

Início

Início_Classes

de matemática, importe raiz quadrada

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:

```
função Parâmetros(molde, lado):  
    molde.lado = lado  
função Area(molde):  
    area = (molde.lado)**2  
    mostre(formatado'Área: {molde.lado}² = {area}.')  
    retorne  
função Diagonal(molde):  
    diagonal = raiz quadrada(((molde.lado)**2)+(molde.lado)**2)  
    mostre(formatado'Diagonal: raiz de {molde.lado}² + {molde.lado}²=  
    {diagonal:.2f}.')  
    retorne  
função Perimetro(molde):  
    perimet = 4*(molde.lado)  
    print(f'Perímetro: 4 x {molde.lado} = {perimet}.')
```

classe Circulo:

```
função Parâmetros(molde, centro, Raio, ax, ay, ca, cb):  
    molde.c = centro  
    molde.Raio = raio  
    molde.ax = ax  
    molde.ay = ay  
    molde.ca = ca  
    molde.cb = cb  
função TesteDoRaio(molde):  
    distancia = raiz quadrada((molde.ax - molde.ay)**2 + (molde.ca -  
    molde.cb)**2)  
    mostre(formatado'Distância: raiz de {molde.ax - molde.ay}² + {self.ca -
```

```

self.cb}^2 = {distancia}')
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
senão:
        se distancia < molde.Raio:
            mostre("O ponto está dentro do círculo")

função Area(molde):
    pi = 3.14159265
    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 isóceles: {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 = raíz 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):
    Ya e p = inteiro(entrada('Digite um número que complete as coordenadas A(Xa,
        B(Xb, Yb): '))
        l.adicione(p)
r1 = Reta(l[0], l[1], l[2], l[3])
r1.Distancia()
mostre(formatado'''
{r1}
''')
r1.EquacaodaReta()
mostre(formatado'''
{r1}
''')
```

de quadrado, importe Quadrado

```

lado = inteiro(entrada('Qual o lado do quadrado? '))
l1 = Quadrado(lado)
l1.Area()
mostre(formatado'''
{l1}
''')
l1.Perimetro()
mostre(formatado'''
{l1}
''')
l1.Diagonal()
mostre(formatado'''
{l1}
''')
```

de circulo, importe Circulo

#O init está na respectiva ordem: centro, raio, ax, ay, ca, cb. Sendo esses últimos quatro os pontos de coordenadas para descobrir distância.

```

x = [ ]
```

```

para c no intervalo(0, 4):
    p = inteiro(entrada('Digite um número que complete as coordenadas A(x, y) e
    C(a, b): '))
    x.adicione(p)
c1 = Circulo(1, 5, x[0], x[1], x[2], x[3])
c1.Area()
mostre(formatado"{c1}
")
c1.TesteDoRaio()
mostre(formatado"
{c1}
")

```

de triangulo, importe Triangulo

#O init está na respectiva ordem: ax, ay, bx, by, cx, cy.

```
x = [ ]
```

```
para c no intervalo(0, 6):
```

```

B(x, y) p = inteiro(entrada('Digite um número que complete as coordenadas A(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()
```

```
mostre(formatado"{t1}
```

```

")

```

```
t1.Area()
```

```
mostre(formatado"{t1}
```

```

")

```

Fim_Main

Fim