L'objectif de ce complément est de présenter les principales instructions du langage de programmation Python rencontrées au lycée et pouvant être utilisées dans le cadre du projet numérique.

1. LES OBJETS DE BASES

Le langage de programmation Python utilise des objets, regroupés par type, qui ont chacun un comportement et un mode d'interaction propres.

1.1. Types de variables et conversions d'un type à l'autre

```
# Affectation : signe =
x = ... # signifie « x prend la valeur ... »

Type integer <int> # nombre entier relatif
int()   # convertit si possible un réel ou un texte en entier

Type float <float> # nombre réel.
float() # convertit si possible un entier ou un texte en réel

Type string <str>  # chaîne de caractères (texte)
                   # suite de caractères d'imprimerie
                   # délimitée par des guillemets
str()   # convertit un nombre en une chaîne de caractères

Type booléen <bool> # logique : ne prend que deux valeurs
                    # True et False (Vrai et Faux)
```

```
x = 2  # Affectation simple

a,b,c = 2.5,2e3,-53 # Affectation multiple

# Affectation combinée à une opération
x += 1 # Affecte x+1 à la variable x
x -= 1 # Affecte x-1 à la variable x
```

```
x = 2             # entier
y = '13'          # chaîne de caractères
z = 9.23          # réel
t = float('2e-3') # renvoie 0.002
z1 = str(x)+y     # renvoie '213' (chaîne)
z2 = x+int(y)     # renvoie 15 (entier)
```

1.2. Opérations numériques

```
+, -, *, / # opérations mathématiques usuelles
**         # puissance
aeN        # a×10^N
//         # quotient de la division euclidienne
%          # reste de la division euclidienne
round(x,n) # arrondi le nombre x à n décimales
           # sans garder tous les 0 décimaux
```

```
m = 23/4  # renvoie 5.75
p = 3**2  # renvoie 9
q = 23//4 # renvoie 5 car 23=4*'5'+3
r = 23%4  # renvoie 3 car 23=4*5+'3'

round(0.489,2) # renvoie 0.49
round(0.499,2) # renvoie 0.5
round(1.000,2) # renvoie 1.0
```

1.3. Entrées et Sorties

• Entrées

```
input('texte')  # affiche texte et prend pour valeur la chaîne
                # de caractères saisie au clavier
int(input('texte'))   # convertit si possible la saisie en entier
float(input('texte')) # convertit si possible la saisie en réel
eval(input('texte'))  # évalue la saisie comme une expression
```

```
# Affiche le texte, convertit en entier la
# valeur saisie et l'affecte à la variable N
N = int(input('Nombre de points ='))
```
Nombre de points = 3

```
# Affiche texte, évalue l'expression saisie
# et affecte à la variable C le résultat
C = eval(input('Calcul ='))
```
Calcul = 2*N

• Sorties

```
print(objet1,objet2,...) # affiche les objets séparés par une tabulation
```

```
print('Le résultat du calcul vaut',C)
```
Le résultat du calcul vaut 6

1.4. Affichage des nombres

```
'{:format}'.format(nombre) # chaîne de caractères (<str>) contenant
                           # le nombre affiché selon le format spécifié
                           # d entier, f réel, e ou E écriture scientifique
                           # .n pour n décimales, + affiche le signe

'%format'% # utilise les mêmes signes que format
           # ne fonctionne pas pour les pourcentages
```

```
'{:04d}'.format(12)      # renvoie 0012
'{:+.3f}'.format(0.01)   # renvoie +0.010
'{:.2e}'.format(0.012)   # renvoie 1.20e-2
'{:+.1%}'.format(-0.12)  # renvoie -12.0%
```

```
print('L\'énergie Em vaut','{:.2E}'.format(4354),'J')
```
L'énergie Em vaut 4.35E+03 J

```
print('L\'énergie Em vaut','%.2e' % 4354,'J')
```
L'énergie Em vaut 4.35e+03 J

1.5. Les listes

```
Type liste <list> # suite d'éléments rangés les uns après les autres
                  # entre crochets séparés par des virgules
```

• Indexation

Dans une liste, la position de chaque élément est repérée par son indice *i* sachant que **l'indice de la première position est 0.**

• Principales fonctions

```
# Si L est une liste
L[i]        # accès à l'élément en position d'indice i
len(L)      # nombre d'éléments (ou longueur) de la liste
L[début inclus:fin exclue:pas] # accès à une partie de la liste
                               # le pas vaut 1 s'il n'est pas précisé
L.append(x) # insère x en dernière position
```

```
L = [10,20,30,40,50,60]   # liste
L[0]  # renvoie 10 le premier
L[1]  # renvoie 20 le deuxième
L[-1] # renvoie 60 le dernier
len(L)# renvoie 6
L[2:5]  # renvoie [30,40,50]
        # indice 2,3 et 4
L[1:-1] # renvoie [20,30,40,50]
        # du deuxième au dernier exclu
L.append(70) # renvoie [10,20,30,40,50,60,70]
```

Remarques. Les chaînes de caractères (<str>), sont indexées comme les listes mais non modifiables. Si `S` est une chaîne de caractères, les fonctions `S[i]`, `len[S]` et `S[début inclus:fin exclue:pas]` s'appliquent à `S`.

```
S = 'Nathan Sirius' # chaîne de caractères
len(S) # renvoie 13 (espace = 1 caractère)
S[3]   # renvoie 'h'
S[0:13:2]# renvoie 'Nta iis' (par pas de 2)
```

• Liste d'entiers

La fonction `range()` génère une liste d'entiers :

```
range(n)     # n premiers entiers de 0 à n-1 inclus
range(n,m)   # entiers de n inclus à m exclu par pas de 1
range(n,m,p) # entiers de n inclus à m exclu par pas de p avec p entier
```

```
range(4)     # renvoie 0,1,2,3
range(1,4)   # renvoie 1,2,3
range(1,7,2) # renvoie 1,3,5
```

1.6. Les tableaux à une dimension (1D)

Pour ranger une collection de nombres les uns à la suite des autres, on utilise les tableaux à « une dimension », c'est-à-dire « à une seule ligne », de la bibliothèque **Numpy**.

```
Type tableau à 1D <ndarray> # suite d'éléments rangés les uns après
                            # les autres dans un tableau à une ligne
                            # indexés en commençant par 0
```

• Création d'un tableau

```
np.array(liste)    # convertit la liste en tableau
np.linspace(a,b,n) # n valeurs régulièrement espacées de a à b inclus
np.arange(Vmin,Vmax,p) # de Vmin inclus à Vmax exclu par pas de p
```

```
np.array([-2e-1,45,1])# renvoie [-0.2 45. 1.]

# 3 valeurs entre 0 et 1.2 INCLUS
np.linspace(0,1.2,3) # renvoie [0. 0.6 1.2]

# De 0 INCLUS à 0.5 EXCLU par pas de 0.2
np.arange(0,0.5,0.2) # renvoie [0. 0.2 0.4]
```

• Principales fonctions

```
#Si T est un tableau à une ligne
T[i]        # accès à l'élément en position d'indice i
len(T)      # nombre d'éléments du tableau
T[début inclus:fin exclue:pas] # accès à une partie du tableau
```

```
T = np.linspace(0,10,3) # renvoie [0. 5. 10.]
T[1]   # renvoie 5.
len(T) # renvoie 3
```

• Opérations mathématiques sur un tableau

```
# T est un tableau et c est un nombre
T+c, T-c , c*T, T/c, T**c # opération sur chaque élément du tableau
# U et T sont deux tableaux à une ligne de même nombre de termes
T+U, T-U , U*T, T/U, T**U # opération terme à terme
```

```
t = np.linspace(0,2,3)
X = 2*t
Y = 0*t
print('t =',t,'X =',X,'Y =',Y)
```

```
t = [0. 1. 2.] X = [0. 2. 4.] Y = [0. 0. 0.]
```

• Fonction NumPy

```
np.fonction() # s'applique séparément à chaque élément du tableau
```

Remarques. Ces fonctions s'appliquent aussi aux objets ressemblant à des tableaux de type «array_like» comme les listes (<list>) de nombres.

```
L = [0,1,4]           # Liste [0,1,4]
T = np.arange(-1,2,1) # Tableau [-1 0 1]
X = np.sqrt(L) # renvoie tableau [0. 1. 2.]
Y = np.abs(T)  # renvoie tableau [1. 0. 1.]
```

2. LES BLOCS D'INSTRUCTIONS

Dans la syntaxe Python, un bloc d'instructions est défini par deux points : suivis à la ligne d'une indentation fixe (1 tabulation ou 4 espaces). La fin de l'indentation indique la fin du bloc d'instructions.

2.1. Fonction : def: ... return

```
def fonction(arg1,arg2,...): # arg='argument'
    <Instructions>           # exécute les instructions
    return résultat          # renvoie le résultat

fonction(valeur1,valeur2,...)# appel de la fonction
```

```
# fonction d'argument la vitesse v en km/h
# retournant la vitesse en m/s
def v_m_s(v):
    v1 = v*10**3/3600
    return v1

v_m_s(90)
```

```
25.0
```

2.2. Test ou instruction conditionnelle : if: ... elif: ... else:

Un test renvoie une valeur booléenne : True ou False.

```
if (test 1):          # si test 1 vérifié
    <Instructions 1> # alors exécute les instructions 1
elif (test 2):        # sinon, si test 2 vérifié
    <Instructions 2> # alors exécute les instructions 2
else:                 # sinon
    < Instructions>  # exécute les dernières instructions
```

```
pH = float(input('pH ='))
```

```
pH = 3.5
```

```
if pH<7:
    print('Acide')
elif pH==7:
    print('Neutre')
else:
    print('Basique')
```

```
Acide
```

```
# Comparateurs
# pour les tests
== # égal
!= # différent
>= # sup ou égal
>  # supérieur
<= # inf égal
<  # inférieur
```

2.3. Le boucle « tant que » : while:

```
while test:          # Tant que test vérifié
    <Instructions>   # exécute les instructions
```

```
# Recherche du plus petit entier dont
# le carré est supérieur ou égal à 1000
n = 0        # Initialise le compteur
while n**2<1000:
    n += 1  # Incrémente le compteur
print(n)
```

```
32
```

2.4. La boucle « pour » : for ... in:

```
for élément in ensemble: # Pour chaque 'élément' dans 'ensemble'
    <Instructions>       # repète les instructions
```

```
liste_de_masses = [2.5,1.8]
for m in liste_de_masses:
    poids = m*9.8
    print(poids)
```

```
24.5
17.64
```

Les ensembles parcourus par in de la boucle for sont des objets dont on peut parcourir les éléments un à un, comme les **chaînes de caractères**, les **listes**, les **tableaux à une dimension** et les **listes d'entier** de la fonction range().

La boucle for et la fonction range() sont utilisées dans la création de listes.

```
# Création de liste avec L.append()
L = []      # Initialise une liste vide
for i in range (1,4):#pour i entier de 1 à 3
    L.append(i*10) # insère i*10
                   # en dernière position
    print ('L =',L)
```

```
L = [10]
L = [10, 20]
L = [10, 20, 30]
```

```
# Création de liste en compréhension
L = [i*10 for i in range(1,4)]
print(L)
```

```
[10, 20, 30]
```

3. REPRESENTATION GRAPHIQUE

Dans de très nombreuses situations en Physique-Chimie, il est utile de tracer la représentation graphique *y=f(x)* d'une grandeur *y* en fonction d'une grandeur *x*.

En langage de programmation Python, une représentation graphique utilise les fonctions et instructions du module `pyplot` de la bibliothèque `matplotlib` habituellement importé sous le préfixe `plt`.

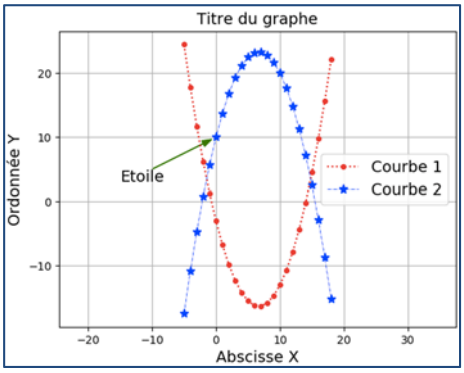
La représentation graphique de la courbe d'équation *y=f(x)*, se fait grâce à l'instruction `plt.plot(x,y,paramètres)` où `x` et `y` sont soit des listes soit des tableaux de nombres contenant les abscisses et les ordonnées des points à représenter et `paramètres` précise l'aspect des points et/ou de la courbe.

Quel que soit le type des objets `x` et `y`, la seule contrainte de l'instruction `plt.plot(x,y)` est qu'ils aient le même nombre de valeurs, c'est-à-dire qu'il y ait autant de valeurs pour les abscisses que de valeurs pour les ordonnées.

3.1. Représentation d'un nuage de points

Les principales instructions de représentation graphique figurent dans le programme suivant :

```
import matplotlib.pyplot as plt #Importe le module pyplot et le renomme plt
X=list(range(-5,19,1))  # Domaine des abscisses liste des entiers x de -5 à 18
Y1=[0.3*x**2-4*x-3 for x in X]  # Domaine des ordonnées de la courbe 1
                                # liste des y=0.3x2-4x-3
Y2=[-0.3*x**2+4*x+10 for x in X]# Domaine des ordonnées de la courbe 2
                                # liste des y=-0.3x2+4x+10
plt.figure('Titre de la figure')# Initialise et nomme la figure
                                # Option : figsize=(largeur,hauteur)
plt.title('Titre du graphe')    # Nomme le graphe
plt.plot(X,Y1,'or:',ms=4,label='Courbe 1')# Nuage de points de coordonnées
                            # dans X et dans Y1, 'o' points 'r' rouges
                            # de taille (ms=markersize) 4, ':'reliés par des
                            # petits points, label=nom de la courbe
plt.plot(X,Y2,'*b-.',lw=0.5, ms=8,label='Courbe 2')# Nuage de points de
                            # coordonnées dans X et Y2, '*' étoiles 'b' bleues
                            # '-.' reliées par des pointillés d'épaisseur
                            # (lw=linewidth) 0.5, label=nom de la courbe
plt.xlabel('Abscisse X')    # Etiquette de l'axe des abscisses
plt.ylabel('Ordonnée Y')    # Etiquette de l'axe des ordonnées
plt.text(-15,3,'Etoile', fontsize =14) # Affiche le texte 'Etoile' en commençant
                            #  au point de coordonnées -15,3, taille (fontsize)
plt.arrow(-10,5,8,4,color='g',head_width=1) # Trace une flèche verte (g) depuis
                            # le point de coordonnées (x=-10,y=5) jusqu'au point
                            # de coordonnées (x+dx=-10+8,y+dy=5+4)
plt.axis('equal')           # Repère orthonormé
plt.grid()                  # Affiche une grille
plt.legend(loc=7, fontsize=14)# Affiche les noms en légende
                            # position loc 7 (droite, centrée), taille (fontsize)
plt.show()                  # Affiche la figure
```



3.2. Modélisation d'un nuage de points expérimentaux

Les coordonnées d'une série de points expérimentaux sont rangées dans deux listes ou deux tableaux à 1D nommés X pour les abscisses et Y pour les ordonnées.

Modéliser le nuage de points consiste à déterminer l'équation mathématique de la courbe qui se rapproche le plus de celle qu'ils tracent.

Exemple : modélisation par une droite d'équation $y = a\,x + b$

La fonction `np.polyfit(X,Y,1)` modélise le nuage de points d'abscisses dans X et d'ordonnées dans Y par une **droite** d'équation $y = a\,x + b$ et renvoie le tableau : `[a b]`.

```
import numpy as np          # Importe la bibliothèque numpy en np
import matplotlib.pyplot as plt # Importe le module pyplot en plt
X = np.array([0,1,2,3,4,5]) # Tableau des abscisses des points expérimentaux
Y = np.array([0.,1.15,2,3,4.1,5.3])#Tableau des ordonnées des points expérimentaux
plt.plot(X,Y,'ro',label='points expérimentaux') # Nuage des points expérimentaux
                            # d'abscisses dans X et d'ordonnées dans Y
                            # sous forme de points rouges non reliés
Modele = np.polyfit(X,Y,1)  # Calcule les paramètres de la droite modélisant le
                            # nuage de points et les range dans le tableau Modele
a,b = [coef for coef in Modele]# Affecte dans cet ordre les paramètres du modèle
                            # aux variables a et b
plt.plot(X,a*X+b,'b-',label='modélisation') #Nuage de points d'abscisses dans X et
                            # d'ordonnées dans a*X+b, en bleu et sous forme reliée
print('y=',round(a,3),'x+(',round(b,3),')')# Affiche l'équation de la droite modèle
plt.grid()                  # Affiche une grille
plt.legend()                # Affiche la légende
plt.show()                  # Affiche la figure
```