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

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

def probFunction(old,new,t):
#   print "probFunction : old : %f new : %f t: %f return : %f exp: %f" %(old,new
, t,-1*(old-new)/t,math.exp(1 *(old-new)/t))
   return math.exp(1 *(old-new)/t)

def neighbour(s):
  if(s==9999):
    return s-1
  elif(s==-9999):
    return s+1
  else:
    if(random.randint(0,1) == 1):
      return s+1
    else:
      return s-1

class Model:
  def schaffer(self,independentVariable):
    global minVal,maxVal
    f1 = independentVariable**2
    f2 = (independentVariable -2)**2
    return (f1+f2)

class BaseLine:
  def __init__(self):
    self.minVal=10000000
    self.maxVal=0

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

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

  def findBaseLine(self):
    model = Model()
    for index in range(0,1000):
      inputRand = random.randint(-10000,10000)
      temp = model.schaffer(inputRand)
      self.minVal=self.returnMin(temp)
      self.maxVal=self.returnMax(temp)
    print("Max: %d Min: %d"%(self.maxVal,self.minVal))

class FindEnergy:
  emax=0

  def __init__(self,minimum,maximum):
    self.minimum = minimum
    self.maximum = maximum
    self.maxVal=0

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

  def evaluate(self,num):
    model = Model()
    temp = model.schaffer(num)
```



```
    energy = (temp -self.minimum)/(self.maximum-self.minimum)
    #print "Energy: %f Temp: %f Self.Max: %f Self.Min: %f Num: %f" %(energy,temp
,self.minimum,self.maximum,num)
    return energy

  def evaluateEmax(self):
    model = Model()
    for index in range(0,1000):
      inputRand = random.randint(-100,100)
      temp = model.schaffer(inputRand)
      energy = (temp - self.minimum)/(self.maximum-self.minimum)
      self.returnMax(energy)
    return self.maxVal

def doSomethingCool():
  base = BaseLine()
  base.findBaseLine()
  energy = FindEnergy(base.minVal,base.maxVal)
  emax = energy.evaluate(10000)
  print emax
#  print emax.evaluate()

#class SimulatedAnnealing:
def evaluate():
    jump = True
    base = BaseLine()
    base.findBaseLine()
    energy = FindEnergy(base.minVal,base.maxVal)
    emax = 0
    print "Base Line Values: Minimum: %f Maximum: %f Emax: %f" %(base.minVal,base.maxVal,e
max)

    s = random.randint(-10000,10000) #Initial State
    e = energy.evaluate(s)          #Initial Enenery
    sb = s                          #Initial Best Solution
    eb = e                          #Initial Best Energy
    k = 1
    kmax = 1000
    count=0
    while(k ≤ kmax ∧ e > emax):
      if(jump==False):
        sn = neighbour(s)
      else:
        sn = random.randint(-10000,10000)
        #jump= False #change
      en = energy.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
    return sb


if __name__ == '__main__':
```

```
    #  doSomethingCool();
145     evaluate()
    #
```