Listes

Correction des exercices :
Note préliminaire : Pour me simplifier la tâche, j'ai utilisé l'éditeur IDLE de Python sur mon ordinateur afin de mettre au point et enregistrer les programmes qui suivent. Les copies d'écran proviennent de cette phase de travail. J'ai ensuite testé les programmes achevés et fonctionnels dans le Workshop de Numworks (onglet « Mes scripts »). Certaines copies d'écran proviennent de ce site mais pour ce qui est de la console, la plupart des lignes étant cachées, il est difficile d'obtenir un rendu satisfaisant.

1) Écrire une fonction qui supprime les doublons d'une liste : [1,2,2,1,2,1] est changée en [1,2].

On peut parcourir la liste L1 de départ et ajouter, dans une liste L2, les éléments s'ils n'y sont pas déjà. Une 2de option serait, pour chaque élément de L1 se demander si il est présent à un rang supérieur, et dans ce cas supprimer les occurrences redondantes (on n'a pas alors besoin d'une liste L2). On peut aussi recopier avec un filtrage la liste L1. Nous allons essayer ces différentes options, mais ce choix d'options n'est pas exhaustif : la mission peut être réalisée encore autrement.

```
def elimine1(L):
    P=[]
    for e in L:
        if e not in P:P.append(e)
    return P

def elimine2(L):
    for e in L:
        if L.count(e)>1:
            for i in range(L.count(e)-1):
                L.remove(e)
    return L

def elimine3(L):
    P=[e for r,e in enumerate(L) if L.index(e)==r]
    return P

def elimine4(L):
    P=set()
    for e in L: P.add(e)
    return list(P)

L1=[1,2,5,8,9,5,2,1,6,4,8,2,4,2,5,4,6,25,15,1,25,3,6,7,8,49,1,2,6]
L2=elimine4(L1)
print("la liste ne contient que",len(L2),"valeurs différentes",L2)
```

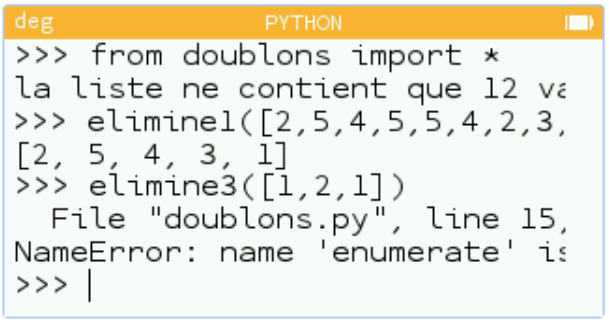
```
la liste ne contient que 12 valeurs différentes [1, 2, 5, 8, 9, 6, 4, 25, 15, 3, 7, 49]
la liste ne contient que 12 valeurs différentes [9, 5, 4, 15, 25, 3, 7, 8, 49, 1, 2, 6]
la liste ne contient que 12 valeurs différentes [1, 2, 5, 8, 9, 6, 4, 25, 15, 3, 7, 49]
la liste ne contient que 12 valeurs différentes [1, 2, 3, 4, 5, 6, 7, 8, 9, 15, 49, 25]
```

Pour réaliser la 3ème option, je me suis aidé de la méthode enumerate(L) de parcours d'une liste L qui donne simultanément le rang (appelé aussi index) et l'élément (appelé aussi valeur). Le filtrage réalisé permet de ne garder que les 1ères occurrences d'un élément. Comparer avec la méthode 2 qui supprime les 1ères occurrences tant qu'il y en a. L'ordre des éléments résiduels est alors différent des 2 autres méthodes car celle-ci ne garde que les dernières occurrences d'un élément.

J'ai ajouté une 4ème méthode qui utilise une autre structure, celle d'ensemble (la structure set). Dans un ensemble, il ne peut y avoir qu'une seule occurrence d'un élément. La procédure élimine ainsi les doublons, en ajoutant les éléments e de L dans l'ensemble P (avec la fonction P.add(e)). Elle renvoie tout de même une liste puisqu'on peut convertir un ensemble en liste. On peut d'ailleurs faire plus simplement, en une seule instruction L=list(set(L)). Cette 4ème méthode renvoie une liste

triée différemment, comme on peut le voir sur les affichages de ces quatre méthodes.

Ces différentes options sont-elles utilisables sur la Numworks ? Je les essaie dans leur Workshop : il s'avère que les deux dernières ne le sont pas encore, du fait de la fonction enumerate pour la 3ème et de la structure set pour la dernière. Voici la copie d'écran obtenue quand on lance le programme (avec l'option elimine2 pour la 1ère ligne). On peut avoir accès aux lignes entières en déplaçant le curseur dessus, mais on ne peut pas obtenir une plus grande fenêtre pour la console.



2) Écrire un programme qui renvoie une main de N cartes prises au hasard dans un jeu de 32 cartes. Par exemple, avec N=2, on devrait pouvoir obtenir [As de Trèfle, Dix de Pique].

Une première approche est la fonction cartes(n) qui choisit n fois une valeur et une couleur dans les listes correspondantes avec la fonction choice(liste).

```
from random import *
def cartes(n):
    coul=['Trèfle','Carreau','Coeur','Pique']
    vale=['7','8','9','10','Valet','Dame','Roi','As']
    main=[]
    for i in range(n):
        carte="{} de {}".format(choice(vale),choice(coul))
        main.append(carte)
    return main

print(cartes(4))
```

```
['9 de Trèfle', 'Valet de Trèfle', '9 de Pique', 'Roi de Pique']
['8 de Carreau', '8 de Carreau', '7 de Pique', 'Dame de Trèfle']
```

Pour éviter d'avoir deux cartes identiques, comme dans le 2ème essai, on peut remplacer la boucle for par une boucle while. La condition d'arrêt du while peut porter sur la longueur de la liste main créée. Avant d'ajouter une nouvelle carte, on vérifie que l'élément n'est pas déjà présent dans la liste. Pour améliorer l'affichage de la liste, je remplace les crochets et les virgules de la liste par un élégant tiret avec la méthode join(liste) qui s'applique à la chaîne de caractères que l'on veut insérer entre les éléments de la liste.

```
from random import *
def cartes(n):
    coul=['Trèfle','Carreau','Coeur','Pique']
    vale=['7','8','9','10','Valet','Dame','Roi','As']
    main=[]
    while len(main)<n:
        carte="{} de {}".format(choice(vale),choice(coul))
        if carte not in main : main.append(carte)
    return main

print(" - ".join(cartes(4)))
```

```
9 de Coeur - Valet de Carreau - Roi de Trèfle - 10 de Coeur
9 de Pique - As de Trèfle - 7 de Coeur - 10 de Carreau
```

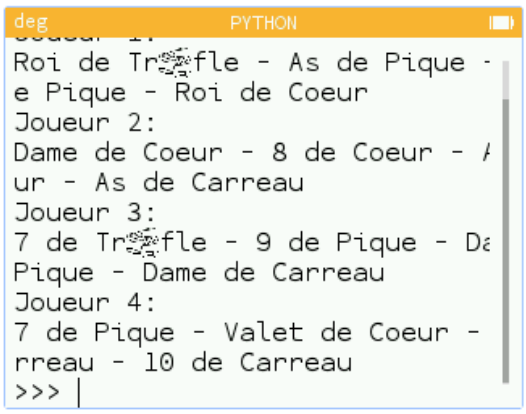
Pour prolonger ce travail, on pourrait écrire un programme qui renvoie P mains différentes de N cartes, les P mains ne pouvant contenir les mêmes cartes. On pourrait ainsi distribuer le jeu entre P joueurs, par exemple cartes(8,4) distribuerait 8 cartes à 4 joueurs.

Pour réaliser ce projet, il serait peut-être plus judicieux de générer une liste contenant l'ensemble des cartes et de tirer ensuite dans cette liste un élément au hasard que l'on supprimerait de la liste (pour éviter l'inconvénient d'une boucle while qui tire très souvent et pour rien des cartes déjà tirées). Le programme qui suit réalise cela, d'une part en parcourant les deux listes (coul et vale) en entiers pour constituer la liste jeu, et d'autre part en tirant un élément de la liste jeu au hasard et en l'y supprimant avec cette instruction à tiroirs : main.append(jeu.pop(randrange(len(jeu)))).

```
from random import *
def cartes(n,p):
    coul=['Trèfle','Carreau','Coeur','Pique']
    vale=['7','8','9','10','Valet','Dame','Roi','As']
    jeu,mains=[],[]
    for c in coul:
        for v in vale:
            jeu.append("{} de {}".format(v,c))
    for i in range(p):
        main=[]
        while len(main)<n:
            main.append(jeu.pop(randrange(len(jeu))))
        mains.append(main)
    return mains

nb_cartes,nb_joueurs=8,4
jeux=cartes(nb_cartes,nb_joueurs)
for i in range(nb_joueurs):
    print("Joueur {}:".format(i+1))
    print(" - ".join(jeux[i]))
```

```
Joueur 1:
Dame de Coeur - Dame de Trèfle - Roi de Coeur - 8 de Coeur - Roi
de Carreau - Dame de Carreau - 10 de Carreau - Valet de Carreau
Joueur 2:
Roi de Pique - Valet de Trèfle - 9 de Trèfle - Roi de Trèfle - 7
de Coeur - 10 de Pique - 9 de Carreau - Dame de Pique
Joueur 3:
8 de Carreau - 9 de Pique - Valet de Coeur - 7 de Pique - 8 de T
rèfle - As de Carreau - 7 de Carreau - As de Trèfle
Joueur 4:
Valet de Pique - As de Pique - 8 de Pique - As de Coeur - 7 de T
rèfle - 10 de Coeur - 10 de Trèfle - 9 de Coeur
```

Ces différents programmes fonctionnent-ils sur la Numworks ? Oui, sans aucun problème, exceptés les deux problèmes mineurs d'affichage des lignes tronquées et des accentuations non supportées (l'accent sur 'Trèfle' donne un joli dessin (voir illustration) et, d'une façon générale, la présence d'accents (ou de tout autre caractère qui sorte du code ASCII) rend impossible la sauvegarde du script sur le Workshop. Il suffit d'éliminer les accents du script pour obtenir un programme fonctionnel. Quant à l'affichage sur des lignes trop longues, on pourrait en tenir compte en prévoyant dans le programme de couper les lignes (ici, il suffirait d'afficher deux cartes par ligne).



3) Écrire un programme qui affiche un petit calendrier de la semaine lorsqu'on lui fournit en argument la date du dimanche. Si on entre 29,4,18 (29 avril 2018), ce programme doit nous afficher Lundi 30 avril 2018, Mardi 1 mai 2018, …, Dimanche 6 mai 2018.

Pas de difficulté majeure ici, sauf qu'il faut penser à traiter tous les cas (semaine à cheval sur deux mois ou deux années, années bissextiles). Ce programme nécessite qu'on lui fournisse le nom et la longueur des mois ainsi que la règle concernant les années bissextiles. J'ai simplifié le problème en ne traitant que les années après le 01/01/2000. Dans cette version, il faut entrer la date du dimanche qui précède la semaine (ce qui était demandé) mais cela amoindrit l'intérêt d'un tel programme.

```
def calendrier(j,m,a):
    nom_jours=['Lundi','Mardi','Mercredi','Jeudi','Vendredi','Samedi','Dimanche']
    nom_mois=['janvier','février','mars','avril','mai','juin','juillet','aout'
              ,'septembre','octobre','novembre','décembre']
    long=[31,28,31,30,31,30,31,31,30,31,30,31]
    if a%4==0 and not a==0:long[1]=29
    semaine,jour,mois,annee=[],j,m,2000+a
    while len(semaine)<7:
        jour+=1
        if jour>long[mois-1]:
            mois+=1
            jour=1
            if mois==13:
                mois=1
                annee+=1
        semaine.append([nom_jours[len(semaine)],str(jour),nom_mois[mois-1],str(annee)])
    return semaine

Cal=calendrier(29,4,18)
for i in range(7):
    print(" ".join(Cal[i]))
    print(" ... ")
```

```
                          Jeudi 3 mai 2018
                           ...
Lundi 30 avril 2018       Vendredi 4 mai 2018
 ...                       ...
Mardi 1 mai 2018          Samedi 5 mai 2018
 ...                       ...
Mercredi 2 mai 2018       Dimanche 6 mai 2018
 ...                       ...
```

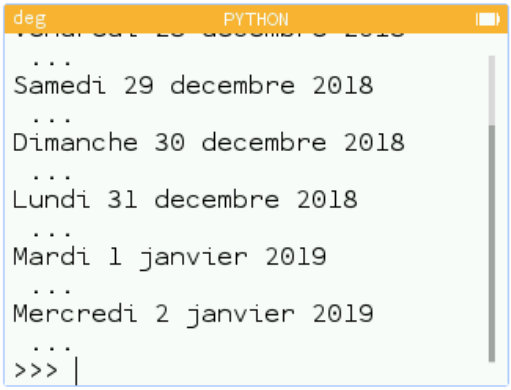
On peut imaginer qu'il serait plus utile si, en fournissant la date du jour, par exemple 27 décembre 2018 sous la forme du triplet (27,12,18), le programme se charge de trouver le jour de la semaine correspondant à cette date et affiche la semaine qui suit cette date.

Dans le programme qui suit, pour déterminer le jour de la semaine, le module jourSemaine(j,m,a) calcule le nombre de jours écoulés entre le 01/01/2000 qui était un samedi et la date du jour (j,m,a).

```
def calendrier(j,m,a):
    if a%4==0:long[1]=29
    semaine,jour,mois,annee=[],j,m,2000+a
    while len(semaine)<7:
        semaine.append([nom_jours[(len(semaine)+JS)%7],str(jour),nom_mois[mois-1],str(annee)])
        jour+=1
        if jour>long[mois-1]:
            mois+=1
            jour=1
            if mois==13:
                mois=1
                annee+=1
    return semaine

def jourSemaine(j,m,a):
    an,mo,jo=0,1,5 #le 1er janvier 2000 était un samedi (j=5)
    while an<a:
        if an%4==0:jo+=366
        else:jo+=365
        an+=1
    while mo<m:
        if a%4==0 and mo==2:jo+=long[mo-1]+1
        else:jo+=long[mo-1]
        mo+=1
    jo+=j-1
    return jo%7

jourJ,moisM,anA=27,12,18 #entrer ici la date du jour désiré (après le 01/01/00)
long=[31,28,31,30,31,30,31,31,30,31,30,31]
nom_jours=['Lundi','Mardi','Mercredi','Jeudi','Vendredi','Samedi','Dimanche']
nom_mois=['janvier','février','mars','avril','mai','juin','juillet','aout'
          ,'septembre','octobre','novembre','décembre']
JS=jourSemaine(jourJ,moisM,anA)
Cal=calendrier(jourJ,moisM,anA)
for i in range(7):
    print(" ".join(Cal[i]))
    print(" ... ")
```

```
Jeudi 27 décembre 2018
 ...
Vendredi 28 décembre 2018
 ...
Samedi 29 décembre 2018
 ...
Dimanche 30 décembre 2018
 ...
Lundi 31 décembre 2018
 ...
Mardi 1 janvier 2019
 ...
Mercredi 2 janvier 2019
 ...
```

Trois listes sont fixes ici et permettent de donner le nom des jours (nom_jours), des mois (nom_mois) et la longueur des mois d'une année ordinaire (long). La liste semaine, par contre, est variable, remplie progressivement avec les jours de la semaine cherchés. La sortie sur une Numworks a t-elle bonne allure ? Assez, jugez-en vous même. En supprimant les lignes de '…' (qui sont bien inutiles) on peut même avoir la semaine en entier. On peut prolonger ce type de programme pour obtenir l'affichage d'un mois ou d'une année-type (sans tenir compte des années bissextiles, il n'y a que 7 types d'années : celles qui commencent un lundi, celles qui commencent un mardi, etc.). On peut aussi déterminer la phase de la Lune...



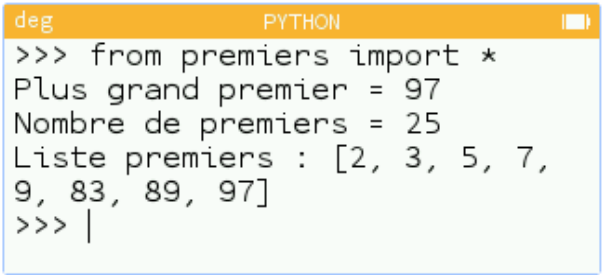
4) Écrire un programme qui détermine la liste des nombres premiers inférieurs ou égaux à un entier *n* donné. Selon la méthode du crible d'Ératosthène, on peut supprimer d'une liste contenant les entiers entre 2 et n tous les multiples de 2, puis supprimer tous les multiples du nombre suivant 2 (sauf erreur, cela doit être 3), etc.

La méthode du crible d'Ératosthène est relativement simple à mettre en œuvre : on parcourt la liste des nombres entiers inférieurs ou égaux à n autant de fois qu'il est possible en supprimant les multiples du premier nombre non supprimé. La liste de départ est obtenue avec list(range(2,n+1)) qui contient [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] lorsque n=12. La suppression des multiples de k=2 (k indique le premier nombre non supprimé) s'obtient en redéfinissant la liste prem, par l'instruction prem=[p for p in prem if p<=k or p%k!=0] qui opère le tri p<=k or p%k!=0 signifiant : soit p est premier (inférieur au dernier nombre non supprimé), soit p n'est pas divisible par k. On recommence l'opération après avoir choisi la nouvelle valeur de k avec k=prem[prem.index(k)+1]. On arrive ainsi à [2, 3, 5, 7, 11] après l'avoir parcourue pour k=2 et k=3.
Le prochain nombre non supprimé est k=5, mais on n'a pas besoin de supprimer ses multiples car le premier qui n'a pas déjà été supprimé est 5×5, un nombre supérieur à n=12. Pour cette raison, on arrête la procédure quand k atteint ou dépasse $\sqrt{n}$.

```
def premiers(n):
    prem=list(range(2,n+1))
    k=2
    nRacine=n**0.5
    while k<nRacine :
        prem=[p for p in prem if p<=k or p%k!=0]
        k=prem[prem.index(k)+1]    # nouveau nombre premier
    return prem

Liste_premiers=premiers(100)
print("Plus grand premier =",Liste_premiers[-1])
print("Nombre de premiers =",len(Liste_premiers))
print("Liste premiers :",Liste_premiers)
```

```
Plus grand premier = 97
Nombre de premiers = 25
Liste premiers : [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

Il y a sans doute de nombreuses autres façons de générer cette liste. Nous en resterons à celle-là qui est assez concise et efficace. Comment se comporte ce programme sur la Numworks ? Très bien, sauf que la plus grande partie de la liste est cachée à l'affichage. On peut penser mieux l'affichage en mettant à profit les 30 caractères affichés par ligne : un nombre par ligne devrait permettre d'avoir un affichage correct (mais est-ce utile?) jusqu'au plus grand nombre premier à 30 chiffres...



5) En partant de l'écriture irréductible d'un nombre rationnel donnée par deux nombres a et b (le numérateur et le dénominateur de la fraction), écrire une fonction qui donne la nature du nombre $\frac{a}{b}$ (entiers, décimal non entier ou rationnel non décimal) et la suite des chiffres qui se répète dans l'écriture décimale d'un rationnel non décimal. Si on entre 1 et 7, la fonction doit renvoyer 142857. Si le nombre est décimal, la fonction donne son écriture décimale.

Il faut effectuer une division euclidienne de a par b jusqu'à reconnaître le reste dans la liste des restes déjà obtenus. Les restes sont mis dans une liste tant qu'ils n'y sont pas déjà. Pour détecter la séquence périodique, on doit juste déterminer le rang du reste qui a permis de détecter la répétition.

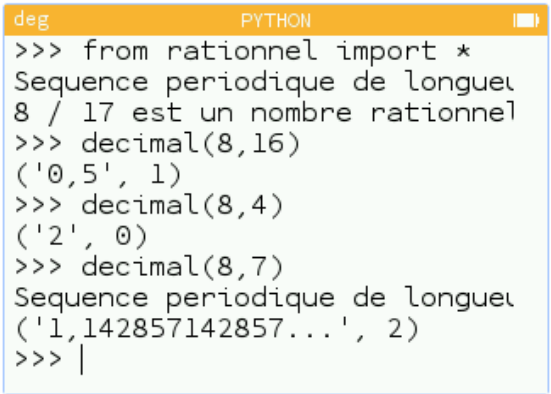
```
def decimal(a,b):
    quotient=str(a//b)  #la partie entière du quotient
    reste=a%b
    texte=0
    if reste!=0:
        suiteRestes=[reste]
        quotient+=","
        while texte==0:
            quotient+=str(reste*10//b)
            reste=(reste*10)%b
            suiteRestes.append(reste)
            if reste==0:
                texte=1
                break
            for i in range(len(suiteRestes)-1) :
                if reste==suiteRestes[i] :
                    long=len(suiteRestes)-i-1
                    sequence=str(quotient[-long:])
                    quotient+=sequence+"..."
                    print("Séquence périodique de longueur "+str(long)+" : "+sequence)
                    texte=2
    return texte,quotient

num,denom=1,7
t,q=decimal(num,denom)
nature=["entier","décimal non entier","rationnel non décimal"]
print(num,"/",denom,"est un nombre "+nature[t],":",q)
```

```
Séquence périodique de longueur 6 : 142857
1 / 7 est un nombre rationnel non décimal : 0,142857142857...
```

La fonction decimal réalise cela et renvoie un quotient décimal contenant deux répétitions de la séquence périodique lorsque le rationnel n'est pas décimal. L'affichage de la séquence et de sa longueur est juste donné en bonus, pour éviter d'avoir à la retrouver ou à la compter.

L'instruction quotient[-long:] permet de retrouver la séquence en découpant dans la chaîne de caractères quotient une tranche de longueur long, en commençant par la fin. Cette instruction utilise le fait que Python considère les chaînes de caractères comme des listes. Par exemple, si t='bonjour' alors t[0] vaut 'b' et t[-2:] vaut 'ur'.

Passons à la Numworks. Encore une fois, tout est correct excepté les limites de lisibilité imposées par l'affichage. Encore une fois, une amélioration notable de l'affichage serait obtenue en coupant les grands nombres toutes les 25 ou 30 décimales (à ajuster) et en concevant des lignes plus courtes pour le texte accompagnateur. Pour la découpe en tranches d'un texte, la fonctionnalité L[d:f] de Python sera utilement mise à profit. Ici, pour le quotient q, on affichera q[:25] sur la ligne 1, puis q[26,50] sur la ligne 2, jusqu'à q[x:] où le rang x est à déterminer.



6) Un problème ancien : Josèphe fut amené à se disposer en cercle avec quarante personnes, sachant qu'en comptant de trois en trois à partir de l'un d'eux, ils devaient se supprimer mutuellement. Le premier à être supprimé est rangé en position 3 et il se fait tuer par celui qui est rangé en position 6 ; celui-ci est le suivant à être supprimé, par celui en position 9. Ainsi de suite, progressivement, toute la troupe est appelée à s'autodétruire. À quelle place doit se ranger Josèphe pour être le dernier et devoir ainsi se supprimer lui-même ?

Commençons par construire la liste initial des 41 numéros (de 1 à 41) attribués aux personnes en cercle. Dans cette liste, on supprime un numéro sur 3 avec la fonction pop(rg) qui élimine le numéro de rang rg et on enregistre ce numéro dans la liste ordre. La longueur de la liste len(initial) changeant à chaque suppression d'une personne, pour savoir le rang rg de la prochaine personne à supprimer, on ajoute p –1 à l'ancien rang (le nombre de personnes sautées) et on se ramène au début de la liste en examinant le reste de la division par len(initial). En effet, arrivés en fin de la liste, il faut continuer comme si le 1er était derrière le dernier. La boucle while qui contient ces deux instructions assure que le processus ne s'achève qu'avec la suppression de la dernière personne. En remplaçant 41 et 3 par les lettres n et p, ce programme pourra être réutilisé avec d'autres valeurs.

```
n,p,rg=41,3,0
initial,ordre=list(range(1,n+1)),[]
while len(initial)>0:
    rg=(rg+p-1)%len(initial)
    ordre.append(initial.pop(rg))
print('Ordre de décimation:',ordre)
#pour savoir dans quel ordre seront supprimés les personnes
passage=n*[0]
for i in range(n): passage[ordre[i]-1]=i+1
print('Ordre de passage:',passage)
```

```
Ordre de décimation: [3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33,
36, 39, 1, 5, 10, 14, 19, 23, 28, 32, 37, 41, 7, 13, 20, 26, 34
, 40, 8, 17, 29, 38, 11, 25, 2, 22, 4, 35, 16, 31]
Ordre de passage: [14, 36, 1, 38, 15, 2, 24, 30, 3, 16, 34, 4,
25, 17, 5, 40, 31, 6, 18, 26, 7, 37, 19, 8, 35, 27, 9, 20, 32,
10, 41, 21, 11, 28, 39, 12, 22, 33, 13, 29, 23]
```

D'après le résultat, Josèphe doit se mettre en 31ème position pour avoir le privilège d'être le dernier à s'autodétruire comme le capitaine d'un navire en perdition qui met un point d'honneur à être le dernier à le quitter. La liste passage qui a été ajoutée à la fin du programme nous donne le rang de l'exécution de chaque personne : le 1er est tué en 14ème, le 2ème est tué en 36ème, le 3ème en 1er, etc.

Ce problème me fait penser à ces ritournelles enfantines où l'on se met en cercle pour désigner celui qui sera le chat (ou que sais-je d'autre) au début de la partie : « plouf plouf, une poule en or c'est toi qui sort » ou bien la plus obscure formule « amstragram pic et pic et colegram bour et bour et ratatam, mistramgram » (je ne suis pas bien sûr de la fin, je cite de mémoire, j'ai trouvé sur internet cette comptine qui termine par « pic » au lieu de « mistramgram »). Tout ça pour dire que le problème de Josèphe se retrouve, de façon moins dramatique, dans nos vies d'enfant. À quel rang faut-il être pour être choisi (équivalent de supprimé en dernier) ? Supposons donc que nous sommes 5 joyeux bambins et que nous faisions la « plouf », une comptine qui fait 10 pieds si je ne m'abuse. J'entre n=5 et p=10 dans le programme et trouve qu'il faut me mettre en avant-dernier (la position 4).

```
n,p,rg=5,10,0
```

```
Ordre de décimation: [5, 2, 3, 1, 4]
Ordre de passage: [4, 2, 3, 5, 1]
```

Une question intéressante : se peut-il toujours que cette procédure de sélection commence et finisse sur la même personne ? Le nombre n de personnes étant connu, comment choisir le pas p pour que le 1er soit le dernier ? La réponse n'est pas évidente sans le programme. Prenons n=5 personnes, cela arrive pour la 1ère fois pour p=4.

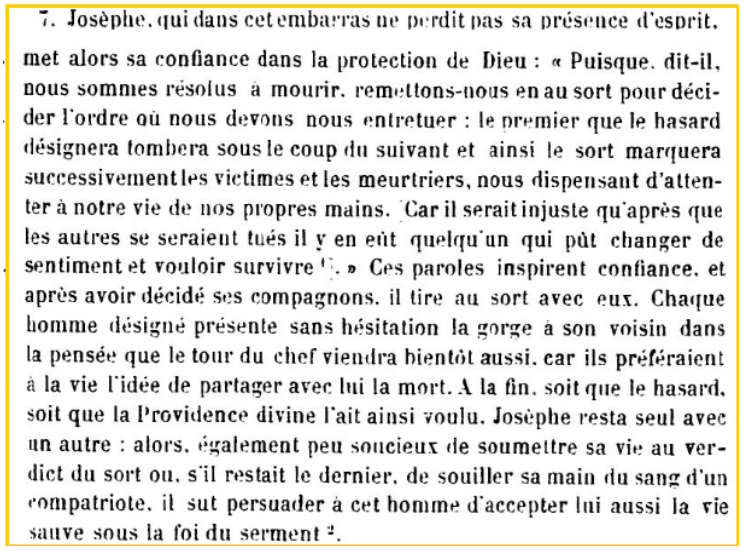
```
Ordre de décimation pour n=5 et p=1: [1, 2, 3, 4, 5]
Ordre de passage: [1, 2, 3, 4, 5]

Ordre de décimation pour n=5 et p=2: [2, 4, 1, 5, 3]
Ordre de passage: [3, 1, 5, 2, 4]

Ordre de décimation pour n=5 et p=3: [3, 1, 5, 2, 4]
Ordre de passage: [2, 4, 1, 5, 3]

Ordre de décimation pour n=5 et p=4: [4, 3, 5, 2, 1]
Ordre de passage: [5, 4, 2, 1, 3]

....
```

Mais si on augmente la valeur de p, d'autres valeurs possibles continuent à arriver. La suivante est p=8, ensuite il faut attendre jusqu'à p=15, etc. Pour éviter de relancer le programme à chaque fois, j'écris une boucle qui le fait automatiquement. Je trouve alors la liste des valeurs possibles pour p quand n=5, qui semble ne pas avoir de fin : 4, 8, 15, 18, 19, 22, 29, 33, 37, 47, …

Et pour les autres valeurs de n ? Il semble que ce soit toujours possible, parfois le plus petit pas possible est plus grand que n (pour n=3, 7, 11, 13, 18, 19, 27, 34, 35, …), la liste est fort irrégulière. Je regarde par curiosité si cette liste est répertoriée dans l'Encyclopédie des suites d'entiers de Sloane. Non, pas encore. D'autres y sont, comme 2, 5, 2, 4, 3, 11, 2, 3, 8, etc. qui est la liste des premières valeurs de p pour que le premier soit le dernier (liste est référencée A187788).

```
pour n=2: [2, 4, 6, 8,      pour n=18: [20, 22, 38,     pour n=34: [75, 126, 21
pour n=3: [5, 6, 11, 12     pour n=19: [34, 38, 54,     pour n=35: [42, 123,
pour n=4: [2, 3, 12, 14     pour n=20: [8, 16, 28,      pour n=36: [13, 16, 70,
pour n=5: [4, 8, 15, 18     pour n=21: [15, 20, 45,     pour n=37: [5, 21, 30,
pour n=6: [3, 5, 14, 16     pour n=22: [10, 24, 30,     pour n=38: [23, 26, 45,
pour n=7: [11, 12, 14,      pour n=23: [7, 38, 45,      pour n=39: [13, 87, 115
pour n=8: [2, 6, 24, 34     pour n=24: [13, 58, 68,     pour n=40: [50, 51, 75,
pour n=9: [3, 4, 15, 16     pour n=25: [11, 22, 24,     pour n=41: [16, 30, 67,
pour n=10: [8, 20, 24,      pour n=26: [13, 16, 50,     pour n=42: [18, 24, 37,
pour n=11: [16, 18, 28,     pour n=27: [45, 76, 81,     pour n=43: [89, 149,
pour n=12: [4, 5, 15,       pour n=28: [18, 67, 95,     pour n=44: [38, 92, 126
pour n=13: [21, 24, 37,     pour n=29: [23, 51, 55,     pour n=45: [8, 21, 28,
pour n=14: [6, 11, 27,      pour n=30: [8, 28, 34,      pour n=46: [39, 94, 140
pour n=15: [5, 25, 30,      pour n=31: [3, 37, 38,      pour n=47: [30, 67, 81,
pour n=16: [2, 4, 10,       pour n=32: [2, 53, 86,      pour n=48: [29, 42, 73,
pour n=17: [11, 15, 25,     pour n=33: [25, 58, 70,     pour n=49: [38, 144,
```

Les références de cette histoire sont complexe : il y eu tout d'abord ce qu'en dit Josèphe[1] lui-même. Ce personnage du 1er siècle parle de tirer au sort l'ordre des tués et mentionne que c'est le voisin qui procède à l'exécution (et non pas un sur trois, ce qui paraît plus logique). Si Josèphe se retrouve en dernier c'est le fruit du hasard... Ensuite, il y eut ce livre de Bachet[2] « Problèmes plaisants et délectables qui se font par les nombres » (1612) qui présente une situation différente mais qui repose sur le même algorithme : 15 chrétiens et 15 turcs dans un bateau trop chargé ; il faut jeter 15 hommes à la mer et on se fie au sort en jetant un homme sur neuf à la mer, les hommes étant rangés en cercle. La question « à quelles places mettre les chrétiens, pour qu'aucun d'eux ne soit jeté à la mer » se traite par le même algorithme. Bachet de Méziriac (1581-1638) aurait reformulé dans son livre ce que Nicolas Chuquet (1445-1487) avait publié en 1484 (problème 146 de son livre, édité pour la 1ère fois en 1880). L'idée de sauter deux hommes sur trois daterait de Cardan, *Practica Arithmeticae*, publié en 1536 ("*on connaît le jeu de Josèphe, qui avec celui-ci infligea la mort à ses compagnons de telle sorte que, dit-on, ceux-ci pensaient qu'elle leur arrivait par le sort, tandis que lui-même, vu que ceux-ci étaient pris au dépourvu, a été sauvé avec un compagnon seulement ; on dispose en cercle autant de petits cailloux qu'on le souhaite et pour deux comptés on en fait sortir un que l'on a choisi*"). J'ai tiré ces informations d'un article de Laurent Signac.



7. Josèphe, qui dans cet embarras ne perdit pas sa présence d'esprit, met alors sa confiance dans la protection de Dieu : « Puisque, dit-il, nous sommes résolus à mourir, remettons-nous en au sort pour décider l'ordre où nous devons nous entretuer : le premier que le hasard désignera tombera sous le coup du suivant et ainsi le sort marquera successivement les victimes et les meurtriers, nous dispensant d'attenter à notre vie de nos propres mains. Car il serait injuste qu'après que les autres se seraient tués il y en eût quelqu'un qui pût changer de sentiment et vouloir survivre [1]. » Ces paroles inspirent confiance, et après avoir décidé ses compagnons, il tire au sort avec eux. Chaque homme désigné présente sans hésitation la gorge à son voisin dans la pensée que le tour du chef viendra bientôt aussi, car ils préféraient à la vie l'idée de partager avec lui la mort. A la fin, soit que le hasard, soit que la Providence divine l'ait ainsi voulu, Josèphe resta seul avec un autre : alors, également peu soucieux de soumettre sa vie au verdict du sort ou, s'il restait le dernier, de souiller sa main du sang d'un compatriote, il sut persuader à cet homme d'accepter lui aussi la vie sauve sous la foi du serment [2].

1 Œuvres complètes de Flavius Josèphe (38-100) trad. en français, rev. et annot. de Théodore Reinach. Tome cinquième, Guerre des juifs. Livres I-III / trad. de René Harmand, Éd. E. Leroux (Paris), pp.290-291

2 Problèmes plaisants & délectables qui se font par les nombres (Troisième éd., rev., simplifiée et augm.) / par Claude-Gaspar Bachet, sieur de Méziriac Bachet, Claude-Gaspard (1581-1638) Éd. Gauthier-Villars (Paris) 1874, pp.118-121

7) On effectue une collection d'objets choisis au hasard parmi N objets numérotés de 1 à N en cherchant à obtenir la collection complète. Par exemple, on tire au hasard un chiffre (entre 0 et 9) jusqu'à ce qu'on obtienne les 10 chiffres au moins une fois chacun. Simuler cette situation avec un programme qui prend N en argument et qui renvoie une liste contenant le nombre d'éléments de chaque sorte. Si on lance cette fonction avec N=10, on doit pouvoir obtenir quelque chose comme [3,5,1,2,1,2,5,3,1,1] qui montre qu'on a obtenu le 1er élément 3 fois, le 2ème 5 fois, … et le 10ème 1 fois.

Pour le moment, la correction de cet exercice n'est pas disponible.
Mes élèves de seconde comprendront pourquoi... ☺

8) On veut parcourir l'ensemble des rationnels strictement positifs en rangeant les fractions $\frac{a}{b}$ dans l'ordre de la somme $s=a+b$ et, pour une même valeur de s, dans l'ordre de a. L'idée est d'aller jusqu'à une certaine fraction $Q=\frac{A}{B}$ et d'afficher le nombre N et le pourcentage p des fractions irréductibles inférieures ou égales à Q. Avec $Q=\frac{2}{5}$, les 17 fractions rangées jusqu'à Q étant $\frac{1}{1}$, $\frac{1}{2}$, $\frac{2}{1}$, $\frac{1}{3}$, $\frac{2}{2}$, $\frac{3}{1}$, $\frac{1}{4}$, $\frac{2}{3}$, $\frac{3}{2}$, $\frac{4}{1}$, $\frac{1}{5}$, $\frac{2}{4}$, $\frac{3}{3}$, $\frac{4}{2}$, $\frac{5}{1}$, $\frac{1}{6}$ et $\frac{2}{5}$ parmi lesquelles 4 sont simplifiables ($\frac{2}{2}$, $\frac{2}{4}$, $\frac{3}{3}$ et $\frac{4}{2}$), on doit obtenir $N=17-4=13$ et $Q=\frac{13}{17}\approx 76{,}5\,\%$. Jusque là, on peut se passer des listes mais on souhaite obtenir l'affichage des fractions irréductibles dans l'ordre numérique croissant.

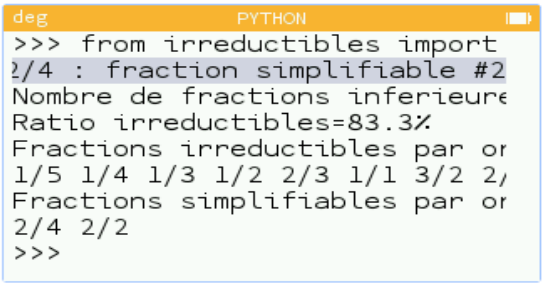
```
def pgcd(a,b):
    reste=a%b
    if reste==0 : return b
    else : return pgcd(b,reste)

def insere(a,b,L):
    rg,val=0,a/b
    for e in L:
        if e[2]>val:break
        else:rg+=1
    L.insert(rg,[a,b,val])

def denombre(n,d):
    som,tot,rang,irr,red=2,0,0,[],[]
    while som<=n+d:
        num,denom=1,som-1
        while num<som:
            PGCD=pgcd(num,denom)
            if PGCD==1:
                rang+=1
                insere(num,denom,irr)
            else:insere(num,denom,red)
            tot+=1
            if num==n and denom==d:return rang,tot,PGCD,irr,red
            num+=1
            denom-=1
        som+=1

a,b=2,6
rng,tot,gcd,Lirr,Lred=denombre(a,b)
if gcd==1:print('{}/{} : fraction irréductible n°{}'.format(a,b,rng))
else:print('{}/{} : fraction simplifiable n°{}'.format(a,b,tot-rng))
print('Nombre de fractions inférieures ou égales : {}'.format(tot))
print('Ratio irreductibles={}%'.format(round(rng/tot*100,1)))
print("Fractions irréductibles par ordre croissant:")
for f in Lirr:
    print('{}/{}'.format(f[0],f[1]),end=" ")
print("\nFractions simplifiables par ordre croissant:")
for f in Lred:
    print('{}/{}'.format(f[0],f[1]),end=" ")
```

```
2/5 : fraction irréductible n°13
Nombre de fractions inférieures ou égales : 17
Ratio irreductibles=76.5%
Fractions irréductibles par ordre croissant:
1/6 1/5 1/4 1/3 2/5 1/2 2/3 1/1 3/2 2/1 3/1 4/1 5/1
Fractions simplifiables par ordre croissant:
2/4 2/2 3/3 4/2
```

Il faut, pour ce programme, utiliser un module qui détermine le *pgcd* de deux entiers : pgcd en est une version récursive. La fonction denombre passe en revue les fractions concernées dans l'ordre du dénombrement et enregistre celles qui sont irréductibles dans une liste qu'il faut obtenir triée dans l'ordre numérique. Pour cette raison, j'enregistre les fractions sous la forme de listes de 3 éléments [a,b,d] où d est une valeur décimale (notation flottante des nombres réels) et j'insère la nouvelle fraction au bon endroit en me servant de d pour les ranger.

L'insertion proprement dite est confiée à la fonction insere qui détermine le rang de l'élément qui a une valeur décimale dépassant pour la première fois celle de l'élément à insérer. Remarquez que la liste irr peut être modifiée dans une autre partie que celle où elle a été déclarée sans qu'il soit nécessaire de renvoyer la dite liste par un bloc return. Ce n'était pas demandé mais le programme construit également la liste triée red des fractions réductibles et donne pour une telle fraction, son rang parmi les fractions réductibles.

Le programme s'exporte bien sur la Numworks, si ce n'est que l'affichage n'est pas bien étudié pour cette dimension d'écran. L'obligation de n'utiliser que des symboles ASCII m'a conduit à modifier l'affichage : au lieu du bon français « 2/4 : fraction simplifiable n°2 », on obtient l'anglicisme « 2/4 : fraction simplifiable #2 ». Why not ?



9) Voici l'énigme MU de Hofstadter[3] : on dispose de MI. À l'étape suivante, on ajoute toutes les chaînes que l'on peut obtenir par une des quatre règles suivantes : ①-Si on possède une chaîne se terminant par I, on peut lui ajouter U à la fin ; ②-Si on possède la chaîne Mx, on peut lui ajouter x pour obtenir la chaîne Mxx ; ③-Si on possède une chaîne contenant III, on peut remplacer III par U ; ④-Si une chaîne contient UU, on peut supprimer ce UU. L'énigme est la suivante : va t-on, à une étape quelconque, obtenir la chaîne MU ? Proposer un programme qui détermine les chaînes s'ajoutant à la collection à chaque étape et qui s'arrête quand MU est trouvé (cela peut ne jamais arriver). Vérifier qu'à l'étape 1 on obtient les chaînes MIU et MII et à l'étape 2 obtient MIUIU, MIIU et MIII.

Cet exercice peut paraître un peu abstrait et/ou arbitraire mais il s'agit du premier exemple de système formel donné par l'auteur dans son monumental livre qui en évoque bien d'autres. Comme il le fait remarquer, les deux premières règles augmentent la longueur des chaînes alors que les deux dernières la font diminuer. Ce peut-il qu'on obtienne MU en partant de MI ?

Le programme qui traduit cette recherche est une boucle non bornée qui pourrait bien être infinie, auquel cas on sera arrêté je suppose assez rapidement avec la Numworks. Sur mon ordinateur, le programme tourne quelques secondes pour obtenir les 7 premières étapes, puis la 8ème, mais la 9ème tarde vraiment à s'achever. Je le laisse travailler tranquillement une vingtaine de minutes pour atteindre cet objectif. La chaîne MU n'a pas été trouvée...

Ici, j'utilise deux listes anciens et nouveaux : en parcourant la 1ère, appliquant les 4 règles à chaque élément, la 2ème liste se constitue et en fin d'étape se déverse dans la 1ère, écrasant celle-ci. Ce fonctionnement est assez simple, ce sont les règles elles-mêmes qui sont plus délicates à traduire :

- m[-1:] désigne le dernier caractère de la chaîne m
- m[1:] désigne la chaîne m privée de son 1er caractère (le fameux x de la règle ②)
- m[0:R]+'U'+m[R+3:] est la chaîne obtenue en substituant U à III se trouvant au rang R
- m[0:R]+m[R+2:] est la chaîne obtenue en éliminant UU se trouvant au rang R

L'instruction R=m.find('III',R+1) recherche III dans la chaîne m à partir du rang R+1 et affecte le rang trouvé à R. Le grand intérêt de la fonction find est de renvoyer 0 si la chaîne n'est pas trouvée au lieu de déclencher une erreur comme le font d'autres fonctions (par exemple la fonction index).

---

3 Douglas Hofstadter, Gödel, Escher, Bach, les Brins d'une Guirlande Éternelle, Dunod 2000, pp.38-47.

Il me reste à tester le fonctionnement de ce programme sur la Numworks. Comme prévu, il déclenche une erreur d'allocation mémoire pendant la 6ème étape. Il n'y a que 232 chaînes à produire mais certaines sont très longues, la règle ② faisant quasiment doubler la longueur d'une chaîne à chaque étape.

```
def regle1(m):
    if m[-1:]=='I'and m+'U' not in nouveaux:
        nouveaux.append(m+'U')
def regle2(m):
    X=m[1:]
    if m+X not in nouveaux:
        nouveaux.append(m+X)
def regle3(m):
    R=m.find('III',0)
    while R>=0:
        if m[0:R]+'U'+m[R+3:] not in nouveaux:
            nouveaux.append(m[0:R]+'U'+m[R+3:])
        R=m.find('III',R+1)
def regle4(m):
    R=m.find('UU',0)
    while R>=0:
        if m[0:R]+m[R+2:] not in nouveaux:
            nouveaux.append(m[0:R]+m[R+2:])
        R=m.find('UU',R+1)

anciens,n=['MI'],0
print('Génération 0:',anciens)
while 'MU' not in anciens:
    nouveaux,n=[],n+1
    for m in anciens:
        regle1(m)
        regle2(m)
        regle3(m)
        regle4(m)
    if n<3:print('Génération {}:'.format(n),nouveaux)
    else:print('Génération {}: {} chaines'.format(n,len(nouveaux)))
    anciens=nouveaux
print('MU trouvé à la génération',n)
```

```
Génération 1: ['MIU', 'MII']
Génération 2: ['MIUIU', 'MIIU', 'MIIII']
Génération 3: 6 chaines
Génération 4: 15 chaines
Génération 5: 48 chaines
Génération 6: 232 chaines
Génération 7: 1544 chaines
Génération 8: 14959 chaines
Génération 9: 203333 chaines
```

Sur le modèle de cette énigme, on peut en créer bien d'autres dont certaines avec solution. Cela me rappelle certains jeux de lettres ou on propose de partir d'une chaîne pour arriver à une autre. Généralement, la règle oblige de changer une lettre à chaque étape, à condition que le mot ait un sens.

```
deg                PYTHON
>>> from myscript import *
Generation 0: ['MI']
Generation 1: ['MIU', 'MII']
Generation 2: ['MIUIU', 'MIIU
Generation 3: 6 chaines
Generation 4: 15 chaines
Generation 5: 48 chaines
  File "myscript.py", line 28,
  File "myscript.py", line 12,
MemoryError: memory allocatior
>>> |
```

10) Vous connaissez le code César : on commence par créer un alphabet de substitution en décalant toutes les lettres d'un certain nombre constituant la clé du code. On peut alors appliquer la substitution à chaque lettre du message à encoder ou bien, en sens inverse, on peut décoder un message. Par exemple, XKJFKQN code le message BONJOUR avec une clé de 4 (E code A, F code B, etc.). Le message suivant XQEBX MUEMZ FQDUQ EXQEB XGEOA GDFQE EAZFX QEYQU XXQGD QE a été encodé avec un code César mais on a perdu la clé. Écrire un programme qui puisse le décrypter. On peut utiliser le principe suivant : en français, la lettre la plus fréquente étant le E, il suffit de déterminer la lettre la plus fréquente pour savoir comment est codé le E et en déduire la clé et le message décodé. Si celui-ci n'a pas de sens, on peut essayer la 2ème lettre la plus fréquente.

Puisque nous allons traiter des caractères, je m'intéresse au code ASCII qui contient tous ceux qu'acceptent la Numworks. Bien sûr, on pourrait se contenter de déclarer une liste alphabet qui ressemblerait à ['a','b',...'z'] en supposant que le texte est préformaté en minuscule. On peut aussi utiliser la fonction upper() qui transforme un caractère minuscule en majuscule (par exemple 'a'.upper() donne 'A') ou d'autres fonctions similaires de la famille des chaînes de caractères. J'ai choisi ici d'utiliser la conversion entre le type chr (caractères) et le type int (entiers) des codes ASCII. Par exemple, ord('a') vaut 97 (le code ASCII de 'a') tandis que chr(97) vaut 'a'. L'observation de la table montre que de 65 à 90 on a les majuscules et de 97 à 122 les minuscules. Pour convertir en majuscule un caractère on peut ajouter 32 à son code ASCII, par exemple chr(ord('a')+32) vaut 'A', mais il faut avouer que c'est moins explicite que 'a'.upper().

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Ces préliminaires étant posés, voici un premier script.

```
def encode(t,c):
    tt=''
    for lettre in t :
        if lettre in ['é','è','ê','ë','È','É','Ê','Ë']:lettre='E'
        if lettre in ['à','À']:lettre='A'
        if lettre in ['ç']:lettre='C'
        lettre=lettre.upper()
        if 64<ord(lettre)<91:
            tt+=chr((ord(lettre)-65-c)%26+65)
        for e in range(len(tt)-len(tt)%5,0,-5):
            tt=tt[:e]+' '+tt[e+1:]
    return tt

texte='abcdefghijklmnopqrstuvwxyz'
clef=4
print('alphabet clair >> CODE ({}):'.format(clef))
print(encode(texte,0))
print(encode(texte,clef))
```

```
alphabet clair >> CODE (4):
ABCDE FGHI JKLM NOPQ RSTU VWXY Z
WXYZA BCDE FGHI JKLM NOPQ RSTU V
>>> encode('bonjour',4)
'XKJFK QN'
>>> encode('Comment vas-tu mère Michel?',4)
'YKIIA JPRW OPQI ANAI EYDA H'
```

J'ai traduit ici l'encodage d'un texte non préformaté où minuscules, majuscules, caractères accentués, chiffres et signes de ponctuation peuvent être présents. Dans le texte encodé, par contre, la ponctuation et les chiffres ont disparus et tout est en majuscule, les lettres étant par ailleurs groupées par 5.

Pourquoi ces choix ? Le texte de l'énoncé semble les suivre et elles sont justifiées car si on conservait la ponctuation, les espaces et la casse des caractères (majuscule/minuscule), le message serait trop facile à décoder. On peut critiquer l'élimination des chiffres qui peuvent constituer une partie importante du message et que l'on ne veut pas perdre. Pour cette raison, je vais ajouter une boucle dans la fonction encode qui aura pour but d'encoder un entier en décalant les chiffres selon la clé du code César (une clé de 10 ou de –10 ne modifiera donc pas l'entier). Les chiffres ont des codes ASCII allant de 48 (0) à 57 (9).

```
def encode(t,c):
    tt=''
    for lettre in t :
        if lettre in ['é','è','ê','ë','È','É','Ê','Ë']:lettre='E'
        if lettre in ['à','À']:lettre='A'
        if lettre in ['ç']:lettre='C'
        lettre=lettre.upper()
        if 64<ord(lettre)<91:
            tt+=chr((ord(lettre)-65-c)%26+65)
        if 47<ord(lettre)<58:
            tt+=chr((ord(lettre)-48-c)%10+48)
        for e in range(len(tt)-len(tt)%5,0,-5):
            tt=tt[:e]+' '+tt[e+1:]
    return tt

texte='abcdefghijklmnopqrstuvwxyz0123456789'
clef=4
print('alphabet clair >> CODE ({}):'.format(clef))
print(encode(texte,0))
print(encode(texte,clef))
```

```
alphabet clair >> CODE (4):
ABCDE FGHI JKLM NOPQ RSTU VWXY Z012 3456 789
WXYZA BCDE FGHI JKLM NOPQ RSTU V678 9012 345
>>> encode("J'ai 1000 amis sur FB et 2 chats",4)
'FWE76 66WI EOOQ NBXA P8YD WPO'
>>> encode("Pi vaut environ 3.141593",4)
'LERWQ PAJR ENKJ 9707 159'
>>> encode("LERWQ PAJR ENKJ 9707 159",-4)
'PIVAU TENV IRON 3141 593'
```

Pour la partie décodage, on va supposer que l'on part d'un texte en majuscules, sans ponctuation exceptés les espaces. L'idée est de trouver la clé supposée perdue car si on connaît la clé, il suffit d'appliquer la clé opposée pour décoder (voir le dernier exemple de l'illustration). Pour éviter d'essayer les 25 clés possibles, l'énoncé suggère de faire une analyse fréquentielle sommaire : on détermine la ou les lettre(s) la(les) plus fréquente(s) qui permettront d'arriver à la clé car cette lettre est supposée coder un E (en espérant que le message ne provient pas de 'La disparition', le roman sans 'e' de G. Perec).

```
def analyse(t):
    alpha,beta=26*[0],26*[0]
    for lettre in t :
        if 64<ord(lettre)<91:
            alpha[ord(lettre)-65]+=1
    print('alpha',alpha)
    val=[]
    for e in alpha:
        if e not in val and e!=0 : val.append(e)
    val.sort(reverse=True)
    j=0
    for n in val:
        start=0
        for i in range(alpha.count(n)):
            beta[j]=alpha.index(n,start)+1
            start=beta[j]
            j+=1
    print('beta',beta)
    return beta

texte='XQEBX MUEMZ FQDUQ EXQEB XGEOA GDFQE EAZFX QEYQU XXQGD QE'
Ordre=analyse(texte)
for i in range(3):
    print('clé={}'.format(Ordre[i]-5),encode(texte,Ordre[i]-5))
```

```
alpha [2, 2, 0, 3, 9, 3, 3, 0, 0, 0, 0, 0, 2, 0, 1, 0, 9, 0, 0, 0, 3, 0, 0, 7, 1, 2]
beta [5, 17, 24, 4, 6, 7, 21, 1, 2, 13, 26, 15, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
clé=0 XQEBX MUEM ZFQD UQEX QEBX GEOA GDFQ EEAZ FXQE YQUX XQGD QE
clé=12 LESPL AISA NTER IESL ESPL USCO URTE SSON TLES MEIL LEUR ES
clé=19 EXLIE TBLT GMXK BXLE XLIE NLVH NKMX LLHG MEXL FXBE EXNK XL
```

La fonction analyse procède au décompte des lettres du message, l'effectif de chacune étant enregistré dans la liste alpha. Je crée ensuite une liste val qui va contenir un seul exemplaire de chacun des effectifs trouvés. Cette liste est ensuite triée dans le sens décroissant (l'option reverse=True de la fonction sort est bien utile ici) afin de permettre le traitement suivant. La liste beta est alors constituée par la liste des lettres les plus fréquentes du message dans l'ordre décroissant. Par exemple, la lettre la plus fréquente du message de l'énoncé est le E (5ème lettre de l'alphabet), suivie, ou plutôt ici ex-aequo, du Q (17ème lettre de l'alphabet), suivie du X (24ème lettre de l'alphabet), etc. J'ai choisis d'afficher les trois premiers choix, espérant que le E est codé par une de ces trois lettres. Il s'avère ici que ce n'est pas le 1er choix (on s'en serait douté car si le E était codé E, le message codé serait en clair...) mais le second. La clé est donc trouvée en enlevant au rang de Q (17) le rang de E (5). Avec cette clé 12, on retrouve le sens du message dans lequel il n'y a qu'à rétablir les césures correctes des mots :

LES PLAISANTERIES LES PLUS COURTES SONT LES MEILLEURES

Suivant la sagesse de cet adage, j'en resterai là pour ces quelques exercices en Python. Pour de plus amples renseignements sur les listes, consulter la littérature en ligne (références dans la feuille d'énoncés). J'oubliais de mentionner le passage par la Workshop : j'ai enlevé les trois lignes de la méthode `encode` qui concernent les caractères accentués (inutiles sur la Numworks) et j'ai remplacé le mot clé par clef. Moyennant ces petits aménagements, le programme fonctionne correctement.