FICHE-OUTIL · O5

Python : les bases

Le programme de physique-chimie demande de savoir **compléter et interpréter** de courts scripts Python (les « capacités numériques »). Cette fiche donne le minimum vital, avec des exemples de physique-chimie. Nous conseillons d'installer **EduPython** et d'y **exécuter puis modifier** les exemples et les scripts du livre : on n'apprend pas un langage en le lisant. Le tracé de graphes est traité dans la fiche **M2**.

---

1. Installer et utiliser EduPython

EduPython réunit dans un même logiciel l'interpréteur Python, l'éditeur et les bibliothèques utiles au lycée. Les indications ci-dessous concernent **Windows**.

1.1 Installation

1. Aller sur le site officiel : https://edupython.free.fr/.
2. Dans la rubrique **Téléchargement**, choisir la dernière version et lancer le fichier d'installation.
3. Suivre les choix proposés par défaut, puis ouvrir EduPython.

Si Windows affiche « Windows a protégé votre ordinateur », vérifier d'abord que le fichier vient bien du site officiel. Cliquer ensuite sur **Informations complémentaires**, puis sur **Exécuter quand même**.

Cap bac — l'indispensable

EduPython est conçu pour Windows. Sur macOS ou Linux, les scripts du livre restent des fichiers Python ordinaires (`.py`) : ils peuvent être ouverts dans un autre environnement Python, mais les menus et boutons ne seront pas exactement les mêmes.

1.2 Ouvrir, modifier et exécuter un script

1. Lancer EduPython, puis choisir **Fichier > Ouvrir** et sélectionner le fichier `.py`.
2. Choisir **Fichier > Enregistrer sous** et ajouter `_mon_travail` au nom du fichier. L'original restera disponible en cas de fausse manipulation.
3. Repérer les lignes signalées par `# À COMPLÉTER` et ne modifier d'abord que celles-ci.
4. Exécuter le script avec la **flèche verte**, par le menu **Exécuter**, ou avec la touche `F5`.
5. Observer le résultat dans la console ou dans la fenêtre graphique. Après une modification, enregistrer puis exécuter à nouveau.

En cas d'erreur, lire la **dernière ligne** du message et repérer le numéro de ligne indiqué. Vérifier en priorité les parenthèses, les deux-points, l'indentation, le point décimal et l'orthographe des variables.

1.3 Télécharger les scripts

1. Ouvrir la page des scripts du chapitre à l'aide de l'adresse courte ou du QR code indiqué dans l'activité Python.
2. Télécharger le fichier `.py` demandé.
3. Ranger les fichiers dans un dossier facile à retrouver, par exemple `C:/Bac/scripts`. Éviter les accents dans les noms de dossiers et de fichiers.
4. Conserver le fichier original et travailler sur une copie.

2. Variables

Une **variable** est une boîte nommée qui contient une valeur. Le signe `=` **affecte** une valeur à une variable (ce n'est *pas* le « égal » mathématique : c'est un ordre, « range cette valeur dans cette boîte »).

```
m = 36.0       # masse d'eau (g)
M = 18.0       # masse molaire de l'eau (g/mol)
n = m / M      # quantité de matière (mol)
print(n)       # affiche : 2.0
```

- Le séparateur décimal est le **point** : `18.0`, jamais `18,0`.
- Tout ce qui suit `#` est un **commentaire**, ignoré par Python.
- `print(...)` affiche une valeur à l'écran.
- Une variable peut être **mise à jour** : après `n = n + 1.0`, la boîte `n` contient son ancienne valeur augmentée de 1.
- L'**écriture scientifique** se manipule comme dans Excel : `2.4E8` correspond au nombre $2{,}4 \times 10^{8}$.

3. Opérateurs

Les opérations s'écrivent avec `+`, `-`, `*` (multiplication, **obligatoire** : `c * V`, pas `cV`), `/` et `**` (puissance).

```
pH = 3.0
h = 10**(-pH)      # [H3O+] = 10^(-pH), en mol/L
print(h)           # affiche : 0.001
```

Les **comparaisons** s'écrivent `<`, `<=`, `>`, `>=`, `==` (égal ?) et `!=` (différent ?). Leur résultat est `True` (vrai) ou `False` (faux) ; elles servent dans les conditions et les boucles (§5 et §7). Attention : `=` affecte, `==` compare.

4. Listes

Une **liste** rassemble plusieurs valeurs dans l'ordre, entre crochets et séparées par des virgules :

```
volumes = [5.0, 10.0, 15.0, 20.0]   # volumes versés (mL)
print(volumes[0])    # affiche : 5.0  (le premier indice est 0 !)
print(volumes[3])    # affiche : 20.0
print(len(volumes))  # affiche : 4   (nombre d'éléments)
```

Très souvent, on **construit** une liste en partant d'une liste **vide**, puis en ajoutant les éléments un à un avec `.append(...)` :

```
mesures = []            # liste vide
mesures.append(12.9)    # mesures vaut [12.9]
mesures.append(13.1)    # mesures vaut [12.9, 13.1]
```

5. Conditions

`if` exécute un bloc d'instructions **seulement si** une condition est vraie ; `elif` (« sinon, si ») et `else` (« sinon ») traitent les autres cas :

```
pH = 5.2
if pH < 7.0:
    print("solution acide")
elif pH == 7.0:
    print("solution neutre")
else:
    print("solution basique")
```

Cap bac — l'indispensable

Le bloc qui dépend du `if` est signalé par le **décalage de début de ligne** (l'**indentation**, 4 espaces) : en Python, l'indentation fait partie de la syntaxe. Ne pas oublier non plus les **deux-points** en fin de ligne `if`, `elif`, `else` (et `for`, `while`).

6. Boucle for

`for` répète un bloc **pour chaque élément** d'une liste. C'est l'outil naturel pour transformer une série de mesures :

```
c = 0.10                    # concentration titrante (mol/L)
volumes = [5.0, 10.0, 15.0, 20.0]   # volumes versés (mL)
quantites = []              # liste vide
for V in volumes:           # pour chaque élément de la liste V
    n = c * V / 1000        # calcul de la quantité correspondante (V converti en L)
    quantites.append(n)     # ajoute à quantités la nouvelle valeur (mol)
print(quantites)
```

À chaque tour, `V` prend la valeur suivante de la liste ; le bloc indenté est exécuté, et la liste `quantites` s'allonge. Pour répéter simplement N fois, on écrit `for i in range(N):` (`i` prend les valeurs $0, 1, \ldots, N-1$).

7. Boucle while

`while` répète un bloc **tant qu'**une condition reste vraie — utile quand on ne sait pas d'avance combien de répétitions il faudra :

```
c = 1.0              # concentration initiale (mol/L)
nb = 0               # compteur de dilutions
while c > 1e-3:      # tant que c dépasse 0.001 mol/L
    c = c / 10       # une dilution au dixième
    nb = nb + 1
print(nb)            # affiche : 3
```

Le bloc doit **faire évoluer** la condition (ici `c` diminue) : sinon la boucle ne s'arrête jamais.

Pièges fréquents

- Le séparateur décimal est le **point** (`0.5`) : la virgule de `[0,5]` crée une liste de deux entiers !
- `=` **affecte**, `==` **compare** : `if pH = 7.0` est une erreur de syntaxe.
- L'**indentation** n'est pas décorative : un bloc mal indenté change le sens du script (ou provoque une erreur).
- Les indices d'une liste commencent à **0** : `volumes[1]` est le **deuxième** élément.
- Une boucle `while` dont la condition ne devient jamais fausse tourne indéfiniment.

Cap bac — l'indispensable

- Variable : `n = m / M` ; affichage : `print(n)`.
- Liste : `L = []` puis `L.append(x)` ; accès `L[0]`, longueur `len(L)`.
- Condition : `if` / `elif` / `else` + deux-points + indentation.
- Répéter sur une liste : `for x in L:` ; répéter tant que : `while condition:`.

---

8. Auto-test

Prédisez le résultat avant d'exécuter (puis vérifiez dans EduPython !).

1. Qu'affiche ce script ?

```
m = 2.0
m = m + 1.0
print(m)
```

2. Qu'affiche `print(volumes[1])` si `volumes = [5.0, 10.0, 15.0]` ?
3. Compléter la ligne marquée `# À COMPLÉTER` pour construire la liste des concentrations en ions oxonium :

```
liste_pH = [1.8, 2.5, 4.6, 7.0]
liste_h = []
for pH in liste_pH:
    h = 10**(-pH)
    # À COMPLÉTER : ajouter h à la liste liste_h
print(liste_h)
```

4. Combien de fois le bloc de cette boucle est-il exécuté ?

```
c = 1.0
nb = 0
while c > 0.1:
    c = c / 2
    nb = nb + 1
print(nb)
```

Corrigés

1. `3.0` : la deuxième ligne remplace le contenu de `m` par son ancienne valeur plus 1.
2. `10.0` : les indices commencent à 0, donc `volumes[1]` est le deuxième élément.
3. `liste_h.append(h)` (indenté de 4 espaces, dans le bloc de la boucle).
4. 4 fois : c vaut successivement $0{,}5$; $0{,}25$; $0{,}125$; $0{,}0625$ — et $0{,}0625 \leq 0{,}1$ arrête la boucle. Le script affiche `4`.