DEVOIR SURVEILLÉ N° 1 — **VENDREDI 22 DÉCEMBRE**
Informatique - MPSI 2 — *Durée : 2 h*

- La calculatrice est interdite.
- Seuls sont autorisés 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 :

a. `3**3`
b. `27//5`
c. `6!=2*3 or 2**3==8`
d. `27%4`

2. L'instruction suivante a été validée dans la console : `s="Informatique"`
Déterminer la valeur des expressions suivantes :

a. `s[4]` **b.** `s[15]` **c.** `s[2:5]` **d.** `s[7:]`

3. On a désormais validé dans la console : `L=[1,2,3]` et `M=[4,5,6,7,8]`
Donner la valeur des expressions suivantes :

a. `L+M`
b. `L*3`

Exercice n° 2 — Vu en TP

1. Ecrire le code de la fonction `sommeNbImpairs` qui prend en paramètres les entiers n1 et n2 tels que n1 $\leqslant$ n2 et qui retourne la somme des entiers impairs compris entre n1 et n2 inclus.

2. Ecrire le code de la fonction `minimum` prenant en paramètre une liste non vide d'entiers et renvoyant son minimum. On proposera deux versions : une version itérative et une version récursive.

3. Ecrire le code de la fonction `compter` prenant en paramètre une chaîne de caractères `s` et un caractère `c` et qui renvoie le nombre d'occurrences de `c` dans `s`.

Exercice n° 3 — **Analyse d'algorithmes**

On considère les deux fonctions suivantes.

```
def f(x) :
    for i in range (3) :
        if x<=10 :
            x=x**2+1
        else :
            x=x//3
    return x
```

```
def g(x,b) :
    if x<b :
        return 0
    else :
        return 1+g(x/b,b)
```

Donner le résultat des appels suivants :

1. `f(1)` **2.** `f(3)` **3.** `g(25,100)` **4.** `g(2023,10)`

Exercice n° 4 — **Suite de Syracuse**

Une suite de Syracuse est définie de la façon suivante : on considère un entier naturel non nul comme premier terme u_0, puis pour tout entier naturel n :

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair.} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases}$$

La conjecture de Syracuse consiste à dire que toute suite de Syracuse finit par atteindre la valeur 1. Cette conjecture va être supposée juste (à ce stade, elle n'est toujours pas démontrée ou infirmée).

1. Ecrire le code de la fonction `syracuse` qui prend en argument un entier `u0` naturel non nul, un entier naturel `n`, et qui calcule le terme u_n de la suite de Syracuse.

2. Ecrire le code de la fonction `tempsVol` qui prend en argument un entier naturel non nul `u0` et qui renvoie le premier entier naturel n (appelé temps de vol) tel que $u_n = 1$.

3. Ecrire le code de la fonction `tpsVolMax` qui prend en argument un entier naturel non nul `n`, et qui renvoie le plus petit entier p de $[\![1;n]\!]$ pour lequel la suite de premier terme $u_0 = p$ a le plus grand temps de vol.

Exercice n° 5 — **Diviseurs non triviaux**

Un diviseur non trivial d'un entier naturel n est un diviseur de n différent de 1 et de n.

1. Ecrire le code de la fonction `DivNT` prenant en argument un entier n et renvoyant la liste des diviseurs non triviaux de cet entier.

2. Ecrire le code de la fonction `sommeCarresDivNT` prenant en argument un entier n et renvoyant la somme des carrés des diviseurs non triviaux de cet entier.

3. Ecrire le code de la fonction `ListeEgalite` prenant en argument un entier n et renvoyant la liste de tous les nombres entiers inférieurs ou égaux à n égaux à la somme des carrés de leurs diviseurs non triviaux.

4. `ListeEgalite(1000)` renvoie la liste ci-dessous :

$$[4,9,25,49,121,169,289,361,529,841,961]$$

Quelle conjecture peut-on faire ?

Exercice n° 6 — Codage de listes

On veut représenter une liste de nombres `L` par une autre liste `C` appelée codage de `L`. Les suites consécutives de valeurs identiques de `L` sont représentées dans `C` par deux nombres `r` et `v` où `r` est le nombre de répétitions de la valeur `v` dans une telle suite. Une valeur `v` de `L` qui ne se répète pas est représentée par `1,v`
Par exemple, le codage de `L=[0,0,0,0,5,-2,-2,-2,0,0,0,0,0]` est donné par la liste `C=[4,0,1,5,3,-2,5,0]`.

1. Quel est le codage de `L=[2,2,3,1,1,1,1,-5,0,0]` ? Quelle liste `L` est codée par `C=[2,1,3,-1,4,0,1,3]` ?
2. Ecrire une fonction `decoder` qui prend comme argument une liste `C` constituée d'un nombre pair de valeurs et qui renvoie la liste `L` dont `C` est le codage.
3. Ecrire une fonction `longueurBloc` qui prend comme arguments une liste `L` et un entier `i` et qui renvoie la longueur du plus long bloc de positions consécutives de la liste `L` à partir de la position `i` dont la valeur est `L[i]`.
4. En utilisant la fonction `longueurBloc`, écrire une fonction `codage` qui prend comme argument une liste `L` et qui renvoie le codage de `L`.

Exercice n° 7 — ADN

Dans ce problème, on s'intéresse aux molécules d'ADN. Dans les cellules vivantes, l'ADN se présente sous la forme d'une double hélice composée de deux brins d'ADN. Dans la suite, on manipule des chaînes de caractères représentants des brins d'ADN donc ne contenant que les quatre bases de l'ADN humain i.e. l'adénine, notée A, la cytosine, notée C, la guanine, notée G, et la thymine, notée T. Voici un exemple d'un brin d'ADN : $ACGGTAGCTAGTTTCGACTGGAGGGGTA$.

1. Écrire une fonction `estADN` prenant en entrée une chaîne de caractères `s` représentant un brin d'ADN et renvoyant un booléen indiquant si elle ne contient aucun autre caractère que les quatre bases A, C, G et T. En particulier, elle renvoie `True` si la chaîne est vide.

Les bases A et T sont dites complémentaires, ainsi que les bases G et C.

2. Écrire une fonction `baseComp` prenant en entrée un caractère `base` représentant l'une des quatre bases de l'ADN et renvoyant sa base complémentaire.

Dans la double hélice, les deux brins se font face. L'un des brins est lu de gauche à droite et l'autre de droite à gauche. De plus, les bases complémentaires de l'un et de l'autre se font face. Par exemple, la double hélice peut être formée des deux brins complémentaires inversés suivants : $GTACA$ et $TGTAC$.

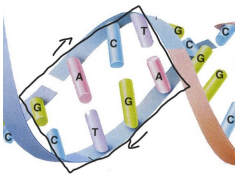

Figure 3 : Double hélice d'ADN : deux brins $GACT$ et $AGTC$.

3. Écrire une fonction `estDoubleHelice` prenant en entrées deux chaînes de caractères `s1` et `s2` représentants deux brins d'ADN et renvoyant un booléen indiquant si ces deux brins forment une double hélice i.e. si `s2` est le brin complémentaire inversé de `s1`. Par exemple, l'appel `estDoubleHelice("GTACA","TGTAC")` renvoie `True`. L'utilisation du test d'égalité sur deux chaînes de caractères est ici interdit.
4. Écrire une fonction `seqCompInv` prenant en entrée une chaîne de caractères `s` représentant un brin d'ADN et renvoyant son brin d'ADN complémentaire inversé. Par exemple, `seqCompInv("GTACA")` renvoie `"TGTAC"`.

Le code génétique est écrit à partir des quatre bases de l'ARN obtenu à partir d'une chaîne représentant un brin d'ADN en remplaçant toutes les bases T par des bases U, les autres bases restant inchangées.

5. Écrire une fonction `transcrit` prenant en entrée une chaîne de caractères `s` représentant un brin d'ADN et renvoyant le brin d'ARN correspondant.
   Par exemple, `transcrit("ACGGTAGCTAGTTTCGACTGGAGGGGTA")` renvoie la chaîne `"ACGGUAGCUAGUUUCGACUGGAGGGGUA"`.

Un brin d'ARN est constitué de suites de trois bases appelées codons comme par exemples : ACG, GUA, CGC, GCU … La lecture des codons s'effectue triplet par triplet. Ainsi le brin $GCUACGGAGCUUCGGAGCACGUAG$ est composé des codons GCU, ACG, GAG, CUU, CGG, AGC, ACG et UAG.

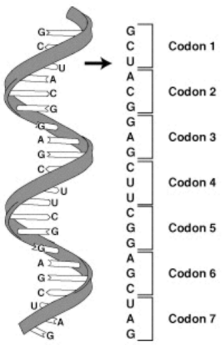

Figure 4 : Brin d'ARN.

6. Écrire une fonction `OccurrencesCodon` prenant en entrée une chaîne de caractères `s` représentant un brin d'ADN et un codon `cod`. Elle renvoie le nombre d'occurrences de `cod` dans le brin d'ARN correspondant au brin `s`. Par exemple, l'appel `OccurrencesCodon("GCTACGGAGCTTCGGAGCACGTAG","ACG")` renvoie 2 et l'appel `OccurrencesCodon("AGTTCGACT","UUC")` renvoie 0.

Les codons constituant l'ARN codent des acides aminés permettant de synthétiser des protéines. Pour synthétiser une protéine, il faut donc lire un brin d'ARN codon par

codon, identifier les acides aminés représentés par ces codons et les assembler afin de former une protéine.
On distingue vingt acides aminés servant de base pour la construction des protéines et qui sont communs à tous les êtres vivants. Chacun de ces vingt acides aminés peut être codé par différents codons. Par exemple, la cystéine est codée par les codons UGU et UGC et l'alanine est codée par les codons GCU, GCC, GCA et GCG. En particulier, un acide aminé est codé par au moins un codon mais deux acides aminés différents ne sont pas nécessairement codés par le même nombre de codons. En revanche, un codon ne peut coder qu'un unique acide aminé. Par exemple, le codon GCU représente uniquement l'alanine.
De plus, le codage d'une protéine sur un brin d'acide aminé termine toujours par un codon spécifique appelé codon "Stop" signifiant que la protéine a complètement été synthétisée. On distingue trois codons différents codant le codon "Stop" : UAA, UAG et UGA.
Ainsi le brin d'ARN $GCU\ ACG\ GAG\ CUU\ CGG\ AGC\ ACG\ UAG$ code la protéine composée d'une alanine (GCU), d'une thréonine (ACG), d'une glutamine (GAG), d'une leucine (CUU), d'une arginine (CGG), d'une sérine (AGC) et d'une thréonine (ACG), le codon UAG étant un codon "Stop".

Dans la suite, on dispose d'une liste `acides_amines` de taille 21 contenant la chaîne "Stop" et les chaînes de caractères représentants les noms des vingt acides aminés. Ainsi `acides_amines[0]="Stop"` et pour $1 \leqslant i < 21$, `acides_amines[i]` est égal au nom du i^{e} acide aminé. A partir de maintenant, on assimilera le codon "Stop" à un acide aminé.
On suppose également disposer d'une liste `codons` de taille 21 telle que pour tout entier $0 \leqslant i < 21$, `codons[i]` soit la liste de tous les codons, donnés sous forme de chaînes de caractères, codant l'acide aminé `acides_amines[i]`.

7. Écrire une fonction `aPourCodons` prenant en entrées la liste `acides_amines` des acides aminés, la liste de listes `codons` des codons représentant les acides aminés et une chaîne de caractères `a` représentant un acide aminé présent dans la liste `acides_amines`. Elle renvoie la liste des codons codant cet acide aminé `a`.

8. Écrire une fonction `codeAcideAmine` prenant en entrées la liste `acides_amines` des acides aminés, la liste de listes `codons` des codons représentant les acides aminés et une chaîne de caractères `cod` de taille 3 représentant un codon présent dans `codons`. Elle renvoie l'acide aminé codé par ce codon `cod`.

9. Écrire une fonction `proteine` prenant en entrées la liste `acides_amines` des acides aminés, la liste de listes `codons` des codons représentant les acides aminés et une chaîne de caractères `s` représentant un brin d'ADN. Elle renvoie la liste des acides aminés composant la protéine pouvant être synthétisée à partir du brin d'ARN correspondant à `s` si cette protéine est terminée et le booléen `False` sinon.