```python
# C:\Users\Utilisateur\Documents\Boulot\Entrainement à l'oral\Oraux Math-Info\Donnés en 2023-2024\Oraux individuels\Exos sans préparation Python\Exercices non préparés tous.py
### Exercices non préparés (10 minutes)

## Exercice 1

import random as rd
def position(n,p):
    pos = 0
    for _ in range(n):
        pos += 2*(rd.random()<p)-1
    return pos
print(position(10,0.4))

def retour_0(n,p):
    k, pos = 1, position(1,p)
    while pos != 0 and k < n:
        pos += position(1,p)
        k += 1
    if k < n:
        return k
    else:
        return -1

print(retour_0(100,0.4))

## Exercice 2

def ldp(n):
    L = []
    for k in range(1,n//2+1):
        if n%k == 0:
            L.append(k)
    return L
print(ldp(100))

def sdp(n):
    return sum(ldp(n))
print(sdp(100))

def parfaits(N):
    L = []
    for n in range(1,N+1):
        if n == sdp(n):
            L.append(n)
    return L
print(parfaits(1000))

```

```
0047| def amicaux(N):
0048|     L = []
0049|     for p in range(2,N+1):
0050|         q = sdp(p)
0051|         if p < q <= N and p == sdp(q):
0052|             L.append((p,q))
0053|     return L
0054| print(amicaux(1000))
0055|
0056| import matplotlib.pyplot as plt
0057| def nuage(N):
0058|     L = amicaux(N)
0059|     Lp, Lq = [c[0] for c in L], [c[1] for c in L]
0060|     plt.plot(Lp,Lq,"o")
0061|     plt.show()
0062| nuage(10000)
0063|
0064| ## Exercice 3
0065|
0066| def u(n,a):
0067|     u = a
0068|     for k in range(n):
0069|         u = u**2/(k+1)
0070|     return u
0071|
0072| import matplotlib.pyplot as plt
0073| Ln = range(3,10)
0074| for k in [1,9,12,16,17,18]:
0075|     plt.figure(k)
0076|     Lu = [u(n,k/10) for n in Ln]
0077|     plt.plot(Ln,Lu,marker='.')
0078|     plt.show()
0079|
0080| plt.figure(1)
0081| N, a = 31, 1.6616
0082| Ln = range(N)
0083| Lu = [u(n,a) for n in Ln]
0084| plt.plot(Ln,Lu,marker='.')
0085| plt.show()
0086|
0087| plt.figure(2)
0088| N, a = 18, 1.6617
0089| Ln = range(N)
0090| Lu = [u(n,a) for n in Ln]
0091| plt.plot(Ln,Lu,marker='.')
0092| plt.show()
0093|
0094| # pour a=1.6616 la suite semble converger vers 0
0095| # pour a=1.6617 la suite semble diverger vers +infini
0096|
```

```
0097| ## Exercice 4
0098|
0099| import numpy as np
0100| def f(p,x):
0101|     n = 100
0102|     return sum([np.sqrt(x+(k/n)**p) for k in range(n)])/n
0103|
0104| def suitef1(p):
0105|     return [f(k,1) for k in range(p+1)]
0106|
0107| import matplotlib.pyplot as plt
0108| p = 30
0109| plt.plot(range(p+1),suitef1(p),marker='.')
0110| plt.show()
0111|
0112| # La suite semble converger vers 1
0113| ##
0114| Lx = np.linspace(0,20)
0115| for p in range(1,7):
0116|     Ly = f(p,Lx)
0117|     plt.plot(Lx,Ly)
0118|     plt.show()
0119|
0120| # On remarque que les fonctions fp sont toutes
croissantes et continues
0121| # pour un x fixé, la suite (fp(1)) est décroissante
0122|
0123| p = 100
0124| for x in [1,4,9,16,25,36]:
0125|     print(f(p,x))
0126|
0127| # il semble que quand p tend vers +infini, fp(x) =
sqrt(x)
0128|
0129| ## Exercice 5
0130|
0131| import random as rd
0132| E = [(i,rd.randint(1,5)) for i in range(32)]
0133| print(E)
0134|
0135| def match(i,j):
0136|     ni, nj = E[i][1], E[j][1]
0137|     if ni == nj:
0138|         return (i,j)
0139|     else:
0140|         p = ni/(2*(ni+nj))
0141|         if rd.random() < p:
0142|             return i
0143|         else:
0144|             return j
```

```
0145|
0146| G = []
0147| CopyE = [couple for couple in E]
0148| for _ in range(8):
0149|     Poule = []
0150|     for _ in range(4):
0151|         equipe = rd.choice(CopyE)
0152|         Poule.append(equipe)
0153|         CopyE.remove(equipe)
0154|     G.append(Poule)
0155| print(G)
0156|
0157| def Poule(k):
0158|     Score = []
0159|     poule = G[k]
0160|     score = [0]*4
0161|     for a in range(3):
0162|         for b in range(a+1,4):
0163|             i , j = poule[a][0], poule[b][0]
0164|             if match(i,j) == (i,j):
0165|                 score[a] += 1
0166|                 score[b] += 1
0167|             elif match(i,j) == i:
0168|                 score[a] += 3
0169|             else:
0170|                 score[b] += 3
0171|     return [(poule[k][0],score[k]) for k in range(4)]
0172|
0173| def affiche_scores():
0174|     for k in range(4):
0175|         print('le résultat de la poule',k,'est',Poule(k))
0176|
0177| ## Exercice 6
0178|
0179| import random as rd
0180| import numpy as np
0181|
0182| def grille():
0183|     G = np.zeros([3,3],int)
0184|     L = list(range(1,10))
0185|     for i in range(3):
0186|         for j in range(3):
0187|             G[i,j] = rd.choice(L)
0188|             L.remove(G[i,j])
0189|     return G
0190| print(grille())
0191|
0192| def estmagique(G):
0193|     s = sum(G[0])
0194|     for i in range(1,3):
```

```
0195|         if sum(G[i]) != s:
0196|             return False
0197|     for j in range(3):
0198|         if sum(G[:,j]) != s:
0199|             return False
0200|     if sum([G[i,i] for i in range(3)]) != s:
0201|         return False
0202|     if sum([G[i,2-i] for i in range(3)]) != s:
0203|         return False
0204|     return True
0205|
0206| for _ in range(10):
0207|     G = grille()
0208|     print(G,estmagique(G))
0209|
0210| def carremagique():
0211|     for a in range(1,10):
0212|         L1 = [b for b in range(1,10) if b != a]
0213|         for b in L1:
0214|             L2 = [c for c in L1 if c != b]
0215|             for c in L2:
0216|                 L3 = [d for d in L2 if d != c]
0217|                 for d in L3:
0218|                     L4 = [e for e in L3 if e != d]
0219|                     for e in L4:
0220|                         L5 = [f for f in L4 if f != e]
0221|                         for f in L5:
0222|                             s = a+b+c
0223|                             g, h, i = s-a-d, s-b-e, s-c-f
0224|                             grille = np.array([[a,b,c],
[d,e,f],[g,h,i]])
0225|                             if estmagique(grille):
0226|                                 return grille
0227| print(carremagique())
0228|
0229| ## Exercice 7
0230|
0231| jeu = [(c,v) for c in ['Pi','Co','Tr','Ca'] for v in
range(1,14)]
0232|
0233| def testAs(carte):
0234|     return carte[1] == 1
0235| print(testAs(rd.choice(jeu)))
0236|
0237| import random as rd
0238| def main():
0239|     reste_jeu = list(jeu)
0240|     m = []
0241|     for _ in range(5):
0242|         carte =
```

```
reste_jeu.pop(rd.randint(0,len(reste_jeu)-1))
0243|             m.append(carte)
0244|         return m
0245| print(main())
0246|
0247| def couleur(m):
0248|     c = m[0][0]
0249|     for k in range(1,5):
0250|         if m[k][0] != c:
0251|             return False
0252|     return True
0253| print(couleur(main()))
0254|
0255| def carre(m):
0256|     for i in range(2):
0257|         val = m[i][1]
0258|         n = 1
0259|         for j in range(i+1,5):
0260|             if m[j][1] == val:
0261|                 n += 1
0262|         if n == 4:
0263|             return True
0264|     return False
0265| print(carre(main()))
0266|
0267| def full(m):
0268|     val = range(13)
0269|     eff_val = [0]*13
0270|     for carte in m:
0271|         for k in range(13):
0272|             eff_val[k] += (carte[1] == val[k])
0273|     return 2 and 3 in eff_val
0274| print(full(main()))
0275|
0276| ## Exercice 8
0277|
0278| nucleotides = ['A','T','G','C']
0279| liste_codons = []
0280| for a in nucleotides:
0281|     for b in nucleotides:
0282|         for c in nucleotides:
0283|             liste_codons.append(a+b+c)
0284| print(liste_codons)
0285|
0286| import random as rd
0287| def sequence(n):
0288|     seq = ''
0289|     for _ in range(n):
0290|         seq += rd.choice(liste_codons)
0291|     return seq
```

```
0292| print(sequence(5))
0293|
0294| def brin(s):
0295|     codon_stop = ['TAG','TAA','TGA']
0296|     n = len(s)
0297|     for i in range(n-2):
0298|         if s[i:i+3] in codon_stop:
0299|             return s[:i+3]
0300|     return -1
0301| print(brin(sequence(10)))
0302|
0303| def complementaire(b1,b2):
0304|     n1, n2 = len(b1), len(b2)
0305|     if n1 != n2:
0306|         return False
0307|     else:
0308|         for k in range(n1):
0309|             if (b1[k],b2[k]) not in [('A','T'),('T','A'),
('C','G'),('G','C')]:
0310|                 return False
0311|     return True
0312| print(complementaire(sequence(10),sequence(10)))
0313|
0314| def crossover(b1,b2,k):
0315|     n1, n2 = len(b1), len(b2)
0316|     if k > min(n1,n2):
0317|         print('pas de croisement possible')
0318|         return b1,b2
0319|     else:
0320|         r1, r2 = b1[k:], b2[k:]
0321|         return b1[:k]+r2, b2[:k]+r1
0322| b1, b2 = sequence(5), sequence(5)
0323| print(b1,b2)
0324| print(crossover(b1,b2,9))
0325|
0326| ## Exercice 9
0327|
0328| def modalites_eff(X):
0329|     m = max(X)
0330|     eff = [0]*(m+1)
0331|     for x in X:
0332|         eff[x] += 1
0333|     modalites = [x for x in range(m+1) if eff[x] !=0]
0334|     effectif = [n for n in eff if n != 0]
0335|     return modalites,effectif
0336| import random as rd
0337| X = [rd.randint(1,10) for _ in range(100)]
0338| print(modalites_eff(X))
0339|
0340| def mode(X):
```

```
0341|     modalite, effectif = modalites_eff(X)
0342|     m = max(effectif)
0343|     return [modalite[k] for k in range(len(modalite)) if
effectif[k] == m]
0344| print(mode(X))
0345|
0346| def moyenne(X):
0347|     return sum(X)/len(X)
0348| print(moyenne(X))
0349|
0350| def mediane(X):
0351|     X.sort()
0352|     m = len(X)//2
0353|     if m % 2 == 1:
0354|         return X[m]
0355|     else:
0356|         return (X[m-1]+X[m])/2
0357| print(mediane(X))
0358|
0359| ## Exercice 10
0360| import random as rd
0361| def  liste_Musee(n):
0362|     L = []
0363|     for _ in range(n):
0364|         M = []
0365|         for _ in range(rd.randint(1,100)):
0366|             oeuvre =
(rd.choice(['P','R']),rd.randint(1,7),rd.randint(1,3))
0367|             M.append(oeuvre)
0368|         L.append(M)
0369|     return L
0370| L = liste_Musee(10)
0371| print(L)
0372|
0373| def reserve(M):
0374|     R = [o for o in M if o[0] == 'R']
0375|     return len(R)
0376| M = L[0]
0377| print(reserve(M))
0378|
0379| def eff_salle(M):
0380|     L = [0]*7
0381|     for o in M:
0382|         L[o[1]-1] += 1
0383|     return L
0384| print(eff_salle(M))
0385|
0386| def classe(L):
0387|     newL = list(L)
0388|     n = len(L)
```

```
0389|     N = [len(M) for M in L]
0390|     Ltriee = []
0391|     while len(N) > 0:
0392|         i = N.index(min(N))
0393|         Ltriee.append(newL[i])
0394|         N.pop(i)
0395|         newL.pop(i)
0396|     return Ltriee
0397| print(classe(L))
0398|
0399| ## Exercice 11
0400|
0401| def crible(L,k):
0402|     return [n for n in L if not n%k == 0]
0403| L = range(1,100)
0404| print(crible(L,2))
0405|
0406| def Eratosthene(n):
0407|     L = list(range(2,n+1))
0408|     prem = 2
0409|     Lprem = [1,2]
0410|     while len(L) > 1 and prem <= n:
0411|         L = crible(L,prem)
0412|         prem = L[0]
0413|         Lprem.append(prem)
0414|     return Lprem
0415| print(Eratosthene(100))
0416|
0417| def test_premier(n):
0418|     return n in Eratosthene(n)
0419| print(test_premier(11))
0420| print(test_premier(10))
0421|
0422| ## Exercice 12
0423|
0424| import random as rd
0425| def Monte_Carlo(N):
0426|     c = 0
0427|     for _ in range(N):
0428|         X, Y = rd.random(),rd.random()
0429|         c += (X**2+Y**2 <= 1)
0430|     return c/N
0431|
0432| # la probabilité de l'événement [X²+Y²<=1] est égale à
l'aire du quart de disque X>=0,Y>=0, X²+Y²<=1 donc égale à pi/4
0433|
0434| print(4*Monte_Carlo(1000000))
0435|
0436| ## Exercice 13
0437|
```

```
0438| import numpy as np
0439| def F(l,k):
0440|     u = np.exp(-l)
0441|     S = u
0442|     for n in range(1,k+1):
0443|         u = l*u/n
0444|         S += u
0445|     return S
0446| print(F(2,1))
0447|
0448| def G(l,x):
0449|     k = 0
0450|     while F(l,k) <= x:
0451|         k += 1
0452|     return k
0453|
0454| print(G(2,0.7))
0455| # G simule la fonction des quantiles de la loi
exponentielle de paramètre l
0456|
0457| ## Exercice 14
0458|
0459| import random as rd
0460| def plateau(n):
0461|     return [rd.randint(0,n-1) for _ in range(n)]
0462| plat = plateau(5)
0463| print(plat)
0464|
0465| def compter(plat):
0466|     c = 0
0467|     for k in range(len(plat)):
0468|         c += plat[k] == k
0469|     return c
0470| print(compter(plat))
0471|
0472| def avancer(plat,p):
0473|     i = 0
0474|     for _ in range(p):
0475|         i = plat[i]
0476|     return i
0477| plat = [2,4,1,3,0]
0478| print(avancer(plat,1))
0479| print(avancer(plat,3))
0480|
0481| ## Exercice 15
0482|
0483| import random as rd
0484| def grille(n):
0485|     L = []
0486|     for _ in range(n):
```

```
0487|         L.append([rd.choice([0,1,None]) for _ in
range(n)])
0488|     return L
0489| G = grille(5)
0490| print(G)
0491|
0492| def case_vide(G):
0493|     c = 0
0494|     for i in range(len(G)):
0495|         for j in range(len(G[i])):
0496|             c += (G[i][j] == None)
0497|     return c
0498| print(case_vide(G))
0499|
0500| def lignes_identiques(G):
0501|     for i in range(len(G)):
0502|         for j in range(i+1,len(G)):
0503|             if G[i] == G[j]:
0504|                 return True
0505|     return False
0506| print(lignes_identiques(G))
0507| G1 = [[0,1,None,0],[1,1,0,None],[0,0,1,1],[1,1,0,None]]
0508| print(lignes_identiques(G1))
0509|
0510| ## Exercice 16
0511|
0512| def saut(p,k):
0513|     if k == 0 or k == 3:
0514|         return k
0515|     elif k == 1:
0516|         if rd.random() < p:
0517|             return 0
0518|         else:
0519|             return 2
0520|     else:
0521|         if rd.random() < p:
0522|             return 1
0523|         else:
0524|              return 3
0525| print(saut(0.1,3))
0526|
0527| def position(p,n):
0528|     Lpos = [1]
0529|     for _ in range(n-1):
0530|         Lpos.append(saut(p,Lpos[-1]))
0531|     return Lpos
0532| print(position(0.4,5))
0533|
0534| ## Exercice 17
0535|
```

```
0536| def strict_croissance(L):
0537|     for i in range(len(L)-1):
0538|         if L[i] >= L[i+1]:
0539|             return False
0540|     return True
0541|
0542| L = [1,2,3]
0543| print(strict_croissance(L))
0544| L = [1,2,2,3]
0545| print(strict_croissance(L))
0546| L = [1,2,4,3]
0547| print(strict_croissance(L))
0548|
0549| def strict_monotone(L):
0550|     Lr = L[::-1]
0551|     return strict_croissance(L) or strict_croissance(Lr)
0552|
0553| L = [3,2,1]
0554| print(strict_monotone(L))
0555| L = [3,2,2,1]
0556| print(strict_monotone(L))
0557| L = [1,2,4,3]
0558| print(strict_monotone(L))
0559|
0560| ## Exercice 18
0561|
0562| def nbvrai(L):
0563|     return sum(L)
0564| print(nbvrai([False,True,True]))
0565|
0566| def blocvrai(L):
0567|     n = nbvrai(L)
0568|     if n == 0:
0569|         return False
0570|     i = L.index(True)
0571|     j, c = i+1, 1
0572|     while j < len(L) and L[j] == True:
0573|         j += 1
0574|         c += 1
0575|     return c == n
0576| L = [False,True,True,True,False,False]
0577| print(blocvrai(L))
0578| L =[False,True,True,True,False,False,True]
0579| print(blocvrai(L))
0580|
0581| ## Exercice 19
0582|
0583| alphabet = 'abcdefghijklmnopqrstuvw'
0584|
0585| def Cesar(C,n):
```

```
0586|     Ccode = ''
0587|     for lettre in C:
0588|         i = alphabet.index(lettre)
0589|         Ccode += alphabet[(i+n)%26]
0590|     return Ccode
0591| C = 'mathematiques'
0592| print(Cesar(C,10))
0593|
0594| def pyramide(C,n):
0595|     ind, nbre = 0, 1
0596|     for _ in range(n):
0597|         if ind + nbre > len(C):
0598|             C += C
0599|         print(C[ind:ind+nbre])
0600|         ind += nbre
0601|         nbre += 1
0602| pyramide(C,12)
0603|
0604| ## Exercice 20
0605|
0606| def budget(L):
0607|     recettes = sum([x for x in L if x > 0])
0608|     depenses = sum([x for x in L if x < 0])
0609|     return recettes, depenses
0610| L = [156.34,-35.78,-26.80,25.90,-132.23,56,25]
0611| print(budget(L))
0612|
0613| import math as m
0614| def decompose(S):
0615|     E = [100,50,20,10,5,2,1]
0616|     C = [50,20,10,5,2,1]
0617|     euro = m.floor(S)
0618|     for v in E:
0619|         if euro//v > 0:
0620|             print(v,' euros :',euro//v)
0621|         euro = euro%v
0622|     cent = int(100*(S - m.floor(S)))
0623|     for v in C:
0624|         if cent//v > 0:
0625|             print(v/100,' euros :',cent//v)
0626|         cent = cent%v
0627| S = 208.74
0628| print(decompose(S))
0629|
0630| ## Exercice 21
0631|
0632| def retire(L,i):
0633|     ote = [L[k] for k in range(i-1,len(L),i)]
0634|     return [x for x in L if x not in ote]
0635| L = [1,3,5,7,9,11,13,15]
```

```
0636| print(retire(L,3))
0637|
0638| def FJ(n):
0639|     L = range(1,n+1)
0640|     for i in range(2,n):
0641|         L = retire(L,i)
0642|     return L
0643| print(FJ(100))
0644|
0645| def U(n):
0646|     return len(FJ(n))
0647|
0648| import matplotlib.pyplot as plt
0649| Ln = range(1,1000)
0650| Lu = [4*n/U(n)**2 for n in Ln]
0651| plt.plot(Ln,Lu,".")
0652| plt.show()
0653|
0654| # la suite semble converger, on pourrait montrer que
c'est vers pi
0655| # un équivalent de U(n) est 2*sqrt(pi)
0656|
0657| ## Exercice 22
0658|
0659| def code_creux(U):
0660|     C, I = [], []
0661|     for k in range(len(U)):
0662|         if U[k] != 0.:
0663|             C.append(U[k])
0664|             I.append(k)
0665|     return [C,I]
0666| U = [1,2,0,5,8,0,0,0,7]
0667| print(code_creux(U))
0668|
0669| def decode_creux(L):
0670|     C, I = L[0], L[1]
0671|     U = [0]*(I[-1]+1)
0672|     for k in range(len(L[0])):
0673|         U[I[k]] = C[k]
0674|     return U
0675| L = [[1, 2, 5, 8, 7], [0, 1, 3, 4, 8]]
0676| print(decode_creux(L))
0677|
0678| ## Exercice 23
0679|
0680| def subdivision(a,b,n):
0681|     h = (b-a)/n
0682|     return [round(a+k*h,2) for k in range(n+1)]
0683| print(subdivision(0,1,100))
0684|
```

```
0685| # on aurait aussi pu écrire
0686| # def subdivision(a,b,n):
0687| #     return np.linspace(a,b,n+1)
0688|
0689| def Lagrange(sub,k):
0690|     def Lk(x):
0691|         P = 1
0692|         for i in range(len(sub)):
0693|             if i != k:
0694|                 P *= (x-sub[i])/(sub[k]-sub[i])
0695|         return P
0696|     return Lk
0697|
0698| def interpol(f,a,b,n):
0699|     Sub = subdivision(a,b,n)
0700|     def L(x):
0701|         return sum([f(Sub[k])*Lagrange(Sub,k)(x) for k in
range(n+1)])
0702|     return L
0703|
0704| import matplotlib.pyplot as plt
0705| import numpy as np
0706| def affiche(f,a,b,n):
0707|     Lx = np.linspace(a,b,int(100*(b-a)))
0708|     Lf = f(Lx)
0709|     plt.plot(Lx,Lf,color = 'red')
0710|     L = interpol(f,a,b,n)(Lx)
0711|     plt.plot(Lx,L,color='black')
0712|     plt.show()
0713|
0714| def f(x):
0715|     return np.sin(x)
0716|
0717| for n in [2,3,5,8,10]:
0718|     plt.figure(n)
0719|     affiche(f,-2*np.pi,2*np.pi,n)
0720|
0721| ## Exercice 24
0722|
0723| import numpy as np
0724| import scipy.integrate as itg
0725| import matplotlib.pyplot as plt
0726|
0727| def h(t):
0728|     return np.sin(t)/t**2
0729|
0730| def f(x):
0731|     return itg.quad(h,x,100)[0]
0732|
0733| for n in range(1,7):
```

```
0734|     plt.figure(n)
0735|     Lx = np.linspace(n*np.pi,(n+1)*np.pi)
0736|     Ly = [f(x) for x in Lx]
0737|     plt.plot(Lx,Ly)
0738|     plt.show()
0739|
0740| def approx(n):
0741|     a, b = n*np.pi, (n+1)*np.pi
0742|     while b-a > 1e-3:
0743|         c = (a+b)/2
0744|         if f(a)*f(c) > 0:
0745|             a = c
0746|         else:
0747|             b = c
0748|     return round(c,3)
0749| print(approx(1))
0750|
0751| ## Exercice 25
0752|
0753| def P(n,x):
0754|     if n == 0:
0755|         return 1
0756|     elif n == 1:
0757|         return 2*x
0758|     else:
0759|         return 2*x*P(n-1,x)-P(n-2,x)
0760|
0761| import numpy as np
0762| import scipy.integrate as itg
0763| def prod_scal(i,j):
0764|     def h(t):
0765|         return 2*np.sqrt(1-t**2)*P(i,t)*P(j,t)/np.pi
0766|     return itg.quad(h,-1,1)[0]
0767|
0768| # def prod(i,j):
0769| #     def h(t):
0770| #         return 2*np.sqrt(1-t**2)*P(i,t)*P(j,t)/np.pi
0771| #     n = 1000
0772| #     return sum([h(-1+2*k/n) for k in range(n)])*2/n
0773|
0774| for i in range(9):
0775|     for j in range(9):
0776|         print(round(prod_scal(i,j),3))
0777|
0778| # on remarque prod(i,j) = 1 si i = j et 0 sinon
0779|
0780| ## Exercice 26
0781|
0782| import random as rd
0783| def deplacement(r,c):
```

```
0784|     if c == (-r,-r):
0785|         return rd.choice([(-r+1,-r),(-r,-r+1)])
0786|     elif c == (-r,r):
0787|         return rd.choice([(-r+1,r),(-r,r-1)])
0788|     elif c == (r,-r):
0789|         return rd.choice([(r-1,-r),(r,-r+1)])
0790|     elif c == (r,r):
0791|         return rd.choice([(r-1,r),(r,r-1)])
0792|     elif c[0] == -r:
0793|         return rd.choice([(-r,c[1]-1),(-r,c[1]+1),(-
r+1,c[1])])
0794|     elif c[0] == r:
0795|         return rd.choice([(r,c[1]-1),(r,c[1]+1),
(r-1,c[1])])
0796|     elif c[1] == -r:
0797|         return rd.choice([(c[0]-1,-r),(c[0]+1,-r),(c[0],-
r+1)])
0798|     elif c[1] == r:
0799|         return rd.choice([(c[0]-1,r),(c[0]+1,r),
(c[0],r-1)])
0800|     else:
0801|         return rd.choice([(c[0]-1,c[1]),(c[0]+1,c[1]),
(c[0],c[1]-1),(c[0],c[1]+1)])
0802| print(deplacement(5,(0,0)))
0803|
0804| def position(r,n):
0805|     pos = (0,0)
0806|     for _ in range(n):
0807|         pos = deplacement(r,pos)
0808|     return pos
0809| print(position(5,10))
0810|
0811| def retour_0(r,n):
0812|     k, pos = 1, deplacement(r,(0,0))
0813|     while pos != (0,0) and k < n:
0814|         pos = deplacement(r,pos)
0815|         k += 1
0816|     if k < n:
0817|         return k
0818|     else:
0819|         return -1
0820| print(retour_0(5,100))
0821|
0822| ## Exercice 27
0823| # import matplotlib.pyplot as plt
0824| # import numpy as np
0825| # Lx = np.linspace(-1,1,500)
0826| # Ly = [np.sqrt(1-x**2) for x in Lx]
0827| # plt.plot(Lx,Ly,color='black',linewidth=3)
0828| # plt.plot([-1,-1/2,1/2,1],
```

```
[0,np.sqrt(3)/2,np.sqrt(3)/2,0],color='black',linewidth=3)
0829| # plt.plot([-1,-1,0,1,1],
[0,1/np.sqrt(3),2/np.sqrt(3),1/np.sqrt(3),0],color='black',line
width=3)
0830| # plt.show()
0831|
0832| def u(n):
0833|     if n == 6:
0834|         return 1/2
0835|     else:
0836|         return np.sqrt((1-np.sqrt(1-(u(n//2))**2))/2)
0837|
0838| def archimede(k):
0839|     n = 6*2**k
0840|     return n*u(n)
0841| print(archimede(10))
0842|
0843| ## Exercice 28
0844|
0845| def Gregory(n,x):
0846|     return sum([((-1)**k*x**(2*k+1))/(2*k+1) for k in
range(n+1)])
0847| print(4*Gregory(1000000,1))
0848|
0849| ## Exercice 29
0850| import random as rd
0851| import matplotlib.pyplot as plt
0852| def Spierpinsky(n):
0853|     x, y = [180], [150]
0854|     a1, a2 = 20, 20
0855|     b1, b2 = 320, 20
0856|     c1, c2 = 170, 280
0857|     for _ in range(n):
0858|         p = rd.random()
0859|         if p < 1/3:
0860|             x.append((a1+x[-1])/2)
0861|             y.append((a2+y[-1])/2)
0862|         elif p < 2/3:
0863|             x.append((b1+x[-1])/2)
0864|             y.append((b2+y[-1])/2)
0865|         else:
0866|             x.append((c1+x[-1])/2)
0867|             y.append((c2+y[-1])/2)
0868|     plt.plot(x,y,'.',color='blue')
0869|     plt.show()
0870| n = 30000
0871| Spierpinsky(n)
0872|
0873| ## Exercice 30
0874| import random as rd
```

```
0875| import matplotlib.pyplot as plt
0876|
0877| def Barnsley(n):
0878|     x, y = [0.5], [0]
0879|     for _ in range(n):
0880|         p = rd.random()
0881|         if p < 0.02:
0882|             x.append(0.5)
0883|             y.append(0.27*y[-1])
0884|         elif p < 0.17:
0885|             x.append(-0.139*x[-1]+0.263*y[-1]+0.57)
0886|             y.append(0.246*x[-1]+0.224*y[-1]-0.036)
0887|         elif p < 0.3:
0888|             x.append(0.17*x[-1]-0.215*y[-1]+0.408)
0889|             y.append(0.222*x[-1]+0.176*y[-1]+0.0893)
0890|         else:
0891|             x.append(0.781*x[-1]+0.034*y[-1]+0.1075)
0892|             y.append(-0.032*x[-1]+0.739*y[-1]+0.27)
0893|     plt.plot(x,y,'.',color='green')
0894|     plt.show()
0895| n = 30000
0896| Barnsley(n)
0897|
0898| ## Exercice 31
0899|
0900| def Mandelbrot(n):
0901|     x, y = [0.5], [0]
0902|     for _ in range(n):
0903|         p = rd.random()
0904|         if p < 0.1:
0905|             x.append(0.05*x[-1])
0906|             y.append(0.6*y[-1])
0907|         elif p < 0.2:
0908|             x.append(0.05*x[-1])
0909|             y.append(1-0.5*y[-1])
0910|         elif p < 0.4:
0911|             x.append(0.46*x[-1]-0.32*y[-1])
0912|             y.append(0.39*x[-1]+0.38*y[-1]+0.6)
0913|         elif p < 0.6:
0914|             x.append(0.47*x[-1]-0.15*y[-1])
0915|             y.append(0.17*x[-1]+0.42*y[-1]+1.1)
0916|         elif p < 0.8:
0917|             x.append(0.43*x[-1]+0.28*y[-1])
0918|             y.append(-0.25*x[-1]+0.45*y[-1]+1)
0919|         else:
0920|             x.append(0.42*x[-1]+0.26*y[-1])
0921|             y.append(-0.35*x[-1]+0.31*y[-1]+0.7)
0922|     plt.plot(x,y,'.',color='brown')
0923|     plt.show()
0924|
```

```
0925| n = 30000
0926| Mandelbrot(n)
0927|
0928| ## Exercice 32
0929| def compresse(L):
0930|     i = 0
0931|     C = [L[0]]
0932|     while i < len(L):
0933|         c = 0
0934|         j = i
0935|         while j < len(L) and L[j] == L[i]:
0936|             c += 1
0937|             j += 1
0938|         C.append(c)
0939|         i = j
0940|     return C
0941|
0942| L = [0,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0]
0943| print(compresse(L))
0944| #
0945| def decompresse(L):
0946|     D = []
0947|     i = 1
0948|     while i < len(L):
0949|         if i%2 == 1:
0950|             D += (L[i]*[L[0]])
0951|         else:
0952|             D += (L[i]*([1-L[0]]))
0953|         i += 1
0954|     return D
0955|
0956| L = [0, 1, 1, 2, 3, 1, 2, 3, 2, 1]
0957| print(decompresse(L))
0958|
0959| #
0960| def remplace(L):
0961|     C = compresse(L)
0962|     if C[0] == 0:
0963|         m = max(C[2::2])
0964|     else:
0965|         m = max(C[1::2])
0966|     ind = sum(C[:C.index(m)])
0967|     L[ind:ind+m]=[2]*m
0968|     return L
0969|
0970| L = [0,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0]
0971| print(L)
0972| print(remplace(L))
0973|
0974| ## Exercice 33
```

```
0975| def nbre_zeros(L,i):
0976|     if L[i] != 0:
0977|         return 0
0978|     else:
0979|         k = i
0980|         c = 0
0981|         while k < len(L) and L[k] == 0:
0982|             c += 1
0983|             k += 1
0984|         return c
0985|
0986| L = [0,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0]
0987| print(nbre_zeros(L,5))
0988|
0989| def nbre_zeros_Max(L):
0990|     return max([nbre_zeros(L,i) for i in range(len(L))])
0991|
0992| L = [0,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0,0,0,0]
0993| print(nbre_zeros_Max(L))
0994|
0995| ## Exercice 34
0996| import math as m
0997| def carres(n):
0998|     return [k**2 for k in range(1,int(m.sqrt(n))+1)]
0999|
1000| print(carres(10))
1001|
1002| def Som_Carre(n):
1003|     L_carres = carres(n//2)
1004|     S_carres = [i+j for i in L_carres for j in L_carres]
1005|     return [k for k in range(n) if k in S_carres]
1006|
1007| print(Som_Carre(100))
1008|
1009| ## Exercice 35
1010| def disjoints(I1,I2):
1011|     return (I1[1] < I2[0]) or (I2[1] < I1[0])
1012|
1013| I1, I2 = [1,3], [4,5]
1014| print(disjoints(I1,I2))
1015| I1, I2 = [1,4], [3,5]
1016| print(disjoints(I1,I2))
1017|
1018| def bien_formee(L):
1019|     for i in range(len(L)-1):
1020|         if L[i][1] >= L[i+1][0]:
1021|             return False
1022|     return True
1023|
1024| L = [[1,3], [4,5],[6,7]]
```

```
print(bien_formee(L))
L = [[1,3], [6,7],[4,5]]
print(bien_formee(L))
L = [[1,3], [3,5],[6,7]]
print(bien_formee(L))

def appartient(x,L):
    for l in L:
        if l[0] <= x <= l[1]:
            return True
    else:
        return False

x, L = 2,[[1,3], [5,6],[7,8]]
print(appartient(x,L))
x, L = 4,[[1,3], [5,6],[7,8]]
print(appartient(x,L))
x, L = 9,[[1,3], [5,6],[7,8]]
print(appartient(x,L))

## Exercice 36
def Que_des_1(n):
    m = 1
    while m % n != 0:
        m = 10*m+1
    return m

print(Que_des_1(7))

def Nbre_de_1(n):
    return len(list(str(Que_des_1(n))))

print(Nbre_de_1(37))

import matplotlib.pyplot as plt

n = 100
Ln = [2*k+1 for k in range(n) if int((2*k+1)%5) != 0]
print(Ln)
L = [Nbre_de_1(x) for x in Ln]
print(L)
plt.plot(Ln,L,marker='o')
plt.show()

## Exercice 37
def u(n,N):
    u = N
    for _ in range(n):
        if u%2 == 0:
            u = int(u/2)
```

```
1075|         else:
1076|             u = 3*u+1
1077|     return u
1078| print(u(100,5))
1079|
1080| def duree(N):
1081|     D = [N]
1082|     n = 1
1083|     while u(n,N) != 1:
1084|         D.append(u(n,N))
1085|         n += 1
1086|     return D + [1]
1087|
1088| print(duree(5))
1089|
1090| import matplotlib.pyplot as plt
1091|
1092| LN = range(1,101)
1093| Ld = [len(duree(N))-1 for N in LN]
1094| plt.plot(LN,Ld,marker='o')
1095| plt.show()
1096|
1097| ## Exercice 38
1098| import matplotlib.pyplot as plt
1099|
1100| def S(n):
1101|     return sum([(-1)**(k+1)/k for k in range(1,n+1)])
1102|
1103| Ln = range(1,50)
1104| Spair = [S(2*n) for n in Ln]
1105| SImpair = [S(2*n+1) for n in Ln]
1106| plt.plot(Ln,Spair,marker='.')
1107| plt.plot(Ln,SImpair,marker='x')
1108| plt.show()
1109|
1110| # on conjecture que les suites extraites sont adjacentes
et donc que (Sn) est convergentes.
1111|
1112| #De plus, pour tout n>0, S_(2n+1)<L<S_2n
1113|
1114| def val_ap():
1115|     n = 0
1116|     while S(2*n+1)-S(2*n) > 10**(-3):
1117|         n += 1
1118|     return (S(2*n)+S(2*n+1))/2
1119| print(val_ap())
1120|
1121| ## Exercice 39
1122| def un_coup(c):
1123|     L = []
```

```
1124|     for i in [-1,1]:
1125|         for j in [-2,2]:
1126|             if 0 <= c[0]+i <= 7 and 0 <= c[1]+j <= 7:
1127|                 L.append((c[0]+i,c[1]+j))
1128|             if 0 <= c[0]+j <= 7 and 0 <= c[1]+i <= 7:
1129|                 L.append((c[0]+j,c[1]+i))
1130|     return L
1131|
1132| c = (4,4)
1133| print(un_coup(c))
1134| c = (0,2)
1135| print(un_coup(c))
1136| c = (0,0)
1137| print(un_coup(c))
1138|
1139| ## Exercice 40
1140| def suivante(L):
1141|     return [(L[i]+L[i+1])%2 for i in range(len(L)-1)]
1142|
1143| L = [1,1,0,0,1,0,1]
1144| print(suivante(L))
1145|
1146| def triangle(L):
1147|     T = [L]
1148|     while len(L) > 0:
1149|         print(L)
1150|         L = suivante(L)
1151|         T.append(L)
1152|     return T
1153|
1154| L = [1,1,0,0,1,0,1]
1155| triangle(L)
1156|
1157| def equilibre(T):
1158|     nbre1, nbre0 = 0, 0
1159|     for l in T:
1160|         nbre1 += sum(l)
1161|         nbre0 += (len(l) - sum(l))
1162|     return nbre1 == nbre0
1163|
1164| L = [1,0,0,0,1,0,1,0]
1165| T = triangle(L)
1166| print(equilibre(T))
1167| L = [1,0,0,0,1,0,1]
1168| T = triangle(L)
1169| print(equilibre(T))
1170|
1171| ## Exercice 41
1172| def couple(n):
1173|     return [(a,b) for a in range(1,n+1) for b in
```

```
range(a,n+1)]
1174|
1175| print(couple(5))
1176|
1177| def pythagoricien(n):
1178|     L = []
1179|     for x in couple(n):
1180|         a, b = x[0], x[1]
1181|         for c in range(1,n+1):
1182|             if a**2 + b**2 == c**2:
1183|                 L.append((a,b,c))
1184|     return L
1185|
1186| print(pythagoricien(10))
1187|
1188| ## Exercice 42
1189| import random as rd
1190| def suivant(k,N):
1191|     return x+2*(rd.random() < x/N)+1
1192|
1193| print(suivant(3,10))
1194|
1195| def simulation(n,N):
1196|     L = [0]
1197|     for _ in range(n):
1198|         L.append(suivant(L[-1],N))
1199|     return L
1200|
1201| print(simulation(10,100))
1202|
1203| ## Exercice 43
1204| def decomposition2(n):
1205|     return [(i,n-i) for i in range(1,n) if i < n/2]
1206|
1207| print(decomposition2(20))
1208|
1209| def decomposition3(n):
1210|     L = []
1211|     for i in range(1,n):
1212|         for l in decomposition2(n-i):
1213|             if l[0] > i:
1214|                 L.append((i,l[0],l[1]))
1215|     return L
1216|
1217| print(decomposition3(20))
1218|
1219| ## Exercice 44
1220| import random as rd
1221| def manche(n):
1222|     return[rd.randint(0,1) for _ in range(n)] # 1 = Pile,
```

```
0 = Face
1223|
1224| print(manche(10))
1225|
1226| def jeu(n):
1227|     c = 0
1228|     while sum(manche(n)) not in [1,n-1]:
1229|         c += 1
1230|     return c
1231|
1232| print(jeu(10))
1233|
1234| def moyenne(n):
1235|     N = 1000
1236|     return sum([jeu(n) for _ in range(N)])/N
1237|
1238| print(moyenne(10))
1239|
1240| ## Exercice 45
1241| def compter(L,x):
1242|     c = 0
1243|     for l in L:
1244|         c += (l == x)
1245|     return c
1246|
1247| L = [rd.randint(1,5) for _ in range(10)]
1248| print(L)
1249| print(compter (L,3))
1250|
1251| def majoritaire(L):
1252|     for x in L:
1253|         if compter (L,x) > len(L)/2:
1254|             return x
1255|     return 0
1256|
1257| L = [rd.randint(1,3) for _ in range(10)]
1258| print(L)
1259|
1260| ## Exercice 46
1261| import random as rd
1262| def numero(n):
1263|     num = 0
1264|     for _ in range(n):
1265|         num += rd.randint(0,1)
1266|     return num
1267|
1268| print(numero(10))
1269|
1270| def repartition(N,n):
1271|     L = [0]*(n+1)
```

```
    for _ in range(N):
        L[numero(n)] += 1
    return L

print(repartition(100,10))

## Exercice 47
# ch = str(u)
def robinson(n):
    u = 0
    for _ in range(n):
        chaine = str(u)
        L = [0]*10
        for c in chaine:
            L[int(c)] += 1
        res = ''
        for j in range(9, -1, -1):
            if L[j] != 0:
                res += str(L[j])+str(j)
        u = int(res)
    return u

print(robinson(2))

def long_Rob(n):
    return len(str(robinson(n)))

print(long_Rob(5))

import matplotlib.pyplot as plt
Ln = range(20)
L = [long_Rob(n) for n in Ln]
plt.plot(Ln,L,marker='.')
plt.show()

## Exercice 48
def Nbmin(passe):
    nb = 0
    for i in passe:
        if 'a' <= i <= 'z':
            nb += 1
    return nb

def NbMaj(passe):
    nb = 0
    for i in passe:
        if 'A' <= i <= 'Z':
            nb += 1
    return nb

```

```
1322| def NbnonAlpha(passe):
1323|     return len(passe)-NbMaj(passe)-Nbmin(passe)
1324|
1325| def longMaj(passe):
1326|     d = 0
1327|     s = 0
1328|     i = 0
1329|     while i < len(passe):
1330|         if 'A' < passe[i] < 'Z':
1331|             s += 1
1332|         else:
1333|             if s > d:
1334|                 d = s
1335|                 s = 0
1336|         i += 1
1337|
1338|     return d
1339|
1340| def longmin(passe):
1341|     d = 0
1342|     s = 0
1343|     i = 0
1344|     while i < len(passe):
1345|         if 'a' < passe[i] < 'z':
1346|             s += 1
1347|         else:
1348|             if s > d:
1349|                 d = s
1350|                 s = 0
1351|         i += 1
1352|
1353|     return d
1354|
1355| def score(passe):
1356|     bonus = (len(passe)-Nbmin(passe))*3+(len(passe)-
NbMaj(passe))*2+(len(passe)-NbnonAlpha(passe))*5
1357|     penalites = longMaj(passe)*3+longmin(passe)*2
1358|     val = bonus-penalites
1359|     if val < 20:
1360|         print('Très faible')
1361|     elif val < 40:
1362|         print('Faible')
1363|     elif val < 80:
1364|         print('Fort')
1365|     else:
1366|         print('Très fort')
1367|
1368| pas = "P@SI_promo2016"
1369| score(pas)
```