Activity 1: Line Creation Algorithms

Contributors go to *Keane Dalisay*, *Nel Alanan* and *Prince Alexander Malatuba*.

🎉 Cheers!

A note on organizing code

```
class LineTemp:

  def __init__(self, x1, y1, x2, y2):
    self.strt_pnt = [x1, y1]
    self.end_pnt = [x2, y2]
```

We used a class to organize the different algorithms as their own methods (functions) when creating the object.

The `self` points to the object when the class is initialized/called into a variable.

```
line = LineTemp(1, 2, 10, 8)
# The self here is the variable (or object) named line
```

Digital Differential Analyzer Algorithm

```
def dda(self):
    plt.title("Digital Differential Analyzer Algorithm")
    plt.xlabel("X-Axis")
    plt.ylabel("Y-Axis")

    strt_x = self.strt_pnt[0]
    strt_y = self.strt_pnt[1]
    # Above is simply (x1, y1)

    dx = abs(self.end_pnt[0] - strt_x) # Distance between (x2) and (x1)
    dy = abs(self.end_pnt[1] - strt_y) # Distance between (y2) and (y1)

    steps = dx if dx >= dy else dy
    # Compare if total distance of (x) is
    # more than total distance of (y)

    x_inc = float(dx / steps)
    y_inc = float(dy / steps)
    # Incremental values of (x) and (y) per point

    frmt = 'bo-'
```

```
    for i in range(steps):
      plt.plot(strt_x, strt_y, frmt)

      strt_x += x_inc
      strt_y += y_inc

    midpoint_x = (self.strt_pnt[0] + self.end_pnt[0]) / 2
    midpoint_y = (self.strt_pnt[1] + self.end_pnt[1]) / 2
    plt.plot(midpoint_x, midpoint_y, 'go-') # Plot midpoint

    plt.show()
```

Bresenham's Algorithm

```
def bresenhams(self):
    plt.title("Bresenham's Algorithm")
    plt.xlabel("X-Axis")
    plt.ylabel("Y-Axis")

    strt_x = self.strt_pnt[0]
    strt_y = self.strt_pnt[1]
    # Above is simply (x1, y1)

    dx = self.end_pnt[0] - strt_x # Distance between (x2) and (x1)
    dy = self.end_pnt[1] - strt_y # Distance between (y2) and (y1)

    decision = (2 * dy) - dx # Decision maker...
    # To pinpoint the next (y) value to take based on the
    # midpoint of the next (x, y) coordinate

    frmt = 'ro-'
    while(strt_x <= self.end_pnt[0]):
      plt.plot(strt_x, strt_y, frmt)

      strt_x += 1

      if decision < 0: # If line is below midpoint
        decision = decision + (2 * dy) # (y) remains the same
      else:
        decision = decision + (2 * dy) - (2 * dx) # If line is above midpoint
        strt_y += 1 # Increment (y) to 1

    midpoint_x = (self.strt_pnt[0] + self.end_pnt[0]) / 2
    midpoint_y = (self.strt_pnt[1] + self.end_pnt[1]) / 2
    plt.plot(midpoint_x, midpoint_y, 'go-') # Plot midpoint

    plt.show()
```

Mid-Point Algorithm

```
def midpoint(self):
    plt.title("Midpoint Algorithm")
    plt.xlabel("X-Axis")
    plt.ylabel("Y-Axis")

    strt_x = self.strt_pnt[0]
    strt_y = self.strt_pnt[1]
    # Above is simply (x1, y1)

    dx = self.end_pnt[0] - strt_x # Distance between (x2) and (x1)
    dy = self.end_pnt[1] - strt_y # Distance between (y2) and (y1)

    decision = (2 * dy) - dx # Decision maker...
    # The algorithm is actually identical
    # to that of Bresenham's

    frmt = 'yo-'
    while(strt_x <= self.end_pnt[0]):
      plt.plot(strt_x, strt_y, frmt)
      strt_x += 1

      if decision < 0: # If line is below midpoint
        decision = decision + (2 * dy) # (y) remains the same
      else:
        decision = decision + 2 * (dy - dx) # If line is above midpoint
        strt_y += 1 # Increment (y) to 1


    midpoint_x = (self.strt_pnt[0] + self.end_pnt[0]) / 2
    midpoint_y = (self.strt_pnt[1] + self.end_pnt[1]) / 2
    plt.plot(midpoint_x, midpoint_y, 'go-') # Plot midpoint

    plt.show()
```

Algorithm's in action

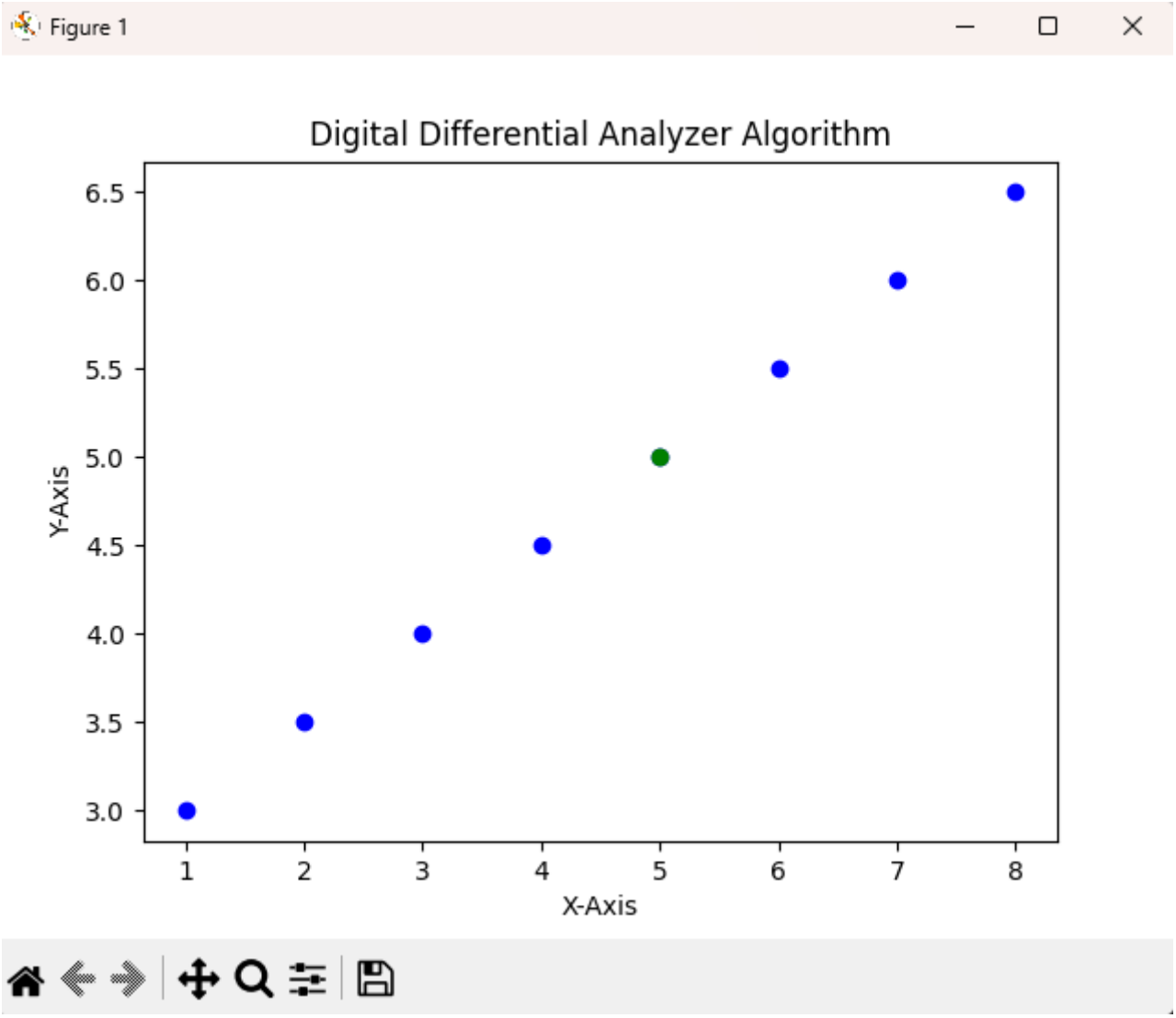


Figure 1
Digital Differential Analyzer Algorithm
Y-Axis
X-Axis
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
1
2
3
4
5
6
7
8

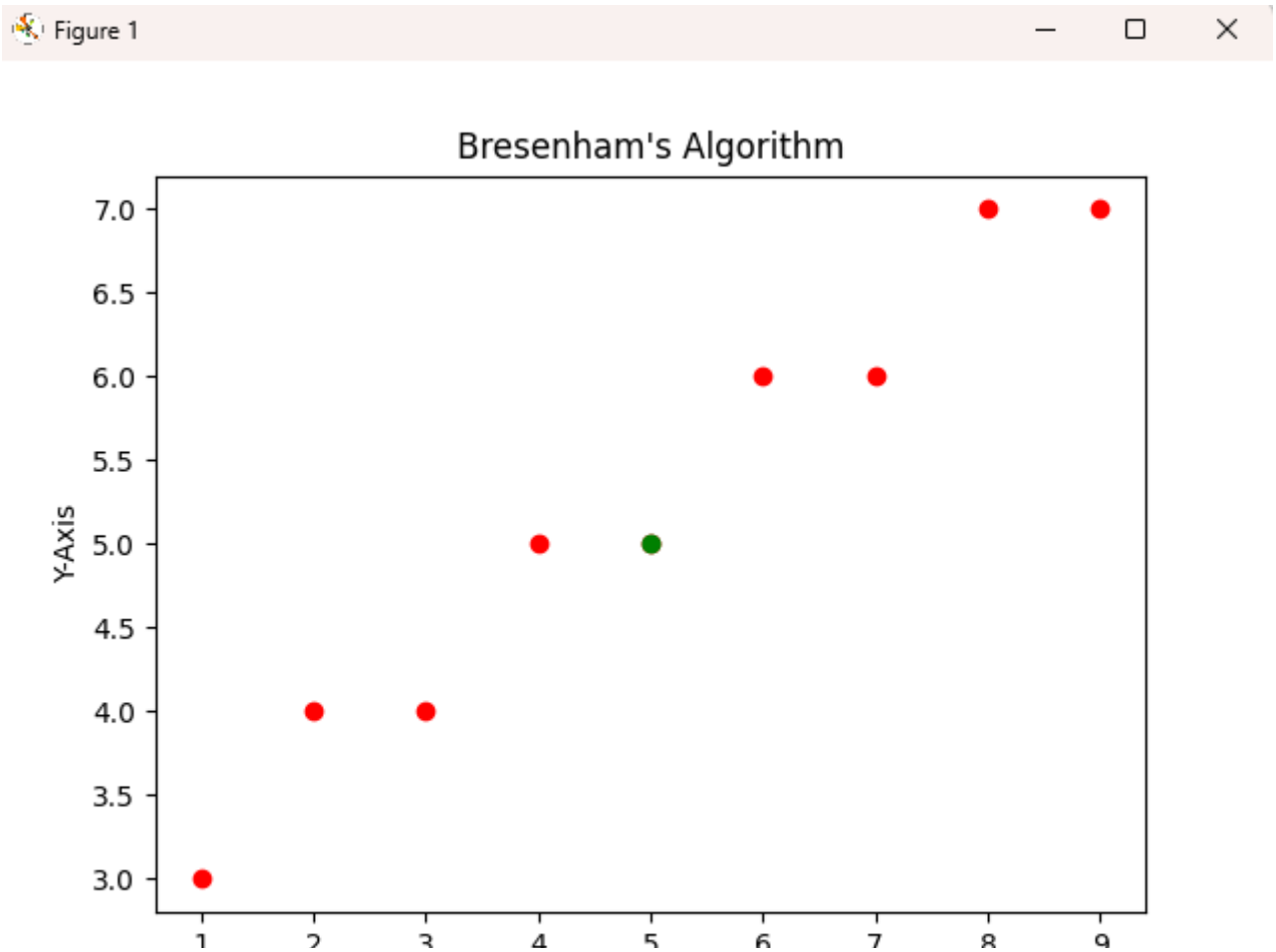
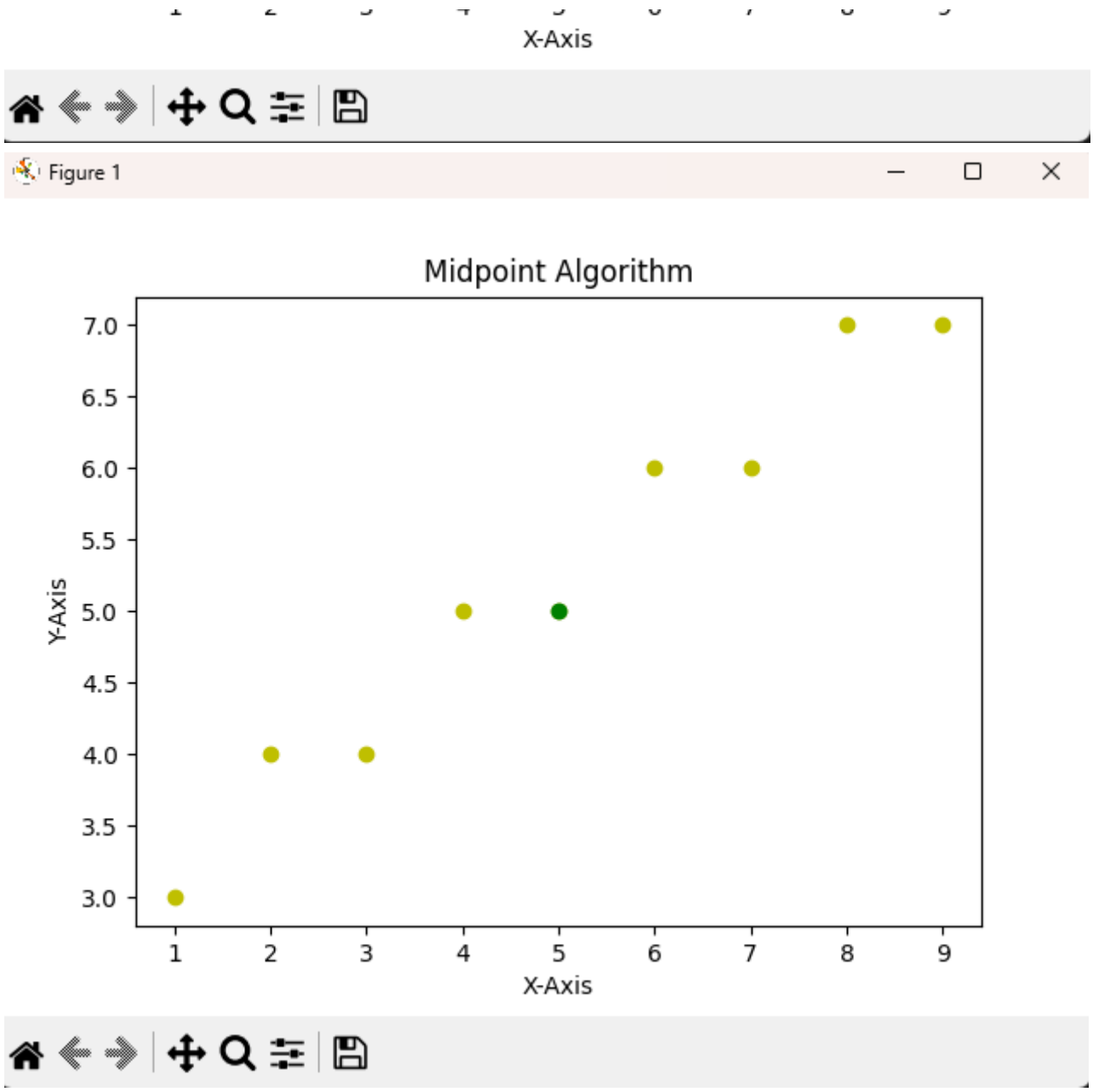


Figure 1
Bresenham's Algorithm
Y-Axis
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
1
2
3
4
5
6
7
8
9


X-Axis
Figure 1
Midpoint Algorithm
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
Y-Axis
1
2
3
4
5
6
7
8
9
X-Axis