MOOC Python

Corrigés de la semaine 2

pythonid (regexp) - Semaine 2 Séquence 2

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

pythonid (bis) - Semaine 2 Séquence 2

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

agenda (regexp) - Semaine 2 Séquence 2

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

phone (regexp) - Semaine 2 Séquence 2

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

url (regexp) - Semaine 2 Séquence 2

```
# en ignorant la casse on pourra ne mentionner les noms de protocoles
# qu'en minuscules
i_flag = "(?i)"

# pour élaborer la chaine (proto1|proto2|...)
protos_list = ['http', 'https', 'ftp', 'ssh', ]
protos      = "(?P<proto>" + "|".join(protos_list) + ")"

# à l'intérieur de la zone 'user/password', la partie
# password est optionnelle - mais on ne veut pas le ':' dans
# le groupe 'password' - il nous faut deux groupes
password    = r"(:(?P<password>[^:]+))?"

# la partie user-password elle-même est optionnelle
user        = r"((?P<user>\w+){password}@)?".format(**locals())

# pour le hostname on accepte des lettres, chiffres, underscore et '.'
# attention à backslaher . car sinon ceci va matcher tout y compris /
hostname    = r"(?P<hostname>[\w\.]+)"

# le port est optionnel
port        = r"(:(?P<port>\d+))?"

# après le premier slash
path        = r"(?P<path>.*)"

# on assemble le tout
url = i_flag + protos + "://" + user + hostname + port + '/' + path
```

libelle - Semaine 2 Séquence 6

```
def libelle(ligne):
    # on enlève les espaces et les tabulations
    ligne = ligne.replace(' ', '').replace('\t','')
    # on cherche les 3 champs
    mots = ligne.split(',')
    # 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 "{}-ème".format(rang)
    return f"{prenom}.{nom} ({msg_rang})"
```

carre - Semaine 2 Séquence 6

```
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])
```

inconnue - Semaine 2 Séquence 6

```
# pour enlever à gauche et à droite une chaine de longueur x
# on peut faire composite[ x : -x ]
# or ici x vaut len(connue)
def inconnue(composite, connue):
    return composite[ len(connue) : -len(connue) ]
```

inconnue (bis) - Semaine 2 Séquence 6

```
# ce qui peut aussi s'écrire comme ceci si on préfère
def inconnue_bis(composite, connue):
    return composite[ len(connue) : len(composite)-len(connue) ]
```

divisible - Semaine 2 Séquence 6

```
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
```

divisible (bis) - Semaine 2 Séquence 6

```
def divisible_bis(a, b):
    if a % b == 0:
        return True
    if b % a == 0:
        return True
    return False
```

morceaux - Semaine 2 Séquence 6

```
def morceaux(x):
    if x <= -5:
        return -x - 5
    elif x <= 5:
        return 0
    else:
        return x / 5 - 1
```

morceaux (bis) - Semaine 2 Séquence 6

```
def morceaux_bis(x):
    if x <= -5:
        return -x - 5
    if x <= 5:
        return 0
    return x / 5 - 1
```

morceaux (ter) - Semaine 2 Séquence 6

```
# on peut aussi faire des tests d'intervalle
# comme ceci  0 <= x <= 10
def morceaux_ter(x):
    if x <= -5:
        return -x - 5
    elif -5 <= x <= 5:
        return 0
    else:
        return x / 5 - 1
```

aplatir - Semaine 2 Séquence 7

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

alternat - Semaine 2 Séquence 7

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

alternat (bis) - Semaine 2 Séquence 7

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

intersect - Semaine 2 Séquence 7

```
def intersect(A, B):
    """
    prend en entrée deux listes de tuples de la forme
    (entier, valeur)
    renvoie la liste des valeurs associées dans A ou B
    aux entiers présents dans A et B
    """
    # pour montrer un exemple de fonction locale:
    # une fonction qui renvoie l'ensemble des entiers
    # présents dans une des deux listes d'entrée
    def keys(S):
        return {k for k, val in S}
    # on l'applique à A et B
    keys_A = keys(A)
    keys_B = keys(B)
    #
    # les entiers présents dans A et B
    # avec une intersection d'ensembles
    common_keys = keys_A & keys_B
    # et pour conclure on fait une union sur deux
    # compréhensions d'ensembles
    return {vala for k, vala in A if k in common_keys} \
         | {valb for k, valb in B if k in common_keys}
```

liste_P (bis) - Semaine 2 Séquence 7

```
# 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
```

produit_scalaire (bis) - Semaine 2 Séquence 7

```
# 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)])

```

produit_scalaire (ter) - Semaine 2 Séquence 7

```
# 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
```