

Início

Início_Classes

de matemática, importe raiz quadrada, número Pi

classe Ponto:

função Parâmetros(molde, x, y):

molde.x = x

molde.y = y

molde.CoordenadasDoPonto = (molde.x, ,molde.y)

classe Reta:

função Parâmetros(molde, ax, ay, bx, by):

molde.ax = ax

molde.ay = ay

molde.bx = bx

molde.by = by

função EqdaReta(molde):

Coeficiente Angular = (molde.by – molde.ay) / (molde.bx - molde.ax)

Coeficiente Linear = molde.ay – (Coeficiente Angular) * molde.ax

Equação da Reta = formatado"{Coeficiente Angular} * x + {Coeficiente Linear}"

retorne Equação da Reta

função Distancia(molde):

distancia = raiz quadrada((molde.bx - molde.ax)**2 + (molde.by - molde.ay)**2)

retorne distancia

classe Quadrado(Retangulo):

função Parâmetros(molde, b):

super().Parâmetros(molde, b)

molde.base = b

função Area(molde):

area = molde.base*molde.base

mostre(formatado'Área: {molde.base}² = {area}.'

retorne

função Diagonal(molde):

diagonal = raiz quadrada(((molde.base)**2)+(molde.base)**2)

mostre(formatado'Diagonal: raiz de 2 * lado = raiz de 2 * {molde.base} =

{diagonal:.2flutuantes}.'

retorne

função Perimetro(molde):

perimet = 4*(molde.base)

print(f'Perímetro: 4 x {molde.base} = {perimet}.'

classe Circulo:

função Parâmetros(molde, raio, centro_x, centro_y):

molde.centro_x = centro_x

molde.centro_y = centro_y

molde.raio = raio

função TesteDoRaio(self, ponto_x, ponto_y):

distancia = raiz quadrada((ponto_x - self.centro_x)**2 + (ponto_y - self.centro_y)**2)

se distancia > molde.raio:

```

        mostre("O ponto está fora do círculo.")
    senão se distancia == molde.raio:
        mostre("O ponto está em cima da circunferência do círculo.")
    senão:
        mostre("O ponto está dentro do círculo.")
    area = pi * (molde.Raio)**2
    mostre(formatado'Área:  $\pi \times r^2 = \{pi\} \times \{molde.Raio\}^2 = \{area\}.$ ')

```

classe Triangulo:

```

    função Parâmetros(molde, ax, ay, bx, by, cx, cy):
        molde.ax = ax
        molde.ay = ay
        molde.bx = bx
        molde.by = by
        molde.cx = cx
        molde.cy = cy
    função LadosDoTriangulo(molde):
        lado1 = raiz quadrada((molde.ax - molde.bx)**2 + (molde.ay - molde.by)**2)
        lado2 = raiz quadrada((molde.bx - molde.cx)**2 + (molde.by - molde.cy)**2)
        lado3 = raiz quadrada((molde.cx - molde.ax)**2 + (molde.cy - molde.ay)**2)
        l = [lado1, lado2, lado3]
        mostre(formatado'''
        Distancia do primeiro ponto até o segundo = {lado1}
        Distancia do segundo ponto até o terceiro = {lado2}
        Distancia do terceiro ponto até o primeiro = {lado3}''')
    função Area(molde):
        lado1 = raiz quadrada((molde.ax - molde.bx)**2 + (molde.ay - molde.by)**2)
        lado2 = raiz quadrada((molde.bx - molde.cx)**2 + (molde.by - molde.cy)**2)
        lado3 = raiz quadrada((molde.cx - molde.ax)**2 + (molde.cy - molde.ay)**2)
        l = [lado1, lado2, lado3]
        se (lado1 + lado2 > lado3) e (lado1 + lado3 > lado2) e (lado2 + lado3 > lado1):
            mostre('Elas conseguem formar um triângulo.')
        se lado1 igual a lado2 igual a lado3:
            mostre('E esse triângulo é equilátero.')
            area = ((lado1)**2)*(raiz quadrada(3))/4
            mostre(formatado'Área do triângulo equilátero: {area}')
```

senão se lado1 igual a lado2 ou lado1 igual a lado3 ou lado2 igual a lado3:

```

            mostre('E esse triângulo é isósceles.')
            maior = [ ]
            maior.adicionar(número máximo(l))
            h = raiz quadrada(lado1**2 + (maior/2)**2)
            area = (maior * h)/2
            mostre(formatado'Área do triângulo isósceles: {area}')
```

senão se lado1 diferente de lado2 diferente de lado3:

```

            mostre('E esse triângulo é escaleno.')
            #Fórmula de Heron para calcular a área do triângulo escaleno.
            semiperimetro = (lado1 + lado2 + lado3)/2
            area = raiz quadrada(semiperimetro*(semiperimetro -
lado1)*(semiperimetro-lado2)*(semiperimetro-lado3))
            mostre(formatado'Área do triângulo escaleno: {area}')
```

senão:

```

            mostre('Não conseguem formar um triângulo.')

```

Fim_Classes

```

Início_Main
de reta, importe Reta
l = [ ]
para c no intervalo(0, 4):
    p = inteiro(entrada('Digite um número que complete as coordenadas A(Xa, Ya) e B(Xb, Yb):
'))
    l.adicione(p)
r1 = Reta(l[0], l[1], l[2], l[3])
r1.Distancia()
r1.EquacaoDaReta()

de quadrado, importe Quadrado

#Diferentemente do retângulo, o quadrado não tem base e altura diferentes. Portanto, basta
saber um dos dados para aplicar as funções.
lado = inteiro(entrada('Qual o lado do quadrado? '))
l1 = Quadrado(lado)
l1.Area()
l1.Perimetro()
l1.Diagonal()

de circulo, importe Circulo

c1 = Circulo(5, 0, 0)
c1.TesteDoRaio(3, 4)
c1.Area()

de triangulo, importe Triangulo

#O init está na respectiva ordem: ax, ay, bx, by, cx, cy.
x = [ ]
para c no intervalo(0, 6):
    p = inteiro(entrada('Digite um número que complete as coordenadas A(x, y), B(x, y) e C(x,
y): '))
    x.adicionar(p)
t1 = Triangulo(x[0], x[1], x[2], x[3], x[4], x[5])
t1.LadosDoTriangulo()
t1.Area()

de retangulo importe Retangulo

b2 = inteiro(entrada('Qual a base do Retângulo? '))
h2 = inteiro(entrada('Qual a altura do Retângulo? '))
rg = Retangulo(b2, h2)
rg.Area()
rg.Diagonal()

```

Fim
Fim_Main