DEVOIR SURVEILLÉ N° 2 — **VENDREDI 19 DÉCEMBRE**
Informatique - MPSI 2 — *Durée : 2 h*

- La calculatrice est interdite.
- Seules sont autorisées l'utilisation des types de bases (entiers, flottants, chaînes de caractères, booléens), la structure de données liste (avec méthodes `append` et `pop`), les structures algorithmiques alternatives et itératives, et l'utilisation d'algorithmes récursifs.
- Si vous introduisez de nouvelles fonctions ou de nouvelles variables, on s'efforcera de leur donner un nom explicite et d'indiquer en commentaire leur rôle.
- Enfin, dans un exercice, il est possible (et même recommandé) d'utiliser les fonctions des questions précédentes, que ces questions aient été traitées ou non.

Exercice n° 1 — Quelques questions indépendantes

1. Donner la valeur des expressions suivantes, en justifiant la réponse :

a. `5**3`

b. `47//8`

c. `25==5*2 or (not(3*2!=6))`

d. `47%8`

2. L'instruction suivante a été validée dans la console : `L=[1,3,5,7,9,11,13,15]`
Déterminer la valeur des expressions suivantes :

a. `L[6]` **b.** `L[2:4]` **c.** `L[4:]` **d.** `L[:4]`

On effectue ensuite les instructions suivantes dans la console : `x=L.pop()` puis `L.append(25)`
Déterminer la valeur des expressions suivantes, en justifiant la réponse :

e. `x`

f. `L[8]`

3. On a désormais validé dans la console : `s1="Hello"` et `s2="World"`
Donner la valeur des expressions suivantes :

a. `s1+" "+s2+"!"`

b. `s1*3`

Exercice n° 2 — Une fonction de tri par sélection

On considère une liste `L` contenant n éléments. Le tri par sélection consiste à faire la démarche suivante :

- on cherche l'élément le plus petit de la liste, puis on échange cet élément avec le premier élément de la liste.

- on cherche le deuxième élément le plus petit de la liste, c'est-à-dire l'élément le plus petit de la liste privée du premier élément, puis on échange cet élément avec le deuxième élément de la liste.
- on poursuit jusqu'à arriver au dernier élément.

1. On considère la liste `L=[3,4,1,5,2]`. Donner les états successifs de cette liste lors du tri par sélection, en partant de la liste initiale et en arrivant à la liste totalement triée.

2. Écrire une fonction `IndiceMinimum(L,i)` prenant en argument une liste `L` et un entier `i` et qui renvoie `False` si l'entier `i` est supérieur ou égal à la longueur de la liste, ou qui renvoie l'indice du plus petit élément de la liste à partir de l'élément d'indice `i`.

Par exemple, si `L=[1,3,0,6,4]`, `IndiceMinimum(L,1)` renverra l'entier 2.

3. Écrire une fonction `tri_selection(L)` prenant en argument une liste `L` et qui trie la liste en place en utilisant la méthode du tri par sélection.

4. Quelle est la complexité de cette fonction ? On l'exprimera à l'aide de la longueur n de la liste `L`.

Exercice n° 3 — Une fonction à déterminer

On considère la fonction suivante, qui implémente un algorithme dont on cherche à déterminer le résultat. Les arguments a et b sont des entiers naturels, avec b non nul.

```
def Mystere(a,b) :
  q = 0
  r = a
  while r >= b :
    q += 1
    r = r - b
  return (q,r)
```

1. Que renvoie `Mystere(23,6)` ? On détaillera dans un tableau donnant les valeurs successives de q et r lors de l'exécution de cette instruction.

2. Montrer que la fonction `Mystere` termine toujours.

3. Que semble faire cette fonction ? En s'appuyant sur un invariant de boucle, justifier la réponse.

4. Écrire une nouvelle fonction `Mystere2(a,b)` qui fait la même chose que la fonction `Mystere`, mais adaptée à un argument `a` qui est un entier relatif, l'argument `b` restant un entier naturel non nul.

5. Écrire une fonction **récursive** `MystereRec(a,b)` ayant le même rôle que la fonction `Mystere2` précédente, mais en ne renvoyant que la valeur de `q`.

Exercice nº 4 — **Palindromes**

Un nombre entier naturel est un palindrome s'il se lit de la même façon dans les deux sens. Par exemple, 12321 et 245542 sont des palindromes, mais pas 12345.

1. Écrire une fonction `miroir(n)` qui prend en argument un entier naturel n et qui renvoie l'entier écrit dans l'autre sens. Par exemple, `miroir(1234)` doit renvoyer le nombre 4321.
2. En déduire une fonction `palindrome(n)` qui prend en argument un entier naturel n et qui renvoie un booléen qui indique si n est un palindrome ou non. Ainsi, `palindrome(12321)` doit renvoyer `True`.
3. Écrire une fonction `ListePalindrome(n)` qui prend en argument un entier naturel n et qui renvoie la liste de tous les palindromes obtenus en faisant un produit de deux nombres à exactement n chiffres.

Exercice nº 5 — **Un peu de cryptographie**

Le but de l'exercice est de s'intéresser à des algorithmes de cryptage et de décryptage de textes. Pour simplifier les différents problèmes posés, on supposera que les messages ne sont constitués que de lettres en minuscules, avec aucun espace ni aucune ponctuation.

Partie I - Traduction d'une chaîne en liste, et vice-versa

Pour effectuer les cryptages, les messages sont transformés en liste de nombres entre 0 et 25 selon la logique suivante : chaque lettre devient une entrée numérique de la liste dans l'ordre d'apparition dans le message, où `"a"` est transformé en 0, `"b"` en 1, etc. jusqu'à `"z"` qui est transformé en 25. Voici un tableau qui pourra servir de référence pour certaines questions :

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Ainsi, `"mot"` est traduit par la liste `[12,14,19]` et la liste `[2,14,3,4]` est la traduction de `"code"`.

On suppose que l'on a validé dans la console la chaîne alphabétique

```
alph="abcdefghijklmnopqrstuvwxyz"
```

1. **a.** Par quelle liste sera représenté le mot `"chat"` ?
   b. Quel est le mot représenté par la liste `[19,0,20,15,4]` ?
2. Écrire une fonction `liste_vers_message(L)` qui prend comme argument une liste `L` d'entiers compris entre 0 et 25, et qui renvoie la chaîne de caractères qui correspond au message qui est représenté par cette liste.
   Par exemple, si `L=[2,14,3,4]`, `liste_vers_message(L)` renvoie la chaîne de caractères `"code"`

3. **a.** Écrire une fonction `lettre_vers_nombre(c)` qui prend en argument un caractère constitué d'une lettre minuscule et qui renvoie le nombre entre 0 et 25 qui lui correspond dans le tableau précédent.

   b. En déduire une fonction `message_vers_liste(s)` qui prend comme argument une chaîne de caractères `s`, et qui renvoie la liste qui correspond à cette chaîne de caractères.

   Par exemple, si `s="mot"`, `message_vers_liste(s)` renvoie la liste `[12,14,19]`

Partie II - Code de César : chiffrement et déchiffrement

Le principe du code de César est le suivant : chaque lettre du message est chiffrée par une lettre à distance fixe. Par exemple, si on choisit une distance de 3, chaque lettre sera remplacée par la lettre située 3 places plus loin : `"a"` devient `"d"`, `"b"` devient `"e"`, etc. Lorsqu'on arrive au bout de l'alphabet, on repart du début, ce qui signifie que `"z"` est chiffré en `"c"` ou encore `"x"` en `"a"`

4. Chiffrer le message `"cesar"` avec une distance de 2.
5. On a reçu le message `"fvyxyw"` en sachant qu'il a été chiffré avec une distance de 4. Quel est le message qui a été chiffré ?
6. Écrire une fonction `chiffre_cesar(message,n)` qui prend comme arguments une chaîne de caractères `message` qui correspond au message à coder et un entier `n` qui correspond à la distance de décalage et qui renvoie le message chiffré en remplaçant chaque lettre par la lettre située à une distance `n`

   Par exemple, `chiffre_cesar("abzx",3)` doit renvoyer `"deca"`.
7. Écrire une fonction `dechiffre_cesar(message,n)` qui prend comme arguments une chaîne de caractères `message` qui correspond au message codé et un entier `n` qui correspond à la distance de décalage et qui renvoie le message déchiffré.
8. On suppose désormais que l'on dispose d'un message chiffré, mais que l'on ne connaît pas le décalage qui a permis de coder le message. Pour tenter de décoder ce message, on adopte une stratégie assez simple : comme on sait que la lettre la plus fréquente en français est la lettre `'e'`, on va d'abord rechercher la lettre qui apparaît le plus dans le message, et on va supposer qu'elle chiffre la lettre `'e'` pour déterminer le décalage.

   a. Écrire une fonction `occurence(L)` qui prend en argument une liste `L` d'entiers compris entre 0 et 25, et qui renvoie une liste `occ` de taille 26 telle que pour tout entier `i` compris entre 0 et 25, `occ[i]` donne le nombre d'occurences du nombre `i` dans la liste `L`

   b. Écrire une fonction `decalage(message)` qui prend en argument une chaîne de caractères `message` qui correspond au message codé, et qui renvoie un entier qui correspond au décalage du chiffrement de César pour ce message, en utilisant la fonction précédente pour déterminer la lettre qui permet de chiffrer `"e"`.

c. En déduire une fonction `dechiffre_cesar_sans_cle(message)` qui prend en argument une chaîne de caractères `message` qui correspond au message chiffré, et qui renvoie une chaîne de caractère qui correspond au message déchiffré.

Partie III - Code de Vigenère

Blaise de Vigenère proposa plus tard une version plus élaborée du chiffrement de César. Le principe est le suivant : les messages sont chiffrés à l'aide d'une clé de plusieurs lettres. La première lettre est décalée par le nombre associé à la première lettre (cf tableau de la partie I), la deuxième lettre est décalée par le nombre associé à la deuxième lettre, etc. Une fois arrivé au bout de la clé, on recommence depuis la première lettre de la clé.

Par exemple, si on choisit comme clé `'mot'`, la première lettre sera décalée de 12 lettres à droite, la deuxième de 14 lettres, la troisième de 19 lettres, puis on recommence avec la 4ème lettre qui sera décalée de 12 lettres à droite, etc.

Voici un autre exemple : on chiffre `'classepreparatoire'` avec la clé `'code'`

c	*l*	*a*	*s*	*s*	*e*	*p*	*r*	*e*	*p*	*a*	*r*	*a*	*t*	*o*	*i*	*r*	*e*
c	*o*	*d*	*e*	*c*	*o*	*d*	*e*	*c*	*o*	*d*	*e*	*c*	*o*	*d*	*e*	*c*	*o*
e	*z*	*d*	*w*	*u*	*s*	*s*	*v*	*g*	*d*	*d*	*v*	*c*	*h*	*r*	*m*	*t*	*s*

La première ligne indique le message à chiffrer, la deuxième ligne indique la lettre de la clé qui est utilisée, et la troisième ligne donne le message chiffré. On voit ainsi que toutes les lettres associées à la lettre c ont été décalées de 2 lettres vers la droite, celles associées à la lettre o ont été décalées de 14 lettres vers la droite, etc.

9. Chiffrer le mot `'informatique'` avec la clé `'banane'`

10. Avec la même clé, déchiffrer le message codé `'caaaait'`

11. Écrire une fonction `chiffrement_vigenere(message,cle)` qui prend en argument un message `message` à chiffrer et une clé `cle`, et qui renvoie le message chiffré à l'aide de la méthode de Vigenère en utilisant la clé `cle`

12. Inversement, écrire une fonction `dechiffrement_vigenere(message,cle)` qui prend en argument un message chiffré `message` et la clé `cle` qui a servi à le chiffrer avec la méthode de Vigenère, et qui renvoie le message déchiffré.

On va supposer désormais que l'on cherche à déchiffrer un message sans connaître la clé. Il faut donc trouver la clé, ce qui se fait en deux temps : on cherche d'abord la longueur de la clé, puis on détermine la clé en utilisant, comme en partie II, le fait que la lettre qui apparaît le plus fréquemment est la lettre `'e'`.

Commençons par déterminer la longueur de la clé. Le principe est le suivant : on regarde dans le message les séquences de 3 lettres successives, et leurs répétitions. L'hypothèse est qu'à chaque fois que la même séquence apparaît, cela correspond aux mêmes lettres codées de la même façon, ce qui veut dire que l'écart entre les deux séquences est un multiple du nombre de lettres de la clé.

Par exemple, si la séquence `'abc'` apparait à l'indice 3 et à l'indice 45, cela veut dire 42 est un multiple du nombre k de lettres de la clé.
En s'intéressant au PGCD de ces différents écarts, on peut alors récupérer le nombre de lettres de la clé.

13. Écrire une fonction `pgcd(a,b)` qui prend en argument deux entiers naturels a et b, et qui renvoie leur PGCD. Il est fortement conseillé de s'appuyer sur l'algorithme d'Euclide.

14. Écrire une fonction `pgcd_distances_repetitions(L,i)` qui prend en argument une liste `L` de longueur `n` d'entiers entre 0 et 25 qui correspond à un message transformé en liste avec la fonction de la question **3.**, et un indice `i` compris entre 0 et `n-3`, et qui renvoie le PGCD de toutes les distances entre les répétitions de la séquence `[L[i],L[i+1],L[i+2]]` dans la suite du texte `[L[i+3],...,L[n-1]]` ou 0 en cas d'absence de répétition.

15. Écrire enfin une fonction `longueur_cle(L)` prenant en argument une liste `L` d'entiers compris entre 0 et 25 correspondant au message chiffré, et qui renvoie la longueur de la clé en s'appuyant sur la question précédente.

Une fois la longueur k de la clé connue, l'idée est simple : les caractères d'indices 0, k, $2k$, etc. sont codés par la même lettre, ensuite, les caractères d'indices 1, $k+1$, $2k+1$, etc. sont codés par la même lettre, etc. En considérant dans chaque cas que la lettre `'e'` sera la plus fréquente, on peut trouver la lettre qui sert à chiffrer `'e'`, puis la lettre qui correspond dans la clé.

16. Écrire une fonction `recherche_cle(L,k)` qui prend en argument une liste `L` de nombres compris entre 0 et 25 qui correspond au message chiffré, et un entier `k` qui donne la longueur de la clé, et qui renvoie la clé de chiffrage.

17. En déduire une fonction `dechiffrage_vigenere(message)` qui prend en argument un message chiffré et qui renvoie le message déchiffré.