In [13]:

```
from math import *
class Vector(object):

    def __init__(self,vx,vy):
        self.x,self.y=vx,vy

    def __str__(self):
        return 'Vector('+str(self.x)+','+str(self.y)+')'

    def dot(self,otro):
        return self.x*otro.x+self.y*otro.y

    def norm(self):
        return sqrt(self.dot(self))

    def unit(self):
        l=self.norm()
        return Vector(self.x/l,self.y/l)

    def ortogonal(self):
        return Vector(-self.y,self.x)

    def is_parallel(self,otro):
        return self.ortogonal().dot(otro)==0

    def rotate(self,angle):
        return Vector(self.x*cos(angle)-self.y*sin(angle),self.x*sin(angle)+self.y*cos(an
gle))


class Punto(object):

    def __init__(self,px,py):
        self.x=px
        self.y=py

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'

    def distance(self,otro):
        if isinstance(otro,Punto):
            return self.vector_to(otro).norm()
        else:
            return otro.distance(self)
    def __add__(self,v):
        '''
        Translate the Point self by the vector v
        '''
        return Punto(self.x+v.x,self.y+v.y)


    def vector_to(self,otro):
        return Vector(otro.x-self.x,otro.y-self.y)
```

In [14]:
```
print Vector(1,0).rotate(pi/2)
```

```
Vector(6.12323399574e-17,1.0)
```

In [15]:
```
p0=Punto(sqrt(2),3)
p1=Punto(12,34)
p2=Punto(10,34)
print p0
```

```
(1.41421356237,3)
```

In [16]:
```
d=p0.distance(p1)
d
```

Out[16]:
```
32.75757736010167
```

In [17]:
```
from sympy import sqrt,pi,cos,sin,simplify
```

In [18]:
```
print Vector(1,0).rotate(pi/2)
p0=Punto(sqrt(2),3)
p1=Punto(12,34)
p2=Punto(10,34)
print p0
d=p0.distance(p1)
d
simplify(d),type(d),d.evalf()
```

```
Vector(0,1)
(sqrt(2),3)
```

Out[18]:
```
(sqrt(-24*sqrt(2) + 1107), sympy.core.power.Pow, 32.7575773601017)
```

In [19]:
```
from sympy import symbols,solve
a=symbols('a')
p0=Punto(a,3)
p1=Punto(8,5)
p2=Punto(10,34)
r=Recta(p0,p0.vector_to(p1))
d=r.distance(p2).simplify()
d.simplify(),d,d.subs(a,solve(d)[0])
```

Out[19]:
```
((-29*a + 228)/sqrt((a - 8)**2 + 4), (-29*a + 228)/sqrt((a - 8)**2 + 4), 0)
```

In [20]:
```
Vector(2,3).is_parallel(Vector(6/sqrt(2),9/sqrt(2)))
```

Out[20]:
```
True
```

In [21]:
```
from sympy import Rational
r=Rational(3,4)
print r
```

```
3/4
```

In [22]:
```
r.__repr__??
```

In [23]:
```
Punto(0,0).distance(Punto(1,1))
```

Out[23]:
```
sqrt(2)
```

In [24]:
```
import math
```

In [25]:
```
Punto(0.,0.).distance(Punto(1.,1.))
```

Out[25]:
```
1.41421356237310
```

In [26]:
```
p=Punto(Rational(1,2),Rational(1,2))
```

In [27]:
```
print p
```

```
(1/2,1/2)
```

In [28]:
```
a,b,c,d,e,f=symbols('a,b,c,d,e,f')
```

In [29]:
```
p1,p2,p3=Punto(a,b),Punto(c,d),Punto(e,f)
t=Triangle(p1,p2,p3)
```

In [30]:
```
v12=p1.vector_to(p2)
v13=p1.vector_to(p3)
v23=p2.vector_to(p3)
```

In [31]:
```
p12=p1+v12.rotate(-2*pi/6)
p13=p1+v13.rotate(2*pi/6)
p23=p2+v23.rotate(-2*pi/6)
#pi,pj,pij son los triángulos equiláteros construidos sobre los lados del triángulo t
```

In [32]:
```
c1=Triangle(p1,p2,p12).center()
c2=Triangle(p1,p3,p13).center()
c3=Triangle(p2,p3,p23).center()
# los centros de los triángulos anteriores
```

In [33]:
```
print pi
```

```
pi
```

In [34]:
```
d12=c1.distance(c2)
d13=c1.distance(c3)
d23=c2.distance(c3)
print d12
print d13
print d23
#las medidas de los lados del triángulo de vértices c1,c2,c3
```

```
sqrt((-c/2 + e/2 - sqrt(3)*(-b + d)/6 - sqrt(3)*(-b + f)/6)**2 + (-d/2 + f/2 + sqrt(3)*(-
a + c)/6 + sqrt(3)*(-a + e)/6)**2)
sqrt((-a/2 + e/2 - sqrt(3)*(-b + d)/6 + sqrt(3)*(-d + f)/6)**2 + (-b/2 + f/2 + sqrt(3)*(-
a + c)/6 - sqrt(3)*(-c + e)/6)**2)
sqrt((-a/2 + c/2 + sqrt(3)*(-b + f)/6 + sqrt(3)*(-d + f)/6)**2 + (-b/2 + d/2 - sqrt(3)*(-
a + e)/6 - sqrt(3)*(-c + e)/6)**2)
```

In [35]:
```
print (d12-d13).simplify(),(d12-d23).simplify()
# El triángulo de vértices c1,c2,c3 es equilátero
# Este resultado se conoce como teorema de Napoleón.
```

```
0 0
```

In []:
```

```