Corrigés de la semaine 2

composite - Semaine 2 Séquence 3

```
# 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) ]
```

divisible - Semaine 2 Séquence 7

```
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(l):
    """
Prend en argument une liste, et retourne la liste modifiée:
 * taille paire: on intervertit les deux premiers éléments
 * taille impaire, on retire le dernier élément
"""
    # si la liste est vide il n'y a rien à faire
    if not l:
        pass
    # si la liste est de taille paire
    elif len(l)%2 == 0:
        # on intervertit les deux premiers éléments
        l[0], l[1] = l[1], l[0]
    # si elle est de taille impaire
    else:
        # on retire le dernier élément
        l.pop()
    # et on n'oublie pas de retourner la liste dans tous les cas
    return l
```

multi_tri - Semaine 2 Séquence 7

```
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):
    """trie toutes les sous listes, dans une direction
    précisée par le second argument"""
    # zip() permet de faire correspondre les éléments
    # de listes avec ceux de reverses
    for liste, reverse in zip(listes, reverses):
        # on appelle sort en précisant reverse=
        liste.sort(reverse=reverse)
    # on retourne la liste de départ
    return listes
```

liste_racines - Semaine 2 Séquence 7

```
from math import e, pi

def liste_racines(p):
    "retourne la liste des racines p-ièmes de l'unité"
    # une simple compréhension fait l'affaire
    # souvenez vous que 1j c'est notre 'i' complexe
    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
def liste_racines_bis(p):
    "retourne la liste des racines p-ièmes de l'unité"
    # on va construire le résultat petit à petit
    # en partant d'une liste vide
    resultat = []
    # pour chaque n dans {0 .. p-1}
    for n in range(p):
        # on ajoute dans le résultat la racine d'ordre n
        resultat.append(e**((2*pi*1j*n)/p))
    # et on retourne le résultat
    return resultat
```

produit_scalaire - Semaine 2 Séquence 7

```
def produit_scalaire(X,Y):
    # initialisation du resultat
    scalaire = 0
    # ici encore avec zip() on peut faire correspondre
    # les X avec les Y
    for x,y in zip(X,Y):
        scalaire += x*y
    # on retourne le résultat
    return scalaire

# Il y a plein d'autres solutions qui marchent aussi
# 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
# pour faire la somme de toute une liste de valeurs
def produit_scalaire_bis(X,Y):
    """retourne le produit scalaire de deux listes de même taille"""
    return sum([x*y for x, y in zip(X, Y)])
```

affichage - Semaine 2 Séquence 8

```
# un élève a remarqué très justement que ce code ne fait pas
# exactement ce qui est demandé, en ce sens qu'avec
# l'entrée correspondant à Ted Mosby on obtient A:><
# je préfère toutefois publier le code qui est en
# 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
    # le plus simple est de les enlever
    s=s.replace(' ', '').replace('\t','')
    # la liste des mots séparés par une virgule
    # nous est donnée par un simple appel à split
    mots = s.split(',')
    # si on n'a même pas deux mots, on retourne None
    if len(mots) < 2:
        return None
    # maintenant qu'on sait qu'on a deux mots
    # on peut extraire le prénom et le nom
    prenom = mots.pop(0)
    nom = mots.pop(0)
    # on veut afficher "??" si l'âge est inconnu
    age = "??"
    # mais si l'âge est précisé dans la ligne
    if len(mots) >= 2:
        # alors on le prend
        age = mots.pop(1)
    # il ne reste plus qu'à formater
    return "N:>{}< P:>{}< A:>{}<".format(nom, prenom, age)
```

carre - Semaine 2 Séquence 8

```
def carre(s):
    # on enlève les espaces et les tabulations
    s = s.replace(' ', '').replace('\t','')
    # la ligne suivante fait le plus gros du travail
    # d'abord on appelle split() pour découper selon les ';'
    # dans le cas où on a des ';' en trop, on obtient dans le
    #    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()
    entiers = [int(token) for token in s.split(";") if token]
    # il n'y a plus qu'à mettre au carré, retraduire en strings,
    # et à recoudre le tout avec join et ':'
    return ":".join([str(entier**2) for entier in entiers])
```