Practica_0

Andrea del Nido García

GTC0: Funciones básicas

Los puntos serán las listas de sus coordenadas. Ejemplo: El punto A=[1,-4].

Un segmento será una lista de dos puntos (sus extremos).

Una recta será una lista de dos puntos cualesquiera de la recta.

Un triángulo será una lista de tres puntos (sus vértices).

Ejercicios:

escribe las siguientes funciones:

1.	dist(A,B)	distancia entre dos puntos A y B
2.	dist2(A,B)	cuadrado de la distancia entre los puntos A y B
3.	sarea(A,B,C)	area signada de los puntos A,B y C
4.	orientation(A,B,C)	orientación del triángulo A,B,C
5.	midPoint(A,B)	punto medio de los puntos A y B
6.	inSegment(P,s)	test de punto P en segmento s
7.	inTriangle(P,t)	test de punto P en triángulo t
8.	segmentIntersectionTest(a,b)	test de intersección de dos segmentos a y b
9.	lineIntersection(r,s)	punto de intersección de dos rectas
10.	incircle(A,B,C,D)	test de inclusión del punto D en el círculo determinado por los puntos A,B,C
11.	circumcenter(A,B,C)	circuncentro del círculo determinado por los puntos A,B,C

1. distancia entre dos puntos

```
def dist(A,B):
    n1 = (B[0] - A[0]) ** 2
    n2 = (B[1] - A[1]) ** 2
    return math.sqrt(n1 + n2)
```

```
# prueba
dist([0,0],[1,1])
```

```
1.4142135623730951
```

```
# prueba
dist([7,5],[4,1])
```

```
5.0
```

2. cuadrado de la distancia entre los puntos

```
def dist2(A,B):
    n1 = (B[0] - A[0]) ** 2
    n2 = (B[1] - A[1]) ** 2
    return n1 + n2
```

```
# prueba
dist2([0,0],[1,1])
```

```
    2
```

```
# prueba
dist2([7,5],[4,1])
```

```
    25
```

3. area signada

```
def sarea(A,B,C):
    return 1/2*(((B[0]-A[0])*(C[1]-A[1]))-((B[1]-A[1])*(C[0]-A[0])))
```

```
# prueba
sarea([1,1],[0,0],[1,0])
```

```
    1/2
```

```
# prueba
sarea([1,0],[0,0],[1,1])
```

```
    -1/2
```

4. orientación de un triángulo

```
def orientation(A,B,C):
    r = 0
    s = sarea(A,B,C)
    if s < r:
        return -1
    else:
        return 1
```

```
#pruebas
orientation([0,0],[1,0],[0,-1]), orientation([1,0],[0,0],[0,-1]),
orientation([1,1],[0,0],[2,2])
```

```
    (-1, 1, 1)
```

```
#pruebas
orientation([0,0],[1,0],[0,-1])
```

```
    -1
```

```
#pruebas
orientation([1,0],[0,0],[0,-1])
```

```
    1
```

5. punto medio de dos puntos

```
def midPoint(A,B):
    return [(A[0] + B[0])/2, (A[1] + B[1])/2]
```

```
#prueba
midPoint([2,4],[4,8])
```

```
    [3, 6]
```

```
#prueba
midPoint([9,2],[1,2])
```

```
    [5, 2]
```

6. test de punto en segmento

```
def inSegment(P,s):
        A = s[0]
        B = s[1]
        if A[0] <= P[0] <= B[0] and sarea(P,A,B) == 0:
            return true
        else:
            return false
```

```
#pruebas
inSegment([1,1],[[0,0],[2,2]]), inSegment([2,2],[[0,0],[2,2]]),
inSegment([1.0000000000000000001,1],[[0,0],[2,2]])
```

```
    (True, True, False)
```

```
#pruebas
inSegment([1,2],[[0,0],[2,2]])
```

```
    False
```

7. test de punto en triángulo

```
def inTriangle(P,t):
    r = [t[0],t[1]]
    s = [t[0],t[2]]
    n = [t[1],t[2]]
    if inSegment(P,r) or inSegment(P,s) or inSegment(P,n):
        return true
    if orientation(t[0],P,t[2]) == orientation(t[2],P,t[1]) ==
orientation(t[1],P,t[0]):
        return true
    else:
        return false
```

```
#pruebas
inTriangle([0,1],[[-2,0],[2,0],[0,2]]),inTriangle([1,1],[[-2,0],[2,0],
[0,2]]),inTriangle([1,1.00000000000000000000000000000000000000000001],
[[-2,0],[2,0],[0,2]])
```

```
    (True, True, False)
```

8. test de intersección de dos segmentos

```
def segmentIntersectionTest(a,b):
    Ar = a[0]
    Br = a[1]
    As = b[0]
    Bs = b[1]

    # Vectores directores de las rectas
    Vr = [Br[0]-Ar[0],Br[1]-Ar[1]]
    Vs = [Bs[0]-As[0],Bs[1]-As[1]]

    # Caso degenerado en el que las rectas son paralelas.
    if Vr == Vs:
        print "Rectas paralelas"
        return false
    elif inSegment(Ar,b) or inSegment(Br,b) or inSegment(As,a) or
inSegment(Bs,a):
        return true
    else:
        return true
```

```
#pruebas
segmentIntersectionTest([[0,0],[2,2]],[[2,0],[0,2]])
```

```
True
```

```
#pruebas
#Se ha considerado que dos rectas coincidentes tienen intersección.
segmentIntersectionTest([[0,0],[2,2]],[[1,1],[1.5,1.5]])
```

```
True
```

```
segmentIntersectionTest([[0,0],[0,2]],[[0,1],[1,1]])
```

```
True
```

```
segmentIntersectionTest([[0,0],[2,2]],[[2,2],[1,0]])
```

```
True
```

```
#pruebas
segmentIntersectionTest([[0,1],[2,1]],[[0,2],[2,2]])
```

```
Rectas paralelas
False
```

9. punto de intersección de dos rectas

```
def lineIntersection(r,s):
    # Puntos de las rectas
    Ar = r[0]
    Br = r[1]
    As = s[0]
    Bs = s[1]

    # Vectores directores de las rectas
    Vr = [Br[0]-Ar[0],Br[1]-Ar[1]]
    Vs = [Bs[0]-As[0],Bs[1]-As[1]]

    # Caso degenerado en el que las rectas son paralelas
    if Vr[0]/Vr[1] != Vs[0]/Vs[1]:
```

```
        x = (((Vs[1]*As[0]-Vs[0]*As[1])*(-Vr[0]))-((Vr[1]*Ar[0]-Vr[0]*Ar[1])*
(-Vs[0])))/((Vr[1]*Vs[0])-(Vs[1]*Vr[0]))
        y = (-Vr[1]*x + Vr[1]*Ar[0]-Vr[0]*Ar[1])/(-Vr[0])
        return [x,y]
    else:
        print "Rectas paralelas"
        return 0
```

```
#pruebas
lineIntersection([[0,0],[2,2]],[[2,0],[0,2]])
```

```
[1, 1]
```

```
lineIntersection([[0,0],[2,2]],[[0,1],[2,2.99]])
```

```
[200.000000000004, 200.000000000004]
```

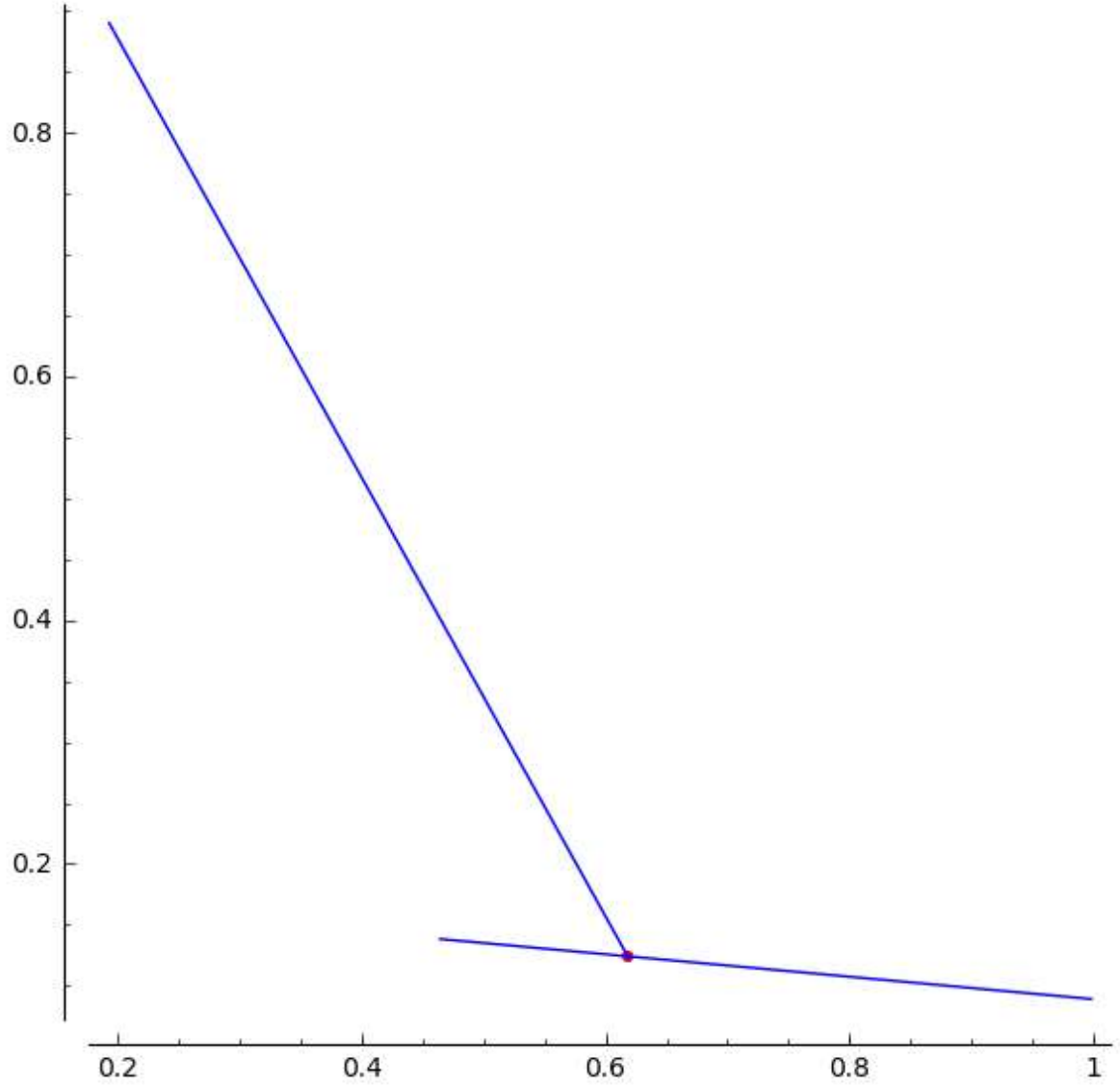
```
lineIntersection([[0,0],[2,2]],[[0,1],[2,3]])
```

```
Rectas paralelas
0
```

```
lineIntersection([[0,0],[2,2]],[[1,1],[3,3]])
```

```
Rectas paralelas
0
```

```
r=[[random(),random()],[random(),random()]]
s=[[random(),random()],[random(),random()]]
line(r,aspect_ratio=1)+line(s)+point(lineIntersection(r,s),color="red",size=20)
```



10. test de inclusión del punto en círculo

Función para obtener el volumen dados cuatro puntos.

```
def svolume(A,B,C,D):
    r = ((A[0]**2) + (A[1]**2))*(D[1]*B[0]+B[1]*C[0]-C[1]*B[0]-D[1]*C[0]-
B[1]*D[0]+C[1]*D[0])+((B[0]**2) + (B[1]**2))*(C[1]*A[0]-D[1]*A[0]-
A[1]*C[0]+D[1]*C[0]-C[1]*D[0])+((C[0]**2) + (C[1]**2))*(D[1]*A[0]-
B[1]*A[0]+A[1]*B[0]-D[1]*B[0]-A[1]*D[0]+B[1]*D[0])+((D[0]**2 + D[1]**2))*
(B[1]*A[0]-C[1]*A[0]-A[1]*B[0]+A[1]*C[0]-B[1]*C[0])
    return r
```

```
#pruebas
svolume([0,0],[0,2],[2,0],[1,1])
```

```
    8
```

```
#incircle test. Entrada cuatro puntos a,b,c,d y salida 1, -1 o 0 dependiendo
de que el punto d sea interior, exterior o este en la frontera del circulo
que determinan a,b y c.

def inCircle(a,b,c,d):
    sigarea = sarea(a,b,c)
    sigvolume = svolume(a,b,c,d)
```

```
    res = sigarea * sigvolume
    solution = 0

    # Comprobar que los tres puntos no estén alineados. El valor devuelto es
2, indicando que no se puede formar un círculo.
    if (a[0] == b[0] == c[0]) or (a[1] == b[1] == c[1]):
        solution = 2
    else:
        if res < 0:
            solution = 1
        elif res > 0:
            solution = -1
        else:
            solution = 0
    return solution
```

```
# pruebas
inCircle([0,0],[0,2],[2,0],[1,1])
```

```
    1
```

```
# pruebas
inCircle([0,0],[0,2],[2,0],[1,1])
```

```
    1
```

```
inCircle([0,0],[0,2],[2,0],[2,2])
```

```
    0
```

```
inCircle([0,0],[0,2],[2,0],[5,5])
```

```
    -1
```

```
inCircle([0,0],[0,2],[0,4],[3,3])
```

```
    2
```

11. circuncentro del círculo determinado por tres puntos

```
def circumcenter(a,b,c):

    # Caso degenerado, los tres puntos están en la misma recta.
    if inSegment(c,[a,b]) or inSegment(b,[a,c]) or inSegment(a,[b,c]):
        print "Tres puntos en la misma recta."
        return 0
    else:
        # Calculamos los puntos medios de los segmento ab y bc.
        m1 = midPoint(a,b)
        m2 = midPoint(b,c)

        # Obtenemos el vector normal a los segmentos ab y bc.
        nr = [(-b[1]+a[1]),(b[0]-a[0])]
        ns = [(-c[1]+b[1]),(c[0]-b[0])]

        # Encontramos las rectas r formada por el punto medio m1 y el vector
normal, perpendicular al segmento ab, igual con la recta s.
        r = [m1,nr]
        s = [m2,ns]
```

```
        # Calculamos la interseccion entre las rectas r y s.
        return lineIntersection(r,s)
```

```
# pruebas
circumcenter([0,0],[1,0],[1,1])
```

```
    [1/3, 1/3]
```

```
circumcenter([0,0],[1,0],[2,0])
```

```
    Tres puntos en la misma recta.
    0
```

```
circumcenter([0,0],[1,0],[2,0.00000000000000001])
```

```
    [-0.000000000000000, 1.00000000000000]
```