MOOC Python 3

Session 2018

Exercice shipdict

shipdict - Semaine 6 Séquence 4

```
1
2   # helpers - used for verbose mode only
3   # could have been implemented as static methods in Position
4   # but we had not seen that at the time
5
6
7   def d_m_s(f):
8       """
9       make a float readable; e.g. transform 2.5 into 2.30'00''
10      we avoid using the degree sign to keep things simple
11      input is assumed positive
12      """
13      d = int(f)
14      m = int((f - d) * 60)
15      s = int((f - d) * 3600 - 60 * m)
16      return "{:02d}.{:02d}'{:02d}''".format(d, m, s)
17
18
19  def lat_d_m_s(f):
20      """
21      degree-minute-second conversion on a latitude float
22      """
23      if f >= 0:
24          return "{} N".format(d_m_s(f))
25      else:
26          return "{} S".format(d_m_s(-f))
27
28
29  def lon_d_m_s(f):
30      """
31      degree-minute-second conversion on a longitude float
32      """
33      if f >= 0:
34          return "{} E".format(d_m_s(f))
35      else:
36          return "{} W".format(d_m_s(-f))
```

shipdict_suite - Semaine 6 Séquence 4

```
1
2
3   class Position(object):
4       "a position atom with timestamp attached"
5
6       def __init__(self, latitude, longitude, timestamp):
7           "constructor"
8           self.latitude = latitude
9           self.longitude = longitude
10          self.timestamp = timestamp
11
12  # all these methods are only used when merger.py runs in verbose mode
13      def lat_str(self):
14          return lat_d_m_s(self.latitude)
15
16      def lon_str(self):
17          return lon_d_m_s(self.longitude)
18
19      def __repr__(self):
20          """
21          only used when merger.py is run in verbose mode
22          """
23          return f"<{self.lat_str()} {self.lon_str()} @ {self.timestamp}>"
24
25      # required to be stored in a set
26      # see https://docs.python.org/3/reference/datamodel.html#object.__hash__
27      def __hash__(self):
28          return hash((self.latitude, self.longitude, self.timestamp))
29
30      # a hashable shall override this special method
31      def __eq__(self, other):
32          return (self.latitude == other.latitude
33                  and self.longitude == other.longitude
34                  and self.timestamp == other.timestamp)
```

shipdict_suite - Semaine 6 Séquence 4

```
1
2
3  class Ship(object):
4      """
5      a ship object, that requires a ship id,
6      and optionnally a ship name and country
7      which can also be set later on
8
9      this object also manages a list of known positions
10     """
11
12     def __init__(self, id, name=None, country=None):
13         "constructor"
14         self.id = id
15         self.name = name
16         self.country = country
17         # this is where we remember the various positions over time
18         self.positions = []
19
20     def add_position(self, position):
21         """
22         insert a position relating to this ship
23         positions are not kept in order so you need
24         to call `sort_positions` once you're done
25         """
26         self.positions.append(position)
27
28     def sort_positions(self):
29         """
30         sort of positions made unique thanks to the set by chronological order
31         for this to work, a Position must be hashable
32         """
33         self.positions = sorted(set(self.positions),
34                                 key=lambda position: position.timestamp)
```

shipdict_suite - Semaine 6 Séquence 4

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

shipdict_suite - Semaine 6 Séquence 4

```
    def add_chunk(self, chunk):
        """
        chunk is a plain list coming from the JSON data
        and be either extended or abbreviated

        based on the result of is_abbreviated(),
        gets sent to add_extended or add_abbreviated
        """
        # here we retrieve the static method through the class
        # this form outlines the fact that we're calling a static method
        # note that
        # self.is_abbreviated(chunk)
        # would work fine just as well
        if ShipDict.is_abbreviated(chunk):
            self.add_abbreviated(chunk)
        else:
            self.add_extended(chunk)

    def sort(self):
        """
        makes sure all the ships have their positions
        sorted in chronological order
        """
        for id, ship in self.items():
            ship.sort_positions()

    def clean_unnamed(self):
        """
        Because we enter abbreviated and extended data
        in no particular order, and for any time period,
        we might have ship instances with no name attached
        This method removes such entries from the dict
        """
        # we cannot do all in a single loop as this would amount to
        # changing the loop subject
        # so let us collect the ids to remove first
        unnamed_ids = {id for id, ship in self.items()
                       if ship.name is None}
        # and remove them next
        for id in unnamed_ids:
            del self[id]
```

shipdict_suite - Semaine 6 Séquence 4

```
    def ships_by_name(self, name):
        """
        returns a list of all known ships with name <name>
        """
        return [ship for ship in self.values() if ship.name == name]

    def all_ships(self):
        """
        returns a list of all ships known to us
        """
        # we need to create an actual list because it
        # may need to be sorted later on, and so
        # a raw dict_values object won't be good enough
        return list(self.values())

```