MÉMENTO PYTHON 3

pour la physique-chimie au lycée

Variables · Affichage · Tests · Boucles · NumPy · Matplotlib

Comment lire ce mémento

Le code s'écrit en **police à chasse fixe**. Les mots en **bleu** sont des instructions Python, les textes en **vert** sont des textes entre guillemets, et les mots en *italique rose* sont à **remplacer par tes propres valeurs ou noms**. Tout ce qui suit un `#` est un commentaire : Python l'ignore.

1 · VARIABLES

Une **variable** est une « boîte » qui porte un nom et qui contient une valeur (nombre ou texte). Le signe `=` n'est **pas** une égalité mathématique : c'est une **affectation**. Python calcule ce qui est à droite et le range dans la variable de gauche.

`N = 2`	Affecte à la variable **N** la valeur entière (type *int*) 2.
`x = 3.14`	Nombre **décimal** (type *float*) : le séparateur décimal est le **point**.
`T = "Bonjour"`	Texte (type *string*) : toujours entre guillemets.
`N = N + 1`	La variable peut être réutilisée : si N valait 2, elle vaut maintenant 3.
`# mon commentaire`	Commentaire : ignoré par Python, utile pour expliquer le programme.

Choisir un nom de variable

Pas d'accent · pas d'espace (on écrit `masse_eau`) · **pas de mot réservé de Python · majuscules et minuscules différentes** : `m` et `M` sont deux variables distinctes. *Piège : « lambda » est un mot réservé de Python : pour une longueur d'onde, écrire* `lam` *ou* `longueur_onde`. Choisis des noms explicites (`masse`, `vitesse`).

Opérations de calcul

`a + b` `a - b`	Addition, soustraction.
`a * b`	**Multiplication** : toujours avec `*` (jamais x ni ×).
`a / b`	Division : le résultat est un décimal.
`a ** b`	Puissance : `2**4` vaut 16.
`a // b` `a % b`	Quotient entier et reste de la division euclidienne : `7//2` vaut 3 et `7%2` vaut 1.

Précision

Python respecte les **priorités** comme en mathématiques, mais il faut des **parenthèses** pour une division : `1/(2*3)` vaut 0,1666… alors que `1/2*3` vaut 1,5.

⚠ Attention

La **virgule** décimale se tape avec un **point** (3,14 s'écrit `3.14`). Une virgule sépare des éléments, jamais les décimales.

2 · AFFICHAGE D'UN TEXTE OU D'UNE VARIABLE

`print("Longueur =", L, "mètres")`	Écrit dans la console le texte, **puis la valeur de L**, puis le texte suivant. Les éléments sont séparés par des **virgules** et Python ajoute un espace entre eux.

`print("%.2e" % N)`	Affiche N en **écriture scientifique** avec 2 décimales (3 chiffres significatifs) : 12345 s'affiche `1.23e+04`.
`print("%.3f" % N)`	Affiche N en écriture décimale avec **3 chiffres après la virgule** : 3.14159 s'affiche `3.142`.
`print(f"{N:.2e}")`	Même résultat que `%.2e`, avec la syntaxe des *f-strings* (le texte commence par `f`).
`print("bonjour\nle monde")`	`\n` **saute une ligne** au milieu d'un texte.

Précision
`%.2e` et `%.3f` modifient seulement **l'affichage**, pas la valeur de la variable. Pour arrondir la valeur elle-même, utiliser `np.around(x, n)` (voir §6).

Puissances de 10 et exposants

`2E4`	vaut 2×10^{4}, soit 20000.0 (le `E` signifie « fois dix puissance »).
`1E-2`	vaut 1×10^{-2}, soit 0.01.
`2**4`	vaut 2^{4}, soit 16.
`10**-4.8`	vaut $10^{-4,8}$, soit environ $1{,}58\times10^{-5}$ (exposant décimal possible).

Lettres grecques et symboles

On peut **afficher** une lettre grecque avec `chr(code)`, mais il vaut mieux ne pas l'utiliser comme nom de variable.

`print(chr(945))`	Affiche la lettre **α** (code 945).
`print("L'angle", chr(945), "vaut", 45, "°")`	Affiche : *L'angle α vaut 45* °

945	946	947	948	949	951	952	954	955	956	957	960	961	963	964	965
α	β	γ	δ	ε	η	θ	κ	λ	μ	ν	π	ρ	σ	τ	υ
966	968	969	916	923	928	931	934	936	937	8771	8800	8804	8805	8734	176
φ	ψ	ω	Δ	Λ	Π	Σ	Φ	Ψ	Ω	≃	≠	≤	≥	∞	°

3 · TESTS ET CONDITIONS

Un test permet d'exécuter des instructions **seulement si** une condition est vraie (*True*) ou fausse (*False*). Les `:` en fin de ligne et l'**indentation** (le décalage de 4 espaces) sont **obligatoires** : tout ce qui est décalé fait partie du test.

`if condition:` `    instructions`	**Test simple** : les instructions ne sont exécutées que si la condition est vraie.

`if condition:`
`    instructions 1`
`else:`
`    instructions 2`
	Si… sinon : instructions 1 si la condition est vraie, instructions 2 dans le cas contraire.

`if condition1:`
`    instructions 1`
`elif condition2:`
`    instructions 2`
`else:`
`    instructions 3`
	Si… sinon si… sinon : plusieurs cas. `elif` n'est testé que si la condition précédente est fausse ; `else` s'applique si **toutes** les conditions sont fausses.

Opérateurs à utiliser dans une condition

`==` `!=`	égal / différent
`<` `>` `<=` `>=`	inférieur, supérieur, inférieur ou égal, supérieur ou égal
`c1 and c2`	**ET** : les deux conditions doivent être vraies.
`c1 or c2`	**OU** : au moins une des deux conditions est vraie.

`not c`	**contraire** de la condition.
`0 < x < 10`	On peut **enchaîner** les comparaisons (x strictement compris entre 0 et 10).

> ⚠️ **Attention**
> Le signe = sert uniquement à **affecter** une valeur à une variable. Pour **tester** une égalité, il faut ==.

Exemple : acide ou basique ?

```
pH = 4.8
if pH < 7:
    print("solution acide")
elif pH == 7:
    print("solution neutre")
else:
    print("solution basique")      # à 25 °C
```

4 • BOUCLES

Une **boucle** répète des instructions. On utilise **for** quand on connaît le nombre de répétitions, **while** quand on ne le connaît pas.

`for i in range(n):` `    instructions`	`i` varie de **0 à n−1** : la boucle est répétée **n fois**. Ex. `range(5)` donne 0, 1, 2, 3, 4.
`for i in range(debut, fin):`	`i` varie de **début à fin−1** : la **borne de fin est exclue**. Ex. `range(2,6)` donne 2, 3, 4, 5.
`for i in range(debut, fin, pas):`	`i` varie de début à fin−1 par **sauts de pas**. Ex. `range(0,10,3)` donne 0, 3, 6, 9 ; `range(10,0,-2)` donne 10, 8, 6, 4, 2.
`for e in tableau:` `    instructions`	e prend successivement **chaque valeur du tableau**.
`while condition:` `    instructions`	Répète les instructions **tant que la condition est vraie**. Il faut **modifier**, dans la boucle, la variable utilisée dans la condition, sinon la boucle est **infinie** (arrêt : touches `Ctrl + C`).

> **Précision**
> `range` ne travaille qu'avec des **nombres entiers**. Pour un pas décimal, utiliser `np.arange` ou `np.linspace` (voir §7).

Exemples

```
somme = 0                          # somme des entiers de 1 à 4
for i in range(1, 5):
    somme = somme + i
print(somme)                       # affiche 10

t = 0                              # instants de 0 à 0,75 s par pas de 0,25 s
while t < 1:
    print(t)
    t = t + 0.25                   # on modifie t : la boucle finit par s'arrêter
```

5 • IMPORTER DES MODULES

Un **module** est une boîte à outils de fonctions supplémentaires. Les `import` se placent **tout au début du programme, une seule fois.** L'alias (`as np`) est un raccourci : on écrit `np.sin` au lieu de `numpy.sin`.

`import random`	Module du **hasard**.
`import numpy as np`	**NumPy**, sous l'alias `np` : tableaux et fonctions mathématiques.
`import matplotlib.pyplot as plt`	**Matplotlib**, sous l'alias `plt` : tracé de courbes.

Le hasard : module random

`random.randrange(a, b)`	Entier au hasard de **a (inclus) à b (exclu)**.
`random.randint(a, b)`	Entier au hasard de **a à b, tous deux inclus**.
`random.random()`	Décimal au hasard entre 0 et 1 (1 exclu).

6 · FONCTIONS MATHÉMATIQUES (module NumPy)

`np.sin(x)` `np.cos(x)` `np.tan(x)`	Fonctions trigonométriques : **x doit être en radians**.
`np.arcsin(x)` `np.arccos(x)` `np.arctan(x)`	Fonctions trigonométriques inverses : le résultat est en **radians**.
`np.radians(x)` `np.degrees(x)`	Convertit des **degrés en radians**, et des radians en degrés.
`np.pi`	Le nombre π.
`np.sqrt(x)`	Racine carrée de x.
`np.log(x)` `np.exp(x)`	**Logarithme népérien** ln(x) ; exponentielle e^x.
`np.log10(x)` `10**x`	Logarithme décimal log(x) ; puissance de 10.
`np.abs(x)`	Valeur absolue.
`np.sign(x)`	Signe : renvoie 1 si x est positif, −1 s'il est négatif (0 si x = 0).
`np.around(x, n)`	Arrondi de x à **n décimales**.
`y ** x`	y puissance x.

Précision

Pour calculer sin(30°) : `np.sin(np.radians(30))`. Le résultat affiché est `0.49999999999999994`, très proche de 0,5 : c'est un **arrondi de l'ordinateur**. On l'affiche proprement avec `np.around(np.sin(np.radians(30)), 3)`, soit `0.5`.

7 · TABLEAUX DE DONNÉES (module NumPy)

En physique-chimie, un **tableau** contient **une grandeur** : par exemple tous les instants `t`, ou toutes les valeurs de la vitesse `v`.

Créer un tableau

`T = np.array([v1, v2, v3])`	Crée « à la main » un tableau avec les valeurs v1, v2, v3.
`T = np.linspace(debut, fin, n)`	Crée automatiquement **n valeurs régulièrement espacées**, **fin incluse**.
`T = np.arange(debut, fin, pas)`	Crée automatiquement les valeurs de début à fin−1 **par pas de pas** (**fin exclue**) ; le pas peut être décimal.
`U = 200 * I`	Crée un tableau U dont chaque valeur vaut **200 fois** celle du tableau I.

Utiliser un tableau

`T[n]`	Valeur de l'élément d'**indice n**. **Les indices commencent à 0** : `T[0]` est le premier élément.
`T[-1]`	**Dernier** élément (`T[-2]` : l'avant-dernier).
`T[n] = valeur`	Remplace la valeur d'indice n.
`T = np.append(T, N)`	Ajoute la valeur N à la fin du tableau.
`len(T)`	**Nombre d'éléments** du tableau (un entier).

`np.max(T)` `np.min(T)` `np.mean(T)` `np.sum(T)`	Maximum, minimum, **moyenne** et **somme** des valeurs du tableau.

Précision

Les calculs se font **élément par élément**, sans boucle : `P = U * I` multiplie les deux tableaux valeur par valeur (ils doivent avoir la **même taille**) et `np.sqrt(T)` applique la racine carrée à toutes les valeurs.

```
t = np.linspace(0, 1, 5)        # [0.   0.25 0.5  0.75 1.  ]
v = 3 * t                       # [0.   0.75 1.5  2.25 3.  ]
print(t[0], v[-1])              # affiche 0.0 3.0
```

8 · TRACER DES COURBES (module Matplotlib)

`plt.plot(X, Y, "style")`	Trace la courbe **Y en fonction de X**. X et Y sont des tableaux de **même taille**.
`plt.plot(X, Y, "style", label="texte")`	Même tracé, avec un texte associé pour la légende.
`plt.legend()`	**Affiche la légende** (à écrire une seule fois, indispensable pour voir les `label`).
`plt.title("titre")`	Titre du graphique.
`plt.xlabel("nom")` `plt.ylabel("nom")`	Noms des axes : **grandeur et unité** (ex. `"t (s)"`).
`plt.axis([xmin, xmax, ymin, ymax])`	Fixe les valeurs minimale et maximale des axes.
`plt.grid()`	Affiche la grille.
`plt.quiver(X, Y, Vx, Vy, color='c', scale=20)`	Trace un **vecteur** d'origine (X, Y) et de coordonnées (Vx, Vy). `scale` est l'échelle, à ajuster par essais.
`plt.savefig("nom.png")`	**Enregistre** l'image (à placer **avant** `plt.show()`).
`plt.show()`	**Affiche la fenêtre** du graphique : indispensable, à placer **tout à la fin** du programme.

Le style d'un tracé : couleur, point, trait

Le style est un texte de **1 à 3 symboles**, à écrire **dans l'ordre : couleur, type de point, type de trait**. Exemple : `"rx-"` trace en **rouge**, avec des **croix**, reliées par un **trait plein**.

Couleur	`"r"`	`"b"`	`"g"`	`"k"`	`"m"`	`"c"`
	rouge	bleu	vert	noir	magenta	cyan
Point	`"o"`	`"."`	`"x"`	`"+"`	`"v"`	
	gros point	petit point	croix	croix +	triangle	
Trait	`"-"`	`"--"`				
	trait plein	pointillés				

Exemple complet

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 2, 50)       # 50 instants entre 0 et 2 s
g = 9.81
v = g * t                       # vitesse en chute libre (m/s)

plt.plot(t, v, "r-", label="v(t)")
plt.title("Vitesse en chute libre")
plt.xlabel("t (s)")
plt.ylabel("v (m/s)")
plt.grid()
plt.legend()
plt.show()                      # toujours en dernier
```

9 • ERREURS FRÉQUENTES

Les 7 oublis classiques

- Une **parenthèse, un crochet ou un guillemet** non fermé.
- Une **virgule** dans un nombre à la place du point.
- La lettre **x** à la place de `*` pour multiplier.
- L'**indentation** non respectée dans un test ou une boucle.
- Les **deux-points** `:` oubliés après `if`, `elif`, `else`, `for`, `while`.
- Un `=` au lieu de `==` dans une condition.
- La **fenêtre Matplotlib** pas fermée : le programme ne se termine pas.

Comprendre le message d'erreur

Python indique la **ligne** de l'erreur (ou celle d'**avant**) et son **type**. Lis toujours la dernière ligne du message.

Message	Cause probable
`SyntaxError`	Syntaxe incorrecte : `:`, parenthèse ou guillemet oublié, virgule à la place du point.
`IndentationError`	Décalage incorrect des lignes dans un `if`, un `for` ou un `while`.
`NameError`	Variable utilisée **avant d'être définie**, faute de frappe, majuscule/minuscule (`T` ≠ `t`), ou module oublié (`import`).
`TypeError`	Opération impossible entre deux types, par exemple un **texte** et un **nombre** : séparer par une virgule dans `print`.
`ZeroDivisionError`	**Division par zéro**.
`ValueError (dans plt.plot)`	`x and y must have same first dimension` : les tableaux X et Y n'ont **pas la même taille**.