

MOOC Python

Corrigés de la semaine 2

pythonid (regexp) - Semaine 2 Séquence 2

```
1 # un identificateur commence par une lettre ou un underscore
2 # et peut être suivi par n'importe quel nombre de
3 # lettre, chiffre ou underscore, ce qui se trouve être \w
4 # si on ne se met pas en mode unicode
5 pythonid_regexp = "[a-zA-Z_]\w*"
```

pythonid (v2) - Semaine 2 Séquence 2

```
1 # on peut aussi bien sûr l'écrire en clair
2 pythonid_bis = "[a-zA-Z_][a-zA-Z0-9_]*"
```

agenda (regexp) - Semaine 2 Séquence 2

```
1 # l'exercice est basé sur re.match, ce qui signifie que
2 # le match est cherché au début de la chaîne
3 # MAIS il nous faut bien mettre \Z à la fin de notre regexp,
4 # sinon par exemple avec la cinquième entrée le nom 'Du Pré'
5 # sera reconnu partiellement comme simplement 'Du'
6 # au lieu d'être rejeté à cause de l'espace
7 #
8 # du coup pensez à bien toujours définir
9 # vos regexps avec des raw-strings
10 #
11 # remarquez sinon l'utilisation à la fin de :? pour signifier qu'on peut
12 # mettre ou non un deuxième séparateur ':'
13 #
14 agenda_regexp = r"\A(?:P<prenom>[-\w]*):(?:P<nom>[-\w]+):?\Z"
```

phone (regexp) - Semaine 2 Séquence 2

```
1 # idem concernant le \Z final
2 #
3 # il faut bien backslasher le + dans le +33
4 # car sinon cela veut dire 'un ou plusieurs'
5 #
6 phone_regexp = r"(\+33|0)(?P<number>[0-9]{9})\Z"
```

url (regexp) - Semaine 2 Séquence 2

```
1 # en ignorant la casse on pourra ne mentionner les noms de protocoles
2 # qu'en minuscules
3 i_flag = "(?i)"
4
5 # pour élaborer la chaine (proto1|proto2|...)
6 protos_list = ['http', 'https', 'ftp', 'ssh', ]
7 protos      = "(?P<proto>" + "|".join(protos_list) + ")"
8
9 # à l'intérieur de la zone 'user/password', la partie
10 # password est optionnelle - mais on ne veut pas le ':' dans
11 # le groupe 'password' - il nous faut deux groupes
12 password    = r"(:(?P<password>[^\:]+))?"
13
14 # la partie user-password elle-même est optionnelle
15 user        = r"((?P<user>\w+){password}@)?".format(**locals())
16
17 # pour le hostname on accepte des lettres, chiffres, underscore et '.'
18 # attention à backslasher . car sinon ceci va matcher tout y compris /
19 hostname    = r"(?P<hostname>[\w\.]*)"
20
21 # le port est optionnel
22 port        = r"(:(?P<port>\d+))?"
23
24 # après le premier slash
25 path        = r"(?P<path>.*)"
26
27 # on assemble le tout
28 url = i_flag + protos + "://" + user + hostname + port + '/' + path
```

inconnue - Semaine 2 Séquence 6

```
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) ]
```

libelle - Semaine 2 Séquence 6

```
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 "{}-ème".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())
```

carre - Semaine 2 Séquence 6

```

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])

```

aplatir - Semaine 2 Séquence 7

```

1 def aplatir(conteneurs):
2     "retourne une liste des éléments des éléments de conteneurs"
3     # on peut concaténer les éléments de deuxième niveau
4     # par une simple imbrication de deux compréhensions de liste
5     return [element for conteneur in conteneurs for element in conteneur]

```

alternat - Semaine 2 Séquence 7

```

1 def alternat(l1, l2):
2     "renvoie une liste des éléments pris un sur deux dans l1 et dans l2"
3     # pour réaliser l'alternance on peut combiner zip avec aplatir
4     # telle qu'on vient de la réaliser
5     return aplatir(zip(l1, l2))

```

alternat (v2) - Semaine 2 Séquence 7

```

1 def alternat_bis(l1, l2):
2     "une deuxième version de alternat"
3     # la même idée mais directement, sans utiliser aplatir
4     return [element for conteneur in zip(l1, l2) for element in conteneur]

```

```
1 def intersect(A, B):
2     """
3     prend en entrée deux listes de tuples de la forme
4     (entier, valeur)
5     renvoie la liste des valeurs associées dans A ou B
6     aux entiers présents dans A et B
7     """
8     # pour montrer un exemple de fonction locale:
9     # une fonction qui renvoie l'ensemble des entiers
10    # présents dans une des deux listes d'entrée
11    def keys(S):
12        return {k for k, val in S}
13    # on l'applique à A et B
14    keys_A = keys(A)
15    keys_B = keys(B)
16    #
17    # les entiers présents dans A et B
18    # avec une intersection d'ensembles
19    common_keys = keys_A & keys_B
20    # et pour conclure on fait une union sur deux
21    # compréhensions d'ensembles
22    return {vala for k, vala in A if k in common_keys} \
23           | {valb for k, valb in B if k in common_keys}
```

```

1 def P(x):
2     return 2 * x**2 - 3 * x - 2
3
4
5 def liste_P(liste_x):
6     """
7     retourne la liste des valeurs de P
8     sur les entrées figurant dans liste_x
9     """
10    return [P(x) for x in liste_x]
11
12 # On peut bien entendu faire aussi de manière pédestre
13 def liste_P_bis(liste_x):
14     liste_y = []
15     for x in liste_x:
16         liste_y.append(P(x))
17     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 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(liste):
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 liste:
9         pass
10    # si la liste est de taille paire
11    elif len(liste) % 2 == 0:
12        # on intervertit les deux premiers éléments
13        liste[0], liste[1] = liste[1], liste[0]
14    # si elle est de taille impaire
15    else:
16        # on retire le dernier élément
17        liste.pop()
18    # et on n'oublie pas de retourner la liste dans tous les cas
19    return liste

```