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

sqrt=math.sqrt

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 ZDT1():
  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 __init__(self,minR=0,maxR=1,n=30):
    self.minR=minR
    self.maxR=maxR
    self.n=n

  def f1(self,lst):
    assert(len(lst)==self.n),"Something's Messed up"
    return lst[0]

  def gx(self,lst):
    n=self.n
    assert(len(lst) == n),"Something's Messed up"
    return (1+ 9*np.sum([lst[i] for i in range(1,n)])/(n-1))
```

```
145   def f2(self,lst):
        n=self.n
        assert(len(lst)≡n),"Something's Messed up"
        gx=self.gx(lst)
        assert(gx≠0),"Ouch! it hurts"
150     return gx * (1- sqrt(lst[0]/gx))

      def evaluate(self,lst):
        return self.f1(lst)+self.f2(lst)/(self.maxVal-self.minVal)

155   def baseline(self,minR=0,maxR=1):
        for x in range(0,90000):
          solution = [(minR + random.random()*(maxR-minR)) for z in range(0,30)]
          self.returnMax(self.f1(solution)+ self.f2(solution))
          self.returnMin(self.f2(solution)+ self.f2(solution))
160
      def info(self):
        return "ZDT1~"

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

      def testgx(self):
        lst = [i for i in range(0,30)]
        print len(lst)
170     sum=0
        for i in range(1,len(lst)):
          sum+=lst[i]
        temp=(1+(9*sum/29))
        assert(self.gx(lst)≡temp),"testgx failed"
175
    class Schaffer:

      def __init__(self,minR=-1e4,maxR=1e4):
        self.minR=minR
180     self.maxR=maxR
        self.minVal=10000000
        self.maxVal=-1e6

      def evaluate(self,listpoint):
185     assert(len(listpoint) ≡ 1),"Something's Messed up"
        var=listpoint[0]
        rawEnergy = (var**2 +(var-2)**2)
        energy = (rawEnergy -self.minVal)/(self.maxVal-self.minVal)
        return energy
190
      def returnMin(self,num):
        if(num<self.minVal):
          return num
        else:
195       return self.minVal

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

      def info(self):
        return "Schaffer~"
205
      def baseline(self,minR,maxR):
        low = self.minR
        high = self.maxR
        for index in range(0,1000000):
210       inputRand =(low + (high-low)*random.random())
          #print "inputRand: %s"%inputRand
          temp = (inputRand**2 +(inputRand-2)**2)
          self.minVal=self.returnMin(temp)
          self.maxVal=self.returnMax(temp)
215     print("Max: %d Min: %d"%(self.maxVal,self.minVal))

      def  neighbour(self,minN,maxN):
```



```
        return minN + (maxN-minN)*random.random()
```