

MOOC Python 3

Session 2018

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 = r"[a-zA-Z_]\w*"
```

pythonid_bis - Semaine 2 Séquence 2

```
1 # on peut aussi bien sûr l'écrire en clair
2 pythonid_bis = r"[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 = 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 = 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 # on utilise ici un raw f-string avec le préfixe rf
16 # pour insérer la regexp <password> dans la regexp <user>
17 user        = rf"((?P<user>\w+){password}@)?"
18
19 # pour le hostname on accepte des lettres, chiffres, underscore et '.'
20 # attention à backslasher . car sinon ceci va matcher tout y compris /
21 hostname    = r"(?P<hostname>[\w\.]*)"
22
23 # le port est optionnel
24 port        = r"(:(?P<port>\d+))?"
25
26 # après le premier slash
27 path        = r"(?P<path>.*)"
28
29 # on assemble le tout
30 url = i_flag + protos + "://" + user + hostname + port + '/' + path
```

url_bis - Semaine 2 Séquence 2

```

1  # merci à sizeof qui a pointé l'utilisation de re.X
2  # https://docs.python.org/fr/3/library/re.html#re.X
3  # ce qui donne une présentation beaucoup plus compacte
4
5  protos_list = ['http', 'https', 'ftp', 'ssh', ]
6
7  url_bis = rf"""(?x)                # verbose mode
8      (?i)                          # ignore case
9      (?P<proto>{"|".join(protos_list)}) # http|https|...
10     ://                            # separator
11     ((?P<user>\w+){password}@)?     # optional user/password
12     (?P<hostname>[\w\.]++)          # mandatory hostname
13     (:(?P<port>\d+))?               # optional port
14     /(?(P<path>.*))                 # mandatory path
15 """

```

label - Semaine 2 Séquence 6

```

1  def label(prenom, note):
2      if note < 10:
3          return f"{prenom} est recalé"
4      elif note < 16:
5          return f"{prenom} est reçu"
6      else:
7          return f"félicitations à {prenom}"

```

label_bis - Semaine 2 Séquence 6

```

1  def label_bis(prenom, note):
2      if note < 10:
3          return f"{prenom} est recalé"
4      # on n'en a pas vraiment besoin ici, mais
5      # juste pour illustrer cette construction
6      elif 10 <= note < 16:
7          return f"{prenom} est reçu"
8      else:
9          return f"félicitations à {prenom}"

```

label_ter - Semaine 2 Séquence 6

```
1 # on n'a pas encore vu l'expression conditionnelle
2 # et dans ce cas précis ce n'est pas forcément une
3 # idée géniale, mais pour votre curiosité on peut aussi
4 # faire comme ceci
5 def label_ter(prenom, note):
6     return f"{prenom} est recalé" if note < 10 \
7         else f"{prenom} est reçu" if 10 <= note < 16 \
8         else f"félicitations à {prenom}"
```

inconnue - Semaine 2 Séquence 6

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

inconnue_bis - Semaine 2 Séquence 6

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

laccess - Semaine 2 Séquence 6

```
1 def laccess(liste):
2     """
3     retourne un élément de la liste selon la taille
4     """
5     # si la liste est vide il n'y a rien à faire
6     if not liste:
7         return
8     # si la liste est de taille paire
9     if len(liste) % 2 == 0:
10         return liste[-1]
11     else:
12         return liste[len(liste)//2]
```

laccess_bis - Semaine 2 Séquence 6

```
1 # une autre version qui utilise
2 # un trait qu'on n'a pas encore vu
3 def laccess_bis(liste):
4     # si la liste est vide il n'y a rien à faire
5     if not liste:
6         return
7     # l'index à utiliser selon la taille
8     index = -1 if len(liste) % 2 == 0 else len(liste) // 2
9     return liste[index]
```

divisible - Semaine 2 Séquence 6

```
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     if a % b == 0:
6         return True
7     # et il faut regarder aussi si a divise b
8     if b % a == 0:
9         return True
10    return False
```

divisible_bis - Semaine 2 Séquence 6

```
1 def divisible_bis(a, b):
2     "renvoie True si un des deux arguments divise l'autre"
3     # on n'a pas encore vu les opérateurs logiques, mais
4     # on peut aussi faire tout simplement comme ça
5     # sans faire de if du tout
6     return a % b == 0 or b % a == 0
```

```
1 def morceaux(x):  
2     if x <= -5:  
3         return -x - 5  
4     elif x <= 5:  
5         return 0  
6     else:  
7         return x / 5 - 1
```

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

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

wc - Semaine 2 Séquence 6

```
1 def wc(string):
2     """
3     Compte les nombres de lignes, de mots et de caractères
4
5     Retourne une liste de ces 3 nombres (notez qu'usuellement
6     on renverrait plutôt un tuple, qu'on étudiera la semaine prochaine)
7     """
8     # on peut tout faire avec la bibliothèque standard
9     nb_lines = string.count('\n')
10    nb_words = len(string.split())
11    nb_bytes = len(string)
12    return [nb_lines, nb_words, nb_bytes]
```

liste_P - Semaine 2 Séquence 7

```
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]
```

liste_P_bis - Semaine 2 Séquence 7

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

carre - Semaine 2 Séquence 7

```
1 def carre(line):
2     # on enlève les espaces et les tabulations
3     line = line.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 la clause 'if token'
9     # enfin on convertit tous les tokens restants en entiers avec int()
10    entiers = [int(token) for token in line.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])
```

carre_bis - Semaine 2 Séquence 7

```
1 def carre_bis(line):
2     # pareil mais avec, à la place des compréhensions
3     # des expressions génératrices que - rassurez-vous -
4     # l'on n'a pas vues encore, on en parlera en semaine 5
5     # le point que je veux illustrer ici c'est que c'est
6     # exactement le même code mais avec () au lieu de []
7     line = line.replace(' ', '').replace('\t','')
8     entiers = (int(token) for token in line.split(";")
9               if token)
10    return ":".join(str(entier**2) for entier in entiers)
```

carre_ter - Semaine 2 Séquence 7

```
1 def carre_ter(ligne):
2     # On extrait toutes les valeurs séparées par des points-
3     # virgules, on les nettoie avec la méthode strip
4     # et on stocke le résultat dans une liste
5     liste_valeurs = [t.strip() for t in ligne.split(';')]
6     # Il ne reste plus qu'à calculer les carrés pour les
7     # valeurs valides (non vides) et les remettre dans une str
8     return ":".join([str(int(v)**2) for v in liste_valeurs if v])
```