Ce TP est en grande partie inspiré d'un ancien TP de sylvain Pelletier : `http://pcsidescartes.fr`

Des bibliothèques sont disponibles pour Python pour le calcul scientifique et le graphisme (dessins de fonctions, d'histogrammes, traitements des images).

La totalité de ces fonctions se trouvent dans la bibliothèque **matplotlib**.

Nous utiliserons précisément la bibliothèque **pylab** qui contient les bibliothèques :

matplotlib pour le graphique donc,

numpy pour la manipulation des tableaux **array**

PIL pour la manipulation d'images.

Pour programmer, il est préférable d'appeler chaque module séparément en leur donnant des alias, on écrira par exemple :

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

pour appeler les deux bibliothèques **matplotlib** et **numpy**.

Pour consulter l'aide sur matplotlib, on utilisera l'aide spécifique à cette bibliothèque ici : `http://matplotlib.org/contents.html`

Exercice 1 :

La fonction la plus importante est la fonction plot. La syntaxe est :

`plot(listeX, listeY) ou plot(listeX, listeY, [options])`

Les entrées sont deux listes de même taille :

— La liste des abscisses listeX : $(x_0, x_1, \cdots, x_{n-1})$

— La liste des ordonnées listeY ; $(y_0, y_1, \cdots, y_{n-1})$

Les n points : $A_0 = (x_0, y_0), A_1 = (x_1, y_1), \cdots$ sont alors reliés par des segments de droites.

Pour obtenir des informations sur les options, il faut consulter l'aide.

(1) Afficher la courbe de la fonction f définie par $f(x) = x$ sur $[0; 3]$ et $f(x) = 6 - x$ sur $[3; 6]$.

(2) Afficher cette fonction en vert avec des pointillés.

(3) En utilisant la fonction **axis**, faire en sorte d'avoir un axe avec autant abscisse qu'en ordonnée (c'est-à-dire, afficher l'axe des ordonnées jusqu'à 6) .

(4) Donner un nom à votre courbe.

(5) Afficher une grille avec la fonction **grid()**.

(6) Enregistre votre courbe dans le fichier graphiqu1.jpg avec **savefig**.

Exercice 2 :

Dessin de courbes et de fonctions

Il faut garder en tête que Python ne fonctionne qu'avec des vecteurs, il est donc impossible de gérer des choses complexes comme *Afficher la Courbe représentative de la fonction f.*

Il faut donc systématiquement passer par deux vecteurs (des listes ou des `array` unidimensionnels) :

— l'un contiendra une liste des abscisses $[x_0, \ldots, x_{n-1}]$,

— l'autre la liste des ordonnées $[f(x_1), \ldots, f(x_n)]$.

Python dessinera alors les segments de droites qui relient les n points de coordonnées $\Big((x_i, f(x_i)\Big)$.

Si il y a suffisamment de points, la courbe ressemblera à une courbe lisse et non à des segments de droites.

De même pour dessiner un cercle, il faut créer la liste des coordonnées des points du cercle. On utilise alors l'équation paramétrique du cercle.

Exécuter (et comprendre) les commandes suivantes :

```
n = 100
x = np.zeros(n)
y = np.zeros(n)
for k in range(n) :
    theta = 2*k*np.pi/n
    x[k] = np.cos(theta)
    y[k] = np.sin(theta)

plt.axis([-1.2,1.2,-1.2, 1.2])
plt.grid()
plt.plot(x,y)
plt.show()
```

Comment faire pour fermer le cercle ?

Exercice 3 :

Opérations sur les array

On rappelle que les array sont les vecteurs gérés par la bibliothèque Numpy. On utilisera donc cette bibliothèque dans la suite.

Le problème qui arrive alors naturellement est de créer rapidement et facilement la liste des abscisses $[x_0, \dots, x_{n-1}]$, et des ordonnées. $[f(x_0), \dots, f(x_{n-1})]$. Ces listes peuvent être des listes de réels ou des `array` unidimensionnels (en fait n'importe quelle structure que Python peut convertir en `array` unidimensionnel).

Une première méthode est, comme on l'a vu, de faire des boucles `for` :

— On crée un array avec les fonctions `ones` ou `zeros` :

```
x = np.ones(nbrPoints) ou x = np.ones(nbrPoints, float)
```

— On remplit les valeurs dans une boucle `for` :

```
for k in range(nbrPoints) :
    x[k] = ...
    y[k] = ...
```

— Il ne reste plus qu'à envoyer ces listes à `plot`.

Une meilleure méthode est d'utiliser :

— les fonctions qui créent des listes de points équirépartis pour la liste des abscisses,

— les opérations sur les `array` pour calculer la liste des ordonnées.

Pour les abscisses, on dispose de deux fonctions à connaître :

— linspace(xmin,xmax,nbrPoints) (« *linéairement espacée* ») qui crée un `array` de taille `nbrPoints`, dont le premier élément est `xmin` et le dernier `xmax`.
On contrôle ainsi le nombre de points de la discrétisation de l'intervalle [`xmin`, `xmax`], le pas est : $\frac{\texttt{xmax} - \texttt{xmin}}{\texttt{nbrPoints} - 1}$.

— arange(xmin,xmax,pas) (« *array range* ») : crée un `array` dont le premier élément est `xmin` et dont chaque élément est distant de `pas`, le dernier étant strictement inférieur à `xmax`.
Cette fonction est donc semblable à `range` sauf que l'on accepte ici un pas non entier, et que la sortie est un `array`.
On contrôle ainsi le pas de la discrétisation de l'intervalle [`xmin`, `xmax`[.
Attention : le dernier point n'est jamais `xmax`.

Pour les opérations sur les `array`, on rappelle que si `x` et `y` sont des `array` et `alpha` est un scalaire, on a :

— x+y est obtenu en ajoutant terme à terme les éléments de `x` et `y`,
— x+alpha est obtenu en ajoutant `alpha` à tous les termes,
— alpha∗x est obtenu en mutlipliant tous les termes par `alpha`,
— x∗y est obtenu en mutlipliant terme à terme les éléments de `x` et `y`,
— cos(x) est obtenu en appliquant la fonction cosinus à tous les éléments de `x`. C'est le comportement pour toutes les fonctions mathématiques (fonctions universelles présentes dans la bibliothèque `numpy`). **Attention** : pour appliquer une fonction qui n'est pas dans la bibliothèque, il est meilleur de revenir à la boucle `for`.
— x∗∗n est obtenu en mettant à la puissance n-ième tous les éléments de x.
— 1/x est obtenu en prenant l'inverse de tous les éléments de x.

Ces fonctionnalités permettent de calculer la liste des images très facilement à partir de la liste des indices.

Par exemple : si on doit dessiner la courbe représentative de la fonction :

$$x \longmapsto \frac{x^3+3x+2}{x^2+1} + x\sin(x) + e^x.$$

sur l'intervalle $[-10, 10]$, on écrit simplement :

```
x = np.linspace(-10, 10, 100)
y =  (x**3+3*x+2) / (x**2+1)  + x*np.sin(x) + np.exp(x)
plt.plot(x,y)
```

La même chose avec une boucle `for` est :

```
pas = (10 - (-10) ) / 99
x = np.zeros(100)
y = np.zeros(100)
for k in range(100):
    x[k] = -10 + k *pas
    y[k] = (x[k]**3 + 3*x[k] + 2) / (x[k]**2 +1) +
           x[k]*np.sin(x[k]) + np.exp(x[k])
```

De même, le dessin du cercle peut se faire avec :

```
theta = np.linspace(0,2*pi,100)
x=np.cos(theta)
y=np.sin(theta)
plt.plot(x,y)
plt.show()
```

ou même :

```
theta = np.linspace(0,2*pi,100)
plt.plot(np.cos(theta), np.sin(theta))
```

remarque

— Du fait des opérations entre `array`, le comportement de += n'est pas celui des listes. Ainsi, l'opération x+=1 ajoute 1 à tous les éléments de x et non à la fin. Pour ajouter une valeur, on peut utiliser la fonction `vecteur = append(vecteur, valeur)`, avec `vecteur` le `array`, et `valeur` la valeur (ou la liste de valeurs) à ajouter.

— On travaille généralement avec environ 100 points. C'est d'ailleurs la valeur par défaut pour la fonction `linspace`. Commencez par cette valeur et ajoutez des points si nécessaires.

— Les fonctions universelles s'écrivent comme les fonctions mathématiques. Attention simplement au logarithme qui s'écrit `log`. Le logarithme en base 10 s'écrit `log10`.
Voir le tableau des fonctions universelles.

— Les opérations entre `array` sont aussi valables pour les `array` en deux dimensions (les matrices donc).
En particulier, le produit $A * B$ n'est pas le produit matriciel, c'est le produit terme à terme. Pour obtenir le produit matriciel, il faut utiliser `dot(A,B)`. Attention aussi : $A+5$ est interprété comme A+5∗ones([n,n]) avec (n, n) la taille de A, et non comme $A + 5I_n$.

— Pour les matrices, on peut aussi utiliser la structure `matrix`, dans laquelle le produit entre `matrix` est le produit matriciel. Néanmoins, il est plutôt conseillé d'utiliser la structure `array`.

— Attention aux ensembles de définitions ! Si vous représentez la fonction $f : x \mapsto \frac{1}{x}$ sur $[-10, 10]$, selon la valeur du pas ou du nombre de points, on peut calculer $f(0)$.

— Du fait des erreurs d'arrondis, si on fait arrange(xmin,xmax,pas), il est possible que le dernier point soit strictement supérieur à `xmax` !

$x \mapsto e^x$	`exp`	exponentiel
$x \mapsto \ln(x)$	`log`	logarithme népérien
$x \mapsto \log(x)$	`log10`	logarithme base 10
$x \mapsto \sqrt{x}$	`sqrt`	racine. Pour la racine n-ième, il faut utiliser x∗∗(1/n)
$x \mapsto \cos(x)$	`cos`	cosinus (idem pour sinus et tangente)
$x \mapsto \arccos(x)$	`arccos`	arcosinus (idem pour arcsinus et arctangente)
$x \mapsto \lfloor x \rfloor$	`floor`	partie entière mathématique. `ceil` et `round` sont les parties entières par excès et approchée.
$x \mapsto \lvert x \rvert$	`abs`	valeur absolue

Dessiner les fonctions suivantes (un dessin par ligne) :

dessin 1 :	$x \mapsto \sin(x)$	$x \mapsto x$	$x \mapsto x - \frac{x^3}{6}$	$x \mapsto x - \frac{x^3}{6} + \frac{x^5}{120}$
dessin 2 :	$x \mapsto e^x$	$x \mapsto 1 + x$	$x \mapsto 1 + x + \frac{x^2}{2}$	$x \mapsto 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$
dessin 3 :	$x \mapsto \ln(1 + x)$	$x \mapsto x$	$x \mapsto x - \frac{x^2}{2}$	$x \mapsto x - \frac{x^2}{2} + \frac{x^3}{3}$

Pour chaque dessin, on choisira convenablement l'intervalle d'étude [`xmin`, `xmax`], ainsi que les axes de manière à montrer que ces polynômes approchent bien ces fonctions au voisinage de 0.

On utilisera des variables pour les valeurs xmin, xmax, ymin, ymax. Pensez aussi à afficher la grille avec : grid()

Exercice 4 :

Représentation d'une série statistique

matplotlib est aussi très utile pour représenter des séries statistiques.

Il y a :

pie , qui permet de dessiner des diagrammes circulaire ou (camembert).

bar , qui permet d'afficher des diagrammes en barres ou diagramme en bâtons

hist , qui permet d'afficher des histogrammes.

Pour l'ensemble des exercices suivants, on utilisera l'aide pour chaque fonction.

Attention, dans cet exercice je vous donne explicitement le type de graphique à utiliser. Pour un TIPE, c'est à vous de trouver le type le plus pertinent

(1) On trouve les résultats des élections législatives dans votre ville sur le site du ministère : `https://www.interieur.gouv.fr/Elections/Les-resultats/Legislatives/elecresult__legislative(path)/legislatives-2017/index.html`

Dessiner un diagramme circulaire qui représente les résultats du 2nd tour de votre ville en prenant en compte l'abstention et les votes blancs/nuls.

(2) On regarde maintenant les résultats du premier tour.

a. Représenter par un diagramme les résultats de votre ville

b. Représenter un diagramme qui distingue les résultats de votre ville avec la ville de votre voisin(e). Pour cela on pourra afficher deux diagrammes différents (un bon exemple se trouve ici : `http://matplotlib.org/examples/api/barchart_demo.html`).

Exercice 5 :

Étude d'une suite implicite avec Python

On considère un réel λ et la suite définie par :

$$\forall n \in \mathbb{N}, u_n \text{ est l'unique solution positive de } \ln(x) + x^n = \lambda.$$

Il s'agit donc d'étudier la suite implicite (u_n) en fonction du paramètre λ.

1. Écrire une fonction dessin(nMax, lam) qui :
   — dessine les fonctions $f_n : x \mapsto \ln(x) + x^n$ pour $n \in [[0, \texttt{nmax}]]$
   — dessine la droite horizontale $y = \texttt{lam}$.
   Petit détail : `lambda` est un mot clé pour Python, il est impossible de l'utiliser comme variable.
2. Visualiser graphiquement le comportement de la suite (u_n). Dans les cas suivants : $\lambda > 1$, $\lambda = 1$, $\lambda \in]0, 1[$, $\lambda < 0$ et $\lambda = 0$.
   Pour chacun des cas, on donnera la monotonie de la suite et la limite. On présentera les résultats sous forme de tableaux.
3. **Résolution d'un problème de géométrie avec Python**
   Soit $\mathcal{C}_1$ le cercle de centre O et de rayon 1 et $\mathcal{C}_2$ le cercle de centre $\Omega \begin{pmatrix} 6 \\ 0 \end{pmatrix}$ et de rayon 2.
   On veut déterminer les tangentes communes aux deux cercles et donner leur point de contact avec $\mathcal{C}_1$ et $\mathcal{C}_2$.
   On va utiliser les représentations paramétriques des cercles :

$$\mathcal{C}_1 = \Big\{(\cos\theta, \sin\theta) \Big| \theta \in [0, 2\pi]\Big\} \qquad \text{et } \mathcal{C}_2 = \Big\{(6 + 2\cos\alpha, 2\sin\alpha) \Big| \alpha \in [0, 2\pi]\Big\}$$

   (a) Dessiner les deux cercles.
   (b) On cherche un couple $(\theta, \alpha) \in [0, 2\pi]^2$ tel que les points A de coordonnées $(\cos\theta, \sin\theta)$, et B de coordonnées $(6 + 2\cos\alpha, 2\sin\alpha)$ vérifient :

$$\overrightarrow{OA} \cdot \overrightarrow{AB} = 0 \qquad \text{et} \qquad \overrightarrow{\Omega B} \cdot \overrightarrow{AB} = 0.$$

   On propose donc la méthode suivante :

— Discrétiser l'intervalle $[0, 2\pi]$ en 200 points équirépartis (on pourra changer cette valeur),
— balayer l'ensemble des $(\theta, \alpha) \in [0, 2\pi]^2$ en utilisant deux boucles `for`.
— Pour chacun de ces couples, calculer les coordonnées des points A et B correspondants, et regarder si ils vérifient les conditions.

Implémenter cette méthode pour trouver les points A et B solutions. Dessiner les tangentes correspondantes sur la figure.

Indications et consignes :
— On utilise des `array` pour les coordonnées des points A et B. L'avantage est que les coordonnées du vecteur $\overrightarrow{AB}$ est B−A (opérations entre `array`).
— Faire une fonction `pscal(u,v)` qui prend en entrée deux `array` de taille 2 (des vecteurs du plan donc) et qui calcule le produit scalaire `u` · `v`.
— On ne peut pas tester si le produit scalaire est nul, mais simplement s'il est inférieur à une valeur de référence `EPS` (en valeur absolue). On cherchera une bonne valeur pour `EPS`. De ce fait, il est possible que l'on trouve plusieurs solutions approchées qui ne correspondent qu'à une vraie solution.
— Afficher à l'écran les valeurs de A, B, θ, α, $\overrightarrow{OA} \cdot \overrightarrow{AB}$ et $\overrightarrow{\Omega B} \cdot \overrightarrow{AB}$.

(c) On constate sur le dessin précédent que :
— Le problème est symétrique par rapport à la droite horizontale $y = 0$, on peut donc se restreindre à $\theta \in [0, \pi]$.
— Pour une valeur de θ donné, il ne peut y avoir que deux α solutions : $\alpha = \theta$ et $\alpha = \pi + \theta$

Corriger la méthode précédente pour déterminer les points A et B solutions.

Comparer les deux méthodes, en particulier en termes de temps de calcul et de précision.

(d) On va chercher maintenant à approcher la solution où A a pour coordonnées $(\cos\theta, \sin\theta)$, et B $\Big(6 + 2\cos\theta, 2\sin\theta\Big)$, c'est-à-dire le cas précédent pour $\alpha = \theta$.

Écrire une fonction qui prend en entrée θ et calcule les produits scalaires $\overrightarrow{OA} \cdot \overrightarrow{AB}$ et $\overrightarrow{\Omega B} \cdot \overrightarrow{AB}$.

Dessiner les deux fonctions $\overrightarrow{OA} \cdot \overrightarrow{AB}$ et $\overrightarrow{\Omega B} \cdot \overrightarrow{AB}$ en fonction de θ.

(e) Faire de même pour le cas où A a pour coordonnées $(\cos\theta, \sin\theta)$, et B $\Big(6 + 2\cos(\theta + \pi), 2\sin(\theta + \pi)\Big)$, c'est-à-dire le cas précédent pour $\alpha = \theta + \pi$.