```python

############################
#          幾何
############################
def sgn(a):
    if a < -eps: return -1
    if a >  eps: return  1
    return 0

class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y
        pass

    def tolist(self):
        return [self.x,self.y]

    def __add__(self,p):
        return Point(self.x+p.x, self.y+p.y)
    def __iadd__(self,p):
        return self + p

    def __sub__(self,p):
        return Point(self.x - p.x, self.y - p.y)
    def __isub__(self,p):
        return self - p

    def __truediv__(self,n):
        return Point(self.x/n, self.y/n)
    def __itruediv__(self,n):
        return self / n

    def __mul__(self,n):
        return Point(self.x*n, self.y*n)
    def __imul__(self,n):
        return self * n

    def __lt__(self,other):
        tmp = sgn(self.x - other.x)
        if tmp != 0:
            return tmp < 0
        else:
            return sgn(self.y - other.y) < 0

    def __eq__(self,other):
        return sgn(self.x - other.x) == 0 and sgn(self.y - other.y) == 0

    def abs(self):
        return math.sqrt(self.x**2+self.y**2)

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

    def det(self,p):
        return self.x * p.y - self.y*p.x

    def arg(self,p):
        return math.atan2(y,x)

# 点の進行方向 a -> b -> c
def iSP(a,b,c):
    tmp = sgn((b-a).det(c-a))
    if tmp > 0:   return 1   # 左に曲がる場合
    elif tmp < 0: return -1  # 右に曲がる場合
    else: # まっすぐ
        if sgn((b-a).dot(c-a)) < 0: return -2 # c-a-b の順
        if sgn((a-b).dot(c-b)) < 0: return  2 # a-b-c の順
        return 0 # a-c-bの順


# ab,cd の直線交差
def isToleranceLine(a,b,c,d):
    if sgn((b-a).det(c-d)) != 0: return 1 # 交差する
    else:
        if sgn((b-a).det(c-a)) != 0: return 0 # 平行
        else: return -1 # 同一直線

# ab,cd の線分交差 重複，端点での交差もTrue
def isToleranceSegline(a,b,c,d):
    return sgn(iSP(a,b,c)*iSP(a,b,d))<=0 and sgn(iSP(c,d,a)*iSP(c,d,b)) <= 0

# 直線ab と 直線cd の交点 (存在する前提)
def Intersection(a,b,c,d):
    tmp1 = (b-a)*((c-a).det(d-c))
    tmp2 = (b-a).det(d-c)
    return a+(tmp1/tmp2)

# 直線ab と 点c の距離
def DistanceLineToPoint(a,b,c):
    return abs(((c-a).det(b-a))/((b-a).abs()))

# 線分ab と 点c の距離
def DistanceSeglineToPoint(a,b,c):
    if sgn((b-a).dot(c-a)) < 0: # <cab が鈍角
        return (c-a).abs()
    if sgn((a-b).dot(c-b)) < 0: # <cba が鈍角
        return (c-b).abs()
    return DistanceLineToPoint(a,b,c)

# 直線ab への 点c からの垂線の足
def Vfoot(a,b,c):
    d = c + Point((b-a).y,-(b-a).x)
    return Intersection(a,b,c,d)

# 多角形の面積
def PolygonArea(Plist):
    #Plist = ConvexHull(Plist)
    L = len(Plist)
    S = 0
    for i in range(L):
        tmpS = (Plist[i-1].det(Plist[i]))/2
        S += tmpS
    return S

# 多角形の重心
def PolygonG(Plist):
    Plist = ConvexHull(Plist)
    L = len(Plist)
    S = 0
    G = Point(0,0)
    for i in range(L):
        tmpS = (Plist[i-1].det(Plist[i]))/2
        S += tmpS
        G += (Plist[i-1]+Plist[i])/3*tmpS
    return G/S

# 多角形 包含点
def InclusionPoint(Plist,p):
    L = len(Plist)
    cnt = 0
    for i in range(L):
        a,b = Plist[i-1],Plist[i]
        if (a.y <= p.y < b.y) or (b.y <= p.y < a.y):
            vt = (p.y-a.y) /(b.y-a.y)
            if p.x < a.x + vt*(b.x-a.x):
                cnt += 1
    return cnt%2 == 1
```

```
147
148 # 凸法
149 def ConvexHull(Plist):
150     Plist.sort()
151     L = len(Plist)
152     qu = deque([])
153     quL = 0
154     for p in Plist:
155         while quL >= 2 and iSP(qu[quL-2],qu[quL-1],p) == 1:
156             qu.pop()
157             quL -= 1
158         qu.append(p)
159         quL += 1
160
161     qd = deque([])
162     qdL = 0
163     for p in Plist:
164         while qdL >= 2 and iSP(qd[qdL-2],qd[qdL-1],p) == -1:
165             qd.pop()
166             qdL -= 1
167         qd.append(p)
168         qdL += 1
169
170     qd.pop()
171     qu.popleft()
172     hidari = list(qd) + list(reversed(qu)) # 左端開始，左回りPlist
173     return hidari
174
```