```
from __future__ import division
import sys
import random
import math
import numpy as np
from models import *
from searchers import *
from options import *
from utilities import *
from sk import *
sys.dont_write_bytecode = True
#Dr.M
rand=  random.random # generate nums 0..1
any=   random.choice # pull any from list
sqrt= math.sqrt  #square root function

def display(modelName,searcher,runTimes,scores,historyhi=[],historylo=[]):
  assert(len(runTimes) == len(scores)),'Ouch! it hurts'
  print "================================================================"
  print "Model Name: %s"%modelName
  print "Searcher Name: %s"%searcher.__name__,
  print "Options Used: ",
  print myoptions[searcher.__name__]
  import time
  print ("Data: %s"%time.strftime("%d/%m/%Y"))
  print "Average running time: %f " %np.mean(runTimes)
  if(len(historyhi)!=0):
    for x in xrange(myModelobjf[modelName]):
      print "Objective No. %d: High: %f Low: %f"%(x+1,historyhi[x],historylo[x])
  #for i in range(0,len(runTimes)):
  #  print "RunNo: %s RunTime: %s Score: %s"%(i+1,runTimes[i],scores[i])
  #print scores
  print xtile(scores,width=25,show=" %1.6f")
  print "================================================================"



def multipleRun():
 r = 1
 for klass in [Schaffer]:#DTLZ7]:#Schaffer, Fonseca, Kursawe, ZDT1,ZDT3,Viennet]:
   #print "Model Name: %s"%klass.__name__
   for searcher in [SA,MaxWalkSat]:
     n = 0.0
     listTimeTaken = []
     listScores = []
     random.seed(6)
     historyhi=[-1e10 for count in xrange(myModelobjf[klass.__name__])]
     historylo=[1e10 for count in xrange(myModelobjf[klass.__name__])]
     for _ in range(r):
       test = searcher(klass(),"display2")

       import time
       t1 = time.time()
       solution,score,model = test.evaluate()
       for x in xrange(model.objf):
         historyhi[x]=max(model.past[x].historyhi,historyhi[x])
         historylo[x]=min(model.past[x].historylo,historylo[x])
         sys.stdout.flush()
       print
       timeTaken = (time.time() - t1) * 1000
       listTimeTaken.append(timeTaken)
       listScores.append(score)
     display(klass.__name__,searcher,listTimeTaken,listScores,historyhi,historylo)
def step2():
    rdivDemo([
      ["Romantic",385,214,371,627,579],
      ["Action",480,566,365,432,503],
      ["Fantasy",324,604,326,227,268],
      ["Mythology",377,288,560,368,320]])

```



```
def part6():
 r=5
 from collections import defaultdict

 for klass in [ZDT1]:#, Fonseca, Kursawe, ZDT1,ZDT3,Viennet]:
   #print "Model Name: %s"%klass.__name__
   for searcher in [SA,MaxWalkSat]:
     eraCollector=defaultdict(list)
     n = 0.0
     listTimeTaken = []
     listScores = []
     random.seed(6)
     historyhi=[-1e10 for count in xrange(myModelobjf[klass.__name__])]
     historylo=[1e10 for count in xrange(myModelobjf[klass.__name__])]

     for count in range(r):
       myoptions['MaxWalkSat']['probLocalSearch']=(count+1)*0.1
       myoptions['SA']['emax']=(count+1)*0.01
       test = searcher(klass(),"display2")
       import time
       t1 = time.time()
       solution,score,model = test.evaluate()
       lastera=[]
       for x in xrange(model.objf):
         temp = searcher.__name__+klass.__name__+str(count)+"f"+str(x+1)
         test=[temp]
         hisIndex=model.past[x].historyIndex
         #print x, hisIndex
         if(len(model.past[x].history[hisIndex-1])!=0):
           lastera.append(test+model.past[x].history[hisIndex-1])
         else:
           lastera.append(test+model.past[x].listing)
       #print lastera
       eraCollector[searcher.__name__+klass.__name__+str(count)]=lastera
       timeTaken = (time.time() - t1) * 1000
       listTimeTaken.append(timeTaken)
       listScores.append(score)
     #display(klass.__name__,searcher,listTimeTaken,listScores)
     #print eraCollector#.keys()
     callrdivdemo(eraCollector)

def callrdivdemo(eraCollector):
  #print "callrdivdemo %d"%len(eraCollector.keys())
  keylist = eraCollector.keys()
  objf = len(eraCollector[keylist[0]])
  variant = len(keylist)
  for x in xrange(objf):
    rdivarray=[]
    for y in xrange(variant):
      #print "Length of array: %f"%len(eraCollector[keylist[y]][x])
      rdivarray.append(eraCollector[keylist[y]][x])
    rdivDemo(rdivarray)

def testGA():
  for klass in [Viennet]:
    test = GA(klass(),"display2")
    test.evaluate()

if __name__ == '__main__':
 # random.seed(1)
 # nums = [random.random()**2 for _ in range(100)]
 # print xtile(nums,lo=0,hi=1.0,width=25,show=" %3.2f")
 # model = ZDT1()
 # model.testgx()
 # for klass in [ZDT1]:
 #   print klass.__name__
 #multipleRun()
 testGA()
 #part6()
 #step2()
"""
Model: Vinnet
Initial Population: 50
```

```
# of crossover in each generation: 20
Crossover probability: 1

  −− *   |  −−−−−−−− , 0.129654,  0.218249,  0.278099,  0.619733,  0.906036
   −−−−− *|    −−−−− , 0.142837,  0.256091,  0.284924,  0.483059,  0.603858
 −   *−−−|−−−−−−−−−  , 0.058453,  0.086322,  0.188811,  0.217728,  0.502102
   * −−−−|−−−−−      , 0.074805,  0.080234,  0.098817,  0.142874,  0.314088
   −−  * |    −−−−− , 0.073295,  0.112665,  0.164241,  0.335644,  0.434142
  − * −−−−|−−−−−−−−−− , 0.058322,  0.073295,  0.081249,  0.124711,  0.374678
−−−− *    |−−−−−     , 0.004657,  0.044779,  0.058284,  0.126250,  0.190066
 − *−−−−−−|−−−−−−−−−− , 0.015735,  0.026406,  0.044110,  0.044191,  0.188385
−*       |      −− , 0.021123,  0.030610,  0.042190,  0.494760,  0.553502
 −−−    * |  −     , 0.032184,  0.103856,  0.229123,  0.428753,  0.478736
 *       | −−−−−−−− , 0.021904,  0.028217,  0.060622,  0.446659,  0.716898
  *   −−−−|−−−−−−    , 0.016239,  0.020727,  0.050713,  0.121313,  0.269193
 −* −−−−−−|         , 0.016240,  0.020808,  0.021127,  0.046757,  0.106734
 *  −−−−−−−|−        , 0.016202,  0.017440,  0.023286,  0.067657,  0.167368
 −*  −−−  |        , 0.012011,  0.016239,  0.020783,  0.048955,  0.068997
 * −−−−−−−−|−−−−−−−−  , 0.014171,  0.016227,  0.019446,  0.038800,  0.235277
 −    *   |         , 0.016215,  0.043754,  0.121925,  0.395675,  0.399153
   *−−−−−−−|−−−−−−−    , 0.015997,  0.022856,  0.061865,  0.069621,  0.277133
  * −−−−−−−|−−−−−−−−  , 0.009435,  0.013996,  0.039269,  0.069554,  0.366202
 −  * −−−−−|−−−−−−−−−− , 0.012377,  0.023472,  0.060187,  0.080512,  0.243279


[−0.18446074481999997, −0.7044678423600002, 1.1999999999999993]
"""
```