```
 1 import math
 2 from Lib.NoMatchFoundError import NoMatchFoundError
 3
 4
 5 class lookupthings():
 6     def __init__(self):
 7         self.x = [[],[],[],[], []]
 8         self.y
    = [[[],[],[],[]], [[],[],[],[]], [[],[],[],[]], [[],
   [],[],[]], [[],[],[],[]]]
 9         self.criterion = 0.25
10         self.criterion_Test_in_ideal4 = 0.7
11         self.Anzahl_Spalten = 5
12
13     def lookup_train_in_ideal(self, source,
   destination):       # source: Instanz train_data;
   destination: Instanz ideal_data
14         """
15         zu den trainingsdaten passende Funktionen in
   den 50 idealen Funktionen finden
16         :param source: Quelldatei -> trainings Daten
17         :param destination: Zieldatei -> ideal Daten
18         """
19
20         # Header schreiben
21         self.x[0] = 'x'
22         for i in range(1, 5):
23             self.y[i][0].append('x')
24             self.y[i][1].append('y')
25             self.y[i][2].append('Differenz')
26             self.y[i][3].append('QuellFunktion')
27
28         # alle yse in Quelle (train, y1 bis y4)
29         for pos_outer_src in range(1,len(source.y)):
30
31             # alle x in Quelle (train_data, x 1 bis
   400)
32             for pos_inner_dest in range(1, len(source
   .x)):
33                 pos_inner_src = pos_inner_dest
34
```

```
35                  # alle y in Ziel (50 ideale) in x Richtung durchlaufen 1 - 50
36                  for pos_outer_dest in range(1, len(destination.y)):
37                      middleValue = (destination.y[pos_outer_dest][pos_inner_dest] + source.y[pos_outer_src][pos_inner_src])/2        # Mittelwert
38                      difference = math.sqrt(0.5 * ((source.y[pos_outer_src][pos_inner_src] - middleValue)**2 + (destination.y[pos_outer_dest][pos_inner_dest] - middleValue)**2))        # quadr. mittlere Abweichung
39                      if difference <= self.criterion:
40                          self.x[pos_outer_src].append(source.x[pos_inner_src])
41                          self.y[pos_outer_src][0].append(source.x[pos_inner_src])        # x-Wert schreiben in y Liste
42                          self.y[pos_outer_src][1].append(destination.y[pos_outer_dest][pos_inner_dest])      # y-Wert schreiben
43                          self.y[pos_outer_src][2].append(difference)     # Differenz schreiben
44                          self.y[pos_outer_src][3].append(destination.y[pos_outer_dest][0])       # Name der Quellfunktion
45                          pass
                                # if Wert <= Sqrt(2)
46                      pass
                                    # y dest
47                  pass
                                        # x_xrc
48              pass
49
50          self.cleanup()  # bereinigen
51
52          print("DONE LOOKUP")
53
54
55      def lookup_test_in_ideal4(self, source, destination):
```

```
56          """
57          testdaten mit den 4 gefundenen idealen
   Funktionen vergleichen
58          :param source: testdaten
59          :param destination: 4 ideale funktionen
60          """
61          self.x = []
62          self.y = [[], [], [], []]
63          # Ueberschriften anlegen
64          self.y[0].append('x')
65          self.y[1].append('y')
66          self.y[2].append('Differenz')
67          self.y[3].append('QuellFunktion')
68          found = False
69
70          # lenFileContent = len(source.file_content)
71          for i in range(len(source.file_content)):
72              xTest = source.file_content[i][0]   # x
   Wert suchen
73
74              for j in range(1, len(destination.y[0])):
75                  xDest = destination.y[0][j]
76                  found = False
77                  if destination.y[0][j] == xTest:
78                      for z in range(1, len(destination
   .y)):  # für alle y se in y bei x
79                          yTest = source.file_content[i
   ][1]
80                          middleValue = (destination.y[
   z][j] + source.file_content[i][1]) / 2.0
81                          difference = math.sqrt(0.5
    * ((source.file_content[i][1] - middleValue) ** 2
    + (destination.y[z][j] - middleValue) ** 2))  #
   quadr. mittlere Abweichung
82                          try:
83                              if difference < self.
   criterion_Test_in_ideal4:
84                                  self.y[0].append(
   xTest)
85                                  self.y[1].append(
   yTest)
```

```
 86                                      self.y[2].append(
    difference)
 87                                      self.y[3].append(
    destination.y[z][0])
 88                                      found = True
 89                                  elif found == False and
    z == len(destination.y)-1:
 90                                      raise
    NoMatchFoundError     # benutzerdefinierte Excpetion
 91
 92                              except NoMatchFoundError:
 93                                  print(NoMatchFoundError
    ().message)
 94
 95                              pass
 96                  else:
 97                      # print("else Pfad")
 98                      if xDest > xTest:
 99                          break       # Schleife
    beenden, wenn x aus Destination > als x aus
    Testdaten, da Daten sortiert sind
100                      pass
101                  pass
102          self.x = self.y[0]
103          self.Anzahl_Spalten = len(self.y)
104
105
106      def cleanup(self):
107          """
108          Funktion bestimmt die am häufigsten
    gefundene ideale Funktion und löscht die nicht
    zugehörigen
109          """
110          i = 0
111          for j in range(1, len(self.y)):     #
    durchlaufe alle y 1-5
112              # die Funktion mit den meisten Matches
    ermitteln
113              zwischending = [[], []]
114              for z in range(1, 51):  # bei 1 starten
    wegen Namen y1 bis y50
```

```
115                 zwischending[0].append('y' + str(z
    ))    # interimsarray mit Funktionsnamen
116                 zwischending[1].append(self.y[j][3].
    count('y' + str(z)))    # zugehörige Anzahl der
    gefundenen Matches
117 
118             max_value = max(zwischending[1])    #
    Maximale Anzahl
119             max_value_index = zwischending[1].index(
    max_value)  # Index der maximalen Anzahl
120             search_value = zwischending[0][
    max_value_index]     # hier steht die ermittelte am
    häufigsten gefundene Funktion
121 
122             for i in range(len(self.y[j][3]), 1, -1
    ):   # lösche die anderen Werte der Funktionen, die
    nicht am häufigsten enthalten sind
123                 if self.y[j][3][i-1] != search_value
    :   # zaehle von hinten her
124                     for todelete in range(len(self.y
    [j])):
125                         del self.y[j][todelete][i-1
    ]    # von i muss 1 subtrahiert werden, da hier der
    Index benötigt wird in i ist aber die gesamte Anzahl
     an Stellen enthalten
126 
127         # Ab hier cleanup um Funktionen createTable
    und dataToTable nutzen zu können -> umformatieren in
     eindimensional (x) und zweidimensional (y)
128         for i in range(1, 5):           #
    Funktionsnamen in y Spalte Platz [0] schreiben als
    Überschrift
129             self.y[i][1][0] = self.y[i][3][1]
130 
131         self.x = []       # x löschen, indem neu
    definiert wird
132         self.x = self.y[1][0]   # es werden die x-
    Werte aus y[0] in neues x geschrieben
133 
134         for i in range(5):
135             del self.y[i][3]        # lösche
```

```
135 Quellfunktion
136             del self.y[i][2]        # lösche
    Differenz
137
138         self.y[0][0] = self.y[1][0]       # x Werte
    in nulltes Array schreiben
139         del self.y[0][1]
140         self.y[0] = self.y[0][0]
141         del self.y[1][0]
142         self.y[1] = self.y[1][0]
143         del self.y[2][0]
144         self.y[2] = self.y[2][0]
145         del self.y[3][0]
146         self.y[3] = self.y[3][0]
147         del self.y[4][0]
148         self.y[4] = self.y[4][0]
149
150         print("DONE cleanup")
```