Université Paris-Sud **Introduction au calcul scientifique**
Licence de Mathématiques **S4 UE Math209 - Année 2016-2017**

Travaux Pratiques en Python 1

Exercice 1 : Prise en main de Python

Le logiciel Spyder est une interface graphique permettant de programmer dans le langage python. Ce logiciel se présente comme suit

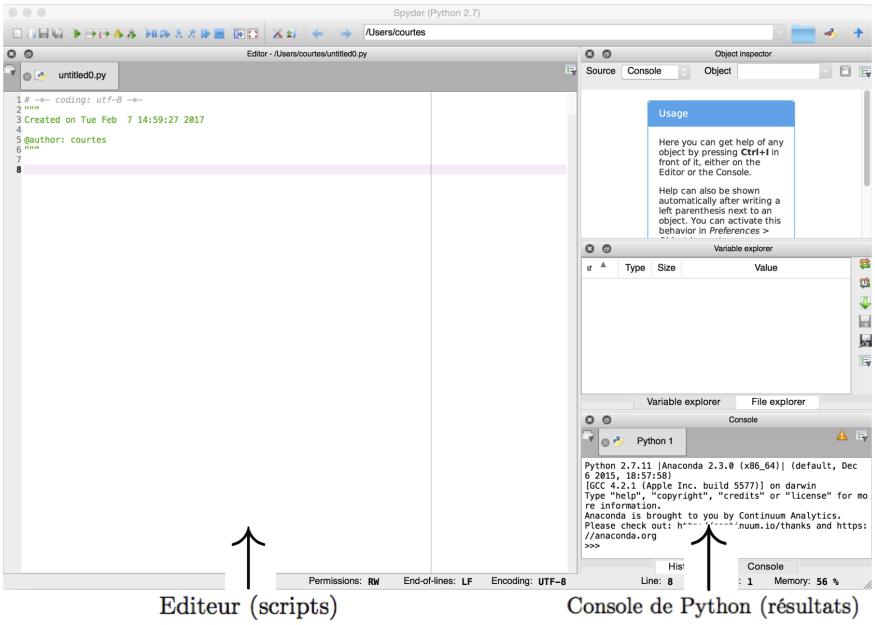

Figure 1: Le logiciel Spyder

Afin de sauvegarder les codes, on les regroupe en scripts qu'on écrit dans l'éditeur de spyder (la partie gauche de l'écran). Une fois le script enregistré et exécuté, le résultat s'affiche dans la console de python (la partie droite de l'écran).
Attention, un script python commence toujours par la ligne

```
# -*- coding: utf-8 -*-
```

(il ne faut pas la supprimer !).

1. *Les opérations de base.* Dans l'éditeur de spyder, tapez les commandes suivantes

```
a=1
b=2
a+=1
print(a)
b-=2
print(b)
a*=4
print(a)
print(a/b)
```

Notez que le résultat ne s'affiche dans la console de python qu'avec le mot clé `print`. Il est intéressant que les résultats ne s'affichent pas automatiquement (notamment quand un calcul nécessite

beaucoup d'étapes intermédiaires). Les commandes `+=, -=, *=` sont des raccourcis.
Le cas de la division est un peu particulier : si `a` et `b` sont des entiers, spyder va faire la divison entière (par exemple `1/2=0`). Pour que la division soit réelle, il faut que `a` et `b` soient des réels, pour cela, on rajoute un `.` (par exemple `1./2.=0.5`).

2. *Les commentaires.* Si une ligne n'a pas vocation à être exécutée par spyder, elle doit être précédée du symbole dièse `#`. Si plusieurs lignes ne doivent pas être exécutées, au lieu de les faire toutes précéder du symbole dièse, on peut les encapsuler entre trois jeux de guillemets
```
"""
mon commentaire
"""
```
Notez que sur la figure 1, la date et l'auteur sont des commentaires placés dans ces trois jeux de guillemets.

3. Affichez dans la console python : `Ceci est mon premier programme`, grâce à la commande `print`. Attention, lorsque vous voulez afficher toute une chaîne de caractère, il faut l'encadrer par des guillemets
```
print("ma chaîne de caractères")
```

4. Commentez cette ligne de code dans l'éditeur pour qu'elle ne s'exécute plus par la suite.

Exercice 2 : Les fonctions
Il est possible de créer des fonctions grâce aux mots clefs `def` et `return`, de la manière suivante

```
def nom_de_ma_fonction(paramètres):
            ce que fait ma fonction
            return resultat
```

Attention, les indentations en python sont très importantes. Toutes les lignes du script doivent commencer le plus à gauche possible, sauf les lignes à l'intérieur de la fonction. L'indentation se fait automatiquement par spyder, ne le changez pas.
Les deux points `:` à la première ligne sont très importants aussi, ne les oubliez pas !

1. Codez la fonction carré : $x \mapsto x^2$
```
def carre(x):
      return x**2
```

   Notez que la puissance s'écrit en python `**` et non `^`.
   Pour l'instant, on a juste défini la fonction, donc rien ne s'affiche lorsque vous exécutez votre code. Pour faire appel à cette fonction, il faut écrire
```
print(carre(2))
```

2. Codez les fonctions $x \mapsto \frac{x}{2}$, $x \mapsto x^3$, $x \mapsto 2x + 9$.

3. Testez-les sur quelques valeurs.

Exercice 3 : Les boucles `if, while, for`
Python permet de faire des boucles conditionnelles avec les mots clés `if, elif` et `else`

```
if condition_1:
            resultat_1
elif condition_2:
            resultat_2
elif condition_3:
            resultat_3
...
else:
            resultat_n
```

Le mot clé `elif` n'est pas obligatoire et peut être utilisé plusieurs fois. Par contre, au sein d'une même boucle conditionnelle `if` et `else` ne doivent être utilisés qu'une seule fois.
Notez l'importance de l'indentation et des deux points `:` (exactement comme pour les fonctions de l'exercice 2).
Attention, il n'y a pas de commande du type «fin», seule l'indentation permet de savoir si la ligne de commande fait partie de la boucle ou non.

égalité	`==`
différence	`!=`
supérieur	`>` ou `>=`
inférieur	`<` ou `<=`

Table 1: Les différents symboles conditionnels

1. Codez la boucle suivante

```
a=235
if (a%2==0):
            print("le nombre est pair")
else:
            print("le nombre est impair")
```

Notez que le reste de `a` modulo `b` s'écrit
`a%b`

2. Testez si 6728 est divisible par 3, par 8 et par 11.

Python permet aussi de faire des boucles `for` ou `while`. Dans une boucle `for`, l'utilisateur sait *a priori* le nombre de fois où l'instruction est réalisée, alors que dans une boucle `while`, l'instruction est réalisée *tant qu'une condition est vraie*. On ne sait pas *a priori* combien de fois cette condition sera vraie. La syntaxe est la suivante

```
for i in I:
        resultat
```
ou
```
while condition:
        resultat
```

Notez l'importance encore une fois de l'indentation et des `:`
Dans une boucle `for`, l'intervalle `I` peut être

- une liste écrite à la main : `[1, 5, 8, 0, 2]`,
- des entiers compris entre n_0 et n_1 par pas de N : `range(n_0, n_1+1, N)`,
- ...

Attention de ne pas oublier le `+1` dans le `range` pour inclure quand même n_1 puisque `range` ne prend jamais en compte le dernier point.

3. Affichez à l'aide d'une boucle `for` tous les entiers pairs de 0 à 100.
4. À l'aide d'une boucle `while`, affichez les élements de la suite $u_{n+1} = 3u_n$ avec $u_0 = 1$, plus petits que 40.

On peut bien sûr imbriquer les boucles `for, while` et `if` entre elles, dans ce cas, il faut faire attention de toujours indenter un cran de plus.

Exercice 4 : Les modules
Le langage python dispose d'un certain nombre de bibliothèques (appelées des modules) pour pouvoir étendre les fonctionnalités. Pour pouvoir utiliser une fonction d'un de ces modules, il faut au préalable l'avoir importé grâce à la commande :

```
import nom_du_module as alias
```

Cette ligne de commande est à mettre en haut du programme, juste après le nom et la date. Le préambule de chaque programme se présente donc comme suit

```
# -*- coding:  utf-8 -*-
"""
created on date
author:  nom
"""
import nom_du_module_1 as alias_1
import nom_du_module_2 as alias_2
...
```

`Alias` est un nom plus court que l'on utilisera toujours pour faire appel à ce module. Pour utiliser une fonction d'un module importé, on écrit

```
alias.nom_de_la_fonction
```

Par exemple, pour utiliser la fonction racine carré (qui se nomme `sqrt`) et qui se trouve dans le module `numpy` (que l'on renommera avec l'alias `np` pour simplifier), il faut écrire

```
import numpy as np
a=np.sqrt(2)
print(a)
```

Testez ce code sur $\sqrt{3}$, $\sqrt{5}$...
Parmi tous les modules existants, nous en importerons deux essentiellement :

```
import numpy as np # pour utiliser des tableaux
import matplotlib.pyplot as plt # pour tracer des figures
```

Exercice 5 : Les tableaux
Dans tout ce qui suit, nous supposerons que le module `numpy` a été importé sous l'alias `np`.

```
import numpy as np
```

1. *Initialisation.* Pour initialiser un tableau (ou vecteur), il existe la commande `np.array`. Il faut ensuite mettre les valeurs du tableau selon les lignes en séparant les lignes par des [ ] et en encapsulant le tout dans des [ ] extérieurs. Par exemple, pour écrire le vecteur $\begin{pmatrix}1.4 & 7 & 0.9\end{pmatrix}$, il faut écrire
   ```
   np.array([1.4, 7, 0.9])
   ```
   Pour écrire le vecteur $\begin{pmatrix}1.4\\7\\0.9\end{pmatrix}$, il faut écrire
   ```
   np.array([[1.4], [7], [0.9]])
   ```
   Pour écrire la matrice $\begin{pmatrix}0 & 1 & 2\\4 & 6 & 0\end{pmatrix}$, il faut écrire
   ```
   np.array([[0, 1, 2], [4, 6, 0]])
   ```

2. Définissez les vecteurs $a = \begin{pmatrix} 1.4 \\ \frac{1}{3} \\ 5 \\ \sqrt{3} \end{pmatrix}$, $b = \begin{pmatrix} 7 \\ \frac{3}{2} \end{pmatrix}$ et la matrice $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$

3. *Tableaux particuliers.* Pour créer un vecteur X de taille N avec seulement des 0, il existe la commande
```
X=np.zeros(N)
```
Que font-les commandes suivantes ? (affichez les différents tableaux `X, Y, Z` et `I`)
```
X=np.zeros(3)
Y=np.zeros((2,4)) # attention aux doubles parenthèses
Z=np.ones(5)
I=np.eye(3)
```

4. *Tableaux linspace.* Le module `numpy` possède la fonction `linspace` qui permet de discrétiser un intervalle $[a, b]$ en N valeurs équidistantes :
```
X=np.linspace(a, b, N)
```
À l'aide de cette fonction, créez un vecteur de taille 27 avec des valeurs équidistantes comprises entre 10 et 99.

5. *Accès à une valeur.* Pour avoir accès à la valeur d'indice `i` du tableau `X`, on utilise des crochets :
```
X[i]
```
Affichez la valeur 14 du vecteur précédent. Puis insérez $\sqrt{7}$ à la place de `X[2]`.

6. *Opérations.* Les opérations usuelles `+, -, *` et `**` sont des opérations termes à termes. Testez les opérations suivantes
```
a=np.ones(3)
print(a+4)
b=np.array([[2, 4], [8, 9.7], [√5, 3.2]])
print(5./2.*b)
print(b**2)
```

7. *Produit scalaire.* Le produit scalaire s'écrit `np.dot(A,B)`. Attention, `*` est un produit terme à terme.
```
a=np.array([1, 2, 3])
b=np.array([2, 3, 4])
print(a*b) # produit terme à terme
print(np.dot(a,b))# produit scalaire
```

8. *Produit matrice vecteur.* Le produit matrice vecteur s'écrit aussi `np.dot(A,B)`. Testez le code suivant
```
a=np.array([[1, 2, 3], [4, 5, 6]])
b=np.array([[1], [1], [1]])
print(a*b) # problème de dimension
print(np.dot(a,b))# produit matrice vecteur
```

Exercice 6 : Les figures

Le module à importer pour pouvoir tracer des figures se nomme `matplotlib.pyplot`, que l'on utilisera toujours sous l'alias `plt`. Nous supposerons à partir de maintenant que les deux modules `numpy` et `matplotlib.pyplot` sont importés.

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

Pour tracer une fonction, il faut d'abord discrétiser l'axe des abscisses grace à la commande `np.linspace`, vue à l'exercice 5. Ensuite, il faut calculer l'ordonnée de toutes ces abscisses ponctuelles $(x_i)_{i \in I}$ et tracer

le nuage de points (x_i, y_i). Le script pour visualiser une fonction s'écrira toujours de la forme suivante :

```
plt.figure(1)            # on numérote ses figures au fur et à mesure
plt.clf()                # on part d'une figure vierge
x=np.linspace(a, b, N) # on discrétise l'abscisse [a, b] avec une précision de N
y=f(x)                   # on calcule les ordonnées de chaques points
plt.plot(x,y)            # on trace le nuage de points
plt.title("mon titre") # titre de la figure
plt.show()               # pour visualiser la figure.
```

La fonction `plt.plot` possède des arguments facultatifs (couleur du graphe, marqueurs pour les points, ligne brisée ou continue, légende des courbes ...), elle s'utilise comme suit

```
plt.plot(x, y, 'ro', label='ma courbe 1')
```

Le `r` correspond à la couleur (rouge ici) et le `o` correspond aux marqueurs des points (des cercles ici). Ce choix peut être remplacé par

couleurs	marqueurs
`r` : rouge	`o` : cercle
`g` : vert	`*` : étoile
`b` : bleu	`-` : ligne continue
`k` : noir	`--` : pointillés
...	...

Table 2: Différentes options de `plt.plot`

Pour les légendes, il faut donner un `label` à chaque courbe et avant la ligne `plt.show()`, écrire
```
plt.legend()
```

1. Définissez la fonction $x \mapsto \sin(x) + 3x$ (le sinus se trouve dans le module `numpy`) à l'aide du mot clé `def` vu à l'exercice 2.
2. Tracez cette fonction sur $[0, 2\pi]$, par une ligne continue en vert (π se trouve aussi dans le module `numpy`).

Par défaut, toutes les courbes seront affichées sur la même figure, si vous ne changez pas de numéro de figure. Il est parfois intéressant de garder la même figure, mais de séparer les courbes dans des fenêtres différentes. C'est la commande `plt.subplot` (à insérer avec `plt.plot`) qui permet cela

```
plt.subplot(i, j, k)
```

Cette commmande crée i fenêtres verticalement et j fenêtres horizontalement et place la courbe sur la fenêtre numéro k (sachant que la première fenêtre est en haut à gauche et qu'elles sont parcourues ligne par ligne)

3. À l'aide de la commande `np.subplot`, affichez la fonction $x \mapsto \cos(\frac{x}{2})$ pour plusieurs précisions N en abscisse.