```
from __future__ import division
import sys
import random
import math
import numpy as np
sys.dont_write_bytecode = True

def say(x):
  sys.stdout.write(str(x)); sys.stdout.flush()

class Fonseca:
  maxVal=−10000
  minVal=10000

  def returnMin(self,num):
    if(num<self.minVal):
      self.minVal=num
      return num
    else:
      return self.minVal

  def returnMax(self,num):
    if(num>self.maxVal):
      self.maxVal=num
      return num
    else:
      return self.maxVal

  def fx(self,listpoint,version):
    n=len(listpoint)
    rootn=(n**0.5)**−1
    sum=0
    for i in range(0,n):
      if version ≡ 1:
        sum+=(listpoint[i]−rootn)**2
      elif version ≡ 2:
        sum+=(listpoint[i]+rootn)**2
      else:
        print "STOP MESSING AROUND"
    return (1 − math.exp(−sum))

  def evaluate(self,listpoint):
    energy = self.fx(listpoint,1)+ self.fx(listpoint,2)
    return (energy−self.minVal)/(self.maxVal−self.minVal)

  def baseline(self,minR,maxR):
    for x in range(0,50000):
      solution = [(minR + random.random()*(maxR−minR)) for z in range(0,3)]
      self.returnMax(self.fx(solution,1)+ self.fx(solution,2))
      self.returnMin(self.fx(solution,1)+ self.fx(solution,2))

  def  neighbour(self,minN,maxN):
    return minN + (maxN−minN)*random.random()

  def info(self):
    return "Fonseca~"

class Kursawe:
  maxVal=−10000
  minVal=10000

  def returnMin(self,num):
    if(num<self.minVal):
      self.minVal=num
      return num
    else:
      return self.minVal

  def returnMax(self,num):
    if(num>self.maxVal):
      self.maxVal=num
      return num
    else:
```

```
      return self.maxVal

  def f1(self,listpoint):
    n=len(listpoint)
    #inspired by ’theisencr’
    return np.sum([−10*math.exp(−0.2*(np.sqrt(listpoint[i]**2 + listpoint[i+1]**
2))) for i in range (0, n−1)])
    return sum

  def f2(self,listpoint):
    a=0.8
    b=3
    n=len(listpoint)
    #inspired by ’theisencr’
    return np.sum([math.fabs(listpoint[i])**a + 5*np.sin(listpoint[i])**b for i
in range (0, n)])


  def evaluate(self,listpoint):
    energy = (self.f1(listpoint)+self.f2(listpoint))
    return (energy−self.minVal)/(self.maxVal−self.minVal)

  def baseline(self,minR,maxR):
    for x in range(0,90000):
      solution = [(minR + random.random()*(maxR−minR)) for z in range(0,3)]
      self.returnMax(self.f1(solution)+ self.f2(solution))
      self.returnMin(self.f2(solution)+ self.f2(solution))

  def  neighbour(self,minN,maxN):
    return minN + (maxN−minN)*random.random()

  def info(self):
    return "Kursawe~"
  def test(self):
    file = open("Kursawe.txt","w")
    for x in range(−5,6):
      for y in range(−5,6):
        for z in range(−5,6):
          solution = [x,y,z]
          file.write("%f\n"%self.evaluate(solution))
    file.close()

class MaxWalkSat():
  model = None
  minR=0
  maxR=0
  random.seed(40)
  def __init__(self,modelName):
    #print "init"
    if modelName ≡ "Fonseca":
      self.model = Fonseca()
      self.minR=−4
      self.maxR=4
      #print "here"
    elif modelName ≡ "Kursawe":
      self.model = Kursawe()
      self.minR=−5
      self.maxR=5
      self.model.test()
      #print "there"
    else:
      print "STOP MESSING AROUND"


  def evaluate(self):
    model = self.model
    print "Model used: %s"%model.info()
    minR=self.minR
    maxR=self.maxR
    maxTries=50
    maxChanges=2000
    n=3
    threshold=0.05
    probLocalSearch=0.75
    bestScore=100
    bestSolution=[]


    print "Value of p: %f"%probLocalSearch
   # model = Fonseca()
    model.baseline(minR,maxR)
    print model.maxVal,model.minVal

    for i in range(0,maxTries): #Outer Loop
      solution=[]
      for x in range(0,n):
        solution.append(minR + random.random()*(maxR−minR))
      #print "Solution: ",
      #print solution
      for j in range(0,maxChanges):      #Inner Loop
         score = model.evaluate(solution)
         #print score
         # optional−start
         if(score < bestScore):
           bestScore=score
           bestSolution=solution

         # optional−end
         if(score < threshold):
           print "threshold reached|Tries: %d|Changes: %d"%(i,j)
           return solution,score

         if random.random() > probLocalSearch:
             c = int(0 + (2−0)*random.random())
             solution[c]=model.neighbour(minR,maxR)
         else:
             tempBestScore=score
             tempBestSolution=solution
             interval = (maxR−minR)/10
             c = int(0 + (2−0)*random.random())
             for itr in range(0,10):
                solution[c] = minR + (itr*interval)*random.random()
                tempScore = model.evaluate(solution)
                if tempBestScore > tempScore:     # score is correlated to max?
                  tempBestScore=tempScore
                  tempBestSolution=solution
             solution=tempBestSolution

    return bestSolution,bestScore

def probFunction(old,new,t):
   return math.exp(1 *(old−new)/t)

class SA():
  model = None
  minR=0
  maxR=0
  random.seed(1)
  def __init__(self,modelName):
    #print "init"
    if modelName ≡ "Fonseca":
      self.model = Fonseca()
      self.minR=−4
      self.maxR=4
      #print "here"
    elif modelName ≡ "Kursawe":
      self.model = Kursawe()
      self.minR=−5
      self.maxR=5
      self.model.test()
      #print "there"
    else:
      print "STOP MESSING AROUND"

  def neighbour(self,solution,minR,maxR):
    returnValue = []
```

```
    n=len(solution)
    for i in range(0,n):
      tempRand = random.random()
      if tempRand <0.33:
        returnValue.append(minR + (maxR − minR)*random.random())
      else:
        returnValue.append(solution[i])
    return returnValue

  def evaluate(self):
    model=self.model
    print "Model used: %s"%(model.info())
    minR = self.minR
    maxR = self.maxR
    model.baseline(minR,maxR)
    print model.maxVal, model.minVal

    s = [minR + (maxR − minR)*random.random() for z in range(0,3)]
    print s
    e = model.evaluate(s)
    emax = 0
    sb = s                         #Initial Best Solution
    eb = e                         #Initial Best Energy
    k = 1
    kmax = 1000
    count=0
    while(k ≤ kmax ∧ e > emax):
      sn = self.neighbour(s,minR,maxR)
      en = model.evaluate(sn)
      if(en < eb):
        sb = sn
        eb = en
        say("!") #we get to somewhere better globally
      tempProb = probFunction(e,en,k/kmax)
      tempRand = random.random()
#      print " tempProb: %f tempRand: %f " %(tempProb,tempRand)
      if(en < e):
        s = sn
        e = en
        say("+") #we get to somewhere better locally
      elif(tempProb ≤ tempRand):
        jump = True
        s = sn
        e = en
        say("?") #we are jumping to something sub−optimal;
        count+=1
      say(".")
      k += 1
      if(k % 50 ≡ 0):
         print "\n"
       #  print "%f{%d}"%(sb,count),
         count=0
    print
    print sb,eb

def doSomethingCool():
   """
 test = MaxWalkSat("Kursawe")
 solution,score = test.evaluate()
 print "Solution: ",
 print solution
 print "Score: ",
 print score
 """
   test = MaxWalkSat("Fonseca")
   solution,score = test.evaluate()
   print "Solution: ",
   print solution
   print "Score: ",
   print score

def step1():
   test = SA("Fonseca")
```

```
    test.evaluate()

    test = SA("Kursawe")
    test.evaluate()

def step2():
    random.seed(24)
    test = MaxWalkSat("Fonseca")
    solution,score = test.evaluate()
    print "Solution: ",
    print solution
    print "Score: ",
    print score

if __name__ ≡ ’__main__’:
  step2();


"""
==========================================================================================
Step1():
Model used: Fonseca~
2.0 0.98516179182
[1.9378527346874188, 2.8846386551360244, 0.38378314132997104]
.!+.?....!+..!+......!+..........?......!+..!+...?.....+.+.?.+...+.+.

!+.?....+.?..+...?...+..+.?.......+..+......+..?..+..?.+.+......+....

?.+.+.+..+..+.....+.?..+..+.....?..+.?.......................

+..........+.?.+.+...+.....?.+.......................+.+.

..........+......?.....+.....+.!+.?...+..........+.?.+.+.+...

+....+.?..+..+.+.+........+........+....+.....+.?.?.......+..+.?..

............?...+..+..+..........+.?....?....+...+..+.....

..+.+..?..?..+.+.?.........?..+........+...+..?...+.?..........+.

......+.?.....+...?.....+........+....................?.

.........+.....?..+...+.....+.+...?........+.+..?..?.+.+.....+.

..+.?............+.+.?.+.+.....+.+........+.....?.........+.+.

+..+.?.......+....+.+..+.........+...?.+...+.+.+....+...+....?..+.!+.

..?.+.+.........+..?.+...+..+..?...+..+......+...?...+.....+...+..

..?...+..........+........?...+.+............?.+........+.

..+..+.?......+.+.+...................?...+...+........+...

.............+...+.....?.+..........+.?..+.+..+..+..+......

.......+....+.....................!+.?.?.+.+...+.....+.?....

+....+....+.?...+...?...+.+..?..+.........+..+.....+.+.?.......+..

.........+....+....+......?...?......+.........+..+....?...

..+..+....+..+.....?..+.+.?...+.....+......+........?.+..+.+...+..

.
[−0.5054875007120163, −0.6684995828260938, −0.7404609583917736] 0.0419534888819
Model used: Kursawe~
20.1443653143 −21.2206622108
[3.3896720622951317, −1.5941655061311302, −2.494047352967057]
.?.+..!+.?.?...+.?.+.+.?...+.?.+..?.+..+.+.?.+.?.?.+.+.?.?..+.?.+.?.+.?.?.+...+..+..?.

+..+.?.+.+...+.?.?..?..+..+.?..+.+.+.+.!+.?.?..?..+.+.+....+.?.+.?..?...?.+.+.?..+..
```

```
.+....?.+.?.?.+...+...+....?..+...?..+...+...+..+.+...?.+.....+.?...+.+.

?..+.+...+.....?.+.?.?...+.+.+.+...+...+.?..+.?..?...+..+.?...+.+..+.+.+.?.?.+.

+..+.......+....+.+.....!+...?.+.+...?..+.+.+.......?.+.+....?.+..+...

.?......+.?.+.+....+..?.+...+..?....+....?..+....?...+.+.+..+.+...+...

..?.?..+.?.+...?.+.....?.+.....+..?.+...+.+...?...+.+...?.+.....?..?..+.

.+..+......+.+.?..+....?.?.+.+.+.+.....+..+...?..+.?.+.+.+....+.+.+....+.?.

.....?.+..+.....?.+..?....+......+.?...+.?.?...+....+.+..+.+..?.....

.+........+.......!+.....!+.....?..?....+.?..+.+.+....+.?...?.+..+..

.......?..+....+.?.+.+.+........?.+.+...+....?...+.+.+..........

......+...+..?...?....+.....+.........?...+.........+....+.

...+.+....?.+..+....+....+.+.+....+...+..?..+...+.?...+.+......?.+.?..

...+.....+..+.......+.........?.+.....+......?....?....+....

.......?..+...?.+.+......+...?.+...+..+.+.+........+..........

.......+...?.+...?.+....?...?..+.+...?..+..+.....?.+.........+...

?......+....?.+.+...+.+.+...............+..+....?...+.+....+..

....?............?..+.+.+...+........?.+..+....+..+.........

?..+.....?.+....+....+..+.+.?.....+........?.....?.+..+.+..+..+....

.+......+............!+.
========================================================================
step3()

Model used: Fonseca~
Value of p: 0.250000
2.0 0.982620191082
threshold reached|Tries: 0|Changes: 248
Solution:  [0.5201622473716911, 0.5223643784465688, 0.7792144027578889]
Score:  0.0482893972315
========================================================================
step4()

Model used: Fonseca~
Value of p: 0.250000
2.0 0.982620191082
threshold reached|Tries: 0|Changes: 248
Solution:  [0.5201622473716911, 0.5223643784465688, 0.7792144027578889]
Score:  0.0482893972315
------------------------------------------------------------------------
Model used: Fonseca~
Value of p: 0.500000
2.0 0.982620191082
threshold reached|Tries: 0|Changes: 437
Solution:  [0.6106730828042144, 0.5371817582767893, 0.7792144027578889]
Score:  0.0479112210915
------------------------------------------------------------------------
Model used: Fonseca~
Value of p: 0.750000
2.0 0.982620191082
threshold reached|Tries: 5|Changes: 51
Solution:  [−0.5600576523890011, −0.3713369731342202, −0.6001432899563435]
Score:  0.0313031798438

The results suggests that the algorithm finds the maximum score faster when value of p is 0.25. Though the efficiency o
f the algorithm seems correlated to value of p (higher the value of p, slower the search is), I am not sure if this is the cas
e with all the models or value ranges.
"""
```