## Corrigés de la semaine 2

```
# Pour calculer inconnue, on extrait une sous-chaine de composite
# qui commence a l'index len(connue)
# qui se termine a l'index len(composite)-len(connue)
# ce qui donne en utilisant une slice
inconnue = composite [ len(connue) : len(composite)-len(connue) ]
#
# on peut aussi faire encore plus simplement
inconnue = composite [ len(connue) : -len(connue) ]
```

```
def divisible(a, b):

"renvoie True si un des deux arguments divise l'autre"

# b divise a si et seulement si le reste

# de la division de a par b est nul

# et il faut regarder aussi si a divise b

return a%b==0 or b%a==0
```

```
spam - Semaine 2 Séquence 7 ----
     def spam(1):
1
         11 11 11
2
     Prend en argument une liste, et retourne la liste modifiée:
3
      * taille paire: on intervertit les deux premiers éléments
4
      * taille impaire, on retire le dernier élément
6
         # si la liste est vide il n'y a rien à faire
         if not 1:
8
             pass
         # si la liste est de taille paire
10
         elif len(1)\%2 == 0:
11
             # on intervertit les deux premiers éléments
             1[0], 1[1] = 1[1], 1[0]
13
         # si elle est de taille impaire
14
         else:
15
             # on retire le dernier élément
16
             1.pop()
17
         # et on n'oublie pas de retourner la liste dans tous les cas
18
         return 1
19
```

```
def multi_tri(listes):

"trie toutes les sous-listes, et retourne listes"

for liste in listes:

# sort fait un effet de bord

liste.sort()

# et on retourne la liste de départ

return listes
```

```
multi_tri_reverse - Semaine 2 Séquence 7
     def multi_tri_reverse(listes, reverses):
1
         """trie toutes les sous listes, dans une direction
2
         précisée par le second argument""
3
         # zip() permet de faire correspondre les éléments
4
         # de listes avec ceux de reverses
         for liste, reverse in zip(listes, reverses):
6
             # on appelle sort en précisant reverse=
             liste.sort(reverse=reverse)
         # on retourne la liste de départ
         return listes
10
```

```
liste_racines - Semaine 2 Séquence 7 -
     from math import e, pi
2
     def liste_racines(p):
3
         "retourne la liste des racines p-ièmes de l'unité"
4
         # une simple compréhension fait l'affaire
5
         # souvenez vous que 1j c'est notre 'i' complexe
6
         return [e**((2*pi*1j*n)/p) for n in range(p)]
     # Il est tout à fait possible aussi de construire les racines pas à pas
     # C'est un peu moins élégant mais ça fonctionne très bien aussi
10
     def liste_racines_bis(p):
11
         "retourne la liste des racines p-ièmes de l'unité"
12
         # on va construire le résultat petit à petit
13
         # en partant d'une liste vide
14
         resultat = []
15
         # pour chaque n dans {0 .. p-1}
         for n in range(p):
17
             # on ajoute dans le résultat la racine d'ordre n
18
             resultat.append(e**((2*pi*1j*n)/p))
19
         # et on retourne le résultat
20
         return resultat
21
```

```
produit_scalaire - Semaine 2 Séquence 7
     def produit_scalaire(X,Y):
1
         # initialisation du résultat
2
         scalaire = 0
3
         # ici encore avec zip() on peut faire correspondre
4
         # les X avec les Y
         for x,y in zip(X,Y):
6
             scalaire += x*y
         # on retourne le résultat
8
         return scalaire
10
     # Il y a plein d'autres solutions qui marchent aussi
11
     # en voici notamment une qui utilise la fonction builtin sum
     # (que nous n'avons pas encore vue, nous la verrons en semaine 4)
     # en voici toutefois un avant-goût: la fonction sum est très pratique
14
     # pour faire la somme de toute une liste de valeurs
15
     def produit_scalaire_bis(X,Y):
16
         """retourne le produit scalaire de deux listes de même taille"""
17
         return sum([x*y for x, y in zip(X, Y)])
18
```

```
affichage - Semaine 2 Séquence 8 =
     # un élève a remarqué très justement que ce code ne fait pas
1
     # exactement ce qui est demandé, en ce sens qu'avec
2
     # l'entrée correspondant à Ted Mosby on obtient A:><
3
     # je préfère toutefois publier le code qui est en
4
     # service pour la correction en ligne, et vous laisse
     # le soin de l'améliorer si vous le souhaitez
     def affichage(s):
         # pour ignorer les espaces et les tabulations
8
         # le plus simple est de les enlever
9
         s=s.replace(' ', '').replace('\t','')
10
         # la liste des mots séparés par une virgule
11
         # nous est donnée par un simple appel à split
         mots = s.split(',')
13
         # si on n'a même pas deux mots, on retourne None
14
         if len(mots) < 2:
15
             return None
16
         # maintenant qu'on sait qu'on a deux mots
17
         # on peut extraire le prénom et le nom
18
         prenom = mots.pop(0)
19
         nom = mots.pop(0)
20
         # on veut afficher "??" si l'âge est inconnu
21
         age = "??"
22
         # mais si l'âge est précisé dans la ligne
23
         if len(mots) >= 2:
24
             # alors on le prend
25
             age = mots.pop(1)
26
         # il ne reste plus qu'à formater
         return "N:>{}< P:>{}< A:>{}<".format(nom, prenom, age)</pre>
28
```

```
carre - Semaine 2 Séquence 8 —
     def carre(s):
1
         # on enlève les espaces et les tabulations
2
         s = s.replace(', ', '').replace('\t','')
3
         # la ligne suivante fait le plus gros du travail
4
         # d'abord on appelle split() pour découper selon les ';'
         # dans le cas où on a des ';' en trop, on obtient dans le
6
              résultat du split un 'token' vide, que l'on ignore
              ici avec le clause 'if token'
         # enfin on convertit tous les tokens restants en entiers avec int()
9
         entiers = [int(token) for token in s.split(";") if token]
10
         # il n'y a plus qu'à mettre au carré, retraduire en strings,
11
         # et à recoudre le tout avec join et ':'
         return ":".join([str(entier**2) for entier in entiers])
13
```