```
#
# Turtle3D.py
#
# Implementation of a 3D Turtle in Python.
# Author: Eric G. Suchanek, PhD
# A part of the program Proteus, https://github.com/suchanek/proteus,
# a program for the manipulation and analysis of macromolecules
# Based on the C implementation originally authored by Eric G. Suchanek PhD
1990
#

import numpy
import math

from Bio.PDB.vectors import Vector
from Bio.PDB.vectors import calc_angle, calc_dihedral

from  proteusPy.proteusGlobals import *

_DOWN_ = -1
_UP_ = 1
_ORIENTATION_INIT = -1

"""
    Return a (left multiplying) matrix that rotates p onto q.

    :param p: moving vector
    :type p: L{Vector}

    :param q: fixed vector
    :type q: L{Vector}

    :return: rotation matrix that rotates p onto q
    :rtype: 3x3 Numeric array

    Examples
    --------
    >>> from Bio.PDB.vectors import rotmat
    >>> p, q = Vector(1, 2, 3), Vector(2, 3, 5)
    >>> r = rotmat(p, q)
    >>> print(q)
    <Vector 2.00, 3.00, 5.00>
    >>> print(p)
    <Vector 1.00, 2.00, 3.00>
    >>> p.left_multiply(r)
    <Vector 1.21, 1.82, 3.03>

    """
```

```
48
49 # Class Definition begins
50
51 class Turtle3D:
52     """3D Turtle."""
53
54     def __init__(self, name="3D_Turtle"):
55         # internal rep is arrays
56         self._position = numpy.array((0.0, 0.0, 0.0), "d")
57         self._heading = numpy.array((1.0, 0.0, 0.0), "d")
58         self._left = numpy.array((0.0, 1.0, 0.0), "d")
59         self._up = numpy.array((0.0, 0.0, 1.0), "d")
60
61         # expose these as Vectors
62         self.position = Vector(0.0, 0.0, 0.0)
63         self.heading = Vector(1.0, 0.0, 0.0)
64         self.left = Vector(0.0, 1.0, 0.0)
65         self.up = Vector(0.0, 0.0, 1.0)
66
67         self._name = name
68         self._pen = _UP_
69         self._orientation = _ORIENTATION_INIT # will be set to 1 or 2 later
… when used for residue building
70         self._recording = False
71         self._tape = []
72
73     def copy_coords(self, source):
74         '''
75         Copy the Position, Heading, Left and Up coordinate system from the
… input source into self.
76         Argument: source: Turtle3D
77         Returns: None
78         '''
79
80         # copy the Arrays
81         self._position = source._position.copy()
82         self._heading = source._heading.copy()
83         self._left = source._left.copy()
84         self._up = source._up.copy()
85
86         # copy the Vectors - create new ones from the source arrays
87         self.position = Vector(source._position)
88         self.heading = Vector(source._heading)
89         self.left = Vector(source._left)
90         self.up = Vector(source._up)
91
92         self._orientation = source._orientation
93
```

```
94      def reset(self):
95          """
96          Reset the Turtle to be at the Origin, with correct Heading, Left
... and Up vectors.
97          Arguments: None
98          Returns: None
99          """
100         self._position = numpy.array((0.0, 0.0, 0.0), "d")
101         self._heading = numpy.array((1.0, 0.0, 0.0), "d")
102         self._left = numpy.array((0.0, 1.0, 0.0), "d")
103         self._up = numpy.array((0.0, 0.0, 1.0), "d")
104
105         # expose these as Vectors
106         self.position = Vector(0.0, 0.0, 0.0)
107         self.heading = Vector(1.0, 0.0, 0.0)
108         self.left = Vector(0.0, 1.0, 0.0)
109         self.up = Vector(0.0, 0.0, 1.0)
110
111         self._pen = _UP_
112         self._orientation = -1
113         self._recording = False
114         self._tape = []
115
116     def Orientation(self):
117         return self._orientation
118
119     def set_orientation(self, orientation):
120         assert orientation == ORIENT_BACKBONE or orientation ==
... ORIENT_SIDECHAIN, f'Orientation must be {ORIENT_BACKBONE} or
... {ORIENT_SIDECHAIN}'
121         self._orientation = orientation
122
123     def Pen(self):
124         if self._pen == _UP_:
125             return('UP')
126         else:
127             return('Down')
128
129     def PenUp(self):
130         self._pen = _UP_
131
132     def PenDown(self):
133         self._pen = _DOWN_
134
135     def Recording(self):
136         return self._recording
137
138     def RecordOn(self):
```

```
139         self._recording = True
140 
141     def RecordOff(self):
142         self._recording = False
143 
144     def ResetTape(self):
145         self._tape = []
146 
147     def setPosition(self, x, y=None, z=None):
148         """
149         Set the Turtle's Position.
150             X is either a list or tuple.
151         """
152         if y is None and z is None:
153             # Array, list, tuple...
154             if len(x) != 3:
155                 raise ValueError("Turtle3D: x is not a list/tuple/array of
    … 3 numbers")
156             self._position = numpy.array(x, "d")
157         else:
158             # Three numbers
159             self._position = numpy.array((x, y, z), "d")
160 
161         self.position = Vector(self._position)
162         return
163 
164     def getPosition(self) -> numpy.array:
165         """
166         Get the Turtle's Position.
167         Return: Turtle's position (Array)
168         """
169         return(self._position)
170 
171     def getVPosition(self) -> Vector:
172         """
173         Get the Turtle's Position.
174         Return: Turtle's position (Array)
175         """
176         return(self._position)
177 
178     def getName(self):
179         """
180         Get the Turtle's Position.
181         """
182         return(self._name)
183 
184     def setName(self, name):
185         """
```

```
186         Set the Turtle'Name.
187 
188         """
189         self._name = name
190 
191     def move(self, distance):
192         """
193         Move the Turtle distance, in direction of Heading
194         """
195         self._position = self._position + self._heading * distance
196 
197     def roll(self, angle):
198         """
199         Roll the Turtle about the heading vector angle degrees
200         """
201 
202         ang = angle * math.pi / 180.0
203         cosang = numpy.cos(ang)
204         sinang = numpy.sin(ang)
205 
206         self._up[0] = cosang * self._up[0] - sinang * self._left[0]
207         self._up[1] = cosang * self._up[1] - sinang * self._left[1]
208         self._up[2] = cosang * self._up[2]- sinang * self._left[2]
209 
210         self._left[0] = cosang * self._left[0] + sinang * self._up[0]
211         self._left[1] = cosang * self._left[1] + sinang * self._up[1]
212         self._left[2] = cosang * self._left[2] + sinang * self._up[2]
213 
214     def yaw(self, angle):
215         """
216         Yaw the Turtle about the up vector (180 - angle) degrees. This is
 … used when building molecules
217         """
218 
219         ang = ((180 - angle) * math.pi) / 180.0
220         cosang = numpy.cos(ang)
221         sinang = numpy.sin(ang)
222 
223         self._heading[0] = cosang * self._heading[0] + sinang *
 … self._left[0]
224         self._heading[1] = cosang * self._heading[1] + sinang *
 … self._left[1]
225         self._heading[2] = cosang * self._heading[2] + sinang *
 … self._left[2]
226 
227         self._left[0] = cosang * self._left[0] - sinang * self._heading[0]
228         self._left[1] = cosang * self._left[1] - sinang * self._heading[1]
229         self._left[2] = cosang * self._left[2] - sinang * self._heading[2]
```

```
230
231     def turn(self, angle):
232         """
233         Turn the Turtle about the up vector angle degrees.
234         """
235
236         ang = (angle * math.pi) / 180.0
237
238         cosang = numpy.cos(ang)
239         sinang = numpy.sin(ang)
240
241         self._heading[0] = cosang * self._heading[0] + sinang *
    … self._left[0]
242         self._heading[1] = cosang * self._heading[1] + sinang *
    … self._left[1]
243         self._heading[2] = cosang * self._heading[2] + sinang *
    … self._left[2]
244
245         self._left[0] = cosang * self._left[0] - sinang * self._heading[0]
246         self._left[1] = cosang * self._left[1] - sinang * self._heading[1]
247         self._left[2] = cosang * self._left[2] - sinang * self._heading[2]
248
249     def pitch(self, angle):
250         """
251         pitch the Turtle about the left vector angle degrees
252         """
253
254         ang = angle * math.pi / 180.0
255         cosang = numpy.cos(ang)
256         sinang = numpy.sin(ang)
257
258         self._heading[0] = self._heading[0] * cosang - self._up[0] * sinang
259         self._heading[1] = self._heading[1] * cosang - self._up[1] * sinang
260         self._heading[2] = self._heading[2] * cosang - self._up[2] * sinang
261
262         self._up[0] = self._up[0] * cosang + self._heading[0] * sinang
263         self._up[1] = self._up[1] * cosang + self._heading[1] * sinang
264         self._up[2] = self._up[2] * cosang + self._heading[2] * sinang
265
266     def _setHeading(self, x, y=None, z=None):
267         """Set the Turtle's Heading.
268             x is either a list or tuple.
269         """
270         if y is None and z is None:
271             # Array, list, tuple...
272             if len(x) != 3:
273                 raise ValueError("Turtle3D: x is not a list/tuple/array of
    … 3 numbers")
```

```
274             self._heading = numpy.array(x, "d")
275         else:
276             # Three numbers
277             self._heading = numpy.array((x, y, z), "d")
278         self.heading = Vector(self.heading)
279         return
280 
281     def _setLeft(self, x, y=None, z=None):
282         """Set the Turtle's Left.
283             x is either a list or tuple.
284         """
285         if y is None and z is None:
286             # Array, list, tuple...
287             if len(x) != 3:
288                 raise ValueError("Turtle3D: x is not a list/tuple/array of
… 3 numbers")
289             self._left = numpy.array(x, "d")
290         else:
291             # Three numbers
292             self._left = numpy.array((x, y, z), "d")
293         self.left = Vector(self._left)
294         return
295 
296     def _setUp(self, x, y=None, z=None):
297         """Set the Turtle's Up.
298             x is either a list or tuple.
299         """
300         if y is None and z is None:
301             # Array, list, tuple...
302             if len(x) != 3:
303                 raise ValueError("Turtle3D: x is not a list/tuple/array of
… 3 numbers")
304             self._up = numpy.array(x, "d")
305         else:
306             # Three numbers
307             self._up = numpy.array((x, y, z), "d")
308         self.up = Vector(self.up)
309         return
310 
311     def unit(self, v):
312         norm = numpy.linalg.norm(v)
313         if norm == 0:
314             return v
315         return v / norm
316 
317     def orient(self, position: numpy.array, heading: numpy.array, left:
… numpy.array):
318         """
```

```
319         Orients the turtle with Position at p1, Heading at p2 and Left at
    p3
320         Arguments:
321             position
322         """
323
324         self._position = position
325
326         temp = heading - position
327         self._heading = self.unit(temp)
328         self.heading = Vector(self._heading)
329
330         temp = left - position
331         self._left = self.unit(temp)
332
333         temp = numpy.cross(self._heading, self._left)
334         self._up = self.unit(temp)
335         self.up = Vector(self._up)
336
337         # fix left to be orthogonal
338         temp = numpy.cross(self._up, self._heading)
339         self._left = self.unit(temp)
340         self.left = Vector(self._left)
341         return
342
343     def orient_at_residue(self, chain, resnumb, orientation):
344         '''
345         Orient the turtle at the specified residue from the input Chain in
346         either orientation 1 or 2.
347
348         Arguments:
349             turtle: input Turtle3D
350             chain: list of Residues in the model, eg: chain = model['A']
351             resnumb: residue number
352             orientation:
353             1 - at Ca heading towards Cb with N at the left
354             2 - at Ca heading towards C with N at the left
355         Returns: None. Turtle internal state is modified
356         '''
357
358         assert self._orientation == 1 or self._orientation == 2,
    f'orient_at_residue() requires Turtle3D to be #1 or #2'
359
360         residue = chain[resnumb]
361         assert residue is not None, f'get_backbone_from_sidechain()
    requires valid residue number'
362
363         # by this point I'm pretty confident I have coordinates
```

```
364         # we pull the actual numpy.array from the coordinates since that's
    … what the
365         # Turtle3D expects
366 
367         n = residue['N'].get_vector().get_array()
368         ca = residue['CA'].get_vector().get_array()
369         cb = residue['CB'].get_vector().get_array()
370         c = residue['C'].get_vector().get_array()
371 
372         if orientation == ORIENT_SIDECHAIN:
373             self.orient(ca, cb, n)
374             self.set_orientation(ORIENT_SIDECHAIN)
375         elif orientation == ORIENT_BACKBONE:
376             self.orient(ca, c, n)
377             self.set_orientation(ORIENT_BACKBONE)
378         return
379 
380     def orient_from_backbone(self, n: numpy.array, ca: numpy.array, cb:
    … numpy.array, c: numpy.array, orientation):
381         '''
382         Orient the turtle at the specified residue from the input Chain in
383         either orientation 1 or 2.
384 
385         Arguments:
386             turtle: input Turtle3D object
387             n: position of n atom
388             ca: position of ca atom
389             c: position of c atom
390             orientation:
391             1 - at Ca heading towards Cb with N at the left
392             2 - at Ca heading towards C with N at the left
393         Returns: None. Turtle internal state is modified
394         '''
395 
396         assert orientation == 1 or orientation == 2, f'orient_at_residue()
    … requires Turtle3D to be #1 or #2'
397 
398         _n = n.copy()
399         _ca = ca.copy()
400         _cb = cb.copy()
401         _c = c.copy()
402 
403         if orientation == ORIENT_SIDECHAIN:
404             self.orient(_ca, _cb, _n)
405             self.set_orientation(ORIENT_SIDECHAIN)
406         elif orientation == ORIENT_BACKBONE:
407             self.orient(_ca, _c, _n)
408             self.set_orientation(ORIENT_BACKBONE)
```

```
409         return
410
411     def to_local(self, global_vec) -> numpy.array:
412         """
413         Returns the Turtle-centered local coordinates for input Global
    … vector (3d)
414         """
415
416         newpos = global_vec - self._position
417         dp1 = numpy.dot(self._heading, newpos)
418         dp2 = numpy.dot(self._left, newpos)
419         dp3 = numpy.dot(self._up, newpos)
420
421         result = numpy.array((dp1, dp2, dp3), "d")
422         return result
423
424     def to_localVec(self, global_vec) -> Vector:
425         """
426         Returns the Turtle-centered local coordinates for input Global
    … vector (3d)
427         """
428
429         newpos = global_vec - self._position
430         dp1 = numpy.dot(self._heading, newpos)
431         dp2 = numpy.dot(self._left, newpos)
432         dp3 = numpy.dot(self._up, newpos)
433
434         return Vector(dp1, dp2, dp3)
435
436     def to_global(self, local) -> numpy.array:
437         """
438         Returns the global coordinates for input local vector (3d)
439         """
440
441         p1 = self._position[0] + self._heading[0] * local[0] +
    … self._left[0] * local[1] + self._up[0] * local[2]
442         p2 = self._position[1] + self._heading[1] * local[0] +
    … self._left[1] * local[1] + self._up[1] * local[2]
443         p3 = self._position[2] + self._heading[2] * local[0] +
    … self._left[2] * local[1] * self._up[2] * local[2]
444
445         return numpy.array((p1, p2, p3), "d")
446
447     def to_globalVec(self, local) -> Vector:
448         """
449         Returns the global coordinates for input local vector (3d)
450         """
451
```

```
452         p1 = self._position[0] + self._heading[0] * local[0] +
… self._left[0] * local[1] + self._up[0] * local[2]
453         p2 = self._position[1] + self._heading[1] * local[0] +
… self._left[1] * local[1] + self._up[1] * local[2]
454         p3 = self._position[2] + self._heading[2] * local[0] +
… self._left[2] * local[1] * self._up[2] * local[2]
455 
456         return Vector(p1, p2, p3)
457 
458     def __repr__(self):
459         """Return Turtle 3D coordinates."""
460         return f"<Turtle: {self._name}\n Position: {self._position},\n
… Heading: {self._heading} \n Left: {self._left} \n Up: {self._up}\n
… Orientation: {self._orientation}\n Pen: {self.Pen()} \n Recording:
… {self._recording}>"
461 
462     def bbone_to_schain(self):
463         '''
464         Function requires turtle to be in orientation #2 (at alpha carbon,
465         headed towards carbonyl, with nitrogen on left) and converts to
… orientation #1
466         (at alpha c, headed to beta carbon, with nitrogen on left.
467 
468         Arguments:
469             turtle: Turtle3D object in orientation #2
470 
471         Returns: modified Turtle3D
472         '''
473 
474         assert self._orientation == 2, f'bbone_to_schain() requires
… Turtle3D to be in orientation #2'
475 
476         self.roll(240.0)
477         self.pitch(180.0)
478         self.yaw(110.0)
479         self.roll(240.0)
480         self.set_orientation(1) # sets the orientation flag
481 
482 
483     def schain_to_bbone(self):
484         '''
485         Function requires turtle to be in orientation #1 (at alpha c,
… headed to beta carbon, with nitrogen on left)
486         and converts to orientation #2 (at alpha carbon, headed towards
… carbonyl, with nitrogen on left).
487 
488         Arguments:
489             None
```

```
490         Returns: modified Turtle3D
491         '''
492 
493         assert self._orientation == 1, f'schain_to_bbone() requires
    … Turtle3D to be in orientation #1'
494 
495         self.pitch(180.0)
496         self.roll(240.0)
497         self.yaw(110.0)
498         self.roll(120.0)
499         self.set_orientation(2) # sets the orientation flag
500         return
501 
502 # End of file
503 
```