Informatique en CPGE (2017-2018)
Corrigé exercices : les listes

Exercice 1

[2,3,4,5,6,7,8,9][2 :6] est la sous-liste [4,5,6,7] et [4,5,6,7][2] est l'élément d'indice 2 de cette sous-liste soit 6 qui est affiché.

Exercice 2

Le code affiche la liste [2, 2, 6, 4, 10, 6, 14, 8, 18].

Pour chaque entier i de 1 à 9, on l'ajoute à la liste **a** s'il est pair ou on ajoute son double à la liste **a** s'il est impair.

Exercice 3

Version 1 (où on utilise les indices des éléments de L)

```
def f(L):
    pairs=[]
    impairs=[]
    for i in range(len(L)):
        if L[i]%2==0:
            pairs.append(L[i])
        else:
            impairs.append(L[i])
    return pairs,impairs
```

Version 2 (où on n'utilise pas les indices des éléments de L puisqu'on n'en a pas besoin)

```
def f(L):
    pairs=[]
    impairs=[]
    for n in L:
        if n%2==0:
            pairs.append(n)
        else:
            impairs.append(n)
    return pairs,impairs
```

Version 3 (avec des listes définies en compréhension)

```
def f(L):
    pairs=[n for n in L if n%2==0]
    impairs=[n for n in L if n%2==1]
    return pairs,impairs
```

Remarque : il est possible de remplacer l'instruction `if n%2==0` par `if not n%2` et l'instruction `if n%2==1` par `if n%2`.

On obtient alors pour les versions 2 et 3 :

```
def f(L):
    pairs=[]
    impairs=[]
    for n in L:
        if n%2:
            impairs.append(n)
        else:
            pairs.append(n)
    return pairs,impairs
```

```
def f(L):
    pairs=[n for n in L if not n%2]
    impairs=[n for n in L if n%2]
    return pairs,impairs
```

Exercice 4
Il suffit de modifier le programme de l'exercice 3.

Version 1

```
def f(L):
    moins6=[]
    plus6=[]
    for i in range(len(L)):
        if len(L[i])<6:
            moins6.append(L[i])
        else:
            plus6.append(L[i])
    return moins6,plus6
```

Version 2 (où on n'utilise pas l'indice des éléments de L puisqu'on n'en a pas besoin)

```
def f(L):
    moins6=[]
    plus6=[]
    for mot in L:
        if len(mot)<6:
            moins6.append(mot)
        else:
            plus6.append(mot)
    return moins6,plus6
```

Version 3 (avec des listes définies en compréhension)

```
def f(L):
    moins6=[mot for mot in L if len(mot)<6]
    plus6=[mot for mot in L if len(mot)>=6]
    return moins6,plus6
```

Exercice 5

1. La fonction ne renvoie rien et affiche les mots un par un.
   Version 1

```
def ordre1(liste):
    while len(liste)>0:
        nblettres=len(liste[0])
        ind=0
        for i in range(len(liste)):
            if len(liste[i])<nblettres:
                nblettres=len(liste[i])
                ind=i
        print(liste[ind])
        liste.remove(liste[ind])

maliste=['toto','bonjour','a','oui','non']
ordre1(maliste)
```

   Version 2

```
def ordre1(liste):
    while len(liste)>0:
        mini=liste[0]
        nblettres=len(mini)
        for mot in liste:
            if len(mot)<nblettres:
                nblettres=len(mot)
                mini=mot
        print(mini)
        liste.remove(mini)

maliste=['toto','bonjour','a','oui','non']
ordre1(maliste)
```

2. On modifie le programme précédent : les mots ne sont plus affichés un à un mais stockés dans une liste qui est renvoyée à la fin.

```
def ordre2(liste):
    liste_ord=[]
    while len(liste)>0:
        mini=liste[0]
        nblettres=len(mini)
        for mot in liste:
            if len(mot)<nblettres:
                nblettres=len(mot)
                mini=mot
        liste_ord.append(mini)
        liste.remove(mini)
    return liste_ord

maliste=['toto','bonjour','a','oui','non']
print(ordre2(maliste))
```

Exercice 6

1. (a) `[1,2,3]+[4,5,6]=[1,2,3,4,5,6]`

1. (b) `2*[1,2,3]=[1,2,3,1,2,3]`

2.

```
def smul(n,liste):
    return [n*x for x in liste]

# ou bien
def smul(n,liste):
    return [n*liste[i] for i in range(len(liste))]

#ou bien
def smul(n,liste):
    prod_liste=[]
    for i in range(len(liste)):
        prod_liste.append(n*liste[i])
    return prod_liste
```

3.

```
def vsom1(liste1,liste2):
    return [liste1[i]+liste2[i] for i in range(len(liste1))]

#ou bien
def vsom1(liste1,liste2):
    som_liste=[]
    for i in range(len(liste1)):
        som_liste.append(liste1[i]+liste2[i])
    return som_liste
```

4.

```
def vdif(liste1,liste2):
    return [liste1[i]-liste2[i] for i in range(len(liste1))]
```

5.
Version 1

```
def vsom2(liste1,liste2):
    n1,n2=len(liste1),len(liste2)
    n=min(n1,n2)
    som_liste=[liste1[i]+liste2[i] for i in range(n)]
    for i in range(n,n1):
        som_liste.append(liste1[i])
    for i in range(n,n2):
        som_liste.append(liste2[i])
    return som_liste
```

Version 2

```
def vsom2(liste1,liste2):
    n1,n2=len(liste1),len(liste2)
    n=min(n1,n2)
    som_liste=[liste1[i]+liste2[i] for i in range(n)]
    if n<n1:
        som_liste+=liste1[n:n1]
    else:
        som_liste+=liste2[n:n2]
    return som_liste
```

Version 3

```
def vsom2(liste1,liste2):
    n1,n2=len(liste1),len(liste2)
    n=min(n1,n2)
    som_liste=[liste1[i]+liste2[i] for i in range(n)]
    som_liste+=liste1[n:n1]
    som_liste+=liste2[n:n2]
    return som_liste
```

Version 4 (en utilisant la fonction `vsom1`)

```
def vsom2(liste1,liste2):
    n1,n2=len(liste1),len(liste2)
    if n1==n2:
        return vsom1(liste1,liste2)
    else:
        if n2<n1:
            liste2=liste2+(n1-n2)*[0]
        else:
            liste1=liste1+(n2-n1)*[0]
    return vsom1(liste1,liste2)
```

Attention : dans toutes ces fonctions les listes passées en paramètres ne sont pas modifiées, mais si, par exemple dans la dernière fonction, on remplace l'instruction `liste2=liste2+(n1-n2)*[0]` par l'instruction `liste2+=(n1-n2)*[0]`, la fonction renvoie un résultat correct mais liste2 est modifiée !

Exercice 7

1.

```
def f1(liste):
    ch=''
    for n in liste:
        ch+=str(n)
    return ch
```

2.

```
def f2(chaine):
    liste=[]
```

```
    for ch in chaine:
        liste.append(int(ch))
    return liste

# ou bien
def f2(chaine):
    return [int(ch) for ch in chaine]
```

3.

```
def f3(liste):
    listeB=[]
    for n in liste:
        b=False
        if n==1:
            b=True
        listeB.append(b)
    return liste

# ou bien
def f3(liste):
    listeB=[]
    for n in liste:
        listeB.append(bool(n))
    return liste

# ou bien
def f3(liste):
    return [bool(n) for n in liste]
```

4.

```
def f4(listeB):
    liste=[]
    for b in listeB:
        n=0
        if b:
            n=1
        listeB.append(n)
    return liste

# ou bien
def f4(listeB):
    liste=[]
    for b in liste:
        listeB.append(int(b))
    return liste

# ou bien
def f4(listeB):
    return [int(b) for b in liste]
```

5.

```
# la liste est modifiée
def decale_droite(liste):
    liste.pop()
    return [0]+liste

# la liste n'est pas modifiée
def decale_droite(liste):
    dd=[0]
    dd+=liste[:len(liste)-1]
    # ou dd+=liste[:-1]
    return dd

# ou bien
def decale_droite(liste):
    return [0]+liste[:-1]
```

Note : en terme de complexité, les instructions `dd=dd+liste[:-1]` et `dd+=liste[:-1]` ne sont pas équivalentes.