*Script 4: The Vector Class created to simplify the Nelder-Mead, Hill Climbing, and Random Search Algorithms for the Rastrigin Function Optimization Problem.*

class Vector(object):

def \_\_init\_\_(self, \*data):

if type(data[0])==Vector:

self.data = list(data[0])

else:

self.data = list(data)

def \_\_repr\_\_(self):

return repr(tuple(self.data))

def print(self, n=0):

print("Vector(", end ="")

for j in range(len(self.data)-1):

print (round(self.data[j],n), end = ", ")

print (round(self.data[j+1],n), end = "")

print(") ")

return self

#--FUNCTIONS--------------------------------------------------------

def length(self):

return (len(self.data))

def scalars(self):

print('scalars =', self.data[:])

return self.data[:]

def vectorList(self):

return self.data

def equals(self, other):

self.data=other.vectorList()

return tuple(self.data)

def vecCost(x,y):

from math import cos

from math import pi

if (-5.12<=x<=5.12) and (-5.12<=y<=5.12):

return (20+ x\*\*2 - 10\*cos(2\*pi\*x) + y\*\*2 - 10\*cos(2\*pi\*y))

return float('inf')

def cost(self):

from math import cos

from math import pi

if (-5.12 <= self.data[0] <= 5.12) and (-5.12 <= self.data[1] <= 5.12):

x = self.data[0]

y = self.data[1]

return (20+ x\*\*2 - 10\*cos(2\*pi\*x) + y\*\*2 - 10\*cos(2\*pi\*y))

return float('inf')

def dist(self, other):

return (self-other).mag()

def dotProd(self, other):

return sum([self.data[j]\*other.data[j] for j in range (len(self.data))])

def crossProd(X,Y):

return Vector (X.data[1]\*Y.data[2] - X.data[2]\*Y.data[1], \

X.data[2]\*Y.data[0] - X.data[0]\*Y.data[2], \

X.data[0]\*Y.data[1] - X.data[1]\*Y.data[0])

def mag(self):

from math import sqrt

return sqrt(sum([j\*j for j in self.data]))

def normalize(self):

m=self.mag()

self.data= (self/m).data

return self

def swap (A,B):

T = Vectors(0)

T.equals(A)

A.equals(B)

B.equals(T)

def sort(vectorList):

#if type(vectorList) != list or len(vectorList) != 3 or type(vectorList[0]) != Vector or type(vectorList[1] != Vector or type(vectorList[2]) != Vector:

#exit('Error: The sort function limited to a list of three Vector elements.')

vectorList.sort(key = Vector.cost)

return vectorList

#--OPERATORS--------------------------------------------------------

def \_\_add\_\_(self, other):

return Vector(\*[self.data[j]+other.data[j] for j in range(len(self.data))])

def \_\_sub\_\_(self, other):

return -other + self

def \_\_mul\_\_(self, entity):

if isinstance(entity, (Vector)):

return self.crossProd(entity)

if isinstance(entity, (int, float)):

return Vector(\*[ j\*entity for j in (self.data) ] )

def \_\_rmul\_\_(self, num):

return self\*num

def \_\_truediv\_\_(self, num):

if num==0: return NotImplemented

return self \*(1.0/num)

def \_\_eq\_\_(self, other):

return (self.data == other.data)

def \_\_ne\_\_(self, other):

return not(self.data == other.data)

def \_\_neg\_\_(self):

return Vector(\*[-j for j in self.data])

def \_\_getitem\_\_(self, index):

return self.data[index]

def \_\_setitem\_\_(self, index, num):

self.data[index]=num

return self