TP.2

[Dictionnaires et Ellipses - écriture de fichiers]

L'objectif de ce TP consiste à compléter des fonctions qui permettront de gérer les paramètres (et propriétés d'une ellipse). La classe `SuperEllipse` n'est en fait qu'un dictionnaire. Un objet `SuperEllipse` pourra être rempli à partir d'un dictionnaire ou d'un fichier. On pourra ajouter des caractéristiques/spécifications à une `SuperEllipse` mais aussi l'écrire dans un fichier. Pour cela, on va compléter un code source existant en réalisant quelques fonctions simples.

I . Questions préliminaires

Avant de commencer, il faudra pouvoir répondre à quelques questions.
Pour cela, il faudra lire et comprendre le code source fourni (voir Listing 1 et Listing 2).

Listing 1 – code source de `main_superEllipse.py`.

```
1
2 from copy import deepcopy
3 import matplotlib.pyplot as plt
4
5 from superEllipse import creerEtOuEffacerUnFichier, afficherUnFichier
6 from superEllipse import SuperEllipse
7
8 # trois ellipses vides et de quoi les remplir
9 # ------------------------------------------
10 ellipse_1 = SuperEllipse()
11 ellipse_1_specificationOriginales = {'xy':(1,2)}
12 ellipse_1_fichier_sauvegarde = "ellipse_1_bis.txt"
13
14 ellipse_2 = SuperEllipse()
15 ellipse_2_specificationOriginales = "unFichierContenantUneSuperEllipse.txt"
16 ellipse_2_specifications_ajout_1 = {"color":'r',"linestyle":':'}
17 ellipse_2_specifications_ajout_2 = {"alpha":.6,"linewidth":4}
18 ellipse_2_fichier_sauvegarde = "ellipse_2_bis.txt"
19
20 ellipse_3 = SuperEllipse()
21 ellipse_3_specificationOriginales = {"xy":(-1,-2),"color":'b',"alpha":.7}
22 ellipse_3_fichier_sauvegarde  = "unFichierVideARemplir.txt"
23
24
25 # on charge une SuperEllipse a partir d'un dictionnaire
26 # ----------------------------------------------------
27 print("\n\nChargement d'une ellipse a partir d'un dictionnaire :")
28 print("  ellipse_1 ............ (cree vide) :",ellipse_1)
29 ellipse_1.aPartirDunDictionnaire(ellipse_1_specificationOriginales)
30 print("  ellipse_1 (remplie dictionnaire) :",ellipse_1)
31
32
33 # Que se passe-t-il quand on utilise "=" ?
34 print("""\nQue se passe-t-il quand on utilise "=" ? : """)
35 ellipse_1bis = ellipse_1
36 print("  id(ellipse_1):",id(ellipse_1))
37 print("  id(ellipse_1bis) :",id(ellipse_1bis))
38 del ellipse_1bis
39
40
41 # on charge une SuperEllipse a partir d'un fichier
42 # -----------------------------------------------
43 print("\n\nChargement d'une SuperEllipse a partir d'un fichier :")
44 print("  ellipse_2  (cree vide) :",ellipse_2)
45 ellipse_2.aPartirDUnNomDeFichier(ellipse_2_specificationOriginales)
46 print("  ellipse_2  (remplie fichier) :",ellipse_2)
47
48
49 # Ajout un element a la fois
```

```
50 # ------------------------------
51 print("\n\nAjouts␣une␣specification␣Ãă␣la␣fois␣:␣")
52 print("␣␣ellipse_1␣␣(avant␣ajout)␣:␣",ellipse_1)
53 ellipse_1.ajouterUneSpecification("angle")
54 ellipse_1.ajouterUneSpecification("color",'m')
55 print("␣␣ellipse_1␣␣(ajout␣angle␣et␣couleur)␣:␣",ellipse_1)
56
57
58 # Ecriture d'ellipse_1 dans un fichier
59 # ------------------------------------
60 print("\n\nEcriture␣de␣Ajout,␣ellipse_1␣dans␣le␣fichier" +
61       """ "{}".""".format(ellipse_1_fichier_sauvegarde))
62 ellipse_1.ecrireDansUnFichier(ellipse_1_fichier_sauvegarde)
63
64
65 # Ajout de plusieurs specifications
66 # ---------------------------------
67 print("\n\nAjout␣de␣plusieurs␣specifications␣simultanee:␣")
68 print("␣␣ellipse_2␣␣(avant␣ajout)␣:␣",ellipse_2)
69 ellipse_2.ajouterDesSpecifications(ellipse_2_specifications_ajout_1)
70 ellipse_2.ajouterDesSpecifications(ellipse_2_specifications_ajout_2)
71 print("␣␣ellipse_2␣␣(ajout␣color+linestyle␣alpha+linewidth)␣:␣",ellipse_2)
72
73
74 # Ecriture d'ellipse_1 dans un fichier
75 # ------------------------------------
76 print("\n\nEcriture␣de␣Ajout,␣ellipse_2␣dans␣le␣fichier" +
77       """ "{}".""".format(ellipse_2_fichier_sauvegarde))
78 ellipse_2.ecrireDansUnFichier(ellipse_2_fichier_sauvegarde)
79
80
81 # Modification d'un fichier contenant une ellipse
82 # -----------------------------------------------
83 print("""\n\nEcrasemant du fichier "{}".""".format(ellipse_3_fichier_sauvegarde))
84 creerEtOuEffacerUnFichier(ellipse_3_fichier_sauvegarde)
85 print("␣␣fichier␣avant␣␣␣␣␣␣␣␣␣(vide)␣:␣")#, end='')
86 afficherUnFichier(ellipse_3_fichier_sauvegarde)
87
88 # ajout au fichier
89 ellipse_2.ajouterAUnFichier(ellipse_3_fichier_sauvegarde)
90 print("␣␣fichier␣apres␣␣␣␣␣␣(ajout␣1)␣:␣")#, end='')
91 afficherUnFichier(ellipse_3_fichier_sauvegarde)
92
93 # creation de la liste de la SuperEllipse
94 ellipse_3.aPartirDunDictionnaire(ellipse_3_specificationOriginales)
95 ellipse_3.ajouterAUnFichier(ellipse_3_fichier_sauvegarde)
96 print("␣␣fichier␣apres␣␣␣␣␣␣(ajout␣2)␣:␣")#, end='')
97 afficherUnFichier(ellipse_3_fichier_sauvegarde)
98
99
100 print("ellipse_1:␣",ellipse_1)
101 print("ellipse_2:␣",ellipse_2)
102 print("ellipse_3:␣",ellipse_3)
103
104 plt.figure()
105 try:
106     ellipse_1.afficher()
107     ellipse_2.afficher()
108     ellipse_3.afficher()
109 except TypeError:
110     print("Il␣va␣falloir␣completer.")
111 plt.xlim([-5,5])
112 plt.ylim([-5,5])
113
114 plt.show()
```

Ici on s'intéresse plus particulièrement au fichier `main_superEllipse.py` (voir Listing 1)

1. A quoi servent les ligne 2 et 3 du fichier `main_superEllipse.py` ? Expliquer.

2. Identifier la ligne du code à laquelle on crée la première `SuperEllipse`.
3. `SuperEllipse` est une classe. Combien de méthodes différentes de cette classe sont-elles appelées ? (Les identifier.)

A la ligne 84 du fichier `main_superEllipse.py`, on fait appel à la fonction `creerEtOuEffacerUnFichier` en lui fournissant pour argument une chaine de caractères (`ellipse_3_fichier_sauvegarde`).
A la ligne 86 du fichier `main_superEllipse.py`, on fait appel à la fonction `affichierUnFichier` en lui fournissant pour argument une chaine de caractères (`ellipse_3_fichier_sauvegarde`).

4. D'où viennent ces deux fonctions ? Sont-elles des méthodes de la classe `SuperEllipse` ? Expliquer.

A partir d'ici, on va s'intéresser plus particulièrement au fichier `superEllipse.py` (voir Listing 2)

Listing 2 – code source (à compléter) de superEllipse.py.

```
1
2  import warnings as wrn
3  from copy import deepcopy
4  import matplotlib as mpl
5  import matplotlib.pyplot as plt
6  import matplotlib.transforms as transforms
7  from    matplotlib.patches import Ellipse
8
9
10 def creerEtOuEffacerUnFichier(unNomDeFichier):
11     fio = open(unNomDeFichier,'w')
12     fio.close()
13
14 def afficherUnFichier(unNomDeFichier):
15     print("\n----------------------------------")
16     print("| ",unNomDeFichier," |")
17     print("----------------------------------")
18     with open(unNomDeFichier, 'r') as file:
19         print(file.read(), end='')
20     print("----------------------------------\n")
21
22
23
24 class SuperEllipse:
25     parameters = ['xy', 'width', 'height', 'angle']
26     properties = ['agg_filter', 'value', 'alpha', 'animated',
27                   'antialiased', 'capstyle', 'clip_box', 'clip_on',
28                   'clip_path', 'color', 'contains', 'edgecolor',
29                   'facecolor', 'figure', 'fill', 'gid',  'hatch',
30                   'in_layout', 'joinstyle', 'label', 'linestyle',
31                   'linewidth', 'path_effects', 'picker',
32                   'rasterized', 'sketch_params', 'snap',
33                   'transform', 'url', 'visible', 'zorder']
34
35     def __init__(self):
36         self.specifications = dict()
37
38     def __str__(self):
39         "pour avoir une Ellipse joliment affichee"
40         return self.specifications.__str__()
41
42     def clear(self):
43         self.specifications.clear()
44
45     def aPartirDUnNomDeFichier(self, filename):
46         '''
47         Pour initialiser la SuperEllipse a partir d'un fichier
48         '''
49         # on ouvre le fichier et on charge les lignes
50         # chaque ligne correspond a un specification
51         # i.e. un couple (clef,valeur) du dictionnaire
52         #
53         # Remarque :
54         # * Dans le fichier, on lit les caractÃÍres qu'il contient.
55         # * On convertit un peu comme on le peut les chaÃőnes de caractÃÍres lues en entier et en flo
56         #
57         with open(filename) as f:
```

```
58              for ligne in f:
59                  items = ligne.split('␣;␣')
60                  assert len(items) == 2
61                  key, valuesList = items[0], items[1]
62                  values = valuesList.split(',')
63                  values[-1] = values[-1][0:-1]
64                  if len(values) == 1:
65                      try :
66                          values[0]  = int(values[0])
67                      except ValueError:
68                          try :
69                              values[0] = float(values[0])
70                          except ValueError:
71                              pass
72                      self.specifications[key] = values[0]
73                  else:
74                      for i in range(len(values)):
75                          try :
76                              values[i] = int(values[i])
77                          except ValueError:
78                              try :
79                                  values[i] = float(values[i])
80                              except ValueError:
81                                  pass
82                      self.specifications[key] = values
83              return self
84
85      def aPartirDunDictionnaire(self, datadict):
86          "Pour␣initialiser␣une␣superEllipse␣a␣partir␣d'un␣dictionnaire"
87          # on copie datadict dans l'attribut specifications
88          # attention a la nature de la copie
89
90          return self
91
92      def ecrireDansUnFichier(self,filename):
93          "Pour␣ecrire␣la␣liste␣dans␣un␣fichier"
94          # on ouvre un fichier
95          # on ecrit chaque specification sur une ligne ligne
96          # on ferme le fichier
97          #
98          # Remarques :
99          #  * On fera quelque chose qui est coherent avec ce qui est ecrit pour la lecture
100         #  * L'ecriture est beaucoup plus simple que la lecture
101         #
102
103
104
105         pass
106
107     def ajouterAUnFichier(self,nomDeFichier):
108         """
109            Pour completer le fichier avec des specifications supplementaires.
110         """
111         # on charge le fichier dans un dictionnaire
112         # on ajoute les specifications i.e. les couples (clef,valeur)
113         # on ecrase le fichier precedent
114
115
116
117
118         pass
119
120     def ajouterUneSpecification(self,specificationAAjouter,qualite=None):
121         '''
122            Pour ajouter une spÃ©cification a la superEllipside
123         '''
124         # tester si la spÃ©cification existe
125         # si elle existe et qu'elle est diffÃ©rente complete on emet un message d'alerte
126         # avec quelque chose du type wrn.warn("Pour le champs {}, on remplace {} par {}"....
```

```
127         # En ajoute la spÃľcifiation (en ecrasant celle qui existait si elle existait)
128         #
129
130
131
132
133         pass
134
135     def ajouterDesSpecifications(self,dictEnPlus):
136         '''
137            Pour ajouter plusieurs specifications a une SuperEllipse
138         '''
139         # Pour chacun des elements de dictEnPlus
140         # et ajout de chaque couple (clef,valeur) comme une specification
141
142
143         pass
144
145     def estElleCoherente(self):
146         # On verifie que les tous les parametres sont presents dans specification
147         # On verifie qu'une specification est soit un parametre soit une propriete
148
149
150         return False
151
152     def ajouterUneValeurParDefaut(self,uneSpecification):
153         if uneSpecification == 'xy':
154             self.ajouterUneSpecification(uneSpecification,(0,0))
155         elif uneSpecification == 'width':
156             self.ajouterUneSpecification(uneSpecification,1)
157         elif uneSpecification == 'height':
158             self.ajouterUneSpecification(uneSpecification,2)
159         elif uneSpecification == 'angle':
160             self.ajouterUneSpecification(uneSpecification,0)
161         elif uneSpecification == 'facecolor':
162             self.ajouterUneSpecification(uneSpecification,'k')
163         elif uneSpecification == 'linewidth':
164             self.ajouterUneSpecification(uneSpecification,1)
165         else:
166             raise ValueError("On n'a pas de valeur par defaut pour '{}'.".format(self.specifications[
167
168     def supprimerUneSpecification(self,uneSpecification):
169         del self.specifications[uneSpecification]
170
171     def nettoyage(self):
172         # on supprime les specification ne correspondant pas a des parametres/proprietes connus
173         # on ajoute les parametres manquants
174
175
176         pass
177
178
179     def afficher(self):
180         # on nettoye i.e. appel de la methode nettoyage
181         # A partir de l'attribut specifications on cree un objet de type matplotlib.patches.Ellipse
182         # on l'ajoute a l'axe courant avec la commande : plt.gca().add_patch(ellipse)
183         self.nettoyage()
184         ellipse = mpl.patches.Ellipse(**self.specifications)
185         plt.gca().add_patch(ellipse)
```

5. Executer le code source fourni. (Vérifier qu'il n'indique aucune erreur.)
6. Combien de fonctions sont-elles incomplètes ?
7. Combien de fonctions ont-elles une valeur de retour ? Que cela implique-t-il pour les autres ?
8. Que modifie la fonction `aPartirDunDictionnaire` ? A quoi faut-il faire attention ? Expliquer.
9. Cette fonction à un argument dénommé `self`. A quoi fait-il référence ?

Dans la suite, on demandera de compléter des fonctions permettant de lire et d'écrire dans des fichiers. On demandera aussi de travailler avec des dictionnaires (type `dict`).

10. Lire la fonction `affichierUnFichier` du fichier `superEllipse.py` (voir Listing 2) et expliquer la ligne 18 de ce fichier.
11. Expliquer ce qu'est un dictionnaire.

II . classe SuperEllipse

On va maintenant s'intéresser à la réalisations des fonctions de la classe `SuperEllipse`.
On devra autant que possible utiliser les fonctions déjà réalisées pour éviter de réécrire plusieurs fois la même chose.

12. Implémenter la fonction `aPartirDunDictionnaire` qui remplira/écrasera un objet `SuperEllipse` à partir du contenu d'un dictionnaire.
13. Vérifier par l'intermédiaire de `main_superEllipse.py` que le résultat est conforme avec ce qui est attendu.
14. Implémenter la fonction `ajouterUneSpecification` qui ajoute une caractéristique/spécification à un objet `SuperEllipse`.
15. Implémenter la fonction `ajouterDesSpecifications`. (On se servira de fonctions existantes.)

(On ne demandera plus de vérifier que le résultat est conforme avec ce qui est attendu. Cela devra tout de même être fait.)

16. Implémenter la fonction `ecrireDansUnFichier` qui écrit une `SuperEllipse` dans un fichier. (On s'inspirera de la fonction `aPartirDUnNomDeFichier`.)
17. Implémenter la fonction `ajouterAUnFichier` (On se servira de fonctions existantes.)

On veut afficher une ellipse à partir d'un objet `SuperEllipse`. Pour cela on va transmettre les informations de l'objet `SuperEllipse` à un objet `Ellipse` fourni par le module `matplotlib` que l'on pourra ensuite afficher.
La méthode `afficher` existe déjà et fait appel à deux méthodes qui seront réalisées dans la suite.

18. Réaliser la fonction `estElleCoherente` qui vérifie que les `specifications` sont cohérentes avec les attributs `parameters` et `properties` de la classe.
    La fonction retournera `False` si les éléments de `parameters` ne sont pas tous contenus dans `specifications` ou si `specifications` possède des clefs non contenues dans `properties` ou `parameters`.
19. `nettoyage` ajoute les paramètres manquant et supprime les spécifications ne correspondant pas à des propriétés.

III . Une figure pleine d'ellipses aléatoire

Ajouter une ou des méthodes permettant de générer une figure contenant des représentations d'ellipses aléatoires.