DEVOIR SURVEILLÉ N° 2 **CORRECTION**
Informatique - MPSI 2

Exercice n° 1

1. **a.** Cela correspond à $\boxed{2^5 = 32.}$
   b. Cela correspond au quotient de la division euclidienne de 38 par 7, donc il s'agit de $\boxed{5.}$
   c. Le premier booléen est associé à $10 = 2 \times 5$, qui est vrai, le second à la négation de $3^2 \neq 9$ qui est faux, et donc la négation est vraie. Avec le connecteur et, la valeur est donc $\boxed{\textbf{True}.}$
   d. Il s'agit du reste de la division euclidienne de 38 par 5 : il s'agit donc de $\boxed{3.}$
2. **a.** $\boxed{\text{'m'}}$
   b. $\boxed{\text{On aura un message d'erreur, car la chaîne est trop courte.}}$
   c. $\boxed{\text{'rogram'}}$
   d. $\boxed{\text{'Progr'}}$
3. **a.** Il faut concaténer les deux listes : $\boxed{[1,2,3,4,5,6,7,8,9,0].}$
   b. Il faut concaténer la liste L par elle-même 2 fois : $\boxed{[1,2,3,4,5,1,2,3,4,5].}$

Exercice n° 2

1.
```
def maximum(L) :
    n=len(L) #n est la longueur de la liste
    M=L[0] # De façon temporaire, le maximum de la liste est
     l'élément d'indice 0
    for i in range (1,n) :
        if L[i]>M :
            M=L[i] #Si le terme d'indice i est plus grand que M, c'est
             le nouveau max
    return M #A la fin de la boucle, M est le maximum de la liste.
```

2.

```
def maximum_rec(L) :
    if len(L)==1 :
        return L[0] # Si la liste a un seul élément, c'est le max
    else :
        M=maximum_rec(L[1:]) #On calcule le max de la liste sans le
         premier terme
        if L[0]<M :
            return M #On compare le premier terme à M, si M est plus
             grand, c'est le max
        else :
            return L[0] #sinon c'est le premier terme
```

3.

```
def posmaximum(L) :
    n=len(L) #n est la longueur de la liste
    M,pos=L[0],0 # De façon temporaire, le maximum de la liste est
     l'élément d'indice 0 et sa position est 0
    for i in range (1,n) :
        if L[i]>M :
            M,pos=L[i],i #Si le terme d'indice i est plus grand que M,
             c'est le nouveau max et i la nouvelle position
        return i #A la fin de la boucle, i est la position du maximum
         de la liste.
```

4.

```
def tteposmaximum(L) :
    n=len(L) #n est la longueur de la liste
    M=L[0] # De façon temporaire, le maximum de la liste est
     l'élément d'indice 0
    S=[0] # De façon temporaire, la liste des positions est
     constitué du nombre 0
    for i in range (1,n) :
        if L[i]>M :
            M=L[i] #Si le terme d'indice i est plus grand que M, c'est
             le nouveau max
            S=[i] #La liste des positions est constitué de i uniquement

        elif L[i]==M :
            S.append(i) #Si le terme d'indice i est égal à M, on ajoute
             i à la liste des indices
    return S #A la fin de la boucle, S est la liste des positions du
     maximum de la liste.
```

Exercice n° 3

1.
```
def divise(a,b) :
    if a==2 :
        return b%a==0 #2 est le seul entier à tester si a vaut 2
    return b%a==0 or divise(a-1,b) #on teste si a divise b, ou alors
    on teste la divisibilité de b par les entiers inférieurs à a-1.
```

2.
```
def premier(n) :
    return not(divise(n-1,n))
```

Remarque : On prouve en arithmétique que l'on peut arrêter les vérifications de divisibilité à $\lfloor\sqrt{n}\rfloor$.

3.
```
def liste_premiers(n) :
    L=[] #On créé la liste qui contiendra les nombres premiers
    for i in range (2,n+1) :
        if premier(i) :
            L.append(i) #Si i est premier, on l'ajoute à la liste
    return L #A la fin de la boucle, L contient tous les nombres
     premiers jusqu'à n.
```

Exercice n° 4

1.
```
def compte_caractere(t) :
    n=len(t) #n est la longueur de la chaine
    c=0 # Cette variable va servir à compter les caractères
    for i in range (n) :
        if t[i]!=' ' :
            c=c+1 #Si le caractère n'est pas un espace, on le compte
    return c #A la fin de la boucle, c est le nombre de caractères.
```

2.
```
def compte_mot(t) :
    n=len(t) #n est la longueur de la chaine
    c=0 # Cette variable va servir à compter les mots
    for i in range (n) :
        if t[i]==' ' :
            c=c+1 #L'espace annonce la fin d'un mot, on compte alors ce
             mot
        elif i==n-1 :
            c=c+1 #Le dernier caractère annonce le dernier mot, que
             l'on compte également
    return c #A la fin de la boucle, c est le nombre de mots.
```

3.
```
def recherche_index_lettre(t,l) :
    n=len(t) #n est la longueur de la chaine
    for i in range (n) :
        if t[i]==l :
            return i #Si le caractère est l, on renvoie l'indice
    return-1 #A la fin de la boucle, l n'a pas été trouvé, on
     renvoie -1
```

4.
```
def recherche_mot_plus_long(t) :
    n=len(t) #n est la longueur de la chaine
    i,longueur,j,k=0,0,0,0 #i est l'indice de départ d'un mot,
     longueur la longueur du mot le plus long, j sera l'indice de
     fin d'un mot et k l'indice déterminé par la fonction
     précédente.
    while k!=-1 and i<n :
        k=recherche_index_lettre(t[i:],' ') #On cherche le premier
         espace à partir de l'indice i, on affecte à k son indice
        j=i+k #On calcule l'indice de fin du mot en sommant l'indice
         de début et l'index de l'espace suivant
        if j-i>longueur :
            longueur=j-i # j-i est la longueur du mot, on la compare à
             celle du mot le plus long
            mot=t[i:j] #On affecte à mot le nouveau mot le plus long
        i=j+1 # Le nouvel indice de départ est situé après l'espace,
         soit en j+1
    return mot #A la fin de la boucle, mot est le mot le plus long
```

Exercice n° 5

1. Proposons le tableau suivant, qui donne les valeurs de a, b et t au début, puis après chaque itération de la boucle.

a	b	t
57	4	0
$57//2 = 28$	$4 \times 2 = 8$	$0 + 4 = 4$
$28//2 = 14$	$8 \times 2 = 16$	4
$14//2 = 7$	$16 \times 2 = 32$	4
$7//2 = 3$	$32 \times 2 = 64$	$4 + 32 = 36$
$3//2 = 1$	$64 \times 2 = 128$	$36 + 64 = 100$
$1//2 = 0$	$128 \times 2 = 256$	$100 + 128 = 228$

2. Utilisons le variant de boucle a : par définition, a est un entier naturel par condition sur l'argument, et le fait que a est le quotient d'une division euclidienne entre

deux entiers naturels. De plus, la suite des valeurs de a est strictement décroissante tant que a est non nul donc elle atteindra 0 : la fonction terminera toujours.

3. Montrons que le terme $a \times b + t$ est un invariant de boucle, toujours égal au produit P des deux arguments.

Avant d'initier la boucle, a et b sont les deux arguments, et t est nul donc on a l'égalité $a \times b + t = a \times b = P$.

Supposons à présent qu'avant une étape de la boucle, $a \times b + t = P$.

On a alors deux cas de figure :

1er cas : a est pair. Dans ce cas, a prend la valeur $a/2$, b prend la valeur $2b$ et t n'est pas modifié. On a alors :

$$a/2 \times 2b + t = a \times b + t = P$$

2eme cas : a est impair. Dans ce cas, a prend la valeur $(a-1)/2$, b prend la valeur $2b$ et t prend la valeur $t+b$. On a alors :

$$(a-1)/2 \times 2b + (t+b) = (a-1) \times b + t + b = a \times b + t = P$$

Dans tous les cas, on retrouve que $a \times b + t$ reste égal à P après une itération de la boucle, ce qui prouve bien que $a \times b + t$ est un invariant de boucle.

Enfin, la boucle s'achève quand $a = 0$. Dans ce cas, l'invariant de boucle est égal à $0 \times b + t = t$, ce qui montre que t est égal à la valeur du produit des arguments à la fin des itérations de la boucle.

4. Posons $n = \lfloor \log_2 a \rfloor$: dans ce cas, $2^n \leqslant a < 2^{n+1}$. Lorsqu'on appelle la fonction, on effectue d'abord une affectation, puis on itère la boucle : on effectue alors 1 test, puis 2 à 3 opérations. La valeur de a respecte alors l'inégalité $2^{n-1} \leqslant a < 2^n$, sauf dans le cas où $n = 0$, qui fait que a passe de 1 à 0 (et la boucle termine). On peut dire que la complexité de cette fonction est en $\mathcal{O}(\log_2 a)$.

Exercice n° 6

Partie I

1.

```
def somme_ligne(M) :
    n=len(M) #n est la longueur de la liste
    s=0 #on initialise s, qui va servir à calculer la somme
    for j in range (n) :
        s=s+M[i][j] #On ajoute à s le terme d'indice j de la ligne
         d'indice i
    return s #A la fin de la boucle, s est la somme recherchée.
```

2.
```
def somme_colonne(M) :
    n=len(M) #n est la longueur de la liste
    s=0 #on initialise s, qui va servir à calculer la somme
    for j in range (n) :
        s=s+M[j][i] #On ajoute à s le terme d'indice i de la ligne
         d'indice j
    return s #A la fin de la boucle, s est la somme recherchée.
```

3.
```
def somme_diag1(M) :
    n=len(M) #n est la longueur de la liste
    s=0 #on initialise s, qui va servir à calculer la somme
    for j in range (n) :
        s=s+M[i][i] #On ajoute à s le terme d'indice i de la ligne
         d'indice i, sur la diagonale
    return s #A la fin de la boucle, s est la somme recherchée.
```

4.
```
def somme_diag2(M) :
    n=len(M) #n est la longueur de la liste
    s=0 #on initialise s, qui va servir à calculer la somme
    for j in range (n) :
        s=s+M[i][n-1-i] #On ajoute à s le terme d'indice i de la ligne
         d'indice n-1-i, sur l'antidiagonale
    return s #A la fin de la boucle, s est la somme recherchée.
```

Partie II

5.
```
def carre_magique(M) :
    n=len(M) #n est la longueur de la liste
    s=somme_diag1(M) #on fixe s comme étant la somme de la diagonale
    if somme_diag2(M)!=s :
        return False #si la somme de l'antidiagonale n'est pas s, le
         carré n'est pas magique
    for i in range (n) :
        if somme_ligne(M,i)!=s or somme_colonne(M,i)!=s :
            return False #si la somme de la ligne ou la colonne d'indice
             i n'est pas s, le carré n'est pas magique
    return True #Si on arrive à la fin de la boucle, le carré est
     magique.
```

Partie III

6.
```
def estPresent(M,e) :
    n=len(M) #n est la longueur de la liste
    for i in range (n) :
        for j in range (n) :
            if M[i][j]==e :
                return True #Si le terme d'indice j de la liste i est
                 l'élement e, e est bien présent
    return False #A la fin de la boucle, e n'a pas été trouvé, et
     n'est pas un élément de la liste.
```

7.
```
def magique_normal(M) :
    n=len(M) #n est la longueur de la liste
    if not(carre_magique(M)) :
        return False #si le carré n'est pas magique, alors il n'est
         pas magique normal.
    for i in range (1,n**2+1) :
        if not (estPresent(M,i)) :
            return False #on fait varier i de 1 à n**2, si i n'est pas
             présent, le carré n'est pas magique normal.
    return True #Si on arrive à la fin de la boucle, le carré est
     magique normal.
```