LAB 02

by Mohammad Akbar

`Cross Product , Dot Product , Collinear , Relate , Intersection` *2-D*

Point Class

In [1]:

```
class Point:
    # ------ Varaiables START ----------------------- #
    x = 0.0
    y = 0.0
    # ------ Varaiables END ------------------------- #
    # ------ Constructor START ---------------------- #
    def __init__( self , x , y ):
        self.x = x
        self.y = y
    # ------ Constructor END ------------------------ #
    # ------ Print Functions START ------------------ #
    def __repr__(self):
        return "( %f , %f )" % (self.x, self.y)
    def __str__(self):
        return "( %f , %f )" % (self.x, self.y)
    # ------ Print Functions END -------------------- #
    def update(self , x , y):
        self.x = x
        self.y = y
```

Vector Class

In [2]:

```
class Vector:
    # ------ Varaiables START ----------------------- #
    x = 0.0
    y = 0.0
    # ------ Varaiables END ------------------------- #
    # ------ Constructor START ---------------------- #
    def __init__( self , x , y ):
        self.x = x
        self.y = y
    # ------ Constructor END ------------------------ #
    # ------ Print Functions START ------------------ #
    def __repr__(self):
        return "< %f , %f >" % (self.x, self.y)
    def __str__(self):
        return "< %f , %f >" % (self.x, self.y)
    # ------ Print Functions END -------------------- #
```

Cross Product

In [3]:

```
def CrossProduct( a: Vector , b: Vector) -> float:
    return a.x * b.y - a.y * b.x
```

In [4]:

```
print(  CrossProduct( Vector( 1 ,-1 ) , Vector( 2 , 3 ) )  )
print(  CrossProduct( Vector( 2 , 0 ) , Vector( 0 , 2 ) )  )
print(  CrossProduct( Vector( 1 , 3 ) , Vector( 4 , 4 ) )  )
```

```
5
4
-8
```

Dot Product

In [5]:

```
def DotProduct( a: Vector , b: Vector) -> float:
    return a.x * b.x + a.y * b.y
```

In [6]:

```
V1 , V2 = Vector(1,-1) , Vector(2,3)
print( DotProduct( V1, V2 ) )

V1 , V2 = Vector(2,0) , Vector(0,2)
print( DotProduct( V1 , V2 ) )

V1 , V2 = Vector(1,3) , Vector(4,4)
print( DotProduct( V1 , V2 ))
```

```
-1
0
16
```

Collinear

In [7]:

```
def IsCollinear( P1 : Point , P2 : Point , P3 : Point ) -> bool:
    return (P3.y - P2.y)*(P2.x - P1.x) - (P2.y - P1.y)*(P3.x - P2.x) == 0.0
```

In [8]:

```
P1 , P2 , P3 = Point(-5,7) , Point(-4,5) , Point(1,-5)
print(IsCollinear( P1 , P2 , P3 ))

P1 , P2 , P3 = Point(2,4) , Point(4,6) , Point(6,9)
print(IsCollinear( P1 , P2 , P3 ))
```

```
True
False
```

Relate

In [9]:

```
def Relate( P1 : Point , P2 : Point , P3 : Point ):
    print( (P3.y - P2.y)*(P2.x - P1.x) - (P2.y - P1.y)*(P3.x - P2.x) )
```

In [10]:

```
P1 , P2 , P3 = Point(-5,7) , Point(-4,5) , Point(1,-5)
Relate(P1,P2,P3)

P1 , P2 , P3 = Point(-30,10), Point(29,-15), Point(15,28)
Relate(P1,P2,P3)
```

```
P1 , P2 , P3 = Point(5,8), Point(3,5), Point(1,3)
Relate(P1,P2,P3)
```

```
0
2187
-2
```

Intersection

In [11]:

```
def Intersection( P1 : Point , P2 : Point , P3 : Point , P4 : Point , P5 : Point ):
    D  = (P2.x - P1.x)*(P3.y - P4.y) - (P1.y - P2.y)*(P4.x - P3.x)
    Dx = (P1.y*P2.x - P1.x*P2.y)*(P3.y - P4.y) - (P1.y - P2.y)*(P3.y*P4.x - P3.x*P4.y)
    Dy = (P2.x - P1.x)*(P3.y*P4.x - P3.x*P4.y) - (P1.y*P2.x - P1.x*P2.y)*(P4.x - P3.x)
    if D == 0.0 and Dx != 0.0 and Dy != 0.0:
        return 1
    elif D == Dx == Dy == 0.0:
        return 2
    else:
        P5.update( Dx/D , Dy/D)
        return 3

P1 , P2 , P3 , P4 , P5 = Point(2,4) , Point(4,8) , Point(1,7) , Point(3,11) , Point(0,0)
print(Intersection(P1 , P2 , P3 , P4 , P5))

P1 , P2 , P3 , P4 , P5 = Point(2,4) , Point(4,8) , Point(6,12) , Point(8,16) , Point(0,0)
print(Intersection(P1 , P2 , P3 , P4 , P5))

P1 , P2 , P3 , P4 , P5 = Point(1,1) , Point(-5,5) , Point(-9,3) , Point(-4,2) , Point(0,0)
print(Intersection(P1 , P2 , P3 , P4 , P5))
print(P5)
```

```
1
2
3
( 1.000000 , 1.000000 )
```

In [ ]: