Symbolic calculation

For simplicity, the following program will use one-dimensionsl functions centered at $x=0$. The generalization to higher dimensions or to multiple centers will be your home work.

You will have to import the `math` python module to be able to use the exponential function:

```
from math import *
```

Create the `Gaussian` class

The first step will be to create an object to describe a Gaussian class, which will describe the Gaussian function:

$$G(x)=\exp(-\alpha x^2)$$

For what follows, the `Gaussian` class will inherit from the `Function` class, which will not be defined yet.

The αparameter will be set at the creation of the object, and the object will be *callable*. This means that the operator *()* will be defined via the `__call__` method, such that the objects will be used as functions.

Here is the interface of the class:

```
class Function(object):
    pass

class Gaussian(Function):

   def __init__(self,exponent):
       # TODO

   def __call__(self,x):
       # TODO

   def __str__(self):
       # TODO
```

The `__str__` method corresponds to the conversion of the object to a string, which will be used to print the symbolic representation of the function. And here is the test program:

```
f1 = Gaussian(1.)
print "F1 = ", f1
for i in range(201):
  x = (i*0.02)-2.
  print x, f1(x)
```

and the expected output:

```
F1 =  exp(-1.000000 x^2)
-2.0 0.0183156388887
-1.98 0.0198331598935
-1.96 0.0214592390801
-1.94 0.0232000696073
  [...]
1.92 0.0250620632626
1.94 0.0232000696073
```

```
1.96 0.0214592390801
1.98 0.0198331598935
2.0 0.0183156388887
```

Multiplying two `Gaussians`

Now, we will implement the multiplication of two `Gaussians`:

$$e^{-\alpha_1 x^2} \times e^{-\alpha_2 x^2} = e^{-(\alpha_1+\alpha_2)x^2}$$

You see that when you multiply a `Gaussian` by another `Gaussian`, the result is a new `Gaussian`. The multiplication of a `Gaussian` object by another `Gaussian` object should return a new `Gaussian` object. Do do that, we will *overload* the `*` operator in the `Gaussian` class:

```
def __mul__(self,other):
   if isinstance(other,Gaussian):
      #TODO
   else:
      raise TypeError
```

The main function is:

```
def main():
  f1 = Gaussian(2.)
  f2 = Gaussian(3.)
  f3 = f1*f2
  print "F1 = ", f1
  print "F2 = ", f2
  print "F3 = ", f3
  for i in range(201):
    x = (i*0.01)-1.
    print x, f1(x)*f2(x), f3(x)
```

and the output sould be:

```
F1 =  exp(-2.000000 x^2)
F2 =  exp(-3.000000 x^2)
F3 =  exp(-5.000000 x^2)
-1.0 0.00673794699909 0.00673794699909
-0.99 0.00744286071006 0.00744286071006
-0.98 0.00821330400345 0.00821330400345
-0.97 0.00905444030644 0.00905444030644
  [...]
0.97 0.00905444030644 0.00905444030644
0.98 0.00821330400345 0.00821330400345
0.99 0.00744286071006 0.00744286071006
1.0 0.00673794699909 0.00673794699909
```

Linear combination

A linear combination can be implemented as a list of pairs. The first element of each pair is a `float`, and the second argument is a `Function`. The `LinearCombination` class will inherit from the `Function` class, and is also a callable class. The `__call__` function should return the value of the linear combination evaluated at x.

The main function is:

```
def main():

  class test_function(Function):

    def __call__(self,x):
     return x

    def __str__(self):
       return "x"

  f = test_function()
  L = LinearCombination()
  print "L = ", L
  print "L(3)=", L(3.)
  L += ( 1., f )
  print "L = ", L
  print "L(3)=", L(3.)
  L += ( -2., f )
  print "L = ", L
  print "L(3)=", L(3.)
```

The output should give:

```
L =
L(3)= 0.0
L =  (1.000000 x x)
L(3)= 3.0
L =  (1.000000 x x) + (-2.000000 x x)
L(3)= -3.0
```

Multiplication of `LinearCombinations`

Implement the product of two linear combinations:

$$\left(\sum_i c_i f_i(x)\right)\left(\sum_j d_j g_j(x)\right) = \sum_i \sum_j c_i d_j f_i(x) g_j(x)$$

The product of a linear combination is a linear combination.

The main program is:

```
def main():

  f1 = Gaussian(1.)
  f2 = Gaussian(2.)
  f3 = Gaussian(3.)
  L1 = LinearCombination()
  L2 = LinearCombination()
  print "L1 = ", L1
  print "L2 = ", L2
  print "L1(1)=", L1(1.)
  print "L2(1)=", L2(1.)
  L1 += ( 1., f1 )
  L2 += ( 2., f1 )
```

```
  print "L1 = ", L1
  print "L2 = ", L2
  print "L1(1)=", L1(1.)
  print "L2(1)=", L2(1.)
  L1 += ( -2., f2 )
  L2 += (  3., f2 )
  print "L1 = ", L1
  print "L2 = ", L2
  print "L1(1)=", L1(1.)
  print "L2(1)=", L2(1.)
  L3 = L1*L2
  print "L3 = ", L3
  print "L3(1) = ", L3(1)
  print "L1(1)*L2(1) = ", L1(1)*L2(1)
```

and the output gives:

```
L1 =
L2 =
L1(1)= 0.0
L2(1)= 0.0
L1 =  (1.000000 x exp(-1.000000 x^2))
L2 =  (2.000000 x exp(-1.000000 x^2))
L1(1)= 0.367879441171
L2(1)= 0.735758882343
L1 =  (1.000000 x exp(-1.000000 x^2)) + (-2.000000 x exp(-2.000000 x^2))
L2 =  (2.000000 x exp(-1.000000 x^2)) + (3.000000 x exp(-2.000000 x^2))
L1(1)= 0.0972088746982
L2(1)= 1.14176473205
L3 =  (2.000000 x exp(-2.000000 x^2)) + (3.000000 x exp(-3.000000 x^2))
+ (-4.000000 x exp(-3.000000 x^2)) + (-6.000000 x exp(-4.000000 x^2))
L3(1) =  0.110989664773
L1(1)*L2(1) =  0.110989664773
```