Corrigés de la semaine 2

inconnue - 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
```

liste_P - Semaine 2 Séquence 7

```
def P(x):
    return 2*x**2 - 3*x - 2

def liste_P(liste_x):
    """
    retourne la liste des valeurs de P
    sur les entrées figurant dans liste_x
    """
    return [ P(x) for x in liste_x ]

# On peut bien entendu faire aussi de manière pédestre
def liste_P_bis(liste_x):
    liste_y = []
    for x in liste_x:
        liste_y.append(P(x))
    return liste_y
```

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
```

produit_scalaire - Semaine 2 Séquence 7

```
def produit_scalaire(X,Y):
    """
    retourne le produit scalaire
    de deux listes de même taille
    """
    # initialisation du résultat
    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):
    return sum([x*y for x, y in zip(X, Y)])

# Et encore une; celle-ci par contre est
# assez peu "pythonique"
# on aime bien en général éviter
# les boucles du genre
# for i in range(l)
#     ... l[i]
def produit_scalaire_ter(X, Y):
    scalaire = 0
    n = len(X)
    for i in range(n):
        scalaire += X[i] * Y[i]
    return scalaire
```

libelle - Semaine 2 Séquence 8

```
def libelle(ligne):
    # on enlève les espaces et les tabulations
    ligne = ligne.replace(' ', '').replace('\t','')
    # on cherche les 3 champs
    mots = ligne.split(',')
    # on enleve les morceaux vides
    mots = [mot for mot in mots if mot]
    # si on n'a pas le bon nombre de champs
    # rappelez-vous que 'return' tout court
    # est équivalent à 'return None'
    if len(mots) != 3:
        return
    # maintenant on a les trois valeurs
    nom, prenom, rang = mots
    # comment presenter le rang
    msg_rang = "1er" if rang == "1" \
               else "2nd" if rang == "2" \
                    else "{}-eme".format(rang)
    return "{prenom}.{nom} ({msg_rang})"\
        .format(nom=nom, prenom=prenom, msg_rang=msg_rang)
    # NOTE:
    # on verra plus tard qu'on pourrait écrire beaucoup
    # plus simplement ici
    # return "{prenom}.{nom} ({msg_rang})"\
    #    .format(**locals())
```

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(";")
               # en éliminant les entrées vides qui correspondent
               # à des point-virgules en trop
               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])
```