Exposure CS 2021 for CS1

Chapter 6 Section 4-8 Slides

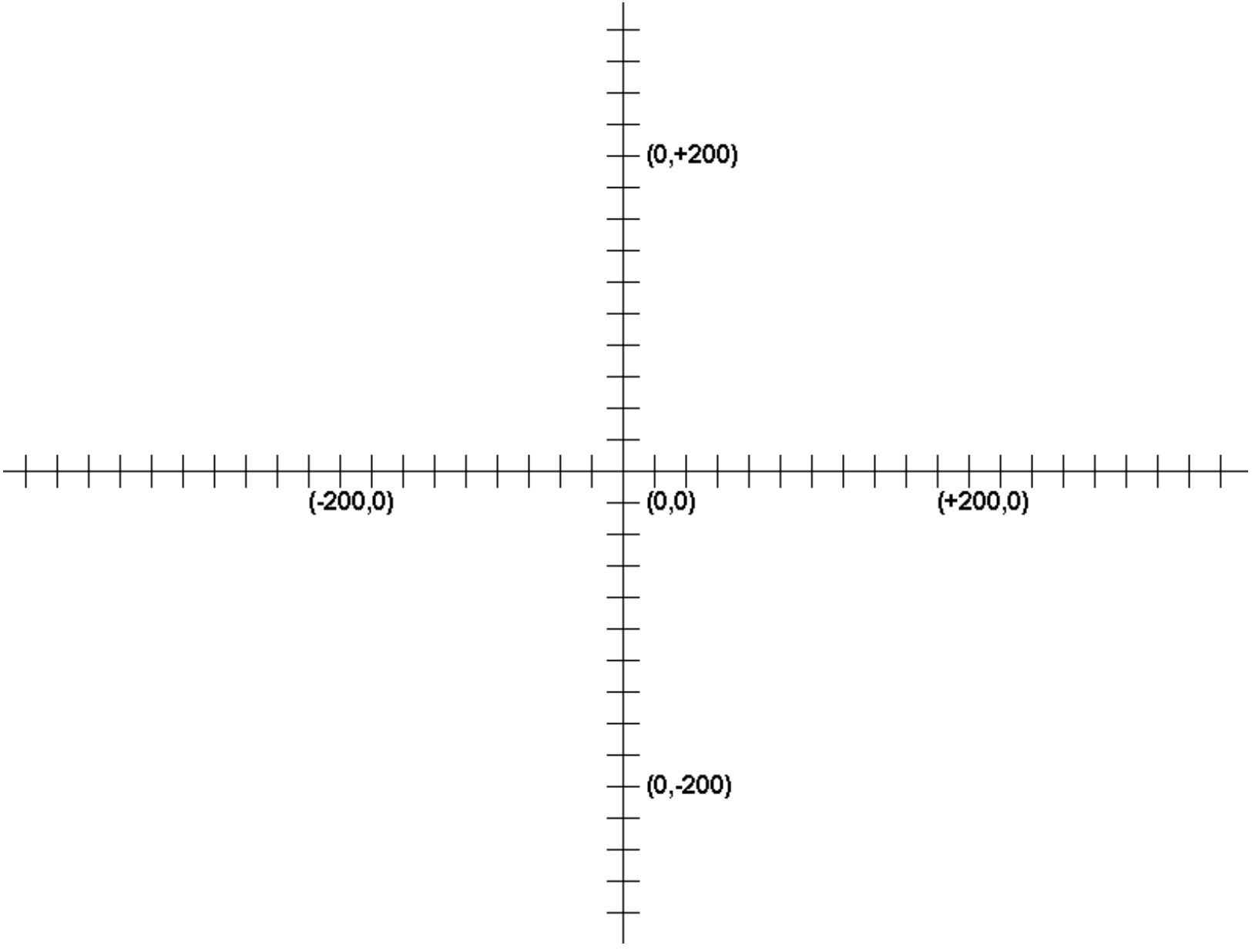
More Python Libraries: Graphics Library Procedures

PowerPoint Presentation
created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science

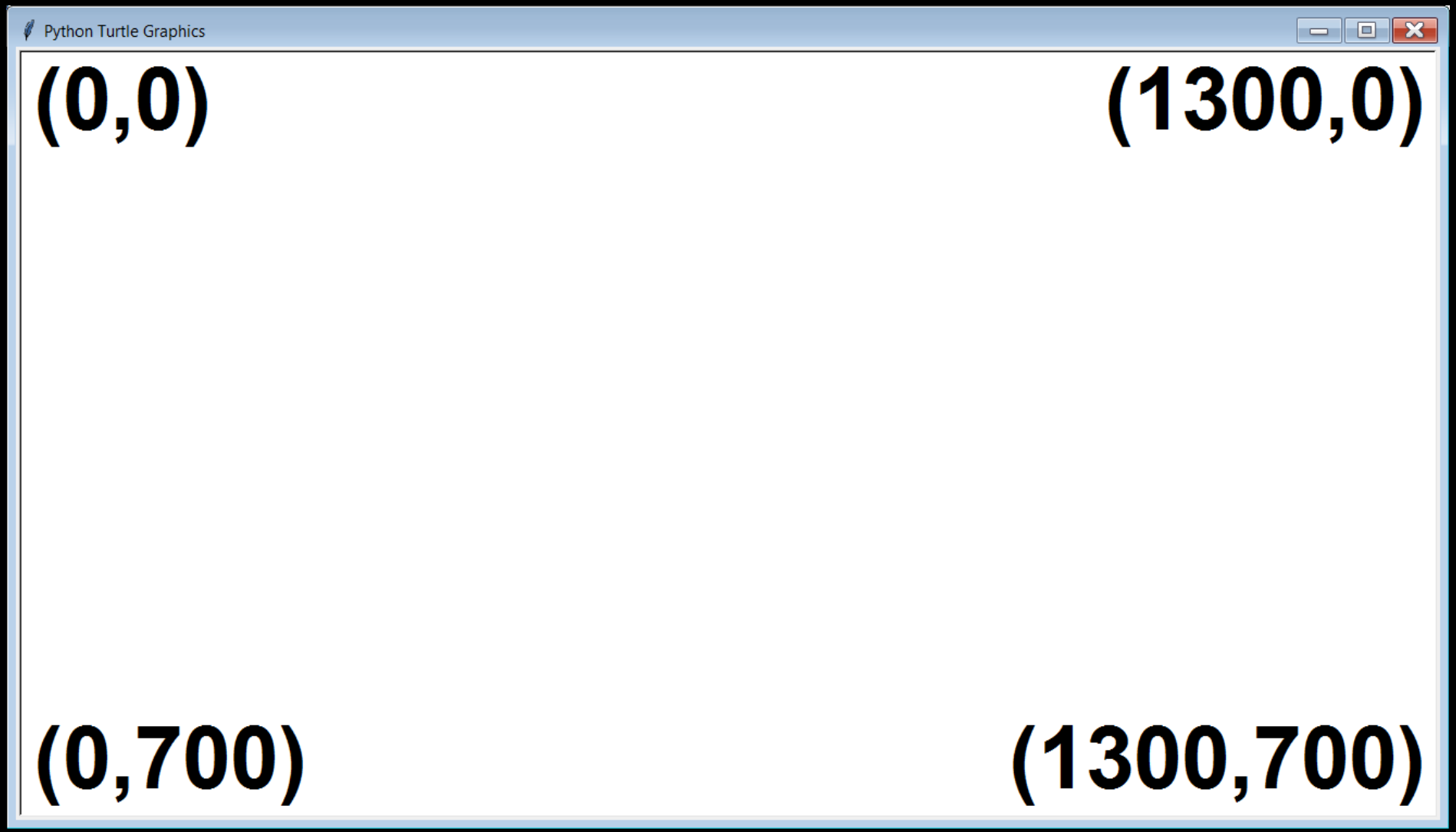
Section 6.4

Introduction to Graphics without the Turtle

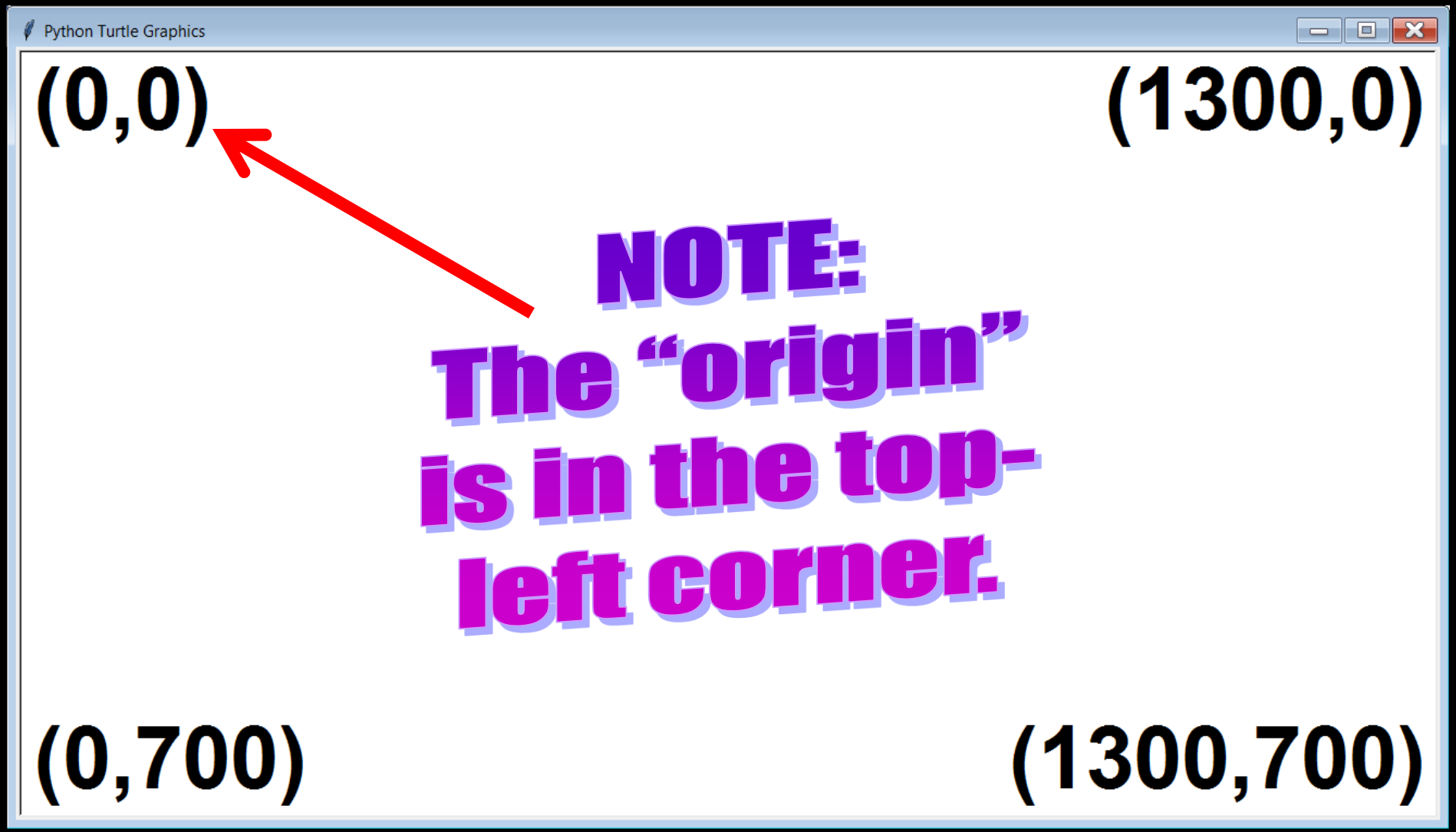
Cartesian Coordinate Grid



Typical Computer Graphics Window



Typical Computer Graphics Window



Python "Traditional Graphics" Disclaimer

There actually is an interface called tkinter that can be used with Python to create graphics output with the traditional coordinate-based system, but that requires some knowledge of *Object Oriented Programming*, which is a topic that is beyond the scope of this first year Computer Science class.

"OOP" is actually a major component of AP® Computer Science-A, which some of you might be taking next year.

Learning Graphics Programming in IntroCS

Learning graphics programming is not simply a fun issue.

Unlike a text-based program, graphics programs are visual.

When you make changes to your program, you see the effects of those changes instantly.

You can learn many sophisticated computer science concepts by manipulating graphics programs.

In order to write programs using traditional, *coordinate-based* graphics, without using *Object Oriented Programming*, we will import Mr. Schram's **Graphics** library.

For now, you will be using subroutines from libraries that have been created by others. In Chapter 9, you will learn how to create your own subroutines and libraries.

```
 1 # GraphicsLibrary01.py
 2 # This program demonstrates the <drawPixel> and
 3 # <drawPoint> procedures of the <Graphics> library.
 4 # Both procedures draw a dot on the computer screen.
 5
 6 # NOTE: The <Graphics> library was created by Mr. Schram
 7 #       and is not part of standard Python.
 8
 9 # NOTE: Most of the procedures in the <Graphics> library
10 #       use integer arguments.
11
12
13 # Required to have access to the
14 # <Graphics> library commands.
15 from Graphics import *
16
17 # Opens a graphics window
18 # with specified dimensions
19 beginGrfx(1300,700)
20
```

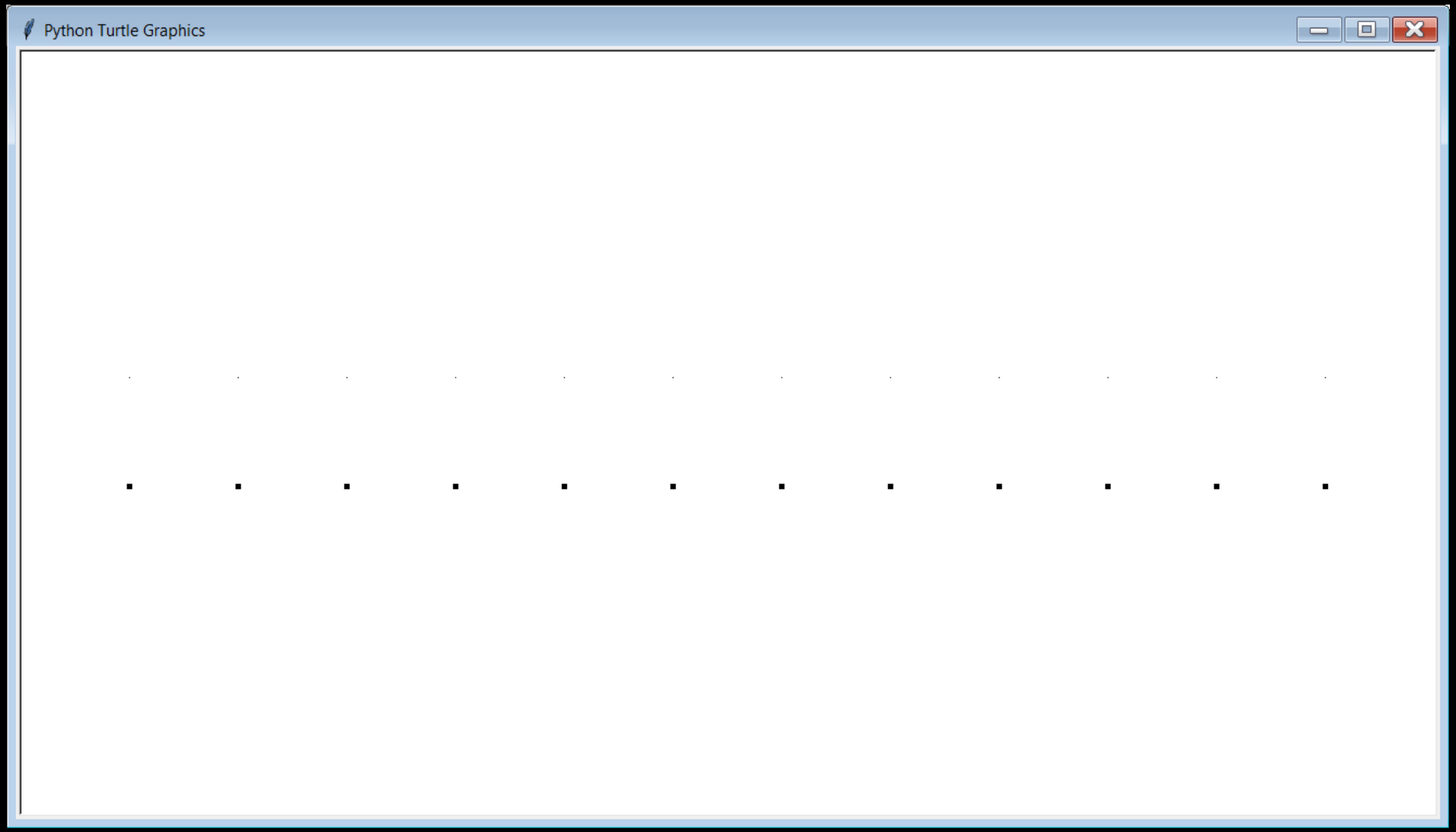
```
21 # Draws tiny individual pixels
22 drawPixel(100,300)
23 drawPixel(200,300)
24 drawPixel(300,300)
25 drawPixel(400,300)
26 drawPixel(500,300)
27 drawPixel(600,300)
28 drawPixel(700,300)
29 drawPixel(800,300)
30 drawPixel(900,300)
31 drawPixel(1000,300)
32 drawPixel(1100,300)
33 drawPixel(1200,300)
34
```

```
35 # Draws small squares
36 drawPoint(100,400)
37 drawPoint(200,400)
38 drawPoint(300,400)
39 drawPoint(400,400)
40 drawPoint(500,400)
41 drawPoint(600,400)
42 drawPoint(700,400)
43 drawPoint(800,400)
44 drawPoint(900,400)
45 drawPoint(1000,400)
46 drawPoint(1100,400)
47 drawPoint(1200,400)
48
```

```
49 # Updates the screen and keeps the graphics
50 # window open when the program is finished.
51 endGrfx()
52
```

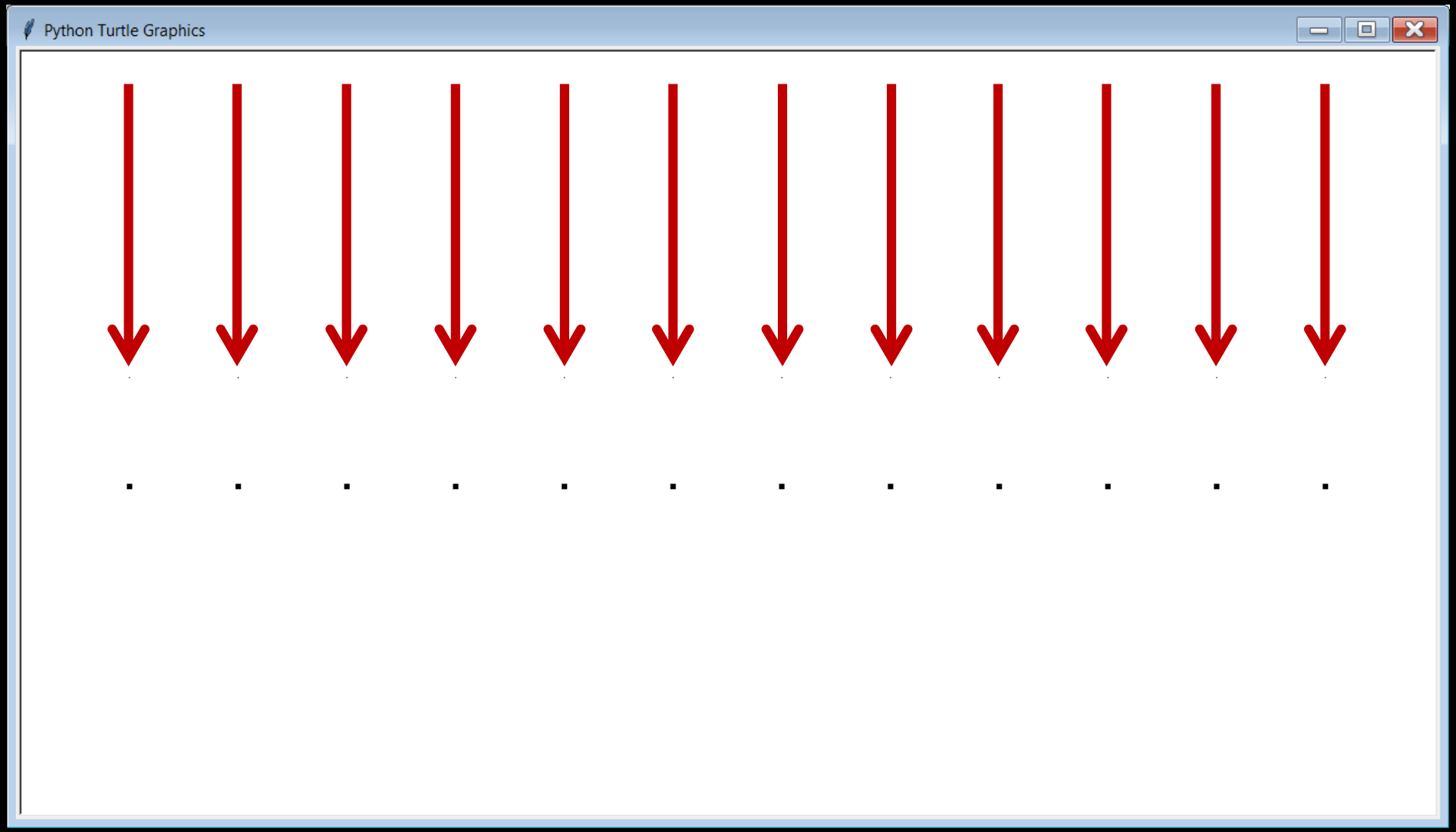
GraphicsLibrary01.py Output

The pixels may be difficult to see.



GraphicsLibrary01.py Output

The pixels may be difficult to see.



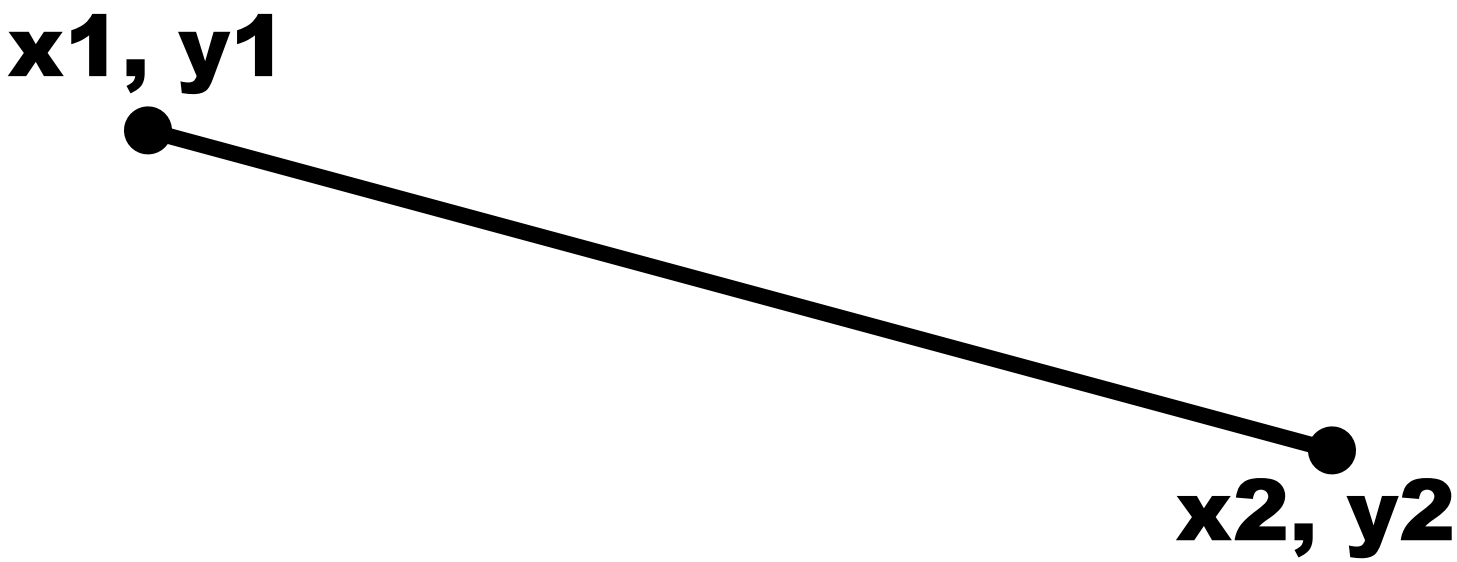
Section 6.5

Drawing Simple Shapes

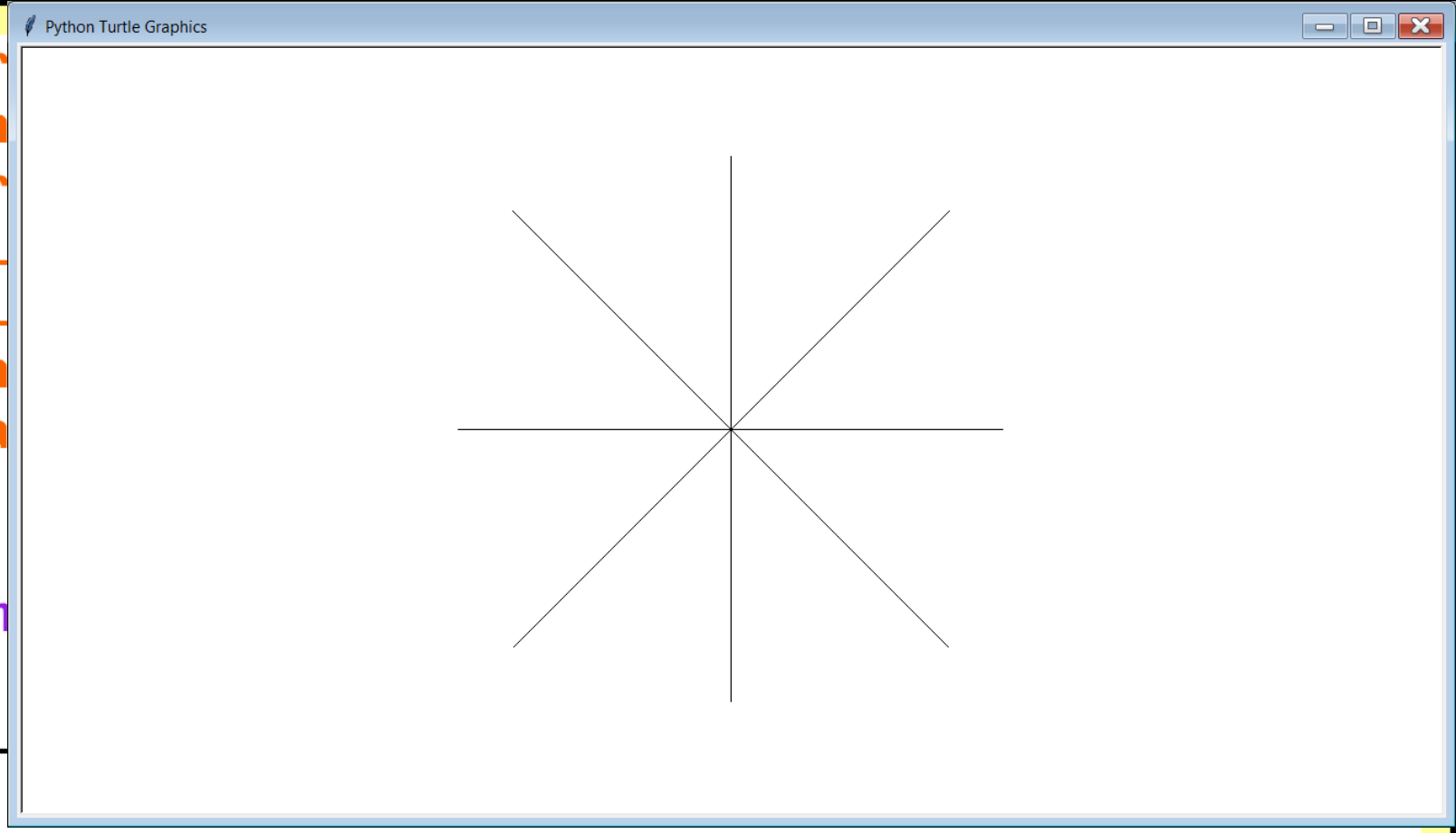
Procedure drawLine

```
drawLine(x1, y1, x2, y2)
```

Draws a line from coordinate (x1,y1) to coordinate (x2,y2)



```
 1 # GraphicsLibrary02.py
 2 # This program demonstrates the <drawLine>
 3 # procedure of the <Graphics> library.
 4 # Lines are drawn from (x1,y1) to (x2,y2)
 5 # with <drawLine(x1,y1,x2,y2)>.
 6 # This program displays a snowflake by
 7 # calling <drawLine> 4 times.
 8
 9
10 from Graphics import *
11
12 beginGrfx(1300,700)
13
14 drawLine(650,100,650,600)
15 drawLine(400,350,900,350)
16 drawLine(450,150,850,550)
17 drawLine(850,150,450,550)
18
19 endGrfx()
```

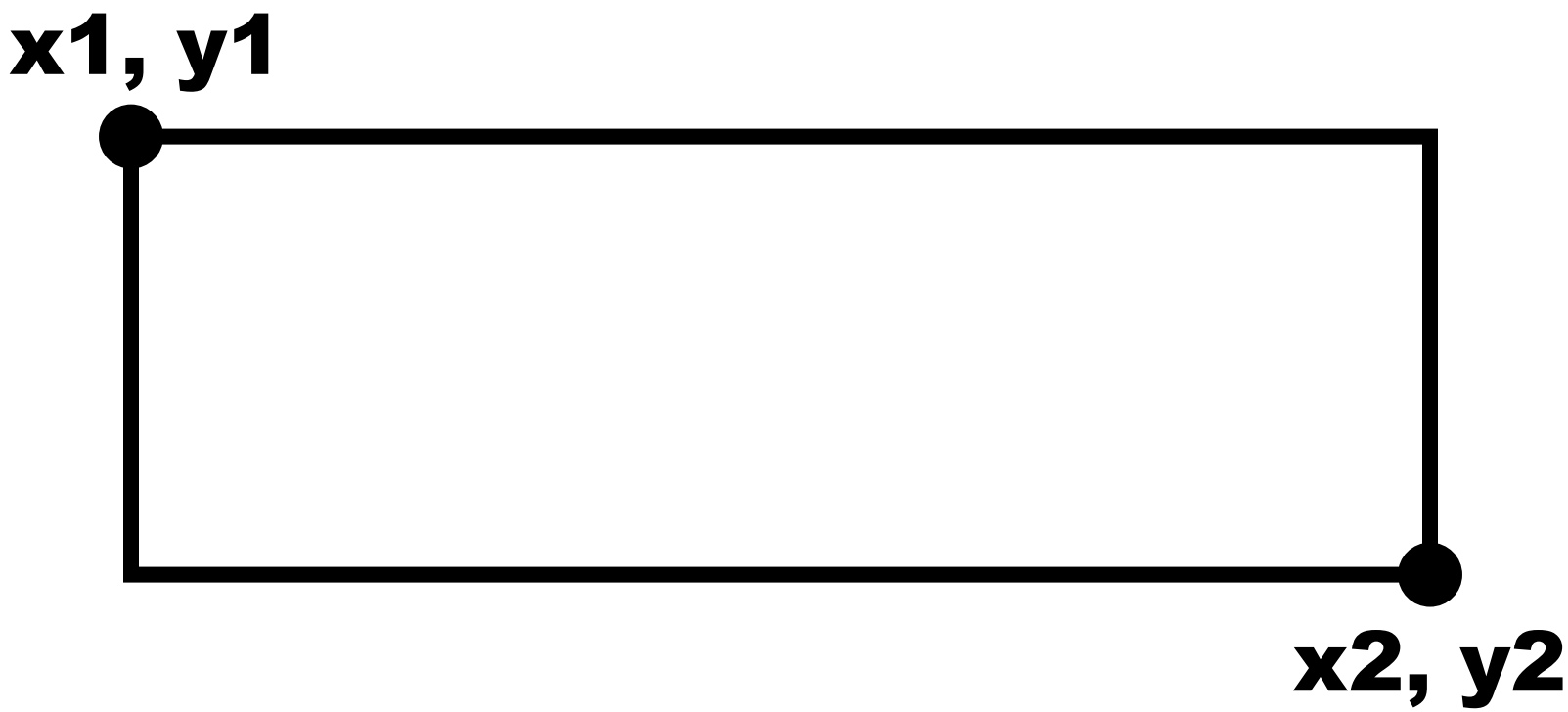



```
1  # Gr
2  # Th
3  # pr
4  # Li
5  # wi
6  # Th
7  # ca
8
9
10 from
11
12 begi
13
14 drawLine(650,100,650,600)
15 drawLine(400,350,900,350)
16 drawLine(450,150,850,550)
17 drawLine(850,150,450,550)
18
19 endGrfx()
```

Procedure drawRectangle

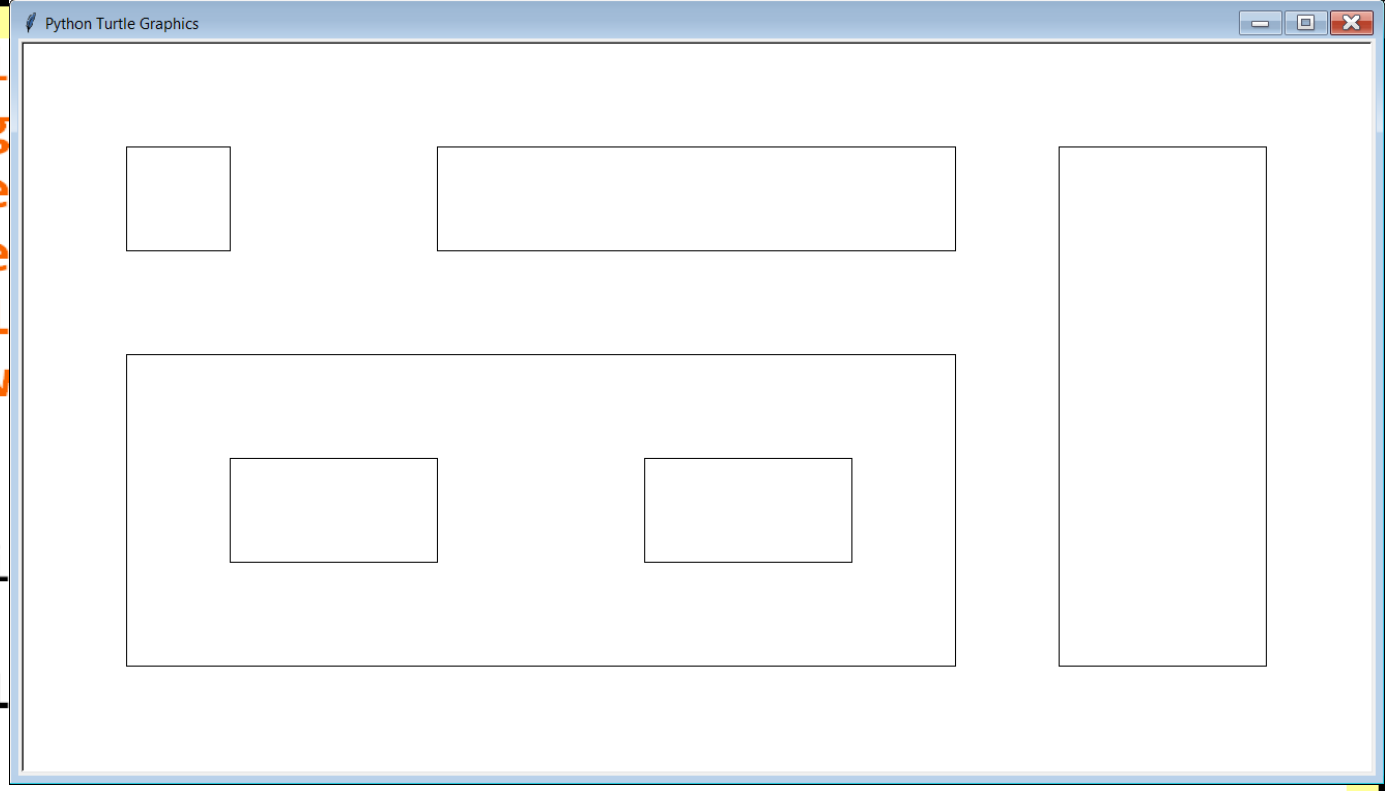
`drawRectangle(x1, y1, x2, y2)`

Draws a rectangle with a top-left corner at coordinate (x1,y1) and a bottom-right hand corner of (x2,y2).



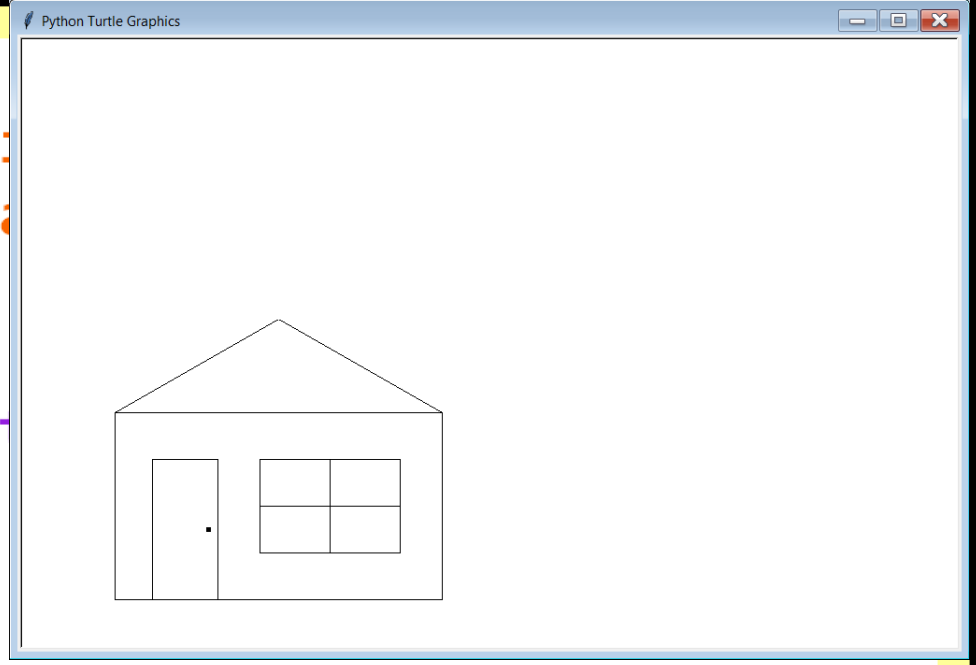
```
 1 # GraphicsLibrary03.py
 2 # This program demonstrates the <drawRectangle>
 3 # procedure of the <Graphics> library.
 4 # Rectangles are drawn from the upper-left-hand
 5 # corner(x1,y1) to the lower-right-hand corner
 6 # (x2,x2) with <drawRectangle(x1,y1,x2,y2)>.
 7
 8
 9 from Graphics import *
10
11 beginGrfx(1300,700)
12
13 drawRectangle(100,100,200,200)
14 drawRectangle(400,100,900,200)
15 drawRectangle(100,300,900,600)
16 drawRectangle(1000,100,1200,600)
17 drawRectangle(200,400,400,500)
18 drawRectangle(600,400,800,500)
19
20 endGrfx()
```

```
1  # GraphicsLi
2  # This prog
3  # procedure
4  # Rectangle
5  # corner(x1
6  # (x2,x2) w
7
8
9  from Graphi
10
11 beginGrfx(1
12
13 drawRectangle(100,100,200,200)
14 drawRectangle(400,100,900,200)
15 drawRectangle(100,300,900,600)
16 drawRectangle(1000,100,1200,600)
17 drawRectangle(200,400,400,500)
18 drawRectangle(600,400,800,500)
19
20 endGrfx()
```



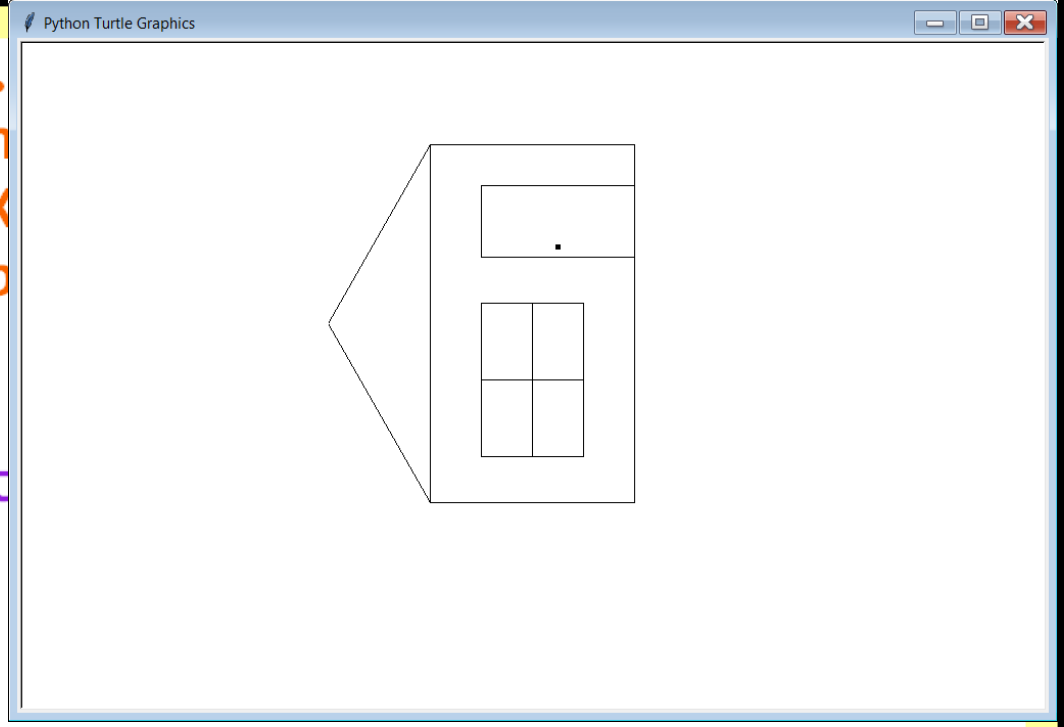
```
 1 # GraphicsLibrary04.py
 2 # This program combines lines, rectangles
 3 # and a point to draw a simple house.
 4
 5
 6 from Graphics import *
 7
 8 beginGrfx(1000,650)
 9
10 drawRectangle(100,400,450,600)
11 drawLine(100,400,275,300)
12 drawLine(450,400,275,300)
13 drawRectangle(255,450,405,550)
14 drawLine(330,450,330,550)
15 drawLine(255,500,405,500)
16 drawRectangle(140,450,210,600)
17 drawPoint(200,525)
18
19 endGrfx()
```

```
1  # GraphicsLibrary04
2  # This program combi
3  # and a point to dra
4
5
6  from Graphics import
7
8  beginGrfx(1000,650)
9
10 drawRectangle(100,400,450,600)
11 drawLine(100,400,275,300)
12 drawLine(450,400,275,300)
13 drawRectangle(255,450,405,550)
14 drawLine(330,450,330,550)
15 drawLine(255,500,405,500)
16 drawRectangle(140,450,210,600)
17 drawPoint(200,525)
18
19 endGrfx()
```



```
 1 # GraphicsLibrary05.py
 2 # This program demonstrates the Logic Error
 3 # that occurs when X and Y values are switched.
 4 # The house is "flipped" diagonally.
 5
 6
 7 from Graphics import *
 8
 9 beginGrfx(1000,650)
10
11 drawRectangle(400,100,600,450)
12 drawLine(400,100,300,275)
13 drawLine(400,450,300,275)
14 drawRectangle(450,255,550,405)
15 drawLine(450,330,550,330)
16 drawLine(500,255,500,405)
17 drawRectangle(450,140,600,210)
18 drawPoint(525,200)
19
20 endGrfx()
```

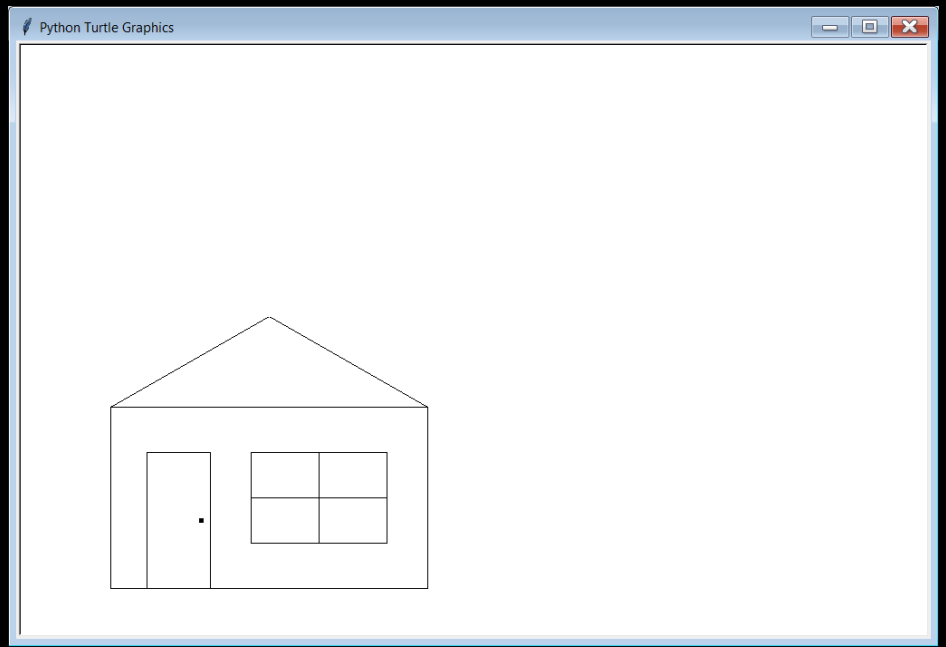
```
1  # GraphicsLibrary05.
2  # This program demon
3  # that occurs when X
4  # The house is "flip
5
6
7  from Graphics import
8
9  beginGrfx(1000,650)
10
11 drawRectangle(400,100,600,450)
12 drawLine(400,100,300,275)
13 drawLine(400,450,300,275)
14 drawRectangle(450,255,550,405)
15 drawLine(450,330,550,330)
16 drawLine(500,255,500,405)
17 drawRectangle(450,140,600,210)
18 drawPoint(525,200)
19
20 endGrfx()
```



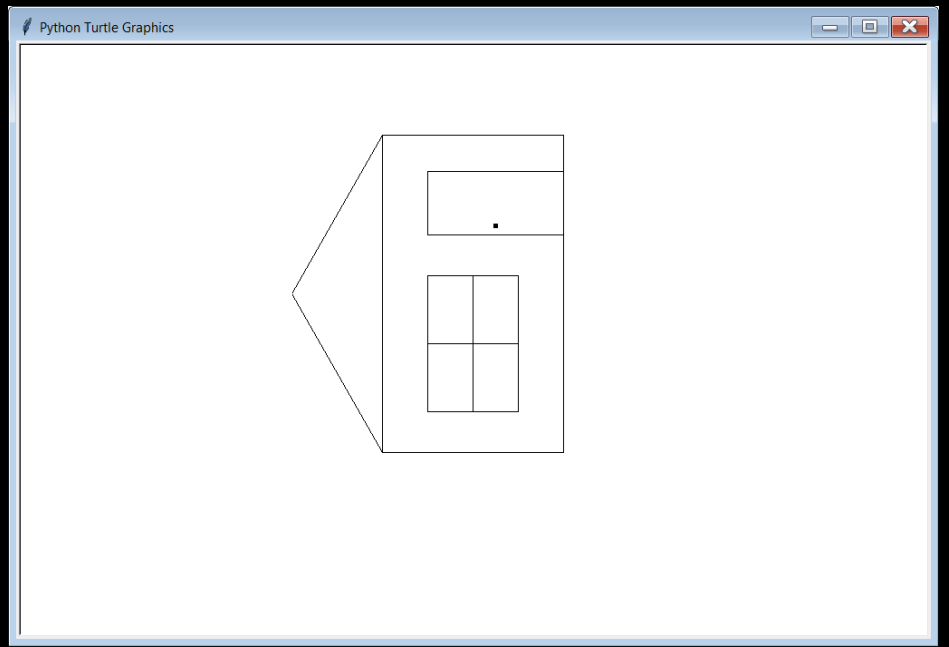
Argument Sequence Matters

GraphicsLibrary04.py vs. GraphicsLibrary05.py

```
drawRectangle(100,400,450,600)
drawLine(100,400,275,300)
drawLine(450,400,275,300)
drawRectangle(255,450,405,550)
drawLine(330,450,330,550)
drawLine(255,500,405,500)
drawRectangle(140,450,210,600)
drawPoint(200,525)
```



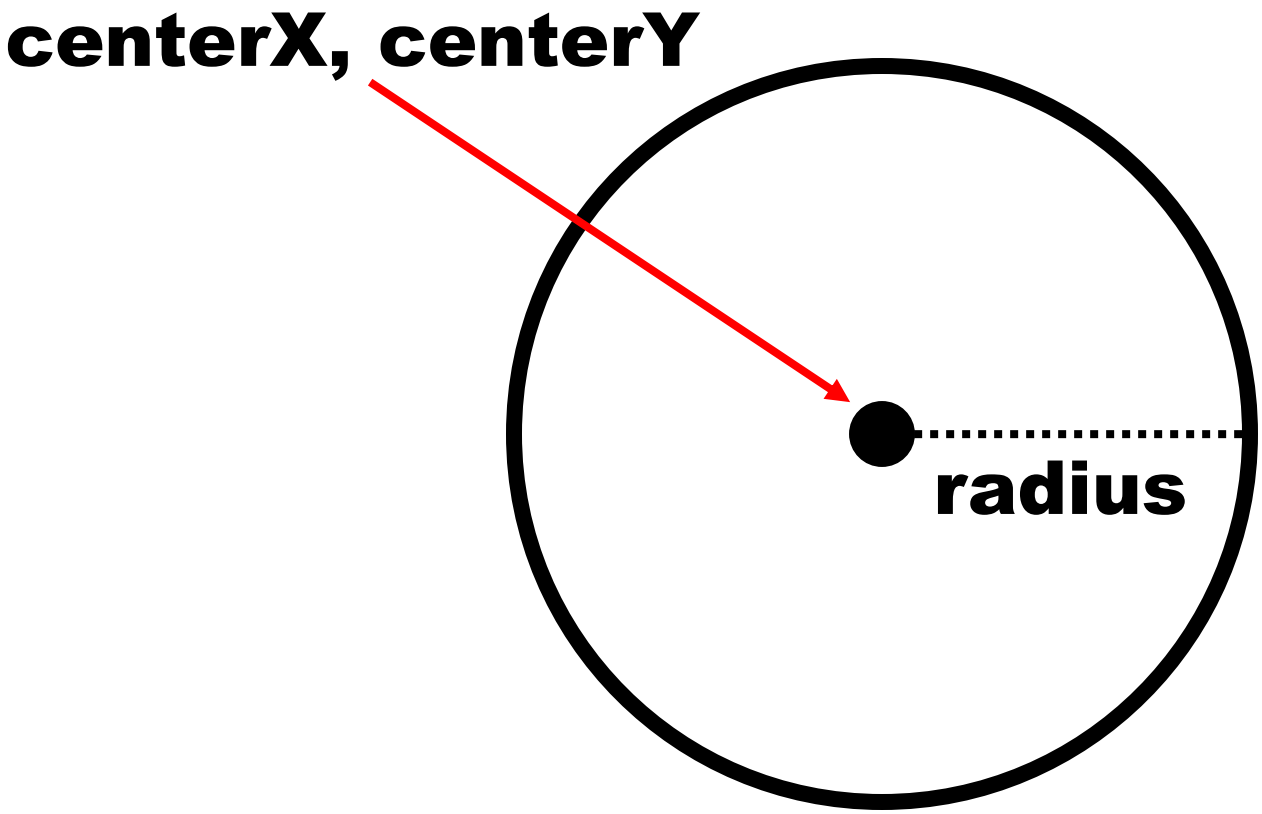

```
drawRectangle(400,100,600,450)
drawLine(400,100,300,275)
drawLine(400,450,300,275)
drawRectangle(450,255,550,405)
drawLine(450,330,550,330)
drawLine(500,255,500,405)
drawRectangle(450,140,600,210)
drawPoint(525,200)
```



Procedure drawCircle

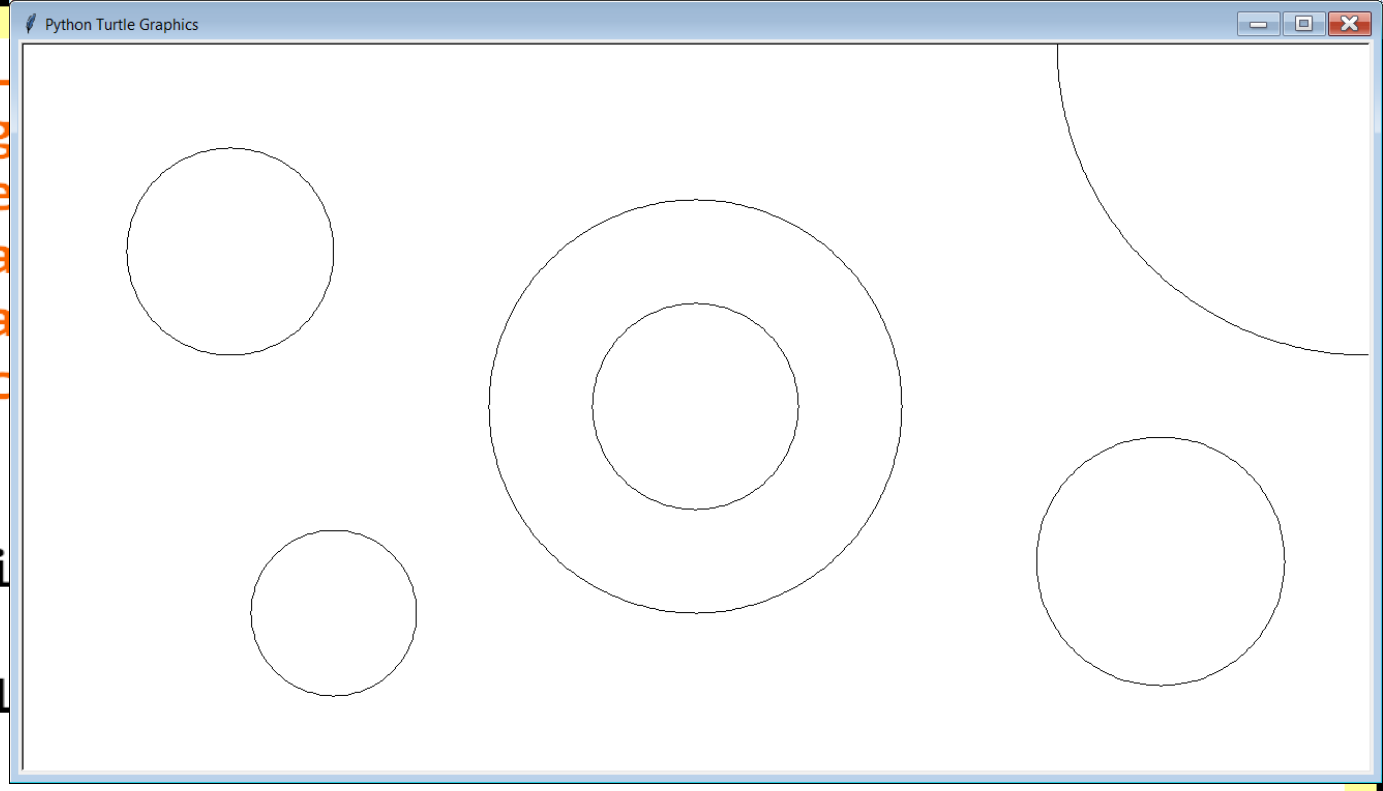
```
drawCircle(centerX, centerY, radius)
```

The location of the circle is specified in its center (centerX,centerY) and the size is specified by the radius.



```
 1 # GraphicsLibrary06.py
 2 # This program demonstrates the <drawCircle>
 3 # procedure of the <Graphics> library.
 4 # Circles are drawn from their center (x,y)
 5 # with a particular radius with
 6 # <drawCircle(x,y,radius)>.
 7
 8
 9 from Graphics import *
10
11 beginGrfx(1300,700)
12
13 drawCircle(200,200,100)
14 drawCircle(1300,0,300)
15 drawCircle(650,350,100)
16 drawCircle(650,350,200)
17 drawCircle(300,550,80)
18 drawCircle(1100,500,120)
19
20 endGrfx()
```

```
 1 # GraphicsL
 2 # This prog
 3 # procedure
 4 # Circles a
 5 # with a pa
 6 # <drawCirc
 7
 8
 9 from Graphi
10
11 beginGrfx(1
12
13 drawCircle(200,200,100)
14 drawCircle(1300,0,300)
15 drawCircle(650,350,100)
16 drawCircle(650,350,200)
17 drawCircle(300,550,80)
18 drawCircle(1100,500,120)
19
20 endGrfx()
```



```
1 # GraphicsLibrary07.py
2 # This program demonstrates the Syntax Error
3 # that occurs when a function is called
4 # with too few arguments.
5
6 from Graphics import *
7
8 beginGrfx(1300,700)
9
10 drawCircle(650,350)
11
12 endGrfx()
13
```

```
  ----jGRASP exec: python GraphicsLibrary07.py
  Traceback (most recent call last):
    File "GraphicsLibrary07.py", line 10, in <module>
      drawCircle(650,350)
  TypeError: drawCircle() missing 1 required
positional argument: 'r'

  ----jGRASP wedge2: exit code for process is 1.
  ----jGRASP: operation complete.
```

```
 8 beginGrfx(1300,700)
 9
10 drawCircle(650,350)
11
12 endGrfx()
13
```

```
1 # GraphicsLibrary08.py
2 # This program demonstrates the Syntax Error
3 # that occurs when a function is called
4 # with too many arguments.
5
6 from Graphics import *
7
8 beginGrfx(1300,700)
9
10 drawCircle(650,350,100,200)
11
12 endGrfx()
13
```

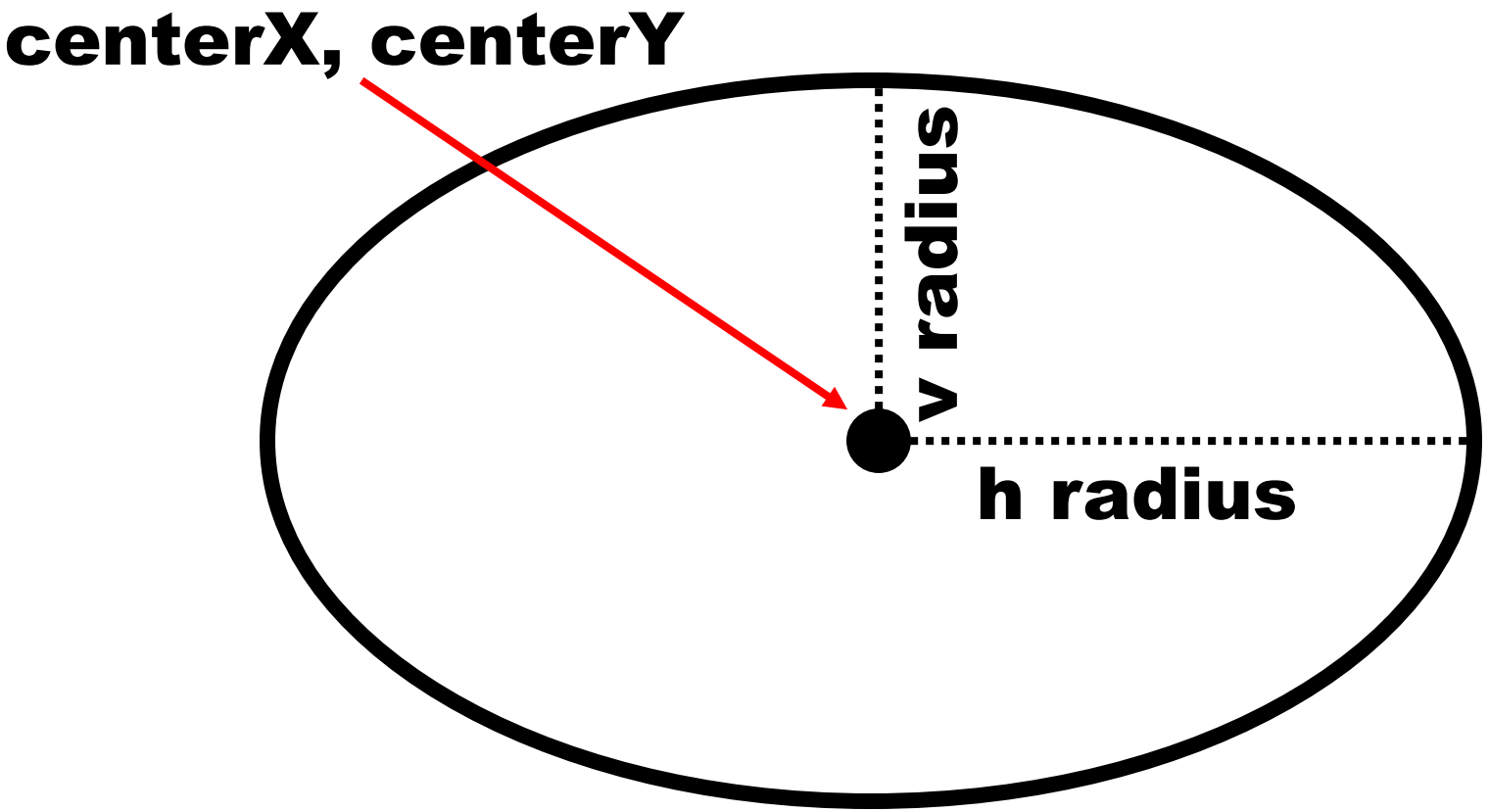
```
  ----jGRASP exec: python GraphicsLibrary08.py
  Traceback (most recent call last):
    File "GraphicsLibrary08.py", line 10, in <module>
      drawCircle(650,350,100,200)
  TypeError: drawCircle() takes 3 positional
arguments but 4 were given

  ----jGRASP wedge2: exit code for process is 1.
  ----jGRASP: operation complete.
```

```
 8 beginGrfx(1300,700)
 9
10 drawCircle(650,350,100,200)
11
12 endGrfx()
13
```

Procedure drawOval

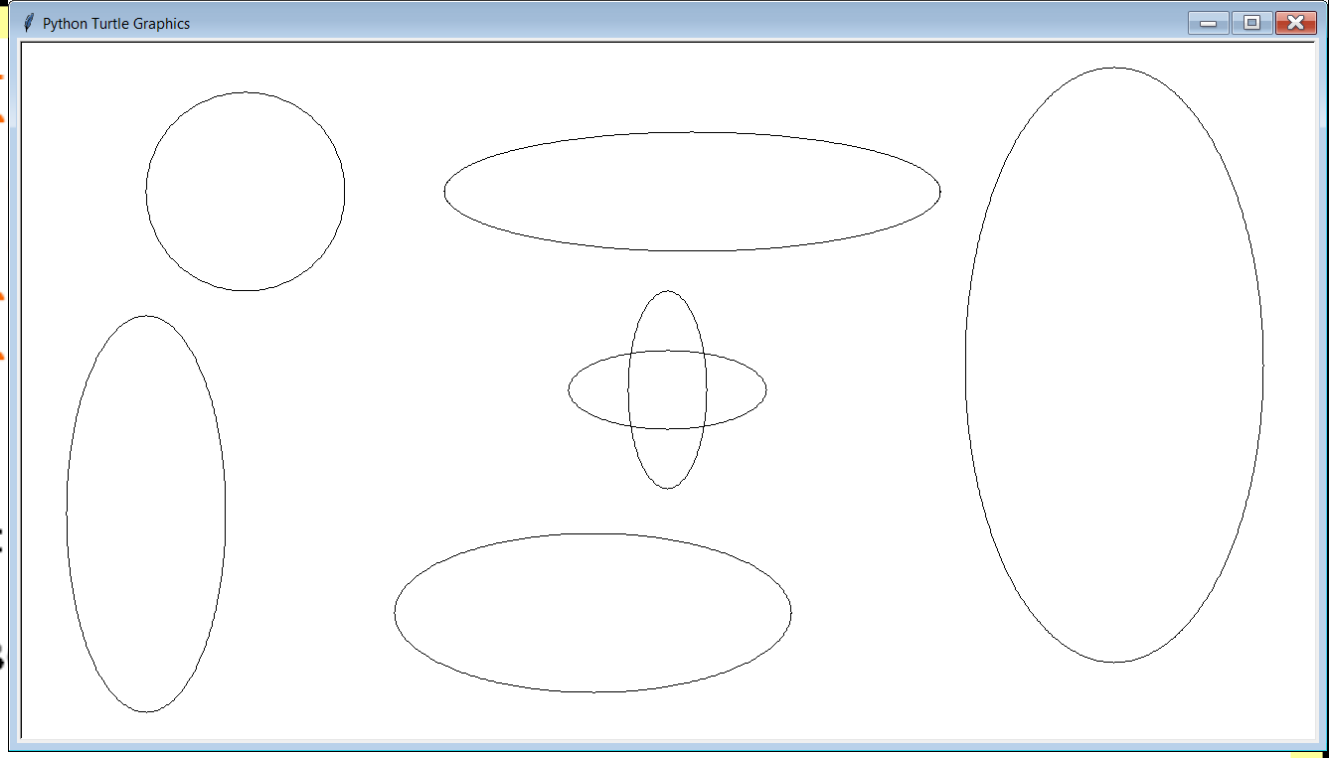
drawOval(centerX, centerY, horizontalRadius, verticalRadius)

The location of the oval is specified in its center (centerX,centerY) and the size is specified by the 2 radii.



```
 1 # GraphicsLibrary09.py
 2 # This program demonstrates the <drawOval>
 3 # procedure of the <Graphics> library.
 4 # Ovals are drawn from their center (x,y)
 5 # with a horizontal radius (hr) and a vertical
 6 # radius (vr) with <drawOval(x,y,hr,vr)>.
 7
 8
 9 from Graphics import *
10
11 beginGrfx(1300,700)
12
13 drawOval(225,150,100,100)
14 drawOval(1100,325,150,300)
15 drawOval(675,150,250,60)
16 drawOval(650,350,40,100)
17 drawOval(650,350,100,40)
18 drawOval(125,475,80,200)
19 drawOval(575,575,200,80)
20
21 endGrfx()
```

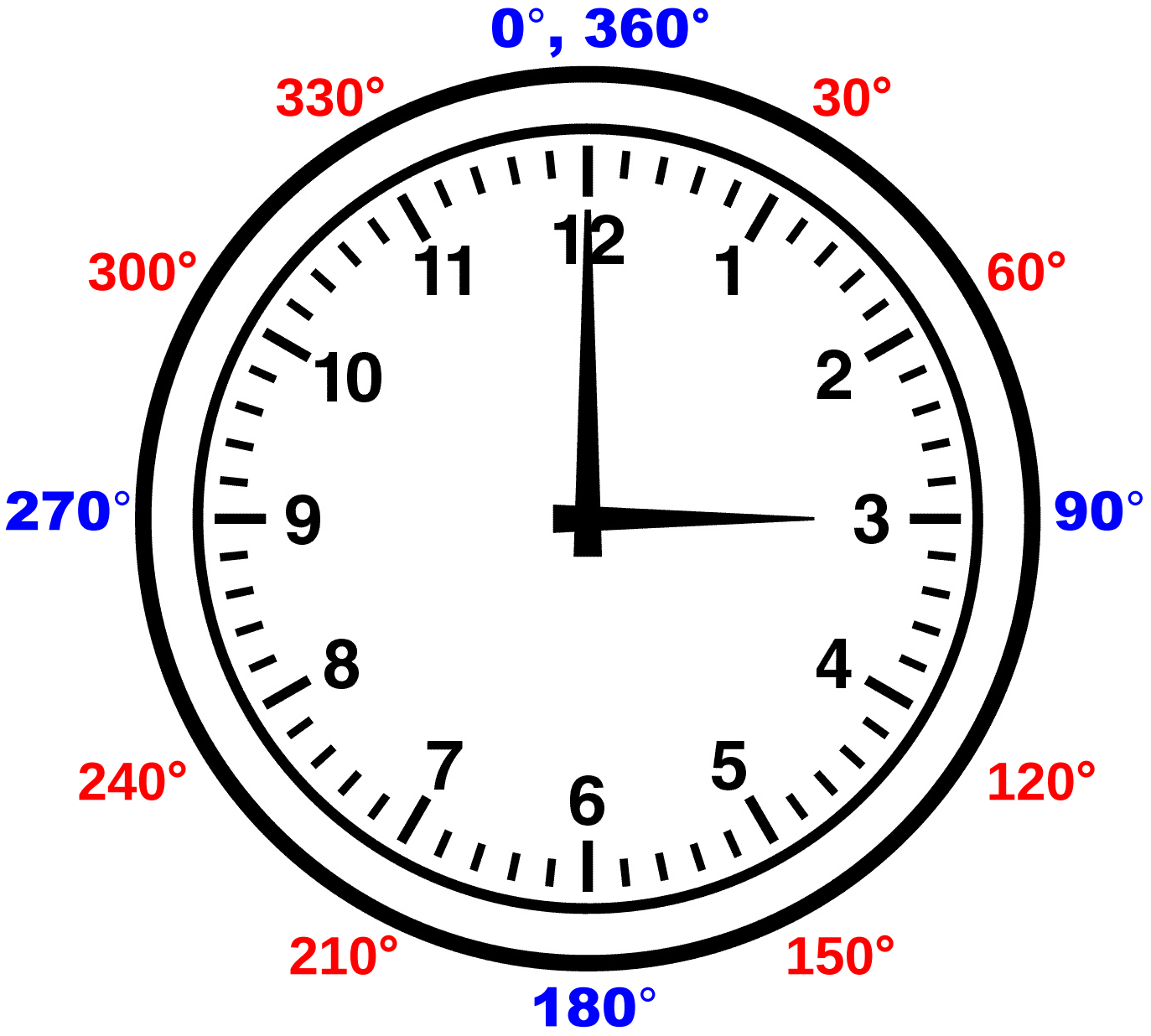
```
 1 # GraphicsLi
 2 # This progr
 3 # procedure
 4 # Ovals are
 5 # with a hor
 6 # radius (vr
 7
 8
 9 from Graphic
10
11 beginGrfx(13
12
13 drawOval(225,150,100,100)
14 drawOval(1100,325,150,300)
15 drawOval(675,150,250,60)
16 drawOval(650,350,40,100)
17 drawOval(650,350,100,40)
18 drawOval(125,475,80,200)
19 drawOval(575,575,200,80)
20
21 endGrfx()
```



Drawing Arcs

An *arc* is a piece of an *oval*.

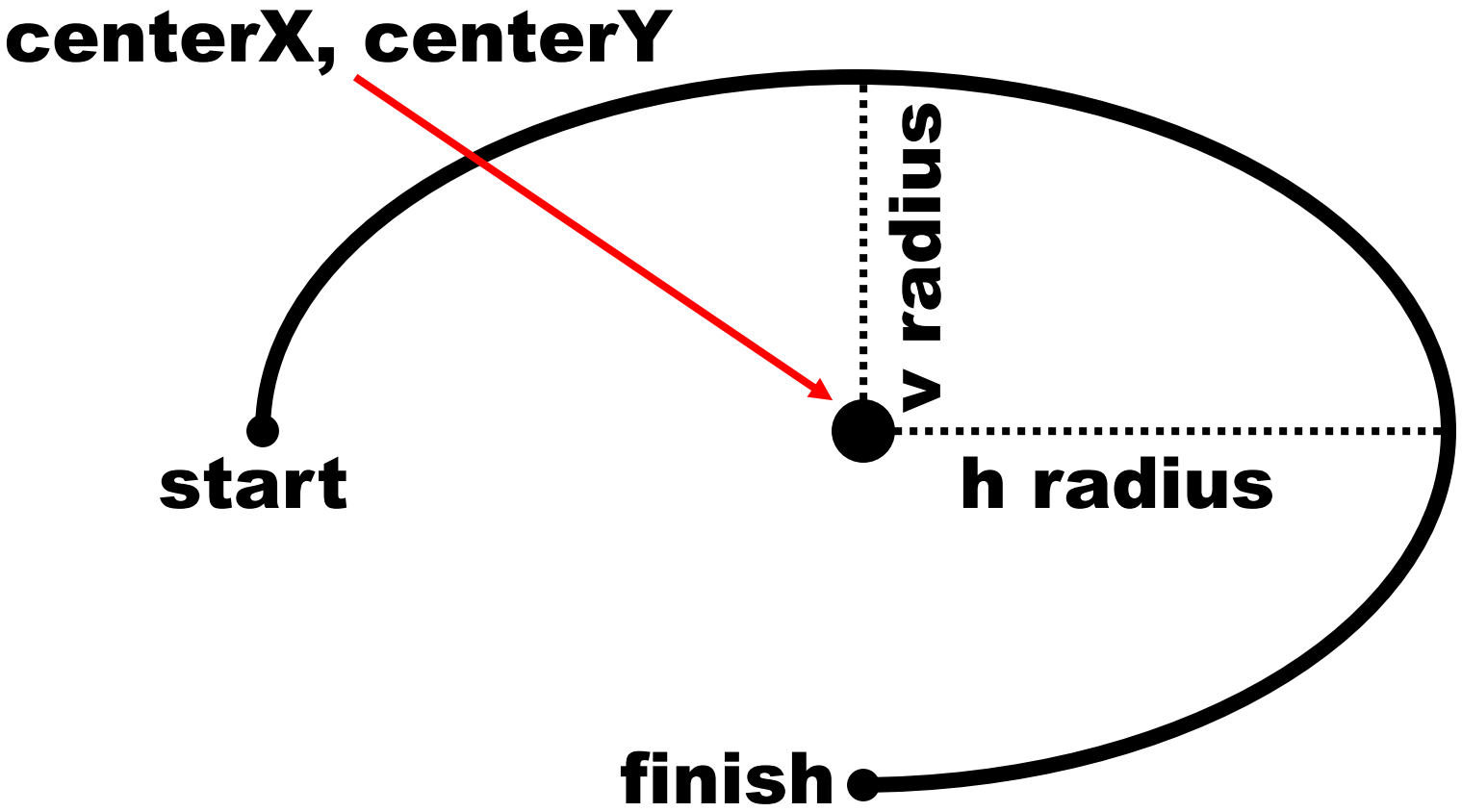
In order to draw an "arc" you need specify where the arc starts and where it stops.



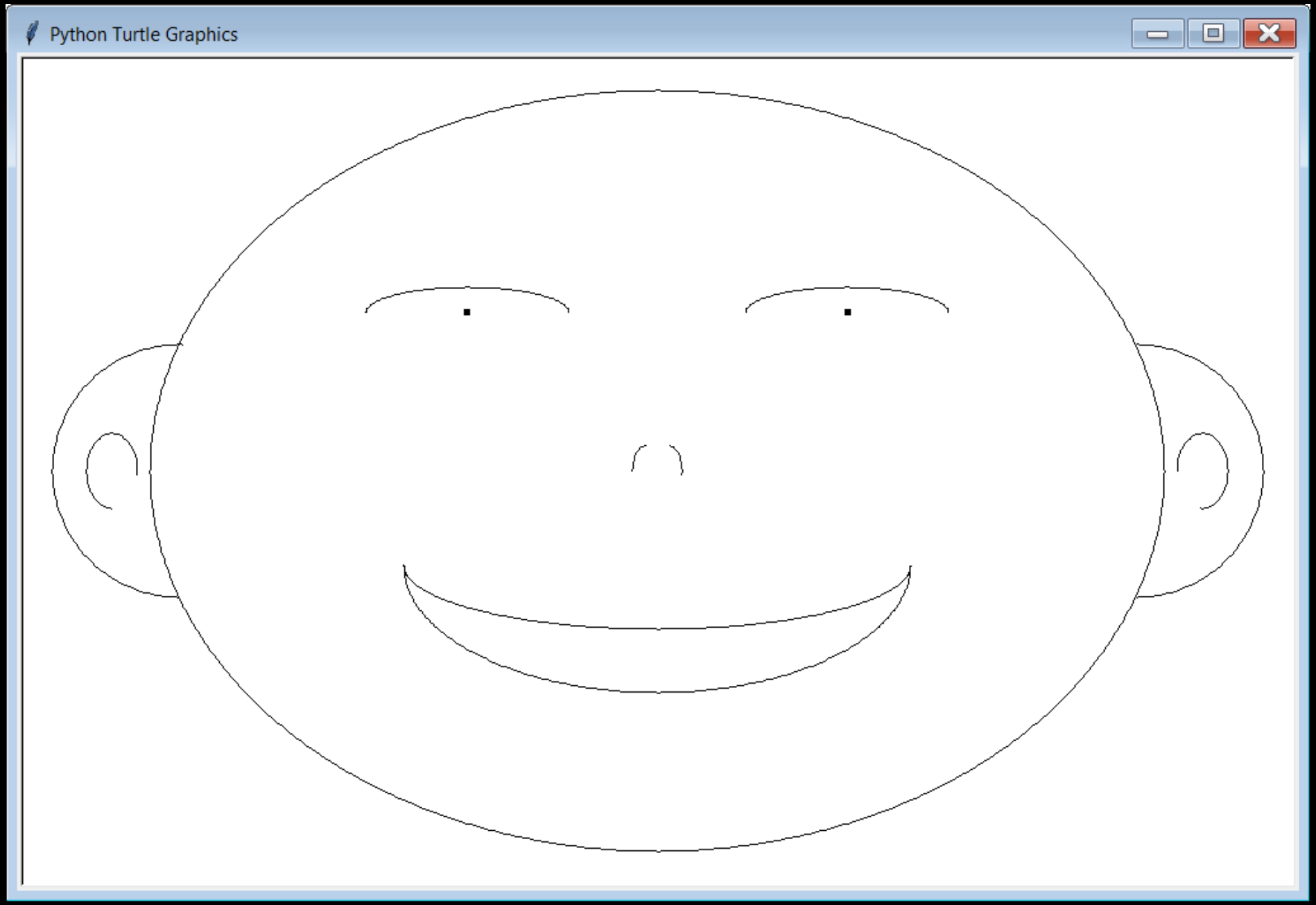
Procedure drawArc

drawArc(centerX, centerY, horizontal radius, vertical radius, start, finish)

Draws part of an oval. The first 4 arguments are the same as drawOval.
start indicates the degree location of the beginning of the arc.
finish indicates the degree location of the end of the arc.



```
 1 # GraphicsLibrary10.py
 2 # This program demonstrates the <drawArc> procedure of the
 3 # <Graphics> library.  An "arc" is a piece of an "oval".
 4 # Like ovals, arcs are drawn from their center (x,y) with
 5 # a horizontal radius (hr) and a vertical radius (vr).
 6 # Arcs also require a starting and stopping degree value.
 7 # This is done with <drawArc(x,y,hr,vr,start,stop)>.
 8
 9
10 from Graphics import *
11
12 beginGrfx(1000,650)
13
14 drawArc(500,325,400,300,0,360)    # complete oval
15 drawArc(500,400,200,50,90,270)    # bottom half of an oval
16 drawArc(500,400,200,100,90,270)
17 drawArc(350,200,80,20,270,90)     # top half of an oval
18 drawArc(650,200,80,20,270,90)
19 drawArc(123,325,100,100,180,0)    # left half of an oval
20 drawArc(878,325,100,100,0,180)    # right half of an oval
21 drawArc(490,325,10,20,270,360)    # top-left 1/4 of an oval
22 drawArc(510,325,10,20,0,90)       # top-right 1/4 of an oval
23 drawArc(70,325,20,30,180,90)      # 3/4 of an oval
24 drawArc(930,325,20,30,270,180)    # different 3/4 of an oval
25 drawPoint(350,200)
26 drawPoint(650,200)
27
28 endGrfx()
```


Python Turtle Graphics

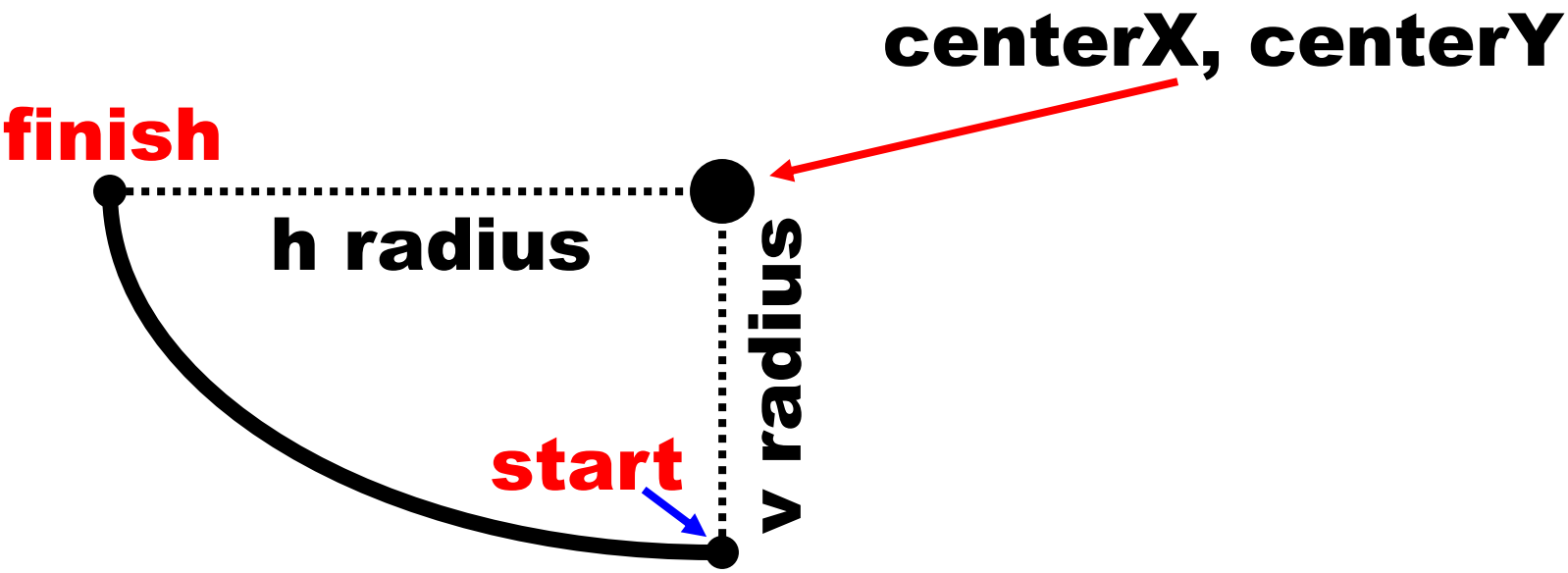
Argument Sequence *Again*

drawArc(centerX, centerY, horizontal radius, vertical radius, start, finish)

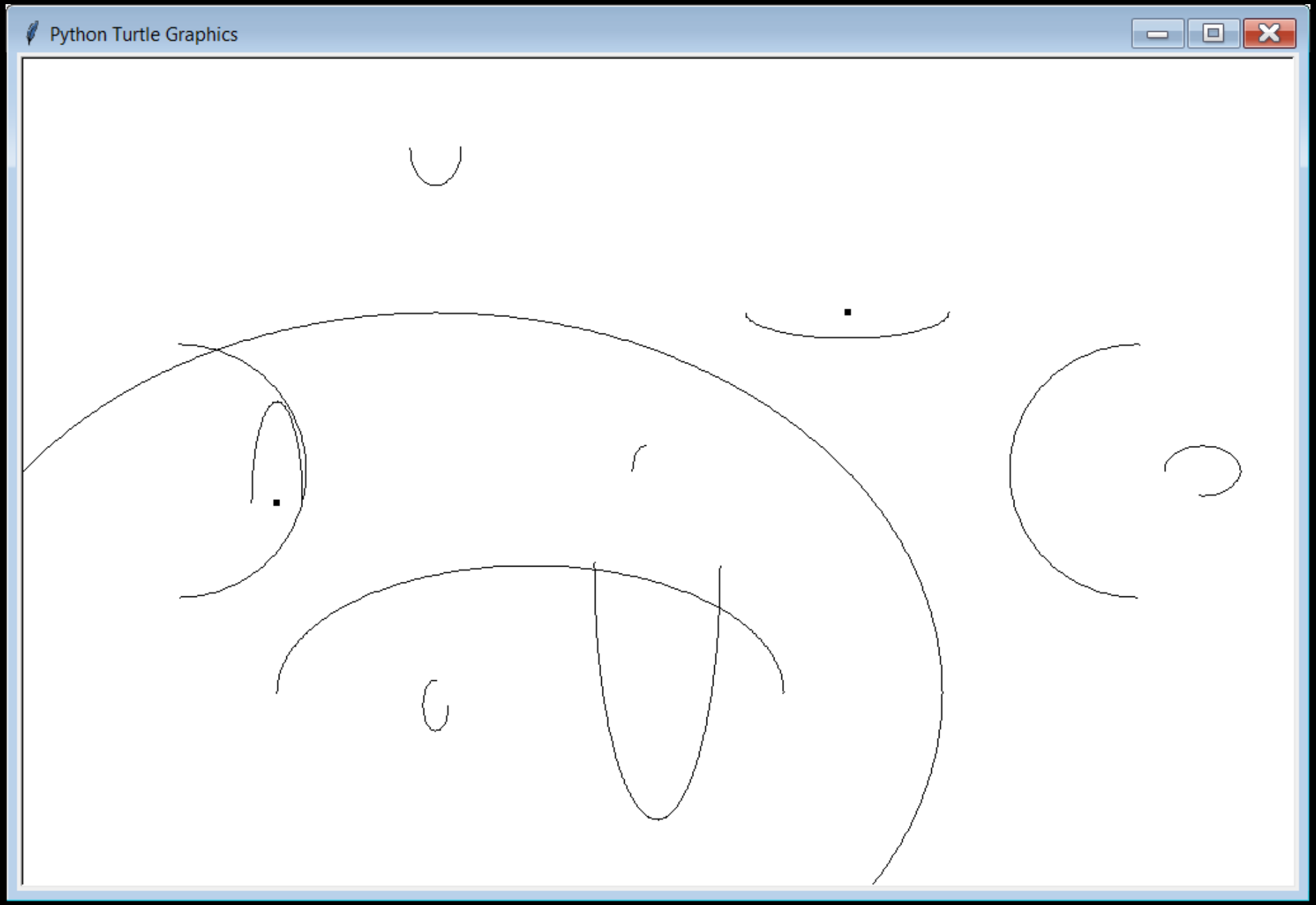
Argument sequence is VERY significant !!!!!!!

Simply switching the order of the **start** and **finish** arguments causes a completely different arc to be drawn.

The next program will graphically demonstrate what happens when arguments are out of order.



```
 1 # GraphicsLibrary11.py
 2 # This program may seem almost identical to the
 3 # previous program which drew the smiley face.
 4 # In reality, it demonstrates what happens when
 5 # arguments are put in the wrong order.  The
 6 # program does execute without any errors, but
 7 # the results are not what you expect.
 8 # This is another example of a Logic Error.
 9
10
11 from Graphics import *
12
13 beginGrfx(1000,650)
14
15 drawArc(325,500,400,300,0,360)
16 drawArc(500,400,50,200,90,270)
17 drawArc(400,500,200,100,270,90)
18 drawArc(200,350,20,80,270,90)
19 drawArc(650,200,80,20,90,270)
20 drawArc(123,325,100,100,0,180)
21 drawArc(878,325,100,100,180,0)
22 drawArc(490,325,10,20,270,360)
23 drawArc(325,510,10,20,90,0)
24 drawArc(325,70,20,30,90,270)
25 drawArc(930,325,30,20,270,180)
26 drawPoint(200,350)
27 drawPoint(650,200)
28
29 endGrfx()
```

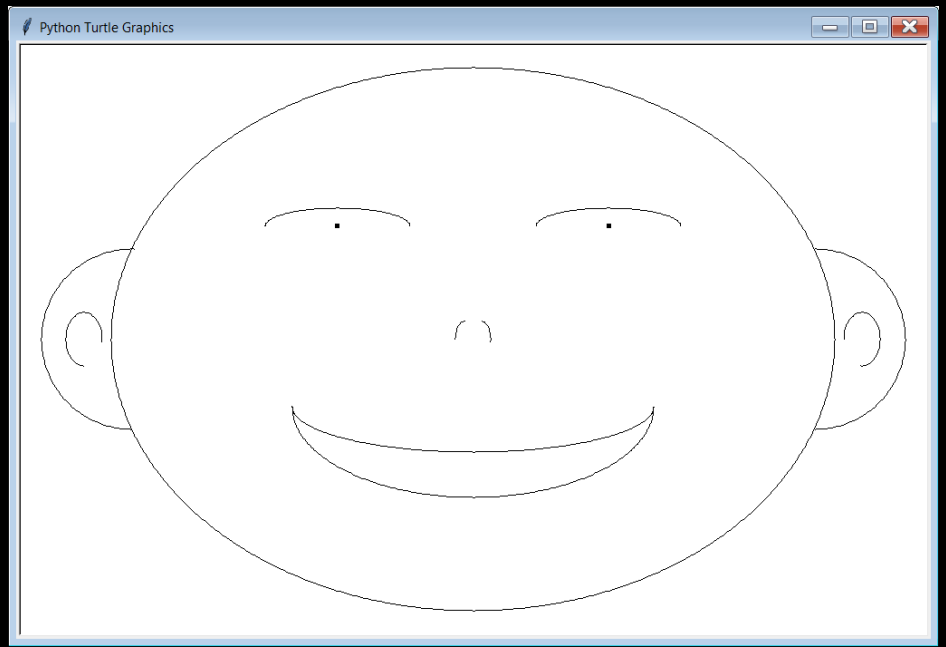

Python Turtle Graphics

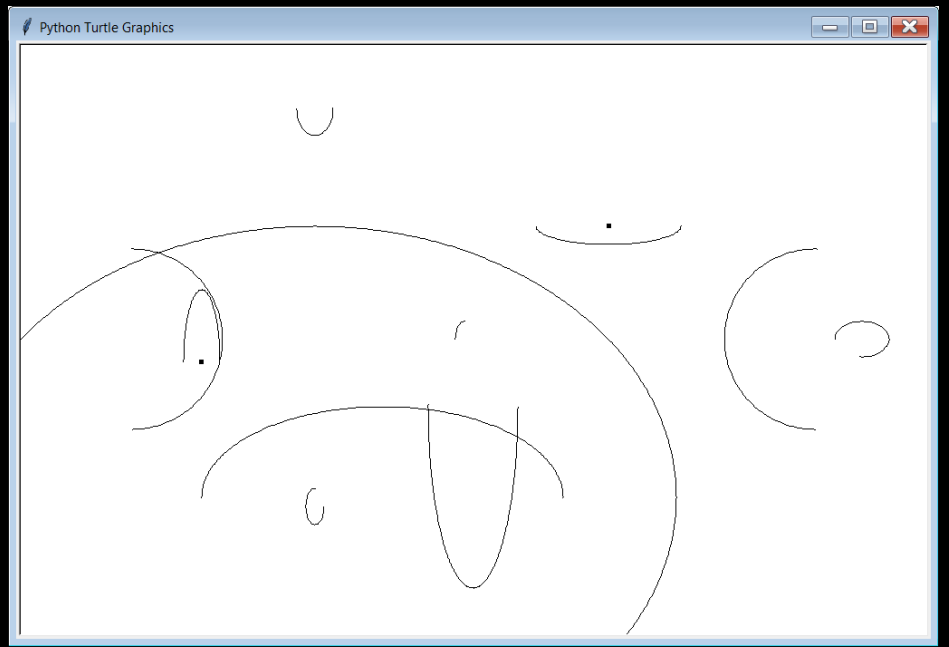
Argument Sequence Matters

GraphicsLibrary10.py vs. GraphicsLibrary11.py

```
drawArc(500,325,400,300,0,360)
drawArc(500,400,200,50,90,270)
drawArc(500,400,200,100,90,270)
drawArc(350,200,80,20,270,90)
drawArc(650,200,80,20,270,90)
drawArc(123,325,100,100,180,0)
drawArc(878,325,100,100,0,180)
drawArc(490,325,10,20,270,360)
drawArc(510,325,10,20,0,90)
drawArc(70,325,20,30,180,90)
drawArc(930,325,20,30,270,180)
drawPoint(350,200)
```

```
drawArc(325,500,400,300,0,360)
drawArc(500,400,50,200,90,270)
drawArc(400,500,200,100,270,90)
drawArc(200,350,20,80,270,90)
drawArc(650,200,80,20,90,270)
drawArc(123,325,100,100,0,180)
drawArc(878,325,100,100,180,0)
drawArc(490,325,10,20,270,360)
drawArc(325,510,10,20,90,0)
drawArc(325,70,20,30,90,270)
drawArc(930,325,30,20,270,180)
drawPoint(200,350)
```

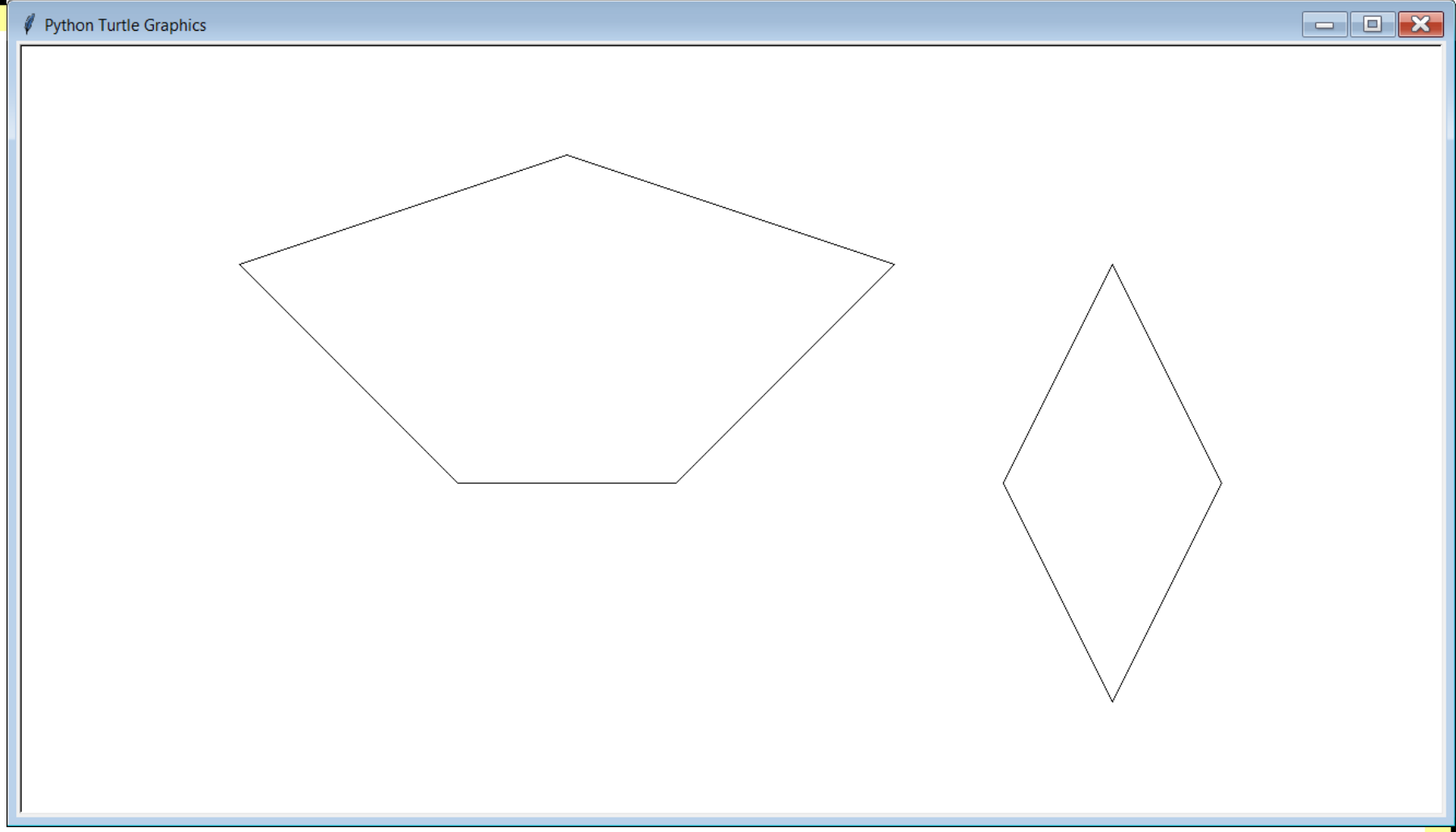




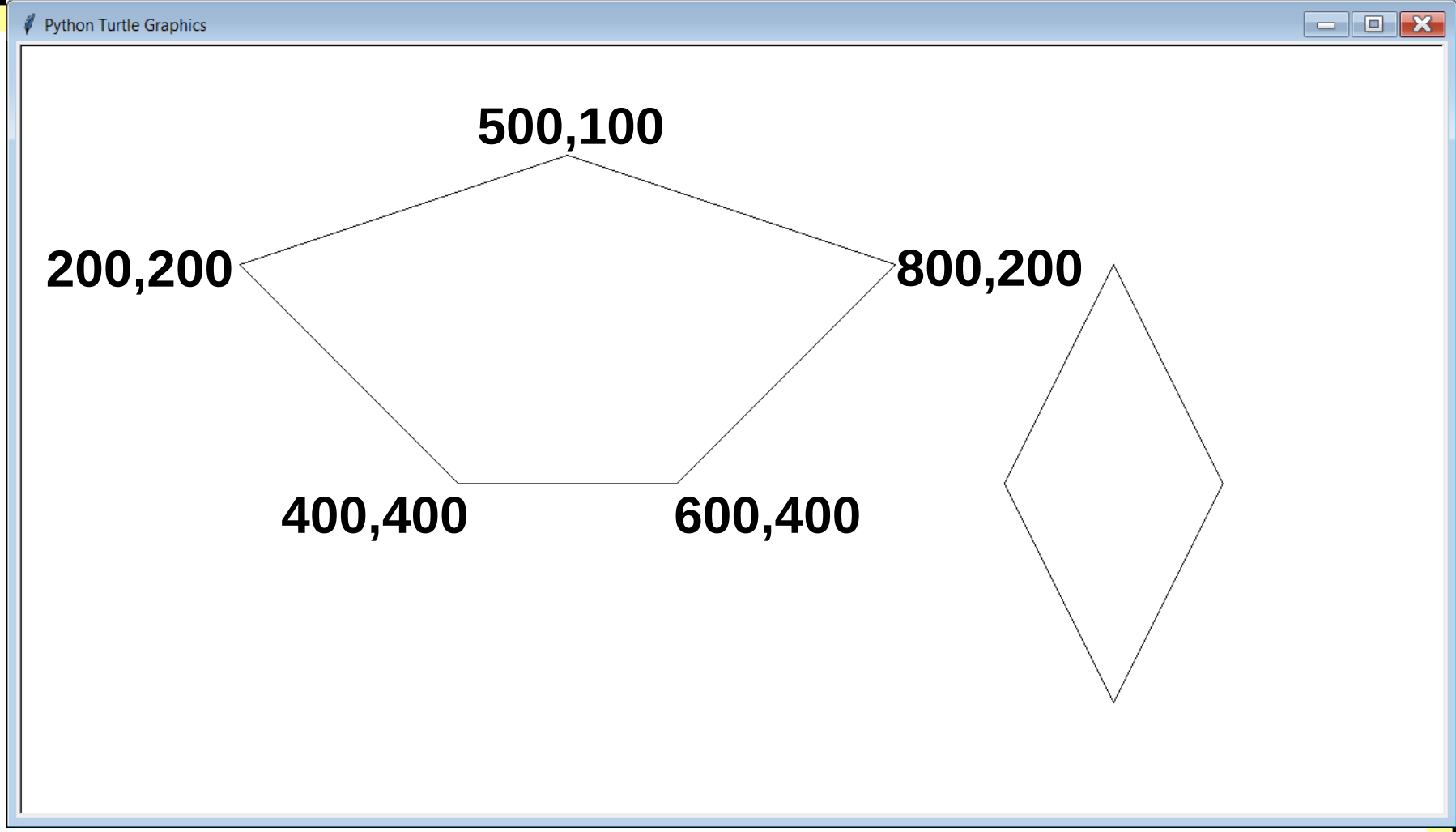
Section 6.6

Drawing Polygons

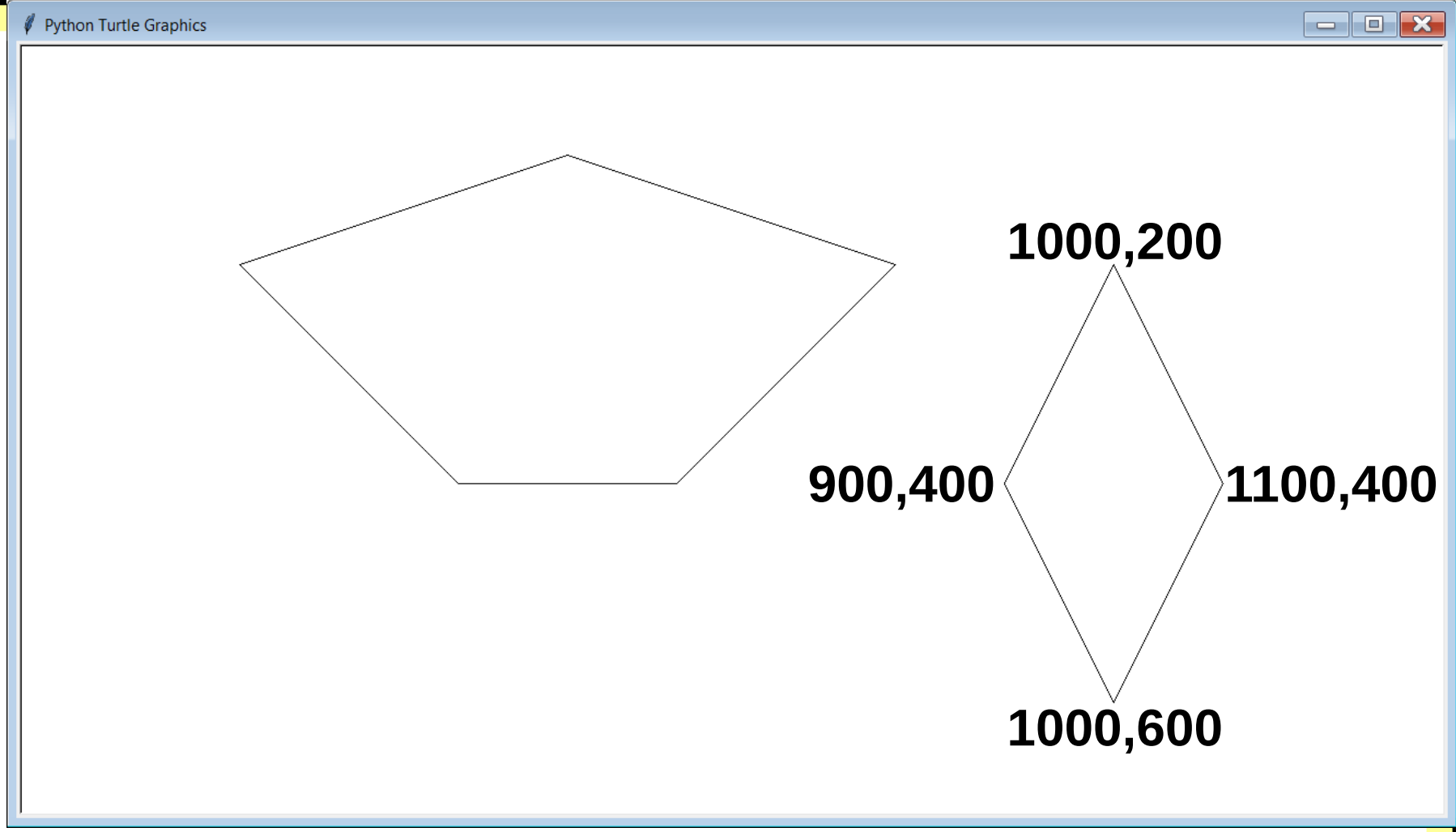
```
1  # GraphicsLibrary12.py
2  # This program demonstrates the <drawPolygon> procedure.
3  # <drawPolygon> can handle 3 or more sets of coordinate
4  # points to draw a triangle, quadrilateral, pentagon,
5  # hexagon, octagon, or any other polygon.
6  # The purpose of the extra set of brackets [] in the
7  # procedure call will be explained in a later chapter.
8
9
10 from Graphics import *
11
12 beginGrfx(1300,700)
13
14 drawPolygon([500,100,800,200,600,400,400,400,200,200])
15 drawPolygon([900,400,1000,200,1100,400,1000,600])
16
17 endGrfx()
18
```




```
drawPolygon([500,100,800,200,600,400,400,400,200,200])
drawPolygon([900,400,1000,200,1100,400,1000,600])

endGrfx()
```

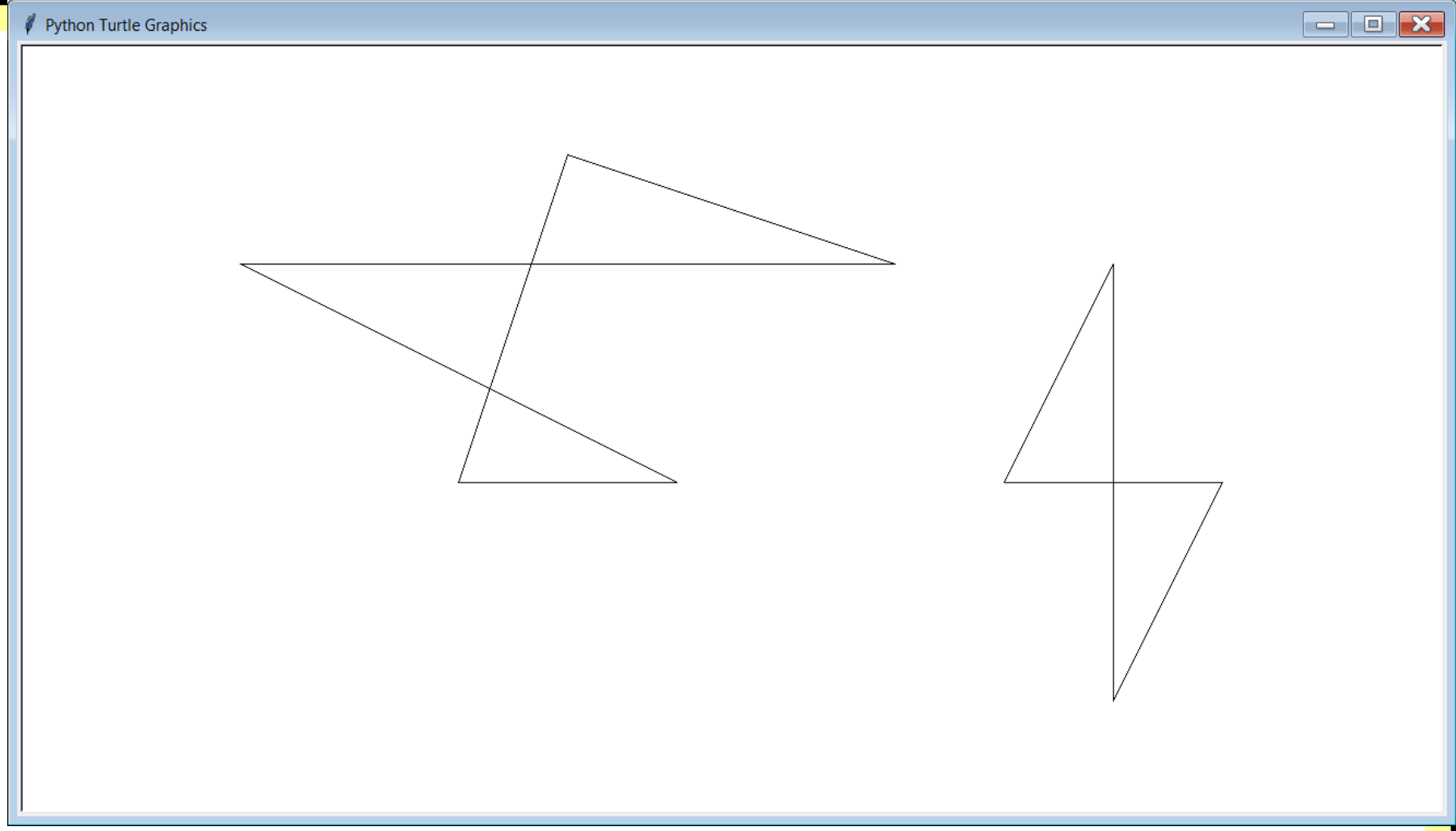



```
drawPolygon([500,100,800,200,600,400,400,400,200,200])
drawPolygon([900,400,1000,200,1100,400,1000,600])

endGrfx()
```

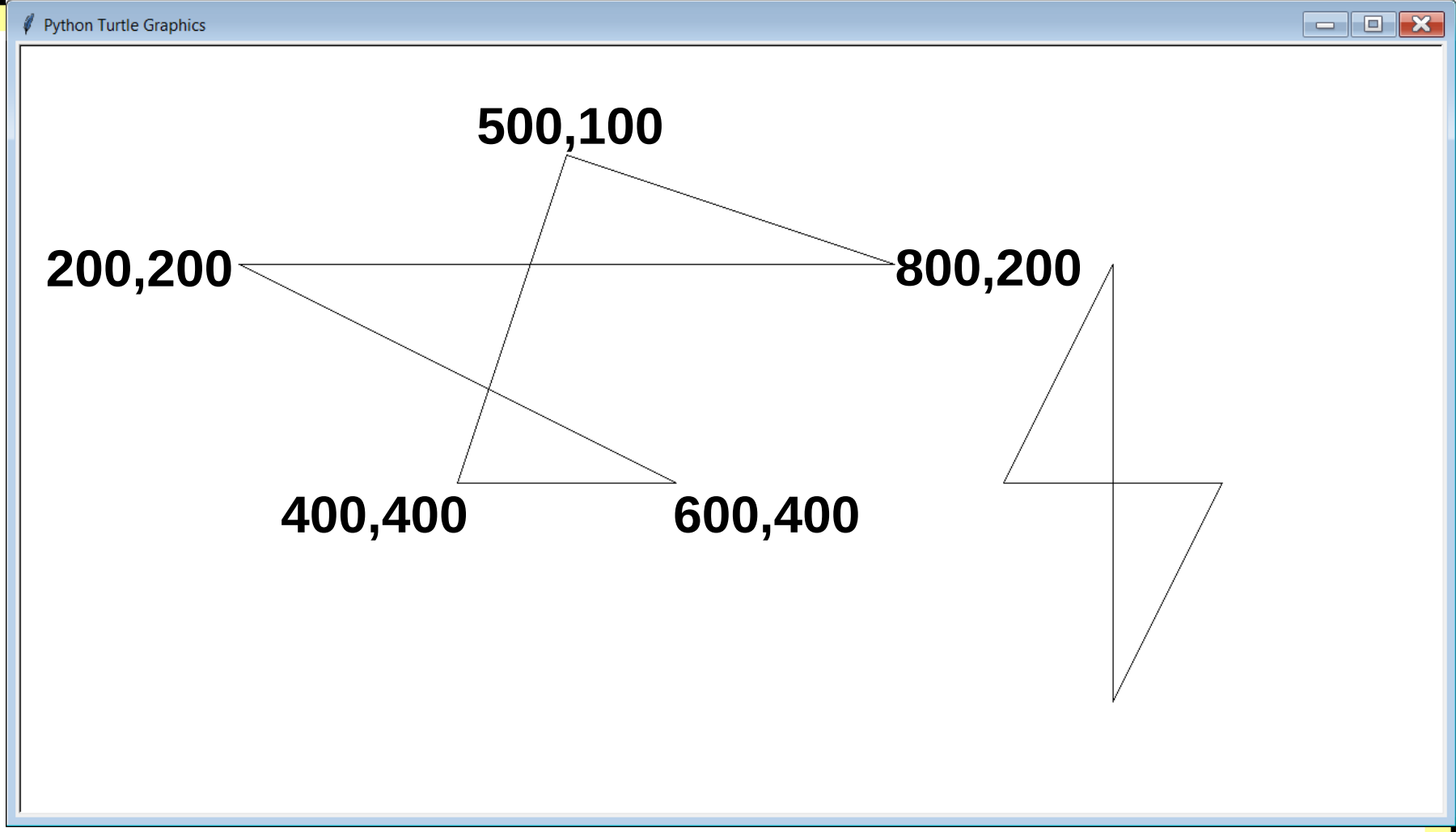



```
drawPolygon([500,100,800,200,600,400,400,400,200,200])
drawPolygon([900,400,1000,200,1100,400,1000,600])

endGrfx()
```

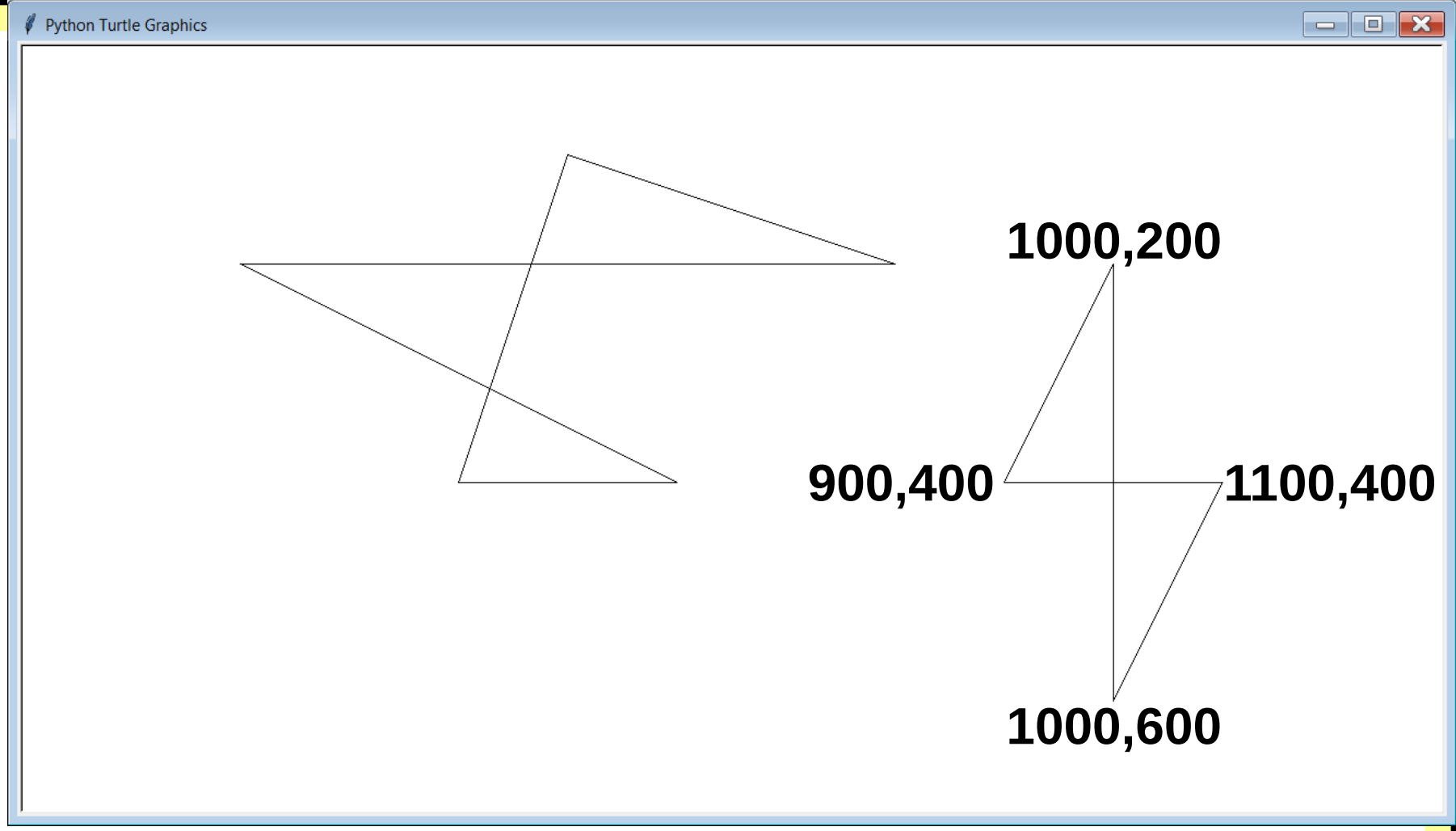
```
1 # GraphicsLibrary13.py
2 # This program demonstrates that the sequence of
3 # the coordinate pairs is significant.  The same
4 # coordinates from the previous program are used
5 # in a different sequence.  The result is that
6 # the display is now very different.
7
8
9 from Graphics import *
10
11 beginGrfx(1300,700)
12
13 drawPolygon([400,400,500,100,800,200,200,200,600,400])
14 drawPolygon([900,400,1000,200,1000,600,1100,400])
15
16 endGrfx()
```




```
drawPolygon([400,400,500,100,800,200,200,200,600,400])
drawPolygon([900,400,1000,200,1000,600,1100,400])

endGrfx()
```




```
drawPolygon([400,400,500,100,800,200,200,200,600,400])
drawPolygon([900,400,1000,200,1000,600,1100,400])

endGrfx()
```



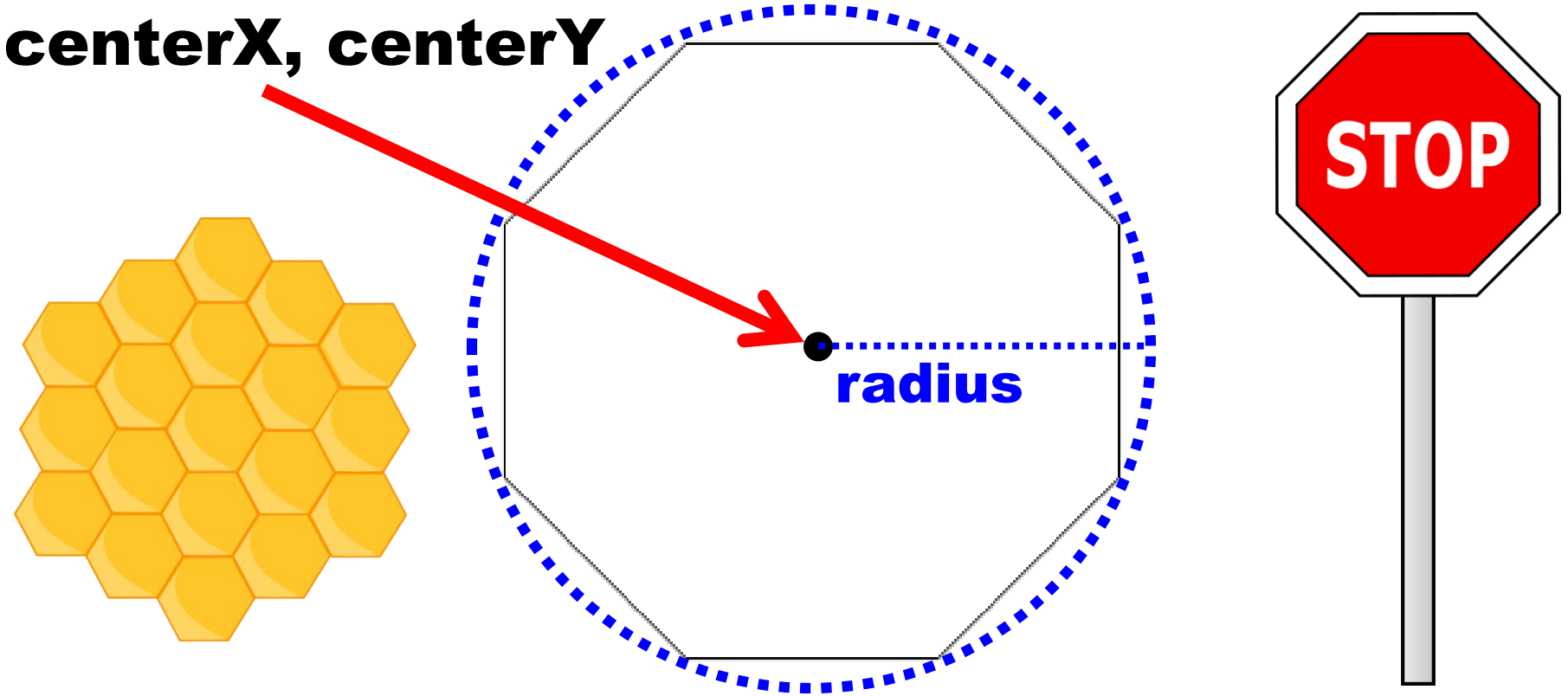

```
drawPolygon([400,400,500,100,800,200,200,200,600,400])
drawPolygon([900,400,1000,200,1000,600,1100,400])

endGrfx()
```

Procedure drawRegularPolygon

drawRegularPolygon(centerX, centerY, radius, numSides)

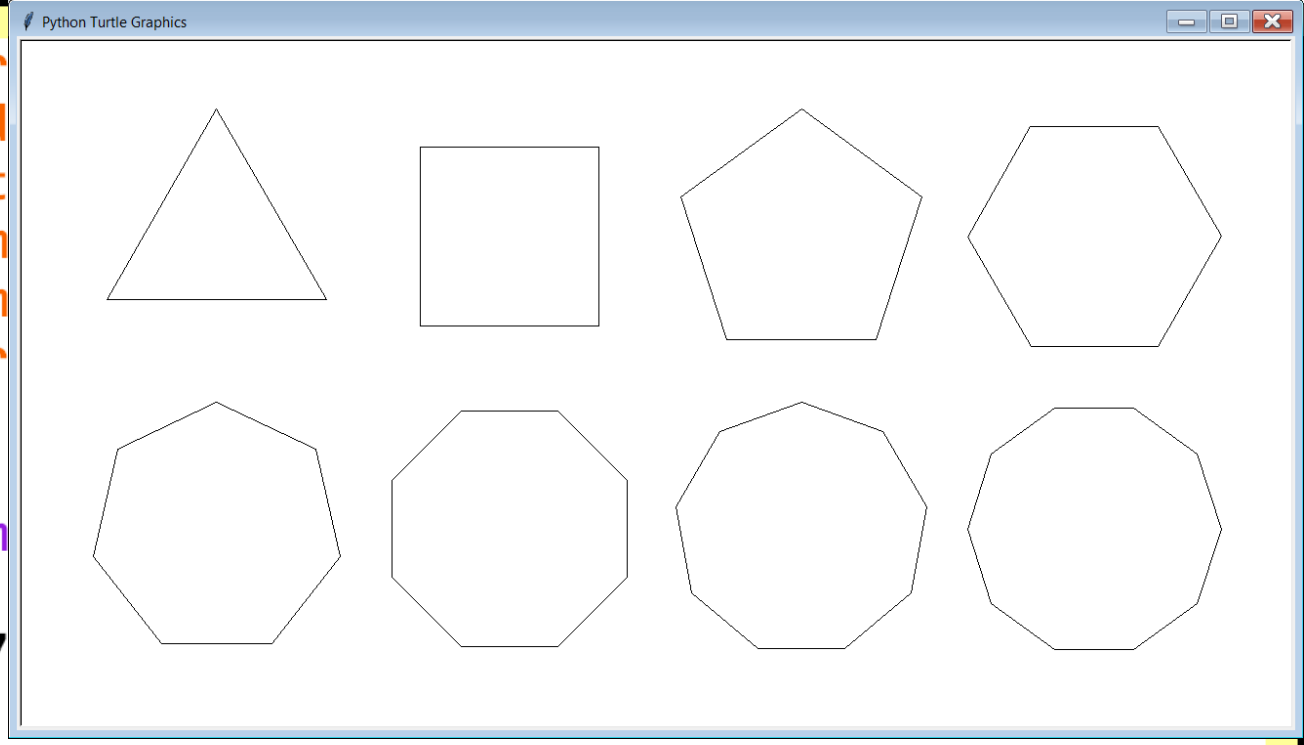
The first 3 arguments for drawRegularPolygon are the same as drawCircle.

The last argument indicates the number of sides.



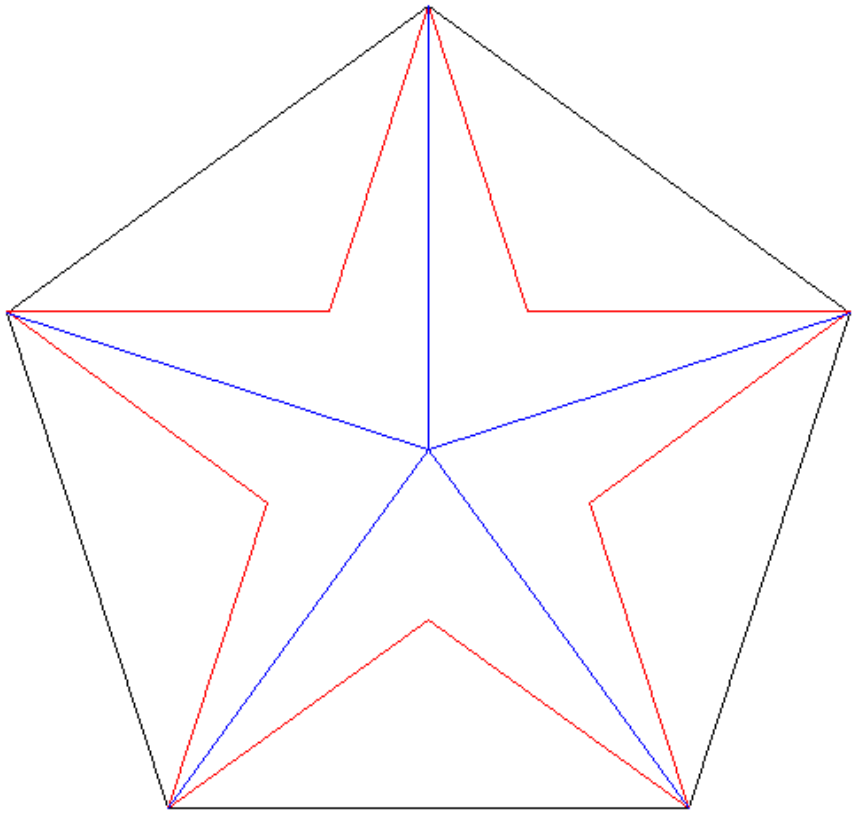
```
 1 # GraphicsLibrary14.py
 2 # This program demonstrates the <drawRegularPolygon>
 3 # procedure of the <Graphics> library.  Regular Polygons
 4 # are drawn from their center (x,y) with a certain radius
 5 # (of the circumscribing circle) and a certain number of
 6 # sides with <drawRegularPolygon(x,y,radius,numSides)>.
 7
 8
 9 from Graphics import *
10
11 beginGrfx(1300,700)
12
13 drawRegularPolygon(200,200,130,3)
14 drawRegularPolygon(500,200,130,4)
15 drawRegularPolygon(800,200,130,5)
16 drawRegularPolygon(1100,200,130,6)
17 drawRegularPolygon(200,500,130,7)
18 drawRegularPolygon(500,500,130,8)
19 drawRegularPolygon(800,500,130,9)
20 drawRegularPolygon(1100,500,130,10)
21
22 endGrfx()
```

```
1  # GraphicsLibrar
2  # This program d
3  # procedure of t
4  # are drawn from
5  # (of the circum
6  # sides with <dr
7
8
9  from Graphics im
10
11 beginGrfx(1300,7
12
13 drawRegularPolygon(200,200,130,3)
14 drawRegularPolygon(500,200,130,4)
15 drawRegularPolygon(800,200,130,5)
16 drawRegularPolygon(1100,200,130,6)
17 drawRegularPolygon(200,500,130,7)
18 drawRegularPolygon(500,500,130,8)
19 drawRegularPolygon(800,500,130,9)
20 drawRegularPolygon(1100,500,130,10)
21
22 endGrfx()
```



Special Regular Polygons

To me, *stars* and *snowflakes* are special kinds of regular polygons. Below you can see that a 5-point star and a 5-point snowflake both fit perfectly inside a 5-sided regular polygon.

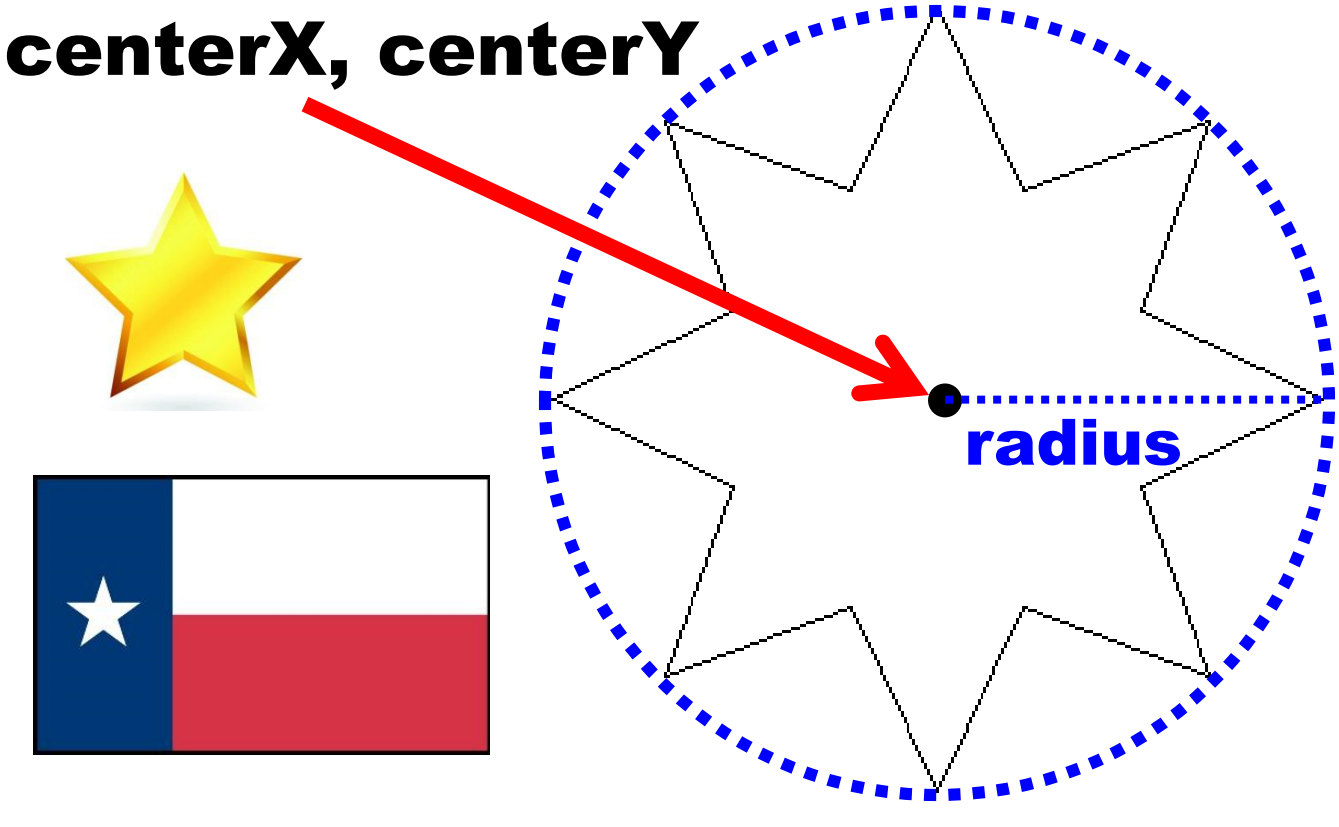


Procedure drawStar

```
drawStar(centerX, centerY, radius, numPoints)
```

The first 3 arguments for drawStar are the same as drawCircle.

The last argument indicates the number of points.

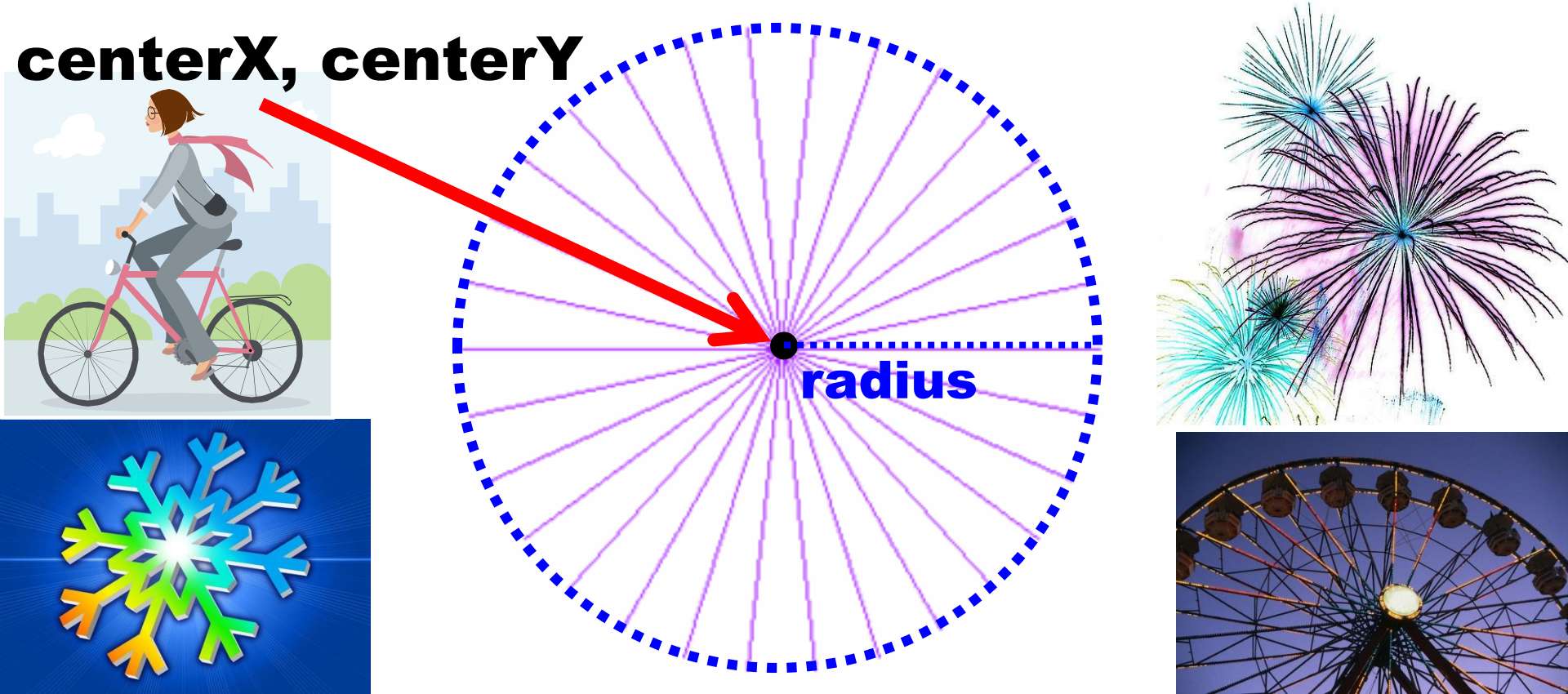


Procedure drawBurst

```
drawBurst(centerX, centerY, radius, numLines)
```

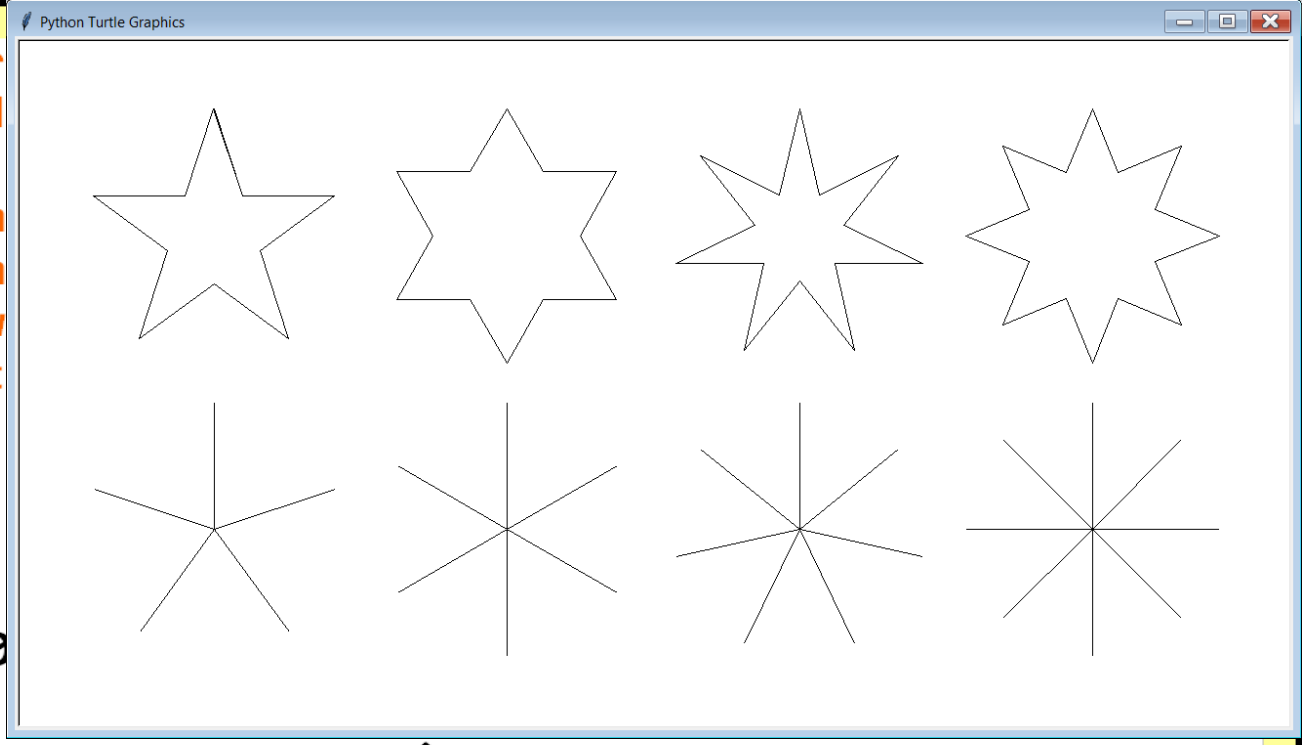
The first 3 arguments for drawBurst are the same as drawCircle.

The last argument indicates the number of lines.



```
1  # GraphicsLibrary15.py
2  # This program demonstrates the <drawStar> & <drawBurst>
3  # procedures of the <Graphics> library.  Stars and bursts
4  # are drawn from their center (x,y) with a certain radius
5  # (of the circumscribing circle) and a certain number of
6  # points/lines with <drawStar(x,y,radius,numPoints)>
7  # and <drawBurst(x,y,radius,numLines)>.
8
9
10 from Graphics import *
11
12 beginGrfx(1300,700)
13
14 drawStar(200,200,130,5)
15 drawStar(500,200,130,6)
16 drawStar(800,200,130,7)
17 drawStar(1100,200,130,8)
18 drawBurst(200,500,130,5)
19 drawBurst(500,500,130,6)
20 drawBurst(800,500,130,7)
21 drawBurst(1100,500,130,8)
22
23 endGrfx()
```

```
1  # GraphicsLibrar
2  # This program d
3  # procedures of
4  # are drawn from
5  # (of the circum
6  # points/lines w
7  # and <drawBurst
8
9
10 from Graphics
11
12 beginGrfx(1300
13
14 drawStar(200,200,130,5)
15 drawStar(500,200,130,6)
16 drawStar(800,200,130,7)
17 drawStar(1100,200,130,8)
18 drawBurst(200,500,130,5)
19 drawBurst(500,500,130,6)
20 drawBurst(800,500,130,7)
21 drawBurst(1100,500,130,8)
22
23 endGrfx()
```



Section 6.7

Fill Procedures and Colors

Procedure fillRectangle

`fillRectangle(x1, y1, x2, y2)`

Draws a SOLID (filled in) rectangle with a top-left corner at coordinate (x1,y1) and a bottom-right hand corner of (x2,y2).

```
 1 # GraphicsLibrary16.py
 2 # This program demonstrates the <fillRectangle>
 3 # procedure of the <Graphics> library.  The
 4 # arguments for <fillRectangle> are exactly
 5 # the same as <drawRectangle>.  Even though
 6 # 6 solid rectangles are drawn, only 4 show
 7 # up on the screen.  Where are the other 2?
 8
 9
10 from Graphics import *
11
12 beginGrfx(1300,700)
13
14 fillRectangle(100,100,200,200)
15 fillRectangle(400,100,900,200)
16 fillRectangle(100,300,900,600)
17 fillRectangle(1000,100,1200,600)
18 fillRectangle(200,400,400,500)
19 fillRectangle(600,400,800,500)
20
21 endGrfx()
```

```
1  # Graphic
2  # This pr
3  # procedu
4  # argumen
5  # the sam
6  # 6 solid
7  # up on t
8
9
10 from Grap
11
12 beginGrfx
13
14 fillRectangle(100,100,200,200)
15 fillRectangle(400,100,900,200)
16 fillRectangle(100,300,900,600)
17 fillRectangle(1000,100,1200,600)
18 fillRectangle(200,400,400,500)
19 fillRectangle(600,400,800,500)
20
21 endGrfx()
```

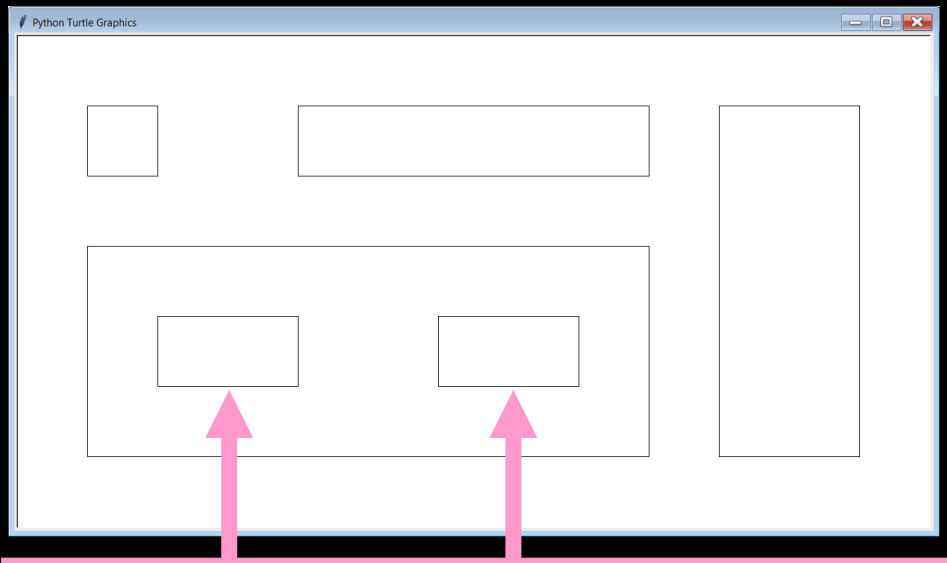
Python Turtle Graphics

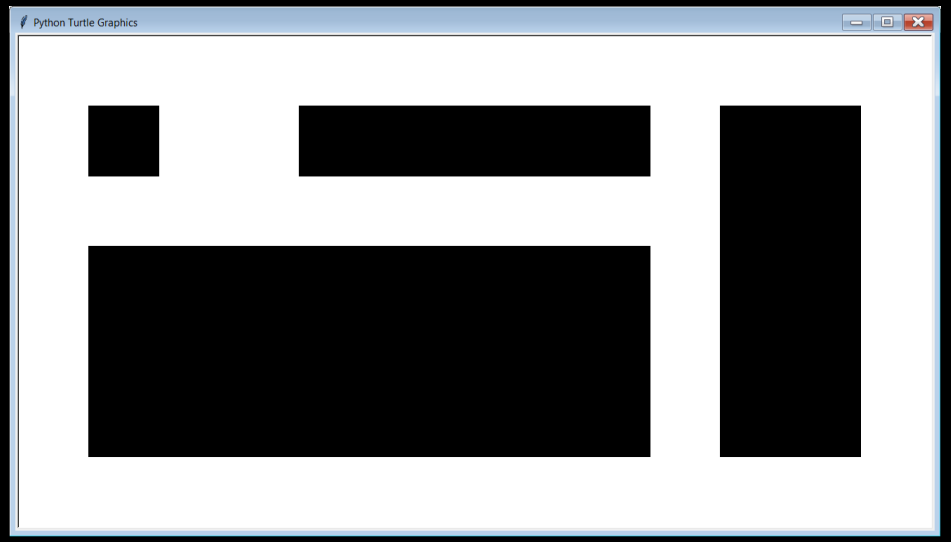
drawRectangle & fillRectangle

GraphicsLibrary03.py vs. GraphicsLibrary16.py

```
drawRectangle(100,100,200,200)
drawRectangle(400,100,900,200)
drawRectangle(100,300,900,600)
drawRectangle(1000,100,1200,600)
drawRectangle(200,400,400,500)
drawRectangle(600,400,800,500)
```

```
fillRectangle(100,100,200,200)
fillRectangle(400,100,900,200)
fillRectangle(100,300,900,600)
fillRectangle(1000,100,1200,600)
fillRectangle(200,400,400,500)
fillRectangle(600,400,800,500)
```

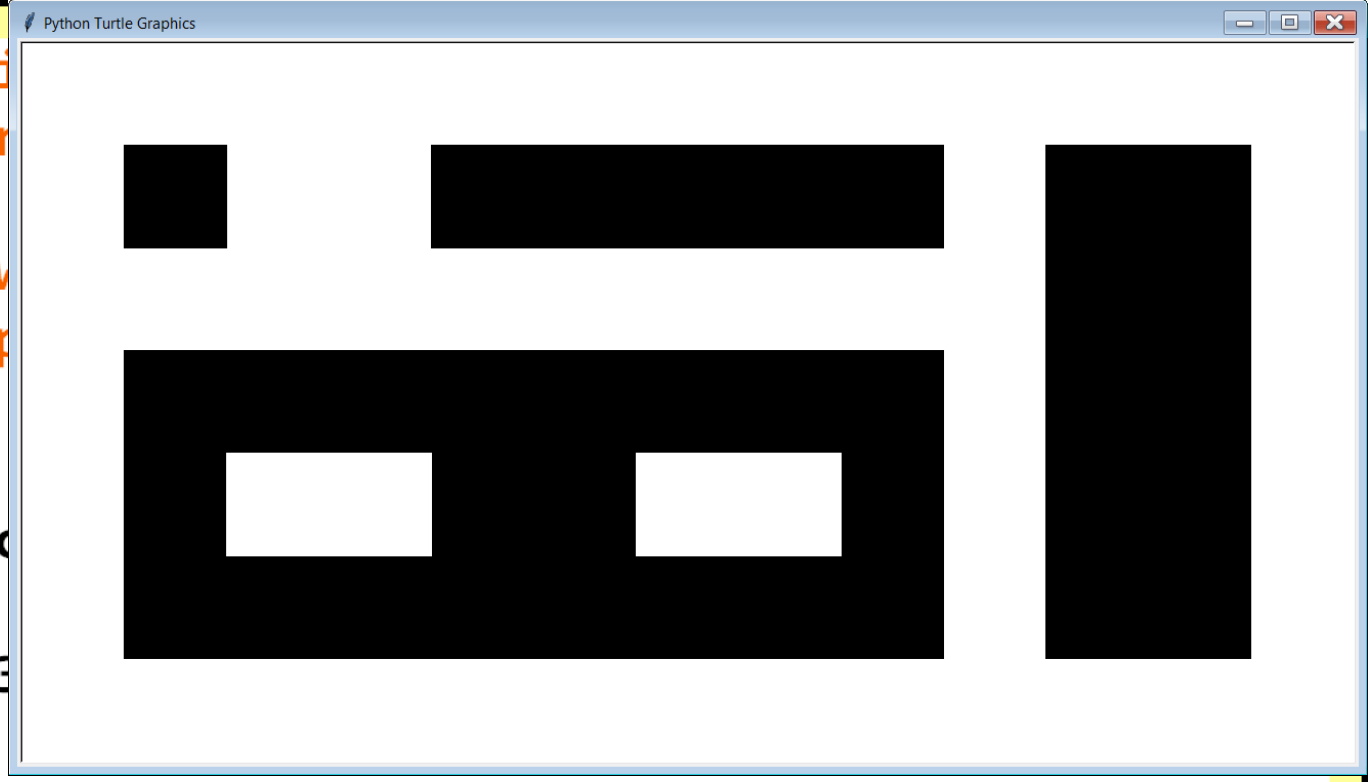




These 2 rectangles do not show up when filled because they are the same color as the rectangle behind them.

```
 1 # GraphicsLibrary17.py
 2 # This program demonstrates the <setColor>
 3 # procedure of the <Graphics> library.
 4 # Now the two missing rectangles from the
 5 # previous program are visible.
 6
 7
 8 from Graphics import *
 9
10 beginGrfx(1300,700)
11
12 fillRectangle(100,100,200,200)
13 fillRectangle(400,100,900,200)
14 fillRectangle(100,300,900,600)
15 fillRectangle(1000,100,1200,600)
16 setColor("white")
17 fillRectangle(200,400,400,500)
18 fillRectangle(600,400,800,500)
19
20 endGrfx()
```

```
 1 # GraphicsLi
 2 # This progr
 3 # procedure
 4 # Now the tw
 5 # previous p
 6
 7
 8 from Graphic
 9
10 beginGrfx(13
11
12 fillRectangle(100,100,200,200)
13 fillRectangle(400,100,900,200)
14 fillRectangle(100,300,900,600)
15 fillRectangle(1000,100,1200,600)
16 setColor("white")
17 fillRectangle(200,400,400,500)
18 fillRectangle(600,400,800,500)
19
20 endGrfx()
```



https://www.w3schools.com/colors/colors_names.asp

HTML Color Names

The 140 colors available in Python are listed here: https://www.w3schools.com/colors/colors_names.asp

Colors Tutorial

- Colors HOME
- Color Names
- Color Values
- Color Groups
- Color Shades
- Color Picker
- Color Mixer
- Color Converter
- Color RGB
- Color HEX
- Color HSL
- Color HWB
- Color CMYK
- Color NCol
- Color Gradient
- Color Theory
- Color Wheels
- Color Hues
- Color Schemes
- Color Palettes
- Color Brands
- Color W3.CSS

Color Names Supported by All Browsers

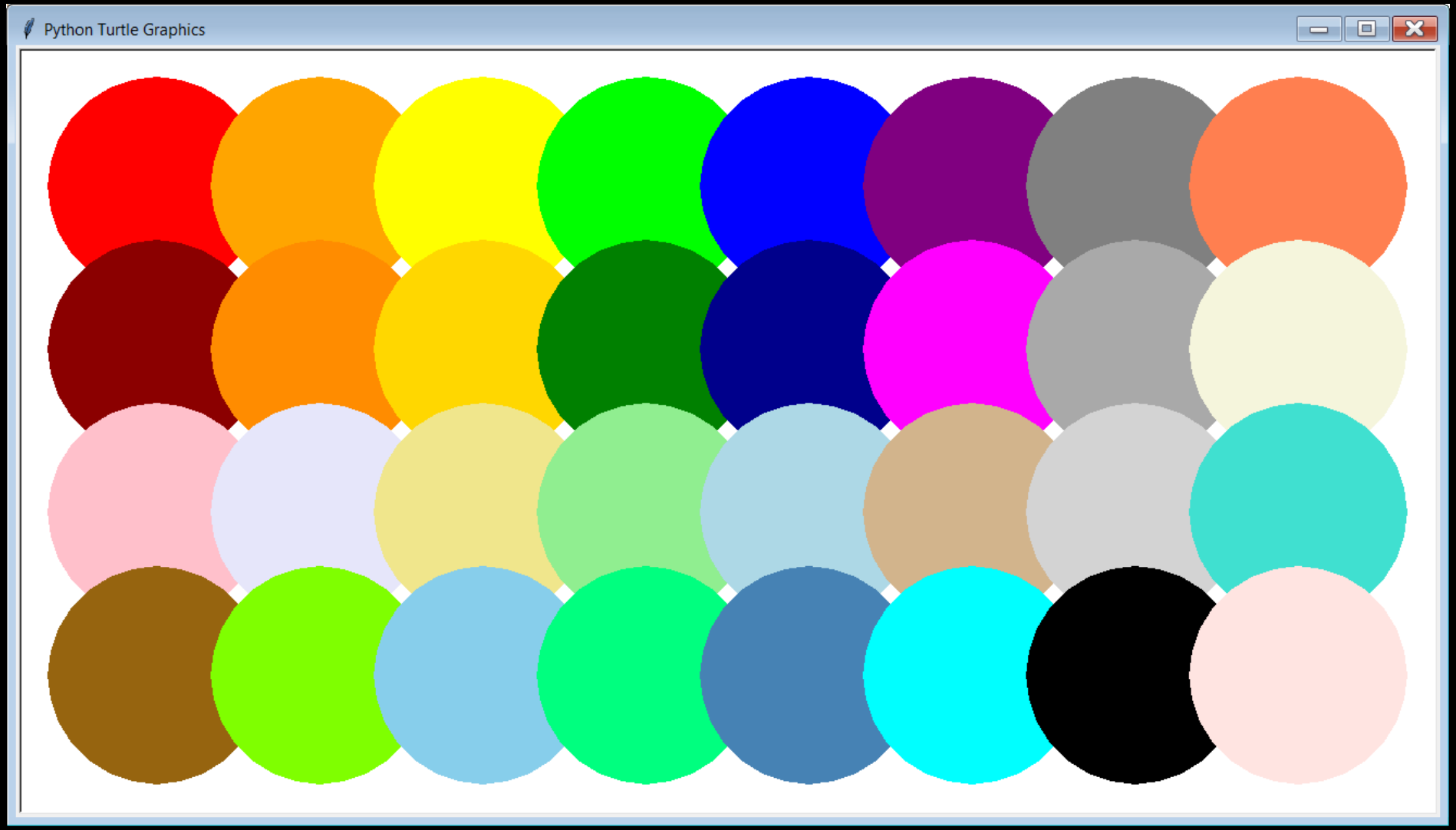
NOTE: Some of these colors, like Aqua, do not work in Python when using a Mac.

Color Name	HEX	Color	Shades	Mix
AliceBlue	#F0F8FF		Shades	Mix
AntiqueWhite	#FAEBD7		Shades	Mix
Aqua	#00FFFF		Shades	Mix
Aquamarine	#7FFFD4		Shades	Mix
Azure	#F0FFFF		Shades	Mix
Beige	#F5F5DC		Shades	Mix
Bisque	#FFE4C4		Shades	Mix
Black	#000000		Shades	Mix
BlanchedAlmond	#FFEBCD		Shades	Mix
Blue	#0000FF		Shades	Mix

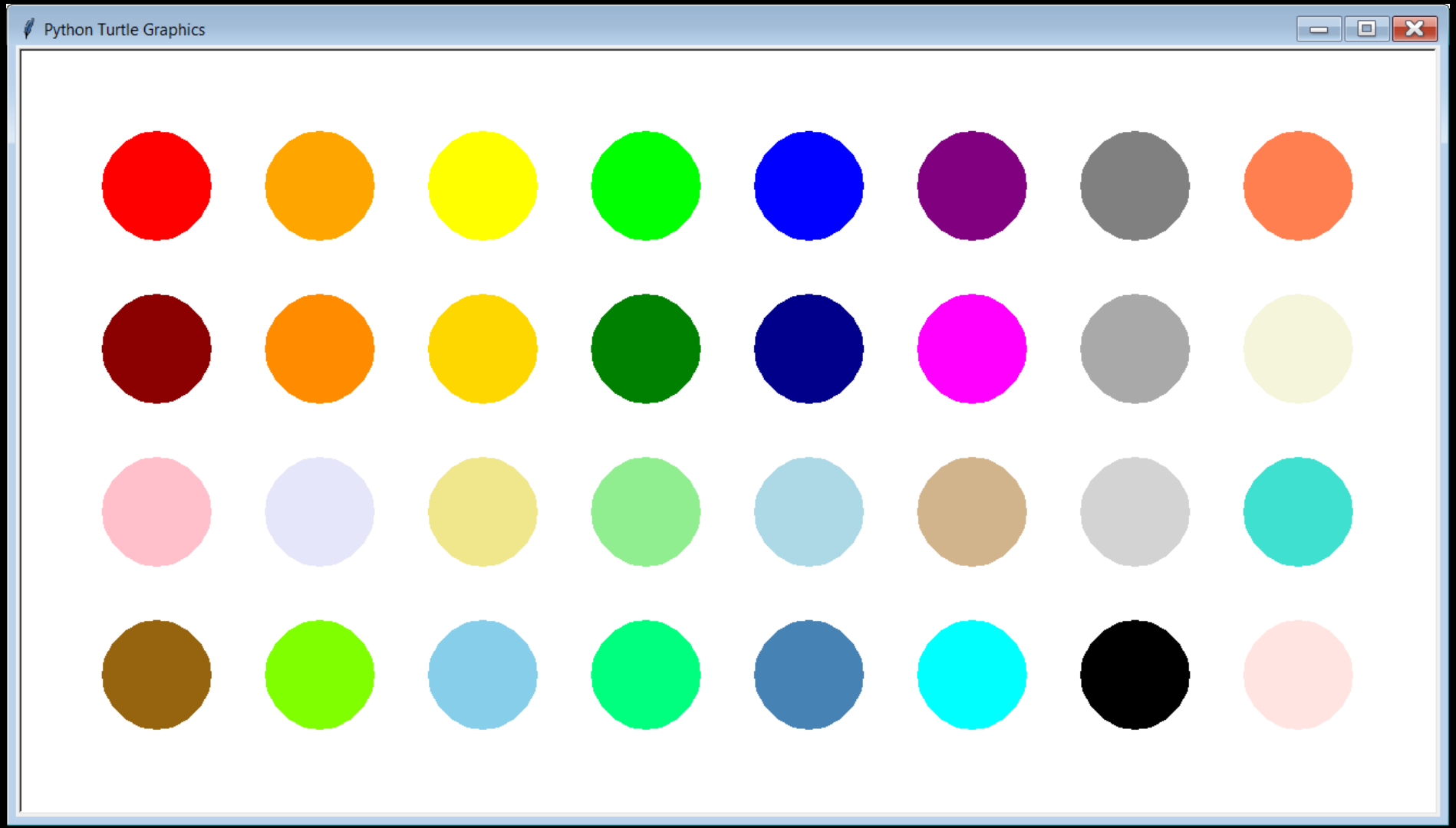
```
 1 # GraphicsLibrary18.py
 2 # This program demonstrates 32 of the colors
 3 # that are available in Python.  For a complete
 4 # list of all 140 colors, visit this website:
 5 # https://www.w3schools.com/colors/colors_names.asp
 6 # This program also demonstrates <fillCircle>.
 7 # It also shows that you can put 2 Python
 8 # commands on the same line if you end the
 9 # first with a semicolon <;>
10
11
12 from Graphics import *
13
14 beginGrfx(1300,700)
15
16 radius = 100
17
18 setColor("red");          fillCircle(125,125,radius)
19 setColor("orange");       fillCircle(275,125,radius)
20 setColor("yellow");       fillCircle(425,125,radius)
21 setColor("green");        fillCircle(575,125,radius)
22 setColor("blue");         fillCircle(725,125,radius)
23 setColor("purple");       fillCircle(875,125,radius)
24 setColor("gray");         fillCircle(1025,125,radius)
25 setColor("coral");        fillCircle(1175,125,radius)
```

```
27 setColor("dark red");       fillCircle(125,275,radius)
28 setColor("dark orange");    fillCircle(275,275,radius)
29 setColor("gold");           fillCircle(425,275,radius)
30 setColor("dark green");     fillCircle(575,275,radius)
31 setColor("dark blue");      fillCircle(725,275,radius)
32 setColor("magenta");        fillCircle(875,275,radius)
33 setColor("dark gray");      fillCircle(1025,275,radius)
34 setColor("beige");          fillCircle(1175,275,radius)
35
36 setColor("pink");           fillCircle(125,425,radius)
37 setColor("lavender");       fillCircle(275,425,radius)
38 setColor("khaki");          fillCircle(425,425,radius)
39 setColor("light green");    fillCircle(575,425,radius)
40 setColor("light blue");     fillCircle(725,425,radius)
41 setColor("tan");            fillCircle(875,425,radius)
42 setColor("light gray");     fillCircle(1025,425,radius)
43 setColor("turquoise");      fillCircle(1175,425,radius)
44
45 setColor("brown");          fillCircle(125,575,radius)
46 setColor("chartreuse");     fillCircle(275,575,radius)
47 setColor("sky blue");       fillCircle(425,575,radius)
48 setColor("spring green");   fillCircle(575,575,radius)
49 setColor("steel blue");     fillCircle(725,575,radius)
50 setColor("cyan");           fillCircle(875,575,radius)
51 setColor("black");          fillCircle(1025,575,radius)
52 setColor("misty rose");     fillCircle(1175,575,radius)
53
54 endGrfx()
```

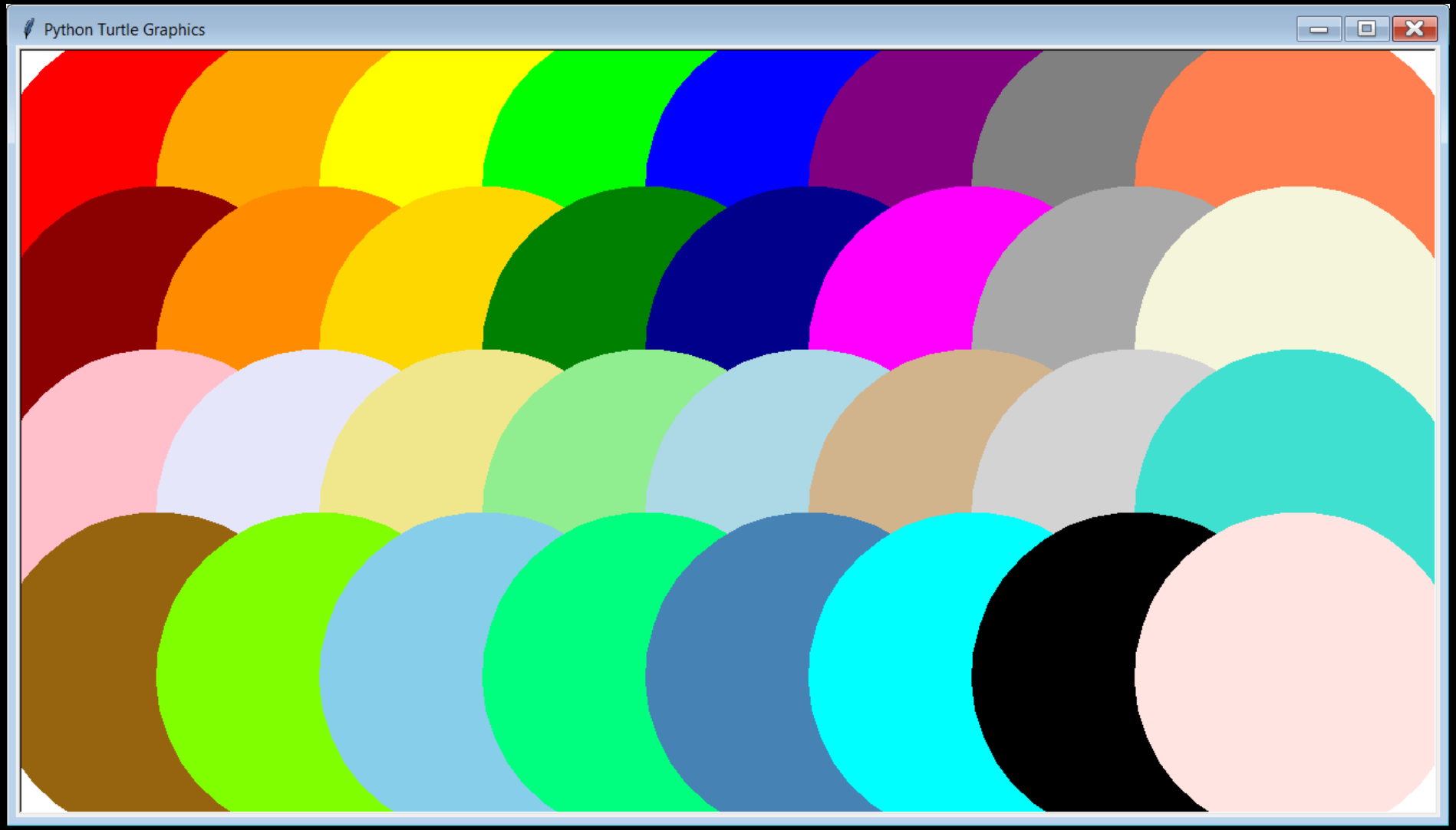
GraphicsLibrary18.py
Current output with radius = 100



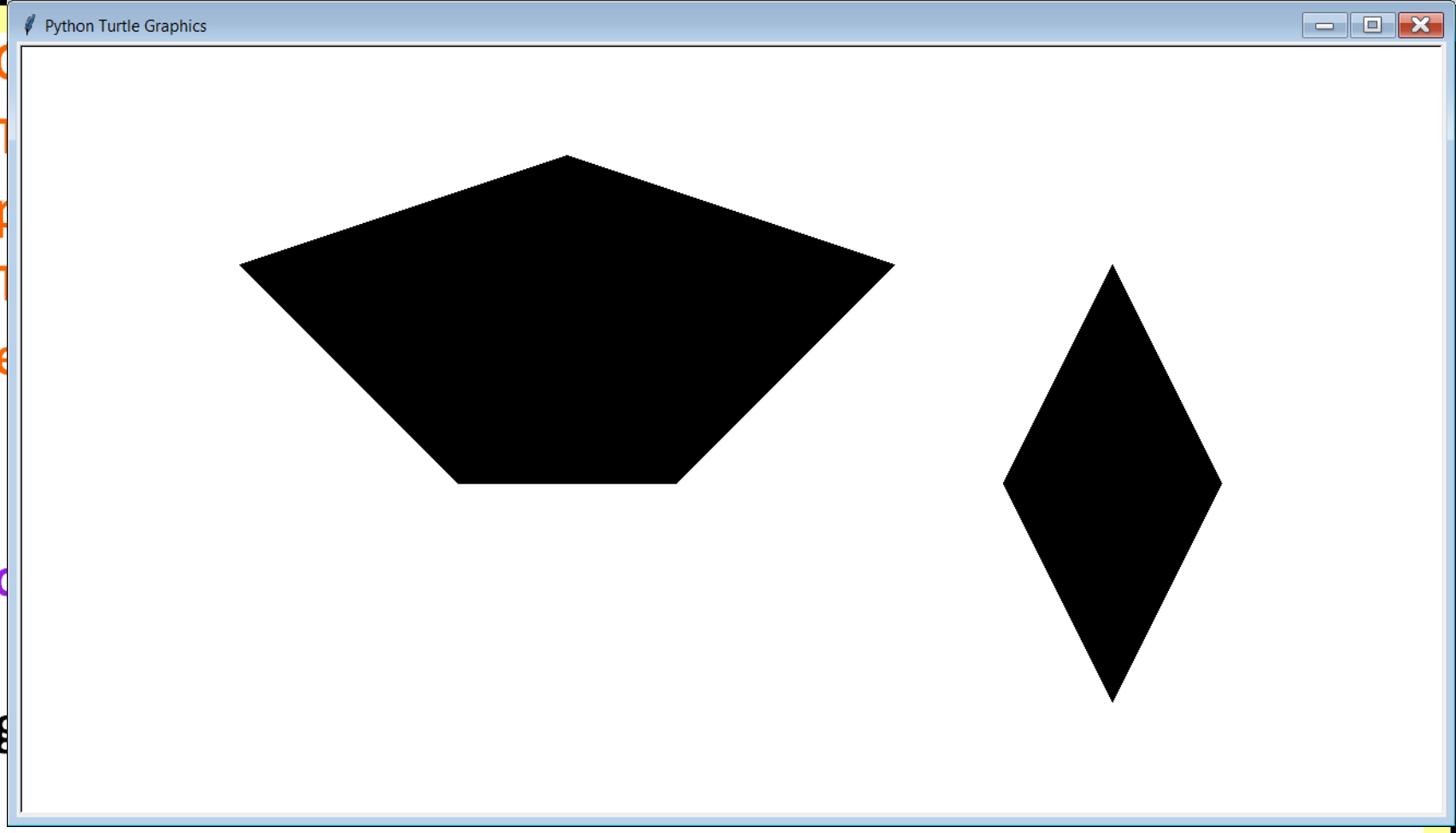
GraphicsLibrary18.py
Output if radius = 50



GraphicsLibrary18.py
Output if radius = 150



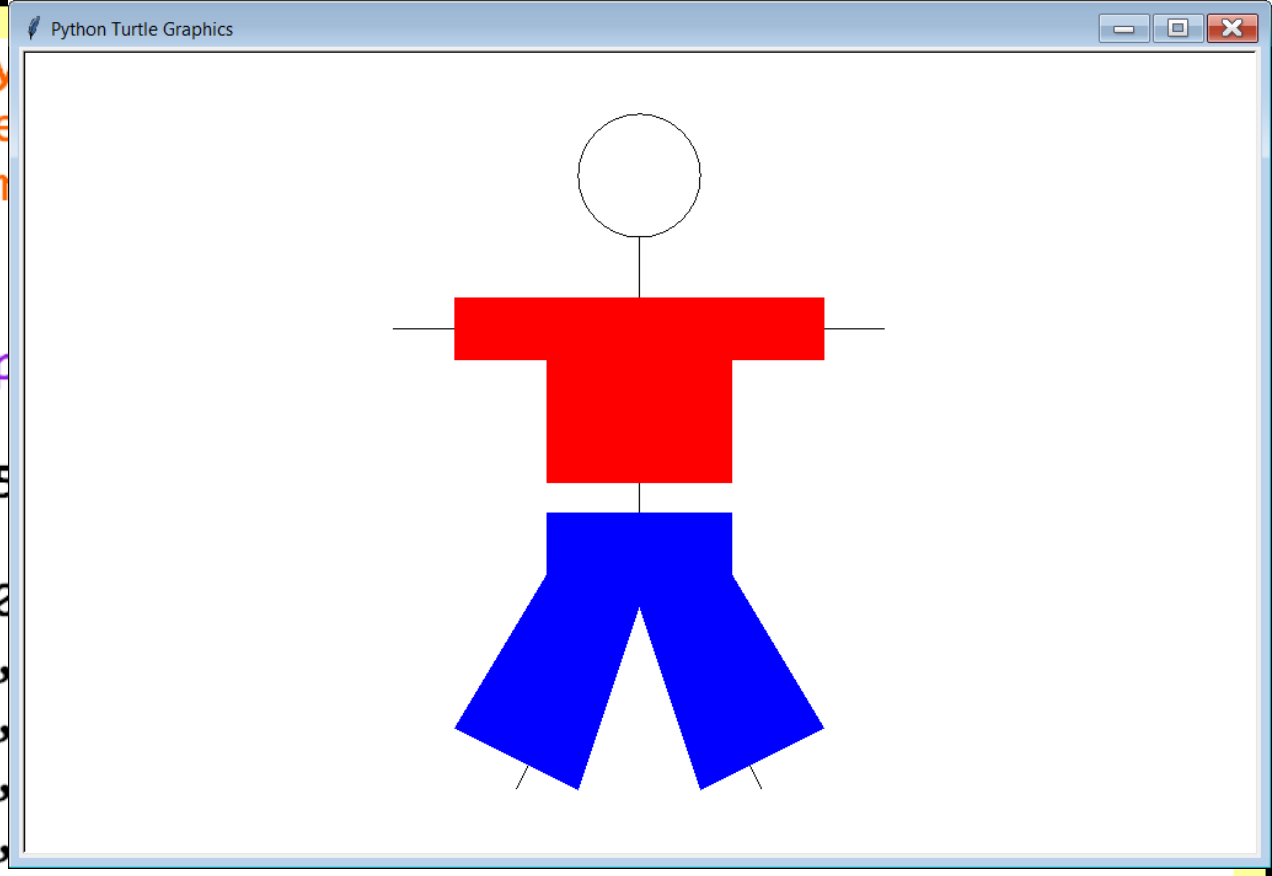
```
1  # GraphicsLibrary19.py
2  # This program demonstrates the <fillPolygon>
3  # procedure of the <Graphics> library.
4  # The arguments for <fillPolygon> are
5  # exactly the same as for <drawPolygon>.
6
7
8  from Graphics import *
9
10 beginGrfx(1300,700)
11
12 fillPolygon([500,100,800,200,600,400,400,400,200,200])
13 fillPolygon([900,400,1000,200,1100,400,1000,600])
14
15 endGrfx()
16
```




```
1  # C
2  # T
3  # p
4  # T
5  # e
6
7
8  fro
9
10 beg
11
12 fillPolygon([500,100,800,200,600,400,400,400,200,200])
13 fillPolygon([900,400,1000,200,1100,400,1000,600])
14
15 endGrfx()
16
```

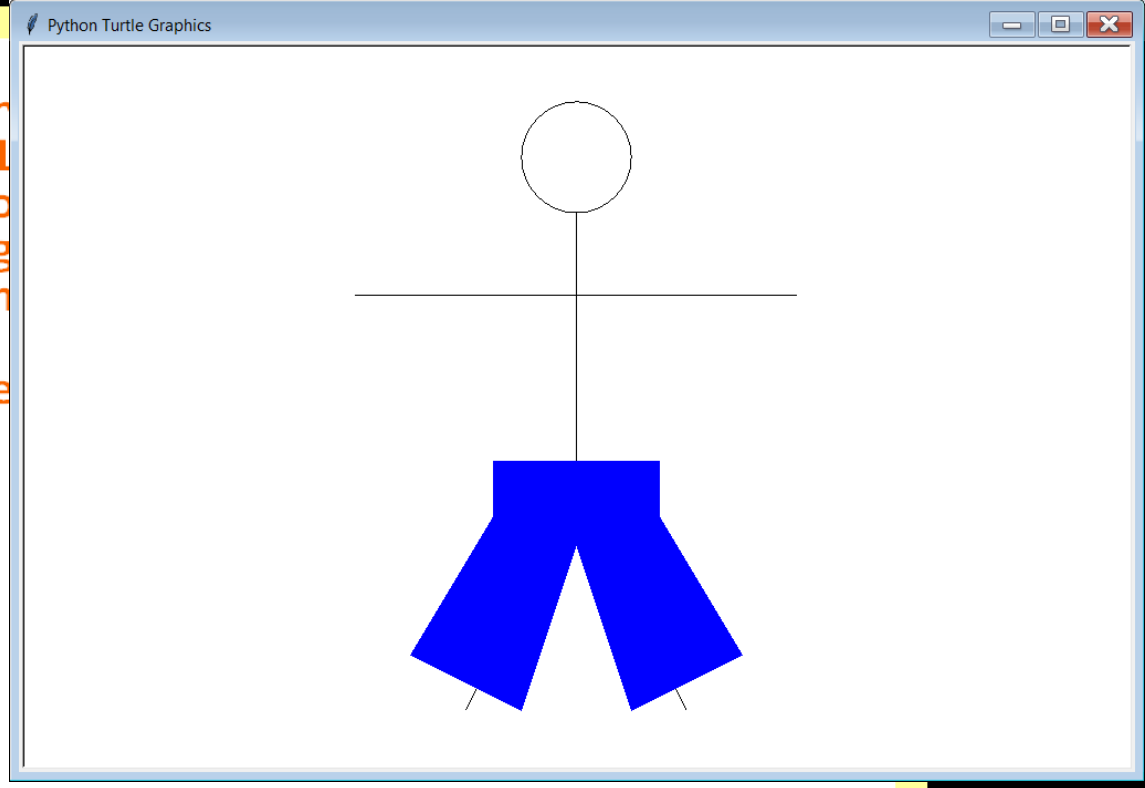
```
 1 # GraphicsLibrary20.py
 2 # This program demonstrates that the different polygons
 3 # in your program can be filled with different colors.
 4
 5
 6 from Graphics import *
 7
 8 beginGrfx(1000,650)
 9
10 drawCircle(500,100,50)
11 drawLine(500,150,500,400)
12 drawLine(500,400,400,600)
13 drawLine(500,400,600,600)
14 drawLine(300,225,700,225)
15 setColor("blue")
16 fillPolygon([425,375,425,425,350,550,450,600,
500,450,550,600,650,550,575,425,575,375])
17 setColor("red")
18 fillPolygon([350,200,650,200,650,250,575,250,
575,350,425,350,425,250,350,250])
19
20 endGrfx()
```

```
1  # GraphicsLibrary
2  # This program de
3  # in your program
4
5
6  from Graphics imp
7
8  beginGrfx(1000,65
9
10 drawCircle(500,10
11 drawLine(500,150,
12 drawLine(500,400,
13 drawLine(500,400,
14 drawLine(300,225,
15 setColor("blue")
16 fillPolygon([425,375,425,425,350,550,450,600,
500,450,550,600,650,550,575,425,575,375])
17 setColor("red")
18 fillPolygon([350,200,650,200,650,250,575,250,
575,350,425,350,425,250,350,250])
19
20 endGrfx()
```



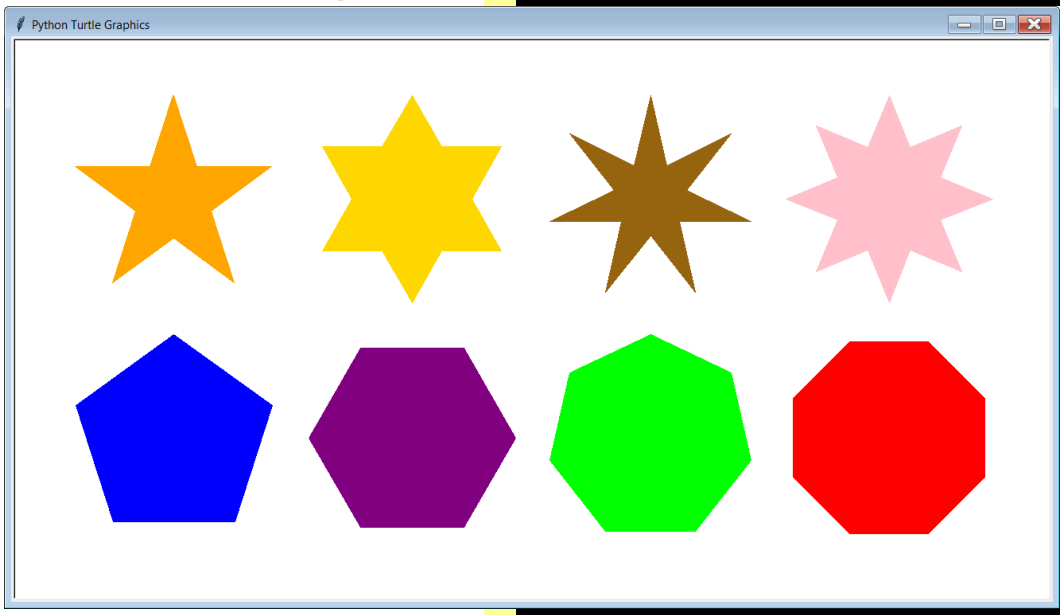
```
 1 # GraphicsLibrary21.py
 2 # This program demonstrates what happens when
 3 # <drawPolygon> or <fillPolygon> is called with
 4 # an incorrect number of integer arguments.
 5 # Since the integer arguments represent
 6 # coordinate points, the number of integer arguments
 7 # must always be even.  Since a polygon must have
 8 # at least 3 sides, the number of integer arguments
 9 # must be at least 6.  If either condition is not
10 # met, a special error message is displayed.
11
12
13 from Graphics import *
14
15 beginGrfx(1000,650)
16
17 drawCircle(500,100,50)
18 drawLine(500,150,500,400)
19 drawLine(500,400,400,600)
20 drawLine(500,400,600,600)
21 drawLine(300,225,700,225)
22 setColor("blue")
23 fillPolygon([425,375,425,425,350,550,450,600,500,450,
550,600,650,550,575,425,575,375])
24 setColor("red")
25 fillPolygon([350,200,650,200,650,250,575,250,575,350,
425,350,425,250,350])
26
27 endGrfx()
```

```
 1 # GraphicsLibrary21.py
 2 # This program demonstr
 3 # <drawPolygon> or <fil
 4 # an incorrect number o
 5 # Since the integer arg
 6 # coordinate points, th
 7 # must always be even.
 8 # at least 3 sides, the
 9 # must be at least 6.
10 # met, a special error
11
12
13 from Graphics import *
14
15 beginGrfx(1000,650)
16
17 drawCircle(500,
18 drawLine(500,15
19 drawLine(500,40
20 drawLine(500,40
21 drawLine(300,22
22 setColor("blue"
23 fillPolygon([42
550,600,650,550,57
24 setColor("red")
25 fillPolygon([350,200,650,200,650,250,575,250,575,350,
425,350,425,250,350])
26
27 endGrfx()
```

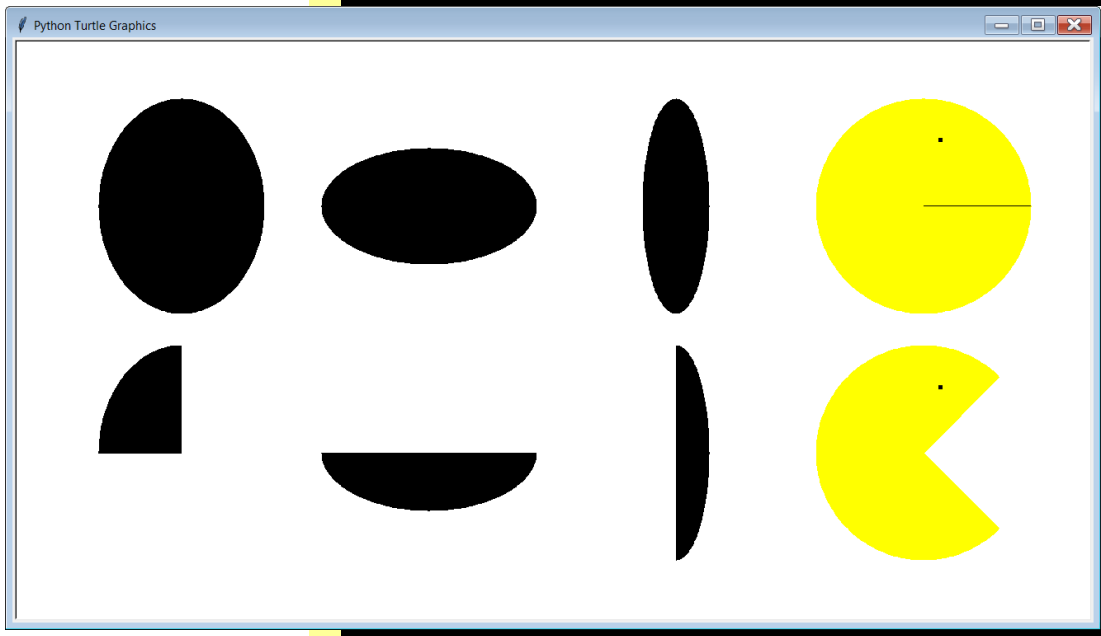



```
 ----jGRASP exec: python GraphicsLibrary21.py
Polygon Error in line 25
When using drawPolygon/fillPolygon
you must have an even # of int arguments.

 ----jGRASP: operation complete.
```

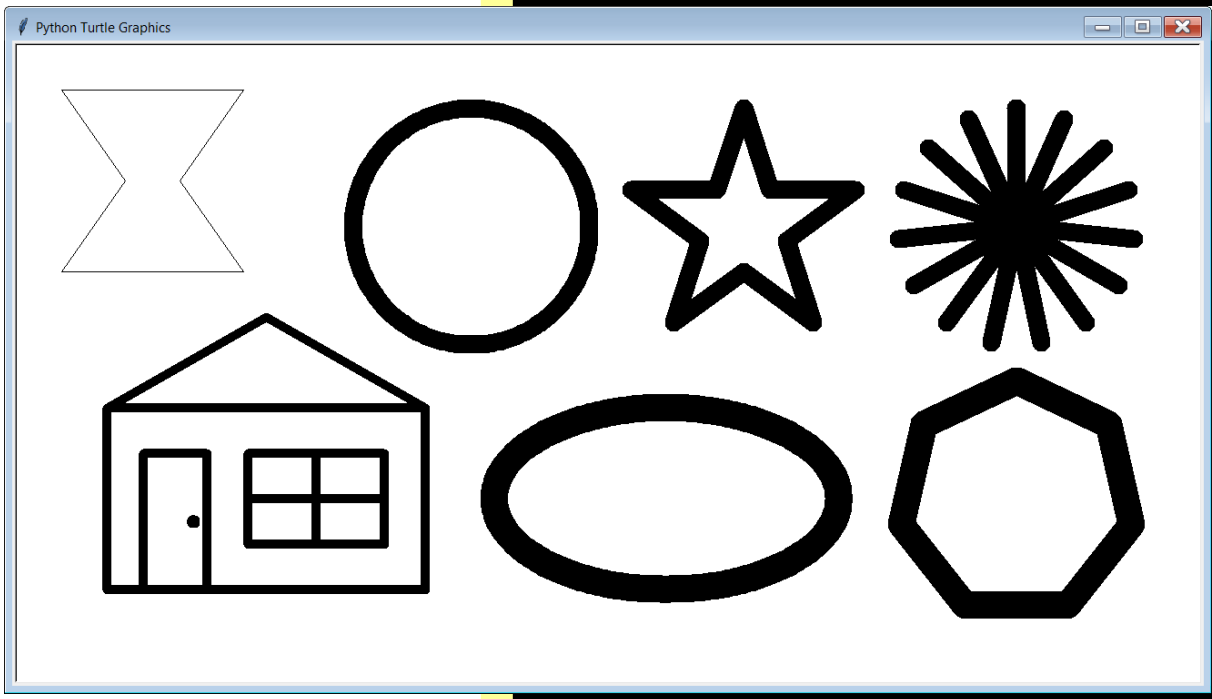
```python
# GraphicsLibrary22.py
# This program demonstrates the <fillRegularPolygon>
# and <fillStar> procedures.  Both of these have the
# same exact arguments as their "draw" counterparts


from Graphics import *

beginGrfx(1300,700)

setColor("orange")
fillStar(200,200,130,5)
setColor("gold")
fillStar(500,200,130,6)
setColor("brown")
fillStar(800,200,130,7)
setColor("pink")
fillStar(1100,200,130,8)
setColor("blue")
fillRegularPolygon(200,500,130,5)
setColor("purple")
fillRegularPolygon(500,500,130,6)
setColor("green")
fillRegularPolygon(800,500,130,7)
setColor("red")
fillRegularPolygon(1100,500,130,8)

endGrfx()
```



```
1  # GraphicsLibrary23.py
2  # This program demonstrates <fillOval> & <fillArc>.
3  # In the same way that an arc is a piece of an oval
4  # a "filled arc" is a piece of a "filled oval".
5
6
7  from Graphics import *
8
9  beginGrfx(1300,700)
10
11 fillOval(200,200,100,130)
12 fillOval(500,200,130,70)
13 fillOval(800,200,40,130)
14 setColor("yellow")
15 fillOval(1100,200,130,130)
16 setColor("black")
17 drawLine(1100,200,1230,200)
18 drawPoint(1120,120)
19 fillArc(200,500,100,130,270,360)
20 fillArc(500,500,130,70,90,270)
21 fillArc(800,500,40,130,0,180)
22 setColor("yellow")
23 fillArc(1100,500,130,130,135,45)
24 setColor("black")
25 drawPoint(1120,420)
26
27 endGrfx()
```



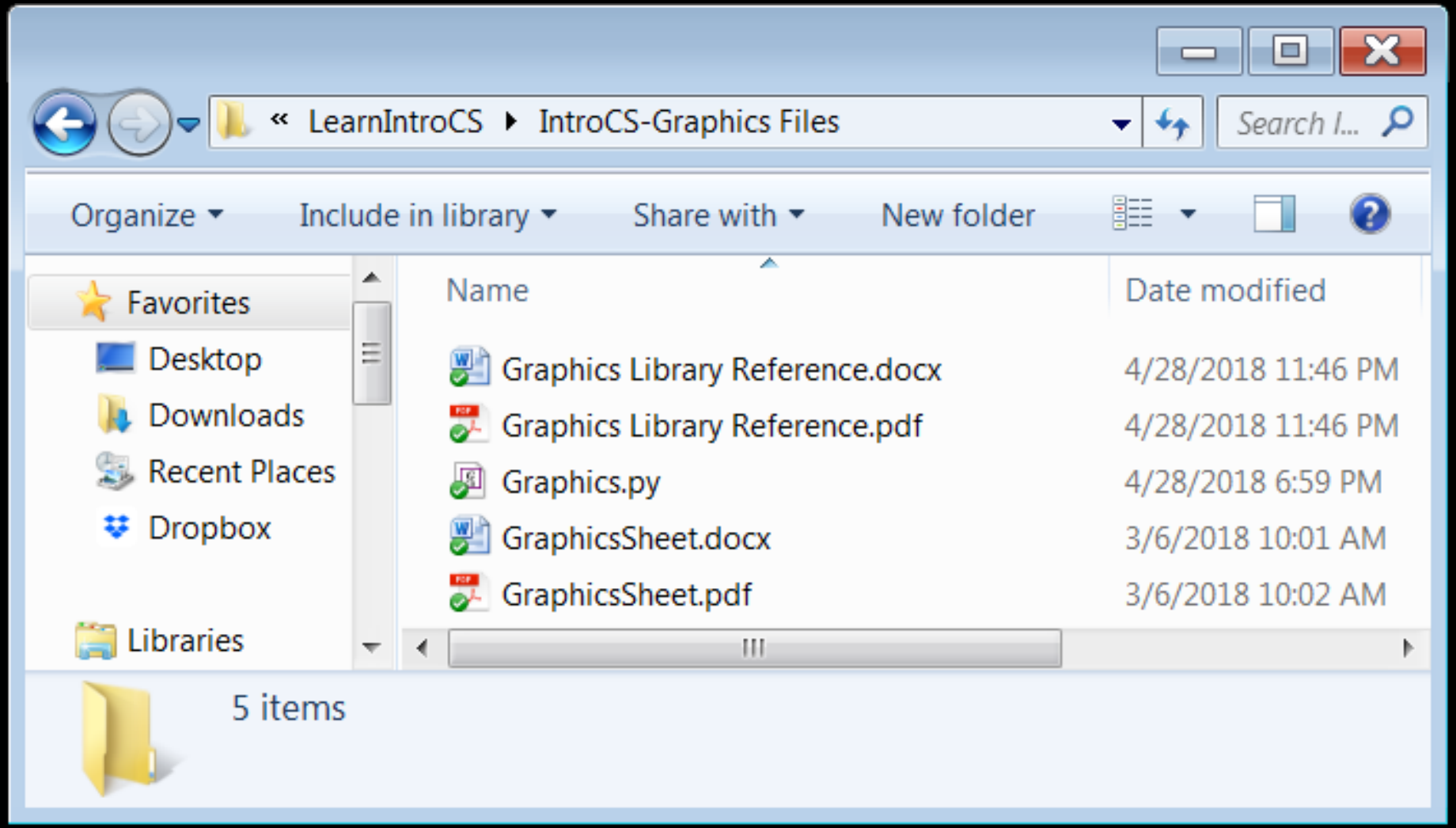
```
1  # GraphicsLibrary24.py
2  # This program demonstrates that the <width>
3  # command from "Turtle Graphics" can also be
4  # used with commands from the <Graphics> library.
5
6
7  from Graphics import *
8
9  beginGrfx(1300,700)
10
11 width(10)
12 drawRectangle(100,400,450,600)
13 drawLine(100,400,275,300)
14 drawLine(450,400,275,300)
15 drawRectangle(255,450,405,550)
16 drawLine(330,450,330,550)
17 drawLine(255,500,405,500)
18 drawRectangle(140,450,210,600)
19 drawPoint(195,525)
20
21 width(20)
22 drawCircle(500,200,130)
23 drawStar(800,200,130,5)
24 drawBurst(1100,200,130,15)
25
26 width(30)
27 drawOval(715,500,190,100)
28 drawRegularPolygon(1100,500,130,7)
29
30 width(1) # back to default
31 drawPolygon([50,50,250,50,180,150,250,250,50,250,120,150])
```



Section 6.8

Graphics Library Reference Information

Graphics Library Reference (Location)



Graphics Library Reference (Legal Stuff)

Mr. Schram's Graphics Library for Python

This version of the **Graphics** library was written by Mr. John Schram 10/3/18 for use in first year Computer Science 1 or Computer Science 1-Honors / PreAPCS.

While built on "Turtle Graphics", the procedures and functions below allow graphics programming with more "Traditional Graphics" commands for greater convenience. It was inspired by a similar graphics library created by Mr. Leon Schram and is designed to operate in a manner similar to that of the **Expo** class that we created for "Exposure Java".

Graphics Library Reference (First Few Procedures)

beginGrfx(windowWidth, windowHeight)

This will create a graphics window whose size is determined by the parameters.
This is always the first command used right after the **Graphics** library is imported.
Example:

```
from Graphics import *
beginGrfx(1300,700)
```

This creates a graphics window that is 1300 pixels wide and 700 pixels tall.

delay(milliseconds)

Updates the graphics window and makes the computer pause for a certain number of milliseconds.
Examples:

```
delay(1000)     # pause for 1 second
delay(2000)     # pause for 2 seconds
delay(500)      # pause for 1/2 of a second
```

drawArc(centerX, centerY, hRadius, vRadius, start, finish)

Draws and arc which looks like a curve.
An ARC is a "piece" of an OVAL.
The first 4 parameters are the same as **drawOval**.
There are 2 additional parameters for the starting degree value and finishing degree of the arc.
0 degrees is at the 12:00 position and the degrees progress in a CLOCKWISE fashion.
(90 degrees is at 3:00, 180 degrees is at 6:00, 270 degrees is at 9:00, 360 degrees is back at 12:00).
Example:

```
drawArc(300,200,100,100,135,45)
```

Draws an open arc which is a 3/4 piece of a circle with a radius of 100 pixels whose center is located at coordinate (300,200).
This arc will resemble the letter "C".

Things to Remember about the Graphics Library

- The **Graphics** library is not part of Python.
- This library was created to allow graphics programming with more traditional, *coordinate-based* graphics commands for greater convenience.
- In order to use the **Graphics** library, the file **Graphics.py** must be in the same folder/directory as the Python file that calls the **Graphics** library subroutines.
- Students will NOT be required to memorize the functions and procedures of the **Graphics** library. They will instead be provided with **Graphics Library Reference** documentation to use during labs and tests.