```
"""
A models file that can be imported to run optimizers
"""
from __future__ import division
import sys
import math, random, numpy as np, scipy as sp
sys.dont_write_bytecode = False
# Define some aliases.
rand=random.uniform
randi=random.randint
exp=math.e
sin=math.sin
sqrt=math.sqrt
pi=math.pi

class modelBasics(object):
  def __init__(i,model):
    i.model=model()
    i.name=model.__name__
  def do_a_randJump(i, e, en, t, k):
    p=exp**(-(e-en)/(t**k))<rand(0,1)
    return p
  def simpleneighbour(self,x,xmax,xmin):
      return xmin+(xmax-xmin)*rand(0,1)
  def neighbour(i,x,xmax,xmin):
    def __new(x,z):
        return xmin+(xmax-xmin)*rand(0,1) if rand(0,1)<1/(i.model.indepSize) els
e x[z]
    x_new=[__new(x,z) for z in xrange(i.model.indepSize)]
    return x_new
  def energy(i,x,emax,emin):
    ener=i.model.score(x);
    e_norm= (ener-emin)/(emax-emin)
    return e_norm
  def baselining(i,model):
    emax=0;emin=1;
    indepSize=i.model.indepSize;
    for _ in xrange(1000):
      x_tmp=[rand(i.model.baselo,i.model.basehi) for _ in xrange(indepSize)]
      ener=i.model.score(x_tmp);
      if ener>emax:
        emax=ener
      elif ener<emin:
        emin=ener
    return emax,emin
  f=open('log_sa_schaffer.txt','w')
  def say(i,x):
    sys.stdout.write(str(x));
    sys.stdout.flush()

class Schaffer(object):

  def __init__(i,hi=100,lo=-100, basehi=1000, baselo=-1000, kooling=0.7, indepSi
ze=1, iterations=2000):
    i.hi, i.lo, i.basehi, i.baselo, i.kooling, i.indepSize, i.iterations= hi, lo
, basehi, baselo, kooling, indepSize, iterations
    random.seed()
  def f1(i,x):
    return x*x
  def f2(i,x):
    return (x-2)**2
  def score(i,x):
    return i.f1(x[0])+i.f2(x[0])
  def eigenschaften(i):
    return i.hi, i.lo, i.kooling, i.indepSize, i.iterations

class Kursawe(object):
  def __init__(i,hi=5,lo=-5,kooling=0.6, a=0.8, b=3, indepSize=3, basehi=1000, b
aselo=-1000, iterations=2000):
    i.hi, i.lo, i.basehi, i.baselo, i.kooling, i.a, i.b, i.indepSize, i.iteratio
ns= hi, lo, basehi, baselo, kooling, a, b, indepSize, iterations
    random.seed()
  def f1(i,x):
```



```
    return np.sum([-10*exp**(-0.2*sqrt(x[z]**2+x[z+1]**2)) for z in xrange(i.ind
epSize-1)])
  def f2(i,x):
    return np.sum([abs(x[z])**i.a+5*sin(x[z]**i.b) for z in xrange(i.indepSize)]
)
  def score(i,x):
    return i.f1(x)+i.f2(x)
  def eigenschaften(i):
    return i.hi, i.lo, i.kooling, i.indepSize, i.iterations

class Fonseca(object):
  def __init__(i,hi=4,lo=-4, basehi=4, baselo=-4, kooling=1.99, indepSize=3, ite
rations=2000):
    i.hi, i.lo, i.basehi, i.baselo, i.kooling, i.indepSize, i.iterations= hi, lo
, basehi, baselo, kooling, indepSize, iterations
    random.seed()
  def f1(i,x):
    return (1-exp**np.sum([(x[z]-1/(np.sqrt(z+1))) for z in xrange(i.indepSize)]
))
  def f2(i,x):
    return (1-exp**np.sum([(x[z]+1/(np.sqrt(z+1))) for z in xrange(i.indepSize)]
))
  def score(i,x):
    return i.f1(x)-i.f2(x)
  def eigenschaften(i):
    return i.hi, i.lo, i.kooling, i.indepSize, i.iterations

class ZDT1(object):
  def __init__(i,hi=1,lo=0, basehi=1, baselo=0, kooling=7e-3, indepSize=30, iter
ations=2000):
    i.hi, i.lo, i.basehi, i.baselo, i.kooling, i.indepSize, i.iterations= hi, lo
, basehi, baselo, kooling, indepSize, iterations
    random.seed()
  def f1(i,x):
    return x[0]
  def g(i,x):
    return (1+9*(np.sum(x[1:]))/(i.indepSize-1))
  def f2(i,x):
    return i.g(x)*(1-sqrt(x[0]/i.g(x)))
  def score(i,x):
    return i.f1(x)+i.f2(x)
  def eigenschaften(i): # German for features
    return i.hi, i.lo, i.kooling, i.indepSize, i.iterations

class ZDT3(object):
  def __init__(i,hi=1,lo=0, basehi=1, baselo=0, kooling=7e-3, indepSize=30, iter
ations=2000):
    i.hi, i.lo, i.basehi, i.baselo, i.kooling, i.indepSize, i.iterations= hi, lo
, basehi, baselo, kooling, indepSize, iterations
    random.seed()
  def f1(i,x):
    return x[0]
  def g(i,x):
    return (1+9*(np.sum(x[1:]))/(i.indepSize-1))
  def f2(i,x):
    return i.g(x)*(1-sqrt(x[0]/i.g(x))-x[0]/i.g(x[0]/i.g(x))*sin(10*pi*x[0]))
  def score(i,x):
    return i.f1(x)+i.f2(x)
  def eigenschaften(i): # German for features
    return i.hi, i.lo, i.kooling, i.indepSize, i.iterations


class Viennet3(object):
  def __init__(i,hi=1,lo=0, basehi=1, baselo=0, kooling=7e-3, indepSize=2, itera
tions=2000):
    i.hi, i.lo, i.basehi, i.baselo, i.kooling, i.indepSize, i.iterations= hi, lo
, basehi, baselo, kooling, indepSize, iterations
    random.seed()
  def f1(i,x):
    return 0.5*x[0]**2+x[1]**2+sin(x[0]**2+x[1]**2)
  def f2(i,x):
    return (3*x[0]-2*x[1]+4)**2/8+(x[0]-x[1]+1)**2/175+15
  def f3(i,x):
```

```
130       return 1/(x[0]+x[1]+1)-1.1*exp**(-x[0]**2-x[1]**2)
      def score(i,x):
        return i.f1(x)+i.f2(x)+i.f3(x)
      def eigenschaften(i): # German for features
        return i.hi, i.lo, i.kooling, i.indepSize, i.iterations
```

```
"""
rDiv Demo
"""

import sk
import random

rand=random.uniform
randi=random.randint

rdivDemo=sk.rdivDemo

Range1=[rand(0,4) for _ in xrange(5)]; Range1.insert(0,'Range1')
Range2=[rand(3,13) for _ in xrange(5)]; Range2.insert(0,'Range2')
Range3=[rand(7,17) for _ in xrange(5)]; Range3.insert(0,'Range3')
Range4=[rand(10,27) for _ in xrange(5)]; Range4.insert(0,'Range4')
Range5=[rand(12,30) for _ in xrange(5)]; Range5.insert(0,'Range5')
Range6=[rand(30,50) for _ in xrange(5)]; Range6.insert(0,'Range6')


def _rDiv():
  rdivDemo([
            Range1,
            Range2,
            Range3,
            Range4,
            Range5,
            Range6,
            ])
_rDiv()
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 15 03:04:43 2014

@author: rkrsn
"""
from __future__ import division
import sys
import math, random, numpy as np, scipy as sp
sys.dont_write_bytecode = False
from models import *
from anzeigen import *
from dynamikliste import *
# from sk import Num
import analyzer

# Define some aliases.
rand = random.uniform
randi = random.randint
exp = math.exp

class SimulatedAnnealer(object):
  def __init__(self, modelName, disp=False, early=False):
    self.modelName = modelName
    self.disp = disp
    self.early = early
  def runSearcher(self):
    modelbasics = modelBasics(self.modelName);
    modelFunction = self.modelName()
    anz = anzeigen();
    hi, lo, kooling, indepSize, iterations = modelFunction.eigenschaften()
    emax, emin = modelbasics.baselining(self.modelName)
    sb = s = [randi(lo, hi) for z in xrange(indepSize)];
    eb = e = modelbasics.energy(s, emax, emin)
    enRec = dynamikliste()  # Creates a growing list.
    enRec[0] = 0;  # Since iterations start from 1, lets initialize enRec[0] to
0
    analyser = analyzer.analyser()
    epochs = 5 if self.early else iterations;
    k = 1;
    while epochs ∧ k < iterations:
      sn = modelbasics.neighbour(s, hi, lo)
      en = modelbasics.energy(sn, emax, emin)
      t = k / iterations
      if en < eb:
        eb, sb, enRec[k] = en, sn, en;
        if self.disp:
          modelbasics.say('!')

      if en < e:
        s, e, enRec[k] = sn, en, eb;
        if self.disp:
          modelbasics.say('+')

      elif modelbasics.do_a_randJump(en, e, t, kooling):  # The cooling factor n
eeds to be reallylow for some reason!!
        s, e, enRec[k] = sn, en, eb;
        if self.disp:
          modelbasics.say('?')
      else:
        enRec[k] = eb
      if self.disp:
        modelbasics.say('.')
      if k % 50 ≡ 0 ∧ k > 50:
        # print enRec[:-10]
        proceed = analyser.isItGettinBetter(enRec[k - 100:])
        if proceed:
          epochs += 1;
        else:
          epochs -= 1;
        # print enRec[k-40:] #
      k = k + 1
      if k % 40 ≡ 0:
```



```
        if self.disp:
          modelbasics.say('\n')  # sa.say(format(sb,'0.2f'))

    if self.disp:
      modelbasics.say('\n'),  # modelbasics.say('Best Value Found '), modelbasic
s.say(sb)

  # Print Energy and best value.
    for i in xrange(k):
      if self.disp:
        if i % 50 ≡ 0:
          print anz.xtile(enRec[i - 50:])
    if self.disp:
      modelbasics.say('\n')
    return eb

class MaxWalkSat(object):
  def __init__(self, modelName, disp=False, early=True, maxTries=100, maxChanges
=100):
    self.modelName = modelName
    self.disp = disp
    self.maxTries = maxTries
    self.maxChanges = maxChanges
  def runSearcher(self):
    modelbasics = modelBasics(self.modelName);
    modelFunction = self.modelName()
    hi, lo, kooling, indepSize, iterations = modelFunction.eigenschaften()
    emax, emin = modelbasics.baselining(self.modelName)
    for i in xrange(self.maxTries):
        # Lets create a random assignment, I'll use list comprehesions here.
        x = xn = xb = [rand(lo, hi) for z in xrange(indepSize)]
        # Create a threshold for energy, let's say thresh=0.1% of emax (which is
 1) for starters
        thresh = 1e-7
        for j in xrange(self.maxChanges):
            # Let's check if energy has gone below the threshold.
            # If so, look no further.
            if modelbasics.energy(xn, emax, emin) < thresh:
                if self.disp:
                  modelbasics.say('.')
                break
            else:
                randIndx = randi(0, indepSize - 1)  # Choose a random part of so
lution x
                if rand(0, 1) > 1 / (indepSize + 1):  # Probablity p=0.33
                    y = xn[randIndx]
                    xn[randIndx] = modelbasics.simpleneighbour(y, hi, lo)
                    if self.disp:
                      modelbasics.say('+')
                    # print 'Random change on', randIndx
                else:
                    # xTmp is a temporary variable
                    xBest = emax;
                    # Step from xmin to xmax, take 10 steps
                    Step = np.linspace(lo, hi, 100)
                    if self.disp:
                      modelbasics.say('!')
                    for i in xrange(np.size(Step)):
                        xNew = xn; xNew[randIndx] = Step[i];
                        if modelbasics.energy(xNew, emax, emin) < xBest:
                            xBest = modelbasics.energy(xNew, emax, emin)
                            xn = xNew

        if modelbasics.energy(xn, emax, emin) < modelbasics.energy(xb, emax, emi
n):
            xb = xn
    return modelbasics.energy(xb, hi, lo)

if __name__ ≡ 'main':
  SimulatedAnnealer(Schaffer)
```

```
from __future__ import division
from a12 import a12

class analyser:
  def __init__(self,less=True):
    self.old=[];
    self.new=[];
    self.less=less;
  def addtolog(self, old, new):
    O = self.old
    O.append(old)
    N = self.new
    N.append(new)
    return O, N
  def bettered(self, new, old):
    roundoff=lambda n: round(n,2)
    def quartiles(lst):
      def p(x) : return int(100*roundoff(lst[x]))
      n  = int(len(lst)*0.25)
      return p(n) , p(2*n) , p(3*n)
    def betterifless():
      p1, median1, p3= quartiles(new)
      IQR1=p3-p1
      p1, median2, p3= quartiles(old)
      IQR2=p3-p1
      return median1<median2, IQR1<IQR2
    def betterifmore():
      p1, median1, p3= quartiles(new)
      IQR1=p3-p1
      p1, median2, p3= quartiles(old)
      IQR2=p3-p1
      return median1>median2, IQR1>IQR2
    def same(): return a12(new, old)≤0.56
    if self.less:
      betterMedian, betterIQR = betterifless()
      return betterMedian, betterIQR, same()
    else:
      betterMedian, betterIQR = betterifmore()
      return betterMedian, betterIQR, same()
  def isItGettinBetter(self, lst):
    self.old, self.new = self.addtolog(lst[0:49], lst[50:])
    out = False
    for old, new in zip(self.old, self.new):
      betterMedian, betterIQR, same = self.bettered(new, old)
      if ((same ∧ ¬ betterIQR) ∨ (¬ same ∧ ¬ betterMedian)): out=False
      elif(¬ same ∧ betterMedian): out=out ∨ False
    return out
```

```
from __future__ import division
from searcher import *
from models import *
import sys
from decimal import *
import numpy as np
from anzeigen import *
from time import gmtime, strftime
import sys, random, math, datetime, time,re
sys.dont_write_bytecode = True


for x in [Schaffer,
          Kursawe, Fonseca, ZDT1, Viennet3]:
  early=True
  eb=30*[0]
  for y in [MaxWalkSat]:#[SimulatedAnnealer, MaxWalkSat]:
    print 'Model: ', x.__name__
    print 'Searcher: ', y.__name__
    print strftime("%a, %d %b %Y %H:%M:%S ", gmtime()), '\n'
    k=x()
    reps=1
    dspl=anzeigen();
    hi, lo, kooling, indepSize, iterations = k.eigenschaften()
    print 'Einstellungen:'
    print 'min=', lo, ', max=', hi, ', Cooling Factor=', kooling, '\n'
    if early: print 'Early Termination!'  , '\n'
    for r in xrange(reps):
      a=y(x,disp=False,early=early)
      eb[r] =  a.runSearcher()

    #print dspl.xtile(eb[1:])
    """for r in xrange(4):
 print dspl.xtile(eb[r:r+50], lo=lo, hi=hi)"""
    print 'Energy: ', "{:.3E}".format(Decimal(str(np.sum(eb)/reps)))

  #

  sys.stdout.write('\n')
```