Let complex number z be $z = x + iy$, in python z=complex(x,y) defines this complex number and real part of z can be retrieved via `z.real`. Note that `z.imag` means the imaginary part of z.
If `vx` is a numpy array then `vx.size` is an attribute that stores the size of the array.

1. Two complex numbers fix a square in xy plane. The center of the square (xc, yc) can be obtained from a complex number $z\_center = xc + iyc$. A second complex number $za = xa + iya$ gives the side of the square. Surface area of the square is $(xa)^2 + (ya)^2 = za\overline{za}$. The first side of the square makes angle $\phi = \arg(za) = \arctan(ya/xa)$ with x axis. Function phase(z) from `cmath` returns argument of z.

   (a) Define class Square . Function `__init__` will have two complex arguments: zcenter and za. The 4 attributes of Square will be:Xc,Yc (coordinates of the center), a (length of a side), phi (the angle that the first side makes with x axis)

   (b) Include a method that will return perimeter of the square.

   (c) Add method called `shift` will return a shifted square. The argument of `shift` will be `xshift,yshift`.

   (d) Add a method that will rotate point (x, y) around the origin by angle ϕ. The algorithm will be : `z0=complex(cos(phi),sin(phi));` `z=z0*complex(x,y);` Real and imaginary part of z are then the x and y coordinate of the rotated point. Method should return a tuple consisting of coordinates of the rotated point. The three arguments of the method are x, y coordinates of the point to be rotated and the rotation angle ϕ.

   (e) Add a method that will return `True` if point `(x,y)` lies inside the square else it will return `False`. *Hint*:Rotate point `(x,y)` by $-\phi$ where ϕ is the fourth attribute of Square. Rotate the center of the square by the same angle $(-\phi)$. Now with simple inequalities you can check if rotated point is inside of the rotated square. For example if `x_rotated` is larger than `Xc_rotated+a/2` then this point is not inside of the square.

   (f) Modify constructor `__init__` so that an independent copy of square1 will be produced by the command, square2=Square(square1). For this purpose the second arguments of `__init__` should be made optional. If type of the first argument is Square then all the attributes of the actual Square must be copied from the first argument.

   (g) What will the following program display:

```
from math import *
%class Square defined here
z=complex(1.,0.)
```

```
zz=z+0.0
s1=Square(z,zz)
s2=Square(s1)
s2.a=100.
print('no operation made on s1.a=',s1.a)
s3=s1
s3.a=100.
print('second time,no operation made on s1.a=',s1.a)
```

2. Two vectors of floats `vx,vy` contain x and y coordinates of N points (vx.size=vy.size=N). Using linear algebra package from numpy write a function which will return coefficients of the polynomial of degree N-1 that passes through these points. Function should return a numpy array of size N. The ordering of coefficients is such that the first element of the array is coefficient of 1^0 and the last is coefficient of x^{N-1}. The arguments of the function will be vx and vy.

3. Write a function whose argument is P(x,C) where C is numpy array containing coefficient of a polynomial according to convention given in problem 2 . It should return the values of the polynomial at x (if x is numpy array then it should return a numpy array of the same size)

4. Write function `PdP(x,C)`that will return a tuple. The first element of the tuple will be value of the polynomial at x, the second element will be the value of the derivative of the polynomial at x. The second argument `C` is a numpy array containing coefficients of the polynomial according to to convention introduced in problem2.

5. According to Lagrange interpolation the the value of polynomial that passes through data points x_k, y_k for $k = 0, 1, , N-1$ at x is

$$\sum_{k=0}^{k=N-1} y_k \left(\prod_{l=0, l\neq k}^{N-1} \frac{(x - x_l)}{x_k - x_l} \right)$$

Program formula above as function `Lagrange(x,xdata,ydata)` where `xdata,ydata` are numpy arrays contianing coordinates of data points. Note that `xdata[i]` $\neq$ `xdata[j]` for `i`$\neq$ `j`. Note that N is size of `xdata` and `ydata`