

MOOC Python

Corrigés de la semaine 2

inconnue - Semaine 2 Séquence 3

```
1 # Pour calculer inconnue, on extrait une sous-chaine de composite
2 # qui commence a l'index len(connue)
3 # qui se termine a l'index len(composite)-len(connue)
4 # ce qui donne en utilisant une slice
5 inconnue = composite [ len(connue) : len(composite)-len(connue) ]
6 #
7 # on peut aussi faire encore plus simplement
8 inconnue = composite [ len(connue) : -len(connue) ]
```

divisible - Semaine 2 Séquence 7

```
1 def divisible(a, b):
2     "renvoie True si un des deux arguments divise l'autre"
3     # b divise a si et seulement si le reste
4     # de la division de a par b est nul
5     # et il faut regarder aussi si a divise b
6     return a%b==0 or b%a==0
```

```

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

```

```

1 def P(x):
2     return 2*x**2 - 3*x - 2
3
4 def liste_P(liste_x):
5     """
6     retourne la liste des valeurs de P
7     sur les entrées figurant dans liste_x
8     """
9     return [ P(x) for x in liste_x ]
10
11 # On peut bien entendu faire aussi de manière pédestre
12 def liste_P_bis(liste_x):
13     liste_y = []
14     for x in liste_x:
15         liste_y.append(P(x))
16     return liste_y

```

```
1 def multi_tri(listes):
2     """
3     trie toutes les sous-listes
4     et retourne listes
5     """
6     for liste in listes:
7         # sort fait un effet de bord
8         liste.sort()
9     # et on retourne la liste de départ
10    return listes
```

```
1 def multi_tri_reverse(listes, reverses):
2     """
3     trie toutes les sous listes, dans une direction
4     précisée par le second argument
5     """
6     # zip() permet de faire correspondre les éléments
7     # de listes avec ceux de reverses
8     for liste, reverse in zip(listes, reverses):
9         # on appelle sort en précisant reverse=
10        liste.sort(reverse=reverse)
11    # on retourne la liste de départ
12    return listes
```

```
1 def produit_scalaire(X,Y):
2     """
3     retourne le produit scalaire
4     de deux listes de même taille
5     """
6     # initialisation du résultat
7     scalaire = 0
8     # ici encore avec zip() on peut faire correspondre
9     # les X avec les Y
10    for x,y in zip(X,Y):
11        scalaire += x*y
12    # on retourne le résultat
13    return scalaire
14
15    # Il y a plein d'autres solutions qui marchent aussi
16    # en voici notamment une qui utilise la fonction builtin sum
17    # (que nous n'avons pas encore vue, nous la verrons en semaine 4)
18    # en voici toutefois un avant-goût: la fonction sum est très pratique
19    # pour faire la somme de toute une liste de valeurs
20    def produit_scalaire_bis(X,Y):
21        return sum([x*y for x, y in zip(X, Y)])
22
23    # Et encore une; celle-ci par contre est
24    # assez peu "pythonique"
25    # on aime bien en général éviter
26    # les boucles du genre
27    # for i in range(l)
28    #     ... l[i]
29    def produit_scalaire_ter(X, Y):
30        scalaire = 0
31        n = len(X)
32        for i in range(n):
33            scalaire += X[i] * Y[i]
34        return scalaire
```

```

1 def libelle(ligne):
2     # on enlève les espaces et les tabulations
3     ligne = ligne.replace(' ', '').replace('\t','')
4     # on cherche les 3 champs
5     mots = ligne.split(',')
6     # on enleve les morceaux vides
7     mots = [mot for mot in mots if mot]
8     # si on n'a pas le bon nombre de champs
9     # rappelez-vous que 'return' tout court
10    # est équivalent à 'return None'
11    if len(mots) != 3:
12        return
13    # maintenant on a les trois valeurs
14    nom, prenom, rang = mots
15    # comment presenter le rang
16    msg_rang = "1er" if rang == "1" \
17                else "2nd" if rang == "2" \
18                else "{}-eme".format(rang)
19    return "{prenom}.{nom} ({msg_rang})"\
20            .format(nom=nom, prenom=prenom, msg_rang=msg_rang)
21    # NOTE:
22    # on verra plus tard qu'on pourrait écrire beaucoup
23    # plus simplement ici
24    # return "{prenom}.{nom} ({msg_rang})"\
25    #     .format(**locals())

```

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