# Initiation au langage PYTHON

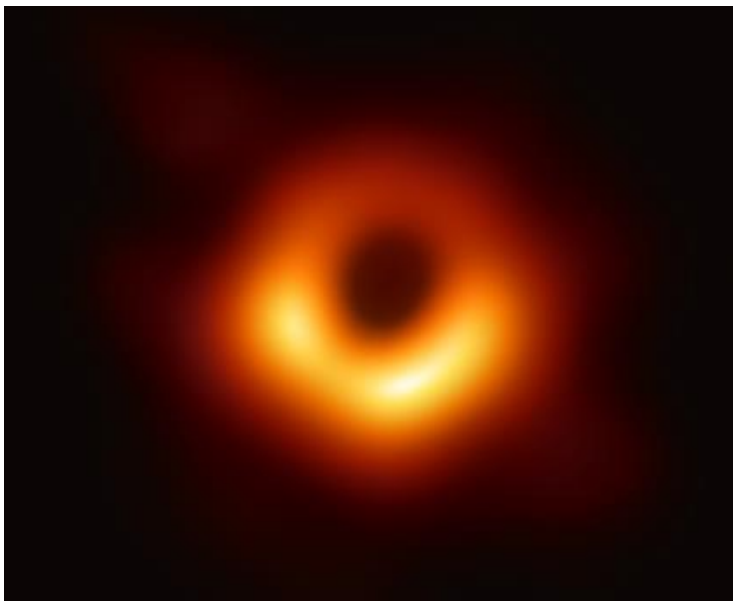
## Introduction

La programmation permet de créer ses propres outils grâce à une succession d'instructions. Pour communiquer ces instructions à l'ordinateur, il existe de nombreux langages qui le permettent. Le Python est un langage de programmation qui est libre et gratuit. Par ailleurs, comparativement à d'autres langages, le Python est assez proche d'un langage naturel.

Ce document vous propose de vous initier à ce langage. Vous allez aborder les bases pour connaitre le socle indispensable à toute programmation.

A part pour la première étape, vous trouverez des exemples et des exercices en lien direct avec la physique ou la chimie. Vous penserez certainement que la plupart des exercices ne nécessitent pas d'utiliser Python pour les résoudre. Vous avez raison car chaque exercice est délibérément très court et ciblé sur une seule notion de Python. Sur des projets plus importants, Python se révèlera simple et puissant.

Les parties « Si vous voulez aller plus loin » se destinent aux enseignants qui ont déjà quelques connaissances en programmation avec un autre langage et qui désirent en savoir un peu plus sur Python. Ces informations ne sont pas nécessaires pour les activités du programme de Seconde et de Première. C'est pourquoi je vous déconseille de les lire si vous n'avez jamais programmé. Si vous en avez envie, vous pourrez toujours les lire plus tard.



*Première photo d'un trou noir, publiée en Avril 2019. Elle est issue d'un traitement codé en Python.*

## Etape n°1 : Ecran

**Ce que vous allez apprendre :**

- Afficher à l'écran un nombre ou une information

**Mots-clés**

- print
- Symbole " ( ) ,

**Activités où cette notion pourra être utilisée :**

- La plupart des activités faites aujourd'hui pour apprendre à programmer.
- 1ère : Activité sur la composition de l'état final d'une transformation chimique totale.

Pour afficher du texte à l'écran, on utilise la fonction **print**. Par exemple, pour afficher le mot *Coucou*, on tape l'instruction **print** suivie de Coucou entre guillemets et entre parenthèses. Les parenthèses que l'on met après une instruction permettent de communiquer des paramètres. Les guillemets définissent qu'il s'agit d'un mot ou d'une phrase.


*Taper la ligne de code ci-dessous :*

```
print("Coucou")
```

*Cliquer sur Exécuter.*
*Recommencer pour afficher à l'écran « Vive la Physique Chimie ! ».*

Si on veut afficher plusieurs phrases avec un seul print, on les sépare par le symbole virgule :

```
print("Coucou", "Guillaume est là")
```

Par défaut, chaque phrase est séparée par le caractère espace.

**Si vous voulez aller plus loin :**

- Si vous souhaitez afficher des guillemets au sein de votre phrase, il faut faire précéder ce caractère par un anti slash \ . Par exemple `print (" Il dit \"Coucou\" à Valérie. ")`
- Vous pouvez aussi utiliser des apostrophes ' à la place des guillemets ".
- `print ( "A" , "B", "C", sep="-")` permet d'afficher A-B-C.

Pour information, on peut ajouter des commentaires à son code. Ces commentaires sont ignorés par Python et sont destinés à un lecteur humain. On note des informations ou des explications. Ainsi, le code est plus facilement compréhensible si on le reprend plus tard. Ils sont aussi utiles pour un autre codeur qui lit notre code. Pour cela, on fait précéder le commentaire du symbole dièse #

Dans l'exemple suivant, Python va exécuter les trois *print* et ignorer les deux commentaires.

```
#Affiche les noms
print("Dupont")
print("Durand") # Durand apparaîtra sur une deuxième ligne
print("Durant")
```

## Etape n°2 : Stockage

**Ce que vous allez apprendre :**

- Stocker une information. Cela peut être un nombre, une phrase, une liste de mesures, ….

**Mots-clés**

- Variable
- Symbole =

**Activités où cette notion pourra être utilisée :**

- Toutes les activités de Seconde
- Toutes les activités de Première

Une variable permet de stocker des informations (nombre, phrase, etc…). Par exemple pour affecter la valeur 50 à la variable nommée x, il faut taper :

```
x = 50
```

Attention, si vous voulez taper un nombre à virgule, il faut utiliser le caractère point car c'est un langage anglais :

```
x = 50.4
```

On peut aussi stocker une phrase :

```
Citation = "Euréka ! "
```

Le nom d'une variable doit être un ensemble de caractères majuscule/minuscule/chiffre et le caractère underscore _ Le nom ne doit pas commencer par un chiffre. Attention, Python est sensible à la casse(minuscule/majuscule), autrement dit les variables appelées citation et Citation sont deux variables différentes.

Pour s'y retrouver facilement, on a toujours intérêt à choisir des noms qui informent sur le contenu. Il vaut mieux nommer position_x plutôt que mavariable.

Nous avions vu la fonction pour afficher à l'écran `print("phrase")`. Quand on veut écrire le contenu d'une variable, il ne faut plus les guillemets, même si la variable contient une phrase. Par exemple `print(x)` ou `print(Citation)`.

Essayons d'afficher à l'écran x (voiture ) = 50 m à partir des deux variables `position_x` et `nom_mobile.`

```
position_x = 50
nom_mobile = "voiture"
print("x (", nom_mobile,  ") =",  position_x, "m")
```

Exercice : 2

*Créer la variable Ec et taper les instructions pour afficher  Ec = 15 J*

**Si vous voulez aller plus loin :**

- Certains mots anglais sont des mots clés réservés au langage Python. Ils sont donc interdits pour nommer des variables : if, else, for, def,,…
- Python a besoin de savoir quel est le type de données de chaque variable. Par exemple, il existe les nombres entiers, les nombres à virgule, les chaines de caractères. Dans les programmes simples, Python « devine » le type. Si c'est un nombre sans virgule, il en déduit que c'est un entier. Si ça commence et se termine par des guillemets, c'est une chaine de caractères, etc…
- Python propose de transformer un type de variable en un autre. Par exemple "14" est une chaine de caractère. Si vous souhaitez que ce soit un nombre entier, on utilise `x=int("14")`
- On peut faire de même avec `float()` pour obtenir un nombre à virgule. On utilise `str()` pour obtenir une chaine de caractères. Il est intéressant de transformer ces types pour les opérations du chapitre suivant.

## Etape n°3 : Les calculs

**Ce que vous allez apprendre :**

- Effectuer une opération

**Mots-clés**

- Opération
- Symboles `+ - * / **`

**Activités où cette notion pourra être utilisée :**

- Toutes les activités de Seconde
- Toutes les activités de Première

Comme le nom « variable » l'indique, ces données peuvent être modifiées grâce à des opérations. Sans cela, les variables perdraient beaucoup de leur intérêt. Sur les nombres, on peut, entre autre, faire addition +, soustraction -, multiplication *, division /, puissance ** . On peut aussi utiliser les parenthèses pour gérer les priorités.

Le programme suivant affiche la puissance d'une résistance.

```
Tension = 3
Intensite = 0.2
P = Tension*Intensite
print (P)
```

Comme vous le voyez, on ne précise pas les unités donc l'élève doit bien réfléchir aux unités avant de programmer les calculs. Si l'intensité est en mA, on pourra effectuer la conversion grâce à Intensite=Intensite/1000

**Si vous voulez aller plus loin :**

- L'opérateur + appliqué à des chaines de caractères permet de concaténer(joindre) des chaines de caractères. Par exemple si on tape `phrase = "Coucou "+"Guillaume "+" et Christophe"`, alors la variable `phrase` contiendra "Coucou Guillaume et Christophe"
- L'opérateur * appliqué à des chaines de caractères permet de multiplier une chaine. Par exemple si `rire = "ah"*4`, rire contiendra "ahahahah"
- Il existe aussi la division entière // et le reste d'une division %

Exercice : 3

*1°) Affecter à des variables les valeurs d'une vitesse(12m/s) et d'une masse (3.5kg). Calculer et afficher la valeur de la quantité de mouvement suivie de kg.m/s*

*2°) Au sein du même programme et avec les mêmes valeurs, calculer et afficher la valeur de l'énergie cinétique suivie de son unité.*

## Etape n°4 : Les mesures en Physique Chimie

**Ce que vous allez apprendre :**

- Stocker en mémoire des séries de mesures.

**Mots-clés**

- Liste
- Symbole [ ]

**Activités où cette notion pourra être utilisée :**

- Toutes les activités de Seconde
- Toutes les activités de Première sauf celle portant sur la chimie

En Physique, quand nous faisons de l'acquisition de données, nous avons de nombreuses valeurs. Il n'est pas envisageable d'utiliser les variables x1, x2, x3, x4, x5, etc…..car on pourrait avoir rapidement des centaines de variables à saisir et à gérer.

Python propose un autre type de variable : les listes.

Le tableau ci-contre va être déclaré en Python grâce à l'instruction suivante :

| Numéro de la mesure | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Abscisse x(m) | 0,5 | 6,9 | 2,4 | 5,1 | -3 |

X = [0.5, 6.9, 2.4, 5.1, -3]

Remarque : comme on l'a déjà vu avec la notation anglaise, les virgules ont été remplacées par un point.

**La virgule est utilisée pour séparer les valeurs.**

Pour accéder à une valeur de la liste, on place entre crochets ce qu'on appelle l'**indice** qui correspond au numéro de la mesure. Par exemple : x[2]

**Mais attention** : l'indice de la première mesure n'est pas 1 mais 0. Donc print(x[2]) affichera 2.4 et pas 6.9

Vous pouvez vous représenter la liste x avec le tableau suivant :

| Indice | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Abscisse x(m) | 0,5 | 6,9 | 2,4 | 5,1 | -3 |

Exercice : 4

*Déclarer une liste nommée v avec ces différentes valeurs puis afficher la troisième mesure :*

| Numéro de la mesure | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Vitesse(m/s) | 0 | 0,4 | 1,2 | 2,8 | 6 | 12,4 |

Si vous avez fait l'acquisition de plusieurs grandeurs simultanément, par exemple abscisse et ordonnée d'un mobile, le plus simple est de déclarer 2 listes : x et y. Il faudra 3 listes s'il y a trois grandeurs, etc…

Exercice : 5

*Déclarer les trois listes : temps, Ec et Ep. Calculer l'énergie mécanique quand temps vaut 5s*

| Temps (s) | 0 | 5 | 10 |
|---|---|---|---|
| Ec (kJ) | 4,3 | 3,1 | 1,9 |
| Ep(J) | 0 | 1200 | 2400 |

**Si vous voulez aller plus loin :**

- Si on tapait le code suivant : Em=Ec+Ep
  Ça ne fonctionnerait pas car l'opérateur + entre deux listes sert à les concaténer(joindre). Donc Em vaudrait [4.3,3.1,1.9,0,1200,2400]
- Et en respectant les unités Em=1000 * Ec+ Ep, Em vaudrait [4.3, 3.1, 1.9, 4.3, 3.1, 1.9, 4.3, 3.1, 1.9, 4.3, 3.1, 1.9, 4.3, 3.1, 1.9, 4.3, 3.1, 1.9, …..(en tout, 1000 fois la liste Ec)…, 0, 1200, 2400]

Vous verrez dans l'étape n°6, une technique pour traiter l'ensemble des valeurs d'une liste et pas seulement pour un indice précis.

Essayons maintenant d'apprendre comment ajouter des valeurs à une liste déjà existante.

Ajouter le code suivant à la fin de l'exercice précédent pour ajouter le temps 15s à la liste temps : temps[3] = 15

Exécuter le programme. Vous avez obtenu le message d'erreur « IndexError: list assignment index out of range ». Voici l'explication. En déclarant temps au tout début, il y avait 3 valeurs : 0, 5 et 10. donc l'indice ne peut valoir que 0, 1 ou 2. En choisissant 3, vous êtes en dehors des indices possibles. Il nous faut donc à présent une méthode pour rajouter des valeurs à une liste. Pour cela, il faut utiliser **append** comme dans l'exemple suivant :

```
temps.append(15)
```

**Si vous voulez aller plus loin :**

- Une fonction est une suite d'instructions que l'on exécute grâce à son nom. Par exemple, `print()` est une fonction.
- **append** est une **méthode**. Une méthode est une fonction qui est ici associée aux listes. C'est pourquoi, il ne faut pas faire `append(15)` mais `temps.append(15)` Cette écriture est typique de ce qu'on appelle la programmation orientée objet. Pour les besoins de Python en seconde et première, il n'est pas utile d'aborder d'autres aspects de la programmation orientée objet mais si vous souhaitez en savoir davantage, vous trouverez des adresses à l'étape 9.
- Il existe une autre technique pour « ajouter » des valeurs mais cela nécessite de connaitre à l'avance le nombre final de mesures.
  `serie_mesure=[0]*50` #permet d'obtenir une liste avec 50 fois la valeur 0
  Dans cet exemple, on peut ensuite modifier les valeurs 0 jusqu'à `serie_mesure[49]`


*Avec append, rajouter les mesures suivantes à l'exercice précédent : 15 s pour temps, 0,7 kJ pour Ec et 3600 J pour Ep. Afficher à l'écran la valeur de Em quand le temps vaut 15s.*

Taper au clavier les différentes valeurs d'une liste peut vite être fastidieux s'il y a beaucoup de mesures. Vous découvrirez dans l'étape n°8, une technique pour récupérer de manière automatique des données enregistrées dans un fichier au format .csv C'est un format disponible depuis la grande majorité des logiciels d'acquisition.

## Etape n°5 : Vrai ou faux ?

**Ce que vous allez apprendre :**

- Faire exécuter des instructions en fonction de certaines conditions.

**Mots-clés**

- if
- else
- Symbole >, <, >=, <=, ==, !=, :
- indentation

**Activités où cette notion pourra être utilisée :**

- 1ère : Activité sur la composition de l'état final d'une transformation chimique totale.

Dans certaines situations, les instructions devront dépendre de certaines conditions. Par exemple, si on veut déterminer quelle est la vitesse la plus élevée pour deux objets a et b, sachant que Va=3m/s et Vb=6km/h. Il faudra aussi mettre cette valeur maximale dans une variable et l'afficher.


Lire le code suivant, le taper sans les commentaires. Vous trouverez des explications détaillées après le code.

```
Va = 3
Vb = 6
Vb = Vb*1000/3600
if Va > Vb : # Si Va est > à Vb alors faire toutes les instructions qui sont indentées(décalées) ci-dessous
      print("L'objet a est plus rapide que l'objet b ")
      Vmax = Va
else : # Sinon, faire toutes les instructions qui sont indentées ci-dessous
      print("L'objet b est plus rapide que l'objet a ")
      Vmax = Vb
print("Cette vitesse maximale est de ", Vmax, " m/s")
```

- Il est indispensable de faire suivre la condition Va>Vb par le symbole deux points.
- Il est aussi nécessaire que toutes les instructions à faire si Va est bien supérieure à Vb, soient indentées(=décalées). Cela peut être une indentation(décalage) automatique grâce à votre éditeur de code. Vous pouvez aussi indenter grâce à la touche tabulation ou espace.
- Par contre, le « `else :` » n'est pas indenté, il doit être aligné avec `if`.
- La dernière ligne qui affiche la vitesse maximale n'est pas indentée parce qu'on veut qu'elle soit toujours exécutée.

Faire varier les valeurs de Va ou de Vb pour que Vb soit plus élevée que Va.

Les différents symboles de tests de conditions :

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

**Si vous voulez aller plus loin :**

- Dans la situation où Va est égal à Vb, l'ordinateur affichera que l'objet b est plus rapide que l'objet a. Il faudrait ajouter un test supplémentaire. Pour cela, nous allons utiliser elif qui est la contraction de « else if » (sinon si)

```
if Va > Vb :
        print("L'objet a est plus rapide que l'objet b.")
elif Va == Vb :
        print("L'objet a va à la même vitesse que l'objet b.")
else :
        print("L'objet b est plus rapide que l'objet a.")
```

On peut utiliser autant de elif qu'on le souhaite mais il ne peut y avoir qu'un seul else (ou pas du tout) à la fin.

Exercice : 8

*Après avoir défini les valeurs des coefficients stœchiométriques de deux réactifs r1=2 et r2=3, et leur quantité de matière n1=0,5mol et n2=0,6mol, calculer si l'avancement final est n1/r1 ou n2/r2 puis l'afficher.*

**Etape n°6 : Les tâches répétitives.**

**Ce que vous allez apprendre :**

- Exécuter un bloc d'instructions plusieurs fois

**Mots-clés**

- for
- in
- indentation
- Symbole :

**Activités où cette notion pourra être utilisée :**

- L'activité de Seconde où il faut représenter les vecteurs vitesse
- Toutes les activités de Première sauf celles portant sur la chimie

L'ordinateur est très doué pour effectuer des tâches répétitives. Les boucles permettent d'exécuter plusieurs fois le même bloc d'instructions. Cela sera particulièrement intéressant pour les activités où il faut parcourir une liste et faire le même calcul pour chaque indice. Par exemple, calculons l'énergie cinétique pour une liste de vitesses. Vous trouverez l'explication du code en dessous :

```
vitesse = [4, 29, 5.3, 12, 8]
m = 3 #masse=3kg
for v in vitesse :
      Ec = 0.5*m*v**2
      print("A la vitesse de ", v, "m/s, l'énergie cinétique vaut", Ec, "J" )
```

Dans l'exemple précédent, la boucle **for** parcourt la totalité de la liste vitesse. A chaque indice, d'une part v reçoit la valeur de la vitesse pour l'indice donné et d'autre part, le bloc indenté en dessous de **for** est exécuté.

Donc dans un premier temps v vaudra 4, Ec sera calculé puis affiché avec v=4. Puis v vaudra 29, Ec sera calculé puis affiché avec v=29. Puis v vaudra 5,3 et ainsi de suite jusqu'à v=8. Et une dernière fois Ec sera calculé et affiché avec v=8.


*On mesure une liste de masses [1.2,2,3.5,4,8]. Afficher la phrase suivante pour chaque masse avec à la place des pointillés les valeurs adéquates : « Si un objet a une masse de …. kg alors son poids est …. N »*

Dans plusieurs situations du programme de Physique Chimie, on ne peut pas utiliser **for** comme dans cette technique. Prenons l'exemple d'un objet en déplacement à une dimension. On veut calculer les vitesses en ayant une liste d'abscisses x, une liste de temps. On calcule $v[i] = \frac{x[i+1]-x[i]}{t[i+1]-t[i]}$

Si on applique cette égalité pour le dernier indice i, on obtiendra une erreur car les valeurs x[i+1] et t[i+1] n'existent pas. Pour pallier cette difficulté, il existe plusieurs solutions. Dans le cas où l'on connait le nombre de mesures, on peut utiliser `range()` qui va permettre de choisir une plage d'indices. Lire le code, les explications détaillées se trouvent après.

```
x = [2, 4, 7, 11, 16, 22]   #il y a 6 valeurs
delta_t = 0.3
for i in range(5) : #i vaudra successivement 0,1,2,3,4
        vitesse = (x[i+1]-x[i])/(delta_t)
        print("A la position",x[i], "m la vitesse est de",vitesse, "m/s")
```

Il y a 6 valeurs d'abscisses. Comme nous venons de le voir, nous ne pouvons pas calculer la vitesse avec l'égalité précédente pour la dernière mesure (indice=5). Par contre, on peut calculer les vitesses pour les indices 0, 1, 2, 3 et 4. C'est exactement ce que permet  for i in range(5) . En effet range(debut,fin) permet d'aller de debut à fin-1 de un en un. Vous pouvez en avoir la confirmation en tapant

```
for i in range(5) :
        print(i)
```

Mais dans le cas où il y a beaucoup de mesures, on ne connait pas forcement l'indice maximal avec précision. La fonction len() permet de déterminer ce nombre de mesures. Dans l'exemple précédent, len(x) renvoie la valeur 6. On peut donc faire :

```
for i in range(len(x)-1) : # len(x)-1 vaut 5 dans cette situation
```

**Si vous voulez aller plus loin :**

- Prenons l'exemple d'un objet en déplacement à une dimension. On veut calculer les vitesses en ayant une liste d'abscisses x, une liste de temps. Cette fois, on souhaite calculer la vitesse avec :
  $v[i] = \frac{x[i+1]-x[i-1]}{t[i+1]-t[i-1]}$ (attention, ce n'est pas le calcul proposé dans le Programme)
  Si on applique cette égalité pour i=0, on obtiendra une erreur car i-1=-1. Or, un indice négatif désigne les éléments à partir de la fin de la liste : l'indice -1 désigne le dernier élément de la liste, l'indice -2 désigne l'avant dernier élément, etc… On peut à nouveau utiliser range() qui va permettre de choisir une plage d'indices. Lire le code, les explications détaillées se trouvent après.

  ```
  x = [2, 4, 7, 11, 16, 22]   #il y a 6 valeurs
  delta_t = 0.3
  for i in range(1,5) : #i vaudra successivement 1,2,3,4
        vitesse = (x[i+1]-x[i-1])/(2*delta_t)
        print("A la position",x[i], "m la vitesse est de",vitesse, "m/s")
  ```

  Il y a 6 valeurs d'abscisses. Comme nous venons de le voir, nous ne pouvons pas calculer la vitesse pour la première mesure(indice=0) et la dernière(indice=5). Par contre, on peut calculer les vitesses pour les indices 1, 2, 3 et 4. C'est ce que permet  for i in range(1,5) .

Nous allons maintenant aborder une activité en Première où il est nécessaire de remplir une nouvelle liste au sein d'une boucle. Il va donc falloir utiliser la fonction append. Mais il est impossible d'utiliser append avec une liste qui n'existe pas encore.

Prenons l'exemple d'un calcul d'une liste de poids à partir d'une liste de masses. Si l'on fait poids.append(9.81*m) au sein de la boucle, on aura un message d'erreur lors de la première exécution de la boucle. En effet, on demande d'ajouter une valeur à une liste poids alors que la liste poids n'existe pas encore. Il faut donc déclarer que poids est une liste avant la boucle for. Ce sera une liste vide (pour l'instant). Pour cela on tape :

poids=[] # Rien entre les crochets

Voilà ce que l'on obtient pour le code complet :

```
masse = [1.2, 2, 3.5, 4, 8]
poids = []
for m in masse :
      poids.append(9.81*m)
print(poids)  #pour afficher la liste des poids
```

Exercice : 10

*Je vous propose de faire une partie d'une activité de 1er. On connait une liste de vitesses et la liste des positions d'un objet qui tombe selon un axe vertical. Il faut déterminer la liste d'énergie cinétique, la liste d'énergie de position et la liste d'énergie mécanique.*

*vitesse_y = [-1, -1.9, -2.9, -3.9, -4.9, -5.8, -6.9] , position_y= [8.23, 8.09, 7.85, 7.5, 7.06, 6.52, 5.89] et masse = 3 kg*

**Si vous voulez aller plus loin :**

- Il existe en Python, une autre boucle beaucoup plus polyvalente que for. Il s'agit de **while** qui signifie **Tant que**.
  On l'utilise avec par exemple :

```
i=0
while Ec<5 and i<len(v):
     Ec=0.5*m*v[i]**2
     print("Pour la mesure n°",i, "l'énergie cinétique vaut",Ec, "J et elle est
inférieure à 5J")
     i=i+1
```

  Comme vous pouvez le voir la boucle while permet d'exécuter un bloc d'instructions, un nombre de fois non défini à l'avance mais qui dépend de certaines conditions. Ici, c'est une double condition séparée par `and`. Il existe aussi `or` et `not`. On aurait bien sûr pu mettre une seule condition mais il faut faire attention à ne pas provoquer de bug en créant une boucle infinie. C'est-à-dire une situation où les conditions sont toujours remplies.

## Etape n°7 : Presque comme des fonctions mathématiques

**Ce que vous allez apprendre :**

- Créer une fonction pour rendre son code plus lisible et léger.

**Mots-clés**

- def
- return
- Symbole :

**Activités où cette notion pourra être utilisée :**

- Tout dépend de ce que les professeurs de Mathématiques et de SNT vont exiger des élèves.

Une fonction est un ensemble d'instructions qu'on appelle grâce à un nom. print(parametre1,parametre2) est une fonction qui permet d'afficher à l'écran la valeur de parametre1 et de parametre2. On peut créer ses propres fonctions. Cela permet d'organiser et de rendre plus lisible son code.

Elles ne sont pas indispensables pour les activités de seconde et de première en Physique Chimie mais si des collègues de SNT ou de mathématiques insistent sur leur rôle, il y a de fortes chances que les élèves veuillent s'en servir.

Nous allons définir une fonction qui permet de calculer l'énergie cinétique. On va l'appeler Ec. Elle nécessitera deux paramètres la masse et la vitesse.

Pour la définir, il faut utiliser l'instruction def qui est l'abréviation de define. Ensuite, on écrit le nom de la fonction. Pour finir, entre parenthèses, on écrit les paramètres dont on aura besoin au sein de la fonction. Voici le code:

```
def Ec(m,v) :
       valeurEc=1/2*m*v**2   #On calcule la valeur de l'énergie cinétique avec du texte indenté
       return valeurEc
```

Le return renvoie valeurEc à l'endroit où a été appelée la fonction.


*Taper le code précédent pour définir la fonction. Ajouter les trois lignes suivantes et exécuter (vous trouverez les explications à la suite ) :*

```
vitesse=5
Ecinetique = Ec(3,vitesse)
print("Si m = 3 kg et v =",vitesse, "m/s alors Ec =",Ecinetique, "J")
```

Analysons la ligne `Ecinetique = Ec(3,vitesse)`. Python va exécuter la fonction Ec que vous aviez défini précédemment. Par ailleurs, m va recevoir 3 et v va recevoir la valeur de vitesse. Une fois que valeurEc est calculée, elle est renvoyée grâce à return vers la variable Ecinetique.

Exercice : 12

*Définir une fonction qui permet de convertir une température en degré Celsius, en Kelvin. L'utiliser pour convertir 21°C et afficher cette valeur en Kelvin.*

Pour information, une fonction peut aussi recevoir en paramètres des listes.

**Si vous voulez aller plus loin :**

- Une fonction peut avoir des paramètres mais ce n'est pas indispensable. Si vous n'en voulez pas, il faut mettre des parenthèses () avec rien entre elles.
- return n'est pas indispensable non plus. On peut faire par exemple une fonction qui se charge d'afficher certaines choses à l'écran mais ne retourne pas de valeur.

**Etape n°8 : Réinventer la roue ?**

**Ce que vous allez apprendre :**

- Utiliser des fonctions déjà existantes qui simplifient la vie du programmeur

**Mots-clés**

- Import
- as
- Toutes les fonctions dont vous retrouvez le récapitulatif dans l'étape suivante.

**Activités où cette notion pourra être utilisée :**

- Toutes les activités de Seconde
- Toutes les activités de Première sauf celle portant sur la chimie

Python possède une très grande bibliothèque de fonctions qui simplifient la vie du programmeur. Ces fonctions sont regroupées par modules. On peut y retrouver par exemple une fonction qui calcule un sinus, une fonction qui permet de lire les fichiers csv et encore beaucoup d'autres. Nous allons voir ici deux modules dont nous pouvons avoir besoin pour le programme de seconde et de première.

## Module math

Ce module permet d'accéder à un grand nombre de fonctions mathématiques (sinus, racine carrée, exponentielle, … ) Pour pourvoir utiliser ce module, il faut taper l'instruction :

```
import math   #import signifie importer
```

A présent, nous pouvons utiliser la fonction math.sin() qui permet de calculer un sinus, math.sqrt qui permet de calculer la racine carrée,…

```
a = math.sin(3.14159/2)
a = math.sin(math.pi/2)   # cette ligne revient au même que la précédente car la constante
nommée math.pi vaut 3.14159265…
b = math.sqrt(9)  # b vaut 3 car sqrt calcule la racine carrée
```

Exercice : 13

*Calculer et afficher le cosinus de π/4 grâce à math.cos()  et math.pi*

**Si vous voulez aller plus loin :**

- On peut aussi utiliser l'instruction :

```
import math as m #remplace le nom math par m
m.sin(0.5*m.pi) # au lieu de taper math. On utilise m. c'est plus court
```

- On peut aussi faire :

```
from math import *
#et ensuite on utilisera directement :
sin(0.5*pi)
```

Mais attention cette technique n'est pas recommandée. En effet, il ne doit pas exister des fonctions ou des variables qui portent le même nom que celles de math( cos, sin, pi, sqrt, etc…)

## Module csv

Voyons maintenant le module pour importer les mesures enregistrées dans un fichier csv (imaginons un fichier avec 3 colonnes pour le temps, l'abscisse et l'ordonnée).

```
import csv
source = open('fichier.csv', 'r') #r signifie que le fichier est ouvert en mode lecture, ça évite de le modifier par erreur
t,x, y = [], [], []   #déclare trois listes vides
for row in csv.reader(source,delimiter=';'): #parcourt le csv ligne par ligne. row est une liste contenant toutes les valeurs d'une ligne
    t1,x1, y1 = map(float,row) #sépare les valeurs de la liste et les « définit » en tant que nombre à virgule
    t.append(t1)   # ajoute t1 à la liste t
    x.append(x1)   # ajoute x1 à la liste x
    y.append(y1)   # ajoute y1 à la liste y
```

Ce morceau de code peut très bien être donné aux élèves pour qu'ils se concentrent sur l'algorithme de la partie Physique Chimie qui suivra cette création des listes.

Cette importation ouvre des perspectives sur une acquisition avec un logiciel tiers (ordi, tablette, smartphone) puis un traitement du csv avec python.

Si le fichier csv n'est pas enregistré dans le même dossier que votre code (fichier .py), vous devez préciser le chemin de votre fichier csv. Attention, si vous êtes sous Windows, il faut remplacer les antislashs \ par des doubles \\ ou par slashs / Par exemple, si votre fichier se trouve -> C:\Users\Travail\Python\fichier.csv, il faut taper :

```
source = open('C:/Users/Travail/Python/fichier.csv', 'r')
```

Attention, cette technique fonctionne uniquement s'il n'y a pas d'en tête avec les noms des grandeurs sur la première ligne de votre fichier .csv. Supprimez cette ligne avant de l'utiliser avec Python ou bien utilisez la technique proposée ensuite dans « Si vous voulez aller plus loin ».

**Si vous voulez aller plus loin :**

- Dans un fichier .csv, il arrive que la première ligne contienne les noms des grandeurs physiques. Il ne faut pas que ces données soient intégrées aux listes de valeurs. Pour cela, il faut remplacer la ligne de la boucle `for` par les trois lignes suivantes :

```
reader = csv.reader(source,delimiter=';')
next(reader, None) # cette ligne permet de passer la première ligne
     for row in reader:
```

- Si le séparateur dans le csv est une virgule, il faut utiliser `delimiter=','`

- Si pour une raison particulière, vous préférez que les élèves accèdent à un fichier .csv hébergé sur internet, il faut connaitre l'url du fichier puis faire :

```
import csv
import requests
source = requests.get('http://www.acamus.net/basket2.csv')
source=source.content.decode('utf-8')
source=source.splitlines()
t,x, y = [], [], []
for row in csv.reader(source,delimiter=';'):
    t1,x1, y1 = map(float,row)
    t.append(t1)
    x.append(x1)
    y.append(y1)
```

Mais ce n'est pas tout, des programmeurs ont créé des bibliothèques pour compléter celle de Python et répondre à des besoins particuliers. Certaines d'entre elles sont très abouties et reconnues dans le domaine scientifique. On va commencer par découvrir Matplotlib.

## Matplotlib

Matplotlib est une bibliothèque qui sert à tracer et visualiser des données. En effet, elle permet d'obtenir des graphiques complets et propres avec peu de lignes de code.

**Pour importer matplotlib, il faut taper :**

```
import matplotlib.pyplot as plt  # On importe matplotlib.pyplot sous le nom plt
```

Ainsi, toutes les fonctions seront appelées en les faisant précéder de `plt`.

**Voici quelques fonctions proposées par matplotlib, utiles pour les Secondes et les Premières :**

**Pour faire apparaître les axes du graphique :**

Il faut taper :

```
plt.axis([xmin, xmax, ymin, ymax])
```

xmin et xmax sont les valeurs minimales et maximales de l'axe des abscisses.

ymin et ymax sont les valeurs minimales et maximales de l'axe des ordonnées.

**Pour tracer une courbe enregistrée dans deux listes, une pour les abscisses et une autre pour les ordonnées :**

x est la liste des abscisses et y la liste des ordonnées. Pour tracer la courbe, il faut taper :

```
plt.plot(x, y, 'r+')
```

Le r de r+ signifie que l'on veut des points rouges (red) D'autres lettres (couleurs) sont disponibles, on peut utiliser par exemple b(lue), g(reen), c(yan), m(agenta), y(ellow), k(black), w(hite).

Le + de r+ signifie que l'on veut des points sous forme de plus (+). On peut aussi utiliser x pour avoir la forme du x, . pour avoir un petit point et o pour avoir un cercle. On peut aussi utiliser - pour avoir une courbe, -- pour avoir une courbe en pointillés. On peut combiner les points et les courbes. Par exemple `'r+--'`

**Pour afficher le graphique créé grâce à matplotlib :**

Une fois que le graphique est terminé, il faut l'afficher à l'écran en tapant :

```
plt.show()
```

Pour résumer, si on veut par exemple afficher la vitesse d'un objet en fonction du temps, il faudra taper :

```
import matplotlib.pyplot as plt
vitesse=[10, 15, 23, 45, 12]
temps=[0, 1, 2, 3, 4]
plt.axis([0, 5, 0, 50])
plt.plot(temps, vitesse, 'r+-')
plt.show()
```


*Grâce aux 3 fonctions précédentes (axis, plot et show), je vous propose de faire l'une des trois activités au programme de Seconde : « Représenter les positions successives d'un système modélisé par un point lors d'une évolution unidimensionnelle ou bidimensionnelle à l'aide d'un langage de programmation. »*

*Pour cela, tracer la trajectoire parabolique dont les positions sont : posx=[-50, -40, -30, -20, -10, 0, 10, 20] et posy=[-123, -78, -43, -17, -1, 4, 0, -13]*

**Pour dessiner des vecteurs :**

Pour tracer des vecteurs, on peut dessiner des flèches en tapant :

```
plt.quiver(x0, y0, deltax, deltay, angles='xy')
```

La commande précédente dessinera une flèche avec les caractéristiques suivantes :

- x0 et y0 sont les coordonnées du point de départ
- deltax et deltay sont les composantes du vecteur selon l'abscisse et l'ordonnée.
- angles='xy' permet d'obtenir une flèche qui a une direction cohérente avec l'échelle des axes. Donc c'est un paramètre à utiliser systématiquement en l'écrivant toujours tel quel.

Si à l'affichage, la taille de la flèche est trop faible ou trop importante, il faudra rajouter deux paramètres :

```
plt.quiver(x0, y0, deltax, deltay, angles='xy', scale=1, scale_units='xy')
```

Si vous voulez agrandir la flèche choisissez une valeur de `scale` inférieure à un. Si vous souhaitez rétrécir la flèche, choisissez une valeur de `scale` supérieure à 1.

Exercice : 15

*Faisons un autre exercice qui fait partie du nouveau programme de seconde : « Représenter des vecteurs vitesse d'un système modélisé par un point lors d'un mouvement à l'aide d'un langage de programmation. »*

*Reprendre les valeurs de posx et posy de l'exercice précédent puis tracer les vecteurs vitesse. Pour cela, vous pouvez calculer les listes de vitesse selon x et y (avec delta_t = 1s ) ou bien prendre les valeurs suivantes qui ont déjà été calculées : vitessex = [10, 10, 10, 10, 10, 10, 10, 10] et vitessey = [50, 40, 30, 21, 11, 1, -9, -19]*

*Pour information, vous aurez besoin d'une boucle for pour tracer tous les vecteurs.*

**Si vous voulez aller plus loin :**

On peut soigner un peu la présentation avec diverses fonctions :

- Pour nommer les axes :

```
plt.xlabel("Nom de l'axe x")
plt.ylabel("Nom de l'axe y")
```

- Pour mettre un titre au graphique :

```
plt.title("Le titre");
```

- Pour ajouter une légende aux courbes :

```
plt.plot(temps,vitesse, 'r+-', 'Vitesse ')   #on nomme la courbe Vitesse
plt.legend()  #la courbe rouge sera légendée avec le nom Vitesse
```

**Pour faire des animations :**

Une des activités de Première consiste à « Simuler à l'aide d'un langage de programmation, la propagation d'une onde périodique ». Il faut donc faire une animation à l'écran. Pour cela, il existe des fonctions de matplotlib destinées aux animations mais elles ne sont pas toujours simples à utiliser. Heureusement, Matplotlib permet une autre technique plus rudimentaire. Elle fonctionne bien dans des situations simples. Elle consiste à dessiner la figure, faire une pause puis effacer l'image. Il faut ensuite dessiner la figure suivante, faire une pause et l'effacer à nouveau. Et ainsi de suite… Pour cela, on utilise les fonctions :

```
plt.cla() # cette fonction efface l'image
…
…                  } ici, on dessine l'image
…
plt.pause(0.01)  # on fait une pause de 0.01 seconde
```

Exercice : 16

Dessiner une flèche qui tourne en recopiant le code suivant :

```
import matplotlib.pyplot as plt
import math
for i in range(100):
     plt.cla() #efface l'image
     plt.axis([-5, 5, -5, 5])
     plt.quiver(0, 0, 4*math.cos(i/10), 4*math.sin(-i/10),
angles='xy', scale=1, scale_units='xy')
     plt.pause(0.01) #la valeur de la pause
plt.show()
```

Exécuter le programme pour voir l'animation. (Si ça ne fonctionne pas, ajouter avant la ligne for : plt.ion(). Cela oblige l'affichage à se mettre à jour après chaque modification. )
Mettre la ligne plt.cla() en commentaire pour la désactiver et observer la différence

Exercice : 17

*Si vous avez le temps : faisons décoller une fusée. Réaliser une animation où une flèche monte verticalement.*

Exercice : 18

Voici maintenant l'animation attendue pour le niveau Première : « Représenter un signal périodique et illustrer l'influence de ses caractéristiques (période, amplitude) sur sa représentation. Simuler à l'aide d'un langage de programmation, la propagation d'une onde périodique. »

Il y a de nombreuses méthodes pour faire cet exercice mais nous allons faire ici une des plus courtes à taper. Commençons par faire les importations et les déclarations :

```
import matplotlib.pyplot as plt
import math

A = 1
longueurOnde = 300
#Attention, le mot « lambda » est déjà utilisé par Python donc on ne peut pas
appelé une variable lambda mais on peut utiliser le symbole λ=
Phi = 3.14
v = 40
T = longueurOnde/v
```

Nous allons faire avancer le temps grâce à une boucle for et à chaque temps, nous allons calculer les ordonnées de la courbe pour toutes les abscisses allant de 0 à x_max= v x t. Autrement dit pour chaque temps, on doit calculer en priorité jusqu'où l'onde s'est propagée.

```
for t in range(200): #le temps va évoluer de 0 à 199
        plt.cla()
        x_max = v*t #on calcule l'abscisse de la propagation
        y = [] #on initialise la liste des ordonnées
        x = [] #pareil pour les abscisses
        for xi in range(0, x_max):
             x.append(xi) # on ajoute xi à la liste des abscisses
             y.append(A*math.sin((2*3.14/longueurOnde)*xi-(2*3.14/T)*t+Phi))
             # dans la ligne précédente, on calcule l'ordonnée en xi
        x.append(1500) # on rajoute le point 1500,0 pour afficher un trait où
        l'onde ne s'est pas encore propagée
        y.append(0)
        plt.axis([-1, 1500, -2, 2])
        plt.plot(x, y, 'r-')
        plt.pause(0.01)

 plt.show()
```

Cet exercice est le plus compliqué des sept présents dans les programmes. Mais il permet de s'approprier les notions physiques de propagation qui sont derrière cette double variation de t et de x. Par ailleurs, une fois terminé, il est aisé de faire varier les caractéristiques (période, amplitude).

**Si vous voulez aller plus loin :**

- On peut bien sûr utiliser une fonction pour calculer les ordonnées. On peut mettre en paramètres seulement x et temps mais on peut aussi ajouter les constantes physiques.

```
def ordOnde(x,temps) : # la version avec seulement x et temps
    A = 1
    longueurOnde = 300
    Phi = 3.141
    v = 40
    T = longueurOnde/v
    Ordonnee = A*math.sin((2*3.14/longueurOnde)*x-(2*3.14/T)*temps+Phi)
    return ordonnee
```

Dans cet exemple avec les deux paramètres, nous n'étions pas obligés de déplacer la déclaration des constantes au sein de la fonction. Cependant, il est préférable qu'au sein d'une fonction toutes les variables proviennent soit d'une déclaration dans la fonction, soit des paramètres de la fonction.

- Pour vous montrer le grand nombre de possibilités pour faire cet exercice, voici une autre solution. Elle permet de calculer avec un pas différent de 1 pour les abscisses, contrairement à la solution de la page précédente.

```
xi=0 #xi va varier de 0 à x_max
dx=0.1 # c'est le delta x entre chaque calcul
x=[]
y=[]
    while xi < x_max :
        y.append(ordOnde(xi, t)) #il faut avoir déclaré la fonction ordOnde
        x.append(xi)
        xi = xi+dx
    y.append(ordOnde(x_max, t))
    x.append(x_max)
```

- On peut aussi utiliser une fonction de matplotlib dédiée à l'animation qui s'appelle FuncAnimation. Le programme sera plus complexe mais l'affichage sera plus fluide que la solution cla()+pause(0.01). Il sera aussi plus simple d'interrompre l'exécution du programme. Pour finir, il sera possible d'enregistrer l'animation sous forme d'une vidéo grâce à la ligne anim.save('monfilm.mp4', fps=30, extra_args=['-vcodec', 'libx264']) à condition d'avoir déjà installé **ffmpeg**.
  Pour plus d'informations, faites une recherche sur internet avec « python FuncAnimation ».
  Si vous souhaitez avoir le programme déjà tapé, vous pouvez le télécharger sur **http://www.acamus.net/animation.py**

**NumPy**

Numpy est une bibliothèque qui se destine à manipuler des tableaux de valeurs. Elle propose des fonctions mathématiques très élaborées. Elle permet donc de créer des programmes scientifiques en économisant des lignes de code. Par ailleurs, on peut faire des calculs sur tous les éléments d'une liste sans utiliser la moindre boucle. C'est pourquoi elle est très utilisée dans le domaine scientifique. Comme Matplotlib, elle fait partie des bibliothèques utilisées pour la photo du trou noir du mois d'Avril 2019.
Cependant, elle présente un inconvénient pour les élèves : elle a un effet « boite noire » qui peut éloigner des concepts de Physique et de Chimie. Il en est de même pour les mathématiques. Donc, il y a peu de chance que nos collègues de mathématiques utilisent cette bibliothèque avec leurs élèves.

C'est pourquoi dans ce document, je ne vous présenterai qu'une seule fonction de Numpy : elle s'appelle `polyfit` et elle fait une régression polynomiale.

Elle peut être utilisée pour l'activité de seconde : « **Représenter un nuage de points associé à la caractéristique d'un dipôle et modéliser la caractéristique de ce dipôle à l'aide d'un langage de programmation.** »

En choisissant la valeur 1 comme troisième argument de la fonction `polyfit`, on obtient une régression linéaire qui permet par exemple de modéliser la caractéristique d'un conducteur ohmique.

Imaginons que `I` soit la liste des intensités qui traversent un conducteur ohmique et U la liste des tensions entre les bornes de ce même dipôle.

```
import numpy as np
coeff = np.polyfit(I, U, 1) # régression linéaire
```

coeff est à présent une liste. La droite modélisée a pour équation : coeff[0] * x + coeff[1] . Attention, c'est bien coeff[0] qui est le coefficient pour la puissance la plus élevée.

Donc pour tracer cette droite modélisée, on peut taper :

```
plt.plot([0, 5], [coeff[1], 5*coeff[0]+coeff[1] ])
```

## Etape n°9 : FAQ, c'est presque terminé !

**Ce que vous allez apprendre :**

- Des solutions et des liens pour résoudre certains problèmes
- Des récapitulatifs

**Mots-clés**

- input
- round

- **Le code tapé par un élève me parait cohérent. Pourtant, il ne s'exécute pas. Pourquoi ?**

  Cela signifie que Python a détecté une erreur dans le code et il est incapable d'interpréter le programme. Vous trouverez un message d'erreur dans la console qui peut vous mettre sur la voie.

  Voici deux erreurs fréquentes :

  - un problème d'indentation avec, par exemple, un espace en début de ligne
  - l'oubli du symbole `:` pour un `for` ou un `if… else`

- **C'est long de mettre en commentaire plusieurs lignes avec #. On peut être plus rapide ?**

  Il faut encadrer le texte avec trois guillemets de chaque côté : """ *commentaire long* """

- **Comment afficher à l'écran des indices ou des exposants pour les formules chimiques ou les unités ?**

  Vous pouvez copier coller ceux présents dans le tableau *Unicode characters* de la page : https://frama.link/symbexp

  Par exemple print("[Fe($H_2O$)$_6$]$^{2+}$") fonctionne. On peut aussi remplacer chaque indice ou exposant par son code **unicode**. Par exemple \u2082 signifie $_2$. Ça donnerait pour l'ion précédent :

  print("[Fe(H\u2082O)\u2086)]\u00B2\u207A")

  Il en est de même pour les caractères grecs : https://frama.link/symbgrec

- **Comment afficher une valeur arrondie ?**

  On utilise la fonction round. Par exemple round(3.14159) vaut 3 et round (3.14159,3) vaut 3.142

- **Comment demander une valeur à un utilisateur ?**

  Nous avons toujours déclaré les grandeurs physiques ou chimiques au sein du code mais on peut aussi demander à l'utilisateur de son programme de taper ces valeurs. Pour cela, il faut utiliser la fonction `input`. Par exemple :

  ```
  nom = input (" Quel est ton nom ? ") #input renvoie une chaîne de caractères
  ```

  Mais si vous souhaitez que l'utilisateur tape un nombre, il va falloir convertir la chaine de caractère renvoyée par `input` en nombre. Par exemple, pour un nombre à virgule :

  ```
  v = float (input (" Combien vaut la vitesse ? ")) # int(input…) pour un nombre entier
  ```

- **Lors des stages de nombreuses questions sont revenues. Les voici avec les réponses :**
  ***Quelle différence existe entre = et == ?***

  = sert à affecter la valeur à droite du signe = à la variable dont le nom est à gauche.
  == est un opérateur qui permet de vérifier si deux variables ont la même valeur ou pas.

```
x = 5
y = 6
if x == y :
   print("Les valeurs sont identiques")
else :
   print("Les valeurs sont différentes")
```

  ***Pourquoi Python fait des erreurs de calculs ?***
  **Il n'y a aucune erreur de calcul** mais certains résultats peuvent sembler pour le moins surprenants.
  Avant de voir l'explication, essayons de deviner ce qui va être affiché avec le code suivant :

```
if 3/9 == 0.003/0.009 :
   print("Ce sont les mêmes valeurs.")
else :
   print("Ce sont des valeurs différentes.")
```

  La réponse est `Ce sont des valeurs différentes` alors que mathématiquement ce sont bien deux valeurs identiques. Cela peut poser problème en particulier dans l'activité de Première "Déterminer la composition de l'état final d'un système siège d'une transformation chimique totale à l'aide d'un langage de programmation"

  Pour comprendre, affichons les deux valeurs suivantes :

```
print(3/9)
print(0.003/0.009)
```

  A l'écran :

```
0.3333333333333333
0.33333333333333337
```

  Effectivement, ce sont bien deux valeurs différentes. Mais pourquoi ce 7 à la fin ?

  Tous les nombres décimaux sont stockés en binaire. Par exemple, puisque : $13 = 1x2^3+1x2^2+0x2^1+1x2^0$, 13 s'écrit en binaire : 1101
  Pour les nombres flottants, c'est le même principe mais avec des puissances négatives de 2, autrement dit $(\frac{1}{2})^n$

  Par exemple $0.625=1x(\frac{1}{2})^1+0x(\frac{1}{2})^2+1x(\frac{1}{2})^3$ donc 0.625 s'écrit 0.101 en binaire. Malheureusement, un nombre décimal (c'est à dire avec un nombre de chiffre fini après la virgule) peut avoir un développement binaire infini. Par exemple 0.1 en décimal devient en binaire une valeur avec un nombre infini de chiffres après la virgule :

$0.1=0x(\frac{1}{2})^1+0x(\frac{1}{2})^2+0x(\frac{1}{2})^3+1x(\frac{1}{2})^4+1x(\frac{1}{2})^5+0x(\frac{1}{2})^6+0x(\frac{1}{2})^7+1x(\frac{1}{2})^8+1x(\frac{1}{2})^9+0x(\frac{1}{2})^{10}+0x(\frac{1}{2})^{11}+1x(\frac{1}{2})^{12}+....$
0.1 s'écrit donc en binaire 0.000110011001...

Par ailleurs un nombre à virgule dans Python, un flottant, se code sur 53 bits. La valeur 0.1 devient donc une valeur arrondie en binaire. Si on convertit à nouveau en décimal, on obtient :
0.1000000000000000055511151231257827021181583404541015625

En conséquence, si on tape

```
print(0.1)
print(0.1+0.1+0.1)
```

A l'écran :

```
0.1
0.30000000000000004
```

En effet, Python 3 affiche pour 0.1, une valeur à nouveau arrondie lors de la conversion binaire -> décimal qui permet de retrouver 0.1 mais l'opération 0.1+0.1+0.1 amplifie l'écart et ce n'est pas 0.3 qui apparait.

**Comment résoudre ce problème ?**
La fonction round affiche une valeur arrondie lorsque les calculs sont terminés : round(valeur, nombre de chiffre après la virgule). Dans le code ci-dessous, print affiche la valeur en arrondissant au quatrième chiffre après la virgule :

```
somme = 0.1+0.1+0.1
print(round(somme,4))
```

A l'écran :

```
0.3
```

En effet, les 0 après le 3 ne sont pas affichés.
Pour information, si l'on fait :

```
somme = round(0.1)+round(0.1)+round(0.1)
print(somme)
```

A l'écran :

```
0.30000000000000004
```

On a retrouvé le même problème car l'arrondi n'a pas été fait sur le résultat final.
Mais attention, pour les mêmes raisons, round renvoie des arrondis qui peuvent sembler étrange :

```
print (round(2.675,2))
print (round(2.685,2))
```

A l'écran :

```
2.67
2.69
```

Donc la fonction round() n'est pas judicieuse pour vérifier des égalités entre deux valeurs.

Si le programme que l'on tape, exige de ne pas rencontrer ces problèmes, Python propose plusieurs solutions mais aucune n'est transparente pour l'élève. On peut par exemple faire appel au module décimal de la bibliothèque Python :

```
import decimal
valeur = decimal.Decimal('0.3')
somme = valeur+valeur+valeur
print(somme)
```

A l'écran :

```
0.9
```

La documentation de Python précise :
Le module decimal « est basé sur un modèle en virgule flottante conçu pour les humains, qui suit ce principe directeur : l'ordinateur doit fournir un modèle de calcul qui fonctionne de la même manière que le calcul qu'on apprend à l'école »

***Comment choisir le nombre de chiffres après la virgule quand on affiche une valeur ?***

```
valeurA = 0.532
valeurB = 0.502
print (round(valeurA,2))
print (round(valeurB,2))
```

A l'écran :

```
0.53
0.5
```

En effet, pour 0.5, le dernier 0 est considéré comme "inutile" donc il n'est pas affiché.
Pour obtenir 0.50, il faut utiliser un formatage spécial dans le print :

```
valeur = 0.502
print ("La valeur arrondie au deuxième chiffre après la virgule est
%.2f"%(valeur))
```

A l'écran :

```
La valeur arrondie au deuxième chiffre après la virgule est 0.50
```

La variable valeur a été affichée en tant que flottant(f) avec 2 chiffres après la virgule(2f). Remarquez la présence des deux symboles "pourcentage". Pour information, ce formatage est apparu dans un sujet 0 des E3C pour les Premières.

***Comment afficher une valeur en notation scientifique ?***

Il faut formater l'affichage ainsi :

```
valeur = 138750
print ("La valeur est %E"%(valeur))
```

A l'écran :

```
La valeur est 1.387500E+05
```

On peut aussi imposer le nombre de chiffre après la virgule :

```
valeur = 138750
print ("La valeur est %.2E"%(valeur))
```

A l'écran :

```
La valeur est 1.39E+05
```

***Peut-on écrire x = x+0.3 ?***

En mathématique, c'est une équation sans solution. Mais en langage informatique le signe = permet d'affecter la valeur de droite à la variable qui est à gauche. Donc dans un premier temps, x+0.3 est évalué puis x reçoit cette nouvelle valeur. Si x valait 2.6, x vaut à présent 2.9 .
On va, par exemple, utiliser cette affectation quand on veut incrémenter une variable dans une boucle avec un pas en particulier.

***Pourquoi utiliser quiver et pas arrow pour dessiner une flèche ?***

On peut très bien utiliser arrow de matplotlib.pyplot. C'est une fonction simple à utiliser et son nom est évocateur pour les élèves. Mais elle présente un inconvénient : dans le cas d'un repère qui n'est pas orthonormé, la flèche a une apparence déformée. Pour qu'elle ait l'apparence attendue par les élèves, il faut calculer une mise à l'échelle qui n'est pas au cœur du programme de Physique Chimie. quiver est une fonction qui permet beaucoup plus de possibilités que arrow. La plupart sont inutiles pour les activités de Seconde et de Première. Mais avec les trois derniers paramètres ci-dessous, la flèche n'est jamais déformée :

```
plt.quiver(x0,y0,deltax,deltay,  angles='xy', scale=1, scale_units='xy')
```

***Comment faire une flèche colorée avec quiver ?***

Il faut ajouter le paramètre color et utiliser les lettres désignant les couleurs comme dans plt.plot :

```
plt.quiver(x0,y0,deltax,deltay,  angles='xy', scale=1, scale_units='xy', color='r')
```

***Quels sont les mots réservés par Python qu'il ne faut pas employer pour le nom d'une variable ?***

Pour nommer une variable, il ne faut pas utiliser les mots qui sont déjà utilisés par Python. Vous pouvez afficher la liste "interdite" grâce au code suivant :

```
import keyword
import builtins
print(dir(builtins),keyword.kwlist)
```

***Comment supprimer ou insérer un élément dans une liste ?***

Il y a eu plusieurs questions portant sur la manipulation des listes.

```
uneliste.pop() # supprime le dernier élément de uneliste
uneliste.insert(i,element)  # insère element dans uneliste à l'indice i
```

Normalement, ce n'est pas nécessaire pour les activités de Seconde et Première. Si vous le souhaitez, vous trouverez toutes les possibilités sur cette documentation de Python.

***Propositions de collègues***

Par ailleurs, des collègues ont proposé des techniques qui n'apparaissent pas dans le cours. Merci à eux, les voici :

- La fonction plt.grid(color='r', linestyle='-', linewidth=2) permet l'affichage d'une grille avec matplotlib.pyplot
- plt.axis("equal") permet d'avoir un repère orthonormé mais il y a deux inconvénients. Bien sûr les courbes peuvent avoir un aspect "écrasé". Par ailleurs, le calcul automatique de la longueur des axes ne prend pas en compte les dimensions des flèches tracées avec quiver ou arrow pour qu'elles apparaissent dans l'image.

Si vous voulez partager des informations, des astuces ou bien des remarques, n'hésitez pas à les envoyer.

- **Quelles notions doit-on connaître pour chaque activité de programmation du programme ?**
  Si vous choisissez d'utiliser le langage Python, vous trouverez la réponse à la page suivante. Vous trouverez ensuite un récapitulatif des fonctions en Python.

**Les couleurs contrastées ne correspondent qu'aux notions utiles. On peut souvent en utiliser d'autres si on le souhaite.**

| | | print | variable | calcul | liste | if | for | def | Matplotlib |
|---|---|---|---|---|---|---|---|---|---|
| SECONDE | Représenter les positions successives d'un système modélisé par un point lors d'une évolution unidimensionnelle ou bidimensionnelle à l'aide d'un langage de programmation. | | | | | | | | |
| SECONDE | Représenter des vecteurs vitesse d'un système modélisé par un point lors d'un mouvement à l'aide d'un langage de programmation | | | | | | | | |
| SECONDE | Représenter un nuage de points associé à la caractéristique d'un dipôle et modéliser la caractéristique de ce dipôle à l'aide d'un langage de programmation. | | | | | | | | + np.polyfit de numpy |
| PREMIERE | Déterminer la composition de l'état final d'un système siège d'une transformation chimique totale à l'aide d'un langage de programmation. | | | | | | | | |
| PREMIERE | Utiliser un langage de programmation pour étudier la relation approchée entre la variation du vecteur vitesse d'un système modélisé par un point matériel entre deux instants voisins et la somme des forces appliquées sur celui-ci. | | | | | | | | |
| PREMIERE | Utiliser un langage de programmation pour effectuer le bilan énergétique d'un système en mouvement. | | | | | | | | |
| PREMIERE | Représenter un signal périodique et illustrer l'influence de ses caractéristiques (période, amplitude) sur sa représentation. Simuler à l'aide d'un langage de programmation, la propagation d'une onde périodique. | | | | | | | pas indispen-sable * | |

*** La création d'une fonction n'est pas indispensable mais cela rend le code plus propre.**

## Déclarations de variables :

```
x = 50  #entier
description =  "Le mélange est homogène"
#chaine de caractères
vitesse  =  45.58  # nombre à virgule,
appelé « flottant »
```

## Conditions :

```
If x >= 9 :
     print(" x est > ou = à 9")
else
     print(" x vaut 9")
```

## Comparaisons possibles :

```
x == 9        x !=  9     x  > 9
x <  9        x >=  9     x  <= 9
```

## Boucles (répétitions) :

```
for i in range (6) :

     print(i)
#affiche 0 1 2 3 4 5

for i in range (2, 6) :
     print(i)
#affiche 2 3 4 5

for i in range (1, 8, 3) :
    print(a)
#affiche 0 3 6 car pas de 3
```

## Fonctions :

```
def poids(m,g) :
    return m * g
p = poids(10,9.81)
```

## Bibliothèques :

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

plt.quiver()  plt.plot()   plt.axis()
plt.xlabel()  plt.ylabel() plt.cla()
plt.legend()  plt.pause()  plt.show()
plt.savefig() plt.grid()
np.polyfit()
```

## Opérations :

```
x =  48 + 5      # 53
x =   3.4 * 8    # 27.2
x =  45  /  6     # 7.5
x =  9 ** 2     # 81
x =  " Vive " + "  la physique " + "  et
la chimie "
#Vive la physique et la chimie
```

## Interactions

```
x = 5.8
print(x)
#5.8
print("v =",x,"m/s")
#v = 5.8 m/s -> remarquez les espaces
nom = input (" Quel est ton nom ? ")
n = int (input (" un entier ? "))
x = float (input (" un flottant ? "))
#int() et float() convertissent la
chaine de caractères renvoyée par input
en entier et en flottant
```

## Listes :

```
abscisse =  [ 4.3 , 5 , 8 , 9 , 7.5 ,
6 ]
# abscisse[0] vaut 4.3
# abscisse[5] vaut 6

somme_tab= [1,2,3]+[4,5]
# [1,2,3,4,5]

for i in range(6):
      print(-0.5*abscisse[i])
# -2.15 -2.5 -4.0 -4.5 -3.75 -3.0
vitesse=[0]*5 #[0,0,0,0,0]
delta_t=1
for i in range(5): # 5 pour i+1
      vitesse[i]=(abscisse[i+1]-
     abscisse[i]) / delta_t
print(vitesse)
# [0.7, 3.0, 1.0, -1.5, -1.5]
#---------------------------
for i in range(1,len(somme_tab)-1):
#len renvoie le nombre de valeurs dans
une liste
      print(somme_tab[i])
#  2 3 4
#---------------------------
for x in abscisse:
      print(x)
# 4.3
# 5
```

# Table des matières