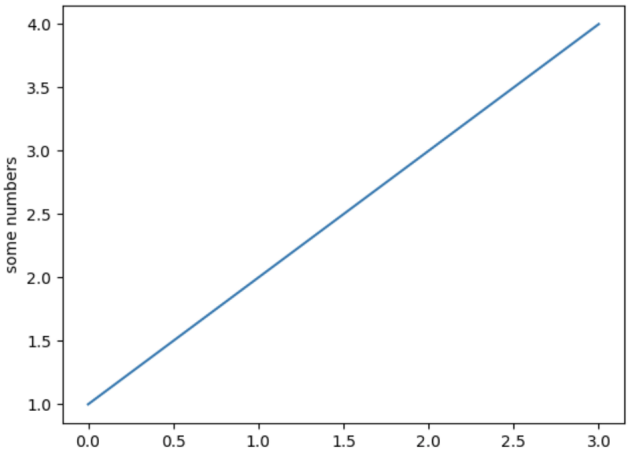
Table des matières

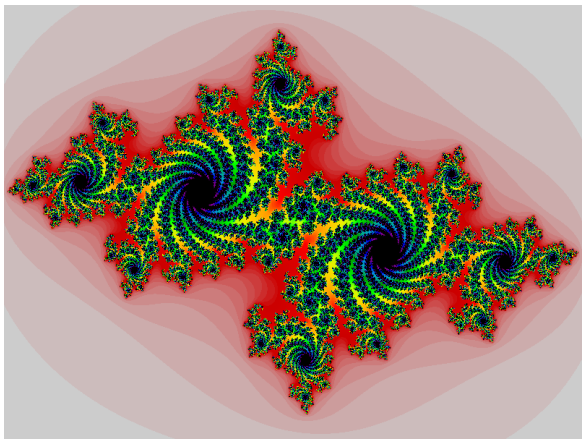
TP 5 : Matplotlib et Numpy

Vidéo d'intro (click !)

Ce TP a pour but de se familiariser avec deux bibliothèque Python : `Matplolib` et `Numpy`. La bibliothéque `Matplolib`[1] sert à faire des graphiques. Du plus simple :



au plus compliqué :



1. Matplotlib est l'abbréviatio, de Mathematic plot library, 'to plot' signifie 'tracer/faire le dessin' en anglais.

Evidemment on commencera par le plus simple et le plus compliqué n'est pas pour tout de suite (mais c'est abordable avant la fin de l'année, DM ? ?...)

La bibliothéque `NumPy`[2] est un module servant à manipuler des matrices ou des tableaux à plusieurs dimension.

Les deux sont liées car pour afficher des graphiques les fonctions de `Matplotlib` prennent en argument des tableaux de chiffres que l'on peut aisément manipuler grâce au module `NumPy`.

I Listes et Tableaux

I. 1 Listes Python

Video Liste

I. 1. a Définition

Les premiers tableaux que l'on va définir sont ce qu'on appelle des *listes*. Ce sont une succession de variables ordonnées dans un tableau à une dimension. En voici les régles :

1. Une liste est définie entre deux crochets.
2. On sépare deux variables à l'aide d'une virgule ','.
3. Les variables d'une liste ne sont pas obligés d'être du même type.

Exemple Voici comment définir deux listes :

```
1 L1=[1,2,67,22]
2 L2=['a','b', -3.14]
```

I. 1. b Appel des éléments

Pour appeler un élément d'une liste on utilise les crochets et le numéro de l'élément de la liste. Comme pour `range`, les listes commencent à l'indice 0. Ainsi

```
1 print(L1[1])
2 ---> 2
3  print(L2[0])
4 ---> 'a'
```

Vidéo Consignes exercices

Exercice 1. *Ecrire un script qui permet d'obtenir -3.14 avec la liste* `L2`

Si on dépasse le nombre d'élement dans la liste, une erreur Python s'affiche

```
1 print(L1[10])
2 ---> IndexError: list index out of range
```

Exercice 2. *Quelle est l'entier naturel maximal n0 que l'on peut utiliser pour que* `L1[n0]` *n'affiche pas d'erreur ?*

On peut aussi appeler les éléments par ordre inverse en utilisant des nombres négatifs :

```
1 print(L1[-1])
2 ---> 22
```

2. pour 'numerical python'

Ici, le dernier élément commence par l'index -1... il faut faire attention aux problèmes d'indices ! De la même façon qu'avec les nombres positifs si on choisit un index négatif trop grand (en valeur absolue) Python affichera un messag d'erreur.

Exercice 3. *Que vaut `L1[-4]` ?*
Quelle est l'entier relatif minimal $n1$ que l'on peut utiliser pour que `L1[n1]` n'affiche pas d'erreur ?

Enfin, on peut obtenir une sous-liste (une liste plus petite contenue dans la liste initiale) en utilisant les deux points ' :'. Soit `L` une liste et a un nombre plus petit que la taille de la liste, l'instruction `L[a:]` est la liste de tous les termes de `L` à partir de l'indice `a` (attention aux indices !)

Exemple Voici ce que ca donne sur un exemple :

```
1 L1=[2:]
2 --->[67,22]
```

Exercice 4. *Que donnent les instructions : `L1[1:]` ? `L1[:-1]` ? `L1[:]` ? `L1[1:3]` ?*

I. 1. c Opérations sur les listes

Vidéo Operation liste

On peut connaitre le nombre d'éléments dans une liste grâce à la commande `len`. On parle alors de la 'taille' ou de la 'longueur' de la liste.

Exemples On obtient avec les listes du paragraphe précedent :

```
1 print(len(L1))
2  ---> 4
3  print(len(L2))
4  ---> 3
```

En particulier, remarquer que le dernier élément s'obtient grâce à `L1(len(L1)-1)` et non pas `L1(len(L1))` qui lui affiche un message d'erreur.

Exercice 5. *Qu'affiche `L2(-len(L2))` ?*

On peut concaténer[3] deux listes grâce à l'opération '+'

```
1 print(L1+L2)
2  ---> [1,2,67,22,'a','b', -3.14]
```

Remarquez que `L1+L2` est différente de `L2+L1`.

Exercice 6. *Soit `K1` et `K2` deux listes telles que `len(K1)=n1` et `len(K2)=n2`. Que vaut `len(K1+K2)` ?*
Soit `K1` une liste de taille $n1$ avec $n1 \geq 2$. Que vaut `len(K1[2:])` ? et `len(K1[:-1])` ?

I. 2 Un premier graphique

Vidéo Graphique

3. Chercher dans un dictionnaire ou sur internet si vous ne connaissez pas le mot

I. 2. a Importer pyplot

Avant de parler des tableaux, on va essayer de faire notre premier graphique. Pour cela il faut importer le module 'matplotlib' et plus particulièrement le sous-module pyplot. Pour cela on utilise la commande suivante

```
import matplotlib.pyplot as plt
```

"import maplotlib.pyplot" permet d'importer la fonction voulue, "as plt" permet d'y faire appelle en raccourci. Cette facon d'appeler la fonction maplotlib.pyplot, même si elle n'est pas obligatoire, elle est ultra standard.

I. 2. b Tracer un graphique

Une fois le sous-module pyplot importé on peut tracer un graphique avec la fonction plot de ce module. La facon la plus simple est de donner à plot deux listes X et Y de même taille et Python tracera alors la suite de points dont les coordonnées sont $(X[i], Y[i])$.

```
plt.plot([0,1,2,3],[3,4,-1,2])
```

'Monsieur... il se passe rien ?' Au même titre que lorsque Python fait un calcul il n'est pas obligé de l'afficher. Quand il fait un dessin il n'est pas obligé de vous le montrer. Il faut le lui demander avec la fonction show.

Réessayer avec

```
plt.plot([0,1,2,3],[3,4,-1,2])
plt.show()
```

Par défaut, la fonction plot relie les points entre eux avec des traits bleus. On peut lui demander de seulement marquer les points et/ou de changer de couleur en rajoutant une variable optionnelle. Essayer le code suivant :

```
plt.plot([0,1,2,3],[3,4,-1,2], 'ro')
plt.show()
```

Voici une liste des couleurs principales :

On peut toutes les trouver sur https://matplotlib.org/3.1.0/gallery/color/named_colors.html

Tester les codes suivants

```
plt.plot([0,1,2,3],[3,4,-1,2], 'bs')
plt.show()
```

et

```
plt.plot([0,1,2,3],[3,4,-1,2], 'g^')
plt.show()
```

Vidéo quelques conseils

Exercice 7. ***Calcul et représentation du n-ième terme d'une suite***
On considère une suite définie par $u_0 \in \mathbb{R}$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = \ln(1+u_n^2)$. Écrire un script qui demande à l'utilisateur un nombre u_0 et un entier n puis qui calcule le terme u_n de la suite.

Créer deux listes X et Y, qui contiennent respecitvement les nombres de 0 à n et les termes u_0 à u_n.

Afficher les termes $u_0, \dots, u_n$ sur un graphique. Quel comportement pouvez-vous conjecturer quand n tend vers $+\infty$?

Vidéo Rappel boucle while

Exercice 8. ***Moyenne arithmético-géométrique.***
Soient a, b des réels tels que $0 < a < b$ On pose $u_0 = a$ et $v_0 = b$. Pour tout $n \in \mathbb{N}$, on définit

$$u_{n+1} = \sqrt{u_n v_n} \qquad et \qquad v_{n+1} = \frac{u_n + v_n}{2}.$$

1. *Calculer et tracer les termes $u_0, \dots, u_{10}$ et $v_0, \dots, v_{10}$ de ces suites sur un même graphique.*
2. *On peut conjecturer (et on pourrait démontrer) que ces deux suites sont adjacentes, qu'elles convergent donc vers la même limite ℓ (cette limite est appelée moyenne arithmético-géométrique de a et b), et qu'on a de plus : $\forall n \in \mathbb{N},\ u_n \leq \ell \leq v_n$.*
   Ainsi, pour obtenir une valeur approchée de la limite ℓ à ε près, il suffit de calculer u_n (valeur approchée par défaut) ou v_n (valeur approchée par excès) tels que : $v_n - u_n < \varepsilon$.
   Écrire un script qui demande les valeurs de a et b ainsi que la précision ε voulue, et qui affiche une valeur approchée de ℓ à ε près, ainsi que le nombre de termes qu'il a fallu calculer.

Vidéo Rappel randint

Exercice 9 (Marche aléatoire)**.** *Demander à l'utilisateur d'entrer un entier n.*
Créer une liste X de taille n comportant les nombres entiers de 0 à n.
Créer une liste Y de taille n dont les éléments sont pris au hasard dans $\{-1, 1\}$.
Créer une liste Z de taille n tel que $Z[i]$ soit la somme des termes de Y entre 0 et i.
Afficher sur un graphique la suite Z. Faites plusieurs essais avec n de plus en plus grand.

I. 3 Tableaux et Numpy

Vidéo Numpy

Afin de faire des graphiques un peu plus compliqué, on va utiliser le module Numpy qui permet de prendre en charge des tableaux un peu plus compliqué que les listes que l'on a vu à la Section I.1.

I. 3. a Importer le module

Comme pour matplotlib nous allons importer le module avec un raccourci

```
1 import numpy as np
```

Cette facon d'appeler la fonction `numpy`, même si elle n'est pas obligatoire, elle est ultra standard.

I. 3. b `np.array`

L'objet de base de la bibliothéque Numpy est l'objet de type `ndarray` pour 'n dimensional array' soit en bon français 'tableaux de dimension n'.

Pour le définir on utilise la commande `array` par exemple :

```
1 A1=np.array([1,2,67,  22])
```

Et la vous allez me dire : 'tout ça pour ça' ! On avait déjà les listes pourquoi s'encombrer de ce machin... par ce qu'il est beaucoup plus simple de faire des opéra tions mathématiques desssus.

```
A1=np.array([1,2,67, 22])
A2=np.array([3,1,33, -22])
print(A1+A2)
--->[4, 3, 100, 0]
```

Au lieu de concaténer les deux tableaux, python les a additionés éléments par éléments ! Et ca change tout ! On verra ca plus en détail avec les matrices au cours de l'année.

On ne pouvait par exemple par soustraire deux listes, ca n'avait pas de sens. On peut en revanche soustraire deux tableaux numpy :

```
A1=np.array([1,2,67, 22])
A2=np.array([3,1,33, -22])
print(A1-A2)
--->[-2, 1, 34, 44]
```

Mieux, on peut appliquer une fonction à tous les éléments d'un coup !

```
A1=np.array([1,2,67, 22])
print(A1**2)
--->[1,4,4489,484]
```

Chaque élément est mis au carré ou encore

```
A1=np.array([0,np.pi,1, np.pi/4])
print(np.cos(A1))
--->[ 1.,-1.,0.54030231,0.70710678]
```

Calcul le cosinus de chaque élément ! (np.pi est la commande pour obtenir π, enfin une approximation de π à 10^{-16} prés.

I. 3. c Fonction `np.linspace`

Vidéo linspace

On va finir ce TP sur cette fonction. Elle permet d'obtenir un tableau Numpy de nombres espacés linéairement. Ainsi `np.linspace(a, b, n)` divise l'intervalle $[a, b]$ en n part égal (en particulier n doit être entier). Tester :

```
x = np.linspace(-1.2, 2, 10)
print(x)
```

On va alors pouvoir tracer des plus belles fonctions. Tester :

```
import numpy as np
import matplotlib.pyplot as plt
X=np.linspace(-2*np.pi, 2*np.pi,1000)
Y=np.cos(X)
plt.plot(X,Y)
plt.show()
```

Magie !

On pourra utiliser ces commandes classiques pour l'aspect du graphique :

- Pour imposer la taille des fenêtres : `plt.axis([xmin,xmax,ymin,ymax])`
- Pour imposer un repère orthonormé : `plt.axis('equal')`
- Pour afficher un quadrillage : `plt.grid()`
- Pour effacer un ancien dessin avant d'en entamer un nouveau : `plt.clf()`

Exercice 10. *Afficher la courbe représentative entre* $[-10, 10]$ *de la fonction*

$$f : x \mapsto e^{-x^2}$$

Exercice 11. *Sur un même graphique, afficher la courbe représentative entre* $[-10, 10]$ *des fonctions*

$$f : x \mapsto \sin(x)$$

et

$$g : x \mapsto x$$

Exercice 12. *Sur un même graphique, afficher la courbe représentative entre* $]-1, 10]$ *des fonctions*

$$f : x \mapsto \ln(1+x)$$

et

$$g : x \mapsto x$$

Exercice 13. *Afficher la courbe représentative de f sur son ensemble de définition avec f définie par :*

$$f : x \mapsto \sqrt{1-x^2}$$

Comment tracer le cercle de centre 0 et de rayon 1 ?

Les Exercices sont à rendre sur

`https://genus2.fr/1BIOA/`

avant samedi 21 novembre 18h.

Exercice Bonus pour ceux qui aiment l'info seulement

On peut vérifier si un élément apparait dans une liste et connaitre sa position dans la liste grâce aux commande `in` et `index`. Voici leur syntaxe respective :

```
1 in L1
——> True
'c' in L2
——> False
```

Dans le cas où l'élément est dans la liste on peut connaitre l'indice de la première fois où il apparait grâce à index :

```
L1.index(1)
——> 0
L2.index('c')
——> ValueError: 'c' is not in list
```

Comme 'c' n'est pas un élément de L2, la deuxième instruction renvoit le message d'erreur disant que c n'appartient pas à la liste.

Le but de l'exercice est de créer une fonction qui code un texte grâce à la règle 'AVOCAT' (A vaut K). Une chaine de caractère est ainsi transformée en appliquant la règle à chaque lettre. A–>K, B–>L, C–>M ... etc quand on arrive à P–> Z on revient au début de l'alphabet et on a Q–>A... et ainsi de suite.

Ainsi la chaine de caractère `'BON COURAGE'` donne après le codage `'LYX MYEBKQO'`

Exercice 14 (Code Secret)**.**

1. *Créeer une liste* `alphabet`*, de taille 26, qui contient toutes les lettres de l'alphabet disons en majsucule (attention, pour qu'une lettre ne soit pas considérée comme une variable il faut mettre des appostrophes de par et d'autre)*
2. *Grace à la fonction* `index` *donner la position de 'K' dans l'alphabet (attention aux indices !)*

3. *A partir de cette liste, définir une liste* `debut` *qui comporte les premières lettre de l'alphabet de 'A' à 'J' et* `fin` *dont les élements vont de 'K' à 'Z'.*

4. *Définir la liste* `alphacode=fin+debut'`*. (Réfléchir à la place de 'K' dans la liste* `alphacode` *par rapport à celle de 'A' dans* `alphabet`*). Que donne l'instruction :*

```
alphacode[alphabet.index('P')]
```

   Créer une fonction `code_lettre` *qui prend en paramètre une lettre et rend sa valeur codée par la règle 'A vaut K' décrite précédemment.*

5. *Que renvoie l'instruction suivante*

```
for lettre in 'coucou a toi':
    print(lettre)
```

6. *Créer une fonction* `code` *qui prend en paramètre une chaine de caractère et qui renvoie le texte codé par la règle décrite précédemment.*

   Il faudra faire attention aux espaces dans la chaine de caractères. En effet une espace `' '` *est un caractère pour Python. Donc si on lui demande* `alphabet.index(' ')` *il va rendre une erreur car l'espace n'est pas dans* `alphabet`*. On pourra pour résoudre ce problème faire une condition dans les instructions de la fonctions : 'si le caractère est une espace alors ... sinon ... . '*

7. *Créer une fonction qui décode un message codé par la règle Avocat.*