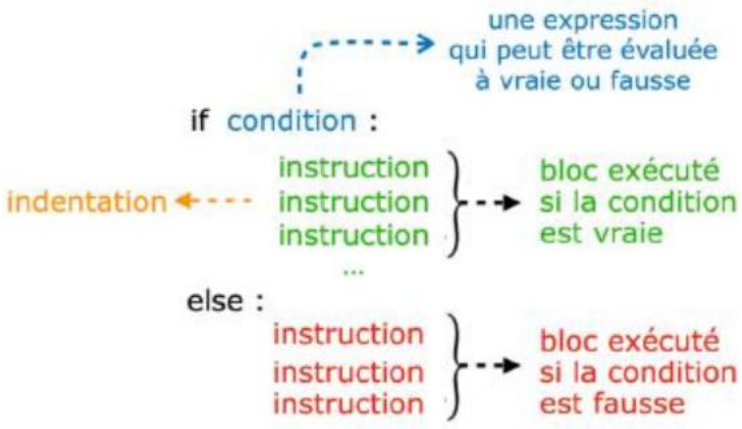
Initiation à Python

« Comprendre un phénomène scientifique, c'est savoir exploiter les données qui le décrivent. »

1- Organisation d'un programme Python

Un programme Python est exécuté ligne par ligne. Certaines instructions permettent cependant de modifier le déroulement du programme : ce sont les structures conditionnelles et les boucles.

Les **blocs d'instructions** sont définis par le décalage vers la droite (indentation). En Python, il n'y a pas d'accolades (comme en Java) : l'indentation est essentielle.



Les commentaires servent à expliquer le programme. Ils commencent par le caractère # et ne sont pas exécutés.

```
print("Python est très lisible")  # commentaire
```

2- Bibliothèques utilisées en sciences

En physique-chimie, on utilise principalement trois bibliothèques :
- numpy : pour les calculs numériques
- matplotlib : pour tracer des graphiques
- math : pour les fonctions mathématiques classiques utilisées en sciences (trigo, racine carrée, log, …)

Pour les graphiques simples, on utilise le sous-module pyplot :

```
import matplotlib.pyplot as plt
```

Quelques méthodes de numpy et de pyplot très utiles :

```
plt.quiver() # trace un vecteur en 2D
plt.plot() # dessine des points et les ajoute à une figure
plt.axis() # renvoie les coordonnées extrêmes des axes : (xmin, xmax, ymin, ymax)
plt.xlabel() plt.ylabel() # précise les titres des axes
plt.legend() # précise la légende
plt.title() # ajoute un titre
plt.show() # affiche une figure à l'écran
plt.axis('equal') # trace un repère orthonormé
plt.axis([xinf, xmax, y inf, ymax]) # précise les valeurs limites des axes
plt.xticks([0,1,2,…]) ou plt.yticks([0,1,2,…]) # choix des graduations suivant x et y
np.polyfit(x,y,d) # realise une régression (ajustement ou modélisation) polynomiale de degré d avec les listes de données x et y
np.linspace(d,f,n) # crée une liste avec n valeurs dans l'intervalle [d,f]
```


3- Variables

Une variable permet de stocker une information (nombre ou texte). L'affectation se fait avec le symbole =.

```
x = 10
y = 3.5
formule = "E = mc²"
```

Attention : Python distingue les majuscules et les minuscules.

4- Affichage à l'écran

L'instruction print() permet d'afficher un message ou la valeur d'une variable.

```
print("La valeur de x est :", x)
```

5- Opérations mathématiques usuelles

Opération	Symbole	Exemple
Addition	+	`Em = Ec + Ep`
Soustraction	–	`Ec = Em – Ep`
Multiplication	*	`U = R*I`
Division	/	`m = P/g`
Puissance	**	`s = l **2`
Division entier	//	`print(10//3) affiche 3`
Reste de division (modulo)	%	`print(10%3) affiche 1`
Incrémentation	+=	`pas+=2 incrémente le pas de 2`
Incrémenter par un facteur	*=	`ech *=2 : multiplie ech par deux`

6- Conditions

L'instruction if permet d'exécuter un bloc seulement si une condition est vraie.

```
if v > 5:
    print("Rapide")
else:
    print("Lent")
```

Pour construire la condition, on peut utiliser des opérateurs de comparaison (voir ci-contre) et si nécessaire des connecteurs logiques (voir ci-dessous).

>	strictement supérieur à
<	strictement inférieur à
>=	supérieur ou égal à
<=	inférieur ou égal à
==	égal à
!=	différent de

Python	Nom logique	Lecture
and	Conjonction logique	"et"
or	Disjonction logique	"ou"
not	Négation logique	"non"

Dans l'exemple ci-dessous, la fonction bissextile prend une année en argument et indique si cette année est bissextile ou non.

Une année est dite bissextile si elle est un multiple de 4, sauf lorsqu'elle est un multiple de 100. Elle reste toutefois bissextile si elle est un multiple de 400.

```
def bissextile(n):
if(n%4==0 and n%100!=0 or n%400==0): print("L'annee est une annee bissextile!")
      # trois conditions combinées
else:
      print("L'annee n'est pas une annee bissextile!")
annee = int(input("Entrez l'annee a verifier:"))
bissextile(annee)
```

7- Boucles

Une boucle permet de répéter plusieurs fois un même bloc d'instructions. Python propose deux boucles : ***for*** (pour) et ***while*** (tant que).

For

Boucle for : utilisée quand le nombre de répétitions est connu.

Quelques exemples

```
for i in range(4):
  print(i) → affiche 0, 1, 2 et 3. range(4) génère quatre valeurs, en commençant à 0.
  La valeur 4 n'est pas incluse.

for i in [0,1,2,3,4]:
  print(i) → affiche 0, 1, 2, 3 et 4. La boucle parcourt tous les éléments de la
liste, exactement tels qu'ils sont écrits.

for i in range(1,10,2):
print(i) → affiche 1, 3, 5, 7, 9. 1 : valeur de départ (incluse), 10 : valeur de fin
  (non incluse) et 2 : pas de la boucle
```

While

Boucle while : utilisée quand on ne connaît pas à l'avance le nombre de répétitions.

Exemple : Une cuve de 1 000 L se remplit de 3 L par jour. Combien de jours faut-il pour la remplir ?

Pour répondre à cette question, on utilise la fonction remplir() suivante :

```
def remplir():
  # initialisation des variables (souvent avec while)
  contenance = 1000
  hauteur = 0
  jour = 0
  while hauteur < contenance:
      jour+=1
      hauteur = hauteur+3
  return jour
  print(remplir())
```

8- Listes

Une liste permet de stocker plusieurs valeurs. **Les indices commencent toujours à 0**.

`t = [0.0, 0.1, 0.2]` → crée une liste avec trois valeurs '0.0' , '0.1' , '0.2'

`t = []` → crée une liste vide

`print(t[1])` → renvoie 0.1

`t.append(0.3)` → ajoute la valeur '0.3' à la fin de la liste t

`len(liste)` donne le nombre d'éléments de la liste.

`t.count(0.1)` → renvoie le nombre d'occurrences de l'élément '0.1' dans la liste t

`t.index(0.1)` → retourne l'indice de l'élément '0.1' dans la liste t (ici 1)

9- Graphique

La fonction plot() sert à préparer le graphique. Elle définit ce que l'on veut tracer (points, lignes, données) et ajoute le graphique à la figure.

La fonction show() permet d'afficher la figure à l'écran.

Il est possible d'utiliser plusieurs instructions plot() pour tracer plusieurs graphiques sur une même figure.

L'instruction show() est appelée une seule fois, à la fin, pour afficher la figure contenant l'ensemble des graphiques.

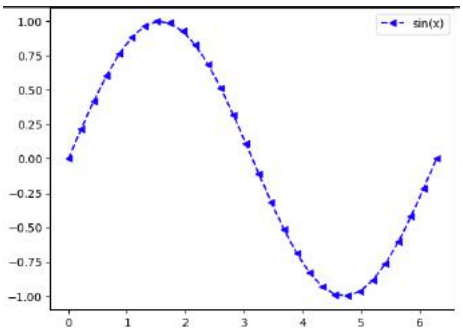
```
import numpy as np
import matplotlib.pyplot as plt
x = [1, 3, 4, 6]
y = [2, 3, 5, 1]
plt.plot(x, y)
plt.show() # affiche la figure à l'écran
```

Syntaxe de la commande plot : plot(liste des abcisses , liste des ordonnées , option de style)

Les options de style pour le graphique

Couleur	'c'	cyan
	'm'	magenta
	'y'	jaune
	'r'	rouge
	'g'	vert
	'b'	Bleu
	'w'	blanc
	'k'	black
Style de ligne	'-'	continue
	'--'	pointillée
	':'	points
	'-.'	alternée point-trait
Type de marqueur	'+'	signe plus
	'o'	cercle non rempli
	'*'	astérisque
	'x'	lettre x
	's'	carré non rempli
	'd'	losange non rempli
	'^'	triangle pointé vers le haut non rempli
	'v'	triangle pointé vers le bas non rempli
	'>'	triangle pointé vers la droite non rempli
	'<'	triangle pointé vers la gauche non rempli
	'p'	pentagramme non rempli
	'h'	hexagramme non rempli

👉 Toutes les options de style doivent être regroupées dans une seule chaîne.




```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.sin(x)
plt.plot(x, y, 'b<--',label = 'sin(x)')
plt.legend()
plt.show()
```

10- Vecteur

On utilisera la fonction `plt.quiver`

Synthaxe : `plt.quiver(x, y, deltax, deltay, angles='xy',scale _units="xy', scale=1)`

- x et y correspondent aux coordonnées des points de départ des vecteurs (listes).
- deltax et deltay représentent les composantes des vecteurs selon l'axe des abscisses et l'axe des ordonnées (listes).
- angles='xy' garantit que la direction des flèches est cohérente avec l'échelle des axes ; ce paramètre doit être utilisé systématiquement.
- scale_units='xy' indique que les unités des vecteurs sont celles des axes ; ce paramètre est également à utiliser systématiquement.
- scale=1 correspond à une échelle 1:1 pour les vecteurs. Une valeur supérieure à 1 réduit la taille des flèches, tandis qu'une valeur inférieure à 1 les agrandit.
- Il est possible de modifier la couleur des flèches en ajoutant le paramètre color='valeur'.

Exemple : Chronophotographie de la trajectoire d'un ballon :

```
import numpy as np
import matplotlib.pyplot as plt
# Données
dt = 0.0667
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359,
1.490, 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207,
1.018,0.797, 0.544, 0.266]
# Calculs des vecteurs vitesses
N = len(x) # Nombre de points de mesures
vx = np.zeros(N) # Initialisation d'un tableau vide de N éléments
vy = np.zeros(N) # Idem
for i in range(1, N-1):
  vx[i]=(x[i+1]-x[i-1])/(2*dt)
  vy[i]=(y[i+1]-y[i-1])/ (2*dt)
# Calcul des vitesses
v = np.sqrt (vx**2+vy**2)
plt.plot(x, y, ".")
plt.xlabel("x (m)")
plt.xlim(0, 2)
plt.ylabel("y (m)")
plt.ylim(0, 2)
plt.title("Trajectoire d'un ballon")
plt.quiver(x, y, vx, vy, angles='xy', scale_units='xy', scale=10, color='red',
width=0.005)
print('Point',i, '-> v=', round(v[i],2), " m/s")
plt.show()
```

11- <u>Regression linéaire</u>

La fonction `linregress` appartient au module stats de la bibliothèque scipy. Elle permet d'effectuer une régression linéaire à partir de données expérimentales.

À partir des valeurs de x et de y, elle calcule les paramètres de la droite de régression ainsi que des indicateurs statistiques associés.

Importation et syntaxe :

```
from scipy.stats import linregress
a, b, rho, p_value, std_err = linregress(x, y)
```

La fonction linregress renvoie cinq valeurs. Les trois premières sont les plus utiles en physique :

- a : coefficient directeur de la droite modèle,
- b : ordonnée à l'origine,
- rho : coefficient de corrélation linéaire ρ, qui mesure la qualité de l'ajustement (plus ρ est proche de 1, meilleure est la modélisation par une droite).

12- Exploitation de données de chronophotographie : tracé de la vitesse et de son évolution

```
import matplotlib.pyplot as plt

dt = 0.067

t = [0.00, 0.067, 0.134, 0.201, 0.268, 0.335]
x = [0.00, 0.14, 0.28, 0.41, 0.55, 0.69]
y = [1.20, 1.35, 1.44, 1.47, 1.44, 1.35]

# --- vitesses entre i et i+1 ---
vx, vy = [], []
x_v, y_v = [], []   # point d'application des vitesses (au point i)

for i in range(len(t) - 1):
    vx.append((x[i+1] - x[i]) / dt)
    vy.append((y[i+1] - y[i]) / dt)
    x_v.append(x[i])
    y_v.append(y[i])

# --- variations de vitesse entre i et i+1 (donc placées au point i+1) ---
Dvx, Dvy = [], []
x_dv, y_dv = [], []  # point d'application de Δv (au point i+1)

for i in range(len(vx) - 1):
    Dvx.append(vx[i+1] - vx[i])
    Dvy.append(vy[i+1] - vy[i])
    x_dv.append(x[i+1])
    y_dv.append(y[i+1])

# --- tracés ---
plt.plot(x, y, marker="x", linestyle="", label="Positions")

# Options importantes : angles + scale_units pour éviter les déformations
# scale : plus grand => flèches plus petites (à ajuster)
plt.quiver(x_v, y_v, vx, vy, angles="xy", scale_units="xy",
           scale=25, color="red", label="Vecteurs vitesse")

plt.quiver(x_dv, y_dv, Dvx, Dvy, angles="xy", scale_units="xy",
           scale=8, color="green", label="Vecteurs variation de vitesse")

plt.axis("equal")
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Chronophotographie : vitesses et variations de vitesse")
plt.legend()
plt.show()
```