Draw 2-D Library

Overview

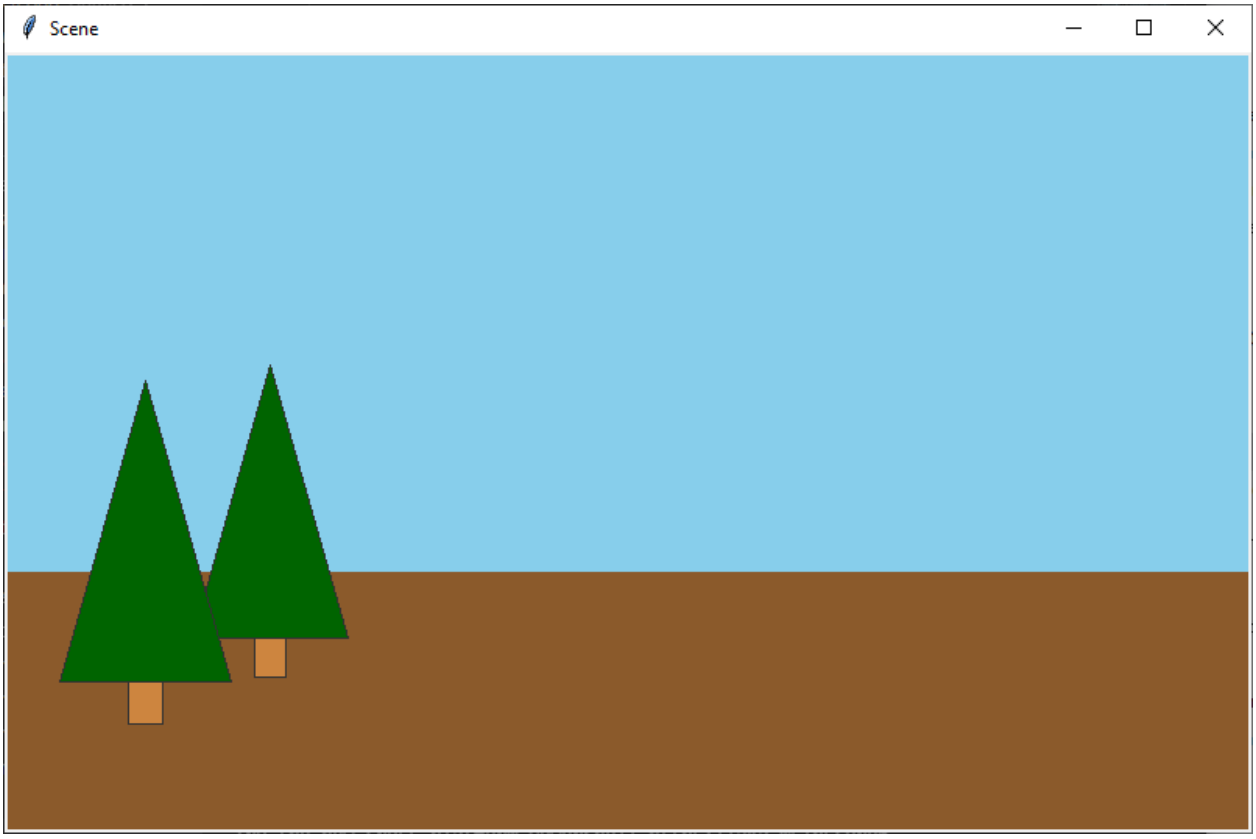
The Draw 2-D library contains functions that draw shapes on a canvas in a computer window. To use the Draw 2-D library, your program must do the following:

1. Import the library's functions
2. Call the `start_drawing` function once. The `start_drawing` function returns a canvas, that your program must pass to each of the `draw_*` functions.
3. Call the `draw_line`, `draw_oval`, `draw_arc`, `draw_rectangle`, `draw_polygon`, and `draw_text` functions as many times as needed. The `draw_*` functions accept Euclidean coordinates that determine where a shape will be drawn on the canvas. The origin of the canvas (0, 0) is in the lower left corner of the canvas.
4. Call the `finish_drawing` function once.

Examples

Outdoor Scene Example

Example program 1 contains all four of the steps needed to use the Draw 2-D library and draws an outdoor scene with two pine trees as shown in this image.



The computer draws the two pine trees because of the following:

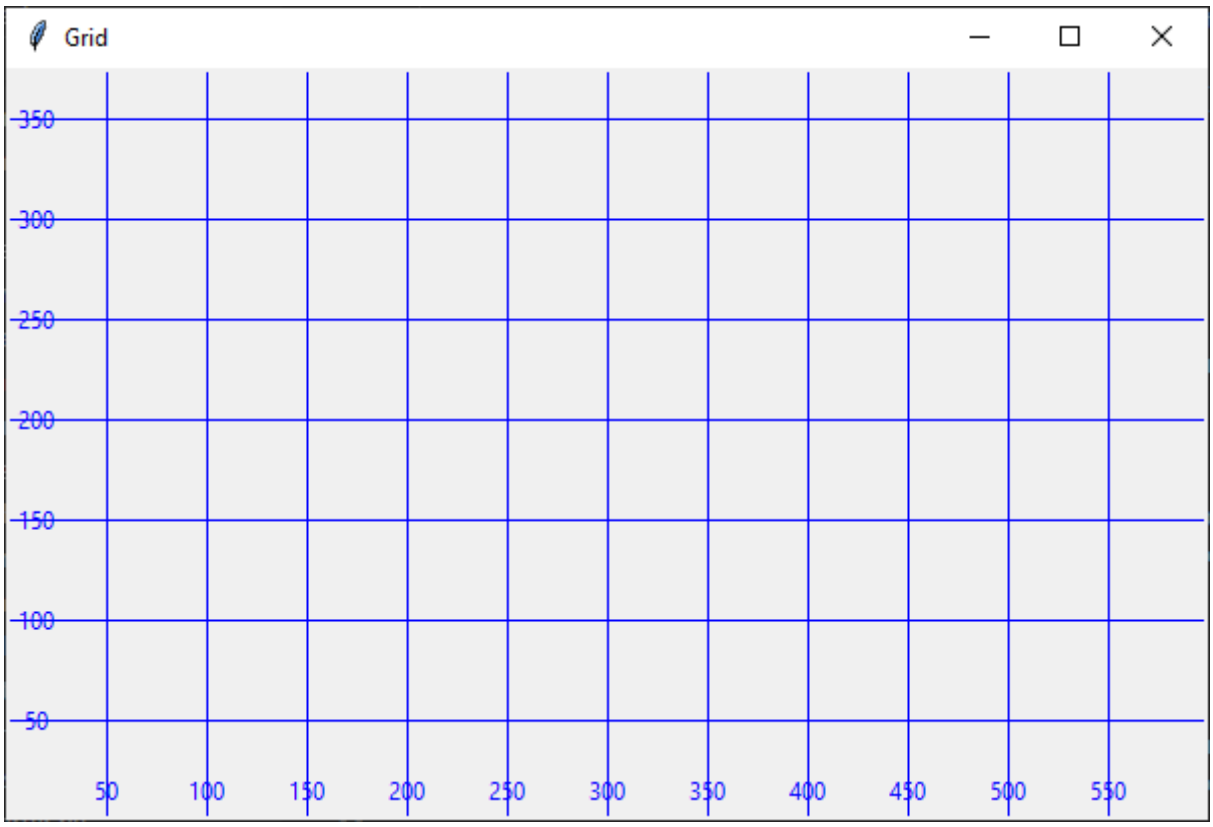
1. The code at line 90 calls the `main` function.
2. At line 20, the `main` function calls the `draw_ground` function.
3. At lines 42 and 49, the `draw_ground` function calls the `draw_pine_tree` function.

```
 1  # Example 1
 2
 3  # Import the functions from the Draw 2-D library
 4  # so that they can be used in this program.
 5  from draw2d import \
 6      start_drawing, draw_line, draw_oval, draw_arc, \
 7      draw_rectangle, draw_polygon, draw_text, finish_drawing
 8
 9
10  def main():
11      scene_width = 800
12      scene_height = 500
13
14      # Call the start_drawing function in the draw2d.py
15      # library which will open a window and create a canvas.
16      canvas = start_drawing("Scene", scene_width, scene_height)
17
18      # Call the draw_sky and draw_ground functions in this file.
19      draw_sky(canvas, scene_width, scene_height)
20      draw_ground(canvas, scene_width, scene_height)
21
22      # Call the finish_drawing function
23      # in the draw2d.py library.
24      finish_drawing(canvas)
25
26
27  def draw_sky(canvas, scene_width, scene_height):
28      """Draw the sky and all the objects in the sky."""
29      draw_rectangle(canvas, 0, scene_height / 3,
30          scene_width, scene_height, width=0, fill="sky blue")
```

```
31
32
33  def draw_ground(canvas, scene_width, scene_height):
34      """Draw the ground and all the objects on the ground."""
35      draw_rectangle(canvas, 0, 0,
36          scene_width, scene_height / 3, width=0, fill="tan4")
37
38      # Draw a pine tree.
39      tree_center_x = 170
40      tree_bottom = 100
41      tree_height = 200
42      draw_pine_tree(canvas, tree_center_x,
43              tree_bottom, tree_height)
44
45      # Draw another pine tree.
46      tree_center_x = 90
47      tree_bottom = 70
48      tree_height = 220
49      draw_pine_tree(canvas, tree_center_x,
50              tree_bottom, tree_height)
51
52
53  def draw_pine_tree(canvas, center_x, bottom, height):
54      """Draw a single pine tree.
55      Parameters
56          canvas: The canvas where this function
57              will draw a pine tree.
58          center_x, bottom: The x and y location in pixels where
59              this function will draw the bottom of a pine tree.
60          height: The height in pixels of the pine tree that
61              this function will draw.
62      Return: nothing
63      """
64      trunk_width = height / 10
65      trunk_height = height / 8
66      trunk_left = center_x - trunk_width / 2
67      trunk_right = center_x + trunk_width / 2
68      trunk_top = bottom + trunk_height
69
70      # Draw the trunk of the pine tree.
71      draw_rectangle(canvas,
72              trunk_left, trunk_top, trunk_right, bottom,
73              outline="gray20", width=1, fill="tan3")
74
75      skirt_width = height / 2
76      skirt_height = height - trunk_height
77      skirt_left = center_x - skirt_width / 2
78      skirt_right = center_x + skirt_width / 2
79      skirt_top = bottom + height
80
81      # Draw the crown (also called skirt) of the pine tree.
82      draw_polygon(canvas, center_x, skirt_top,
83              skirt_right, trunk_top,
84              skirt_left, trunk_top,
85              outline="gray20", width=1, fill="dark green")
86
87
88  # Call the main function so that
89  # this program will start executing.
90  main()
```

Coordinate Grid

Example program 2 draws a grid that can help a programmer determine the correct coordinates for an object in a scene.



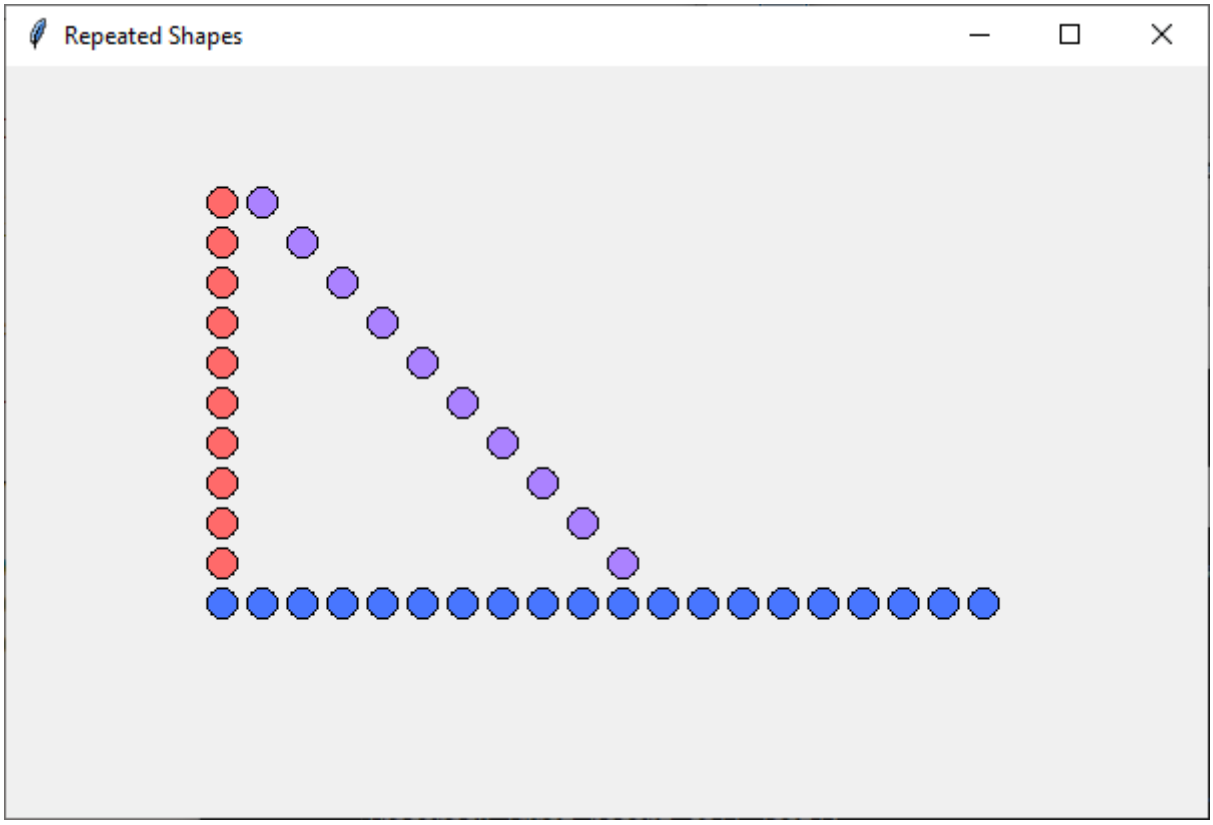
In example 2, the `for` loop at lines 24–26 causes the computer to draw the vertical lines in the grid. The `for` loop at lines 30–32 causes the computer to draw the vertical lines in the grid.

```
1  # Example 2
2
3  from draw2d import \
4      start_drawing, draw_line, draw_text, finish_drawing
5
6  def main():
7      scene_width = 600
8      scene_height = 375
9
10     # Call the start_drawing function in the draw2d.py
11     # library which will open a window and create a canvas.
12     canvas = start_drawing("Grid", scene_width, scene_height)
13
14     draw_grid(canvas, scene_width, scene_height, 50)
15
16     # Call the finish_drawing function
17     # in the draw2d.py library.
18     finish_drawing(canvas)
19
20
21 def draw_grid(canvas, width, height, interval, color="blue"):
22     # Draw a vertical line at every x interval.
23     label_y = 15
24     for x in range(interval, width, interval):
25         draw_line(canvas, x, 0, x, height, fill=color)
26         draw_text(canvas, x, label_y, f"{x}", fill=color)
27
28     # Draw a horizontal line at every y interval.
29     label_x = 15
30     for y in range(interval, height, interval):
31         draw_line(canvas, 0, y, width, y, fill=color)
32         draw_text(canvas, label_x, y, f"{y}", fill=color)
33
34
35 # Call the main function so that
```

```
36  # this program will start executing.
37  main()
```

Row, Column, and Diagonal of Shapes

The program in example 3 uses three `for` loops and the `draw_oval` function to draw circles in a row, a column, and a diagonal.



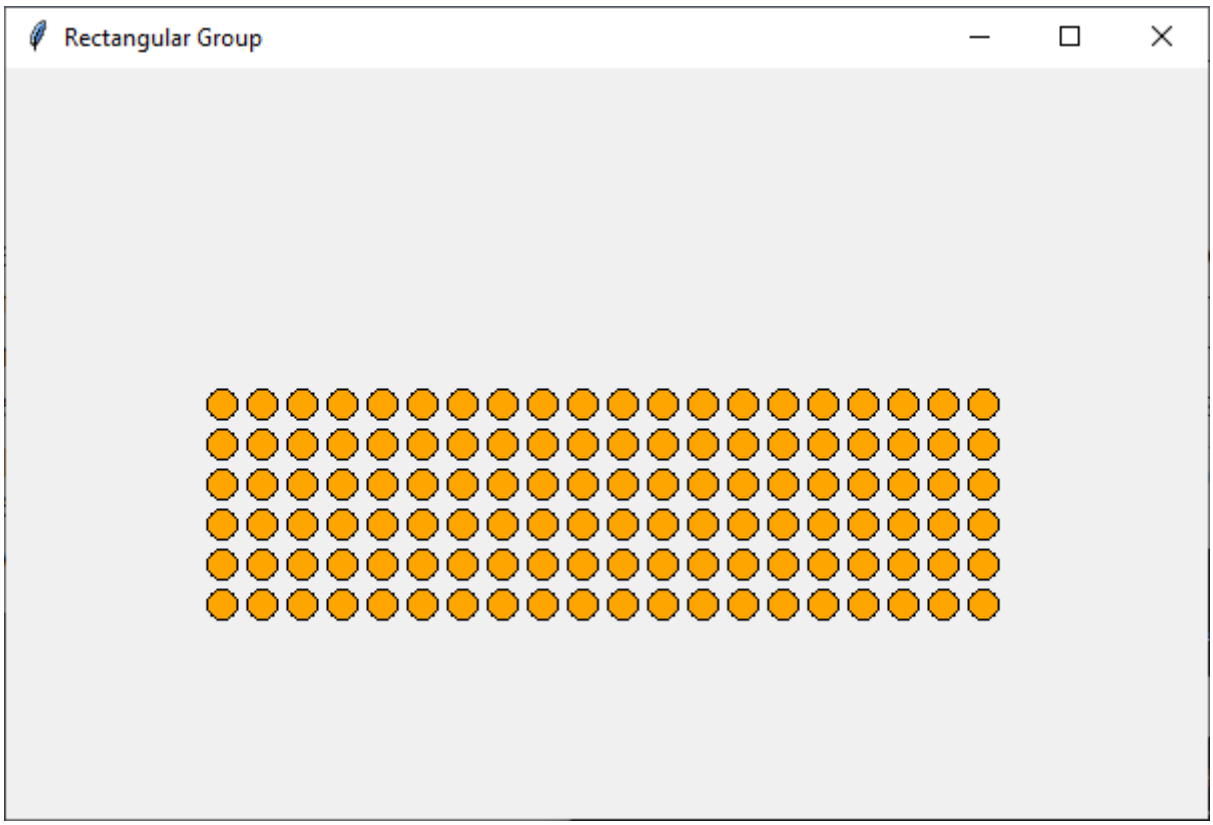
Comparing the three `for` loops at lines 17–20, 25–28, and 33–37 will help you understand how you can use a `for` loop to draw many shapes. Of course, your program can call any draw function within a `for` loop and draw any shape you like.

```
 1  # Example 3
 2
 3  from draw2d import start_drawing, draw_oval, finish_drawing
 4
 5  def main():
 6      # Call the start_drawing function in the draw2d.py
 7      # library which will open a window and create a canvas.
 8      canvas = start_drawing("Repeated Shapes", 600, 375)
 9
10      diameter = 15
11      space = 5
12      interval = diameter + space
13
14      # Draw a row of 20 circles.
15      x = 100
16      y = 100
17      for i in range(20):
18          draw_oval(canvas, x, y, x + diameter, y + diameter,
19                    fill="royalBlue1")
20          x += interval
21
22      # Draw a column of 10 circles.
```

```
23      x = 100
24      y = 120
25      for i in range(10):
26          draw_oval(canvas, x, y, x + diameter, y + diameter,
27                  fill="indianRed1")
28          y += interval
29
30      # Draw a diagonal of 10 circles.
31      x = 120
32      y = 300
33      for i in range(10):
34          draw_oval(canvas, x, y, x + diameter, y + diameter,
35                  fill="mediumPurple1")
36          x += interval
37          y -= interval
38
39      # Call the finish_drawing function
40      # in the draw2d.py library.
41      finish_drawing(canvas)
42
43
44  # Call the main function so that
45  # this program will start executing.
46  main()
```

Rectangular Group of Shapes

The program in example 4 uses two `for` loops and the `draw_oval` function to draw a rectangular group of circles.



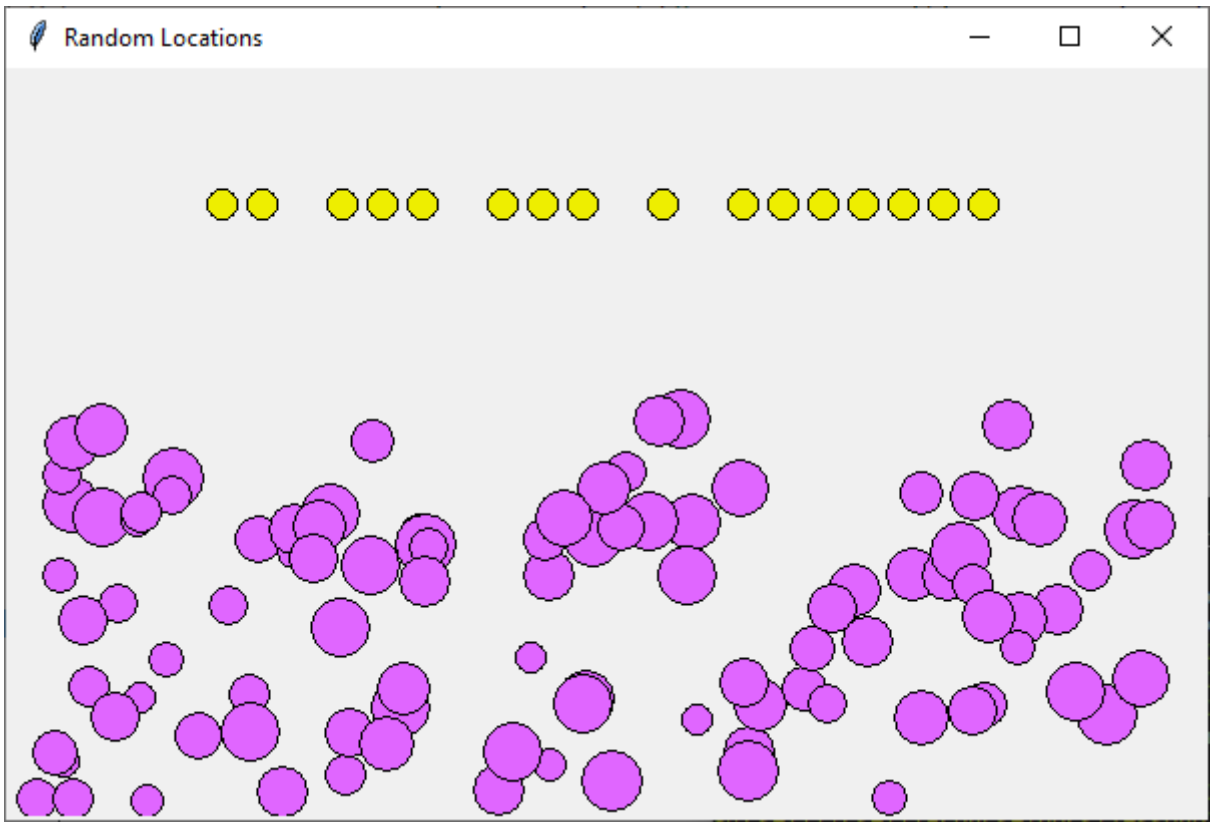
The first `for` loop starts at line 16 The second `for` loop is nested inside the first at lines 18–21. The computer executes the first `for` loop once for each row in the group. The computer executes the second `for` loop once for each cell in the group. Because the loops are nested, the computer executes the statements inside the second loop at lines 19–21 120 times (6 × 20). In other words, the

program in example 4 will draw six rows with 20 circles in each row, which makes 120 circles.

```
# Example 4

from draw2d import start_drawing, draw_oval, finish_drawing

def main():
    # Call the start_drawing function in the draw2d.py
    # library which will open a window and create a canvas.
    canvas = start_drawing("Rectangular Group", 600, 375)

    diameter = 15
    space = 5
    interval = diameter + space

    # Draw a rectangular series of circles.
    y = 100
    for row in range(6):
        x = 100
        for cell in range(20):
            draw_oval(canvas, x, y,
                      x + diameter, y + diameter, fill="orange")
            x += interval
        y += interval

    # Call the finish_drawing function
    # in the draw2d.py library.
    finish_drawing(canvas)


# Call the main function so that
# this program will start executing.
main()
```

Randomly Located Shapes

The program in example 5 draws a row of circles but some of the circles are missing because the computer randomly chose not to draw some of the circles. It also draws 100 circles, each with a random diameter at a randomly chosen location.



In example 5 at line 24, the call to the `random.randint` function causes the computer to calculate a random number between 1 and 5 inclusive. The `if` statement at line 25 allows the computer to draw a circle if the random number is greater than 1. In other words, if the random number is 1, the computer will not draw a circle, which means about 20% (1 / 5) of the circles in the row will not be drawn.

In example 5, the statements at lines 36–41, cause the computer to draw 100 circles, each with a random location and diameter. At lines 37–39, the computer calculates a random location (*x* and *y*) and a random diameter. Then at lines 40–41 the computer uses those random numbers to draw a circle.

```
 1  # Example 5
 2
 3  from draw2d import start_drawing, draw_oval, finish_drawing
 4  import random
 5
 6  def main():
 7      scene_width = 600
 8      scene_height = 375
 9
10      # Call the start_drawing function in the draw2d.py
11      # library which will open a window and create a canvas.
12      canvas = start_drawing("Random Locations",
13              scene_width, scene_height)
14
15      diameter = 15
16      space = 5
17      interval = diameter + space
18
19      # Draw a row of circles with
20      # some of the circles missing.
21      x = 100
22      y = 300
23      for i in range(20):
24          number = random.randint(1, 5)
```

```
        if number > 1:
            draw_oval(canvas, x, y,
                    x + diameter, y + diameter, fill="yellow2")
        x += interval

    half_height = round(scene_height / 2)
    min_diam = 15
    max_diam = 30

    # Draw 100 circles, each with
    # a random location and diameter.
    for i in range(100):
        x = random.randint(0, scene_width - max_diam)
        y = random.randint(0, half_height)
        diameter = random.randint(min_diam, max_diam)
        draw_oval(canvas, x, y, x + diameter, y + diameter,
                fill="mediumOrchid1")

    # Call the finish_drawing function
    # in the draw2d.py library.
    finish_drawing(canvas)


# Call the main function so that
# this program will start executing.
main()
```

Function Reference

The following functions are contained in the Draw 2-D library.

`start_drawing(title, width, height)`

Create a window with a canvas where a program can draw 2-dimensional shapes.

Parameters

- **title**: the title that will appear in the window's title bar
- **width**: the width in pixels of the canvas
- **height**: the height in pixels of the canvas

Return

- the new canvas

Example

```
canvas = start_drawing("Scene", 800, 500)
```

`draw_line(canvas, x0, y0, x1, y1, … xn, yn, width=1, fill="black")`

Draw a that line goes through the series of points (x0, y0), (x1, y1), ... (xn, yn)

Parameters

- **canvas**: the canvas returned from the `start_drawing` function
- **x0, y0, x1, y1, ... xn, yn**: the coordinates for the segments of a line that this function will draw
- **width**: the line's width. The default is 1 pixel.
- **fill**: the line's color. The default is black.

Return

- nothing

Example

```
draw_line(canvas, 0, 5, 100, 50, width=10, fill="blue")
```

```
draw_oval(canvas, x0, y0, x1, y1, width=1, outline="black", fill="")
```

Draw an oval (ellipse) inside the bounding box defined by the coordinates (x0, y0), (x1, y1)

Parameters

- **canvas**: the canvas returned from the `start_drawing` function
- **x0, y0, x1, y1**: the coordinates of a bounding box for the oval that this function will draw
- **width**: the width of the oval's outline. The default is 1 pixel. A width of 0 will draw an oval without an outline.
- **outline**: the color of the oval's outline. The default is black.
- **fill**: the color of the oval's interior. The default is "" which means transparent.

Return

- nothing

Example

```
draw_oval(canvas, 100, 100, 300, 200, fill="pink")
```

```
draw_arc(canvas, x0, y0, x1, y1, start=0, extent=90,
        width=1, outline="black", fill="")
```

Draw a wedge shaped slice taken from an oval (ellipse) defined by the bounding box with coordinates (x0, y0), (x1, y1).

Parameters

- **canvas**: the canvas returned from the `start_drawing` function
- **x0, y0, x1, y1**: the bounding box for the oval from which this function will draw a wedge
- **width**: the width of the oval's outline; default is 1 pixel

- **outline**: the color of the oval's outline; default is black
- **fill**: the color of the oval's interior; default is "" which means transparent

Return

- nothing

Example

```
draw_arc(canvas, 100, 100, 300, 200, start=180, extent=270)
```

```
draw_rectangle(canvas, x0, y0, x1, y1,
        width=1, outline="black", fill="")
```

Draw a rectangle with two of its corners at (x0, y0), (x1, y1).

Parameters

- **canvas**: the canvas returned from the `start_drawing` function
- **x0, y0, x1, y1**: the bounding box for the rectangle that this function will draw
- **width**: the width of the rectangle's outline; default is 1 pixel
- **outline**: the color of the rectangle's outline; default is black
- **fill**: the color of the rectangle's interior; default is "" which means transparent

Return

- nothing

Example

```
draw_rectangle(canvas, 100, 100, 300, 200, width=5)
```

```
draw_polygon(canvas, x0, y0, x1, y1, x2, y2, … xn, yn,
        width=1, outline="black", fill="")
```

Draw a polygon with vertices (x0, y0), (x1, y1), ... (xn, yn). The polygon is always a closed polygon with the same quantity of segments as vertices. In other words, the segments are defined as follows: (x0, y0) -> (x1, y1) -> ... -> (xn, yn) -> (x0, y0)

Parameters

- **canvas**: the canvas returned from the `start_drawing` function
- **x0, y0, x1, y1, x2, y2, ... xn, yn**: the coordinates for the vertices of the polygon that this function will draw
- **width**: the width of the polygon's outline; default is 1 pixel
- **outline**: the color of the polygon's outline; default is black
- **fill**: the color of the polygon's interior; default is "" which means transparent

Return

- nothing

Example

```
draw_polygon(canvas,
        100, 100, 200, 100, 200, 150, fill="green")
```

`draw_text(canvas, center_x, center_y, text, fill="black")`

Draw text with the center of the text at (center_x, center_y).

Parameters

- **canvas**: the canvas returned from the `start_drawing` function
- **center_x, center_y**: the coordinates for the center point where this function will draw text
- **text**: the text to be drawn. To force a line break, include a newline character ("\n").

Return

- nothing

Example

```
draw_text(canvas, 250, 200, "Olá", fill="purple")
```

`finish_drawing(canvas)`

Call all functions that are necessary to display the drawing on the computer's monitor.

Parameters

- **canvas**: the canvas returned from the `start_drawing` function

Return

- nothing

Example

```
finish_drawing(canvas)
```

Colors

The Draw 2-D library supports the following named colors.

lightCoral	chocolate1	orange2	chartreuse	turquoise4	dodgerBlue1	violet	snow1	gray51
indianRed1	chocolate2	orange3	chartreuse2	darkSlateGray1	dodgerBlue2	magenta	snow2	gray50
indianRed2	chocolate	orange4	chartreuse3	darkSlateGray2	dodgerBlue3	magenta2	snow3	gray49
indianRed	chocolate3	darkOrange	chartreuse4	darkSlateGray3	dodgerBlue4	magenta3	white	gray48
indianRed3	saddleBrown	darkOrange1	mintCream	darkSlateGray4	cornflowerBlue	magenta4	gray99	gray47
indianRed4	seashell	darkOrange2	honeydew1	darkSlateGray	royalBlue	orchid1	gray98	gray46
firebrick1	seashell2	darkOrange3	honeydew2	lightCyan1	royalBlue1	orchid2	gray97	gray45
firebrick2	seashell3	darkOrange4	honeydew3	lightCyan2	royalBlue2	orchid	gray96	gray44
firebrick3	seashell4	paleGoldenrod	honeydew4	lightCyan3	royalBlue3	orchid3	gray95	gray43
firebrick	peachPuff	goldenrod1	seaGreen1	lightCyan4	royalBlue4	orchid4	gray94	gray42
firebrick4	peachPuff2	goldenrod2	seaGreen2	cyan1	blue1	maroon1	gray93	gray41
red1	peachPuff3	goldenrod	seaGreen3	cyan2	blue2	maroon2	gray92	gray40
red2	peachPuff4	goldenrod3	mediumSeaGreen	cyan3	blue3	maroon3	gray91	gray39
red3	tan	goldenrod4	seaGreen	cyan4	blue4	maroon	gray90	gray38
red4	tan1	darkGoldenrod1	darkSeaGreen1	cadetBlue1	navyBlue	maroon4	gray89	gray37
rosyBrown1	sandyBrown	darkGoldenrod2	darkSeaGreen2	cadetBlue2	midnightBlue	deepPink1	gray88	gray36
rosyBrown2	tan2	darkGoldenrod3	darkSeaGreen3	cadetBlue3	lavender	deepPink2	gray87	gray35
rosyBrown3	tan3	darkGoldenrod	darkSeaGreen	cadetBlue	slateBlue1	deepPink3	gray86	gray34
rosyBrown	tan4	darkGoldenrod4	darkSeaGreen4	cadetBlue4	slateBlue2	deepPink4	gray85	gray33
rosyBrown4	floralWhite	gold1	paleGreen1	ghostWhite	slateBlue3	hotPink1	gray84	gray32
brown1	linen	gold2	paleGreen	aliceBlue	slateBlue4	hotPink	gray83	gray31
brown2	oldLace	gold3	paleGreen2	powderBlue	mediumPurple1	hotPink2	gray82	gray30
brown3	antiqueWhite	gold4	paleGreen3	lightBlue1	mediumPurple2	hotPink3	gray81	gray29
brown	antiqueWhite1	lemonChiffon1	paleGreen4	lightBlue	mediumPurple	hotPink4	gray80	gray28
brown4	antiqueWhite2	lemonChiffon2	forestGreen	lightBlue2	mediumPurple3	paleVioletRed1	gray79	gray27
tomato1	antiqueWhite3	lemonChiffon3	green1	lightBlue3	mediumPurple4	paleVioletRed	gray78	gray26
tomato2	antiqueWhite4	lemonChiffon4	green2	lightBlue4	purple1	paleVioletRed2	gray77	gray25
tomato3	cornsilk	khaki1	green3	deepSkyBlue	purple2	paleVioletRed3	gray76	gray24
tomato4	cornsilk2	khaki	green4	deepSkyBlue2	purple3	paleVioletRed4	gray75	gray23
coral	cornsilk3	khaki2	green	deepSkyBlue3	purple4	violetRed1	gray74	gray22
coral1	cornsilk4	khaki3	darkGreen	deepSkyBlue4	purple	violetRed2	gray73	gray21
coral2	bisque1	darkKhaki	springGreen1	lightSkyBlue1	blueViolet	violetRed	gray72	gray20
coral3	bisque2	khaki4	springGreen2	lightSkyBlue2	darkViolet	violetRed3	gray71	gray19
coral4	bisque3	ivory1	springGreen3	lightSkyBlue3	mediumOrchid1	violetRed4	gray70	gray18
salmon	bisque4	ivory2	springGreen4	lightSkyBlue4	mediumOrchid2	lavenderBlush1	gray69	gray17
salmon1	burlywood1	ivory3	aquamarine1	skyBlue	mediumOrchid	lavenderBlush2	gray68	gray16
salmon2	burlywood2	ivory4	aquamarine2	skyBlue1	mediumOrchid3	lavenderBlush3	gray67	gray15
salmon3	burlywood3	beige	aquamarine3	lightSkyBlue	mediumOrchid4	lavenderBlush4	gray66	gray14
salmon4	burlywood4	lightYellow1	aquamarine4	skyBlue2	darkOrchid1	pink	gray65	gray13
lightSalmon1	burlywood	lightYellow2	azure1	skyBlue3	darkOrchid2	pink1	gray64	gray12
lightSalmon2	blanchedAlmond	lightYellow3	azure2	skyBlue4	darkOrchid3	pink2	gray63	gray11
darkSalmon	papayaWhip	yellow1	azure3	lightSteelBlue	darkOrchid4	pink3	gray62	gray10
lightSalmon3	moccasin	yellow2	azure4	steelBlue1	thistle1	pink4	gray61	gray9
lightSalmon4	navajoWhite	yellow3	paleTurquoise1	steelBlue2	thistle2	lightPink	gray60	gray8
orangeRed1	navajoWhite2	yellow4	paleTurquoise2	steelBlue3	thistle	lightPink1	gray59	gray7
orangeRed2	navajoWhite3	oliveDrab1	paleTurquoise3	steelBlue	thistle3	lightPink2	gray58	gray6
orangeRed3	navajoWhite4	oliveDrab2	paleTurquoise4	steelBlue4	thistle4	lightPink3	gray57	gray5
orangeRed4	wheat	oliveDrab3	lightSeaGreen	slateGray1	plum1	lightPink4	gray56	gray4
sienna1	wheat1	oliveDrab4	turquoise	slateGray2	plum2	mistyRose1	gray55	gray3
sienna2	wheat2	darkOliveGreen	turquoise1	slateGray3	plum	mistyRose2	gray54	gray2
sienna3	wheat3	greenYellow	turquoise2	lightSlateGray	plum3	mistyRose3	gray53	gray1
sienna	wheat4	lawnGreen	darkTurquoise	slateGray	plum4	mistyRose4	gray52	black
sienna4	orange	limeGreen	turquoise3	slateGray4				