Chapter 2: Drawing Basic Shapes.

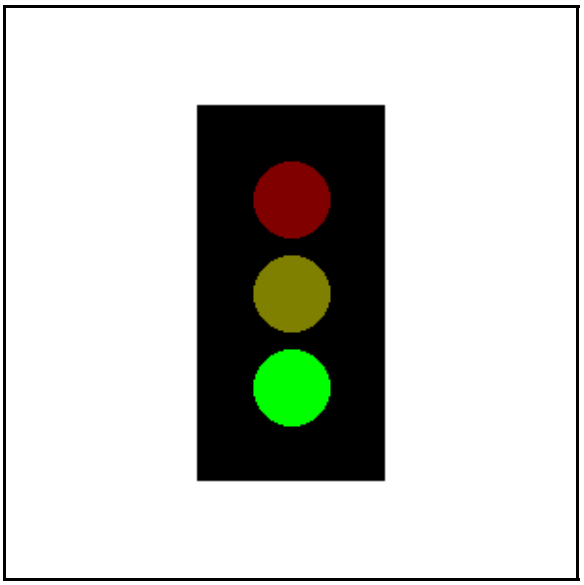
In this chapter we will be getting graphical. You will learn how to draw rectangles, circles, lines and points of various colors. These programs will get more and more complex, so you will also learn how to save your programs to long term storage and how to load them back in so you can run them again or change them.

Drawing Rectangles and Circles:

Let's start the graphics off by writing a graphical program that will draw a traffic light, specifically a green light.

```
1     # traffic_light.kbs
2     # Show a traffic light and say a message.
3
4     clg
5
6     color black
7     rect 100,50,100,200
8
9     color darkred
10    circle 150,100,20
11
12    color darkyellow
13    circle 150,150,20
14
15    color green
16    circle 150,200,20
17
18    say "Green light. You may go."
```

Program 9: Traffic Light



Sample Output 9: Traffic Light

Let's go line by line through the program above. The first and second lines are called remark or comment statements. A remark is a place for the programmer to place comments in their computer code that are ignored by the BASIC-256. They are a good place to describe what complex blocks of code is doing, the program's name, why we wrote a program, or who the programmer was.


```
#
rem
```

The **#** and **rem** statements are called remarks. A remark statement allows the programmer to put comments about the code they are working on into the program. The computer sees the **#** or **rem** statement and will ignore all of the rest of the text on the line.

On line four you see the **clg** statement. It is much like the **cls** statement from Chapter 1, except that the **clg** statement will clear the graphic output area of the screen.


```
clg
clg color_name
clg rgb( red, green, blue )
```

The **clg** statement erases the graphics output area so that we have a clean place to do our drawings.

You may optionally define a color after the **clg** statement and it will set the entire graphics output window to that color.

Lines six, nine, twelve, and fifteen contain the simple form of the **color** statement. It tells BASIC-256 what color to use for the next drawing action. You may define colors either by using one of the eighteen standard color names or you may create one of over 16 million different colors by mixing the primary colors of light (red, green, and blue) together.

When you are using the numeric method to define your custom color be sure to limit the values from 0 to 255. Zero (0) represents no light of that component color and 255 means to shine the maximum. Bright white is represented by 255, 255, 255 (all colors of light) where black is represented by 0, 0, 0 (no colors at all). This numeric representation is known as the RGB triplet. Illustration 3 shows the named colors and their RGB values.


```
color color_name
color rgb( red, green, blue )
```

color can also be spelled **colour**.

The **color** statement allows you to set the color that will be drawn next. You may follow the **color** statement with a color name (black, white, red, darkred, green, darkgreen, blue, darkblue, cyan, darkcyan, purple, darkpurple, yellow, darkyellow, orange, darkorange, grey/gray, darkgrey/darkgray). You may also specify over 16 million different colors using the RGB() function by specifying how much red, blue, and green should be used.

Color Name and RGB Values	Color Name and RGB Values
black (0,0,0)	white (255,255,255)
red (255,0,0)	darkred (128,0,0)
Green (0,255,0)	darkgreen (0,128,0)
blue (0,0,255)	darkblue (0,0,128)
cyan (0,255,255)	darkcyan (0,128,128)
purple (255,0,255)	darkpurple (128,0,128)
yellow (255,255,0)	darkyellow (128,128,0)
orange (255,102,0)	darkorange (170,51,0)
grey/gray (164,164,164)	darkgrey/darkgray (128,128,128)

Illustration 3: Color Names

The graphics display area, by default is 300 pixels wide (x) by 300 pixels high (y). A pixel is the smallest dot that can be displayed on your computer monitor. The top left corner is the origin (0,0) and the bottom right is (299,299). Each pixel can be represented by two numbers, the first (x) is how

far over it is and the second (y) represents how far down. This way of marking points is known as the Cartesian Coordinate System to mathematicians.

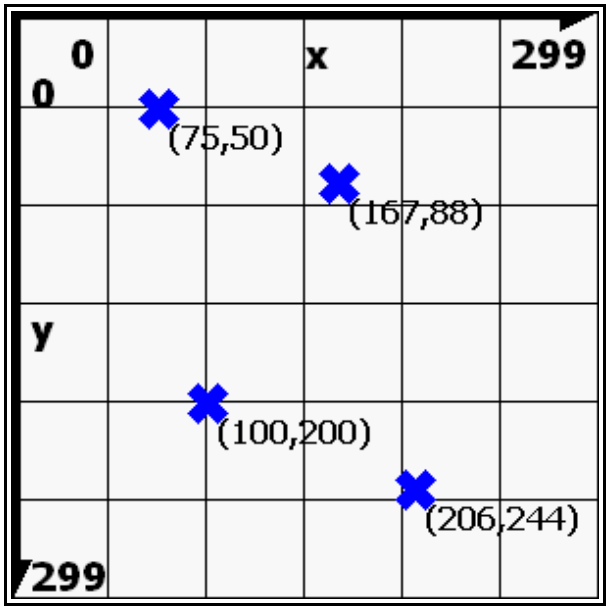


Illustration 4: The Cartesian Coordinate System of the Graphics Output Area

Explore

You can display grid lines on the Graphics Output Area of the screen by checking the "Graphics Window Grid Lines" option on the View menu.

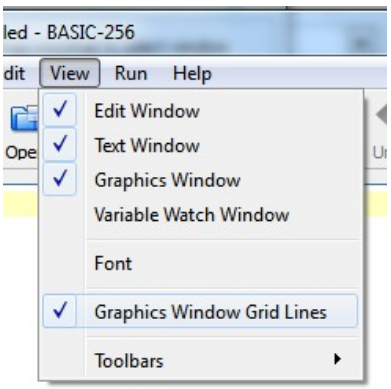


Illustration 5: Grid Lines Menu Option

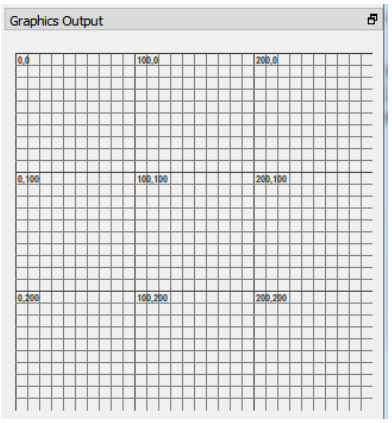


Illustration 6: Graphics Output Grid Lines

The next statement we will discuss (line 7) is **rect**. It is used to draw rectangles on the screen. It takes four numbers separated by commas; (1) how far over the left side of the rectangle is from the left edge of the graphics area, (2) how far down the top edge is, (3) how wide and (4) how tall. All four numbers are expressed in pixels (the size of the smallest dot that can be displayed).

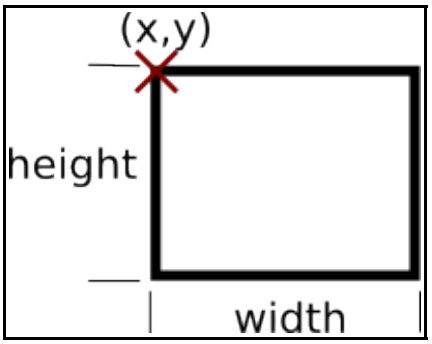


Illustration 7: Rectangle

You can see that the rectangle in the program starts at the point (100,50), is 100 pixels wide and 200 pixels tall.


```
rect x, y, width, height
```

The **rect** statement uses the current drawing color and places a rectangle on the graphics output window. The top left corner of the rectangle is specified by the first two numbers and the width and height is specified by the other two arguments.

Lines 10, 13 and 16 of Program 9 introduce the **circle** statement to draw a circle. It takes three numeric arguments, the first two represent the Cartesian coordinates for the center of the circle and the third the radius in pixels.

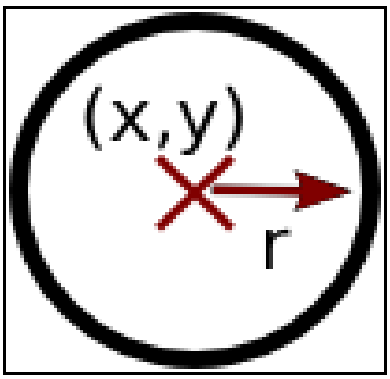


Illustration 8: Circle


```
circle x, y, radius
```

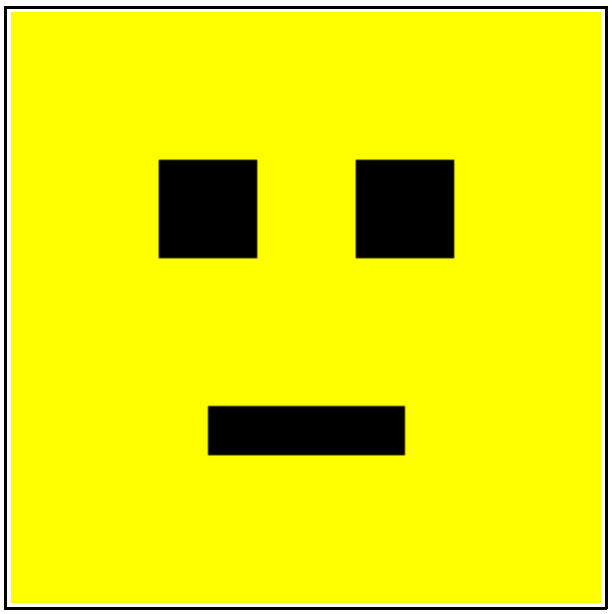
The **circle** statement uses the current drawing color and draws a filled circle with its center at (x, y) with the specified radius.

Some Other Programs Using Circles and Rectangles

Here are a couple of sample programs that use the new statements **clg**, **color**, **rect** and **circle**. Type the programs in and modify them. Make them a frowning face, alien face, or look like somebody you know.

```
1     # rectanglesmile.kbs
2
3     # make the screen yellow
4     clg yellow
5
6     # draw the mouth
7     color black
8     rect 100,200,100,25
9
10    # put on the eyes
11    color black
12    rect 75,75,50,50
13    rect 175,75,50,50
14
15    say "Hello."
```

Program 10: Face with Rectangles



Sample Output 10: Face with Rectangles

```
1     # circlesmile.kbs
2
3     # clear the screen
4     clg white
5
6     # draw the face
7     color yellow
8     circle 150,150,150
9
10    # draw the mouth by drawing a big black circle
11    # and then covering up the to part to leave
12    # a smile
13    color black
14    circle 150,200,70
15    color yellow
16    circle 150,150,70
17
```

```
18    # draw the eyes
19    color black
20    circle 100,100,30
21    circle 200,100,30
```

Program 11: Smiling Face with Circles

Sample Output 11: Smiling Face with Circles

Saving Your Program and Loading it Back:

Now that the programs are getting more complex, you may want to save them so that you can load them back in the future.

You may store a program by using the Save button  on the tool bar or Save option on the File menu. A dialog will display asking you for a file name, if it is a new program, or will save the changes you have made (replacing the old file).

If you do not want to replace the old version of the program and you want to store it using a new name you may use the Save As option on the File menu to save a copy with a different name.

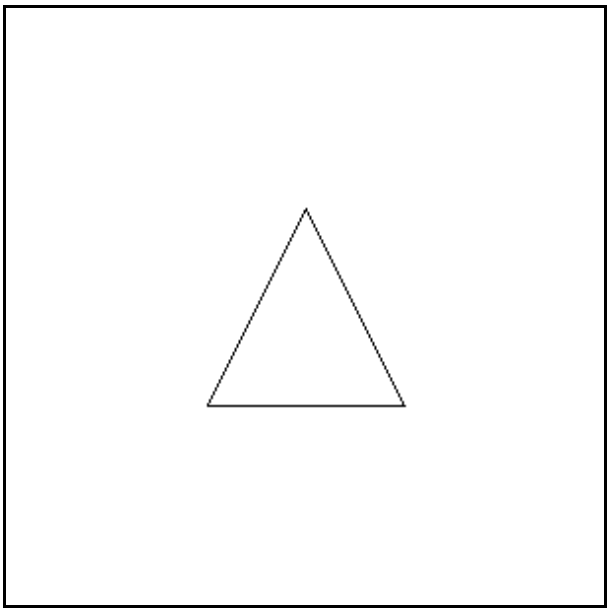
To load a previously saved program you would use the Open button  on the tool bar or the Open option on the File menu.

Drawing with Lines:

The next drawing statement is **line**. It will draw a line one pixel wide, of the current color, from one point to another point. Program 12 shows an example of how to use the **line** statement.

```
1     # triangle.kbs - draw a triangle
2
3     clg
4
5     color black
6     line 150, 100, 100, 200
7     line 100, 200, 200, 200
8     line 200, 200, 150, 100
```

Program 12: Draw a Triangle



Sample Output 12: Draw a Triangle

`line start_x, start_y, finish_x, finish_y`

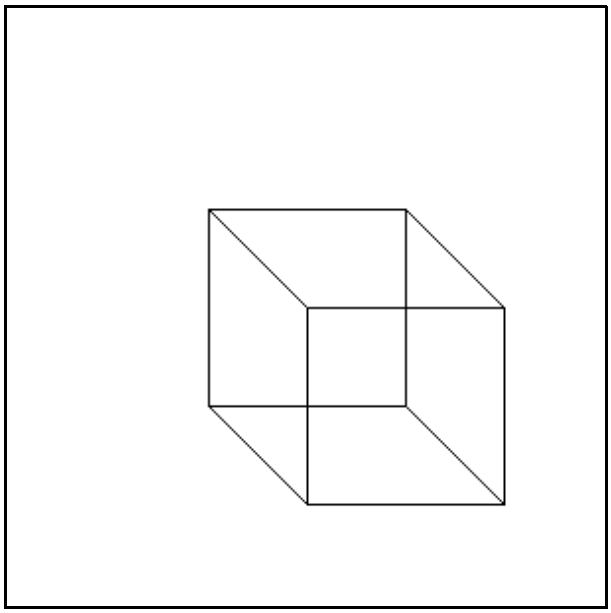
Draw a line one pixel wide from the starting point to the ending point, using the current color.

The next program is a sample of what you can do with many lines. It draws a cube on the screen.

```
1     # cube.kbs - draw a cube
2
3     clg
4     color black
```

```
5
6     # draw back square
7     line 150, 150, 150, 250
8     line 150, 250, 250, 250
9     line 250, 250, 250, 150
10    line 250, 150, 150, 150
11
12    # draw front square
13    line 100, 100, 100, 200
14    line 100, 200, 200, 200
15    line 200, 200, 200, 100
16    line 200, 100, 100, 100
17
18    # connect the corners
19    line 100, 100, 150, 150
20    line 100, 200, 150, 250
21    line 200, 200, 250, 250
```

Program 13: Draw a Cube



Sample Output 13: Draw a Cube

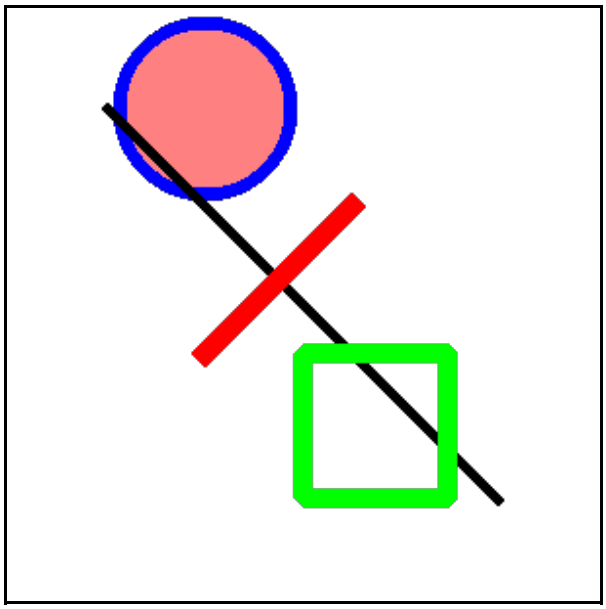
Setting Line Width and Drawing Shape Borders:

By default the width of a line drawn in BASIC256 is one pixel (dot) wide. The **penwidth** statement can be used to change the way lines (and borders around shapes) are drawn.

The following program will illustrate the **penwidth** statement, a more complex use of the **color** statement and an example of the special color **clear**.

```
1     # shapeoutline.kbs
2     # draw shapes with an outline
3
4     clg
5
6     # darw a pink circle with blue background
7     penwidth 7
8     color blue, rgb(255,128,128)
9     circle 100,50,44
10
11    # draw a thick black line
12    color black
13    penwidth 5
14    line 50,50,250,250
15
16    # draw another thick red line
17    color red
18    penwidth 10
19    line 175,100,100,175
20
21    # draw a green square that is not filled
22    color green, clear
23    penwidth 10
24    rect 150,175,75,75
```

Program 14: Penwidth and Shape Outline



Sample Output 14: Penwidth and Shape Outline

`penwidth n`

Changes the width of the drawing pen. The pen represents the width of a line being drawn and also the width of the outline of a shape.


```
color pen_color, fill_color
```

Earlier in this chapter we saw the color statement with a single color. When only a single color is specified then both the pen and the fill color are set to the same value. You may define the pen and fill colors to be different colors by using the color statement with two colors.


```
clear
```

The special color **clear** may be used in the color statement to tell BASIC256 to only draw the border of a shape. Just set the fill color to clear.

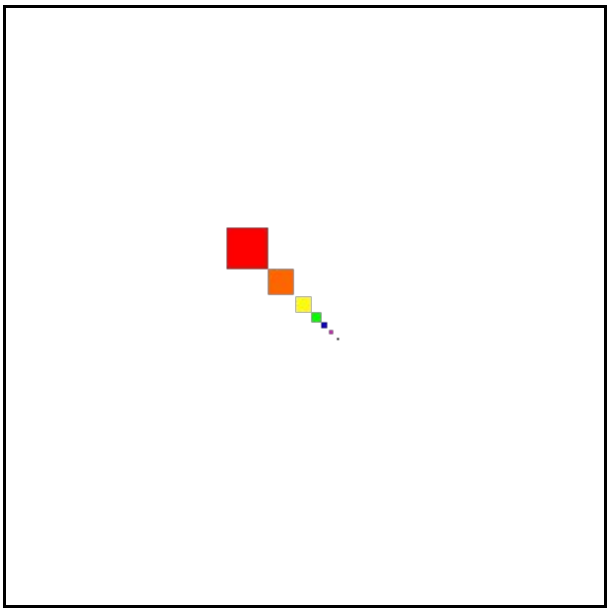
Setting Individual Points on the Screen:

The last graphics statement covered in this chapter is **plot**. The **plot** statement sets a single pixel (dot) on the screen. For most of us these are so small, they are hard to see. Later we will write programs that will draw groups of pixels to make very detailed images.

```
1     # pointplot.kbs - use plot to draw points
2
3     clg
4
5     color red
6     penwidth 21
7     plot 120,120
8
```

```
9     color orange
10    penwidth 13
11    plot 137,137
12
13    color yellow
14    penwidth 8
15    plot 149,149
16
17    color green
18    penwidth 5
19    plot 155,155
20
21    color blue
22    penwidth 3
23    plot 159,159
24
25    color purple
26    penwidth 2
27    plot 163,163
28
29    color black
30    penwidth 1
31    plot 166,166
```

Program 15: Use Plot to Draw Points



Sample Output 15: Use Plot to Draw Points


```
plot x, y
```

Draws a point on the screen in the current pen color with the current pen width.

Big Program

At the end of each chapter there will be one or more big programs for you to look at, type in, and experiment with. These programs will contain only topics that we have covered so far in the book.

This "Big Program" takes the idea of a face and makes it talk. Before the program will say each word the lower half of the face is redrawn with a different mouth shape. This creates a rough animation and makes the face more fun.

```
# talkingface.kbs
color yellow
rect 0,0,300,300
color black
rect 75,75,50,50
rect 175,75,50,50

#erase old mouth
color yellow
rect 0,150,300,150
# draw new mouth
color black
rect 125,175,50,100
# say word
say "i"

color yellow
rect 0,150,300,150
color black
rect 100,200,100,50
say "am"

color yellow
rect 0,150,300,150
color black
rect 125,175,50,100
say "glad"

```

```
29    color yellow
30    rect 0,150,300,150
31    color black
32    rect 125,200,50,50
33    say "you"
34
35    color yellow
36    rect 0,150,300,150
37    color black
38    rect 100,200,100,50
39    say "are"
40
41    color yellow
42    rect 0,150,300,150
43    color black
44    rect 125,200,50,50
45    say "my"
46
47    # draw whole new face with round smile.
48    color yellow
49    rect 0,0,300,300
50    color black
51    circle 150,175,100
52    color yellow
53    circle 150,150,100
54    color black
55    rect 75,75,50,50
56    rect 175,75,50,50
57    say "friend"
```

Program 16: Big Program - Talking Face

Sample Output 16: Big Program - Talking Face

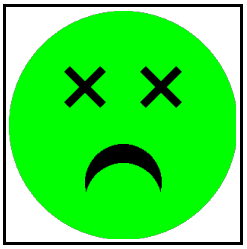
Exercises:

r e t a n i d r o o c
e e a r a e l c r u m
m e l c r i c e s s r
a c k v c e c c u y o
r y j l n t i i t p l
k a g t a h d h w l o
q n e n p a g i q o c
y r g a r i d p j t e
c l r e e t s a v e h
e g p h h u e n i l d
j r x p e n w i d t h

center, circle, clear, clg, color, coordinate, cyan, graphics, height, line, penwidth, plot, radius, rectangle, remark, save, width

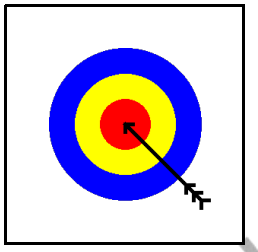
Problems

1. Type in the code for Program 11: Smiling Face with Circles (on page 24) and modify it to display Mr. Yuck. You may need to use the **penwidth** statement to make the lines you draw thicker.



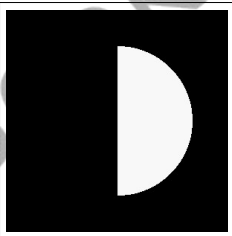
2. Write a program to draw a square and then say "square". Clear the graphics screen, draw a circle, and say "circle". Then clear the graphics screen draw several lines (in any pattern you would like) and say "lines".

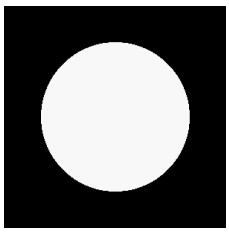
3. Use colors, lines, and circles to draw an archery target with an arrow in the center. Once the arrow is drawn make the computer say "Bullseye!".

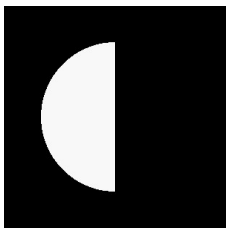


4. Write a program that draws each of the quarters of the moon (new moon, first quarter, full moon, and third quarter) and speaks the name for the quarter. Hint: Draw the moon as a circle and then draw a rectangle over the part you do not want.


New Moon


First Quarter


Full Moon


Third Quarter