Memento PYTHON pour la PHYSIQUE-CHIMIE

Partie A : Quelques éléments de syntaxe

- La **console** est un interpréteur actif qui permet de tester des instructions de façon isolée (une ligne de code)
- L'éditeur permet d'entrer plusieurs instructions avant de les exécuter. Les sorties et affichages éventuels apparaissent dans la console. Le « code » du programme écrit dans l'éditeur est exécuté lorsque l'on clique sur le bouchon flèche verte.
- Les erreurs ne sont pas nos ennemies ! Il faut savoir les lire, les comprendre et les interpréter pour déboguer rapidement les scripts. (La ligne incriminée est indiquée dans le message d'erreur !)

I- Variables et valeurs

En programmation, les ***variables*** peuvent posséder des types différents : **entier (integer), nombre décimal (float), chaînes de caractères (string)**, **liste (list), tableau (array),** etc…
Ces types sont automatiquement affectés en fonction de leur valeur.
Par exemple, l'instruction `x = 4` ***affecte*** la valeur de type 'entier' (integer) à la variable x.
Inutile pour nous de déclarer le type.
NB : Par convention, le nom d'une variable doit commencer par une minuscule et ne contenir aucun caractère spécial ni accentué. Seul le tiret-bas _ est autorisé.

L'explorateur de variables permet de suivre le type de la variable ainsi que sa valeur. C'est pratique !!

Type	Exemples d'instruction	Explication
String (Str)	"bonjour" "4" "Salut ?! "	Une **chaîne de caractères** alphanumériques est toujours indiquée entre guillemets. Les caractères de la chaine sont numérotés depuis l'indice 0. On accède au $i^{ième}$ caractère d'une chaine c avec l'instruction `c[i]`. L'indice -1 désigne le dernier élément de la chaîne. Par exemple `"bonjour"[0]` renvoie le caractère `"b"`.
Integer (Int)	4	Nombre entier
Float	3.14	Nombre décimal. La virgule est indiquée par un point.
list	["bonjour", 8, 5.2]	**Liste** d'éléments introduits par des crochets et séparés entre eux par des virgules. Comme pour une chaine de caractères (string), les éléments d'une liste sont numérotés **depuis l'indice 0 et l'indice -1 correspond au dernier élément**. Dans l'exemple, `L[0]` renvoie `bonjour` et `L[-1]` renvoie `5.2`
Boolean (bool)	True False	**Variable logique** prenant comme unique valeur True ou False (avec une majuscule en première lettre).

II- Opérateurs et comparateurs usuels

A) Les opérateurs usuels

*La plupart des opérateurs mathématiques courants (+, -, * et /) sont disponibles sous Python. Par convention, on met **un espace avant et après chaque opérateur**.*
Un calcul impliquant des nombres décimaux (« flottants ») donne obligatoirement une valeur de type float en sortie de calcul, même lorsque le résultat est entier .
L'instruction round(x,n) permet d'arrondir une valeur x à n décimales.

Exemples d'opérations	Explications
`[1,2,3] * 4`	→ Renvoie ici `[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]` La multiplication d'une liste par un entier renvoie une liste contenant les éléments répétés autant de fois que l'entier indiqué. NB : l'opérateur *4 ne correspond pas à multiplier chaque terme par 4 (pour cela on utilisera le type array ,voir en partie C)
`2**4`	→ Renvoie ici 16 L'opérateur ** renvoie la puissance.
1.5E-3	Les puissances de 10 sont aussi disponibles. Il est ici écrit 1.5×10^{-3}
`round(124.187,2)`	→ Renvoie ici 124.19 NB : ne pas confondre `.` et `,`

B) Les comparateurs usuels

*La plupart des **comparateurs** mathématiques usuels sont disponibles en Python. Leur exécution renvoie toujours un **booléen** (True ou False) lors du test.*

Opérateurs	Explications et exemples
>, >=, <, <=	Supérieur à, supérieur ou égal à, inférieur à, inférieur ou égal à.
==	La double égalité permet de tester l'égalité entre les valeurs <u>et les types</u> de 2 variables (le signe = simple étant déjà réservé à l'affectation). Une tolérance s'applique sur la comparaison entre un entier et un flottant. 2 == 2 renvoie True [4, 5, 6] == "4, 5, 6" renvoie False 1.0 == 1 renvoie True NB : ***Dans la mesure du possible, on évitera d'associer flottants et comparateurs !!***
!=	Opérateur de différence. Compare les variables et renvoie True si <u>la valeur ou le type</u> sont différents. Une tolérance s'applique entre un entier et un flottant. [4, 5, 6] != [4, 5, 6] renvoie False "bonjour" != "aurevoir" renvoie True 4 == 4.0

NB : conseil d'un prof de physique... éviter l'opérateur ==

III- Quelques instructions utiles

Il existe plusieurs fonctions, méthodes et instructions particulièrement utiles pour l'exploitation de Python en Physique, et présentes de façon native (sans appel de bibliothèque).

Python dispose de nombreux outils permettant de réaliser certaines opérations fréquentes rapidement :

- *la méthode* `.pop(n)` *qui enlève l'élément d'indice n*
- ***la méthode*** `.append(x)` ***qui ajoute x en fin de liste***
- *les instructions* `sum`*,* `min` *et* `max` *qui portent bien leur nom*
- *l'instruction* `len` *qui permet de récupérer la taille de la liste.*

La technique de ***slicing*** `L[a:b]` *permet de récupérer tous les éléments d'une liste de l'indice a à l'indice b-1.*

Exemples	Explications
len(L) / len(mot) max(L) min(L) sum(L)	len : renvoie le nombre d'éléments d'une **liste** ou d'une chaine de caractères. max : renvoie le maximum d'une liste min : renvoie le minimum d'une liste sum : renvoie la somme des éléments d'une liste Pour la liste [1,3,5,7], `len(L)` renvoie 4, sum(L) renvoie 16 etc.

L[a:b]
mot[a:b]
	Méthode de ***slicing (***renvoyant une séquence allant de l'indice (a) à l'indice (b-1) d'une liste ou d'une chaine de caractères. Si rien n'est indiqué avant le signe : , le découpage commence au début. Si rien n'est indiqué après le signe :, le découpage va jusqu'à la fin.
	Pour la liste `L=[2, 4, 6, 8, 10]`
	L'instruction `L[2:4]` renvoie `[6, 8]`
	`L[:3]` renvoie `[2, 4, 6]`
	`L[2:]` renvoie `[6, 8, 10]`

L[n] = x	Réaffecte la valeur x à la $n^{ième}$ valeur de la liste L.
L.append(x)	**Ajoute l'élément x en dernière position de la liste L.**
L.pop(n)	Enlève l'élément de la liste L situé à l'indice n (et le renvoie, de sorte qu'il puisse être réaffecté si besoin).
del L[n]	Enlève et efface l'élément de la liste L situé à l'indice n
x in L x not in L	Teste la présence de la valeur x dans L (de type chaine, ou liste) et renvoie un booléen True ou False. `"a" in "bonjour"` renvoie False `1 not in [2, 4, 6]` renvoie True

Partie B : Quelques structures fondamentales

IV- La structure de test : if… elif… else

Derrière les **mots clefs `if`** et **`elif`**, la condition est nécessairement un booléen : on rentre alors dans le ***bloc d'instructions*** (qui doit être ***indenté* de 4 espaces, soit 1 tabulation**) si et seulement si le booléen est True. Dans le cas contraire, le script passe au test suivant.

- Si la condition1 est vraie, alors le bloc d'instructions1 est exécuté et on sort du test lorsque l'***indentation*** se termine (les lignes elif et else ne sont alors pas traitées).
- Si la condition1 est fausse, le script passe au test elif condition2 : à nouveau, le bloc indenté n'est traité que si la condition2 est vraie puis le test se termine.

Un exemple de structure if … else

```
x = 4
if x > 0 :
      print("x positif")
elif x < 0 :
      print("x négatif")
else :
      print("x nul")
```

V- La structure de boucle bornée : FOR

La boucle **for** est une structure élémentaire en programmation, qui permet de gérer la répétition d'un même bloc d'instructions. **Cette structure est utilisée quand le nombre de répétitions (ou nombre d'*itérations)* est connu à l'avance**. Les boucles for utilisent des variables, appelées ***compteur de boucle***, qui changeront de valeur à chaque itération (le compteur est ***incrémenté*** à chaque itération).

Exemples d'instruction	Explication
for j in [5, 7, 3, 2] : print(j)	Le **compteur** j est initialement affecté de la valeur 5. L'instruction (indentée) est ensuite réalisée (ici un simple affichage de j) puis la variable j prend la valeur suivante 7. etc. La boucle s'arrête quand la liste de j a été entièrement parcourue → renvoie 5 7 3 2
for i in range(3,10) : print(val)	Le compteur i est initialement affecté de la valeur 3. Une fois le bloc d'instructions exécuté une première fois, la variable i est **incrémentée** et prend la valeur de l'**entier** suivant : 4. **La boucle s'arrête quand le compteur atteint la valeur (n-1), l'avant dernière,** ici 9. L'instruction range(3,10) s'apparente donc au tableau de valeurs : [3, 4, 5, 6, 7, 8, 9] que balaie j. Il est possible de définir un pas (entier) en 3ème paramètre.

Un exemple de boucle for

```
Z=[]
for i in range(0,11):
    z = 2*(i/10)
    Z.append(round(z,2))
print(Z)

Y=[]
t=0
```

```
for i in range(0,11):
    y = 2*t
    Y.append(round(y,2))
    t = t + 0.10
print(Y)
```

VI- La structure de boucle non bornée : WHILE

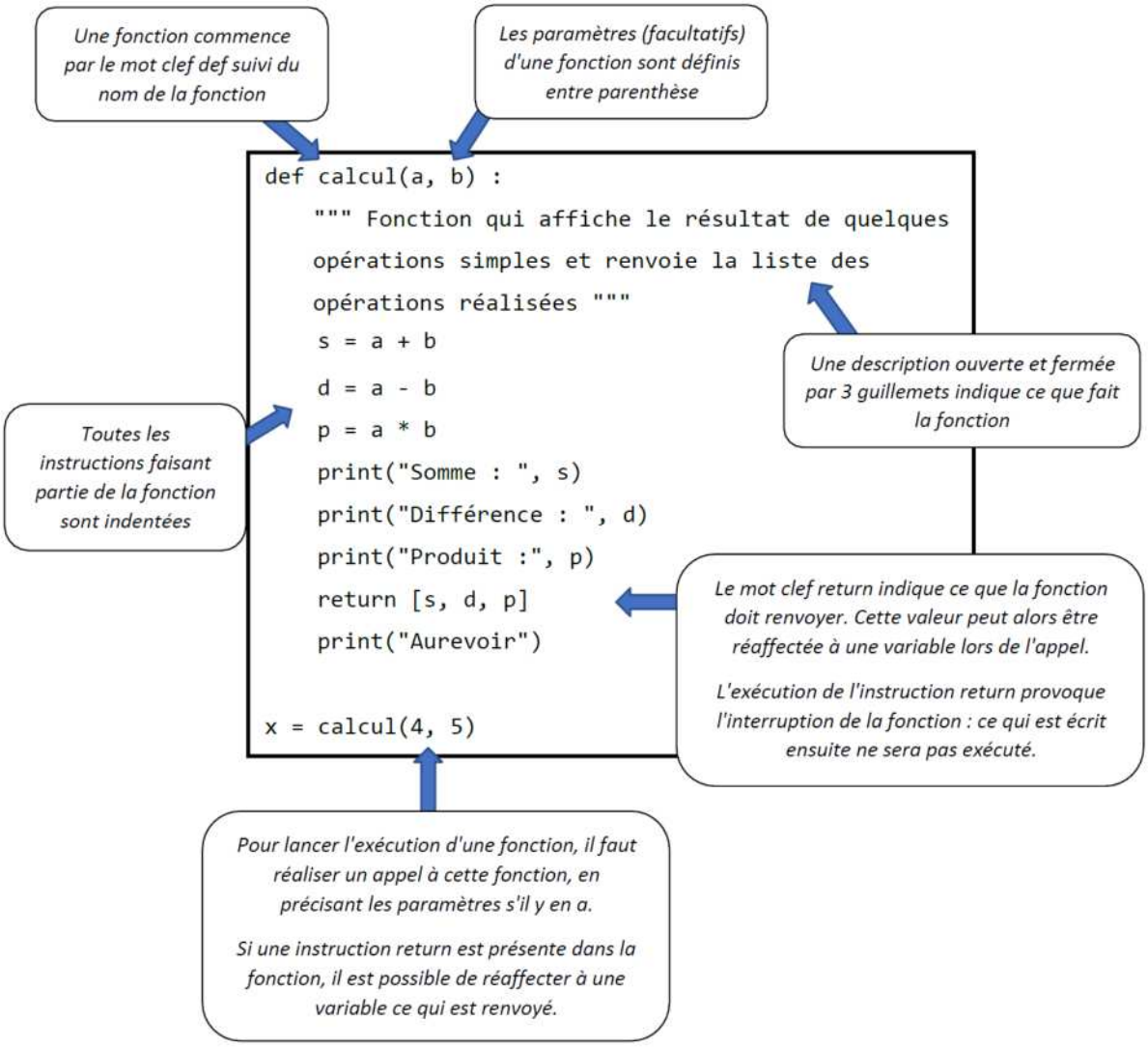
La **boucle while** est une structure utilisée quand un bloc d'instructions doit s'exécuter tant qu'une certaine condition, appelée ***condition d'exécution*** ou **condition d'arrêt**, est vérifiée. Dans ce type de boucle, le compteur doit être défini manuellement avant la boucle et doit ensuite être réactualisé avant la prochaine itération.
Si la condition d'arrêt est mal posée et si celle-ci est toujours vraie, la boucle peut continuer indéfiniment : on parle alors de **boucle infinie**.

Exemples	Explication
c = 1 while c <= 10 : print("Itération numéro",c) c = c + 1 print ("Sortie de boucle : c vaut",c)	Le **compteur**, ici appelé c, est initialisé avant la boucle. La boucle while est exécutée tant qu'une condition est vérifiée : ici tant que $c \leq 10$. A chaque itération, les instructions sont exécutées et le compteur c est incrémenté de 1. Quand c atteint 11, la condition d'arrêt n'est plus vérifiée et la boucle s'interrompt.
i = 9 while i != 0 : print("i vaut", i) i = i – 2 print("Ne sera jamais exécuté !")	Dans ce cas, le compteur i vaut initialement 9 puis il est décrémenté de 2 à chaque itération (i vaudra donc 9, 7, 5, 3, 1, -1, -3, -5…) La condition d'arrêt est mal posée car i est toujours différent de 0 : la boucle est donc infinie. **Une fois le script lancé, le seul moyen de l'interrompre est d'interrompre le script (carré rouge ou raccourci CTRL+C)**

VII- Les fonctions

A) Définir une fonction :

En programmation, on utile des fonctions pour regrouper un ensemble d'instructions en une seule. On peut ainsi imaginer les fonctions comme des objets dont l'exécution va provoquer l'exécution d'un ensemble d'autres instructions qui auront été définies auparavant.
Pratiquer cette ***modularité*** dans un script permet de gagner en lisibilité, de gagner du temps lors du débogage mais aussi de réutiliser simplement et rapidement des suites d'instructions.
Prenons immédiatement un exemple pour illustrer l'écriture d'une fonction :



Partie C : Les bibliothèques scientifiques

Les ***bibliothèques*** sont des ensembles de scripts (que nous appellerons ***modules)*** rédigés par un tiers, mettant à disposition de nouveaux outils qui n'étaient pas présents. Par exemple, la bibliothèque math mettra à disposition un outil « sqrt » qui permettra de calculer en une instruction la racine carrée d'un nombre. Le nombre de bibliothèques et de modules disponibles en Python est colossal et permet de faire à peu près tout ce qu'il est possible d'imaginer : tracer, gérer des documents html, des bases de données, représenter des séries de valeur, faire du calcul formel, faire du traitement de l'image, etc.
Nous ne nous intéresserons ici qu'aux modules ayant un intérêt dans le cadre d'un enseignement de Physique- Chimie au lycée et à certaines de leurs "fonctionnalités".

Pour pouvoir utiliser les objets mis à disposition par une bibliothèque, il est tout d'abord nécessaire de l'importer. Cette importation peut se faire de différentes manières.

- Solution 1 : Effectuer l'instruction `import.`
  Pour exemple la bibliothèque math, contenant une fonction sqrt() permettant de calculer la racine carrée d'un nombre est importée par l'instruction `import math` placée en début de script. Toutes les fonctions, tous les modules et toutes les constantes de la bibliothèque math sont alors importés. Toutefois, il sera nécessaire de préciser à chaque fois qu'une fonction est utilisée que celle- ci se trouve dans la bibliothèque importée.
  Ainsi, il faudra entrer l'instruction `math.sqrt(25)` pour pouvoir récupérer la valeur 5.0
- Solution 2 : Effectuer l'instruction `from math import *`, qui permet d'appeler la bibliothèque une fois pour toutes, sans nécessairement préciser le nom de la bibliothèque devant les fonctions qu'on appelle *(Mise en garde : il existe des fonctions intégrées dans 2 bibliothèques qui n'ont pas toutes à fait les mêmes propriétés, risque de conflit)*

Si le nom de la bibliothèque est trop long, il est possible d'utiliser un alias, c'est-à-dire une forme contractée du nom qui viendra la remplacer lorsque vous y ferez appel.

IX - La bibliothèque numpy

La bibliothèque numpy est une bibliothèque permettant un traitement numérique rapide et efficace des données scientifiques. Elle donne accès à de nouveaux types de tableaux (multidimensionnels), notamment le type **array**, pratiques pour la gestion de vecteurs, le calcul matriciel et le travail sur les polynômes. Elle permet également la gestion de routines de haut niveau (fonctions spéciales, outils statistiques, algèbre linéaire, etc.).

List ou Array ?
La majorité des situations rencontrées au lycée peut être gérée avec des variables de type list, toutefois le type array peut se révéler très utile dans certaines situations :

- Pour construire rapidement un tableau de valeurs réparties uniformément (temps ou positions).
- Pour trouver rapidement les zéros d'un polynôme ou modéliser une courbe.
- Pour réaliser des opérations simples et rapides avec les vecteurs (somme vectorielle par exemple).

Un choix pédagogique est ici à faire entre :

- Se limiter à utiliser le type list : ayant pour avantage d'être également utilisé en math et en SNT et généralement assez bien compris des élèves, mais alourdissant grandement certains calculs portant sur des tableaux de valeurs.
- Se lancer dans l'utilisation du type array : ayant pour avantage de faire gagner beaucoup de temps et de lisibilité (et incontournable en prépa/IUT), mais ajoutant une difficulté syntaxique supplémentaire.

Conclusion : Cette bibliothèque n'est pas abordée dans les programmes de maths et de SNT (donc petite perte de temps pour l'introduire) mais elle facilite grandement la manipulation des tableaux

Instructions	Explication
import numpy as np	Permet d'importer la bibliothèque numpy et d'utiliser l'alias np
T = np.array([[a1, a2], [b1, b2], [c1, c2]])	Permet de construire un tableau T de type array en le définissant ligne par ligne : $\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{pmatrix}$
T = np.array(L) **L = list(T)**	**Permet de transformer immédiatement une list L en tableau T de type array.** **Inversement, transforme un tableau de type array en liste.** **Remarque : on peut garder le même nom. L'instruction T=list(T) change la liste T en un tableau de même nom.**
T + MT – M T * k T / k	Attention, un tableau array ne se comporte pas comme une liste : les opérations sont réalisées terme à terme et il n'est pas possible de concaténer directement 2 tableaux. Ainsi, l'instruction T + M somme chacun des termes de T et M. De même T* k multiplie chaque élément du tableau T par k.
np.linspace(a, b, n)	**Crée un tableau de n valeurs entre a et b (incluses), réparties de façon homogène.**
np.arange(a, b, h)	Crée un tableau de valeurs entre a et b (exclue), espacées d'un pas de h
T.size	Renvoie le nombre d'éléments contenus dans le tableau T
np.zeros(n)	Crée le tableau de taille n ne contenant que des 0
np.ones(n)	Crée le tableau de taille n ne contenant que des 1
np.zeros([n,p])	Crée la matrice de taille $n \times p$ ne contenant que des 0
p = np.poly1d([an, …,a1, a0])	Affecte à la variable p le polynôme $P(x) = a_n x^n + \cdots + a_2 x^2 + a_0$
np.roots([an, …,a1, a0]) *ou* np.roots(p)	Renvoie les racines du polynôme $P(x) = a_n x^n + \cdots + a_2 x^2 + a_0 = 0$
np.polyfit(x, y, n)	**Renvoie un tableau contenant les coefficients du polynôme d'ordre n modélisant au mieux la fonction y = f(x).**
np.sqrt, np.cos, np.sin, np.tan, np.exp, np.log, np.pi, np.e, …	Les fonctions et les constantes mathématiques usuelles … **Ces fonctions (numpy) acceptent également des tableaux (list ou array) en argument**, *par exemple np.sqrt(L) pour calculer les racines carrées de tous les éléments de la liste L* !

X - La bibliothèque math

La bibliothèque math est utilisée pour avoir accès aux fonctions et constantes mathématiques élémentaires (racine carré, fonction gamma, exponentielle, logarithme, constante pi, fonctions trigonométriques, conversion degré/radians, etc.).
La plupart de ces fonctions est également présente dans la bibliothèque numpy (voir paragraphe suivant)

Instructions	Explication
import math	Permet d'importer et d'utiliser les objets de la bibliothèque math
math.sin(x) math.cos(x) math.tan(x) math.asin(y) math.acos(y) math.atan(y)	Permet d'utiliser les fonctions trigonométriques cosinus, sinus, tangente, arc sinus, arc cosinus et arc tangente. Attention, les valeurs de x pour les fonctions sinus, tangente et cosinus sont nécessairement **en radians**.
math.exp(x)	Permet d'obtenir l'exponentielle de x : e^x
math.log(x, n) math.log10(x) math.log2(x)	Permet d'obtenir le logarithme de x en base n. Par défaut math.log(x) est en base e. Permet d'obtenir les logarithmes en base 10 ou 2 de x (plus précis).
Math.e math.pi math.inf et - math.inf	Permet d'utiliser les constantes e et π Permet de travailler avec $+\infty$ et $-\infty$
math.sqrt(x)	Renvoie la racine carrée de x
math.hypot(x, y)	Renvoie la norme euclidienne $\sqrt{x^2 + y^2}$
math.degrees(x) math.radians(x)	Renvoie la conversion de l'angle en degrés (x étant en radians). Renvoie la conversion de l'angle en radians (x étant en degrés).

XI - La bibliothèque matplotlib

La bibliothèque matplotlib est utile pour la représentation, en 2D ou 3D, de fonctions mathématiques avancées ou d'ensembles de valeurs numériques. Elle possède de nombreux modules mais celui qui nous intéresse dans le cadre d'un enseignement au lycée est pyplot.

Instructions	Explication
import matplotlib.pyplot as plt	Permet d'importer la bibliothèque pyplot et d'utiliser son alias plt
plt.plot(x, y)	Trace la courbe $y = f(x)$. Paramètres optionnels : color : "black", "red", "blue", … modifie la couleur du tracé linewidth : "1", "2", "3" … modifie l'épaisseur du tracé linestyle :"-", "--", ":", "." modifie l'apparence du tracé marker : "+", "x", ".", "o", "v" modifie la forme des points marqués label : "courbe"1", "courbe2", … indique une légende pour chaque courbe. L'instruction plt.legend() doit figurer dans le script. *Remarque :* Il est possible de condenser les paramètres : plt.plot(x, y, "b:x") permet d'obtenir une ligne pointillée bleue avec des points en croix
plt.show()	Affiche les courbes mises en mémoire avec plt.plot(). Si plusieurs courbes étaient en traitement, elles sont superposées.
plt.grid()	Rend visible le quadrillage sur la figure
plt.legend()	Permet d'afficher les labels en légende lorsqu'ils ont été indiqués en paramètre label dans plt.plot()
plt.xlabel("votre texte", fontsize=12) plt.ylabel("votre texte", fontsize=12)	Affiche le "texte" en abscisse ou en ordonnée et précise sa taille.
plt.title("texte", fontsize=12)	Affiche "texte" comme titre de courbe et indique sa taille
plt.axis([x0, x1, y0, y1])	Limite l'affichage en abscisse sur $[x_0, x_1]$ et en ordonnées sur $[y_0, y_1]$ plt.axis("tight") permet de gérer le cadrage automatiquement plt.axis("equal") permet d'obtenir un repère orthonormé.
plt.xlim(a,b) plt.ylim(a,b)	Remplace l'instruction plt.axis() en indiquant l'intervalle $[a, b]$ utilisé pour le tracé en abscisses ou en ordonnées.
plt.text(x, y, "texte")	Permet d'afficher un texte sur la figure à la position (x, y)
plt.savefig("name")	Permet de sauvegarder la figure obtenue sous le nom "name".
plt.arrow(x, y, dx, dy)	Trace une flèche ayant comme point de départ (x, y) et allant jusqu'au point (x+dx, y+dy). Paramètres optionnels (mais indispensables en méca…) : color : "black", "red", "blue", etc. head_length : longueur de la tête de flèche, par défaut 0.0045 head_width : épaisseur de la tête de flèche, par défaut 0.003$