Correction Oraux Banque PT

August 4, 2020

[1]:
```
from math import *
import random
import numpy as np
import time,os
import matplotlib.pyplot as plt
```

1 Exercice 1

Q1) Ecrire une fonction permettant de calculer le $n-$ième terme d'une suite est une question classique. Cependant, il faut faire attention aux cas aux bords. Pour $n = 0$, il y a 0 itération de boucle : la fonction retourne bien $u_0 = 0$. Pour $n = 1$, il y a 1 itération de boucle (car range(1,2)=[1]) : la fonction retourne bien u_1. Pour $n = n$, il y a n itérations de boucle (car range(1,n+1)=[1,$\cdots$,n]) : la fonction retourne bien u_n.

[2]:
```
def terme(n):
    u=0
    for k in range(1,n+1):
        u=u+(-1)**(k-1)/(2*k-1)
    return u

print("Le 3-ième terme de la suite vaut S_3=",terme(3))
```

```
Le 3-ième terme de la suite vaut S_3= 0.8666666666666667
```

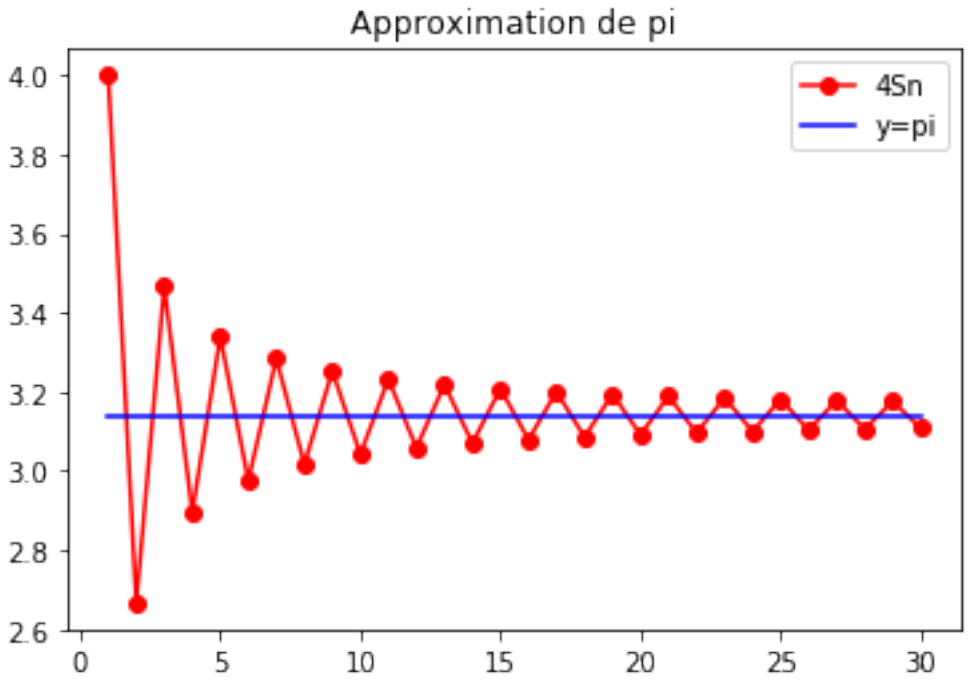
Q2) Dire que S_n est une approximation de $\frac{\pi}{4}$ d'erreur inférieure à $\frac{1}{2n+1}$ signifie que $\left|S_n - \frac{\pi}{4}\right| \leq \frac{4}{2n+1}$ i.e. $|4S_n - \pi| \leq \frac{4}{2n+1}$ Pour que $|4S_n - \pi| \leq \varepsilon$, il suffit de choisir n tel que $\frac{4}{2n+1} \leq \varepsilon$. Signalons au passage que calculer $4S_n$ pour n suffisamment grand permet d'obtenir une valeur approchée de π, et qu'en conséquence il ne faut surtout pas utiliser la valeur de π dans l'écriture d'une fonction permettant de calculer une valeur approchée de π !

[3]:
```
def approx_pi(eps):
    n=1
    while (4/(2*n+1)>=eps):
        n=n+1
    return 4*terme(n)

print("Une approximation de pi à 10e-5 près est",approx_pi(10e-5))
```

```
Une approximation de pi à 10e-5 près est 3.1415426535898248
```

Q3) Les listes en compréhension, bien que hors-programme, sont très commodes. Pour définir la liste X des abscisses, on aurait pu aussi utiliser np.linspace() du module numpy. Il est utile de savoir légender une figure un minimum.

[4]:
```python
def graphe_pi(n):
    #listes en compréhension
    X=[k for k in range(1,n+1)]
    Y=[4*terme(k) for k in X]
    plt.plot(X,Y,'r',marker='o',label='4Sn')
    plt.plot([1,n],[pi,pi],'b',label='y=pi')
    plt.title("Approximation de pi")
    plt.legend()
    plt.show()
    return

graphe_pi(30)
```



2 Exercice 2

Q1) L'utilisation de return judicieusement placé permet d'éviter des else et l'utilisation de variables intermédiaires. C'est une façon de coder à privilégier.

[5]:
```
def degre(P):
    if P[-1]!=0:
        return len(P)-1
    return -1

print("Le degré du polynôme X^5+5X^2-3X+1 est",degre([1,-3,5,0,0,5]))
print("Le degré du polynôme nul est",degre([0]))
```

```
Le degré du polynôme X^5+5X^2-3X+1 est 5
Le degré du polynôme nul est -1
```

Q2) Les listes en compréhension, bien que hors-programme, sont particulièrement commodes pour avoir un code concis et clair.

[6]:
```
def derive(P):
    return [(i+1)*P[i+1] for i in range(len(P)-1)]

print("La liste associée à la dérivée du polynôme X^5+5X^2-3X+1␣
 ↪est",derive([1,-3,5,0,0,5]))
```

```
La liste associée à la dérivée du polynôme X^5+5X^2-3X+1 est [-3, 10, 0, 0, 25]
```

Q3) Pour retirer un élément d'une liste L, il y a plusieurs façons de faire : - L.pop(i) retire l'élément d'indice i de la liste L ET le renvoie en valeur de retour, - L.remove(x) supprime l'élément x de la liste L, - del L[i:j+1] supprime les éléments d'indice i à j (inclus).

[7]:
```
def somme(P,Q):
    p,q=len(P),len(Q)
    #result est la liste des coeff du polynôme P+Q
    result=[]
    if p>=q:
        result=P.copy()
        for k in range(q):
            result[k]=result[k]+Q[k]
    else:
        result=Q.copy()
        for k in range(p):
            result[k]=result[k]+P[k]
    #retrait des éventuels 0 en fin de liste, en prenant la convention que la␣
 ↪liste du polynôme nul est [0]
    while result[-1]==0 and len(result)>1:
        result.pop(-1)
    return result

print("La liste associée à la somme de 7X^3+X^2+4X-1 et -2X+5␣
 ↪est",somme([5,-2],[-1,4,1,7]))
```

```
La liste associée à la somme de 7X^3+X^2+4X-1 et -2X+5 est [4, 2, 1, 7]
```

Q4) Sauf mention contraire, il faut privilégier les fonctions natives de Python. Typiquement, il est

préférable d'utiliser la fonction native sum(L) pour calculer la somme des éléments de la liste L, plutôt que de boucler.

[8]:
```
def eval(P,y):
    #liste des a_kx^k si P=[a_0,a_1,...,a_n]
    L=[P[k]*y**k for k in range(len(P))]
    return sum(L)

print("Si P=X^2+2X+5, alors l'évaluation en 5 donne P(5)=",eval([5,2,1],5))
```

```
Si P=X^2+2X+5, alors l'évaluation en 5 donne P(5)= 40
```

Q5) Rappelons que le coefficient devant X^k d'un produit PQ est donné par la formule : $c_k = \sum_{i=0}^{k} a_i b_{k-i} = \sum_{\text{t.q. } i+j=k}^{i,j} a_i b_j$. Comparer le code suivant au vôtre !

[9]:
```
def produit(P,Q):
    p,q=len(P),len(Q)
    #création d'une liste de p+q-1 éléments
    result=(p+q-1)*[0]
    for i in range(p):
        for j in range(q):
            result[i+j]+=P[i]*Q[j]
    return result

print("La liste associée au produit des polynômes X^2+1 et 7X-2␣
 ↪est",produit([1,0,1],[-2,7]))
```

```
La liste associée au produit des polynômes X^2+1 et 7X-2 est [-2, 7, -2, 7]
```

Q6) Coder l'algorithme d'Euclide pour déterminer le reste d'une division euclidienne entre deux polynômes, ou le pgcd de deux entiers, est un classique à savoir faire.

[10]:
```
def euclide(P,Q):
    q=degre(Q)
    while degre(P)>=degre(Q):
        p=degre(P)
        #création de la liste associée au polynôme B=-X^{p-q}dom(P)/dom(Q)
        B=[0]*(p-q)
        B.append(-P[-1]/Q[-1])
        #P+BQ pour retirer le monôme dominant de P
        P=somme(P,produit(B,Q))
    return P

print("La liste associée au reste de la division euclidienne de X^5-1 par X-1␣
 ↪est",euclide([-1,0,0,0,0,1],[-1,1]))
print("La liste associée au reste de la division euclidienne de X^5-X+1 par␣
 ↪X^2+X+2 est",euclide([1,-1,0,0,0,0,0,0,0,0,1],[2,1,0,1]))
```

```
La liste associée au reste de la division euclidienne de X^5-1 par X-1 est [0.0]
La liste associée au reste de la division euclidienne de X^5-X+1 par X^2+X+2 est
[13.0, -1.0, -11.0]
```

3 Exercice 3

Q1) Commençons par déterminer une équation cartésienne de la droite (AB) : Soit $M(x,y)$ un point quelconque du plan. Alors on a les équivalences suivantes : $M(x,y) \in (AB) \Leftrightarrow \det(\overrightarrow{AM}, \overrightarrow{AB}) = 0_{\mathbb{R}} \Leftrightarrow \begin{vmatrix} x-a & b-a \\ y-f(a) & f(b)-f(a) \end{vmatrix} = 0_{\mathbb{R}} \Leftrightarrow (x-a)(f(b)-f(a)) - (y-f(a))(b-a) = 0_{\mathbb{R}}$ Ainsi, une équation cartésienne de la droite (AB) est donnée par : $(f(b)-f(a))x + (a-b)y + (-a(f(b)-f(a)) + f(a)(b-a)) = 0_{\mathbb{R}}$. En nullifiant y, il vient que l'abscisse c du point d'intersection de la droite (AB) avec l'axe des abscisses est donné par :

$$c = a - \frac{f(a)(b-a)}{f(b)-f(a)} = \frac{af(b)-bf(a)}{f(b)-f(a)}$$

[11]:
```
#Q2)
def itera(f,a,b):
    return (a*f(b)-b*f(a))/(f(b)-f(a))
```

Q3) Coder une fonction donnant le terme général d'une suite récurrente linéaire d'ordre est un must-know indispensable. Une façon de faire est d'utiliser des listes comme suit :

[12]:
```
def secante(N,f,u0,u1):
    Y=[u0,u1]
    for k in range(2,N):
        Y.append(itera(f,Y[k-2],Y[k-1]))
    return Y

print(secante(10,lambda x:x**2-2,0,2))
```

```
[0, 2, 1.0, 1.3333333333333333, 1.4285714285714286, 1.4137931034482758,
1.41421143847487, 1.4142135626888699, 1.414213562373095, 1.4142135623730951]
```

Q4b) Il suffit de prendre N suffisamment grand pour que u_N soit une valeur approchée de $\sqrt{2}$. Par exemple, $u_9 = 1.4142135623730951$ est une valeur approchée de $\sqrt{2}$.

Q5a)b) Pour que u_n soit une valeur approchée de $\sqrt{2}$ à 10^{-6} près i.e. $|u_n - \sqrt{2}| \leq 10^{-6}$, il suffit de choisir n tel que $2^{-\varphi^n} \leq 10^{-6}$. On a les équivalences suivantes :

$$2^{-\varphi^n} \leq 10^{-6} \Leftrightarrow -\varphi^n \ln(2) \leq -6 \Leftrightarrow \varphi^n \geq \frac{6}{\ln(2)} \Leftrightarrow n \geq \frac{\ln(6) - \ln(\ln(2))}{\ln(\varphi)}$$

Il suffit alors de choisir $n = E\left(\frac{\ln(6) - \ln(\ln(2))}{\ln(\varphi)}\right) + 1$, où $E(\cdot)$ désigne la fonction partie entière, pour être certain que u_n soit une approximation de $\sqrt{2}$ à 10^{-6} près.

[13]:
```
#Q5c)
def approx(a,b,p):
```

```
    n=0
    phi=(1+sqrt(5))/2
    while 2**(-phi**n)>10**(-p):
        n=n+1
    return n

print("Une valeur approchée de racine de 2 à 10e-6 près␣
 ↪est",secante(approx(0,2,6),lambda x:x**2-2,0,2)[-1])
print(sqrt(2))
```

```
Une valeur approchée de racine de 2 à 10e-6 près est 1.41421143847487
1.4142135623730951
```

4 Exercice 4

[14]:
```
#Q1) et Q2)
mon_fichier=open("pi.txt",'r')
decpi=mon_fichier.read()
mon_fichier.close()

print("Les 10 premières décimales de pi sont",decpi[1:11])
print("Les 10 'dernières' décimales de pi sont",decpi[-11:])
print("La longueur de la chaîne de caractères est",len(decpi))
```

```
Les 10 premières décimales de pi sont 1415926535
Les 10 'dernières' décimales de pi sont 5845602850

La longueur de la chaîne de caractères est 2401
```

Q3) La recherche d'une sous-chaine dans une chaîne de caractères est un algorithme au programme qu'il faut maîtriser ! On propose ici un algorithme naïf qui a le mérite d'être facile à comprendre, mais a une subtilité de taille (l'ordre des conditions dans la boucle while est important, comprenez-vous pourquoi ?).

[15]:
```
def appart(L,M):
    p,n = len(L),len(M)
    for i in range(n-p+1):
        j=0
        while j<p and L[j]==M[i+j]:
            j=j+1
        if j==p:
            return i
    return -1
```

Q4) On peut ou bien utiliser la fonction précédemment écrite ou utiliser le mot-clé "in" pour tester l'appartenance d'une sous-chaîne à une chaîne de caractères.

[16]:
```
print("314159" in decpi)
print("123456" in decpi)
print(appart("314159",decpi))
print(appart("123456",decpi))
```

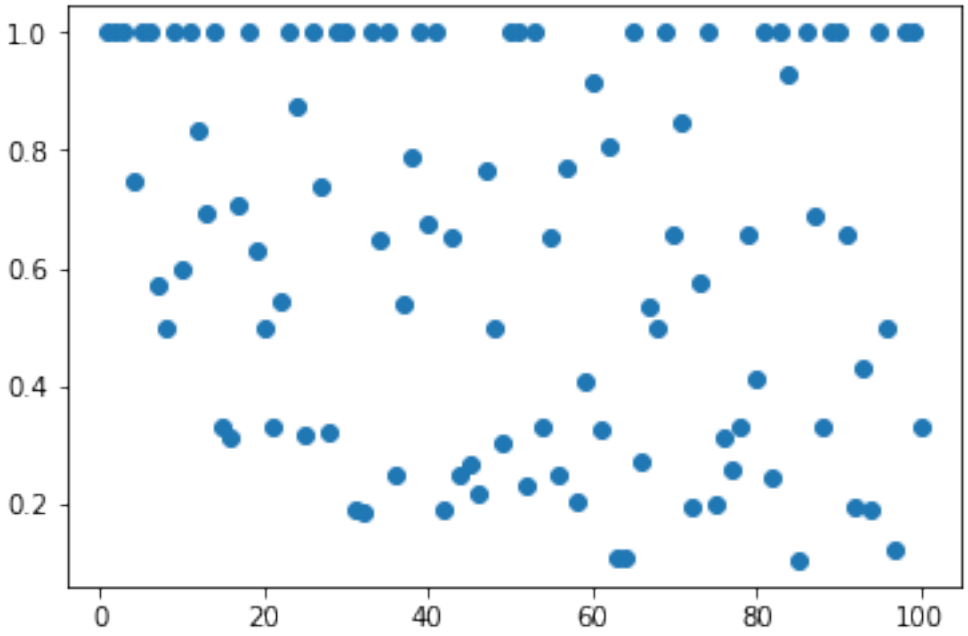
```
True
False
0
-1
```

5 Exercice 5

[17]:
```
#Q1)
def spirale(L):
    M=[]
    for i in range(len(L)//2):
        M.append(L[-1-i])
        M.append(L[0+i])
    if len(L)%2==1:
        M.append(L[len(L)//2])
    return M


#Q2)
L=[i for i in range(1,7)]
M=[i for i in range(1,8)]
for i in range(1,5):
    print(spirale(L))
    L=spirale(L)
    print(spirale(M))
    M=spirale(M)


#Q3)
def periode(n):
    L=spirale(list(range(1,n+1)))
    i=1
    while L!=list(range(1,n+1)):
        L=spirale(L)
        i=i+1
    return i


#Q4)
X=[i for i in range(1,101)]
Y=[periode(i)/i for i in X]
plt.plot(X,Y,'o')
```

```
plt.show()


#Q5)
print("La valeur de n comprise entre 1 et 100 minimisante est",Y.
 ↪index(min(Y))+1)
```

```
[6, 1, 5, 2, 4, 3]
[7, 1, 6, 2, 5, 3, 4]
[3, 6, 4, 1, 2, 5]
[4, 7, 3, 1, 5, 6, 2]
[5, 3, 2, 6, 1, 4]
[2, 4, 6, 7, 5, 3, 1]
[4, 5, 1, 3, 6, 2]
[1, 2, 3, 4, 5, 6, 7]
```




```
La valeur de n comprise entre 1 et 100 minimisante est 85
```

6 Exercice 6

[18]:
```
print(list(str(173471)))
```

```
['1', '7', '3', '4', '7', '1']
```

Ainsi, on observe que la commande list(str(n)) permet de récupérer la liste des chiffres qui composent n sous forme de chaîne de caractères.

Pour récupérer le chiffre des unités d'un nombre n donné : - on commence par créer la liste L des chiffres qui composent n sous forme de chaîne de caractères à l'aide de la commande $L = list(str(n))$, - puis on prend le dernier élément à l'aide de la commande L[-1], - enfin on transtype à l'aide de la fonction int().

[19]:
```
#Q2)
L=list(str(2019))
print(L)
print(int(L[-1]))
print(int(L[-2]))
```

```
['2', '0', '1', '9']
9
1
```

[20]:
```
#Q3)
def change(n):
    L=[int(i) for i in list(str(n))[::-1]]
    for i in range(1,len(L),2):
        if 2*L[i]<10:
            L[i]=2*L[i]
        else:
            L[i]=sum([int(j) for j in list(str(2*L[i]))])
    L_ch="".join([str(i) for i in L])
    return int(L_ch[::-1])

#Q4)
n=10000
L=[i for i in range(n+1) if change(i)==i]
print("La liste de tous les nombres inférieurs à 10000 inchangés par change␣
 ↪est\n",L,"de longueur",len(L))
```

```
La liste de tous les nombres inférieurs à 10000 inchangés par change est
 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
101, 102, 103, 104, 105, 106, 107, 108, 109, 190, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 290, 291, 292,
293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308,
309, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404,
405, 406, 407, 408, 409, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500,
501, 502, 503, 504, 505, 506, 507, 508, 509, 590, 591, 592, 593, 594, 595, 596,
597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 690, 691, 692,
693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708,
709, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804,
805, 806, 807, 808, 809, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900,
901, 902, 903, 904, 905, 906, 907, 908, 909, 990, 991, 992, 993, 994, 995, 996,
997, 998, 999, 9000, 9001, 9002, 9003, 9004, 9005, 9006, 9007, 9008, 9009, 9090,
9091, 9092, 9093, 9094, 9095, 9096, 9097, 9098, 9099, 9100, 9101, 9102, 9103,
9104, 9105, 9106, 9107, 9108, 9109, 9190, 9191, 9192, 9193, 9194, 9195, 9196,
```

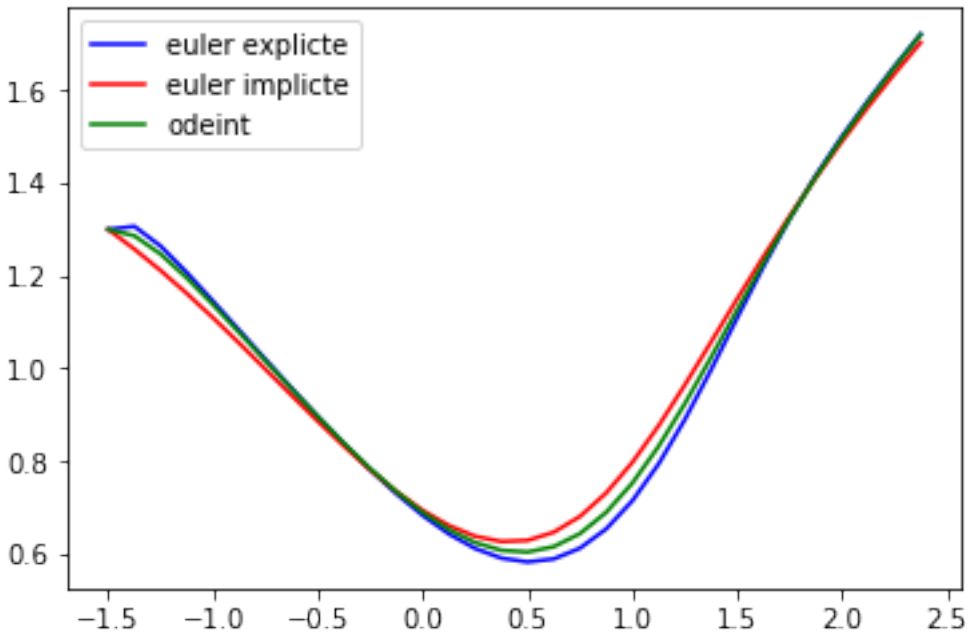
```
9197, 9198, 9199, 9200, 9201, 9202, 9203, 9204, 9205, 9206, 9207, 9208, 9209,
9290, 9291, 9292, 9293, 9294, 9295, 9296, 9297, 9298, 9299, 9300, 9301, 9302,
9303, 9304, 9305, 9306, 9307, 9308, 9309, 9390, 9391, 9392, 9393, 9394, 9395,
9396, 9397, 9398, 9399, 9400, 9401, 9402, 9403, 9404, 9405, 9406, 9407, 9408,
9409, 9490, 9491, 9492, 9493, 9494, 9495, 9496, 9497, 9498, 9499, 9500, 9501,
9502, 9503, 9504, 9505, 9506, 9507, 9508, 9509, 9590, 9591, 9592, 9593, 9594,
9595, 9596, 9597, 9598, 9599, 9600, 9601, 9602, 9603, 9604, 9605, 9606, 9607,
9608, 9609, 9690, 9691, 9692, 9693, 9694, 9695, 9696, 9697, 9698, 9699, 9700,
9701, 9702, 9703, 9704, 9705, 9706, 9707, 9708, 9709, 9790, 9791, 9792, 9793,
9794, 9795, 9796, 9797, 9798, 9799, 9800, 9801, 9802, 9803, 9804, 9805, 9806,
9807, 9808, 9809, 9890, 9891, 9892, 9893, 9894, 9895, 9896, 9897, 9898, 9899,
9900, 9901, 9902, 9903, 9904, 9905, 9906, 9907, 9908, 9909, 9990, 9991, 9992,
9993, 9994, 9995, 9996, 9997, 9998, 9999, 10000] de longueur 401
```

7 Exercice 7

Q1) Cf cours pour l'explication de la méthode d'Euler.

[21]:
```
a=1.3
h=1/8

def F(t,y):
    return t**2-y**3



#Q2)
def euler_explicit(t0,h,b,y0,F):
    X,Y=[t0],[y0]
    while t0+h<b:
        y0=y0+h*F(t0,y0)
        t0=t0+h
        X.append(t0)
        Y.append(y0)
    return X,Y

X,Yexp=euler_explicit(-1.5,h,2.5,a,F)
plt.plot(X,Yexp,'b',label='euler explicte')



#Q6)
def euler_implicit(t0,h,b,y0,F):
    X,Y=[t0],[y0]
    while t0+h<b:
        y0=y0+h*F(t0+h,y0+h*F(t0,y0))
        t0=t0+h
```

```
        X.append(t0)
        Y.append(y0)
    return X,Y

X,Y=euler_implicit(-1.5,h,2.5,a,F)
plt.plot(X,Y,'r',label='euler implicte')



#Q3)
from scipy.integrate import odeint

def F2(y,t):
    return t**2-y**3

X=euler_explicit(-1.5,h,2.5,a,F)[0]
Yexa=odeint(F2,a,X)
plt.plot(X,Yexa,'g',label='odeint')

plt.legend()
plt.show()
```



8 Exercice 8

La fonction set() permet de transtyper une chaîne de caractères ou une liste en l'ensemble (au sens mathématique et donc sans répétition) de ses éléments.

[22]:
```
#Q1)
def coupure(L):
    M=[]
    i=0
    while i<len(L):
        start=L[i]
        j=1
        while i+j<len(L) and L[i+j]==start:
            j=j+1
        if i+j<len(L):
            M.append(i+j)
        i=i+j
    return M

print(coupure([1,1,1,2,2,3,2,2,3,3,3]))
print(coupure("aaeeeabcbccaaabbaaceee"))


#Q2)
def CP(L):
    Ind=coupure(L)
    M=[Ind[0],L[0]]
    for i in range(1,len(Ind)):
        M.append(Ind[i]-Ind[i-1])
        M.append(L[Ind[i-1]])
    M.append(len(L)-Ind[-1])
    M.append(L[-1])
    return M

print(CP("aaeeeabcbccaaabbaaceee"))


#Q3)
def nbelem(L):
    return len(set(L))

print("Le nombre d'éléments distincts de 'acadabra' est",nbelem("acadabra"))
print("Le nombre d'éléments distincts de [1,1,1,2,2,3,2,3,3]␣
 ↪est",nbelem([1,1,1,2,2,3,2,3,3]))
```

```
[3, 5, 6, 8]
[2, 5, 6, 7, 8, 9, 11, 14, 16, 18, 19]
[2, 'a', 3, 'e', 1, 'a', 1, 'b', 1, 'c', 1, 'b', 2, 'c', 3, 'a', 2, 'b', 2, 'a',
```

```
1, 'c', 3, 'e']
Le nombre d'éléments distincts de 'acadabra' est 5
Le nombre d'éléments distincts de [1,1,1,2,2,3,2,3,3] est 3
```

9 Exercice 9

[23]:
```python
#Q1)
def nbdiv(n):
    L=[]
    for k in range(1,n+1):
        if n%k==0:
            L.append(k)
    return len(L)

print("Le nombre de diviseurs de 60 est",nbdiv(60))
```

```
Le nombre de diviseurs de 60 est 12
```

[24]:
```python
#Q2)
#écriture d'une fonction intermédiaire permettant de tester si un entier n est␣
 ↪riche ou pas
def est_riche(n):
    Max=nbdiv(n)
    for i in range(1,n):
        if nbdiv(i)>=Max:
            return False
    return True

def riches(K):
    L=[]
    total=0
    i=1
    while total<K:
        if est_riche(i):
            L.append([i,nbdiv(i)])
            total=total+1
        i=i+1
    return L

print("La liste des 16 premiers nombres riches avec leur nombre de diviseurs␣
 ↪est\n",riches(16))
```

```
La liste des 16 premiers nombres riches avec leur nombre de diviseurs est
 [[1, 1], [2, 2], [4, 3], [6, 4], [12, 6], [24, 8], [36, 9], [48, 10], [60, 12],
[120, 16], [180, 18], [240, 20], [360, 24], [720, 30], [840, 32], [1260, 36]]
```

[25]:
```
#Q3)
def facteurs_premiers(n):
    L=[]
    for i in range(2,n+1):
        while n%i==0 :
            n=n//i
            L.append(i)
    return L

print("La liste donnant la décomposition en facteurs irréductibles de 360␣
 ↪est",facteurs_premiers(360))
```

```
La liste donnant la décomposition en facteurs irréductibles de 360 est [2, 2, 2,
3, 3, 5]
```

[26]:
```
#Q4
def facteurs_premiers2(n):
    L=facteurs_premiers(n)
    S=set(L)
    result=[]
    for i in S:
        result.append([i,L.count(i)])
    return result

print("La liste donnant la décomposition en facteurs irréductibles de 360␣
 ↪est",facteurs_premiers2(360))
```

```
La liste donnant la décomposition en facteurs irréductibles de 360 est [[2, 3],
[3, 2], [5, 1]]
```

[27]:
```
#Q5)
print("La liste donnant la décomposition en facteurs irréductibles de 360␣
 ↪est",facteurs_premiers2(360))
print("Le nombre de diviseurs de 360 est",nbdiv(360))
```

```
La liste donnant la décomposition en facteurs irréductibles de 360 est [[2, 3],
[3, 2], [5, 1]]
Le nombre de diviseurs de 360 est 24
```

Soit $n = \prod_i p_i^{\alpha_i}$ la décomposition en produit de facteurs irréductibles de l'entier n. Alors le nombre de diviseurs de n est donné par $\prod_i(\alpha_i + 1)$. Par exemple, $360 = 2^3 \times 3^2 \times 5^1$ admet $(3+1) \times (2+1) \times (1+1) = 24$ diviseurs.

10 Exercice 10

[28]:
```
#Q1)
def retourner(n):
    return int(str(n)[::-1])

print("Le retourné du nombre 419 est",retourner(419))
```

```
Le retourné du nombre 419 est 914
```

[29]:
```
#Q2)
def est_palindrome(n):
    return n==retourner(n)

print(est_palindrome(200))
print(est_palindrome(1991))
```

```
False
True
```

[30]:
```
#Q3)
def palindromes(N):
    return [i for i in range(1,N+1) if est_palindrome(i)]

print("La liste des palindromes inférieurs ou égaux à 200␣
 ↪est\n",palindromes(200))
```

```
La liste des palindromes inférieurs ou égaux à 200 est
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111, 121,
131, 141, 151, 161, 171, 181, 191]
```

[31]:
```
#Q4)
def construit_palindrome(n):
    while not est_palindrome(n):
        n=n+retourner(n)
    return n

print("Le palindrome construit à partir de 39 est",construit_palindrome(39))
```

```
Le palindrome construit à partir de 39 est 363
```

[32]:
```
#Q5)
def construit_palindrome2(n):
    step=0
    while not est_palindrome(n):
        n=n+retourner(n)
        step=step+1
    return (n,step)
```

```
print("Le palindrome construit à partir de 39 et le nombre d'itérations␣
 ↪nécessaires sont donnés par",construit_palindrome2(39))
```

```
Le palindrome construit à partir de 39 et le nombre d'itérations nécessaires
sont donnés par (363, 2)
```

[33]:
```
#Q6)
def construit_palindrome3(n,K):
    step=0
    while not est_palindrome(n) and step<K:
        n=n+retourner(n)
        step=step+1
    return (n,step)

print(construit_palindrome3(39,10))
print(construit_palindrome3(196,10))
```

```
(363, 2)
(18211171, 10)
```

11 Exercice 11

Q1) La fonction chiffres(n) renvoie la liste des chiffres qui constitue l'entier n. On propose une fonction chiffres_bis(n) alternative qui est plus élégante que celle proposée par l'énoncé.

[34]:
```
def chiffres(n):
    if n==0:
        return []
    L=[]
    while n!=0:
        L.append(n%10)
        n=n//10
    return L[::-1]

print(chiffres(7246))

def chiffres_bis(n):
    return [int(i) for i in str(n)]
```

```
[7, 2, 4, 6]
```

[35]:
```
#Q2) et Q3)
def narcisse(n):
    return n==sum([i**len(str(n)) for i in chiffres(n)])

print("93084 est un nombre narcissique :",narcisse(93084))
```

```
93084 est un nombre narcissique : True
```

[36]:
```
#question 4)
for i in range(10001):
    if narcisse(i):
        print(i,"est un nombre narcissique:",narcisse(i))
```

```
0 est un nombre narcissique: True
1 est un nombre narcissique: True
2 est un nombre narcissique: True
3 est un nombre narcissique: True
4 est un nombre narcissique: True
5 est un nombre narcissique: True
6 est un nombre narcissique: True
7 est un nombre narcissique: True
8 est un nombre narcissique: True
9 est un nombre narcissique: True
153 est un nombre narcissique: True
370 est un nombre narcissique: True
371 est un nombre narcissique: True
407 est un nombre narcissique: True
1634 est un nombre narcissique: True
8208 est un nombre narcissique: True
9474 est un nombre narcissique: True
```

[37]:
```
#Q5)
def narcis_suiv(n,N):
    for i in range(n+1,N+1):
        if narcisse(i):
            return i
```

[38]:
```
#Q6)
n=10000
L=[i for i in range(1,n+1) if narcisse(i) and len([j for j in range(1,i+1) if␣
 ↪i%j==0])==2]
print("La liste des nombres narcissiques inférieurs ou égaux à 10000 et␣
 ↪premiers est",L)
```

```
La liste des nombres narcissiques inférieurs ou égaux à 10000 et premiers est
[2, 3, 5, 7]
```

12 Exercice 12

[39]:
```
#Q1)
def disjoints(L1,L2):
    if L1[1]<L2[0]:
        return True
    elif L1[1]>=L2[0] and L2[1]<L1[0]:
        return True
    return False

print(disjoints([1,2],[3,5]))
```

```
True
```

[40]:
```
#Q2)
def fusion(L1,L2):
    return [min([L1[0],L2[0]]),max([L1[1],L2[1]])]

print(fusion([1,2],[-1,5]))
```

```
[-1, 5]
```

Q3a) L_1 n'est pas bien formée car ses deux derniers éléments ne sont pas disjoints, L_2 n'est pas bien formée car ses premier et dernier éléments ne sont pas disjoints, L_3 est bien formée car ses éléments sont deux à deux disjoints et vérifient la condition de croissance voulue.

[41]:
```
#Q3b)
def verifie(L):
    for i in range(len(L)-1):
        if L[i][1]>L[i+1][0]:
            return False
    return True

print(verifie([[0,1],[2,5],[3,6]]))
print(verifie([[2,5],[0,1],[3,6]]))
print(verifie([[0,1],[2,3],[4,6]]))
```

```
False
False
True
```

[42]:
```
#Q4)
def appartient(x,L):
    for i in range(len(L)):
        if L[i][0]<=x<=L[i][1]:
            return True
    return False
```

```
print(appartient(-2,[[0,1],[2,3],[4,6]]))
print(appartient(7,[[0,1],[2,3],[4,6]]))
print(appartient(5.5,[[0,1],[2,3],[4,6]]))
```

```
False
False
True
```

13 Exercice 13

[43]:
```
#Q1)
def LDP(n):
    return [i for i in range(1,n) if n%i==0]


#Q2)
def SDP(n):
    return sum(LDP(n))


#Q3)
def parfaits(K):
    L=[]
    for i in range(1,K+1):
        if SDP(i)==i:
            L.append(i)
            print(i,"est parfait")
    return L


#Q4)
def amicaux(K):
    L=[]
    for i in range(1,K+1):
        for j in range(i+1,K+1):
            if i==SDP(j) and j==SDP(i):
                L.append((i,j))
                print("le couple",(i,j),"est amical")
    return L
```

14 Exercice 14

[44]:
```
#Q2)
def somme(n):
    return sum([int(i) for i in list(str(n))])
```

```python
#Q3)
def adequat(n):
    return somme(n)%10==0


#Q4)
def modification(n):
    if not adequat(n):
        unit=0
        for i in range(10):
            if (i+somme(n)-int(str(n)[-1]))%10==0:
                unit=i
                break
        return int(str(n)[:-1]+str(unit))
    return n


#Q5)
from random import randint

for i in range(10):
    p=randint(10000,100000)
    print(p,somme(p),modification(p),adequat(modification(p)))


#Q6)
n=10000
L=[1 for i in range(n) if adequat(randint(1,100001))]
print("La proportion de nombre adéquations est",sum(L)/n)
```

```
69689 38 69681 True
59768 35 59763 True
16836 24 16832 True
58210 16 58214 True
17030 11 17039 True
56024 17 56027 True
93085 25 93080 True
66540 21 66549 True
96345 27 96348 True
62785 28 62787 True
La proportion de nombre adéquations est 0.0988
```

[ ]: