de la programmation. Elles permettent de **stocker et manipuler des données**. En maîtrisant les variables, tu pourras écrire des programmes plus dynamiques et puissants.

Afficher ses variables avec print

> *La fonction `print()` est une des bases de la programmation en Python. Elle permet d'afficher du texte, des nombres ou des résultats sur l'écran, ce qui est essentiel pour communiquer avec l'utilisateur ou vérifier le fonctionnement d'un programme.*

Objectif

Apprendre à utiliser `print()` pour afficher différentes informations de manière claire et organisée sur l'écran.

Introduction à `print()`

La fonction `print()` permet d'afficher du contenu dans le terminal. Sa syntaxe de base est la suivante :

```
print(valeur)
```

(Attention aux paranthèses)

Exemple :

```
print("Bonjour tout le monde !")
```

Résultat :

```
Bonjour tout le monde !
```

Afficher plusieurs valeurs

On peut afficher plusieurs valeurs en les séparant par des virgules. Par défaut, `print()` ajoute un espace entre chaque valeur.

Exemple :

```
print("Bonjour", "tout", "le", "monde", "!")
```

Résultat :

```
Bonjour tout le monde !
```

Affichage de variables

On peut également utiliser `print()` pour afficher le contenu de variables.

Exemples :

```
annee = 2025
print(annee) # affiche: 2025
print("Nous sommes en", annee) # affiche: Nous sommes en 2025
```

```
nom = "Alice"
age = 25
print("Nom :", nom, "Âge :", age)
```

Résultat :

```
Nom : Alice Âge : 25
```

Changer le séparateur entre les valeurs

Par défaut, les valeurs affichées avec `print()` sont **séparées par un espace** (`" "`). On peut modifier cela grâce au paramètre `sep` .

Exemple :

```
print("Python", "est", "génial", sep="-")
```

Résultat :

```
Python-est-génial
```

Modifier la fin de ligne

Par défaut, `print()` termine chaque affichage par un saut de ligne (`\n`). On peut changer cela grâce au paramètre `end` .

Exemple :

```
print("Bonjour", end=" ")
print("tout le monde !")
```

Résultat :

```
Bonjour tout le monde !
```

À retenir

- La fonction `print()` est utilisée pour afficher du contenu.
- Elle peut afficher des chaînes, des nombres, des variables, ou des calculs.
- Les paramètres `sep` et `end` permettent de personnaliser l'affichage.

Avec une bonne maîtrise de `print()`, tu peux rendre tes programmes interactifs et clairs pour l'utilisateur ou pour toi-même !

Manipuler les variables en Python

> *Une variable permet de stocker une valeur temporairement. Et, surtout, elle permet à votre programme de manipuler cette valeur.*

Les variables peuvent être manipulées de différentes manières en Python. Par exemple, nous pouvons leur **attribuer de nouvelles valeurs**, les **utiliser** dans des opérations mathématiques, les **concaténer** pour former des chaînes de caractères, etc.

Voici des exemples de manipulation de variables :

```
# Attribuer une nouvelle valeur à une variable
age = 26

# Utiliser des variables dans des opérations mathématiques
a = 10
b = 20
somme = a + b
difference = b - a
produit = a * b
quotient = b / a

# Concaténer des variables pour former une chaîne de caractères
prenom = "Jean"
nom = "Dupont"
nom_complet = prenom + " " + nom
```

Principes de base des manipulations

Une variable garde sa valeur jusqu'à la fin du programme ou jusqu'à ce qu'on lui en affecte une nouvelle.

Le simple fait d'utiliser une variable **ne change pas** sa valeur.

Valeurs et variables: à toi de jouer (2) 💪

> *Dans ces exercices, tu vas apprendre à **modéliser une situation réelle** avec des variables, à **effectuer un calcul simple**, puis à **afficher le résultat**. Avant d'exécuter le programme, tu dois **prédire ce que Python va afficher**, comme si tu étais l'ordinateur.*

> *Un programme commence toujours par **ce que je sais** (les variables), puis **ce que je calcule**, et enfin **ce que j'affiche**.*

L'objectif est clair pour l'élève : 👉 **réfléchir avant de coder** 👉 **identifier les variables nécessaires** 👉 **écrire le calcul** 👉 **afficher le résultat**

Dans chaque exercice :

- identifie **les informations à connaître**
- crée **une variable par information**
- réalise **le calcul demandé**
- affiche **le résultat final**

Voici 5 énoncés d'exercices Python à contexte scientifique, sans boucles ni fonctions :

🧪 Exercice 1 — Loi d'Ohm

Un circuit électrique contient une résistance de **470 Ω** soumise à une tension de **9 V**.

1. Calcule l'intensité du courant (en ampères) à l'aide de la loi d'Ohm : `U = R × I`
2. Affiche le résultat avec la phrase : `"L'intensité du courant est de X ampères."`
3. Calcule ensuite la puissance dissipée par la résistance : `P = U × I` et affiche-la.

🌍 Exercice 2 — Chute libre

Un objet est lâché sans vitesse initiale depuis une hauteur de **80 mètres**.

On donne : `g = 9.81 m/s²`

La formule de la durée de chute est : `t = sqrt(2h / g)`

1. Importe le module `math` et utilise `math.sqrt()` pour calculer la durée de la chute.
2. Calcule la vitesse d'impact : `v = g × t`
3. Affiche les deux résultats en arrondissant à 2 décimales avec `round()`.

🧬 Exercice 3 — Concentration d'une solution

En chimie, la concentration molaire est donnée par : `C = n / V`

où `n` est la quantité de matière en moles et `V` le volume en litres.

On dissout **0.5 mol** de sel dans **0.25 L** d'eau.

1. Calcule la concentration molaire `C`.
2. On dilue ensuite cette solution en ajoutant **0.75 L** d'eau supplémentaires. Calcule la nouvelle concentration.
3. Affiche les deux concentrations avec leurs unités `mol/L`.

🌡️ Exercice 4 — Conversion de températures

Les trois échelles de température sont liées par les formules :

- Celsius → Fahrenheit : `F = C × 9/5 + 32`
- Celsius → Kelvin : `K = C + 273.15`

1. Stocke une température de **-40 °C** dans une variable.
2. Convertis-la en Fahrenheit et en Kelvin.
3. Affiche les trois valeurs. Que remarques-tu pour -40° ?

🔭 Exercice 5 — Distance en années-lumière

La lumière parcourt **299 792 458 m/s**. Une année-lumière est la distance parcourue par la lumière en un an.

1. Calcule le nombre de secondes dans une année (365 jours).
2. Calcule la valeur d'une année-lumière en mètres.
3. L'étoile Proxima Centauri est à **4.24 années-lumière**. Calcule sa distance en mètres et affiche-la en **notation scientifique** avec `f"{valeur:.2e}"`.

📦 Les différentes façons d'importer

1. Importer le module entier

```python
import math

resultat = math.sqrt(16)   # on préfixe avec "math."
print(resultat)            # 4.0
```

2. Importer une fonction spécifique

```python
from math import sqrt

resultat = sqrt(16)        # plus besoin du préfixe "math."
print(resultat)            # 4.0
```

3. Importer plusieurs éléments d'un coup

```python
from math import sqrt, pi, cos

print(pi)                  # 3.141592653589793
print(sqrt(25))            # 5.0
print(cos(0))              # 1.0
```

4. Importer tout un module avec un alias (raccourci)

```python
import math as m

print(m.sqrt(9))           # 3.0
print(m.pi)                # 3.141592653589793
```

🔬 Modules utiles en sciences

MODULE	CONTIENT	EXEMPLE
`math`	`sqrt`, `pi`, `cos`, `sin`, `log`, `exp`...	`math.pi` → 3.14...
`random`	nombres aléatoires	`random.randint(1, 6)`
`statistics`	`mean`, `median`, `stdev`...	`statistics.mean([1,2,3])`

✅ Pour les exercices précédents

L'**exercice 2** (chute libre) est le seul qui nécessite un import :

```python
import math

h = 80
g = 9.81

t = math.sqrt(2 * h / g)
print(t)
```

Les exercices 1, 3, 4 et 5 n'ont besoin **d'aucun import** — les opérateurs `+`, `-`, `*`, `/` suffisent !

Important

> *En programmation, on ne commence jamais par le calcul. On commence par **identifier les informations**, puis on **les stocke dans des variables**, ensuite seulement on **calcule et on affiche**.*

Bien nommer ses variables en Python

> *Les conventions de nommage des variables en Python rendent votre code lisible et compréhensible. Elles permettent de suivre un ensemble de règles et de pratiques recommandées pour donner des noms significatifs à vos variables. Dans cet article, nous allons explorer à la fois la syntaxe obligatoire et les conventions préconisées par PEP et Google.*

Syntaxe Obligatoire

En Python, il existe quelques règles de syntaxe obligatoires que vous devez respecter lors de la déclaration de vos variables :

1. **Les noms de variables doivent commencer par une lettre (a-z, A-Z) ou un trait de soulignement (`_`), aussi appelé "tiret du bas" ou "GB *underscore*").** Par exemple, `nom_variable` est valide, mais `1variable` ne l'est pas.
2. **Les noms de variables ne peuvent contenir que des lettres, des chiffres (0-9) et des traits de soulignement (_).** Les caractères spéciaux comme les espaces ou les symboles sont interdits. Par exemple, `ma_variable_1` est valide, mais `ma variable !` ne l'est pas.
3. **Les noms de variables sont sensibles à la casse.** Cela signifie que `ma_variable` et `Ma_Variable` sont considérés comme deux variables distinctes.
4. **Vous ne pouvez pas utiliser un mot réservé de Python comme nom de variable**. Par exemple, `print` , `if` ,.... sont des noms de variables incorrects.

Conventions Préconisées

Il existe également des **conventions** de codage pour le langage Python pour le nommage des variables. Les conventions présentées ici sont celles **préconisées** par la communauté de développeurs Python.

Ces conventions de nommage sont **facultatives** mais **fortement recommandées**. Si chacun est libre d'utiliser ses propres conventions, utiliser des conventions "reconnues" internationalement rendront votre code plus lisible pour les autres.

Peu importe les conventions que vous utilisez, l'important est d'**être cohérent** dans votre code, votre projet et dans votre équipe. Choisissez votre convention une fois pour toutes et respectez-la.

Voici quelques-unes des conventions de nommage généralement recommandées pour les variables :

1. **Utiliser des noms de variables en minuscules avec des traits de soulignement ("tirets du bas", GB *underscores*) pour séparer les mots.** Par exemple, `ma_variable_de_test` plutôt que `MaVariableDeTest` .

💡 Quand on écrit un nom de variable avec des tirets du bas, on appelle ça du "**snake case**".

2. **Pour les constantes, utilisez des majuscules avec des traits de soulignement.** Par exemple, `VALEUR_MAXIMALE` plutôt que `valeur_maximale` .

3. **Éviter d'utiliser des caractères uniques comme noms de variables, sauf pour des variables temporaires.** Par exemple, `i` est couramment utilisé pour les indices dans les boucles, mais il est préférable d'utiliser un nom plus descriptif pour les variables importantes.
4. **Soyez descriptif dans le choix des noms de variables.** Un nom de variable comme `compteur` est plus informatif que simplement `c`.
5. De manière générale, on évite les articles dans les noms: `aire_du_rectangle` -> `aire_rectangle`

Conclusion

En conclusion, les conventions de nommage des variables en Python sont essentielles pour maintenir un **code propre et lisible**. En suivant la syntaxe obligatoire et en adoptant les conventions préconisées par la communauté, vous faciliterez la compréhension de votre code par vous-même et par les autres développeurs. N'oubliez pas que la cohérence est essentielle, alors choisissez une convention et suivez-la tout au long de votre projet.

Pratique

Exercice : Nommer les Variables en Respectant les Conventions Python

Vous allez vous entraîner à nommer des variables en respectant les conventions de nommage en Python. Ces conventions aident à rendre votre code plus clair, lisible et maintenable.

En Python, les variables doivent être nommées en utilisant le style **snake_case**, c'est-à-dire avec des mots en minuscules séparés par des underscores `_`. Par exemple, pour "Le nombre de kilomètres parcourus", on utilise `km_parcourus`.

CONSIGNES

Pour chaque phrase ci-dessous, trouvez un nom de variable approprié en respectant les conventions Python.

1. Le nombre total d'élèves dans la classe
2. La longueur du rectangle en centimètres
3. La largeur de la salle de sport
4. Le poids de l'objet en kilogrammes
5. Le montant total des achats
6. La température de l'eau en degré Celsius
7. La vitesse maximale autorisée sur la route
8. Le nombre d'articles en stock
9. Le nom du professeur principal
10. L'âge de l'utilisateur
11. La durée du trajet en heures
12. Le niveau de batterie du téléphone
13. La date de naissance de l'utilisateur
14. Le pourcentage de réussite de l'examen
15. Le nom de la ville de résidence

16. Le nombre de pages dans le livre
17. La quantité de sucre en grammes
18. Le prix d'un billet de cinéma
19. La taille de l'écran en pouces
20. Le taux de croissance de la population
21. Le code postal de l'adresse
22. Le nombre de jours de vacances
23. La distance entre deux villes en kilomètres
24. Le numéro de la carte d'identité
25. Le niveau sonore en décibels
26. Le temps de réponse du serveur en millisecondes
27. Le prix de vente d'un produit
28. Le nombre de personnes présentes dans la salle
29. Le taux d'inflation annuel
30. La durée de vie moyenne d'une pile en heures

Exemple de réponse attendue :

- Pour "Le nombre de kilomètres parcourus", la variable est `km_parcourus`.
- Pour "La température extérieure", la variable est `temperature_exterieure`.

OBJECTIF

Entraînez-vous à bien nommer les variables de façon claire et explicite pour que d'autres développeurs puissent comprendre facilement à quoi elles servent.

N.B. : Dans cet exercice, il n'est pas nécessaire d'implémenter un programme en Python ; concentrez-vous uniquement sur le nommage des variables en respectant les conventions de style Python.

Python - Opérations mathématiques de base

> *Python est un langage de programmation polyvalent qui peut être utilisé pour effectuer diverses opérations mathématiques. Dans ce cours, nous allons explorer les opérations mathématiques de base en Python, notamment l'addition (+), la soustraction (-), la multiplication (*), la division (/), la division entière (//), le modulo (%), et l'exponentiation (**).*

Voici un article de cours sur la syntaxe de base du langage Python pour effectuer des opérations mathématiques de base.

Pour faire court: les opérations mathématiques en Python fonctionnent **comme en math**, y compris la **priorité des opérations**.

Addition (+)

L'addition est l'opération qui permet de combiner deux nombres pour obtenir leur somme. En Python, l'opérateur d'addition est le signe "+". Voici un exemple :

```
a = 5
b = 3
resultat = a + b
print(resultat)  # Affiche 8
```

Soustraction (-)

La soustraction est l'opération qui permet de trouver la différence entre deux nombres. En Python, l'opérateur de soustraction est le signe "-". Voici un exemple :

```
x = 10
y = 4
difference = x - y
print(difference)  # Affiche 6
```

Multiplication (*)

La multiplication est l'opération qui permet de trouver le produit de deux nombres. En Python, l'opérateur de multiplication est le signe "*" (astérisque). Voici un exemple :

```
p = 7
q = 2
produit = p * q
print(produit)  # Affiche 14
```

Division (/)

La division permet de trouver le quotient de deux nombres. En Python, l'opérateur de division est le signe "/" (slash). Voici un exemple :

```
dividende = 8
diviseur = 2
quotient = dividende / diviseur
print(quotient)  # Affiche 4.0
```

Division Entière (//)

La division entière permet de trouver le quotient de deux nombres en ignorant la partie décimale. En Python, l'opérateur de division entière est le signe "//". Voici un exemple :

```
numerator = 15
denominator = 4
quotient_integer = numerator // denominator
print(quotient_integer)  # Affiche 3
```

Modulo (%)

L'opérateur modulo (%) permet de trouver le reste de la division de deux nombres. Voici un exemple :

```
n = 17
m = 5
reste = n % m
print(reste)  # Affiche 2
```

Exposants (**)

L'exponentiation permet d'élever un nombre à une puissance donnée. En Python, l'opérateur d'exponentiation est le double astérisque "**". Voici un exemple :

```
base = 2
exposant = 3
resultat_expo = base ** exposant
print(resultat_expo)  # Affiche 8
```

Séparateur de décimales

En Belgique, nous utilisons la virugle (`,`) pour séparer la partie entière de la partie décimale d'un nombre réel (ce qu'on appelle un nombre à... virgule).

En Python, on utilise le point (`.`) pour séparer la partie entière de la partie décimale d'un nombre réel.

```
pi = 3.1415
nombre_or = 1.61803398875
```

Priorité des opérations

Les règles de **priorité** sont les **mêmes qu'en math**: **les `*` et `/` sont calculés avant les `+` et les `-`**.

Combien donnera le calcul suivant?

```
total = 5 + 2 * 3

print(total)
```

21? Réessaye 😀 . La réponse et 11. Etant donné l'ordre des opérations, Python calcule d'abord `2 * 3` et l'additionne ensuite à 5.

Les parenthèses permettent de changer l'ordre de priorité.

Combien donnera le calcul suivant?

```
total = (5 + 2) * 3

print(total)
```

11? Réessaye 😀 . La réponse et 21. Etant donné l'ordre des opérations, Python calcule d'abord l'intérieur des parenthèses (`5 + 2`) et multliplie le résultat par 3.

Les paranthèses à l'intérieur des parenthèses sont exécutées en premier.

L'affectation (signe " `=` ") est toujours effectuée en dernier. Autrement dit, Python va toujours effectuer les opérations à droite du `=` d'abord.

Conclusion

Ce cours vous a introduit aux opérations mathématiques de base en Python. Vous pouvez maintenant effectuer des opérations d'addition, de soustraction, de multiplication, de division, de division entière, de modulo, et d'exponentiation en utilisant Python. Ces concepts sont essentiels pour résoudre des problèmes mathématiques et scientifiques à l'aide de la programmation. N'hésitez pas à pratiquer ces opérations pour les maîtriser davantage.

Python: Les conditions

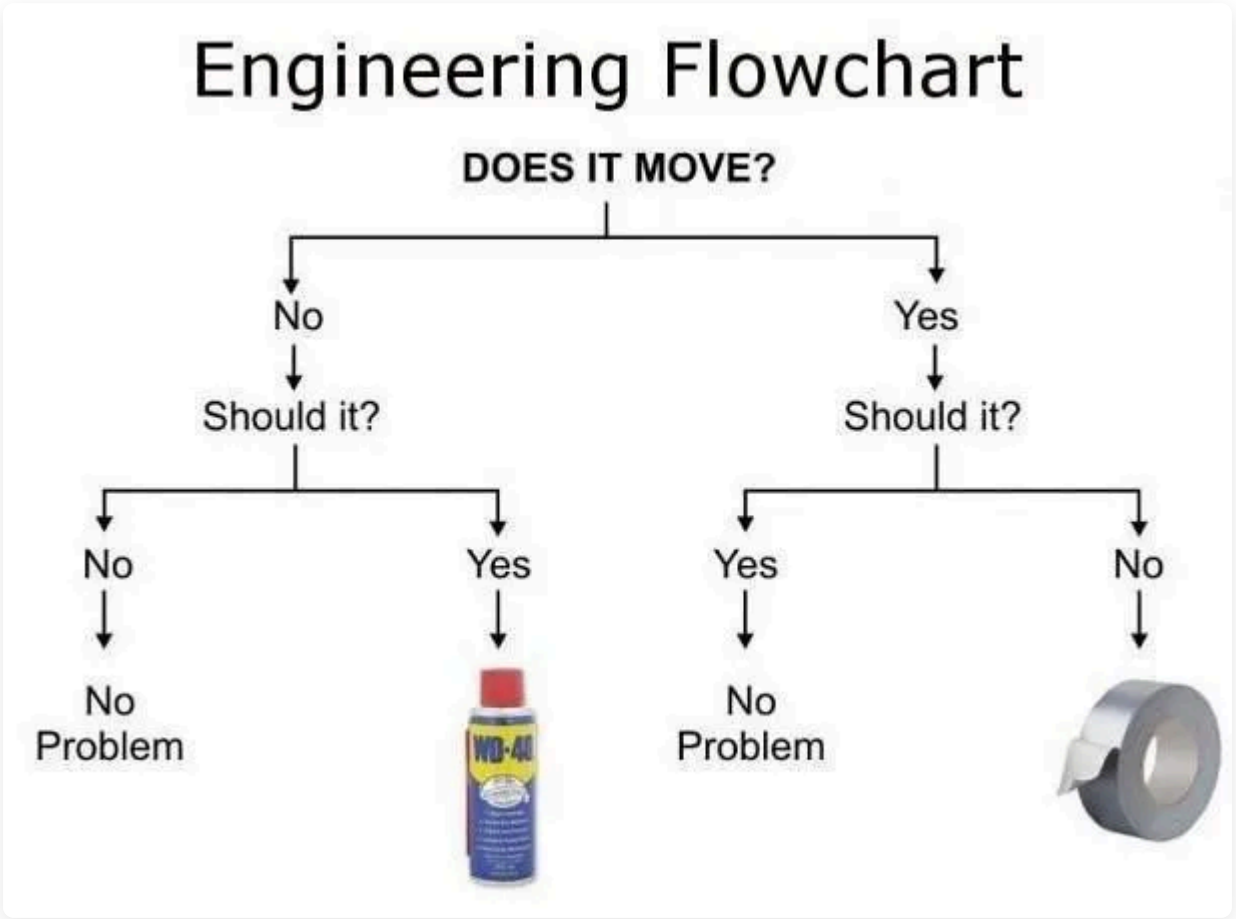
> *Lorsque nous débutons l'apprentissage de la programmation, l'un des concepts fondamentaux à maîtriser est celui des conditions. Elles sont au cœur de la prise de décision dans nos codes, permettant à nos programmes d'agir différemment selon les situations rencontrées. Dans cet article, nous explorerons l'importance et l'utilité des conditions en algorithmique, avec un focus particulier sur leur mise en œuvre en Python.*

Les conditions sont au cœur de la prise de décision dans nos codes, permettant à nos programmes d'agir différemment selon les situations rencontrées.

Qu'est-ce qu'une condition en programmation?

Une condition en programmation est une expression qui évalue si une proposition est **vraie** ou **fausse**. Cela permet au programme de **décider** de l'exécution ou non d'un bloc de code. En Python, comme dans la plupart des langages de programmation, cela se traduit souvent par l'utilisation d'instructions `if`, `elif`, et `else` (si, sinon si, sinon).

En algorithmique, c'est ce que l'on appelle une **structure conditionnelle**.



L'utilité des conditions

Les conditions rendent nos programmes intelligents et **flexibles** et leur permettent de **prendre des décisions**.

Adaptation aux données entrantes

Dans les applications du monde réel, les données ne sont pas statiques; elles varient constamment. Les conditions permettent à nos programmes de **s'adapter** à ces données dynamiques. Par exemple, dans une application météo, les recommandations vestimentaires peuvent changer en fonction de la température actuelle.

Validation des entrées

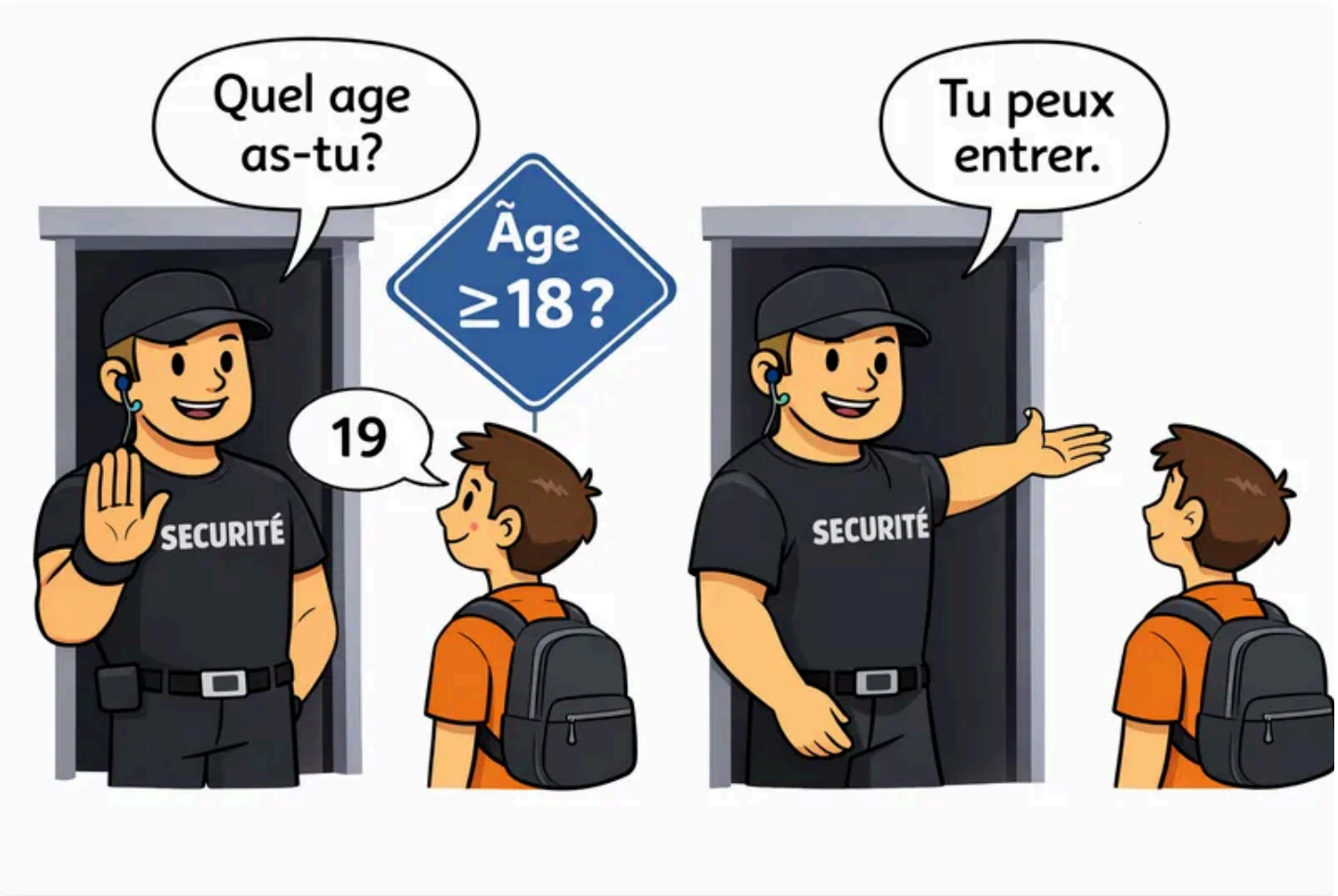
Un autre usage crucial des conditions est la **validation des entrées utilisateur**. Avant de traiter les données fournies par l'utilisateur, il est sage de vérifier leur validité. Les conditions aident à assurer que notre programme ne traite que des entrées appropriées, évitant ainsi des erreurs potentielles.

Logique de contrôle de flux

Les conditions sont essentielles pour **contrôler le flux d'un programme**. Elles permettent de réaliser des boucles, des sauts conditionnels et des arrêts, en fonction de la logique métier de notre application. Cela rend le code non seulement plus lisible mais aussi plus efficace.

Mise en œuvre en Python

Python offre une syntaxe claire et concise pour la mise en œuvre des conditions. Voici un exemple:




```
age = 19
if age >= 18:
    print("Vous pouvez entrer.")
```

`Vous pouvez entrer.` s'affiche uniquement si age est plus grand que 18. Essaie avec `age = 17`

```
age = 18
if age >= 18:
    print("Vous êtes majeur.")
else:
    print("Vous êtes mineur.")
```

Dans cet exemple, le programme vérifie si l'âge de l'utilisateur est supérieur ou égal à 18. Selon le résultat, il affiche un message approprié. Python supporte également les conditions complexes avec l'utilisation des opérateurs logiques `and`, `or`, et `not`, permettant de combiner plusieurs conditions.

Conclusion

Les conditions sont un pilier de l'algorithmique et de la programmation. Elles injectent de la logique et de la flexibilité dans nos codes, permettant de construire des applications dynamiques qui répondent intelligemment à diverses situations. En maîtrisant l'usage des conditions, en particulier en Python, on ouvre la porte à la création de programmes plus complexes et plus fonctionnels, capables de résoudre des problèmes réels et d'améliorer l'interaction utilisateur.

Python: Les conditions simples

> *Les conditions sont un aspect fondamental de la programmation en Python. Elles permettent à votre programme de prendre des décisions en fonction de certaines conditions ou de réagir différemment en fonction de situations spécifiques. Dans cet article, nous allons explorer les structures de contrôle conditionnelles en Python, notamment les instructions `if`, `elif` (else if), et `else`.*

L'instruction `if`

L'instruction `if` est utilisée pour exécuter un bloc de code si une condition spécifique est vraie (`True`). Voici un exemple :

```
age = 15
if age >= 18:
    print("Vous êtes majeur.")
```

Dans cet exemple, le code dans le bloc d'instructions sous `if` ne sera exécuté que si la condition `age ≥ 18` est vraie.

L'instruction `else`

L'instruction `else` est utilisée pour exécuter un bloc de code lorsque la condition de l'instruction `if` est fausse. Voici un exemple :

```
age = 15
if age >= 18:
    print("Vous êtes majeur.")
else:
    print("Vous êtes mineur.")
```

Dans ce cas, si la condition `age ≥ 18` est fausse, le bloc de code sous `else` sera exécuté.

L'instruction `elif` (else if)

L'instruction `elif` est utilisée pour gérer **plusieurs conditions** alternatives. Elle vous permet de tester plusieurs conditions **les unes après les autres jusqu'à ce que l'une d'entre elles soit vraie**. Voici un exemple :

```
note = 80
if note >= 90:
    print("A")
elif note >= 80:
    print("B")
elif note >= 70:
    print("C")
else:
    print("D")
```

Dans cet exemple, le code teste la note et affiche la lettre correspondante en fonction de la plage de notes.

Combinaison de Conditions

Vous pouvez également combiner des conditions en utilisant les opérateurs logiques `and`, `or`, et `not`. Voici un exemple :

```
age = 25
if age >= 18 and age <= 60:
    print("Vous êtes dans la tranche d'âge active.")
```

Cette condition vérifie à la fois si l'âge est supérieur ou égal à 18 et inférieur ou égal à 60.

Pratique

PERMIS DE CHASSE

Reprends l'exercice sur le **permis de chasse**. A la fin du programme, écris le texte " `Vous avez perdu votre permis` " si le nombre de points restants est inférieur ou égal à zéro.

DIVISION

Reprends l'exercice sur la **division entière**. Teste ton programme avec les valeurs 20 et 0. Que se passe-t-il ? Maintenant, *upgrade* ton programme de la façon suivante: Si le diviseur est différent de zéro, effectue la division, sinon, affiche le message " `Impossible de diviser par 0` ".

LES NOMBRES

Étant donné un nombre entier - imprimez "STRICTEMENT POSITIF" s'il est strictement positif. - imprimez "IMPAIR" s'il est impair et imprimez "PAIR" sinon.

IMC

Reprends l'exercice sur l'IMC. Adapte la sortie finale pour afficher l'interprétation de l'IMC selon l'OMS:

Indice de masse corporelle (IMC)	Interprétation (d'après l'OMS)
moins de 18,5	Insuffisance pondérale (maigreur)
18,5 à 25	Corpulence normale
25 à 30	Surpoids
30 à 35	Obésité modérée
35 à 40	Obésité sévère
plus de 40	Obésité morbide ou massive

Conclusion

Les structures de contrôle conditionnelles en Python vous permettent de prendre des décisions dans votre code en fonction de conditions spécifiques. Vous pouvez utiliser les instructions `if`, `elif`, et `else` pour gérer différentes situations. La maîtrise des conditions est essentielle pour écrire des programmes flexibles et réactifs. N'hésitez pas à pratiquer et à expérimenter pour approfondir vos connaissances en programmation Python.

Python: Opérations de comparaison

> *Les opérateurs de comparaison en Python (et dans de nombreux autres langages) vous permettent de comparer des valeurs pour évaluer des conditions et prendre des décisions dans votre code. Dans cet article, nous allons explorer les opérateurs de comparaison en Python, comment les utiliser et à quoi ils servent.*

Qu'est-ce qu'un Opérateur de Comparaison?

En programmation, un opérateur de comparaison est un **symbole** ou un mot-clé qui permet de **comparer deux valeurs ou expressions**. Le résultat de cette comparaison est une valeur booléenne, c'est-à-dire `True` (vrai) ou `False` (faux). Les opérateurs de comparaison vous permettent de répondre à des questions telles que "Est-ce que ces valeurs sont **égales** ?" ou "Est-ce que cette valeur est **plus grande** que celle-là ?".

En Python, voici les principaux opérateurs de comparaison :

- `==` : Égal à
- `≠` : Différent de
- `<` : Inférieur à
- `>` : Supérieur à
- `≤` : Inférieur ou égal à
- `≥` : Supérieur ou égal à
- `... ≤ x ≤ ...` : intervalle

Exemples d'Utilisation

Voyons quelques exemples d'utilisation de ces opérateurs :

Égal à (`==`)

```
a = 5
b = 5
resultat = (a == b)  # Cela renverra True car a est égal à b.
print("Résultat:", resultat) # True
```

💡 L'opérateur de comparaison est `==` (2x le signe `=`) **afin de ne pas le confondre avec l'opérateur d'affectation**.

Taper ce code dans un éditeur:

```
a = 5
b = 15

print ( a == b)
print ( a = b)
```

Que se passe-t-il à la ligne 4 et à la ligne 5 ?

Différent de (`!=`)

```
x = 10
y = 20
resultat = (x != y)  # Cela renverra True car x est différent de y.
print("Résultat:", resultat) # True
```

Inférieur à (`<`)

```
age = 15
limite_age = 18
resultat = (age < limite_age)  # Cela renverra True car age est inférieur à limite_age.
print("Résultat:", resultat) # True
```

Supérieur à (`>`)

```
note = 85
note_passage = 60
resultat = (note > note_passage)  # Cela renverra True car note est supérieur à note_passag
print("Résultat:", resultat) # True
```

Inférieur ou égal à (`<=`)

```
temps = 25
limite_temps = 30
resultat = (temps <= limite_temps)  # Cela renverra True car temps est inférieur ou égal à
print("Résultat:", resultat) # True
```

Supérieur ou égal à (`>=`)

```
nombre1 = 50
nombre2 = 50
resultat = (nombre1 >= nombre2)  # Cela renverra True car nombre1 est supérieur ou égal à n
print("Résultat:", resultat) # True
```

Intervalles

En Python, il est possible de vérifier si une valeur se trouvent entre deux bornes, comme dans l'expression suivante :

```
-50 <= temperature <= 50
```

Cette syntaxe permet de vérifier en une seule étape si une valeur se trouve dans un certain intervalle. Cela revient à combiner deux comparaisons logiques à l'aide de l'opérateur logique `and`. Concrètement, l'expression ci-dessus est équivalente à :

```
-50 <= temperature and temperature <= 50
```

EXPLICATION TECHNIQUE :

1. **Lecture fluide** : L'utilisation d'une double inégalité améliore la lisibilité du code, car elle s'apparente à une notation mathématique couramment utilisée.
2. **Évaluation de gauche à droite** : Python évalue les expressions de gauche à droite. D'abord, il vérifie si `-50 ≤ temperature`. Si cette condition est vraie, il continue avec `temperature ≤ 50`. Si l'une des deux conditions échoue, l'évaluation s'arrête immédiatement.
3. **Type de valeur retournée** : Le résultat de la comparaison est un booléen (`True` ou `False`).

EXEMPLE CONCRET :

Voici un exemple qui vérifie si une température est dans une plage réaliste pour des mesures météorologiques :

```
temperature = 25

if -50 <= temperature <= 50:
    print("La température est dans la plage acceptable.")
else:
    print("La température est hors plage !")
```

Dans cet exemple, la valeur `25` répond à la condition `-50 ≤ temperature ≤ 50`, donc le programme affichera :

```
La température est dans la plage acceptable.
```

CAS D'UTILISATION :

- Vérification de plages de valeurs numériques (par exemple, température, âge, scores, etc.).
- Validation de données dans des intervalles bien définis.
- Simplification des conditions multiples pour améliorer la lisibilité du code.

BONNES PRATIQUES :

- Préférez cette notation compacte lorsque vous travaillez avec des comparaisons d'intervalles.
- Assurez-vous que cette syntaxe est claire pour votre public cible ou documentez-la si nécessaire pour éviter toute confusion.

Combinaison d'Opérateurs de Comparaison

Vous pouvez également combiner plusieurs opérateurs de comparaison pour créer des expressions plus complexes. Par exemple, vous pouvez utiliser les opérateurs `and` et `or` pour combiner des conditions.

```
age = 25
sexe = "Femme"
resultat = (age >= 18) and (sexe == "Femme")  # Cela renverra True si l'âge est supérieur o
print("Résultat:", resultat) # True
```

Lire l'article complet

Pratique

Voici une série de comparaisons. Essaie d'abord de deviner le résultat de chacune d'entre elles. Tape ensuite ce code dans un éditeur pour vérifier tes hypothèses.

```
a = 5
b = 7
print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

Bonus

Quel est le résultat du code suivant?

```
a = 25
b = "25"
print (a == b)
```

La réponse se trouve ici

Conclusion

Les opérateurs de comparaison sont des outils puissants pour évaluer des conditions et prendre des décisions dans vos programmes Python. Ils vous permettent de comparer des valeurs de manière simple et efficace. En comprenant comment utiliser ces opérateurs, vous serez en mesure d'écrire un code plus précis et plus adapté à vos besoins. N'hésitez pas à les utiliser pour résoudre des problèmes et créer des applications plus sophistiquées.

Python: Conditions / Exercices de base

> *Voici une série d'exercices de base et cohérents pour travailler **`if / else`**, pensés pour s'exercer au début. Chaque situation a un **sens concret**, mobilise uniquement des **opérations mathématiques**, des **`input()`**, et **une seule condition** par exercice, sans `and` / `or`, sans boucles.*

Consignes générales

Envoyez vos fichiers dans le dossier `Python/Conditions simples/Exercices 01`.

01 – Vérifier si un nombre est positif ou négatif

`01_positif_negatif.py`

L'utilisateur encode un nombre. Afficher `"positif"` s'il est strictement supérieur à 0, sinon afficher `"négatif ou nul"`.

---

02 – Tester la majorité

`02_majorite.py`

L'utilisateur encode son âge. Afficher `"majeur"` si l'âge est au moins 18, sinon afficher `"mineur"`.

---

03 – Comparer deux nombres

`03_comparer_deux_nombres.py`

L'utilisateur encode deux nombres. Afficher `"le premier est plus grand"` si le premier est strictement supérieur au second, sinon afficher `"le second est plus grand ou égal"`.

---

04 – Pair ou impair

`04_pair_impair.py`

L'utilisateur encode un nombre entier. Afficher `"pair"` si le nombre est divisible par 2, sinon afficher `"impair"`.

---

05 – Note réussie ou ratée

`05_note_reussite.py`

L'utilisateur encode une note sur 20. Afficher `"réussi"` si la note est au moins 10, sinon afficher `"raté"`.

▸ 06 – Température extérieure

`06_temperature.py`

L'utilisateur encode une température en degrés Celsius. Afficher `"il fait froid"` si la température est inférieure à 10, sinon afficher `"il fait bon"`.

▸ 07 – Mot trop long ou non

`07_longueur_mot.py`

L'utilisateur encode un mot. Afficher `"mot long"` si le mot contient plus de 5 caractères, sinon afficher `"mot court"`.

Pour calculer la taille du mot, utiliser : `len(mot)`

▸ 08 – Solde bancaire suffisant

`08_solde_bancaire.py`

L'utilisateur encode un montant d'argent disponible. Afficher `"paiement accepté"` si le montant est au moins 50, sinon afficher `"paiement refusé"`.

▸ 09 – Comparaison avec zéro

`09_comparaison_zero.py`

L'utilisateur encode un nombre. Afficher `"non nul"` si le nombre est différent de 0, sinon afficher `"nul"`.

▸ 10 – Distance autorisée

`10_distance_autorisee.py`

L'utilisateur encode une distance en kilomètres. Afficher `"trajet court"` si la distance est inférieure ou égale à 10, sinon afficher `"trajet long"`.

▸ 11 – Budget suffisant

`11_budget_suffisant.py`

L'utilisateur encode un budget et un prix. Afficher `"achat possible"` si le budget est supérieur ou égal au prix, sinon afficher `"achat impossible"`.

▸ 12 – Présence d'un caractère spécial

`12_caractere_special.py`

L'utilisateur encode un mot. Afficher `"contient un point"` si le caractère `"."` est présent dans le mot, sinon afficher `"ne contient pas de point"`.

▸ 13 – Code secret

`13_code_secret.py`

L'utilisateur encode un nombre. Afficher `"code correct"` si le nombre est égal à `1234`, sinon afficher `"code incorrect"`.

▸ 14 – Joueur dans l'écran

`14_joueur_dans_ecran.py`

On connaît :

- `position_joueur`
- `largeur_ecran`

Afficher `"dans l'écran"` si le joueur dépasse **à droite** (`position_joueur > largeur_ecran`), sinon afficher `"ok"`.

(Version volontairement limitée à une seule condition, sans `and`)

▸ 15 – Déplacement avec retour à gauche

`15_deplacement_ecran.py`

L'utilisateur encode :

- `position_joueur`
- `largeur_ecran`
- `deplacement`

Calculer la nouvelle position. Si le joueur dépasse la largeur de l'écran, le remettre à `0`. Afficher la position finale.

▸ 16 – Trésor trouvé

`16_tresor.py`

On connaît `case_tresor`. L'utilisateur encode `position_joueur`. Afficher `"trésor trouvé"` si les deux valeurs sont égales, sinon `"rien ici"`.

▸ 17 – Points de vie après attaque

`17_points_de_vie.py`

On connaît `points_de_vie` et `attaque`. Calculer les points de vie restants. Afficher `"vivant"` si le résultat est strictement supérieur à 0, sinon `"mort"`.

18 – Multiples et parité

`18_multiples.py`

L'utilisateur encode un entier.

Afficher :

- `"pair"` ou `"impair"`
- `"multiple de 2"` si divisible par 2
- `"multiple de 3"` si divisible par 3
- `"multiple de 4"` si divisible par 4
- `"multiple de 5"` si divisible par 5
- `"multiple de 6"` si divisible par 6

Exemple :

```
Nombre : 12

pair
multiple de 2
multiple de 3
multiple de 4
multiple de 6
```

19 – Carré ou rectangle

`19_carre_rectangle.py`

L'utilisateur encode :

- `largeur`
- `hauteur`

Afficher `"Carré"` si les deux valeurs sont identiques, sinon afficher `"Rectangle"`.

20 – Achat possible ou non

`20_achat_possible.py`

L'utilisateur encode :

- `argent_disponible`
- `prix_article`
- `quantite`

Calculer le prix total de l'achat. Afficher `"achat possible"` si l'argent est suffisant, sinon `"argent insuffisant"`.

Exemple :

```
Argent disponible : 75
Prix d'un article : 15
Articles achetés : 4

Prix total : 100
Argent insuffisant
```

Python: Les conditions (combinaisons)

> *Dans le monde de la programmation, la capacité à combiner plusieurs conditions représente une compétence essentielle pour construire des logiciels complexes et réactifs. Cet article explore la théorie et l'utilité de cette pratique, en se concentrant sur sa mise en œuvre et ses avantages dans le développement de logiciels.*

Introduction à la combinaison de conditions

La programmation conditionnelle permet aux développeurs de **diriger le flux d'exécution d'un programme** selon que certaines **conditions** sont remplies ou non. En combinant plusieurs de ces conditions, on peut créer des instructions **plus précises** et adaptatives, capables de gérer des **situations complexes** avec élégance et efficacité.

Pourquoi combiner des conditions?

La combinaison de conditions permet de traiter des cas spécifiques qui **dépendent de plusieurs facteurs**. Imaginons une application de commerce en ligne où l'accès à certaines promotions est conditionné non seulement par le statut de membre du client mais aussi par le montant de son panier d'achat. En utilisant une combinaison de conditions, l'application peut facilement déterminer si ces critères sont remplis.

La syntaxe de base en Python

Python, avec sa syntaxe claire et intuitive, facilite la combinaison de conditions. Les opérateurs logiques tels que `and`, `or`, et `not` sont utilisés pour combiner des conditions simples en expressions plus complexes.

```
if membre == True and montant_achat >= 50:
    appliquer_promotion()
```

Dans cet exemple, la fonction `appliquer_promotion` n'est exécutée que si l'utilisateur est un membre **ET** que le montant de son achat est supérieur ou égal à 50. Cela illustre comment une combinaison de conditions peut être utilisée pour contrôler le flux d'exécution d'un programme de manière précise.

Conclusion

La combinaison de conditions est une technique puissante qui, bien maîtrisée, permet de créer des logiciels plus intelligents, réactifs et adaptés aux besoins spécifiques des utilisateurs. En suivant les bonnes pratiques de développement et en faisant preuve de rigueur dans la conception et les tests, les développeurs peuvent exploiter pleinement le potentiel de cette approche pour résoudre des problèmes complexes et offrir des expériences utilisateur enrichissantes.

Python - Les opérateurs logiques

> *Les opérateurs logiques permettent d'évaluer des expressions booléennes et prendre des décisions en fonction de ces évaluations. En Python, les opérateurs logiques les plus couramment utilisés sont `and`, `or`, `xor` et `not`. Dans cet article, nous explorerons chacun de ces opérateurs et leur utilisation dans des situations pratiques.*

💡 Un opérateur logique permet de "combiner" 2 ou plusieurs valeurs booléennes et renvoie une valeur booléenne.

L'opérateur `and`

L'opérateur `and` retourne `True` si **les deux** expressions qu'il relie **sont vraies**, sinon il retourne `False`.

Table de vérité pour l'opérateur `and` :

A	B	A AND B
✅ True	✅ True	✅ True
✅ True	❌ False	❌ False
❌ False	✅ True	❌ False
❌ False	❌ False	❌ False

Exemple :

```
x = 5
y = 10
result = (x < 10) and (y > 5)
print(result)  # Output: True
```

Dans cet exemple, l'expression `(x < 10) and (y > 5)` est vraie car les deux conditions sont satisfaites : `x` est inférieur à 10 et `y` est supérieur à 5.

Il est bien sûr possible de combiner plus de 2 comparaisons, comme dans l'exemple suivant:

Vérification de l'éligibilité pour un concours

Imaginons qu'on vérifie si une personne est éligible pour participer à un concours. Les critères sont les suivants :

- L'âge doit être supérieur ou égal à 18 ans.
- Elle doit être résidente en Belgique.
- Ca doit être sa première participation au concours.

```python
age = 20
residence = "Belgique"
premiere_participation = True

if age >= 18 and residence == "Belgique" and premiere_participation:
    print("Vous êtes éligible pour participer au concours.")
else:
    print("Vous ne remplissez pas les critères pour participer au concours.")
```

Il est bien sûr possible de **combiner plusieurs comparaisons**. Dans le cas du `and`, l'expression sera considérée comme `True` si **TOUTES les conditions sont** `True`.

L'opérateur `or`

L'opérateur `or` retourne `True` si **au moins l'une** des expressions qu'il relie **est vraie**, sinon il retourne `False`.

Table de vérité pour l'opérateur `or` :

A	B	A OR B
✅ True	✅ True	✅ True
✅ True	❌ False	✅ True
❌ False	✅ True	✅ True
❌ False	❌ False	❌ False

Voici un exemple :

```python
x = 5
y = 10
result = (x > 10) or (y < 20)
print(result)  # Output: True
```

Dans cet exemple, l'expression `(x > 10) or (y < 20)` est vraie car au moins l'une des conditions est satisfaites : `y` est inférieur à 20.

Il est bien sûr possible de **combiner plusieurs comparaisons**. Dans le cas du `or`, l'expression sera considérée comme `True` si **au moins UNE des conditions est** `True`.

Exemple:

```
x = 5
y = 10
z = 15

if x > 0 and y < 5 and z == 15:
    print("Toutes les conditions sont remplies.")
else:
    print("Au moins une condition est fausse.")
```

L'opérateur `xor`

L'opérateur `xor` (ou exclusif) retourne `True` si une et une seule des expressions qu'il relie est vraie, sinon il retourne `False`. En Python, il n'existe pas d'opérateur `xor` natif, mais vous pouvez le simuler en utilisant l'opérateur `≠` (différent de).

Table de vérité pour l'opérateur `xor` :

A	B	A XOR B
✅ True	✅ True	❌ False
✅ True	❌ False	✅ True
❌ False	✅ True	✅ True
❌ False	❌ False	❌ False

Voici un exemple :

```
x = True
y = False
result = x != y
print(result)  # Output: True
```

Dans cet exemple, l'expression `x ≠ y` est vraie car une des variables est vraie et l'autre est fausse.

L'opérateur `not`

L'opérateur `not` **inverse la valeur** d'une expression booléenne. Si l'expression est vraie, `not` la rendra fausse, et vice versa.

Table de vérité pour l'opérateur `not` :

A	NOT A
✅ True	❌ False
❌ False	✅ True

Voici un exemple :

```
x = True
result = not x
print(result)  # Output: False
```

Dans cet exemple, la valeur de `x` est vraie, mais `not x` inverse cette valeur pour retourner `False`.

Ordre de priorité

Lorsque vous utilisez **plusieurs opérateurs logiques** dans une expression Python, il est important de comprendre leur ordre de priorité, également appelé **précédence des opérateurs**. Cela **détermine l'ordre dans lequel les opérateurs sont évalués dans une expression**. Voici l'ordre de priorité des opérateurs logiques, du plus haut au plus bas :

1. **not** : L'opérateur `not` a la plus haute précédence. Il est utilisé pour inverser la valeur d'une expression booléenne.
2. **and** : Ensuite vient l'opérateur `and`. L'expression est évaluée de gauche à droite et l'évaluation s'arrête dès qu'une valeur `False` est trouvée, car la conjonction de deux expressions nécessite que les deux soient vraies.
3. **or** : Enfin, l'opérateur `or` a la plus basse précédence. De la même manière que l'opérateur `and`, l'expression est évaluée de gauche à droite et l'évaluation s'arrête dès qu'une valeur `True` est trouvée, car la disjonction de deux expressions nécessite que au moins une soit vraie.

Il est important de noter que l'utilisation de parenthèses peut modifier l'ordre d'évaluation dans une expression. Les parenthèses ont la priorité la plus élevée et sont évaluées en premier. Par conséquent, vous pouvez utiliser des parenthèses pour spécifier explicitement l'ordre dans lequel les opérateurs doivent être évalués.

En comprenant l'ordre de priorité des opérateurs logiques, vous pouvez écrire des expressions booléennes complexes de manière claire et précise, en vous assurant que l'évaluation se déroule comme prévu.

Conclusion

Les opérateurs logiques en Python peuvent évaluer des expressions booléennes et vous aider à prendre des décisions dans vos programmes. En comprenant comment utiliser `and`, `or`, `xor` et `not`, vous pouvez écrire du code plus expressif et plus concis tout en contrôlant le flux de votre programme en fonction des conditions logiques.

Exercices

Devine le résultat de ces programmes **SANS LES EXÉCUTER** dans Python.

Ensuite, vérifie ton hypothèse en tapant le programme (PAS DE COPIER-COLLER) et en déboguant à l'aide de la touche `F7`.

1. **Exercice 1 :**

```
x = 5
y = 10
result = (x > 3) and (y < 20)
print(result)
```

Devinez le résultat du programme.

2. **Exercice 2 :**

```
a = True
b = False
c = True
result = a and b or c
print(result)
```

Devinez le résultat du programme.

3. **Exercice 3 :**

```
x = 15
y = 25
result = (x < 10) or (y > 20)
print(result)
```

Devinez le résultat du programme.

4. **Exercice 4 :**

```
a = True
b = False
result = not a or b
print(result)
```

Devinez le résultat du programme.

5. **Exercice 5 :**

```
x = 8
y = 12
result = (x % 2 == 0) and (y % 3 == 0)
print(result)
```

Devinez le résultat du programme.

6. **Exercice 6 :**

```
a = False
b = True
result = a or (not b)
print(result)
```

Devinez le résultat du programme.

7. **Exercice 7 :**

```
x = 20
y = 30
result = (x > 10) and (y < 25) or (x + y == 50)
print(result)
```

Devinez le résultat du programme.

8. **Exercice 8 :**

```
a = True
b = True
result = (not a) or (not b)
print(result)
```

Devinez le résultat du programme.

9. **Exercice 9 :**

```
x = 5
y = 10
z = 15
result = (x + y > 10) and (y - z < 0) or (x * z == 75)
print(result)
```

Devinez le résultat du programme.

10. **Exercice 10 :**

```
a = True
b = False
result = (a and b) or (not a and not b)
print(result)
```

Devinez le résultat du programme.

Ces exercices vous permettront de tester votre compréhension des opérateurs logiques en Python en essayant de prédire le résultat de chaque expression booléenne.

Python: Conditions / Combinaisons / Exercices

> *Exercices supplémentaires sur les conditions*

Voici une série d'exercices basés sur les structures conditionnelles `if`, `else`, et `elif` et le opérateurs de combinaison `and`, `or` et `not` en Python, orientés autour du thème des jeux vidéo. Ces exercices visent à renforcer la compréhension des structures de contrôle de flux dans un contexte ludique et engageant.

ces exercices sur le thème des jeux vidéo sont conçus pour vous aider à pratiquer la combinaison de deux ou trois conditions dans des instructions `if` à l'aide des opérateurs `OR` et `AND`.

Consignes générales

Dans les exercices suivants, quand l'utilisateur doit faire un choix parmi plusieurs propositions, vous devez

- créer une liste qui contient la liste des choix possibles
- afficher un menu de sélection Exemple:

  ```
  1. Épée
  2. Arc
  3. Hache
  ```

  Autre exemple:

  ```
  Épée
  Arc
  Hache
  ```

- Pour le menu, choisissez la version qui correspond le mieux à ce que vous avez déjà vu en classe.
- demander son choix à l'utilisateur (`int(input( ... ))`)
- vérifier que le choix est correct. Si vous avez déjà vu les boucles, vous pouvez répétez l'étape de sélection tant que l'utilisateur effectue un choix non valide.
- Envoyez vos fichiers dans le dossier `Python/Conditions combinaisons`.

Exercice 1: Sélection de personnage

Nom de fichier: `01.personnage.py` Demandez à l'utilisateur de choisir un personnage entre "Guerrier", "Mage" ou "Archer". Si l'utilisateur choisit "Guerrier" ou "Mage", affichez "Vous avez choisi un personnage de combat à distance.".

Exercice 2: Niveau d'accès

Nom de fichier: `02.niveau.py` Un jeu vidéo propose trois niveaux d'accès: "Débutant", "Intermédiaire", "Avancé" et "Expert". Demandez à l'utilisateur de saisir son niveau. Si l'utilisateur n'est ni "Débutant" ni "Intermédiaire", informez-le qu'il accède au niveau "Ninja".

Exercice 3: Équipe de héros

Nom de fichier: `03.equipe.py` Dans un jeu, l'utilisateur peut choisir de rejoindre l'équipe des "Héros" ou des "Vilains". Si l'utilisateur saisit "Héros" et son âge est supérieur ou égal à 18, affichez "Bienvenue dans l'équipe des Héros, adulte!".

Exercice 4: Arme spéciale

Nom de fichier: `04.arme_speciale.py` Un joueur peut choisir une arme parmi "Épée", "Arc", ou "Bâton". Si le joueur choisit "Épée" ou "Arc" et a plus de 100 points, il obtient une version améliorée de l'arme..

Exercice 5: Porte secrète

Nom de fichier: `05.porte_secrete.py` Dans un jeu d'aventure, pour ouvrir une porte secrète, le joueur doit posséder une clé "d'or" ou "d'argent" et connaître le mot de passe. Vérifiez si le joueur remplit les conditions.

Exercice 6: Capacité de personnage

Nom de fichier: `06.capacite.py` Un personnage peut être "Rapide" ou "Fort". Si le personnage est "Rapide" et a moins de 50 points de santé ou s'il est "Fort" et a plus de 50 points de santé, il survit.

Exercice 7: Accès au donjon

Nom de fichier: `07.donjon.py` Le donjon est ouvert aux joueurs de niveau 20 à 30 ou ceux qui ont un objet spécial. Demandez au joueur son niveau et s'il possède l'objet spécial (Oui/Non). Utilisez `OR` et `AND`.

Exercice 8: Mode nocturne

Nom de fichier: `08.mode_nocturne.py` Dans un jeu, le mode nocturne est activé si le joueur est dans la "Forêt sombre" ou le "Château hanté" et qu'il est entre 22 heures et 5 heures. Utilisez `AND` et `OR`.

Exercice 9: Compétition de jeu

Nom de fichier: `09.competition.py` Une compétition est ouverte aux joueurs de "niveau 5" ou plus qui jouent soit à "LoL" soit à "Apex". Vérifiez l'éligibilité. Utilisez `AND` et `OR`.

Exercice 10: Choix de quête

Nom de fichier: `10.quete.py` Un joueur doit choisir entre la quête "Chasse au trésor" ou "Sauvetage du prince". Si le joueur choisit "Chasse au trésor" et possède une carte, ou "Sauvetage du prince" et a une épée, il peut commencer. Utilisez `AND` et `OR`.

Exercice 11: Potion magique

Nom de fichier: `11.potion_magique.py` Une potion magique est disponible si le joueur a soit des "herbes rares" soit des "cristaux magiques" et plus de 200 points de mana. Utilisez `AND` et `OR`.

▸ Exercice 12: Accès VIP

Nom de fichier: `12.acces_vip.py` L'accès VIP est accordé si le joueur a terminé soit le niveau "Dragon de feu" soit le "Labyrinthe des ombres" et a un score supérieur à 5000 points. Utilisez `AND` et `OR`.

Python - Les opérateurs logiques - Évaluation paresseuse

> *L'évaluation paresseuse, appelée aussi short-circuit evaluation, est une technique utilisée par les langages de programmation comme Python pour optimiser l'exécution des expressions logiques. Elle consiste à arrêter l'évaluation d'une expression logique dès qu'il n'est plus nécessaire de continuer, en fonction du résultat déjà déterminé.*

Évaluation Paresseuse (Short-Circuit Evaluation)

L'évaluation paresseuse, appelée aussi **short-circuit evaluation**, est une technique utilisée par les langages de programmation comme Python pour optimiser l'exécution des expressions logiques. Elle consiste à **arrêter l'évaluation d'une expression logique dès qu'il n'est plus nécessaire de continuer**, en fonction du résultat déjà déterminé.

Principe de base

L'évaluation paresseuse repose sur les règles fondamentales des opérateurs logiques `and` et `or` :

1. **Pour `and` :**
   - Si une condition est `False`, le résultat final sera forcément `False`, peu importe les autres conditions restantes. Il est inutile de continuer l'évaluation.
   - Exemple :

     ```
     condition1 and condition2 and condition3
     ```

     - Si `condition1` est `False`, Python s'arrête et ignore `condition2` et `condition3`.
2. **Pour `or` :**
   - Si une condition est `True`, le résultat final sera forcément `True`, peu importe les autres conditions restantes. Il est inutile de continuer l'évaluation.
   - Exemple :

     ```
     condition1 or condition2 or condition3
     ```

     - Si `condition1` est `True`, Python s'arrête et ignore `condition2` et `condition3`.

Avantages de l'évaluation paresseuse

1. **Amélioration des performances :**
   - L'évaluation paresseuse évite de calculer inutilement les conditions restantes, ce qui économise du temps et des ressources.
   - Cela est particulièrement utile si certaines conditions sont complexes ou coûteuses à évaluer (exemple : fonctions ou requêtes vers une base de données).
2. **Sécurité et gestion des erreurs :**
   - Permet d'éviter des erreurs en ne procédant pas à des évaluations inutiles.
   - Exemple : Protéger une division par zéro.

```
a = 0
b = 5
if a != 0 and b / a > 1:  # Python n'exécute pas b / a si a == 0
    print("Condition remplie")
else:
    print("Condition non remplie")
```

Exemples Concrets

Teste ces exemples avec Thonny en mode debug, en exécutant ces codes pas à pas (`F7`). Tu verras effectivement que toutes les conditions ne sont pas testées.

1. Évaluation avec `and`

```
x = 5
y = 10

if x > 0 and y / x > 1:
    print("Conditions valides")
else:
    print("Au moins une condition est invalide")
```

DÉROULEMENT :

1. Python évalue `x > 0` → Résultat : `True` .
2. Passe à `y / x > 1` → Résultat : `True` .
3. Le résultat final est donc `True` et l'instruction `print("Conditions valides")` est exécutée.

Mais si `x = 0` :

1. Python évalue `x > 0` → Résultat : `False` .
2. Python s'arrête immédiatement sans évaluer `y / x` (prévenant une division par zéro).

2. Évaluation avec `or`

```
def test():
    print("Fonction exécutée")
    return True

if True or test():
    print("Condition remplie")
```

DÉROULEMENT :

1. La première condition (`True`) est évaluée.
2. Python s'arrête immédiatement et **n'appelle pas** la fonction `test()` .

Résultat :

```
Condition remplie
```

La phrase `"Fonction exécutée"` n'apparaît pas.

---

3. Combinaison d'expressions

Un cas combinant des calculs et des protections contre des erreurs possibles.

```
a = 0
b = 10

if a != 0 and (b / a > 5 or b > 20):
    print("Condition remplie")
else:
    print("Condition non remplie")
```

DÉROULEMENT :

1. Python commence par évaluer `a ≠ 0` . Si c'est `False` , il s'arrête immédiatement, sans risquer une division par zéro.

Limites et précautions

1. **Effets secondaires :**
    - Si une condition dans une expression logique contient des fonctions qui modifient l'état d'une variable ou effectuent des actions (exemple : impression, écriture de fichiers), ces actions ne seront pas exécutées si Python n'évalue pas cette partie de l'expression.

```python
def modifier_variable():
    global x
    x = 10
    return True

x = 0
if False and modifier_variable():  # modifier_variable() n'est jamais exécutée
    print("Condition remplie")

print(x)  # Résultat : x reste 0
```

2. **Ordre des conditions :**
    - Placez les conditions les plus simples ou les plus probables en premier pour maximiser les avantages de l'évaluation paresseuse.

En résumé

L'évaluation paresseuse est une optimisation qui :

- Améliore les performances en arrêtant les calculs inutiles.
- Préserve la sécurité dans certains cas (comme la gestion des divisions par zéro).
- Fonctionne naturellement avec les opérateurs logiques `and` et `or`, en respectant leurs comportements.

Ce concept est particulièrement utile pour écrire du code efficace et sécurisé tout en minimisant les erreurs potentielles.

Introduction à Python

> *Python est un langage de programmation interprété, polyvalent et convivial qui a gagné en popularité au fil des ans. Créé par Guido van Rossum et publié pour la première fois en 1991, Python est devenu l'un des langages les plus utilisés dans le monde de la programmation en raison de sa simplicité, de sa lisibilité et de sa grande communauté de développeurs.*

Les avantages de Python

Simplicité et lisibilité

L'une des principales caractéristiques de Python est sa syntaxe simple et concise, qui le rend facile à apprendre et à utiliser même pour les débutants. La lisibilité du code Python est également un avantage majeur, permettant aux développeurs de comprendre rapidement le fonctionnement d'un programme.

Polyvalence

Python est un langage polyvalent qui peut être utilisé dans une grande variété de domaines, notamment le développement web, l'analyse de données, l'intelligence artificielle, l'automatisation, la robotique et bien plus encore. Sa polyvalence en fait un outil indispensable pour de nombreux développeurs et entreprises.

Grande bibliothèque standard

Python dispose d'une vaste bibliothèque standard qui offre des fonctionnalités prêtes à l'emploi pour diverses tâches de programmation. Cela permet aux développeurs d'économiser du temps en utilisant des modules et des packages déjà développés plutôt que de réinventer la roue.

Communauté active

La communauté Python est l'une des plus grandes et des plus actives dans le monde de la programmation. Cette communauté dynamique contribue au développement continu de Python en créant de nouveaux packages, en fournissant des ressources d'apprentissage et en soutenant les nouveaux arrivants.

Qui utilise Python et pourquoi ?

Développeurs web

Python est largement utilisé dans le développement web pour la création de sites web, d'applications web et de services web. Des frameworks populaires comme Django et Flask permettent de développer rapidement des applications web robustes et évolutives.

Scientifiques des données

Python est devenu le langage de choix pour de nombreux scientifiques des données en raison de ses puissantes bibliothèques d'analyse de données telles que NumPy, Pandas et Matplotlib. Ces outils facilitent l'exploration, la manipulation et la visualisation des données.

Ingénieurs en intelligence artificielle et en machine learning

Python est largement utilisé dans le domaine de l'intelligence artificielle et de l'apprentissage automatique en raison de sa facilité d'utilisation et de ses bibliothèques spécialisées telles que TensorFlow, Keras et scikit-learn. Ces bibliothèques permettent de développer et de déployer des modèles d'apprentissage automatique de manière efficace.

Développeurs de logiciels et d'applications

De nombreuses entreprises utilisent Python pour le développement de logiciels et d'applications en raison de sa polyvalence, de sa productivité et de sa capacité à s'intégrer facilement avec d'autres langages et technologies.

La popularité croissante de Python

Python connaît une popularité croissante grâce à ses nombreux avantages et à sa polyvalence. Selon divers indices de popularité, tels que l'indice TIOBE et l'enquête Stack Overflow Developer Survey, Python figure parmi les langages de programmation les plus populaires et les plus demandés dans l'industrie.

Sa popularité continue de croître en raison de son adoption par de grandes entreprises comme Google, Facebook, Amazon et Netflix, qui utilisent Python pour diverses applications et services.

En conclusion, Python est bien plus qu'un simple langage de programmation. C'est un outil puissant et polyvalent qui continue de gagner en popularité grâce à sa simplicité, sa lisibilité, sa grande communauté et sa polyvalence. Que vous soyez un débutant en programmation ou un développeur expérimenté, Python offre une multitude d'opportunités pour créer des applications innovantes et résoudre des problèmes complexes dans divers domaines.

Notes de cours

- Télécharger les notes de cours "officielles"

EXERCICES DU JOUR

- Exercices sur les variables
- Pour les courageux
- Exercices sur les conditions
- Exercices sur les boucles

ELECTRONIQUE

Les capteurs

> *Un capteur est un dispositif qui **détecte et mesure une grandeur physique ou une caractéristique de son environnement**, puis **convertit** cette information en un signal utilisable par un système électronique. Les capteurs jouent un rôle essentiel en robotique, car **ils permettent aux robots de percevoir** et d'interagir avec leur environnement.*

Utilité des capteurs

Les capteurs sont utilisés pour fournir aux robots des informations sur le monde qui les entoure. Ils permettent au robot de **recueillir des données sur des grandeurs physiques** telles que la lumière, le son, la température, la pression, la distance, etc. Ces informations sont ensuite utilisées **pour prendre des décisions**, effectuer des actions et interagir avec l'environnement de manière intelligente.

Les capteurs permettent aux robots de **détecter des objets**, d'analyser des caractéristiques de l'environnement, de suivre des lignes, d'éviter des obstacles, de mesurer des paramètres, de percevoir des mouvements, de reconnaître des formes et de nombreux autres aspects nécessaires à leur fonctionnement autonome.

Différents types de capteurs

On distingue généralement deux types principaux de capteurs en fonction de la manière dont ils fournissent des informations : les **capteurs analogiques** et les **capteurs numériques**.

- **Capteurs analogiques** : Les capteurs analogiques fournissent une sortie continue qui varie en fonction de la grandeur physique mesurée. Ils produisent **une tension ou un courant proportionnel à la grandeur mesurée**. Les valeurs de sortie varient généralement de manière linéaire ou non linéaire en fonction de la grandeur mesurée. Les signaux analogiques nécessitent souvent une conversion en signaux numériques pour être traités par un système électronique.
- **Capteurs numériques** : Les capteurs numériques fournissent une sortie discrète, généralement sous forme de bits (0 et 1) ou de valeurs numériques prédéfinies. Ils utilisent des composants électroniques internes pour convertir la grandeur physique en signaux numériques. Les capteurs numériques sont souvent plus précis, plus faciles à utiliser et peuvent être directement connectés à des microcontrôleurs ou à d'autres systèmes électroniques.

Il existe une grande variété de capteurs disponibles, chacun étant conçu pour mesurer une grandeur spécifique. Parmi les autres types de capteurs couramment utilisés en robotique, on peut citer les capteurs de mouvement, les capteurs de proximité, les capteurs de gaz, les capteurs de pression, les capteurs de température, les capteurs de lumière, les capteurs de force, les capteurs de couleur, les capteurs de son, les capteurs d'inclinaison, etc.

L'utilisation appropriée des capteurs permet aux robots de percevoir leur environnement, d'interagir de manière autonome et d'exécuter des tâches spécifiques en fonction des informations recueillies.

Principaux capteurs utilisés en robotique

Voici une liste des principaux capteurs utilisés en robotique, accompagnée d'une brève description de chacun :

Capteur de distance ultrasonique : Ce capteur utilise des ondes sonores pour mesurer la distance entre le robot et un objet environnant. Il est largement utilisé pour l'évitement d'obstacles et la navigation.

En classe, nous avons utilisé le capteur ultra-son HC-SR04, qui est un modèle très répandu et bon marché.

Capteur infrarouge : Les capteurs infrarouges détectent les rayonnements infrarouges émis par les objets. Ils sont utilisés pour la détection d'obstacles, le suivi de lignes, la détection de mouvement, etc.

Capteur de lumière : Les capteurs de lumière mesurent l'intensité de la lumière ambiante. Ils peuvent être utilisés pour détecter la luminosité de l'environnement et ajuster le comportement du robot en conséquence.

Capteur de température : Ce capteur mesure la température de l'environnement. Il est utilisé dans diverses applications telles que la surveillance de la température, la régulation thermique et le contrôle climatique.

En classe, nous avons utilisé le capteur de température et d'humidité DHT11, qui est un modèle très répandu et bon marché.

Capteur de pression : Les capteurs de pression mesurent la force appliquée sur une surface. Ils sont utilisés dans les applications de contrôle de la force, de détection de contact et de mesure de la pression atmosphérique.

Capteur de mouvement : Ces capteurs détectent les mouvements, les inclinaisons ou les rotations du robot. Ils sont couramment utilisés pour la détection de mouvement, la stabilisation et le contrôle de l'orientation.

Capteur de son : Les capteurs de son détectent les variations de pression acoustique dans l'environnement. Ils peuvent être utilisés pour détecter des sons spécifiques, effectuer une reconnaissance vocale ou analyser l'acoustique de l'environnement.

Capteur de couleur : Ces capteurs permettent de détecter les couleurs des objets. Ils sont utilisés dans les applications de tri, d'identification d'objets et de suivi de lignes basé sur la couleur.

Capteur de proximité : Les capteurs de proximité détectent la présence d'objets à proximité sans contact physique direct. Ils sont utilisés pour la détection d'obstacles, la détection de présence et l'activation de fonctions basées sur la proximité.

Capteur de gaz : Ces capteurs détectent la présence et la concentration de gaz dans l'environnement. Ils sont utilisés dans les applications de surveillance de la qualité de l'air, de détection de fuites de gaz, etc.

Oui, il existe d'autres capteurs utilisés en robotique. Voici quelques exemples supplémentaires :

Capteur d'inclinaison : Ces capteurs mesurent l'inclinaison ou l'orientation du robot par rapport à la gravité. Ils sont utilisés pour la stabilisation, la détection de l'attitude et le contrôle de l'équilibre.

Capteur de force : Les capteurs de force mesurent la force appliquée sur une surface ou un objet. Ils sont utilisés dans les applications de manipulation d'objets, de contrôle de la pression et de rétroaction haptique.

Capteur d'humidité : Ces capteurs mesurent le niveau d'humidité dans l'air ou dans un matériau. Ils sont utilisés dans les applications de contrôle de l'humidité, d'irrigation intelligente et de surveillance environnementale.

Capteur de gaz toxique : Ces capteurs spécifiques détectent la présence de gaz toxiques ou dangereux dans l'environnement. Ils sont utilisés dans les applications de sécurité industrielle, de surveillance environnementale et de détection de fuites.

Capteur de mouvement oculaire : Ces capteurs utilisent la technologie de suivi oculaire pour détecter et mesurer les mouvements des yeux. Ils sont utilisés dans les interfaces homme-machine avancées et les applications de réalité virtuelle.

Capteur d'odeur : Les capteurs d'odeur, également appelés capteurs de gaz odorants, peuvent détecter et analyser différentes odeurs. Ils sont utilisés dans des applications telles que l'analyse de la qualité de l'air, la détection d'odeurs spécifiques et le contrôle des émissions odorantes.

Capteur de vitesse : Ces capteurs mesurent la vitesse de déplacement du robot. Ils sont utilisés pour le contrôle de la vitesse, la détection de mouvement rapide et le suivi de la trajectoire.

Capteur de vision : Les capteurs de vision, tels que les caméras haute résolution, sont utilisés pour capturer des images et des vidéos de l'environnement du robot. Ils sont essentiels pour la perception visuelle, la détection d'objets, la reconnaissance faciale, etc.

Chaque capteur a des caractéristiques spécifiques et est utilisé pour des applications particulières en fonction des besoins du robot et de son environnement. L'utilisation de différents capteurs permet d'obtenir une perception complète et précise du monde extérieur, ce qui est essentiel pour des opérations robotiques avancées. Ces capteurs sont largement utilisés en robotique et offrent une multitude de possibilités pour la perception et l'interaction du robot avec son environnement. En fonction des besoins spécifiques de chaque projet, il est possible d'utiliser différents capteurs pour obtenir les informations nécessaires et permettre au robot d'interagir de manière intelligente avec le monde qui l'entoure.

BME280: pression atmosphérique et altitude

> *Dans cette activité, tu vas découvrir comment mesurer la pression atmosphérique avec le capteur BME280 et utiliser ces données pour calculer l'altitude, une compétence essentielle pour comprendre et analyser la descente d'un CanSat lors de son vol.*

🎯 Objectif du cours

Dans ce cours, tu vas apprendre à :

- lire la **pression atmosphérique** avec le capteur **BME280** via `raw_values`,
- comprendre la relation entre pression et altitude,
- calculer l'**altitude** en utilisant une formule standard.

Ces compétences te serviront pour la **mission primaire** du CanSat.

Comprendre la pression atmosphérique

🌍 Qu'est-ce que la pression atmosphérique ?

La pression atmosphérique, c'est le **poids de l'air** au-dessus de toi.

- Au niveau de la mer : environ **1013 hPa**
- En altitude, la pression **diminue** car la colonne d'air est plus petite.

👉 Au CanSat, la pression va donc diminuer beaucoup *pendant la montée* puis augmenter pendant la descente.

Le capteur BME280

Mesures disponibles

- Température (°C)
- Pression (hPa)
- Humidité (%)

Communication

Il utilise le **bus I2C**, qui ne demande que deux fils : SDA et SCL.

▸ Exemple de branchement (Pico → BME280)

BME280	PICO
VCC	3.3V
GND	GND
SCL	GP9
SDA	GP8

Lire les données *numériques* avec `raw_values`

Le module BME280 utilisé dans le CanSat propose :

```
temp, pressure, humidity = bme.raw_values
```

Il retourne :

- la température en **°C**,
- la pression en **hPa**,
- l'humidité en **%**.

👉 Pas besoin de convertir des chaînes : c'est prêt pour les calculs.

▸ Exemple de code

```
from machine import I2C, Pin
from bme280 import BME280
import time

i2c = I2C(0, scl=Pin(9), sda=Pin(8))
bme = BME280(i2c=i2c)

while True:
    temp, pressure, humidity = bme.raw_values
    print("Température (°C):", temp)
    print("Pression (hPa):", pressure)
    print("Humidité (%):", humidity)
    print("-------------------------")
    time.sleep(1)
```

🔍 Comprendre le calcul de l'altitude à partir de la pression

Pour estimer l'altitude à partir de la pression mesurée par le BME280, on utilise une formule barométrique simplifiée. Mais avant de l'utiliser, il faut comprendre un point essentiel : **la pression au niveau de la mer n'est pas une constante !**

🌊 Qu'est-ce que la pression MSL ?

MSL signifie **Mean Sea Level pressure** = pression ramenée au niveau de la mer. C'est la valeur que les stations météo publient dans les bulletins.

IMPORTANT :

- La valeur "standard" est **1013.25 hPa**, mais elle varie **chaque jour**, selon la météo.
- Lors d'un **anticyclone**, la pression peut monter jusqu'à **1035 hPa**.
- Lors d'une **dépression**, elle peut descendre vers **980 hPa** (ou moins).

👉 Si tu utilises la valeur **standard 1013.25 hPa** alors que la pression réelle du jour est par exemple 1002 hPa, ton altitude sera fausse de plusieurs **dizaines de mètres**.

📡 Où trouver la pression MSL du jour ?

Voici des sources fiables (utiliser la station météo la plus proche) :

- **IRM – Belgique** https://www.meteo.be Chercher "pression atmosphérique" → valeur en hPa.
- **OpenWeatherMap** (API ou application) https://openweathermap.org
- **Météociel** https://www.meteociel.fr/observations-meteo/pression.php
- **Applications smartphone** : Météo & Radar, Windy, etc.

👉 Dans un projet CanSat, on choisit la station météo la plus proche du lieu du **lancement**.

🧮 La formule utilisée

La relation pression ↔ altitude vient de la physique de l'atmosphère :

$$\text{altitude} = 44330 \times \left(1 - \left(\frac{P}{P_0}\right)^{\frac{1}{5.255}}\right)$$

- **P** = pression mesurée par le BME280 (en hPa)
- **P_0** = pression MSL (pression au niveau de la mer, en hPa)
- **altitude** en mètres

POURQUOI ÇA FONCTIONNE ?

L'air est comprimé par son propre poids. Plus on monte :

- la densité de l'air diminue,

- la pression diminue.

La formule ci-dessus exprime précisément cette diminution.

▸ 📘 Exemple concret

Imaginons que tu mesures :

- Pression du capteur : **955 hPa**
- Pression MSL du jour (IRM) : **1002 hPa**

$$altitude = 44330 \times \left(1 - (955/1002)^{1/5.255}\right)$$

→ Résultat ≈ **423 m**

Si tu avais utilisé 1013.25 hPa par défaut :

→ ≈ **500 m** (erreur de +77 m)

▸ 🌡️ Pourquoi la pression MSL change ?

Parce que la pression dépend de la masse d'air au-dessus de la région :

- Air chaud → plus léger → **basse pression**
- Air froid → plus lourd → **haute pression**
- Mouvements atmosphériques → variations permanentes

Donc, **tu dois calibrer ton capteur** en mettant à jour le `P0` **chaque jour** ou avant chaque vol.

🌍 Faut-il utiliser la pression MSL du lieu ?

Oui, mais il faut bien comprendre ce que cela signifie.

▸ 🔹 1) La station météo te donne *la pression ramenée au niveau de la mer*

Quand tu vas sur IRM, OpenWeather, Windy, etc., la pression affichée n'est **pas** la pression réellement mesurée à cet endroit. Les stations appliquent un calcul pour "ramener" la pression mesurée au **niveau de la mer (MSL)** afin de pouvoir comparer deux stations situées à des altitudes différentes.

Exemple :

- Pression mesurée à Bastogne (500 m) ≈ 955 hPa
- Pression ramenée au niveau de la mer (MSL) ≈ 1010 hPa

👉 C'est cette **pression MSL** (1010 hPa) qu'il faut utiliser dans la formule d'altitude.

Pourquoi ? Parce que la formule barométrique **suppose que P_0 correspond à la pression au niveau de la mer**, pas à la pression "au sol" à 350 ou 500 m d'altitude.

🔹 Alors quelle valeur utiliser dans le CanSat ?

✓ Option 1 (recommandée par l'ESA & IRM)

➡️ **P_0 = pression MSL du jour fournie par la station météo la plus proche du site de lancement.**

Sources :

- IRM Belgique : https://www.meteo.be
- Windy : https://www.windy.com
- OpenWeatherMap
- Stations météo universitaires locales

C'est la méthode "officielle".

🔹 Option 2 : calibration locale pour plus de précision

Les conditions météo changent en permanence. Même la valeur MSL donnée par IRM peut entraîner une erreur (surtout si la station est loin).

👉 Solution : **calibrer P_0 sur place**.

ÉTAPES :

1. Tu regardes la **vraie altitude** du site (Google Maps, géoportail, panneau local).
2. Tu mesures la pression du BME280 sur place : `P_mesure`.
3. Tu ajustes P_0 pour que la formule te redonne cette altitude.

EXEMPLE

Lieu du lancement : altitude réelle = **510 m** Pression mesurée BME280 = **955.4 hPa**

Tu cherches la valeur P_0 qui donne 510 m :

[510 = 44330 \times \left(1 - (955.4/P_0)^{1/5.255}\right)]

Résultat → **P_0 ≈ 1010.6 hPa**

👉 Tu mets `P0 = 1010.6` dans ton code.

C'est la méthode la **plus précise** possible.

🔹 5) Mini-algorithme conseillé pour le CanSat

1. Récupérer la pression MSL du jour (IRM).
2. Mesurer la pression réelle au sol avec le BME280.
3. Calculer l'altitude affichée → comparer avec l'altitude connue.
4. **Ajuster P_0 automatiquement** dans le code pour corriger l'erreur.
5. Laisser P_0 constant pendant tout le vol.

Calculer l’altitude à partir de la pression

On utilise la formule barométrique simplifiée :

$$\text{altitude} = 44330 \times \left(1 - \left(\frac{P}{P_0}\right)^{\frac{1}{5.255}}\right)$$

- **P** : pression mesurée (hPa)
- **P_0** : pression au niveau de la mer (par défaut 1013.25 hPa)

Code "complet" avec altitude

```
from machine import I2C, Pin
from bme280 import BME280
import time

i2c = I2C(0, scl=Pin(9), sda=Pin(8))
bme = BME280(i2c=i2c)

P0 = 1013.25   # pression standard au niveau de la mer

while True:
    temp, P, hum = bme.raw_values

    # ICI: CALCULE L'ALTITUDE AVEC LA FORMULE SIMPLIFIEE

    print(f"Température: {temp:.2f} °C")
    print(f"Pression: {P:.2f} hPa")
    print(f"Altitude estimée: {altitude:.2f} m")
    print("------------------------------")

    time.sleep(1)
```

Exemple d'interprétation

PRESSION MESURÉE (HPA)	ALTITUDE ESTIMÉE
1013	0 m
1000	~110 m
950	~520 m
900	~980 m
800	~1900 m

👉 Pendant le vol du CanSat, tu vas observer une **chute rapide de pression**, ce qui te permet de **recalculer l'altitude en temps réel**.

À retenir

- Le BME280 renvoie des **valeurs numériques** grâce à `raw_values`.
- La pression diminue quand l'altitude augmente.
- On peut calculer l'altitude **en direct** pendant le vol.
- Ces données sont la base de la **mission primaire** du CanSat.

💡 Exercice

1. Lire les valeurs du capteur avec `raw_values`.
2. Afficher température + pression + altitude chaque seconde.
3. Monter un étage avec le capteur et observer comment l'altitude change.
4. Tester à l'extérieur : comparer avec l'altitude donnée par Google Maps.

DHT11: Simulation du Capteur

> *Tu vas apprendre à utiliser un **objet Python** qui simule un capteur de température et d'humidité (le DHT11). Cela te permettra d'écrire du code **exactement comme si tu avais un vrai capteur branché** sur un Raspberry Pi Pico.*

Utiliser la classe `DHT11` en Python

Tu dois d'abord télécharher le fichier Python ci-joint. Ensuite, crée un nouveau fichier python dans Thonny que tu appelles "simulation_dht11.py" et sauvegarde-le dans ton dossier "Téléchargements".

Étape 1 : importer le module

Quand on veut utiliser une classe qui se trouve dans un autre fichier Python, on doit **l'importer**.

👉 Imaginons que ton fichier avec la classe `DHT11` s'appelle `dht11.py`. Alors, tu dois écrire en haut de ton programme :

```
from dht11 import DHT11
```

Étape 2 : créer un objet capteur

Maintenant, tu peux **créer un capteur simulé**. Il faut lui donner le numéro de la broche (pin) sur laquelle il est censé être branché :

```
capteur = DHT11(pin=15)
```

Même si la simulation n'utilise pas vraiment la broche, on garde ce paramètre pour ressembler au vrai capteur.

Étape 3 : demander une mesure

Avant de lire la température ou l'humidité, tu dois **lancer une mesure** :

```
capteur.measure()
```

C'est comme si tu envoyais l'ordre au vrai capteur de "prendre une photo" de l'air ambiant.

Étape 4 : lire les valeurs

Une fois la mesure faite, tu peux demander :

```
print("Température :", capteur.temperature(), "°C")
print("Humidité :", capteur.humidity(), "%")
```

Chaque appel va te donner la **dernière valeur mesurée**.

Exemple complet

```
from dht11 import DHT11

# Création d'un capteur simulé
capteur = DHT11(pin=15)

# Faire 3 mesures
for i in range(3):
    capteur.measure()  # on lance une nouvelle mesure
    print("Température :", capteur.temperature(), "°C")
    print("Humidité :", capteur.humidity(), "%")
    print("---")
```

À retenir

1. On **importe** la classe depuis son fichier.
2. On **crée un objet** `DHT11`.
3. On appelle `measure()` pour lancer une mesure.
4. On récupère les valeurs avec `temperature()` et `humidity()`.

👉 Quand tu passeras au vrai Raspberry Pi Pico, tu n'auras presque rien à changer : seule la ligne `from dht11 import DHT11` sera remplacée par l'import du vrai module.

Veux-tu que je t'écrive aussi la **version avec le vrai module `dht` de MicroPython** pour montrer le passage simulation → réel ?

DHT11: Capteur de température et d'humidité

> *Cet article présente le DHT11, un capteur de température et d'humidité très répandu et facile à utiliser.*

- Le capteur DHT11 est un capteur d'humidité et de température numérique économique et couramment utilisé.
- Il fournit des lectures de température et d'humidité relativement précises à partir d'une seule broche de données.

Caractéristiques

Le capteur DHT11 est un capteur de température et d'humidité très couramment utilisé dans de nombreuses applications. Voici quelques-unes de ses caractéristiques principales :

1. **Mesure de la température et de l'humidité** : Le capteur DHT11 permet de mesurer à la fois la température et l'humidité relative de l'environnement dans lequel il est placé.
2. **Précision** : Le capteur DHT11 offre une **précision de ±2°C** pour la mesure de la température et de **±5%** pour la mesure de l'humidité relative.
3. **Plage de mesure** : La plage de mesure de la température du capteur DHT11 est généralement de **0°C à 50°C**, tandis que la plage de mesure de l'humidité relative est de **20% à 90%**.
4. **Alimentation** : Le capteur DHT11 fonctionne avec une tension d'alimentation de **3,3V à 5V**, ce qui le rend compatible avec de nombreux microcontrôleurs et cartes de développement.
5. **Sortie numérique** : Le capteur DHT11 utilise une **sortie numérique** pour transmettre les données de température et d'humidité. Il envoie les informations sous forme de signal série numérique.
6. **Temps de réponse** : Le capteur DHT11 a un **temps de réponse relativement lent** par rapport à d'autres capteurs de température et d'humidité. Il peut prendre plusieurs secondes pour effectuer une mesure et transmettre les données.
7. **Connectivité** : Le capteur DHT11 utilise un protocole de communication à **un seul fil**, ce qui le rend **facile à intégrer** dans des projets électroniques et des systèmes embarqués.

Il convient de noter que le capteur DHT11 est un **capteur d'entrée de gamme** largement utilisé pour des applications simples de mesure de température et d'humidité. Si une précision plus élevée ou une plus grande plage de mesure est requise, il existe d'autres capteurs plus avancés disponibles sur le marché (tels que le DHT22 ou le BME280)

DHT22: alternative plus précise

Parmi les modèles plus "haut de gamme", on citera par exemple le **DHT22**. Également connu sous le nom d'AM2302, il offre une **meilleure précision** et une **plus large plage de mesure** par rapport au DHT11. Il peut mesurer la température avec une précision de **±0,5°C** et l'humidité avec une précision de ±2%. De plus, le DHT22 a une plage de mesure de température de **-40°C à 80°C** et une plage d'humidité de **0% à 100%**.

Configuration matérielle

- Connectez la broche de données du capteur DHT11 au GPIO du Raspberry Pi Pico (par exemple, GPIO2).
- Alimentez le capteur DHT11 en connectant le fil rouge au 3,3 V du Pico et le fil noir à la masse (GND).

Installation de la bibliothèque DHT11

- Téléchargez la bibliothèque dht11 pour MicroPython à partir du dépôt GitHub officiel (https://github.com/mcauser/micropython-dht11).
- Copiez le fichier dht11.py sur votre Raspberry Pi Pico.

Exemple de code pour la lecture des données du capteur DHT11

```
import machine
import dht
import time

# Initialize the DHT sensor.
capteur = dht.DHT11(machine.Pin(2))

while True:
    capteur.measure()  # Trigger a measurement.
    print("Température:", capteur.temperature())
    print("Humidité:", capteur.humidity())
    time.sleep(2)
```

Explication du code

Bien sûr, voici une explication pour ce code :

- `import machine, import dht, import time` : Ces trois instructions importent les modules nécessaires. `machine` est un module qui permet d'interagir avec le matériel de l'appareil, `dht` est un module pour travailler avec le capteur de température et d'humidité DHT11, et `time` est un module pour manipuler le temps (par exemple, pour créer des délais).
- `capteur = dht.DHT11(machine.Pin(2))` : Cette ligne crée une instance de l'objet DHT11 appelée `capteur`. Le DHT11 est connecté au pin 2 de la machine (en supposant que vous utilisiez un ESP8266 ou un ESP32, où le pin 2 est généralement disponible).
- `while True:` : Ceci commence une boucle infinie. Le code à l'intérieur de cette boucle continuera à s'exécuter indéfiniment.
- `capteur.measure()` : Cette ligne indique au capteur DHT11 de commencer une mesure. Le capteur effectue une lecture de la température et de l'humidité de l'air.
- `print("Température:", capteur.temperature())` : Cette ligne affiche la température mesurée par le capteur. La méthode `temperature()` renvoie la température mesurée lors du dernier appel à `capteur.measure()`.
- `print("Humidité:", capteur.humidity())` : De même, cette ligne affiche l'humidité mesurée par le capteur. La méthode `humidity()` renvoie l'humidité mesurée lors du dernier appel à `capteur.measure()`.
- `time.sleep(2)` : Pour finir, cette ligne fait une pause dans l'exécution du programme pendant 2 secondes. C'est important car le capteur DHT11 a besoin d'un certain temps entre les lectures pour garantir des mesures précises. En général, il est recommandé d'attendre au moins 2 secondes entre chaque lecture.

Donc, en résumé, ce code effectue une lecture continue de la température et de l'humidité avec un capteur DHT11, et imprime ces valeurs toutes les 2 secondes. Ressources supplémentaires

Infos supplémentaires

- Tutoriel vidéo sur l'utilisation du capteur DHT11 avec MicroPython et Raspberry Pi Pico : https://youtu.be/hZ9zdvTRnTw
- Documentation de la bibliothèque DHT11 pour MicroPython : https://github.com/mcauser/micropython-dht11

Ces instructions vous permettront de lire les données de température et d'humidité à partir du capteur DHT11 à l'aide de MicroPython sur votre Raspberry Pi Pico.

HC-SR04: Capteur de distance à ultra-sons

> *Cet article présente HC-SR04, un capteur de distance à ultra-sons très répandu et facile à utiliser.*

Introduction au capteur de distance HC-SR04

- Le capteur de distance HC-SR04 est un capteur ultrasonique populaire utilisé pour mesurer la distance entre le capteur et un objet.
- Il utilise des ondes sonores ultrasoniques pour déterminer la distance, en émettant un signal et en mesurant le temps nécessaire pour que l'écho revienne.

Configuration matérielle

- Connectez la broche de déclenchement (Trig) du capteur HC-SR04 à un GPIO du Raspberry Pi Pico (par exemple, GPIO2).
- Connectez la broche d'écho (Echo) du capteur HC-SR04 à un autre GPIO du Pico (par exemple, GPIO3).
- Alimentez le capteur HC-SR04 en connectant le fil rouge au 5V du Pico et le fil noir à la masse (GND).

Exemple de code pour la mesure de distance avec le capteur HC-SR04

```
import machine
import utime

# Définir les broches GPIO pour le déclenchement et l'écho
trigger_pin = machine.Pin(2, machine.Pin.OUT)
echo_pin = machine.Pin(3, machine.Pin.IN)

# Fonction pour mesurer la distance
def mesure_distance():
    # Générer une impulsion de déclenchement
    trigger_pin.low()
    utime.sleep_us(2)
    trigger_pin.high()
    utime.sleep_us(10)
    trigger_pin.low()

    # Attendre le signal d'écho
    while echo_pin.value() == 0:
        signal_depart = utime.ticks_us()

    while echo_pin.value() == 1:
        signal_arrivee = utime.ticks_us()

    # Calculer la durée de l'écho
    duree_echo = signal_arrivee - signal_depart

    # Calculer la distance en fonction de la durée de l'écho
    distance = (duree_echo * 0.0343) / 2

    return distance

while True:
    # Mesurer la distance avec le capteur HC-SR04
    distance = mesure_distance()

    # Afficher la distance mesurée
    print("Distance : {:.2f} cm".format(distance))

    # Attendre avant de refaire une mesure
    utime.sleep(1)
```

Explication du code

- Importez les modules nécessaires, notamment machine pour accéder aux fonctionnalités du Raspberry Pi Pico et utime pour la gestion du temps.
- Définissez les broches GPIO pour le déclenchement (Trig) et l'écho (Echo) du capteur HC-SR04.
- Définissez une fonction mesure_distance() pour effectuer la mesure de distance.
- Dans la fonction mesure_distance(), générez une impulsion de déclenchement en mettant la broche de déclenchement à l'état bas puis haut puis à nouveau bas.
- Attendez le signal d'écho en mesurant le temps auquel l'écho commence et se termine.

Calculez la durée de l'écho en soustrayant le temps de départ du temps d'arrivée. 7. Calculez la distance en utilisant la formule de conversion de la durée de l'écho en distance (en supposant une vitesse du son de 343 m/s).

- Dans la boucle principale, appelez la fonction mesure_distance() pour mesurer la distance avec le capteur HC-SR04.
- Affichez la distance mesurée et attendez avant de refaire une mesure.

Utilisation du capteur de distance HC-SR04 avec un module open source

Installation du module open source

- Téléchargez le module open source hc_sr04 pour MicroPython à partir du dépôt GitHub officiel (https://github.com/rsc1975/micropython-hcsr04).
- Copiez les fichiers hc_sr04.py et pulseio.py sur votre Raspberry Pi Pico.

Exemple de code pour la mesure de distance avec le capteur HC-SR04

```
import machine
import utime
from hc_sr04 import HCSR04

# Définir les broches GPIO pour le déclenchement et l'écho
trigger_pin = machine.Pin(2)
echo_pin = machine.Pin(3)

# Créer une instance du capteur HC-SR04
capteur_distance = HCSR04(trigger_pin, echo_pin)

while True:
    # Mesurer la distance avec le capteur HC-SR04
    distance = capteur_distance.distance_cm()

    # Afficher la distance mesurée
    print("Distance : {:.2f} cm".format(distance))

    # Attendre avant de refaire une mesure
    utime.sleep(1)
```

Explication du code

- Importez les modules nécessaires, notamment machine pour accéder aux fonctionnalités du Raspberry Pi Pico et utime pour la gestion du temps.
- Importez la classe HCSR04 du module hc_sr04 pour utiliser le capteur HC-SR04.
- Définissez les broches GPIO pour le déclenchement (Trig) et l'écho (Echo) du capteur HC-SR04.
- Créez une instance du capteur HC-SR04 en passant les broches GPIO comme arguments.
- Dans la boucle principale, appelez la méthode distance_cm() de l'instance du capteur pour mesurer la distance.
- Affichez la distance mesurée et attendez avant de refaire une mesure.

Ressources supplémentaires

- Tutoriel vidéo sur l'utilisation du capteur de distance HC-SR04 avec MicroPython et Raspberry Pi Pico : https://youtu.be/YAF3o3SsCmk
- Dépôt GitHub du module open source hc_sr04 pour MicroPython : https://github.com/rsc1975/micropython-hcsr04

Ces instructions vous permettront de mesurer la distance à l'aide du capteur HC-SR04 en utilisant un module open source avec MicroPython sur votre Raspberry Pi Pico.

Ressources supplémentaires

- Tutoriel vidéo sur l'utilisation du capteur de distance HC-SR04 avec MicroPython et Raspberry Pi Pico : https://youtu.be/YAF3o3SsCmk

- Bibliothèque hcsr04 pour MicroPython sur GitHub : https://github.com/rsc1975/micropython-hcsr04

Ces instructions vous permettront de mesurer la distance à l'aide du capteur HC-SR04 en utilisant MicroPython sur votre Raspberry Pi Pico.

Les micro-contrôleurs

> *Un microcontrôleur est une unité de commande intégrée utilisée pour **contrôler le fonctionnement d'un robot**. Il s'agit d'un petit ordinateur sur une seule puce (SoC) qui combine un **microprocesseur**, de la **mémoire** et des **interfaces d'entrée/sortie**.*

En tant qu'unité de commande, le microcontrôleur **exécute les programmes et les instructions** qui définissent le comportement du robot. Il reçoit des informations provenant des capteurs du robot, traite ces données et envoie des commandes aux actionneurs pour effectuer des actions spécifiques.

Les microcontrôleurs sont souvent utilisés dans les robots en raison de leur **taille compacte**, de leur **faible consommation d'énergie** et de leur capacité à exécuter des tâches en **temps réel**. Ils offrent une solution intégrée pour contrôler les mouvements, les interactions avec l'environnement et les communications du robot.

Les microcontrôleurs peuvent être programmés à l'aide de différents langages, tels que **C/C++**, Python, **Micro-Python**, ou des langages de programmation visuels basés sur des blocs. Les programmeurs écrivent des codes qui spécifient le comportement du robot, y compris la prise de décision, les mouvements, la détection d'obstacles et les réponses aux événements.

En résumé, un microcontrôleur est l'**unité de commande centrale** d'un robot, responsable de l'exécution des programmes, de la gestion des capteurs et des actionneurs, et de la coordination des opérations pour permettre au robot d'accomplir des tâches spécifiques.

Micro:bit :

- Le micro:bit est un microcontrôleur largement utilisé pour l'apprentissage de la robotique, en particulier pour les débutants.
- Il est petit, abordable et **dispose de nombreux capteurs intégrés **tels que l'accéléromètre, le magnétomètre et les boutons.
- Le micro:bit utilise une programmation par blocs conviviale, ce qui le rend facile à apprendre pour les débutants.
- Il est souvent utilisé pour créer des projets robotiques simples tels que des robots suiveurs de ligne ou des robots éviteurs d'obstacles.

En savoir plus (notes de cours)

ESP8266 :

- L'ESP8266 est un microcontrôleur WiFi populaire utilisé dans la robotique et l'Internet des objets (IoT).
- Il dispose d'une connectivité WiFi intégrée, ce qui permet aux robots de se connecter à des réseaux sans fil et d'interagir avec d'autres appareils.
- L'ESP8266 est compatible avec plusieurs langages de programmation, tels que Arduino IDE et MicroPython.
- Il est souvent utilisé pour des projets de robotique avancés nécessitant une communication sans fil, tels que des robots contrôlés par smartphone ou des robots de surveillance.

Raspberry Pi Pico :

- Le Raspberry Pi Pico est un microcontrôleur développé par la Fondation Raspberry Pi.
- Il est basé sur le microcontrôleur RP2040 et dispose de nombreuses fonctionnalités, telles que des broches d'E/S polyvalentes et une programmation en MicroPython et C/C++.
- Le Raspberry Pi Pico est abordable, polyvalent et peut être utilisé pour des projets robotiques de taille moyenne à avancée.
- Il est souvent utilisé pour créer des robots autonomes, des robots avec des capteurs avancés et des projets de robotique embarquée.

En savoir plus (notes de cours)

LEGO Mindstorms :

- LEGO Mindstorms est une **plateforme de robotique éducative** très populaire pour les enfants et les débutants.
- Elle utilise des briques LEGO spéciales, des moteurs et des capteurs pour construire des robots.
- Les robots LEGO Mindstorms peuvent être programmés à l'aide d'un logiciel convivial basé sur des blocs graphiques.
- Cette plateforme est idéale pour enseigner les bases de la robotique et de la programmation, en mettant l'accent sur la créativité et la construction physique.

Thymio :

- Thymio est un **robot éducatif** conçu pour l'apprentissage de la robotique et de la programmation.
- Il est doté de capteurs et d'actionneurs, tels que des capteurs de proximité, des détecteurs de lumière et des moteurs.
- Thymio peut être programmé à l'aide d'un logiciel visuel ou d'un langage de programmation textuel, ce qui permet d'explorer différents niveaux de complexité.
- Ce robot est largement utilisé dans les écoles pour enseigner la programmation, la logique et les concepts de base de la robotique.

Ces microcontrôleurs offrent une excellente opportunité d'apprentissage pour les étudiants dans le domaine de la robotique. Ils sont abordables, faciles à programmer et offrent des fonctionnalités adaptées à différents niveaux de compétence.

Le Raspberry Pi Pico

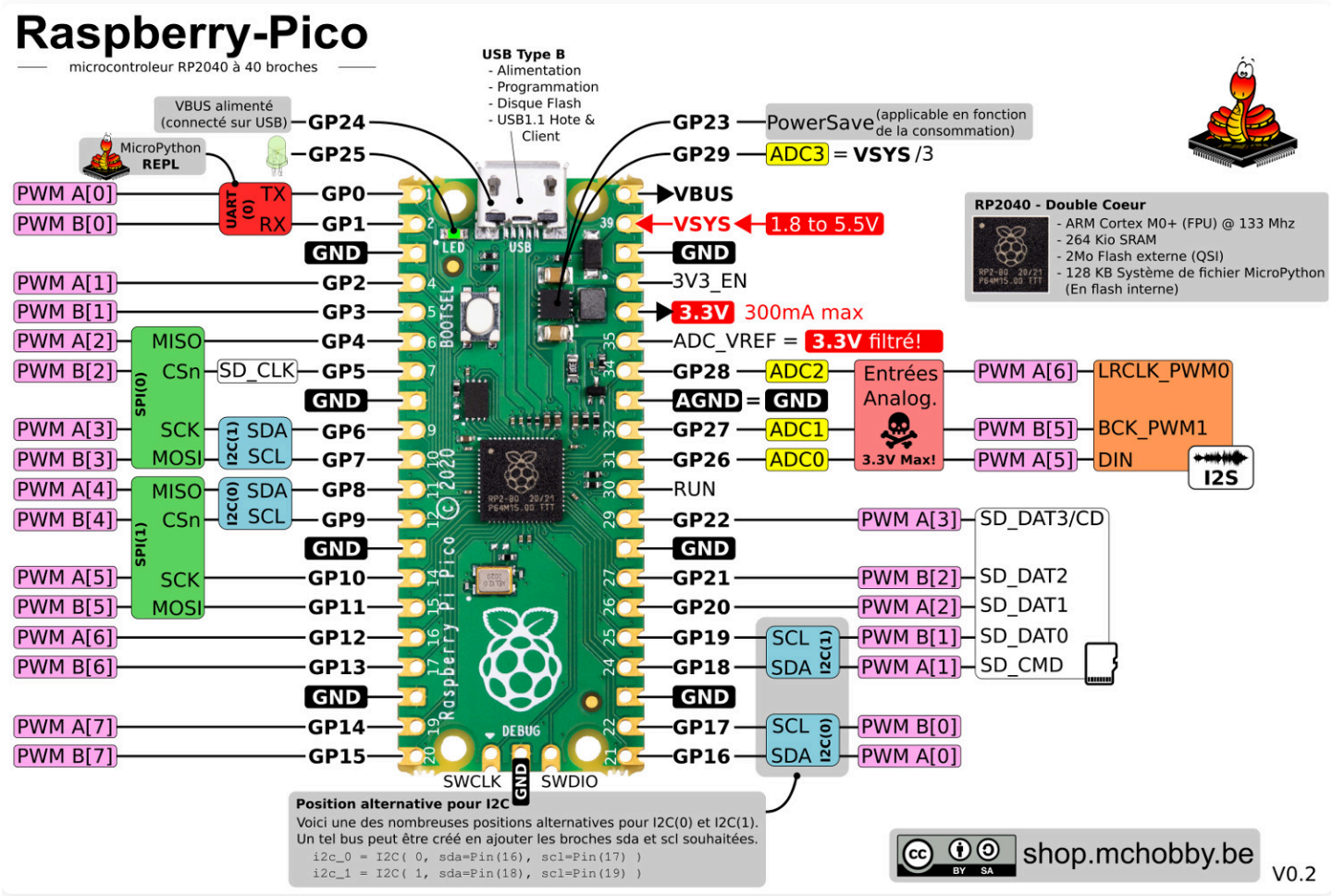
Les Broches du Raspberry Pi Pico en MicroPython

> *Sur cette page, nous allons explorer les différentes broches du Pico, leur numéro, leur fonction (analogique, PWM, numérique), et comment les utiliser en MicroPython.*

Le Raspberry Pi Pico dispose de plusieurs broches GPIO (General-Purpose Input/Output) qui peuvent être utilisées pour diverses tâches en MicroPython.

Broches GPIO du Raspberry Pi Pico

Le Raspberry Pi Pico possède 26 broches GPIO que vous pouvez utiliser pour interagir avec des composants électroniques et des capteurs. Voici une liste des broches GPIO disponibles sur le Pico :



Utilisation des Broches en MicroPython

Configuration d'une broche en mode numérique

Pour configurer une broche en mode numérique (entrée ou sortie), vous pouvez utiliser la bibliothèque `machine` en MicroPython. Par exemple, pour configurer la broche GP0 en tant que sortie, vous pouvez utiliser le code suivant :

```
import machine

# Configurer GP0 en sortie
broche_gp0 = machine.Pin(0, machine.Pin.OUT)
```

MicroPython offre plusieurs méthodes pour contrôler la tension de sortie :

```
from machine import Pin

Pin.on() et Pin.off()
Pin.high() et Pin.low()
Pin.value(valeur)
```

Les noms de fonctions sont explicites :

```
pin_led = Pin(16, mode=Pin.OUT)

# Impose une tension de 3.3V en sortie (état logique haut)
pin_led.on()
pin_led.high()
pin_led.value(1)

# Impose une tension de 0V en sortie (état logique bas)
pin_led.off()
pin_led.low()
pin_led.value(0)
```

Utilisation des broches en mode PWM

Le Raspberry Pi Pico prend en charge la modulation de largeur d'impulsion (PWM) sur certaines broches. Cela permet de contrôler la vitesse des moteurs, la luminosité des LED, etc. Voici comment configurer une broche en mode PWM :

```
import machine

# Configurer GP2 en mode PWM
broche_gp2 = machine.Pin(2)
pwm = machine.PWM(broche_gp2)
```

Lecture de valeurs analogiques

Pour lire des valeurs analogiques à partir des broches GPIO, vous pouvez utiliser l'ADC (Convertisseur Analogique-Numérique) intégré au Pico.

Note/Rappel: Le RPi Pico possède 4 broches analogiques: GP26->29

Voici comment configurer une broche pour lire une valeur analogique :

```
import machine

# Configurer GP26 pour la lecture analogique
broche_gp26 = machine.Pin(26)
adc = machine.ADC(broche_gp26)

# Lire la valeur analogique
valeur_analogique = adc.read_u16()
```

Le code suivant permet de lire la valeur d'un potentiomètre analogique:

```
from machine import ADC, Pin
from utime import sleep
pot = ADC(26)
conversion_factor = 3.3 / 65535

while True:
    raw = pot.read_u16()
    volts = raw * conversion_factor
    print('Raw: {} '.format(raw), 'Voltage {:.1f}V'.format(volts))
    sleep(1)
```

Et ce code permet de changer la fréquence de clignotement d'une LED en fonction de la valeur lue sur un potentiomètre:

```
from machine import ADC, Pin
from utime import sleep

pot = ADC(26)
led = Pin(25, Pin.OUT)

def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)

while True:
    raw = pot.read_u16()
    delay = map(raw, 0, 65535, 0, 1)
    print('Raw: {} '.format(raw), 'Delay {:.1f}s'.format(delay))
    led.value(1)
    sleep(delay)
    led.value(0)
    sleep(delay)
```

Conclusion

Les broches GPIO du Raspberry Pi Pico offrent de nombreuses possibilités pour interagir avec le monde physique en utilisant MicroPython. Que ce soit pour contrôler des LED, des moteurs, ou pour lire des capteurs analogiques, vous avez maintenant les connaissances de base pour commencer à créer vos propres projets avec le Pico. Explorez davantage et expérimentez avec les différentes broches pour donner vie à vos idées.

MicroPython sur RPi Pico avec Thonny Python

///

Prérequis

- Assurez-vous d'avoir un Raspberry Pi Pico fonctionnel et un câble USB pour le connecter à votre ordinateur.
- Téléchargez et installez l'IDE Thonny Python sur votre ordinateur à partir du site officiel de Thonny (https://thonny.org/).

Installation du firmware MicroPython

- Visitez le site officiel de MicroPython pour Raspberry Pi Pico (https://micropython.org/download/rp2-pico/) et téléchargez le dernier firmware MicroPython (.uf2) pour le Raspberry Pi Pico.

Connexion du Raspberry Pi Pico

- Connectez votre Raspberry Pi Pico à votre ordinateur à l'aide du câble USB. Assurez-vous que le commutateur "BOOTSEL" est enfoncé pour mettre le Pico en mode Bootloader.

Flashage du firmware MicroPython

- Ouvrez l'IDE Thonny Python sur votre ordinateur.
- Dans Thonny, cliquez sur le bouton "Sélectionner l'interpréteur" en haut à droite.
- Dans la fenêtre de sélection de l'interpréteur, choisissez "Raspberry Pi Pico (MicroPython)".
- Maintenant, dans Thonny, cliquez sur le bouton "Flasher" en haut.
- Dans la fenêtre de flashage, sélectionnez le firmware MicroPython (.uf2) que vous avez précédemment téléchargé.
- Cliquez sur "OK" pour flasher le firmware sur votre Raspberry Pi Pico.

Écriture et exécution du code MicroPython

- Une fois le firmware MicroPython installé, le Raspberry Pi Pico redémarrera automatiquement.
- Dans Thonny, vous verrez le Pico répertorié comme un périphérique connecté.
- Créez un nouveau fichier en cliquant sur "Nouveau" dans le menu Fichier.
- Écrivez votre code MicroPython dans le nouvel onglet du fichier.
- Pour exécuter le code, cliquez sur le bouton "Exécuter" ou appuyez sur F5.

Interaction avec le Raspberry Pi Pico

- Vous pouvez utiliser la console REPL (Read-Eval-Print Loop) pour interagir avec le Raspberry Pi Pico en temps réel.
- Dans Thonny, cliquez sur l'icône de console en bas de la fenêtre pour ouvrir la console REPL.
- Vous pouvez saisir des commandes directement dans la console pour interagir avec les broches GPIO, les capteurs ou les actionneurs connectés au Raspberry Pi Pico.

Débogage et surveillance

- Thonny Python offre également des fonctionnalités de débogage et de surveillance pour vous aider à identifier et corriger les erreurs dans votre code MicroPython.
- Vous pouvez placer des points d'arrêt, examiner les variables et suivre l'exécution pas à pas pour comprendre le comportement de votre code.

Ces étapes vous permettront de configurer et d'utiliser MicroPython sur votre Raspberry Pi Pico en utilisant l

Raspberry Pi Pico et les Ports GPIO

///

- Le Raspberry Pi Pico est un microcontrôleur à faible coût développé par la Fondation Raspberry Pi.
- Il est basé sur le microcontrôleur RP2040 et offre de nombreuses fonctionnalités pour les projets de robotique et d'électronique.

Les Ports GPIO (General-Purpose Input/Output)

- Le Raspberry Pi Pico dispose de **26 broches GPIO (0 à 25)** qui peuvent être configurées en tant qu'**entrées ou sorties** pour interagir avec d'autres composants électroniques.
- Les broches GPIO peuvent être utilisées pour **lire des signaux provenant de capteurs externes ou pour contrôler des actionneurs** tels que des moteurs ou des LED.
- Chaque broche GPIO peut être configurée individuellement et prend en charge différents modes de fonctionnement, tels que l'**entrée**, la **sortie**, la **PWM** (modulation de largeur d'impulsion), l'I2C, le SPI, etc.

Utilisation des Ports GPIO

- Configuration des broches : Avant d'utiliser une broche GPIO, il est nécessaire de la **configurer dans le mode de fonctionnement souhaité** en utilisant des bibliothèques de programmation appropriées, telles que RP2PIO ou RP2GPIO pour le Raspberry Pi Pico.
- Lecture des broches d'entrée : Les broches GPIO configurées en tant qu'entrées peuvent être utilisées pour lire des signaux provenant de capteurs externes. On peut les lire en utilisant des fonctions de lecture appropriées dans le langage de programmation utilisé, telles que input() en Python.
- Contrôle des broches de sortie : Les broches GPIO configurées en tant que sorties peuvent être utilisées pour contrôler des actionneurs, tels que des moteurs ou des LED. On peut les contrôler en utilisant des fonctions d'écriture appropriées dans le langage de programmation utilisé, telles que output() en Python.

Exemples d'utilisation des Ports GPIO

- Contrôle de LED : On peut connecter une LED à une broche GPIO configurée en tant que sortie et utiliser la fonction d'écriture pour allumer ou éteindre la LED.
- Lecture de bouton : On peut connecter un bouton à une broche GPIO configurée en tant qu'entrée et utiliser la fonction de lecture pour détecter si le bouton est enfoncé ou relâché.
- Contrôle de moteur : On peut utiliser les broches GPIO configurées en tant que sorties pour contrôler la vitesse et la direction d'un moteur en utilisant la modulation de largeur d'impulsion (PWM) pour créer des signaux analogiques simulés.

Modes Analogique, Digital et PWM

Mode Analogique

- Le mode analogique est utilisé pour représenter et manipuler des **signaux continus** qui peuvent prendre une gamme de valeurs différentes.
- Dans le contexte des microcontrôleurs, le mode analogique permet de **mesurer ou de générer des signaux analogiques**.
- Lorsqu'un microcontrôleur est configuré en mode analogique, il peut **lire des tensions** provenant de capteurs analogiques tels que des capteurs de température ou de lumière.
- Les microcontrôleurs peuvent également générer des signaux analogiques en utilisant des *convertisseurs numérique-analogique* (CNA) pour contrôler des dispositifs tels que des amplificateurs ou des moteurs.

Mode Digital

- Le mode digital est utilisé pour représenter des signaux qui ne peuvent prendre que **deux valeurs distinctes : 0 et 1 (ou HIGH et LOW)**.
- Les microcontrôleurs sont capables de traiter des signaux digitaux, ce qui leur permet de communiquer avec d'autres composants électroniques tels que des capteurs, des actionneurs ou d'autres microcontrôleurs.
- Les broches GPIO configurées en tant que sorties digitales peuvent envoyer des signaux à d'autres composants, tandis que les broches configurées en tant qu'entrées digitales peuvent recevoir des signaux provenant d'autres sources.

Modulation de Largeur d'Impulsion (PWM)

- La modulation de largeur d'impulsion (PWM) est une technique utilisée pour **simuler des signaux analogiques en utilisant des signaux digitaux**.
- Elle est couramment utilisée pour contrôler la vitesse ou l'intensité de dispositifs tels que des moteurs, des LED ou des servomoteurs.
- Dans un signal PWM, **la largeur de l'impulsion varie dans le temps**, tandis que la période de l'impulsion reste constante.

- En ajustant le rapport cyclique (rapport entre la durée de l'impulsion et la période), on peut contrôler la moyenne pondérée du signal, ce qui permet de varier la vitesse ou l'intensité du dispositif contrôlé.

Utilisation dans la Robotique

- En robotique, les **signaux analogiques** sont souvent utilisés **pour mesurer des grandeurs** telles que la **température**, la **luminosité** ou la **distance**.
- Les signaux digitaux sont utilisés pour la communication entre composants électroniques et la **lecture d'états** de capteurs, tels que les **boutons** ou les **interrupteurs**.
- La modulation de largeur d'impulsion (**PWM**) est utilisée pour **contrôler précisément la vitesse des moteurs** ou la **luminosité des LED**, ce qui est essentiel dans de nombreux projets robotiques.

Il est important de comprendre ces différents modes et techniques, car ils permettent aux microcontrôleurs de s'adapter à différentes situations et de contrôler divers dispositifs dans les projets robotiques.

Ressources supplémentaires

- Documentation officielle du Raspberry Pi Pico : https://www.raspberrypi.org/documentation/pico/
- Tutoriels et exemples de projets : Le site officiel du Raspberry Pi Pico propose une variété de tutoriels et de projets pour vous aider à démarrer avec les ports GPIO et d'autres fonctionnalités du microcontrôleur.

Utilisation du Raspberry Pi Pico avec MicroPython

> *Dans cet article de cours, nous allons explorer comment utiliser le Raspberry Pi Pico avec MicroPython et vous montrer des exemples de programmes pour lire et écrire des valeurs sur les broches GPIO (General-Purpose Input/Output).*

Le Raspberry Pi Pico est un microcontrôleur puissant et abordable qui peut être programmé avec MicroPython, un langage de programmation convivial pour les débutants.

Prérequis

Avant de commencer, assurez-vous d'avoir les éléments suivants à votre disposition :

- Un Raspberry Pi Pico
- Un ordinateur avec MicroPython installé (vous pouvez télécharger MicroPython sur le site officiel)
- Un câble micro USB pour la connexion entre l'ordinateur et le Raspberry Pi Pico

Installation de MicroPython

Ceci est une procédure d'installation manuelle que nous n'avons pas vue en classe. Je la laisse pour que vous sachiez que ça existe, mais il ne faut pas la connaître pour le cours.

1. Téléchargez la dernière version de MicroPython depuis le site officiel (https://micropython.org/).
2. Connectez votre Raspberry Pi Pico à votre ordinateur à l'aide du câble micro USB. Le Pico devrait être reconnu comme un périphérique de stockage USB.
3. Copiez le fichier MicroPython que vous avez téléchargé sur le Pico. Il doit avoir une extension ".uf2".
4. Déconnectez le Pico de votre ordinateur, puis reconnectez-le. Vous verrez un nouveau périphérique de stockage appelé "RPI-RP2".
5. Supprimez le fichier "CURRENT.UF2" s'il existe et renommez votre fichier MicroPython téléchargé en "CURRENT.UF2". Cela permettra au Pico de démarrer automatiquement avec MicroPython.
6. Déconnectez à nouveau le Pico de l'ordinateur, puis reconnectez-le. Il devrait maintenant exécuter MicroPython.

Programme de base

Commençons par un exemple simple de programme en MicroPython pour allumer et éteindre une LED connectée à une broche GPIO du Raspberry Pi Pico.

```
import machine
import utime

# Définir la broche GPIO à utiliser
led = machine.Pin(25, machine.Pin.OUT)

# Boucle infinie pour clignoter la LED
while True:
    led.toggle()  # Inverse l'état de la LED
    utime.sleep(1)  # Attendre 1 seconde
```

Dans ce programme, nous utilisons le module `machine` pour gérer les broches GPIO. Nous définissons la broche 25 comme une sortie (`machine.Pin.OUT`). Ensuite, nous utilisons une boucle infinie pour alterner l'état de la LED entre allumé et éteint toutes les secondes à l'aide de la méthode `toggle()` .

Lecture de valeurs analogiques

Vous pouvez également lire des valeurs analogiques à partir de broches GPIO pour mesurer des grandeurs telles que la luminosité avec un capteur de lumière. Voici un exemple pour lire une valeur analogique à partir de la broche 26 :

```
import machine

# Définir la broche GPIO à utiliser
analog_pin = machine.Pin(26)

# Configurer l'ADC (Convertisseur Analogique-Numérique)
adc = machine.ADC(analog_pin)

# Lire et imprimer la valeur analogique
while True:
    valeur = adc.read_u16()  # Lecture de la valeur analogique (0-65535)
    print("Valeur analogique : ", valeur)
```

Dans cet exemple, nous utilisons le module `machine` pour configurer l'ADC (Convertisseur Analogique-Numérique) sur la broche 26. La valeur analogique est lue à l'aide de la méthode `read_u16()` et affichée dans la console.

Conclusion

Le Raspberry Pi Pico avec MicroPython offre une excellente plateforme pour l'apprentissage de la programmation et de l'électronique. Vous pouvez créer une variété de projets intéressants en utilisant les broches GPIO pour contrôler des composants électroniques. N'hésitez pas à explorer davantage et à créer vos propres projets en utilisant ces concepts de base.

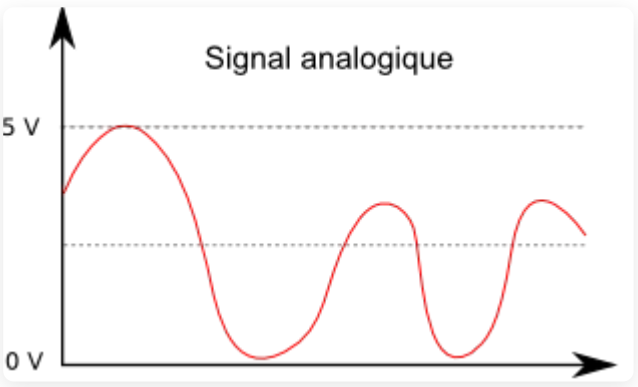
Les différents types de signaux

Les microcontrôleurs sont des composants essentiels dans de nombreux systèmes électroniques, des appareils ménagers aux robots industriels. Pour interagir avec leur environnement, ces microcontrôleurs utilisent différents types de signaux. Dans cet article, nous allons explorer en détail les trois principaux types de signaux utilisés : les signaux analogiques, les signaux digitaux et les signaux à modulation de largeur d'impulsion (PWM). Nous verrons comment ils fonctionnent, leurs applications et les bonnes pratiques pour les utiliser.

1. Signaux Analogiques

Qu'est-ce qu'un Signal Analogique ?

Un signal analogique est un **signal continu** qui peut prendre une **infinité de valeurs** dans une plage donnée. Contrairement aux signaux digitaux, qui ne peuvent prendre que des valeurs discrètes (0 ou 1), les signaux analogiques peuvent représenter des variations continues, comme la température, la luminosité ou le son.



Exemple de Signal Analogique

Imaginez un capteur de température analogique. La sortie de ce capteur est une tension continue qui varie en fonction de la température mesurée. Par exemple, à 20°C, la sortie pourrait être de 2V, à 25°C, elle pourrait être de 2.5V, et ainsi de suite.

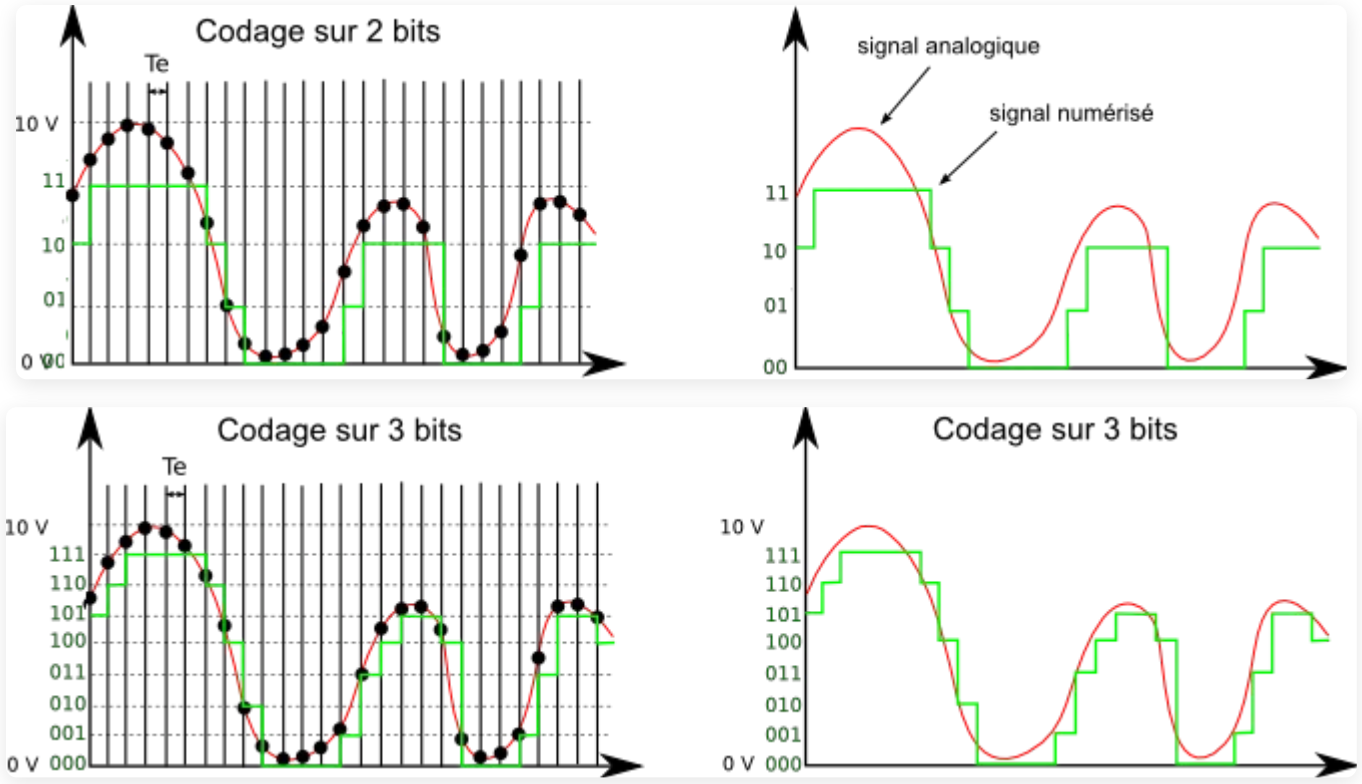
Utilisation dans les Microcontrôleurs

Les microcontrôleurs ne peuvent pas traiter directement les signaux analogiques. Ils doivent **convertir ces signaux** en valeurs numériques à l'aide d'un **convertisseur analogique-numérique (ADC)**. Une fois le signal converti, le microcontrôleur peut le traiter de manière numérique.

L'ADC **convertit une tension analogique continue** en une valeur numérique discrète que le microcontrôleur peut traiter. Cette conversion se fait en **échantillonnant** le signal analogique à des intervalles de temps réguliers et en attribuant à chaque échantillon une valeur numérique correspondante.

Chaque échantillon est attribué à la valeur numérique la plus proche parmi un ensemble de valeurs discrètes possibles.

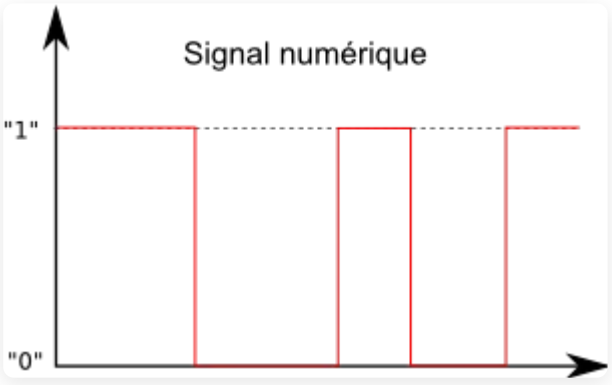
Une caractéristique important des ADC est leur **résolution** : C'est le **nombre de bits de la sortie numérique**. Une résolution de **10 bits** signifie que la sortie peut prendre **1024** valeurs différentes (2^10). **Plus la résolution est élevée, plus la conversion est précise**.



2. Signaux Digitaux

Qu'est-ce qu'un Signal Digital ?

Un signal **digital**, ou **numérique**, ne peut prendre que deux valeurs distinctes : **0 (faible) ou 1 (élevé)** (en tout cas en ce qui concerne les i/o des micro-controlleurs). Ces signaux sont utilisés pour représenter des états binaires ou des données discrètes.



Exemple de Signal Digital

Considérons un **interrupteur** connecté à un microcontrôleur. Lorsque l'interrupteur est fermé, la sortie est à 1 (élevé), et lorsqu'il est ouvert, la sortie est à 0 (faible).

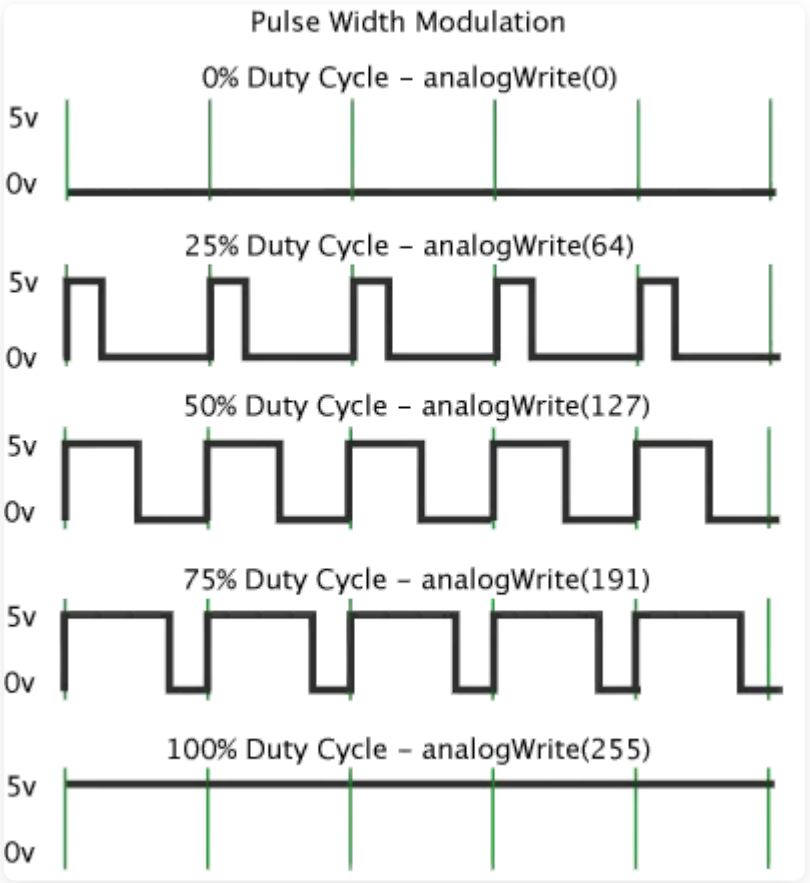
Utilisation dans les Microcontrôleurs

Les microcontrôleurs traitent naturellement les signaux digitaux. Ils peuvent lire les états des broches d'entrée/sortie (I/O) et exécuter des actions en fonction de ces états.

3. Signaux à Modulation de Largeur d'Impulsion (PWM)

Qu'est-ce que le PWM ?

La **modulation de largeur d'impulsion** (GB ***PWM ou pulse width modulation***) est une technique où la largeur des impulsions d'un signal est modulée pour représenter une **valeur analogique moyenne**. Le **rapport cyclique** (GB ***duty cycle***) détermine la **proportion de temps** pendant laquelle le signal est à l'état haut (1) par rapport à l'état bas (0).



Le graphe ci-dessus montre l'équivalence entre différents duty cycles et la valeur équivalente en analogique (analogWrite).

Exemple de PWM

Supposons que nous ayons une LED contrôlée par un signal PWM. En modifiant le rapport cyclique, nous pouvons contrôler la luminosité de la LED. Un rapport cyclique de 50% signifie que le signal est haut pendant 50% du temps et bas pendant les 50% restants, ce qui correspond à une luminosité moyenne de la LED.

Utilisation dans les Microcontrôleurs

Les microcontrôleurs peuvent générer des signaux PWM à l'aide de modules dédiés. Ces signaux sont souvent utilisés pour le contrôle de la puissance, comme dans les moteurs, les LEDs ou les régulateurs de tension.

Conclusion

Les microcontrôleurs utilisent des signaux **analogiques**, **digitaux** et **PWM** pour interagir avec leur environnement. Chaque type de signal a ses propres caractéristiques, avantages et inconvénients. Les signaux analogiques permettent de représenter des valeurs continues, les signaux digitaux sont parfaits pour les états binaires et la communication, et les signaux PWM offrent une solution efficace pour le contrôle de puissance et la génération de valeurs analogiques moyennes.

En comprenant comment utiliser ces différents types de signaux, vous pouvez concevoir des systèmes plus robustes et efficaces.

INTELLIGENCE ARTIFICIELLE

Chatbot, assistant intégré, agent : qui fait quoi ?

> *ChatGPT, Claude Code, GitHub Copilot, Cursor, Gemini, Operator… Tous reposent sur des modèles de langage, mais ils ne se ressemblent pas du tout à l'usage. Ce cours pose les trois grandes familles d'outils IA, ce que chacune sait faire, et comment choisir selon ton besoin — pour le code et au-delà.*

Quand on parle d'« IA », on parle souvent de **ChatGPT**. Mais ChatGPT n'est qu'une **catégorie** d'outil IA parmi d'autres. La même technologie sous-jacente (un modèle de langage, ou *LLM*) peut être emballée de **plusieurs façons** très différentes, qui ne servent pas du tout aux mêmes besoins.

Comprendre ces différences, c'est éviter trois pièges courants :

- utiliser un **chatbot** pour une tâche qui demanderait un **agent** → tu passes la moitié du temps en copier-coller
- utiliser un **agent** pour une tâche que résoudrait un **chatbot** → tu déploies un bulldozer pour planter un clou
- choisir un outil sur sa **hype** sans regarder ce qu'il fait vraiment → tu paies pour des fonctionnalités que tu n'utilises pas

---

🧠 Le point commun : un modèle de langage

Sous le capot, tous ces outils s'appuient sur un **LLM** (*Large Language Model*) — un modèle entraîné à prédire le texte le plus probable à partir d'un contexte. GPT-4, Claude Opus, Gemini, Llama, Mistral… ce sont des LLM.

Un LLM brut, c'est juste une fonction : on lui donne du texte, il complète. Pour devenir un outil utilisable, il faut l'**emballer** dans une interface. C'est cet **emballage** qui définit la catégorie d'outil — pas le modèle.

> 💡 *Le même modèle (par exemple Claude Sonnet 4.6) peut alimenter **un chatbot** (Claude.ai), **un assistant intégré** (Cursor) ou **un agent** (Claude Code). C'est le **type d'interface** qui change, pas l'intelligence du modèle.*

🗨️ Famille 1 — Le chatbot

Ce que c'est

Une **interface de conversation** dans un navigateur ou une appli mobile. Tu écris, le modèle répond. Rien ne sort de cette fenêtre — pour appliquer un résultat, **c'est à toi de copier-coller**.

EXEMPLES	URL
ChatGPT	chatgpt.com
Claude	claude.ai
Gemini	gemini.google.com
Le Chat (Mistral)	chat.mistral.ai
Perplexity	perplexity.ai

Ce qu'il fait bien

- **Expliquer** un concept (en cours, à la maison, sur un sujet inconnu)
- **Brainstormer** une idée, une structure, un plan
- **Rédiger** un texte, un email, une dissertation, un script
- **Traduire** ou **reformuler**
- **Analyser** un document ou une image que tu lui colles
- **Réviser** : tu lui demandes de te poser des questions sur un cours
- **Coder** par bouts copiés-collés (snippet par snippet)

Ses limites

- N'a **aucun accès** à ton disque, ton terminal, internet (sauf option *web search*), tes outils
- Tout résultat doit être **copié-collé manuellement** vers ta vraie destination
- La conversation se perd quand tu fermes l'onglet (sauf si tu sauvegardes)
- Tu dois lui **donner** le contexte à chaque fois (ses derniers échanges sont sa seule mémoire)

Quand l'utiliser

> *Quand le travail final est **du texte que tu vas relire et utiliser à la main**.*

Exemple : tu prépares un exposé, tu veux une explication claire d'un concept, ou tu demandes une review de ta dissertation. Le chatbot est parfait.

Famille 2 — L'assistant intégré (autocomplétion / inline)

Ce que c'est

L'IA est **intégrée dans un outil que tu utilises déjà** (ton éditeur de code, ton traitement de texte, ton mail). Elle te propose des **complétions** au fil de la frappe, ou répond à des questions **sans quitter** l'application.

EXEMPLES	INTÉGRÉ DANS
GitHub Copilot	VS Code, JetBrains, Visual Studio
Cursor (chat + tab)	Son propre éditeur (fork de VS Code)
Windsurf	Son propre éditeur
JetBrains AI	IntelliJ, Rider, PyCharm…
Microsoft 365 Copilot	Word, Excel, Outlook, Teams
Notion AI	Notion
Gmail Smart Compose	Gmail

Ce qu'il fait bien

- **Suggérer la suite** de ce que tu écris (variable, fonction, paragraphe…)
- **Compléter du code** répétitif : boucles, getters/setters, tests évidents
- **Répondre à une question** sur ton fichier ouvert sans changer d'app
- **Rester rapide** : pas besoin de prompt long, tu travailles en flot

Ses limites

- Vision **locale**, souvent limitée au fichier ouvert ou aux fichiers proches
- Ne **prend pas l'initiative** : c'est toi qui pilote ligne par ligne
- Peut **inventer** une API inexistante si le contexte est trop court (*hallucination*)
- Lié à un outil précis — si tu changes d'éditeur, tu perds l'assistant

Quand l'utiliser

> *Quand tu veux **garder le contrôle ligne par ligne** mais accélérer la frappe et éviter les répétitions.*

Exemple : tu codes une fonction, Copilot devine la suite et te la propose en gris — `Tab` pour accepter. Ou : tu écris un mail dans Gmail, l'IA complète ta phrase.

🤖 Famille 3 — L'agent

Ce que c'est

L'IA **agit** dans ton environnement. Elle lit des fichiers, exécute des commandes, modifie du code, fait des appels réseau — sous ton contrôle, mais **de sa propre initiative** quand tu lui donnes un objectif de haut niveau.

EXEMPLES (CODE)	PLATEFORME
Claude Code	Terminal, Anthropic
Codex CLI	Terminal, OpenAI
Gemini CLI	Terminal, Google
Cline / **Roo Code**	VS Code (open source)
Aider	Terminal (open source)

EXEMPLES (HORS CODE)	ACTION
Claude Computer Use	Contrôle ton bureau (souris, clavier, screenshots)
OpenAI Operator	Pilote un navigateur web pour faire des tâches
Browser Use	Bibliothèque open source, agents web
n8n / Make + IA	Automatisations complexes avec étapes IA

Ce qu'il fait bien

- **Tâches multi-étapes** où l'ordre dépend des résultats intermédiaires
- **Modifier plusieurs fichiers** de façon cohérente dans un projet
- **Lancer des commandes** (tests, build, git, scripts shell)
- **Naviguer sur le web** et accomplir des tâches concrètes (remplir un formulaire, comparer des prix…)
- **Itérer seul** : essayer, lire l'erreur, corriger, retenter

Ses limites

- Peut faire des **bêtises irréversibles** s'il n'est pas surveillé (`rm` , push forcé, suppression de données…)
- **Plus lent** par opération qu'un chatbot ou Copilot (beaucoup d'appels au modèle pour une tâche)
- **Consomme beaucoup** de quota / d'API (un agent peut faire 50 appels là où un chatbot en fait 1)
- Demande une **vraie supervision** : tu dois relire ce qu'il fait, pas juste accepter

Quand l'utiliser

> *Quand tu peux **décrire un objectif** plutôt qu'une séquence d'étapes, et que tu acceptes de **superviser** ce qui se passe.*

Exemple : *« ajoute un bouton Copier au générateur de mot de passe, écris un test, lance les tests »*. L'agent ouvre les fichiers, modifie, lance les tests, lit la sortie, ajuste. Tu valides à la fin.

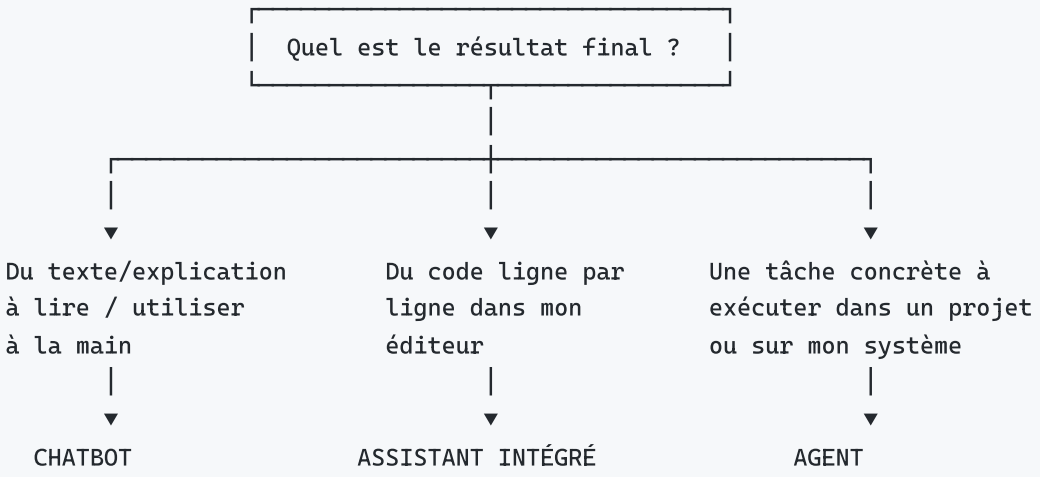
📊 Tableau comparatif

CRITÈRE	CHATBOT	ASSISTANT INTÉGRÉ	AGENT
Interface	Navigateur / appli	Dans ton outil	Terminal / IDE
Touche à ton système ?	Non	Édition locale ciblée	Oui, fichiers + commandes
Prise d'initiative	Aucune	Faible	Forte
Granularité	Échange par échange	Ligne par ligne	Tâche complète
Vitesse perçue	Rapide	Très rapide (inline)	Plus lent (multi-étapes)
Coût (intensif)	Faible	Moyen	Élevé
Risque	Quasi nul	Faible	Réel (à superviser)
Mémoire du projet	Toi tu la fournis	Fichier ouvert + proches	Lit le codebase

---

🎯 Comment choisir : la question à se poser

Avant de sortir un outil, demande-toi :



▸ Quelques cas concrets

BESOIN	BON OUTIL
Comprendre une notion de cours	**Chatbot** (ChatGPT, Claude)
Réviser pour un examen	**Chatbot** (génère un quiz, corrige-moi)
Rédiger un mail délicat	**Assistant intégré** (Gmail/Outlook) ou **chatbot**
Écrire une fonction `triBulle` en C#	**Assistant intégré** (Copilot, Rider AI)
Refactoriser un projet sur plusieurs fichiers	**Agent** (Claude Code, Codex CLI)
Faire passer des tests qui échouent	**Agent**
Comparer 3 produits sur des sites différents	**Agent web** (Operator, Browser Use)
Traduire un texte	**Chatbot**
Résumer un PDF de 50 pages	**Chatbot** (avec upload)
Générer 50 fichiers à partir d'un template	**Agent**
Compléter une formule Excel compliquée	**Assistant intégré** (Copilot Excel)
Brainstormer un sujet de TFE	**Chatbot**
Mettre à jour les imports cassés après un rename	**Agent**

🌍 Et au-delà du code ?

Les trois familles existent pour **tous les domaines**, pas juste le code :

- **Recherche / analyse** : un **chatbot** comme Perplexity cite ses sources ; un **agent** comme Operator peut naviguer pour toi.
- **Écriture** : Notion AI (intégré), ChatGPT (chatbot), agents de rédaction long-format (encore expérimentaux).
- **Bureautique** : Microsoft 365 Copilot, Google Duet AI — intégrés à Word, Excel, Sheets…
- **Image / vidéo** : DALL·E, Midjourney, Sora — ce sont des modèles spécialisés, mais l'**emballage** suit les mêmes catégories (interface chat ou intégrée dans un outil).
- **Vie quotidienne** : agents qui planifient des réservations, comparent des assurances, remplissent des formulaires administratifs.

Le réflexe est toujours le même : **quel est le résultat que je veux ? Faut-il que l'IA agisse, ou juste qu'elle me réponde ?**

💸 Et les coûts ?

FAMILLE	MODÈLE ÉCONOMIQUE TYPIQUE
Chatbot	Gratuit avec quotas, abonnement ~20 €/mois pour les versions Pro
Assistant intégré	Abonnement (Copilot : ~10 €/mois ; Cursor : ~20 €/mois ; gratuit pour les étudiants chez certains)
Agent	Soit abonnement (Claude Code Pro), soit **paiement à l'usage** via API (peut grimper vite — un projet long peut coûter quelques €)

> 💡 ***Astuce étudiant** : GitHub Copilot est **gratuit avec un compte étudiant** (GitHub Student Pack). Les chatbots ont tous une version gratuite suffisante pour l'usage scolaire. Pour les agents, les comptes gratuits ont des quotas qui suffisent à découvrir.*

⚠️ Ce que ces outils ne sont pas

Avant de tomber dans le piège du marketing :

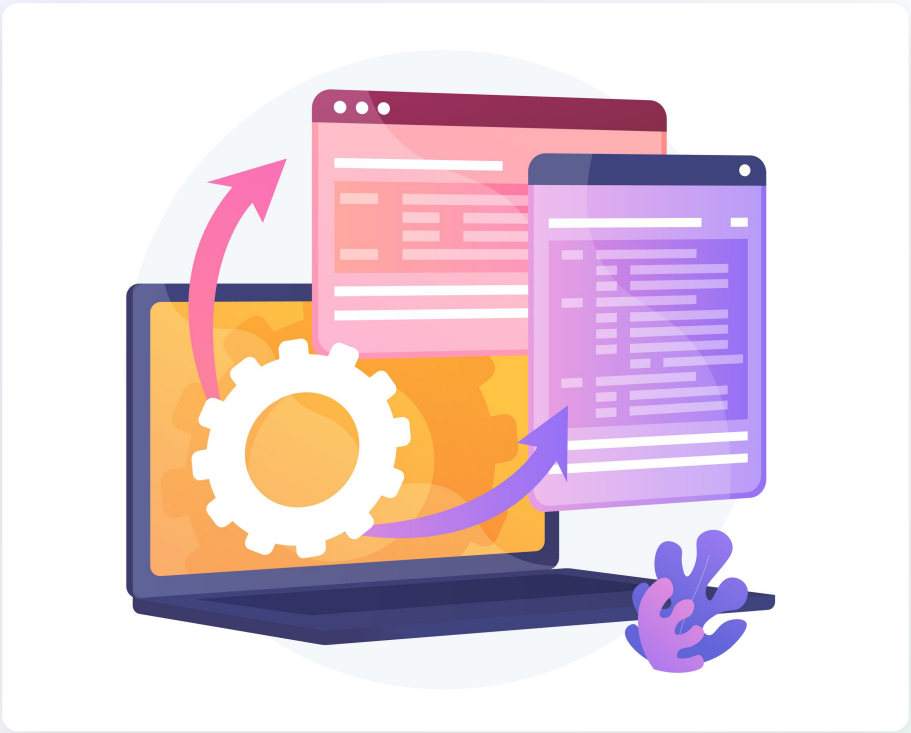
- **Pas une source de vérité.** Tous peuvent **halluciner** — inventer une fonction, une date, un nom de loi. Vérifie ce qui compte.
- **Pas un substitut à la compréhension.** Si tu rends du code que tu ne comprends pas, tu n'apprends rien et tu seras coincé au premier bug.
- **Pas une boîte noire à laquelle déléguer.** Tu restes responsable de ce qui sort de chez toi.
- **Pas conscient.** Le modèle ne « comprend » pas comme un humain — il prédit du texte plausible. Ça suffit pour énormément de tâches, mais ça ne pense pas.

✍️ À retenir

- Tous ces outils reposent sur des **LLM**, mais leur **emballage** détermine ce qu'ils savent faire.
- **Chatbot** : on discute, on copie-colle. Idéal pour expliquer, rédiger, brainstormer.
- **Assistant intégré** : il complète ce que tu fais dans ton outil. Idéal pour rester rapide ligne par ligne.
- **Agent** : il agit dans ton projet ou ton système. Idéal pour les tâches multi-étapes décrites en langage naturel.
- **Bon outil = bon résultat avec le moins d'effort**. Réfléchis au type de résultat avant de choisir.
- Les trois familles existent **hors du code** : bureautique, recherche, image, vie quotidienne.
- Tu restes **responsable** de ce que tu produis avec ces outils — toujours.

Suite

L'article suivant (Coder son projet avec un agent IA) te fait passer à la pratique : installer un agent sur ta machine et l'utiliser pour de vrai sur un projet.



SYSTÈMES D'EXPLOITATION

Fichiers et dossiers

> *Un **fichier** est un **ensemble de données** (texte, image, vidéo, programme, …) enregistré sur un support de stockage (disque dur, clé USB, carte mémoire…). Un **dossier** (ou répertoire) sert à **organiser les fichiers***

Objectifs

- Comprendre ce qu'est un fichier et un dossier.
- Savoir lire la structure d'un disque dur (arborescence).
- Faire la différence entre chemin absolu et chemin relatif.

Qu'est-ce qu'un fichier ?

Un fichier (GB ***file***) est un **ensemble de données** (texte, image, vidéo, programme, …) enregistré sur un support de stockage (disque dur, clé USB, carte mémoire…).

Chaque fichier a :

- **Un nom** (ex. `cours`)
- **Une extension** (ex. `.docx`, `.jpg`, `.mp3`) qui indique son type.

Le nom + l'extension donnent le **nom de fichier complet**.

Nom de fichier complet = [NOM.DU.FICHIER].[EXT]

Exemples :

- Nom complet: `rapport.docx` → nom de fichier = `rapport`, extension = `docx`
- Nom complet: `photo.jpg` → nom de fichier = `photo`, extension = `jpg`

Le nom de fichier est tout ce qui se trouve **AVANT le dernier point** du nom de fichier complet.

L'extension du fichier se trouve **APRÈS le dernier point** du nom du fichier. Elle indique le type de fichier et permet de déterminer le programme pour l'ouvrir. Par exemple, .docx pour un document Word ou .jpg pour une image.

Exemple:

Qu'est-ce qu'un dossier ?

Un dossier (GB ***folder***), aussi appelé répertoire (GB ***directory***) sert à **organiser les fichiers**.

- Un dossier peut contenir **des fichiers** et **d'autres dossiers**.
- On parle de **dossier parent** pour les dossiers qui contiennent d'autres dossiers.

- On parle de **dossier enfant (ou sous-dossier)** pour les dossiers qui sont à l'intérieur d'un autre.

💡 En résumé : **les dossiers contiennent des fichiers et parfois d'autres dossiers**.

Arborescence des dossiers

Les dossiers sont organisés comme un **arbre** :

- En haut, on trouve la **racine** (GB ***root***), qui est le point de départ, le dossier "le plus en haut".
- Chaque dossier peut contenir d'autres dossiers ou fichiers.

Exemple sous Windows :

```
C:\
 └── Utilisateurs
      └── Adrien
      |    └── Documents
      |    |    └── notes.txt
      |    └── Téléchargements
      └── liste.txt
```

Ici :

- `C:\` est la racine du disque C.
- `Utilisateurs` est un enfant de `C:\`.
- `Utilisateurs` a 2 enfants:
  - le dossier `Adrien`
  - le fichier `liste.txt`
- `Adrien` est l'enfant de `Utilisateurs`.
- `Adrien` a 2 dossiers enfants:
  - `Documents` est l'enfant de `Adrien`.
  - `Téléchargements` est l'enfant de `Adrien`.
- `Adrien` est donc le parent de `Documents` et `Téléchargements`
- `notes.txt` est un fichier dans `Documents`.

Les chemins d'accès

Un **chemin** indique où se trouve un fichier ou dossier.

Chemin absolu

C'est le chemin **complet** depuis la racine. Exemple :

```
C:\Utilisateurs\Adrien\Documents\notes.txt
```

▸ Chemin relatif

C'est le chemin **par rapport à l'endroit où l'on se trouve**.

- `..` signifie « remonter d'un dossier ».

Exemple : Si je suis dans `C:\Utilisateurs\Adrien` et que je veux lire le fichier `notes.txt` qui se trouve dans `Documents`, je peux écrire :

```
type Documents\notes.txt
```

Si je veux remonter d'un cran pour afficher le fichier `liste.txt` dans `C:\Utilisateurs`, je peux écrire :

```
type ..\liste.txt
```

À retenir

- Un fichier = données enregistrées avec un nom et une extension.
- Un dossier contient des fichiers et/ou d'autres dossiers.
- Organisation en **arborescence** (racine → parents → enfants).
- **Chemin absolu** = depuis la racine.
- **Chemin relatif** = depuis l'endroit où tu es déjà (`..` pour remonter).

Chasses aux trésors

> *Apprends à parcourir des arborescences de fichier et les bases de la ligne de commande. Sois le 1er à résoudre ces énigmes!*

Les aventures et énigmes à résoudre sont envoyées dans ton espace personnel sur https://fichiers.ttrinfo.be.

CANSAT

Liens utiles

Liens utiles

- https://www.innoviris.brussels/fr/program/cansat-belgium
- Wiki Cansat: https://wiki.mchobby.be/index.php?title=ENG-CANSAT-PICO-MISSION1-RECEIVE
- https://meteologix.com/be/observations/pressure-qnh.html
- Altitude Jodoigne: https://fr-be.topographic-map.com/map-vncz/Jodoigne/?center=50.72818%2C4.87376&zoom=16&popup=50.72866%2C4.87617

Documentations

Cansat - Projets paralèlles

> *Projets pour les élèves qui ne participent pas à CanSat.*

Voici pour **les projets 1 et 8** :

👉 une **MISSION complète** (façon CanSat) 👉 un **CAHIER DES CHARGES précis**, clair et exploitable en classe 👉 niveau et esprit équivalent au concours CanSat (mesures + datas + analyse + vraie mission finale)

Projet 1 — GreenSat : Mini-station météo autonome

🎯 Mission

Construire une **station météo autonome** capable de mesurer l'environnement autour de l'école et de transmettre les données en temps réel à une station au sol. L'équipe doit **collecter**, **transmettre**, **enregistrer** et **analyser** les données scientifiques pour comprendre les variations météo locales.

La mission finale consiste à **installer la station dehors pendant 24 h**, récupérer les données et produire un **rapport scientifique** (graphiques + analyse + interprétation).

📘 Cahier des charges

FONCTIONNALITÉS OBLIGATOIRES

- Lire et enregistrer au minimum :
  - Température (±0.5°C)
  - Humidité relative
  - Pression atmosphérique
  - Luminosité
- Enregistrer les données toutes les **30 secondes à 2 minutes**.
- Stocker les données sur **carte SD** *ou* transmettre à un PC via :
  - LoRa
  - 433 MHz
  - Bluetooth Low Energy
  - USB Série (solution de secours)

COMPOSANTS TECHNIQUES REQUIS

- Microcontrôleur (Raspberry Pi Pico conseillé)
- Capteur combiné (BME280 / DHT22 + BMP280 / SHT31…)
- Capteur de luminosité (BH1750 ou LDR)
- Horloge interne (RTC optionnel mais conseillé)

- Module de communication (LoRa / 433 MHz / BLE)
- Boîtier étanche IP45 minimum (impression 3D autorisée)

CONTRAINTES TECHNIQUES

- La station doit fonctionner **en extérieur** pendant au moins **24 h**.
- Résister à la pluie légère (grille anti-gouttes, aérations protégées).
- Alimentation autonome :
  - batterie + powerbank
  - ou panneau solaire (option bonus)

FORMAT DE DONNÉES

Chaque enregistrement DOIT contenir :

```
timestamp ; temperature ; humidite ; pression ; luminosite
```

TESTS OBLIGATOIRES

- Test de calibration des capteurs
- Test d'endurance (30 minutes de fonctionnement continu)
- Test de transmission (si radio)

LIVRABLES

- Code documenté
- Schéma électronique
- Boîtier fonctionnel
- Fichier CSV final
- Rapport scientifique :
  - graphiques
  - interprétation des variations
  - comparaison au site météo officiel
  - limites, erreurs possibles

Projet 8 — NoiseMap : Cartographie sonore de l'école

🎯 Mission

Construire un **sonomètre autonome** capable de mesurer le niveau sonore en différents lieux du bâtiment. Objectif final : réaliser une **carte thermique du bruit dans l'école** selon l'heure et l'endroit, puis proposer des **solutions d'amélioration acoustique**.

La mission finale consiste à **déployer plusieurs capteurs** dans l'école (couloirs, réfectoire, cour…) pendant une journée et analyser les données.

📘 Cahier des charges

FONCTIONNALITÉS OBLIGATOIRES

- Mesure du **niveau sonore en dB** (valeur simple, pas une analyse spectrale).
- Enregistrement d'une valeur toutes les **5 à 30 secondes**.
- Stockage sur carte SD *ou* transmission sans-fil.
- Détection des pics sonores (portes, cris, cloches, chariots…).
- Création d'une **HeatMap** :
  - x = lieu
  - y = moment de la journée
  - couleur = niveau sonore moyen

COMPOSANTS TECHNIQUES

- Microcontrôleur (Pico conseillé)
- Microphone analogique ou I²S (MAX4466, INMP441…)
- Circuit de lissage (si micro analogique)
- Carte SD ou module radio (LoRa / BLE)
- Petit boîtier discret et ventilé
- Batterie pour 8–24 h

CONTRAINTES TECHNIQUES

- Le capteur doit pouvoir être placé dans un couloir sans attirer l'attention.
- Le micro ne peut PAS enregistrer l'audio brut (RGPD).
- Seules des **mesures de niveau** sont autorisées (pas d'enregistrement son → juste amplitude).
- Résolution recommandée : ±2 dB.
- L'équipe doit définir :
  - les zones de mesure
  - les moments de mesure
  - les conditions de calibration

FORMAT DES DONNÉES

```
timestamp ; lieu ; niveau_dB
```

TESTS OBLIGATOIRES

- Calibration :
  - mesure à 1 m d'une source connue (cloche, sonomètre smartphone)

- Test de stabilité
- Test de variation rapide (bruit soudain)

LIVRABLES

- Code documenté
- Schéma électronique
- Photo ou dessin du boîtier
- CSV ou fichier JSON final
- Carte thermique (Excel / Python Matplotlib)
- Rapport d'analyse :
  - zones les plus bruyantes
  - heures critiques
  - causes probables
  - propositions d'amélioration (affiches "silence", changement de mobilier, mousse acoustique…)

TECHNOLOGIES

3D et impression

> *De la modélisation 3D sur ordinateur à l'objet physique qui sort d'une imprimante : on apprend à dessiner en volume avec TinkerCad, à comprendre comment fonctionne une imprimante 3D, et on termine par la fabrication d'un petit objet.*

TinkerCad — modéliser ton premier objet

> *Premier contact avec la modélisation 3D : tu fabriques un porte-clés personnalisé avec tes initiales. En le construisant, tu apprends à manipuler la vue, à empiler des solides, à percer, à aligner et à redimensionner — tout ce qu'il faut pour modéliser un objet simple.*

TinkerCad est un outil de **modélisation 3D** gratuit qui tourne dans le navigateur. Pas d'installation, pas de licence : tu te crées un compte (ou tu en utilises un fourni par le prof) et tu commences. Il est volontairement simple — pas de fonctions ultra avancées — mais il suffit largement à dessiner les objets qu'on va imprimer.

Dans ce cours, on apprend en construisant : tu vas modéliser un **porte-clés à tes initiales**. Pendant la construction, tu rencontres tous les gestes de base.

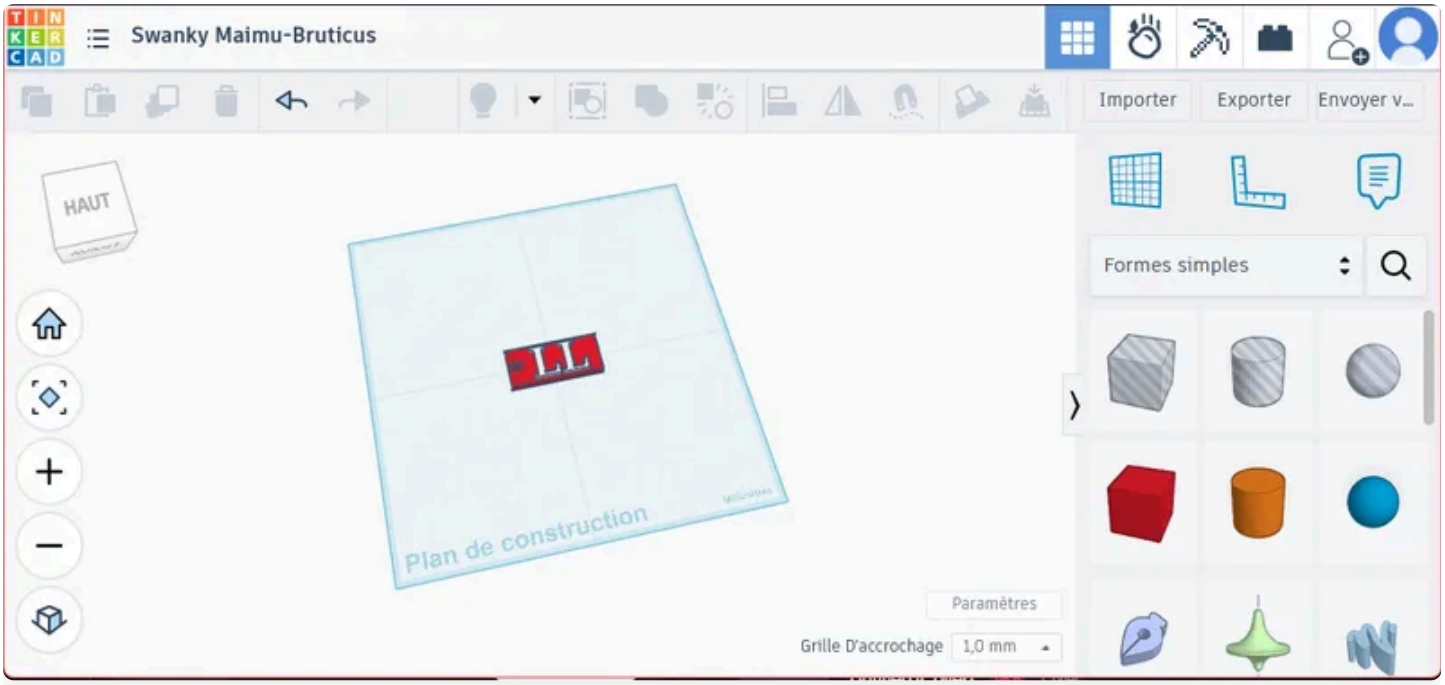
🎯 Objectifs

À la fin de ce cours, tu seras capable de :

1. Te repérer dans l'espace 3D de TinkerCad (zoomer, tourner, déplacer la vue).
2. Ajouter des **solides** et des **perçages** depuis la bibliothèque.
3. **Déplacer**, **redimensionner** et **aligner** des formes les unes par rapport aux autres.
4. Combiner solides et perçages pour creuser un objet (faire un trou, graver un texte).
5. Exporter ton modèle au format **STL** pour l'envoyer vers une imprimante 3D.

Partie 1 — L'interface et la vue 3D

Une fois connecté sur tinkercad.com, clique sur **Créer** → **Conception 3D**. TinkerCad t'ouvre un atelier vide.



Le **plan de construction** est ce grand sol quadrillé. C'est ton espace de construction, gradué en **millimètres** : un cube de 20 mm de côté sera bel et bien un cube de 2 cm une fois imprimé.

> 💡 ***Si tu as déjà vu une imprimante 3D**, pense au plan de travail comme au **plateau d'impression** : sa taille en TinkerCad correspond à la zone où l'imprimante peut déposer du plastique. Tout ce qui dépasse du plateau ne pourra pas être imprimé.*

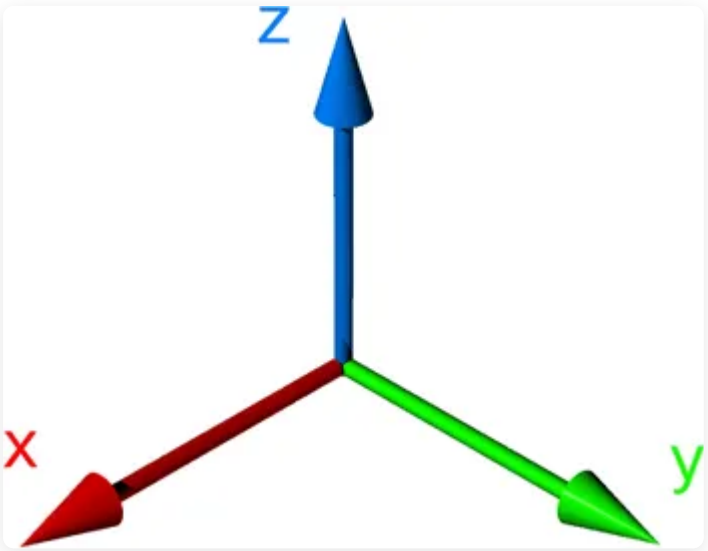
▸ Bouger la vue avec la souris

Quatre gestes à mémoriser tout de suite — c'est ce que tu vas faire le plus souvent :

GESTE SOURIS	EFFET
Clic gauche	Sélectionner / manipuler les objets (déplacer, redimensionner, lever)
Molette	Zoom avant / arrière
Clic droit maintenu + glisser	Faire tourner la vue (orbiter)
Clic molette maintenu + glisser (ou Shift + clic droit)	Déplacer la vue (sans rotation)

> 💡 ***Si tu es perdu :** en haut à gauche, le petit cube ViewCube te permet de revenir à une vue propre (clique sur la face "Haut", "Avant", etc.). Le bouton **Maison** revient à la vue par défaut.*

Les axes X, Y, Z



Trois directions :

- **X** — droite ↔ gauche
- **Y** — devant ↔ derrière
- **Z** — haut ↔ bas

Quand tu sélectionnes un objet, des petites flèches apparaissent dans ces trois directions : ce sont les axes. C'est avec elles que tu vas déplacer, redimensionner ou lever l'objet.

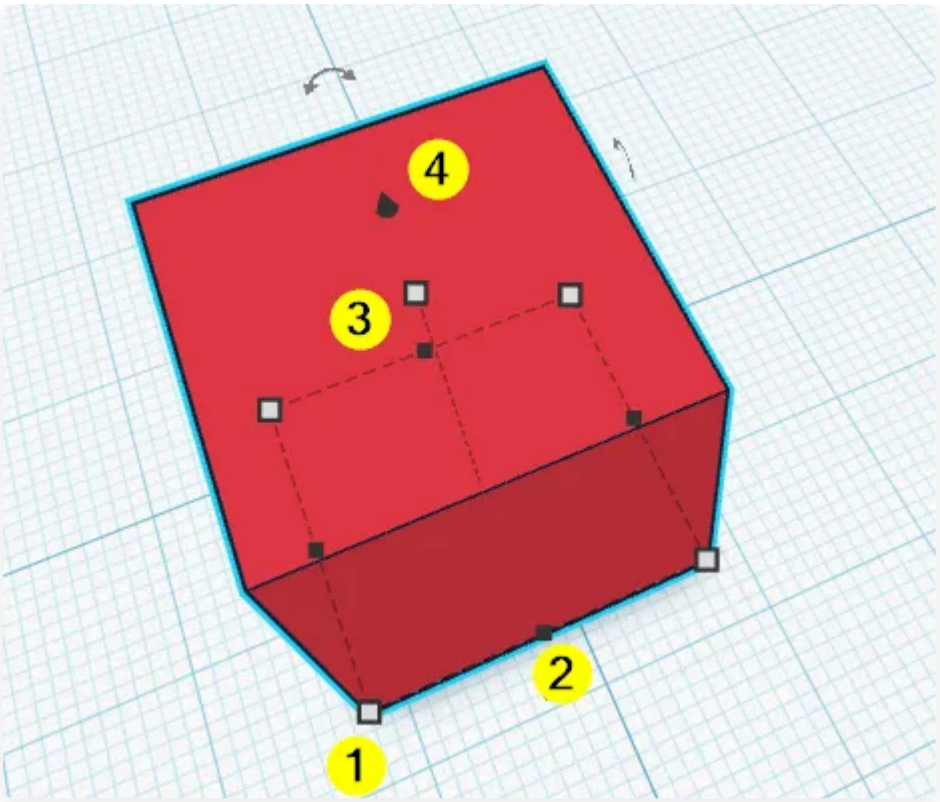
Partie 2 — Ajouter une première forme

À droite de l'écran, la **bibliothèque des formes** liste tous les solides disponibles : cube, cylindre, sphère, prisme, texte, etc.

Étape 1 : glisse un **cube rouge** depuis la bibliothèque vers le plan de travail. Tu obtiens un cube de 20 × 20 × 20 mm posé au sol.

Sélectionner, déplacer, lever

Clique sur le cube pour le sélectionner. Plusieurs éléments apparaissent autour de lui :



- des **poignées** : petits carrés blancs dans les coins [1], noirs sur les arêtes [2] et un blanc sur le dessus [3]. C'est ce qui sert à le **redimensionner** ;
- un **cône noir** pointé vers le haut [4], séparé du corps de la forme — c'est ce qui sert à le **lever** (le déplacer verticalement, sur l'axe Z). La pointe du cône devient claire au survol.

Pour déplacer le cube sur le plan de travail :

- Clique au **centre** du cube, maintiens, et glisse — il suit la souris en restant collé au sol.
- Ou utilise les **flèches du clavier** pour des déplacements précis de **1 mm** (ou **1 cm** avec Maj enfoncé).
- Tu peux fixer un seul axe de déplacement en appuyant sur la touche **Shift/Maj** en même temps.

Pendant que tu déplaces le cube, deux indications de distance s'affichent à côté de lui : une par axe du plan de travail. Le sol a en effet **deux axes** — **X** (de gauche à droite) et **Y** (de l'avant vers l'arrière). Le troisième axe, **Z**, est la hauteur — c'est celui qu'on utilise avec le cône noir pour lever l'objet. Quand tu glisses ton cube sur le sol, tu te déplaces donc en X et en Y simultanément ; avec Shift/Maj enfoncé, tu n'en bouges plus qu'un seul à la fois.

Pour lever le cube au-dessus du plan :

- Saisis le **cône noir** [4] du dessus et tire-le vers le haut. Une indication de hauteur apparaît : c'est la valeur Z.
- Pour le reposer au sol, tire-le vers le bas jusqu'à 0.

Pour changer ses dimensions : clique et tire sur une des poignées blanches ou noires (voir ci-dessous).

> ⚠️ ***Attention :*** *si tu cliques sur une* ***poignée*** *au lieu du corps de la forme, tu vas la redimensionner au lieu de la déplacer. Si ton objet se déforme alors que tu voulais juste le bouger, fais* ***Ctrl + Z*** *et recommence en cliquant bien au centre.*

Redimensionner

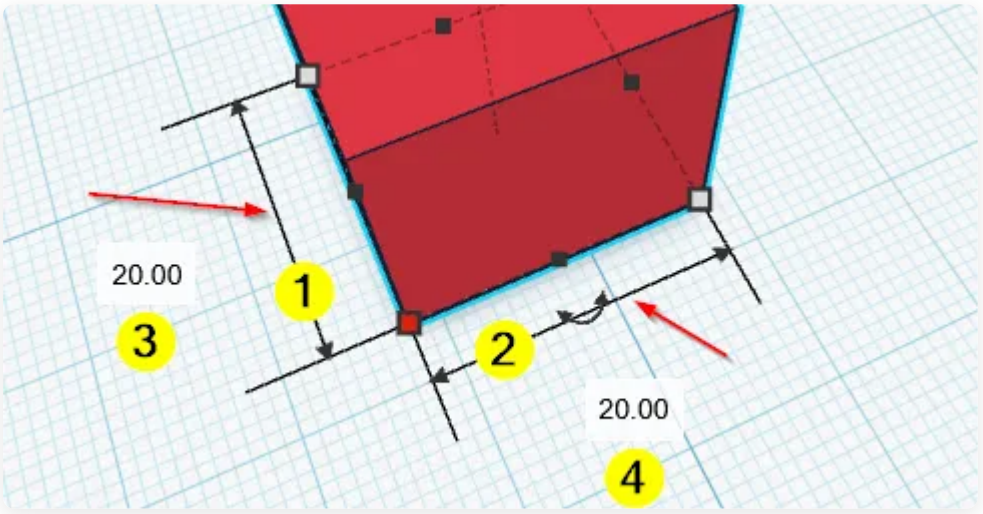
Les poignées blanches [1] et noires sur les côtés [2] étirent la forme dans une ou deux directions. Les poignées blanches **en coin** (celles du bas) permettent de redimensionner les 2 côtés qu'elles touchent en même temps. Les poignées noires permettent de changer la dimension sur un seul axe. Appuie sur **Shift/Maj** pour que les autres dimensions s'adaptent en même temps : ta forme garde alors les mêmes proportions.

Pour un porte-clés, on veut une base **rectangulaire plate**. Sélectionne ton cube, puis tape dans la case de dimension qui apparaît :

- **Largeur (X)** : 50 mm
- **Profondeur (Y)** : 20 mm
- **Hauteur (Z)** : 4 mm

Tu obtiens une plaque rectangulaire, fine et allongée. C'est la base de ton porte-clés.

Quand tu modifies une dimension, une ligne de mesure apparaît [1+2] (ou deux lignes si tu modifies deux dimensions en même temps).

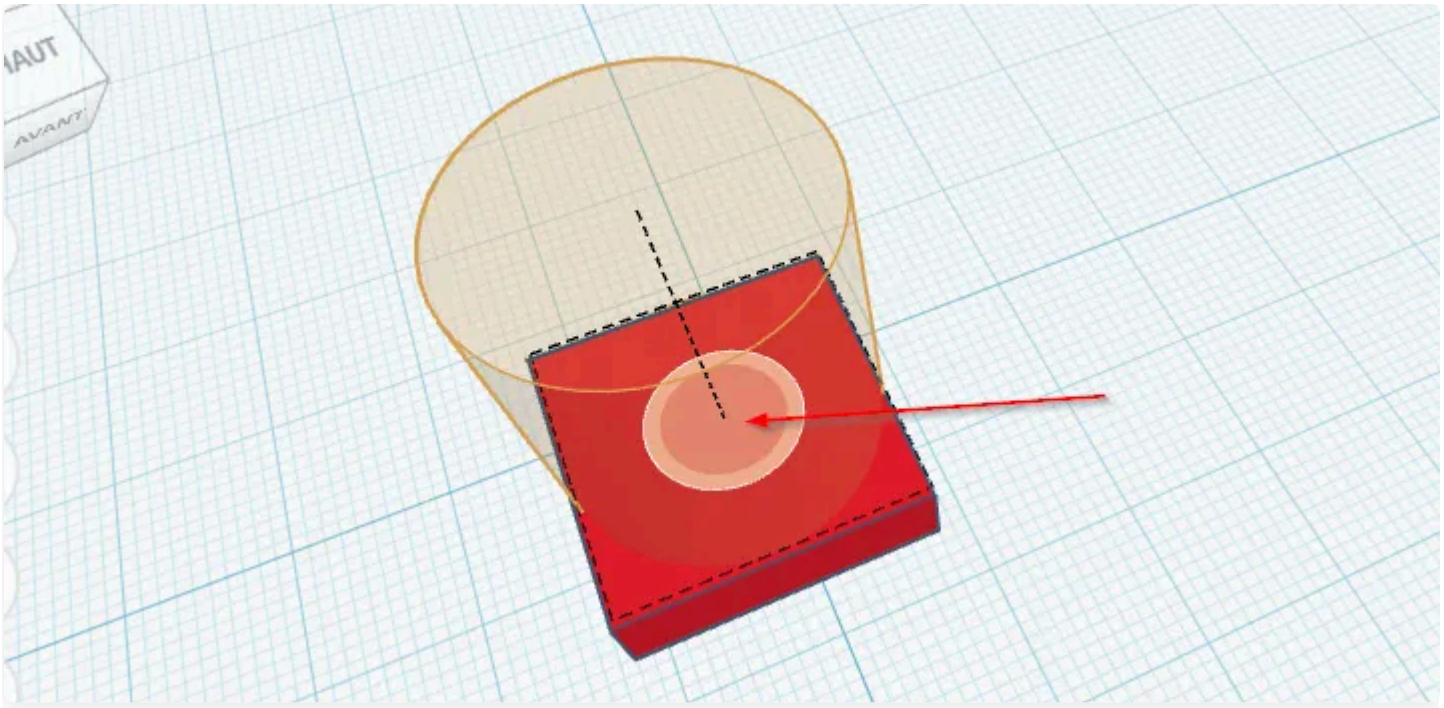


> 💡 ***Astuce :*** *tu peux toujours cliquer directement sur les chiffres affichés sous la ligne de mesure [3+4] pour entrer une valeur précise au clavier. C'est plus rapide et plus précis que de tirer à la souris.*

Partie 3 — Ajouter du texte (tes initiales)

Poser le texte directement sur la plaque

TinkerCad a une astuce très pratique : quand tu **glisses une nouvelle forme depuis la bibliothèque**, tu peux la déposer sur n'importe quelle surface d'un objet déjà présent — elle se posera dessus, pas au sol :



> ⚠️ ***C'est seulement au moment de la création*** *que ça fonctionne. Un objet déjà posé au sol ne se collera* ***pas*** *automatiquement sur un autre objet, même si tu le glisses au-dessus. Pour reposer un objet existant, il faut utiliser le* ***cône noir*** *(voir plus bas).*

1. Dans la bibliothèque, cherche la forme **Texte** (Text en anglais).
2. Glisse-la depuis la bibliothèque, **directement au-dessus de la plaque**, sans la lâcher.
3. Pendant que tu déplaces, regarde l'objet : une **ombre arrondie** sous le curseur indique la surface de dépôt. Quand l'ombre est sur le dessus de la plaque, relâche.
4. Le texte se pose à hauteur Z = **4 mm**, posé sur la plaque, et non au sol.

▸ Régler le contenu du texte

Une zone de réglage apparaît à droite quand le texte est sélectionné :

- **Text** : tape tes initiales (ex. `LL`).
- **Police** : laisse celle par défaut, ou choisis une plus lisible.
- **Hauteur (Z)** : 4 mm (la même que la plaque).

Redimensionne ensuite ton texte pour qu'il soit un peu plus petit que la plaque, par exemple **20 × 10 × 4 mm**.

▸ Si tu t'es trompé et que le texte est posé au sol

Pas de panique — c'est facile à corriger avec le **cône noir** :

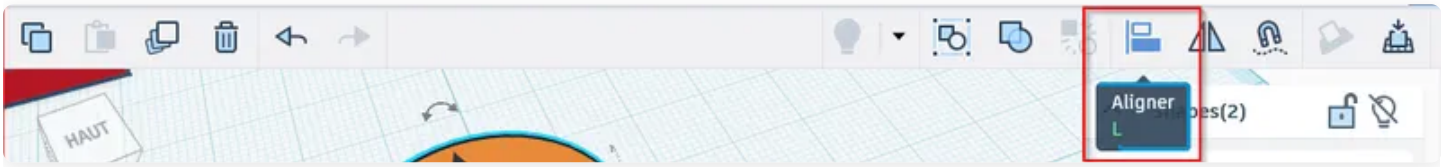
1. Sélectionne le texte.
2. Saisis le **cône noir** au-dessus de la forme (la flèche vers le haut, séparée du corps).
3. Tire-le vers le haut jusqu'à une hauteur Z de **4 mm**. Tu peux aussi cliquer sur la valeur affichée et la taper directement au clavier.

Tu as maintenant la plaque + le texte qui dépasse de 4 mm. Ton porte-clés commence à ressembler à quelque chose.

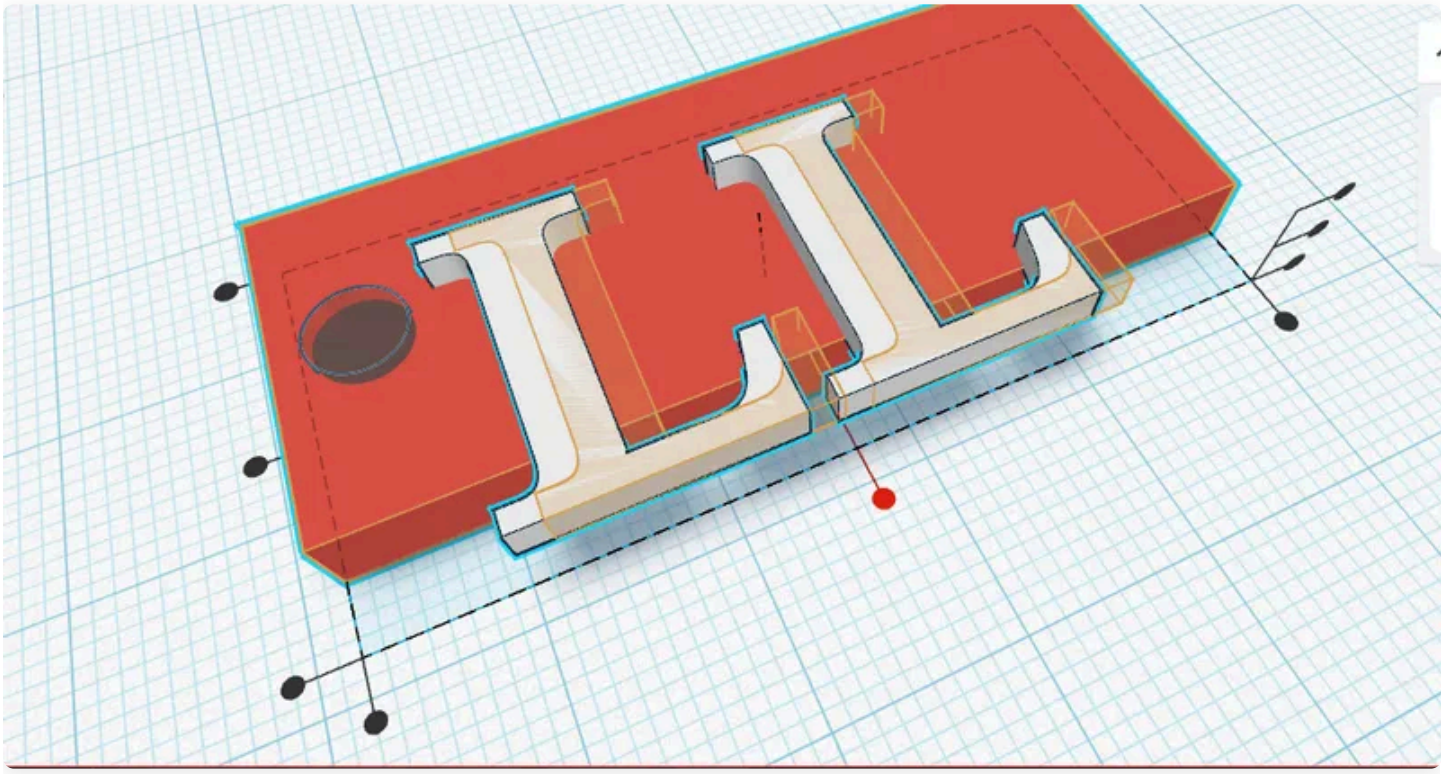
Partie 4 — Aligner deux objets

À l'œil, ton texte n'est probablement pas parfaitement centré sur la plaque. TinkerCad a un outil dédié : **Aligner**.

1. Sélectionne **les deux objets** (la plaque ET le texte). Pour ça : clique le premier, puis Shift + clic sur le second. Ou trace un rectangle de sélection autour des deux.
2. Clique sur le bouton **Aligner** en haut à droite (ou appuie sur **L**).



Des petites poignées noires apparaissent autour de la sélection : **3 par direction** (début, milieu, fin), pour chacun des axes X, Y et Z.



Choisir l'objet qui ne bouge pas

Par défaut, l'alignement utilise un **point central de la sélection** : les deux objets vont se rapprocher l'un de l'autre. Souvent, ce n'est pas ce que tu veux — tu veux que la **plaque reste à sa place** et que **le texte vienne se centrer dessus**.

Pour ça : pendant que tu es en mode Aligner, **clique sur la plaque**. Elle est désignée comme **objet de référence** (un petit indicateur visuel apparaît dessus) : c'est elle qui ne bougera plus, et les autres objets s'aligneront par rapport à elle.

Centrer le texte sur la plaque

Avec la plaque désignée comme référence :

3. Clique sur la poignée du **milieu** dans le sens de la **largeur** (axe X, de gauche à droite) pour centrer le texte horizontalement.
4. Clique sur la poignée du **milieu** dans le sens de la **profondeur** (axe Y, d'avant en arrière) pour centrer le texte de l'autre côté.

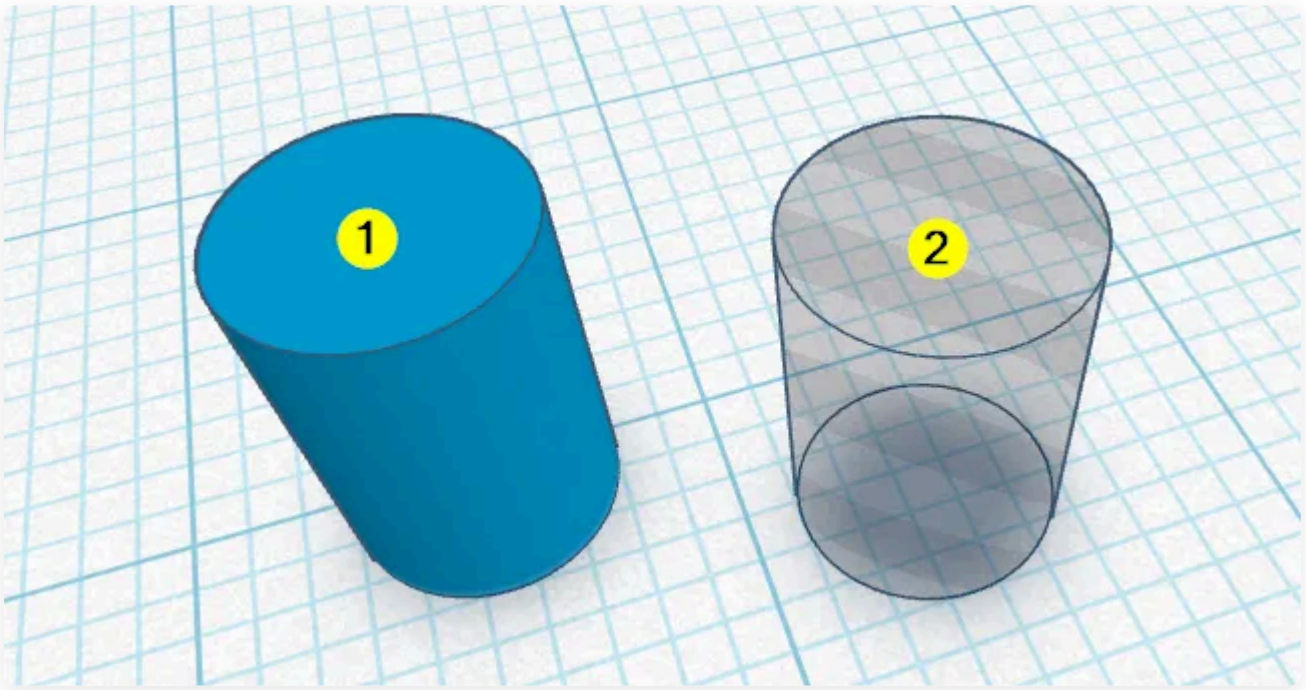
Le texte est maintenant parfaitement centré sur la plaque. Appuie sur **Échap** ou clique ailleurs pour sortir du mode Aligner.

> 🧠 ***À retenir :*** *la fonction Aligner agit sur* ***tous les objets sélectionnés en même temps****. C'est l'outil le plus utile pour avoir un travail propre — utilise-le systématiquement à la fin de chaque étape, en désignant bien l'objet de référence par un clic.*

Partie 5 — Percer : le trou pour l'anneau

Un porte-clés sans trou pour l'anneau de clés, ça ne sert à rien. C'est ici qu'on découvre une mécanique importante de TinkerCad : les **perçages**.

Solide vs perçage



Dans TinkerCad, chaque forme est soit :

TYPE	APPARENCE	EFFET
Solide [1]	Couleur pleine	Ajoute de la matière
Perçage [2]	Hachuré, semi-transparent	Enlève de la matière à ce qui se trouve dessous

Un perçage **n'est pas un trou tout seul** : c'est une forme qui dit "tout ce que je touche, tu l'effaces". Pour que le trou existe vraiment, il faut **regrouper en union** un solide et un perçage ensemble.

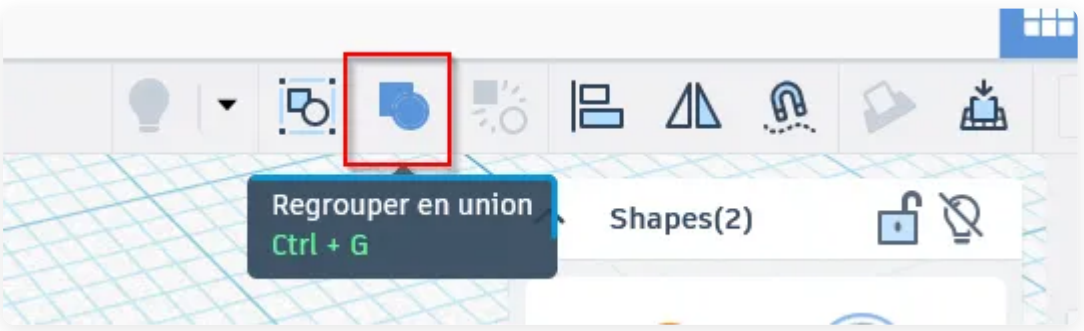
Créer le trou

1. Glisse un **cylindre** depuis la bibliothèque.
2. Redimensionne-le : diamètre **6 mm** (donc 6 mm en X et en Y), hauteur **5 mm** (légèrement plus que la plaque pour bien la traverser).
3. Avec le cylindre sélectionné, dans le petit panneau en haut à droite, clique sur **Perçage** (l'option à côté de "Solide"). Le cylindre devient hachuré.
4. Déplace-le sur la partie **gauche** de la plaque, là où ira l'anneau.
5. Vérifie qu'il **traverse bien** la plaque de haut en bas (la base du cylindre doit être au niveau du sol, ou même en dessous).

Grouper pour faire le trou

Le trou n'apparaîtra pas tant que tu n'auras pas dit à TinkerCad de **combiner** la plaque et le perçage.

1. Sélectionne **la plaque ET le cylindre-perçage** (Shift + clic, ou rectangle de sélection).
2. Clique sur **Regrouper en union** en haut à droite (ou **Ctrl + G**).



Boum : la plaque est désormais percée. Tu peux faire tourner la vue pour vérifier que le trou la traverse vraiment.

> ⚠️ ***Si le trou n'apparaît pas :*** *c'est presque toujours parce que le cylindre est encore réglé en Solide, ou parce qu'il ne traverse pas complètement la plaque. Annule (**Ctrl + Z**), vérifie les deux points, et recommence.*

> 💡 ***Tu peux dissocier*** *avec **Ctrl + Maj + G** si tu t'es trompé. Tant que tu ne quittes pas TinkerCad, tu peux aussi toujours revenir en arrière (**Ctrl + Z**).*

Partie 6 — Finir le porte-clés

À ce stade, tu as :

- une plaque rectangulaire percée à gauche ;
- un texte avec tes initiales posé dessus.

Si tu veux un seul objet qui se tient ensemble, sélectionne **les deux**, puis **Regrouper en lot** (Ctrl + G). Tu obtiens un seul objet, ton porte-clés.

Variantes possibles

- **Texte gravé au lieu de relevé** : passe ton texte en *Perçage* avant de regrouper avec la plaque. Au lieu de sortir, les lettres seront creusées dedans.
- **Coins arrondis** : utilise la forme *Round Roof* ou *Cylindre* pour adoucir les angles de la plaque. Ou change le rayon dans les propriétés de la forme.
- **Forme particulière** : remplace le rectangle par une étoile, un cœur, un logo de classe…

Partie 7 — Exporter pour l'impression

Quand ton modèle te plaît, il faut le sortir au format **STL** — le format universel des fichiers à imprimer en 3D.

1. En haut à droite, clique sur **Exporter**.
2. Choisis **Tout dans la conception** (sinon TinkerCad n'exporte que ce qui est sélectionné).
3. Sélectionne le format **.STL**.
4. Le fichier se télécharge dans ton dossier *Téléchargements*.

Ce fichier `.stl`, c'est ce que ton prof (ou toi) va ouvrir dans un **slicer** pour le préparer à l'impression. On verra ça dans le cours suivant.

> ***Format OBJ*** *: TinkerCad propose aussi l'export* `.obj`*. Pour l'impression 3D, reste sur* ***STL*** *— c'est le standard universel et celui que tous les slicers acceptent sans broncher.*

Exercices

Exercice 1 — Reproduire le porte-clés

Suis pas à pas les 7 parties ci-dessus, sans sauter d'étapes. À la fin, tu dois pouvoir créer :

- une plaque de 50 × 20 × 4 mm ;
- tes initiales en relief (ou gravées) sur le dessus ;
- un trou de 6 mm de diamètre traversant la plaque ;
- un fichier `.stl` exporté.

Exercice 2 — Un trophée empilé

Modélise un **trophée** en empilant quatre formes : un **cylindre**, un **cône** et **deux sphères**. Pour chaque pièce après la première, glisse-la **directement depuis la bibliothèque** sur la surface de la pièce précédente — utilise la technique de l'ombre arrondie pour qu'elle se pose dessus, pas au sol.

Proposition de proportions :

- **Socle** : un **cylindre** plat et large (Ø 40 mm, hauteur 8 mm), posé au sol.
- **Globe** : une **grosse sphère** (Ø 30 mm), posée au centre du socle.
- **Flamme** : un **cône** (Ø 15 mm, hauteur 25 mm), pointe vers le haut, posé sur le globe.
- **Boule finale** : une **petite sphère** (Ø 10 mm), posée sur la pointe du cône.

Utilise l'outil **Aligner** à chaque étape pour que chaque pièce soit bien centrée sur la précédente (sélectionne les deux objets, désigne le bas comme référence, puis clique sur les poignées centrales en X et en Y).

Quand tout est en place et bien aligné, sélectionne le tout et **regroupe en lot** (Ctrl + G) pour obtenir un seul objet.

Exercice 3 — Une boîte vide avec une paroi de 3 mm

Modélise une **boîte** :

- Base : un cube, 50 mm de large, 50 mm de haut.
- Intérieur creusé : un autre cube en *Perçage*, légèrement plus petit (épaisseur de mur de 3 mm) — détermine la taille correcte de ce cube toi-même.
- Aligne les deux, regroupe en union, et tu as une boîte solide avec un fond.

Exercice 4 — Un porte-stylos hexagonal

Modélise un **porte-stylos** :

- Base : un prisme à 6 côtés (forme *Polygon* avec 6 côtés), 50 mm de large, 60 mm de haut.

- Intérieur creusé : un autre prisme à 6 côtés en *Perçage*, légèrement plus petit (épaisseur de mur de 3 mm) — détermine la taille correcte toi-même.
- Aligne les deux, regroupe en union, et tu as un porte-stylos solide avec un fond.

▸ Exercice 5 — Un dé à 6 faces

Modélise un **dé à jouer** classique de 16 × 16 × 16 mm :

- Un cube de 16 mm de côté.
- Sur chaque face, le nombre de points correspondant (1 sur la face du dessus, 6 sur celle du dessous, etc., la somme des faces opposées doit faire 7).
- Les points sont des **petites sphères en perçage** (diamètre 3 mm), creusées dans le cube.

Cet exercice te force à travailler **toutes les faces** d'un objet — pas seulement le dessus. Utilise le ViewCube pour faire pivoter la vue entre chaque face.

✍️ À retenir

- TinkerCad travaille en **millimètres** sur un plan de travail quadrillé — ce que tu vois à l'écran, c'est la taille réelle de l'objet imprimé. Le plan de travail correspond au plateau de l'imprimante.
- Gestes souris : **clic gauche** = manipuler les objets, **molette** = zoom, **clic droit** = orbiter, **Shift + clic droit** (ou clic molette) = déplacer la vue.
- Pour poser un objet sur la surface d'un autre, glisse-le **directement depuis la bibliothèque** au-dessus de cet objet — l'ombre sous le curseur montre la surface de dépôt. Un objet déjà existant ne se posera pas tout seul : il faut le lever avec le **cône noir**.
- Chaque forme est soit **Solide** (ajoute de la matière), soit un **Perçage** (enlève de la matière).
- Un perçage n'a d'effet qu'une fois **regroupé** (Ctrl + G) avec le solide qu'il doit creuser.
- L'outil **Aligner** (L) aligne plusieurs objets entre eux — utilise-le systématiquement.
- On exporte au format **STL** pour passer à l'impression.

Suite

Maintenant que tu sais modéliser un objet, il reste à comprendre **comment l'imprimer**. Dans le prochain cours, on regarde comment fonctionne une imprimante 3D, ce qu'est un slicer, et quels filaments existent pour faire le bon choix.

L'impression 3D — comment ça marche

> *Une vue d'ensemble de l'impression 3D grand public : le processus complet du modèle au filament fondu, les principales technologies d'imprimantes, et les types de filaments que tu vas croiser le plus souvent.*

Tu sais maintenant modéliser un objet dans TinkerCad. Mais entre ton fichier `.stl` et l'objet en plastique qui sort de la machine, il y a plusieurs étapes que la plupart des gens ignorent. Ce cours met les choses au clair : ce qu'il se passe vraiment, quelles machines existent, et avec quels matériaux on imprime.

🎯 Objectifs

À la fin de ce cours, tu seras capable de :

1. Décrire les **étapes** entre un fichier 3D et un objet imprimé.
2. Expliquer ce qu'est un **slicer** et à quoi il sert.
3. Distinguer les principales **technologies** d'impression 3D grand public (FDM, résine).
4. Choisir un type de **filament** adapté à un objet donné (jouet, pièce technique, prototype…).

Partie 1 — Le principe : empiler des couches

Toutes les imprimantes 3D grand public reposent sur le même principe : **construire l'objet couche par couche, du bas vers le haut**. Au lieu d'enlever de la matière (comme une fraise ou un tour), on en ajoute petit à petit. C'est ce qu'on appelle la **fabrication additive** (GB ***additive manufacturing***).

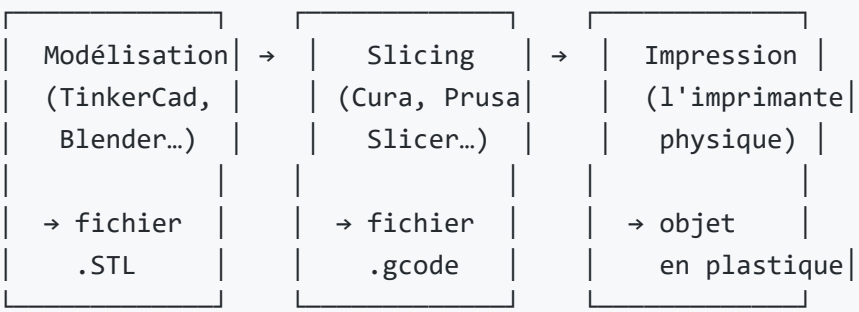
```
Couche 30   ──────────      ← dernière déposée
Couche 29   ──────────────
Couche 28   ──────────────
 ...
Couche 03   ──────────────
Couche 02   ──────────────
Couche 01   ──────────      ← première déposée sur le plateau
            ══════════════
               plateau
```

Chaque couche fait typiquement entre **0,1 mm et 0,3 mm** d'épaisseur. Plus la couche est fine, plus l'objet sera détaillé — mais plus l'impression sera longue. C'est un des compromis classiques de l'impression 3D : **qualité contre temps**.

> 💡 ***Ordre de grandeur :*** *un objet de la taille d'une balle de ping-pong (40 mm de haut) en couches de 0,2 mm = 200 couches. Si chaque couche prend 30 secondes, l'impression dure 1 h 40. Une vraie pièce technique avec des couches fines peut tourner toute la nuit.*

Partie 2 — Le processus complet

Du modèle à l'objet, il y a trois étapes :



Étape 1 — Modéliser

C'est ce que tu as appris à faire dans le cours précédent : dessiner ton objet en 3D dans TinkerCad (ou Blender, Fusion 360, FreeCAD…) et l'**exporter en STL**.

Le fichier STL contient juste **la forme** de l'objet — un maillage de triangles qui décrit la surface. Aucune information sur la couleur, le matériau, ou comment l'imprimer.

Étape 2 — Trancher (slicer)

Le STL ne suffit pas à piloter une imprimante. Il faut maintenant le découper en **tranches horizontales** — c'est le boulot du **slicer**.

Le slicer est un logiciel qui :

1. Prend ton fichier STL en entrée.
2. Te demande **quelle imprimante** tu utilises, **quel filament**, **quelle épaisseur de couche**, etc.
3. Découpe virtuellement l'objet en couches horizontales.
4. Calcule, pour chaque couche, **le chemin exact** que la tête d'impression doit suivre.
5. Génère un fichier **G-code** : la liste des mouvements à faire, dans l'ordre.

Les slicers grand public les plus connus :

SLICER	PARTICULARITÉ
Cura	Le plus populaire, gratuit, développé par Ultimaker
PrusaSlicer	Excellent et gratuit, optimisé pour les imprimantes Prusa mais marche avec d'autres
OrcaSlicer	Récent, dérivé de PrusaSlicer, très utilisé pour les imprimantes Bambu Lab
Bambu Studio	Le slicer maison de Bambu Lab

Dans le slicer, on règle plein de paramètres importants :

- **Épaisseur des couches** (0,1 à 0,3 mm).
- **Remplissage** (GB ***infill***) — l'intérieur de l'objet est creux et rempli d'un motif (souvent en nid d'abeille) à un taux qui va de 10 % à 100 %. Plus le taux est élevé, plus l'objet est solide… et plus l'impression est longue et consomme de filament.
- **Supports** — pour les parties en surplomb (un toit, un bras tendu). Le slicer ajoute des "échafaudages" en plastique qu'on casse à la fin.
- **Vitesse** — combien de mm/s la tête se déplace.
- **Température** — varie selon le filament.

Étape 3 — Imprimer

Une fois le G-code prêt, on l'envoie à l'imprimante (par carte SD, USB, WiFi…). La machine exécute les mouvements un par un, dépose le plastique fondu, et après quelques minutes ou quelques heures, l'objet est terminé.

> 🧠 ***À retenir :*** *un slicer ne **dessine pas** : il **traduit** ton modèle 3D en instructions machine. C'est le pont obligatoire entre le monde 3D et le monde physique. Aucune imprimante grand public ne sait lire un STL directement.*

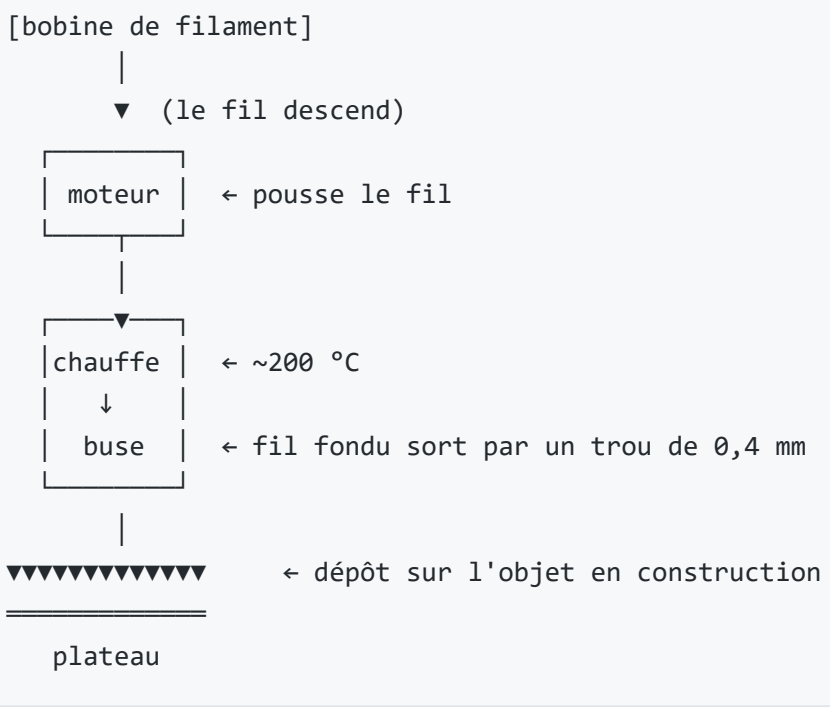
Partie 3 — Les types d'imprimantes 3D grand public

Il existe plusieurs technologies, mais deux dominent largement le grand public.

FDM — la plus répandue

FDM = *Fused Deposition Modeling* (dépôt de fil fondu). C'est la technologie de **95 %** des imprimantes 3D que tu croiseras à l'école, à la maison, au FabLab.

Principe : un **fil de plastique** est tiré depuis une bobine, poussé dans une tête chauffée (la **buse** ou GB ***nozzle***), fondu autour de 200 °C, et déposé sur le plateau couche par couche.



Avantages :

- Bon marché (machines à partir de 200-300 €, filament autour de 20 €/kg).
- Simple à utiliser et à entretenir.
- Bonne variété de filaments.

Limites :

- Les couches sont **visibles** à l'œil — surface jamais parfaitement lisse.
- Précision limitée (la buse fait 0,4 mm, on ne fera pas de bijou ultra détaillé avec).

Exemples de marques : Creality (Ender, K1), Prusa, Bambu Lab, Anycubic, Voron (à monter soi-même).

▸ Résine (SLA / MSLA / DLP)

Une autre famille, plus chère mais plus précise.

Principe : un bac contient de la **résine liquide** photosensible. Une **lumière UV** (laser ou écran LCD) durcit la résine couche par couche. Le plateau remonte au fur et à mesure, tirant l'objet hors du bac.

Avantages :

- Précision **bien supérieure** au FDM (couches autour de 0,025 mm, soit 8× plus fines).
- Surfaces très lisses, idéales pour des figurines, des bijoux, des prototypes détaillés.

Limites :

- Plus cher : machine et résine.
- La résine liquide est **toxique** avant durcissement : gants, lunettes, ventilation obligatoires.
- Étapes de post-traitement : lavage à l'alcool, durcissement final aux UV.
- Inadapté aux grandes pièces mécaniques (résine plus cassante).

Exemples de marques : Anycubic Photon, Elegoo Mars, Phrozen, Formlabs.

Comparatif rapide

CRITÈRE	FDM	RÉSINE (SLA/MSLA)
Matériau	Fil plastique en bobine	Liquide photosensible
Précision typique	0,1–0,3 mm par couche	0,01–0,05 mm par couche
Coût machine	200–800 €	250–600 €
Coût matière (kg)	~20 €/kg	~50 €/kg
Vitesse	Moyenne à rapide	Moyenne
Sécurité	Buse chaude, mais peu de risque	Résine toxique : gants/ventilation
Idéal pour	Pièces fonctionnelles, prototypes, gros volumes	Figurines, bijoux, détails fins

À l'école, c'est presque toujours du **FDM**. C'est ce qu'on utilisera ici.

Partie 4 — Les principaux filaments FDM

Quand tu achètes une bobine, tu te retrouves face à une jungle de sigles. Voici les quatre filaments que tu vas croiser le plus souvent.

PLA — le filament de base

PLA = *Polylactic Acid* (acide polylactique). C'est le filament **par défaut** quand on débute. Fabriqué à partir d'amidon de maïs, il est en partie **biodégradable**.

- **Température d'impression :** ~200 °C.
- **Plateau :** chauffé à 50-60 °C (ou pas chauffé du tout sur certaines machines).
- **Rigidité :** bonne mais cassant — il se fend plus qu'il ne plie.
- **Tenue à la chaleur :** mauvaise — déforme à partir de 50-60 °C (donc à éviter pour un objet laissé dans une voiture en été).
- **Odeur à l'impression :** légère, plutôt agréable (un peu sucrée).

Idéal pour : prototypes, déco, jouets, objets d'usage normal à l'intérieur. **C'est ce qu'on utilise à l'école.**

PETG — le polyvalent

PETG = *Polyethylene Terephthalate Glycol*. Le grand cousin du plastique des bouteilles d'eau (PET) avec un G ajouté qui le rend imprimable.

- **Température d'impression :** ~230 °C.
- **Rigidité :** souple, moins cassant que le PLA.
- **Tenue à la chaleur :** meilleure que le PLA (jusqu'à ~70-80 °C).
- **Résistance à l'humidité :** bonne — utilisable en extérieur.

Idéal pour : pièces qui prennent des chocs (clip de fixation, bras articulé), contenants alimentaires (sous certaines conditions), objets exposés au soleil.

ABS — le technique

ABS = *Acrylonitrile Butadiene Styrene*. Le plastique des Lego, du tableau de bord de ta voiture, des coques d'électroménager.

- **Température d'impression :** ~240-260 °C.
- **Plateau :** chauffé à 100 °C minimum, **enceinte fermée** recommandée.
- **Rigidité :** très solide et résistant aux chocs.
- **Tenue à la chaleur :** excellente (jusqu'à ~100 °C).
- **Inconvénient majeur : odeur très désagréable et nocive** à l'impression. À utiliser uniquement dans une pièce ventilée ou avec une imprimante en enceinte fermée.

Idéal pour : pièces mécaniques, pièces exposées à la chaleur, pièces qui doivent durer.

TPU — le souple

TPU = *Thermoplastic Polyurethane*. Un filament **flexible** — l'objet imprimé se plie comme du caoutchouc.

- **Température d'impression :** ~220 °C.
- **Souplesse :** variable selon le grade (de la semelle de chaussure au pneu).
- **Difficulté :** délicat à imprimer (le fil souple a tendance à s'emmêler dans l'extrudeur).

Idéal pour : coques de téléphone, joints, semelles, jouets antichoc, pneus de petits véhicules.

Comparatif rapide

FILAMENT	TEMPÉRATURE	SOLIDITÉ	CHALEUR	TOXICITÉ ODEUR	NIVEAU
PLA	200 °C	Cassant	Faible	Faible	Débutant
PETG	230 °C	Souple	Bonne	Faible	Intermédiaire
ABS	250 °C	Très bon	Excellent	Forte	Avancé
TPU	220 °C	Flexible	Moyenne	Faible	Avancé

> 🧠 ***À retenir :*** *à l'école et pour 95 % de tes premiers projets, c'est **PLA**. Il pardonne tout, sent presque rien, et donne de beaux résultats. Tu changeras de filament quand un projet précis le demandera.*

Partie 5 — Quelques pièges classiques

Avant de te lancer, sache que l'impression 3D **rate parfois**. C'est normal. Les ratés les plus fréquents :

- **L'objet ne tient pas au plateau** → la première couche s'arrache. Cause : plateau pas propre, mal nivelé, ou trop loin de la buse. Solution : on règle le **nivellement** (manuel ou automatique) et on nettoie le plateau à l'alcool.

- **Spaghetti monstrueux** → l'imprimante continue d'extruder dans le vide après que l'objet s'est décollé. Toujours rester à proximité au début d'une impression.
- **Décalage de couches** → toutes les couches au-dessus d'un certain niveau sont décalées de quelques mm. Cause : un choc, ou une courroie détendue.
- **Surplombs ratés** (GB ***overhangs***) → des parties en l'air s'affaissent. Solution : ajouter des **supports** dans le slicer, ou orienter l'objet autrement avant d'imprimer.
- **Filament humide** → bruits crépitants à l'impression, surface granuleuse. Le filament absorbe l'humidité de l'air. Solution : garder les bobines dans une boîte fermée avec un sachet déshydratant.

Aucun de ces problèmes n'est dramatique : c'est l'apprentissage classique de la machine.

✍️ À retenir

- L'impression 3D grand public dépose la matière **couche par couche**, du bas vers le haut.
- Trois étapes : **modéliser** (TinkerCad → STL), **trancher** (slicer → G-code), **imprimer**.
- Le **slicer** est obligatoire : il traduit un modèle 3D en mouvements machine.
- **FDM** (fil fondu) domine le grand public — bon marché, polyvalent, surfaces visibles en couches.
- **Résine** (SLA/MSLA) est plus précise mais plus chère et plus dangereuse à manipuler.
- Les filaments courants : **PLA** (débutant, déco), **PETG** (extérieur, robuste), **ABS** (technique, chaleur), **TPU** (flexible).
- À l'école : **PLA**, presque toujours.

Suite

Tu as les concepts, tu sais modéliser et tu sais ce que fait une imprimante. Il est temps de tout mettre en œuvre : dans le prochain travail, tu vas **concevoir et faire imprimer ton premier objet**, dans un format compact qu'on peut produire en classe.

Projet : ton premier objet imprimé

> *Tu modélises et fais imprimer un petit objet de ton choix, qui doit tenir dans un cube de 4 × 4 × 4 cm. Objectif : aller du brouillon papier à l'objet physique, en passant par TinkerCad et le slicer.*

Tu as appris à modéliser dans TinkerCad et tu comprends comment fonctionne une imprimante 3D. Place à la pratique : tu vas **concevoir, modéliser et imprimer** un petit objet — quelque chose d'utile, de joli, ou de décoratif, mais qui rentre dans des dimensions imposées.

🎯 Objectifs

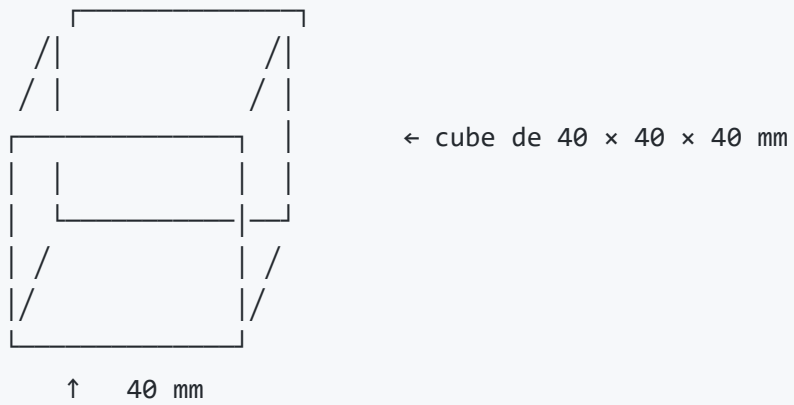
À la fin de ce projet, tu seras capable de :

1. **Concevoir** un objet en partant d'une idée (croquis papier, mesures, contraintes).
2. **Modéliser** cet objet dans TinkerCad en respectant des dimensions précises.
3. **Préparer** ton fichier pour l'impression (export STL, vérification).
4. **Présenter** ton objet imprimé et expliquer tes choix de conception.

📋 Contraintes du projet

Dimensions

Ton objet doit tenir **entièrement** dans un cube imaginaire de **4 × 4 × 4 cm** (donc 40 × 40 × 40 mm dans TinkerCad).



Pas de pièce qui dépasse, même un peu. La raison est très concrète : on imprime plusieurs projets en même temps, et chacun doit rester dans son carré.

Temps d'impression

Pour que tout le monde passe, ton objet doit **s'imprimer en moins d'une heure** sur l'imprimante de la classe. Le slicer t'annonce ce temps quand tu ouvres ton fichier — il faudra le vérifier avant la mise en file.

Si l'estimation dépasse l'heure, tu peux jouer sur :

- **Baisser le remplissage** (GB ***infill***) — par exemple 10 % au lieu de 20 %.
- **Augmenter l'épaisseur de couches** — passer de 0,2 mm à 0,25 ou 0,3 mm.
- **Simplifier ton modèle** — supprimer un détail qui prend du temps sans rien apporter.

Filament

Filament imposé : **PLA**. Tu choisis la couleur parmi ce qui est en stock le jour de l'impression.

🧠 Quelques pistes d'objets

Tu choisis librement, mais voici des idées pour démarrer si tu manques d'inspiration :

IDÉE	POURQUOI C'EST INTÉRESSANT
Porte-clés personnalisé	Tu reprends le cours TinkerCad et tu le personnalises.
Pion de jeu de société	Cylindre, sphère, perçage — toutes les bases.
Mini-pot pour cactus	Cylindre + perçage intérieur — utile et joli.
Support pour téléphone (mini)	Plus difficile : géométrie en équilibre, angles.
Pièce de réparation	Un clip cassé, un crochet, une cale — résoudre un vrai problème.
Dé personnalisé	6 faces gravées au choix (initiales, logos, symboles).
Médaillon ou pendentif	Plat, joli, traversé par un anneau.
Logo en relief	Le logo de ta série, de l'école, d'un groupe — utilisable comme aimant de frigo.
Range-câbles	Petit clip qui se fixe sur un bureau pour tenir un câble USB.

> 💡 ***Astuce :*** *un bon projet de débutant a* ***peu de surplombs*** *(peu de parties en l'air qui auraient besoin de supports). Plus la pièce ressemble à une "pyramide vue d'en haut" (large en bas, plus fin en haut), plus elle s'imprime sans soucis.*

📝 Étapes du projet

Étape 1 — Croquis papier (10 min)

Avant de toucher à TinkerCad, dessine ton idée **sur papier**.

- Trois vues : **dessus**, **face**, **côté**.
- Note les **dimensions principales** (en mm) à côté de chaque côté important.
- Vérifie que tout rentre dans 40 × 40 × 40 mm.

Cette étape semble inutile — elle ne l'est pas. Modéliser sans plan, c'est passer 30 minutes à bricoler des cubes en se demandant à quoi on voulait arriver. Avec un croquis devant les yeux, la modélisation prend 10 minutes.

▸ Étape 2 — Modélisation dans TinkerCad

Crée ton modèle en suivant ton croquis. Pendant le travail :

- Active la **règle** dans TinkerCad (icône Règle, en haut à droite) pour vérifier tes dimensions à tout moment.
- Utilise systématiquement l'outil **Aligner** pour que rien ne soit "à peu près" centré.
- Regroupe les éléments par étapes — pas tout d'un coup à la fin.

Quand tu penses avoir fini, **fais tourner ta vue** sous tous les angles. Cherche les détails qui ne vont pas : une face qui dépasse, un trou bouché, un texte mal aligné…

▸ Étape 3 — Export et vérification

1. Sélectionne **tout** ton objet (Ctrl + A).
2. **Exporter** → STL → "Tout dans la conception".
3. Nomme le fichier : `<TonNom>-<NomObjet>.stl` (ex. `lambrechts-porte-cles.stl`).

Avant de le rendre, fais une **dernière vérification** : ouvre le STL dans le slicer de la classe (Cura, PrusaSlicer, ou autre).

- Le **temps d'impression** est-il < 1 heure ?
- L'objet **tient-il bien** sur le plateau dans le bon sens ?
- Y a-t-il besoin de **supports** ?

Si quelque chose cloche, retourne dans TinkerCad et corrige.

▸ Étape 4 — Impression

Une fois la file d'attente arrivée à ton tour, le prof lance l'impression. Pendant ce temps, tu observes — c'est instructif. Regarde particulièrement :

- La **première couche** : c'est elle qui décide si toute l'impression va réussir.
- Les **mouvements de la tête** sur les premiers étages : comment la pièce prend forme.

▸ Étape 5 — Présentation

Quand ton objet est imprimé et nettoyé (supports cassés s'il y en a, fil parasite enlevé), tu le présentes en classe en répondant à ces questions :

1. **À quoi sert ton objet ?** (ou : qu'est-ce qu'il représente ?)
2. **Quelle a été la partie la plus difficile à modéliser ?** Comment tu l'as résolue ?
3. **Qu'est-ce que tu changerais** si tu devais le refaire ?
4. **Combien de temps** a duré l'impression ? Avec quelle épaisseur de couche et quel remplissage ?

✅ Critères de réussite

Ton projet est validé si :

- L'objet **tient** dans 40 × 40 × 40 mm.
- L'impression **prend moins d'une heure**.
- Le fichier STL est **propre** (un seul objet bien défini, pas de bouts en l'air ou de faces qui se chevauchent bizarrement).
- L'objet imprimé est **identifiable** — on reconnaît clairement ce que tu voulais faire.
- Tu peux **expliquer tes choix** lors de la présentation.

▸ Grille d'évaluation détaillée

CRITÈRE	POINTS
Croquis papier clair, avec cotes	/3
Respect des dimensions (40 × 40 × 40 mm max)	/3
Qualité de la modélisation (alignement, propreté)	/4
Originalité / pertinence du choix d'objet	/3
Export STL correct et imprimable	/3
Présentation orale (clarté, recul sur le travail)	/4
Total	**/20**

🏆 Pour aller plus loin

Si tu as fini avant les autres :

- **Personnaliser** : ajoute du texte, un logo, un détail qui te ressemble.
- **Imprimer en plusieurs couleurs** : prévois deux pièces séparées (par exemple un anneau et un texte) qu'on peut imprimer dans deux couleurs et emboîter.
- **Optimiser** : essaie de descendre sous **30 minutes** d'impression sans sacrifier la solidité.
- **Tester une variante** : refais ton objet avec une autre orientation sur le plateau — la qualité change-t-elle ?
- **Préparer un second objet** différent du premier (autre forme, autre usage). Le meilleur des deux sera évalué.

⚠️ Quelques rappels avant de te lancer

> ⚠️ ***L'imprimante chauffe à 200 °C.*** *Ne touche **jamais** la buse pendant ou après une impression. Le plateau peut aussi être chaud — attends qu'il refroidisse avant de décoller ta pièce.*

> ⚠️ ***Reste réaliste sur la durée.*** *Si ton premier slicing annonce 4 h, tu n'as pas une heure de marge — tu as 3 h de trop. Simplifie maintenant, pas le jour J.*

> 💡 ***Le STL ne sait pas mentir.*** *Si l'objet rate dans le slicer (faces fantômes, "non-manifold"), c'est qu'il y a un problème dans TinkerCad : deux solides qui se touchent juste à un point, un perçage qui ne traverse pas, etc. Repasse en revue les regroupements.*

Suite

Une fois ton premier objet en main, tu auras vraiment fait le tour : du dessin papier au plastique fini. Pour la suite, tu pourras t'attaquer à des objets plus complexes — assemblages de plusieurs pièces, mécaniques (charnières, engrenages), ou pièces fonctionnelles qui remplacent un objet cassé chez toi.

GESTION DE PROJETS

Introduction à la Gestion de Projet

La **gestion de projet** est l'ensemble des méthodes, outils et pratiques permettant de **planifier, organiser et piloter un projet** afin d'atteindre des objectifs précis dans un délai et avec des ressources définies. Un projet se distingue par son caractère **temporaire** (il a un début et une fin) et par sa **finalité unique** (il vise à produire un livrable spécifique, comme un logiciel, une application ou un service).

En informatique, la gestion de projet est essentielle pour **structurer le développement de logiciels, la mise en place d'infrastructures ou la conception de systèmes complexes**. Sans une gestion efficace, les projets risquent d'être mal organisés, de dépasser les délais ou le budget, ou encore de ne pas répondre aux attentes des utilisateurs. Que ce soit avec des approches **classiques** (en cascade) ou **agiles**, bien gérer un projet informatique permet d'optimiser le travail des équipes, d'anticiper les risques et d'assurer la réussite du produit final. 🚀

Le carré des contraintes

> *Au-delà du triangle d'or : le modèle à quatre sommets de Jurgen Appelo pour piloter un projet.*

Pourquoi ce cours ?

Dans n'importe quel projet en équipe, tu vas vivre la même tension : trop à faire, pas assez de temps, une équipe imparfaite, et un client (ou un prof, ou un public) qui veut quand même un résultat de qualité.

Ce cours te donne un **outil mental** pour rendre ces tensions visibles, en parler, et faire des choix lucides. Cet outil s'appelle le **carré des contraintes**, proposé par Jurgen Appelo dans *Management 3.0* (2010) en évolution du classique **triangle d'or**.

Principe fondamental

> ***Si un sommet varie dans un sens, soit les 2 sommets adjacents varient dans le même sens, soit le sommet opposé varie dans le sens inverse.***

C'est la règle unique qui régit toute la mécanique du modèle. Le reste du cours en dérive.

Objectifs d'apprentissage

À la fin de cette séquence, tu seras capable de :

1. **Identifier** les quatre sommets du carré des contraintes et donner leur équivalent en anglais et en français.
2. **Expliquer** le principe fondamental d'arbitrage : adjacents même sens / opposé sens inverse.
3. **Décomposer** la notion de ressources entre les outils et la capacité humaine (formule à quatre facteurs).
4. **Analyser** un projet en identifiant le sommet modifié et les sommets compensateurs.
5. **Argumenter** un choix d'arbitrage dans une réunion d'équipe agile en s'appuyant sur le carré.

Cinq notions-clés à retenir

#	NOTION	IDÉE CENTRALE
1	**Triangle d'or**	Le modèle classique : Scope, Time, Cost. Critiqué par Appelo car la qualité y est implicite.
2	**Carré des contraintes**	Quatre sommets explicites : **Scope, Time, Quality, Resources**.
3	**Adjacents vs opposé**	Si je modifie un sommet, soit les deux adjacents bougent dans le même sens, soit l'opposé bouge en sens inverse.
4	**Ressources = outils × humains**	Une équipe sans outils ne produit pas, des outils sans équipe non plus. La capacité humaine, elle, suit la formule `N × C × M × T` — multiplicative.
5	**Pas d'arbitrage gratuit**	Aucun projet ne maximise les quatre sommets en même temps. Le rôle du chef de projet est de rendre les arbitrages **explicites et négociés**.

---

1. Du triangle au carré

Le triangle d'or classique (Iron Triangle)

Depuis les années 1960, la gestion de projet se résume souvent à trois contraintes :

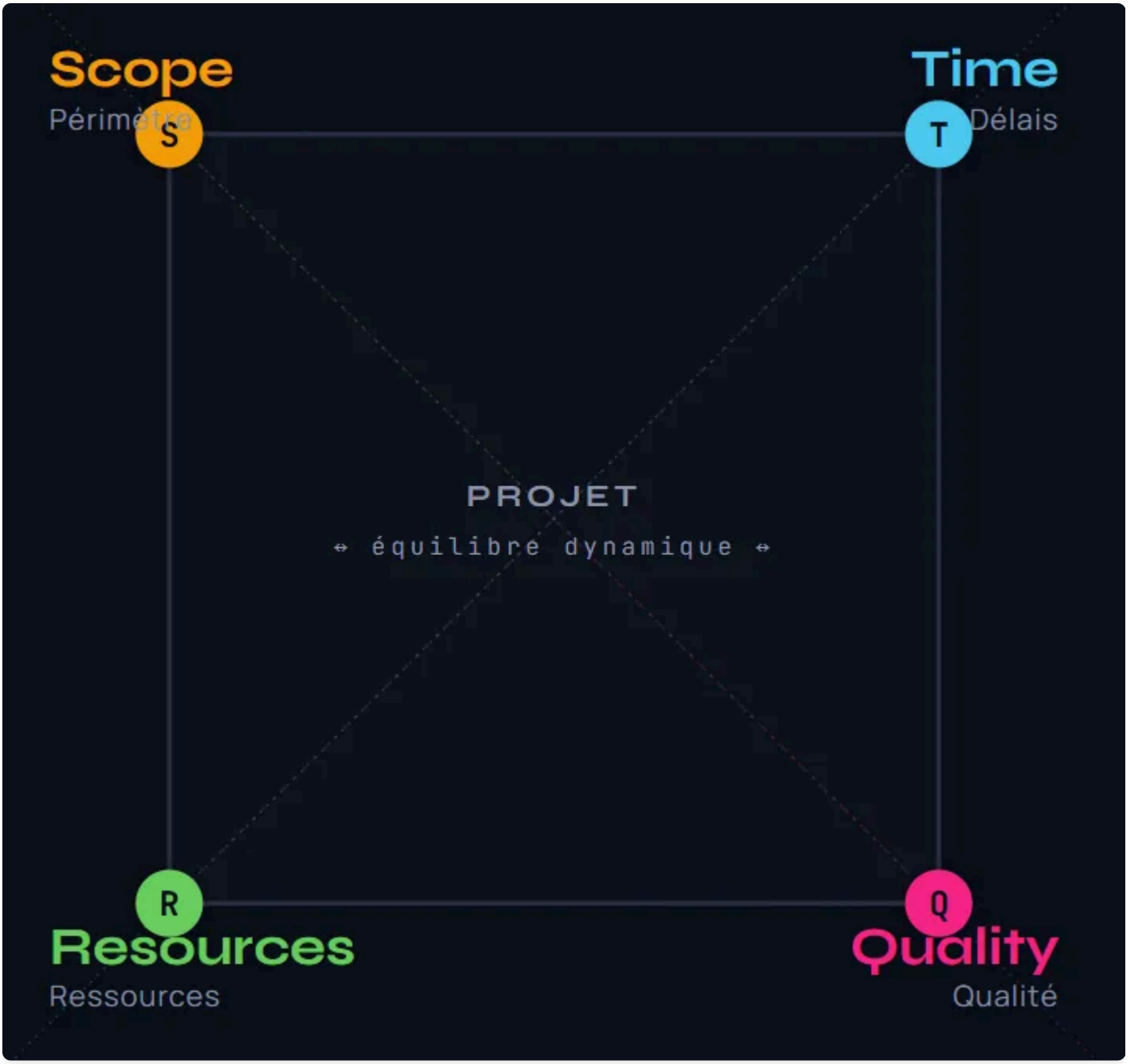
- **Scope** (périmètre) — ce qu'on doit livrer
- **Time** (délai) — combien de temps on a
- **Cost** (coût) — combien ça coûte

> *« Vite, pas cher, beaucoup de fonctionnalités — choisis-en deux. »*

C'est élégant, mais Appelo trouve que **la qualité y est traitée comme un sous-produit silencieux**. Elle se dégrade quand on serre les trois autres, mais personne n'en parle ouvertement.

Le carré d'Appelo

Appelo propose d'ajouter explicitement la **Qualité** comme quatrième sommet :



Voisinages :

SOMMET	ADJACENTS (CÔTÉ DU CARRÉ)	OPPOSÉ (DIAGONALE)
Scope	Time, Resources	Quality
Time	Scope, Quality	Resources
Quality	Time, Resources	Scope
Resources	Scope, Quality	Time

Cette topologie est essentielle : c'est elle qui détermine *qui* doit bouger *quand* tu modifies un sommet.

2. Les quatre sommets en détail

↖ Sommet haut-gauche : Scope *(Périmètre)*

Définition. L'ensemble des fonctionnalités, livrables et exigences qui définissent **ce que** le projet doit produire. C'est la *frontière* du projet : ce qui est inclus, ce qui est exclu.

Vocabulaire.

- GB *Scope, requirements, features, deliverables*
- FR Périmètre, exigences, fonctionnalités, livrables

Exemple. Un site de réservation avec catalogue, calendrier, paiement en ligne, comptes utilisateurs, scan QR et back-office → **scope large**. Le même site avec uniquement le catalogue et le calendrier → **scope réduit**.

Piège classique. Le **scope creep** : le périmètre grossit en douce au fil du projet (« on pourrait juste ajouter… »). Sans arbitrage, c'est le délai, la qualité ou les ressources qui paient l'addition.

↗ Sommet haut-droite : Time *(Délais)*

Définition. La durée totale disponible pour livrer le projet. Inclut la **date de fin** (deadline), les **jalons** intermédiaires, et le **rythme** de travail.

Vocabulaire.

- GB *Time, schedule, deadline, milestones*
- FR Délais, temps imparti, échéance, jalons

Exemple. Une soirée d'entreprise est prévue le 14 février. Cette date **ne bouge pas**. Si le client demande une nouvelle catégorie de questions trois semaines avant, c'est le scope ou la qualité qui devra s'ajuster — jamais le délai.

Particularité. Le temps est souvent **la contrainte la plus rigide** : un salon, une rentrée scolaire, une obligation légale, un lancement marketing — ces dates sont externes au projet et ne se négocient pas.

↘ Sommet bas-droite : Quality *(Qualité)*

Définition. Le niveau d'exigence du livrable : robustesse du code, ergonomie, accessibilité, performance, sécurité, design, absence de bugs, conformité aux normes.

Vocabulaire.

- GB *Quality, robustness, usability, performance*
- FR Qualité, robustesse, ergonomie, performance

Exemple. Une app médicale doit être validée **sans bug critique** — la qualité est imposée par la loi. Un prototype présenté en interne peut tolérer des bugs mineurs : on ajuste après.

Le piège de la qualité sacrifiée. Quand on dit *« on livrera vite, on corrigera après »*, on crée de la **dette technique** : du code mal écrit qu'il faudra payer (avec intérêts) plus tard. Appelo insiste : rendre la qualité explicite, c'est la rendre négociable **en conscience**, pas par défaut.

↙ Sommet bas-gauche : Resources *(Ressources)*

Définition. Tout ce qui est mobilisé pour réaliser le projet : équipe humaine, outils, matériel, budget, locaux.

Vocabulaire.

- GB *Resources, people, tools, capacity, budget*
- FR Ressources, équipe, outils, capacité, budget

C'est le sommet **le plus mal compris** — et c'est sur lui qu'on va passer un peu plus de temps.

---

3. Zoom — Décomposer les ressources

> ⚠️ ***Note importante.*** *Ce qui suit est un* ***enrichissement personnel*** *du modèle d'Appelo,* ***pas la version d'origine****. Dans Management 3.0, Appelo traite les ressources de manière assez globale — équipe et budget réunis. La décomposition ci-dessous est une grille de lecture plus fine pour mieux piloter ce qu'il y a derrière la simple étiquette « ressources ».*

Deux grandes familles

Pour un projet, les ressources se regroupent en deux grandes familles :

1. **Les outils** (`O`) — tout ce qui n'est pas humain mais qui permet de produire : matériel (ordinateurs, serveurs, hardware spécialisé), logiciels (IDE, suites de design, outils de build), infrastructure (cloud, CI/CD, bases de données), locaux, licences, accès aux specs et à la documentation, budget.
2. **La capacité humaine** (`H`) — ce que l'équipe peut réellement produire, qui dépend de quatre facteurs détaillés ci-dessous.

Les deux dimensions se combinent de manière **multiplicative** :

> $R = O \times H$

Pourquoi multiplicatif ? Parce qu'une équipe top sans serveur, sans IDE, sans accès aux specs → bloquée. Et des outils flambant neufs sans personne pour les utiliser → rien ne sort. Si l'une des deux dimensions tend vers zéro, l'ensemble tend vers zéro.

La capacité humaine : quatre facteurs

> $H = N \times C \times M \times T$

Où :

- **N — Nombre de personnes** dans l'équipe (de 1 à n)
- **C — Compétence** : ce qu'elles savent faire dans le domaine du projet
- **M — Motivation** : leur envie réelle d'avancer sur *ce* projet
- **T — Temps disponible** : la part de leur temps de travail qu'elles peuvent vraiment y consacrer

⚠️ Une remarque essentielle sur les pourcentages

Dans les exemples qui suivent, on utilise des pourcentages (« compétence à 95 % », « motivation à 0 % »…). **Ces chiffres ne sont pas des mesures absolues.** Personne ne calcule scientifiquement « 78 % de motivation » ou « 42 % de compétence » — c'est impossible.

Les pourcentages sont là **uniquement pour faciliter la compréhension** : ils permettent de visualiser intuitivement les ordres de grandeur et l'effet multiplicatif. En pratique, trois niveaux suffisent souvent — *élevé / moyen / faible* — voire juste la question *« est-ce qu'il manque quelque chose ? »*.

L'important n'est **pas la précision du chiffre**. C'est de reconnaître que **tous les facteurs jouent en même temps**, et que **tout facteur très bas fait chuter l'ensemble**.

Pourquoi c'est *multiplicatif* et pas *additif*

C'est la clé du modèle. Une multiplication a une propriété particulière : **si un seul facteur est à zéro, le résultat est zéro**, peu importe les autres.

Regardons quatre cas pour s'en convaincre :

CAS 1 — LE DÉVELOPPEUR EXPERT MAIS DÉMOTIVÉ

> *1 personne · compétence très élevée ·* ***motivation nulle*** *· temps total disponible →* `H ≈ 1 × 0,95 × 0 × 1 = 0`

Le projet n'avance pas. Aucune productivité. La compétence sans motivation = zéro. C'est **la grande leçon d'Appelo** : la motivation n'est pas un « bonus » qu'on ajoute, c'est un **facteur multiplicatif**. Sans elle, tout le reste est neutralisé.

CAS 2 — L'ÉQUIPE MOTIVÉE MAIS SANS COMPÉTENCE

> *5 personnes ·* ***compétence très faible*** *· motivation maximale · temps total disponible →* `H ≈ 5 × 0,05 × 1 × 1 = 0,25`

L'enthousiasme ne remplace pas le savoir-faire. Une équipe ultra-motivée mais qui découvre une techno la veille du sprint ne livrera pas une plateforme robuste.

CAS 3 — L'EXPERT SANS TEMPS DISPONIBLE

> *1 personne · compétence maximale · motivation maximale ·* ***temps disponible très faible*** *(autres prioritaires) →* `H ≈ 1 × 1 × 1 × 0,05 = 0,05`

Un excellent profil à quelques % de son temps fournit la même chose qu'un débutant à quelques % : presque rien. Le temps disponible est crucial — et souvent surestimé.

CAS 4 — L'ÉQUIPE ÉQUILIBRÉE

> *5 personnes · compétence assez bonne · motivation bonne · temps partiel →* `H ≈ 5 × 0,7 × 0,8 × 0,6 ≈ 1,7`

Soit environ **17 %** d'un maximum théorique de 10 personnes à plein régime sur tous les facteurs. C'est une équipe **correcte et réaliste** : on ne travaille jamais à fond sur tous les axes en même temps.

Comment évaluer ta propre équipe

Avant chaque sprint, pose-toi (en équipe) les quatre questions :

1. **N —** Sommes-nous combien à vraiment travailler dessus ? (Distinguer les présents des actifs.)
2. **C —** Maîtrisons-nous les technos nécessaires, ou faut-il apprendre en chemin ?
3. **M —** Est-ce que tout le monde *a envie* d'avancer ? Y a-t-il un sujet bloquant ?
4. **T —** Combien d'heures réelles peut-on y consacrer cette semaine ?

Et n'oublie pas la première moitié de la formule : **a-t-on les outils nécessaires ?** Sans accès au serveur de test, sans la bonne licence logiciel, sans la doc à jour, même la meilleure équipe sera bloquée.

Si l'un des facteurs (outils inclus) est très bas → c'est lui qu'il faut adresser **en priorité** avant d'ajouter du scope.

---

4. La mécanique d'arbitrage

C'est le **cœur du modèle**. Pour chaque situation, on applique le principe fondamental : **soit** les deux adjacents bougent dans le même sens (🟢 vert), **soit** l'opposé bouge en sens inverse (🔴 rouge).

Situation A — Le périmètre augmente (Scope ↑)

> *Un client ajoute une fonctionnalité en cours de projet.*

🟢 **Soit les deux adjacents augmentent :**

- ↑ **Time** — il faut plus de temps
- ↑ **Resources** — il faut plus de ressources

🔴 **Soit l'opposé diminue :**

- ↓ **Quality** — la qualité baisse en compensation

Situation B — Le délai se raccourcit (Time ↓)

> *La direction avance la deadline de trois semaines.*

🟢 **Soit les deux adjacents diminuent :**

- ↓ **Scope** — on réduit les fonctionnalités
- ↓ **Quality** — on accepte une qualité moindre

🔴 **Soit l'opposé augmente :**

- ↑ **Resources** — on ajoute des renforts

▸ Situation C — Les ressources diminuent (Resources ↓)

> *Un développeur quitte l'équipe à mi-projet.*

🟢 **Soit les deux adjacents diminuent :**

- ↓ **Scope** — on coupe des features
- ↓ **Quality** — la qualité chute

🔴 **Soit l'opposé augmente :**

- ↑ **Time** — on allonge le délai

▸ Situation D — La qualité doit monter (Quality ↑)

> *Audit qualité imposé, ou cible plus exigeante.*

🟢 **Soit les deux adjacents augmentent :**

- ↑ **Time** — il faut plus de temps pour peaufiner
- ↑ **Resources** — plus de relecture, plus de tests

🔴 **Soit l'opposé diminue :**

- ↓ **Scope** — on livre moins, mais mieux

▸ Le rôle du chef d'équipe

À chaque demande de changement, **trois questions à se poser à voix haute** :

1. Quel sommet veut bouger, et dans quel sens ?
2. Qui sont les adjacents, qui est l'opposé ?
3. Quel scénario de compensation choisit-on : **adjacents même sens** ou **opposé sens inverse** ?

L'erreur classique : *« on garde tout pareil et on encaisse ».* Ça n'existe pas. Le carré rend l'arbitrage **explicite** au lieu de le subir en silence.

---

5. Lien avec l'écoresponsabilité numérique

Un sommet souvent oublié mais qui devrait être discuté : **toute fonctionnalité ajoutée a un coût environnemental**.

- Plus de scope → plus de code, plus de serveurs, plus de stockage, plus de bande passante consommée.
- Plus de ressources humaines → plus de déplacements, de hardware, de chauffage de bureaux.
- Plus de qualité (notamment perf) → souvent moins de gaspillage énergétique côté utilisateur.

Quand tu négocies le scope en équipe, pose la question : **cette fonctionnalité justifie-t-elle son empreinte ?** Le carré devient alors aussi un outil d'arbitrage éthique. C'est la posture **Lean & Green** de la gestion de projet moderne.

6. Exercices

Exercice 1 — Application du principe

Pour chaque scénario, indique les **deux scénarios de compensation possibles** selon le principe fondamental :

1. La qualité doit augmenter (audit imposé).
2. Le scope doit diminuer (le client coupe une feature).
3. Les ressources doivent augmenter (recrutement d'un dev).
4. Le délai s'allonge (livraison repoussée).

Réponse attendue par scénario :

- 🟢 **Adjacents :** quels sommets et dans quel sens ?
- 🔴 **Opposé :** quel sommet et dans quel sens ?

Exercice 2 — Diagnostic d'équipe

Évalue la capacité réelle de ton équipe agile pour la semaine prochaine. Pour chaque facteur, contente-toi d'un niveau *élevé / moyen / faible* (pas besoin de pourcentages précis) :

- **Outils** disponibles (matériel, logiciels, accès aux specs)
- **N** — nombre de personnes vraiment actives
- **C** — compétence collective sur les technos requises
- **M** — motivation
- **T** — temps réellement disponible

→ Identifie le **maillon faible** (le facteur le plus bas) et propose **une action concrète** pour le faire remonter.

Exercice 3 — Négociation

Le client demande une fonctionnalité supplémentaire à 10 jours de la livraison. Rédige un message Slack ou e-mail expliquant **les deux options possibles** (adjacents même sens / opposé sens inverse) et **ta recommandation argumentée**.

Exercice 4 — Étude de cas

Choisis un projet réel que tu connais (un projet open source, le développement d'un jeu vidéo connu, un grand chantier public…). Analyse-le à travers le carré :

- Quel sommet a été **préservé à tout prix** ?
- Quel sommet a été **sacrifié** ?
- Quel scénario d'arbitrage a été appliqué (adjacents même sens, ou opposé sens inverse) ?

Pour aller plus loin

- **Jurgen Appelo**, *Management 3.0 — Leading Agile Developers, Developing Agile Leaders*, Addison-Wesley, 2010 (chap. 1 & 4).
- **Jurgen Appelo**, *Managing for Happiness*, Wiley, 2016 (sur la motivation comme facteur multiplicatif).
- **The Mythical Man-Month**, Fred Brooks, 1975 (la fameuse loi : *« ajouter des développeurs à un projet en retard le retarde davantage »* — illustration du modèle multiplicatif).
- **The Phoenix Project**, Gene Kim et al. (sur la dette technique et l'arbitrage qualité dans un contexte DevOps).

« Le triangle vous dit ce que vous pouvez optimiser. Le carré vous rappelle ce que vous sacrifiez. »

Méthodes agiles

> *L'**Agilité** est une approche moderne de gestion de projet qui repose sur la flexibilité, l'adaptabilité et la collaboration. Contrairement aux méthodes traditionnelles, qui suivent un plan rigide du début à la fin, les méthodes **Agiles** permettent d'ajuster le projet en cours de route en fonction des retours et des priorités changeantes.*

Qu'est-ce que l'Agilité en gestion de projet ?

L'**Agilité** est une approche moderne de gestion de projet qui repose sur la flexibilité, l'adaptabilité et la collaboration. Contrairement aux méthodes traditionnelles, qui suivent un plan rigide du début à la fin, les méthodes **Agiles** permettent d'ajuster le projet en cours de route en fonction des retours et des priorités changeantes.

L'Agilité est particulièrement utilisée dans le développement logiciel, mais elle s'étend aujourd'hui à d'autres domaines comme le marketing, la gestion de produit ou encore l'industrie.

Les principes des méthodes Agiles

Les méthodes Agiles sont basées sur le **Manifeste Agile** (2001), qui repose sur **quatre valeurs fondamentales** :

1. **Les individus et leurs interactions** plus que les processus et les outils.
2. **Un logiciel fonctionnel** plus qu'une documentation exhaustive.
3. **La collaboration avec les clients** plus que la négociation contractuelle.
4. **L'adaptation au changement** plus que le suivi d'un plan rigide.

L'Agilité encourage donc une approche plus dynamique et humaine du travail, axée sur la satisfaction du client et l'amélioration continue.

Les principales méthodes Agiles

Plusieurs méthodologies appliquent les principes de l'Agilité. Voici les plus connues :

1 Scrum

- Fonctionne en **sprints** (cycles courts de 1 à 4 semaines).
- Rôles clés : **Scrum Master** (facilitateur), **Product Owner** (vision du produit) et **équipe de développement**.
- Utilise un **tableau Scrum** pour suivre les tâches et une **réunion quotidienne** (Daily Scrum) pour ajuster le travail.

2 Kanban

- Utilise un **tableau Kanban** pour visualiser l'avancement des tâches.

- Favorise un **flux de travail continu**, en limitant le nombre de tâches en cours (**WIP – Work In Progress**).
- Permet une grande flexibilité, car il n'impose pas de cycles rigides.

3️⃣ Extreme Programming (XP)

- Axé sur les **bonnes pratiques du développement logiciel** (tests automatisés, programmation en binôme, intégration continue).
- Vise à améliorer la qualité du code et la réactivité aux changements.

4️⃣ Lean

- Inspiré du **système de production de Toyota**.
- Objectif : **réduire les gaspillages** et optimiser la valeur apportée au client.
- Met l'accent sur l'amélioration continue et la simplification des processus.

Pourquoi utiliser une méthode Agile ?

✅ **Plus de flexibilité** : on peut adapter le projet aux évolutions et aux imprévus.
✅ **Une meilleure collaboration** : les équipes travaillent ensemble et avec les clients.
✅ **Livraison plus rapide** : des petites versions du projet sont livrées régulièrement.
✅ **Amélioration continue** : les erreurs sont corrigées rapidement et les processus sont optimisés au fur et à mesure.

Les méthodes Agiles permettent donc de **répondre plus efficacement aux besoins des clients** tout en favorisant un travail d'équipe plus fluide et motivant. 🚀

Les User Stories

> *Une user story est une description simple, en langage naturel courant, d'un besoin ou d'une attente d'un utilisateur ou d'un client, dans le cadre d'un projet logiciel ou d'un produit.*

Pourquoi les user stories existent

As-tu déjà construit une fonctionnalité pour te rendre compte, trop tard, que **ce n'est pas ce que l'utilisateur voulait** ? C'est précisément ce problème que les **user stories** cherchent à résoudre.

Les user stories sont :

- **petites**
- **simples**
- **puissantes**
- et surtout **centrées sur l'utilisateur**

Elles permettent aux équipes de rester concentrées sur **la valeur réelle pour l'utilisateur**, et non sur ce que l'équipe *suppose* être utile.

Elles fonctionnent :

- en Agile,
- en méthodologie hybride,
- et même dans des projets encore peu structurés.

Définition d'une user story

Une **user story** est :

> *une **courte description**, en langage simple, d'une **fonctionnalité ou capacité**, racontée **du point de vue de l'utilisateur**.*

Son objectif principal est clair :

👉 **s'assurer que ce que l'on construit répond réellement aux besoins des utilisateurs**.

Ce qu'une user story n'est PAS

- ❌ ce n'est pas une spécification complète
- ❌ ce n'est pas une description technique
- ❌ ce n'est pas une liste détaillée de règles

✅ Une user story est **un point de départ pour une discussion**, pas une documentation figée.

Le problème des exigences (requirements)

Dans tout projet, une question revient sans cesse :

> *Comment s'assurer que les personnes qui construisent le produit comprennent réellement ce que les utilisateurs veulent ?*

Les user stories apportent une réponse simple :

- elles favorisent une **communication humaine, claire et centrée sur les besoins**
- elles évitent un "dump" de détails techniques incompréhensibles
- elles **font le lien** entre :
  - ceux qui **ont un besoin**
  - et ceux qui **construisent la solution**

👉 Elles comblent le fossé entre métier et technique.

Le format classique : WHO – WHAT – WHY

Les user stories suivent un format volontairement simple :

```
As a [WHO]
I want / I can [WHAT]
So that [WHY]
```

En français :

> ***En tant que*** *[qui]* ***Je veux / je peux*** *[quoi]* ***Afin de*** *[pourquoi]*

Rôle de chaque partie

- **WHO (Qui)** L'utilisateur ou le rôle concerné → "utilisateur enregistré", "administrateur", "nouvel utilisateur", etc.
- **WHAT (Quoi)** Ce que l'utilisateur veut pouvoir faire → une action, une capacité, un besoin
- **WHY (Pourquoi)** La valeur, le bénéfice, la raison → ce que cela apporte à l'utilisateur

Règle fondamentale

🚫 **Pas de HOW (comment)**

On **ne décrit pas la solution technique**. On décrit **le besoin et la valeur**, pas l'implémentation.

Qui écrit les user stories ?

Réponse courte :

👉 **le Product Owner**

Pourquoi ?

- il est responsable du **backlog**
- il garantit que chaque story est :
  - claire
  - utile
  - alignée avec la vision produit

Réponse complète (la plus importante)

✅ **Les bonnes user stories sont un travail d'équipe**

- n'importe qui peut proposer une story
- développeurs, designers, testeurs doivent participer
- les meilleures stories émergent souvent :
  - d'une discussion
  - d'un tableau blanc
  - d'un "Et si l'utilisateur faisait plutôt ça ?"

👉 Une user story devient vraiment utile **quand elle est enrichie collectivement**.

Les 3 C's des user stories (Ron Jeffries)

Pour bien comprendre ce qu'est *vraiment* une user story, Ron Jeffries a défini les **3 C's** :

1️⃣ Card (la carte)

À l'origine, les user stories étaient écrites sur des **cartes physiques**.

Pourquoi ?

- elles sont **simples**
- manipulables
- regroupables
- jetables

Elles donnent un message important :

👉 **Une user story n'est pas un document permanent**

Même aujourd'hui, dans les outils numériques :

- une story **n'est pas une spécification figée**
- **elle n'est pas le requirement**
- elle est seulement **un pointeur vers le besoin**

▸ 2️⃣ Conversation (la conversation)

C'est **le cœur du processus**.

Chaque user story est :

> ***une promesse d'avoir une conversation avant de développer***

Questions typiques :

- "Peux-tu nous en dire plus ?"
- "À quoi ressemble le succès ?"
- "Et si l'utilisateur fait X ?"
- "Que se passe-t-il dans ce cas ?"

⚠️ Sans cette conversation :

- le Product Owner est tenté d'écrire des specs énormes
- on perd tout l'intérêt des stories

Important :

> *Le fait que le PO "veuille quelque chose" ne signifie pas que l'équipe doit l'implémenter tel quel.*

L'équipe doit :

- questionner
- challenger
- proposer d'autres solutions

👉 **La conversation fait partie intégrante de la story.**

▸ 3️⃣ Confirmation (la confirmation)

La confirmation correspond aux **critères d'acceptation**.

Ils répondent à la question :

> *Comment saura-t-on que la story est terminée, et bien terminée ?*

Exemples :

- "La story doit faire ceci"
- "Elle ne doit pas faire cela"
- "Si X arrive, alors Y doit se produire"

✅ Ces critères :

- émergent naturellement des conversations
- donnent de la clarté
- évitent les malentendus

Exemples de user stories (connexion à un site)

Story de base

> *En tant qu'utilisateur enregistré, je dois me connecter afin d'accéder au système.*

Autres points de vue

- **Mot de passe oublié**

  > *En tant qu'utilisateur distrait, je peux demander un rappel de mot de passe afin de pouvoir me connecter même si je l'ai oublié.*

- **Nouvel utilisateur**

  > *En tant que nouvel utilisateur, je peux créer un compte afin de me connecter et enregistrer mes préférences.*

- **Utilisateur fidèle (expérience positive)**

  > *En tant qu'utilisateur revenant pour la énième fois, je reçois une récompense afin de me sentir reconnu et encouragé à continuer.*

👉 Remarque : La story dit *"nième fois"* sans préciser 3, 5 ou 20. Ce détail sera décidé **plus tard**, au moment de l'implémentation.

Où sont les détails si la story est si courte ?

Une user story est souvent **une seule phrase**.

C'est normal.

- les détails viennent :
  - pendant les conversations
  - via les critères d'acceptation
  - au moment où la story est prête à être développée

✅ La brièveté est une force, pas une faiblesse.

À retenir (synthèse)

- Une user story est **courte, simple, centrée utilisateur**
- Elle décrit **WHO – WHAT – WHY**, jamais le HOW
- Elle est :
  - un **pointeur vers un besoin**
  - une **promesse de conversation**
  - validée par des **critères de confirmation**
- Les 3 C's sont fondamentaux :
  - **Card**
  - **Conversation**
  - **Confirmation**
- Les user stories sont un **outil de collaboration**, pas juste de documentation

---

La méthode Kanban

Qu'est-ce que la méthode Kanban ?

La méthode **Kanban** est une approche de gestion du travail qui permet de visualiser, organiser et optimiser les flux de travail. Elle est particulièrement utilisée dans le domaine du développement logiciel, de la gestion de projet et même dans la gestion de tâches personnelles.

Kanban repose sur l'utilisation d'un **tableau Kanban**, qui est un outil visuel permettant de suivre l'avancement des tâches. L'objectif est d'améliorer l'efficacité et la fluidité du travail en identifiant rapidement les blocages et en limitant le travail en cours (**WIP – Work In Progress**).

Le principe de la méthode Kanban

Kanban repose sur quelques principes clés :

1. **Visualisation du travail** : chaque tâche est représentée sous forme de carte (ou post-it) placée sur un tableau, qui est divisé en plusieurs colonnes représentant les différentes étapes du processus.
2. **Limiter le travail en cours (WIP – Work In Progress)** : il ne faut pas surcharger une étape avec trop de tâches pour éviter les goulots d'étranglement.
3. **Gérer le flux** : l'objectif est de maintenir un flux de travail fluide et d'éviter les blocages.
4. **Rendre explicites les règles du processus** : chaque équipe doit définir clairement ses règles pour garantir une bonne organisation.

5. **Amélioration continue** : Kanban est une méthode évolutive, qui s'adapte aux besoins en identifiant et en corrigeant les inefficacités.

L'objectif de la méthode Kanban

L'objectif principal de Kanban est **d'améliorer la productivité et l'efficacité en rendant le travail plus fluide et transparent**. En utilisant Kanban, une équipe peut :

- Réduire le temps nécessaire pour accomplir une tâche.
- Identifier rapidement les blocages.
- Adapter le processus de travail aux besoins réels.
- Travailler de manière plus collaborative et efficace.

Comment mettre en œuvre Kanban de façon simple ?

1. Créer un tableau Kanban

Le tableau Kanban peut être **physique** (un tableau blanc avec des post-it) ou **numérique** (via des outils comme Trello, Jira, ou Notion). Il est divisé en colonnes qui représentent les différentes étapes du processus de travail.

Un exemple classique de tableau Kanban :

À FAIRE (TO DO)	EN COURS (IN PROGRESS)	TERMINÉ (DONE)
Rédiger le rapport	Corriger un bug	Publier l'article
Créer une maquette	Développer une fonctionnalité	Valider le projet

---

2. Ajouter les tâches sous forme de cartes

Chaque tâche est représentée sous forme d'une **carte** (ou post-it). Une carte contient généralement :

- Un titre clair de la tâche.
- Une description si nécessaire.
- Une personne assignée.
- Une date limite (optionnelle).

---

3. Déplacer les tâches au fur et à mesure de leur avancement

Les tâches avancent progressivement à travers les colonnes du tableau en fonction de leur état d'avancement. Par exemple :

1. Une nouvelle tâche commence dans la colonne **"À faire"**.
2. Lorsqu'une personne commence à travailler dessus, elle la déplace dans **"En cours"**.

3. Une fois la tâche terminée, elle est placée dans "**Terminé**".

▸ 4. Limiter le travail en cours (WIP)

Une règle importante de Kanban est de **limiter le nombre de tâches en cours**. Par exemple, si une équipe fixe une limite de 3 tâches en cours par personne, cela évite la surcharge et permet de mieux se concentrer sur les priorités.

▸ 5. Améliorer continuellement le processus

Kanban encourage une réflexion continue sur l'amélioration du flux de travail. Quelques questions à se poser régulièrement :

- Y a-t-il des blocages fréquents dans une étape ?
- Une colonne est-elle souvent surchargée ?
- Pouvons-nous mieux répartir les tâches ?
- Le tableau reflète-t-il bien notre manière de travailler ?

En ajustant les règles et les limites, l'équipe optimise progressivement son travail.

Conclusion : ce qu'il faut retenir

✅ **Kanban est une méthode visuelle de gestion du travail** qui repose sur un tableau avec des tâches organisées en colonnes.
✅ Son objectif est **d'optimiser le flux de travail et d'améliorer l'efficacité** en limitant le travail en cours.
✅ Il suffit de **créer un tableau**, d'y ajouter des tâches sous forme de cartes et de les faire avancer d'une colonne à l'autre.
✅ La méthode encourage une **amélioration continue** en adaptant les règles aux besoins de l'équipe.

Kanban est une méthode simple mais puissante, idéale pour mieux organiser ton travail et celui de ton équipe ! 🚀