Table des matières

BUREAUTIQUE

PowerPoint

MULTIMÉDIA

Les Images Numériques

- Gestion de la transparence
- Images numériques : matricielles vs vectorielles
- Les formats d'images matriciels
- Compression avec et sans perte
- Les images vectorielles
- Résolution et définition d'une image : différences et implications

PROGRAMMATION EN PYTHON

- Python
- Améliorez Vos Affichages avec le module Rich
- Bien débuter: Python au Lycée - Vidéos
- Python: Les outils pour développer
- Thonny Python
- Thonny Python - Déboguer son code
- Les variables en Python
- 🎯 Python: variables, opérations, print() et input(): Objectifs
- Afficher ses variables avec print
- Les types de variables
- Manipuler les variables en Python
- Manipuler valeurs et variables: exemples guidés 🚀
- Valeurs et variables: à toi de jouer 💪
- Bien nommer ses variables en Python
- Pourquoi 25 n'est-il pas égal à "25" ?
- Demander des valeurs à l'utilisateur
- Variables et opérations - Exercices
- Python - Opérations mathématiques de base

© 2025 Mr. Lambrechts

Python: Les chaines de caractères
Python: Rechercher dans une chaîne de caractères
Python: Les conditions
🎯 Conditions en Python: objectifs d'apprentissage.
Python: Les conditions simples
Python: Opérations de comparaison
Python: Opérations de comparaison - Exemples
Python: Conditions / Exercices de base
Python: Les conditions (combinaisons)
Python - Les opérateurs logiques
Python: Conditions / Combinaisons / Exercices
Python: Les boucles
Python: Les boucles for
Python: Les boucles for (exercices 💪)
Python: Compter et cumuler des valeurs dans une boucle
Python: Les boucles for sur des listes
Python: Afficher un menu de sélection
Projets Python
🦸 Superhéros : couleurs, bonus et combat
Python: L'opérateur 'f' et les f-strings
Introduction à Python

HARDWARE

Les périphériques
Le clavier azerty

JEUX VIDÉO

Tuto PyGame
Code de base d'un jeu PyGame
Introduction à la boucle de jeu (game loop)
Gestion des coordonnées en pixels dans un jeu vidéo
Afficher un personnage à l'écran en PyGame
Gérer les entrées au clavier
Déplacer un personnage en PyGame
Afficher une couleur en fond d'écran en PyGame

Afficher des images en fond d'écran en PyGame
Empêcher le joueur de sortir de l'écran avec pygame
Utiliser un joystick dans PyGame (bonus facultatif)
Générer un carré d'or sur l'écran
Détecter une collision simple (un joueur et un carré)
Générer plusieurs carrés d'or sur l'écran
Détecter plusieurs collisions avec des carrés
Ajouter une musique et des sons
Générer plusieurs pièces d'or sur l'écran

SYSTÈMES D'EXPLOITATION

Fichiers et dossiers
Organisez votre vidéothèque (2025)
Les outils Windows
Introduction à la ligne de commande Windows

TECHNOLOGIES

3D et impression
TinkerCad — modéliser ton premier objet
L'impression 3D — comment ça marche
Projet : ton premier objet imprimé



BUREAUTIQUE

PowerPoint

Le fichier se trouve en pièce jointe.

MULTIMÉDIA

Les Images Numériques

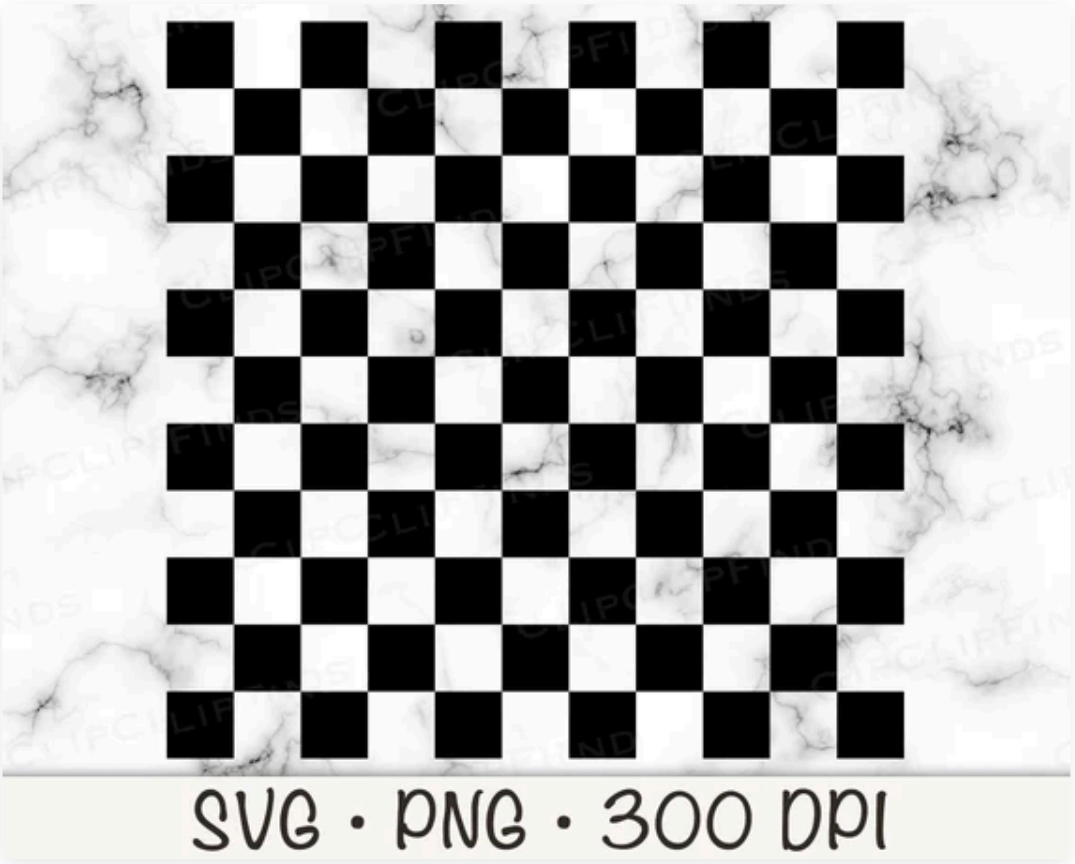
Les images

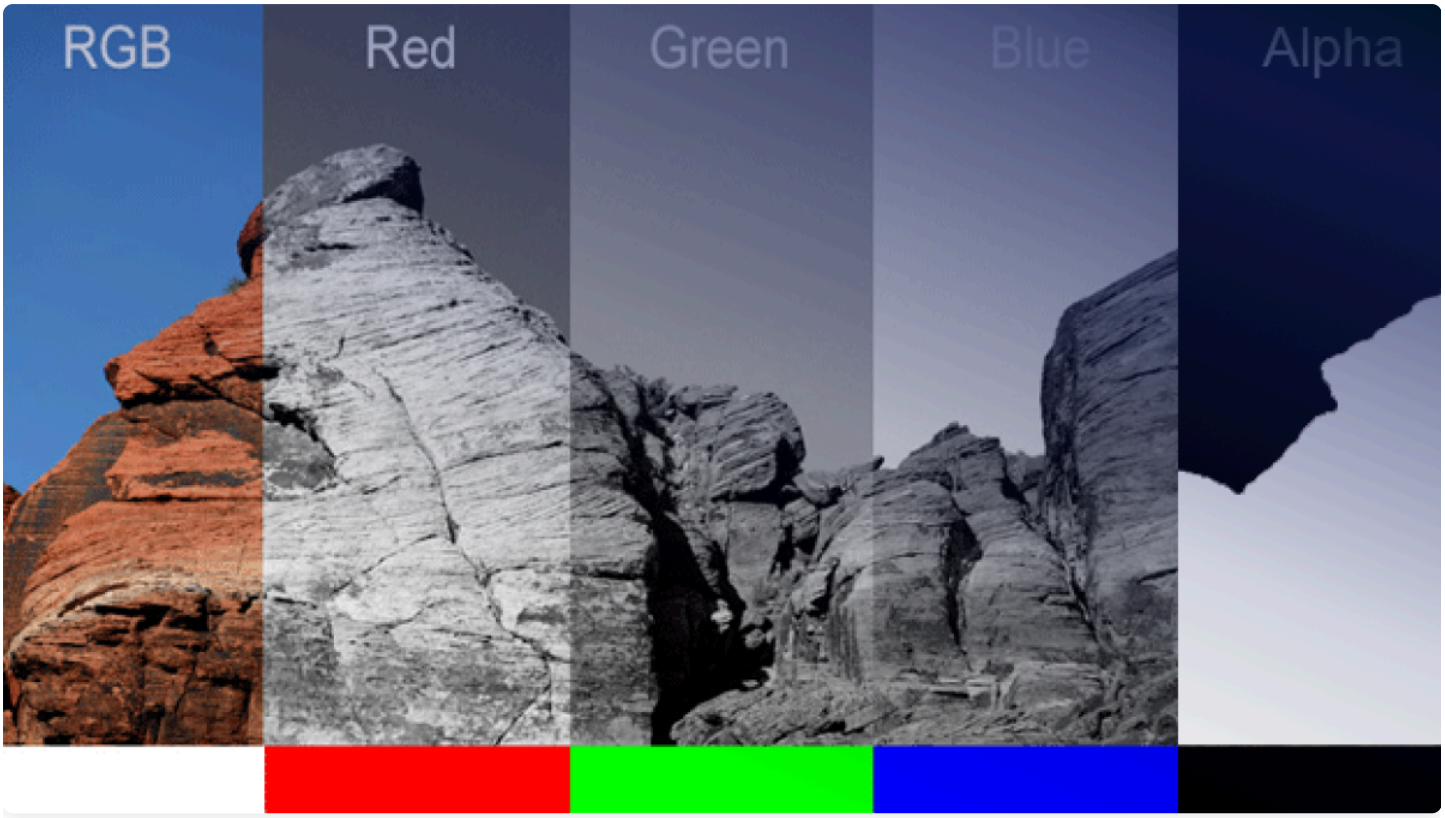
Gestion de la transparence

> *Dans cet article, nous allons comprendre ce qu'est la **transparence dans une image**, comment elle fonctionne techniquement, et pourquoi certains formats (comme PNG ou WebP) la supportent alors que d'autres (comme JPEG) ne le permettent pas.*

Qu'est-ce que la transparence ?

Image







La transparence permet de rendre **certaines zones d'une image invisibles**.

👉 Concrètement :

- une partie de l'image peut laisser apparaître **ce qu'il y a derrière**
- très utile pour superposer des images

Exemple :

- un logo sans fond
- une icône posée sur une interface

Comment ça fonctionne ?

Le canal alpha

Une image classique contient 3 informations par pixel :

- Rouge (R)
- Vert (G)
- Bleu (B)

👉 On parle de **RGB**

Pour gérer la transparence, on ajoute une 4e information :

- **Alpha (A)**

👉 On parle alors de **RGBA**

▸ Valeur de transparence

Le canal alpha définit **le niveau de visibilité** :

- 0 → totalement transparent (invisible)
- 255 → totalement opaque (visible)
- entre les deux → semi-transparent

👉 Cela permet des effets comme :

- ombres
- flous
- dégradés transparents

Tous les formats ne gèrent pas la transparence

- **JPEG : pas de transparence ❌**




Limitation



Le format **JPEG** :

- ❌ ne supporte **pas la transparence**
- remplace les zones transparentes par une couleur (souvent blanc)

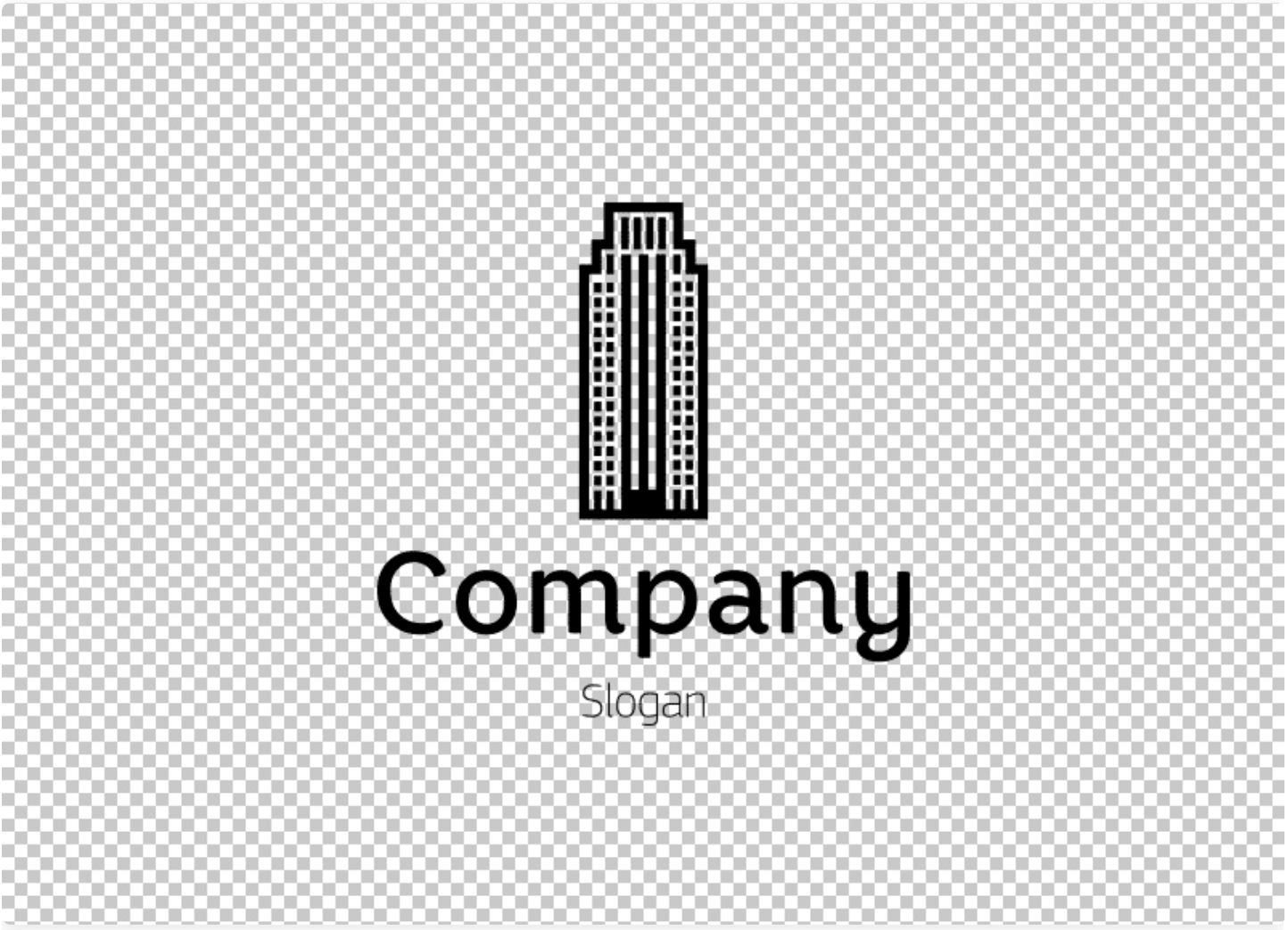
👉 Conséquence :

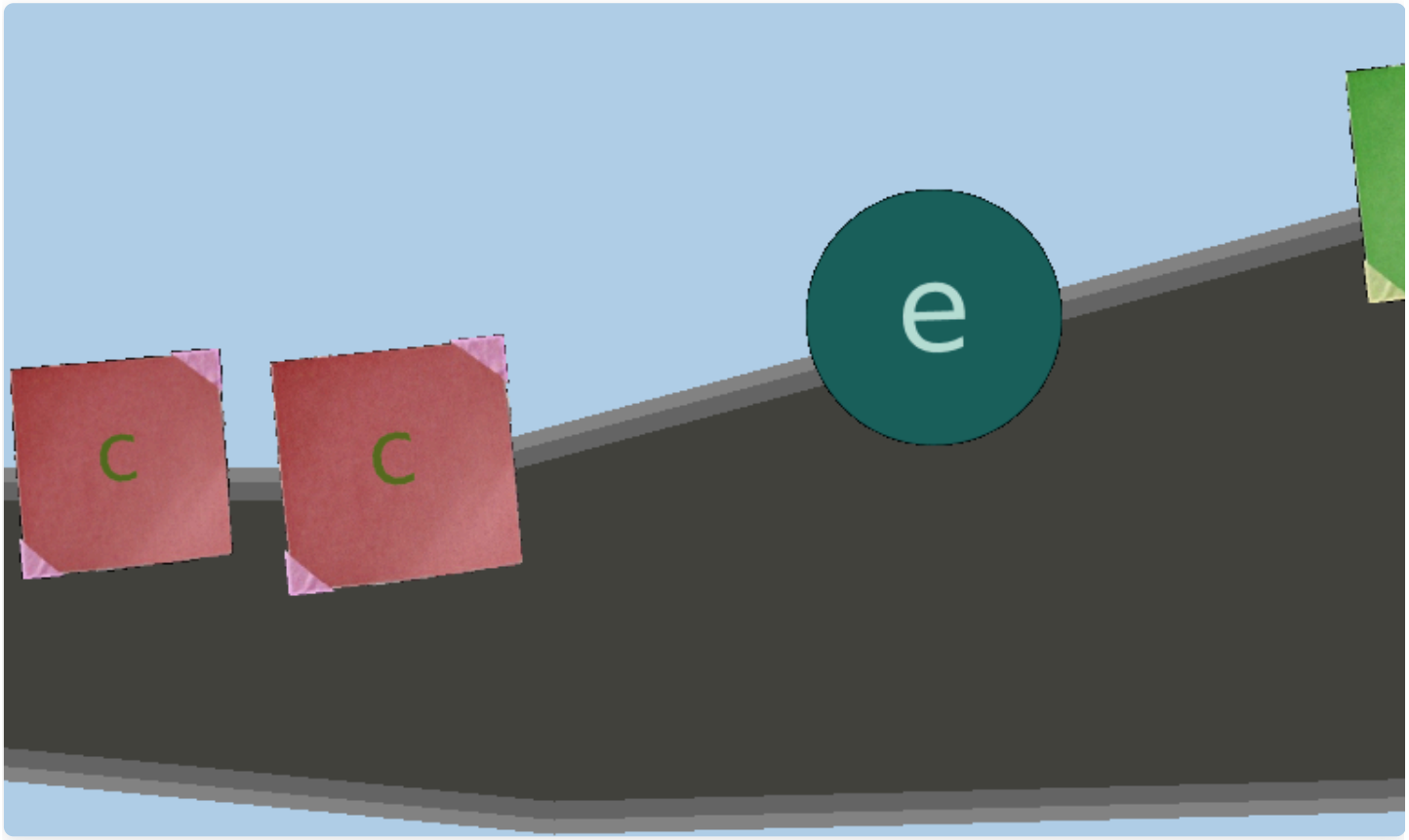
- impossible d'avoir un logo "découpé"
- fond toujours visible

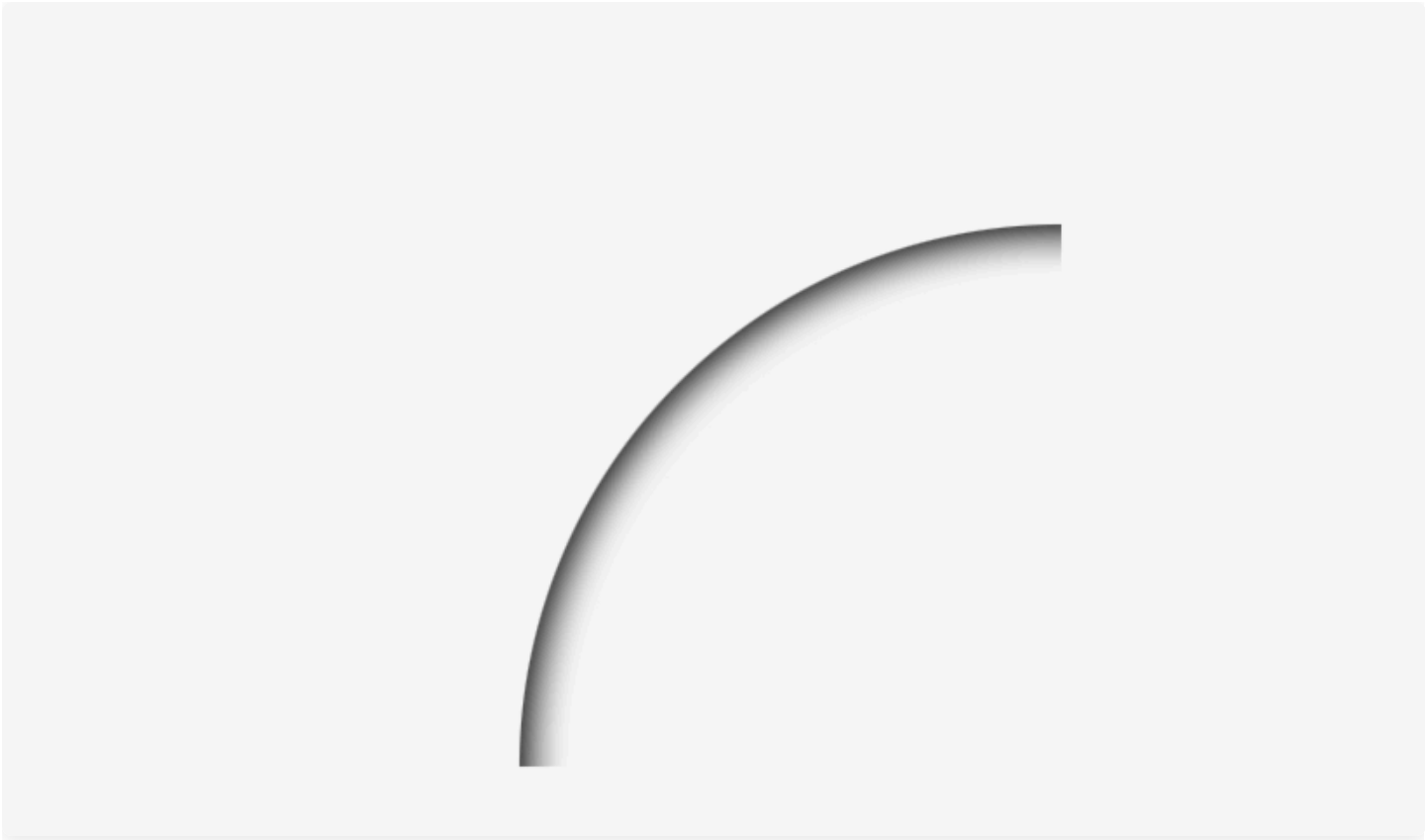
✓ Utilisation :

- photos
- images complètes (pas besoin de transparence)

- **PNG : transparence avancée ✓**







Image

Le format **PNG** :

- ✓ supporte la transparence complète (canal alpha)
- ✓ gère les **transparences progressives** (semi-transparent)
- ✓ qualité sans perte

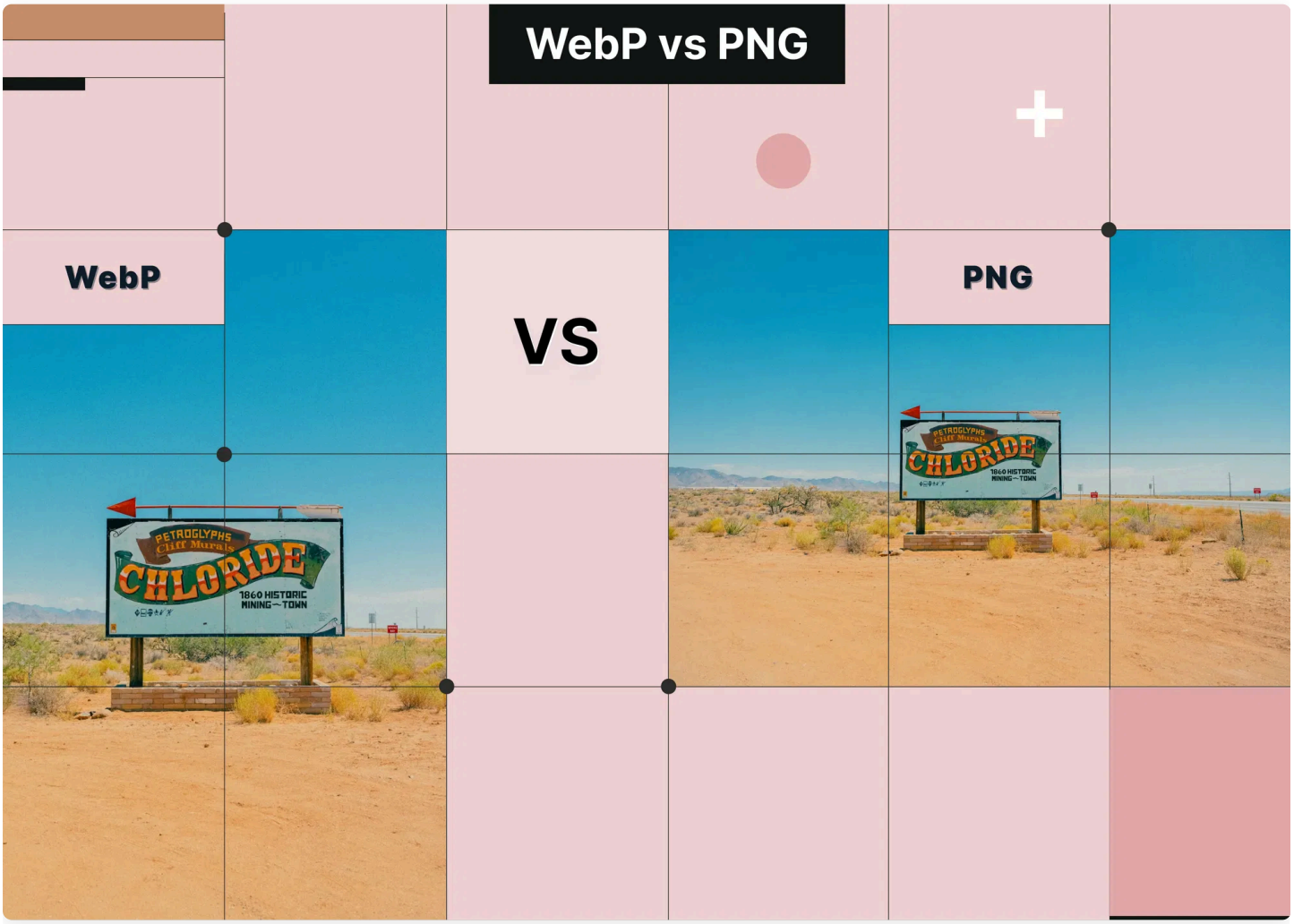
👉 Avantages :

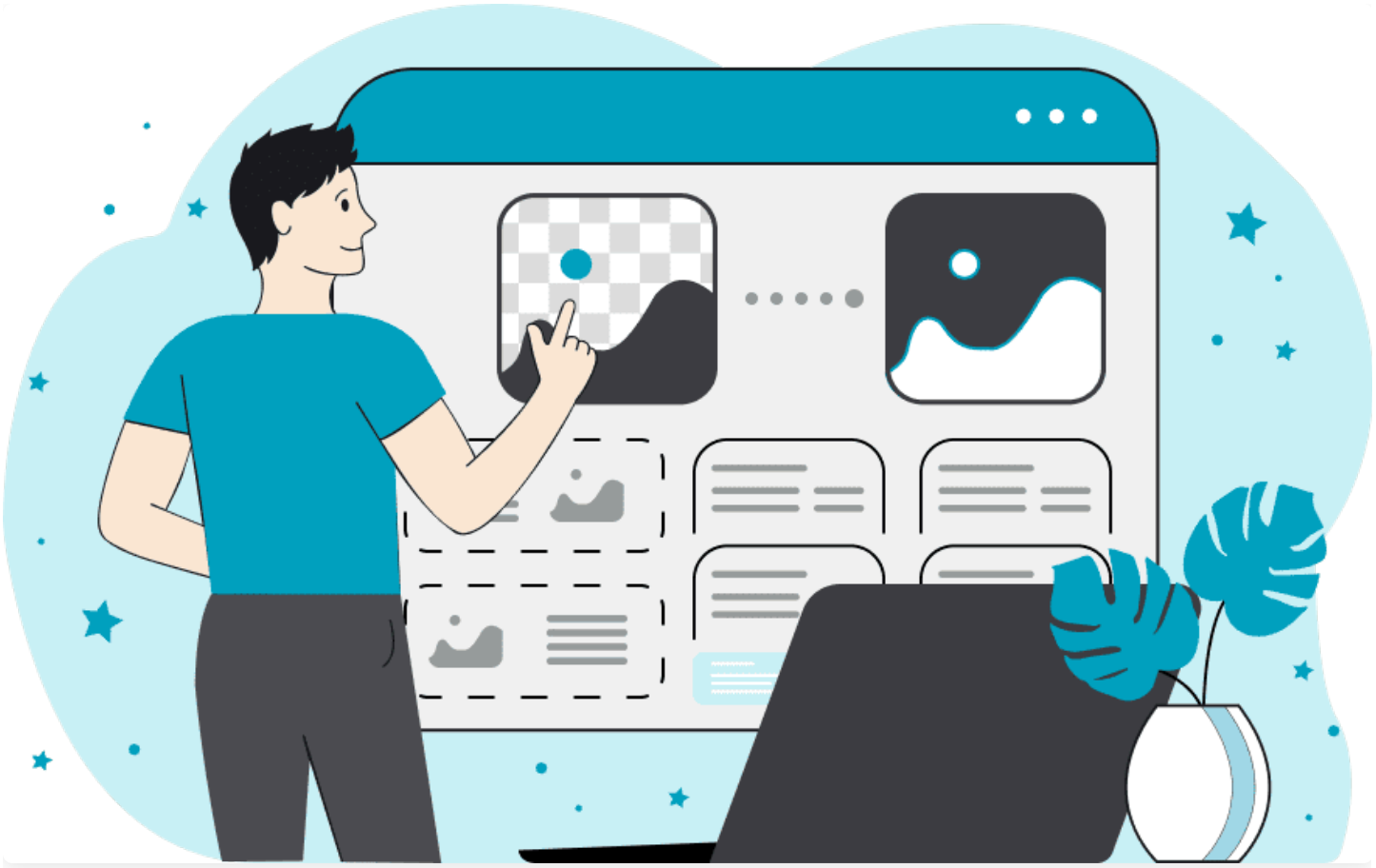
- contours propres
- effets visuels précis

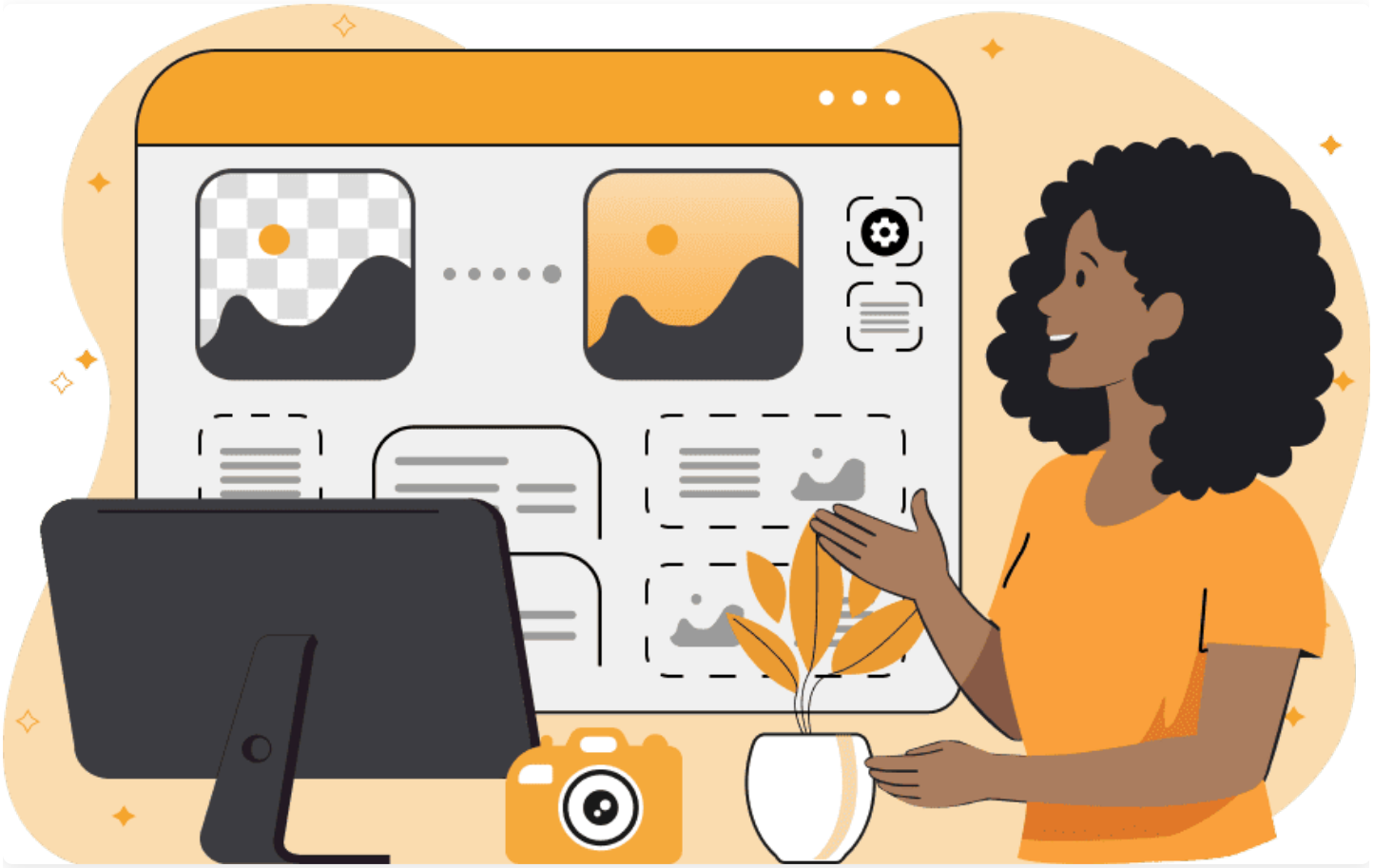
✓ Utilisation :

- logos
- icônes
- interfaces
- éléments graphiques

- WebP : transparence moderne ✓







Le format **WebP** (développé par Google) :

- ✓ supporte la transparence (comme PNG)

- ✓ compression plus efficace
- ✓ possible avec ou sans perte

👉 Avantages :

- fichiers plus légers
- idéal pour le web moderne

✓ Utilisation :

- sites web optimisés
- applications web

Comparaison rapide

FORMAT	TRANSPARENCE	TYPE	USAGE
JPEG	❌ Non	Compression avec perte	Photos
PNG	✓ Oui (précise)	Sans perte	Logos, UI
WebP	✓ Oui	Avec ou sans perte	Web moderne

Bonnes pratiques

- ✓ Utiliser **PNG** pour les images avec transparence de qualité
- ✓ Utiliser **WebP** pour optimiser les performances web
- ❌ Ne jamais utiliser **JPEG** pour un logo ou une icône

👉 Règle simple :

- besoin de transparence → **PNG ou WebP**
- photo classique → **JPEG**

En résumé

La transparence permet de rendre **une partie d'une image invisible**, grâce au **canal alpha**.

- JPEG → pas de transparence
- PNG → transparence précise et fiable
- WebP → transparence + optimisation

👉 Bien gérer la transparence est essentiel pour :

- le design web
- les interfaces
- les logos et éléments graphiques

Si tu veux, je peux te faire :

- un **TP simple (tester les 3 formats en HTML)**
- ou un **exercice visuel où les élèves doivent choisir le bon format**

Images numériques : matricielles vs vectorielles

> *Cet article explore les différences clés entre les deux types d'images principaux: les images matricielles, idéales pour les photographies et les illustrations numériques, et les images vectorielles, optimales pour les logos et les graphiques, en mettant l'accent sur leur utilisation appropriée en fonction des besoins spécifiques du projet.*

Dans cet article, nous allons comprendre comment sont représentées les images numériques et apprendre à choisir le bon type d'image selon le contexte (web, photo, design…).

Pourquoi s'intéresser aux formats d'image ?

Les images numériques sont utilisées dans de nombreux domaines :

- 📸 photographie
- 🎮 multimédia et jeux vidéo
- 🎨 conception graphique
- 🌐 développement web

Un **format d'image** est une manière standard de **stocker et transmettre une image**.

👉 Chaque format possède ses propres caractéristiques :

- qualité
- poids du fichier
- possibilités (transparence, animation…)
- cas d'utilisation

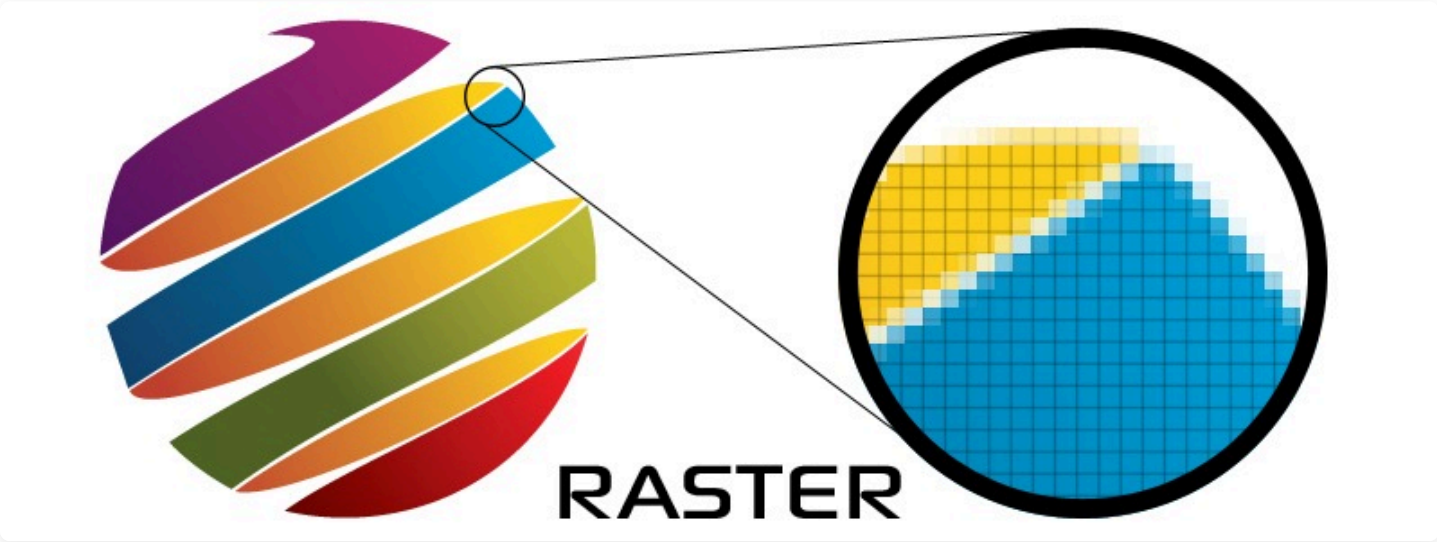
Deux grandes familles d'images

Il existe deux types principaux d'images numériques :

- les **images matricielles** (bitmap)
- les **images vectorielles**

👉 Comprendre cette différence est **fondamental** pour choisir le bon format.

Les images matricielles (bitmap / raster)



Une image matricielle est composée d'une **grille de pixels**.

- Chaque pixel = une couleur
- L'ensemble des pixels forme l'image

👉 On peut voir cela comme une **mosaïque numérique**

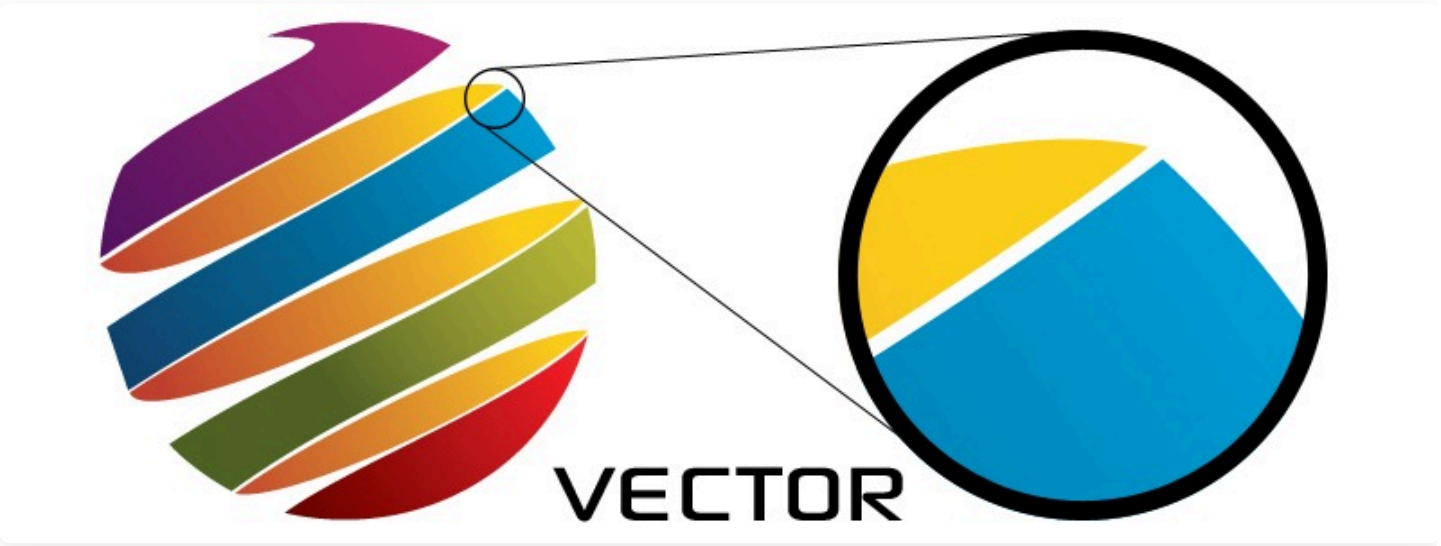
Caractéristiques

- Basée sur des **pixels**
- Qualité dépend du **nombre de pixels**
- **Perte de qualité** lors de l'agrandissement (pixelisation)

Utilisations typiques

- photographies
- captures d'écran
- images réalistes

Les images vectorielles



Une image vectorielle est basée sur des **formes géométriques** :

- lignes
- courbes
- points
- polygones

Ces éléments sont définis par des **formules mathématiques**.

👉 L'image est "recalculée" en permanence, peu importe sa taille.

Caractéristiques

- Basée sur des **équations mathématiques**
- **Aucune perte de qualité** lors du redimensionnement
- Fichiers souvent **plus légers** (pour des formes simples)

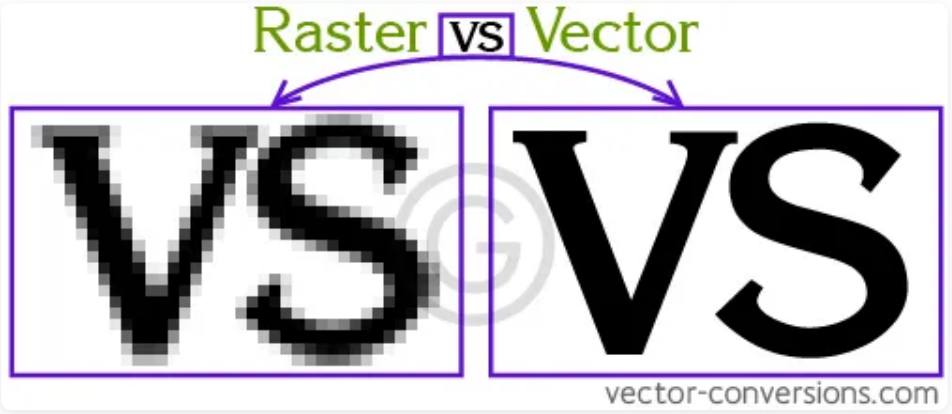
Utilisations typiques

- logos
- icônes
- illustrations
- schémas

Formats courants

- SVG
- AI
- EPS

Différences essentielles



- **Structure**
  - Matricielle → pixels
  - Vectorielle → formes mathématiques
- **Qualité lors du zoom**
  - Matricielle → perte de qualité
  - Vectorielle → aucune perte
- **Type d'image**
  - Matricielle → images complexes (photos)
  - Vectorielle → images simples (logos, schémas)
- **Poids du fichier**
  - Matricielle → souvent plus lourd
  - Vectorielle → souvent plus léger (selon complexité)

Comment choisir ?

👉 La règle la plus importante :

- **Photo → matriciel**
- **Logo / schéma → vectoriel**

Exemples concrets

- Une photo de paysage → **matriciel**
- Un logo d'entreprise → **vectoriel**
- Une interface d'application → souvent **mix des deux**

Conclusion

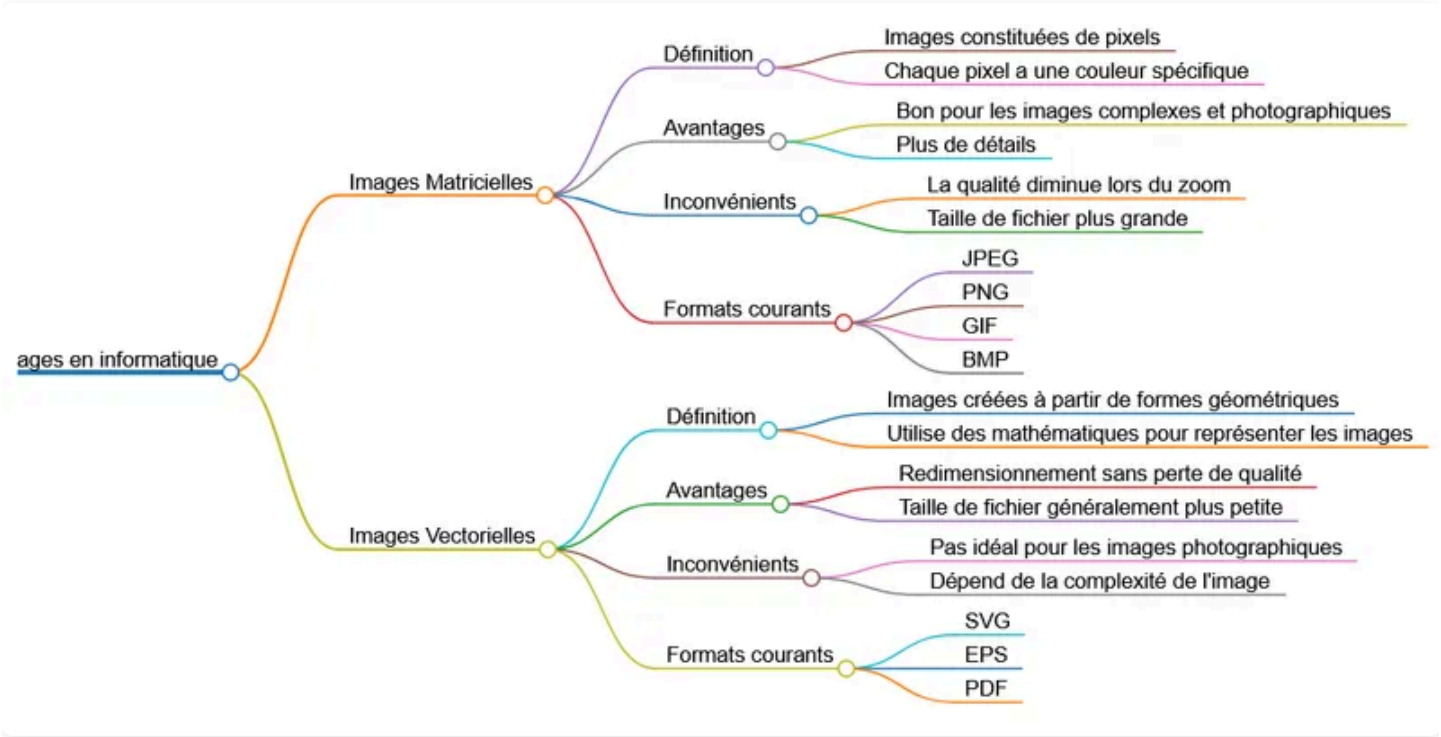
Les images numériques reposent sur deux approches très différentes :

- les **images matricielles**, composées de pixels, idéales pour les images réalistes
- les **images vectorielles**, basées sur des formules mathématiques, parfaites pour les formes nettes et redimensionnables

👉 Le choix dépend toujours de l'usage.

Bien choisir le type d'image permet :

- d'obtenir une meilleure qualité
- de réduire le poids des fichiers
- d'éviter des problèmes d'affichage (flou, pixelisation)



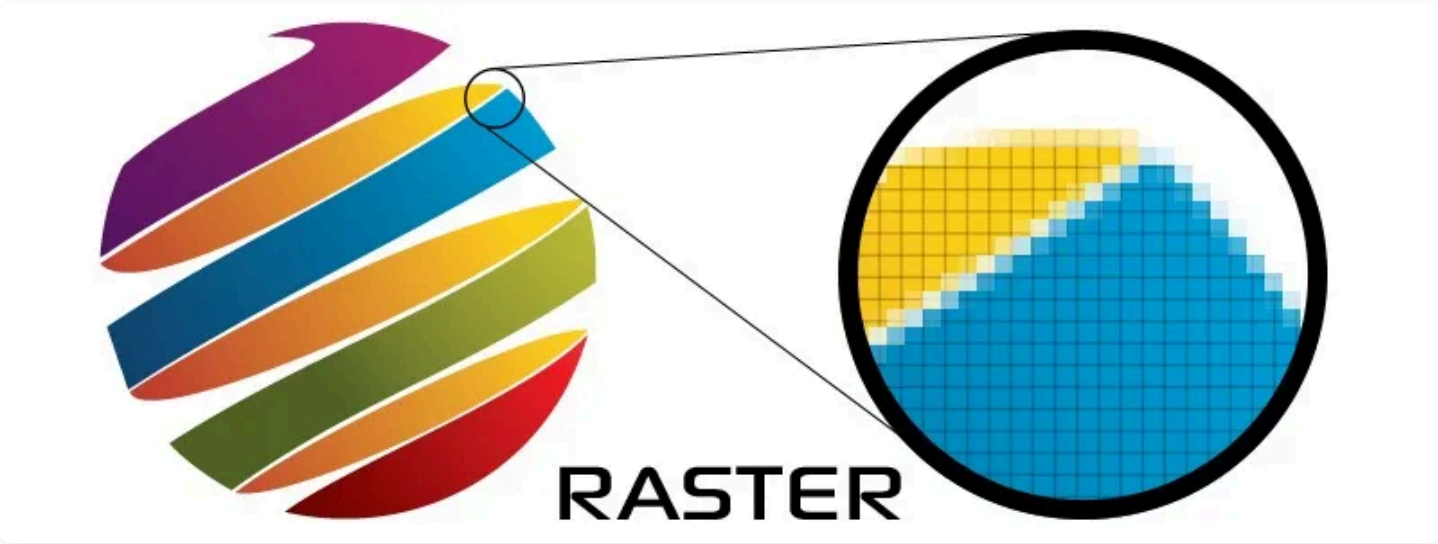
Les formats d’images matriciels

> *Les images matricielles, également appelées **images bitmap**, **images raster** ou **images à base de pixels**, sont un type d'image numérique composé d'une **grille de pixels**, où **chaque pixel contient des informations de couleur**. Les images matricielles sont couramment utilisées pour les **photographies**, les illustrations numériques et les captures d'écran.*

Dans cet article, nous allons comprendre comment sont stockées les images que vous utilisez tous les jours (photos, captures d’écran, images web) et comment choisir le bon format selon le besoin.

Qu’est-ce qu’une image matricielle ?

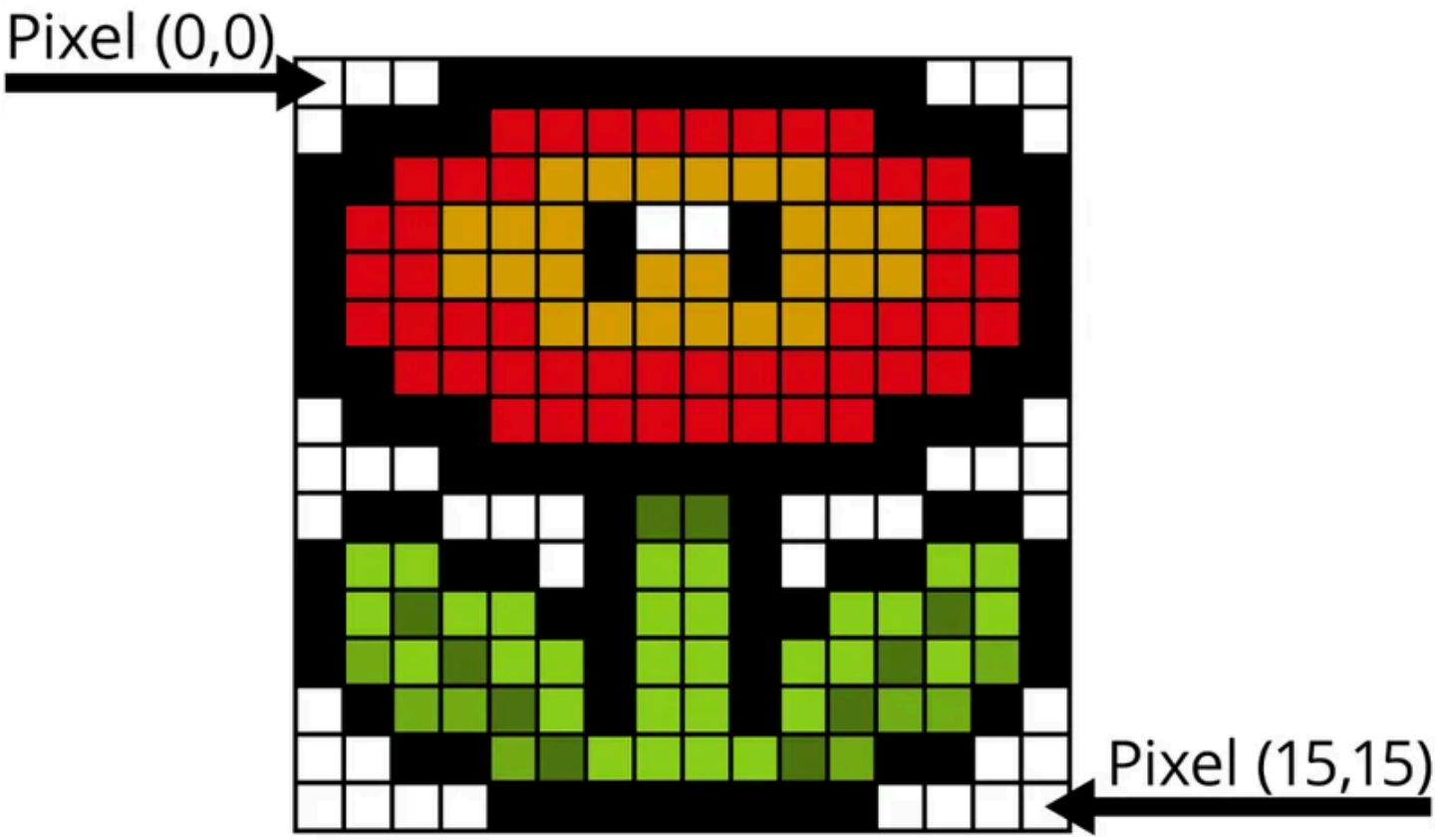
Une image matricielle (aussi appelée **bitmap** ou **raster**) est une image composée d’une **grille de pixels**.



> *Un **pixel** est le plus petit élément d’une image numérique. C'est aussi le plus petit élément affichable sur un écran.*

Chaque pixel contient une **couleur précise**.

👉 Une image est donc simplement une **grande mosaïque de pixels colorés**.



Plus il y a de pixels, plus l'image peut contenir de détails.

Comprendre la taille et la qualité d'une image

La définition (ou dimensions)

La définition correspond au **nombre de pixels** dans l'image. Elle généralement spécifiée sous la forme `l x h` (largeur x hauteur).

Exemple :

- 1920 × 1080 pixels → 1920 pixels en largeur → 1080 pixels en hauteur

👉 Nombre total de pixels = 1920 × 1080 = 2.073.600 pixels ≈ **2 mégapixels (MP)**

La résolution

La résolution correspond à la **densité de pixels** sur une surface (souvent en **ppi** ou **dpi**).

👉 Important :

- **Définition = quantité de pixels**
- **Résolution = densité des pixels**

L'agrandissement et la pixelisation

Quand on agrandit une image matricielle :

- on ne crée pas de nouveaux détails
- les pixels deviennent visibles

👉 Résultat : **image floue ou “carrée”** → c’est la **pixelisation**





Les principaux formats d’images matriciels

JPEG (ou JPG)

- Format **le plus utilisé pour les photos**
- Compression **avec perte**
- Très bon **rapport qualité / poids**
- Supporte des **millions de couleurs**

👉 Idéal pour :

- photos

- images réalistes
- web (images légères)

PNG

- Compression **sans perte**
- Qualité **parfaite (pas de dégradation)**
- Supporte la **transparence** (fond invisible)
- Fichiers souvent **plus lourds**

👉 Idéal pour :

- logos
- interfaces
- captures d'écran
- images avec texte

GIF

- Limité à **256 couleurs**
- Supporte la **transparence**
- Permet les **animations**

👉 Idéal pour :

- petites animations
- mèmes
- images simples

WebP

- Format moderne développé par Google
- Compression **avec ou sans perte**
- Plus **efficace** que JPEG et PNG
- Supporte :
  - transparence
  - animations

👉 Idéal pour :

- le web moderne
- optimiser les performances (chargement rapide)

FORMAT	TYPE DE COMPRESSION	QUALITÉ	TRANSPARENCE	ANIMATION	NOMBRE DE COULEURS	POIDS DES FICHIERS	CAS D'USAGE IDÉAL
JPEG (JPG)	Avec perte	Bonne (mais dégradation possible)	❌ Non	❌ Non	Millions	Léger	Photos, images réalistes, web
PNG	Sans perte	Excellente (aucune perte)	✅ Oui	❌ Non	Millions	Lourd	Logos, interfaces, captures d'écran, texte
GIF	Sans perte (mais limité)	Faible à moyenne	✅ Oui	✅ Oui	256 couleurs	Léger à moyen	Animations simples, mèmes
WebP	Avec **ou** sans perte	Excellente	✅ Oui	✅ Oui	Millions	Très léger	Web moderne, optimisation des performances

.table .table-striped .table-bordered .table-hover .w-100 .mx-auto}

Quand utiliser chaque format ?

SITUATION	FORMAT CONSEILLÉ
Photo	JPEG
Logo / image avec transparence	PNG
Animation simple	GIF
Site web optimisé	WebP

Cas d'utilisation des images matricielles

On les retrouve partout :

- 📸 Photographie numérique
- 🖥️ Interfaces et sites web
- 🎮 Jeux vidéo
- 📱 Réseaux sociaux
- 🖥️ Captures d'écran

Bonnes pratiques

- ✓ Choisir le format selon le **type d'image**
- ✓ Éviter d'agrandir une image trop petite
- ✓ Compresser les images pour le web
- ❌ Éviter les formats lourds comme **BMP** (non compressé)

En résumé

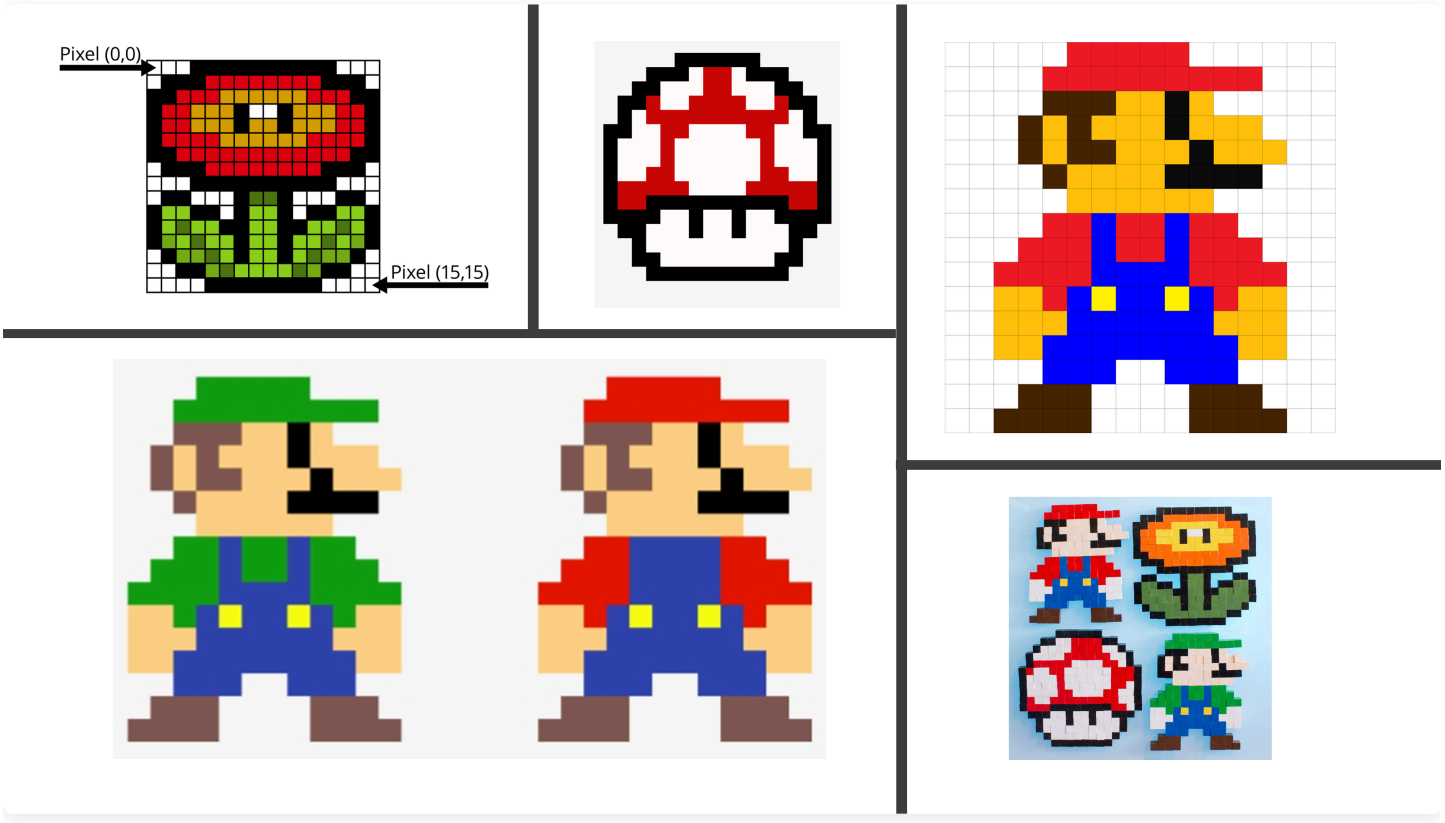
Une image matricielle est une image composée de **pixels organisés en grille**.

- Plus il y a de pixels → plus de détails
- L'agrandissement → perte de qualité
- Le format influence :
  - la qualité
  - le poids
  - les fonctionnalités (transparence, animation)

👉 Ce type d'image est indispensable pour représenter des images **réalistes et détaillées**, comme les photographies.

Exercice

Pour le jeu, essaye de reproduire 2 de ces images dans le logiciel IcoFX.



Compression avec et sans perte

> *Dans ce chapitre, tu vas comprendre comment réduire la taille des images tout en choisissant le bon compromis entre qualité et poids.*



🎯 Objectifs

À la fin, tu dois être capable de :

- comprendre ce qu'est la compression d'image
- distinguer **compression avec perte** et **sans perte**
- choisir le bon type selon la situation
- reconnaître les formats les plus courants

📸 Pourquoi compresser une image ?

Une image (photo, dessin…) peut être **très lourde** : plusieurs mégaoctets.

👉 Problèmes :

- lente à charger sur un site
- longue à envoyer (mail, réseau)
- prend beaucoup de place

👉 Solution : **la compression**

➡️ Elle permet de **réduire la taille du fichier**, parfois **jusqu'à 20 fois moins** !

📐 Taille d'une image non compressée

Une image est composée de **pixels**.

Chaque pixel contient :

- Rouge (R)
- Vert (V)
- Bleu (B)

👉 Chaque couleur = **1 octet** 👉 Donc 1 pixel = **3 octets**

▸ Exemple

Image de **24 mégapixels** :

- 24 000 000 pixels × 3 octets
- = **72 000 000 octets ≈ 72 Mo**

👉 Une seule image = **72 Mo** sans compression 😱

➡️ D'où l'importance de compresser !

⚖️ Deux types de compression

Il existe **2 méthodes principales** :

- 🟢 **Sans perte (lossless)**
- 🔴 **Avec perte (lossy)**

🟢 Compression sans perte

▸ 🧠 Principe

Aucune donnée n'est supprimée.

👉 L'image est optimisée intelligemment 👉 Mais **reste strictement identique** après décompression

▸ ✅ Avantages

- Qualité parfaite
- Aucune perte d'information

▸ ❌ Inconvénients

- Fichiers plus lourds

- Compression moins efficace

📁 Formats

- PNG
- TIFF
- BMP

🔴 Compression avec perte

🧠 Principe

Certaines informations sont **supprimées définitivement**.



👉 L'image est allégée 👉 Mais **légèrement dégradée**

✅ Avantages

- Fichiers beaucoup plus légers
- Idéal pour le web et le partage

❌ Inconvénients

- Perte de qualité (visible si trop compressé)
- Impossible de retrouver l'image originale

📁 Formats

- JPEG
- WebP

🧭 Comment choisir ?

SITUATION	CHOIX RECOMMANDÉ
Impression, archivage, qualité maximale	Sans perte
Web, réseaux sociaux, envoi rapide	Avec perte

📝 À retenir

- **Sans perte** → qualité parfaite, fichiers plus lourds
- **Avec perte** → fichiers légers, qualité réduite

🧠 Synthèse (l'essentiel)

- La compression sert à **réduire la taille des images**
- Sans perte → aucune dégradation
- Avec perte → qualité réduite mais gain important
- Formats :
  - JPEG / WebP → avec perte
  - PNG / TIFF / BMP → sans perte
- Le choix dépend du besoin : **qualité ou légèreté**

Les images vectorielles

> *Dans cet article, nous allons découvrir les images vectorielles, un type d'image numérique distinct qui repose sur l'utilisation de formes géométriques, de courbes et de lignes définies par des formules mathématiques.*

Les images vectorielles sont des images numériques **basées sur des formes géométriques** (lignes, courbes, polygones), définies à l'aide de **formules mathématiques**.

Contrairement aux images matricielles, elles **ne sont pas composées de pixels**.

> 👉 *Une image vectorielle n'est pas "dessinée pixel par pixel" : elle est **décrite mathématiquement**, puis affichée par l'ordinateur.*

Définition et caractéristiques

Formes géométriques

Une image vectorielle est composée de :

- lignes
- courbes
- points
- formes (cercles, rectangles, polygones…)

Chaque élément est défini par des **équations mathématiques**.

👉 Cela permet de :

- déplacer
- transformer
- modifier

sans diminuer la qualité.

Redimensionnement sans perte

C'est la caractéristique principale :

- agrandir → qualité intacte
- réduire → qualité intacte

👉 L'image est **recalculée** à chaque affichage.

➡️ On parle d'images **indépendantes de la résolution**

Taille des fichiers

Les fichiers vectoriels sont souvent :

- **plus légers** que les images matricielles (pour des dessins simples)

👉 Car ils stockent :

- des **instructions (formes, coordonnées)** et non une couleur pour chaque pixel

---

Limitations

Les images vectorielles ne sont **pas adaptées** pour :

- les photographies
- les images très détaillées
- les textures complexes

👉 Elles ne peuvent pas reproduire facilement :

- des nuances fines
- des dégradés réalistes
- du bruit visuel complexe

Principaux formats

SVG (Scalable Vector Graphics)

- Format basé sur **XML**
- Compatible avec les navigateurs web
- Redimensionnable **sans perte**
- Léger et très utilisé sur le web

👉 Idéal pour :

- icônes
- logos
- interfaces web

EPS (Encapsulated PostScript)

- Format historique utilisé en graphisme
- Très compatible avec de nombreux logiciels
- Utilisé en impression et en design

👉 Idéal pour :

- logos professionnels
- fichiers destinés à l'impression

AI (Adobe Illustrator)

- Format propriétaire d'Adobe
- Utilisé pour la création graphique avancée
- Nécessite souvent Adobe Illustrator pour l'édition

👉 Idéal pour :

- projets professionnels
- design complexe

Cas d'utilisation

Les images vectorielles sont utilisées pour :

- 🎨 illustrations
- 🧩 icônes
- 🏷️ logos
- 📊 schémas et diagrammes
- 🌐 éléments d'interface web

Recommandations

- ✓ Utiliser **SVG** pour le web (léger, flexible, moderne)
- ✓ Utiliser **EPS** pour la compatibilité et l'impression
- ✓ Utiliser **AI** pour les projets professionnels complexes

👉 Règle simple :

- formes simples → vectoriel
- images réalistes → matriciel

En résumé

Une image vectorielle est une image **définie par des formules mathématiques** décrivant des formes géométriques.

Ses points forts :

- redimensionnement **sans perte de qualité**
- fichiers souvent **légers**
- idéal pour les éléments graphiques

Sa limite principale :

- **peu adaptée aux images complexes** comme les photos

👉 C'est le format incontournable pour tout ce qui doit rester **net, précis et adaptable à toutes les tailles**.

Résolution et définition d'une image : différences et implications

La **résolution** et la **définition** sont deux termes couramment utilisés pour décrire la qualité d'une image numérique. Bien qu'ils soient souvent utilisés de manière interchangeable, ils ont des significations distinctes. Dans ce cours, nous expliquerons la différence entre la résolution et la définition d'une image et comment ces concepts influencent la qualité et l'apparence des images numériques.

Définition d'une image

La définition d'une image fait référence au **nombre total de pixels** qui composent l'image Elle est généralement exprimée en largeur x hauteur (par exemple, 1920x1080 pixels)

Implications

- Plus la définition est élevée, plus l'image contient de détails et d'informations
- Une définition plus élevée peut entraîner des fichiers d'image plus volumineux
- La définition d'une image détermine sa taille native et peut influencer la qualité lors du redimensionnement

Résolution d'une image

La résolution d'une image fait référence au **nombre de pixels par unité de surface** (généralement par pouce ou par centimètre).

Elle est souvent **exprimée en DPI** (*dots per inch*, points par pouce) pour l'impression ou en **PPI** (*pixels per inch*, pixels par pouces) pour les images affichées sur des écrans.

Implications

- La résolution influence la qualité de l'image lorsqu'elle est affichée ou imprimée
- Une résolution plus élevée produit des images plus nettes et plus détaillées
- La résolution est particulièrement **importante pour l'impression**, où une résolution insuffisante peut entraîner une qualité médiocre

Comparaison entre définition et résolution

Différences

- La définition concerne le nombre total de pixels, tandis que la résolution se rapporte au nombre de pixels par unité de surface
- La définition affecte directement la taille native de l'image, tandis que la résolution influence la qualité de l'image lors de l'affichage ou de l'impression

Interdépendance

- La résolution et la définition sont liées : **une image de haute définition peut être affichée ou imprimée avec une résolution élevée** pour produire une image de haute qualité
- Ajuster la résolution sans modifier la définition peut entraîner une perte de qualité, car les pixels sont étirés ou compressés pour s'adapter à la nouvelle résolution

Considérations pratiques

- Pour les **écrans** numériques, **une résolution de 72 DPI est généralement suffisante**, tandis que **pour l'impression, une résolution d'au moins 300 DPI est recommandée**
- Adapter la définition et la résolution en fonction des besoins spécifiques du projet pour obtenir la meilleure qualité d'image possible

Conclusion

Comprendre la différence entre la résolution et la définition d'une image est essentiel pour choisir les paramètres appropriés en fonction du support et de l'usage souhaité.

La définition concerne le nombre total de pixels de l'image, tandis que la résolution se rapporte au nombre de pixels par unité de surface. Les deux concepts sont liés et ont un impact sur la qualité de l'image lorsqu'elle est affichée ou imprimée. Il est important de prendre en compte la définition et la résolution lors de la création, de l'édition ou de l'impression d'images numériques pour garantir une qualité optimale et des résultats professionnels. En adaptant ces paramètres en fonction des besoins spécifiques de votre projet, vous pouvez obtenir des images nettes, détaillées et attrayantes sur divers supports.



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)

Améliorez Vos Affichages avec le module Rich

> *Le module **Rich** est un outil puissant pour ceux qui souhaitent améliorer l'affichage de leurs programmes Python dans la console. Ce module vous permet de rendre vos sorties plus lisibles et esthétiques en ajoutant des couleurs, des tableaux, des barres de progression, des arbres de dossiers, et bien plus encore. Dans cet article, nous allons explorer quelques-unes des fonctionnalités de base que vous pouvez utiliser avec la fonction `print()`.*

Installation du Module Rich

Avant de commencer, vous devez installer le module Rich.

Dans Thonny

Allez dans le menu Outils/Gérer les paquets, cherchez "rich" et installer le paquet.

L'affichage des couleurs n'est pas pris en charge par la console de Thonny. Il faudra exécuter vos programmes dans une console système (`CTRL-T`).

Dans Python (global):

Vous pouvez l'installer facilement via `pip` :

```
pip install rich
```

Une fois installé, vous pouvez commencer à utiliser les fonctionnalités de Rich dans vos scripts Python.

Dans PyCharm (5e et 6e TTR)

Le module est facilement installable depuis le package manager.

Utilisation de Rich avec `print()`

L'un des principaux avantages de Rich est sa compatibilité avec la fonction `print()`. Contrairement à la fonction `print()` standard de Python, Rich permet de formater facilement le texte, de le colorier et de l'organiser en tableaux.

Exemple de Base : Texte Coloré

Pour commencer, voyons comment afficher du texte en couleur. Avec Rich, c'est aussi simple que d'utiliser la fonction `print()` :

```
from rich import print

print("[bold red]Ceci est un texte en rouge gras[/bold red]")
```

Dans cet exemple, le texte "Ceci est un texte en rouge gras" sera affiché en rouge et en gras. Le module Rich utilise des balises similaires à celles de HTML pour définir le formatage du texte.

Codes couleurs

Voici une liste des principales couleurs disponibles avec le module Rich, que vous pouvez utiliser pour styliser le texte dans vos sorties console :

COULEURS DE BASE

- **red** : Rouge
- **green** : Vert
- **blue** : Bleu
- **yellow** : Jaune
- **magenta** : Magenta (Rose violacé)
- **cyan** : Cyan (Bleu clair)
- **white** : Blanc
- **black** : Noir

COULEURS CLAIRES

- **bright_red** : Rouge clair
- **bright_green** : Vert clair
- **bright_blue** : Bleu clair
- **bright_yellow** : Jaune clair
- **bright_magenta** : Magenta clair
- **bright_cyan** : Cyan clair
- **bright_white** : Blanc très clair
- **grey** : Gris

NUANCES DE GRIS

- **grey0** : Noir pur
- **grey1** à **grey23** : Différentes nuances de gris allant de plus foncé à plus clair
- **grey100** : Blanc pur

AUTRES COULEURS PRÉDÉFINIES

- **orange1** : Orange vif
- **purple** : Violet
- **brown** : Marron
- **pink** : Rose
- **turquoise** : Turquoise

Ajouter des Styles au Texte

Rich vous permet également d'ajouter plusieurs styles à votre texte. Vous pouvez combiner des couleurs, du gras, de l'italique, du souligné, etc.

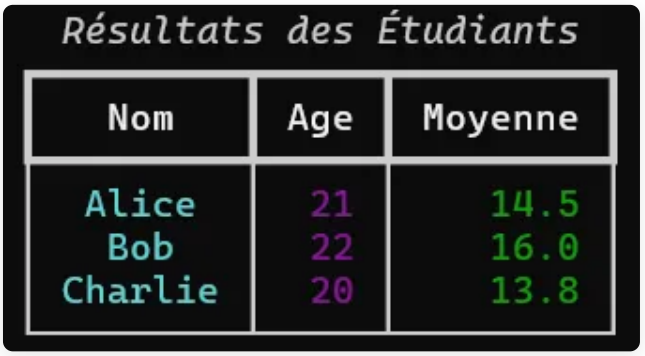
```
print("[italic blue]Texte en bleu et italique[/italic blue]")
print("[underline green]Texte en vert souligné[/underline green]")
print("[bold italic yellow]Texte en jaune gras et italique[/bold italic yellow]")
```

Ces options sont idéales pour mettre en évidence des parties spécifiques de vos sorties.

Affichage de Tableaux

L'une des fonctionnalités les plus intéressantes de Rich est la possibilité d'afficher des tableaux. Ceci est particulièrement utile pour afficher des données structurées.

Voici un exemple:




```python
from rich.table import Table

table = Table(title="Résultats des Étudiants")

table.add_column("Nom", justify="right", style="cyan", no_wrap=True)
table.add_column("Age", justify="right", style="magenta")
table.add_column("Moyenne", justify="right", style="green")

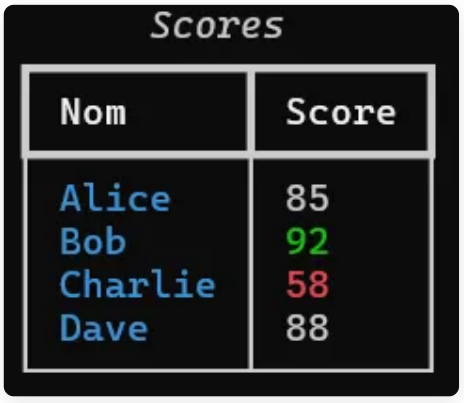
table.add_row("Alice", "21", "14.5")
table.add_row("Bob", "22", "16.0")
table.add_row("Charlie", "20", "13.8")

print(table)
```

Ce code créera un tableau stylisé avec des colonnes "Nom", "Age", et "Moyenne", chacune ayant sa propre couleur. Le tableau sera automatiquement ajusté pour être bien lisible dans la console.

Les styles conditionnels

Il est facile de personnaliser vos couleurs et styles en fonction des valeurs que afficher.



Il suffit de créer une variable `style` et de l'utiliser dans une f-string:

```python
from rich.table import Table
from rich import print

# Création d'un tableau avec du style conditionnel
table = Table(title="Scores avec Mise en Évidence")

table.add_column("Nom", style="cyan")
table.add_column("Score", style="magenta")

# Ajouter des lignes avec mise en évidence conditionnelle
scores = [("Alice", 85), ("Bob", 92), ("Charlie", 58), ("Dave", 88)]

for name, score in scores:
  if score >= 90:
    style = "bold green"
  elif score < 60:
    style = "bold red"
  else:
    style =  "white"

  table.add_row(name, f"[{style}]{score}[/{style}]")

print(table)
```

Créer des encadrés

Il est tout à fait possible d'encadrer du texte avec le module **Rich**. Pour cela, vous pouvez utiliser la classe `Panel` qui permet d'afficher du texte ou même d'autres objets Rich (comme des tableaux, des arbres, etc.) à l'intérieur d'un encadré, avec des bordures personnalisables.

Encadrement d'un texte:

Voici un exemple simple où l'on encadre du texte avec un `Panel` :



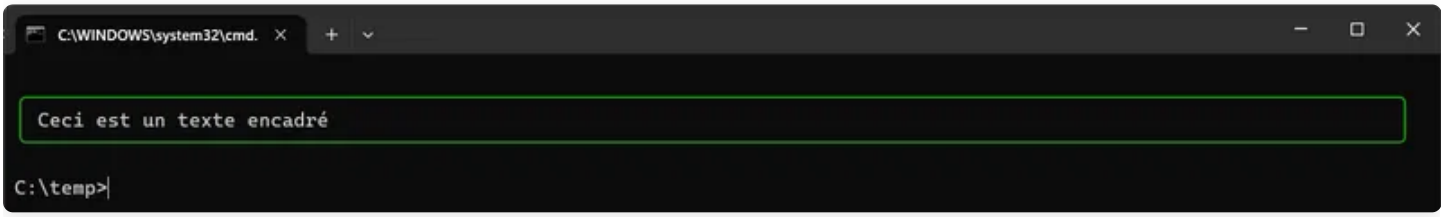

```python
from rich import print
from rich.panel import Panel

# Création d'un texte encadré
encadre = Panel("Ceci est un texte encadré")

# Affichage du texte encadré
print(encadre)
```

Changer la couleur de la bordure

Vous pouvez changer la couleur de la bordure grâce au paramètre `boder_style` :



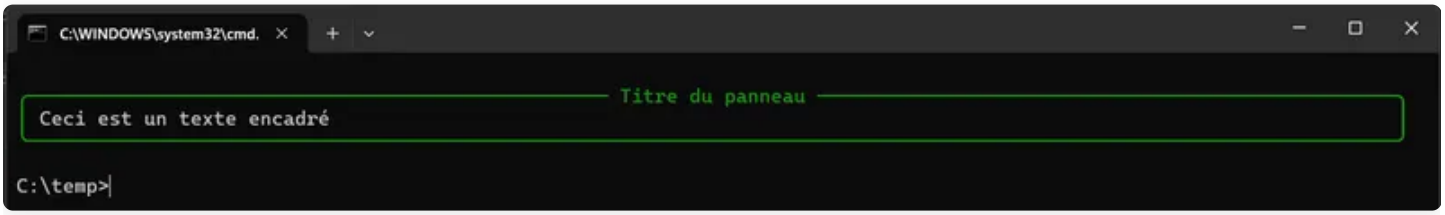

```python
from rich import print
from rich.panel import Panel

# Création d'un texte encadré avec un titre
encadre = Panel("Ceci est un texte encadré", border_style="green")

# Affichage du texte encadré
print(encadre)
```

Ajouter un titre au panel

Voici un exemple où l'on ajoute un titre au texte avec un `Panel` :




```python
from rich import print
from rich.panel import Panel

# Création d'un texte encadré avec un titre
encadre = Panel("Ceci est un texte encadré", title="Encadré", border_style="green")

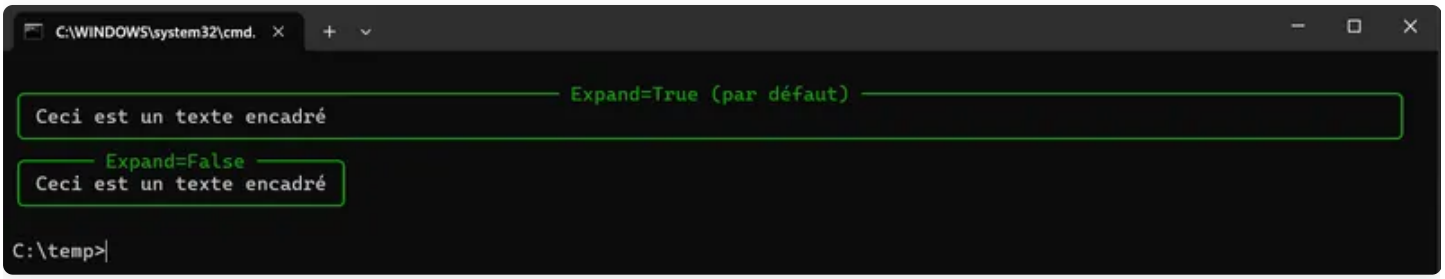
# Affichage du texte encadré
print(encadre)
```

Personnalisation du `Panel`

Le `Panel` offre plusieurs options pour personnaliser l'encadrement :

- `title` : Permet d'ajouter un titre au-dessus du panel.
- `border_style` : Définit la couleur des bordures.
- `padding` : Définit l'espacement intérieur entre le texte et les bordures du panel.
- `expand` : Si `True`, le panel occupera toute la largeur disponible de la console.
- `subtitle` : Ajoute un sous-titre en bas du panel.

Exemple avec expand




```python
from rich import print
from rich.panel import Panel

# Création d'un texte encadré
encadre = Panel(
    "Voici un texte important avec un encadrement stylisé.",
    title="Encadré Principal",
    border_style="bold red",
    expand=False
)

# Affichage du texte encadré
print(encadre)
```

Ces exemples montrent comment vous pouvez utiliser `Panel` pour encadrer du texte ou d'autres éléments, en ajoutant ainsi une touche esthétique et organisée à vos sorties dans la console. Cette fonctionnalité est particulièrement utile pour mettre en avant des informations importantes ou pour structurer des interfaces en ligne de commande plus complexes.

Utiliser des Barres de Progression

Rich facilite également l'affichage de barres de progression, ce qui est très utile pour indiquer l'avancement de certaines tâches.

```python
from rich.progress import track
import time

for step in track(range(10), description="Traitement..."):
    time.sleep(0.1)
```

Ici, la fonction `track()` de Rich enveloppe une boucle pour afficher une barre de progression pendant que la boucle s'exécute. Cela permet de donner un retour visuel immédiat sur la progression d'une tâche.

Conclusion

Le module Rich est un outil formidable pour tous ceux qui cherchent à améliorer l'affichage de leurs programmes Python dans la console. En utilisant Rich avec la fonction `print()`, vous pouvez facilement ajouter des couleurs, des styles, des tableaux, des barres de progression, et bien plus encore à vos sorties. Ces fonctionnalités permettent de rendre vos programmes non seulement plus esthétiques mais aussi plus lisibles et compréhensibles.

En explorant plus en profondeur les possibilités offertes par Rich, vous pouvez améliorer significativement l'expérience utilisateur de vos scripts Python. Alors, n'hésitez pas à l'essayer dans vos prochains projets !

Ressources

Pour aller plus loin, voici quelques ressources utiles pour apprendre à utiliser le module Rich :

- Documentation officielle de Rich
- Référence rapide des balises de style
- Exemples de code avec Rich sur GitHub

Essayez ces fonctionnalités dans vos projets et vous verrez combien elles peuvent transformer l'affichage de vos scripts Python!

Exercices

Fiche personnage D&D

Vous allez recréer une version simplifiée et colorée de votre fiche personnage D&D à l'aide de Rich:

```
╔═══════════════════╗
║ Votre personnage: ║
╚═══════════════════╝
 - Nom: Prof
 - Race: Humain
 - Classe: Guerrier
 - Points d'XP:  450
 - Niveau:  2
 - Objet Magique: Excalibur

╔═══════════════════╗
║ Caractéristiques: ║
╚═══════════════════╝
 - Force:         10
 - Dextérité:     18
 - Constitution:  17
    BONUS:        +3
 - Intelligence:  6
 - Sagesse:       19
 - Charisme:      13
 - Points de vie: 13
```

1. Encadrez les titres de sections
2. Le nom de la caratéristique doit être affiché en blanc.
3. La valeur doit être affichée en jaune

Etape 2:

1. Utilisez des couleurs différentes pour chaque race
2. Affichez les valeurs des caractéristiques <10 en rouge, les autres en vert

Bien débuter: Python au Lycée - Vidéos

Voici une série de vidéos d'introduction en français. Elles te permettront de bien débuter en Python.

Afin de profiter au maximum de ces vidéos et pour optimiser ton apprentissage, il est important de noter qu'**il est impossible d'apprendre à programmer juste en regardant des vidéos**. **LA MEILLEURE SOLUTION**: Ouvre un éditeur de code Python (Thonny ou VS Code par exemple) et **code en même temps que l'instructeur!**

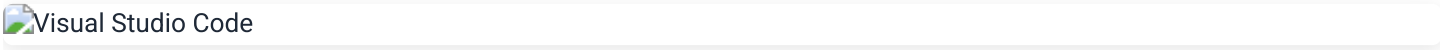
Vidéos "Python au lycée"

- 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 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.

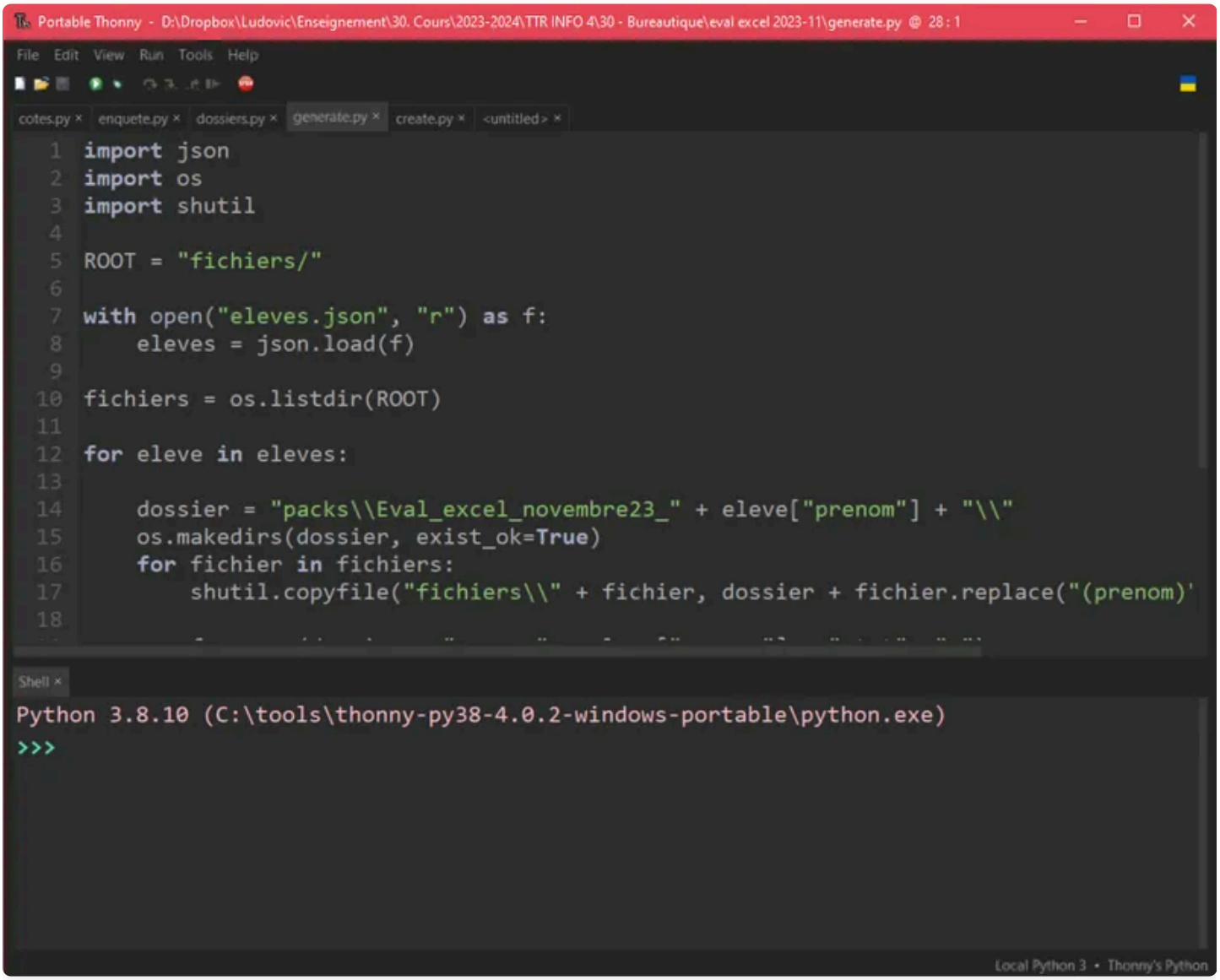
Configuration de Visual Studio Code pour Python

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

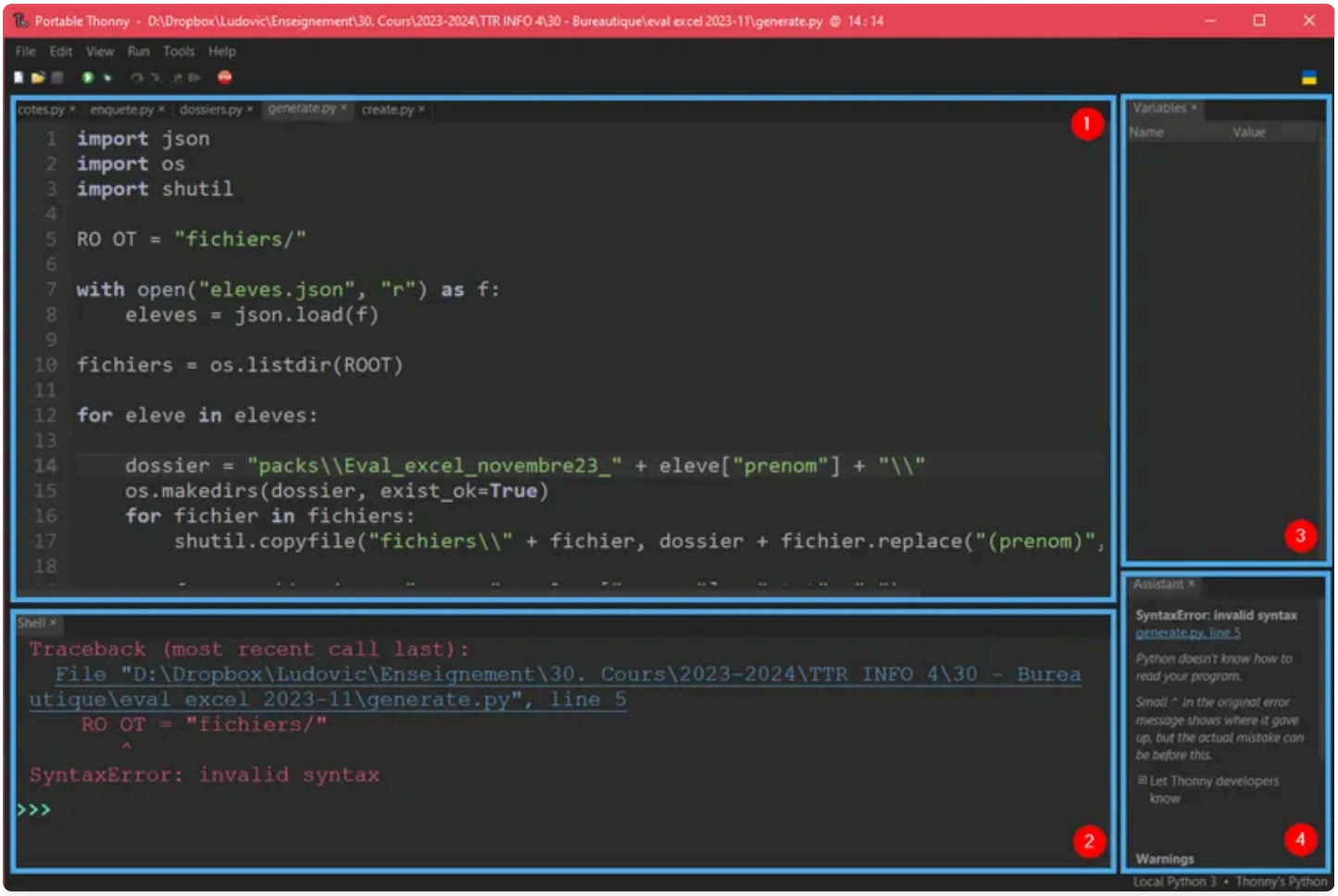
Thonny est livré avec un gestionnaire de packages/modules intégré qui facilite l'installation de modules Python supplémentaires directement depuis l'IDE, ce qui vous permet d'étendre les fonctionnalités de Python selon vos besoins.

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

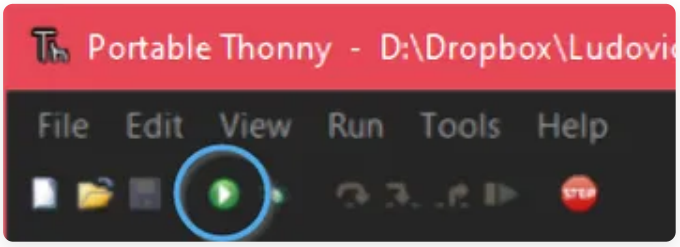
Téléchargez et installez Thonny à partir du site officiel de Thonny. Une fois l'installation terminée, lancez Thonny depuis votre menu d'applications.

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

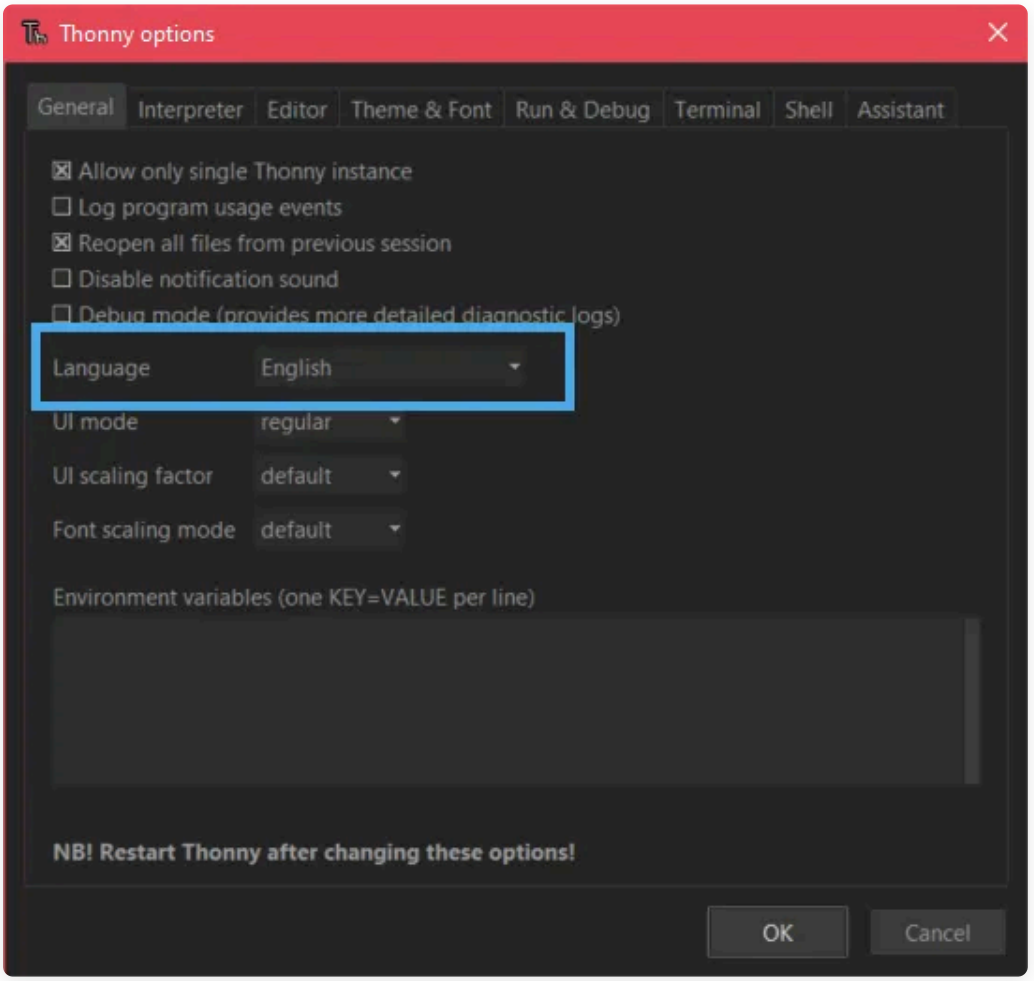
Si vous avez besoin de fonctionnalités supplémentaires, utilisez le gestionnaire de packages intégré pour installer les modules Python nécessaires à votre projet.

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.



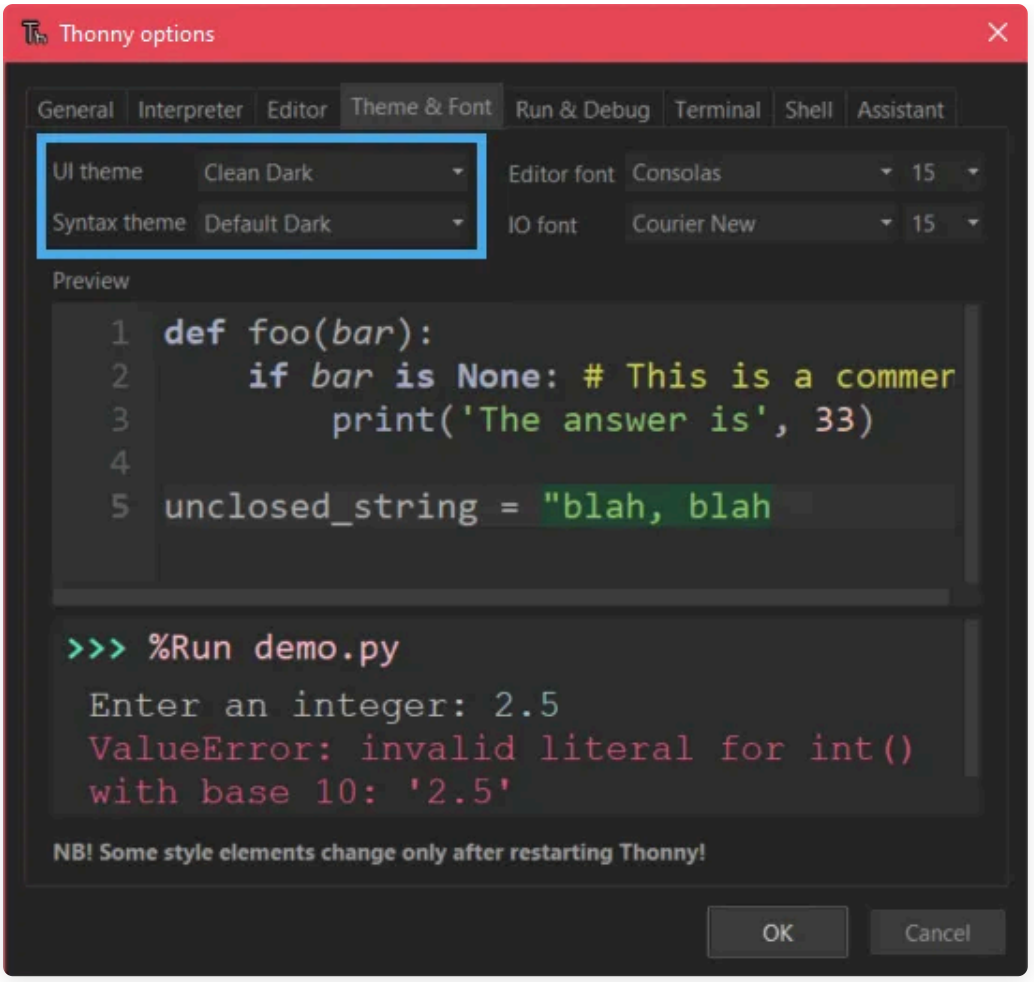
Vous devrez redémarrer Thonny pour que ce changement soit pris en compte.

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 - Déboguer son code

> *Le débogage de code est une compétence essentielle pour tout développeur informatique. Qu'il s'agisse de résoudre des erreurs de syntaxe simples ou de traquer des bogues complexes, savoir comment déboguer efficacement son code peut faire toute la différence entre un projet réussi et des heures de frustration. Dans cet article, nous allons explorer les concepts de base du débogage de code, ainsi que comment utiliser les outils de débogage disponibles dans l'environnement de développement intégré (IDE) Thonny.*

Comprendre le débogage de code

Le débogage de code consiste à **identifier**, **localiser** et **résoudre les erreurs** (ou "bugs") présentes dans un programme informatique. Ces erreurs peuvent prendre différentes formes, telles que des erreurs de syntaxe, des erreurs logiques ou des comportements inattendus du programme. Le processus de débogage implique généralement l'utilisation d'outils spécifiques, tels que des débogueurs, des impressions de messages de débogage ou des tests unitaires, pour isoler et corriger ces erreurs.

Les étapes du débogage de code

1. **Reproduire le problème**: La première étape pour déboguer un code est de reproduire le problème. Cela implique d'**identifier les conditions ou les actions** qui provoquent l'erreur ou le comportement inattendu dans le programme.
2. **Isoler la source du problème**: Une fois que le problème a été reproduit, l'étape suivante consiste à **isoler la source du problème** en identifiant les lignes de code spécifiques ou les parties du programme qui contribuent à l'erreur.
3. **Utiliser des outils de débogage**: Les outils de débogage, tels que les débogueurs intégrés aux IDE, sont **extrêmement utiles** pour examiner l'exécution du programme, surveiller les valeurs des variables et détecter les erreurs à l'exécution.
4. **Apporter des modifications et tester**: Une fois que la source du problème a été identifiée, le développeur peut apporter des modifications au code pour **corriger l'erreur**. Il est important de tester ces modifications pour s'assurer qu'elles résolvent effectivement le problème sans introduire de nouveaux bugs.

Déboguer son code dans Thonny

Thonny est un IDE convivial et puissant pour Python, qui offre plusieurs outils de débogage pour aider les développeurs à identifier et à résoudre les erreurs dans leur code.

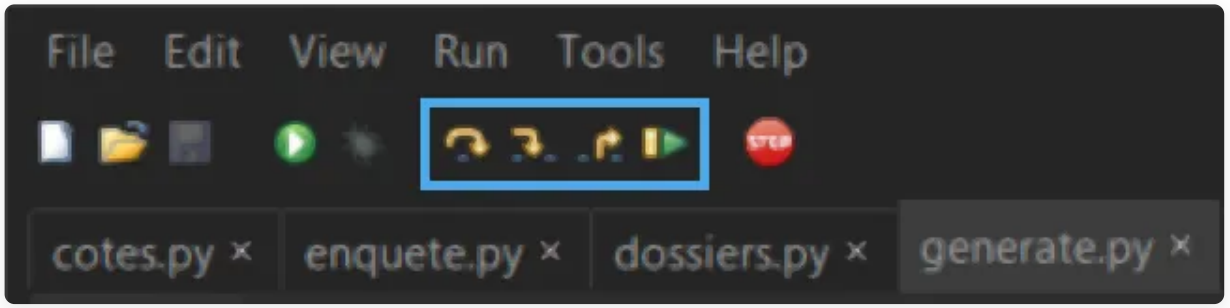
Thonny comprend un débogueur intégré qui permet aux développeurs d'exécuter **leur code pas à pas**, autrement dit **ligne par ligne**, d'**observer les valeurs des variables** à chaque étape, et de **mettre en pause l'exécution du programme** pour inspecter l'état du code.

Pour utiliser le débogueur dans Thonny, il suffit d'exécuter votre programme en **mode "debug"** grâce au bouton de la barre d'outils et/ou de placer des **points d'arrêt** à des endroits stratégiques dans le code, puis de lancer le programme en mode débogage.

Les différentes possibilités d'exécution pas à pas dans Thonny

L'exécution pas à pas, également connue sous le nom de "step-by-step execution" en anglais, est une fonctionnalité essentielle des débogueurs intégrés dans les environnements de développement intégrés (IDE) comme Thonny. Cette fonctionnalité permet aux développeurs **d'exécuter leur code ligne par ligne**, en observant et en analysant l'état du programme à chaque étape.

Ces fonctionnalités sont accessibles via la barre d'outils (et les raccourcis clavier correspondants):



MODE "STEP OVER"



Thonny offre la possibilité d'exécuter le code pas à pas en mode "**Step Over**" (étape suivante). Lorsque les développeurs utilisent cette option, **le débogueur exécute la ligne de code actuelle, puis s'arrête à la ligne suivante, sans entrer dans les appels de fonction ou de méthode**. Cela permet aux développeurs de suivre l'exécution de leur code de manière linéaire, en évitant les détails internes des fonctions ou des méthodes appelées.

MODE "STEP INTO"



Thonny propose également l'option d'exécuter le code pas à pas en mode "**Step Into**" (entrer dans). Avec cette option, le débogueur **entre dans chaque appel de fonction ou de méthode rencontré** pendant l'exécution du code, permettant ainsi aux développeurs d'inspecter le comportement interne de ces fonctions ou méthodes. Cela peut être utile pour comprendre le flux d'exécution du programme et identifier les erreurs potentielles à l'intérieur des fonctions appelées.

MODE "STEP OUT"



Enfin, Thonny offre la possibilité d'exécuter le code pas à pas en mode "**Step Out**" (sortir). Lorsque les développeurs utilisent cette option, le débogueur **exécute le reste du code de la fonction actuelle jusqu'à ce qu'il atteigne la ligne de retour ou la fin de la fonction**, puis s'arrête à la ligne suivante dans la fonction appelante. Cela permet aux développeurs de passer rapidement à la prochaine étape du processus de débogage, en évitant de parcourir chaque ligne de code à l'intérieur de la fonction actuelle.

En utilisant ces différentes options d'exécution pas à pas dans Thonny, les développeurs peuvent analyser et déboguer leur code de manière efficace et précise, ce qui contribue à la création de logiciels de haute qualité et fiables.

MODE "RESUME"



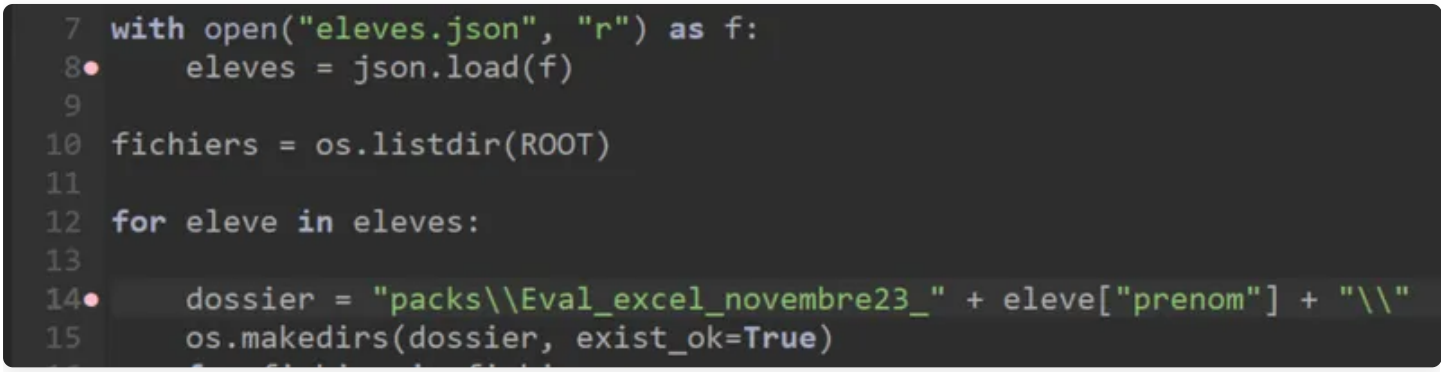
Si vous avez terminé de déboguer votre code, vous pouvez demander à Thonny de continuer à exécuter (resume) votre pro-

gramme jusqu'à la fin, sans s'arrêter.

▸ Les points d'arrêts (GB *breakpoints*)

Les points d'arrêt, également appelés **breakpoints** en anglais, sont des outils essentiels dans le processus de débogage de code. Dans Thonny, les développeurs peuvent placer des points d'arrêt à des endroits spécifiques de leur code où ils souhaitent mettre en pause l'exécution du programme afin d'inspecter l'état des variables et des données.

Pour ce faire, il suffit de cliquer sur le numéro de la ligne de code correspondante. Un marqueur apparaît dans la marge:



Une fois que le programme atteint un point d'arrêt, l'**exécution est suspendue automatiquement** et le développeur peut examiner le contexte de l'exécution, observer les valeurs des variables et évaluer l'état du code.

Les points d'arrêt permettent aux développeurs de suivre pas à pas l'exécution de leur programme, ce qui est particulièrement utile pour identifier les erreurs et les comportements inattendus. En plaçant stratégiquement des points d'arrêt dans leur code, les développeurs peuvent **cibler spécifiquement les parties du programme qu'ils souhaitent examiner** de plus près, ce qui accélère le processus de débogage et améliore l'efficacité globale du développement logiciel.

▸ Affichage des messages de débogage

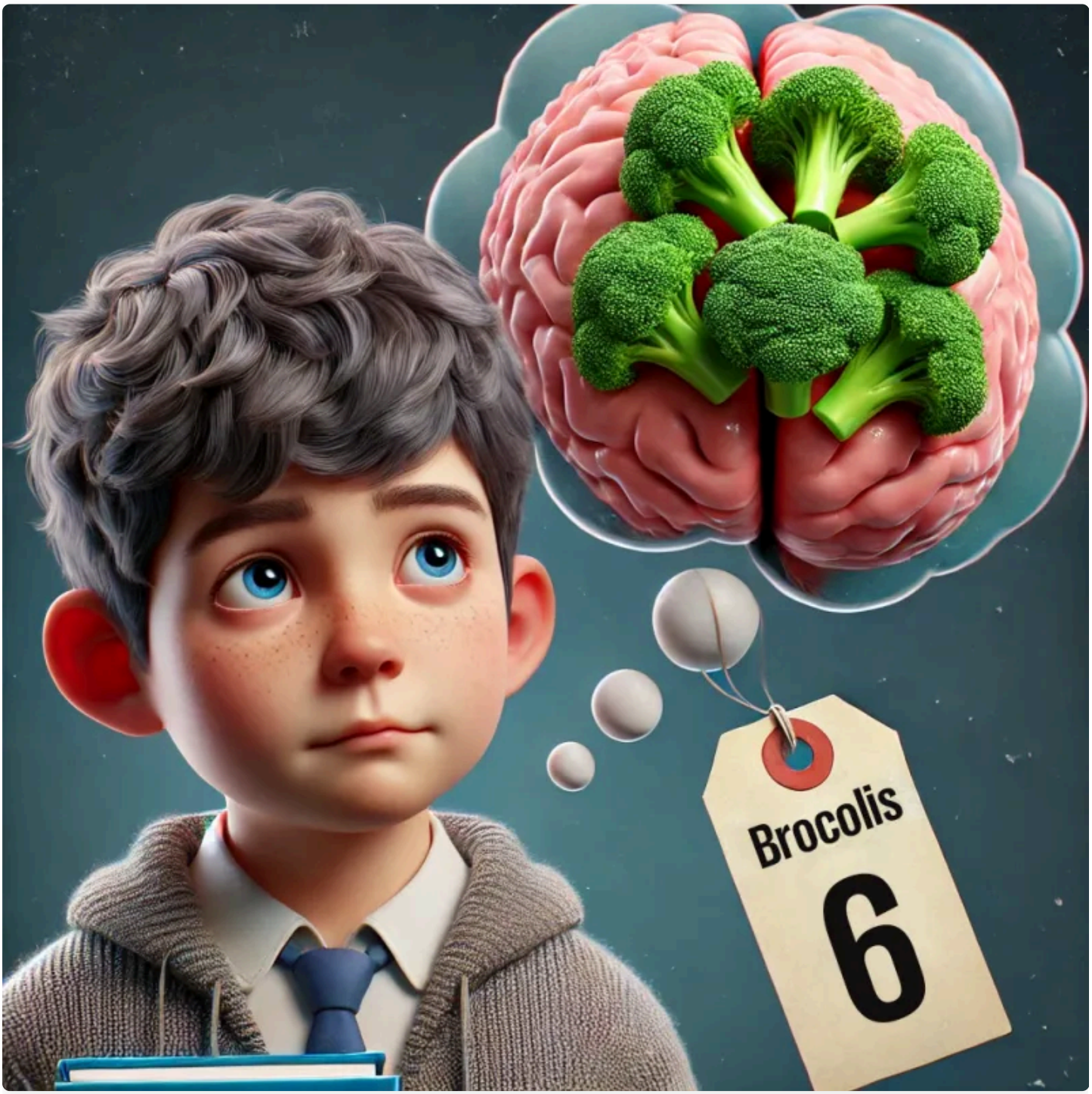
En plus du débogueur intégré, Thonny permet également aux développeurs d'insérer des messages de débogage dans leur code à l'aide de la fonction `print()`. Ces messages peuvent être utilisés pour afficher des informations sur l'exécution du programme à des points spécifiques, ce qui peut aider à identifier les erreurs ou les comportements inattendus.

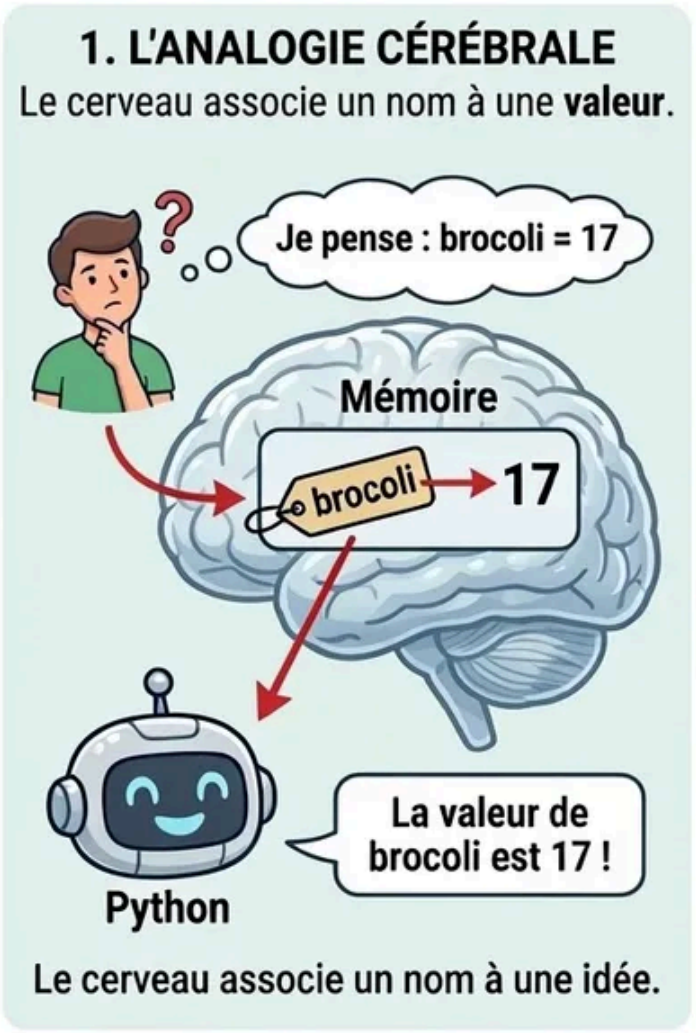
Conclusion

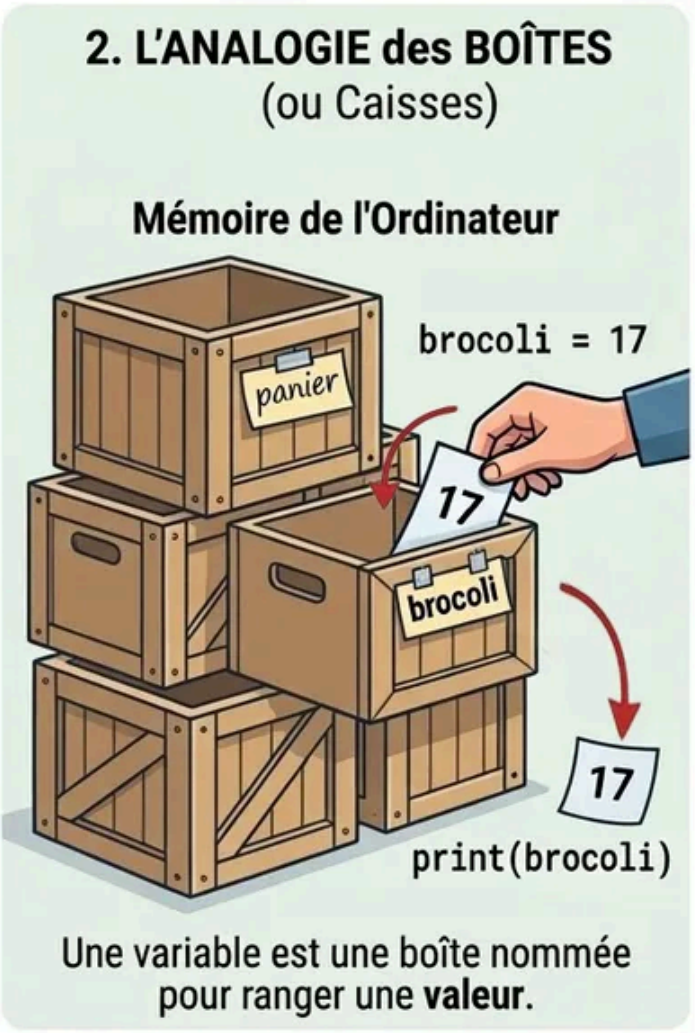
Le débogage de code est une compétence essentielle pour tout développeur informatique. En comprenant les concepts de base du débogage et en utilisant les outils appropriés, les développeurs peuvent identifier et résoudre efficacement les erreurs dans leur code, ce qui contribue à la création de logiciels de haute qualité et fiables. Avec son débogueur intégré et ses autres outils de débogage, Thonny est un excellent choix pour les développeurs Python qui cherchent à améliorer leur efficacité et leur précision lors du débogage de leur code.

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**.

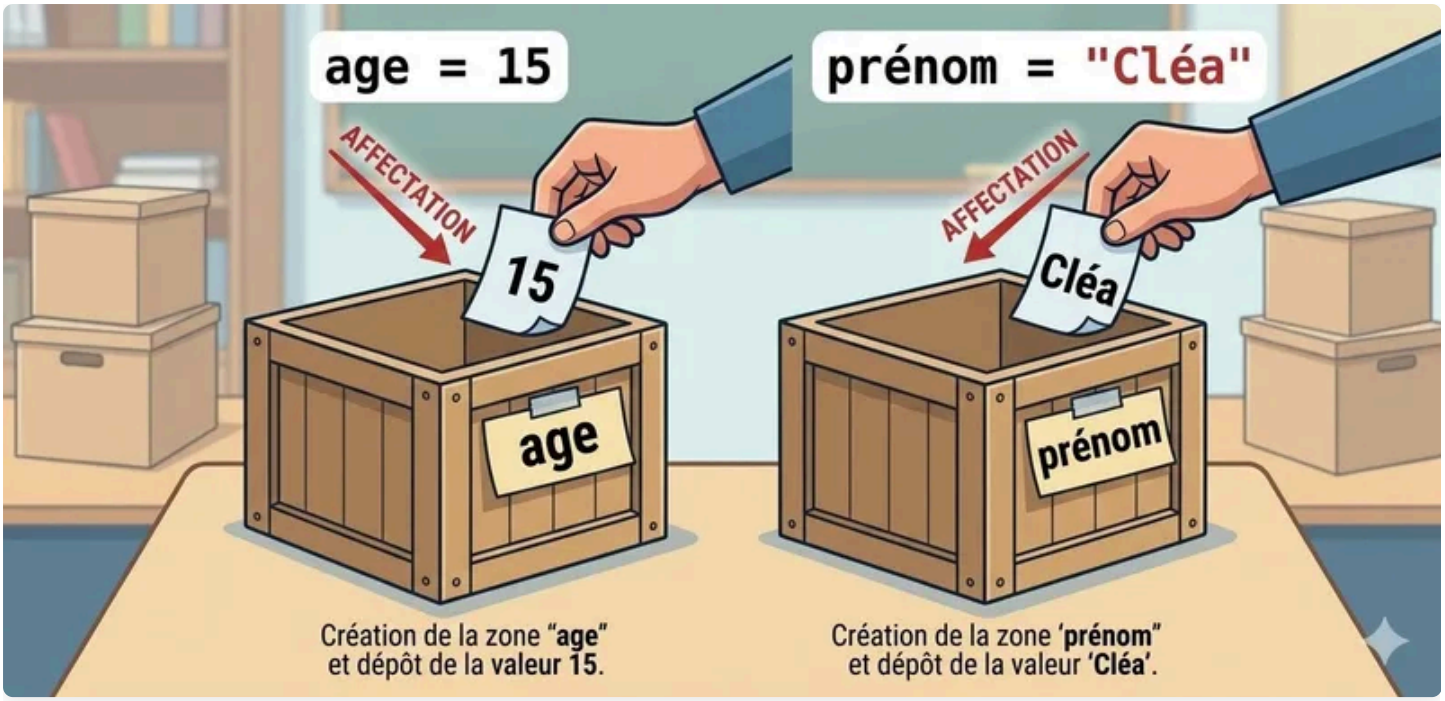
Premier exemple

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`,

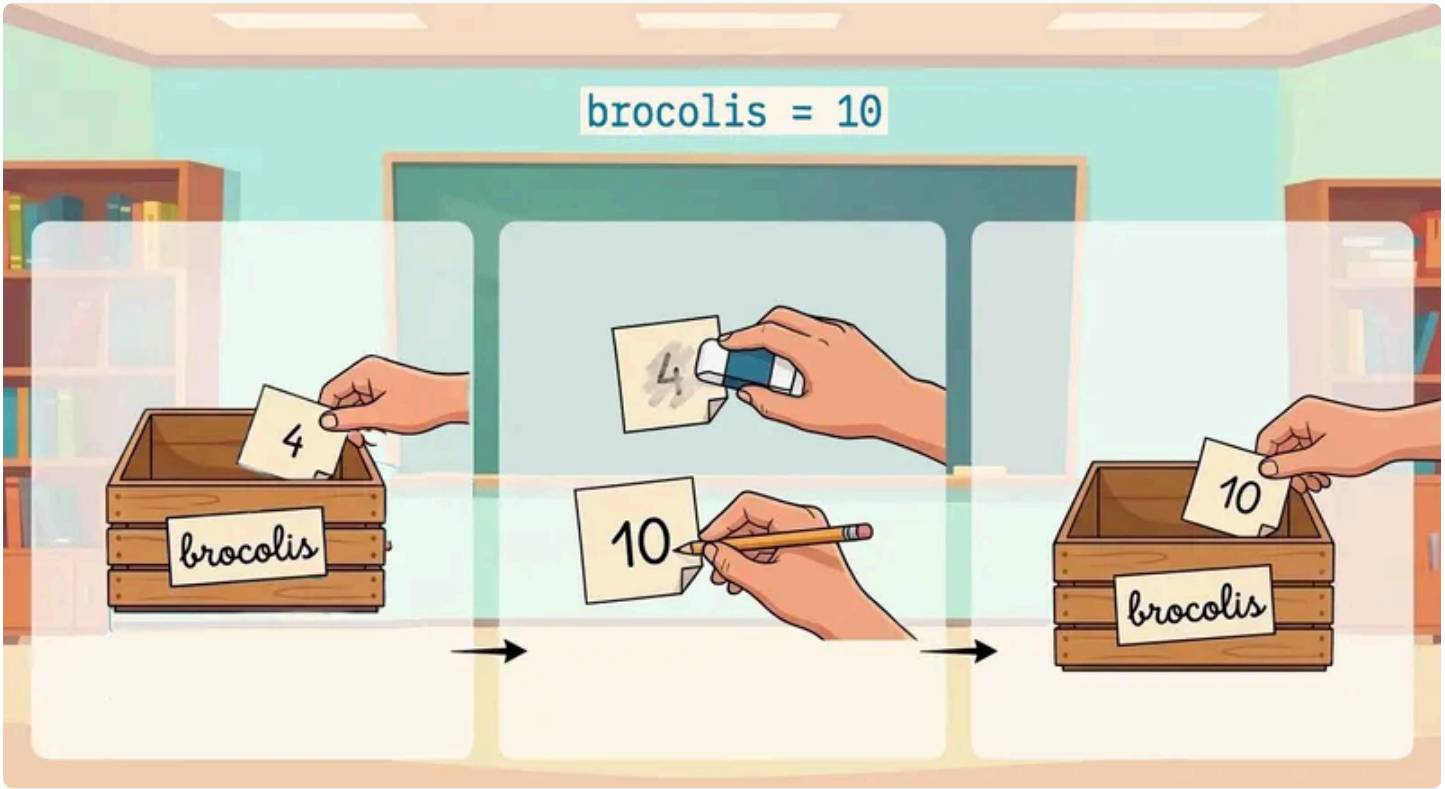


Ensuite, on pourra récupérer ces valeurs pour les afficher ou les utiliser dans un calcul par exemple.

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.

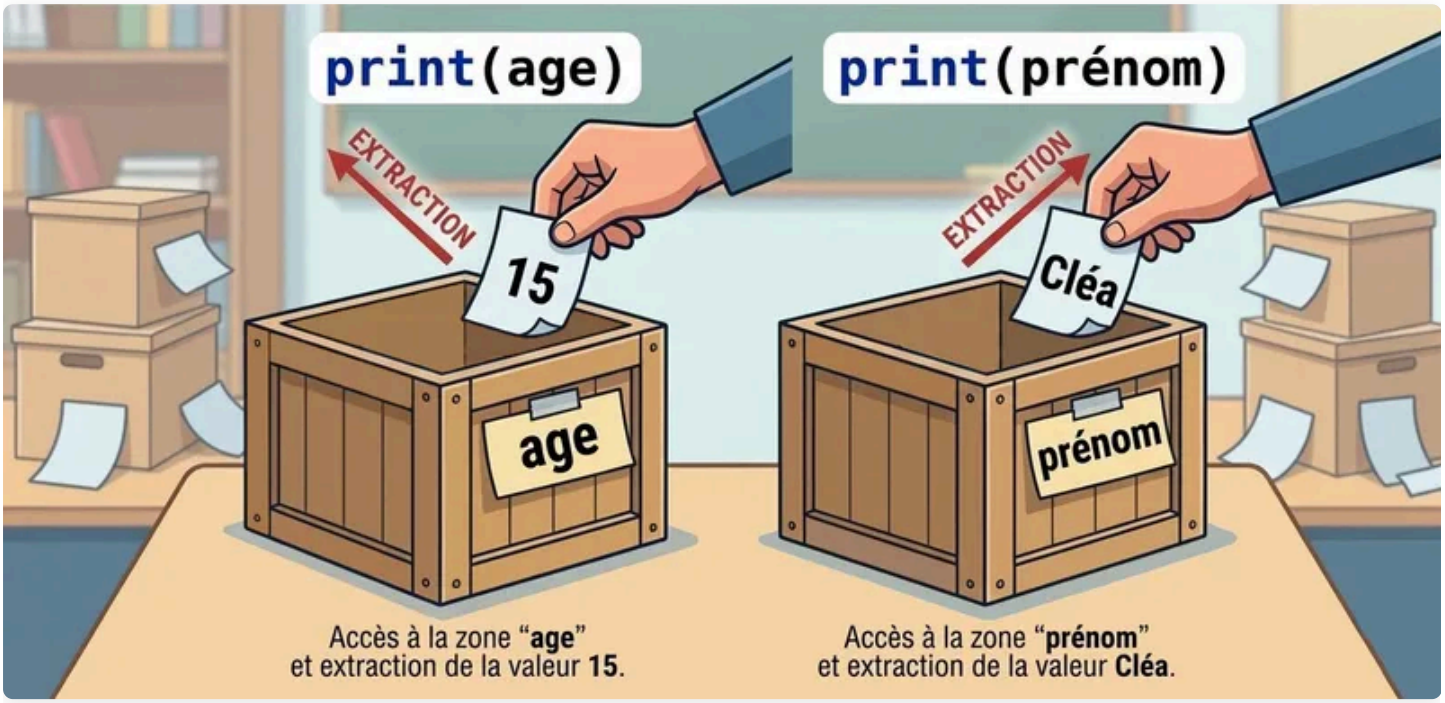
Cela peut peut être dû au fait que:

- 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
```

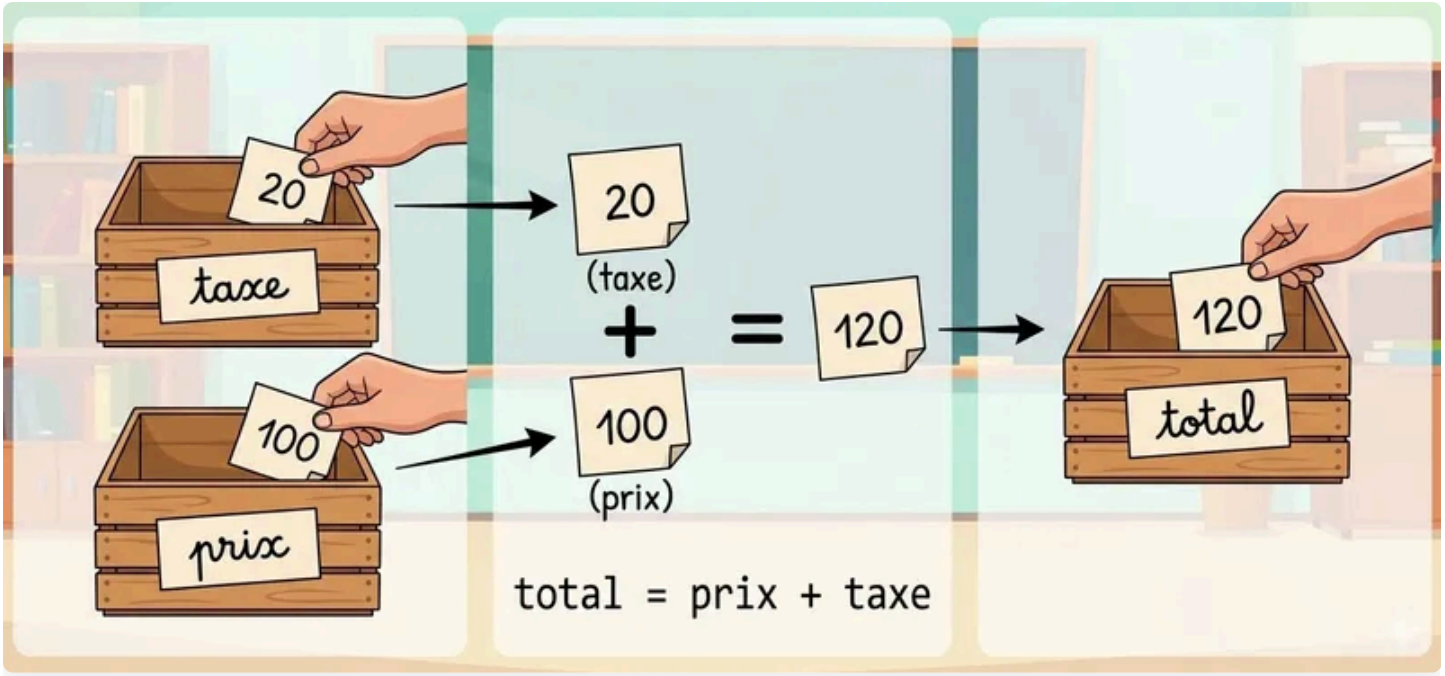


Explications

- `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.

🎯 Python: variables, opérations, print() et input(): Objectifs

> *Ce document définit les objectifs progressifs liés à la compréhension et à la manipulation des variables, des types, des chaînes de caractères, des opérations mathématiques et des entrées/sorties en Python.*

A. Variables : définition et fonctionnement

Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'écrire une variable et de lui affecter une valeur simple.	☐	☐	☐	
Je suis capable d'afficher la valeur d'une variable avec `print()`.	☐	☐	☐	
Je suis capable de modifier la valeur d'une variable existante.	☐	☐	☐	
Je sais reproduire un exemple donné contenant plusieurs variables.	☐	☐	☐	

Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer ce qu'est une variable (zone mémoire associée à un nom).	☐	☐	☐	
Je sais expliquer ce que signifie le symbole `=` en Python.	☐	☐	☐	
Je suis capable de décrire ce qui se passe lorsqu'une variable change de valeur.	☐	☐	☐	
Je sais différencier une variable d'une valeur littérale.	☐	☐	☐	

Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser plusieurs variables pour résoudre un problème simple.	☐	☐	☐	
Je suis capable d'anticiper le résultat d'un programme utilisant plusieurs affectations successives.	☐	☐	☐	
Je suis capable d'expliquer un code simple écrit par un autre élève.	☐	☐	☐	

B. Types de variables

Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de créer une variable contenant un entier.	☐	☐	☐	
Je suis capable de créer une variable contenant un nombre décimal.	☐	☐	☐	
Je suis capable de créer une variable contenant une chaîne de caractères.	☐	☐	☐	
Je suis capable de créer une variable contenant un booléen.	☐	☐	☐	

Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais identifier le type d'une variable à partir de sa valeur.	☐	☐	☐	
Je sais expliquer la différence entre 25 et `"25"`.	☐	☐	☐	
Je sais expliquer le rôle des guillemets.	☐	☐	☐	
Je sais expliquer à quoi sert un booléen.	☐	☐	☐	

Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de choisir le type adapté à une situation donnée.	☐	☐	☐	
Je suis capable de corriger une erreur liée à une confusion de types.	☐	☐	☐	
Je suis capable d'expliquer les conséquences d'un mauvais choix de type dans un contexte professionnel.	☐	☐	☐	

C. Chaînes de caractères (str)

Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de déclarer une chaîne de caractères avec des guillemets.	☐	☐	☐	
Je suis capable d'afficher une chaîne avec `print()`.	☐	☐	☐	
Je suis capable de stocker un texte dans une variable.	☐	☐	☐	
Je suis capable de concaténer deux chaînes avec l'opérateur `+`.	☐	☐	☐	

▸ Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais définir ce qu'est une chaîne de caractères.	☐	☐	☐	
Je sais expliquer la différence entre addition numérique et concaténation de chaînes.	☐	☐	☐	
Je sais expliquer pourquoi `"Bonjour" + "Pierre"` fonctionne.	☐	☐	☐	
Je sais expliquer pourquoi `"Age : " + 25` provoque une erreur.	☐	☐	☐	

▸ Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de construire un message personnalisé à partir de plusieurs variables.	☐	☐	☐	
Je suis capable d'utiliser `str()` pour convertir un nombre avant concaténation.	☐	☐	☐	
Je suis capable de produire un affichage clair combinant texte et données numériques.	☐	☐	☐	

D. Affichage avec `print()`

▸ Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'afficher un texte avec `print()`.	☐	☐	☐	
Je suis capable d'afficher une variable.	☐	☐	☐	
Je suis capable d'afficher plusieurs éléments séparés par une virgule.	☐	☐	☐	
Je suis capable de reproduire un affichage du type `print("Périmètre :", perimetre)`.	☐	☐	☐	

▸ Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer la différence entre `,` et `+` dans `print()`.	☐	☐	☐	
Je sais expliquer pourquoi une conversion est parfois nécessaire.	☐	☐	☐	
Je suis capable de prédire le résultat d'une instruction `print()`.	☐	☐	☐	

▸ Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de produire un affichage structuré et lisible.	☐	☐	☐	
Je suis capable d'améliorer la qualité d'affichage d'un programme existant.	☐	☐	☐	

E. Entrée utilisateur avec `input()`

▸ Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser `input()` avec un message clair terminé par `:` ou `?`.	☐	☐	☐	
Je suis capable de stocker la valeur saisie dans une variable.	☐	☐	☐	

▸ Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer que `input()` renvoie une chaîne de caractères.	☐	☐	☐	
Je sais expliquer pourquoi une conversion est nécessaire pour calculer.	☐	☐	☐	
Je suis capable d'utiliser `int()` et `float()` pour convertir une entrée.	☐	☐	☐	

▸ Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de demander plusieurs valeurs et de les utiliser dans un calcul.	☐	☐	☐	
Je suis capable d'anticiper une erreur liée à une absence de conversion.	☐	☐	☐	
Je suis capable de concevoir une interaction utilisateur claire et professionnelle.	☐	☐	☐	

F. Manipuler les nombres et les opérateurs mathématiques

▸ Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser `+`, `-`, `*`, `/`.	☐	☐	☐	
Je suis capable d'utiliser `**`, `//`, `%`.	☐	☐	☐	

Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer la différence entre `/` et `//`.	☐	☐	☐	
Je sais expliquer le rôle du modulo `%`.	☐	☐	☐	
Je suis capable de prédire le résultat d'une expression mathématique.	☐	☐	☐	

Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de résoudre un problème concret à l'aide d'expressions combinées.	☐	☐	☐	
Je suis capable d'optimiser une expression pour la rendre plus lisible.	☐	☐	☐	

G. Bien nommer ses variables

Niveau 1 — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'écrire un nom respectant la syntaxe Python.	☐	☐	☐	
Je suis capable d'utiliser un nom explicite simple.	☐	☐	☐	

Niveau 2 — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer l'importance de noms clairs et cohérents.	☐	☐	☐	

Niveau 3 — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'améliorer la lisibilité d'un programme en renommant correctement les variables.	☐	☐	☐	
Je suis capable d'expliquer l'importance de la lisibilité dans un contexte professionnel.	☐	☐	☐	

H. Autonomie / Transfert

Niveau 3 — Généraliser / Transférer

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'écrire un programme interactif combinant variables, chaînes, calculs et affichage.	☐	☐	☐	
Je suis capable d'anticiper les erreurs liées aux types et aux conversions.	☐	☐	☐	
Je suis capable de transférer ces compétences vers un autre langage.	☐	☐	☐	

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 !

Les types de variables

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.



Les types de variables en Python

Objectifs

- Rappel de ce qu'est une variable et à quoi elle sert.
- Découvrir les **types de données de base en Python**.
- Apprendre à identifier et utiliser correctement chaque type.
- Comprendre pourquoi le type est important dans un programme.

RAPPEL 💡 Une **variable** est comme une boîte dans laquelle tu ranges une valeur. Chaque boîte a un **nom** (ex. : `age`, `nom`, `prix`) et contient une **valeur** (ex. : `15`, `"Alice"`, `9.99`).

En Python, tu crées une variable simplement en écrivant :

```
age = 15
nom = "Alice"
prix = 9.99
```

Ici :

- `age` contient le nombre entier `15`
- `nom` contient du texte `"Alice"`
- `prix` contient un nombre décimal `9.99`

Les principaux types de variables en Python

1. Les entiers (`int`)

Ce sont des nombres **sans virgule** (positifs ou négatifs). Exemples :

```
a = 42
b = -7
```

→ Utilisés pour compter des objets, représenter un âge, un score…

---

2. Les nombres décimaux (`float`)

Ce sont des nombres **à virgule** (en Python, on écrit avec un point). Exemples :

```
temperature = 19.5
pi = 3.14159
```

→ Utilisés pour des mesures, des calculs scientifiques ou financiers.

---

3. Les chaînes de caractères (`str`)

Ce sont des **textes** entourés de guillemets (`" "` ou `' '`). Exemples :

```
prenom = "Alice"
phrase = 'Bonjour tout le monde'
```

→ Utilisés pour afficher des messages, stocker des noms, etc.

---

4. Les booléens (`bool`)

Ils ne peuvent prendre que **deux valeurs** : `True` (vrai) ou `False` (faux). Exemples :

```
est_connecte = True
a_faim = False
```

→ Utilisés pour représenter une condition (oui/non, allumé/éteint, etc.).

---

5. Les listes (`list`)

Ce sont des **ensembles de valeurs** rangées dans un ordre. Exemples :

```
notes = [12, 15, 18, 9]
prenoms = ["Alice", "Bob", "Charlie"]
```

→ Utilisées pour regrouper plusieurs informations sous un seul nom.

** On les apprendra plus tard**, mais comme ça tu sais que ça existe 😉

Pourquoi le type est important ?

Le type de la variable détermine **ce que tu peux faire avec elle**.

Exemple :

```
age = 15
nom = "Alice"

print(age + 5)     # ok → 20
print(nom + " !")  # ok → Alice !
print(age + nom)   # erreur → on ne peut pas additionner un nombre et un texte
```

À retenir

1. Une variable est un **nom qui stocke une valeur**.

2. Les types de base en Python sont :
    - `int` (entiers),
    - `float` (décimaux),
    - `str` (chaînes de caractères),
    - `bool` (vrai/faux),
    - `list` (listes).
3. Le type détermine les **opérations possibles** sur la variable.

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.

Manipuler valeurs et variables: exemples guidés 🚀

> *Exemples pour explorer la manipulation des variables*

Petits calculs : `print(7*44)`

PLACES DANS UNE SALLE

Combien de personnes une salle de cinéma peut-elle accueillir?

```
# 7 rangées de 44 sièges
print(7 * 44)
```

Étapes

- On a 7 rangées.
- Chaque rangée a 44 sièges.
- Total = 7 × 44 → on affiche le total.

GOBELETS POUR UN ÉVÉNEMENT

On achète 6 packs de 30 gobelets au magasin.

```
# 6 packs de 30 gobelets
print(6 * 30)
```

Étapes

- 6 packs.
- 30 gobelets par pack.
- Total = 6 × 30.

AUTOCOLLANTS IMPRIMÉS

On a 10 feuilles avec chacune 44 autcollants. Combien avons-nous d'autocollants?

```
# 10 feuilles de 44 autocollants
print(10 * 44)
```

Étapes

- 10 feuilles.
- 44 autocollants par feuille.
- Total = 10 × 44.

MINUTES D'ENTRAÎNEMENT

Je m'entraîne pendant 60 minutes tous les jours de la semaine. Pendant combien de minutes je m'entraîne ?

```
# 7 séances de 60 minutes
print(7 * 60)
```

Étapes

- 7 séances.
- 60 minutes par séance.
- Total = 7 × 60 minutes.

DISTANCE EN TOURS DE PISTE

```
# 7 tours de 44 mètres (petite piste)
print(7 * 44)
```

Étapes

- 7 tours.
- 44 m par tour.
- Distance totale = 7 × 44 m.

Print avec variables

PRIX TOTAL

```
prix_total = 12.5
print(prix_total)
```

Étapes

- `prix_total` est une variable.
- Elle contient 12.5 (euros).
- `print(prix_total)` affiche la valeur.

DISTANCE

```
distance_km = 8
print(distance_km)
```

Étapes

- `distance_km` contient 8.
- On affiche la distance.

NOMBRE DE PARTICIPANTS

Un évènement est organisé sur 4 jours. Chaque jour il peut accueillir 100 participants.

```
participants = 4 × 100
print(participants)
```

Étapes

- `participants` contient 400.
- On affiche le nombre.

Étapes

- On crée les variables
- On calcul le résultat...
- ... et on l'affiche

Assignation de valeurs et `print` (avec calcul simple et résultat)

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.

Exercice — Calcul du prix des courses

Tu fais les courses du jour. Chaque article a un **prix**, et tu en achètes une **quantité** donnée.

Consigne

- Identifie les informations nécessaires pour calculer le prix total.
- Stocke chaque information dans une variable.
- Calcule le total à payer.
- Affiche le résultat avec `print`.

```
prix = 3
quantite = 7
total = prix * quantite
print(total)
```

Zone de prédiction

👉 Total affiché : `_____`

Aide à la compréhension

- `prix` représente le prix d'un article
- `quantite` représente le nombre d'articles
- `total` correspond au calcul : prix × quantité

Exercice — Distance parcourue

Un véhicule roule à vitesse constante pendant un certain temps.

Consigne

- Stocke la vitesse et le temps dans des variables.
- Calcule la distance parcourue.
- Affiche la distance.

```
vitesse = 60
temps = 2
distance = vitesse * temps
print(distance)
```

Zone de prédiction

👉 Distance affichée : _____

Aide à la compréhension

- La distance se calcule avec : vitesse × temps
- Le résultat est exprimé en kilomètres

Exercice — Recette d'un événement

Lors d'un événement, des tickets sont vendus.

Consigne

- Stocke le prix d'un ticket.
- Stocke le nombre de tickets vendus.
- Calcule la recette totale.
- Affiche le montant.

```
prix_ticket = 11
tickets = 40
recette = prix_ticket * tickets
print(recette)
```

Zone de prédiction

👉 Recette affichée : _____

Aide à la compréhension

- Chaque ticket rapporte le même montant
- La recette est un total d'argent gagné

Exercice — Calcul d'un salaire

Une personne travaille plusieurs heures avec un taux horaire fixe.

Consigne

- Stocke le taux horaire dans une variable.
- Stocke le nombre d'heures travaillées.
- Calcule le salaire total.
- Affiche le salaire.

```
taux = 14
heures = 5
salaire = taux * heures
print(salaire)
```

Zone de prédiction

👉 Salaire affiché : _____

Aide à la compréhension

- Le salaire dépend du nombre d'heures
- Plus on travaille, plus le salaire augmente

Exercice — Pages imprimées

Un document est imprimé en plusieurs copies.

Consigne

- Stocke le nombre de pages d'un document.
- Stocke le nombre de copies.
- Calcule le nombre total de pages imprimées.
- Affiche le résultat.

```
pages = 18
copies = 6
total_pages = pages * copies
print(total_pages)
```

Zone de prédiction

👉 Nombre de pages affiché : _____

Aide à la compréhension

- Chaque copie contient le même nombre de pages
- Le total correspond à toutes les pages imprimées

Message clé

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

Changer la valeur d'une variable (la "mise à jour")

STOCK QUI DIMINUE

```
stock = 20
print(stock)

stock = 15
print(stock)
```

Étapes

- Au début : stock = 20 → on affiche 20.
- Ensuite on *remplace* la valeur : stock = 15 → on affiche 15.

DISTANCE QUI AUGMENTE (ON AJOUTE UN TRAJET)

```
distance = 5
print(distance)

distance = distance + 3
print(distance)
```

Zone de prédiction --> 👉 `Valeur affichée : _____`

Étapes

- Distance initiale : 5 km.
- Puis on ajoute 3 km → 8 km.
- La variable prend la nouvelle valeur.

PRIX APRÈS RÉDUCTION

```
prix = 80
print(prix)

prix = prix - 10
print(prix)
```

Zone de prédiction --> 👉 `Valeur affichée : _____`

Étapes

- Prix initial : 80.
- On applique une réduction de 10 → 70.

POINTS DANS UN JEU

```
points = 0
print(points)

points = points + 50
print(points)
```

Étapes

- Départ : 0 point.
- Gain : +50 → 50 points.

TEMPÉRATURE MISE À JOUR

```
temperature = 18
print(temperature)

temperature = 21
print(temperature)
```

Étapes

- Il faisait 18°C.
- Nouvelle mesure : 21°C.
- On affiche les deux valeurs successives.

Valeurs et variables: à toi de jouer 💪🏻

> *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**

Exercice — Courses au supermarché

Tu achètes plusieurs articles identiques.

Consigne

- Tu connais le **prix d'un article**.
- Tu connais la **quantité achetée**.
- Calcule le **prix total à payer**.
- Affiche le total.

Aide

👉 Quelles sont les **2 valeurs indispensables** pour faire le calcul ? 👉 Quel calcul mathématique faut-il utiliser ?

Exercice — Distance parcourue en voiture

Une voiture roule à vitesse constante.

Consigne

- Tu connais la **vitesse** (en km/h).
- Tu connais le **temps de trajet** (en heures).
- Calcule la **distance parcourue**.

- Affiche la distance.

Aide

👉 Distance = vitesse × temps

Exercice — Temps total de sport

Tu fais plusieurs séances de sport identiques.

Consigne

- Tu connais la **durée d'une séance** (en minutes).
- Tu connais le **nombre de séances**.
- Calcule le **temps total d'entraînement**.
- Affiche le résultat.

Exercice — Vente de tickets

Lors d'un événement, des tickets sont vendus.

Consigne

- Tu connais le **prix d'un ticket**.
- Tu connais le **nombre de tickets vendus**.
- Calcule la **recette totale**.
- Affiche le montant gagné.

Exercice — Salaire journalier

Une personne travaille plusieurs heures dans la journée.

Consigne

- Tu connais le **taux horaire**.
- Tu connais le **nombre d'heures travaillées**.
- Calcule le **salaire du jour**.
- Affiche le salaire.

Exercice — Pages à imprimer

Un document est distribué à toute une classe.

Consigne

- Tu connais le **nombre de pages du document**.
- Tu connais le **nombre d'élèves**.
- Calcule le **nombre total de pages à imprimer**.
- Affiche le total.

Exercice — Points dans un jeu

Dans un jeu, un joueur gagne le même nombre de points à chaque niveau.

Consigne

- Tu connais le **nombre de points par niveau**.
- Tu connais le **nombre de niveaux terminés**.
- Calcule le **score total du joueur**.
- Affiche le score.

Exercice — Consommation de carburant

Une machine consomme une quantité fixe de carburant par heure.

Consigne

- Tu connais la **consommation par heure**.
- Tu connais le **nombre d'heures de fonctionnement**.
- Calcule la **consommation totale**.
- Affiche le résultat.

Exercice — Boîtes de rangement

Du matériel est stocké dans des boîtes identiques.

Consigne

- Tu connais le **nombre de boîtes**.
- Tu connais le **nombre d'objets par boîte**.
- Calcule le **nombre total d'objets stockés**.
- Affiche le total.

Exercice — Temps de visionnage

Tu regardes une série avec des épisodes de même durée.

Consigne

- Tu connais la **durée d'un épisode**.
- Tu connais le **nombre d'épisodes regardés**.
- Calcule le **temps total de visionnage**.
- Affiche le temps total.

Voici **5 nouveaux exercices**, **sans code**, sur le **thème de l'astronomie**, formulés exactement dans le même esprit : réflexion → variables → calcul → affichage. Ils sont adaptés à des **débutants complets** et exploitables tels quels en classe.

Exercices – Astronomie : variables et calculs

Dans chaque exercice :

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

Exercice — Distance parcourue par un satellite

Un satellite tourne autour de la Terre à vitesse constante.

Consigne

- Tu connais la **vitesse du satellite** (en km par heure).
- Tu connais le **temps de déplacement** (en heures).
- Calcule la **distance parcourue** par le satellite.
- Affiche la distance.

Aide

👉 Distance = vitesse × temps

Exercice — Tours de la Terre

Un satellite met toujours le même temps pour faire un tour complet de la Terre.

Consigne

- Tu connais le **nombre de tours effectués**.
- Tu connais la **durée d'un tour** (en minutes).
- Calcule le **temps total passé en orbite**.

- Affiche le résultat.

Exercice — Observations astronomiques

Un astronome observe le ciel plusieurs nuits de suite.

Consigne

- Tu connais la **durée d'une observation** (en minutes).
- Tu connais le **nombre de nuits d'observation**.
- Calcule le **temps total d'observation**.
- Affiche le temps total.

Exercice — Panneaux solaires d'un satellite

Un satellite est équipé de plusieurs panneaux solaires identiques.

Consigne

- Tu connais le **nombre de panneaux solaires**.
- Tu connais la **puissance d'un panneau**.
- Calcule la **puissance totale produite**.
- Affiche le résultat.

Exercice — Messages envoyés depuis l'espace

Une sonde spatiale envoie des données vers la Terre à intervalles réguliers.

Consigne

- Tu connais le **nombre de messages envoyés par heure**.
- Tu connais le **nombre d'heures de communication**.
- Calcule le **nombre total de messages envoyés**.
- Affiche le total.

Ces exercices mobilisent : les variables, les opérateurs arithmétiques, `print()`, `round()`, `math`, et le formatage de chaînes — sans aucune boucle ni fonction à définir.

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.

Pourquoi 25 n'est-il pas égal à "25" ?

> *Dans le monde de la programmation, la distinction entre des valeurs telles que 25 (un nombre entier) et "25" (une chaîne de caractères) peut sembler évidente pour les programmeurs expérimentés, mais elle peut être source de confusion pour les débutants. Dans ce chapitre, nous allons expliquer pourquoi 25 et "25" sont deux choses différentes en Python.*

Type de Données

La principale raison pour laquelle 25 n'est pas égal à "25" réside dans le concept de type de données en Python. Chaque valeur en Python appartient à un type de données spécifique, qui définit la nature de cette valeur et comment elle peut être traitée.

- **25 est un nombre entier (int)** : Lorsque nous écrivons simplement `25` dans notre code, Python comprend que nous travaillons avec un **nombre entier**. Les nombres entiers sont des valeurs numériques sans guillemets, et ils sont utilisés pour effectuer des calculs mathématiques.
- **"25" est une chaîne de caractères (str)** : D'autre part, lorsque nous entourons une valeur avec des guillemets, comme `"25"`, Python interprète cette valeur comme une **chaîne de caractères**. Les chaînes de caractères sont utilisées pour représenter du texte et peuvent contenir des lettres, des chiffres, des espaces, des symboles, etc.

Comparaison de Types Différents

La différence fondamentale entre 25 et "25" réside dans leur type de données respectif. **Si vous comparez un nombre entier à une chaîne de caractères, Python considère qu'ils sont différents en raison de leurs types différents**.

Voici un exemple pour illustrer cela :

```
nombre = 25
chaine = "25"

if nombre == chaine:
    print("Les valeurs sont égales.")
else:
    print("Les valeurs ne sont pas égales.")
```

Le résultat de ce code sera "Les valeurs ne sont pas égales." parce que Python compare un nombre entier (`nombre`) à une chaîne de caractères (`chaine`) et constate que leurs types sont différents.

💡 D'une certaine manière, c'est comme si vous écriviez `25` et `vingt-cing` ... on a l'idée que c'est la même chose, mais on voit bien qu'on n'a pas **écrit** la même chose.

Conversion entre Types

Si vous souhaitez comparer ces deux valeurs en tant que nombres, vous devez les convertir en un même type. Vous pouvez utiliser les fonctions `int()` ou `float()` pour convertir une chaîne de caractères en un nombre.

Par exemple :

```
nombre = 25
chaine = "25"

chaine_convertie = int(chaine)  # Convertir la chaîne en un nombre entier

if nombre == chaine_convertie:
    print("Les valeurs sont égales.")
else:
    print("Les valeurs ne sont pas égales.")
```

Maintenant, le résultat sera "Les valeurs sont égales." car nous avons converti la chaîne "25" en un nombre entier, les deux valeurs sont donc comparables.

En conclusion, la différence entre 25 et "25" réside dans leur type de données. 25 est un nombre entier, tandis que "25" est une chaîne de caractères. Lorsque vous effectuez des comparaisons en Python, assurez-vous de prendre en compte le type de données des valeurs que vous comparez, et si nécessaire, effectuez des conversions pour les rendre compatibles. Cela vous permettra de travailler efficacement avec différents types de données dans vos programmes Python.

Demander des valeurs à l'utilisateur

> *La fonction `input()` est l'un des outils les plus basiques mais essentiels que vous rencontrerez en apprenant à programmer en Python. Elle vous permet d'interagir avec l'utilisateur en récupérant des données depuis la console. Dans cet article, nous allons explorer en détail la fonction `input()`, comment l'utiliser efficacement, et comment convertir les données entrées en utilisant `int()` et `float()`.*

Qu'est-ce que la Fonction `input()` ?

En Python, `input()` est une fonction intégrée qui permet à un programme de **recevoir des données entrées par l'utilisateur depuis le clavier**. Elle attend que l'utilisateur saisisse une ligne de texte, la valide en tapant la **touche "entrée"** et retourne cette ligne sous forme de **chaîne de caractères (string)**. Voici un exemple simple :

```
nom_utilisateur = input("Entrez votre nom : ")
print("Bonjour, " + nom_utilisateur + "!")
```

Dans cet exemple, `input("Entrez votre nom : ")` affiche le message "Entrez votre nom :" à l'utilisateur et **attend** que l'utilisateur entre son nom. La valeur entrée est ensuite stockée dans la variable `nom_utilisateur` **sous forme de chaîne de caractères** et affichée dans un message de salutation.

💥 La fonction `input` retourne une chaîne de caractères GB ***string***. Si vous voulez utiliser cette valeur pour effectuer des calculs, il faut d'abord la convertir.

Conversion avec `int()` et `float()`

Lorsque vous utilisez `input()` pour obtenir des données numériques de l'utilisateur, celles-ci sont **toujours stockées sous forme de chaînes de caractères (strings)**. Si vous avez besoin de travailler avec ces données en tant que nombres, vous devez les **convertir** en utilisant les fonctions `int()` ou `float()`.

- `int()` est utilisé pour convertir une chaîne en un nombre entier (int).
- `float()` est utilisé pour convertir une chaîne en un nombre décimal (float).

Voici des exemples de conversion avec `int()` et `float()` :

```
age_texte = input("Entrez votre âge : ")
age = int(age_texte)  # Convertir la chaîne en un entier

prix_texte = input("Entrez le prix : ")
prix = float(prix_texte)  # Convertir la chaîne en un nombre décimal (float)
```

L'utilisation de `int()` et `float()` permet de traiter les données entrées par l'utilisateur comme des nombres, ce qui est souvent nécessaire pour effectuer des calculs ou manipuler des valeurs numériques.

Si l'utilisateur entre des nombres invalides, cela provoquera le plantage de votre programme. Pour l'instant on peut s'en contenter. Partons du principe pour ce cours que l'on reçoit des données valides. Sachez qu'il existe des moyens d'éviter ces problèmes.

Conclusion

La fonction `input()` permet d'interagir avec l'utilisateur et obtenir des données d'entrée dans vos programmes Python. Lorsque vous travaillez avec des données numériques, n'oubliez pas de les convertir en utilisant `int()` ou `float()` pour les manipuler correctement. Avec `input()` et la conversion appropriée, vous pouvez créer des applications interactives et conviviales qui répondent aux besoins de vos utilisateurs.

Quelques exercices

1. **Salutation personnalisée**: Écris un programme qui demande à l'utilisateur son prénom et affiche : `Bonjour, [prénom] !`
2. **Présentation complète**: Demande à l'utilisateur son prénom et son nom, puis affiche une phrase complète : `Tu t'appelles [prénom] [nom].`
3. **Réponse humoristique**: Demande à l'utilisateur son animal préféré et réponds : `[animal] ? C'est un choix intéressant !`
4. **Couleur préférée**: Demande la couleur préférée de l'utilisateur et affiche une phrase : `[couleur] est une belle couleur !`
5. **Deviner une profession**: Demande à l'utilisateur ce qu'il aimerait faire comme métier et réponds : `Devenir [métier] ? Super idée !`

---

1. **Double d'un nombre**: Demande un nombre à l'utilisateur et affiche son double.
2. **Addition simple**: Demande deux nombres à l'utilisateur et affiche leur somme.
3. **Calcul de périmètre**: Demande la longueur d'un côté d'un carré et calcule son périmètre.
4. **Multiplication**: Demande deux nombres à l'utilisateur et affiche leur produit.
5. **Moyenne de trois nombres**: Demande trois nombres et calcule leur moyenne.
6. **Temps en secondes**: Demande un temps en minutes et affiche le nombre de secondes.
7. **Conversion km → miles**: Demande une distance en kilomètres et affiche la conversion en miles : *1 km ≈ 0.621371 miles*
8. **Conversion Celsius → Fahrenheit**: Demande une température en Celsius et affiche la conversion en Fahrenheit: *F = C × 9/5 + 32*

Variables et opérations - Exercices

> *Voici quelques exercices pour pouvoir... t'exercer avec les variables en Python*

Objectifs

- Savoir créer et utiliser des variables en Python.
- Comprendre les types de données de base : nombres entiers (`int`) et nombres décimaux (`float`).
- Réaliser des calculs simples avec les opérateurs (`+` , `-` , `*` , `/` , `//` , `%` , `**`).
- Afficher les résultats avec `print()` .

Exercices

Niveau 1 – Variables simples

1. Crée une variable `age` qui contient ton âge et affiche-la.
2. Crée deux variables `nom` et `prenom` , puis affiche ton prénom et ton nom sur une seule ligne.
3. Stocke la valeur `12` dans une variable `x` et affiche son contenu.
4. Modifie la valeur de `x` en ajoutant 5, puis affiche à nouveau `x` .

Niveau 2 – Opérations de base

5. Crée deux variables `a = 8` et `b = 3` , puis affiche la somme de `a` et `b` .
6. Avec les mêmes variables, affiche la différence (`a - b`).
7. Affiche le produit (`a * b`).
8. Affiche le quotient (`a / b`). Quelle est la différence avec `a // b` ? Teste les deux.
9. Calcule le reste de la division de `a` par `b` (utilise `%`).
10. Calcule `2` à la puissance `5` (utilise `**`).

Niveau 3 – Combiner les variables

11. Crée deux variables `longueur = 7` et `largeur = 4` . Calcule et affiche l'aire du rectangle.
12. Calcule le périmètre de ce rectangle.
13. Stocke le rayon d'un cercle dans une variable `r = 5` . Calcule l'aire du cercle avec la formule `pi * r**2` (utilise `3.14` pour `pi`).
14. Crée une variable `celsius = 20` . Convertis-la en Fahrenheit avec la formule `F = C * 9/5 + 32` .
15. Calcule la moyenne de trois nombres : 12, 18 et 25. Stocke le résultat dans une variable `moyenne` .

© 2025 Mr. Lambrechts

▸ Niveau 4 – Exercices pratiques

16. Crée une variable `minutes = 135`. Convertis-la en heures et minutes (par exemple 2h15).
17. Calcule le prix total de 7 pommes coûtant chacune 0.80€. Affiche le résultat avec un message clair.
18. Un étudiant a eu les notes suivantes : 14, 9, 16, 11. Calcule la moyenne et affiche-la.
19. Crée une variable `salaire = 2000`. Calcule le salaire après une augmentation de 5%.
20. Crée une variable `capital = 1000`. Calcule la valeur après 3 ans avec un intérêt annuel de 4% (formule : `capital * (1 + taux) ** années`).

👉 Chaque exercice peut être testé directement dans un fichier Python avec un simple `print()`.

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 chaines de caractères

> *Les chaînes de caractères en Python sont utilisées pour représenter du texte et sont omniprésentes dans la programmation. Dans cet article, nous explorerons en détail ce qu'est une chaîne de caractères en Python, comment les créer et les utiliser dans vos programmes, avec des exemples concrets à chaque étape.*

Qu'est-ce qu'une Chaîne de Caractères ?

Une chaîne de caractères en Python est une séquence de caractères encadrée par des guillemets simples `' '` ou doubles `" "`. Par exemple :

```
chaine_simple = 'Bonjour'
chaine_double = "à tous"
```

Les chaînes de caractères peuvent contenir n'importe quel caractère, y compris des lettres, des chiffres, des symboles et des espaces.

Les `""` et `''` permettent de définir une chaine sur **une seule ligne**.

Création de Chaînes de Caractères

Pour créer une chaîne de caractères en Python, vous pouvez simplement l'assigner à une variable en utilisant des guillemets simples ou doubles. Voici quelques exemples :

```
chaine1 = 'Hello World'
chaine2 = "Python"
chaine3 = "12345"
```

Sur plusieurs lignes

Pour définir une chaîne de caractères sur plusieurs lignes, il faut utiliser des `""" """` (triples guillemets):

```
texte = """ Cette chaîne est définie
sur plusieurs
lignes
et
ça fonctionne
quand-même !
"""
```

Python: Rechercher dans une chaîne de caractères

> *Dans ce cours tu vas apprendre à utiliser les opérateurs **`in`** et **`not in`** pour vérifier si une **lettre**, un **mot**, ou une **sous-chaîne** est présent(e) dans une autre chaîne de caractères.*

Objectif

Apprendre à utiliser les opérateurs `in` et `not in` pour vérifier si une **lettre**, un **mot**, ou une **sous-chaîne** est présent(e) dans une autre chaîne de caractères.

Principe

L'opérateur `in` répond à une question simple :

> *Est-ce que cet élément se trouve dans cette chaîne ?*

Il renvoie une **valeur booléenne** :

- `True` → si l'élément est présent
- `False` → sinon

Exemple simple

```
mot = "Python"

print("P" in mot)     # True  → 'P' est bien dans "Python"
print("p" in mot)     # False  → 'p' est bien dans "Python"
print("y" in mot)     # True
print("z" in mot)     # False → 'z' n'y est pas
print("Py" in mot)    # True  → sous-chaîne présente
print("thon" in mot)  # True
```

💡 Le test fonctionne **aussi bien avec des caractères individuels qu'avec des mots entiers**. 💡 Python fait la différence entre majuscules et minuscules. On dit que la comparaison est **sensible à la casse** (***case sensitive***)

Exemple avec `not in`

L'opérateur inverse :

> *Est-ce que cet élément **n'est pas** dans la chaîne ?*

```
phrase = "Bonjour à tous"

print("a" not in phrase)   # False → car il y a bien un 'a'
print("z" not in phrase)   # True  → il n'y a pas de 'z'
```

Exemple – Vérifier la présence d'un mot

```
texte = "Python est un langage puissant"

if "langage" in texte:
    print("Le mot 'langage' est présent dans la phrase.")
```

Résultat :

```
Le mot 'langage' est présent dans la phrase.
```

Exemple – Vérifier si un mot contient une majuscule

```
mot = "Informatique"

if "I" in mot:
    print("Ce mot commence bien par une majuscule.")
```

À retenir 🧠

EXPRESSION	SIGNIFICATION	RÉSULTAT
`"a" in "chat"`	le caractère `'a'` est-il présent ?	✅ `True`
`"z" in "chat"`	`'z'` est-il présent ?	❌ `False`
`"ch" in "chat"`	la sous-chaîne `'ch'` existe ?	✅ `True`
`" " in "Hello World"`	y a-t-il un espace ?	✅ `True`
`"Python" not in "JavaScript"`	Python est-il absent ?	✅ `True`

Bonnes pratiques

- Les comparaisons avec `in` sont **sensibles à la casse** (`"p"` ≠ `"P"`).
- Tu peux **normaliser** tes chaînes avec `.lower()` ou `.upper()` pour éviter les erreurs :

```
mot = "Python"
print("p" in mot.lower())  # True
```

- Combine `in` avec `if`, `for`, ou même des **conditions multiples** :

```
if "@" in email and "." in email:
    print("Adresse email valide.")
```

En résumé

BUT	OUTIL	EXEMPLE
Vérifier la présence d'un caractère	`in`	`"a" in "chat"`
Vérifier l'absence d'un caractère	`not in`	`"z" not in "chat"`
Vérifier la présence d'un mot	`in`	`"Python" in phrase`

Python: Les conditions

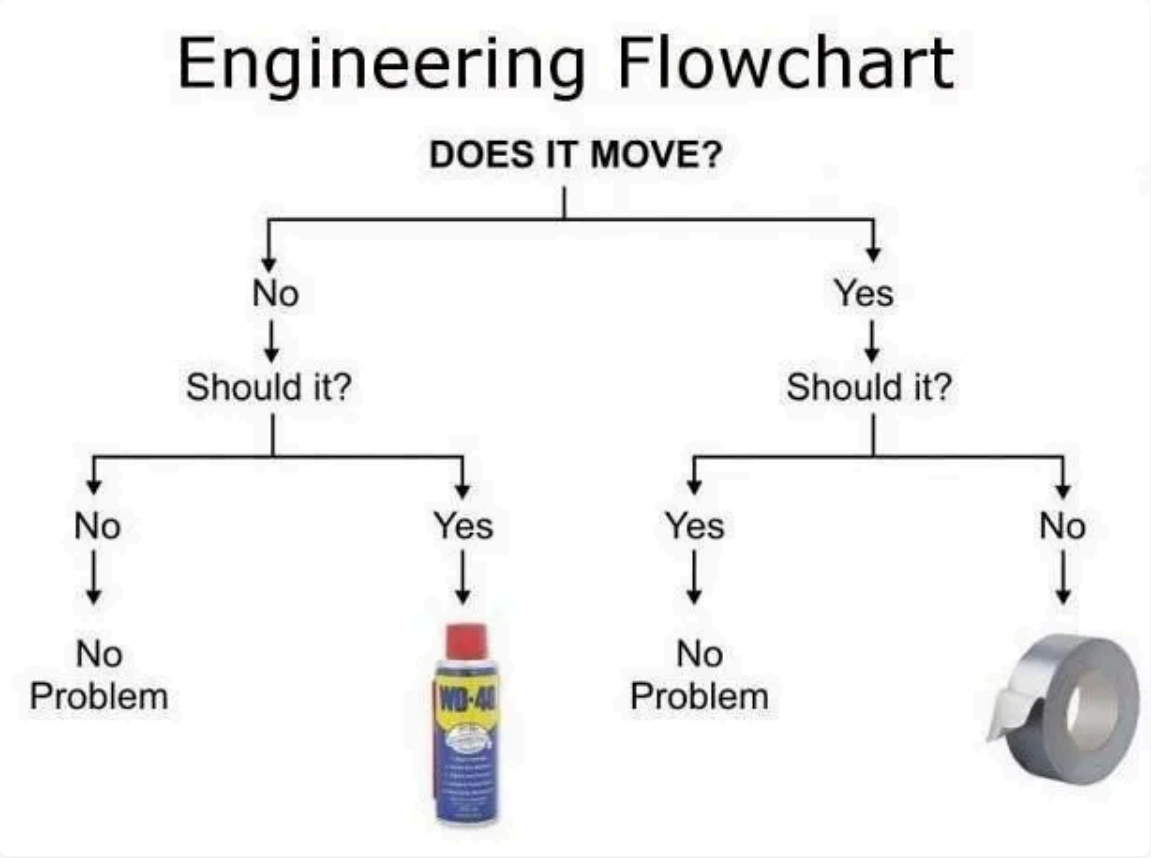
> *Lorsque nous débutons l'apprentissage de la programmation, l'un des concepts fondamentaux à maîtriser est celui des conditions. Elles sont au cœur de la prise de décision dans nos codes, permettant à nos programmes d'agir différemment selon les situations rencontrées. Dans cet article, nous explorerons 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

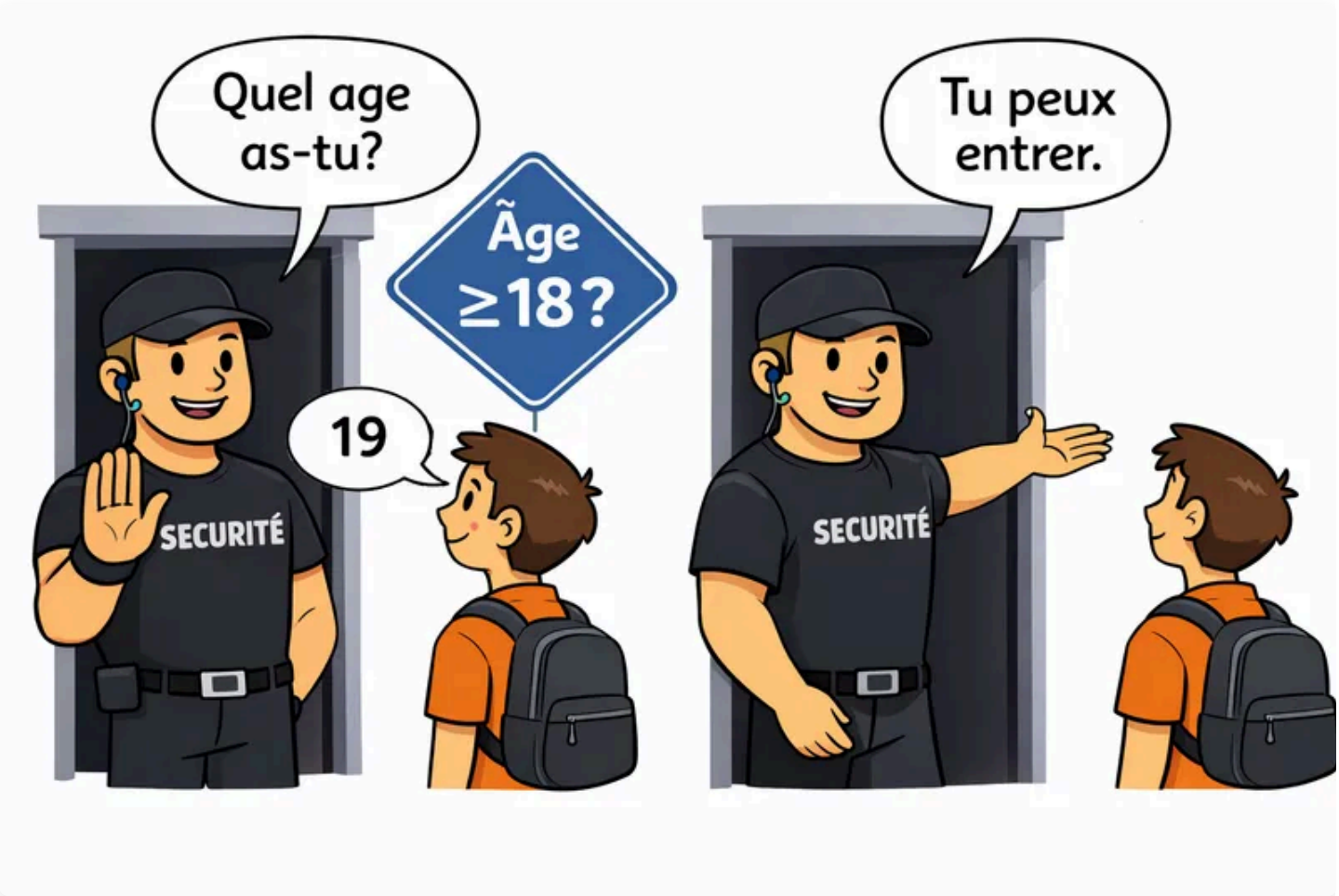
Un autre usage crucial des conditions est la **validation des entrées utilisateur**. Avant de traiter les données fournies par l'utilisateur, il est sage de vérifier leur validité. Les conditions aident à assurer que notre programme ne traite que des entrées appropriées, évitant ainsi des erreurs potentielles.

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:




```python
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.

🎯 Conditions en Python: objectifs d'apprentissage.

> *Ce document définit les objectifs progressifs liés à la compréhension et à l'utilisation des structures conditionnelles en Python.*

A. Principe des conditions en programmation

Niveau — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'identifier une condition simple dans un programme Python.	☐	☐	☐	
Je suis capable de reproduire un exemple utilisant une condition `if`.	☐	☐	☐	
Je suis capable d'exécuter un programme contenant une condition et d'observer son comportement.	☐	☐	☐	

Niveau — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer ce qu'est une condition en programmation.	☐	☐	☐	
Je sais expliquer qu'une condition produit un résultat booléen (`True` ou `False`).	☐	☐	☐	
Je sais expliquer le rôle des conditions dans la prise de décision d'un programme.	☐	☐	☐	

Niveau — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'identifier les situations où une condition est nécessaire dans un programme.	☐	☐	☐	
Je suis capable d'expliquer comment une condition influence le déroulement d'un programme.	☐	☐	☐	

B. L'instruction `if`

Niveau — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'écrire une instruction `if` simple.	☐	☐	☐	
Je suis capable d'écrire une condition comparant une variable à une valeur.	☐	☐	☐	
Je suis capable d'écrire un programme qui exécute un bloc de code uniquement si une condition est vraie.	☐	☐	☐	

Niveau — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer le rôle du mot-clé `if`.	☐	☐	☐	
Je sais expliquer pourquoi le code sous `if` ne s'exécute que si la condition est vraie.	☐	☐	☐	
Je sais expliquer la structure logique d'un test conditionnel.	☐	☐	☐	

Niveau — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser `if` pour contrôler le comportement d'un programme.	☐	☐	☐	
Je suis capable d'anticiper le résultat d'un programme contenant une condition.	☐	☐	☐	

C. L'instruction `else`

Niveau — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'écrire une structure `if / else`.	☐	☐	☐	
Je suis capable d'écrire un programme qui affiche un résultat différent selon la condition.	☐	☐	☐	
Je suis capable de reproduire un programme utilisant une alternative simple.	☐	☐	☐	

Niveau — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer le rôle de `else`.	☐	☐	☐	
Je sais expliquer pourquoi un seul des deux blocs (`if` ou `else`) est exécuté.	☐	☐	☐	
Je sais expliquer la logique « soit / soit » d'une structure conditionnelle.	☐	☐	☐	

Niveau — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser `if / else` pour résoudre un problème simple.	☐	☐	☐	
Je suis capable d'expliquer le fonctionnement d'un programme contenant une décision binaire.	☐	☐	☐	

D. Opérateurs de comparaison

Niveau — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser `==` pour tester l'égalité.	☐	☐	☐	
Je suis capable d'utiliser `≠` pour tester une différence.	☐	☐	☐	
Je suis capable d'utiliser `<` et `>` dans une condition.	☐	☐	☐	
Je suis capable d'utiliser `≤` et `≥` dans une condition.	☐	☐	☐	

Niveau — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer la différence entre `=` et `==`.	☐	☐	☐	
Je sais expliquer le rôle d'un opérateur de comparaison.	☐	☐	☐	
Je sais expliquer que le résultat d'une comparaison est un booléen.	☐	☐	☐	

Niveau — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de choisir l'opérateur de comparaison adapté à une situation.	☐	☐	☐	
Je suis capable d'anticiper le résultat d'une comparaison dans un programme.	☐	☐	☐	

E. Utiliser des conditions avec des variables

Niveau — Reproduire

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable de tester la valeur d'une variable dans une condition.	☐	☐	☐	
Je suis capable d'utiliser une variable numérique dans un test `if`.	☐	☐	☐	
Je suis capable d'utiliser une valeur saisie par l'utilisateur dans une condition.	☐	☐	☐	

Niveau — Comprendre

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je sais expliquer comment une variable influence une condition.	☐	☐	☐	
Je sais expliquer comment une condition permet de vérifier une information.	☐	☐	☐	

Niveau — Généraliser / Combiner

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'utiliser une condition pour valider une donnée.	☐	☐	☐	
Je suis capable d'utiliser une condition pour contrôler le comportement d'un programme.	☐	☐	☐	

F. Autonomie / Transfert

Niveau — Généraliser / Transférer

OBJECTIF	ÉLÈVE	PAIR	PROF	RESSOURCE
Je suis capable d'écrire un programme utilisant `if` et `else` pour prendre une décision.	☐	☐	☐	
Je suis capable d'anticiper le comportement d'un programme conditionnel.	☐	☐	☐	
Je suis capable d'expliquer le fonctionnement d'une structure conditionnelle complète.	☐	☐	☐	

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_passage.
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 limite_
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 nombre2.
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 ou égal
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: Opérations de comparaison - Exemples

> *Quelques exemples pour s'entraîner*

Opérateurs de comparaison (avec valeurs, `print`)

Ici, `print( ... )` affiche **True** ou **False**.

ÂGE

```
print(16 >= 18)
```

Zone de prédiction --> 👉 `Valeur affichée : ________`

Étapes

- Question : “16 est-il supérieur ou égal à 18 ?”
- Réponse : False → on affiche False.

BUDGET SUFFISANT

```
print(50 >= 49.99)
```

Zone de prédiction --> 👉 `Valeur affichée : ________`

Étapes

- “Ai-je assez avec 50€ pour payer 49.99€ ?”
- True → on affiche True.

VITESSE LIMITE

```
print(72 > 50)
```

Étapes

- “72 est-il au-dessus de 50 ?”
- True → on affiche True (excès de vitesse).

TEMPÉRATURE “FROID”

```
print(8 < 10)
```

Zone de prédiction --> 👉 `Valeur affichée : ________`

Étapes

- “8 est-il plus petit que 10 ?”
- True → on affiche True.

CODE CORRECT

```
print(1234 == 9999)
```

Zone de prédiction --> 👉 `Valeur affichée : _____`

Étapes

- “1234 est-il égal à 9999 ?”
- False → on affiche False.

Opérateurs de comparaison (avec variables, `print`)

MAJORITÉ

```
age = 16
print(age >= 18)
```

Zone de prédiction --> 👉 `Valeur affichée : _____`

Étapes

- `age` vaut 16.
- On teste `age ≥ 18`.
- Affiche False.

ACHAT POSSIBLE

```
argent = 35
prix = 40
print(argent >= prix)
```

Zone de prédiction --> 👉 `Valeur affichée : _____`

Étapes

- On a 35€, le prix est 40€.
- Test : “35 >= 40 ?”
- Affiche False.

MOT LONG

```
mot = "bonjour"
print(len(mot) > 5)
```

Zone de prédiction --> 👉 `Valeur affichée : _____`

Étapes

- `len(mot)` calcule le nombre de caractères.
- “bonjour” → 7 caractères.
- 7 > 5 → True.

DISTANCE TRAJET COURT

```
distance = 12
print(distance <= 10)
```

Zone de prédiction --> 👉 `Valeur affichée : ________`

Étapes

- Distance = 12 km.
- Test : “12 <= 10 ?”
- False.

NOTE RÉUSSIE

```
note = 9
print(note >= 10)
```

Zone de prédiction --> 👉 `Valeur affichée : ________`

Étapes

- Note = 9/20.
- Test : “9 >= 10 ?”
- False.

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.

```
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 :

```
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 boucles

> *Les boucles sont des structures de contrôle fondamentales en programmation, permettant de **répéter l'exécution d'un bloc de code** un certain nombre de fois ou tant qu'une condition spécifiée reste vraie. Elles sont largement utilisées dans tous les langages de programmation pour automatiser des tâches répétitives, traiter des ensembles de données et réaliser des opérations itératives sur des structures complexes.*

Les boucles sont largement utilisées dans tous les langages de programmation **pour automatiser des tâches répétitives** et **traiter des ensembles de données**.

Utilité des Boucles en Programmation

En programmation, tu as souvent besoin de répéter une action plusieurs fois. Sans boucle, tu dois copier-coller le même code plusieurs fois, ce qui rend la maintenance difficile.

Compter sans boucle

Pour afficher les nombres de 1 à 5, tu écris :

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

Chaque instruction est rédigée individuellement.

Compter avec une boucle

Utilise une boucle `for` pour automatiser la répétition :

```
for i in range(1, 6):
    print(i)
```

La boucle parcourt automatiquement les nombres de 1 à 5. Tu écris le code une seule fois, ce qui simplifie la modification et améliore la lisibilité.

Les avantages des boucles

1. **Gain de temps** : Moins de code à écrire.
2. **Facilité de modification** : Modifier la plage de répétition est simple.
3. **Lisibilité améliorée** : Le code est plus structuré et compréhensible.
4. **Réduction des erreurs** : Moins de répétitions manuelles réduit le risque d'erreurs.

Utilité des boucles

Répétition de tâches

Les boucles permettent d'exécuter un bloc de code plusieurs fois sans avoir à le répéter manuellement. Cela permet d'économiser du temps et des efforts en automatisant des tâches répétitives.

Traitement de données

Les boucles sont couramment utilisées pour parcourir des ensembles de données, tels que des listes, des tableaux ou des fichiers, et effectuer des opérations sur chaque élément de manière itérative. Cela permet de traiter efficacement de grandes quantités de données.

Les Types de Boucles en Programmation

Il existe principalement deux types de boucles en programmation :

1. Boucles `for`

Les boucles `for` sont utilisées pour parcourir une séquence de données, telle qu'une liste, un tuple ou un dictionnaire, et exécuter un bloc de code pour chaque élément de la séquence. Elles sont couramment utilisées lorsque le nombre d'itérations est connu à l'avance ou lorsqu'il est nécessaire de parcourir une séquence de données.

2. Boucles `while`

Les boucles `while` sont utilisées pour exécuter un bloc de code tant qu'une condition spécifiée reste vraie. Elles sont couramment utilisées lorsque le nombre d'itérations n'est pas connu à l'avance ou lorsqu'une condition spécifique doit être vérifiée avant chaque itération.

Utilisation des Boucles en Python

En Python, les boucles sont largement utilisées pour diverses tâches de programmation en raison de leur simplicité et de leur flexibilité. Python offre les boucles `for` et `while`, ainsi que des fonctions intégrées telles que `range()` pour faciliter la création de boucles complexes.

À retenir

- Une boucle permet de répéter des instructions de manière automatique.
- La syntaxe de base en Python pour une boucle `for` est `for variable in range(début, fin)`.
- Utilise les boucles pour rendre ton code plus efficace et facile à maintenir.

En conclusion, les boucles sont des outils essentiels en programmation, offrant un moyen efficace de **répéter des tâches**, de traiter des données et de contrôler le flux d'exécution d'un programme.

Python: Les boucles for

> *Quand un programme commence à devoir répéter une action plusieurs fois, copier-coller la même ligne devient vite absurde. La boucle `for` est la réponse propre à ce problème : elle décrit une fois ce qu'il faut faire, et Python s'occupe de le refaire autant de fois qu'il faut.*

*Tu vas voir ici **pourquoi** on a inventé les boucles `for`, **comment** elles se construisent, et tu termineras en écrivant ta première table de multiplication automatique.*

🎯 Objectifs

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

1. Reconnaître les situations où une boucle `for` remplace utilement du copier-coller.
2. Écrire une boucle `for` qui exécute une action **un nombre précis de fois**.
3. Utiliser `range()` avec un, deux ou trois paramètres.
4. **Parcourir** une chaîne de caractères, lettre par lettre.
5. Résoudre seul un petit problème classique (table de multiplication) avec une boucle.

Partie 1 — Le problème : copier-coller, ça ne tient pas

Imagine qu'on te demande d'afficher 5 fois **le même message** — par exemple « Joyeux anniversaire, Léa ! » pour faire une carte un peu folle. Sans boucle, ça donne :

```
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
```

Cinq lignes identiques. Ça marche, mais maintenant :

- **Et s'il en fallait 200 fois ?** Tu vas vraiment écrire 200 `print` ?
- **Et si on veut changer le texte** (mettre « Bon anniversaire » à la place) ? Il faudrait modifier les 200 lignes.
- **Et si on ne sait qu'au moment de l'exécution combien de fois répéter** (l'utilisateur entre le nombre) ? Tu ne peux même pas écrire les `print` à l'avance.

> 🧠 ***La répétition manuelle ne passe pas à l'échelle.*** *Dès qu'on commence à copier-coller plus de deux ou trois lignes presque identiques, c'est le signe qu'il faut sortir une boucle.*

La boucle `for` répond exactement à ça : tu décris **une seule fois** ce qu'il faut faire, et Python s'occupe de le refaire autant de fois que tu veux.

> 💡 ***Et si chaque ligne était différente ?*** *Par exemple, afficher 5 prénoms d'élèves :* `Alice`, `Bob`, `Charlie` *… Ça se résout aussi avec une boucle, mais on a besoin d'une* ***liste*** *pour stocker les prénoms. Les listes arrivent dans un cours suivant — pour l'instant, on se concentre sur le cas « même action, répétée N fois ».*

Partie 2 — Première boucle : répéter N fois

Pour démarrer simple, on combine `for` avec la fonction `range()` pour répéter l'action **un nombre précis de fois** :

```
for i in range(5):
    print("Joyeux anniversaire, Léa !")
```

Résultat à l'écran :

```
Joyeux anniversaire, Léa !
Joyeux anniversaire, Léa !
Joyeux anniversaire, Léa !
Joyeux anniversaire, Léa !
Joyeux anniversaire, Léa !
```

Cinq fois. C'est exactement le programme du début, mais en **deux lignes** au lieu de cinq. Et si tu remplaces `range(5)` par `range(200)`, tu obtiens 200 lignes — sans rien changer d'autre.

▸ Décortiquons cette boucle

```
for      i      in     range(5)     :          ← en-tête (« la promesse »)
└──┬──┘ └─┬─┘ └──┬──┘ └─────┬─────┘
 mot-clé  variable          séquence à parcourir

    print("Bonjour")                          ← corps (indenté de 4 espaces)
```

ÉLÉMENT	RÔLE
`for`	Mot-clé qui démarre la boucle
`i`	Variable qui prend une nouvelle valeur à chaque tour
`in`	Mot-clé qui sépare la variable de la séquence
`range(5)`	Génère la séquence `0, 1, 2, 3, 4`
`:`	Termine l'en-tête (obligatoire en Python)
Corps indenté	Le code à répéter à chaque tour

> ⚠️ ***L'indentation est obligatoire.*** *Le corps de la boucle DOIT être indenté (4 espaces, ou une tabulation, mais reste cohérent). Sans indentation, Python ne sait pas où finit la boucle.*

Partie 3 — Anatomie d'une exécution

Reprends ce code, mais cette fois on **utilise** la variable `i` :

```
for i in range(5):
    print("Tour numéro", i)
```

Résultat :

```
Tour numéro 0
Tour numéro 1
Tour numéro 2
Tour numéro 3
Tour numéro 4
```

À chaque tour, `i` prend une valeur différente :

TOUR	VALEUR DE `I`	LIGNE AFFICHÉE
1	0	`Tour numéro 0`
2	1	`Tour numéro 1`
3	2	`Tour numéro 2`
4	3	`Tour numéro 3`
5	4	`Tour numéro 4`

> 💡 ***Pourquoi ça commence à 0 ?*** *Parce que* `range(5)` *produit* `0, 1, 2, 3, 4` *— cinq valeurs en partant de 0. C'est une convention en programmation : on compte à partir de 0. On verra dans la partie 5 comment forcer un autre point de départ.*

Partie 4 — Parcourir une chaîne de caractères

Il existe une autre situation où une boucle `for` est très utile : quand on veut **traiter chaque élément** d'une suite, sans forcément savoir combien il y en a.

Une chaîne de caractères, c'est exactement ça : une suite de lettres.

```
mot = "Python"

for lettre in mot:
    print(lettre)
```

Résultat :

```
P
y
t
h
o
n
```

Ici, on n'a pas écrit `range(6)`. On a écrit `for lettre in mot`. Python sait **tout seul** parcourir la chaîne lettre par lettre, peu importe sa longueur. Si le mot fait 6 lettres, la boucle s'exécute 6 fois. S'il en fait 50, elle s'exécute 50 fois.

> 🧠 ***À retenir*** *:* `for` *ne sert pas seulement à compter. Sa vraie nature est de dire « pour chaque élément de cette suite, fais ceci ». Une chaîne, une liste, un fichier ouvert… tout ça se parcourt avec* `for` *.*

▸ Exemple — Lister les lettres d'un mot

Tu pourrais essayer ça :

```python
mot = "ornithorynque"

for lettre in mot:
    print("Une lettre :", lettre)
```

Sans rien savoir d'autre, ton programme va afficher 13 lignes. Tu n'as pas eu à écrire « 13 » dans ton code — c'est Python qui s'adapte.

Partie 5 — La fonction `range()` en détail

Tu as vu `range(5)`. Mais `range()` peut prendre **un, deux ou trois** paramètres. Chaque forme est utile dans des contextes différents.

▸ Forme 1 — `range(fin)` : de 0 à fin − 1

```python
for i in range(5):
    print(i)
```

→ `0, 1, 2, 3, 4`

▸ Forme 2 — `range(début, fin)` : de début à fin − 1

```python
for i in range(1, 6):
    print(i)
```

→ `1, 2, 3, 4, 5`

> 💡 ***Très pratique*** *pour compter de 1 à 10 « comme un humain », sans commencer à 0.*

▸ Forme 3 — `range(début, fin, pas)` : avec un incrément personnalisé

```
for i in range(0, 20, 2):
    print(i)
```

→ `0, 2, 4, 6, 8, 10, 12, 14, 16, 18`

Le pas peut être **négatif** pour aller à reculons :

```
for i in range(10, 0, -1):
    print(i)
```

→ `10, 9, 8, 7, 6, 5, 4, 3, 2, 1` (parfait pour un compte à rebours)

▸ Récapitulatif

FORME	GÉNÈRE	EXEMPLE TYPIQUE
`range(n)`	`0, 1, …, n - 1`	Répéter n fois
`range(a, b)`	`a, a+1, …, b - 1`	Compter de 1 à 10
`range(a, b, pas)`	`a, a+pas, …` (jusqu'à < `b`)	Pairs, impairs, à rebours

> ⚠️ ***La valeur de fin n'est jamais incluse.*** `range(1, 11)` *va de 1 à 10, pas jusqu'à 11. Ça surprend au début, mais c'est ce qui permet d'écrire* `range(len(mot))` *sans erreur.*

🧪 Exercices

▸ Exercice guidé — La table de 7

Énoncé. Écris un programme qui affiche la **table de multiplication par 7**, de 7 × 1 jusqu'à 7 × 10, dans ce format :

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70
```

Avant de coder, prends 30 secondes pour réfléchir. **Ne lis pas la suite tout de suite** — essaie de réfléchir à ces trois questions :

1. Combien de lignes vais-je afficher au total ?
2. Qu'est-ce qui **change** d'une ligne à l'autre ?
3. Qu'est-ce qui **reste pareil** ?

ÉTAPE 1 — IDENTIFIER LA SÉQUENCE

Dix lignes à afficher : pour `i` qui va de **1 à 10**. C'est exactement ce que produit `range(1, 11)` (rappel : la valeur de fin n'est pas incluse, donc 11 pour aller jusqu'à 10).

Premier brouillon, juste pour vérifier que la boucle tourne bien :

```
for i in range(1, 11):
    print(i)
```

Exécute. Tu dois voir afficher les nombres de 1 à 10. Si oui, **la boucle est bonne** — il ne reste plus qu'à changer ce qu'on affiche.

ÉTAPE 2 — CALCULER LE PRODUIT

À chaque tour, on veut afficher `7 × i`. En Python, le `×` s'écrit `*` :

```
for i in range(1, 11):
    print(7 * i)
```

Tu obtiens :

```
7
14
21
28
35
42
49
56
63
70
```

Les bons résultats sont là. Mais le format n'est pas celui demandé.

ÉTAPE 3 — COMPOSER LA LIGNE COMPLÈTE

On veut afficher `7 x 1 = 7`, `7 x 2 = 14`, etc. La fonction `print` accepte plusieurs arguments séparés par des virgules : elle les affiche tous séparés par un espace.

```
for i in range(1, 11):
    print(7, "x", i, "=", 7 * i)
```

Résultat :

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

C'est exactement ce qui était demandé.

ÉTAPE 4 (BONUS) — PLUS PROPRE AVEC UNE F-STRING

Quand la ligne à afficher devient longue, les `print(a, "x", b, ...)` deviennent illisibles. Python a une syntaxe plus élégante : les **f-strings** (chaînes formatées).

```python
for i in range(1, 11):
    print(f"7 x {i} = {7 * i}")
```

Le préfixe `f` devant les guillemets active la magie : tout ce qui est entre `{ }` est évalué et inséré dans la chaîne. C'est exactement la même sortie qu'avant, mais beaucoup plus lisible.

> 🧠 ***Habitue-toi aux f-strings dès maintenant.*** *C'est la façon moderne d'afficher du texte mêlé à des variables en Python. Tu vas en croiser dans presque tous les exercices à partir d'ici.*

POUR ALLER PLUS LOIN

Quelques variantes à tester par toi-même :

- **Table choisie** : remplace `7` par une variable `n = 7` en début de programme. Tu pourras changer la table en modifiant une seule ligne.
- **Table interactive** : remplace `n = 7` par `n = int(input("Quelle table ? "))`. Le programme demande à l'utilisateur quelle table afficher.
- **Toutes les tables de 1 à 10** : imbrique deux boucles `for` (mais ça, c'est pour un cours suivant).

▸ Exercice 2 — Prédiction

Avant de lancer chacun de ces programmes, **prévois sur papier** ce qu'il va afficher. Puis vérifie.

```python
for i in range(5):
    print(i)
```

```python
for i in range(1, 5):
    print(i)
```

```python
for i in range(1, 11, 2):
    print(i)
```

```python
for i in range(10, 0, -1):
    print(i)
print("BOUM !")
```

▸ Exercice 3 — Initiales en chaîne

Écris un programme qui demande ton prénom à l'utilisateur, puis affiche chaque lettre en majuscule, **une par ligne**. Indice : la méthode `.upper()` sur une lettre la met en majuscule.

```
Ton prénom ? Ludovic
L
U
D
O
V
I
C
```

✍️ À retenir

- Une boucle `for` répète un bloc de code, **soit un nombre précis de fois**, **soit pour chaque élément d'une séquence**.
- La syntaxe générale est :

  ```python
  for variable in sequence:
      instructions
  ```

- `range(n)` produit `0, 1, …, n - 1`. `range(a, b)` part de `a`. `range(a, b, pas)` ajoute un incrément (positif ou négatif).
- La **valeur de fin n'est jamais incluse** : `range(1, 11)` s'arrête à 10.
- Une chaîne de caractères est **itérable** : `for lettre in mot` parcourt les lettres une par une.
- L'indentation du corps de la boucle est **obligatoire**.
- Les **f-strings** (`f"… {variable} …"`) sont la façon moderne d'insérer des variables dans du texte.

Suite

Tu sais maintenant *répéter*. La prochaine étape : faire **évoluer une variable** d'un tour à l'autre. Ça permet de **compter** (combien de voyelles dans un mot ?) et **cumuler** (quelle est la somme des notes ?). C'est ce qu'on découvre dans le prochain cours.

Python: Les boucles for (exercices 💪)

> *Voici des **exercices progressifs**, directement alignés avec l'article sur les boucles `for`. Ils vont du **très guidé** vers le **raisonnement autonome**, sans introduire de notions non vues.*

Ces exercices te permettent de t'entraîner à utiliser les boucles `for` pour répéter des actions et parcourir des séquences (`range`, chaînes de caractères).

▸ 01 – Dire bonjour plusieurs fois

`01_bonjour_5_fois.py`

Afficher le mot `"Bonjour"` **5 fois** à l'aide d'une boucle `for`.

▸ 02 – Compter de 0 à 4

`02_compter_0_a_4.py`

Afficher les nombres de `0` à `4`, chacun sur une ligne.

▸ 03 – Compter de 1 à 10

`03_compter_1_a_10.py`

Afficher les nombres de `1` à `10` inclus, chacun sur une ligne.

▸ 04 – Afficher les nombres pairs

`04_nombres_pairs_0_20.py`

Afficher tous les nombres pairs entre `0` et `20` (inclus).

▸ 05 – Table de multiplication de 5

`05_table_5.py`

Afficher la table de multiplication de `5` (de `5 × 1` à `5 × 10`).

▸ 06 – Table de multiplication (nombre de l'utilisateur)

`06_table_nombre_utilisateur.py`

Demander un nombre à l'utilisateur et afficher sa table de multiplication (de `0` à `10`).

▸ 07 – Carrés des nombres

`07_carres_1_a_10.py`

Afficher le carré des nombres de `1` à `10`.

Exemple attendu :

```
1 → 1
2 → 4
3 → 9
```

▸ 08 – Parcourir un mot

`08_parcourir_python.py`

À partir du mot `"Python"`, afficher chaque lettre sur une ligne différente.

▸ 09 – Parcourir un mot donné par l'utilisateur

`09_parcourir_mot_utilisateur.py`

Demander un mot à l'utilisateur et afficher chaque lettre sur une nouvelle ligne.

▸ 10 – Compter les lettres d'un mot

`10_compter_lettres_mot.py`

Demander un mot à l'utilisateur et afficher le **nombre total de lettres**, en comptant avec une boucle `for` (**sans utiliser** `len()`).

▸ 11 – Afficher une phrase caractère par caractère

`11_phrase_caractere_par_caractere.py`

À partir de la phrase `"Je code en Python"`, afficher chaque caractère (y compris les espaces), un par ligne.

▸ 12 – Compter les lettres "a"

`12_compter_a.py`

Demander un mot à l'utilisateur et compter combien de fois la lettre `"a"` apparaît.

▸ 13 – Somme des nombres

`13_somme_1_a_100.py`

Calculer la somme des nombres de `1` à `100` à l'aide d'une boucle `for`.

▸ 14 – Afficher les positions d'un mot

`14_positions_programmation.py`

Afficher les nombres de `0` jusqu'à la longueur du mot `"Programmation"` moins 1.

15 – Mot vertical numéroté

`15_mot_vertical_numerote.py`

Demander un mot à l'utilisateur et afficher chaque lettre précédée de son numéro de position (à partir de 1).

Exemple :

```
1 : P
2 : y
3 : t
4 : h
5 : o
6 : n
```

---

16 – Exercice bonus : Le jeu du 21 (version `for`)

`16_bonus_jeu_21_for.py`

Le joueur commence avec un score de `0`. Le jeu se déroule en **5 tours maximum** à l'aide d'une boucle `for`.

À chaque tour :

- un nombre aléatoire entre **1 et 6** est généré ;
- ce nombre est ajouté au score ;
- le tirage et le score actuel sont affichés.

À la fin du jeu :

- si le score est **exactement 21** → afficher `"Bravo, 21 !"`
- si le score est **inférieur à 21** → afficher `"Pas assez"`
- si le score est **supérieur à 21** → afficher `"Trop élevé"`

Contraintes :

- utiliser une boucle `for`
- utiliser une variable de cumul `score`
- pas de boucle `while`
- pas de `break`

Python: Compter et cumuler des valeurs dans une boucle

> *Lorsqu'on utilise une boucle, on ne fait pas que répéter des instructions. Très souvent, on veut aussi **compter des éléments** ou **additionner des valeurs** au fil des répétitions. Cet article explique comment fonctionnent ces mécanismes, comment les coder correctement, et surtout comment **raisonner** pour éviter les erreurs.*

Pourquoi compter ou cumuler dans une boucle ?

Une boucle `for` exécute un bloc de code **plusieurs fois**. Il est donc naturel de vouloir :

- **compter** combien de fois quelque chose se produit (ex. : nombre de lettres, nombre de valeurs, nombre d'essais) ;
- **cumuler** des valeurs (ex. : somme de nombres, total de points, score final).

👉 Ces deux idées reposent sur **le même principe fondamental**.

Le principe clé : une variable qui évolue

Pour compter ou additionner, on utilise toujours une **variable qui change de valeur** à chaque itération.

Cette variable doit :

1. être **initialisée avant la boucle** ;
2. être **modifiée à chaque tour de boucle**.

Compter : augmenter de 1 à chaque fois

Exemple simple : compter jusqu'à 5

```
count = 0

for i in range(5):
    count = count + 1

print(count)
```

Ce qui se passe réellement

- `count` commence à `0`
- À chaque passage dans la boucle, on ajoute `1`
- Après 5 passages, `count` vaut `5`

👉 La ligne importante est :

```
count = count + 1
```

On peut la lire comme :

> *« le nouveau `count` devient l'ancien `count` plus 1 »*

Cumuler : additionner une valeur qui change

▸ Exemple : somme des nombres de 1 à 5

```
total = 0

for i in range(1, 6):
    total = total + i

print(total)
```

▸ Interprétation

- `total` commence à `0`
- `i` prend successivement les valeurs `1`, `2`, `3`, `4`, `5`
- À chaque tour, on ajoute `i` au total

👉 Ici, la ligne clé est :

```
total = total + i
```

Cela signifie :

> *« j'ajoute la valeur actuelle de `i` au total existant »*

Différence entre compter et cumuler

OBJECTIF	VARIABLE MODIFIÉE	CE QU'ON AJOUTE
Compter	`count`	toujours `+1`
Cumuler	`total`	une valeur (`i` , un score, un prix, etc.)

Erreur fréquente à éviter ❌

Réinitialiser la variable dans la boucle

```
for i in range(5):
    count = 0      # ❌ erreur
    count = count + 1
```

➡️ Ici, `count` est remis à `0` à chaque tour. Résultat final : `count` vaut toujours `1`.

👉 **L'initialisation doit toujours être faite avant la boucle.**

Astuce essentielle : raisonner sur papier 🖍️

Quand un programme devient difficile à suivre, **il faut le simuler**.

Méthode recommandée

1. Trace une **ligne par variable**
2. Fais évoluer les valeurs **itération par itération**

Exemple de simulation sur papier

Code :

```
total = 0

for i in range(1, 4):
    total = total + i
```

Tableau de suivi :

TOUR	I	TOTAL AVANT	TOTAL APRÈS
1	1	0	1
2	2	1	3
3	3	3	6

➡️ Résultat final : `6`

👉 Cette méthode permet de :

- comprendre ce que fait réellement le programme ;
- repérer immédiatement les erreurs de raisonnement ;
- expliquer son code à quelqu'un d'autre.

Cas très courant : compter sous condition

Exemple : compter le nombre de lettres `"a"` dans un mot.

```
mot = "banane"
count = 0

for lettre in mot:
    if lettre == "a":
        count = count + 1

print(count)
```

Ici :

- la boucle parcourt chaque lettre ;
- le compteur n'augmente **que si la condition est vraie**.

À retenir

- Pour **compter**, on ajoute toujours `1` à une variable compteur.
- Pour **cumuler**, on ajoute une valeur qui change à une variable total.
- La variable de comptage ou de cumul :
  - est initialisée **avant** la boucle ;
  - est modifiée **dans** la boucle.
- En cas de doute, **simuler sur papier** avec un tableau est la meilleure stratégie.
- Comprendre l'évolution des variables est plus important que mémoriser le code.

Si tu veux, je peux ensuite :

- proposer **une série d'exercices ciblés "compter / cumuler"**,
- écrire un **article jumeau** sur les erreurs fréquentes,
- ou relier cette notion à des **cas concrets de jeu ou de statistiques**.

Python: Les boucles for sur des listes

> *Une boucle `for` ne sert pas seulement à répéter N fois la même chose. Sa vraie force, c'est de **parcourir une collection** d'éléments — une liste de prénoms, de prix, de notes — et de traiter chacun à tour de rôle. C'est ce qu'on apprend ici.*

*Dans le cours précédent, on a vu `for i in range(N)` pour répéter N fois la même action. Mais quand chaque tour de boucle traite un élément **différent** — un prénom, un prix, un nom d'animal — il faut une **liste**. C'est l'outil de stockage le plus basique de Python, et il marche main dans la main avec `for`.*

🎯 Objectifs

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

1. Créer une **liste** Python avec les crochets `[ ]`.
2. **Parcourir** une liste avec une boucle `for`.
3. Appliquer la **convention de nommage** : nom de la liste au pluriel, variable d'itération au singulier.
4. Faire varier le traitement dans la boucle (afficher, modifier, conditionner).
5. Reconnaître qu'une liste peut contenir n'importe quel type de valeurs.

Partie 1 — Le problème : un truc différent à chaque tour

Reprends l'idée du cours précédent. On voulait afficher des prénoms d'élèves :

```
print("Alice")
print("Bob")
print("Charlie")
print("Diane")
print("Eli")
```

Avec ce qu'on a vu jusqu'ici (`for + range`), on **ne sait pas faire** : `for i in range(5)` répèterait 5 fois la **même** ligne. Mais nous, on veut **5 lignes différentes**.

Il manque un endroit où **stocker les cinq prénoms à l'avance**. C'est exactement le rôle d'une **liste**.

Partie 2 — Créer une liste

Une liste se crée avec des **crochets** `[ ]` et des éléments séparés par des virgules :

```
prenoms = ["Alice", "Bob", "Charlie", "Diane", "Eli"]
```

Quelques détails à remarquer :

- Le nom de la variable est **au pluriel** : `prenoms`. C'est une convention universelle en Python (et dans la plupart des langages), parce que la variable contient **plusieurs valeurs**.
- Chaque élément est entre **guillemets** : on stocke des chaînes de caractères.
- L'ordre est conservé : `prenoms[0]` vaut `"Alice"`, `prenoms[1]` vaut `"Bob"`, etc.

> 💡 ***Vocabulaire*** *: un élément d'une liste s'appelle aussi un* ***item*** *(GB* ***item****). On dit « parcourir les items » ou « traiter chaque item ».*

Partie 3 — Parcourir une liste avec `for`

C'est ici que tout se met en place. Pour traiter chaque prénom de la liste à tour de rôle :

```
prenoms = ["Alice", "Bob", "Charlie", "Diane", "Eli"]

for prenom in prenoms:
    print(prenom)
```

Résultat :

```
Alice
Bob
Charlie
Diane
Eli
```

Exactement ce qu'on voulait — **et la boucle s'adapte automatiquement** au nombre d'éléments. Si tu ajoutes 195 prénoms à la liste, la boucle s'exécute 200 fois sans rien changer d'autre.

▸ La convention de nommage : pluriel / singulier

Regarde bien les deux noms de variables dans la boucle :

```
for prenom in prenoms:
#   └────┘     └─────┘
#  singulier   pluriel
#  (un item)   (la collection)
```

C'est **la** convention Python pour les boucles `for` sur listes. Elle se lit en français naturel :

> *« Pour chaque* ***prénom*** *dans la liste des* ***prénoms****, fais… »*

Autres exemples :

LISTE (PLURIEL)	VARIABLE DE BOUCLE (SINGULIER)	LECTURE
`animaux`	`animal`	« Pour chaque animal dans animaux »
`prix`	`prix_unitaire` (ou `p`)	« Pour chaque prix dans prix »
`notes`	`note`	« Pour chaque note dans notes »
`eleves`	`eleve`	« Pour chaque élève dans élèves »
`voitures`	`voiture`	« Pour chaque voiture dans voitures »

> 🧠 ***Pourquoi cette convention ?*** *Un lecteur qui découvre ton code doit pouvoir comprendre en deux secondes la nature de chaque variable.* `for animal in animaux` *est instantanément clair.* `for x in liste1` *est obscur. Adopte le pluriel/singulier dès le début, c'est gratuit et ça change la lisibilité.*

Partie 4 — Que faire dans la boucle ?

À l'intérieur de la boucle, tu peux faire **tout ce que tu veux** avec l'élément courant. Voici les usages les plus fréquents.

4.1 — Afficher chaque élément

C'est l'exemple qu'on vient de voir :

```
animaux = ["Chat", "Lapin", "Capibara", "Dragon de Komodo"]

for animal in animaux:
    print(animal)
```

4.2 — Afficher avec une mise en forme

```
animaux = ["Chat", "Lapin", "Capibara"]

# On utilise une f-string pour entourer le nom de texte
for animal in animaux:
    print(f"🐾 {animal} a été ajouté au zoo.")
```

Résultat :

```
🐾 Chat a été ajouté au zoo.
🐾 Lapin a été ajouté au zoo.
🐾 Capibara a été ajouté au zoo.
```

4.3 — Transformer chaque élément

```
prenoms = ["alice", "bob", "charlie"]

# La méthode .upper() met une chaîne en majuscules
for prenom in prenoms:
    print(prenom.upper())
```

Résultat :

```
ALICE
BOB
CHARLIE
```

4.4 — Conditionner le traitement

Tu peux combiner `for` avec un `if` pour ne traiter qu'une partie des éléments :

```
notes = [12, 8, 17, 4, 15, 19, 11]

# On affiche uniquement les notes en échec (< 10)
for note in notes:
    if note < 10:
        print(f"⚠️ Note insuffisante : {note}/20")
```

Résultat :

```
⚠️ Note insuffisante : 8/20
⚠️ Note insuffisante : 4/20
```

> 💡 ***Tu remarques*** *? La boucle parcourt* ***toutes*** *les notes, mais l'* `if` *filtre celles qu'on veut afficher. C'est la combinaison la plus utilisée en pratique.*

Partie 5 — Une liste peut contenir n'importe quoi

Les listes ne sont pas limitées aux chaînes. Tu peux y mettre des nombres, des booléens, des choses mélangées.

5.1 — Liste de nombres

```
prix = [2.50, 4.99, 1.20, 8.75, 3.00]

# Affichage formaté
for p in prix:
    print(f"{p:.2f} €")
```

Résultat :

```
2.50 €
4.99 €
1.20 €
8.75 €
3.00 €
```

> ℹ️ **Le `:.2f` dans la f-string** *force 2 chiffres après la virgule. Très pratique pour les montants.*

5.2 — Liste de booléens

```python
reponses = [True, False, True, True, False]

for reponse in reponses:
    if reponse:
        print("✓ Correct")
    else:
        print("✗ Faux")
```

5.3 — Liste hétérogène (à éviter en général)

Python autorise des listes avec des types mélangés :

```python
truc = ["Alice", 17, True, 3.14]

for element in truc:
    print(element)
```

Ça fonctionne, mais **évite-le** : une liste qui mélange tout, c'est un signe qu'on aurait dû utiliser une autre structure (un dictionnaire, par exemple — on verra plus tard).

> 🧠 ***Bonne pratique** : une liste = **une collection d'éléments du même type**, qui représentent la même chose.*

🧪 Exercices

Exercice guidé — Le panier de courses

Énoncé. Tu as la liste des courses à faire :

```python
courses = ["pain", "lait", "œufs", "fromage", "tomates"]
```

Écris un programme qui affiche :

```
À acheter :
- pain
- lait
- œufs
- fromage
- tomates

Total : 5 articles
```

ÉTAPE 1 — IDENTIFIER CE QU'IL FAUT

Trois choses à faire :

1. Afficher une ligne d'en-tête (`À acheter :`).
2. Parcourir la liste et afficher chaque article précédé d'un `-` .
3. Afficher une ligne de total.

Le **point 2** est le cœur de la boucle.

ÉTAPE 2 — LA BOUCLE QUI AFFICHE LES ARTICLES

```
courses = ["pain", "lait", "œufs", "fromage", "tomates"]

for course in courses:
    print(f"- {course}")
```

Note le **pluriel/singulier** : `for course in courses` .

Résultat à ce stade :

```
- pain
- lait
- œufs
- fromage
- tomates
```

ÉTAPE 3 — AJOUTER L'EN-TÊTE ET LE TOTAL

L'en-tête se met **avant** la boucle, le total **après**. La fonction `len()` donne la taille d'une liste.

```
courses = ["pain", "lait", "œufs", "fromage", "tomates"]

print("À acheter :")

for course in courses:
    print(f"- {course}")

print(f"\nTotal : {len(courses)} articles")
```

Le `\n` au début de la dernière f-string ajoute une ligne vide avant le total.

Résultat final :

```
À acheter :
- pain
- lait
- œufs
- fromage
- tomates

Total : 5 articles
```

🧠 ***Le squelette à retenir*** *: en-tête → boucle → bilan. C'est un pattern qu'on retrouve partout (factures, bulletins, statistiques…).*

▸ Exercice 2 — Le zoo

Tu as la liste suivante :

```
animaux = ["Chat", "Lapin", "Capibara", "Lion", "Dragon de Komodo"]
```

Écris un programme qui affiche pour chaque animal :

```
🐾 Chat fait partie du zoo. (4 lettres)
🐾 Lapin fait partie du zoo. (5 lettres)
🐾 Capibara fait partie du zoo. (8 lettres)
🐾 Lion fait partie du zoo. (4 lettres)
🐾 Dragon de Komodo fait partie du zoo. (16 lettres)
```

Indice : `len(animal)` te donne le nombre de caractères du nom (espaces compris).

▸ Exercice 3 — Notes au-dessus de la moyenne

Pour la liste suivante :

```
notes = [11, 7, 15, 18, 4, 13, 9, 16]
```

Affiche **uniquement** les notes ≥ 10, dans ce format :

```
✓ 11/20
✓ 15/20
✓ 18/20
✓ 13/20
✓ 16/20
```

Indice : combine `for` et un `if`.

▸ Exercice 4 — Inventaire avec prix

À partir des listes suivantes (qu'on suppose alignées dans le même ordre — produit i ↔ prix i) :

```
produits = ["Pain", "Lait", "Œufs", "Fromage"]
prix     = [1.50, 1.20, 2.80, 4.99]
```

Écris un programme qui affiche :

```
- Pain     : 1.50 €
- Lait     : 1.20 €
- Œufs     : 2.80 €
- Fromage  : 4.99 €
```

Indice : ici on a deux listes à parcourir en parallèle. Utilise `range(len(produits))` et accède aux deux listes par leur index — ou découvre seul la fonction `zip()` si tu es curieux.

✍️ À retenir

- Une **liste** se crée avec `[ ]` : `prenoms = ["Alice", "Bob"]`.
- Une boucle `for` parcourt **tous les éléments**, un par un, dans l'ordre.
- **Convention de nommage** : liste au **pluriel** (`animaux`), variable d'itération au **singulier** (`animal`).
- À l'intérieur de la boucle, tu peux **afficher**, **transformer**, **conditionner** — tout ce que tu veux faire avec l'élément courant.
- La boucle s'adapte **automatiquement** à la taille de la liste : pas besoin d'écrire combien d'éléments.
- Une liste peut contenir n'importe quel type, mais **par bonne pratique** elle contient des éléments de même nature.
- `len(liste)` te donne le nombre d'éléments.

Suite

Tu sais maintenant parcourir une liste. Une application directe très fréquente : **afficher un menu de sélection à l'utilisateur** — par exemple les choix d'un jeu, les rayons d'un magasin, les options d'une appli. C'est l'objet du cours suivant, qui montre la progression *menu manuel → menu avec une liste → menu numéroté avec* `enumerate()`.

Python: Afficher un menu de sélection

> *Un menu dans une console, c'est l'application la plus fréquente du combo *liste + boucle for*. On part d'un menu écrit à la main, on le remplace par un menu généré depuis une liste, puis on lui ajoute des numéros automatiques avec `enumerate()`. Et à la fin, l'utilisateur peut vraiment choisir.*

*Presque toutes les applications console montrent un menu à un moment ou l'autre : un jeu (« Nouvelle partie / Charger / Quitter »), une appli de quiz (« Choisis ta catégorie »), un script utilitaire (« Que veux-tu faire ? »). Ce cours montre la **bonne façon** de l'écrire — et surtout, comment passer du « ça marche pour 3 options » au « ça marche pour 30 ».*

🎯 Objectifs

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

1. Écrire un menu console avec un titre, des options et un choix utilisateur.
2. Reconnaître pourquoi le menu « écrit à la main » devient ingérable au-delà de 3-4 options.
3. Générer un menu **depuis une liste**, avec une boucle `for`.
4. Ajouter une **numérotation automatique** grâce à `enumerate()`.
5. Récupérer le choix de l'utilisateur et l'utiliser pour accéder à l'option sélectionnée.

Partie 1 — Version 1 : le menu écrit à la main

Imagine que tu codes un mini-zoo et que tu dois afficher la liste des animaux disponibles pour que l'utilisateur en choisisse un. Première approche, naïve, mais qui marche :

```
print("Choisis un animal :")
print("Chat")
print("Lapin")
print("Capibara")
```

Résultat :

```
Choisis un animal :
Chat
Lapin
Capibara
```

Ça marche pour 3 animaux. Mais regarde ce qui cloche déjà :

- **Pas de numéros** : comment l'utilisateur exprime son choix ? Il doit taper le mot exact ? Avec ou sans accent ?
- **Pas évolutif** : si on ajoute un animal, il faut ajouter une ligne `print`.
- **Couplage fort** : le menu est figé dans le code, on ne peut pas le générer depuis un fichier ou une saisie.

> 🧠 ***Le symptôme*** *: plusieurs* `print` *qui se ressemblent comme deux gouttes d'eau. C'est exactement le signal qui dit « remplace-moi par une boucle ! ».*

Partie 2 — Version 2 : le menu via une liste + `for`

Première amélioration : ranger les animaux dans une **liste** et les afficher avec une boucle.

```
# La liste des animaux du zoo
animaux = ["Chat", "Lapin", "Capibara"]

print("Choisis un animal :")

# La boucle affiche un animal par ligne
for animal in animaux:
    print(animal)
```

Résultat identique à la version 1 :

```
Choisis un animal :
Chat
Lapin
Capibara
```

À première vue, c'est plus long que la version 1. **Mais** :

- **Pour ajouter un animal**, il suffit de l'ajouter à la liste : `animaux = ["Chat", "Lapin", "Capibara", "Lion"]`. La boucle l'affichera tout seul.
- **Pour 30 animaux**, ce code fait toujours 5 lignes.
- **Si la liste vient d'ailleurs** (un fichier, une base de données, une saisie utilisateur), seul `animaux = …` change. Le menu reste le même.

> 💡 ***Convention*** *rappelée du cours précédent : la liste est au* ***pluriel*** *(`animaux`), la variable de boucle au* ***singulier*** *(`animal`). Tu peux lire le code à voix haute : « pour chaque animal dans animaux, affiche-le ».*

C'est mieux, mais l'utilisateur ne sait toujours pas quoi taper pour choisir. Il faut un **numéro** devant chaque option.

Partie 3 — Version 3 : numéroter avec `enumerate()`

Ce qu'on veut afficher maintenant :

```
Choisis un animal :
1. Chat
2. Lapin
3. Capibara
```

Pour avoir un numéro qui s'incrémente automatiquement à chaque tour, Python fournit la fonction `enumerate()`. Elle prend une liste et te donne, à chaque tour, **deux valeurs** : l'index et l'élément.

```
animaux = ["Chat", "Lapin", "Capibara"]

print("Choisis un animal :")

for (i, animal) in enumerate(animaux):
    print(f"{i}. {animal}")
```

Résultat :

```
Choisis un animal :
0. Chat
1. Lapin
2. Capibara
```

Tu remarques quelque chose : la numérotation commence à **0**. C'est parce que Python compte à partir de 0 (l'index d'un élément dans une liste). Pour un humain, on préfère commencer à 1.

▸ Décortiquons `enumerate()`

```
for (i, animal) in enumerate(animaux):
#    └──────┬──────┘    └──────┬──────┘
#  deux variables     enumerate produit
#  par tour           des paires (index, élément)
```

À chaque tour de boucle, `enumerate` donne une **paire** :

TOUR	`I`	`ANIMAL`
1	0	`"Chat"`
2	1	`"Lapin"`
3	2	`"Capibara"`

> *i* **Sans** `enumerate()` *tu aurais dû gérer un compteur à la main :* `i = 0; for animal in animaux: print(i, animal); i += 1`*. C'est plus long, plus fragile, et personne ne fait ça en Python —* `enumerate` *est l'outil dédié.*

▸ Commencer à 1 au lieu de 0

Deux possibilités :

SOLUTION A — AJOUTER 1 DANS LE `PRINT`

```
for (i, animal) in enumerate(animaux):
    print(f"{i + 1}. {animal}")
```

SOLUTION B — LE PARAMÈTRE `START` DE `ENUMERATE()`

```
for (i, animal) in enumerate(animaux, start=1):
    print(f"{i}. {animal}")
```

Les deux donnent le même résultat. La solution B est légèrement plus propre — `start=1` exprime clairement l'intention.

```
Choisis un animal :
1. Chat
2. Lapin
3. Capibara
```

Partie 4 — Récupérer le choix de l'utilisateur

Le menu est joli, mais à quoi sert-il si l'utilisateur ne peut pas répondre ? Ajoutons la **saisie**.

```
animaux = ["Chat", "Lapin", "Capibara"]

print("Choisis un animal :")

for (i, animal) in enumerate(animaux, start=1):
    print(f"{i}. {animal}")

# Récupère la réponse, convertit en entier
choix = int(input("Ton choix ? "))

# Retrouve l'animal correspondant (attention : -1 car les listes commencent à 0)
animal_choisi = animaux[choix - 1]

print(f"Tu as choisi : {animal_choisi}")
```

Déroulement à l'exécution :

```
Choisis un animal :
1. Chat
2. Lapin
3. Capibara
Ton choix ? 2
Tu as choisi : Lapin
```

> ⚠️ ***Pourquoi*** `choix - 1` ***?*** *L'utilisateur tape* `2` *pour le deuxième animal. Mais dans la liste, le deuxième animal est à l'index* `1` *(les listes commencent à 0). D'où le* `-1`*. Ne te trompe pas dans ce petit décalage, c'est la source d'erreur n°1 sur les menus.*

▸ Cas problématique : que se passe-t-il si l'utilisateur tape n'importe quoi ?

- Il tape `0` → `animaux[-1]` → renvoie le **dernier** animal (`Capibara`). Comportement bizarre mais pas plantage.
- Il tape `99` → `animaux[98]` → **plantage** (`IndexError`).
- Il tape `chien` → `int("chien")` → **plantage** (`ValueError`).

Pour gérer ces cas proprement, il faut ajouter une **validation** — c'est typiquement le rôle d'une boucle `while` qui redemande tant que la réponse est invalide. On voit ça dans le cours sur les boucles `while` avec validation.

Partie 5 — Le programme complet, commenté

Voici la version finale, avec commentaires pédagogiques :

```
# --- Données du menu (modifie cette liste pour changer le menu) ---
animaux = ["Chat", "Lapin", "Capibara", "Lion", "Dragon de Komodo"]

# --- Affichage du menu ---
print("=== ZOO DU LYCÉE ===")
print("Choisis un animal à observer :")
print()  # ligne vide

for (i, animal) in enumerate(animaux, start=1):
    print(f"  {i}. {animal}")

print()

# --- Saisie du choix ---
choix = int(input("Ton choix ? "))

# --- Traitement du choix ---
animal_choisi = animaux[choix - 1]
print(f"\n→ Tu pars observer un {animal_choisi}. Bonne visite !")
```

Exécution :

```
=== ZOO DU LYCÉE ===
Choisis un animal à observer :

  1. Chat
  2. Lapin
  3. Capibara
  4. Lion
  5. Dragon de Komodo

Ton choix ? 3

→ Tu pars observer un Capibara. Bonne visite !
```

Note que pour **ajouter un animal**, il suffit de l'ajouter à la liste `animaux`. Tout le reste du programme s'adapte tout seul. **C'est ça, l'élégance d'un menu géré par boucle.**

🧪 Exercices

▸ Exercice guidé — Le menu du restaurant

Énoncé. Écris un programme qui :

1. Affiche le menu d'un restaurant avec ces plats : `["Lasagnes", "Salade César", "Burger", "Pizza margherita", "Tarte aux fruits"]`
2. Numérote chaque plat à partir de 1.
3. Demande à l'utilisateur son choix.
4. Affiche un message du type : `→ Commande validée : Pizza margherita.`

ÉTAPE 1 — RÉFLÉCHIR AU PLAN

Ce qu'il faut :

- Une **liste** des plats.
- Une **boucle for + enumerate** pour afficher avec numéros.
- Un **input** pour le choix.
- Un accès à la liste avec `[choix - 1]` pour récupérer le plat.

ÉTAPE 2 — LE SQUELETTE

```
plats = ["Lasagnes", "Salade César", "Burger", "Pizza margherita", "Tarte aux fruits"]

print("Menu du restaurant :")
# TODO : boucle d'affichage

# TODO : saisie du choix

# TODO : affichage de la commande
```

ÉTAPE 3 — LA BOUCLE D'AFFICHAGE

```
for (i, plat) in enumerate(plats, start=1):
    print(f"  {i}. {plat}")
```

(Convention pluriel/singulier : `plats` → `plat`.)

ÉTAPE 4 — LA SAISIE

```
choix = int(input("\nQuel plat veux-tu ? "))
```

ÉTAPE 5 — LA VALIDATION FINALE

```
plat_choisi = plats[choix - 1]
print(f"→ Commande validée : {plat_choisi}.")
```

ÉTAPE 6 — LE PROGRAMME COMPLET

```
plats = ["Lasagnes", "Salade César", "Burger", "Pizza margherita", "Tarte aux fruits"]

print("Menu du restaurant :")

for (i, plat) in enumerate(plats, start=1):
    print(f"  {i}. {plat}")

choix = int(input("\nQuel plat veux-tu ? "))
plat_choisi = plats[choix - 1]

print(f"→ Commande validée : {plat_choisi}.")
```

Exécution :

```
Menu du restaurant :
  1. Lasagnes
  2. Salade César
  3. Burger
  4. Pizza margherita
  5. Tarte aux fruits

Quel plat veux-tu ? 4
→ Commande validée : Pizza margherita.
```

▸ Exercice 2 — Le menu d'un jeu

Écris un menu de jeu vidéo avec les options :

```
options = ["Nouvelle partie", "Charger une partie", "Options", "Crédits", "Quitter"]
```

Le menu doit ressembler à ça :

```
╔═══════════════════╗
║   MENU PRINCIPAL  ║
╚═══════════════════╝
  1. Nouvelle partie
  2. Charger une partie
  3. Options
  4. Crédits
  5. Quitter

Ton choix > 1
Tu as choisi : Nouvelle partie
```

▸ Exercice 3 — Menu de classes du lycée

À partir de la liste :

```
classes = ["3TTR", "4TTR", "5TTR", "6TTR", "5TQ", "6TQ", "3GMS", "4GMS", "5GMS", "6GMS"]
```

Écris un programme qui demande à l'élève sa classe et affiche le message :

```
Classes disponibles :
  1. 3TTR
  2. 4TTR
  ...
  10. 6GMS

Ta classe ? 5
→ Bienvenue en 5TTR !
```

▸ Exercice 4 — Bonus : éviter la duplication

Tu veux pouvoir afficher **plusieurs menus différents** dans ton programme (menu principal, sous-menu options, etc.) — sans copier-coller la logique d'affichage. Écris une fonction `afficher_menu(titre, options)` qui prend un titre et une liste, et fait l'affichage. Indice : c'est juste la mise en commun de tout ce qu'on a vu, dans un `def`. (Si tu n'as pas encore vu les fonctions, garde cet exercice pour plus tard.)

✍️ À retenir

- Un menu console se construit avec **trois briques** : une liste d'options, une boucle `for + enumerate`, un `input`.
- `enumerate(liste)` donne à chaque tour une paire `(index, élément)` — utile dès qu'il faut un numéro devant les éléments.
- `enumerate(liste, start=1)` commence la numérotation à 1 — plus naturel pour un humain.
- Pour récupérer l'option choisie : `options[choix - 1]` (attention au `-1` puisque les listes commencent à 0).
- Pour ajouter ou retirer une option, on **modifie la liste** — rien d'autre ne change.
- La gestion des erreurs de saisie (lettres au lieu de numéros, numéro hors limites) demande une boucle `while` avec validation — sujet du cours suivant.

Suite

Le menu fonctionne, mais il **plante** si l'utilisateur tape n'importe quoi. Pour gérer ça proprement — redemander tant que la saisie est invalide — il nous faut une autre famille de boucles : **les boucles** `while`. On y vient.

Projets Python

> *Tu trouveras ici les consignes des différents projets à réaliser.*

🦸 Superhéros : couleurs, bonus et combat

> *Tu ajoutes trois nouvelles fonctionnalités à ton générateur de superhéros : des couleurs dans la console avec Rich, un bonus de caractéristique lancé au dé, et un combat final contre un boss.*

Ces étapes complètent ton programme de superhéros. Tu pars du code que tu as déjà écrit (origine, caractéristiques `force`, `agilite`, `intelligence`, `endurance`…) et tu lui ajoutes de la couleur, un bonus et un combat.

🎯 Objectifs

1. Colorier les affichages importants de la console avec le module Rich.
2. Appliquer un bonus aléatoire à une caractéristique selon l'origine du héros.
3. Résoudre un combat contre un boss à l'aide d'une boucle `for` et d'une condition.

🎨 Avant tout : ajouter des couleurs avec Rich

Ce qu'on veut faire

On veut colorier certains textes dans la console pour rendre le programme plus lisible — sans changer la structure du code.

Installation dans Thonny

Rich n'est pas installé par défaut. Dans Thonny :

1. Menu **Outils** → **Gérer les paquets…** (ou **CTRL+T**)
2. Tape `rich` dans la barre de recherche
3. Clique sur **Installer**

L'import à ajouter

Ajoute cette ligne **tout en haut** du fichier, avec les autres imports :

```
from rich import print
```

> 💡 *Cette seule ligne remplace silencieusement le `print()` de Python par celui de Rich. **Ton code existant ne change pas du tout.** La différence : tu peux maintenant ajouter des balises de couleur dans tes textes.*

Comment colorier du texte

On entoure le texte à colorier avec des balises entre crochets :

```
print("[bold green]Victoire ![/bold green]")
print("[bold red]Défaite...[/bold red]")
print(f"Score : [yellow]{score}[/yellow]")
print("[dim]Texte discret[/dim]")
```

Les couleurs : `red` `green` `yellow` `cyan` `magenta` `blue` `white` Les styles : `bold` (gras) · `italic` · `dim` (atténué)

> ⚠️ ***Règle de lisibilité :*** *colorie uniquement les valeurs importantes (résultats, erreurs, succès). Garde le texte descriptif en blanc normal. Trop de couleurs = aucune couleur.*

ÉTAPE 5.5 — Bonus de caractéristique

▸ Ce qu'on veut faire

Ton origine t'accorde un bonus sur une caractéristique. On lance un dé à 6 faces (d6) et on ajoute le résultat à la stat correspondante.

Chaque origine booste une stat différente :

ORIGINE	STAT BOOSTÉE
Mutation génétique	Force
Technologie	Agilité
Magie ancienne	Intelligence
Radioactivité	Endurance

▸ Quelle structure utiliser ?

On a besoin du **nom** de la stat boostée pour l'afficher → liste parallèle. On a besoin de **modifier la variable** → `if/elif` (comme à l'étape 5).

> 💡 *Les deux listes* `origines` *et* `boost_noms` *doivent être définies au même endroit que les autres,* ***tout en haut du programme****.*

Le code

```
# À ajouter en haut, avec les autres listes
boost_noms = ["Force", "Agilité", "Intelligence", "Endurance"]

# --- Étape 5.5 ---
# On récupère le nom de la stat boostée grâce à l'index du choix
stat_boostee = boost_noms[choix]

# On lance un d6
bonus = random.randint(1, 6)

# On applique le bonus à la bonne variable
if choix == 0:
    force += bonus
elif choix == 1:
    agilite += bonus
elif choix == 2:
    intelligence += bonus
elif choix == 3:
    endurance += bonus

print(f"[bold yellow] Bonus d6 : +{bonus} en {stat_boostee}[/bold yellow]")
print(f"[dim]{stat_boostee} passe à {intelligence}[/dim]")
```

💡 `force += bonus` *est une écriture courte pour* `force = force + bonus`.

Exemple d'exécution

```
 Bonus d6 : +4 en Intelligence     ← en jaune gras
Intelligence passe à 22            ← en gris discret
```

ÉTAPE 8 — Combat final

Ce qu'on veut faire

Ton héros affronte un boss. Le combat se résout en deux étapes :

1. On calcule le **nombre de coups** = `force + intelligence`
2. On lance autant de **d6** que de coups — la somme représente les dégâts infligés
3. Si les dégâts dépassent les PV du boss → victoire, sinon défaite

Quelle structure utiliser ?

On répète le même lancer un certain nombre de fois. ➡️ Le nombre de répétitions est connu à l'avance → boucle `for`.

Pour la victoire ou la défaite → résultat différent selon une condition → `if/else`.

Formule des PV du boss

Le boss doit être **difficile mais pas impossible** à battre. La formule choisie garantit environ **50 % de chances de victoire** :

```
boss_pv = nb_coups * 3 + random.randint(0, nb_coups)
```

> 🧠 ***Pourquoi ça marche ?***
>
> *Chaque d6 donne en moyenne **3.5**. Donc la somme des lancers tourne autour de `nb_coups × 3.5`. Le boss a entre `nb_coups × 3` et `nb_coups × 4` PV — ce qui le place juste autour de cette moyenne. Résultat : une chance sur deux de gagner, quelle que soit la valeur de `nb_coups`.*

Le code

```
nb_coups = force + intelligence
boss_pv  = nb_coups * 3 + random.randint(0, nb_coups)

print(f"\n[bold red]👹 Boss — Points de vie : {boss_pv}[/bold red]")
print(f"Tu lances [white]{nb_coups}[/white] dés (force {force} + intelligence {intelligence})")

# Lancer nb_coups dés et additionner les résultats
total_degats = 0
for i in range(nb_coups):
    total_degats += random.randint(1, 6)

print(f"Dégâts infligés : [bold white]{total_degats}[/bold white]")

# Résultat du combat
if total_degats > boss_pv:
    print(f"[bold green]✓ Combat GAGNÉ ! ({total_degats} > {boss_pv})[/bold green]")
else:
    print(f"[bold red]✗ Combat PERDU. ({total_degats} ≤ {boss_pv})[/bold red]")
```

Exemple d'exécution

```
👹 Boss — Points de vie : 78      ← en rouge gras
Tu lances 24 dés (force 9 + intelligence 15)
Dégâts infligés : 83
✓ Combat GAGNÉ ! (83 > 78)        ← en vert gras
```

🚀 Bonus — Tournoi de plusieurs combats

Au lieu d'un seul combat, ton héros dispute **3 combats** contre des boss différents. On compte ses victoires.

```python
NB_COMBATS = 3
victoires  = 0

for combat in range(NB_COMBATS):
    # Chaque boss a ses propres PV
    pv_boss = nb_coups * 3 + random.randint(0, nb_coups)
    degats  = sum(random.randint(1, 6) for _ in range(nb_coups))

    if degats > pv_boss:
        victoires += 1
        print(f"  Combat {combat + 1} : [green]Victoire[/green] ({degats} vs {pv_boss})")
    else:
        print(f"  Combat {combat + 1} : [red]Défaite[/red]  ({degats} vs {pv_boss})")

print(f"\n[bold]Résultat final : {victoires}/{NB_COMBATS} victoires[/bold]")
```

💡 `sum( ... for _ in range(n))` *est une façon compacte de lancer* `n` *dés et d'additionner les résultats d'un coup. C'est équivalent à la boucle* `for` *ci-dessus — à toi de choisir celle que tu préfères.*

▸ Exemple d'exécution

```
  Combat 1 : Victoire (88 vs 81)    ← Victoire en vert
  Combat 2 : Défaite  (71 vs 79)    ← Défaite en rouge
  Combat 3 : Victoire (92 vs 85)    ← Victoire en vert

Résultat final : 2/3 victoires
```

✍️ À retenir

- Le module **Rich** colore tes affichages sans changer la logique de ton code.
- Une **liste parallèle** (`boost_noms`) sert à retrouver un texte à partir d'un index.
- `+=` est un raccourci pour ajouter une valeur à une variable existante.
- Une **boucle** `for` répète un lancer un nombre de fois connu à l'avance.
- Un `if/else` choisit entre deux issues (victoire / défaite).

Python: L'opérateur 'f' et les f-strings

> *Lorsque vous programmez en Python, vous rencontrerez probablement l'opérateur 'f' dans le contexte des chaînes de caractères formatées. Cet opérateur, également appelé 'f-string', est un outil puissant et pratique pour formater les chaînes de caractères de manière claire et concise. Dans cet article, nous allons explorer en profondeur l'utilisation et les avantages de l'opérateur 'f' en Python.*

Qu'est-ce qu'un f-string ?

Un f-string est une **syntaxe spéciale de formatage de chaînes de caractères** introduite dans Python 3.6. Il permet d'incorporer des expressions Python directement dans les chaînes de caractères, ce qui simplifie grandement le processus de création de chaînes formatées. Pour créer un f-string, **vous ajoutez simplement un " `f` " ou " `F` " avant le début de la chaîne** et vous utilisez des **accolades** " `{}` " pour insérer des expressions Python à l'intérieur de la chaîne.

Voici un exemple simple d'utilisation de l'opérateur "f" :

```
name = "Alice"
age = 30
print(f"Bonjour, je m'appelle {name} et j'ai {age} ans.")
```

On peut aussi stocker le résultat dans une variable. Voici l'exemple précédent adapté:

```
name = "Alice"
age = 30

salution = f"Bonjour, je m'appelle {name} et j'ai {age} ans."

print(salutation)
```

Dans cet exemple, les variables `name` et `age` sont directement incorporées dans la chaîne de caractères à l'aide de l'opérateur "f". Lorsque vous exécutez ce code, vous obtenez la sortie suivante :

```
Bonjour, je m'appelle Alice et j'ai 30 ans.
```

Comme vous pouvez le voir, les valeurs des variables sont automatiquement substituées dans la chaîne formatée.

Formatage des nombres avec l'opérateur "f"

En plus de formater les chaînes de caractères, l'opérateur " `f` " peut également être utilisé **pour formater des nombres de manière précise**. Vous pouvez spécifier la précision des nombres flottants, le nombre de chiffres après la virgule, et même le format des nombres entiers.

Voici quelques exemples de formatage de nombres avec l'opérateur "f" :

```python
pi = 3.141592653589793
print(f"La valeur de pi avec 2 décimales : {pi:.2f}")
# Sortie : La valeur de pi avec 2 décimales : 3.14

percentage = 0.45678
print(f"Le pourcentage avec 1 décimale : {percentage:.1%}")
# Sortie : Le pourcentage avec 1 décimale : 45.7%

large_number = 1000000
print(f"Le grand nombre formaté avec séparateur de milliers : {large_number:,}")
# Sortie : Le grand nombre formaté avec séparateur de milliers : 1,000,000
```

Dans ces exemples, nous utilisons des spécificateurs de formatage après les deux-points ":" pour définir le format des nombres. Par exemple, ".2f" spécifie que nous voulons afficher le nombre avec deux décimales, "%.1f" spécifie une décimale pourcentage, et "," spécifie l'utilisation du séparateur de milliers pour les grands nombres entiers.

Spécifier le nombre de décimales des floats

Pour afficher une variable de type `float`, vous pouvez utiliser le spécificateur `.xf`, où `f` signifie '`float` et `x` représente le nombre de décimales à afficher. Notez que le résulat est **arrondi**!

```python
pi = 3.141592653589793
print(f"La valeur de pi avec 2 décimales : {pi:.2f}")
# Sortie : La valeur de pi avec 2 décimales : 3.14

print(f"La valeur de pi avec 2 décimales : {pi:.4f}")
# Sortie : La valeur de pi avec 2 décimales : 3.1416
```

Utilisation des pourcentages avec les f-strings

L'opérateur "f" en Python offre une manière simple et efficace d'afficher des pourcentages dans vos chaînes de caractères formatées. Vous pouvez formater les pourcentages avec précision en utilisant des spécificateurs de formatage appropriés dans les f-strings.

Affichage d'un pourcentage avec une décimale

```python
percentage = 0.45678
print(f"Le pourcentage avec 1 décimale : {percentage:.1%}")
# Sortie : Le pourcentage avec 1 décimale : 45.7%
```

Dans cet exemple, le spécificateur de formatage ".1%" indique à Python d'afficher le pourcentage avec une seule décimale.

Affichage d'un pourcentage avec deux décimales

```python
percentage = 0.789
print(f"Le pourcentage avec 2 décimales : {percentage:.2%}")
# Sortie : Le pourcentage avec 2 décimales : 78.90%
```

Ici, le spécificateur ".2%" est utilisé pour afficher le pourcentage avec deux décimales.

Affichage d'un pourcentage sans décimales

```
percentage = 0.25
print(f"Le pourcentage sans décimales : {percentage:.0%}")
# Sortie : Le pourcentage sans décimales : 25%
```

En spécifiant ".0%", vous pouvez afficher le pourcentage sans décimales.

Utilisation de pourcentages dans des calculs

Les f-strings permettent également d'incorporer des pourcentages dans des calculs et d'afficher les résultats formatés :

```
total = 1500
percentage_discount = 0.20
discount_amount = total * percentage_discount
print(f"Vous avez droit à une réduction de {percentage_discount:.0%}, ce qui correspond à {discou
# Sortie : Vous avez droit à une réduction de 20%, ce qui correspond à 300 euros.
```

Dans cet exemple, nous calculons le montant de la réduction en multipliant le total par le pourcentage de réduction, puis affichons le pourcentage formaté dans la chaîne de caractères.

En utilisant les spécificateurs de formatage appropriés avec les f-strings, vous pouvez facilement manipuler et afficher des pourcentages de manière précise dans vos programmes Python.

Avantages de l'opérateur "f"

L'utilisation de l'opérateur "f" présente plusieurs avantages :

1. Lisibilité améliorée

Les f-strings rendent le code plus lisible en permettant l'insertion directe de variables et d'expressions Python dans les chaînes de caractères. Cela rend le code plus concis et plus facile à comprendre.

2. Plus de clarté

En utilisant des f-strings, vous pouvez éviter les constructions de chaînes de caractères compliquées avec des opérations de concaténation. Cela rend votre code plus clair et moins sujet aux erreurs.

3. Performances améliorées

Les f-strings sont généralement plus performants que d'autres méthodes de formatage de chaînes de caractères, comme les méthodes `format()` ou `%`. Cela est dû à leur implémentation optimisée dans Python.

Bonnes pratiques

Lorsque vous utilisez des f-strings, gardez à l'esprit les bonnes pratiques suivantes :

- Utilisez des noms de variables descriptifs pour améliorer la lisibilité de votre code.
- Évitez d'incorporer des expressions trop complexes dans les f-strings, car cela peut rendre le code difficile à comprendre.

- Assurez-vous que les expressions incorporées sont sûres et ne présentent pas de risques de sécurité, surtout si elles proviennent d'une source externe.

Conclusion

En résumé, l'opérateur "f" en Python est un outil extrêmement utile pour formater les chaînes de caractères de manière claire et concise, ainsi que pour formater des nombres avec précision. En utilisant des f-strings, vous pouvez améliorer la lisibilité de votre code, éviter les erreurs de formatage et améliorer les performances de votre application. En incorporant les bonnes pratiques, vous pouvez tirer pleinement parti de la puissance de l'opérateur "f" dans vos projets Python.

Exercices

Enregistrez vos fichiers dans `Python \ Exercices \ f-strings`.

Le gardien de phare

Reprenez l'exercice du Gardien de phare et réécrivez la `#sortie` en utilisant une f-string.

Nom du fichier: `gardien-fstring-[prenom].py`

Aire et circonférence du cercle

Reprenez l'exercice sur le calcul de l'aire et de la circonférence du cercle.

Utilisez la valeur de `pi` offerte par le module `math` (`from math import pi`).

Réécrivez la `#sortie` en utilisant une f-string pour obtenir le résultat suivant (valeurs arrondies à 2 décimales):

```
Rayon? 10
Circonférence: 62.83
Aire: 314.16
```

Nom du fichier: `cercle-fstring-[prenom].py`

Calcul de la TVA

Ecrivez un programme qui demande un prix unitaire hors TVA et le nombre d'unités à l'utilisateur et permet de calculer une remise, le montant de la TVA et le montant TVA comprise.

```
taux_remise = 0.03
taux_tva = 0.21

# Votre code
```

Résultat attendu:

```
Prix unitaire HTVA? 100
Nombre de pièces? 10

Montant: 1000€
Remise 3%: 30€

Montant HTVA: 970€
TVA 21%: 203.70€

Montant TVAC: 1173.70€
```

Remarquez comment les taux de remise et de TVA ainsi que les montants sont affichés: `3%`, `21%`, `203.70€` (2 chiffres après la virgule).

Nom du fichier: `tva-fstring-[prenom].py`

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



HARDWARE

Les périphériques

Le clavier azerty

> *Le clavier AZERTY belge est l'outil principal pour saisir du texte sur les ordinateurs en Belgique. Bien qu'il ressemble fortement au clavier français, il possède ses propres particularités, notamment dans la manière d'écrire les accents, les symboles et les caractères spéciaux.*



Le clavier **AZERTY belge** est le clavier standard utilisé en Belgique. Il ressemble beaucoup au clavier français, mais présente plusieurs différences importantes :

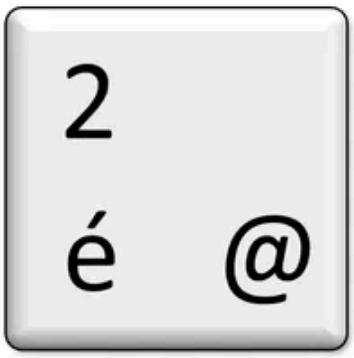
• Des caractères supplémentaires accessibles via **AltGr**, • Une disposition permettant d'écrire facilement en **français**, **néerlandais** et **allemand**, • Plusieurs combinaisons pour produire des **accents**, des **symboles mathématiques** ou des caractères moins courants.

L'objectif de ce cours est d'apprendre à utiliser efficacement ce clavier, y compris les majors, les accents, les symboles spéciaux et les combinaisons de touches les plus utiles.

Majors = les caractères du haut de la touche, obtenus avec Shift.

Structure d’une touche : ce qu’elle contient





Sur le clavier belge, chaque touche peut afficher **jusqu’à trois caractères** :

1. **En bas à gauche** : caractère obtenu par un appui simple.
2. **En haut à gauche** : caractère obtenu avec **Shift** (majuscule).
3. **En bas à droite** : caractère obtenu avec **AltGr**.

Exemple : la touche **E** peut afficher

- `e` (simple)
- `E` (Shift)
- `€` (AltGr)
- parfois un caractère supplémentaire selon les versions (rare)

Les touches de modification

Maj / Shift



La touche **Shift** sert à : • taper le caractère affiché **en haut** de la touche (*majors*), • produire une **majuscule** (tant qu'elle est appuyée)

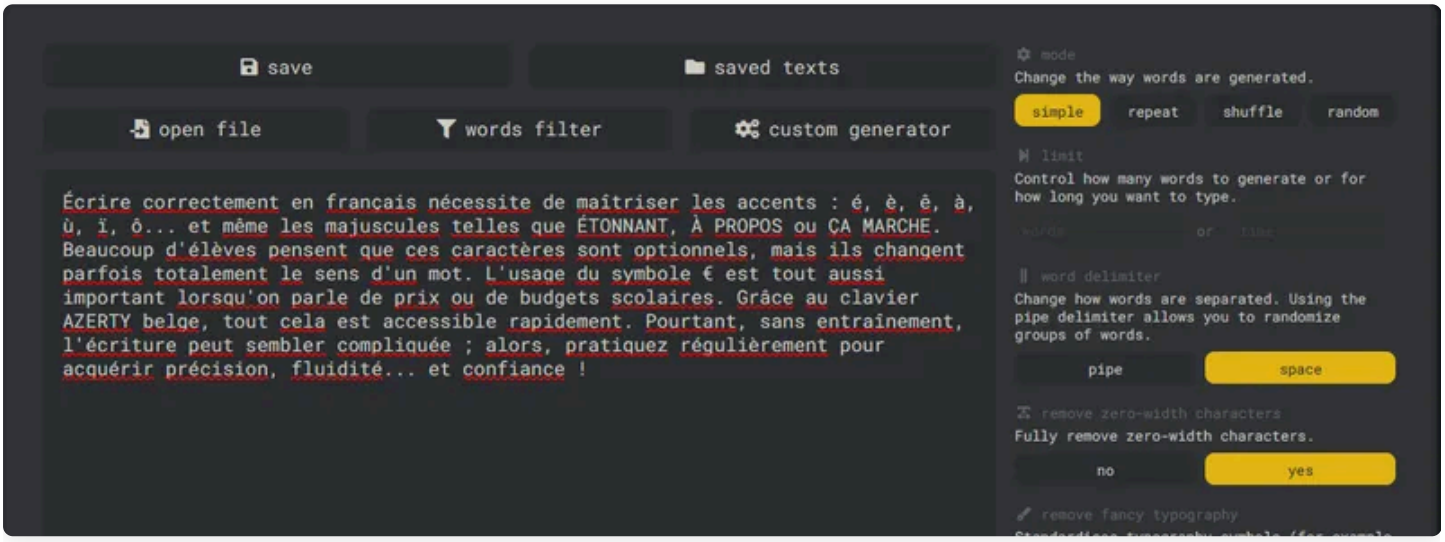
:buld: Astuce : pour écrire plusieurs lettres en majuscules, utilise **Caps Lock**. Elle est aussi appelée **Shift Lock**, **Verr. Maj**...

Pour n'écrire **que une ou quelques majuscules**, privilégie la touche "shift"

AltGr (Alt Graphique)



La touche **AltGr** permet d'obtenir les caractères affichés **en bas à droite** de certaines touches. Elle se trouve à drotie de la barre d'espace.



Exemples essentiels : • `AltGr` + `2` → `@` • `AltGr` + `3` → `#` • `AltGr` + `E` → `€` • `AltGr` + `9` → `{` • `AltGr` + `0` → `}` • `AltGr` + `^` → `[` • `AltGr` + `$` → `]` • `AltGr` + `<` → `\`

Les accents sur le clavier belge

Le clavier belge permet d'écrire **tous les accents français** facilement. Il faut distinguer deux familles :

Les voyelles accentuées « toutes faites »

Certaines lettres accentuées existent **directement sur des touches**.

• `é` : touche `2` • `à` : touche `0` • `ç` : touche `9`

Les accents « morts » (accent d'abord, lettre ensuite)

Un **accent mort** est une touche qui n'affiche rien immédiatement, mais applique un accent à la lettre suivante.

C'est aussi de cette manière que tu peux taper des **majuscules accentuées**.

ACCENT AIGU (´)

Touche : ´ (à droite de la touche M)

1. Appuie sur ´
2. Tape la voyelle → `ó`, `á`, `ú`, `É` etc.

ACCENT GRAVE (`)

Shift + touche ´ Puis tape une voyelle → `è`, `ì`, `ò`, `È` …

ACCENT CIRCONFLEXE (^)

AltGr + ^ Puis la voyelle → `â`, `ê`, `î`, `ô`, `û`

TRÉMA (¨)

Shift + ¨ Puis tape une voyelle → `ë`, `ï`, `ö`, `` ` `` …

💡 Pourquoi d'abord appuyer sur shift? Parce que le symbole `¨` se trouve en haut de la touche!

Faire des MAJUSCULES accentuées

Le clavier belge **permet facilement** les majuscules accentuées, indispensables en français.

Méthode :

1. Appuie sur l'accent correspondant (accent mort),
2. Appuie sur **Shift** en même temps que la lettre.

Exemples : • `´` + **Shift + E** → **É** • `^` + **Shift + A** → **Â** • `` ` `` + **Shift + A** → **À** • `¨` + **Shift + U** → **Ü**

Astuce : si cela ne fonctionne pas dans un logiciel, c'est que celui-ci ne supporte pas les accents morts. Word, LibreOffice et navigateurs les gèrent correctement.

Troois touches spéciales très utiles

Voici trois paragraphes prêts à insérer dans ton article de cours, rédigés dans un style pédagogique et adapté à des élèves.

▸ La touche Entrée (Enter)

La touche **Entrée**, souvent représentée par une flèche incurvée vers le bas et la gauche, sert à **valider** une action ou à **passer à la ligne**.

Dans un traitement de texte (Word, Google Docs…), elle crée un **nouveau paragraphe**.

Dans un formulaire, elle valide un champ ou envoie les données.

Dans les interfaces de fichiers, elle permet également d'**ouvrir un dossier** ou un programme sélectionné.

On peut considérer qu'il s'agit de la touche qui "confirme" une intention : taper Entrée revient à dire *OK, fais-le*.

Sur certains claviers, il existe deux touches Entrée : une grande à droite du clavier principal et une plus petite sur le pavé numérique, qui ont exactement la même fonction.

▸ La touche Retour arrière (Backspace)

La touche **Retour arrière**, représentée par une flèche vers la gauche, sert à **effacer le caractère situé *avant* le curseur**.

C'est l'équivalent d'un "effaceur" qui supprime ce qui vient d'être écrit.

Elle est différente de la touche **Delete**, qui efface *le caractère situé après le curseur*.

Retour arrière est la touche la plus utilisée lorsqu'on tape du texte : elle permet de corriger rapidement les fautes, les lettres en trop, les majuscules oubliées, etc.

Son comportement est instantané : chaque pression supprime un caractère, et si on la maintient, l'effacement se répète automatiquement.

▸ La touche Tabulation (Tab)

La touche **Tab** sert à insérer une **tabulation**, c'est-à-dire un déplacement horizontal du curseur vers la droite d'une distance fixe.

Dans un traitement de texte, cela crée une **indentation** ou aligne des éléments (listes, tableaux simples).

Dans les formulaires web, Tab permet de **passer au champ suivant** sans utiliser la souris (par exemple : du prénom vers le nom, puis vers l'email).

Dans les éditeurs de code (Python, HTML, etc.), Tab sert à **indenter le code**, ce qui est indispensable pour conserver une structure lisible et correcte.

Enfin, combinée avec **Shift**, Tab permet de revenir au champ précédent ou de réduire une indentation dans un éditeur.

Le pavé numérique

Si ton clavier en possède un :

• Active-le avec **Num Lock**. • Les touches du pavé permettent des chiffres rapides sans Shift. • Le pavé contient aussi :

- `/` (diviser)
- `*` (multiplier)
- `-`
- `+`
- `Enter` (pratique en Excel)

Conclusion

Le clavier AZERTY belge offre :

• une grande facilité pour écrire en **français**, **néerlandais**, **allemand**, • un accès rapide aux caractères informatiques via **AltGr**, • une gestion complète des **accents**, y compris des **majuscules accentuées**, • une organisation élégante des symboles.

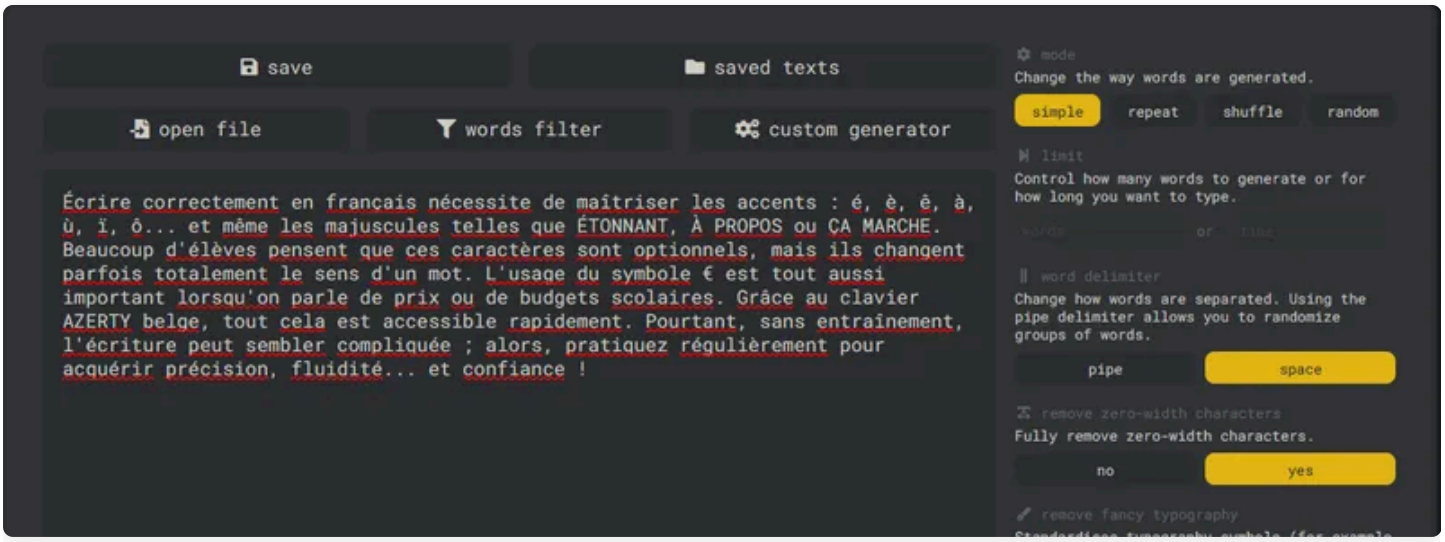
Avec un peu de pratique, il devient un outil très efficace au quotidien.

MonkeyType

Va sur le site MonkeyType, click sur la langue (english) -> français, puis "custom" et "Change".

Colle le texte suivant: `Écrire correctement en français nécessite de maîtriser les accents : é, è, ê, à, ù, ï, ô… et même les majuscules telles que ÉTONNANT, À PROPOS ou ÇA MARCHE. Beaucoup d'élèves pensent que ces caractères sont optionnels, mais ils changent parfois totalement le sens d'un mot. L'usage du symbole € est tout aussi important lorsqu'on parle de prix ou de budgets scolaires. Grâce au clavier AZERTY belge, tout cela est accessible rapidement. Pourtant, sans entraînement, l'écriture peut sembler compliquée ; alors, pratiquez régulièrement pour acquérir précision, fluidité… et confiance !`

Clique sur "Save"



JEUX VIDÉO

Tuto PyGame

> *Cette série d'articles va vous permettre d'apprendre les bases de pygame.*

Dans cette série de petits tutoriels, tu vas apprendre à:

- gérer les entrées-sorties dans ton jeu
- gérer les coordonnées d'un écran
- afficher une image de fond
- afficher un personnage
- déplacer ton personnage sur l'écran
- le faire réapparaître s'il sort de l'écran
- l'empêcher de sortir de l'écran
- gérer la gravité (la chute)

Structure du projet

Pour organiser un projet PyGame, il est recommandé d'utiliser une structure claire avec des dossiers dédiés. Voici une hiérarchie simple :

Crée un nouveau dossier qui s'appelle PyGame-Intro avec les sous-dossiers suivants.

```
PyGame-Intro/
|── images/
|── sons/
|── musiques/
```

Explication des dossiers

- **images/** : Stocke les fichiers d'images (PNG, JPG, etc.) pour les sprites, arrière-plans, icônes...
- **sons/** : Contient les effets sonores du jeu (bruits de pas, collisions, tirs...)
- **musiques/** : Regroupe les musiques d'ambiance et les bandes sonores du jeu.

Cette organisation facilite la gestion des ressources et améliore la clarté du projet.

Code de base d'un jeu PyGame

> *Voici le code de base minimum nécessaire pour pouvoir développer un jeu avec PyGame.*

Tous les jeux que tu vas développer avec pygame auront cette structure minimale de base.

Il n'est pas nécessaire de connaître la structure par cœur dans le cadre de ce cours, mais il est très intéressant de comprendre comment elle fonctionne.

Ton programme ci-dessous fonctionnera si ton écran est noir avec une petite boîte carrée affichée.

```
import pygame
import sys

# Initialisation
pygame.init()

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((800, 600))
clock = pygame.time.Clock()

# Boucle de jeu
running = True
while running:

    # Permet de quitter le programme proprement
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Mise à jour de l'écran
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

> 🚀 `pygame.init()`

▸ Description

La fonction `pygame.init()` permet d'**initialiser tous les modules** de la bibliothèque PyGame nécessaires au bon fonctionnement du jeu (graphismes, son, joystick, etc.). C'est la **toute première chose à faire** avant d'utiliser PyGame.

📜 Syntaxe

```
pygame.init()
```

🎯 À quoi ça sert ?

PyGame est composée de plusieurs modules : affichage, son, police de texte, gestion des entrées…
`pygame.init()` va essayer d'activer tous ces modules. Sans cette étape, beaucoup de fonctions PyGame ne fonctionneront pas correctement (voire pas du tout).

✅ Exemple

```
import pygame
pygame.init()
```

Une fois que c'est fait, tu peux créer une fenêtre, charger des images, jouer des sons, etc.

🧱 `screen = pygame.display.set_mode()`

La fonction `pygame.display.set_mode()` permet de **créer la fenêtre principale** du jeu, là où tous les éléments seront affichés. C'est l'une des premières fonctions à appeler après avoir initialisé PyGame avec `pygame.init()`.

📜 Syntaxe

```
surface = pygame.display.set_mode((largeur, hauteur))
```

📌 Paramètres

PARAMÈTRE	TYPE	DESCRIPTION
`(largeur, hauteur)`	`tuple`	La taille de la fenêtre de jeu, en pixels (largeur x hauteur)

✅ Exemple

```
screen = pygame.display.set_mode((800, 600))
```

- Crée une fenêtre de **800 pixels de large** et **600 pixels de haut**.
- La variable `screen` contient la surface sur laquelle on pourra dessiner (`blit` , `draw` , etc.).

💡 Bonnes pratiques

- Stocke toujours le résultat dans une variable (`screen`) : c'est **la surface principale** sur laquelle tu dessines.
- Utilise des **variables** pour la taille de l'écran (ex. `LARGEUR_ECRAN = 800`) pour pouvoir les réutiliser ailleurs.

🕒 `clock = pygame.time.Clock()`

Cette instruction crée un **objet "horloge"** dans PyGame.
Il est utilisé pour **contrôler la vitesse d'exécution du jeu**, c'est-à-dire le **nombre d'images par seconde** (FPS = frames per second).

🎯 À quoi ça sert ?

Dans un jeu, il est important que la boucle de jeu **ne tourne pas trop vite**.
Sinon, le jeu serait **trop rapide**, ou tournerait à une vitesse différente selon l'ordinateur.

Avec un objet `Clock`, tu peux **ralentir volontairement** la boucle de jeu pour avoir un rythme stable.

🔁 `while running:`

📄 Ce que ça fait

Ce code crée une **boucle infinie contrôlée par une variable**. C'est une **structure de base dans un jeu vidéo**, souvent appelée la **boucle de jeu** (*game loop*).

🧠 Explication

```
running = True
while running:
    # instructions exécutées en boucle
```

- `running` est une variable booléenne (vrai ou faux).
- Tant que `running` vaut `True`, le code **à l'intérieur de la boucle** continue à s'exécuter.
- Dès qu'on met `running = False`, la boucle s'arrête.

🎮 À quoi ça sert dans un jeu ?

C'est la **boucle principale du jeu**. Elle tourne en continu pour :

1. Lire les entrées clavier/souris
2. Mettre à jour les positions, les collisions, les scores…
3. Redessiner l'écran

🔄 `for event in pygame.event.get():`

Ce groupe d'instructions permet de vérifier tous les événements qui se passent dans pygame. Dans un premier temps, pour notre première série d'exercices, le plus important est simplement de se rappeler qu'**elle permet de quitter le jeu proprement**.

🕒 `clock.tick(60)`

`clock.tick(60)` limite la boucle à **60 tours par seconde**, donc le jeu s'exécutera à **60 FPS** maximum.

🔄 `pygame.display.flip()`

📄 Description

`pygame.display.flip()` met à **jour l'écran complet** avec tout ce qui a été dessiné depuis le début de la boucle. C'est l'étape **indispensable** pour que le joueur voie les changements à l'écran : sans elle, rien ne s'affiche.

🎯 Pourquoi c'est important ?

PyGame utilise un système de **double affichage** (*double buffering*) :

- Tu dessines **dans l'ombre** (sur une surface mémoire invisible).
- Ensuite, `flip()` **montre à l'écran** ce que tu viens de préparer.

Cela évite les scintillements et rend le jeu fluide.

✅ Exemple typique dans une boucle de jeu

```
while running:
    screen.fill((0, 0, 0))               # Effacer l'écran
    pygame.draw.rect(screen, rouge, ...) # Dessiner un élément
    pygame.display.flip()                # Mettre à jour l'affichage
```

💡 Bonnes pratiques

- Utilise **une seule fois** `flip()` à la fin de chaque boucle.
- Dessine **tout** avant de l'appeler.

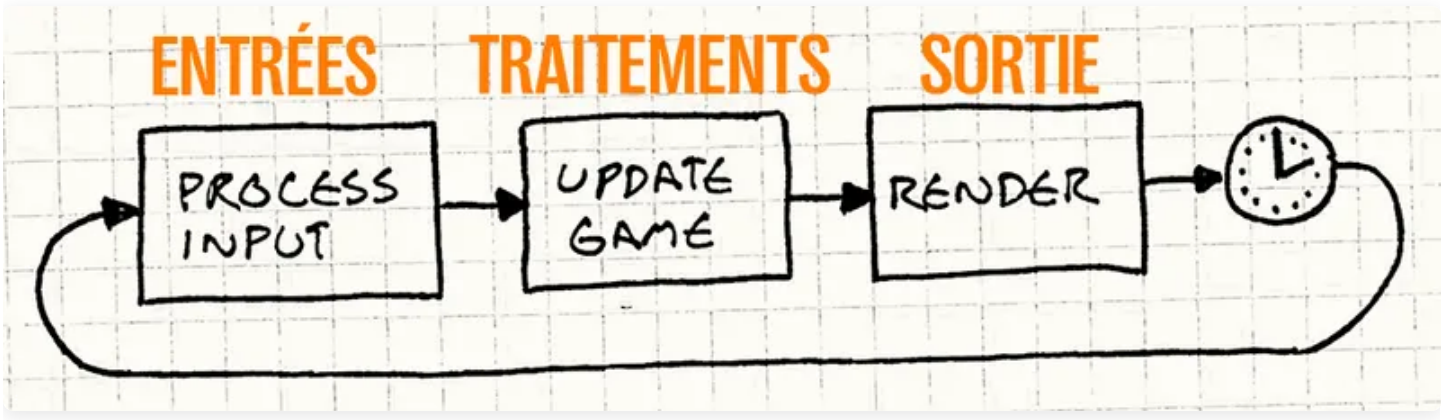
Introduction à la boucle de jeu (game loop)

> *La boucle de jeu, ou **game loop** en anglais, est l'un des concepts les plus fondamentaux dans le développement de jeux vidéo. Que vous utilisiez un moteur de jeu avancé ou que vous programmiez un jeu en 2D avec une bibliothèque comme PyGame, la boucle de jeu est essentielle pour réagir aux actions du joueur, gérer la logique du jeu et mettre à jour les éléments graphiques.*

Qu'est-ce qu'une boucle de jeu ?

Une boucle de jeu est une structure répétitive qui fait tourner le jeu en continu jusqu'à ce que le joueur quitte. Elle permet de :

1. **Lire les entrées du joueur** (clavier, souris...).
2. **Mettre à jour l'état du jeu** (déplacements, collisions...).
3. **Rafraîchir l'affichage** pour montrer les changements à l'écran.



Structure d'une boucle de jeu

1. **Initialisation** (création de la fenêtre, chargement des ressources...)
2. **Boucle principale** :
   - Gérer les entrées
   - Mettre à jour le jeu
   - Redessiner l'écran
3. **Sortie propre** (fermeture du jeu)

Exemple simple en PyGame

Ton programme ci-dessous fonctionnera si ton écran est noir avec une petite boîte carrée affichée.

```python
import pygame
import sys

# Initialisation
pygame.init()

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((800, 600))
clock = pygame.time.Clock()

# position initiale du joueur
player_x = 100
player_y = 100

player_color = (255, 0, 255)  # Rose

# Boucle de jeu
running = True
while running:

    # Permet de quitter le programme proprement
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Mise à jour de l'écran
    # Dessiner le joueur (rectangle)
    pygame.draw.rect(screen, player_color, (player_x, player_y, 50, 50))
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

 `pygame.draw.rect()`

La fonction `pygame.draw.rect()` permet de **dessiner un rectangle** sur une surface (par exemple l'écran de jeu). C'est très utile pour représenter un joueur, un mur, un bouton ou tout autre élément graphique simple.

📄 Syntaxe

```
pygame.draw.rect(surface, couleur, rectangle)
```

📌 Paramètres

PARAMÈTRE	TYPE	DESCRIPTION
`surface`	`pygame.Surface`	L'endroit où le rectangle doit être dessiné (souvent l'écran principal, par exemple `screen`).
`couleur`	`tuple`	La couleur du rectangle, sous forme `(R, G, B)`, avec des valeurs entre 0 et 255.
`rectangle`	`tuple` ou `pygame.Rect`	La position et la taille du rectangle : `(x, y, largeur, hauteur)`. `x, y` = coin supérieur gauche. `largeur`, `hauteur` = taille en pixels.

✅ Exemple

```
pygame.draw.rect(screen, player_color, (player_x, player_y, 50, 50))
```

- `screen` : l'écran sur lequel on dessine.
- `player_color` : une couleur, par exemple `(255, 0, 0)` pour rouge.
- `(player_x, player_y, 50, 50)` : on dessine un rectangle de 50×50 pixels à la position `(player_x, player_y)`.

🖼️ Résultat

Cela dessine un carré rouge (ou de la couleur choisie) à l'écran, à la position indiquée, de taille 50x50 pixels. Il est visible **à la prochaine mise à jour de l'affichage** avec `pygame.display.flip()`.

💡 Bonnes pratiques

- **Garde la boucle claire** : sépare les entrées, la logique et l'affichage (**entrée**, **traitements**, **sorties**).
- **Utilise un framerate constant** (clock.tick(60)) pour éviter les variations de vitesse.
- **Ne surcharge pas la boucle** : place les calculs lourds dans des fonctions à part.

Exercices à essayer

- Change les coordonnées (`player_x` et `player_y`) du joueur pour changer sa position à l'écran.
  - Identifie l'impact de ces changement sur ton personnage
- Affiche plusieurs rectangles avec des tailles et des couleurs différentes
- Grâce à une boucle `for` affiche 10 rectangles à des positions aléatoires (au hasard)
- Pour les ninjas: affiche 10 rectangles avec des positions, tailles et couleurs aléatoires

Cette boucle de jeu est la base pour créer un jeu interactif. Ajoute des éléments et expérimente !

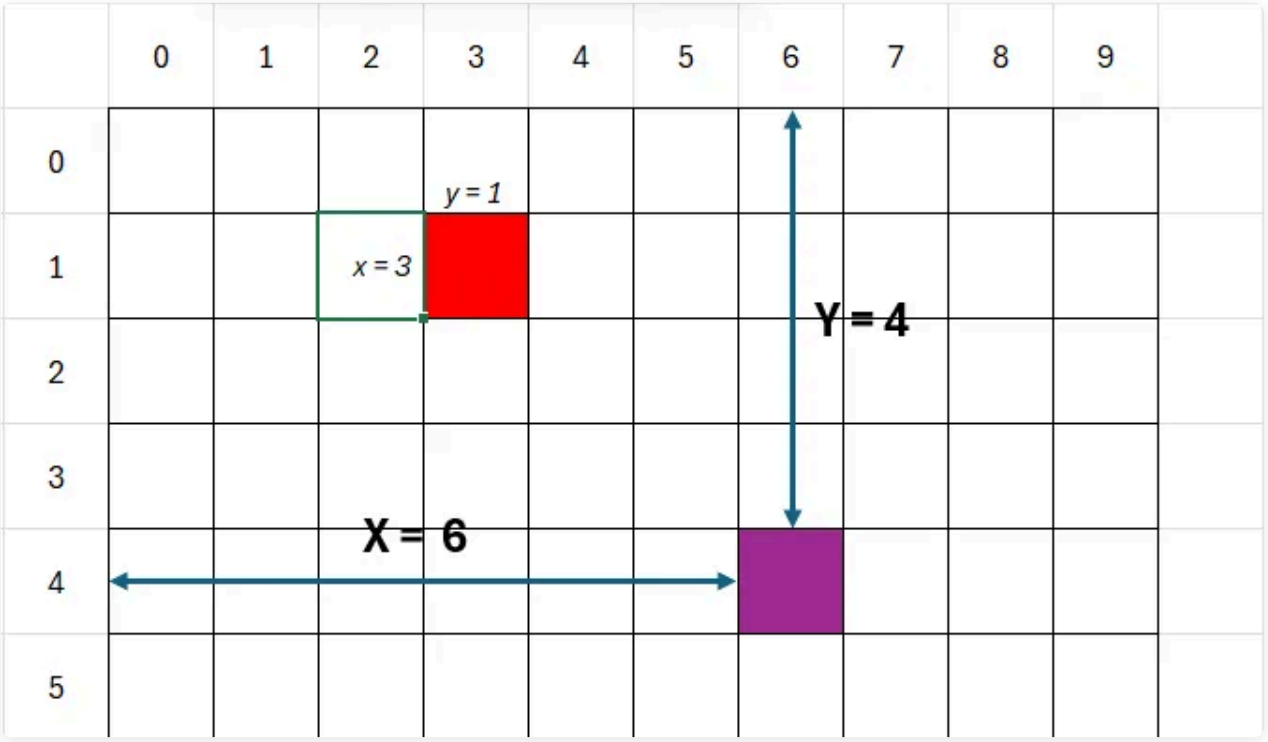
Gestion des coordonnées en pixels dans un jeu vidéo

> *Dans un jeu vidéo, chaque élément visuel est placé sur un écran en fonction d'un système de coordonnées en pixels. Comprendre ce système est essentiel pour positionner correctement les objets, gérer les déplacements et éviter qu'ils ne sortent des limites de l'affichage.*

Système de coordonnées sur un écran

Dans un jeu vidéo, l'écran est une grille de pixels où chaque point est repéré par une coordonnée **(x, y)** :

- **x** : position **horizontale** (0 à gauche, augmente vers la droite).
- **y** : position **verticale** (0 en haut, augmente vers le bas).



Exemple : dans une fenêtre de **800×600** pixels :

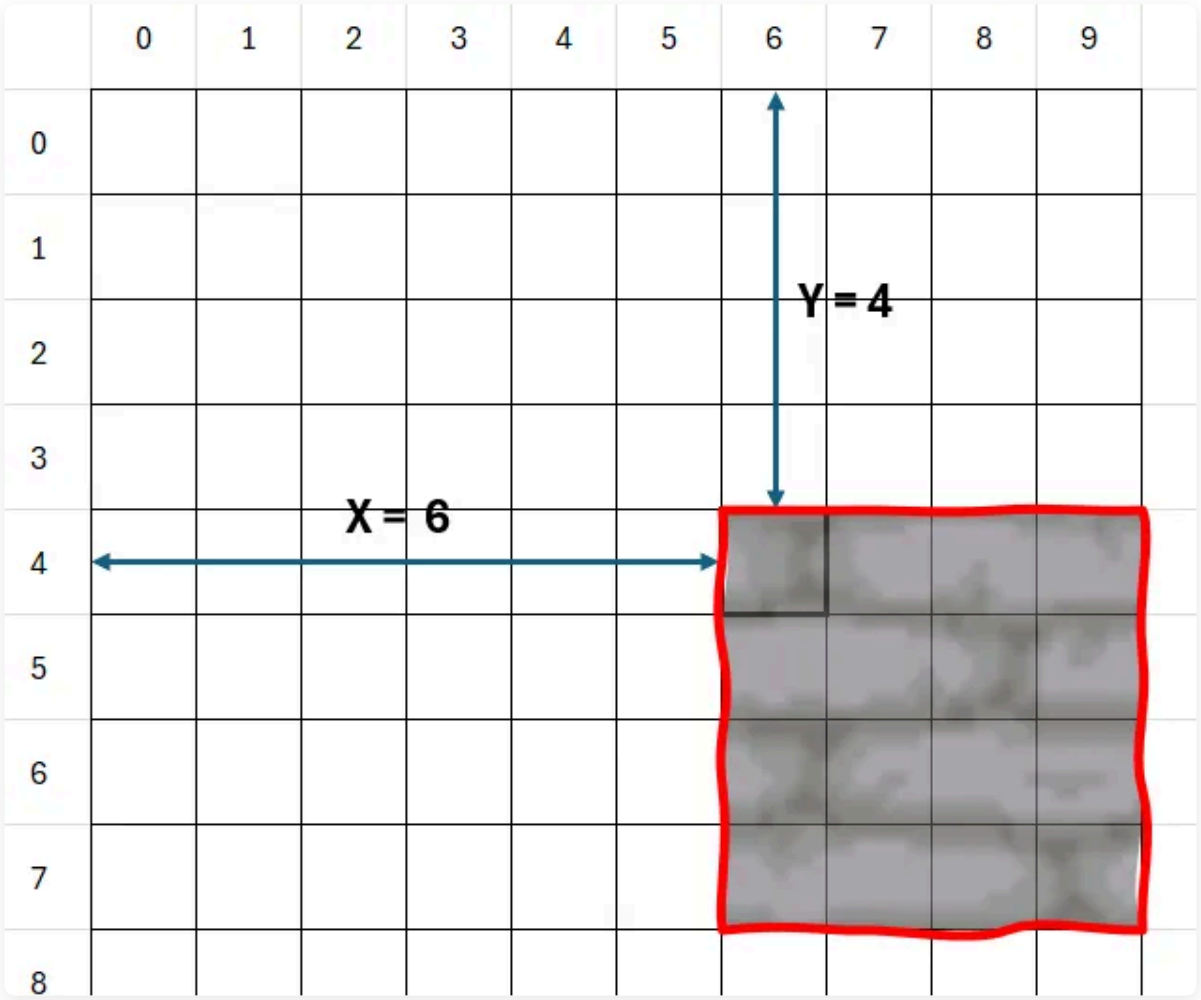
- (0,0) est en haut à gauche.
- (800,0) est en haut à droite.
- (0,600) est en bas à gauche.
- (800,600) est en bas à droite.

Positionner un élément

Dans PyGame, on place un élément en utilisant `blit()` :

```
screen.blit(image, (x, y))
```

L’image sera affichée **avec son coin supérieur gauche** à (x, y).



Conclusion

La gestion des coordonnées est essentielle pour déplacer et positionner les objets dans un jeu. Comprendre le référentiel et maîtriser les limites de l’écran permettent de mieux contrôler le gameplay et l’affichage.

Afficher un personnage à l'écran en PyGame

> *L'affichage d'un personnage est une étape essentielle dans un jeu vidéo. Voici comment le charger et l'afficher dans une fenêtre PyGame.*

Pour pouvoir afficher un personnage à l'écran, il faut connaître sa position et l'image utilisée pour le représenter.

Comme on l'a aperçu précédemment, la position du joueur est gérée via des coordonnées x et y. Ces coordonnées seront stockées dans des variables.

Pygame permet de charger facilement une image.

🔥 Dans la plupart des cas, tu devras utiliser des images avec un **fond transparent**. Tu utiliseras donc généralement des images au format `PNG`.

Charger et afficher une image

1. **Charger l'image du personnage** :

```
player = pygame.image.load('images/player.png')
```

Tu peux télécharger des images de personnages depuis la section téléchargement. Remplace le nom de fichier par celui dont tu as besoin et déplace l'image dans le dossier `images` de ton de projet (dossier que tu créeras au besoin).

2. **Définir sa position initiale** :

```
player_x = 100
player_y = 100
```

Player en anglais signifie joueur en français. On a choisi d'appeler la variable comme ça parce qu'elle représente la position `x` et la position `y` d'un player. On aurait pu l'appeler `joueur_x`, `perso_x`, `personnage_x`... Au choix, tant que ça a du sens pour s'y retrouver plus tard.

3. **L'afficher à l'écran** :

```
screen.blit(player, (player_x, player_y))
```

Exemple complet

```
import pygame
import sys

# Initialisation
pygame.init()

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((800, 600))
player = pygame.image.load('player.png')

clock = pygame.time.Clock()

# position initiale du joueur
player_x = 100
player_y = 100

# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Mise à jour de l'écran
    screen.blit(player, (player_x, player_y))
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

Ce code est identique à celui réalisé dans l'introduction. Dans les articles suivants, nous allons animer le personnage et ajouter un fond d'écran.

Astuces

- Assure-toi que l'image du personnage est au format `PNG` avec un **fond transparent**.

Avec ces bases, ton personnage apparaît à l'écran, prêt pour être animé et déplacé !

Gérer les entrées au clavier

> *Dans un jeu vidéo, gérer les entrées clavier est essentiel pour contrôler les déplacements du personnage.*

Gestion des entrées clavier dans PyGame

La notion de **vitesse** est également importante : elle détermine combien de pixels le personnage se déplacera à chaque action. Une vitesse plus élevée entraînera des déplacements plus rapides et plus grands à l'écran, tandis qu'une vitesse plus faible permettra des mouvements plus précis et contrôlés.

Déplacement avec les flèches

Utilise les constantes de PyGame pour détecter les touches fléchées :

```
keys = pygame.key.get_pressed() # Récupère la liste de toutes les touches qui sont appuyées par le

if keys[pygame.K_LEFT]: # gauche
    player_x -= vitesse
if keys[pygame.K_RIGHT]: # droite
    player_x += vitesse
if keys[pygame.K_UP]: # haut
    player_y -= vitesse
if keys[pygame.K_DOWN]: # bas
    player_y += vitesse
```

- `K_LEFT` : gauche
- `K_RIGHT` : droite
- `K_UP` : haut
- `K_DOWN` : bas

Dans la boucle de jeu

Gérer les entrées utilisateurs est généralement la première chose que tu fais dans ta boucle de jeu. Voici la boucle de jeu originale des exercices précédents adaptées avec la gestion des touches:

```python
import pygame
import sys

# Initialisation
pygame.init()

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((800, 600))
player = pygame.image.load('player.png')

clock = pygame.time.Clock()

# position initiale du joueur
player_x = 100
player_y = 100
vitesse = 5

# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT] or keys[pygame.K_q]:
        player_x -= vitesse
    if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
        player_x += vitesse
    if keys[pygame.K_UP] or keys[pygame.K_z]:
        player_y -= vitesse
    if keys[pygame.K_DOWN] or keys[pygame.K_s]:
        player_y += vitesse

    # Mise à jour de l'écran
    screen.blit(player, (player_x, player_y))
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

Pourquoi je peins mon écran ?

Il manque un petit quelque chose dans la gestion de l'affichage de notre jeu. A chaque fois qu'on dessine l'image à une nouvelle position, Pygame se contente de dessiner l'image une nouvelle fois sur l'écran. Et en fonction de la vitesse de déplacement, celle ci vient se dessiner au dessus des autres images dessinnées précédemment. Ce qui donne un petit peu l'effet pinceau.

Pour corriger ce problème: à chaque fois, avant d'afficher ton personnage, il faut entièrement recolorier/effacer le fond de l'écran. Dans un premier temps, on se contente de le colorier en noir et on apprendra plus tard comment changer la couleur et afficher des images de fond.

```
ecran.fill((0, 0, 0))  # Effacer l'écran en noir
```

A ajouter juste avant la ligne `screen.blit(player, (player_x, player_y))`.

Touches gamer habituelles

Pour une expérience de jeu plus complète, on utilise souvent les touches suivantes :

```
if keys[pygame.K_z]:  # Monter ou avancer
    player_y -= vitesse
if keys[pygame.K_q]:  # Gauche
    player_x -= vitesse
if keys[pygame.K_s]:  # Bas
    player_y += vitesse
if keys[pygame.K_d]:  # Droite
    player_x += vitesse
```

- **ZQSD** : déplacements haut, gauche, bas, droite

Bonnes pratiques

- Regroupe la gestion des entrées dans une seule section de la boucle de jeu.
- Ajuste la variable `vitesse` pour contrôler la rapidité des mouvements.
- Sépare clairement le traitement des entrées du reste du code pour une meilleure lisibilité.

Avec ces bases, tu peux efficacement contrôler ton personnage dans PyGame !

Déplacer un personnage en PyGame

> *Le déplacement d'un personnage est une des bases de l'interactivité dans un jeu.*

Chaque mouvement modifie la position du personnage en ajustant ses coordonnées **(x, y)** sur l'écran.

Mouvements horizontaux

- Une augmentation de `x` le fait avancer vers la droite (+)
- une diminution de `x` le fait reculer vers la gauche (-)

Mouvements verticaux

- une augmentation de `y` le déplace vers le bas (+)
- une diminution de `y` le fait monter (-)

C'est une bonne pratique de stocker la vitesse dans une variable. Ainsi, tu pourras changer la vitesse à laquelle le personnage se déplace à l'écran en changeant le nombre de pixels de chaque déplacement.

```
vitesse =  5
```

Voici comment gérer ces déplacements en PyGame avec les touches du clavier.

Capturer les entrées clavier

On utilise `pygame.key.get_pressed()` pour détecter les touches enfoncée :

```
    # Touches pressées en continu
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        player_x -= vitesse
    if keys[pygame.K_RIGHT]:
        player_x += vitesse
    if keys[pygame.K_UP]:
        player_y -= vitesse
    if keys[pygame.K_DOWN]:
        player_y += vitesse
```

Exemple complet

```python
import pygame
pygame.init()

# Initialisation de la fenêtre
largeur_ecran = 800
hauteur_ecran = 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Déplacement du personnage")

# Charger le personnage
player = pygame.image.load('player.png')

player_x = 100
player_y = 100
vitesse = 5

# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Touches pressées en continu
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        player_x -= vitesse
    if keys[pygame.K_RIGHT]:
        player_x += vitesse
    if keys[pygame.K_UP]:
        player_y -= vitesse
    if keys[pygame.K_DOWN]:
        player_y += vitesse

    screen.fill((0, 0, 0))  # Effacer l'écran
    screen.blit(player, (player_x, player_y))  # Afficher le personnage
    pygame.display.flip()  # Rafraîchir l'écran

pygame.quit()
```

Astuces

- Utiliser `pygame.key.get_pressed()` permet un mouvement fluide.
- Limiter la position du personnage pour qu'il ne sorte pas de l'écran.
- Ajuster `vitesse` pour accélérer ou ralentir le déplacement.

Avec ces bases, ton personnage peut se déplacer librement à l’écran !

Afficher une couleur en fond d'écran en PyGame

> *Un fond d'écran va rendre ton jeu plus joli. Utiliser une couleur comme fond d'écran est une méthode simple et efficace.*

1. **Définir une couleur de fond** :

```
background_color = (50, 50, 50)  # Gris foncé
```

2. **Remplir l'écran** :

```
screen.fill(background_color)
```

Exemple complet

```
import pygame
pygame.init()

# Initialisation de la fenêtre
largeur_ecran, hauteur_ecran = 800, 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
background_tile = pygame.image.load('background_tile.png')
background_color = (50, 50, 50)

# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill(background_color)  # Étape 1 : fond uni

    pygame.display.flip()  # Rafraîchir l'écran

pygame.quit()
```

▸ Théorie des couleurs en informatique (RGB)

En informatique, la couleur est définie par le modèle **RGB (Red, Green, Blue)**. Chaque couleur est une combinaison de trois valeurs :

- **Rouge (R)**
- **Vert (G)**
- **Bleu (B)**

Chaque composante peut varier de **0 à 255**, où 0 signifie absence de couleur et 255 représente l'intensité maximale.

EXEMPLES DE COULEURS

- **Couleurs primaires**
  - Rouge : `(255, 0, 0)`
  - Vert : `(0, 255, 0)`
  - Bleu : `(0, 0, 255)`
- **Couleurs secondaires (mélange de deux primaires)**
  - Magenta (Rouge + Bleu) : `(255, 0, 255)`
  - Jaune (Rouge + Vert) : `(255, 255, 0)`
  - Cyan (Vert + Bleu) : `(0, 255, 255)`
- **Autres couleurs**
  - Blanc (toutes les couleurs à 255) : `(255, 255, 255)`
  - Noir (absence de couleur) : `(0, 0, 0)`
  - Gris (valeurs égales) : `(128, 128, 128)`

Méthode 2 : Utiliser un motif répétitif

Dans un jeu vidéo, il est parfois utile d'utiliser une image en motif (pattern) qui se répète pour couvrir tout l'écran. Voici comment procéder en PyGame.

Tu peux télécharger des images de fond dans la section téléchargement. Dans les exemples suivants, remplace le nom de fichier par celui dont tu as besoin.

1. **Charger l'image du motif** :

```
background_tile = pygame.image.load('background.png')
```

2. **Dessiner l'image en répétition** :

```
for x in range(0, largeur_ecran, background_tile.get_width()):
    for y in range(0, hauteur_ecran, background_tile.get_height()):
        screen.blit(background_tile, (x, y))
```

Charger et répéter une image en fond

1. **Charger l'image du motif** :

```
background_tile = pygame.image.load('background.png')
```

2. **Dessiner l'image en répétition** :

```
for x in range(0, largeur_ecran, background_tile.get_width()):
    for y in range(0, hauteur_ecran, background_tile.get_height()):
        screen.blit(background_tile, (x, y))
```

▸ Exemple complet

```
import pygame
pygame.init()

# Initialisation de la fenêtre
largeur_ecran, hauteur_ecran = 800, 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
background_tile = pygame.image.load('background.png')

# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Remplir l'écran avec le motif répété
    for x in range(0, largeur_ecran, background_tile.get_width()):
        for y in range(0, hauteur_ecran, background_tile.get_height()):
            screen.blit(background_tile, (x, y))

    pygame.display.flip()  # Rafraîchir l'écran

pygame.quit()
```

Astuces

- Choisis une image qui s'assemble bien pour éviter les coupures visibles.
- Assure-toi que la taille du motif est optimisée pour éviter des répétitions trop évidentes.
- Pour un effet plus dynamique, ajoute un léger déplacement du motif à chaque frame pour simuler un défilement.

Avec cette technique, tu peux créer facilement des arrière-plans immersifs et fluides !

Bonus

Pour les plus courageux: essaye d'afficher un mur autours de la fenêtre.

Cela peut se faire avce 3 boucles:

1. Une pour le mur du haut
2. Une pour les murs verticaux
3. Une pour le mur du bas

Afficher des images en fond d'écran en PyGame

> *Un fond d'écran va rendre ton jeu plus joli. Utilisez des images donnera un meilleur résultat qu'une simple couleur de fond..*

Utiliser un motif répétitif

Dans un jeu vidéo, il est parfois utile d'utiliser une image en motif (pattern) qui se répète pour couvrir tout l'écran. Voici comment procéder en PyGame.

Tu peux télécharger des images de fond dans la section téléchargement. Dans les exemples suivants, remplace le nom de fichier par celui dont tu as besoin.

1. **Charger l'image du motif** :

```
background_tile = pygame.image.load('background.png')
```

2. **Dessiner l'image en répétition** :

```
for x in range(0, largeur_ecran, background_tile.get_width()):
    for y in range(0, hauteur_ecran, background_tile.get_height()):
        screen.blit(background_tile, (x, y))
```

Le code fait exactement ça :

- Il commence tout en haut à gauche de l'écran.
- Il pose ton image carrée une fois.
- Ensuite, il avance vers la droite et pose à nouveau ton image, juste à côté de la première, sans laisser d'espace.
- Il fait ça jusqu'à remplir toute la ligne du haut.
- Quand il arrive au bout, il descend d'une ligne, et recommence depuis le début.
- Il fait ça ligne après ligne, jusqu'à ce que tout l'écran soit rempli.

Comme ça, tu obtiens un beau motif avec ton image, qui se répète partout sur l'écran !

▸ Exemple complet

```python
import pygame
import sys

# Initialisation
pygame.init()
largeur_ecran = 800
hauteur_ecran = 600

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
clock = pygame.time.Clock()

# position initiale du joueur
player_x = 100
player_y = 100
vitesse = 5

# Chargement des images
player = pygame.image.load('player.png')
background_tile = pygame.image.load('sol_paves.png')


# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Mise à jour de l'écran
    # Remplir l'écran avec le motif répété
    for x in range(0, largeur_ecran, background_tile.get_width()):
        for y in range(0, hauteur_ecran, background_tile.get_height()):
            screen.blit(background_tile, (x, y))

    screen.blit(player, (player_x, player_y))
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

:: box-light 💣 on ajouté les variables `largeur_ecran` et `hauteur_ecran`

::

Exemple avec déplacement du personnage

```
import pygame
import sys

# Initialisation
pygame.init()
largeur_ecran = 800
hauteur_ecran = 600

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
clock = pygame.time.Clock()

# position initiale du joueur
player_x = 100
player_y = 100
vitesse = 5

# Chargement des images
player = pygame.image.load('player.png')
background_tile = pygame.image.load('sol_paves.png')


# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT] or keys[pygame.K_q]:
        player_x -= vitesse
    if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
        player_x += vitesse
    if keys[pygame.K_UP] or keys[pygame.K_z]:
        player_y -= vitesse
    if keys[pygame.K_DOWN] or keys[pygame.K_s]:
        player_y += vitesse

    # Mise à jour de l'écran
    # Remplir l'écran avec le motif répété
    for x in range(0, largeur_ecran, background_tile.get_width()):
        for y in range(0, hauteur_ecran, background_tile.get_height()):
            screen.blit(background_tile, (x, y))

    screen.blit(player, (player_x, player_y))
```

```
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

Astuces

- Choisis une image qui s'assemble bien pour éviter les coupures visibles.
- Assure-toi que la taille du motif est optimisée pour éviter des répétitions trop évidentes.
- Pour un effet plus dynamique, ajoute un léger déplacement du motif à chaque frame pour simuler un défilement.

Avec cette technique, tu peux créer facilement des arrière-plans immersifs et fluides !

Bonus

Pour les plus courageux: essaye d'afficher un mur autours de la fenêtre.

Cela peut se faire avce 3 boucles:

1. Une pour le mur du haut
2. Une pour les murs verticaux
3. Une pour le mur du bas

Empêcher le joueur de sortir de l'écran avec pygame

> *Lorsque tu déplaces un personnage à l'écran, il peut sortir des limites si aucune précaution n'est prise.*

Objectif

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

- **Limiter les déplacements** d'un personnage pour qu'il reste visible à l'écran dans un jeu pygame.
- Comprendre comment utiliser la **taille du personnage** pour gérer les limites de déplacement.

Situation initiale et problème à résoudre

Lorsque tu déplaces un personnage à l'écran, il peut sortir des limites si aucune précaution n'est prise. Dans ton code actuel, rien n'empêche le joueur d'aller au-delà des bords de l'écran. Nous allons voir comment fixer ce problème simplement et efficacement.

Points essentiels à maîtriser

1. Pourquoi limiter le déplacement ?

Si le personnage sort des limites, l'expérience du joueur est affectée. Le joueur peut se perdre ou ne plus être capable de revenir à l'écran. Limiter les déplacements permet une meilleure gestion du jeu et améliore l'expérience utilisateur.

2. Ce qu'il faut prendre en compte :

Pour empêcher efficacement ton personnage de sortir de l'écran, tu dois connaître :

- **La largeur et la hauteur de l'écran**

```
largeur_ecran = 800
hauteur_ecran = 600
```

- **La largeur et la hauteur du personnage** (obtenues à partir de l'image chargée) :

```
largeur_joueur = player.get_width()
hauteur_joueur = player.get_height()
```

3. Vérifier les limites de déplacement

Tu dois ajouter une vérification après chaque déplacement pour t'assurer que le personnage reste toujours à l'intérieur de l'écran :

```
# Limite à gauche
if player_x < 0:
    player_x = 0

# Limite à droite
if player_x > largeur_ecran - largeur_joueur:
    player_x = largeur_ecran - largeur_joueur

# Limite en haut
if player_y < 0:
    player_y = 0

# Limite en bas
if player_y > hauteur_ecran - hauteur_joueur:
    player_y = hauteur_ecran - hauteur_joueur
```

On peut aussi effectuer des calculs différemment:

```
# Limite à droite
if player_x + largeur_joueur > largeur_ecran:
    player_x = largeur_ecran - largeur_joueur

# Limite en bas
if player_y + hauteur_joueur > hauteur_ecran:
    player_y = hauteur_ecran - hauteur_joueur
```

▸ 4. Intégration complète dans ton code

Voici comment intégrer concrètement cette logique dans ton code existant :

```python
import pygame
import sys

# Initialisation
pygame.init()
largeur_ecran = 800
hauteur_ecran = 600

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
clock = pygame.time.Clock()

# Position initiale du joueur
player_x = 100
player_y = 100
vitesse = 5

# Chargement des images
player = pygame.image.load('player.png')
background_tile = pygame.image.load('sol_paves.png')

# Dimensions du joueur
largeur_joueur = player.get_width()
hauteur_joueur = player.get_height()

# Boucle de jeu
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT] or keys[pygame.K_q]:
        player_x -= vitesse
    if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
        player_x += vitesse
    if keys[pygame.K_UP] or keys[pygame.K_z]:
        player_y -= vitesse
    if keys[pygame.K_DOWN] or keys[pygame.K_s]:
        player_y += vitesse

    # Limiter les déplacements du joueur à l'écran
    player_x = max(0, min(player_x, largeur_ecran - largeur_joueur))
    player_y = max(0, min(player_y, hauteur_ecran - hauteur_joueur))

    # Mise à jour de l'écran
    for x in range(0, largeur_ecran, background_tile.get_width()):
        for y in range(0, hauteur_ecran, background_tile.get_height()):
```

```
            screen.blit(background_tile, (x, y))

    screen.blit(player, (player_x, player_y))
    pygame.display.flip()

    clock.tick(60)

pygame.quit()
sys.exit()
```

5. Bonnes pratiques à retenir

- Toujours vérifier la position du joueur après son déplacement.
- Utiliser des fonctions simples (`max` et `min`) pour clarifier ton code.
- Adapter ces limites en fonction de la taille réelle du personnage, sinon celui-ci disparaîtra partiellement ou totalement hors écran.

Synthèse à retenir :

✅ Prends en compte les dimensions du personnage pour établir des limites précises.
✅ Vérifie la position à chaque déplacement.
✅ Utilise des fonctions comme `max()` et `min()` pour simplifier ton code.

Grâce à ces notions simples, tu peux garantir que ton personnage ne quittera jamais l'écran pendant une partie !

Utiliser un joystick dans PyGame (bonus facultatif)

> *Un joystick (ou joypad, manette) peut être utilisé dans un jeu PyGame pour déplacer un personnage. Ce périphérique est souvent composé de sticks analogiques et de boutons. PyGame permet d'accéder à ces éléments via son module `joystick`.*

Initialisation du joystick

Avant de lire les mouvements du joystick, il faut l'activer dans PyGame :

```
pygame.init()
pygame.joystick.init()

# Activer le premier joystick détecté
joystick = pygame.joystick.Joystick(0)
joystick.init()
```

> ⚠️ *Ton joystick doit être branché **avant** de lancer le programme.*

Lire les mouvements du stick

La majorité des joysticks ont **deux axes analogiques** (gauche/droite et haut/bas). On peut lire ces axes avec :

```
x_axis = joystick.get_axis(0)  # Axe horizontal
y_axis = joystick.get_axis(1)  # Axe vertical
```

Chaque axe renvoie une valeur entre `-1.0` (à gauche ou en haut) et `1.0` (à droite ou en bas). Le centre correspond à `0.0`.

Appliquer le mouvement au joueur

Voici comment déplacer un joueur avec un joystick, en tenant compte d'une vitesse :

```
vitesse = 5
player_x += x_axis * vitesse
player_y += y_axis * vitesse
```

Le personnage se déplace plus ou moins vite selon l'inclinaison du stick. Cela donne un mouvement fluide et progressif.

Exemple de boucle de jeu avec joystick

```
import pygame

pygame.init()
pygame.joystick.init()
joystick = pygame.joystick.Joystick(0)
joystick.init()

screen = pygame.display.set_mode((800, 600))
player = pygame.image.load("joueur.png")
x, y = 400, 300
vitesse = 5
clock = pygame.time.Clock()
running = True

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    x_axis = joystick.get_axis(0)
    y_axis = joystick.get_axis(1)

    x += x_axis * vitesse
    y += y_axis * vitesse

    screen.fill((0, 0, 0))
    screen.blit(player, (x, y))
    pygame.display.flip()
    clock.tick(60)

pygame.quit()
```

Bonnes pratiques

- **Test de zone morte** : comme le joystick n'est jamais parfaitement stable, on peut ignorer les petites valeurs proches de 0 (zone morte).

```
if abs(x_axis) < 0.1:
    x_axis = 0
```

- **Limiter la position** pour que le personnage ne sorte pas de l'écran (voir article précédent).
- Vérifie combien de joysticks sont branchés avec `pygame.joystick.get_count()` si tu veux gérer plusieurs manettes.

Avec cette base, tu peux facilement ajouter un support joystick dans ton jeu PyGame et permettre aux joueurs de se déplacer autrement qu’au clavier.

Générer un carré d'or sur l'écran

> *Voici un article sur la **génération d'un carré d'or** à l'écran, première étape avant la détection de collisions. L'objectif est de te préparer à gérer des interactions simples entre objets dans PyGame.*

Avant d'apprendre à détecter les collisions, il faut d'abord afficher des objets avec lesquels le joueur pourra interagir. Ici, on va afficher un **carré d'or** sous forme de **rectangle jaune**. Cela permet de se concentrer sur la logique sans devoir gérer des images.

🎯 Objectif

Afficher un carré d'or à l'écran à une positions fixe ou aléatoire.

🧠 Ce que tu dois déjà savoir faire

✓ Créer une fenêtre PyGame
✓ Afficher un personnage (même en rectangle)
✓ Utiliser des coordonnées `(x, y)`

✅ Étape 1 : Un seul carré

On commence avec un carré de 20x20 pixels, couleur dorée `(255, 215, 0)` :

```
taille_carre = 20
carre_x = 300
carre_y = 200
couleur_carre = (255, 215, 0)
```

Dans la boucle de jeu, on dessine le carré :

```
pygame.draw.rect(screen, couleur_carre, (carre_x, carre_y, taille_carre, taille_carre))
```

🎲 Étape 3 : Génération aléatoire

On peut aussi générer le carré à une position aléatoire :

```python
import random

carre_x = random.randint(0, largeur_ecran - taille_carre)  # largeur_ecran - largeur du carré pou
carre_y = random.randint(0, hauteur_ecran - taille_carre)  # hauteur_ecran - hauteur du carré pou
```

On laisse une marge pour que la carré ne sorte pas de l'écran.

🧠 Bonnes pratiques

- Utilise des `pygame.Rect` dès le départ : c'est pratique pour afficher ET détecter les collisions.
- Garde la même taille pour toutes les carrés pour simplifier les calculs.

Exemple complet

Voici le code complet qui te permet de générer un carré au hasard en reprenant le code utilisé dans les exemples précédents.

```python
import pygame
import random
import sys

# Initialisation de PyGame
pygame.init()

# Définir la taille de la fenêtre
largeur_ecran = 800
hauteur_ecran = 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Pièces d'or")

# Définir la couleur et la taille des carrés (pièces)
couleur_carre = (255, 215, 0)  # Doré
taille_carre = 20

# Génération du carré
carre_x = random.randint(0, largeur_ecran - taille_carre)  # largeur_ecran - largeur du carré pour
carre_y = random.randint(0, hauteur_ecran - taille_carre)  # hauteur_ecran - hauteur du carré pour

# Boucle de jeu
clock = pygame.time.Clock()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Effacer l'écran
    screen.fill((0, 0, 0))

    # Dessiner le carré
    pygame.draw.rect(screen, couleur_carre, (carre_x, carre_y, taille_carre, taille_carre))

    # Mettre à jour l'affichage
    pygame.display.flip()
    clock.tick(60)

# Quitter proprement
pygame.quit()
sys.exit()
```

➡ SOON La suite

Dans les prochains articles, tu apprendras à :

- vérifier si le **joueur touche le carré** (`colliderect`)

- **faire disparaître** le carré touché
- **augmenter un score**

Détecter une collision simple (un joueur et un carré)

> *Commencer par la détection de collision avec **un seul carré** est parfait débuter.*

Maintenant que tu sais déplacer un personnage dans PyGame, voyons comment détecter **si le joueur touche un objet** à l'écran. On commence avec **un seul objet** : un carré doré qui représente une pièce à ramasser.

🎯 Objectif

- Comprendre comment vérifier si **le joueur touche un objet**
- Utiliser un **rectangle invisible** pour détecter la collision entre le joueur (image) et le carré (rectangle)

✅ Étape 1 : Créer un seul carré

On dessine un carré jaune à l'écran avec :

```
carre_x = 300
carre_y = 200
taille_carre = 20
couleur_carre = (255, 215, 0)

# Pour détecter une collision, on crée un rectangle :
carre_rect = pygame.Rect(carre_x, carre_y, taille_carre, taille_carre)
```

Ce rectangle représente la **zone** où la pièce est dessinée. Dans l'exemple du code précédent, on s'est contenté de dessiner un carré. Ici pour pouvoir détecter La collision avec le personnage, il faut enregistrer ce carré dans une variable pour effectuer nos tests plus tard.

✅ Étape 2 : Créer le rectangle du joueur

Même si le joueur est une **image**, on doit créer un rectangle invisible autour de lui avec :

```
player_rect = pygame.Rect(player_x, player_y, joueur_image.get_width(), joueur_image.get_height()
```

Ceci doit se faire dans la boucle. A chaque fois que le joueur a changé de position.

✅ Étape 3 : Tester la collision

On utilise la méthode `.colliderect()` pour savoir si les deux rectangles se touchent :

```
if player_rect.colliderect(carre_rect):
    print("Le joueur touche la pièce !")
```

🧪 Exemple complet avec un seul carré

Voici un petit jeu complet : un joueur (image) peut se déplacer et toucher un carré jaune. Quand c'est le cas, un message s'affiche dans la console.

```python
import pygame
import sys

# Initialisation
pygame.init()

# Fenêtre
largeur_ecran = 800
hauteur_ecran = 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Collision simple")

# Joueur (image)
joueur_image = pygame.image.load("joueur.png")
player_x = 100
player_y = 100
vitesse = 5

# Taille du joueur (automatique via l'image)
player_largeur = joueur_image.get_width()
player_hauteur = joueur_image.get_height()

# Carré doré (pièce)
carre_x = 300
carre_y = 200
taille_carre = 20
couleur_carre = (255, 215, 0)

# Créer le rectangle du carré
carre_rect = pygame.Rect(carre_x, carre_y, taille_carre, taille_carre)

# Boucle de jeu
clock = pygame.time.Clock()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Gérer les touches
    touches = pygame.key.get_pressed()
    if touches[pygame.K_LEFT]:
        player_x -= vitesse
    if touches[pygame.K_RIGHT]:
        player_x += vitesse
    if touches[pygame.K_UP]:
        player_y -= vitesse
    if touches[pygame.K_DOWN]:
        player_y += vitesse
```

```
    # Créer le rectangle du joueur (à jour à chaque boucle)
    player_rect = pygame.Rect(player_x, player_y, player_largeur, player_hauteur)

    # Vérifier la collision
    if player_rect.colliderect(carre_rect):
        print("Le joueur touche la pièce !")

    # Affichage
    screen.fill((0, 0, 0))  # fond noir

    # Dessiner le carré doré
    pygame.draw.rect(screen, couleur_carre, carre_rect)

    # Dessiner le joueur
    screen.blit(joueur_image, (player_x, player_y))

    pygame.display.flip()
    clock.tick(60)

pygame.quit()
sys.exit()
```

🧠 À retenir

- Tu peux détecter une collision entre un **rectangle et une image** grâce à `pygame.Rect()`
- La méthode `.colliderect()` renvoie `True` si deux rectangles **se touchent**
- Cette logique fonctionne même si le joueur est une **image** !

➡ SOON Prochaine étape

Tu es prêt à passer à :

- plusieurs carrés à l'écran
- supprimer un carré touché
- compter les points

Générer plusieurs carrés d’or sur l’écran

> *Voici un article sur la **génération de plusieurs carrés d’or** à l’écran.*

Tu vas d’abord afficher des objets avec lesquels le joueur pourra interagir. Ici, on va afficher des **carrés d’or** sous forme de **rectangles jaunes**. Cela permet de se concentrer sur la logique sans devoir gérer des images.

🎯 Objectif

Afficher plusieurs carrés d’or à l’écran à des positions fixes ou aléatoires, sous forme de rectangles.

🧠 Ce que tu dois déjà savoir faire

✓ Créer une fenêtre PyGame
✓ Afficher un personnage (même en rectangle)
✓ Utiliser des coordonnées `(x, y)` ✓ Afficher un carré à l'écran

🧩 Étape 2 : Plusieurs carrés à l'écran

On crée une **liste de rectangles**, chaque rectangle représentant un carré.

```
liste_carres = [
    pygame.Rect(100, 150, taille_carre, taille_carre),
    pygame.Rect(300, 220, taille_carre, taille_carre),
    pygame.Rect(500, 350, taille_carre, taille_carre)
]
```

Et dans la boucle de jeu, on les affiche tous :

```
for carre in liste_carres:
    pygame.draw.rect(screen, couleur_carre, carre)
```

Le fait de travailler avec une liste, que l'on verra plus tard, permet de travailler de façon systématique et automatique sur plusieurs objets. "En même temps".

🎲 Étape 3 : Génération aléatoire

On peut aussi générer des carrés à des positions aléatoires :

```python
import random

liste_carres = []
for _ in range(5):  # 5 carrés aléatoires
    x = random.randint(0, largeur_ecran - taille_carre)  # largeur_ecran - largeur du carré pour r
    y = random.randint(0, hauteur_ecran - taille_carre)  # hauteur_ecran - hauteur du carré pour r
    rect_carre = pygame.Rect(x, y, taille_carre, taille_carre)
    liste_carres.append(rect_carre)
```

On laisse une marge pour que la carré ne sorte pas de l'écran.

🧠 Bonnes pratiques

- Utilise des `pygame.Rect` dès le départ : c'est pratique pour afficher ET détecter les collisions.
- Garde la même taille pour toutes les carrés pour simplifier les calculs.
- Commence avec 2-3 carrés fixes pour tester avant d'ajouter du hasard.

Exemple complet

Voici le code complet qui te permet de générer les carrés au hasard en reprenant le code utilisé dans les exemples précédents.

```python
import pygame
import random
import sys

# Initialisation de PyGame
pygame.init()

# Définir la taille de la fenêtre
largeur_ecran = 800
hauteur_ecran = 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Pièces d'or")

# Définir la couleur et la taille des carrés (pièces)
couleur_carre = (255, 215, 0)  # Doré
taille_carre = 20

# Génération aléatoire de 5 carrés
liste_carres = []
for _ in range(5):
    x = random.randint(0, largeur_ecran - taille_carre)
    y = random.randint(0, hauteur_ecran - taille_carre)
    rect_carre = pygame.Rect(x, y, taille_carre, taille_carre)
    liste_carres.append(rect_carre)

# Boucle de jeu
clock = pygame.time.Clock()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Effacer l'écran
    screen.fill((0, 0, 0))

    # Dessiner tous les carrés de la liste
    for carre in liste_carres:
        pygame.draw.rect(screen, couleur_carre, carre)

    # Mettre à jour l'affichage
    pygame.display.flip()
    clock.tick(60)

# Quitter proprement
pygame.quit()
sys.exit()
```

SOON La suite

Dans le prochain article, tu apprendras à :

- vérifier si le **joueur touche un carré** (`colliderect`)
- **faire disparaître** le carré touché
- **augmenter un score**

Détecter plusieurs collisions avec des carrés

> *En repartant des exercices précédents nous allons détecter les collisions avec plusieurs carrés.*

Rappelle-toi, la méthode que nous utilisons pour détecter les collisions consiste à comparer les positions de deux rectangles entre eux:

```
if player_rect.colliderect(carre):
    # Collision détectée
```

Pour détecter les collisions avec tous les objets qu'on a créés, il suffit de passer chacun d'eux en revue à l'aide d'une boucle.

```
for carre in liste_carres:
    if player_rect.colliderect(carre):
        print("Le joueur touche une pièce !")
```

Ceci doit se faire bien sûr **dans la boucle de jeu**. Le code a été ajouté à la ligne 56 ci-dessous:

```python
import pygame
import random
import sys

# Initialisation
pygame.init()

# Fenêtre
largeur_ecran = 800
hauteur_ecran = 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Collisions multiples")

# Joueur (image)
joueur_image = pygame.image.load("joueur.png")
player_x = 100
player_y = 100
vitesse = 5
player_largeur = joueur_image.get_width()
player_hauteur = joueur_image.get_height()

# Couleur et taille des pièces
couleur_carre = (255, 215, 0)  # doré
taille_carre = 20

# Génération de plusieurs carrés aléatoires
liste_carres = []
for _ in range(5):
    x = random.randint(0, largeur_ecran - taille_carre)
    y = random.randint(0, hauteur_ecran - taille_carre)
    rect_carre = pygame.Rect(x, y, taille_carre, taille_carre)
    liste_carres.append(rect_carre)

# Boucle de jeu
clock = pygame.time.Clock()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Déplacements
    touches = pygame.key.get_pressed()
    if touches[pygame.K_LEFT]:
        player_x -= vitesse
    if touches[pygame.K_RIGHT]:
        player_x += vitesse
    if touches[pygame.K_UP]:
        player_y -= vitesse
    if touches[pygame.K_DOWN]:
```

```
        player_y += vitesse

    # Créer le rectangle du joueur à chaque boucle
    player_rect = pygame.Rect(player_x, player_y, player_largeur, player_hauteur)

    # Vérifier les collisions avec chaque carré
    for carre in liste_carres:
        if player_rect.colliderect(carre):
            print("Le joueur touche une pièce !")

    # Affichage
    screen.fill((0, 0, 0))  # fond noir

    # Dessiner les carrés
    for carre in liste_carres:
        pygame.draw.rect(screen, couleur_carre, carre)

    # Afficher le joueur
    screen.blit(joueur_image, (player_x, player_y))

    pygame.display.flip()
    clock.tick(60)

pygame.quit()
sys.exit()
```

Ajouter une musique et des sons

> *Une musique de fond permet de rendre ton jeu plus vivant et immersif. PyGame propose une méthode simple pour jouer un fichier audio pendant que la boucle de jeu tourne.*

🎯 Objectif

Savoir jouer une musique au démarrage du jeu, en boucle.

✅ Étapes à suivre

1. Préparer le fichier audio

- Utilise un fichier **au format** `.mp3` **ou** `.ogg`
- Place-le dans un soud-dossier " `musique` " là où se trouve ton script Python (par exemple : `musique.mp3`)

2. Initialiser le module audio de PyGame

```
pygame.mixer.init()
```

3. Charger et jouer la musique

```
pygame.mixer.music.load("musique.mp3")   # Charger le fichier
pygame.mixer.music.play(-1)              # Jouer en boucle (-1 = boucle infinie)
```

Tu peux aussi arrêter la musique avec :

```
pygame.mixer.music.stop()
```

🔁 Exemple minimal

```
import pygame

pygame.init()
pygame.mixer.init()

pygame.mixer.music.load("musique.mmp3")
pygame.mixer.music.play(-1)

screen = pygame.display.set_mode((800, 600))
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
```

🎵 Bonnes pratiques

- **Convertis** ton fichier si nécessaire (en `.ogg` ou `.mp3` pour plus de compatibilité)
- Lance la musique **après l'initialisation** de `pygame` et `mixer`
- Évite les musiques trop longues ou trop lourdes

Générer plusieurs pièces d'or sur l'écran

> *Voici un article sur la **génération de carrés d'or** à l'écran, première étape avant la détection de collisions. L'objectif est de te préparer à gérer des interactions simples entre objets dans PyGame.*

Cet article va te guider dans le passage des **rectangles simples** à des **images de pièces d'or** dans un jeu PyGame. Il explique les modifications nécessaires pour continuer à détecter les collisions, tout en utilisant des visuels plus réalistes.

Remplacer les carrés par des images (pièces d'or)

Jusqu'à présent, tu as appris à dessiner des pièces avec des **rectangles** simples (`pygame.draw.rect`). C'est très pratique pour apprendre, mais pour rendre ton jeu plus attrayant, tu peux les remplacer par de **vraies images de pièces** (par exemple un fichier `piece.png`).

🎯 Objectif

- Afficher des **images** de pièces à la place des rectangles
- Garder la **détection de collision** fonctionnelle
- Bien comprendre comment gérer les collisions avec des objets dessinés à partir d'images

---

📦 Étape 1 : Préparer l'image

- Ajoute une image dans ton dossier de projet, par exemple : `piece.png`
- Cette image doit être **carrée** ou de petite taille (ex. : 20×20 pixels)

🧱 Étape 2 : Charger l'image

```
piece_image = pygame.image.load("piece.png")
taille_carre = piece_image.get_width()
```

🎯 Étape 3 : Dessiner les images

Dans la boucle de jeu, on affiche chaque pièce à l'aide de :

```
for piece in liste_pieces:
    screen.blit(piece_image, (piece["x"], piece["y"]))
```

💡 Étape 4 : Détecter les collisions avec l'image du joueur

Comme la version de la détection des collisions que nous utilisons nécessite des rectangles, nous allons créer à nouveau un rectangle invisible autour de chaque pièce d'or.

Le joueur étant aussi une image, on continue d'utiliser un rectangle invisible pour vérifier les collisions :

```
player_rect = pygame.Rect(player_x, player_y, joueur_image.get_width(), joueur_image.get_height()

for x, y in liste_pieces:
    piece_rect = pygame.Rect(x, y, taille_piece, taille_piece)

    if player_rect.colliderect(piece_rect):
        print("Collision avec une pièce !")
```

Puis :

```
for piece in liste_pieces:
    if player_rect.colliderect(piece["rect"]):
        print("Le joueur a touché une pièce !")
```

🧪 Exemple complet avec images

```
import pygame
import random
import sys

pygame.init()

# Fenêtre
largeur_ecran = 800
hauteur_ecran = 600
screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Collisions avec images")

# Joueur
joueur_image = pygame.image.load("joueur.png")
player_x = 100
player_y = 100
vitesse = 5
player_largeur = joueur_image.get_width()
player_hauteur = joueur_image.get_height()

# Pièce (image)
piece_image = pygame.image.load("piece.png")
taille_carre = piece_image.get_width()

# Génération des pièces
liste_pieces = []
for _ in range(5):
    x = random.randint(0, largeur_ecran - taille_carre)
    y = random.randint(0, hauteur_ecran - taille_carre)
    liste_pieces.append((x, y))

# Boucle de jeu
clock = pygame.time.Clock()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Déplacements
    touches = pygame.key.get_pressed()
    if touches[pygame.K_LEFT]:
        player_x -= vitesse
    if touches[pygame.K_RIGHT]:
        player_x += vitesse
    if touches[pygame.K_UP]:
        player_y -= vitesse
```

```
    if touches[pygame.K_DOWN]:
        player_y += vitesse

    # Créer le rectangle invisible du joueur
    player_rect = pygame.Rect(player_x, player_y, player_largeur, player_hauteur)

    # Vérification des collisions
    for x, y in liste_positions_pieces:
        piece_rect = pygame.Rect(x, y, taille_piece, taille_piece)
        if player_rect.colliderect(piece_rect):
            print("Collision avec une pièce !")

    ############### Affichage ################
    screen.fill((0, 0, 0))  # fond noir

    # Afficher les pièces (images)
    for x, y in liste_positions_pieces:
        screen.blit(piece_image, (x, y))

    # Afficher le joueur
    screen.blit(joueur_image, (player_x, player_y))

    pygame.display.flip()
    clock.tick(60)

pygame.quit()
sys.exit()
```

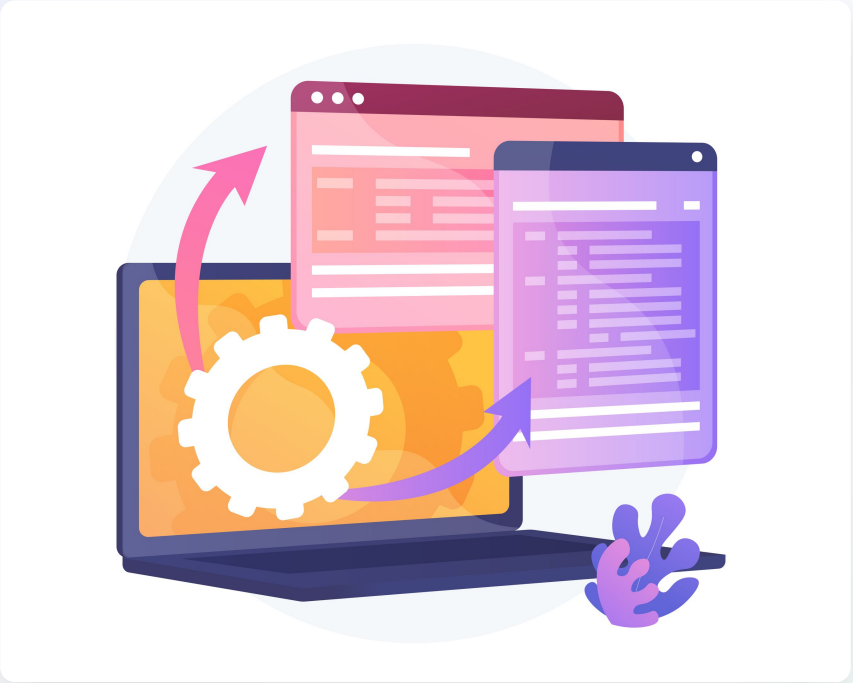
✅ Ce qu'il faut retenir

AVANT (RECTANGLE)	MAINTENANT (IMAGE)
`pygame.draw.rect( ... )`	`screen.blit(image, (x, y))`
`Rect` = l'objet affiché	`Rect` = une boîte invisible autour de l'image
Collision avec `colliderect()`	✅ fonctionne toujours avec un rectangle

➡️ Prochaine étape

Tu peux maintenant :

- faire disparaître les pièces touchées
- ajouter un score
- jouer un son lors d'une collision



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 :

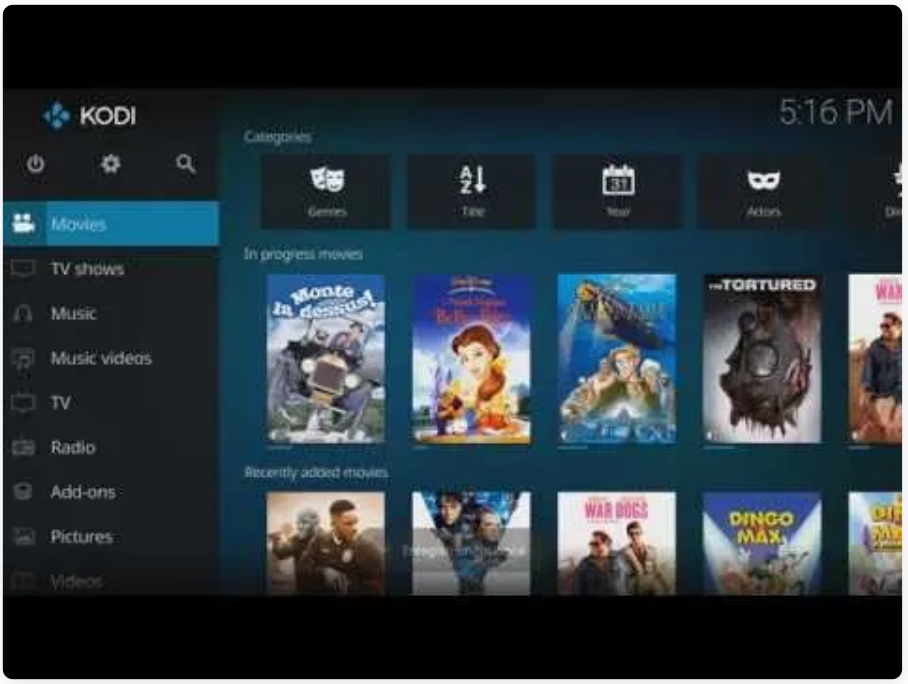
```
type ..\liste.txt
```

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).

Organisez votre vidéothèque (2025)

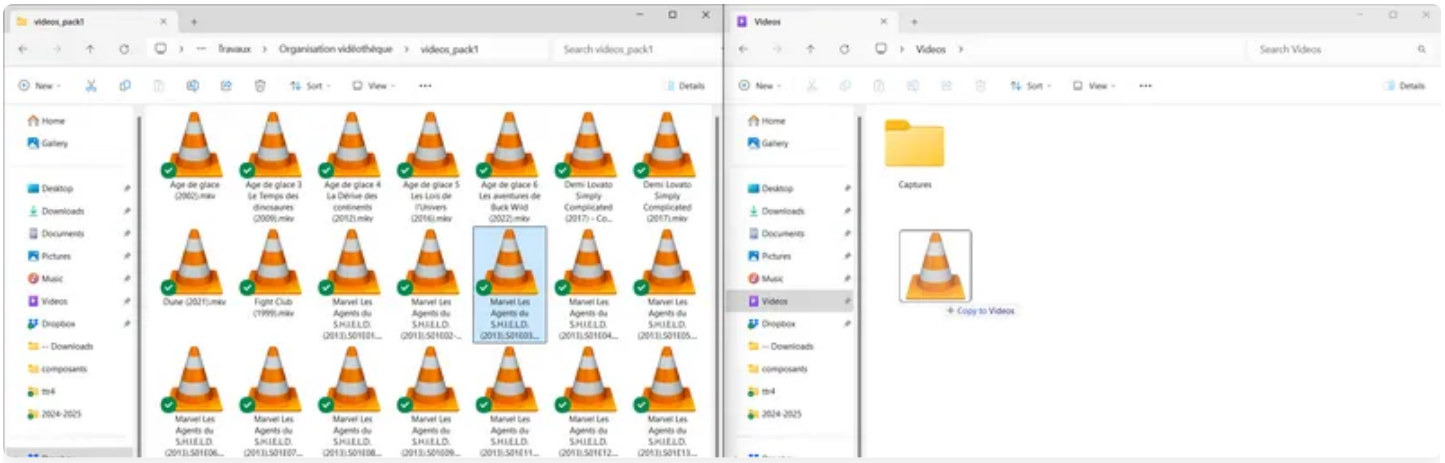
Dans cet exercice, tu vas apprendre à organiser une collection de fichiers vidéo en créant une arborescence de dossiers logique et structurée à l'aide de l'Explorateur de fichiers. Tu devras trier des fichiers, les déplacer, les renommer et les ranger de façon méthodique en suivant les consignes. Tu auras également l'occasion d'utiliser des fonctionnalités de l'Explorateur, comme le 'glisser-déposer' (drag & drop), et tu découvriras comment utiliser deux fenêtres simultanément pour travailler plus efficacement.



Objectifs d'Apprentissage

À la fin de cet exercice, tu seras capable de :

- Organiser des fichiers et des dossiers sur un disque.
- Analyser et comprendre un article technique décrivant une structure de fichiers.
- Créer des dossiers à l'aide de l'explorateur de fichiers.
- Sélectionner, copier, déplacer et renommer des dossiers et des fichiers.
- Utiliser deux fenêtres de l'explorateur de fichiers simultanément pour travailler plus efficacement.



En bonus :

- La méthode de rangement pourra t'aider à créer ton propre centre multimédia à la maison.

Instructions

Pour te permettre de suivre ta progression, utilise le `Watcher` fourni dans l'archive (voir instructions en bas de page).

1. **Télécharge ton fichier de travail sur** fichiers.ttrinfo.be
2. **Décompresse l'archive** : Une fois téléchargée, décompresse l'archive sur ton disque dans un dossier de ton choix. ⚠️ Vérifie ensuite que tu travailles dans le dossier extrait, et PAS dans l'archive.
3. **Création de l'arborescence de dossiers** : En suivant les recommandations ci-dessous, crée une arborescence de dossiers qui te permettra d'organiser ta vidéothèque. Voici quelques conseils pour la création des dossiers :
   - Tu peux créer tous les dossiers en une seule fois, puis y déplacer les fichiers.
   - Ou, tu peux les créer au fur et à mesure que tu classes tes fichiers.
   - Mélange les deux méthodes si tu préfères. Trouve la stratégie qui te convient le mieux !

Astuce: tu peux aussi y aller progressivement, niveau par niveau. Par exemple, tu peux d'abord déplacer tous les films dans le dossiers `Films` et toutes les séries dans le dossier `Séries` (niveau 1).

Puis pour les films, passe au niveau 2: crée les dossiers des sagas et déplaces-y les films correspondants. Crée les dossiers de chaque film et déplace le fichier correspondant.

Puis pour les séries, passe au niveau 2: crée un dossier par série et déplces-y tous les fichiers de la série. Ensuite, au 3e niveau, crée les dossiers des saisons et déplaces-y les fichiers correspondants.

🤓 Utilise deux fenêtres de l'Explorateur de fichiers : ce sera beaucoup plus simple pour glisser-déposer tes documents d'un endroit à l'autre.

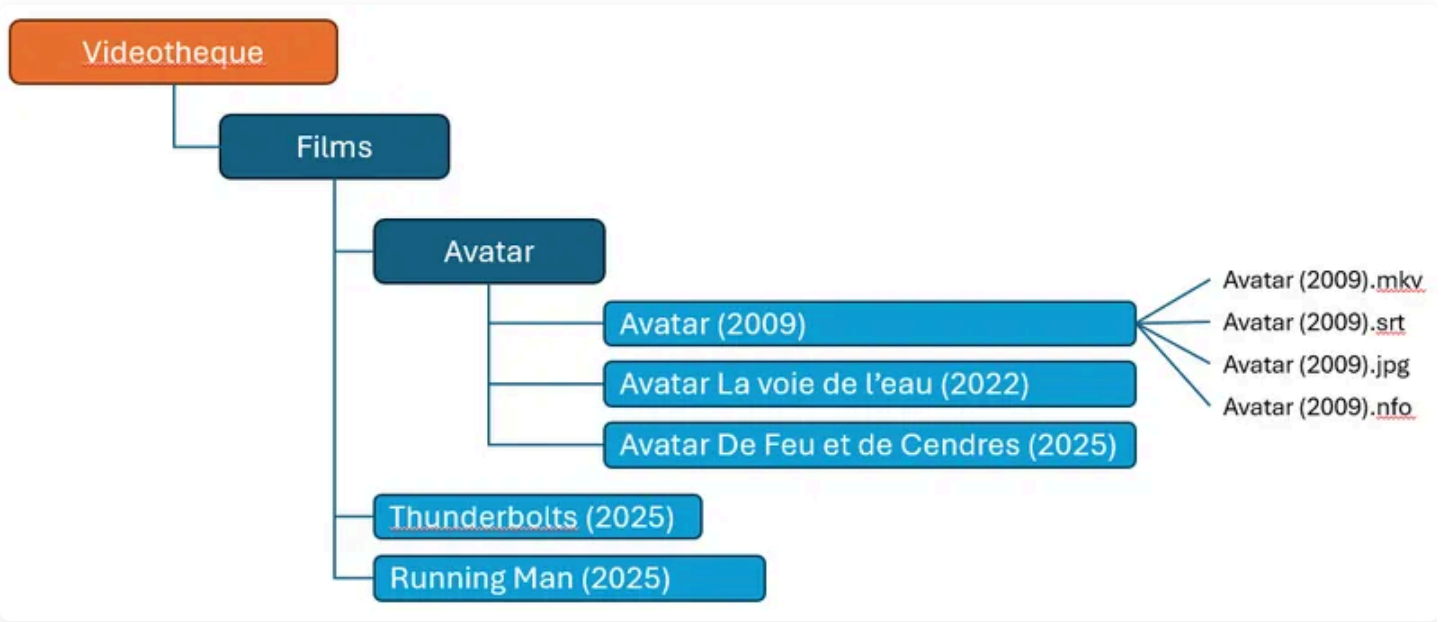
Imagine que ton ordinateur est comme ta chambre : tu as une caisse à gauche et une caisse à droite. Tu prends des objets dans la première et tu les déplaces dans la seconde. Avec deux explorateurs ouverts, c'est exactement la même chose, mais pour organiser tes fichiers.

1. **Classement des fichiers** : Utilise le "glisser-déposer" (drag & drop) pour déplacer les fichiers dans les dossiers appropriés. Pour plus de rapidité, ouvre deux fenêtres de l'Explorateur de fichiers : une pour l'arborescence de dossiers que tu crées, et l'autre pour les fichiers que tu veux trier.
2. **Besoin d'aide ?** : Consulte tes notes de cours sur l'utilisation de l'Explorateur de fichiers. N'oublie pas que ton professeur est également là pour répondre à tes questions si nécessaire.

Consignes

Ces consignes reposent sur **le bon sens et des conventions** largement acceptées dans la communauté des utilisateurs de médiathèques. L'idée principale est de séparer les différents types de médias (films, séries, concerts, etc.) dans des dossiers distincts, puis de les organiser à l'intérieur selon des critères spécifiques (par exemple, les films d'une même saga regroupés dans un dossier, les séries classées par saisons). Cette méthode te permettra non seulement de retrouver facilement tes fichiers à la main, mais aussi d'**optimiser leur reconnaissance par des logiciels comme Kodi**.

Les films



Les films doivent se trouver dans le dossier `Films`.

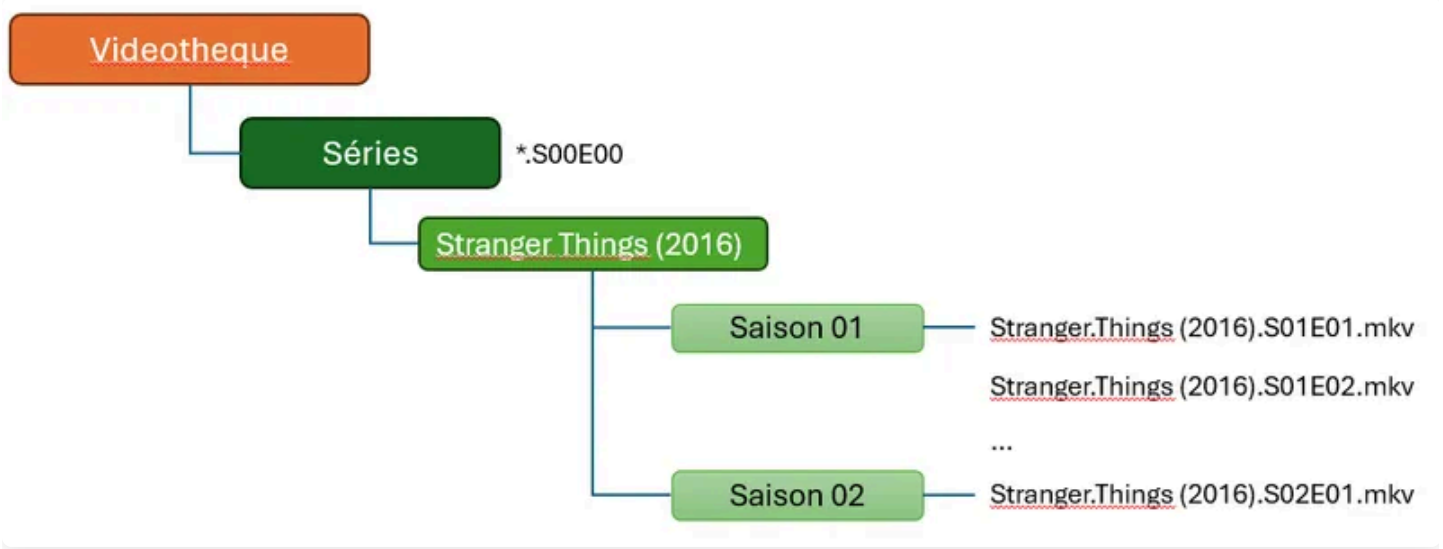
S'il n'y a qu'un seul film, place-le dans son propre dossier (nom identique à celui du film). Ex: `Thundebolts (2025).mkv` --> dossier `Thundebolts (2025)`.

S'il y a plusieurs films d'une même "série", il s'agit d'une **saga**. Dans ce cas les films doivent déplacés **dans un répertoire qui porte le nom de la saga**. Exemples de saga: Avatar, Hunger Games...

Tous les fichiers en rapport avec un même film doivent se trouver dans le même dossier.

Les séries

Tu peux identifier une série à son nom de fichier qui se termine par S00E00 --> S00 représente le numéro de saison et E00 le numéro de l'épisode dans la saison.



Chaque série doit être rangée dans son propre dossier.

Dans le dossier d'une série, il faut créer un dossier pour chaque saison, qui contient tous les fichiers des épisodes de la saison. Contrairement aux films, pas besoion de créer un dossier pour chaque épisode.

Suivi de progression

Pour suivre ta progression en temps réel, lance le programme `Watcher.exe` qui se trouve dans ton archive. Celui-ci va vérifier automatiquement si les fichiers sont à la bonne place et envoyer tes statistiques sur un serveur.

Le classement est affiché en classe.

IMPORTANT: Le classement sert à créer une petite compétition sympathique entre vous. Mais le plus important, c'est ceci : **si tu n'es pas encore en haut du classement, cela signifie simplement que tu es en train d'apprendre. Et c'est une très bonne nouvelle, parce que le but de l'activité, justement, c'est de progresser. Chaque place gagnée montre que tu avances et que tes efforts paient**. Et à la fin, ce qui compte, c'est que tu es parti·e de 0 fichiers classés. Quelle belle progression!

Et si tu as terminé l'exercice, va expliquer comment faire à quelqu'un qui a besoin d'aide!

Raccourcis claviers utiles

- `F2` : renommer un fichier
- `CTRL-SHIFT-N` : créer un nouveau dossier
- `Win- →` : déplacer la fenêtre vers la droite
- `Win- ←` : déplacer la fenêtre vers la gauche

Séquence utile pour créer le dossier d'un film/d'une série:

- `F2` (renomme)
- `CTRl-C` (copier le nom sélectionné, sans l'extension)
- `ESC` ou `ECHAP` ou `ENTRÉE` (qui le mode sélection)
- `CTRL-SHIFT-N` (créer un nouveau dossier)
- `CTRL-V` coller le nom du fichier
- `ENTRÉE` valider la création du dossier

Les outils Windows

Introduction à la ligne de commande Windows

> *La ligne de commande (ou **cmd**) est une fenêtre où tu peux écrire des instructions directement à l'ordinateur.*

Objectifs

- Comprendre ce qu'est la ligne de commande.
- Savoir lire et interpréter l'**invite de commande** (prompt).
- Apprendre à naviguer dans les dossiers avec quelques commandes de base.
- Afficher le contenu d'un fichier texte simple.

La ligne de commande et l'invite (prompt)

La ligne de commande (ou **cmd**) est une fenêtre où tu peux écrire des instructions directement à l'ordinateur.

Chaque fois que tu ouvres l'invite de commande, tu vois apparaître une ligne qui ressemble à ceci :

```
C:\Utilisateurs\Adrien>
```

Cette ligne est appelée le **prompt**.

Elle te donne deux informations importantes :

- **Le lecteur courant** : ici `C:` → ton disque principal.
- **Le répertoire courant** : ici `\Utilisateurs\Adrien` → le dossier dans lequel tu te trouves actuellement.

Tout ce que tu tapes ensuite (commandes comme `dir`, `cd`, etc.) va s'exécuter **dans/à partir de ce dossier courant**.

On peut comparer cela à un GPS : le prompt t'indique ta « position actuelle » dans l'arborescence des dossiers.

Navigation dans les dossiers

Un ordinateur organise ses fichiers dans une **arborescence de dossiers** :

- Le **répertoire parent** est le dossier qui contient d'autres dossiers.
- Les **répertoires enfants** sont les sous-dossiers qu'il contient.

Exemple :

```
C:\
 └── Utilisateurs
      └── Adrien
           └── Documents
```

Si ton prompt indique :

```
C:\Utilisateurs\Adrien>
```

- Le **parent** est `C:\Utilisateurs`.
- Un **enfant** est `Documents`.

Les commandes de base

- `dir`

Affiche la liste des fichiers et sous-dossiers présents dans le dossier courant.

Exemple :

```
C:\Utilisateurs\Adrien> dir
```

- `cd [nom_dossier]`

Permet d'entrer dans un **répertoire enfant**.

```
C:\Utilisateurs\Adrien> cd Documents
C:\Utilisateurs\Adrien\Documents>
```

Remarque que le dossier courant dans le *prompt* a été mis à jour (`C:\Utilisateurs\Adrien` --> `C:\Utilisateurs\Adrien\Documents`).

- `cd ..`

Permet de remonter d'un cran, dans le **répertoire parent**.

```
C:\Utilisateurs\Adrien\Documents> cd ..
C:\Utilisateurs\Adrien>
```

Remarque que le dossier courant dans le *prompt* a été mis à jour (`C:\Utilisateurs\Adrien\Documents` --> `C:\Utilisateurs\Adrien`).

- `type [nom_fichier]`

Affiche le contenu d'un fichier **texte** directement dans la console.

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

Cette commande n'a vraiment d'intérêt que pour les fichiers "textes". Les fichiers Word, PowerPoint, images, musiques... ne s'afficheront pas correctement avec `type`.

Ce qu'il faut retenir

- Le **prompt** t'indique toujours où tu te trouves (`lecteur` et `répertoire courant`).
- `dir` → liste le contenu du dossier.
- `cd nom_dossier` → descendre dans un sous-dossier.
- `cd ..` → remonter au dossier parent.
- `type fichier.txt` → lire un fichier **texte**.

Veux-tu que je prépare maintenant une **fiche d'exercices guidés** (progressive, étape par étape) pour que tes élèves manipulent directement le prompt et ces 4 commandes ?



TECHNOLOGIES

© 2025 Mr. Lambrechts

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.

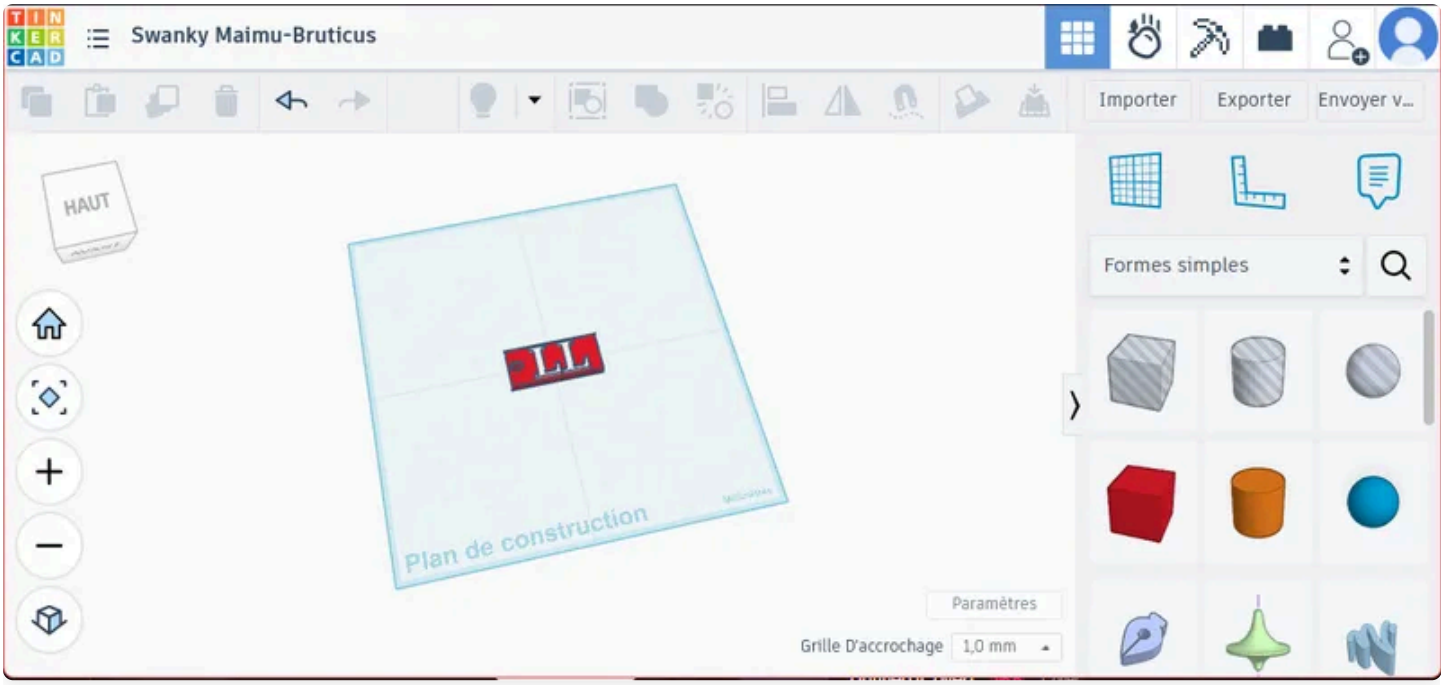
🎯 Objectifs

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 Tinker-Cad correspond à la zone où l'imprimante peut déposer du plastique. Tout ce qui dépasse du plateau ne pourra pas être imprimé.*

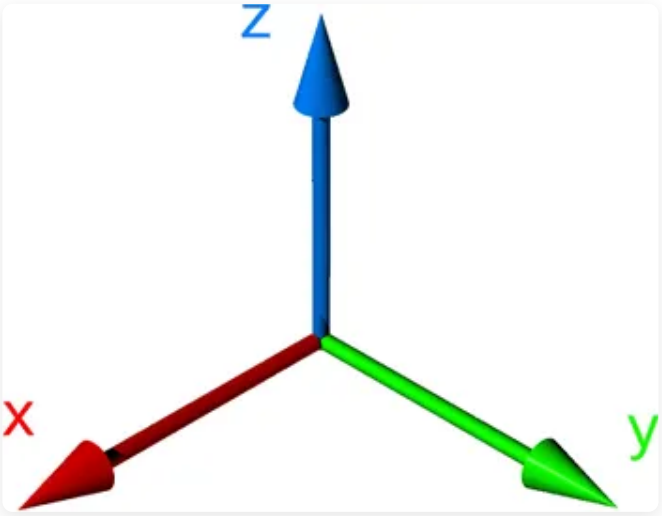
▸ Bouger la vue avec la souris

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.

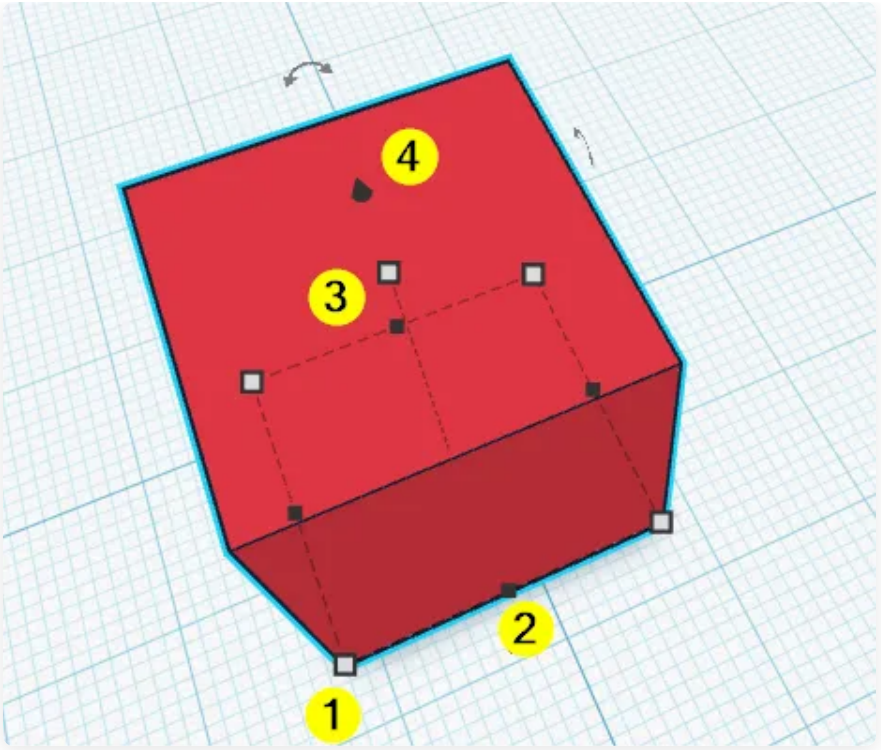
Partie 2 — Ajouter une première forme

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

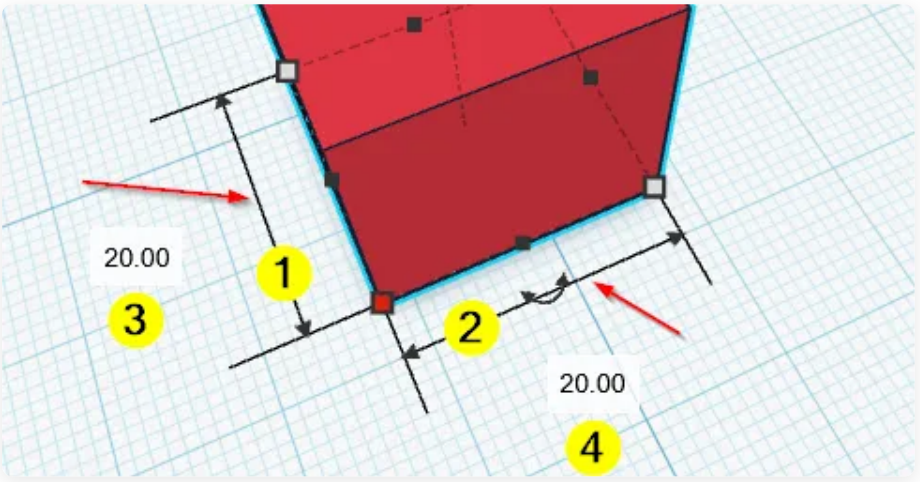
Les poignées blanches [1] et noires sur les côtés [2] étirent la forme dans une ou deux directions. Les poignées blanches **en coin** (celles du bas) permettent de redimensionner les 2 côtés qu'elles touchent en même temps. Les poignées noires permettent de changer la dimension sur un seul axe. Appuie 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).

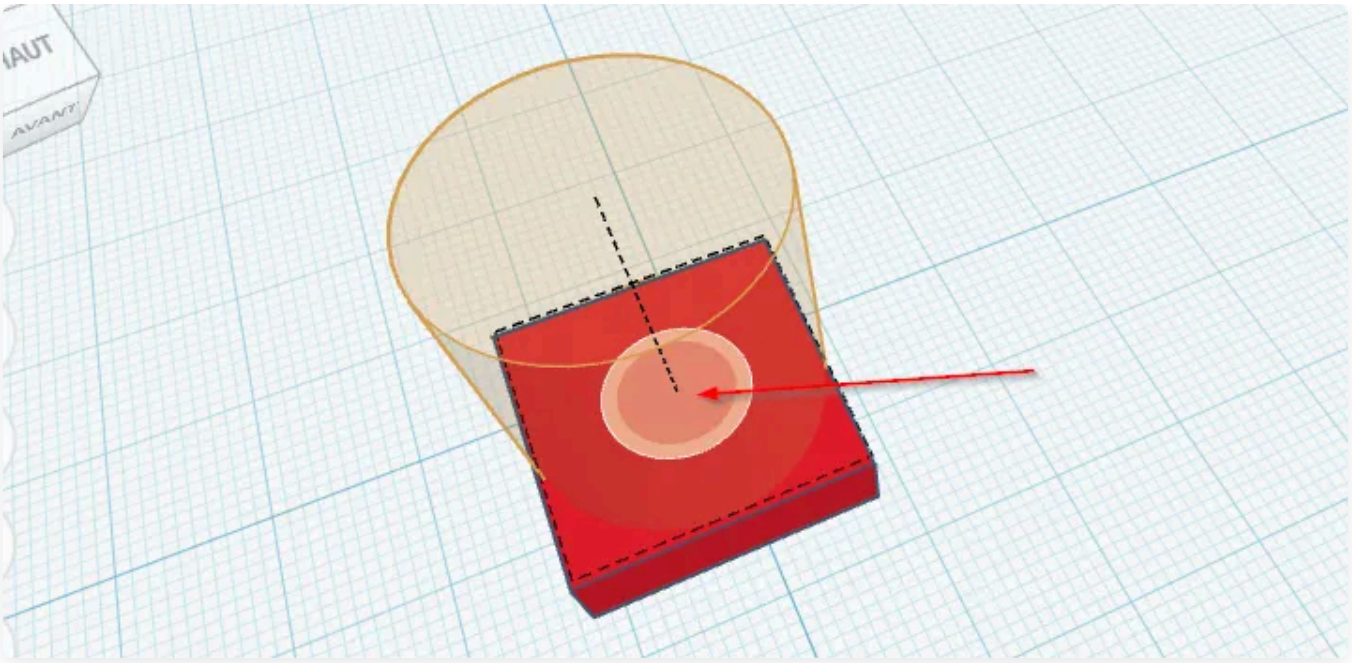


> 💡 ***Astuce :*** *tu peux toujours cliquer directement sur les chiffres affichés sous la ligne de mesure [3+4] pour entrer une valeur précise au clavier. C'est plus rapide et plus précis que de tirer à la souris.*

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** :

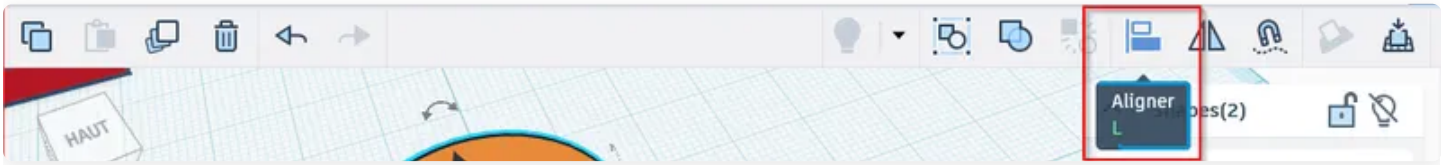
1. Sélectionne le texte.
2. Saisis le **cône noir** au-dessus de la forme (la flèche vers le haut, séparée du corps).
3. Tire-le vers le haut jusqu'à une hauteur Z de **4 mm**. Tu peux aussi cliquer sur la valeur affichée et la taper directement au clavier.

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

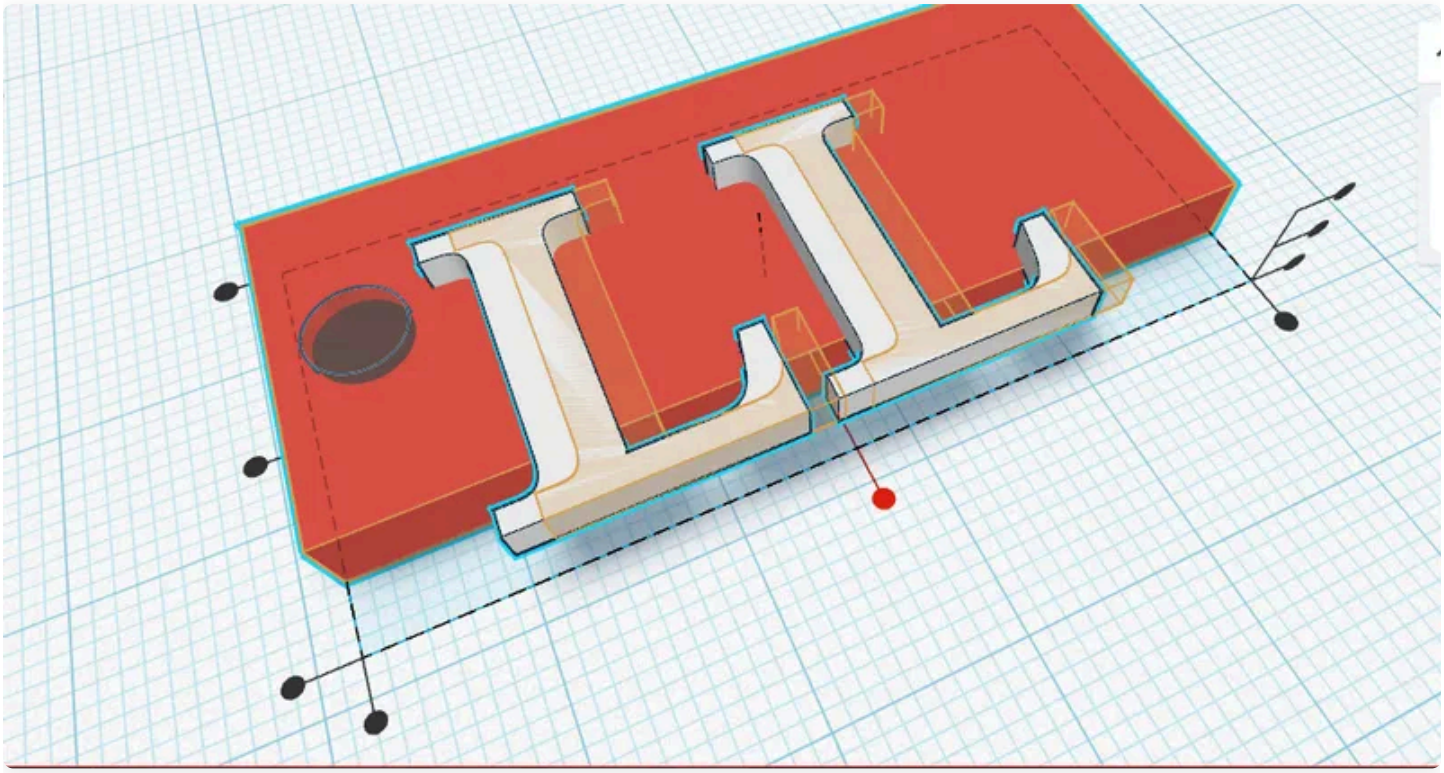
Partie 4 — Aligner deux objets

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.

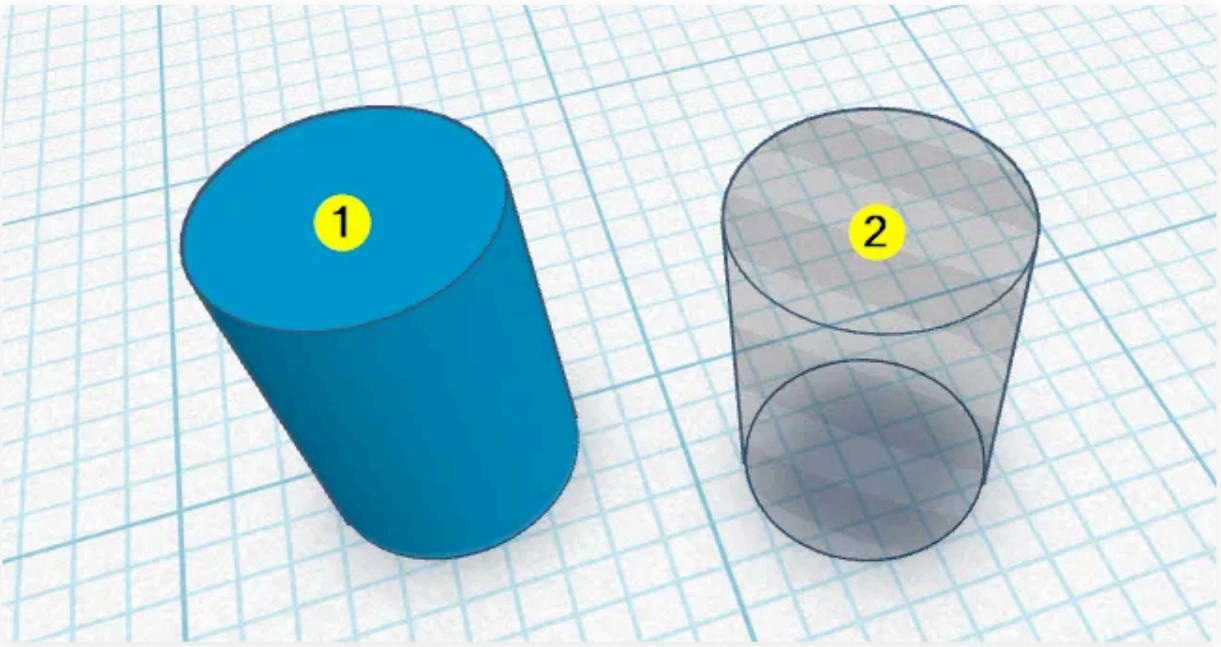
Le texte est maintenant parfaitement centré sur la plaque. Appuie sur **Échap** ou clique ailleurs pour sortir du mode Aligner.

> 🧠 ***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.

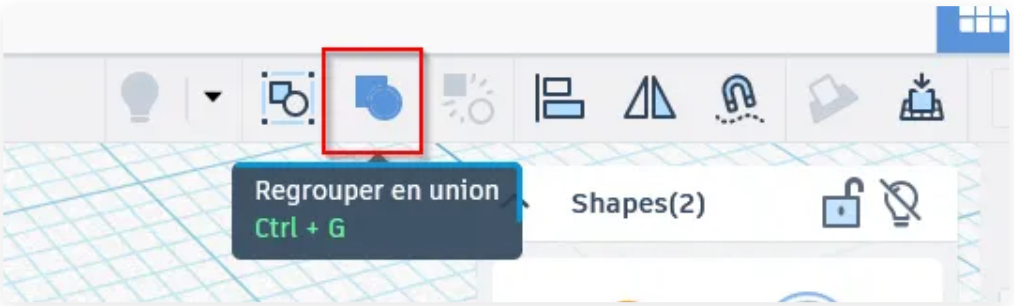
5. Vérifie qu'il **traverse bien** la plaque de haut en bas (la base du cylindre doit être au niveau du sol, ou même en dessous).

▸ 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.

> ℹ️ ***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***).

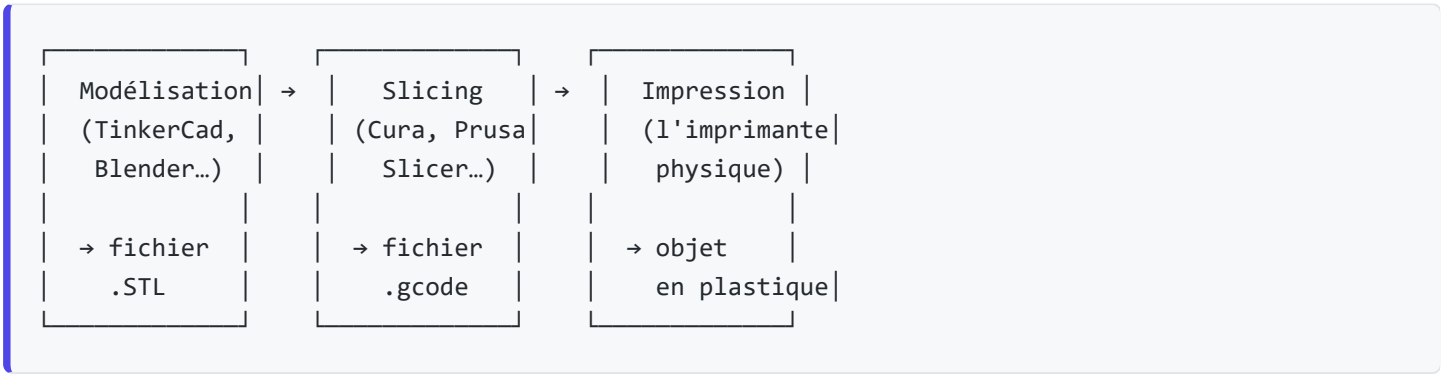
```
Couche 30   ▬▬▬▬▬▬▬▬▬▬▬▬     ← dernière déposée
Couche 29   ▬▬▬▬▬▬▬▬▬▬▬▬
Couche 28   ▬▬▬▬▬▬▬▬▬▬▬▬▬▬
 ...
Couche 03   ▬▬▬▬▬▬▬▬▬▬▬▬▬▬
Couche 02   ▬▬▬▬▬▬▬▬▬▬▬▬
Couche 01   ▬▬▬▬▬▬▬▬▬▬▬▬     ← première déposée sur le plateau
            ══════════════
               plateau
```

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.*

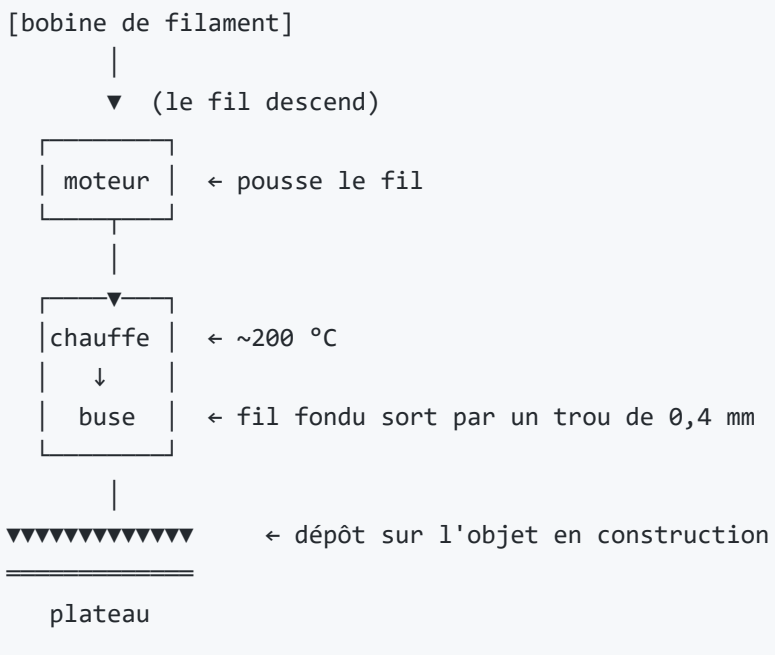
Partie 3 — Les types d'imprimantes 3D grand public

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

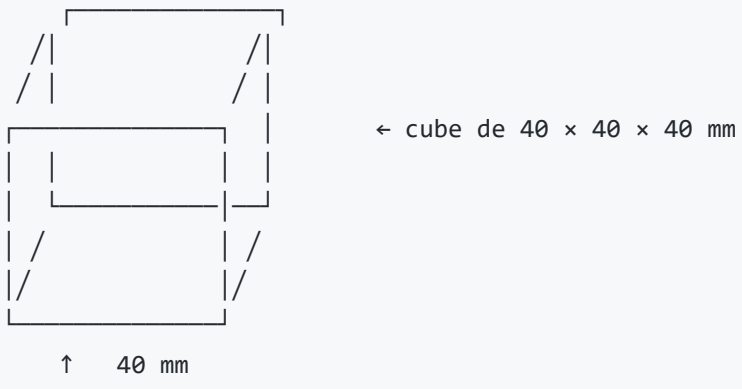
À la fin de ce projet, tu seras capable de :

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.