1 Installation et démarrage

Logiciel : nous utiliserons durant ces séances de programmation l'éditeur de code Pyzo, interfacé avec Miniconda qui est un interpréteur du langage Python dans sa version 3.

Ces deux logiciels sont des logiciels libres. Vous pouvez les installer sur tout ordinateur tournant sous Linux, Windows, ou OS X.

Les logiciels et un tutoriel d'installation en 3 étapes sont disponibles ici :

`http://www.pyzo.org/start.html`

Premier test : tapez dans la fenêtre de gauche le code

```
print("Bonjour à tous !")
```

et exécutez-le avec Ctrl+E ... le message de bienvenue s'affiche dans la fenêtre supérieure droite.

Sauvegarde de vos programmes : créez puis utilisez un répertoire `Programmation` et enregistrez-y les programmes avec Ctrl+S, en leur donnant des noms explicites (par exemple de la forme TP3exo2.py).

Vous pouvez retrouver les corrections rédigées en séance par les enseignants dans :

`EchangeEnsEtu/Informatique/Programmation/Groupes/A` (ou `B`, `C`, `D`, `E` selon votre groupe).

Ressources : pour compléter (et approfondir largement) cet aide-mémoire, vous pouvez consulter en ligne ou télécharger au format pdf l'ouvrage « Apprendre Python 3.0 » de Gérard Swinnen sur

`https://python.developpez.com/cours/apprendre-python3/`

2 Variables et opérations élémentaires

2.1 Nombres

```
a=3                 # variable entière
b=6.0               # variable réelle
c=1.25e-3           # réel saisie en notation scientifique
x=y=5.2             # deux variables, une valeur
u,v = 3, 4.12       # deux variables, deux valeurs

s = a+b             # somme
m = a-b             # différence
p = a*b             # produit
carre = a**2        # carré
d = a/b             # division
q = a//b            # division entière
r = a%b             # reste de la division

u,v = v,u                   # échange des valeurs
```

Les règles de priorité usuelles s'appliquent dans les calculs :

```
print(a/(b+v**2)-d)
```

calcule d'abord le carré de `v`, puis la somme avec `b`, puis le quotient de `a` par ce résultat, puis soustrait `d`, et affiche le résultat.

2.2 Chaînes de caractères

Définir une chaîne de caractères :

```
chaine  = "ici du texte" # avec guillemets
chaine2 = 'ici aussi'    # ou bien avec apostrophes
chaine3 = 'en\ndeux lignes' # avec un saut de ligne
print(chaine3)   # affichage
```

```
chaine4 = "pour écrire une longue chaîne sur deux \
lignes utiliser le caractère anti-slash"

chaine5 = "mettre une apostrophe ' dans une chaîne"
chaine6 = 'mettre des guillemets " dans une chaîne'
```

Accès à une partie de chaîne, caractère par caractère.
Les caractères sont numérotés à partir de 0 :

```
print( chaine[2] )   # affiche le 3ème caractère
print( chaine[1:6] ) # affiche du 2ème au 5ème
print( chaine[3:] )  # affiche à partir du 4ème
print( chaine[:4] )  # affiche les 4 premiers
```

Convertir des données d'un type à l'autre :

```
a='124'    # chaîne de caractère
print(type(a))  # affiche int (entier),
                # float (réel), string (chaîne), ...
print(a+1) # erreur : a et 1 sont de types différents
a=int(a)   # conversion en entier
print(a+1) # l'addition devient possible
```

3 Affichage

La fonction `print` permet des affichages simples :

```
print(a)        # affiche la valeur de a
print(a,b)      # affiche deux valeurs
print("valeurs ", a, " et ", b) # nombres et chaînes
```

ou des affichages formatés, mais la syntaxe se complique !

Pour cela on indique une chaîne (entre guillemets ou apostrophes), dans laquelle on place des `{}` pour réserver un emplacement pour un nombre (ou autre) et on y indique son format d'affichage souhaité, et on indique à la fin dans les paramètres de l'instruction `.format` avec quoi remplir ces emplacements.

```
a=12345.67891011
b=9.987e-5

# affichage avec 2 décimales :
print("a = {:.2f} ".format(a))

# affichage avec 3 décimales, 15 caractères en tout :
print("a = {:15.3f} ".format(a))

# affichage en notation scientifique :
print("a = {:e}".format(a))

# notation scientifique avec 2 décimales :
print("b = {:.2e} ".format(b))

# mixer les notations scientifiques et décimales :
print("a = {:.2e} et b = {:3f} ".format(a,b))

# ...et aussi placer symboles et textes :
print('15 espaces ici {}, \
et 20 signes "-" là : {} '.format(" "*15, "-"*20))
```

4 Saisie au clavier :

`input("message")` affiche la chaîne de caractères passée en paramètre, et renvoie la chaîne tapée au clavier par l'utilisateur.

Pour la saisie d'un nombre, on convertit ensuite explicitement la chaîne en entier ou en réel :

```
a=input("donnez une valeur")
a=int(a)        # conversion en entier
a=float(a)      # conversion en réel
```

On peut résumer ces deux étapes en une seule ligne :

```
a=int(input("donnez une valeur"))   # saisit un entier
a=float(input("donnez une valeur")) # saisit un réel
```

5 Tests

On dispose d'opérateurs de comparaison :

```
a==b     # teste l'égalité
a!=b     # ou la non-égalité
a>b
a<b
a>=b     # supérieur ou égal
a<=b
```

que l'on peut utiliser avec l'instruction conditionnelle `if` accompagnée éventuellement de `elif` et/ou `else` :

```
if a>0 :
        print("a est strictement positif")
elif a==0 :
        print("a est nul")
else :
        print("a est strictement négatif")
```

Exemple d'un test de parité : pour savoir si un nombre est pair, on regarde si le reste de la division par 2 est nul.

```
a=int(input("donnez un nombre entier"))
if a%2 == 0: # le reste de la division par 2 est nul ?
        print("il est pair")
else:
        print("il est impair")
```

6 Fonctions mathématiques prédéfinies

On place en début de programme

```
from math import *
```

pour importer les fonctions mathématiques (le * indique de les importer tout, on peut utiliser une liste explicite des fonctions que l'on utilisera).

Quelques exemples :

```
print(pi,e)          # pi et e sont prédéfinis
print(sqrt(3))       # racine carrée
print(sin(pi/6))     # sinus (d'un angle en radian)
print(log(e/2))      # logarithme népérien ln
print(fabs(2.1))     # valeur absolue
print(atan(1))       # arctangente
```

7 Boucles

7.1 `while`

Une boucle `while` répète une série d'instructions tant qu'une condition est vérifiée.

```
# répète tant que la valeur saisie est négative :
a=-1
while a<0:
        a=float(input("donner un nombre positif"))
        print("vous avez saisi le nombre", a)
print("vous avez bien saisi une valeur positive, ", a)
```

(noter l'importance de la présence ou l'absence de tabulation devant les instructions `print`)

On peut utiliser `while` et une variable pour faire varier un nombre entre deux entiers :

```
# compter jusqu'à 10 :
a=0
while a<10:
        a=a+1
        print(a)
```

7.2 `for`

Dans ce dernier cas, la boucle `for` couplée à la fonction `range` fournissent une solution moins universelle mais plus élégante que la boucle `while` :

```
for nombre in range(10): # 10 nombres entiers, de 0 et 9
        print nombre
```

Variantes :

```
for nombre in range(5,12):       # de 5 à 11
        print nombre

for nombre in range(12,1,-1):    # de 12 à 2
        print nombre

for nombre in range(2,24,3):     # par pas de 3
        print nombre
```

Afficher 200 valeurs régulièrement réparties entre pi/6 et pi/4 :

```
import math     # importe le module
        # contenant les fonctions mathématiques
for i in range(200):
        print( pi/6 + (pi/4 - pi/6)*i/199 )
```

8 Fonctions

8.1 Fonction sans paramètre ni valeur renvoyée

```
def mapremierefonction():  # pour la définir
        print("affiche un texte")

mapremierefonction()       # pour l'utiliser
```

8.2 Avec paramètre mais sans valeur renvoyée

```
def affichecarre(nombre):
        print("le carré de ", nombre, " est ", nombre**2)

affichecarre(5)
```

8.3 Avec paramètre et renvoi de valeur

Il est souvent préférable de dissocier calcul et affichage ainsi :

```
def carre(nombre):              # calcul et retour de valeur
        return nombre**2        # mais sans affichage

print("le carré de 5 est ", carre(5))
                # affiche le résultat renvoyé
```

8.4 Avec plusieurs paramètres

```
# fonction à qui on fournit deux réels x et y,
#    et qui renvoie le module du complexe x + iy

def module(x, y):
        return sqrt(x**2 + y**2)
```

8.5 Avec des variables locales

On peut utiliser dans la fonction une variable, qui n'existe pas en dehors de la fonction, et qui sert à faire des calcul ou stocker des résultats intermédiaires :

```
def sinc(x):    # fonction sinus cardinal
        if x != 0:
                y=sin(x)/x
        else:
                y=1
        return y

y=3
print(sinc(1), y) # expliquer l'affichage obtenu !
```

9 Listes

Définir une liste :

```
uneliste = [ 1, 'a', 3.2, 5] # on peut mélanger les types
uneliste = []           # liste vide
```

Pour ajouter une valeur en fin de liste :

```
uneliste.append(12)
```

Afficher et manipuler une liste :

```
print( uneliste )       # affiche la liste
print( uneliste[1:3] )   # affiche du 2ème au 4ème élémer
uneliste.sort() # trie une liste de nombres
uneliste.reverse()      # renverse l'ordre d'une liste
```

Détermine le rang d'une valeur présente dans la liste (erreur si la valeur n'est pas présente, le premier rang si la valeur apparaît plusieurs fois) :

```
uneliste.index(17)
```

Enlève un élément d'une liste (ne fait rien s'il n'est pas présent) :

```
uneliste.remove(32)
```

Déterminer le nombre d'éléments d'une liste :

```
len(uneliste)
```

10 Tracé de courbes

Pour créer une liste `X` contenant `nombre` abscisses régulièrement réparties entre deux nombres `min` et `max` : on ajoute les valeurs une par une à une liste initialement vide :

```
X = []  # liste vide
for i in range(nombre):
        X.append( min + i*(max-min)/(nombre-1) )
print(X)   # affichage (peu lisible)
```

Pour créer une liste d'abscisse et leurs images par la fonction `cos` :

```
X = []
Y = []
for i in range(nombre):
        x = min + i*(max-min)/(nombre-1)
        X.append( x )
        Y.append( cos(x) )
```

Si la liste `X` existe déjà, et que l'on veut les ordonnées correspondantes dans une liste `Y` :

```
Y = []
for x in X:
        Y.append( cos(x) )
```

Pour tracer la courbe associée :

```
import matplotlib.pyplot as pplot

pplot.grid(True)
pplot.plot(X,Y)
pplot.show()
```

11 Fichiers

11.1 lecture directe

Pour relire un fichier, on peut l'ouvrir, récupérer l'ensemble des lignes du fichier dans une liste `lignes`, et le refermer :

```
fichier = open('essai.txt', 'r')
lignes = fichier.readlines()
fichier.close()
```

Si le fichier `essai.txt` contient un nombre par ligne, on peut créer la liste des valeurs présentes dans le fichier ainsi :

```
fichier = open('essai.txt', 'r')
X=[]
for x in fichier.readlines():
        X.append( float(x) )
fichier.close()
```

Si les données sont rangées à raison de deux nombres par ligne séparés par une espace, on peut utiliser en plus `split()` pour remplacer chaque ligne par une liste composée des deux nombres de la ligne :

```
fichier = open('courbe.txt','r')

X=[]
Y=[]

for ligne in fichier.readlines():
    ligne = line.split()
    X.append(float(ligne[0]))
    Y.append(float(ligne[1]))

fichier.close()
```

11.2 fonctions `lecture`

Écrire une fonction `lecture` permet de ne pas refaire à chaque fois ce travail dans le programme principal :

```
def lecture(chaine):
    fichier = open(chaine,'r')
    X=[]
    for ligne in fichier.readlines():
        X.append( float(ligne) )
    fichier.close()
    return(X)


X = lecture('courbe.txt')
```

et en version "deux valeurs par ligne" :

```
def lecture2(chaine):
    fichier = open(chaine,'r')
    X=[]
    Y=[]
    for ligne in fichier.readlines():
        ligne = line.split()
        X.append(float(ligne[0]))
        Y.append(float(ligne[1]))
    fichier.close()
    return(X,Y)


X,Y = lecture2('courbe.txt')
```

11.3 fonctions `sélecteur de fichier`

```
from tkinter import *
from tkinter.filedialog import askopenfilename

nomdufichier = askopenfilename()
```

renvoie dans la variable `nomdufichier` le chemin d'accès désigné par l'utilisateur.

On peut spécifier un argument optionnel `initialdir` pour choisir le répertoire par défaut dans lequel s'ouvre le sélecteur de fichiers et limiter les fichiers qui apparaissent aux fichiers texte :

```
nomdufichier = askopenfilename
(initialdir = 'X:/Informatique/Programmation/Donnees/',
filetypes=[("texte",'*.txt;*.text')])
```