Table des matières

BUREAUTIQUE

PowerPoint

Le fichier se trouve en pièce jointe.

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)

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!**

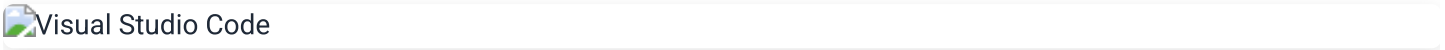
Vidéos "Python au lycée"

- 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

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.

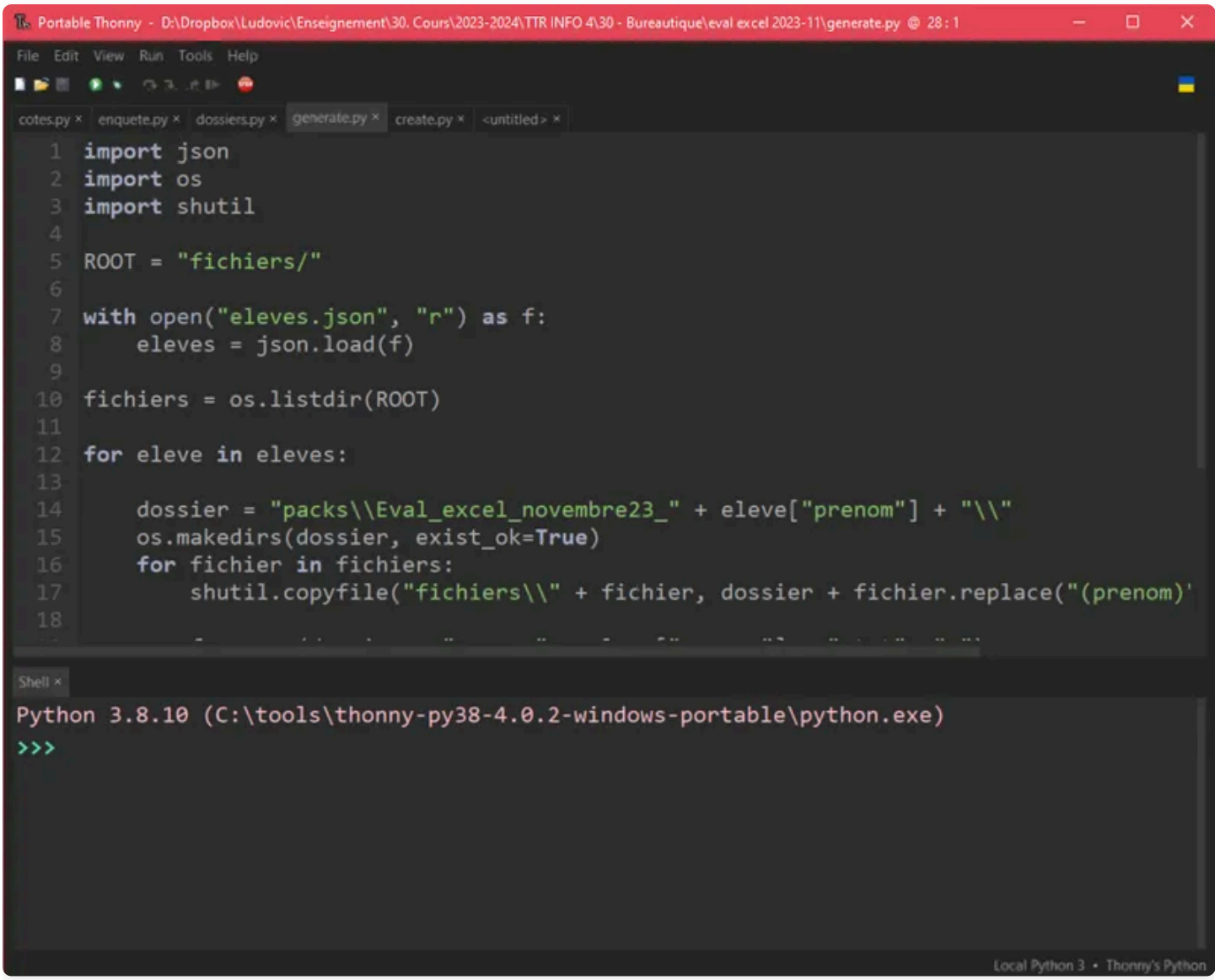
Configuration de Visual Studio Code pour Python

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

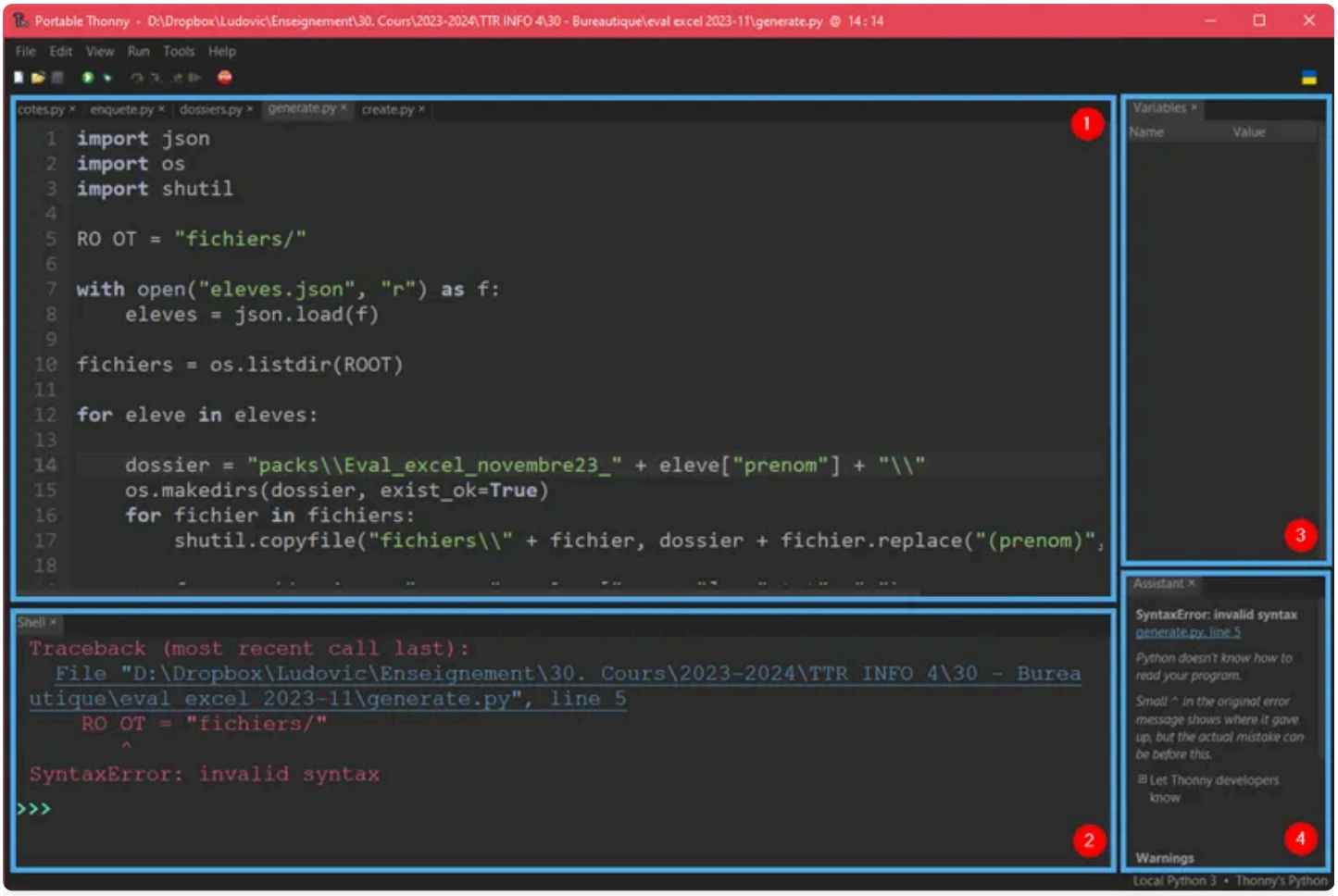
Thonny est livré avec un gestionnaire de packages/modules intégré qui facilite l'installation de modules Python supplémentaires directement depuis l'IDE, ce qui vous permet d'étendre les fonctionnalités de Python selon vos besoins.

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

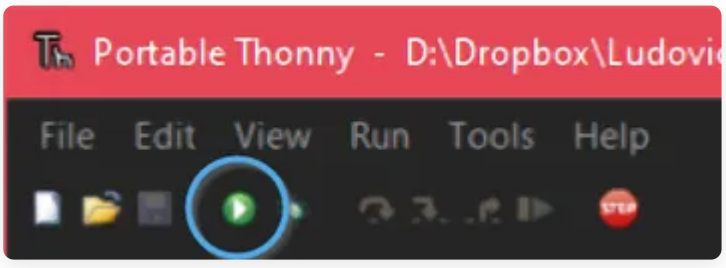
Téléchargez et installez Thonny à partir du site officiel de Thonny. Une fois l'installation terminée, lancez Thonny depuis votre menu d'applications.

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

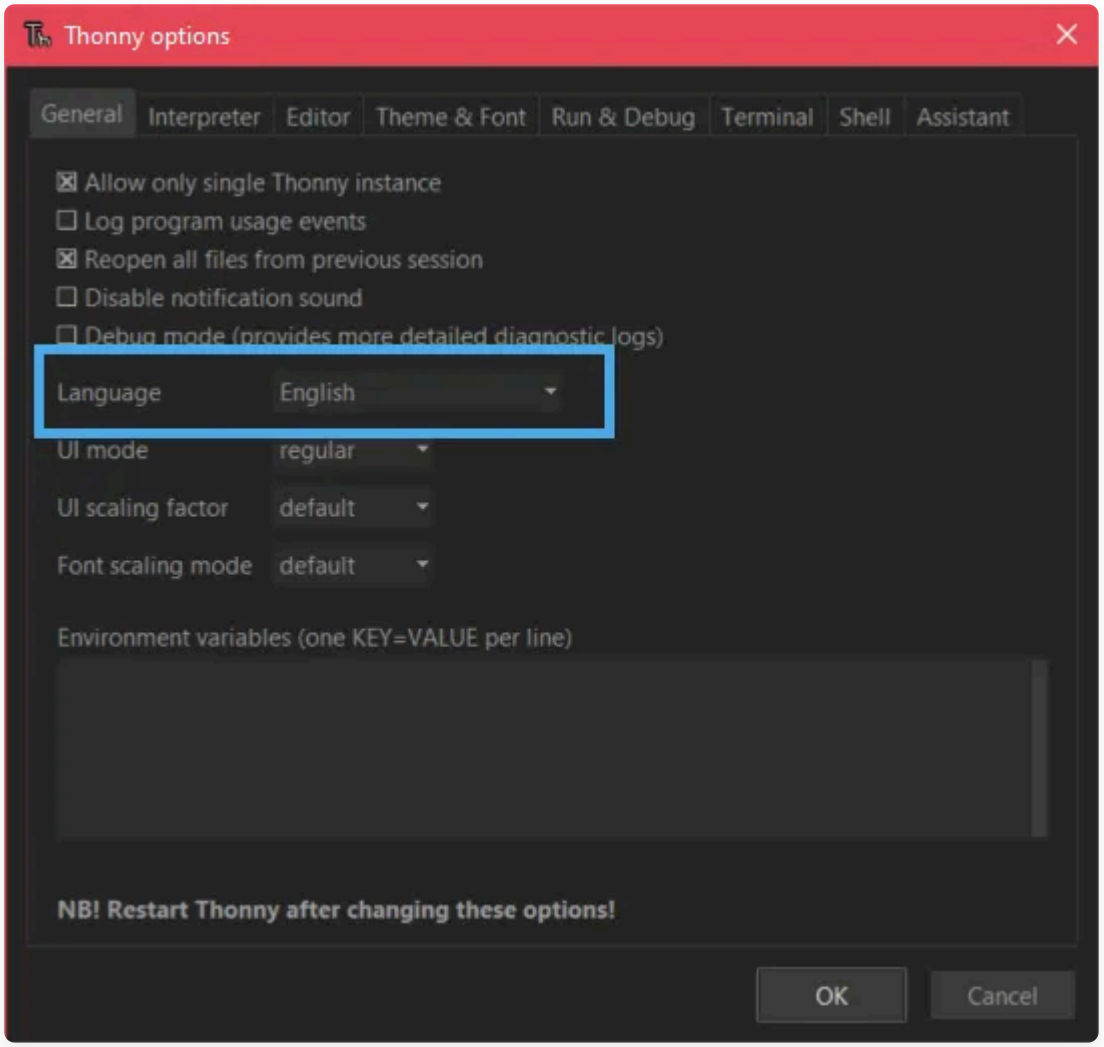
Si vous avez besoin de fonctionnalités supplémentaires, utilisez le gestionnaire de packages intégré pour installer les modules Python nécessaires à votre projet.

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.

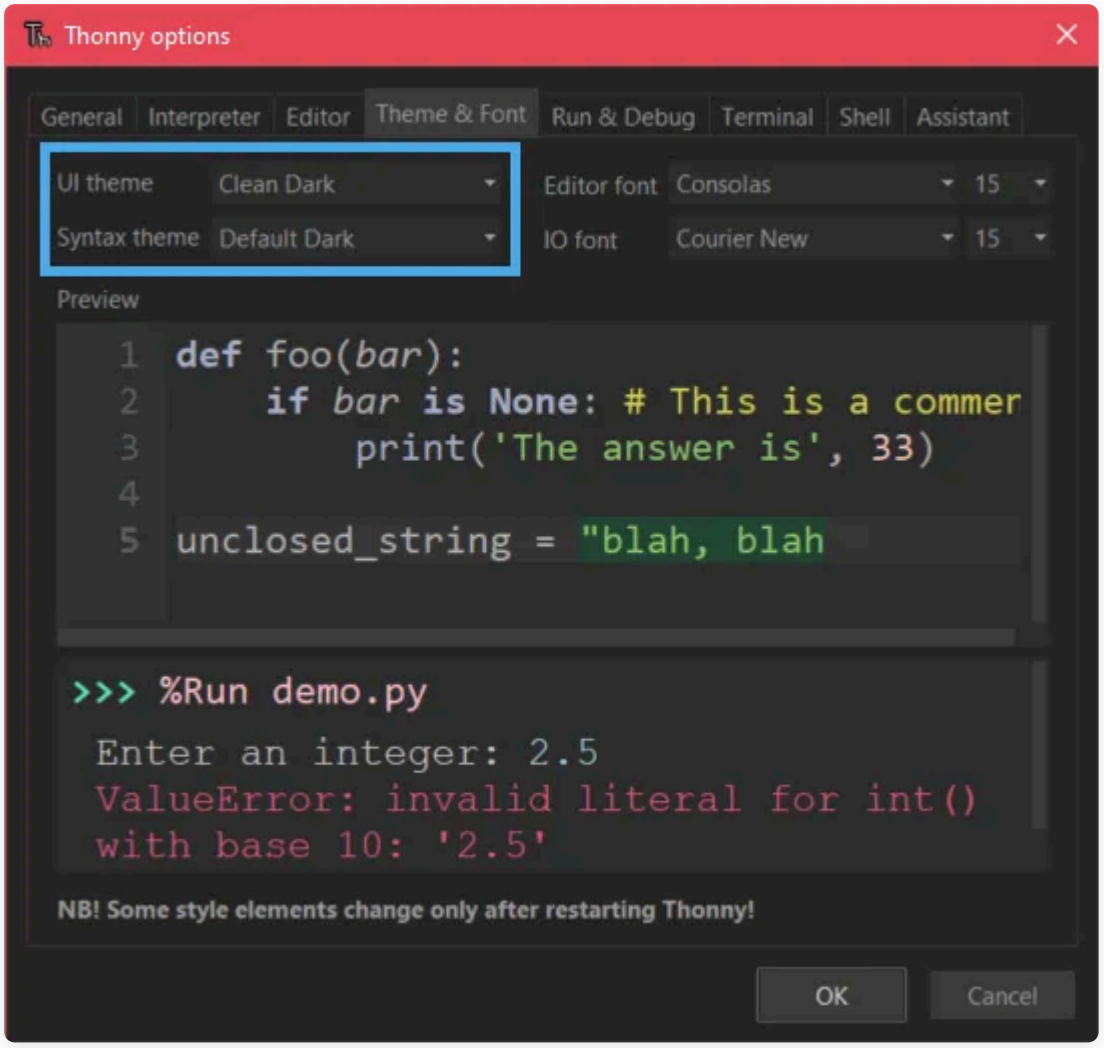
Change le thème d'affichage

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.

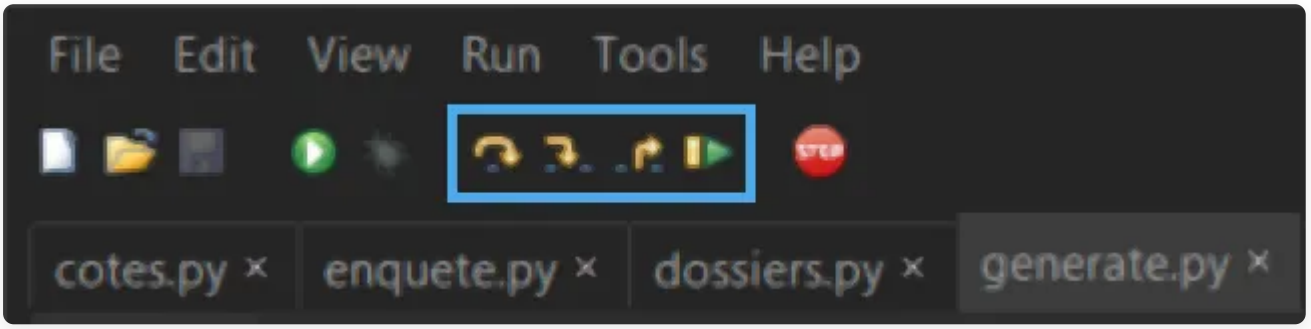
Thonny comprend un débogueur intégré qui permet aux développeurs d'exécuter **leur code pas à pas**, autrement dit **ligne par ligne**, d'**observer les valeurs des variables** à chaque étape, et de **mettre en pause l'exécution du programme** pour inspecter l'état du code.

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 programme 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:

```
7  with open("eleves.json", "r") as f:
8      eleves = json.load(f)
9
10 fichiers = os.listdir(ROOT)
11
12 for eleve in eleves:
13
14     dossier = "packs\\Eval_excel_novembre23_" + eleve["prenom"] + "\\"
15     os.makedirs(dossier, exist_ok=True)
```

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

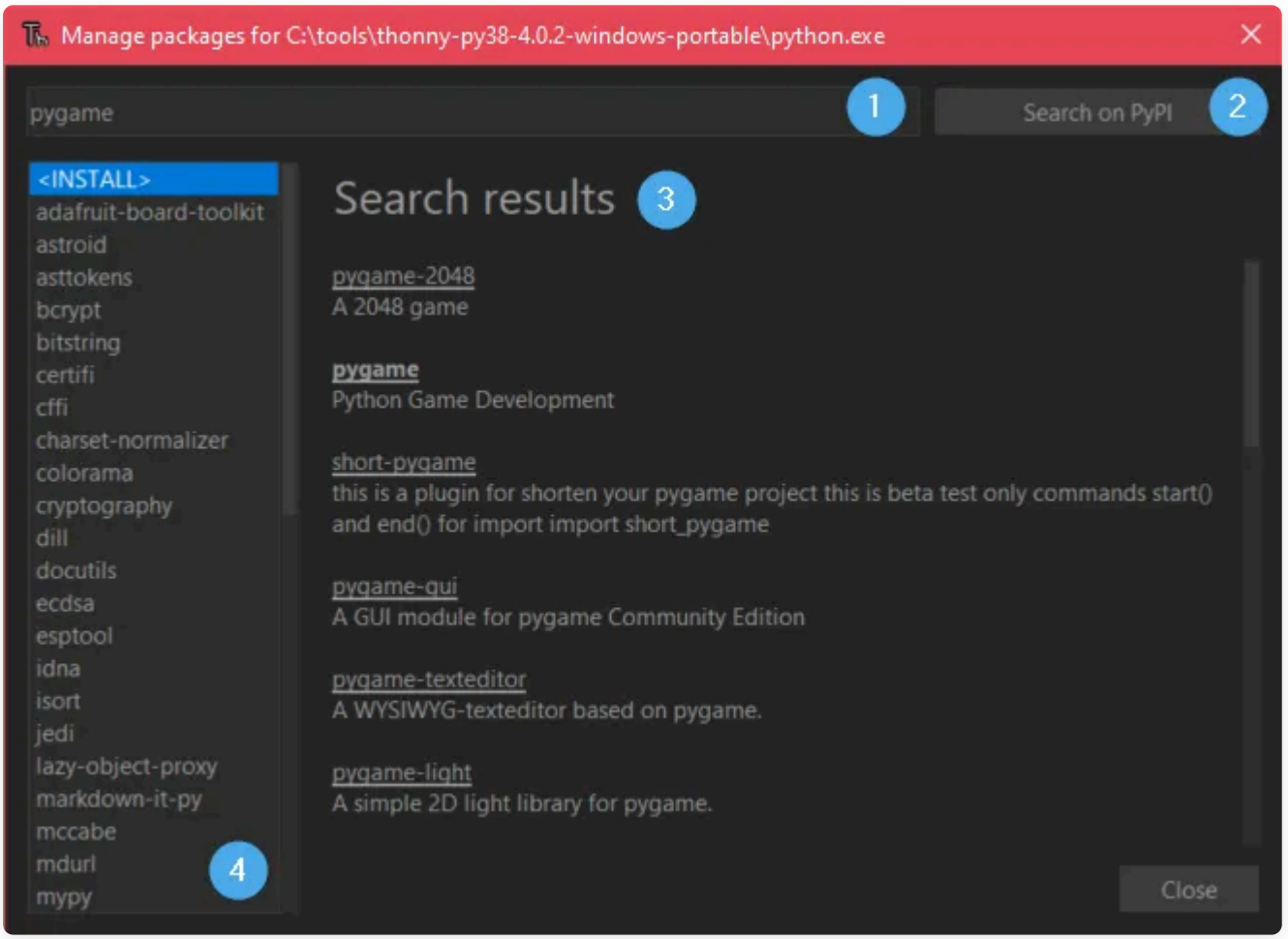
Le débogage de code est une compétence essentielle pour tout développeur informatique. En comprenant les concepts de base du débogage et en utilisant les outils appropriés, les développeurs peuvent identifier et résoudre efficacement les erreurs dans leur code, ce qui contribue à la création de logiciels 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.

Thonny Python - Installer des modules

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

Accéder à la gestion des packages

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

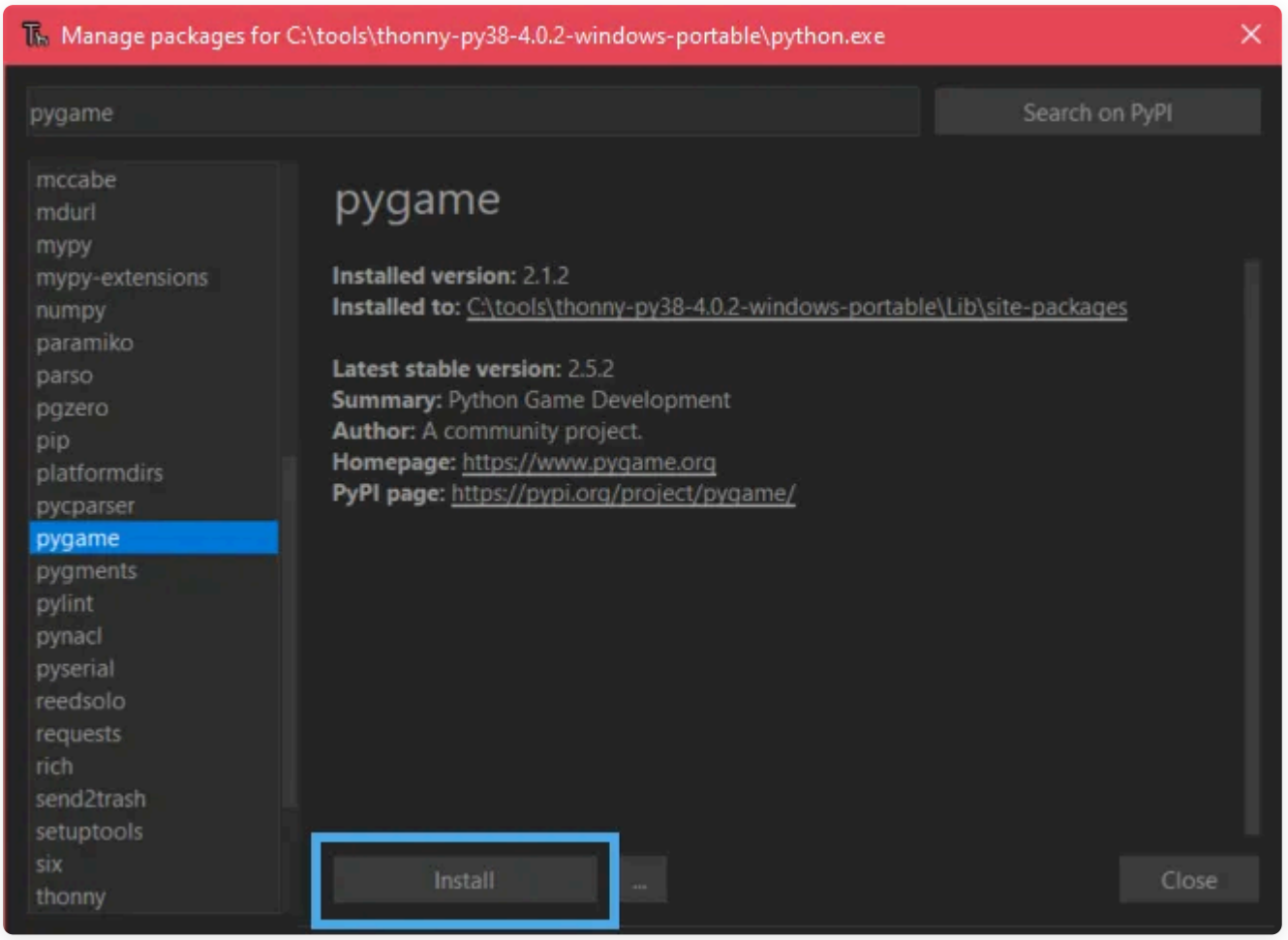


Étape 3: Rechercher et installer un module Python

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

`[2]` pour valider votre recherche

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



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

Étape 4: Vérifier l'installation du module

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

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

```
import pygame
```

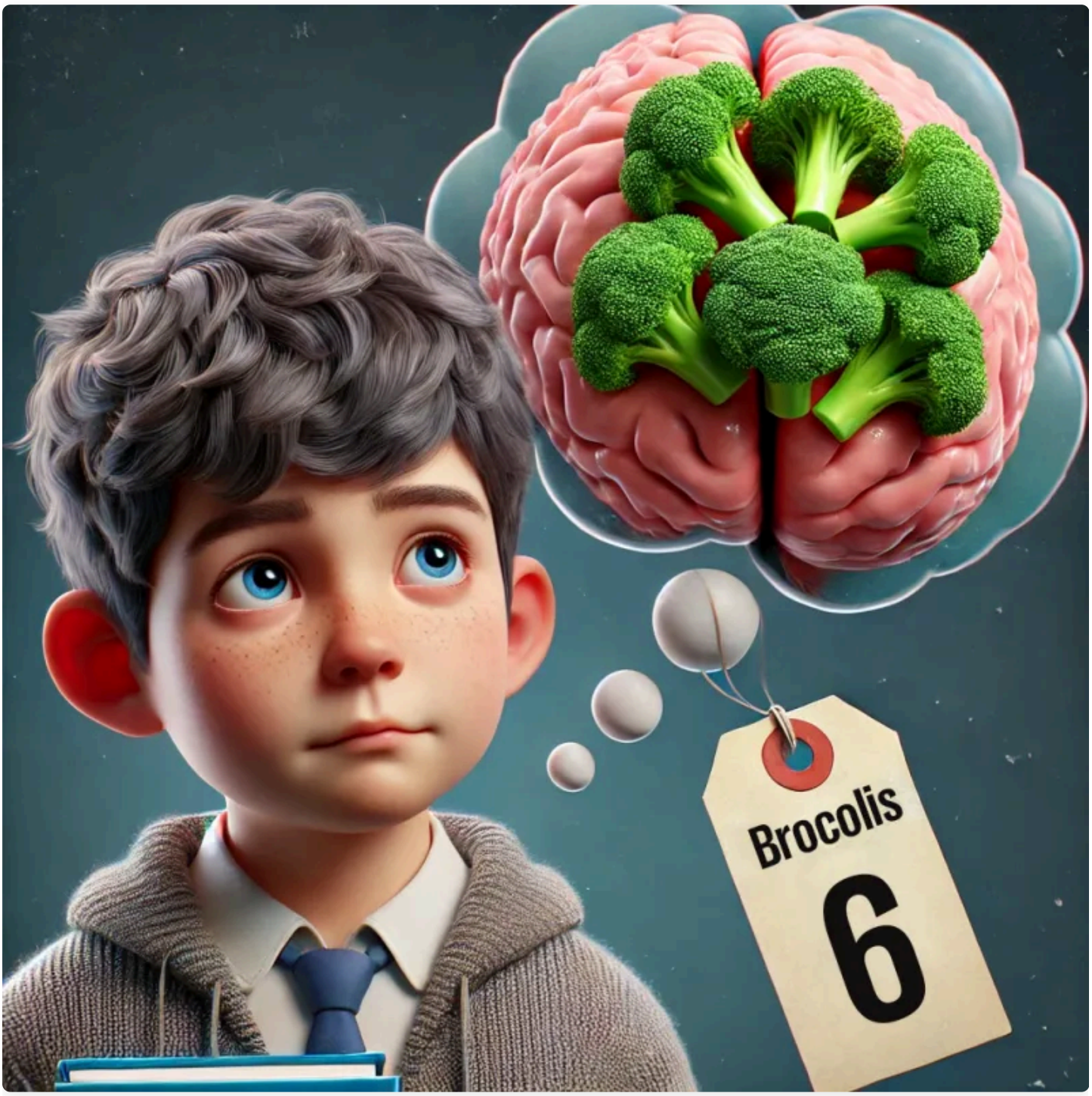
Si vous ne voyez pas d'erreurs, cela signifie que le module a été installé avec succès et que vous pouvez commencer à l'utiliser dans vos projets Python.

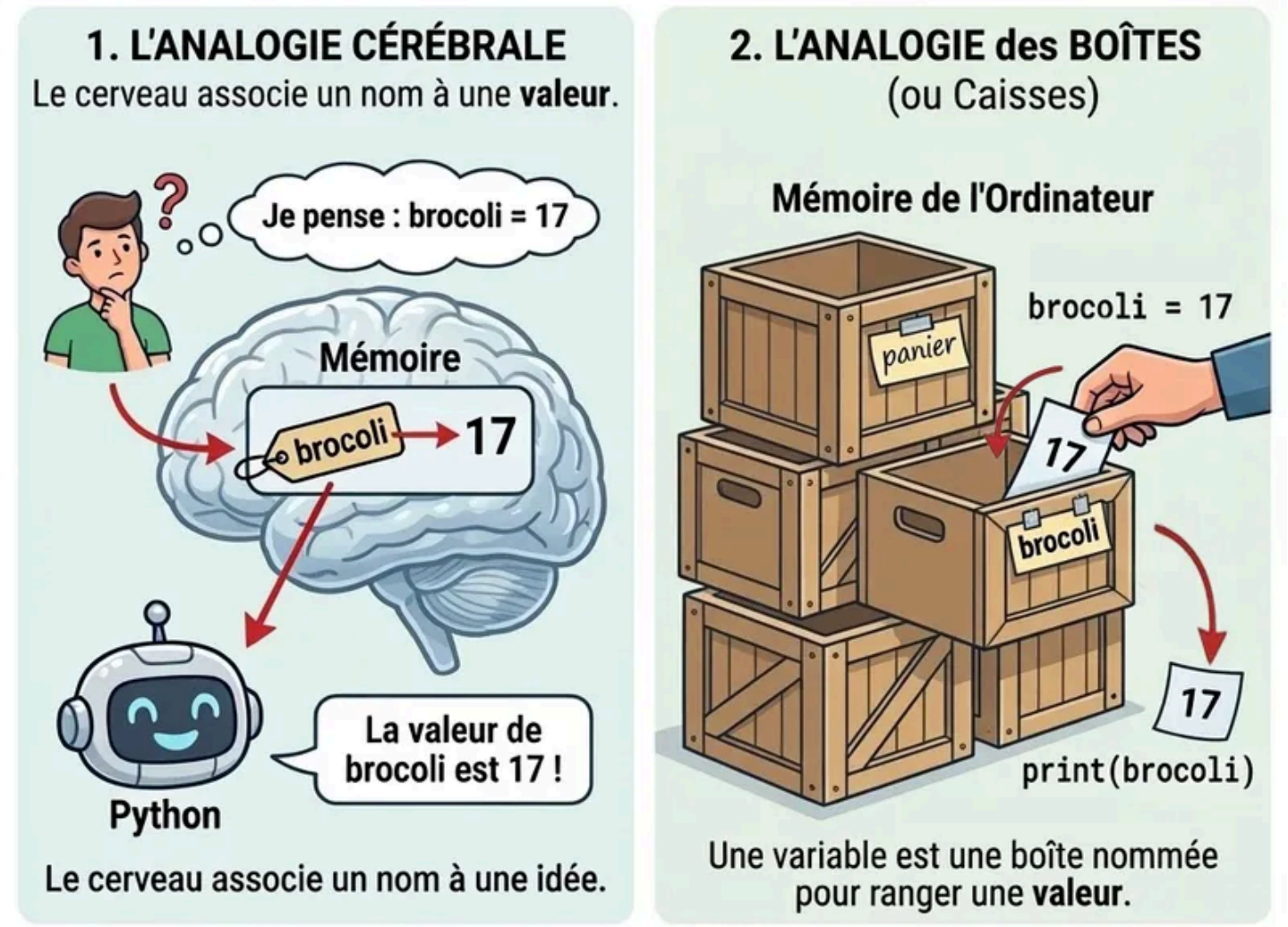
Conclusion

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

Les variables en Python

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





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

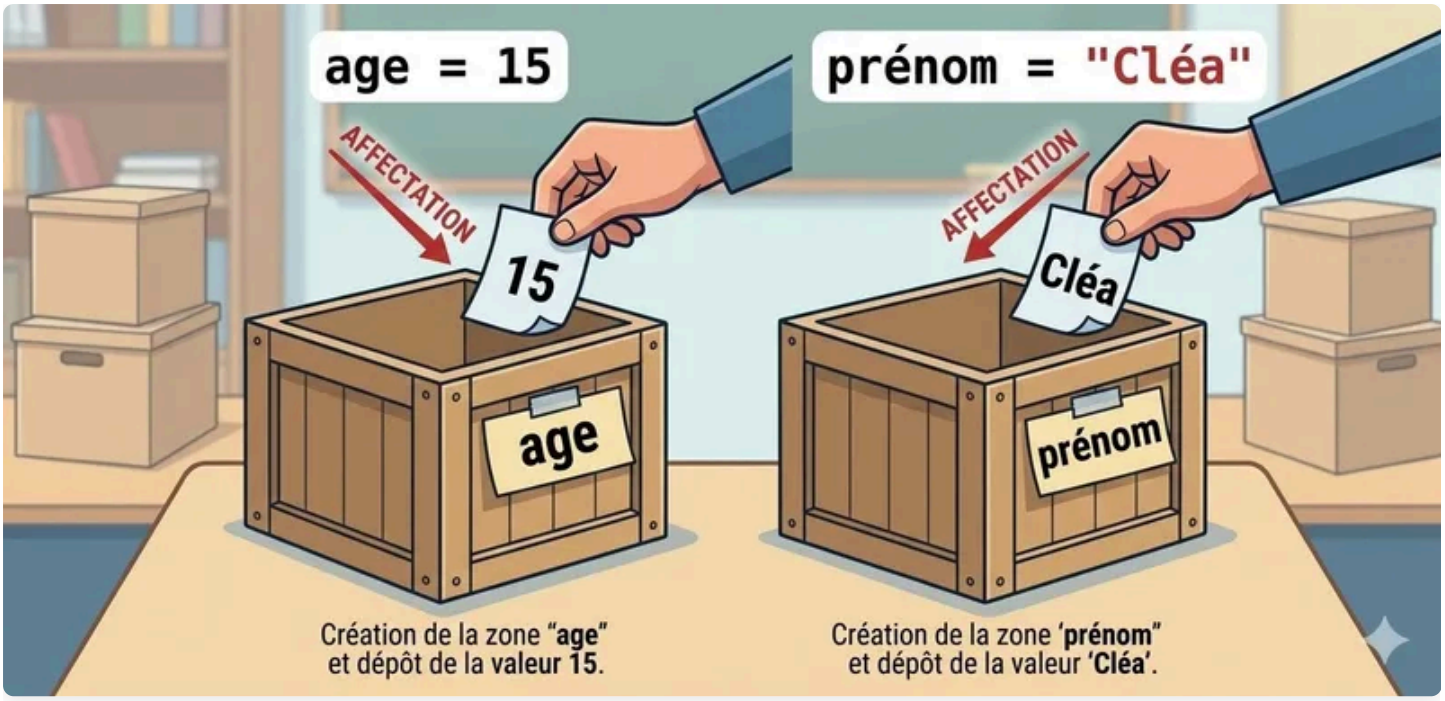
▸ Premier exemple

Prenons un exemple pour illustrer ce concept en Python :

```
nom = "Cléa"
age = 15
```

Dans cet exemple :

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

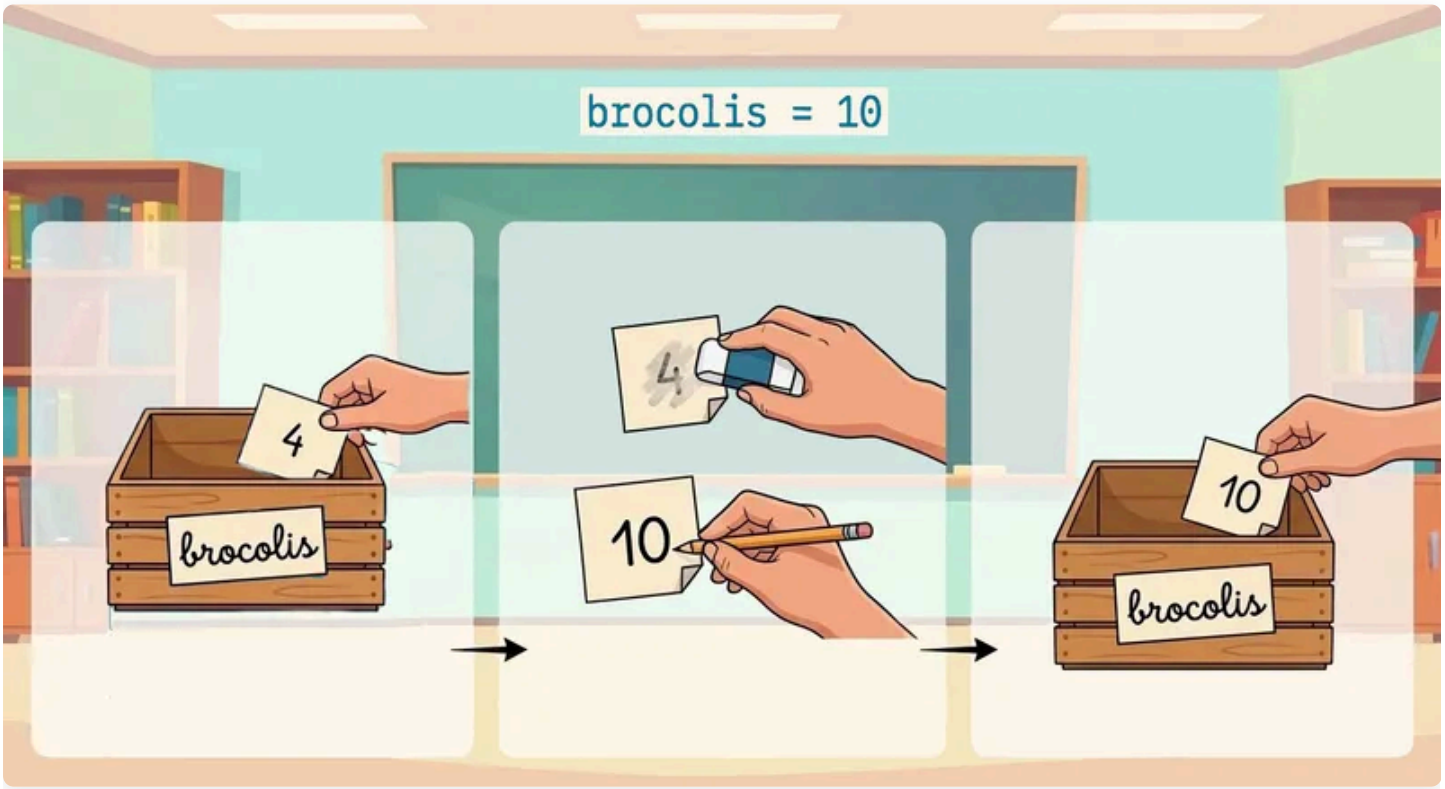


Ensuite, on pourra récupérer ces valeurs pour les afficher ou les utiliser dans un calcul par exemple.

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

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

```
brocolis = 10
```



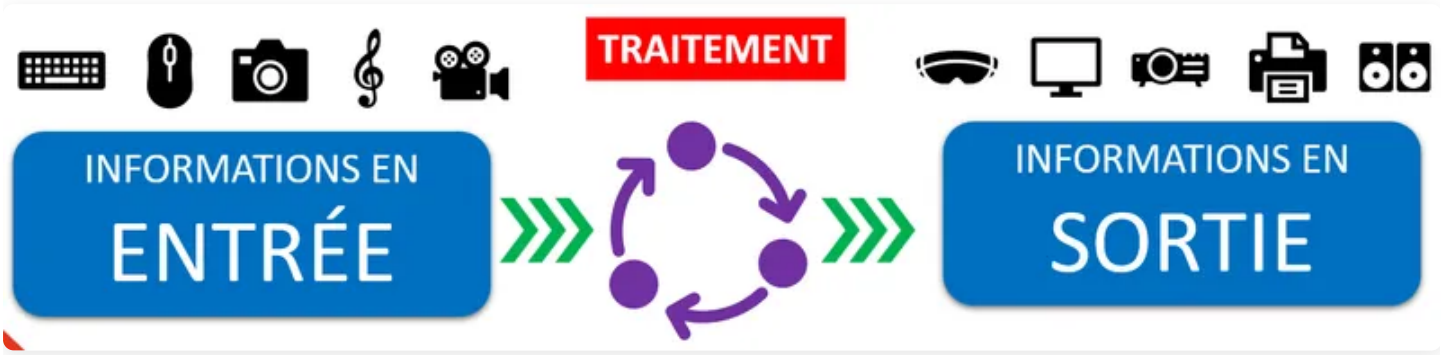
Importance des variables

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

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

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

Souviens-toi de ce schéma:



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

Les données sont stockées dans des variables.

Déclarer ses Variables

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

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

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

```
age = 25
```

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

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

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

>>>
```

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

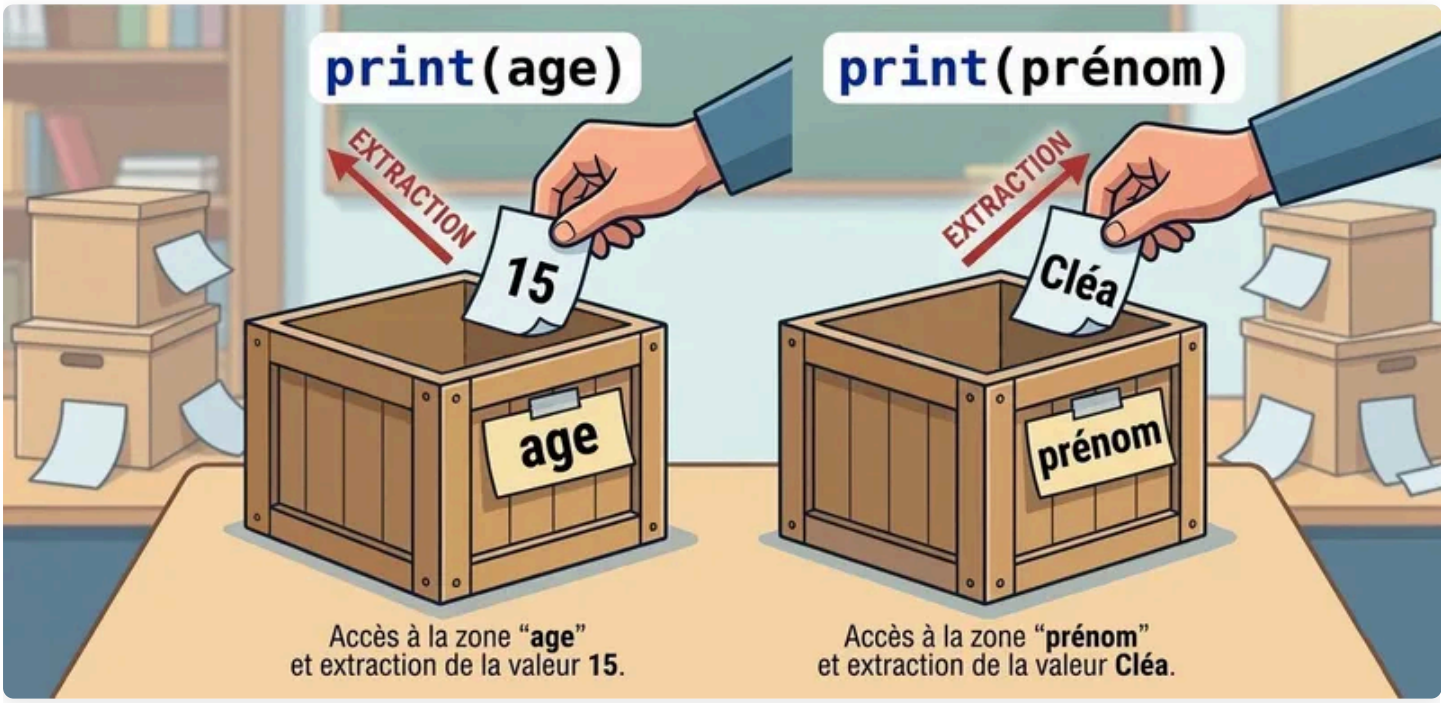
Cela peut peut être dû au fait que:

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

Utiliser les Variables

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

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

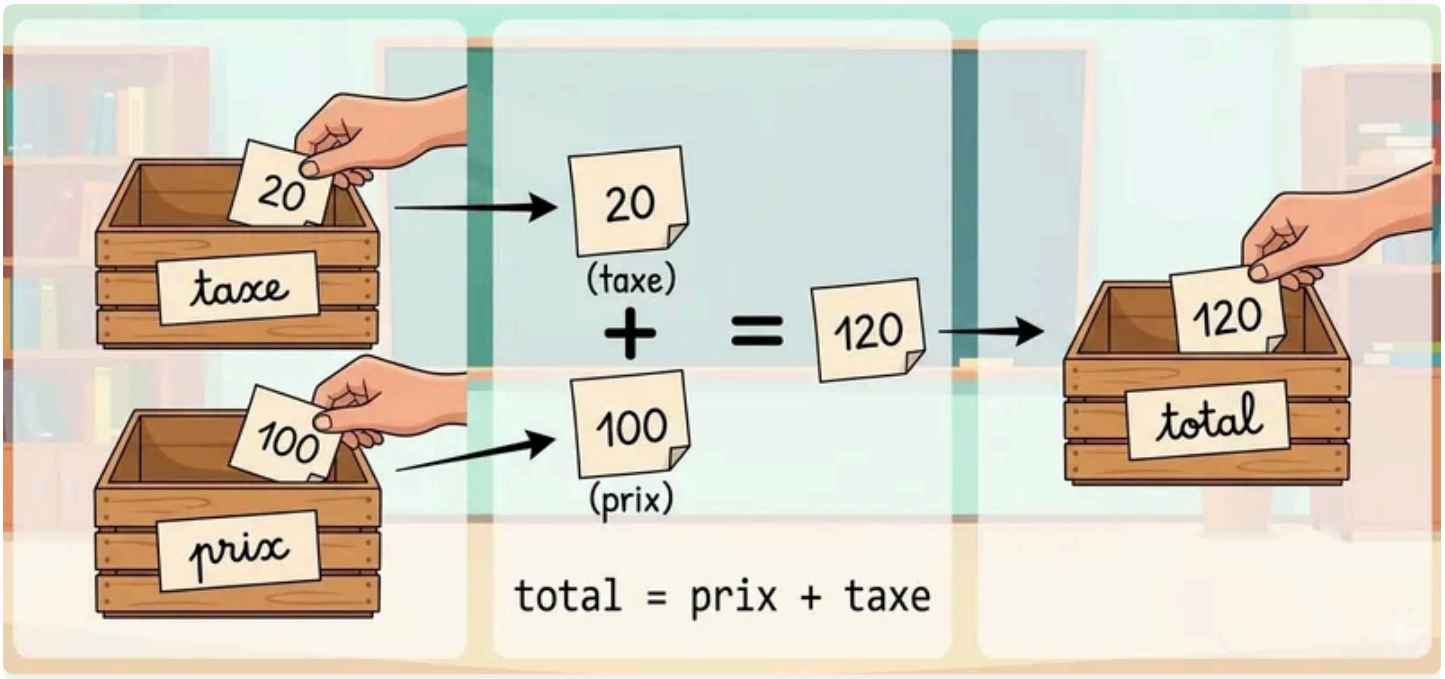


Explications

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

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

print (total) # Affiche 120
```



Explications

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

Ce code affiche:

```
120
```

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

```
brocolis = 18

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

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

Conclusion

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

Afficher ses variables avec print

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

Objectif

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

Introduction à `print()`

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

```
print(valeur)
```

(Attention aux paranthèses)

Exemple :

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

Résultat :

```
Bonjour tout le monde !
```

Afficher plusieurs valeurs

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

Exemple :

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

Résultat :

```
Bonjour tout le monde !
```

Affichage de variables

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

Exemples :

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

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

Résultat :

```
Nom : Alice Âge : 25
```

Changer le séparateur entre les valeurs

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

Exemple :

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

Résultat :

```
Python-est-génial
```

Modifier la fin de ligne

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

Exemple :

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

Résultat :

```
Bonjour tout le monde !
```

À retenir

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

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

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 :

```python
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.

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` .

▸ 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 conditions

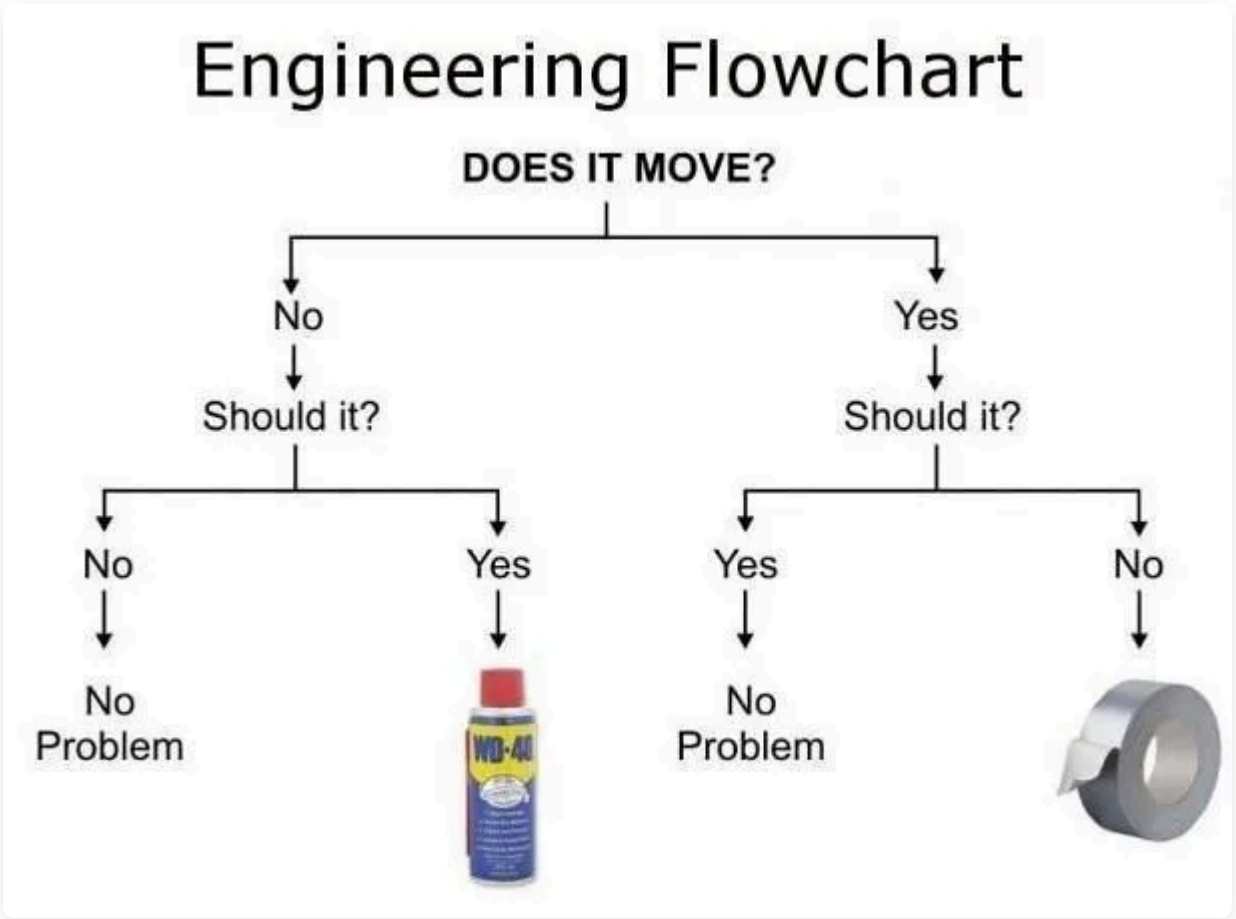
> *Lorsque nous débutons l'apprentissage de la programmation, l'un des concepts fondamentaux à maîtriser est celui des conditions. Elles sont au cœur de la prise de décision dans nos codes, permettant à nos programmes d'agir différemment selon les situations rencontrées. Dans cet article, nous explorerons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

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

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

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

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



L'utilité des conditions

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

Adaptation aux données entrantes

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

Validation des entrées

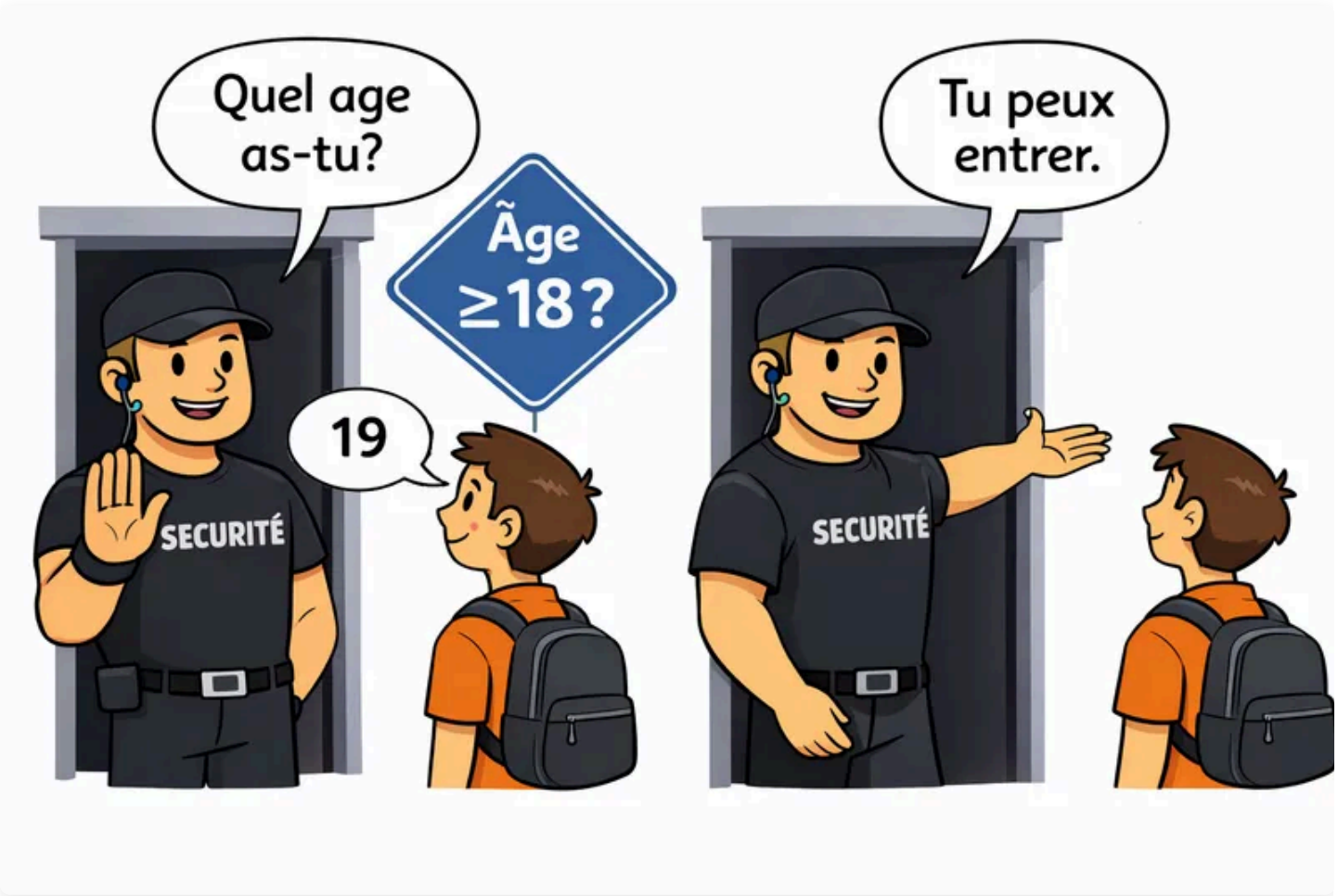
Un autre usage crucial des conditions est la **validation des entrées utilisateur**. Avant de traiter les données fournies par l'utilisateur, il est sage de vérifier leur validité. Les conditions aident à assurer que notre programme ne traite que des entrées appropriées, évitant ainsi des erreurs potentielles.

Logique de contrôle de flux

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

Mise en œuvre en Python

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




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

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

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

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

Conclusion

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

Python: Les conditions simples

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

L'instruction `if`

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

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

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

L'instruction `else`

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

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

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

L'instruction `elif` (else if)

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

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

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

Combinaison de Conditions

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

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

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

Pratique

PERMIS DE CHASSE

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

DIVISION

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

LES NOMBRES

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

IMC

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

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

Conclusion

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

Python: Opérations de comparaison

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

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

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

En Python, voici les principaux opérateurs de comparaison :

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

Exemples d'Utilisation

Voyons quelques exemples d'utilisation de ces opérateurs :

Égal à (`==`)

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

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

Taper ce code dans un éditeur:

```
a = 5
b = 15

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

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

Différent de (!=)

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

Inférieur à (<)

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

Supérieur à (>)

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

Inférieur ou égal à (<=)

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

Supérieur ou égal à (>=)

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

Intervalles

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

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

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

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

EXPLICATION TECHNIQUE :

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

EXEMPLE CONCRET :

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

```
temperature = 25

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

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

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

CAS D'UTILISATION :

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

BONNES PRATIQUES :

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

Combinaison d'Opérateurs de Comparaison

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

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

Lire l'article complet

Pratique

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

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

Bonus

Quel est le résultat du code suivant?

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

La réponse se trouve ici

Conclusion

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

Python: 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
```

Introduction à Python

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

Les avantages de Python

Simplicité et lisibilité

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

Polyvalence

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

Grande bibliothèque standard

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

Communauté active

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

Qui utilise Python et pourquoi ?

Développeurs web

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

Scientifiques des données

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

Ingénieurs en intelligence artificielle et en machine learning

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

Développeurs de logiciels et d'applications

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

La popularité croissante de Python

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

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

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

Notes de cours

- Télécharger les notes de cours "officielles"

EXERCICES DU JOUR

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

ELECTRONIQUE

Les actionneurs

> *Un actionneur est un composant ou un dispositif qui convertit un signal ou une commande électrique, pneumatique ou hydraulique en une action physique ou mécanique. Les actionneurs jouent un rôle essentiel en robotique, car ils permettent aux robots d'agir sur leur environnement et d'exécuter des tâches spécifiques.*

Utilité des actionneurs

Les actionneurs sont utilisés pour **générer des mouvements**, des forces ou des **actions physiques** en réponse à des signaux de commande. Ils permettent aux robots **d'interagir activement avec leur environnement** en effectuant des tâches précises. Les actionneurs sont essentiels pour la manipulation d'objets, le déplacement, la propulsion, le mouvement des articulations, le contrôle de la précision, etc.

Les actionneurs permettent aux robots d'**exercer des forces**, de produire des mouvements linéaires ou rotatifs, d'effectuer des changements d'état ou de position, d'appliquer des pressions, de saisir des objets, d'émettre des signaux et bien plus encore. Ils sont les "**moteurs**" qui permettent aux robots de **réaliser des actions concrètes dans le monde réel**.

Différents types d'actionneurs

Il existe plusieurs types d'actionneurs utilisés en robotique. Voici quelques-uns des plus courants :

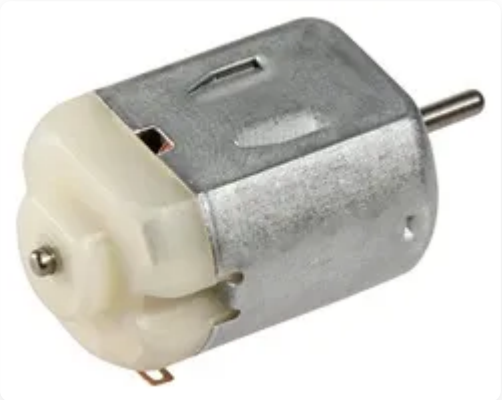
- **Actionneurs électriques** : Les actionneurs électriques, tels que les **moteurs électriques**, sont alimentés par de l'énergie électrique. Ils peuvent générer des mouvements rotatifs, linéaires ou vibratoires, et sont couramment utilisés dans **les bras robotiques**, les **roues motrices** et d'autres systèmes mécaniques.
- **Actionneurs pneumatiques** : Les actionneurs pneumatiques **utilisent de l'air comprimé** pour générer des forces et des mouvements. Les **vérins pneumatiques** sont un exemple d'actionneur pneumatique couramment utilisé pour produire des mouvements linéaires.
- **Actionneurs hydrauliques** : Les actionneurs hydrauliques **utilisent un fluide** hydraulique pour générer des forces et des mouvements. Les **vérins hydrauliques** sont fréquemment utilisés pour produire des **mouvements linéaires puissants** dans les robots industriels et d'autres applications nécessitant une force élevée.
- **Actionneurs piezoélectriques** : Les actionneurs piezoélectriques utilisent des matériaux piézoélectriques pour convertir une tension électrique en un **mouvement mécanique précis**. Ils sont utilisés dans des applications nécessitant **un contrôle très précis**, tels que les systèmes de positionnement fin et les microrobots.
- **Actionneurs électromagnétiques** : Les actionneurs électromagnétiques **utilisent des champs magnétiques** pour générer des mouvements ou des forces. Les **électroaimants** et les actionneurs à solénoïde sont des exemples courants d'actionneurs électromagnétiques.

Principaux actionneurs utilisés en robotique

- **Moteurs électriques** : Les moteurs électriques, tels que les moteurs à courant continu (DC) et les moteurs pas à pas, sont les actionneurs les plus couramment utilisés en robotique. Ils convertissent l'énergie électrique en mouvement mécanique pour produire des rotations, des translations ou des mouvements précis.
- **Relais**: Les relais sont des composants utilisés pour contrôler des circuits électriques puissants avec de faibles signaux électriques. Ils **fonctionnent comme des interupteurs**. Ils fonctionnent en utilisant une bobine électromagnétique qui active des contacts électriques pour **fermer ou ouvrir le circuit**. Les relais sont utilisés pour isoler les circuits de commande des circuits de puissance, ce qui permet de contrôler des charges élevées sans endommager les circuits de commande.
- **Vérins pneumatiques** : Les vérins pneumatiques utilisent l'air comprimé pour générer un mouvement linéaire. Ils sont souvent utilisés pour effectuer des mouvements rapides et puissants dans les applications robotiques nécessitant une force importante.
- **Vérins hydrauliques** : Les vérins hydrauliques utilisent un fluide hydraulique sous pression pour générer un mouvement linéaire puissant. Ils sont utilisés dans les applications nécessitant une force élevée et un contrôle précis.
- **Actionneurs électromagnétiques** : Les actionneurs électromagnétiques, tels que les électroaimants et les actionneurs à solénoïde, utilisent des champs magnétiques pour produire un mouvement linéaire ou rotatif. Ils sont couramment utilisés dans les mécanismes de commutation, les systèmes de verrouillage et d'autres applications.

Les 2 types de moteurs principaux

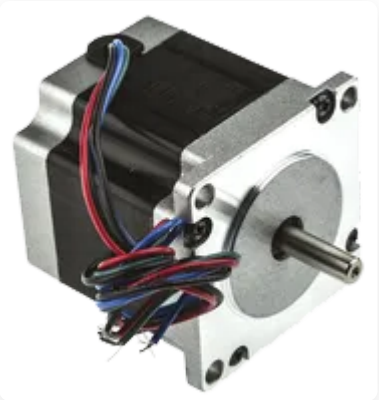
Moteurs à courant continu (DC)



Les moteurs à courant continu sont largement utilisés en robotique en raison de leur **simplicité**, de leur **coût abordable** et de leur contrôle précis de la **vitesse et de la direction**. Ces moteurs fonctionnent en utilisant un courant continu pour créer un champ magnétique qui génère un **couple de rotation**. Les moteurs DC sont disponibles dans différentes tailles et puissances, ce qui les rend polyvalents pour diverses applications robotiques, allant des petits robots autonomes aux bras robotiques industriels.

C'est ce genre de moteur que l'on utilise pour **faire tourner les roues du robot**.

Moteurs pas à pas et servomoteurs



Ces moteurs sont utilisés lorsque **la précision de positionnement est primordiale**. Ces moteurs se déplacent par incréments discrets, appelés "pas", en réponse à des signaux de commande spécifiques. Ils sont composés de bobines et d'un rotor magnétique, et fonctionnent en appliquant des impulsions électriques séquentielles aux bobines pour créer un champ magnétique rotatif.

Les moteurs pas à pas offrent un contrôle précis de la position et **peuvent maintenir une position sans consommer d'énergie** supplémentaire. Ils sont couramment utilisés dans les imprimantes 3D, les systèmes de positionnement, les robots de précision et les mécanismes nécessitant un contrôle précis de la rotation.

Il existe des différences entre les moteurs pas à pas et les servomoteurs. la principale différence réside dans la façon dont ces moteurs sont contrôlés et dans leur capacité à maintenir une position précise. **Les servomoteurs sont contrôlés en fonction de la position souhaitée**, tandis que **les moteurs pas à pas se déplacent par incréments précis** en réponse à des signaux de commande spécifiques. Les servomoteurs sont utilisés pour un positionnement précis à **un angle spécifique**, tandis que les moteurs pas à pas offrent une précision de positionnement élevée pour des **mouvements précis** et contrôlés.

Comparaison

Chaque type de moteur présente des avantages et des limites, et leur sélection dépend des exigences spécifiques de l'application robotique. Les **moteurs à courant continu** sont adaptés aux applications nécessitant un **contrôle de vitesse et de direction**, tandis que les **moteurs pas à pas** sont privilégiés pour leur **précision de positionnement**. Le choix du moteur dépendra donc des besoins spécifiques de conception, de contrôle et de précision du robot.

Conclusion

Chaque type d'actionneur a ses propres caractéristiques, avantages et limites, et est utilisé en fonction des besoins spécifiques du robot et de l'application. L'utilisation appropriée des actionneurs permet aux robots d'interagir avec le monde réel, d'exécuter des tâches et de réaliser des mouvements précis et contrôlés.

Feux de circulation

> *Cet article explique comment brancher et contrôler les 3 led du module 'feu de signalisation'*

Pour contrôler un feu de signalisation (traffic lights) à l'aide d'une Raspberry Pi Pico en MicroPython, vous aurez besoin de trois LED (rouge, jaune, verte) et de résistances appropriées pour les connecter (le composant reçu en classe est prêt à l'emploi). Voici un exemple de code qui montre comment cela peut être réalisé :

Matériel nécessaire :

- Raspberry Pi Pico.
- Un composant "Led traffic lights"
- Câbles de connexion.

Connexion des composants :

Connectez la LED rouge à une broche GPIO (par exemple GPIO2), la LED jaune à une autre broche (par exemple GPIO3) et la LED verte à une autre broche (par exemple GPIO4).

Code MicroPython pour contrôler les LEDs :

```
import machine
import utime

# Configuration des broches pour chaque LED
led_rouge = machine.Pin(2, machine.Pin.OUT)
led_jaune = machine.Pin(3, machine.Pin.OUT)
led_verte = machine.Pin(4, machine.Pin.OUT)

def feu_rouge():
    led_rouge.on()
    led_jaune.off()
    led_verte.off()
    utime.sleep(5)  # Rouge pour 5 secondes

def feu_jaune():
    led_rouge.off()
    led_jaune.on()
    led_verte.off()
    utime.sleep(2)  # Jaune pour 2 secondes

def feu_vert():
    led_rouge.off()
    led_jaune.off()
    led_verte.on()
    utime.sleep(5)  # Vert pour 5 secondes

while True:
    feu_rouge()
    feu_jaune()
    feu_vert()
```

Explication du code :

- Les LED sont initialisées en tant que sorties de broches GPIO.
- Trois fonctions (`feu_rouge` , `feu_jaune` , `feu_vert`) sont définies pour contrôler les états des LED.
- Chaque fonction active une LED tout en éteignant les autres, simulant ainsi un feu de signalisation.
- Une boucle infinie (`while True`) alterne entre les trois états, imitant le comportement d'un feu de signalisation réel.

N'oubliez pas de tester soigneusement votre circuit et votre code dans un environnement sûr pour éviter tout court-circuit ou dommage au Raspberry Pi Pico.

Introduction aux buzzers

> *Dans cet article, nous allons poser les bases nécessaires pour comprendre ce qu'est un buzzer, à quoi il sert et comment il est utilisé avec une carte microcontrôleur comme la Raspberry Pi Pico. Cet article est un **prérequis conceptuel** : il ne cherche pas encore à produire des sons complexes, mais à clarifier les différences fondamentales entre les types de buzzers et leurs implications côté matériel et côté code.*

Qu'est-ce qu'un buzzer ?

Un buzzer est un **composant électronique de sortie** dont le rôle est de produire un son.

On le rencontre très souvent dans des systèmes embarqués :

- bips de confirmation
- alertes sonores
- signaux d'erreur
- minuteries
- interfaces homme-machine simples

Contrairement à un haut-parleur, un buzzer est conçu pour être **simple à piloter**, avec peu de composants externes.

À quoi sert un buzzer dans un projet embarqué ?

Dans un projet pédagogique ou industriel, le buzzer sert principalement à :

- donner un **retour immédiat** à l'utilisateur
- signaler un **événement important** sans écran
- attirer l'attention (alarme, avertissement)

Il est particulièrement utile lorsque :

- l'affichage est absent ou minimal
- le système doit rester simple et robuste

Les deux grandes familles de buzzers

Il existe deux types de buzzers très courants en électronique embarquée :

- le **buzzer actif**
- le **buzzer passif**

Ils se ressemblent physiquement, mais leur fonctionnement interne et leur usage en programmation sont très différents.

Le buzzer actif

Un buzzer actif contient un **oscillateur interne**.

Cela signifie qu'il est capable de produire un son **tout seul**, dès qu'il est alimenté.

Caractéristiques principales

- une seule tonalité possible
- fréquence interne fixe
- pilotage très simple
- aucun réglage de hauteur sonore

Côté code

Pour faire sonner un buzzer actif :

- on met la broche à l'état HIGH → il sonne
- on met la broche à l'état LOW → il se tait

Il se pilote donc comme une **LED**, mais au lieu de s'allumer, il émet un son.

Le buzzer passif

Un buzzer passif **ne contient aucun oscillateur**.

Il ne peut pas produire de son sans un signal électrique approprié.

Caractéristiques principales

- nécessite un signal PWM (= qui varie)
- la fréquence du signal détermine la hauteur du son
- le rapport cyclique influence le volume
- permet de produire des sons variés, des alertes et des mélodies

Côté code

Avec un buzzer passif, le microcontrôleur doit :

- générer un signal périodique
- choisir la fréquence
- gérer la durée

Cela demande plus de code, mais offre beaucoup plus de possibilités.

Différences fondamentales entre actif et passif

Buzzer actif

- autonome
- simple
- une seule note
- peu flexible

Buzzer passif

- dépend du microcontrôleur
- plus complexe
- fréquences multiples
- très flexible

Pourquoi a-t-on souvent l'impression que le buzzer actif sonne plus fort ?

C'est une impression fréquente, mais trompeuse.

Un buzzer actif :

- fonctionne directement à sa fréquence de résonance
- est souvent alimenté avec une tension stable
- est optimisé pour produire un son immédiatement audible

Un buzzer passif :

- est souvent utilisé avec des réglages prudents
- est piloté en basse tension
- dépend fortement du signal généré par le code

Un buzzer passif **peut sonner aussi fort, voire plus fort**, mais uniquement s'il est correctement piloté (fréquence, tension, duty cycle).

Choisir le bon buzzer selon le projet

Utiliser un **buzzer actif** si :

- un simple bip suffit
- le code doit rester très simple
- on veut juste un signal sonore

Utiliser un **buzzer passif** si :

- on veut contrôler la hauteur du son
- produire des alertes complexes
- jouer des mélodies
- expérimenter avec le PWM

Erreurs fréquentes

- Confondre buzzer actif et passif
- Essayer de changer la fréquence d'un buzzer actif
- Utiliser un buzzer passif sans PWM
- Conclure que le buzzer passif est "moins puissant" par nature

---

Ce qu'il faut retenir

- Un buzzer est un dispositif de sortie sonore simple
- Actif et passif ont des usages très différents
- Le type de buzzer détermine directement le **type de code**
- La complexité sonore vient du **pilotage**, pas du composant seul

---

Dans le prochain article, nous utiliserons un **buzzer actif** pour produire des bips, gérer le rythme et construire progressivement un signal de type Morse, uniquement à l'aide de durées et de pauses.

Utiliser un buzzer actif (KY-012)

> *Dans cet article, nous allons apprendre à utiliser un **buzzer actif KY-012** avec une Raspberry Pi Pico. L'objectif est de comprendre comment produire des bips sonores, gérer le rythme et utiliser le temps comme information, jusqu'à construire un **code Morse simplifié**.*

Contrairement au buzzer passif, ici **la fréquence n'est pas contrôlable** : le buzzer actif ne sait produire **qu'une seule note**.

Rappel : principe du buzzer actif

Un buzzer actif contient un oscillateur interne. Dès qu'il est alimenté, il émet un son à une fréquence fixe.

Conséquence directe

- Pas de PWM
- Pas de fréquence à choisir
- Uniquement ON / OFF

Côté code, on travaille uniquement avec :

- l'état de la broche
- la durée pendant laquelle elle reste active

Connexion du KY-012

- `S` (Signal) → GPIO15
- `-` (GND) → GND
- `+` (VCC) → 3.3 V

Contrairement au buzzer passif, le buzzer actif **doit être alimenté**.

Premier son : un bip simple

On commence par le cas le plus élémentaire : un bip, puis arrêt.

```
from machine import Pin
import utime

buzzer = Pin(15, Pin.OUT)

buzzer.value(1)     # buzzer ON
utime.sleep(1)      # 1 seconde
buzzer.value(0)     # buzzer OFF
```

À retenir

- `value(1)` alimente le buzzer
- `value(0)` coupe le son
- la durée dépend uniquement du `sleep`

Bip + pause en boucle

On introduit maintenant la répétition.

```
from machine import Pin
import utime

buzzer = Pin(15, Pin.OUT)

while True:
    buzzer.value(1)
    utime.sleep(0.5)

    buzzer.value(0)
    utime.sleep(0.5)
```

Le buzzer émet un bip régulier, comparable à un métronome.

Bip court et bip long

La différence entre un bip court et un bip long est **uniquement temporelle**.

```
from machine import Pin
import utime

buzzer = Pin(15, Pin.OUT)

while True:
    # bip court
    buzzer.value(1)
    utime.sleep(0.2)
    buzzer.value(0)
    utime.sleep(0.3)

    # bip long
    buzzer.value(1)
    utime.sleep(0.6)
    buzzer.value(0)
    utime.sleep(0.8)
```

Introduire la notion de rythme

À ce stade, on ne change jamais le son lui-même. On ne joue que sur :

- la durée du bip
- la durée du silence

Avec un buzzer actif, **le rythme est l'unique information sonore**.

C'est exactement le principe du code Morse.

Principe du Morse (version pédagogique)

Le code Morse repose sur :

- un son court
- un son long
- des pauses

Pour simplifier :

- point → bip court
- trait → bip long

Nous n'utilisons pas ici les durées normalisées strictes, mais une version **compréhensible et lisible**.

Dictionnaire Morse simplifié

On associe chaque lettre à une séquence de symboles.

```
morse = {
    'S': '...',
    'O': '---',
    'I': '..',
    'N': '-.',
    'F': '..-.'
}
```

Jouer “SOS”

```
from machine import Pin
import utime

buzzer = Pin(15, Pin.OUT)

morse = {
    'S': '...',
    'O': '---'
}

dot = 0.2
dash = 0.6
pause = 0.3

while True:
    for letter in "SOS":
        for symbol in morse[letter]:
            buzzer.value(1)
            if symbol == '.':
                utime.sleep(dot)
            else:
                utime.sleep(dash)

            buzzer.value(0)
            utime.sleep(pause)

        utime.sleep(0.6)   # pause entre lettres
```

Jouer “INFO”

```
morse = {
    'I': '..',
    'N': '-.',
    'F': '..-.',
    'O': '---'
}

message = "INFO"
```

Le reste du code est identique : seul le dictionnaire et le message changent.

Ce que cet article a introduit

- pilotage ON / OFF
- gestion du temps
- notion de rythme
- dictionnaire comme table de correspondance
- boucle imbriquée

Erreurs fréquentes

- Essayer de modifier la fréquence d’un buzzer actif
- Oublier la pause entre lettres
- Alimenter le buzzer actif sans GND commun

Ce qu’il faut retenir

- Un buzzer actif ne produit qu’un seul son
- Toute l’information passe par la durée
- Le Morse est une application directe de cette logique
- Simplicité matérielle ≠ pauvreté pédagogique

Dans le prochain article, nous utiliserons un **buzzer passif (KY-006)** pour introduire la fréquence, le volume et la génération de sons évolutifs, jusqu’à des mélodies complètes.

Utiliser un buzzer passif (KY-006)

> *Dans cet article, nous passons au **buzzer passif KY-006**, beaucoup plus flexible que le buzzer actif. Il permet de **contrôler la hauteur du son**, le volume et les variations dans le temps. C'est avec lui que l'on peut produire des **alertes évoluées** et, progressivement, de **véritables mélodies**.*

Principe du buzzer passif

Un buzzer passif **ne produit aucun son tout seul**. Il doit être piloté par un **signal PWM**.

Avec un buzzer passif :

- la **fréquence** du PWM détermine la hauteur du son
- le **duty cycle** influence le volume
- la **durée** définit le rythme

On contrôle donc **trois paramètres**, contre un seul pour le buzzer actif.

Connexion du KY-006

- `S` (Signal) → GPIO15 (PWM)
- `-` (GND) → GND
- `+` → non connecté (c'est le `S` qui sert à générer le signal dans notre cas)

Initialisation de base

```
from machine import Pin, PWM
import utime

buzzer = PWM(Pin(15))
buzzer.duty_u16(2000)
```

À ce stade, le buzzer est prêt, mais **silencieux** tant qu'aucune fréquence n'est définie.

Alerte simple sur deux tons (ambulance)

On alterne deux fréquences distinctes.

```
while True:
    buzzer.freq(600)
    utime.sleep(0.5)

    buzzer.freq(900)
    utime.sleep(0.5)
```

Alerte avec montée progressive de fréquence

```
while True:
    for freq in range(400, 1000, 20):
        buzzer.freq(freq)
        utime.sleep(0.02)
```

La fréquence augmente progressivement, ce qui donne une sensation de montée de tension.

Alerte avec descente progressive de fréquence

```
while True:
    for freq in range(1000, 400, -20):
        buzzer.freq(freq)
        utime.sleep(0.02)
```

Montée puis descente de fréquence

```
while True:
    for freq in range(400, 1000, 20):
        buzzer.freq(freq)
        utime.sleep(0.02)

    for freq in range(1000, 400, -20):
        buzzer.freq(freq)
        utime.sleep(0.02)
```

Alerte à deux tons avec durée qui diminue

La durée de chaque note diminue de **25 % à chaque itération**.

```
duration = 2.0

while duration > 0.3:
    buzzer.freq(600)
    utime.sleep(duration)

    buzzer.freq(900)
    utime.sleep(duration)

    duration = duration * 0.75
```

Diminuer aussi la pause

```
duration = 2.0
pause = 1.0

while duration > 0.3:
    buzzer.freq(600)
    utime.sleep(duration)
    utime.sleep(pause)

    buzzer.freq(900)
    utime.sleep(duration)
    utime.sleep(pause)

    duration = duration * 0.75
    pause = pause * 0.75
```

Introduction aux notes de musique

On peut associer des noms de notes à des fréquences.

```
notes = {
    'C': 262,
    'D': 294,
    'E': 330,
    'F': 349,
    'G': 392,
    'A': 440,
    'B': 494,
    'C5': 523,
    'D5': 587,
    'E5': 659,
    'F5': 698,
    'G5': 784,
    'A5': 880
}
```

Mélodie simple, rythme monotone

```
melody = ['C', 'D', 'E', 'F', 'G', 'A', 'G', 'F']

for note in melody:
    buzzer.freq(notes[note])
    utime.sleep(0.4)
```

Introduire la durée avec des tuples

Chaque élément contient :

- la note
- la durée

```
melody = [
    ('C', 0.4),
    ('D', 0.4),
    ('E', 0.6),
    ('C', 0.6)
]
```

Lecture :

```
for note, duration in melody:
    buzzer.freq(notes[note])
    utime.sleep(duration)
```

Frère Jacques (8 premières notes)

```
melody = [
    ('C', 0.4), ('D', 0.4), ('E', 0.4), ('C', 0.4),
    ('C', 0.4), ('D', 0.4), ('E', 0.4), ('C', 0.4)
]

for note, duration in melody:
    buzzer.freq(notes[note])
    utime.sleep(duration)
```

Joyeux anniversaire (6 premières notes)

```
melody = [
    ('G', 0.4), ('G', 0.2),
    ('A', 0.6),
    ('G', 0.6),
    ('C5', 0.6),
    ('B', 1.0)
]

for note, duration in melody:
    buzzer.freq(notes[note])
    utime.sleep(duration)
```

Le finale

```
melody = [
    ('A4', 0.3), ('B4', 0.3), ('D5', 0.4), ('B4', 0.4),
    ('F5', 0.6), ('F5', 0.6),
    ('E5', 0.8)
]

notes.update({
    'A4': 440,
    'B4': 494,
    'D5': 587,
    'E5': 659,
    'F5': 698
})

for note, duration in melody:
    buzzer.freq(notes[note])
    utime.sleep(duration)
```

Erreurs fréquentes

- oublier d'utiliser le PWM
- confondre fréquence et durée
- jouer trop fort dès le début
- croire qu'une mélodie est autre chose qu'une suite de fréquences et de temps

---

Ce qu'il faut retenir

- le buzzer passif est entièrement piloté par le code
- fréquence = hauteur du son
- durée = rythme
- les mélodies sont des données, pas de la magie
- une structure simple permet des résultats complexes

Avec ces trois articles, les élèves disposent désormais :

- d'une vision claire des buzzers
- d'une progression logique
- d'un terrain idéal pour introduire abstraction, fonctions… et surprises sonores ultérieures

Les capteurs

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

Utilité des capteurs

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

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

Différents types de capteurs

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

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

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

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

Principaux capteurs utilisés en robotique

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

BME280: pression atmosphérique et altitude

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

🎯 Objectif du cours

Dans ce cours, tu vas apprendre à :

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

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

Comprendre la pression atmosphérique

🌍 Qu'est-ce que la pression atmosphérique ?

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

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

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

Le capteur BME280

Mesures disponibles

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

Communication

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

▸ Exemple de branchement (Pico → BME280)

BME280	PICO
VCC	3.3V
GND	GND
SCL	GP9
SDA	GP8

Lire les données *numériques* avec `raw_values`

Le module BME280 utilisé dans le CanSat propose :

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

Il retourne :

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

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

▸ Exemple de code

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

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

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

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

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

🌊 Qu'est-ce que la pression MSL ?

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

IMPORTANT :

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

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

📡 Où trouver la pression MSL du jour ?

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

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

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

🧮 La formule utilisée

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

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

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

POURQUOI ÇA FONCTIONNE ?

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

- la densité de l'air diminue,

- la pression diminue.

La formule ci-dessus exprime précisément cette diminution.

▸ 📘 Exemple concret

Imaginons que tu mesures :

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

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

→ Résultat ≈ **423 m**

Si tu avais utilisé 1013.25 hPa par défaut :

→ ≈ **500 m** (erreur de +77 m)

▸ 🌡️ Pourquoi la pression MSL change ?

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

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

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

🌍 Faut-il utiliser la pression MSL du lieu ?

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

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

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

Exemple :

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

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

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

🔹 Alors quelle valeur utiliser dans le CanSat ?

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

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

Sources :

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

C'est la méthode "officielle".

🔹 Option 2 : calibration locale pour plus de précision

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

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

ÉTAPES :

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

EXEMPLE

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

Tu cherches la valeur P_0 qui donne 510 m :

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

Résultat → **P_0 ≈ 1010.6 hPa**

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

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

🔹 5) Mini-algorithme conseillé pour le CanSat

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

Calculer l’altitude à partir de la pression

On utilise la formule barométrique simplifiée :

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

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

Code "complet" avec altitude

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

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

P0 = 1013.25   # pression standard au niveau de la mer

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

    # ICI: CALCULE L'ALTITUDE AVEC LA FORMULE SIMPLIFIEE

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

    time.sleep(1)
```

Exemple d'interprétation

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

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

À retenir

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

💡 Exercice

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

DHT11: Simulation du Capteur

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

Utiliser la classe `DHT11` en Python

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

Étape 1 : importer le module

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

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

```
from dht11 import DHT11
```

Étape 2 : créer un objet capteur

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

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

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

Étape 3 : demander une mesure

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

```
capteur.measure()
```

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

Étape 4 : lire les valeurs

Une fois la mesure faite, tu peux demander :

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

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

Exemple complet

```
from dht11 import DHT11

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

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

À retenir

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

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

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

DHT11: Capteur de température et d'humidité

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

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

Caractéristiques

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

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

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

DHT22: alternative plus précise

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

Configuration matérielle

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

Installation de la bibliothèque DHT11

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

Exemple de code pour la lecture des données du capteur DHT11

```
import machine
import dht
import time

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

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

Explication du code

Bien sûr, voici une explication pour ce code :

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

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

Infos supplémentaires

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

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

HC-SR04: Capteur de distance à ultra-sons

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

Introduction au capteur de distance HC-SR04

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

Configuration matérielle

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

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

```
import machine
import utime

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

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

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

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

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

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

    return distance

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

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

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

Explication du code

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

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

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

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

Installation du module open source

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

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

```
import machine
import utime
from hc_sr04 import HCSR04

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

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

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

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

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

Explication du code

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

Ressources supplémentaires

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

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

Ressources supplémentaires

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

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

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

KY-004: Bouton

> *Cet article présente le KY-004, un bouton poussoir facile à utiliser.*

Le module KY-004 est un bouton poussoir simple qui peut être utilisé dans de nombreux projets avec un Raspberry Pi Pico en MicroPython. Voici un exemple de code qui montre comment utiliser le KY-004 pour allumer et éteindre une LED en appuyant sur le bouton.

Matériel nécessaire :

- Raspberry Pi Pico.
- Module KY-004 Button.
- Le feu rouge
- Câbles de connexion et breadboard.

Connexion des composants :

- Connectez les LED des feux de circulation
- Connectez la broche `S` (Signal) du KY-004 à une autre broche GPIO (par exemple GPIO3).
- Connectez la broche `-` (GND) du KY-004 à GND de la Pico.

Code MicroPython :

```
from machine import Pin
import utime

# Configuration de la broche pour la LED et le bouton
led_rouge = Pin(2, Pin.OUT)
button = Pin(3, Pin.IN)

led_state = False

def toggle_led():
    global led_state
    led_state = not led_state
    led_rouge.value(led_state)

# Boucle principale
while True:
    if button.value() == 1:  # Vérifie si le bouton est appuyé
        toggle_led()         # Change l'état de la LED
        utime.sleep(0.5)     # Anti-rebond (debounce), évite les changements d'état multipl
```

Explication :

- La broche de la LED est configurée en sortie (`Pin.OUT`), et la broche du bouton en entrée (`Pin.IN`).
- La fonction `toggle_led` change l'état de la LED à chaque appel.
- Dans la boucle principale, le programme vérifie si le bouton est appuyé. Si c'est le cas, il appelle `toggle_led` pour changer l'état de la LED.
- Un délai de 0.5 seconde après chaque appui sur le bouton évite les rebonds (changement d'état multiple dû à la pression physique du bouton).

Ce code est un bon exemple pour apprendre les bases de la lecture des entrées numériques et le contrôle des sorties numériques en MicroPython.

Quel est le proglème avec ce code?

Pour améliorer la gestion du bouton, essaie cette version: bouton avec lecture unique