Aide-Mémoire Python du Cours

Ce document regroupe les concepts de programmation, les fonctions intégrées et les bibliothèques que nous avons abordés dans ce cours.

1. Variables, Types et Structures de Données

Les variables stockent des informations. Le type de la donnée modifie les opérations que l'on peut effectuer.

- `int` (entier) : Nombres entiers (ex. `10`, `-3`).
- `float` (flottant) : Nombres à virgule (ex. `3.14`, `-0.5`).
- `str` (chaîne de caractères) : Texte placé entre guillemets ou apostrophes (ex. `"Bonjour"`, `'A'`).
  - **f-strings** (chaînes formatées) : `f"Valeur : {x}"` évalue la variable `x` directement dans la chaîne.
  - Méthodes utiles : `.startswith(prefix)`, `.split(sep)` (découpe une chaîne en liste).
- `list` (liste) : Collection ordonnée et modifiable d'éléments (ex. `[1, 2, 3]`).
  - `.append(element)` : Ajoute un élément à la fin de la liste.
- `tuple` (tuple) : Collection ordonnée mais **non modifiable** d'éléments (ex. `(1, 2)`).

Slicing (Découpage) : Permet d'extraire une partie d'une chaîne, d'une liste ou d'un tableau NumPy.

- Syntaxe : `collection[début:fin:pas]` (la `fin` est exclue).
- Exemples : `liste[0:3]` (3 premiers éléments), `liste[::-1]` (inverser la liste).

Type et Conversion :

- `type(obj)` : Retourne le type de l'objet.
- `int(x)`, `float(x)`, `str(x)`, `list(x)` : Convertissent `x` vers le type spécifié.

---

2. Opérateurs Mathématiques

- `+` (Addition), `-` (Soustraction)
- `*` (Multiplication), `/` (Division flottante)
- `//` (Division entière, ex. `5//2` donne `2`)
- `**` (Puissance, ex. `2**3` donne `8`)
- `%` (Modulo, reste de la division entière, ex. `5%2` donne `1`)

(Note : Pour les opérations avancées sur des séries de données, nous utilisons systématiquement NumPy plutôt que le module `math` de base).

---

3. Structures de Contrôle

Les structures de contrôle gèrent le flux d'exécution du code : conditions et boucles.

Conditions

Elles permettent d'exécuter du code différemment selon qu'un test est Vrai ou Faux.

```
if condition:
    # exécuté si la condition est Vraie
elif autre_condition:
    # exécuté si la première est Fausse et celle-ci est Vraie
else:
    # exécuté si aucune des conditions n'est Vraie
```

Boucles

- `for` (boucle pour) : Parcourt une collection (liste, tableau) ou un nombre d'itérations connu.
  - `range(début, fin, pas)` : Génère une séquence d'entiers. Idéal avec `for` (ex. `range(5)` donne 0 à 4).
  - `enumerate(iterable)` : Parcourt une collection en renvoyant à la fois l'**indice** et la **valeur** (`for index, val in enumerate(liste):`).
- `while` (boucle tant que) : Répète un bloc de code tant qu'une condition reste Vraie.

---

4. Fonctions Personnalisées

Créer ses propres fonctions avec le mot-clé `def`.

```
def ma_fonction(arg1, arg2):
    """
    Docstring (Chaîne de documentation) :
    Explique ce que fait la fonction, ses arguments et sa valeur de retour.
    """
    resultat = arg1 + arg2
    return resultat
```

---

5. Fonctions Intégrées (Built-ins)

- `print(valeur, ...)` : Affiche les valeurs dans la console.
- `len(obj)` : Renvoie le nombre d'éléments d'une liste, d'un tableau, d'une chaîne, etc.

- `sum(obj)`, `min(obj)`, `max(obj)` : Somme, minimum et maximum d'une collection. *(En pratique, on privilégie `np.sum()`, `np.min()`, `np.max()` avec NumPy).*
- `abs(x)` : Renvoie la valeur absolue.
- `round(nombre, ndigits)` : Arrondit `nombre` avec `ndigits` chiffres après la virgule.
- `repr(obj)` : Renvoie une représentation textuelle (souvent détaillée) d'un objet.
- `open(fichier, mode)` : Ouvre un fichier (ex. `mode="r"` pour lecture).

---

6. Bibliothèques Externes (Modules)

Les bibliothèques nécessitent un `import` avant d'être utilisées.

NumPy (`import numpy as np`)

Bibliothèque de référence pour la manipulation de tableaux (arrays) et le calcul numérique.

Création et Manipulation de Tableaux :

- `np.array(liste)` : Convertit une liste Python en tableau NumPy.
- `np.arange(start, stop, step)` : Crée un tableau de nombres avec un pas constant.
- `np.linspace(start, stop, num)` : Crée un tableau de `num` valeurs réparties uniformément entre `start` et `stop`.
- `np.zeros(shape)` / `np.ones(shape)` : Crée un tableau rempli de zéros ou de uns avec les dimensions `shape` (ex. `(3, 3)`).
- `np.ones_like(a)` : Crée un tableau de uns avec la même forme (dimensions) que le tableau `a`.
- `np.meshgrid(x, y)` : Génère des grilles de coordonnées N-D depuis des vecteurs 1D (essentiel pour les graphiques 3D).
- `np.column_stack(tup)` / `np.concatenate(tup, axis)` : Assemble des tableaux (en colonnes, ou bout à bout).
- `np.where(condition, x, y)` : Retourne les éléments choisis dans `x` ou `y` (ou leurs indices) en fonction de la `condition`.

Algèbre Linéaire et Matrices :

- `np.diag(v)` : Extrait la diagonale d'un tableau ou construit une matrice diagonale.
- `np.trace(a)` : Calcule la somme de la diagonale d'une matrice.
- `np.linalg.norm(x)` : Calcule la norme (longueur) d'un vecteur ou d'une matrice.
- `np.linalg.solve(A, b)` : Résout de manière exacte le système d'équations linéaires matricielles $Ax = b$.

- `np.linalg.lstsq(a, b)` : Donne la solution des moindres carrés pour $ax = b$ (régression linéaire).
- `np.linalg.cond(x)` : Calcule le conditionnement d'une matrice (sa sensibilité aux erreurs numériques).

Statistiques, Maths et Ajustements :

- `np.mean(a)`, `np.std(a)`, `np.cov(m)` : Calculent respectivement la moyenne, l'écart-type et la matrice de covariance.
- `np.sum(a)`, `np.min(a)`, `np.max(a)`, `np.argmax(a)` : Opérations globales (somme, min, max, indice du max).
- `np.exp(x)`, `np.log(x)`, `np.log10(x)`, `np.sqrt(x)`, `np.round(a)` : Fonctions mathématiques sur tableaux complets.
- `np.polyfit(x, y, deg)` : Régression polynomiale ; retourne les coefficients (ex. `deg=1` pour une droite).

Précision, Différences et Mémoire :

- `np.allclose(a, b, rtol, atol)` : Vérifie si les éléments de deux tableaux sont "égaux" (très proches), palliant les imprécisions des flottants.
- `np.shares_memory(a, b)` : Vérifie si deux tableaux pointent vers les mêmes données en RAM.
- `np.float32` / `np.finfo(dtype)` : Type de nombre flottant simple précision, et fonction pour inspecter ses limites numériques (epsilon, min, max).

Aléatoire (`np.random`) :

- `np.random.seed(seed)` : Définit la graine de l'aléatoire pour rendre les résultats reproductibles.
- `np.random.uniform(low, high, size)` : Génère des nombres aléatoires selon une distribution uniforme.
- `np.random.rand(d0, d1, ...)` / `np.random.randn(...)` : Distributions uniformes [0,1) ou normales (gaussiennes).

Lecture de Fichiers :

- `np.loadtxt(fname, delimiter, skiprows)` : Charge des données depuis un fichier texte (comme un CSV).
  - `delimiter=','` : Indique le séparateur de colonnes (souvent la virgule).
  - `skiprows=1` : Essentiel pour ignorer la ou les premières lignes (ex: en-têtes de colonnes textuels).

Matplotlib (`import matplotlib.pyplot as plt`)

Bibliothèque standard pour la création de graphiques et la visualisation de données.

Création et Structure :

- `plt.figure(figsize=(l, h))` : Initialise une figure avec une taille spécifique.
- `plt.subplots(nrows, ncols, figsize)` : Crée à la fois la figure principale et plusieurs sous-graphiques ("Axes") selon une grille.

Types de Graphiques 2D :

- `plt.plot(x, y, color, label, linestyle, marker)` : Trace une courbe reliant les points. Arguments utiles pour l'apparence et la légende.
- `plt.scatter(x, y, c, s, marker)` : Trace un nuage de points sans les relier. `c` pour la couleur, `s` pour la taille.
- `plt.loglog(x, y)` : Échelle logarithmique sur les DEUX axes (X et Y).
- `plt.semilogx(x, y)` (ou `semilogy`) : Échelle logarithmique sur un seul axe.
- `plt.axvline(x, color, linestyle)` / `plt.axhline(y)` : Trace une ligne d'une extrémité à l'autre du graphique, verticale ou horizontale.
- `plt.contourf(X, Y, Z, cmap)` : Dessine des courbes de niveau remplies (cartes de chaleur 2D). `cmap` définit la palette de couleurs.
- `plt.pcolormesh(X, Y, Z, cmap, shading)` : Grille colorée pseudo-couleur (une alternative plus rapide à `contourf`).

Configuration du Graphique :

- `plt.title(label)`, `plt.xlabel(xlabel)`, `plt.ylabel(ylabel)` : Ajout du titre et des légendes d'axes.
  - *Note:* Sur un objet Axe (via `subplots`), on utilise `ax.set_title()`, `ax.set_xlabel()`, etc.
- `plt.grid(True)` : Affiche la grille de repère.
- `plt.legend()` : Génère la légende (fonctionne si `label="..."` a été défini lors du traçage).
- `plt.colorbar()` : Ajoute une échelle de barres de couleurs au bord d'un `contourf` ou `pcolormesh`.
- `plt.annotate(text, xy)` : Ajoute une annotation textuelle pointant vers des coordonnées `xy` spécifiques.
- `plt.tight_layout()` : Ajuste automatiquement l'espace de la fenêtre pour éviter que les éléments ne débordent.
- `plt.show()` : Affiche physiquement la fenêtre graphique.

Graphiques 3D (`mpl_toolkits.mplot3d`) :

- `fig.add_subplot(111, projection='3d')` : Crée un repère tridimensionnel (Objet axe 3D).
- `ax.plot_surface(X, Y, Z, cmap)` : Trace une surface lisse en 3 dimensions à partir de grilles de coordonnées.

Autres Modules Standards

- `time` :

  - `time.perf_counter()` : Horloge avec la plus haute précision disponible (idéal pour mesurer précisément le temps d'exécution d'un bloc de code).
- `os` / `sys` :
  - `os.chdir(path)` : Change le dossier de travail actuel (Current Working Directory). Utile si les fichiers de données ne sont pas trouvés.