

TP2 - Premiers pas en Python

Etienne Dagorn*

Table des matières

1	Rappels sur Python et le terminal	2
	Rappels sur Python et le terminal	2
1.1	Comprendre l'arborescence des dossiers	2
1.2	Interpréteur Python vs scripts	2
2	Exercices – Manipulation de listes et dictionnaires	4
3	Cheatsheet Python & Terminal	11

*etienne.dagorn@univ-lille.fr

1 Rappels sur Python et le terminal

1.1 Comprendre l'arborescence des dossiers

Un ordinateur organise tous ses fichiers (documents, images, scripts Python, etc.) dans une **arborescence de dossiers**, un peu comme des **cartons rangés dans des boîtes et des sous-boîtes**.

- Au sommet, il y a un **dossier racine** :
 - Windows : souvent C:\;
 - Mac/Linux : / (appelé *à* root *à*).
- À l'intérieur se trouvent des **sous-dossiers**, chacun pouvant contenir des fichiers (.txt, .py, images) ou d'autres sous-dossiers. **Exemple d'arborescence** :

```
Documents
|
+- Cours_Python
  +- TP1
    +- script1.py
  +- TP2
    +- data.csv
  +- TP3
    +- notes.txt
+- Images
  +- photo.png

Images photo.png
```

Idée clé

Chaque fichier possède un **chemin** qui indique son emplacement, par exemple : Documents/TP3/notes.txt. Le terminal et Python ont toujours besoin de savoir dans quel dossier ils *à* travaillent *à* : c'est le **dossier courant**.

1.2 Interpréteur Python vs scripts

Avant de poursuivre, faisons un point sur le **terminal** (ou *invite de commandes* / *shell*) et son rôle dans l'exécution de Python.

1. Qu'est-ce que le terminal ?

- Le terminal est une **interface texte** qui permet de communiquer avec le système d'exploitation.
- On y tape des **commandes** que le système exécute : par exemple `pwd` pour connaître le dossier courant, `ls` (Linux/Mac) ou `dir` (Windows) pour lister les fichiers.
- C'est l'outil *à* derrière *à* Visual Studio Code, Jupyter ou Spyder : ces IDE appellent Python mais le système d'exploitation passe toujours par le shell.

2. Lancer Python

Deux façons principales :

1. **Mode interactif** : dans le terminal, taper `python` ou `python3` $\Rightarrow$ l'invite change (`>>>`) et on peut tester des lignes de code une par une.
2. **Exécuter un script** : taper `python mon_script.py` $\Rightarrow$ Python lit tout le fichier et l'exécute.

Astuce : pour quitter le mode interactif, taper `exit()` ou `Ctrl+D` (Mac/Linux) ou `Ctrl+Z` puis Entrée (Windows).

3. Dossiers et chemins

- Chaque terminal est “positionné” dans un **dossier courant** (ou *working directory*), visible avec `pwd`.
- Pour changer de dossier : `cd nom_du_dossier`; revenir en arrière : `cd ...`
- Lorsque Python ouvre ou enregistre un fichier (ex : `open("data.txt")`), il cherche d'abord dans le dossier courant ; d'où l'importance de se placer au bon endroit.

4. Commandes utiles à connaître

<code>pwd</code>	affiche le chemin du dossier courant
<code>ls</code> ou <code>dir</code>	liste les fichiers présents
<code>cd Dossier</code>	change de dossier courant
<code>python script.py</code>	exécute le fichier Python
<code>exit()</code>	quitte l'interpréteur Python

À retenir

Le terminal est le **point d'entrée du système** ; Python est un programme que l'on appelle via le terminal, même si les IDE cachent cette étape. Comprendre où l'on se trouve (chemin/dossier) est essentiel pour lire et écrire des fichiers correctement.

2 Exercices – Manipulation de listes et dictionnaires

Exercice 1 – Création d’une liste

Crée une liste contenant les noms de 5 étudiant.e.s, puis :

- Affiche le premier et le dernier élément.
- Ajoute un nouvel étudiant à la liste.
- Supprime le deuxième étudiant.
- Trie la liste par ordre alphabétique.
- Affiche la longueur de la liste et l’index de “Alice” si elle est présente.

Aide

Pense aux indices 0 et -1. Mots-clés utiles : `append()`, `pop()`, `sort()`, `len()`, opérateur `in`, `index()`.

Bonus léger – Aide

Parcours la liste d’origine et ajoute un élément dans une *nouvelle* liste seulement s’il n’y est pas déjà (`if x not in ...`).

Exercice 2 – Moyenne des notes

Crée une liste de notes (ex : `[12, 14, 10, 8]`), puis :

- Calcule la moyenne (arrondie à 1 décimale).
- Affiche les notes strictement supérieures à la moyenne.
- Calcule la note minimale et maximale.
- Calcule l’écart-type (*sans bibliothèque*).

Aide

Mots-clés : `sum()`, `len()`, `round()`, compréhension de liste, `min()`, `max()`.

Pour l’écart-type (population) : moyenne m , puis variance $= \frac{1}{n} \sum (x - m)^2$, et racine carrée (`** 0.5`).

Pense à gérer la *liste vide* avant toute division.

Bonus – Aide

Teste `if len(L) == 0` avant de calculer la moyenne ; affiche un message clair au lieu d’une division par zéro.

Exercice 3 – Noms en majuscules

Écris une boucle qui parcourt une liste de noms et affiche chaque nom en majuscules.

- Crée une *nouvelle* liste `noms_maj`.
- Affiche uniquement les prénoms commençant par la lettre A (insensible à la casse).

- Donne la longueur de chaque prénom ; quel est le plus long ?

Aide

Mots-clés : `.upper()`, `.lower()`, `.startswith()`, `len()`, compréhension de liste, `max(liste, key=len)` (ou boucle avec comparaison).

Exercice 4 – Liste de carrés

Crée une liste contenant les carrés des nombres de 1 à 10, en utilisant une boucle **ou** une compréhension de liste.

- Ne garde que les carrés pairs.
- Calcule la somme des carrés.

Aide

Mots-clés : `range()`, puissance `**2`, filtre avec `x % 2 == 0`, `sum()`.

Deux voies possibles : boucle `for` + `append()`, ou compréhension de liste.

Bonus – Aide

Crée une liste de tuples `(n, n**2)` via une compréhension : `[(n, n**2) for n in ...]`.

Exercice 5 – Dictionnaire simple

Crée un dictionnaire représentant un étudiant :

```
etudiant = {"nom": "Alice", "age": 22, "note": 14}
```

Puis :

- Affiche le nom et la note.
- Modifie la note (+1 point).
- Ajoute une clé `"present"` avec la valeur `True`.
- Utilise `get()` pour lire une clé absente avec une valeur par défaut.

Aide

Accès par clé : `d["cle"]`. Mise à jour : `d["note"] = ...`. Ajouter une clé : affectation simple. `d.get("cle", "default")` évite l'erreur si la clé n'existe pas.

Bonus – Aide

`pop("note")` pour récupérer et supprimer la note, puis créer `"notes"` en liste et calculer la moyenne avec `sum()/len()`.

Exercice 6 – Liste de dictionnaires

Crée une liste de 3 étudiant.e.s (dictionnaires avec `nom` et `notes` = liste).

- Pour chaque étudiant, calcule la moyenne.
- Affiche le classement décroissant par moyenne.

— En cas d'égalité, classe par ordre alphabétique du nom.

Aide

Calcule la moyenne avec `sum()/len()` et ajoute-la comme nouvelle clé.

Tri : `sorted(liste, key=...)` avec une `lambda` qui renvoie un *tuple* comme clé, par ex. `(-moyenne, nom)`.

Bonus – Aide

Écris une fonction `ajouter_etudiant(...)` qui nettoie le nom (`strip()`, `title()`) et convertit les notes en float.

Exercice 7 – Comptage de mots

Demande une phrase à l'utilisateur, puis :

- Normalise en minuscules, enlève `,. ; : ! ?`.
- Compte les occurrences de chaque mot dans un dictionnaire.
- Affiche les **3** mots les plus fréquents.

Aide

`.lower()` pour normaliser; `.replace()` pour ôter la ponctuation; `.split()` pour découper.

Dico de fréquences : `d[mot] = d.get(mot, 0) + 1`.

Top 3 : `sorted(d.items(), key=lambda kv: -kv[1])[0:3]`.

Bonus – Aide

Avant de compter, *ignore* les mots très fréquents via un `set` de “stopwords” et un `if mot not in stopwords`.

3 Cheatsheet Python & Terminal

Terminal – Commandes essentielles

- `pwd` – affiche le **chemin du dossier courant** (où l'on se trouve).
- `ls` – liste le contenu du dossier courant (Mac/Linux).
- `dir` – liste le contenu du dossier courant (Windows).
- `cd <nom_dossier>` – **changer de dossier**.
- `cd ..` – remonter d'un dossier.
- `cd /` – aller à la racine du système.
- `mkdir <nom_dossier>` – créer un nouveau dossier.
- `rm <nom_fichier>` – supprimer un fichier (irréversible).
- `rm -r <nom_dossier>` – supprimer un dossier et son contenu.

Astuce : Python exécute le code dans le **dossier courant**. Si votre script ne trouve pas un fichier, vérifiez que vous êtes dans le bon dossier avec `pwd`.

Python – Listes

- Créer : `L = ["Alice", "Bob", "Clara"]`
- Accès : `L[0]` (premier), `L[-1]` (dernier)
- Ajouter : `L.append("Farid")`
- Retirer & renvoyer : `retire = L.pop(1)`
- Trier (in-place) : `L.sort()`
- Longueur : `len(L)`
- Test appartenance : `"Alice" in L`
- Compréhension : `carres = [n**2 for n in range(1,11)]`

Python – Dictionnaires

- Créer : `d = {"nom": "Alice", "note": 14}`
- Accès : `d["note"]` (KeyError si absente)
- Ajouter/modifier : `d["age"] = 22`
- Supprimer+renvoyer : `d.pop("note")`
- Lecture sûre : `d.get("ville", "inconnue")`
- Itérations : `d.keys()`, `d.values()`, `d.items()`
- Test clé : `"note" in d`

Boucles & Conditions

- `for x in it: ...` `for i,x in enumerate(L, start=1): ...`
- `for i in range(5): (0..4)`
- `while condition: ...`
- `if ...: elif ...: else:`
- `break` (sortir) | `continue` (sauter)
- Ternaire : `a if cond else b`
- Vide? `if L:` (False si liste vide)

Affichage & Chaînes

- `print(obj, ...)`
`print("\n"+"="*72)`
- f-strings : `f"Note = {note}"`
- `s.upper()`, `s.lower()`, `s.capitalize()`, `s.title()`
- `s.strip()` (espaces), `s.replace(",", " ")`, `s.split()`
- `ch.isalpha()` (lettre?)

Sommes

- `sum(L)`, `min(L)`, `max(L)`
- Générateur : `sum(v["prix"]*v["stock"] for v in inventaire.values())`
- `round(x, 2)` Puissance `n**2` Modulo `x % 2 == 0`

Tri & clés de tri

- `L.sort()` (in-place) vs `sorted(L)` (nouvelle liste)
- Tri simple : `sorted(d.items(), key=lambda kv: kv[1], reverse=True)`
- Multi-critères : `sorted(eleves, key=lambda d: (-d["moyenne"], d["nom"]))`
- Top-*k* : `sorted(...)[ : k]`

Ensembles (stopwords)

- Définir : `stop = {"le", "la", "et", "c'est"}`
- Test rapide : `if mot in stop: continue`

Fonctions utilitaires

→ Définir :

```
— def moyenne(L):
    return sum(L)/len(L) if
    len(L)>0 else None

— def mediane(L):
    T=sorted(L);          n=len(T);
    mid=n//2;
    return (None if n==0 else
    T[mid] if n%2 else (T[mid-
    1]+T[mid])/2)

— def mode(L):
    if not L: return None
    c={};
    for x in L: c[x]=c.get(x,0)+1
    f=max(c.values()); M=[k for k,v
    in c.items() if v==f]
    return M[0] if len(M)==1 else M
```

Patrons fréquents du TP

```
→ Inventaire (prix & stock) :
    inventaire["riz"]["prix"] *= 0.9

→ Validation-then-apply :
    ok=True; for prod,q in
    cmd.items():
        if prod not in inv or
        inv[prod]["stock"]<q: ok=False;
        break
    if ok: for prod,q in cmd.items():
        inv[prod]["stock"] -= q

→ Compteur de mots :
    compte={}; for m in mots:
        compte[m]=compte.get(m,0)+1

→ Classement dict par valeurs :
    sorted(d.items(), key=lambda kv:
    -kv[1])
```

Conseils pratiques

- ✓ Sauvegardez souvent votre fichier .py.
- ✓ Privilégiez des noms explicites : prix_moyen, compte_mots.
- ✓ Vérifiez le dossier courant avant lectures/écritures.
- ✓ Commentez avec # ; testez par petites étapes.