

# Tous les corrigés

## Table des matières

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

multi\_tri\_reverse - Semaine 2 Séquence 7

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

liste\_racines - Semaine 2 Séquence 7

```
1 from math import e, pi
2
3 def liste_racines(p):
4     "retourne la liste des racines p-ièmes de l'unité"
5     # une simple compréhension fait l'affaire
6     # souvenez vous que 1j c'est notre 'i' complexe
7     return [e**((2*pi*1j*n)/p) for n in range(p)]
8
9 # Il est tout à fait possible aussi de construire les racines pas à pas
10 # C'est un peu moins élégant mais ça fonctionne très bien aussi
11 def liste_racines_bis(p):
12     "retourne la liste des racines p-ièmes de l'unité"
13     # on va construire le résultat petit à petit
14     # en partant d'une liste vide
15     resultat = []
16     # pour chaque n dans {0 .. p-1}
17     for n in range(p):
18         # on ajoute dans le résultat la racine d'ordre n
19         resultat.append(e**((2*pi*1j*n)/p))
20     # et on retourne le résultat
21     return resultat
```

```

1 def produit_scalaire(X,Y):
2     # initialisation du résultat
3     scalaire = 0
4     # ici encore avec zip() on peut faire correspondre
5     # les X avec les Y
6     for x,y in zip(X,Y):
7         scalaire += x*y
8     # on retourne le résultat
9     return scalaire
10
11 # Il y a plein d'autres solutions qui marchent aussi
12 # en voici notamment une qui utilise la fonction builtin sum
13 # (que nous n'avons pas encore vue, nous la verrons en semaine 4)
14 # en voici toutefois un avant-goût: la fonction sum est très pratique
15 # pour faire la somme de toute une liste de valeurs
16 def produit_scalaire_bis(X,Y):
17     """retourne le produit scalaire de deux listes de même taille"""
18     return sum([x*y for x, y in zip(X, Y)])

```

```
1  # un élève a remarqué très justement que ce code ne fait pas
2  # exactement ce qui est demandé, en ce sens qu'avec
3  # l'entrée correspondant à Ted Mosby on obtient A:><
4  # je préfère toutefois publier le code qui est en
5  # service pour la correction en ligne, et vous laisse
6  # le soin de l'améliorer si vous le souhaitez
7  def affichage(s):
8      # pour ignorer les espaces et les tabulations
9      # le plus simple est de les enlever
10     s=s.replace(' ', '').replace('\t','')
11     # la liste des mots séparés par une virgule
12     # nous est donnée par un simple appel à split
13     mots = s.split(',')
14     # si on n'a même pas deux mots, on retourne None
15     if len(mots) < 2:
16         return None
17     # maintenant qu'on sait qu'on a deux mots
18     # on peut extraire le prénom et le nom
19     prenom = mots.pop(0)
20     nom = mots.pop(0)
21     # on veut afficher "???" si l'âge est inconnu
22     age = "???"
23     # mais si l'âge est précisé dans la ligne
24     if len(mots) >= 2:
25         # alors on le prend
26         age = mots.pop(1)
27     # il ne reste plus qu'à formater
28     return "N:>{ }< P:>{ }< A:>{ }<".format(nom, prenom, age)
```

carre - Semaine 2 Séquence 8

```
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(";") if token]
11    # il n'y a plus qu'à mettre au carré, retraduire en strings,
12    # et à recoudre le tout avec join et ':'
13    return ":".join([str(entier**2) for entier in entiers])
```

index - Semaine 3 Séquence 2

```
1 def index(bateaux):
2     """
3     Calcule sous la forme d'un dictionnaire indexé par les ids
4     un index de tous les bateaux présents dans la liste en argument
5     Comme les données étendues et abrégées ont toutes leur id
6     en première position on peut en fait utiliser ce code
7     avec les deux types de données
8     """
9     # c'est une simple compréhension de dictionnaire
10    return {bateau[0]:bateau for bateau in bateaux}
```

index - Semaine 3 Séquence 2

```
1 def index2(bateaux):
2     """
3     La même chose mais de manière itérative
4     """
5     # si on veut décortiquer
6     resultat = {}
7     for bateau in bateaux:
8         resultat [bateau[0]] = bateau
9     return resultat
```

```
1 def merge(extended, abbreviated):
2     """
3     Consolide des données étendues et des données abrégées
4     comme décrit dans l'énoncé
5     Le coût de cette fonction est linéaire dans la taille
6     des données (longueur des listes)
7     """
8     # on initialise le résultat avec un dictionnaire vide
9     result = {}
10    # pour les données étendues
11    for ship in extended:
12        # on affecte les 6 premiers champs
13        # et on ignore les champs de rang 6 et au delà
14        id, latitude, longitude, timestamp, name, country = ship[:6]
15        # on crée une entrée dans le résultat,
16        # avec la mesure correspondant aux données étendues
17        result[id] = [name, country, (latitude, longitude, timestamp)]
18    # maintenant on peut compléter le résultat avec les données abrégées
19    for id, latitude, longitude, timestamp in abbreviated:
20        # et avec les hypothèses on sait que le bateau a déjà été
21        # inscrit dans le résultat, donc result[id] doit déjà exister
22        # et on peut se contenter d'ajouter la mesure abrégée
23        # dans l'entrée correspondant dans result
24        result[id].append((latitude, longitude, timestamp))
25    # et retourner le résultat
26    return result
```



```
1 def merge2(extended, abbreviated):
2     """
3     Une deuxième version, linéaire également
4     """
5     # on initialise le résultat avec un dictionnaire vide
6     result = {}
7     # on remplit d'abord à partir des données étendues
8     for ship in extended:
9         id = ship[0]
10        # on crée la liste avec le nom et le pays
11        result[id] = ship[4:6]
12        # on ajoute un tuple correspondant à la position
13        result[id].append(tuple(ship[1:4]))
14    # pareil que pour la première solution,
15    # on sait d'après les hypothèses
16    # que les id trouvées dans abbreviated
17    # sont déjà présentes dans le resultat
18    for ship in abbreviated:
19        id = ship[0]
20        # on ajoute un tuple correspondant à la position
21        result[id].append(tuple(ship[1:4]))
22    return result
```

```

1 def merge3(extended, abbreviated):
2     """
3     Une troisième solution
4     à cause du tri que l'on fait au départ, cette
5     solution n'est plus linéaire mais en  $O(n \cdot \log(n))$ 
6     """
7     # ici on va tirer profit du fait que les id sont
8     # en première position dans les deux tableaux
9     # si bien que si on les trie,
10    # on va mettre les deux tableaux 'en phase'
11    #
12    # c'est une technique qui marche dans ce cas précis
13    # parce qu'on sait que les deux tableaux contiennent des données
14    # pour exactement le même ensemble de bateaux
15    #
16    # on a deux choix, selon qu'on peut se permettre ou non de
17    # modifier les données en entrée. Supposons que oui:
18    extended.sort()
19    abbreviated.sort()
20    # si ça n'avait pas été le cas on aurait fait plutôt
21    # extended = extended.sorted() et idem pour l'autre
22    #
23    # il ne reste plus qu'à assembler le résultat
24    # en découpant des tranches
25    # et en les transformant en tuples pour les positions
26    # puisque c'est ce qui est demandé
27    return {
28        e[0] : e[4:6] + [ tuple(e[1:4]), tuple(a[1:4]) ]
29        for (e,a) in zip (extended, abbreviated)
30    }

```

```

1 def diff(extended, abbreviated):
2     """Calcule comme demandé dans l'exercice, et sous formes d'ensembles
3     (*) les noms des bateaux seulement dans extended
4     (*) les noms des bateaux présents dans les deux listes
5     (*) les ids des bateaux seulement dans abbreviated
6     """
7     ### on n'utilise que des ensembles dans tous l'exercice
8     # les ids de tous les bateaux dans extended
9     # une compréhension d'ensemble
10    extended_ids = {ship[0] for ship in extended}
11    # les ids de tous les bateaux dans abbreviated
12    # idem
13    abbreviated_ids = {ship[0] for ship in abbreviated}
14    # les ids des bateaux seulement dans abbreviated
15    # une difference d'ensembles
16    abbreviated_only_ids = abbreviated_ids - extended_ids
17    # les ids des bateaux dans les deux listes
18    # une intersection d'ensembles
19    both_ids = abbreviated_ids & extended_ids
20    # les ids des bateaux seulement dans extended
21    # ditto
22    extended_only_ids = extended_ids - abbreviated_ids
23    # pour les deux catégories où c'est possible
24    # on recalcule les noms des bateaux
25    # par une compréhension d'ensemble
26    both_names = \
27        {ship[4] for ship in extended if ship[0] in both_ids}
28    extended_only_names = \
29        {ship[4] for ship in extended if ship[0] in extended_only_ids}
30    # enfin on retourne les 3 ensembles sous forme d'un tuple
31    return extended_only_names, both_names, abbreviated_only_ids

```

```

1  # le module this est implémenté comme une petite énigme
2  # comme le laissent entrevoir les indices, on y trouve
3  # (*) dans l'attribut 's' une version encodée du manifeste
4  # (*) dans l'attribut 'd' le code à utiliser pour décoder
5  #
6  # ce qui veut dire qu'en première approximation on pourrait
7  # obtenir une liste des caractères du manifeste en faisant
8  #
9  # [ this.d [c] for c in this.s ]
10 #
11 # mais ce serait le cas seulement si le code agissait sur
12 # tous les caractères; comme ce n'est pas le cas il faut
13 # laisser intacts les caractères dans this.s qui ne sont pas
14 # dans this.d (dans le sens "c in this.d")
15 #
16 # je fais exprès de ne pas appeler l'argument this pour
17 # illustrer le fait qu'un module est un objet comme un autre
18 #
19
20 def decode_zen(this_module):
21     "décode le zen de python à partir du module this"
22     # la version encodée du manifeste
23     encoded = this_module.s
24     # le 'code'
25     code = this_module.d
26     # si un caractère est dans le code, on applique le code
27     # sinon on garde le caractère tel quel
28     # aussi, on appelle 'join' pour refaire une chaîne à partir
29     # de la liste des caractères décodés
30     return ''.join([code[c] if c in code else c for c in encoded])

```

```

1  # une autre version qui marche aussi, en utilisant
2  # dict.get(key, default)
3  def decode_zen2(this):
4      return "".join([this.d.get(c, c) for c in this.s])

```

```

1 def dispatch1(a, b):
2     """dispatch1 comme spécifié"""
3     # si les deux arguments sont pairs
4     if a%2 == 0 and b%2 == 0:
5         return a*a + b*b
6     # si a est pair et b est impair
7     elif a%2 == 0 and b%2 != 0:
8         return a*(b-1)
9     # si a est impair et b est pair
10    elif a%2 != 0 and b%2 == 0:
11        return (a-1)*b
12    # sinon - c'est que a et b sont impairs
13    else:
14        return a*a - b*b

```

```

1 def dispatch2(a, b, A, B):
2     """dispatch2 comme spécifié"""
3     # les deux cas de la diagonale \
4     if (a in A and b in B) or (a not in A and b not in B):
5         return a*a + b*b
6     # sinon si b n'est pas dans B
7     # ce qui alors implique que a est dans A
8     elif b not in B:
9         return a*(b-1)
10    # le dernier cas, on sait forcément que
11    # b est dans B et a n'est pas dans A
12    else:
13        return (a-1)*b

```

```

1 def comptage(in_filename, out_filename):
2     """
3     retranscrit le fichier in_filename dans le fichier out_filename
4     en ajoutant des annotations sur les nombres de lignes, de mots
5     et de caractères
6     """
7     # on ouvre le fichier d'entrée en lecture
8     # on aurait pu mettre open (in_filename, 'r')
9     with open(in_filename) as input:
10        # on ouvre la sortie en écriture
11        with open(out_filename, "w") as output:
12            # initialisations
13            lineno = 0
14            total_words = 0
15            total_chars = 0
16            # pour toutes les lignes du fichier d'entrée
17            for line in input:
18                # on maintient le nombre de lignes
19                # qui est aussi la ligne courante
20                lineno += 1
21                # autant de mots que d'éléments dans split()
22                nb_words = len(line.split())
23                total_words += nb_words
24                # autant de caractères que d'éléments dans la ligne
25                nb_chars = len(line)
26                total_chars += nb_chars
27                # on écrit la ligne de sortie; pas besoin
28                # de newline (\n) car line en a déjà un
29                output.write("{}: {}: {}:{}\n".\
30                            format(lineno, nb_words, nb_chars, line))
31            # on écrit la ligne de synthèse
32            output.write("{}: {}: {}{}\n".\
33                        format(lineno, total_words, total_chars))

```

```

1 def pgcd(a, b):
2     "le pgcd de a et b par l'algorithme d'Euclide"
3     # l'algorithme suppose que a >= b
4     # donc si ce n'est pas le cas
5     # il faut inverser les deux entrées
6     if b > a :
7         a, b = b, a
8     # boucle sans fin
9     while True:
10        # on calcule le reste
11        r = a % b
12        # si le reste est nul, on a terminé
13        if r == 0:
14            return b
15        # sinon on passe à l'itération suivante
16        a, b = b, r

```

```

1 from operator import mul
2
3 def numbers(liste):
4     """
5     retourne un tuple contenant
6     (*) la somme
7     (*) le produit
8     (*) le minimum
9     (*) le maximum
10    des éléments de la liste
11    """
12
13    return (
14        # la builtin 'sum' renvoie la somme
15        sum(liste),
16        # pour la multiplication, reduce est nécessaire
17        reduce(mul, liste, 1),
18        # les builtin 'min' et 'max' font ce qu'on veut aussi
19        min(liste),
20        max(liste)
21    )

```

validation - Semaine 4 Séquence 3

```
1 def validation(f, g, entrees):
2     """
3     retourne une liste de booléens, un par entree dans entrees
4     qui indique si f(entree) == g(entree)
5     """
6     # on vérifie pour chaque entrée si f et g retournent
7     # des résultats égaux avec ==
8     # et on assemble le tout avec une comprehension de liste
9     return [f(entree) == g(entree) for entree in entrees]
```

aplatir - Semaine 4 Séquence 4

```
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 4 Séquence 4

```
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 - Semaine 4 Séquence 4

```
1 def alternat2(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 values(S):
12        return {i for i, val in S}
13    # on l'applique à A et B
14    val_A = values(A)
15    val_B = values(B)
16    #
17    # bien sûr on aurait pu écrire directement
18    # val_A = {i for i, val in A}
19    # val_B = {i for i, val in B}
20    #
21    # les entiers présents dans A et B
22    # avec une intersection d'ensembles
23    common_keys = val_A & val_B
24    # et pour conclure on fait une union sur deux
25    # compréhensions d'ensembles
26    return {vala for a, vala in A if a in common_keys} \
27        | {valb for b, valb in B if b in common_keys}

```

```

1 import math
2
3 def distance(*args):
4     "la racine de la somme des carrés des arguments"
5     # avec une compréhension on calcule la liste des carrés des arguments
6     # on applique ensuite sum pour en faire la somme
7     # vous pourrez d'ailleurs vérifier que sum ([]) = 0
8     # enfin on extrait la racine avec math.sqrt
9     return math.sqrt(sum([x**2 for x in args]))

```

```

1 def doubler_premier(f, first, *args):
2     """
3     renvoie le résultat de la fonction f appliquée sur
4     f(2 * first, *args)
5     """
6     # une fois qu'on a écrit la signature on a presque fini le travail
7     # en effet on a isolé la fonction, son premier argument, et le reste
8     # des arguments
9     # il ne reste qu'à appeler f, après avoir doublé first
10    return f(2*first, *args)

```

```

1 def doubler_premier2(f, first, *args, **keywords):
2     """
3     équivalent à doubler_premier
4     mais on peut aussi passer des arguments nommés
5     """
6     # c'est exactement la même chose
7     return f(2*first, *args, **keywords)
8
9 # Complément - niveau avancé
10 # ----
11 # Il y a un cas qui ne fonctionne pas avec cette implémentation,
12 # quand le premier argument de f a une valeur par défaut
13 # *et* on veut pouvoir appeler doubler_premier
14 # en nommant ce premier argument
15 #
16 # par exemple - avec f=muln telle que définie dans l'énoncé
17 #def muln(x=1, y=1): return x*y
18
19 # alors ceci
20 #doubler_premier2(muln, x=1, y=2)
21 # ne marche pas car on n'a pas les deux arguments requis
22 # par doubler_premier2
23 #
24 # et pour écrire, disons doubler_permier3, qui marcherait aussi comme cela
25 # il faudrait faire une hypothèse sur le nom du premier argument...

```

```
1 def validation2(f, g, argument_tuples):
2     """
3     retourne une liste de booléens, un par entree dans entrees
4     qui indique si f(*tuple) == g(*tuple)
5     """
6     # c'est presque exactement comme validation, sauf qu'on s'attend
7     # à recevoir une liste de tuples d'arguments, qu'on applique
8     # aux deux fonctions avec la forme * au lieu de les passer directement
9     return [f(*tuple) == g(*tuple) for tuple in argument_tuples]
```

```

1  # une troisième implémentation de RPCProxy
2
3  class Forwarder(object):
4      """
5      Une instance de la classe Forwarder est un callable
6      qui peut être utilisée comme une méthode sur l
7      class RPCProxy
8      """
9      def __init__(self, rpc_proxy, methodname):
10         """
11         le constructeur mémorise l'instance de RPCProxy
12         et le nom de la méthode qui a été appelée
13         """
14         self.methodname = methodname
15         self.rpc_proxy = rpc_proxy
16
17     def __call__(self, *args):
18         """
19         en rendant cet objet callable, on peut l'utiliser
20         comme une méthode de RPCProxy
21         """
22         print "Envoi à {} \nde la fonction {} -- args= {}".\
23             format(self.rpc_proxy.url, self.methodname, args)
24         return "retour de la fonction " + self.methodname
25
26 class RPCProxy(object):
27     """
28     Une troisième implémentation de RPCProxy qui sous-traite
29     à une classe annexe 'Forwarder' qui se comporte comme
30     une *factory* de méthodes
31     """
32     def __init__(self, url, login, password):
33         self.url = url
34         self.login = login
35         self.password = password
36
37     def __getattr__(self, methodname):
38         """
39         Crée à la volée une instance de Forwarder
40         correspondant à 'methodname'
41         """
42         return Forwarder(self, methodname)

```

```

1  from __future__ import print_function
2
3  # helpers - used for the verbose mode only
4  # could have been implemented as static methods in Position
5  # but we had not seen that at the time
6  def d_m_s(f):
7      """
8      make a float readable; e.g. transform 2.5 into 2.30'00''
9      we avoid using the degree sign to keep things simple
10     input is assumed positive
11     """
12     d = int (f)
13     m = int((f-d)*60)
14     s = int( (f-d)*3600 - 60*m)
15     return "{:02d}.{:02d}'{:02d}''".format(d,m,s)
16
17  def lat_d_m_s(f):
18     if f>=0:      return "{} N".format(d_m_s(f))
19     else:         return "{} S".format(d_m_s(-f))
20
21  def lon_d_m_s(f):
22     if f>=0:      return "{} E".format(d_m_s(f))
23     else:         return "{} W".format(d_m_s(-f))

```

```

1 class Position(object):
2     "a position atom with timestamp attached"
3
4     def __init__(self, latitude, longitude, timestamp):
5         "constructor"
6         self.latitude = latitude
7         self.longitude = longitude
8         self.timestamp = timestamp
9
10    # all these methods are only used when merger.py runs in verbose mode
11    def lat_str(self): return lat_d_m_s(self.latitude)
12    def lon_str(self): return lon_d_m_s(self.longitude)
13
14    def __repr__(self):
15        """
16        only used when merger.py is run in verbose mode
17        """
18        return "<{} {} @ {}>".format(self.lat_str(),
19                                       self.lon_str(), self.timestamp)

```

```

1 class Ship(object):
2     """
3     a ship object, that requires a ship id,
4     and optionnally a ship name and country
5     which can also be set later on
6
7     this object also manages a list of known positions
8     """
9     def __init__(self, id, name=None, country=None):
10         "constructor"
11         self.id = id
12         self.name = name
13         self.country = country
14         # this is where we remember the various positions over time
15         self.positions = []
16
17     def add_position(self, position):
18         """
19         insert a position relating to this ship
20         positions are not kept in order so you need
21         to call 'sort_positions' once you're done
22         """
23         self.positions.append(position)
24
25     def sort_positions(self):
26         """
27         sort list of positions by chronological order
28         """
29         self.positions.sort(key=lambda position: position.timestamp)

```

```

1 class ShipDict(dict):
2     """
3     a repository for storing all ships that we know about
4     indexed by their id
5     """
6     def __init__(self):
7         "constructor"
8         dict.__init__(self)
9
10    def __repr__(self):
11        return "<ShipDict instance with {} ships>".format(len(self))
12
13    def is_abbreviated(self, chunk):
14        """
15        depending on the size of the incoming data chunk,
16        guess if it is an abbreviated or extended data
17        """
18        return len(chunk) <= 7
19
20    def add_abbreviated(self, chunk):
21        """
22        adds an abbreviated data chunk to the repository
23        """
24        id, latitude, longitude, _, _, _, timestamp = chunk
25        if id not in self:
26            self[id] = Ship(id)
27        ship = self[id]
28        ship.add_position (Position (latitude, longitude, timestamp))
29
30    def add_extended(self, chunk):
31        """
32        adds an extended data chunk to the repository
33        """
34        id, latitude, longitude = chunk[:3]
35        timestamp, name = chunk[5:7]
36        country = chunk[10]
37        if id not in self:
38            self[id] = Ship(id)
39        ship = self[id]
40        if not ship.name:
41            ship.name = name
42            ship.country = country
43        self[id].add_position (Position (latitude, longitude, timestamp))

```



```

1  def add_chunk(self, chunk):
2      """
3      chunk is a plain list coming from the JSON data
4      and be either extended or abbreviated
5
6      based on the result of is_abbreviated(),
7      gets sent to add_extended or add_abbreviated
8      """
9      if self.is_abbreviated(chunk):
10         self.add_abbreviated(chunk)
11     else:
12         self.add_extended(chunk)
13
14     def sort(self):
15         """
16         makes sure all the ships have their positions
17         sorted in chronological order
18         """
19         for id, ship in self.iteritems():
20             ship.sort_positions()
21
22     def clean_unnamed(self):
23         """
24         Because we enter abbreviated and extended data
25         in no particular order, and for any time period,
26         we might have ship instances with no name attached
27         This method removes such entries from the dict
28         """
29         # we cannot do all in a single loop as this would amount to
30         # changing the loop subject
31         # so let us collect the ids to remove first
32         unnamed_ids = { id for id, ship in self.iteritems()
33                         if ship.name is None }
34         # and remove them next
35         for id in unnamed_ids:
36             del self[id]

```

```

1  def ships_by_name(self, name):
2      """
3      returns a list of all known ships with name <name>
4      """
5      return [ ship for ship in self.values() if ship.name == name ]
6
7  def all_ships(self):
8      """
9      returns a list of all ships known to us
10     """
11     return self.values()
12
13  def sort_ships_by_name (self, ships):
14     """
15     New in version 2.0
16
17     given a list of ships, returns a sorted version
18     this uses sorted() so a shallow copy is returned
19
20     sorting criteria is first on names, and then with
21     identical ship names use ship id instead
22
23     """
24     # to be completely deterministic, we cannot use only
25     # key=lambda ship: ship.name
26     # because of duplicate names in the fleet
27     # use good old cmp instead
28     def ship_compare (s1, s2):
29         return -1 if s1.name < s2.name \
30             else 1 if s1.name > s2.name \
31             else s1.id - s2.id
32     return sorted (ships, cmp = ship_compare)
33

```

regex\_pythonid - Semaine 6 Séquence 6

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

regex\_pythonid - Semaine 6 Séquence 6

```
1 # on peut aussi bien sûr l'écrire en clair
2 regex_pythonid2 = "[a-zA-Z_][a-zA-Z0-9_]*"
```

regex\_specials - Semaine 6 Séquence 6

```
1 # il faut commencer par exactement 2 underscores
2 # donc le caractère suivant doit être une lettre
3 # ensuite on peut mettre ce qu'on veut comme alphanumérique,
4 # mais avant les deux derniers underscores on ne peut pas avoir
5 # un underscore
6 # enfin pour traiter le cas où la partie centrale est réduite
7 # à un seul caractère, on met une option - avec ()?
8 regex_specials = "__[a-zA-Z](\w*[a-zA-Z0-9])?__"
```

```

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 backslaher . 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 regex_url = i_flag + protos + "://" + user + hostname + port + '/' + path

```