Réflexe :
Importer les librairies nécessaires à la représentation graphique en Python.

Entrée[5]:
```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

TP Python 8 : Représentation graphique de suites

Correction

Partie I : Échauffement

Exercice 1

On s'intéresse à la suite (u_n) définie par $u_n = \left(1 + \frac{1}{n}\right)^n$ pour $n \in \mathbb{N}^*$.

1. Montrer que $\lim\limits_{n \to +\infty} u_n = e$.
2. Écrire un programme qui renvoie la liste `rangs` des entiers de 1 à 50 compris et la liste `termes` des 50 premiers termes de cette suite.
3. Compléter le script précédent pour tracer un graphique, où l'on voit les cinquante premiers termes de la suite (u_n) en fonction de n.
4. Compléter le programme pour afficher sur le même graphique l'asymptote horizontale d'équation $y = e$.

Rappel : *Pour tracer des points et non des lignes brisées, on pensera à mettre en option* `'x'` *ou autre symbole de son choix dans la fonction* `plot` *(cf. TP précédents).*

Entrée[22]:

```
rangs = np.linspace(1,50,50)
termes = []
asymptote = []

for k in range(len(rangs)):
    u = (1+1/rangs[k])**rangs[k]
    termes.append(u)
    # la liste "asymptote" est une liste contenant toujours le mé
    # il faut qu'il y ait autant de fois le nombre e dans cette l
    # dans la liste "termes" : autrement dit, cinquante fois.
    asymptote.append(np.exp(1))

# On pouvait aussi définir rangs à l'aide de range:
# rangs = range(1,51)
# Cela ne change rien et est tout à fait acceptable.

plt.clf()
plt.plot(rangs,termes,'r+')
plt.plot(rangs, asymptote,'b')
plt.show()
```

Partie II : Représentation "en escalier"

On s'intéresse ici à une suite récurrente (u_n), définie par un certain terme initial $u_0 \in \mathbb{R}$ puis par la relation de récurrence :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = f(u_n),$$

où f est une fonction donnée.

On souhaite représenter la construction terme-à-terme d'une telle suite comme on le ferait sur papier.

II.1) La fonction `np.arange`

- La fonction `arange` de la bibliothèque `numpy` est une fonction prête à l'emploi qui permet de générer une liste de valeurs.
- Elle prend trois variables en entrée : la commande `range(a,b,c)` énumère les entiers a, $a+c$, $a+2c$, ... $a+nc$ où n est le plus grand entier vérifiant $a+nc < b$ (autrement dit, on parcourt l'ensemble des entiers de a à $b-1$ avec un pas de c).
- C'est une alternative à la fonction `np.linspace` . On utilisera `np.arange` si l'on souhaite utiliser un pas de discrétisation particulier, ou bien `np.linspace` si l'on veut un certain nombre de points dans notre discrétisation.

II.2) Mise en application

Exercice 2
Exécuter le script suivant. Que fait-il?

Entrée[30]:

```
# on définit une fonction g
def g(x):
    return 1/(1+x+x**2)

# on définit une liste u des termes d'une suite initialisée à 1,
u=[1]
# on définit le nombre d'itérations
n=7

# on calcule les n itérations de la suite avec une boucle
for i in range(0,n):
    u.append(g(u[i]))
    print('u[',i,']=',u[i]) # on affiche le calcul des termes, or
# on définit les points d'abcisses qui vont servir pour tracer le
pas= 0.01
x=np.arange(0,1,0.01)
# on définit la liste des ordonnées
gx=[g(a) for a in x]

# on trace les courbes
plt.clf()
plt.plot(x, gx, 'b-',label='g(x)')
plt.plot(x, x, 'r-',label='y=x')

# on trace maintenant des traits noirs, k correspond à "black"
for i in range(0,n-1):
    plt.plot([u[i], u[i+1]], [u[i+1], u[i+1]], 'k-')
    plt.plot([u[i+1], u[i+1]], [u[i+1], u[i+2]], 'k-')
    plt.plot([u[i], u[i]], [0, u[i+1]], 'k--')
# il manque le dernier trait alors on le rajoute à la main
plt.plot([u[n-1], u[n-1]], [0, u[n]], 'k--')

# on rajoute des commentaires sur le graphe (Hors Programme)
plt.annotate('$u_0$', xy=(u[0], 0), xytext=(u[0]+.1, .1),arrowpro
plt.annotate('$u_1$', xy=(u[1], 0), xytext=(u[1]-.1, .1),arrowpro

# on met un titre (Hors Programme)
plt.title('Etude de ...')

# on affiche le dessin
plt.legend(loc='upper right')
plt.show()
```

```
u[ 0 ]= 1
u[ 1 ]= 0.3333333333333333
u[ 2 ]= 0.6923076923076923
u[ 3 ]= 0.4604904632152589
u[ 4 ]= 0.5978923350778832
u[ 5 ]= 0.5114127954884086
u[ 6 ]= 0.5640298397826446
```

Exercice 3

Écrire un programme qui représente la construction de la suite récurrente (v_n) définie par $v_0 = 6$ et la fonction $f : x \mapsto \frac{x^2}{10} + 1$

Entrée[38]:

```
# on définit une fonction f
def f(x):
    return (x**2/10) +1

# on définit une liste u des termes d'une suite initialisée à 1,
v=[6]
# on définit le nombre d'itérations
n=7

# on calcule les n itérations de la suite avec une boucle
for i in range(0,n):
    v.append(f(v[i]))
    print('v[',i,']=',v[i]) # on affiche le calcul des termes, or
# on définit les points d'abcisses qui vont servir pour tracer le
pas= 0.01
x=np.arange(0,7,0.01)
# on définit la liste des ordonnées
y=[f(k) for k in x]

# on trace les courbes
plt.clf()
plt.plot(x, y, 'b-',label='f(x)')
plt.plot(x, x, 'r-',label='y=x')

# on trace maintenant des traits noirs, k correspond à "black"
for i in range(0,n-1):
    plt.plot([v[i], v[i+1]], [v[i+1], v[i+1]], 'k-')
    plt.plot([v[i+1], v[i+1]], [v[i+1], v[i+2]], 'k-')
    plt.plot([v[i], v[i]], [0, v[i+1]], 'k--')
# il manque le dernier trait alors on le rajoute à la main
plt.plot([v[n-1], v[n-1]], [0, v[n]], 'k--')

# on rajoute des commentaires sur le graphe (Hors Programme)
plt.annotate('$v_0$', xy=(v[0], 0), xytext=(v[0]+.1, .1),arrowpro
plt.annotate('$v_1$', xy=(v[1], 0), xytext=(v[1]+.1, .1),arrowpro

# on met un titre (Hors Programme)
plt.title('Etude de ...')

# on affiche le dessin
plt.legend(loc='upper right')
plt.show()
```

```
v[ 0 ]= 6
v[ 1 ]= 4.6
v[ 2 ]= 3.1159999999999997
v[ 3 ]= 1.9709455999999999
v[ 4 ]= 1.3884626558159359
v[ 5 ]= 1.1927828546595443
v[ 6 ]= 1.1422730938369772
```

Partie III : Cabinet de curiosités

Exercice 4 : Suite de Fibonacci

La suite de Fibonacci est une suite énoncée au début du XIIIe siècle en Italie par Leonardo Fibonacci, aussi appelé Leonardo Pisano, dans son traité mathématique "*Liber Abaci*" (1202). Elle apparaît dans un problème récréatif sur l'étude d'une population de lapins :

« Quelqu'un a déposé un couple de lapins dans un certain lieu, clos de toutes parts, pour savoir combien de couples seraient issus de cette paire en une année, car il est dans leur nature de générer un autre couple en un seul mois, et qu'ils enfantent dans le second mois après leur naissance. »

Cette suite se définit facilement en posant $F_0 = 0$, $F_1 = 1$ et via la relation de récurrence :

$$F_{n+2} = F_{n+1} + F_n, \qquad \forall n \geq 2.$$

1. À l'aide de la relation de récurrence, exprimer le terme général F_n pour tout $n \geq 2$.
2. Compléter la fonction `fibonacci` ci-dessous qui prend en argument un entier naturel n et renvoie la liste contenant les $n + 1$ premiers termes de la suite de Fibonacci.
   NB : il s'agit d'une fonction écrite naïvement : il existe des écritures beaucoup plus concises, mais pas encore atteignable en cette période de l'année.
3. En ajoutant à la suite de la fonction `fibonacci` un programme adéquat, représenter graphiquement les trentes premiers termes de la suite de Fibonacci.

Entrée[83]:

```python
def fibonacci(n):
    liste = []

    if n==0:
        liste.append(0)

    if n==1:
        liste.append(0)
        liste.append(1)
    else:
        liste.append(0)
        liste.append(1)
        for k in range(2,n+1):
            Fn = liste[k-1]+liste[k-2]
            liste.append(Fn)

    return liste

fibonacci(1)


# 10 premiers termes de Fibonacci
rangs = np.linspace(1,31,32)  # Nombre de termes à calculer
F = fibonacci(31)


# Tracé des résultats
plt.clf()
plt.plot(rangs, F, marker='o', label="Suite de Fibonacci")
plt.title("Représentation graphique de la suite de Fibonacci")
plt.xlabel("n (indice)")
plt.ylabel("Fibonacci(n)")
plt.show()

```

Exercice 5 : Termes alternés

On considère la suite (u_n) définie pour tout $n \in \mathbb{N}^*$ par $u_n = \dfrac{(-1)^n}{n}$.

1. Écrire une fonction Python `suite_alt` qui prend en argument un entier naturel n et renvoie le n-ième terme u_n de la suite $(u_n)_{n\in\mathbb{N}^*}$.
2. Écrire une programme Python permettant de représenter les trentes premiers termes de cette suite, en affichant en rouge (`'r'`) les termes d'ordre pair et en bleu (`'b'`) les termes d'ordre impair.

→Indice ←

Entrée[87]:

```
def suite_alt(n):
    if n == 0:
        print("Erreur, n doit être non nul")
        return
    else:
        u = ((-1)**n)/n
    return u

U_pair = []
rangs_pairs = []
U_impair = []
rangs_impairs = []


for n in range(1,31):
    u = suite_alt(n)
    if n%2 ==0:
        U_pair.append(u)
        rangs_pairs.append(n)
    else :
        U_impair.append(u)
        rangs_impairs.append(n)

plt.clf()
plt.plot(rangs_pairs,U_pair,'+r')
plt.plot(rangs_impairs,U_impair,'+b')
plt.title("Termes de la suite alternée ((-1)**n)/n")
plt.legend(["pairs","impairs"])
plt.show()
```