Exercice n° 1

1. **a.** Cela correspond à $\boxed{5^3 = 125.}$
   b. Cela correspond au quotient de la division euclidienne de 47 par 8, donc il s'agit de $\boxed{5.}$
   c. Le premier booléen est associé à $25 = 2 \times 5$, qui est faux, le second à la négation de $3 \times 2 \neq 6$ qui est faux, et donc la négation est vraie. Avec le connecteur ou, la valeur est donc $\boxed{\textbf{True}.}$
   d. Il s'agit du reste de la division euclidienne de 47 par 8 : il s'agit donc de $\boxed{7.}$
2. **a.** $\boxed{13}$
   b. $\boxed{[5,7]}$
   c. $\boxed{[9,11,13,15]}$
   d. $\boxed{[1,3,5,7]}$
   e. $\boxed{15}$
   f. $\boxed{\text{La liste contient 8 éléments, donc } L[8] \text{ renverra un message d'erreur.}}$
3. **a.** $\boxed{\text{"Hello World!"}}$
   b. $\boxed{\text{"HelloHelloHello"}}$

Exercice n° 2

1. Regardons étape par étape.
   - L'élément minimal est 1, échangé avec 3, d'où la liste `L=[1,4,3,5,2]`
   - L'élément minimal à partir du 2ème indice est 2, échangé avec 4, d'où la liste `L=[1,2,3,5,4]`
   - L'élément minimal à partir du 3ème indice est bien placé, on ne change rien.
   - L'élément minimal à partir du 4ème indice est 4, échangé avec 5, d'où la liste `L=[1,2,3,4,5]`
   - La liste est désormais triée.

2.
```
def IndiceMinimum(L,i) :
    n=len(L) #n est la longueur de la liste
    if i>=n :
        return False # On renvoie False si l'indice i est trop grand
    m,ind=L[i],i # De façon temporaire, le minimum de la liste est
     l'élément d'indice i, ind est l'indice i
    for k in range (i,n) :
        if L[k]<m :
            m,ind=L[k],k #Si le terme d'indice k est plus petit que m,
             c'est le nouveau min, et k est le nouvel indice du min
    return ind #A la fin de la boucle, ind est l'indice du minimum.
```

3.
```
def tri_selection(L) :
    n=len(L) #n est la longueur de la liste
    for i in range (n) :
        k=IndiceMinimum(L,i) # On détermine l'indice du minimum à
         partir de i
        L[i],L[k]=L[k],L[i] # On échange le minimum avec le terme
         d'indice i
```

4. La complexité de la fonction `IndiceMinimum` étant $\mathcal{O}(n)$, en rajouant la boucle de la fonction du tri par sélection, on en déduit que la complexité est en $\mathcal{O}(n^2)$.

Exercice n° 3

1. Proposons le tableau suivant, qui donne les valeurs de q et r, puis après chaque itération de la boucle.

q	r
0	23
1	17
2	11
3	5

On en déduit que `Mystere(23,6)` renvoie le tuple `(3,5)`

2. Utilisons le variant de boucle r : au début de la boucle, r est un entier naturel car il est égal à a, puis à chaque itération de la boucle, il diminue de l'entier naturel b donc r reste un entier. De plus, b est non nul, donc r décroit strictement à chaque itération de la boucle : r finira donc par devenir strictement inférieur à b.

3. Considérons l'invariant de boucle Inv_n : « Après n itérations de la boucle, le terme $b \times q + r$ est égal à l'entier a donné en argument. »

Inv_0 : Avant d'initier la boucle, q est nul et r vaut a donc on a l'égalité $b \times q + r = 0 + a = a$.

Soit $n \in \mathbb{N}$ et supposons que Inv_n est vérifié. On a donc, après n itérations de la boucle, $b \times q + r = a$.

Il y a alors deux cas à considérer.

1er cas : $r \geqslant b$.

Dans ce cas, q prend comme nouvelle valeur $q+1$, et r prend comme nouvelle valeur $r-b$.

On a alors $b \times (q+1) + (r-b) = b \times q + b + r - b = b \times q + r + b - b = b \times q + r = a$.

On en déduit que Inv_{n+1} est bien vérifié.

2ème cas : $r < b$.

La boucle s'achève quand $r < b$. Dans ce cas, $a = bq + r$ avec r strictement inférieur à b. De plus, à la dernière itération de la boucle, la valeur de r était supérieure ou égale à b, donc la nouvelle valeur $r - b$ est positive, ce qui prouve que la dernière valeur de r vérifie $0 \leqslant r < b$.

Ainsi, $b \times q + r$ est bien un invariant de boucle, et vaut toujours a, et la boucle s'arrête quand $0 \leqslant r < b$: par unicité de la division euclidienne, on en déduit que (q, r) est le couple quotient/reste de la division euclidienne.

4. Lorsque r est strictement négatif, il faut augmenter la valeur de r de b tout en diminuant la valeur de q de -1 pour obtenir le couple quotient/reste de la division euclidienne, ce qui donne la fonction suivante :

```
def Mystere2(a,b) :
    q = 0
    r = a
    if r >= 0 :
        while r >= b :
            q += 1
            r = r - b
    else :
        while r < 0 :
            q -= 1
            r = r + b
    return (q,r)
```

5.

```
def MystereRec(a,b) :
    if 0 <= a < b :
        return 0
    if a >=b :
        return 1+MystereRec(a-b,b)
    else :
        return-1+MystereRec(a+b,b)
```

Exercice n° 4

1.
```
def miroir(n) :
    out=0 # On initialise un entier
    while n!=0 :
        p=n%10 # On détermine le chiffre des unités de n
        out=out*10+p # On multiplie par 10 pour décaler à gauche les
         décimales de out, et on ajoute p en unités
        n=n//10 # On attribue à n le quotient de la DE par 10, ce qui
         revient à retirer le chiffre des unités
    return out
```

2.
```
def palindrome(n) :
    return n==miroir(n)
```

3.
```
def ListePalindrome(n) :
    L=[] #On créé la liste qui contiendra les palindromes
    for i in range (10**(n-1),10**n-1) :
        for j in range (10**(n-1),10**n-1) :
            if palindrome(i*j) #On teste si le produit ij est un
             palindrome :
                L.append(i*j) #Dans ce cas, on l'ajoute à la liste
    return L
```

Exercice n° 5

Partie I

1. a. "chat" sera représenté par la liste [2,7,0,19]

   b. La liste représente le mot "taupe"

2.
```
def liste_vers_message(L) :
    n=len(L) #n est la longueur de la liste
    out=' ' On initialise une chaine de caractères vide
    for i in range (n) :
        out=out+alph[L[i]] #On ajoute au bout de la chaîne out le
         caractère qui correpond au nombre i
    return out #A la fin de la boucle, out est la chaîne de
     caractères associée à la liste.
```

3. a.
```
def lettre_vers_nombre(c) :
    for i in range (25) :
        if c==alph[i] :
            return i #Si le caractère c coïncide avec celui d'alph
             d'indice i, il faut renvoyer i
```

b.

```
def message_vers_liste(s) :
    n=len(s) #n est la longueur de la chaîne de caractères
    out=[] On initialise une liste vide
    for i in range (n) :
        out.append(lettre_vers_nombre(s[i])) #On ajoute à la liste
         le nombre entre 0 et 25 associé au caractère s[i]
    return out #A la fin de la boucle, out est la liste associée
     à la chaîne de caractères.
```

Partie II

4. Le message devient après chiffage "eguct"

5. Le message, une fois déchiffré, est "brutus"

6.

```
def chiffre_cesar(message,n) :
    L=message_vers_liste(message) #On transforme le message en liste
     de nombres
    p=len(L) #p est la longueur de la liste
    for i in range (n) :
        L[i]=(L[i]+n)%26 #On ajoute à chaque nombre de la liste
         l'entier n, et on donne le reste de la division euclidienne
         avec 26 pour avoir un entier entre 0 et 25
    return liste_vers_message(L) #A la fin de la boucle, la liste
     est chiffrée, et on renvoie la liste traduite en chaîne de
     caractères
```

7.

```
def dechiffre_cesar(message,n) :
    return chiffre_cesar(message,-n) #Déchiffrer revient à retirer
     le décalage au lieu de l'ajouter : on peut réutiliser la
     fonction précédente.
```

8. a.

```
def occurence(L) :
    occ=[0 for i  in range (26)] #on initialise une liste de 26
     zéros
    n=len(L) #L est la longueur de la liste
    for i in range (n) :
        occ[L[i]]+=1 #on ajoute 1 au nombre d'éléments L[i] déjà
         comptés
    return occ #Si on arrive à la fin de la boucle, occ donne le
     nombre d'occurences de chaque entier entre 0 et 25.
```

b.

```
def decalage(message) :
    L=message_vers_liste(message) #on transforme le message en
     une liste L de nombres
    occ=occurence(L) #on détermine les occurences des nombres
    M,ind=occ[0],0 #On fixe temporairement le maximum à l'indice
     0
    for i in range (1,26) :
        if occ[i]>M :
            M,ind=occ[i],i
        #Si le terme d'indice i est plus grand que le maximum,
         c'est le nouveau maximum
    return (ind-4)%26 #ind est l'indice qui correspond à la
     lettre e, dont l'indice est 4 : le décalage est la
     différence modulo 26
```

c.

```
def dechiffre_cesar_sans_clé(message) :
    p=decalage(message) #On calcule la clé de chiffrage
    return dechiffre_cesar(message,p) #On utilise la fonction qui
     déchiffre avec la clé.
```

Partie III

9. On obtient après chiffrement : "jnsoeqbtvqhi"

10. Le message déchiffré est "bananes"

11.

```
def chiffrement_vigenere(message,cle) :
    M=message_vers_liste(message) #On traduit le message en une
     liste de nombres
    C=message_vers_liste(cle) #Même chose pour la clé
    n,p=len(M),len(C) #On note n et p les longueurs des listes M et
     C
    for i in range (n) :
        M[i]=(M[i]+C[i%p])%26 #On change les valeurs de la liste
         message en ajoutant les valeurs de la clé
    messagechiffre=liste_vers_message(M) #On retraduit la liste en
     message
    return messagechiffre
```

12.

```
def dechiffrement_vigenere(message,cle) :
    M=message_vers_liste(message) #On traduit le message en une liste de nombres
    C=message_vers_liste(cle) #Même chose pour la clé
    n,p=len(M),len(C) #On note n et p les longueurs des listes M et C
    for i in range (n) :
        M[i]=(M[i]-C[i%p])%26 #On change les valeurs de la liste message en retirant les valeurs de la clé
    messagedechiffre=liste_vers_message(M) #On retraduit la liste en message
    return messagedechiffre
```

13.

```
def pgcd(a,b) :
    while b!=0 :
        a,b=b,a%b #Tant que le reste est non nul, on remplace a et b par b et le reste de la DE de a par b
    return a #A la fin de la boucle, a est le dernier reste non nul, donc le pgcd.
```

14.

```
def pgcd_distances_repetition(L,i) :
    out,ind=0,i #On initialise le pgcd à 0 et l'indice de référence
    n=len(L) # n est la longueur de la liste
    for k in range (i,n-3) :
        if L[i:i+3]==L[k:k+3] :
            out,ind=pgcd(out,k-ind),k #Par associativité, le pgcd est le pgcd de out (pgcd temporaire) et k-ind, nombre de lettres entre les deux paquets identiques
    return out
```

15.

```
def longueur_cle(L) :
    n=len(L)
    out=0 #On initialise la longueur de la clé à 0
    for i in range (n-2) :
        out=pgcd(out,pgcd_distances_repetition(L,i)) #On détermine le pgcd des répétitions de la chaine L[i:i+2], et le nouveau pgcd est le pgcd de ce nombre avec out(pgcd temporaire)
    return out
```

16.

```
def recherche_cle(L,k) :
    cle='' #On initialise la clé à une chaîne vide
    M=[[] for i in range(k)] #On initialise k listes vides
    n=len(L)
    for i in range (n) :
        M[i%k].append(L[i]) #On répartit les caractères du message
         dans les différentes sous-listes
    for j in range (k) :
        p=decalage(liste_vers_message(M[j])) #On détermine le décalage
         pour la sous-liste M[j]
        cle=cle+alph[p] #On ajoute le caractère correspondant au
         décalage pour obtenir la clé
    return cle
```

17.

```
def dechiffrage_vigenere(message) :
    L=message_vers_liste(message) #On traduit le message en une
     liste de nombres
    k=longueur_cle(L) #On détermine la longueur de la clé
    cle=recherche_cle(L,k) #On détermine la clé
    return dechiffrement_vigenere(message,clé)
```