Initiation à la programmation en python

mecamic

Table des matières

Premier programme Python I

1. Choix de l'environnement de programmation

Langage retenu *Fondamental*

Un langage de programmation est nécessaire pour l'écriture des programmes : un langage simple d'usage, interprété, concis, libre et gratuit, multi-plateforme, largement répandu, riche de bibliothèques adaptées aux thématiques étudiées et bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif est nécessaire. Le langage choisi est **Python**.

Python, un langage interprété

Chaque programme est un ensemble d'instructions, qu'il s'agisse d'additionner deux nombres ou d'envoyer une requête sur Internet. Les compilateurs et les interpréteurs prennent le code lisible par l'homme et le convertissent en code machine lisible par l'ordinateur.

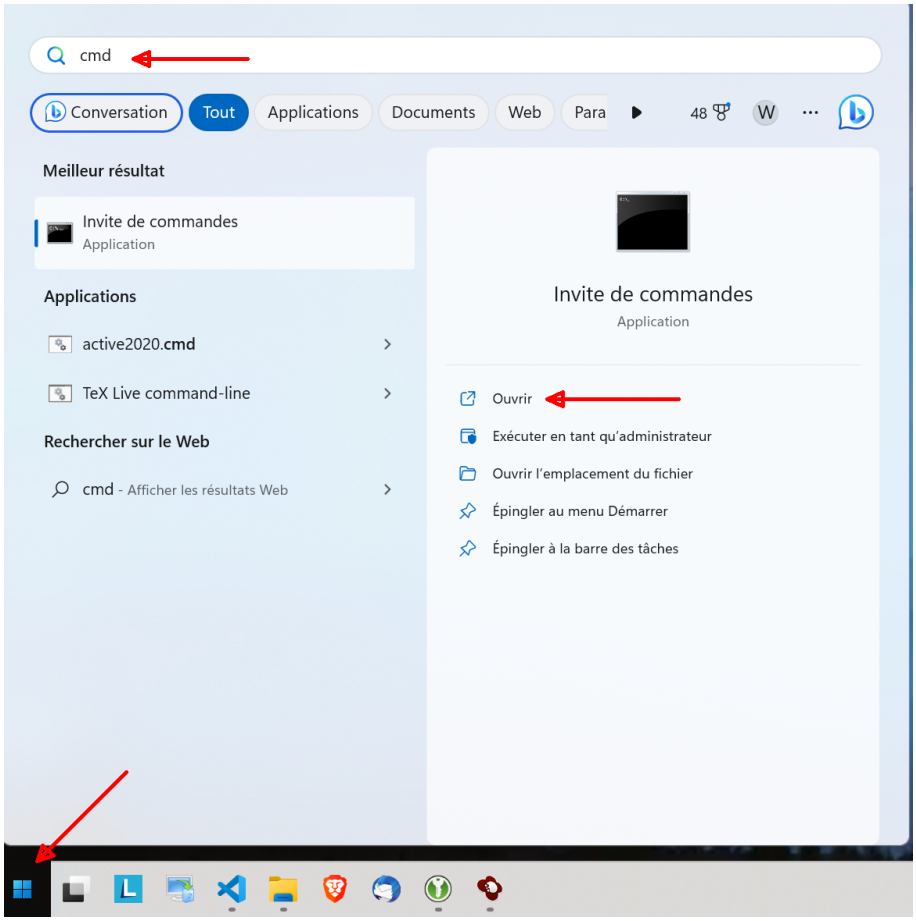
Dans un langage compilé, la machine cible traduit directement le programme. Dans un langage interprété, le code source n'est pas directement traduit par la machine cible. Au lieu de cela, un programme différent, appelé l'interpréteur, lit et exécute le code.

- Les langages compilés sont convertis directement en code machine que le processeur peut exécuter. Par conséquent, ils ont tendance à être plus rapides et plus efficaces à exécuter que les langages interprétés. Ils permettent également au développeur de mieux contrôler les aspects matériels, comme la gestion de la mémoire et l'utilisation du processeur. Par contre il faudra produire un code différent suivant la machine qui l'exécutera. Ces langages doivent recompilés après chaque modification. Des exemples de langages purement compilés sont C, C++, Go...
- Les langages avec interpréteurs parcourent un programme ligne par ligne et exécutent chaque commande. Les langages interprétés étaient autrefois nettement plus lents que les langages compilés, mais, cet écart se réduit.. Un langage interprété pourra fonctionné avec n'importe quel système. Des exemples de langages interprétés sont PHP, Ruby, Python et JavaScript.

Environnement de développement

Un environnement de développement comprends au minimum :

1. Un éditeur (de texte) de code source est un logiciel permettant la rédaction de texte brut, sans aucune mise en forme.
2. Le shell (Anglicisme informatique) est l'Interface utilisateur d'un système d'exploitation, principalement destinée à lancer d'autres programmes et gérer leurs interactions. Le terme est généralement utilisé pour parler d'une interface en ligne de commande. On peut aussi nommer cette partie par le terme console ou terminal. Sous windows il suffit de taper « cmd » pour l'ouvrir.



Fenêtre de commande

Environnement de développement intégré

Un environnement de développement intégré, ou IDE, est un logiciel de création d'applications, qui rassemble des outils de développement fréquemment utilisés dans une seule interface utilisateur graphique (GUI). Un IDE se compose habituellement des éléments suivants :

- Éditeur de code source : un éditeur de texte qui aide à la rédaction du code logiciel, avec des fonctions telles que la coloration syntaxique avec repères visuels, la saisie automatique en fonction du langage et la vérification de bogues dans le code pendant la rédaction.
- Utilitaires d'automatisation de version locale : des utilitaires qui permettent d'automatiser des tâches simples et reproductibles lors de la création d'une version locale du logiciel à destination du développeur lui-même, par exemple la compilation du code source en code binaire, la mise en paquet du code binaire et l'exécution de tests automatisés.
- Débogueur : un programme qui permet de tester d'autres programmes en affichant l'emplacement des bogues dans le code d'origine.

En python l'IDE s'appelle l'IDLE qui veut dire :

Python's Integrated Development and Learning Environment.

Thonny

Pour notre part nous utiliserons Thonny[1] . Qui est un environnement dédié à la programmation python et également adapté à la gestion des microcontrôleurs.

Utilisation de la clef USB

Pour installer et commencer à utiliser Thony je vous renvois vers les cours spécifiques à ce sujet .

Quelques liens utiles

Remarque

Le site vitascience[2] permet une initiation informatique en passant d'un éditeur graphique de type scratch à un éditeur textuel. Il est orienté vers l'Internet des objets avec le pilotage de microcontrôleurs.

Trinket[3] permet de programmer en python 3 sans rien installer. Vous trouverez ici un tutoriel[4] qu'il faut télécharger, puis décompresser. **Cet outil peut vous être utile chez vous**.

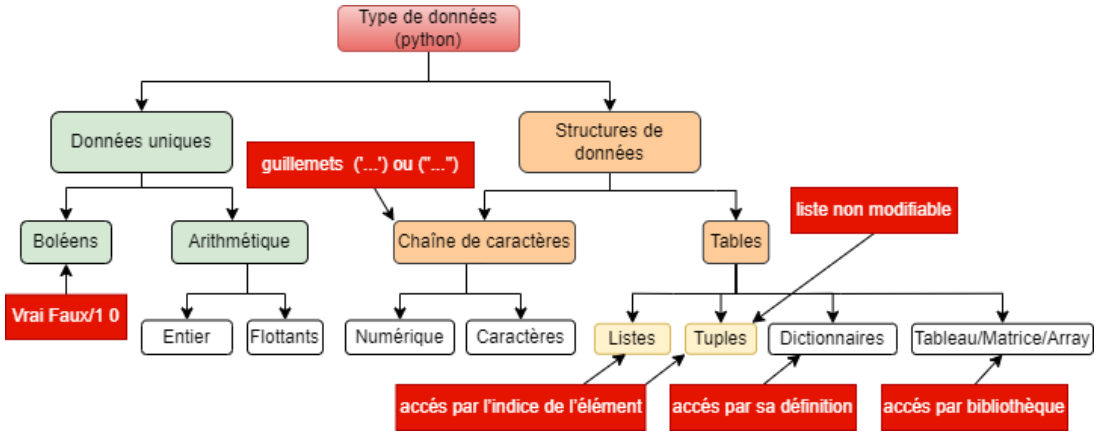
Le site france-ioi.org[5] permet au travers de son parcours lycée en python d'appréhender l'ensemble de formation python du lycée. **Si vous voulez apprendre python en autodidacte cet outil est fait pour vous**. Le site futurecoder.io[6] permet également d'apprendre le python en partant du début.

Pythontutor[7] est un outil en ligne qui permet de déboguer votre programme python ligne après ligne. Voici un tutoriel[8] pour s'en servir. **Il faut l'avoir sous la main**.

La référence du tutoriel[9] python.

2. Quelles sont les données traitées

Suivant la nature des éléments traités on peut distinguer les types de données suivantes. Ce tableau est orienté vers les données traitées avec python, mais on retrouve ce type de classement d'autres langages.



Types de donnée

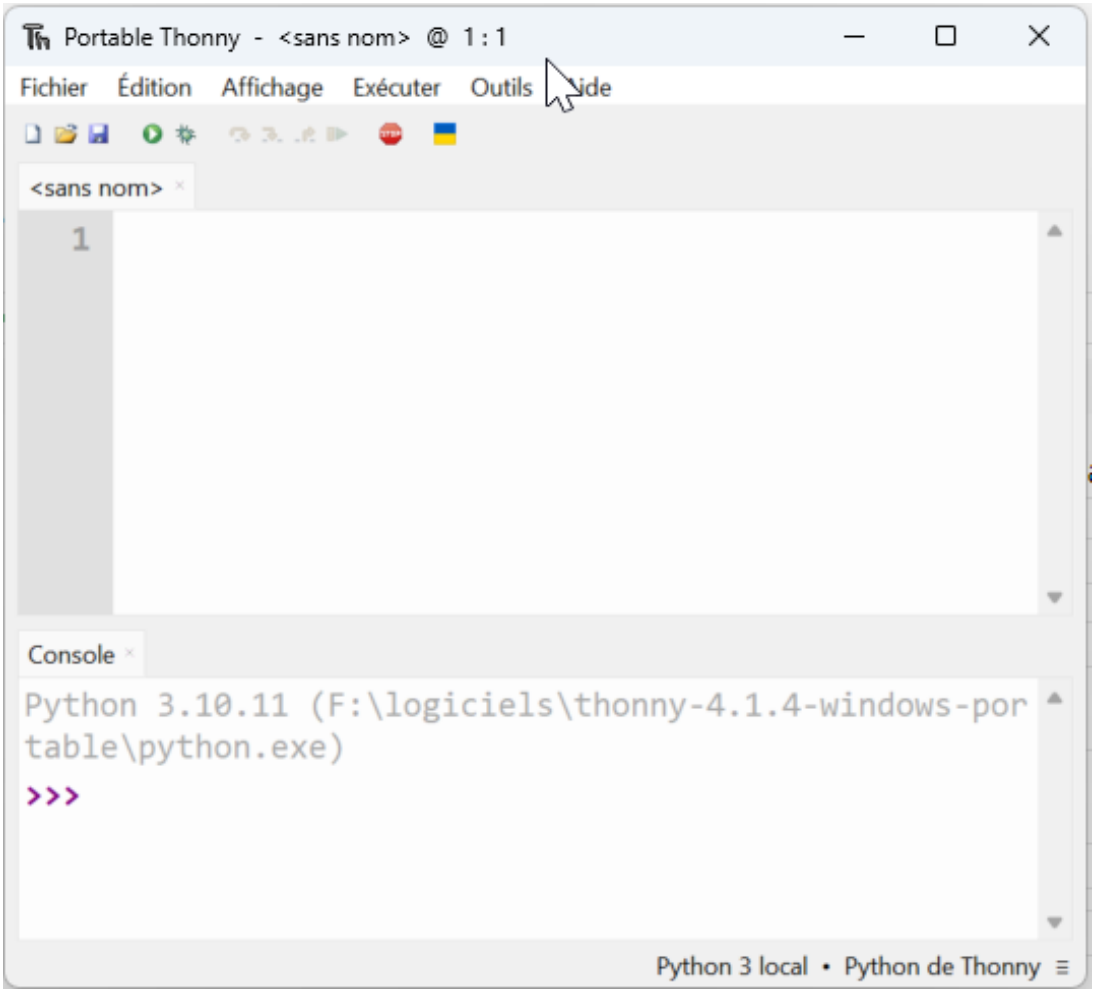
1. https://thonny.org/
2. https://fr.vittascience.com/
3. https://trinket.io/features/python3
4. https://wendling.xyz/nextcloud/s/yW8SMPDZz4PDBb7
5. http://www.france-ioi.org/algo/chapters.php
6. https://fr.futurecoder.io/
7. http://pythontutor.com/
8. https://robertvandeneynde.be/parascolaire/pythontutor.html
9. https://docs.python.org/fr/3/tutorial/index.html

3. Le premier programme

Premier démarrage

Au premier démarrage Thonny comporte deux fenêtres :

1. Dans la partie supérieure **l'éditeur** qui comporte les scripts du programme.
2. Dans la partie inférieure **la console ou shell** où s'affiche les résultats de l'exécution de votre code.



Fenêtres initiales Thonny

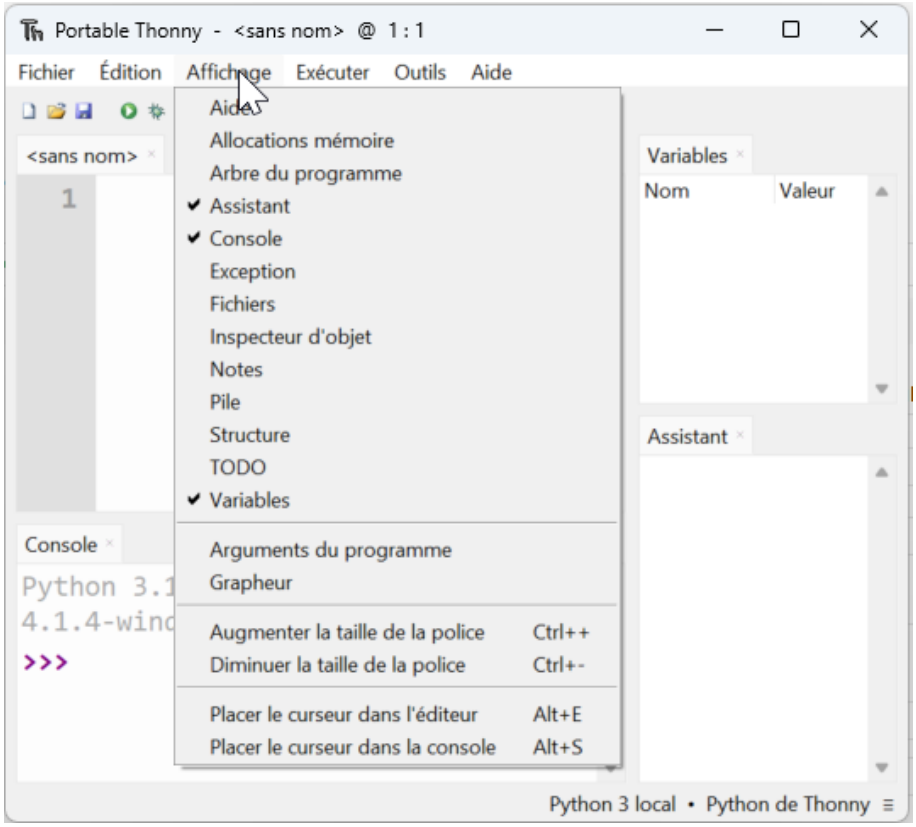
Vous noterez la présence de la flèche verte qui permet l'exécution d'un programme.

Exécuter un programme

Affichage complémentaire

Pour avoir un affichage plus complet vous pouvez afficher deux fenêtres supplémentaires :

1. La fenêtre qui les variables utilisées
2. La fenêtre assistant qui fournie une aide à la rédaction et à l'interprétation des scripts.



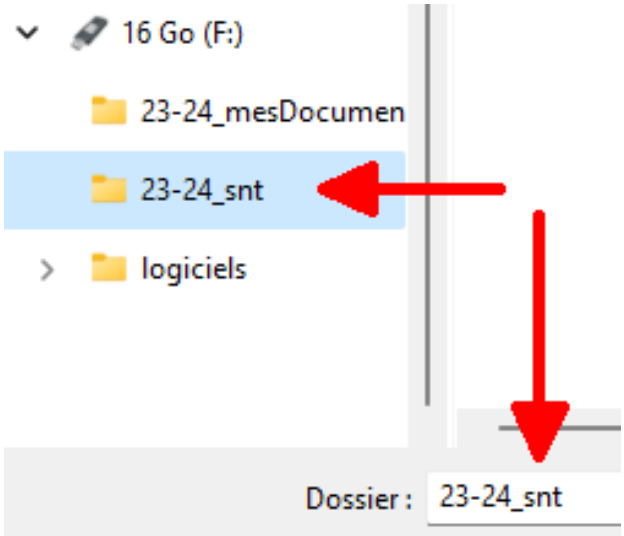
Affichage complémentaire

Créer le premier programme

Il faut à ce moment créer le fichier « helloworld.py ». Pour cela copier coller le programme ci-dessous dans la partie éditeur.

```
1 print("Hello, world")
```

- Puis enregistrer le sous « helloworld.py » dans votre dossier de travail.



Dossier sur la clef

- Puis exécutez le.
- Vérifier le résultat qui apparait dans le fenêtre shell.

Utilisez Trinket

Faite la même chose en utilisant le site trinket[10].

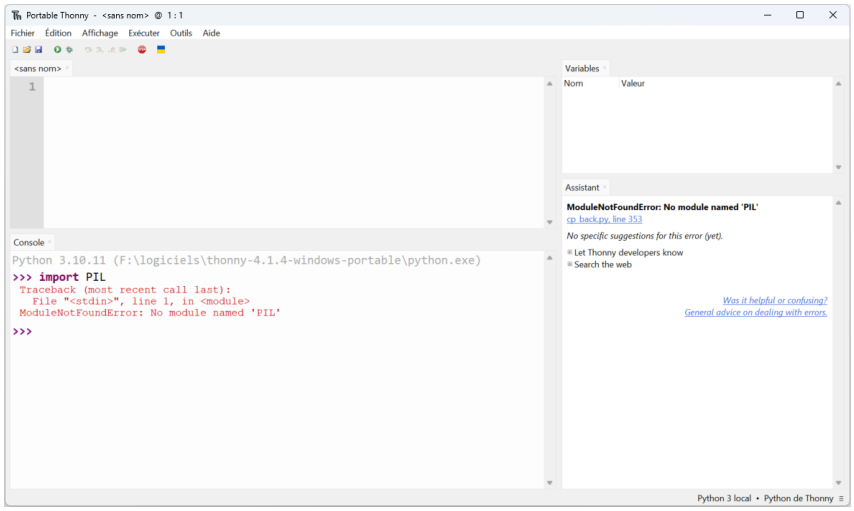
4. Installer une bibliothèque dans Thonny

Dans tout ce tutoriel, nous installons la bibliothèque `pillow` qui se charge en utilisant `import PIL`. Si vous souhaitez installer une autre bibliothèque, remplacez simplement toutes les occurrences de `pillow` par la bibliothèque que vous souhaitez installer.

Vérifiez que la bibliothèque n'est pas déjà installée.

Dans le shell ; la partie basse de la fenêtre, écrivez `import PIL` et validez.

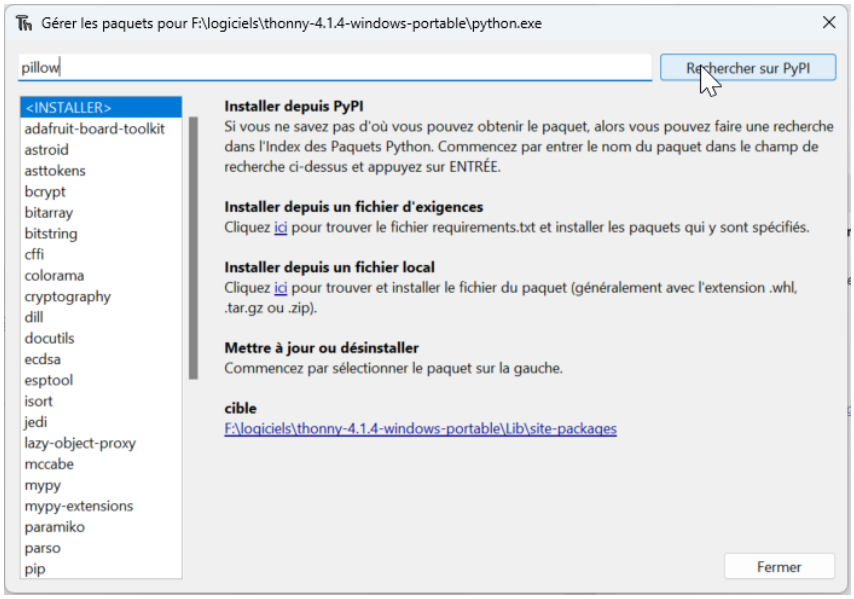
- Si aucun message ne s'affiche, la bibliothèque est déjà installée. Pas besoin d'aller plus loin.
- Si un message d'erreur s'affiche, qui se termine par `ModuleNotFoundError: No module named 'PIL'`, suivez les instructions suivantes pour installer la bibliothèque.



Bibliothèque non installée

Installez la bibliothèque désirée

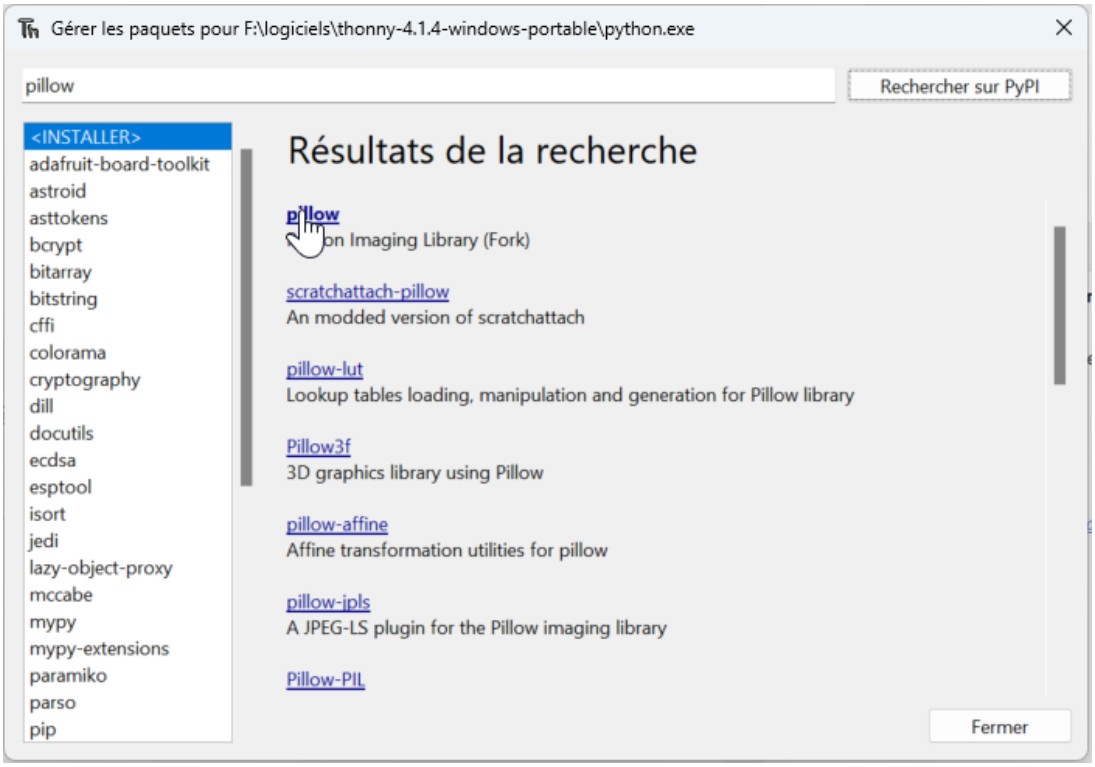
- Allez dans le menu Outils , puis Gérer les paquets...
- Dans la fenêtre qui s'ouvre, inscrivez `Pillow`, lancez la recherche.



Lancer la recherche

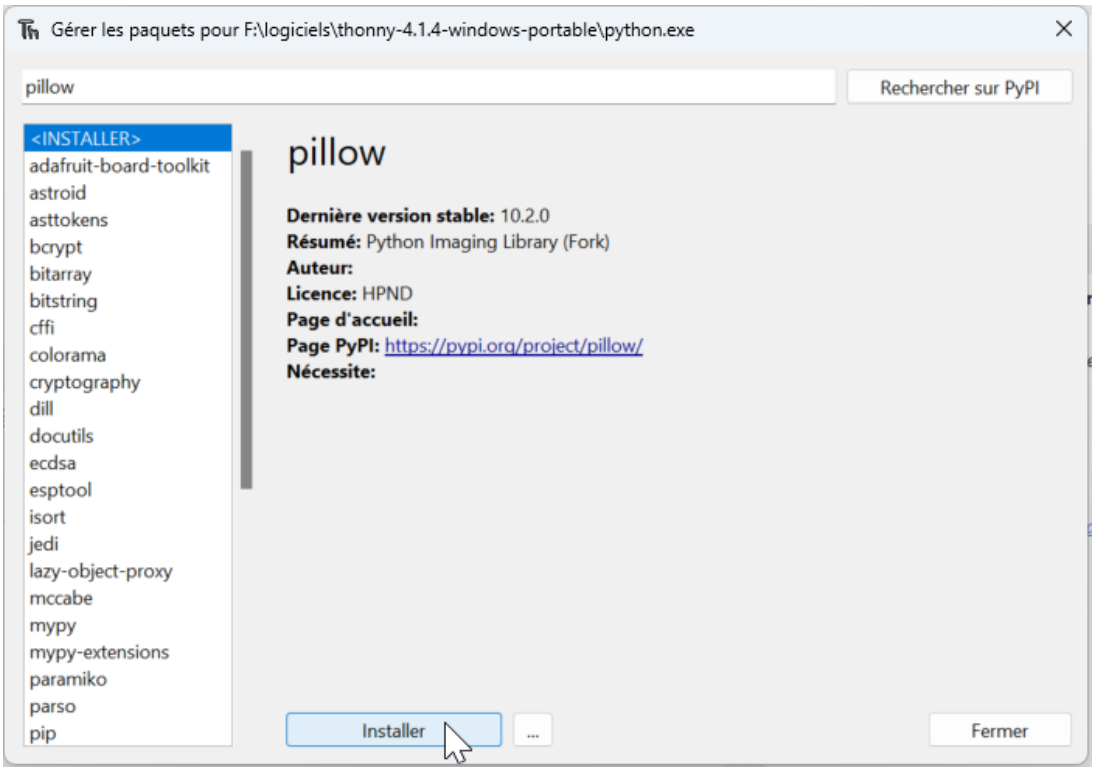
10. https://trinket.io/

- Choisir la bibliothèque



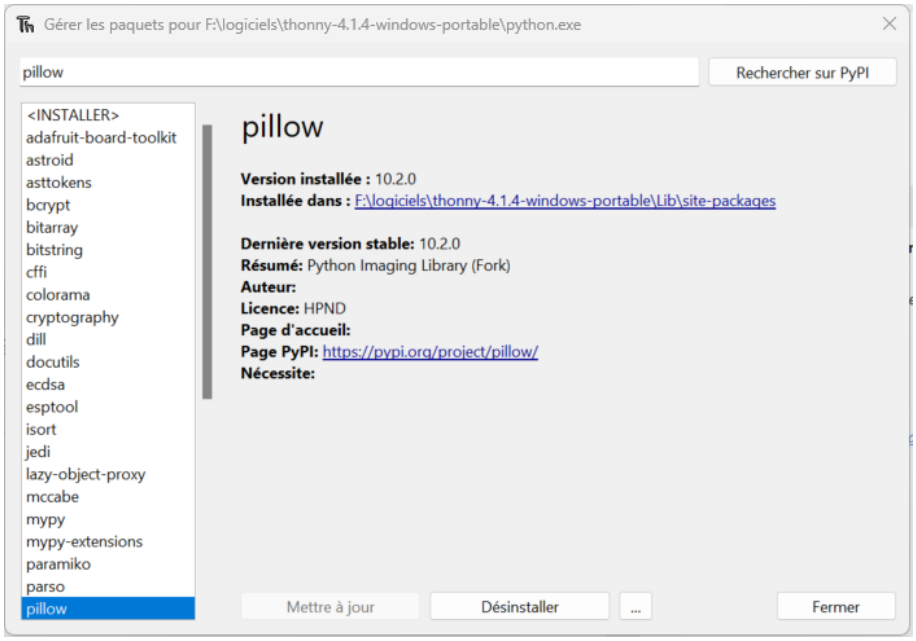
Choisir la bibliothèque

- Puis cliquez sur le bouton `Installer`l pour l'installer.



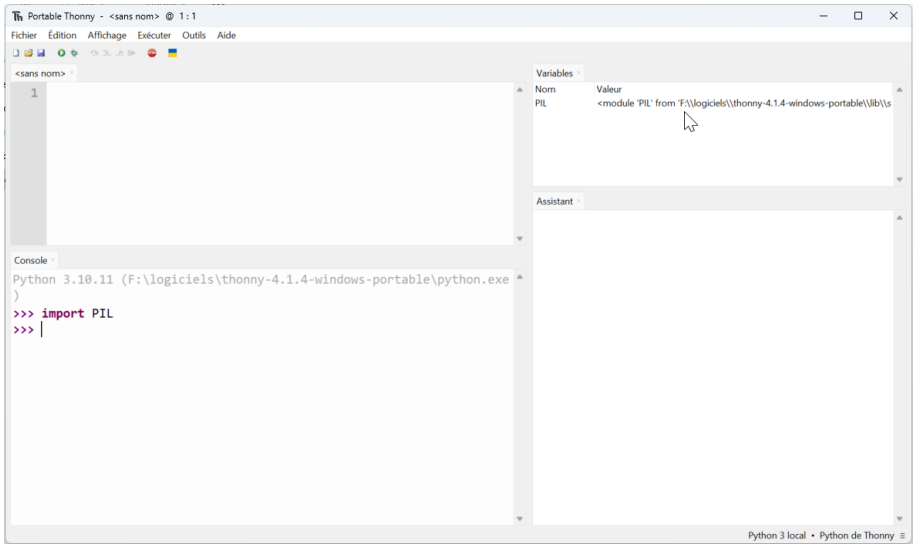
Installer la bibliothèque

- La bibliothèque est installée elle apparaît dans la liste de gauche, et il y a un bouton pour la désinstaller.



Bibliothèque installée

- Quittez Thonny
- Rédémarrez le
- Dans le shell, écrivez à nouveau `import PIL`. Si rien ne s'affiche, tout a été installé correctement.



Vérification de l'installation

5. Exercice : A faire Mes premiers codes

Utiliser la fonction `print` pour afficher la phrase "J'apprends Python !"

Calculer le résultat de `17 + 35 * 2`

6. Exercice : Synthèse premier programme en python

Choix de l'environnement de programmation

Un langage de programmation est nécessaire pour l'écriture des programmes : un langage simple d'usage, interprété, concis, libre et gratuit, , largement répandu, riche de adaptées aux thématiques étudiées et bénéficiant d'une d'auteurs dans le monde éducatif est nécessaire. Le langage choisi est .

Environnement de programmation

Pour programmer en python il faudra installer .

Pyzo

Nous installerons également fourni une aide à la programmation. En effet cet interface à l'aide de ses :

- , l'endroit où l'on .
- qui permet de se déplacer
- , fenêtre dans laquelle

De plus on dispose

- qui permet de donner des informations sur les erreurs de code.

Trinket

Sur le web[11] permet de programmer en python à partir d'un .

Affichage du code

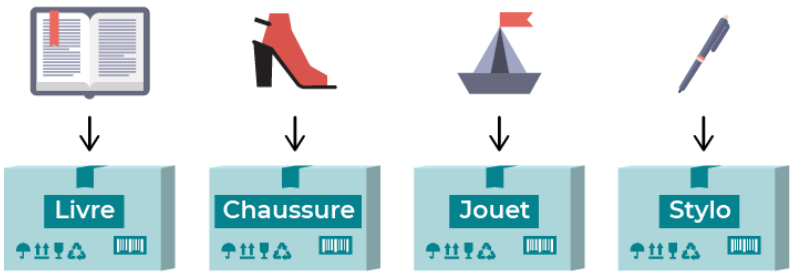
- Pour du code on utilise la commande ; avec l'expression à imprimer
- Si je veux afficher du il faut le placer entre '' ''

11. https://trinket.io/

Les variables et les données II

1. Les variables

Une variable, c'est comme une boîte. C'est un moyen d'enregistrer les données. Cet enregistrement permet par la suite de les récupérer et de les classer par type de même nature. Un nom d'utilisateur, des tickets d'avion encore disponibles, le jour de la semaine, un inventaire... Toutes ces données sont enregistrées dans les variables.



Les variables

C'est donc une petite information ; une donnée temporaire que l'on stocke dans une case de la mémoire de l'ordinateur. On dit qu'elle est "variable" car c'est une valeur qui peut changer pendant le déroulement du programme.

Ecriture d'une variable *Fondamental*

Ici on affecte à la variable livre le contenu Alice au pays des merveilles

```
1 livre = "Alice au pays des merveilles"
```

Définition

Une variable est composée de deux éléments :

- un nom : livre
- une valeur : Alice au pays des merveilles
    - Cette valeur peut être de différents types : texte, nombre chiffre.

2. Modifiez la valeur d'une variable

On peut donc facilement modifier le contenu d'une variable

```
1 livre = "Alice au pays des merveilles"
2 print(livre)
```

Vous devez obtenir :

Alice au pays des merveilles

Nous allons maintenant modifier la valeur de cette variable.

```
1 livre = "Blanche-Neige et les Sept Nains"
2 print(livre)
```

Vous devez obtenir :

Blanche-Neige et les Sept Nains

3. Afficher des variables

Nous avons vu comment afficher une seule variable avec la fonction « print » .

Mais parfois, on a besoin d'afficher des messages mélangeant des variables et des chaînes de caractères. On va faire cela avec la « f-string ».

Commande f-string *Définition*

Pour afficher des variables, la f-string permet d'insérer facilement les variables dans la chaîne de caractères à afficher. Une f-string est une chaîne de caractères précédée d'un f (ou F), et contenant des expressions entre accolades {} qui seront évaluées lors de l'exécution du programme.

```
1 nom = "Dupont"
2 prenom = "Jean"
3 age = 30
4
5 print(f"Bonjour, je m'appelle {prenom} {nom} et j'ai {age} ans.")
```

Vous devez obtenir :

Bonjour, je m'appelle Jean Dupont et j'ai 30 ans.

4. Nommage des variables

- Utilisez des noms descriptifs

  Au lieu de quantite, ou pire ; qte, ajoutez des détails ; quantite_en_stock, solde_actuel
- Utilisez des mots complets
- Suivez une convention d’appellation communément adoptée en informatique

  En utilisant le tiret bas _ pour séparer les mots
- Commencez avec une lettre ou le tiret bas

  En effet une variable ne peut pas commencer par un chiffre
- Utilisez uniquement des caractères alphanumériques et des tirets bas et surtout pas d'accents
- N’oubliez pas que les noms de variables sont sensibles à la casse

  Faite attention à l'utilisation des majuscules et des minuscules ; Une bonne approche est donc de ne pas utiliser les majuscules.

5. Type de données

Les nombres

Les types de données les plus simples, ou primitifs, utilisés dans Python sont :

- Les entiers (Integers en anglais)
    - 1
    - 4
    - 3914
- Les virgules flottantes (Float en anglais). **Attention il faut mettre des points et non des virgules comme séparateur.**
    - 3.14
    - 536.36
    - 99.9

Vous pouvez réaliser toutes sortes d'opérations arithmétiques entre ces nombres en voici quelques-unes :

- x + y : la somme de x et y
- x - y : la différence entre x et y
- x * y : le produit de x et y
- x / y : le quotient de x et y
- x % y : donne le reste de x divisé par y

Les chaînes de caractères

Une chaîne c'est une information généralement du texte entourée de guillemets soit simples soit doubles. Quelques exemples de variables de type string ou chaîne de caractères en anglais :

- `salutations = "bonjour !"`
- `le_professeur = "Paul-Emile"`
- `prenom = "Sam"`
- `citation_inspirante = "vous ratez 100 % des opportunités que vous ne saisissez pas"`

Lorsqu'une apostrophe est présente dans la chaîne, comme dans le message "J'aime Python.", si on délimite la chaîne par une apostrophe, python stoppera la chaîne à la seconde apostrophe rencontrée, ce qui provoquera une erreur.

Nous disposons de plusieurs façons de contourner le problème.

- La première consiste à délimiter la chaîne par des guillemets doubles ou triples.

```
1 a = "J'aime Python."
2 a
3 "J'aime Python."
```

- La seconde consiste à échapper l'apostrophe à l'intérieur de la chaîne à l'aide d'un \. Nommée backslash ou encore antislah ou contre-oblique ou barre oblique inversée ou encore barre d'échappement.

```
a = 'J\'aime Python.'
a
"J'aime Python."
```

Les booléens

Un booléen n'a que deux options de valeur : True (vrai) ou False (faux).

- Par exemple, si vous voulez avoir une variable qui enregistre le fait qu'il y a du soleil dehors ou pas, vous pouvez dire,

S'il n'y a pas de soleil.

```
climat_ensoleille = False
```

Déterminer le type de données

Pour déterminer le type de données on utilise la commande "type()" qui renverra le type le type de donnée contenu dans la variable.

6. Opération sur les entiers et les virgules flottantes

Voici les opérations courantes :

- x+y
- x-y
- x*y
- x/y : le quotient de x divisé par y
- x%y : le reste de x divisé par y

7. Exercice : A faire Variables et données

Déclarez la variable `couleur` avec la valeur `'verte'`

Modifiez la valeur de la variable `couleur` à `'jaune'`

Affichez la valeur de la variable couleur avec la fonction `print`

Ecrivez votre nom dans une variable appelée `nom`

Utilisez la fonction `print` pour afficher le contenu de la varibale `nom` dans le terminal.

Stockez le résulat de `x + y` dans une variable notée `z`

Affichez le résultat de la variable `z` dans le terminal à l'aide de la fonction `print`

8. Exercice : Synthèse variables et données

Les variables

C'est une que l'on stocke dans une case de la mémoire de l'ordinateur. On dit qu'elle est " " car c'est une valeur qui pendant le déroulement du .

Si on affecte à la à la variable le contenu

= " "

Nommage des variables

- Utilisez des noms descriptifs : quantite_en_stock
- Utilisez des mots complets
- Utilisez uniquement des caractères et des et surtout

Type de données

- Les entiers ou
- Les ou float
- Les ou strings
- Les booléens

Les opérations

On retrouve les quatre opérations de base plus l'opération qui donne le reste de x divisé par y : x%y

Listes et tuples

III

1. Les listes

Qu'est-ce qu'une liste et pourquoi l'utiliser ?

Les pommes, les oranges, les poires. 🍎 🍊 🍐 . Une liste enregistre une collection d'objets auxquels vous voulez accéder plus tard.

Fondamental

Dans Python, on utilise des crochets `[]` pour indiquer une liste. Le code suivant crée une liste de différentes plateformes de réseaux sociaux et la sauvegarde dans une variable appelée :

- plateformes_sociales

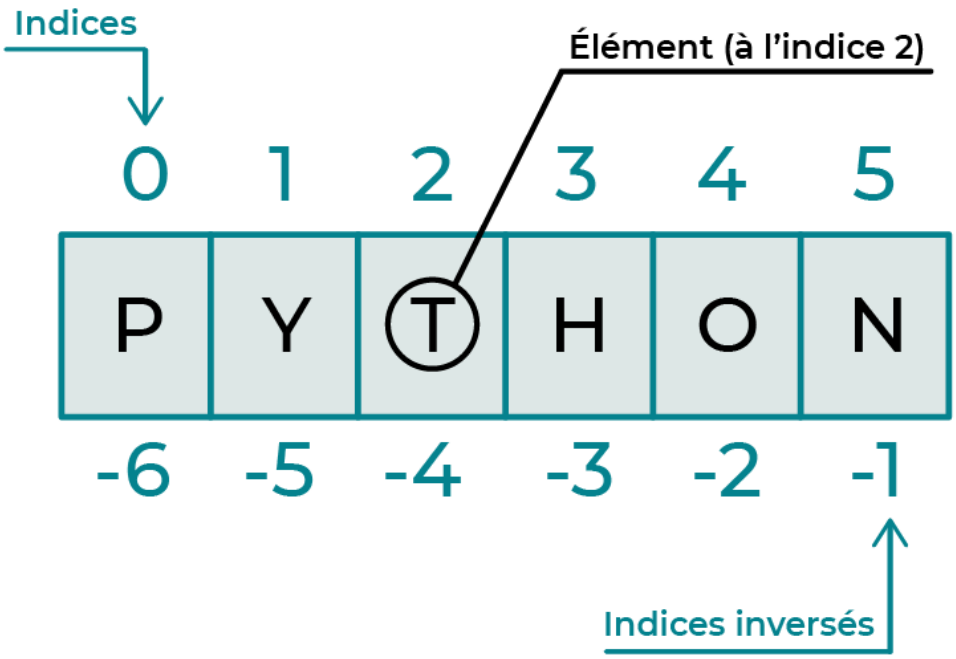
```
1 plateformes_sociales = ["Facebook", "Instagram", "Snapchat", "Twitter"]
```

2. Accès aux éléments d'une liste

Fondamental

Pour accéder aux éléments d'une liste, on utilise un indice. Chaque élément a un indice qui lui correspond, selon sa position dans la liste. Vous obtenez la valeur de cet indice avec la syntaxe suivante « liste[indice] ». Elle vous renverra la valeur de la liste qui est à la position de l'indice.

En faisant attention au fait que le **premier indice est repéré par un 0**.



Indices

Accédez aux caractères d'une chaîne comme un élément d'une liste *Remarque*

Les indices fonctionnent aussi avec les chaînes de caractères dans une variable. Chaque caractère correspond à un indice qui va de zéro à la longueur de la chaîne.

```
langage_de_programmation ="PYTHON"
langage_de_programmation[2]
"T"
langage_de_programmation[-4]
"T"
```

3. Modifier une liste

Méthode

On fait appel à une « méthode » pour ajouter, retirer ou trier des éléments, c'est à une méthode de liste.

Nous verrons les méthodes dans leur ensemble plus tard pour l'instant il s'agit simplement de réaliser quelques opérations de base.

Modifier une case d'une liste *Fondamental*

Pour modifier une case d'une liste, il suffit d'utiliser l'indice de la case que l'on souhaite modifier, et d'y affecter la nouvelle valeur à l'aide de l'opérateur d'affectation (=), tout comme pour une variable.

```
plateformes_sociales[2] = "LinkedIn"
print(plateformes_sociales)
["Facebook", "Instagram", "LinkedIn", "TikTok"]
```

Ajouter une plateforme de réseau social *Fondamental*

append()

```
plateformes_sociales.append("TikTok")
print(plateformes_sociales)
["Facebook", "Instagram", "Snapchat", "Twitter", "TikTok"]
```

Retirer un élément spécifique d'une liste *Fondamental*

remove()

```
plateformes_sociales.remove("Snapchat")
print(plateformes_sociales)
["Facebook", "Instagram", "Twitter", "TikTok"]
```

Connaître la longueur de la liste *Fondamental*

len()

```
print(len(plateformes_sociales))
4
```

Trier les éléments de la liste *Fondamental*

sort().

Le tri se fait sur alphabétiquement pour les listes de chaînes et numériquement pour les listes de nombres.

```
plateformes_sociales.sort()
```

Après avoir fait le tri il fau utiliser la commande print pour voir le résultat.

Il y a beaucoup d'autres méthodes que vous pouvez utiliser avec les listes. Vous les trouverez dans la **documentation Python**[12].

4. Tuples

Beaucoup des propriétés des tuples sont similaires à celles des listes. Par exemple, les listes et tuples utilisent tous deux les indices.

- La différence principale est que les tuples **ne peuvent pas être modifiés**
- Les listes par contre sont **modifiables**.

Fondamental

Au lieu des crochets [ ], ils se caractérisent par les parenthèses ( ) .

```
plateformes_sociales_tuple = ("Facebook", "Instagram", "TikTok", "Twitter")
```

Trouvez un élément

Fondamental

Pour savoir si un élément est présent dans une liste ou un tuple, on peut utiliser l'opérateur « in » . Cet opérateur retourne « True » si l'élément est présent dans la séquence sinon « False ».

```
nombres = [1,2,3,4,5]
5 in nombres
True
8 in nombres
False
```

5. Exercice : A faire Travailler avec des listes

```
liste_nombres = [1, 6, 98, 52, 1045, 3]

# 1) Classez liste_nombres du plus petit au plus grand avec la méthode sort !

# 2) Supprimez le premier élément de la liste grâce à la méthode pop. P.S.: pour supprimer un élément à un index donné, on écrit liste.pop(index)

# 3) Ajoutez le chiffre 1097 à la liste liste_nombres grâce à la méthode de liste append !

# 4) Récupérez le deuxième élément de la liste liste_nombres dans la variable deuxieme_element
deuxieme_element =
print(deuxieme_element) # la console devrait afficher "6" !

# 5) Affichez en une ligne la longueur de la liste liste_nombres après avoir supprimé le premier élement. Vous devriez obtenir 6 logiquement ! P.S.: pour calculer la longueur d'une liste, on écrit len(liste)
```

6. Exercice : Synthèse listes

Les listes

Dans Python, on utilise `[ ]` pour indiquer une liste

Opération sur les listes

La liste est numéroté de 0 à n de gauche à droite et de -1 à -n de droite à gauche

- Pour avoir accès à élément il faut indiquer son numéro renvoi le troisième élément de la liste
- Pour ajouter un élément on utilise plateformes_sociales("TikTok")
- Pour retirer un élément on utilise plateformes_sociales("TikTok")
- Pour connaître la longueur de la liste on utilise print((plateformes_sociales))
- Pour trier par ordre croissant on utilise plateformes_sociales.()

Il faut au final utiliser la commande pour voir le résultat.

Les tuples

La principale différence est que les tuples sont , alors que les listes sont .

Les conditions IV

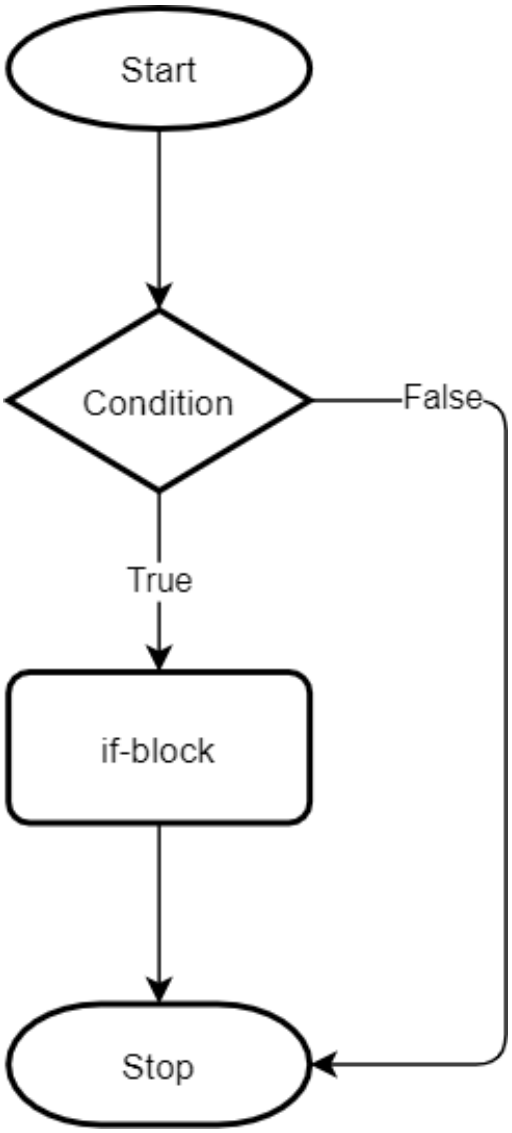
1. Conditions

Déroulement du programme

Certaines lignes d'un programme ne seront lues une fois seulement, d'autres plusieurs fois., autres encore pourraient être complètement ignorées, tout dépend de la façon dont elles sont codées.

Condition avec l'instruction if (si) *Fondamental*

Avec une instruction **if**, vous pouvez exécuter certaines lignes de code uniquement si une certaine condition est **vraie (True)**. Si cette condition est **fausse (False)**, le code ne s'exécutera pas.



Logigramme si

```
1 if ensoleille:
2     print("on va à la plage !")
```

La ligne 2 s'exécute uniquement si la condition `ensoleille` est **True**.

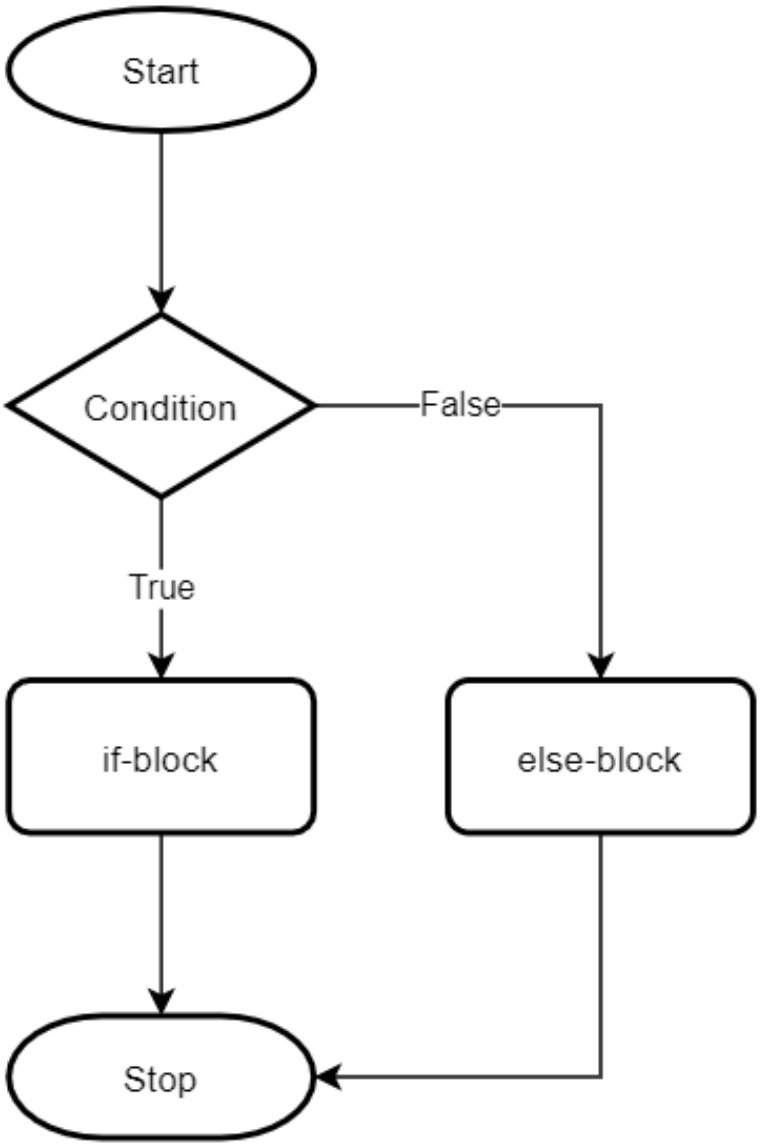
12. https://docs.python.org/fr/3/tutorial/datastructures.html

Conditions avec les instructions if else (si sinon) *Fondamental*

Quand est-ce que « else » intervient ?

Avec l'exemple du climat, allons plus loin pour y ajouter une instruction « else » :

- S'il fait beau dehors, je vais à la plage.
- Sinon (par exemple s'il pleut), je reste à la maison !



Logigramme si sinon

```
if mon_booleen:
   # exécuter le code quand mon_booleen est vrai
else:
   # exécuter le code quand mon_booleen est faux
```

Donc pour afficher une instruction sous une certaine condition, les étapes sont les suivantes :

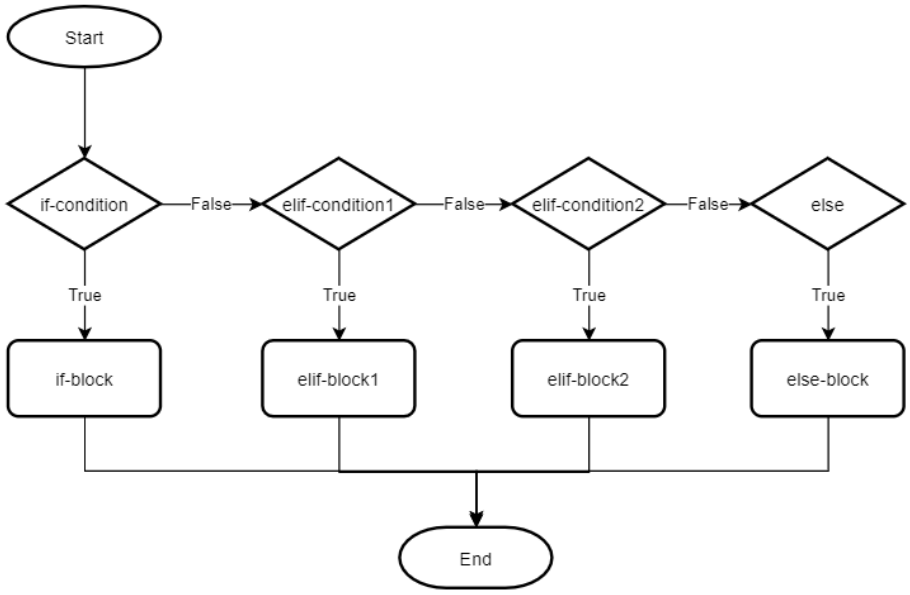
```
ensoleille = True

if ensoleille:
   print("on va à la plage !")
else:
   print("on reste à la maison !")
```

Dans cet extrait de code, puisque ensoleille est vrai, on va à la plage ! va s'afficher dans votre terminal. Si sa valeur était fausse, le code afficherait on reste à la maison !

2. Conditions alternatives

Conditions alternatives en ajoutant une clause elif (sinon) *Fondamental*

Les instructions **if/elif/else** vous permettent de définir des conditions multiples. Le mot-clé **elif** permet d'ajouter autant de conditions que vous voulez. Vous devez ensuite terminer avec une instruction **else**.



Logigramme if elif ese

Vous voulez aller à la plage s'il fait chaud dehors, et que vous voulez faire un bonhomme de neige s'il neige. Et sinon, vous restez à la maison.

```
ensoleille = False
neige = True

if ensoleille:
   print("on va à la plage !")
elif neige:
   print("on fait un bonhomme de neige")
else:
   print("on reste à la maison !")
```

Ce code va vérifier si ensoleille est vrai, et parce que c'est faux, il va vérifier si neige est vrai. Comme neige est vrai, le code va afficher faire un bonhomme de neige. Mais si neige était aussi faux, le programme aurait exécuté l'instruction finale **else** et affiché on reste à la maison !

3. Conditions multiples avec des opérateurs logiques

Opérateurs liés aux fonctions logiques

Pour vérifier plusieurs conditions pour un seul résultat dans la même instruction if, vous pouvez utiliser les opérateurs logiques :

- **and** vérifie si deux conditions sont toutes les deux vraies.
- **or** vérifie si au moins une condition est vraie.
- **not** vérifie si une condition n'est pas vraie (c'est-à-dire fausse).

Fondamental

Admettons que vous voulez aller à la plage seulement s'il y a du soleil et que c'est le weekend. Mais s'il y a du soleil et que nous sommes au milieu de la semaine, vous devez être au travail.

```
avec_soleil = True
en_semaine = False

if avec_soleil and not en_semaine:
   print("on va à la plage !")
elif avec_soleil and en_semaine:
   print("on va au travail !")
else:
   print("on reste à la maison !")
```

4. Conditions complexes avec expressions comparatives

Egalités ou inégalités

Les expressions comparatives vous permettent de comparer différentes expressions entre elles et d'évaluer si une expression est vraie ou fausse.

- Égal à ; a == b
- Différent ; a != b
- Inférieur ; a < b
- Inférieur ou égal à ; a <= b
- Supérieur à ; a > b
- Supérieur ou égal à ; a >= b

Fondamental

```
nombre_de_sieges = 30
nombre_dinvites = 25

if nombre_dinvites < nombre_de_sieges:
   # autoriser plus d'invités
else:
   # ne pas autoriser plus d'invités
```

5. Indentation

Fondamental

Après une instruction if, tout le code qui suit doit être indenté, c'est-à-dire qu'il doit être décalé par rapport à la marge. Sinon Python ne fait pas la différence entre l'instruction et le code à exécuter.

6. Exercice : A faire Contrôlez le déroulement de programmes

```
# 1) Comprenez le code ci-dessous
a = True
b = False
c = True

if a and b:
    x = 5
```

```
elif not c:
    x = 4
elif a:
    x = 8
else:
    x = 7

# 2) Déduisez-en la valeur de x et assignez la dans la variable nommée "solution"
solution =

# 3) Vérifiez votre intuition en faisant tourner le code (> "Run")
print(f"{solution} est la bonne valeur de x !" if solution == x else "Raté")

```

```
# 1) Comprenez le code ci-dessous
a = 3
b = 7
c = 5

vrai_ou_faux = (a < b or b != c) and c >= b

# 2) Déduisez-en la valeur de vrai_ou_faux et assignez la dans la variable nommée "solution"
solution =

# 3) Vérifiez votre intuition en faisant tourner le code (> "Run")
print(f"{solution} est la bonne valeur de vrai_ou_faux !" if solution == vrai_ou_faux else "Raté")

```

7. Exercice : Synthèse sur les conditions

Instruction if else

Avec une , vous pouvez exécuter certaines lignes de code uniquement si une certaine . Si cette , le code .

Clause elif

Les instructions vous permettent de définir . Le mot-clé vous permet d'ajouter autant de conditions que vous voulez. Vous devez ensuite terminer avec .

Conditions multiples avec opérateurs

Pour vérifier pour dans la , vous pouvez utiliser les opérateurs logiques :

- vérifie si deux conditions .
- vérifie si condition est vraie.
- vérifie si (c'est-à-dire fausse).

Expressions comparatives

- Égal à :
- Différent à :
- Inférieur à :

- Inférieur ou égal à :
- Supérieur :
- Supérieur ou égal à :

Les boucles V

1. Quand utiliser des boucles

Pour répéter un ensemble d'instructions ;

- Quand on sait combien de fois doit avoir lieu la répétition. On utilise dans ce cas la boucle **for** qui peut être combiné avec l'instruction **range**.
- Pour exécuter du code tant qu'une certaine condition est vérifiée. On utilise dans ce cas la boucle **while.**

2. Boucle for

for in *Fondamental*

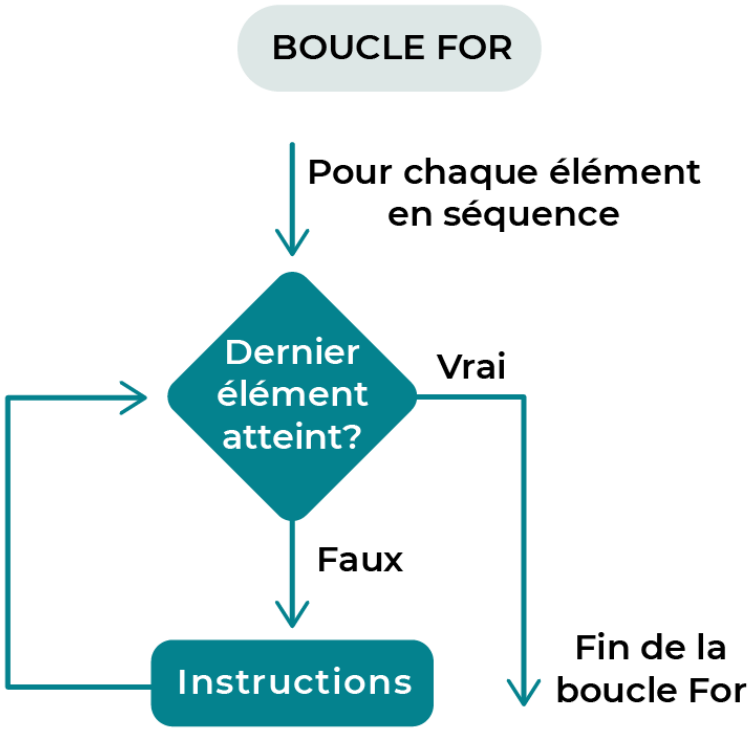
La boucle **for** vous permet d'exécuter du code un nombre spécifique de fois. Une boucle **for** est utilisée pour itérer sur une séquence. Ça peut être une liste, un tuple, un dictionnaire ou même une chaîne de caractères.

Si vous voulez afficher tous les éléments dans une liste, le code ressemblera à ça :

```
1 races_de_chien = ["golden retriever", "chihuahua", "terrier", "carlin"]
2 for chien in races_de_chien:
3     print(chien)
```

Dans ce code, chaque élément dans races_de_chien sera affiché dans le terminal.

- chien est un nom de variable qui se met à jour pour être l'élément suivant à chaque fois que la boucle se répète.



Logigramme for

Vous pouvez faire le même genre de boucle **for** si vous voulez passer en boucle chaque caractère d'une chaîne.

for range

Exemple

Pour boucler un certain nombre de fois, vous pouvez utiliser la fonction **range()**. Elle renverra une séquence de nombres qui vont de 0 à ce nombre moins un.

```
for x in range(5):
    print(x)
```

Ce code affichera 0, 1, 2, 3, 4 en séquence.

Exemple

```
for x in range(100):
    print(f"{x} bouteilles de bières au mur !")
```

Les accolades {} ci-dessus prennent n'importe quelle valeur dans la variable x et la remplace (n'oubliez pas le "f" au début de la donnée string qui signifie format). Le Le code afficher sera :

```
0 bouteilles de bières au mur !
1 bouteilles de bières au mur !
…
99 bouteilles de bières au mur !
```

Attention

La fonction **range()** est réglée sur 0 par défaut pour la valeur de début. Vous pouvez la modifier en ajoutant un autre nombre entier **range(4, 10)**. Cette plage renvoie les valeurs de 4 à 9 ; sans inclure 10.

3. Boucle while

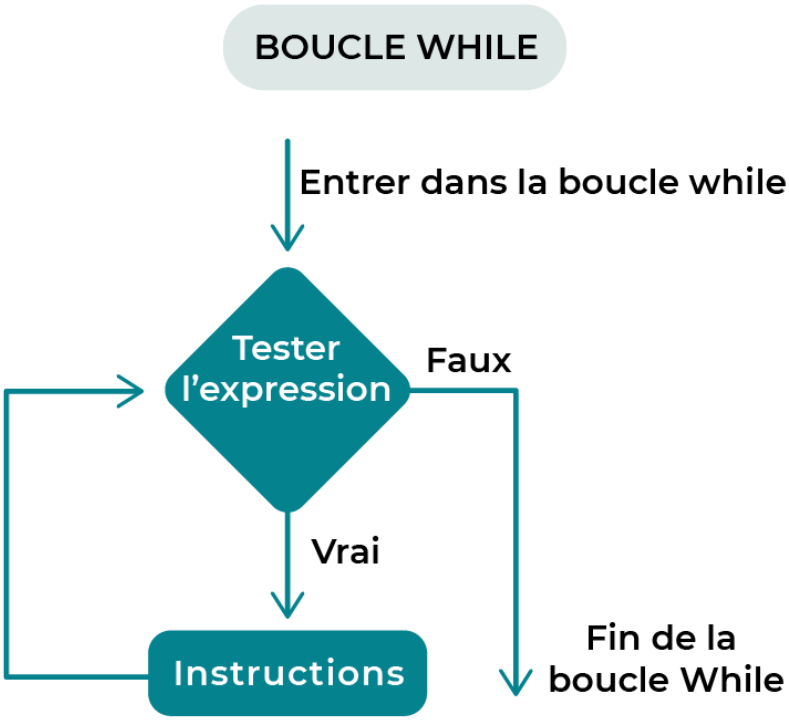
Fondamental

Le code dans l'instruction **while** s'exécute jusqu'à ce que la condition devienne fausse.

L'extrait de code ci-dessous vérifie la capacité actuelle et l'augmente d'une unité jusqu'à ce que la capacité maximale soit atteinte (+= 1 augmente la valeur actuelle de 1).

```
capacite_maximale = 10
capacite_actuelle = 3
while capacite_actuelle < capacite_maximale:
    capacite_actuelle += 1
```

Comme cette capacite_actuelle commence à 3, ce code s'exécute 7 fois jusqu'à ce que capacite_actuelle atteigne10.



Logigramme while

Les boucles infinies ***Attention***

Si la condition que vous avez réglée est toujours vraie, la boucle va s'exécuter pour toujour !

```
x = 0
while x != 5:
    x += 2
```

Dans cette situation, x n'atteindra jamais 5.

4. Exercice : A faire Des boucles

Répétez des tâches facilement à l'aide de boucles

```
"""
Le but de l'exercice est de calculer la somme des 100 premiers entiers
naturels !

Pour information :
vous êtes sur les pas du célèbre mathématicien Gauss
https://fr.wikipedia.org/wiki/Somme_(arithm%C3%A9tique)
"""

# 1) Utilisez une boucle et la fonction "range" pour calculer la somme.
# Testez et récupérez le résultat en faisant tourner le code (> "Run")

# 2) Assignez le résultat obtenu dans la variable "solution" pour vérification
solution =

# Ne touchez pas le print ci-dessous :)
print(f"{solution} est la bonne valeur de la somme !" if solution == (100 *
101) / 2 else "Raté")

```

5. Exercice : Synthèse sur les boucles

Boucle for

- Si la boucle [ ] la condition de [ ] elle est donc [ ] et [ ] de la boucle.
- Une boucle for permet de répéter du code [ ]

Boucle while

- Le code dans l'instruction while s'exécute jusqu'à ce que la [ ]
- Une boucle while permet de répéter du code jusqu'à ce qu' [ ]

Les fonctions VI

1. Les fonctions

Qu'est-ce qu'une fonction

Une fonction est un bloc de code avec un but spécifique et un nom pertinent. Quand vous appelez cette fonction, vous exécutez le code qu'elle contient. Les fonctions vous laissent saisir des paramètres pour exécuter le même code sur différentes valeurs. Il y a différents types de fonctions dans Python :

- Les fonctions intégrées fournies à Python. Par exemple la fonction range()
- Les fonctions définies par l'utilisateur que les développeurs créent.

Vous pouvez créer des fonctions avec ou sans paramètres d'entrée.

Définissez une fonction

Une fonction est un moyen de réutiliser un ensemble d'instructions répétables.

Fondamental

Vous la définissez :

- avec le mot-clé def
- le nom de la fonction
- des parenthèses
- deux-points

Si la fonction a besoin d'un paramètre ou plus, il faut les saisir à l'intérieur des parenthèses, séparés par des virgules.

```
1 def add(a, b):
2     return a + b
```

Lorsque la fonction est définie, vous pouvez l'appeler en y saisissant des valeurs en tant que paramètres. Ces valeurs sont appelées **arguments**. La fonction renverra les valeurs ajoutées ensemble.

```
1 add(1,3)
2 4
3 add(403,123)
4 526
```

Quand une valeur est renvoyée dans une fonction, vous pouvez la sauvegarder dans une variable.

```
1 total = add(1,4)
```

Quand utiliser les fonctions

Les fonctions sont un moyen de répéter des fonctionnalités et de séparer du code dans des petits modules différents, qui ne font qu'une chose à la fois.

2. Exercice : Synthèse sur les fonctions

Définition d'une fonction

On défini une fonction avec , le de la fonction, et

Pourquoi utiliser une fonction

- Les fonctions sont un moyen de répéter des fonctionnalités et de séparer du code dans des modules différents.
- Vous pouvez créer des fonctions avec ou sans paramètres d'entrée.
- Les fonctions sont (code écrit) **appelées** (code exécuté) et peuvent des informations (une valeur est donnée comme résultat).

Ecrivez le code correctement VII

1. Ne vous répétez pas

Le premier commandement de la programmation est le DRY : ne vous répétez pas (Don't Repeat Yourself en anglais).

Exemple

Admettons que vous voulez calculer le taux de conversion et l'afficher dans plusieurs campagnes. Vous pouvez le copier et coller pour chaque campagne comme dans le code ci-dessous :

```
taux_de_conversion1 = premiere_campagne['visiteurs'] /
premiere_campagne['conversions']
taux_de_conversion2 = seconde_campagne['visiteurs'] /
seconde_campagne['conversions']
taux_de_conversion3 = troisieme_campagne['visiteurs'] /
troisieme_campagne['conversions']
```

Si vous commencez à écrire du code comme ça, la première chose à faire est de le transformer en fonction. Vous pouvez écrire une fonction :

```
def calculer_taux_de_conversion(campagne):
    taux_de_conversion = campagne['visiteurs'] / campagne['conversions']
    return taux_de_conversion

calculer_taux_de_conversion(premiere_campagne)
calculer_taux_de_conversion(seconde_campagne)
calculer_taux_de_conversion(troisieme_campagne)
```

2. Le principe de responsabilité unique

Quand vous organisez quelque chose, il vaut mieux avoir un endroit où ranger chaque objet, car vous savez qu'il sera plus facile à retrouver. C'est pareil pour écrire des fonctions.

Chaque fonction doit être responsable d'une seule et unique fonctionnalité, et rien de plus.

3. Commentez votre code

Les commentaires sont un moyen très pratique de rappeler, à vous et aux autres développeurs, ce qu'un morceau de code est censé faire.

Fondamental

Les commentaires à ligne unique sont indiqués par #

```
#ceci est un commentaire
```

Les commentaires à lignes multiples sont entourés de trois guillemets """

```
"""
Ceci est un
commentaire à lignes multiples
"""
```

4. Respectez les standards du code

En général, vous lisez du code bien plus souvent que vous n'en écrivez. C'est pour ça que c'est important d'avoir des lignes directrices pour améliorer sa lisibilité et le rendre aussi cohérent que possible.

Le guide de style officiel pour le code Python est un document appelé PEP8[13]. Il contient les règles et les bonnes pratiques qui à standardisent l'écriture en Python. Cette approche est synthétisée le document appelé PEP 20

- Explicite est meilleur qu'implicite.
- Simple est mieux que complexe.
- Les cas spéciaux ne sont pas assez spéciaux pour enfreindre les règles.
- L'l'ambiguïté n'est pas de mise, refusez la tentation de deviner.
- Maintenant, c'est mieux que jamais.
- Si la mise en œuvre est difficile à expliquer, c'est une mauvaise idée.
- Si la mise en œuvre est facile à expliquer, cela peut être une bonne idée.
- ...

5. Exercice : Synthèse écriture

Ne vous répétez pas

Utilisez des

Une chose à la fois

Chaque fonction doit être responsable , et rien de plus.

Commenter!

- commentaire à , il faut mettre en début de ligne :
- commentaires à sont entourés de trois guillemets :

Les packages VIII

1. Package

Un package ou bibliothèque en français est une collection de fonctions qui peuvent être ajoutées au code Python. **VOIR VSC**

Installez les packages avec Pip

Pour installer un package avec pip dans votre terminal, utilisez la méthode suivante

```
pip install <nom-du-package>
```

Pour voir les packages déjà installés, vous pouvez écrire le code qui suit :

```
pip freeze
```

Il va afficher une liste de tous les packages existants, qu'on appelle dépendances, dans votre terminal.

Importez les packages dans votre code

Après l'installation d'un paquet il faut l'importer, avec la commande import tout en haut de votre fichier de code.

```
import <nom-du-package>
```

Appel d'une fonction d'un package

```
nom-du-package.nom-fonction
```

Et pour importer une seule fonction appelée à partir d'un package :

```
from <package> import <fonction>
```

Vous n'avez plus besoin d'écrire *requests* avant fonction() parce que la méthode elle-même a déjà été importée.

13. https://peps.python.org/pep-0008/