MOOC Python

Corrigés de la semaine 3

comptage - Semaine 3 Séquence 1

```
def comptage(in_filename, out_filename):
    """
    retranscrit le fichier in_filename dans le fichier out_filename
    en ajoutant des annotations sur les nombres de lignes, de mots
    et de caractères
    """
    # on ouvre le fichier d'entrée en lecture
    # on aurait pu mettre open(in_filename, 'r')
    with open(in_filename, encoding='utf-8') as input:
        # on ouvre la sortie en écriture
        with open(out_filename, "w", encoding='utf-8') as output:
            # initialisations
            total_words = 0
            total_chars = 0
            # pour toutes les lignes du fichier d'entrée
            # le numéro de ligne commence à 1
            for lineno, line in enumerate(input, 1):
                # autant de mots que d'éléments dans split()
                nb_words = len(line.split())
                total_words += nb_words
                # autant de caractères que d'éléments dans la ligne
                nb_chars = len(line)
                total_chars += nb_chars
                # on écrit la ligne de sortie; pas besoin
                # de newline (\n) car line en a déjà un
                output.write("{}:{}:{}:{}"
                             .format(lineno, nb_words, nb_chars, line))
            # on écrit la ligne de synthèse
            # lineno est une variable de boucle, elle "fuite"
            # on peut donc utiliser sa dernière valeur
            # mais remarquez que ce code ne fonctionnerait
            # pas sur un fichier vide, ou on aurait lineno non définie
            output.write("{}:{}:{}\n"
                         .format(lineno, total_words, total_chars))
```

index - Semaine 3 Séquence 4

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

index (bis) - Semaine 3 Séquence 4

```
def index_bis(bateaux):
    """
    La même chose mais de manière itérative
    """
    # si on veut décortiquer
    resultat = {}
    for bateau in bateaux:
        resultat [bateau[0]] = bateau
    return resultat
```

merge - Semaine 3 Séquence 4

```
def merge(extended, abbreviated):
    """
    Consolide des données étendues et des données abrégées
    comme décrit dans l'énoncé
    Le coût de cette fonction est linéaire dans la taille
    des données (longueur commune des deux listes)
    """
    # on initialise le résultat avec un dictionnaire vide
    result = {}
    # pour les données étendues
    # on affecte les 6 premiers champs
    # et on ignore les champs de rang 6 et au delà
    for id, latitude, longitude, timestamp, name, country, *ignore in extended:
        # on crée une entrée dans le résultat,
        # avec la mesure correspondant aux données étendues
        result[id] = [name, country, (latitude, longitude, timestamp)]
    # maintenant on peut compléter le résultat avec les données abrégées
    for id, latitude, longitude, timestamp in abbreviated:
        # et avec les hypothèses on sait que le bateau a déjà été
        # inscrit dans le résultat, donc result[id] doit déjà exister
        # et on peut se contenter d'ajouter ls mesure abrégée
        # dans l'entrée correspondant dans result
        result[id].append((latitude, longitude, timestamp))
    # et retourner le résultat
    return result
```

merge (bis) - Semaine 3 Séquence 4

```python
def merge_bis(extended, abbreviated):
    """
    Une deuxième version, linéaire également
    """
    # on initialise le résultat avec un dictionnaire vide
    result = {}
    # on remplit d'abord à partir des données étendues
    for ship in extended:
        id = ship[0]
        # on crée la liste avec le nom et le pays
        result[id] = ship[4:6]
        # on ajoute un tuple correspondant à la position
        result[id].append(tuple(ship[1:4]))
    # pareil que pour la première solution,
    # on sait d'après les hypothèses
    # que les id trouvées dans abbreviated
    # sont déja présentes dans le resultat
    for ship in abbreviated:
        id = ship[0]
        # on ajoute un tuple correspondant à la position
        result[id].append(tuple(ship[1:4]))
    return result
```

merge (ter) - Semaine 3 Séquence 4

```
def merge_ter(extended, abbreviated):
    """
    Une troisième solution
    à cause du tri que l'on fait au départ, cette
    solution n'est plus linéaire mais en O(n.log(n))
    """
    # ici on va tirer profit du fait que les id sont
    # en première position dans les deux tableaux
    # si bien que si on les trie,
    # on va mettre les deux tableaux 'en phase'
    #
    # c'est une technique qui marche dans ce cas précis
    # parce qu'on sait que les deux tableaux contiennent des données
    # pour exactement le même ensemble de bateaux
    #
    # on a deux choix, selon qu'on peut se permettre ou non de
    # modifier les données en entrée. Supposons que oui:
    extended.sort()
    abbreviated.sort()
    # si ça n'avait pas été le cas on aurait fait plutôt
    # extended = extended.sorted() et idem pour l'autre
    #
    # il ne reste plus qu'à assembler le résultat
    # en découpant des tranches
    # et en les transformant en tuples pour les positions
    # puisque c'est ce qui est demandé
    return {
        e[0] : e[4:6] + [ tuple(e[1:4]), tuple(a[1:4]) ]
        for (e,a) in zip (extended, abbreviated)
        }
```

parse_graph - Semaine 3 Séquence 4

```
from collections import defaultdict

def parse_graph(filename):
    g = defaultdict(list)
    with open(filename) as f:
        for line in f:
            begin, value, end = line.split()
            g[begin].append( (end, int(value)))
    return g
```

diff - Semaine 3 Séquence 5

```
def diff(extended, abbreviated):
    """Calcule comme demandé dans l'exercice, et sous formes d'ensembles
    (*) les noms des bateaux seulement dans extended
    (*) les noms des bateaux présents dans les deux listes
    (*) les ids des bateaux seulement dans abbreviated
    """
    ### on n'utilise que des ensembles dans tous l'exercice
    # les ids de tous les bateaux dans extended
    # une compréhension d'ensemble
    extended_ids = {ship[0] for ship in extended}
    # les ids de tous les bateaux dans abbreviated
    # idem
    abbreviated_ids = {ship[0] for ship in abbreviated}
    # les ids des bateaux seulement dans abbreviated
    # une difference d'ensembles
    abbreviated_only_ids = abbreviated_ids - extended_ids
    # les ids des bateaux dans les deux listes
    # une intersection d'ensembles
    both_ids = abbreviated_ids & extended_ids
    # les ids des bateaux seulement dans extended
    # ditto
    extended_only_ids = extended_ids - abbreviated_ids
    # pour les deux catégories où c'est possible
    # on recalcule les noms des bateaux
    # par une compréhension d'ensemble
    both_names = \
          {ship[4] for ship in extended if ship[0] in both_ids}
    extended_only_names = \
          {ship[4] for ship in extended if ship[0] in extended_only_ids}
    # enfin on retourne les 3 ensembles sous forme d'un tuple
    return extended_only_names, both_names, abbreviated_only_ids
```

diff (bis) - Semaine 3 Séquence 5

```
def diff_bis(extended, abbreviated):
    """
    Idem avec seulement des compréhensions
    """
    extended_ids =     {ship[0] for ship in extended}
    abbreviated_ids =  {ship[0] for ship in abbreviated}
    abbreviated_only = {ship[0] for ship in abbreviated
                        if ship[0] not in extended_ids}
    extended_only =    {ship[4] for ship in extended
                        if ship[0] not in abbreviated_ids}
    both =             {ship[4] for ship in extended
                        if ship[0] in abbreviated_ids}
    return extended_only, both, abbreviated_only
```