

# **Rasterização de Linhas**

# Equação da reta

- Lei de formação:

$$y = f(x) = ax + b$$

- 'a' → coeficiente angular
- 'b' → coeficiente linear
- 'x' → variável independente

# Equação da reta

- Exemplo:

$$f(x) = 2x + 4$$

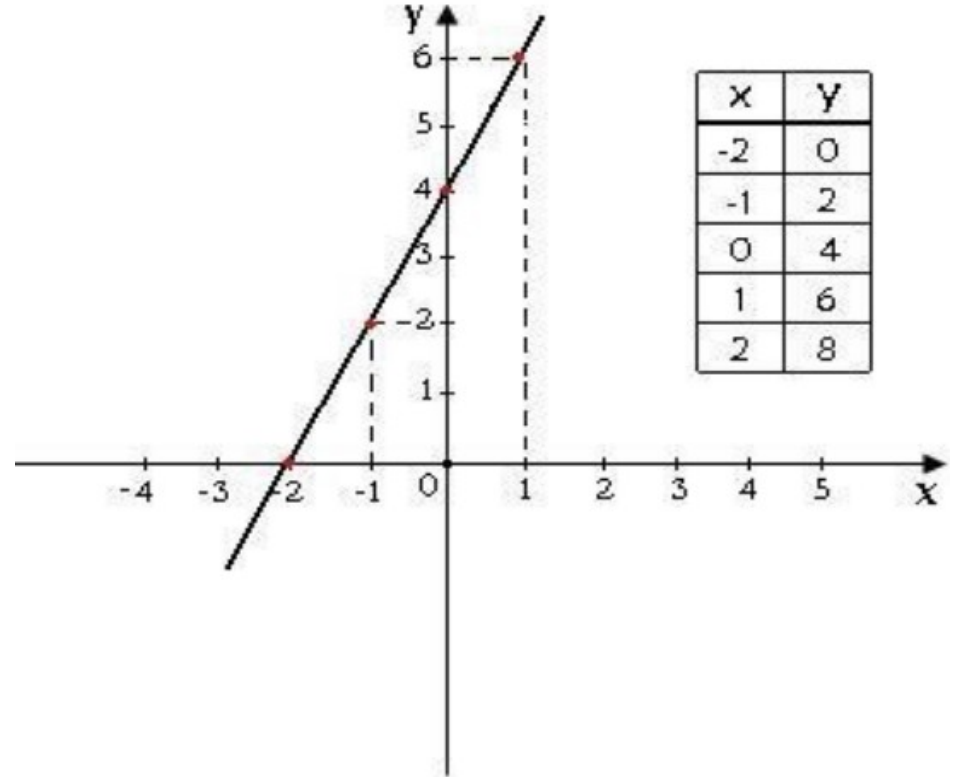
- $f(x) = 2(-2) + 4 = 0$

- $f(x) = 2(-1) + 4 = 2$

- $f(x) = 2(0) + 4 = 4$

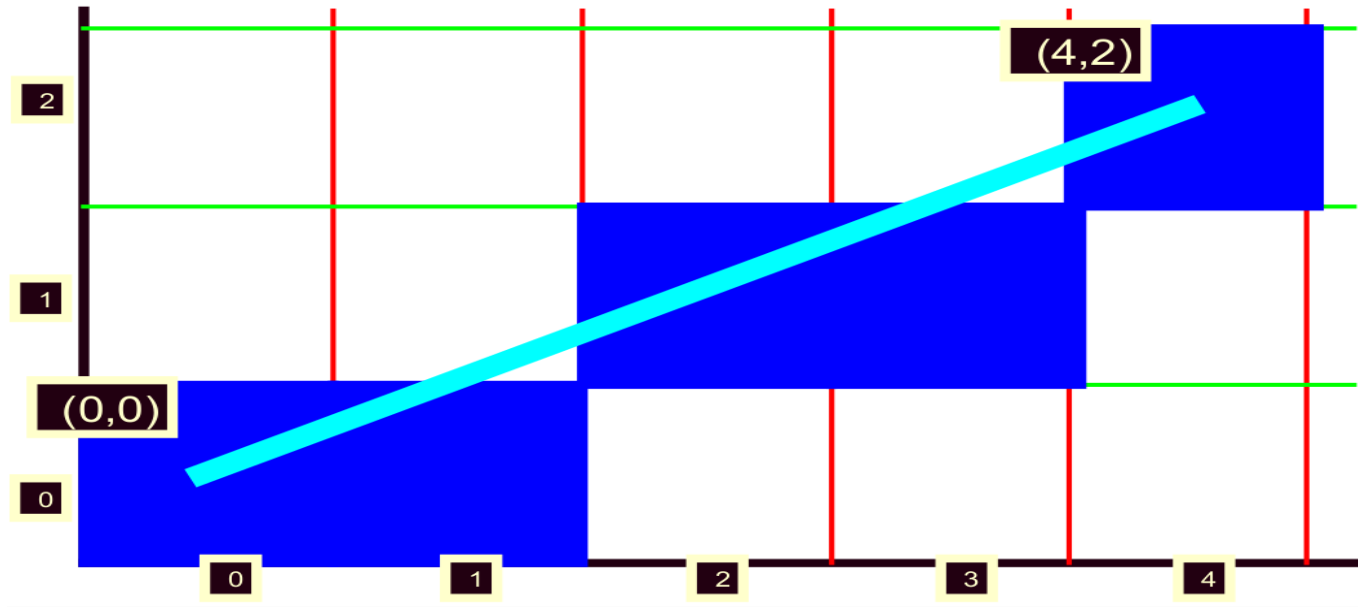
- $f(x) = 2(1) + 4 = 6$

- $f(x) = 2(2) + 4 = 8$



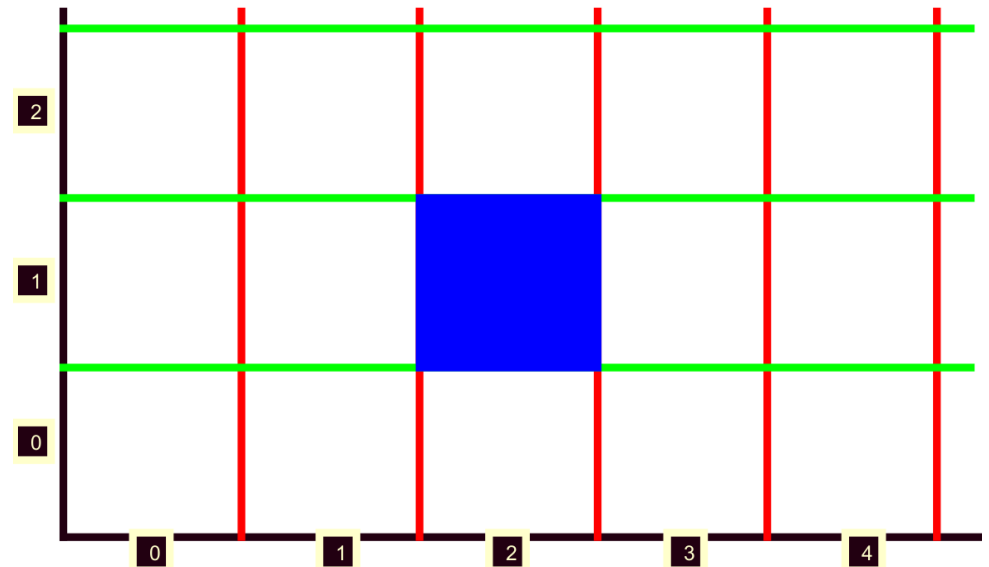
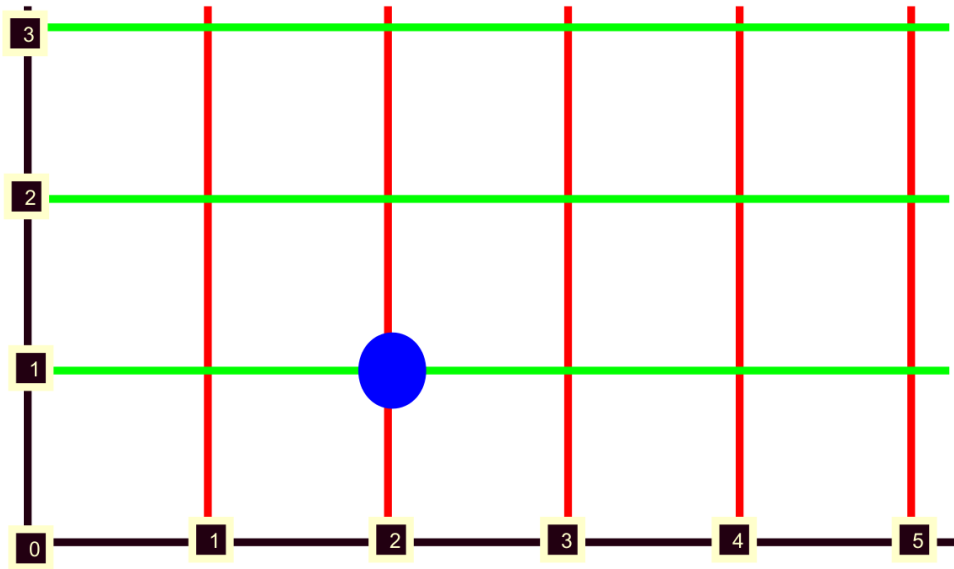
# Rasterização

- Converte informação de vértices/arestas em pixels a serem mostrados na imagem



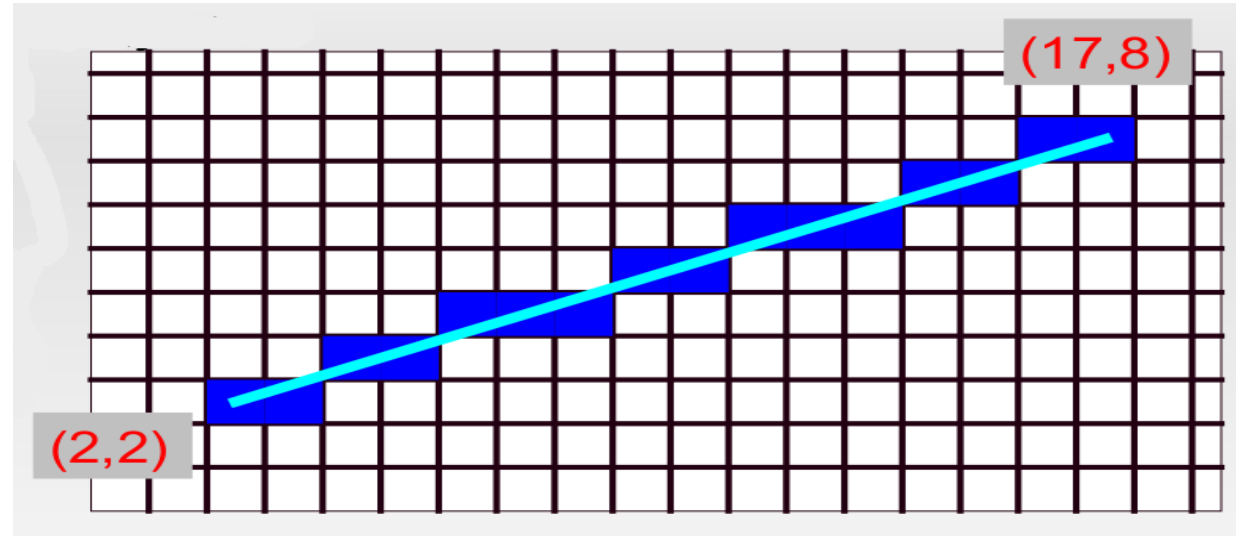
# Rasterização

- Poderíamos obter a reta mapeando cada ponto do SRU para o SRD, mas:



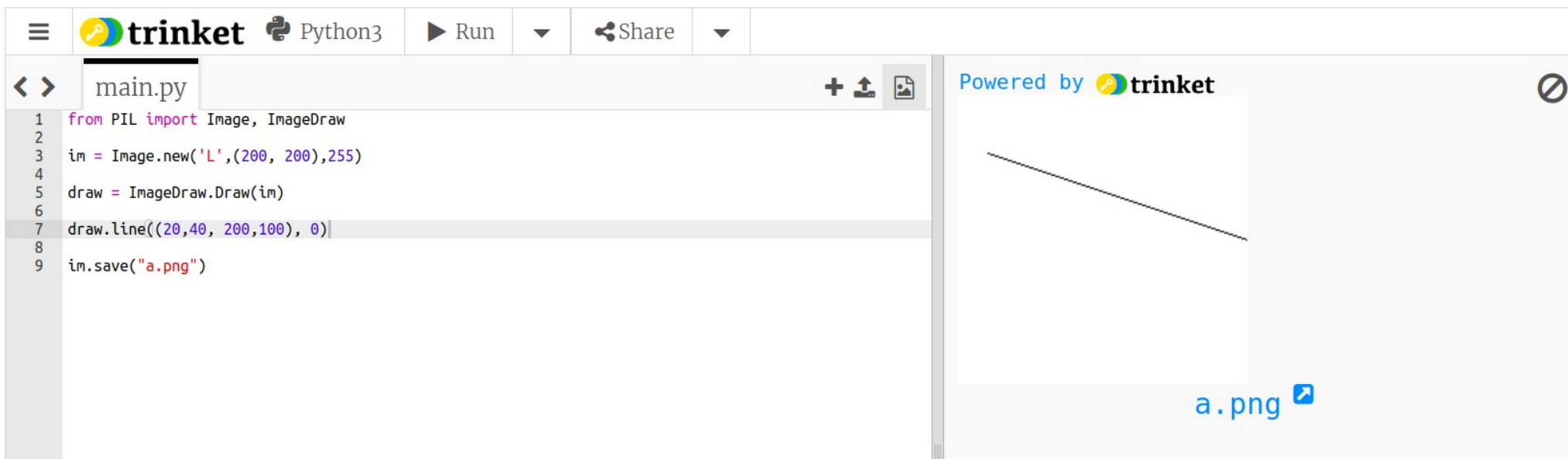
# Rasterização

- Objetivo:
  - Aparência contínua
  - Uniformidade
  - Próximo a linha ideal
  - Rapidez de rasterização



# Rasterização

- Método Slope-Intercept
- Método DDA
- Algoritmo de Bresenham



The image shows a screenshot of the Trinket Python IDE. The interface includes a top navigation bar with the Trinket logo, Python3 version, a Run button, and a Share button. Below this is a file browser showing 'main.py'. The code editor contains the following Python code:

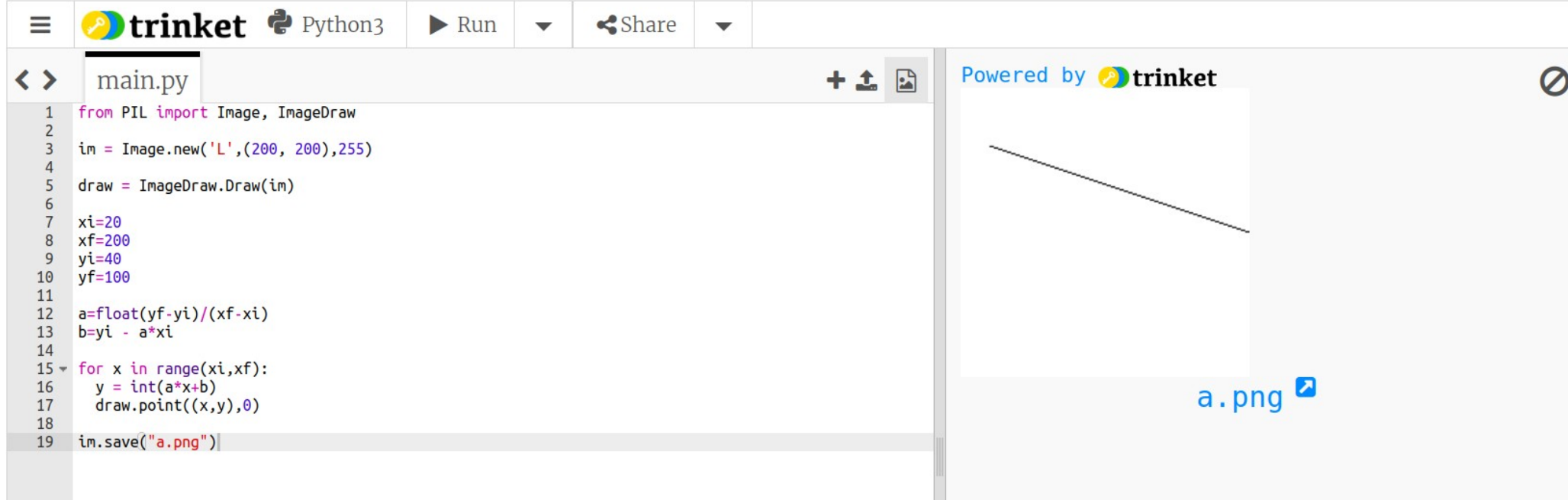
```
1 from PIL import Image, ImageDraw
2
3 im = Image.new('L', (200, 200), 255)
4
5 draw = ImageDraw.Draw(im)
6
7 draw.line((20, 40, 200, 100), 0)
8
9 im.save("a.png")
```

The right side of the IDE displays the output of the code, which is a 200x200 pixel grayscale image containing a single black line. The line starts at the coordinates (20, 40) and ends at (200, 100). Below the image, the filename 'a.png' is shown with a download icon. The output area is powered by Trinket.

# Método Slope-Intercept

- Da equação:  $y = f(x) = ax + b$

xi=20  
xf=200  
yi=40  
yf=100



The image shows a Trinket Python IDE interface. The left pane displays a Python script named 'main.py' with the following code:

```
1 from PIL import Image, ImageDraw
2
3 im = Image.new('L', (200, 200), 255)
4
5 draw = ImageDraw.Draw(im)
6
7 xi=20
8 xf=200
9 yi=40
10 yf=100
11
12 a=float(yf-yi)/(xf-xi)
13 b=yi - a*xi
14
15 for x in range(xi,xf):
16     y = int(a*x+b)
17     draw.point((x,y),0)
18
19 im.save("a.png")
```

The right pane shows the output of the script, a white canvas with a black line segment. The line starts at the point (20, 40) and ends at (200, 100). Below the canvas, the text 'a.png' is displayed with a download icon. The interface includes a top navigation bar with the Trinket logo, 'Python3', 'Run', and 'Share' buttons. The right pane is titled 'Powered by trinket'.



# Método Slope-Intercept

- Da equação:  $y = f(x) = ax + b$

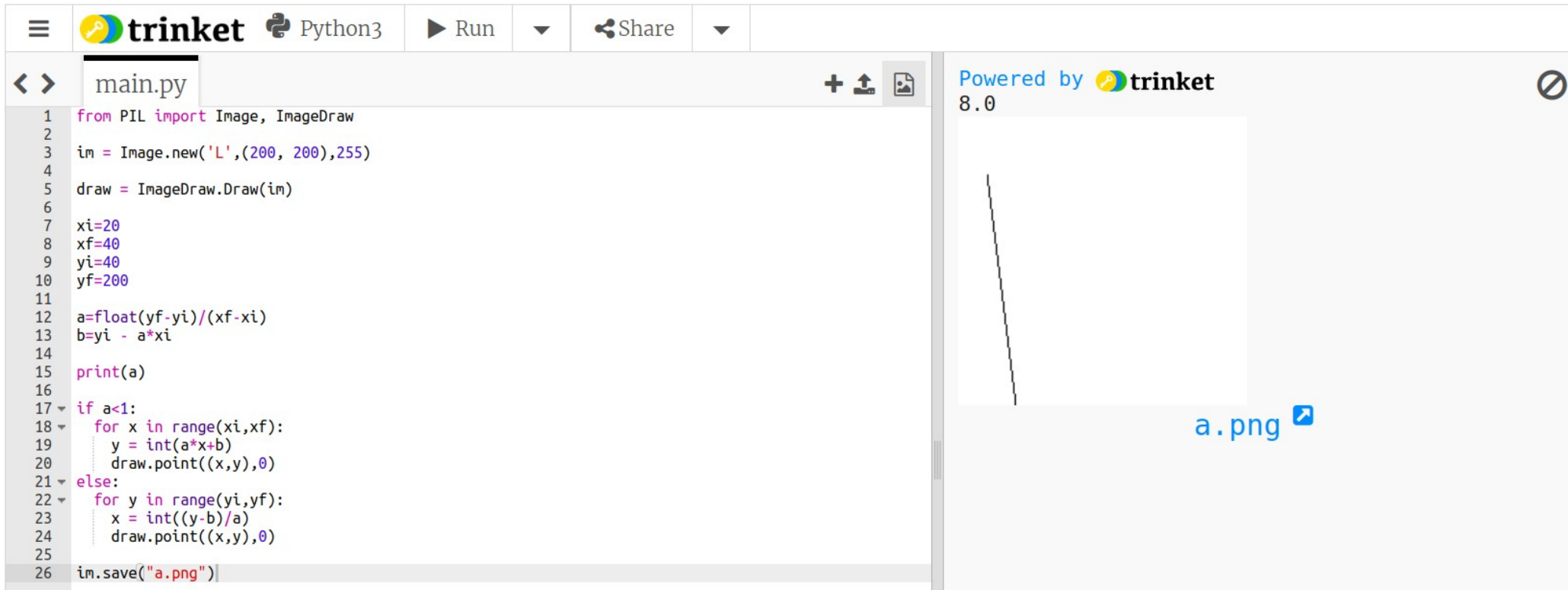
$x_i=20$   
 $x_f=40$   
 $y_i=40$   
 $y_f=200$

**O que acontece?**

# Método Slope-Intercept

- Da equação:  $y = f(x) = ax + b$

xi=20  
xf=40  
yi=40  
yf=200



The image shows a Trinket Python IDE interface. The left pane displays a Python script named 'main.py' with the following code:

```
1 from PIL import Image, ImageDraw
2
3 im = Image.new('L',(200, 200),255)
4
5 draw = ImageDraw.Draw(im)
6
7 xi=20
8 xf=40
9 yi=40
10 yf=200
11
12 a=float(yf-yi)/(xf-xi)
13 b=yi - a*xi
14
15 print(a)
16
17 if a<1:
18     for x in range(xi,xf):
19         y = int(a*x+b)
20         draw.point((x,y),0)
21 else:
22     for y in range(yi,yf):
23         x = int((y-b)/a)
24         draw.point((x,y),0)
25
26 im.save("a.png")
```

The right pane shows the output of the script, which is a plot of a line segment. The plot is titled 'Powered by trinket 8.0' and shows a vertical line segment on a white background. Below the plot, the filename 'a.png' is displayed with a download icon.

# Método DDA

- Melhora o método anterior
- DDA → Digital Differential Analyzer
  - Foca na equação:  $a = dy/dx$
- Objetivo:
  - Fazer uma operação a menos dentro do ciclo

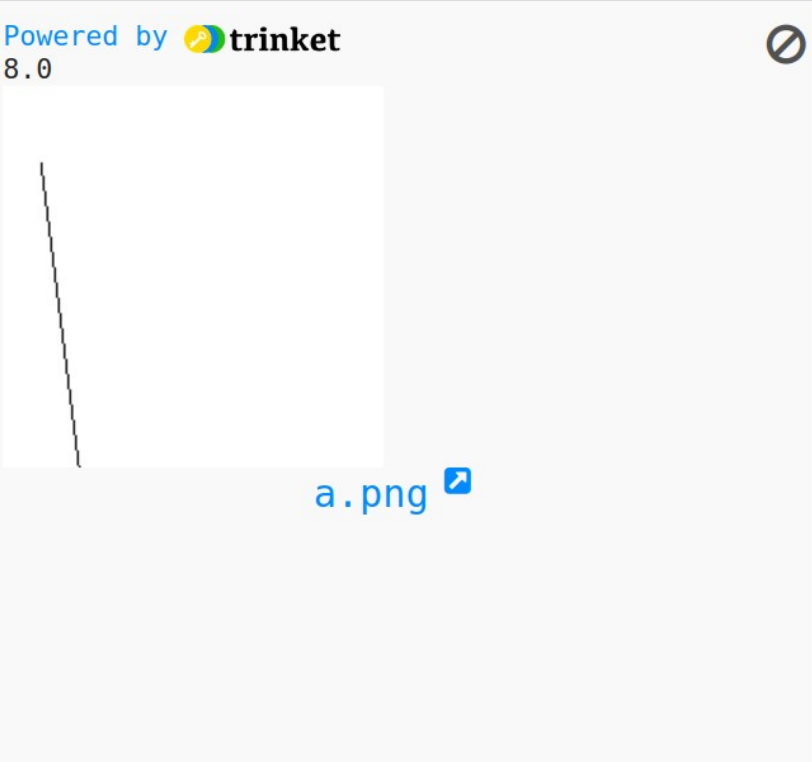
# Método DDA

Python3 Run Share

main.py

```
1 from PIL import Image, ImageDraw
2
3 im = Image.new('L',(200, 200),255)
4
5 draw = ImageDraw.Draw(im)
6
7 xi=20
8 xf=40
9 yi=40
10 yf=200
11
12 a=float(yf-yi)/(xf-xi)
13 b=yi - a*xi
14
15 print(a)
16
17 if a<1:
18     y=yi
19     for x in range(xi,xf):
20         y+=a
21         draw.point((x,y),0)
22 else:
23     x=xi
24     for y in range(yi,yf):
25         x+=1/a
26         draw.point((x,y),0)
27
28 im.save("a.png")
```

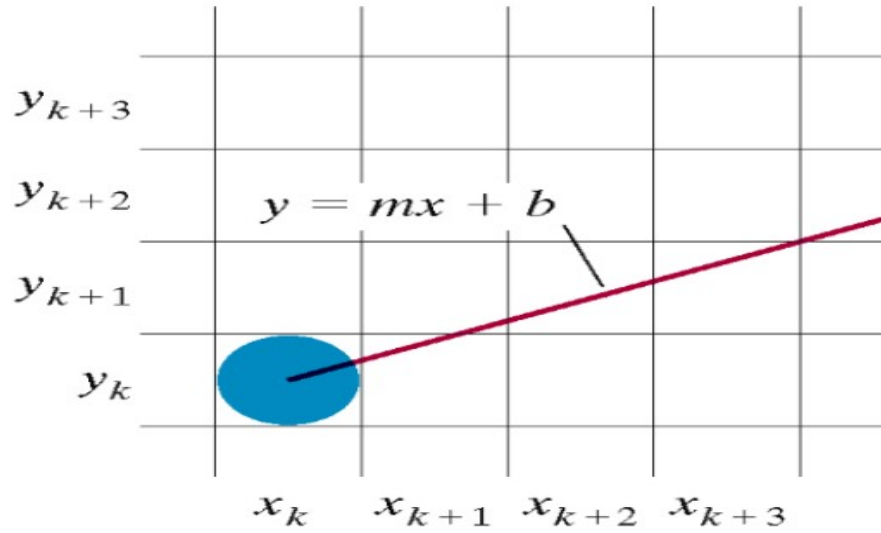
Powered by trinket 8.0



a.png

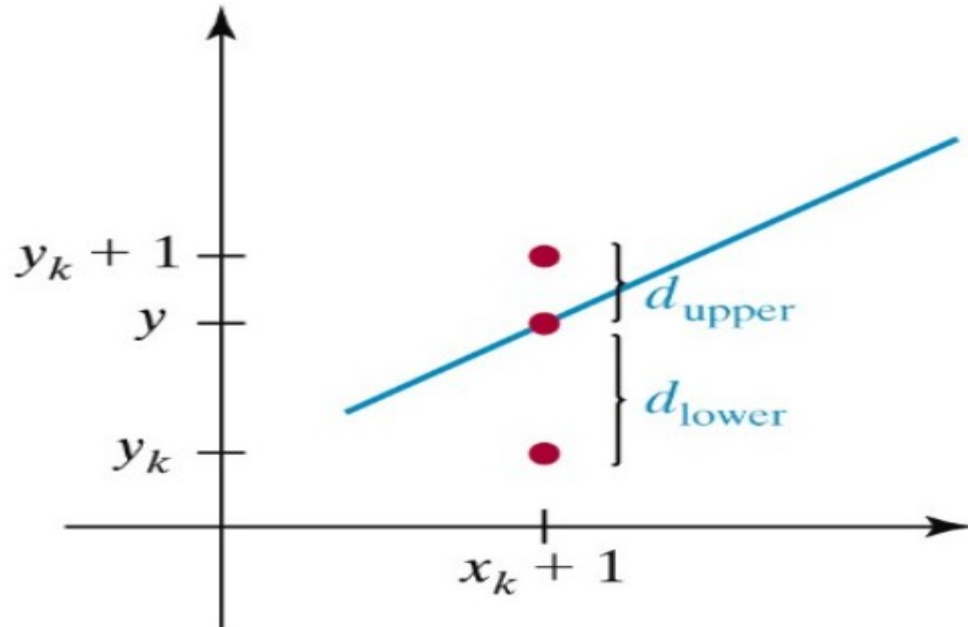
# Algoritmo de Bresenham

- Algoritmo rápido eficiente e preciso
- Melhora o DDA para usar aritmética inteira



# Algoritmo de Bresenham

- $d$  (upper, lower) → distâncias entre o valor real e discretizado



# Algoritmo de Bresenham

- Decisão de qual pixel preencher:
  - Sinal de  $d_{\text{lower}} - d_{\text{upper}}$

$$d_{\text{lower}} - d_{\text{upper}} \left\{ \begin{array}{l} + \rightarrow y_k + 1 \\ - \rightarrow y_k \end{array} \right.$$

# Algoritmo de Bresenham

- Cálculo de:  $d_{\text{lower}} - d_{\text{upper}}$

$$y = m(x_k + 1) + b \quad (\text{cálculo da coordenada } y \text{ na linha } x_k + 1)$$

$$d_{\text{lower}} = y - y_k = m(x_k + 1) + b - y_k$$

$$d_{\text{upper}} = (y_k + 1) - y = y_{k+1} - m(x_k + 1) - b$$

$$d_{\text{lower}} - d_{\text{upper}} = 2m(x_k + 1) - 2y_k + 2b - 1$$

$$d_{\text{lower}} - d_{\text{upper}} = 2 \left( \frac{dy}{dx} \right) (x_k + 1) - 2y_k + 2b - 1$$



# Algoritmo de Bresenham

- Parâmetro de decisão:  $p_k$   
 $p_k = dx (d_{lower} - d_{upper})$
- Sinal é o mesmo de:  $d_{lower} - d_{upper}$ 
  - Cálculo mais simples

$$p_k = dx ( 2 ( dy / dx )(x_k + 1) - 2y_k + 2b - 1 )$$

$$p_k = 2 \, dy \, x_k - 2 \, dx \, y_k + c$$

Onde  $c$  é uma constante;  $c = 2 \, dy + 2 \, dx \, b - dx$

# Algoritmo de Bresenham

- Achando  $p_{k+1}$

$$\left\{ \begin{array}{ll} p_{k+1} = p_k + 2 \, dy & p_k < 0 \rightarrow \text{negativo} \\ p_{k+1} = p_k + 2 \, (dy - dx) & p_k \geq 0 \rightarrow \text{positivo} \end{array} \right.$$

- $p_k = 2 \cdot dy + dx$                        $k = 0$

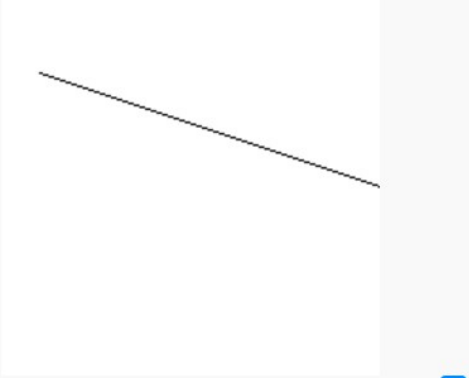
# Algoritmo de Bresenham

trinket Python3 Run Share

main.py

```
1 from PIL import Image, ImageDraw
2
3 im = Image.new('L',(200, 200),255)
4
5 draw = ImageDraw.Draw(im)
6
7 xi=20
8 xf=200
9 yi=40
10 yf=100
11
12 dx=abs(xf-xi)
13 dy=abs(yf-yi)
14 p=2*dy-dx
15
16 if xi>xf:
17     x=xf
18     y=yf
19     xf=xi
20 else:
21     x=xi
22     y=yi
23
24 draw.point((x,y),0)
25 while x<xf:
26     x+=1
27     if p<0:
28         p+=2*dy
29     else:
30         y+=1
31         p+=2*(dy-dx)
32     draw.point((x,y),0)
33
34 im.save("a.png")
35 # From Hearn & Baker's Computer Graphics with OpenGL, 3rd Edition
```

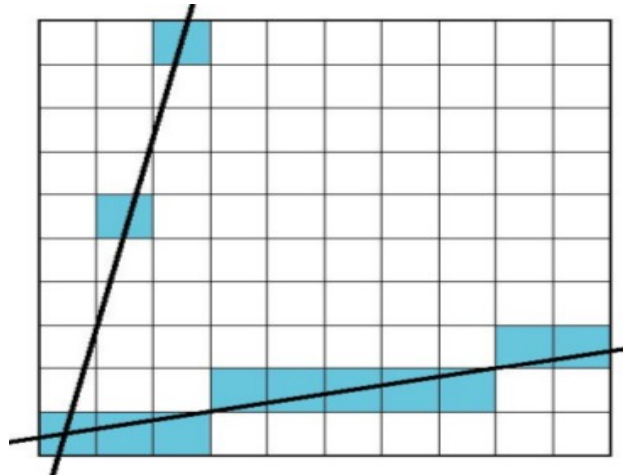
Powered by trinket



a.png

# Algoritmo de Bresenham

- Problema
  - Para cada  $x$ , desenhar o pixel no melhor  $y$



- Solução: simetria  $\rightarrow$  trocar  $x$  por  $y$

# Exercício

- Aplicar o algoritmo de Bresenham para:

$x_i=20$   
 $x_f=40$   
 $y_i=40$   
 $y_f=200$

Solução: simetria → trocar x por y