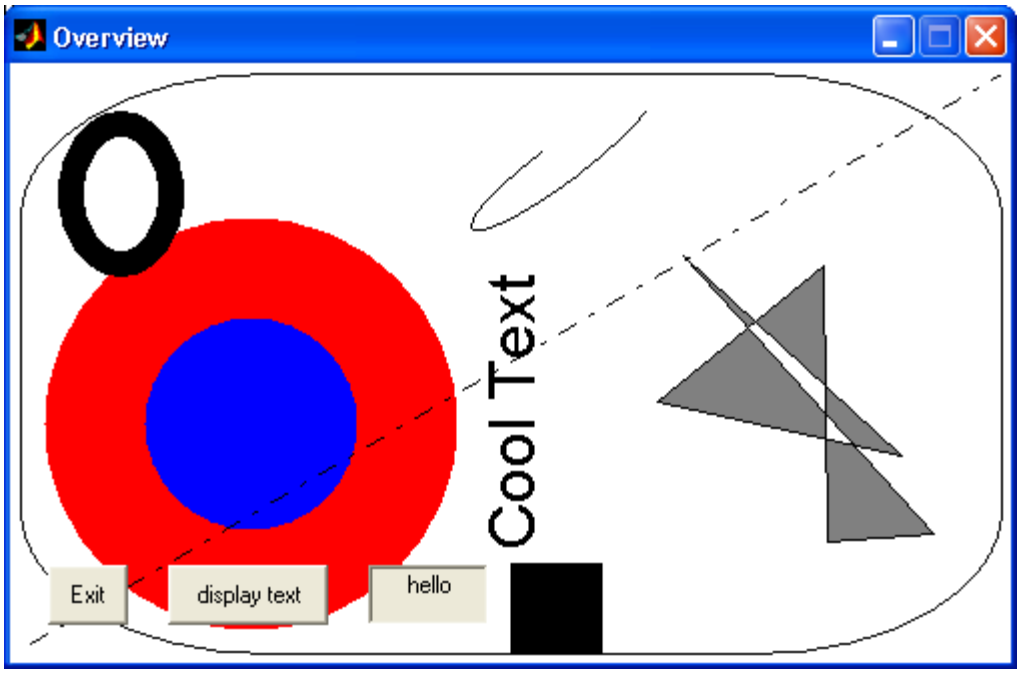
Graphics Programming with MatlabGraph

by
Maj Jason Moore, LtCol Tom Schorsch and Maj Tom Rathbun

In Computer Science 110, we will be creating graphics programs using a simple graphics package called MatlabGraph, which was written by Tom Schorsch and Tom Rathbun at USAFA. This reading explains how to get started using MatlabGraph, and it describes MatlabGraph routines you will need this semester.

Overview of MatlabGraph

All MatlabGraph commands are relative to a special graphics window, an example of which is depicted below. In this graphics window you can draw lines, rectangles, circles, arcs, ellipses, text, buttons, and edit boxes. These graphics objects can be of various sizes colors, styles, rotations and filled or unfilled. You can make your graphics interactive and animated. This reading describes how to use MatlabGraph and presents many examples to aid your understanding.



In the Graphics window above, there are several filled and unfilled graphic objects. MatlabGraph creates links to each graphics object so we can change their properties at a later time. We can change colors, position, styles and such.

You can interact with the mouse by determining the position of the mouse in the graphics window and determining if and where a mouse button was clicked. Special graphics objects like buttons and edit boxes are also available to interact with the user. You can also simulate movement in the graphics window by drawing an object then repeatedly changing it position slightly so it looks like it is moving.

There is help available by typing either `help MatlabGraph` or `help MatlabGraphFull`. The full version displays all the parameters for each function.

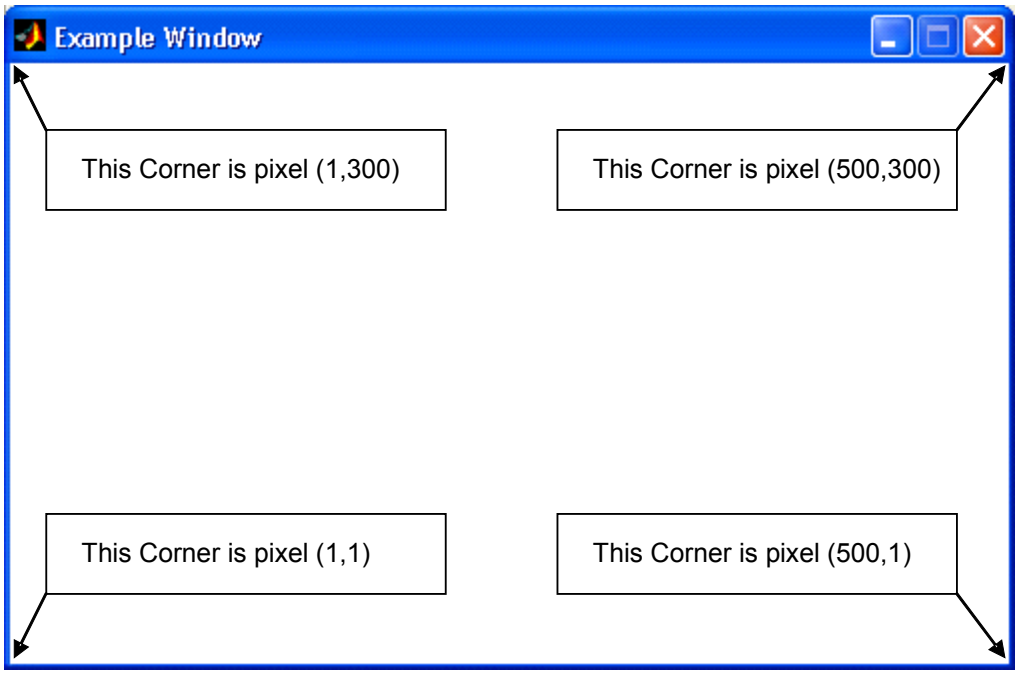
Getting Started with MatlabGraph

To use MatlabGraph you must create a graphics window. This graphics window must be created prior to calling any other MatlabGraph function.

```
Open_Graph_Window(width, height, name, x_origin, y_origin)
%
% width      - The width of the window in pixels
% height     - The height of the window in pixels
% name       - A string for the name of the window
% x_origin   - The x position of the window on your computer screen
% y_origin   - The y position of the window on your computer screen
```

For example the following function call creates the window below. Notice the last two parameters `(x_origin y_origin)` are not used. They are optional parameters.

```
Open_Graph_Window(500, 300, 'Example Window')
```



The graphics window always starts off with a white background. If a Matlab program calls function `Open_Graph_Window` with different values for `width` and `height` then the window will be sized appropriately. The origin of the graphics window's (X, Y) coordinate system is at the bottom left-hand corner of the window. The X-axis starts from 1 going left to right. The Y-axis starts from 1 going bottom to top.

At the end of your Matlab program you should close the graphics window using the following command:

```
Close_Graph_Window;
```

All other calls to MatlabGraph functions need to be between the calls that create and close the graphics window. The following example program illustrates a sequence of graphics function calls in your Matlab program.

```
function Empty_Window

% open graph window
Open_Graph_Window(400,300, 'Empty Window');

% display a text message in blue
Draw_Text(5, 5, 'Press any key to exit', 'Color', 'blue',…
'FontSize',24);

% the program stops here and waits for a key to be typed
Wait_For_Key;

% close the window
close_graph_window;
```

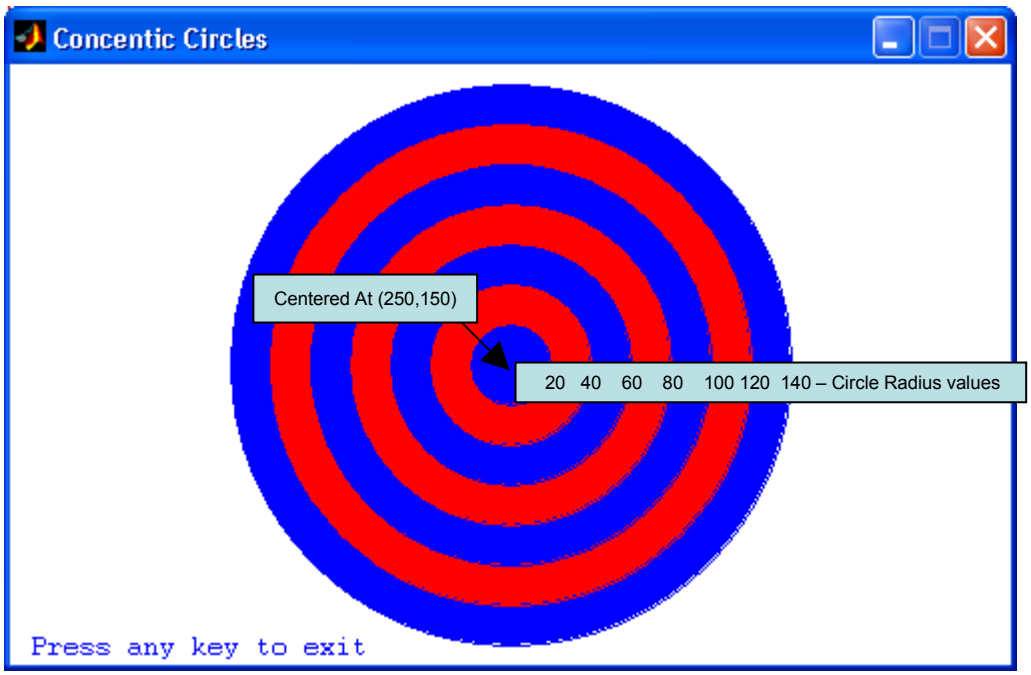
Displaying Graphic Objects with MatlabGraph

MatlabGraph has a collection of functions that make graphical shapes and text appear on the graphics window. All of these functions let the programmer specify the (X,Y) locations where the item is to appear in the graphic window.

As an example, one MatlabGraph function can draw a filled circle that is centered at an (X,Y) coordinate in the graphics window and that has a particular radius and color. The following seven calls to function `Draw_Circle` produce a collection of concentric circles that alternate between red and blue in color and that decrease in size.

```
% open the graphics window
Open_Graph_Window(500, 400, 'Concentric Circles')

% draw centered, concentric red and blue filled circles
Draw_Circle(250, 150, 140, 'Color', 'blue', 'Filled', true);
Draw_Circle(250, 150, 120, 'Color', 'red',  'Filled', true);
Draw_Circle(250, 150, 100, 'Color', 'blue', 'Filled', true);
Draw_Circle(250, 150, 80,  'Color', 'red',  'Filled', true);
Draw_Circle(250, 150, 60,  'Color', 'blue', 'Filled', true);
Draw_Circle(250, 150, 40,  'Color', 'red',  'Filled', true);
Draw_Circle(250, 150, 20,  'Color', 'blue', 'Filled', true);
```

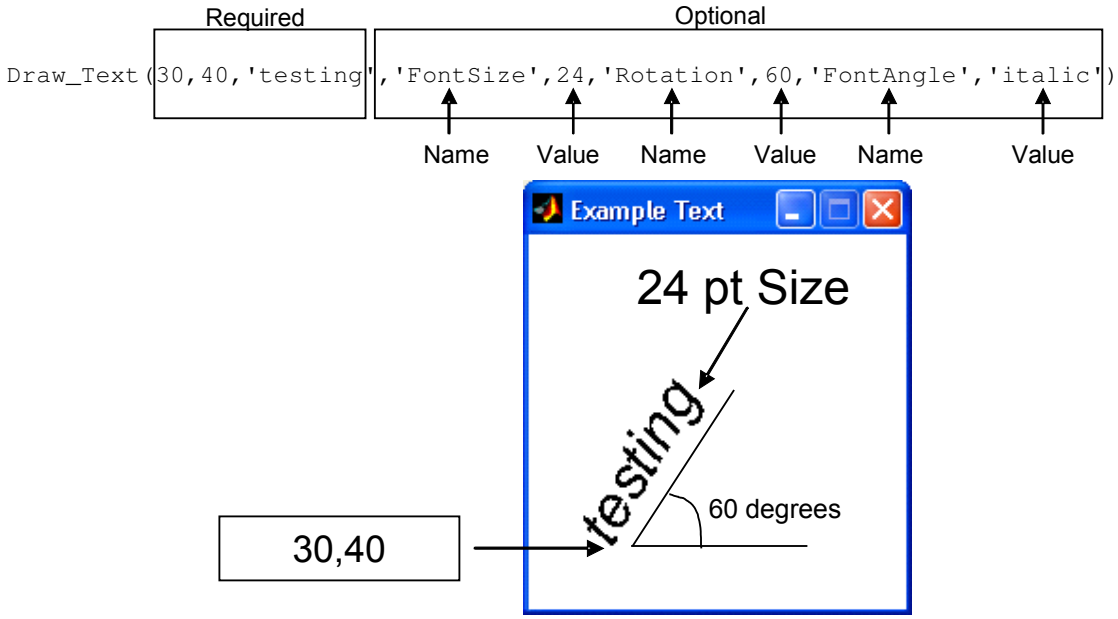
The effect of making these seven calls can be seen in the annotated graphic window below.



The drawing routines available in MatlabGraph are:

`Draw_Line`	Draws a line in the desired color given the two endpoint coordinates
`Draw_Box`	Draws a rectangle in the desired color given the upper left and lower right (X,Y) coordinates
`Draw_Circle`	Draws a circle given its center and radius
`Draw_Ellipse`	Draws an ellipse given the lower left and upper right (X,Y) coordinates of the rectangle bounding the ellipse.
`Draw_Arc`	Draws an arc (a section of an ellipse) given the upper left and lower right (X,Y) coordinates of the rectangle bounding the ellipse and starting and stopping coordinates.
`Draw_Ploygon`	Draws a series of lines to complete a polygon in the desired color given an array of coordinates
`Draw_Text`	Draws a string of characters, where the given (x,y) coordinate marks the lower left corner of the text
`Clear_Window`	Erases (clears) the entire graphics window.

All of the MatlabGraph drawing functions have required and optional parameters. The required parameters are for information like location and size. The optional parameters are for information like color, filled, style, etc. This type of information has default values which MatlabGraph uses unless you override it. For example in `Draw_Line` you must specify the `x1, y1, x2`, and `y2` endpoint values, but `color, style, filled` and `thickness` have default values. Optional parameters are entered slightly different than required parameters. Required parameters go first, followed by optional parameters. Optional parameters are entered in pairs; the optional parameter name is first followed by its value. In the following function call to `Draw_Text`, the first three parameters are required and the last three are optional.



MatlabGraph uses default values for optional parameters if not specified. The default values are:

Name	Value	Possible Values
'Color'	'black'	See next table
'Rotation'	0	0 – 360 (degrees)
'Filled'	false	true or false
'Thickness'	0.5	> 0 (measured in pixel width)
'Style'	'-' Solid	'-' - Solid , '--' - dashed ':' - dotted, '-.' - dash-dotted
'Rounded'	[0,0]	[0-1, 0-1] (0-no rounding, 1-fully rounded)
'HorizontalAlignment'	'left'	'left', 'center', 'right'
'FontName'	'Helvetica'	Fonts loaded in the operating system
'FontSize'	10	> 4 (measured in points)
'FontAngle'	'normal'	'normal', 'italic', 'oblique'
'FontWeight'	'normal'	'light', 'normal', 'demi', 'bold'
'VerticalAlignment'	'middle'	'top', 'cap', 'middle', 'baseline', 'bottom'

To find out which parameters are required and which are optional for each draw function you can look at the list of function calls at the end of this reading or you can type help Function name from the Matlab command line, for example, in this case below, `x, y,` and `txt` are required and the rest are optional.

```
>> help Draw_Text
  h = Draw_Text(x, y, txt)
  Inputs
         x                    - X coordinate of the lower left corner
                                for the text unless HorizontalAlignment
                                is changed
         y                    - Y coordinate of the lower left corner
                                for the text unless HorizontalAlignment
                                is changed
         txt                  - The text to print
  Optional
         'Color'              - color of the circle - Default 'black'
                                colors are 'yellow', 'magenta', 'cyan',
                                'red', 'green', 'blue', and 'white'
         'Rotation'           - The rotation of the text - Default 0
         'HorizontalAlignment'- [ {left} | center | right ]
         'FontName'           - The name of the text - Default Helvetica
         'FontSize'           - The size of the text - Default 10
         'FontAngle'          - [ {normal} | italic | oblique ]
         'FontWeight'         - [ light | {normal} | demi | bold ]
         'VerticalAlignment'  - [ top | cap | {middle} | baseline |
                                bottom ]
  Output
         h     - Is a handle to the graphics object
```

There are two ways to change from the default value for an optional parameter. The first by listing it and its value when you call the function, as was done in the example on the previous page. The second way is to change the default value itself. This is accomplish by

calling the function `Change_Default_Value` and specifying the name and its new value. For example, to change default color to green, the call would look like this:

```
Change_Default_Value('Color', 'green')
```

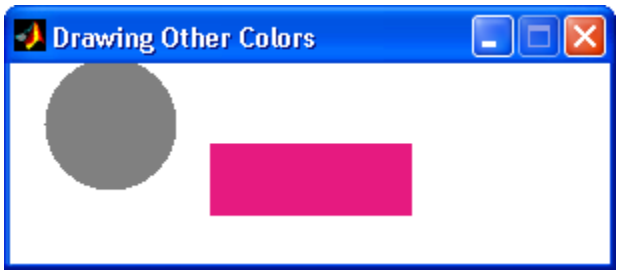
The change to the default value stays in effect until the graphics window is closed and reopened or `Change_Default_Value` is called again to change the value to something else.

There are three ways to specify color. The first is by the full name, the second is by the single character value and the third is by an array of RGB (Red, Green, Blue) values. The following table demonstrates how to specify colors:

Full Name	Short Name	RGB Value
'black'	'k'	[0, 0, 0]
'blue'	'b'	[0, 0, 1]
'green'	'g'	[0, 1, 0]
'cyan'	'c'	[0, 1, 1]
'red'	'r'	[1, 0, 0]
'yellow'	'y'	[1, 1, 0]
'white'	'w'	[1, 1, 1]
'magenta'	'm'	[1, 0, 1]

Additional colors are available by using the RGB method and specifying values between 0 and 1 for each RGB value. The follow example illustrates using RGB values.

```
Open_Graph_Window(300, 100, 'Drawing Other Colors');

% This color is gray
Draw_Circle(50, 70, 30, 'Color', [0.5,0.5,0.5], 'Filled', true);

% this color is purplish
Draw_Box (100, 25, 200, 60, 'Color', [0.9,0.1,0.5], 'Filled', true);
```



Example Calls to Display Graphics Objects

To draw a blue line from coordinate (15,99) to coordinate (12,13), type in the following:

```
Draw_Line(15, 99, 12, 13, 'Color', 'blue');
```

We can also use variables to pass the coordinates to the drawing functions. For example, if a blue line is to be drawn from (1,1) to the point whose x and y coordinates are stored in variables `My_X` and `My_Y`, respectively, we would use the following call. A different line will be drawn depending on the values of variables `My_X` and `My_Y`.

```
Draw_Line(1, 1, My_X, My_Y, 'Color', 'blue');
```

To draw a box, we pass the upper left corner coordinates to X1 and Y1, and the lower right corner coordinates to X2 and Y2. Here, for example, we draw a cyan-colored box:

```
Draw_Box(38, 100, 76, 20, 'Color', 'cyan');
```

The box provides our first chance to give an example of a filled object. The default is to not be filled. It we specify true for the 'Filled' parameter then the interior is filled with the same color as the object itself. Here we draw a box filled with black (what can we infer from the coordinates used in the call to `Draw_Box`, assuming variables `Width` and `Height` were used to create the graphic window?):

```
Draw_Box(Width/2 - 10, Height/2 + 10, Width/2 + 10, Height/2 - 10,
         'Color', 'black', 'Filled', true);
```

To draw a circle, we specify the coordinates of the center and the length of the radius. Here we draw a filled green circle with radius of 9 pixels centered at the point with coordinates specified in the variables `X_Circle` and `Y_Circle`:

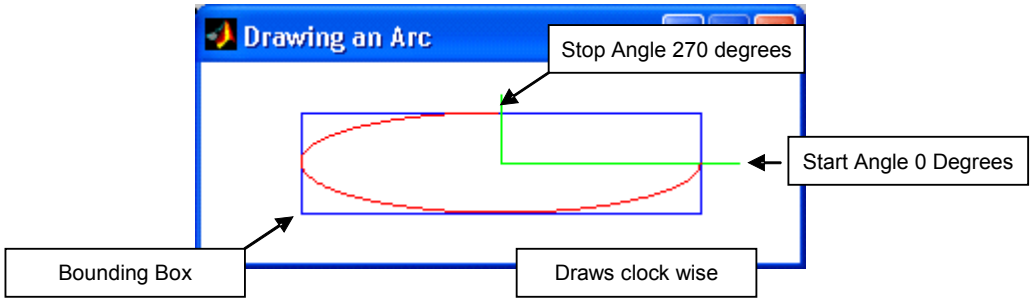
```
Draw_Circle(X_Circle, Y_Circle, 9, 'Color', 'green', 'Filled', true);
```

Drawing ellipses and arcs may be slightly less intuitive than the previous shapes. For ellipses, we specify the (x, y) coordinates of the smallest rectangle bounding the ellipse. The figure below illustrates how an ellipse is drawn; it was created using the Matlab code below the figure. Note in the figure below a box is drawn first and a filled ellipse is drawn with the same coordinates as the box. The ellipse then "fills" the box as shown.

```
Open_Graph_Window(300, 100, 'Drawing an Ellipse');

% Draw a box and an ellipse using the same X, Y coordinates
Draw_Box    (50, 25, 250, 75, 'Color', 'blue', 'Filled', false);
Draw_Ellipse(50, 25, 250, 75, 'Color', 'red', 'Filled', true);
```

For arcs we specify the bounding box (as we did for an ellipse). We also specify starting and stopping angles. The drawn arc starts on the ellipse at the start angle and ends at the stop angle. The Arc is drawn in a clockwise direction.

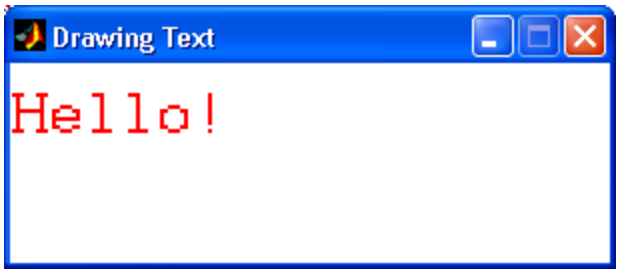
The figure below illustrates how an arc is drawn; it was created using the Matlab code below the figure.




```
% opens the graph window
Open_Graph_Window(300, 100, 'Drawing an Arc');

% Draws the bounding box
Draw_Box (50, 25, 250, 75, 'Color', 'blue', 'Filled', false);

% draws the line intersection with the start angle
Draw_Line (150, 50, 270, 50, 'Color', 'green');

% draws the line intersecting with the stop angle
Draw_Line (150, 50, 150, 85, 'Color', 'green');

% draws the angle
Draw_Arc (50, 25, 250, 75, 0, 270, 'Color', 'red', 'Filled', false);
```

Often it's nice to put text on our graphics window. We can't use `disp` or `fprintf` to write to the graphics window. Instead, we use a function that lets us specify where to put the text in the graphics window, what color the text should be, and the size of the text. Note that the (X, Y) coordinates passed to the `Draw_Text` routine specify the lower left corner of the text's position if using the default HorizontalAlignment. Assuming that the height of the graphics window is stored in the variable named `Height`, the example below places a red greeting in the upper left corner of the window with a font size of 24.

```
Draw_Text(1, Height - 30, 'Hello! ', 'Color', 'red', 'FontSize', 24);
```



We can erase all drawing objects in the graphics window by calling procedure `Clear_Graph_Window`. This function collects all the links to all the objects in the window and deletes them one at a time.

Input with MatlabGraph

While MatlabGraph is fairly simple, it is powerful enough to draw some complicated pictures. However, in addition to being able to draw pictures, we want to make our graphic programs interactive – so the user can direct what happens on the window. We will now discuss input routines for both the mouse and the keyboard. The input routines available are:

`Wait_For_Key`	A function that simply waits until a key is pressed.
`Key_Hit`	A function returning the Boolean value `true` if a key has been pressed since the last key was processed. Normally the procedure `Get_Key` should be called when `Key_Hit` is true.
`Get_Key`	A function that returns the character typed by the user. If no character is ready to be processed, `Get_Key` waits until the user presses a key.
`Wait_For_Mouse_Button`	A function that simply waits until either mouse button is pressed.
`Mouse_Button_Pressed`	A function that returns the Boolean value `true` if either button has been pressed since the last button was processed. Normally the procedure `Get_Mouse_Button` should be called when `Mouse_Button_Pressed` is true.
`Get_Mouse_Button`	A function that takes either button and returns the coordinates of a click of that button. If no click is ready to be processed, Get_Mouse_Button waits until the user presses the button.
`Get_Mouse_Location`	A procedure that immediately returns the current (X,Y) coordinates of the mouse.

It is important to note that some of the above calls will halt your program until a key or mouse button is pressed. Other calls do not wait but merely return a true or false or (X,Y) coordinates. If you do not want your program to wait for a key press or mouse button click you can use `Key_Hit` or `Mouse_Button_Pressed` to check whether such an event has occurred.

Example Calls to Keyboard Input Routines

If we want a program to wait for a keystroke, we simply use `Wait_For_Key`.

```
Wait_For_Key;
```

If we want our program to interpret the key that is pressed and maybe do something based on which key is pressed we use the `Get_Key` function. The `Get_Key` function will also stop the program until a key is pressed. That key is then assigned to a variable. In the example below it is assigned to the variable `Char_Var`.

```
  % Wait for the user to enter a key and assign it to the variable
Char_Var

  Char_Var = Get_Key;
```

Your program can now use the value of the variable `Char_Var` to perform some action, perhaps to draw a circle if a "C" is pressed and draw a box if a "B" is pressed.

```
  if Char_Var == 'C'
    Draw_Circle(100, 100, 20, 'Color', 'blue');
  elseif Char_Var == 'B'
    Draw_Box(10, 10, 20, 20, 'Color', 'blue');
  end
```

If we want to see if a character is available before committing our program to act on it, we use the function `Key_Hit` in conjunction with function `Get_Key`. By checking `Key_Hit`, before calling `Get_Key`, as demonstrated in the code below, we avoid stopping the program and waiting for keyboard input.

```
  if Key_Hit
    Char_Var = Get_Key;
    % Now do something with the character in Char_Var
  end
```

Example Calls to Mouse Input Routines

Mouse input is similar to keyboard input. If we simply want to wait for a mouse button to be pressed, we use procedure `Wait_For_Mouse_Button`. The following code will halt your program until the mouse button is pressed.

```
  Wait_For_Mouse_Button;
```

Often it may be useful to determine where the mouse was located when the mouse button was pressed. The procedure `Get_Mouse_Button` will stop the program until a mouse button is pressed and then return the X, Y position at which the mouse button was clicked. The following code will wait for a mouse button and then place the X and Y pixel coordinates of the mouse position at the time of the click into `X_Var and Y_Var` variables.

```
  [X_Var, Y_Var] = Get_Mouse_Button;
```

As with keyboard input, we may not want to commit to waiting for a mouse button input unless we know the mouse button has already been pressed. We can check to see if a mouse button input is waiting by ensuring `Mouse_Button_Pressed` is true before calling `Get_Mouse_Button`. For example, we may wish to act on the next user input, but without knowing whether it will take the form of a keystroke or a mouse button. The example below handles this situation.

```
if Mouse_Button_Pressed
   [X_Var, Y_Var] = Get_Mouse_Button;

   % code would be inserted here to act on the button click
 elseif Key_Hit
   The_Key = Get_Key;
   % code would be inserted here to act on the keystroke
 end
```

If we just want to know where the mouse is on the graphics window at any particular moment, we can use `Get_Mouse_Location` which returns the X and Y coordinates of the mouse. The following call places the current position of the mouse in variable `X_Var` and `Y_Var`.

```
[X_Var, Y_Var] = Get_Mouse_Location;
```

Special Input with MatlabGraph

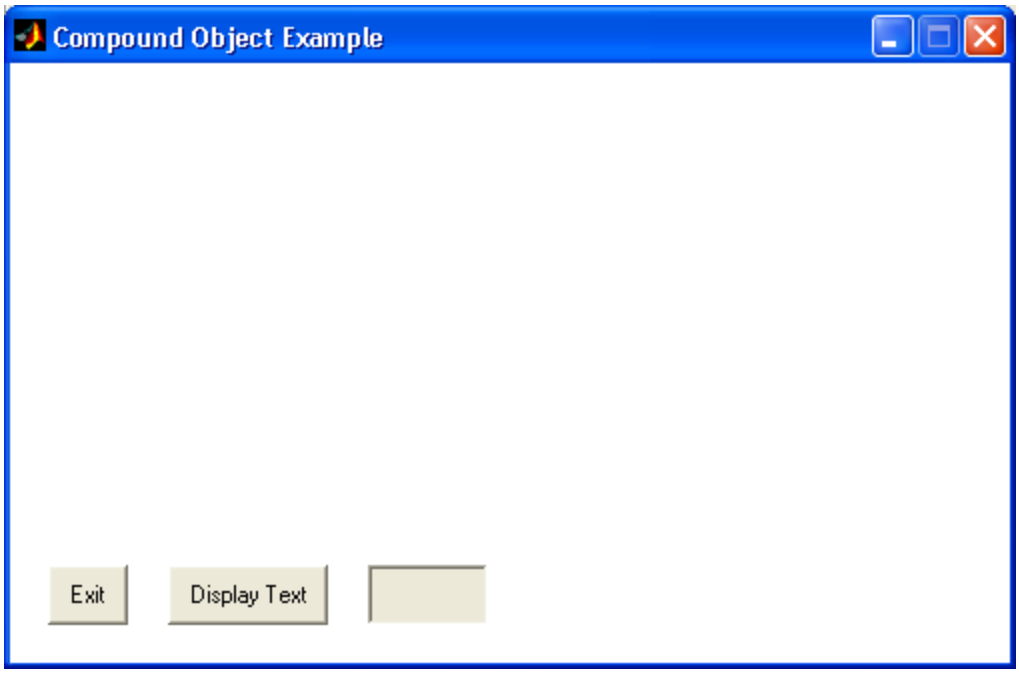
If would be nice to have different mouse clicks mean different actions, for example, to click to exit or to click to start over. With our current set of function calls it would take some work to accomplish this. However, MatlabGraph provides two compound objects that make this easy. A compound object combines a graphics object with mouse and key inputs, these two objects are Buttons and Text Boxes. The functions available to create and use these compound objects are:

`Create_Button`	Places a graphical Button object on the display window.
`Test_Button`	Returns `true` if the button has been clicked and `false` otherwise.
`Create_Text_Box`	Places a graphics Edit Box object on the display window
`Test_Text_Box`	Returns `true` if the Edit Box has been typed in and `false` otherwise
`Get_Text_Box`	Returns a string of whatever characters were typed in the Edit Box.
`Clear_Text_Box`	Clears the Edit box

There are two steps in using these compound objects. First, you create them and second you test so see if the user has interacted with them. The creation looks like this

```
Open_Graph_Window(500,300,'Compound Object Example');

h1 = Create_Button(20,20,40,30, 'Exit');
h2 = Create_Button(80,20,80,30, 'Display Text');
h3 = Create_Text_Box(180,20,60,30, 'User Name');
```

The names of the buttons and text boxes are very important. The name is used in the test function, so you must use the same exact name when testing. Executing the above code results in the following.



In the second step you must set up a loop to continually test for the user interacting with the compound graphics objects. The test function return a true or false value so you can use them in an if statement. Be sure to put a small pause in the loop to give Matlab a chance to do some background processing or Matlab may lock up. An example loop follows.

```
while true

    pause(0.01)

    if Test_Button('Exit')
        % do something
    end
    if Test_Button('Display Text')
        % do something
    end
    if Test_text_Box('User Name')
        % do something
    end
end
```

In the case of the Text Box there are two additional functions to get the contents and clear the text. The following example demonstrates their use:

```
Typed_string = Get_Text_Box('User Name')
str = ['You typed ' Typed_string ' in the text box'];
Draw_Text(50, 200, str);
Clear_Text_Box('User Name');
```

Graphics Objects and Handles

If you wanted to keep track of many different pages in a book, you might put tabs on those pages to get there quickly. MatlabGraph does a similar thing width graphics objects. MatlabGraph creates a link (Called a handle) to each graphics object, stores the handle, and returns that handle like a tab when a drawing command is executed. If you specify a return variable you can also save the handle to the object you just drew. In the following code, h stores the handle to the graphics object generated by `Draw_Circle`.

```
h = Draw_Circle(x, y, 30, 'Color', 'Red', 'Filled', true);
```

With an object's handle you have complete mastery over that object. You can change its color, location, visibility, and even delete it. You can use the function `Select_Object` to get a user selected graphics object's handle from MatlabGraph. This function waits for a mouse click and returns the handle of the object that the mouse was on when clicked. If no graphics object was at the clicked location then it will return the handle to the window. Once you get the handle you can compare it or use it to change the properties of the object. For example the following code determines if you clicked on a circle after you have drawn it.

```
% draw circle
h = Draw_Circle(250, 150, 140, 'b', true);

% select an object
newH = Select_Object;

% compare handles
if h == newH
    % do something
end
```

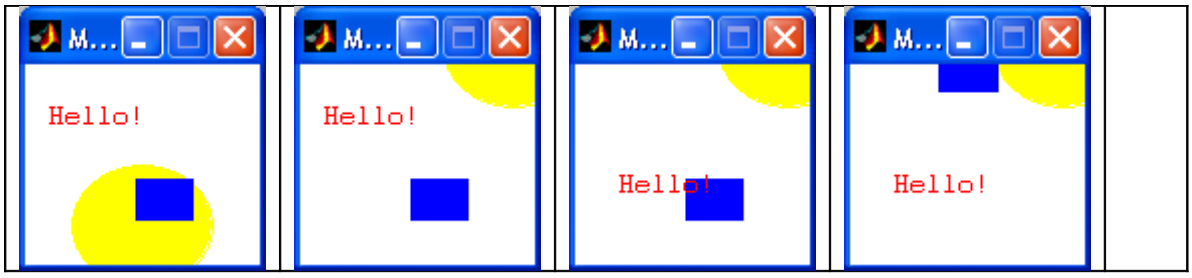
Change Object properties

When a graphics object is drawn, MatlabGraph stores properties about the object. If these values are changed then MatlabGraph automatically updates the display of the graphics object. The following functions are available to get and change object properties

`Change_Object_Color`	A function that changes the color of an object
`Delete_Object`	A function that deletes the graphics object.
`Hide_Object`	A function that makes the object invisible, but it is still there. `Show_Object` restores its visibility
`Move_Object_By`	A function that changes the position of an object by x and y offset values.
`Move_Object_To`	A function that changes the position of an object to a new x and y location. Does not work for arcs and polygons
`Show_Object`	A function that makes an object visible. If the object is already visible then it has no effect.
`Select_Object`	Waits for a mouse click and returns the handle to the object selected or the window if now object was clicked on

The following code lets the user select an object and then moves that object by a random amount. The sequence of pictures below shows the results.

```
Open_Graph_Window(100, 100,'Move Example');

% draw some objects
Draw_Text(10, 70, 'Hello!', 'red', 0, 12);
Draw_Circle(50, 20, 30, 'yellow', true);
Draw_Box    (50, 25, 70, 40, 'blue',true);

% loop for three moves
for i = 1:3
    % get the handle of the object that was selected
    h = Select_Object;
    % calculate two random values
    x = Get_Random(1,99,'integer');
    y = Get_Random(1,99,'integer');
    % use the handle to move the object
    Move_Object_To(h,x,y)
end
```

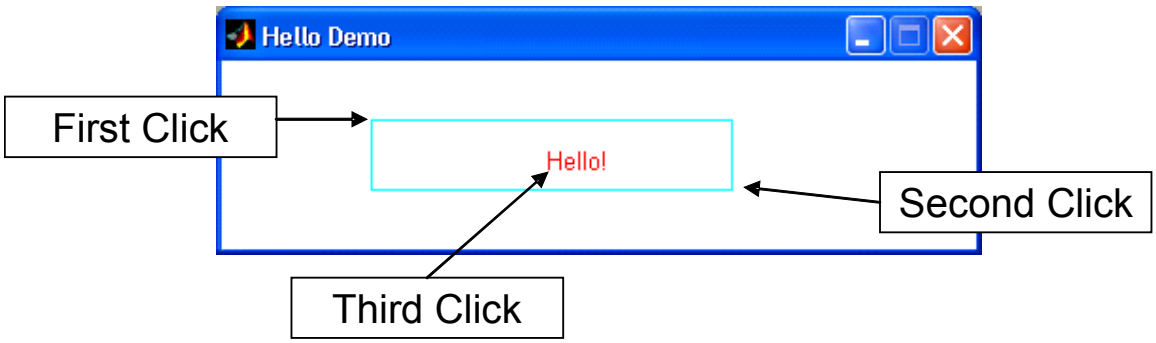

M...
Hello!
M...
Hello!
M...
Hello!
M...
Hello!

Example program: Hello_Demo

Below is a complete program demonstrating many of the above calls. This example waits for left mouse clicks to define two corners of a rectangle and the position of some text. It then leaves the graphics window open until the user presses a key.

```
function Hello_Demo
% This program opens a 400x100 graphics window.  It then waits
% for two consecutive left mouse clicks, saves their locations,
% and uses them to draw a cyan box.  It then waits for one more
% mouse click and places the word 'Hello' in magenta at that
% position.  Finally, it quits when the user presses a key.

% These constants hold the dimensions of the graphics window
Max_X = 400;
Max_Y = 100;


disp('This program will open a graphics window.');
disp('It will then wait for two left mouse clicks.');
disp('These mouse clicks will define the corners');
disp('of a rectangle which will be drawn on the');
disp('window.  The program will then wait');
disp('for another mouse click, this one defining');
disp('the location of a greeting. After the greeting');
disp('is displayed, the program will wait for you');
disp('to press a key, at which time it will quit.');


Open_Graph_window(Max_X,Max_Y,'Hello Demo');

% Get first two mouse clicks and draw a box
[X_Upper_Left, Y_Upper_Left] = Get_Mouse_Button;
[X_Lower_Right, Y_Lower_Right] = Get_Mouse_Button;
Draw_Box(X_Upper_Left,Y_Upper_Left,X_Lower_Right,Y_Lower_Right,
'Color','cyan');

% Get 3rd mouse click and display text at that location
[X_Text, Y_Text] = Get_Mouse_Button;
Draw_Text(X_Text,Y_Text,'Hello!', 'Color','red');

% waits until the user presses a key to finish the program
Wait_For_Mouse_Button;
Close_Graph_Window;
```

Below is a sample run of the program with the three clicks annotated.



Animation with MatlabGraph

Sometimes we want to make an object move across the graphic window. Recall that movement can be simulated by drawing an object, then changing the position property of the object. To make objects move smoothly and at a reasonable speed, we might want to try moving the object more than one pixel at a time and with a small delay between times when the object is drawn. We can accomplish this with a loop like the one in the program below, which moves a red circle horizontally across the graphic window:

```
function Animate_Ball
% This program opens a 400x100 graphics window.  It then draws
% an animation of a circle traveling horizontally across the
% graphic window.  The program will quit when the user presses a key.

% These constants hold the dimensions of the graphics window
Max_X = 400;
Max_Y = 100;

disp('This program will open a graphics window.');
disp('It will then position a circle at the left of the');
disp('window and then make the circle move across the');
disp('window by drawing and erasing the circle in.');
disp('different positions.');
disp('A key press will end the program.');

Open_Graph_Window (Max_X, Max_Y, 'Animate Ball');

% initialize the position of the circle to the left of the window
X_Pos = 1;
Y_Pos = Max_Y/2;


% draw a red circle
handle = Draw_Circle(X_Pos,Y_Pos, 3, 'Color', 'red', 'Filled', true);

while true

    pause(0.05);  % wait a bit so the user can see the circle

    % Update X_pos here, 4 seems a good number of pixels
    % We don't update Y_pos, since we are moving horizontally
    X_Pos = X_Pos + 4;

    % Move the ball
    Move_Object_To(handle,X_Pos,Y_Pos)

    % Exit when the center of the ball goes off the right side of the
    % window.
    if X_Pos > Max_X
        break
    end
end

% The following waits until the user presses a key before ending
Wait_For_Key;
Close_Graph_Window;
```

How could this program be changed so that the ball "bounced" off of the sides of the graphic window? (hint… start adding a "– 4" to the `X_Pos` each time.)

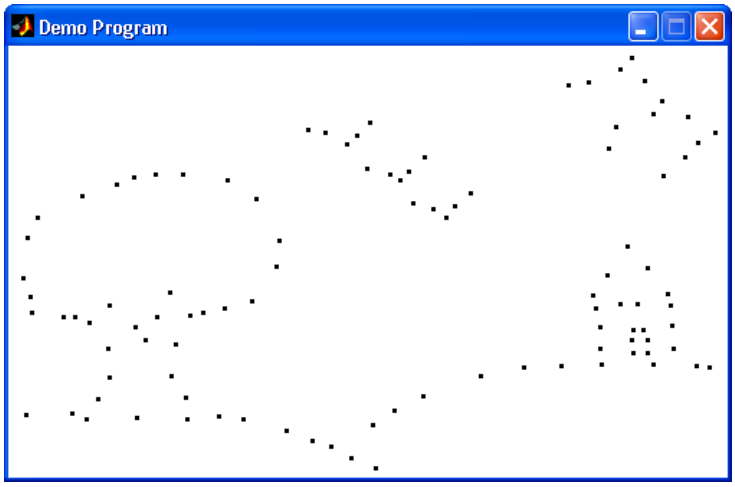
One note on the `pause` function in the above program: the pause is important to have time to see the movement. It is also important to give Matlab some time to do some background processing. No delay will cause Matlab to lock up.

Example program: Draw_Demo

In this section we show two more complete programs to help you understand how to use MatlabGraph for more interesting programs. The next example draws a black box wherever the left mouse button is clicked. It quits when the right mouse button is clicked.

```
function Draw_Demo
% This program opens a 500x300 graphics window.  It paints a black
% box everywhere the user left clicks in the window.  Finally, it
% exits when the user types a key

% These constants hold the dimensions of the graphics window
Max_X = 500;
Max_Y = 300;

Open_Graph_window(Max_X, Max_Y,'Demo Program');

while true
    if Key_Hit
        break;
    end

    % Draw a small Black Box where the left button pressed
    if Mouse_Button_Pressed
        [X_Var, Y_Var] = Get_Mouse_Button;
        Draw_Box(X_Var-1,Y_Var-1,X_Var+1,Y_Var+1, 'Filled', true);
    end

    % important to have a pause in the loop
    pause(0.01)
end
Close_Graph_Window;
```

An un-artistic attempt to draw a picture using the Draw_Demo program!


Demo Program

Summary of all MatlabGraph procedure and function calls.

Graphic window opening and closing functions

```
h=Open_Graph_Window(width, height, name, x_origin, y_origin)
%
% Opens a graphics window, must be called before any other grahics
%       command
%
% Inputs
%        width      - The width of the window in pixels
%        height     - The height of the window in pixels
%        name       - A string for the name of the window
%        x_origin   - The x value for the location on the screen for
the %                     window
%        y_origin   - The y value for the location on the screen for
the %                     window

% Close_Graph_Window(h)
%
% Deletes all objectsthen closes the graph window
%
% Inputs
%         h     - Is a handle to the graphics window - Default is
%                 current window
```

Keyboard input functions

```
Wait_For_Key
%
% Waits for nay key to be press then continues
%

pressed = Key_Hit
%
% Returns a 1 if a key was typed and a 0 otherwise
%
% Inputs
%
% Output
%       pressed - true false value of whether a key was pressed or not

c = Get_key
%
% waits for a key click and return the character typed
%
% Inputs
%
% Output
%       c - character that was type on the keyboard
```

Mouse input functions

```
Wait_For_Mouse_Button
```

```
%
% Waits for a mouse button then returns
%



pressed = Mouse_Button_Pressed
%
% Returns a 1 if either mouse button was press or a 0 otherwise
%
% Inputs
%
% Output
%       pressed - true false value of whether a mouse button was pressed
%                 or not

[x,y] = Get_Mouse_Button
%
% Waits for a mouse click then return the x and y coordinates of where
%       is was clicked
% Inputs
%
% Output
%       x - X position of the mouse
%       y - Y position of the mouse

[x,y] = Get_mouse_Location
%
% Returns the x and y coordinates of the current mouse location
%
% Inputs
%
% Output
%       x - X position of the mouse
%       y - Y position of the mouse
```

Compound Graphics Objects for Input/Output

```
h = Create_Text_Box(x,y,width,height,name,value)
%
% Create a text box that allows to user to enter text in
%
% Inputs
%        x      - X coordinate of the lower left hand point of the box
%        y      - Y coordinate of the lower left hand point of the box
%        width  - The width of the text box
%        height - The height of the text box
%        name   - the name used to identify the text box
%        value  - An initial value to be placed in the text box
% Output
%        h     - Is a handle to the graphics object
%

Clear_Text_Box(name)
%
% Clears the text in the text box
```

```
%
% Inputs
%        name   - the name used to identify the text box
% Output
%        none
%

Test_Text_Box(name)
%
% Determines if the user typed in the text box
%
% Inputs
%        name   - the name used to identify the text box
% Output
%        pressed - true or false value

str = Get_Text_Box(name)
%
% Gets the contents of the text box
%
% Inputs
%        name   - the name used to identify the text box
% Output
%        str - a string with the contents of the text box

h = Create_Button(x,y,width,height,name)
%
% Create a button that allows to user to click the mouse in
%
% Inputs
%        x      - X coordinate of the lower left hand point of the
%                 button
%        y      - Y coordinate of the lower left hand point of the
%                 button
%        width  - The width of the button
%        height - The height of the button
%        name   - the name of the button and it is used to identify the
%                 button
% Output
%        h     - Is a handle to the graphics object
%

Test_Button(name)
%
% Determines if the user clicked on the button
%
% Inputs
%        name   - the name used to identify the button
% Output
%        pressed - true or false value
```

Drawing functions

```
h = Draw_Line(x1, y1, x2, y2)
%
% Draws a line
%
% Inputs
%        x1            - X coordinate of the starting point
%        y1            - Y coordinate of the starting point
%        x2            - X coordinate of the stopping point
%        y2            - Y coordinate of the stopping point
% Optional
%        'Color'       - color of the circle - Default black
%                        colors are 'yellow', 'magenta', 'cyan', 'red',
%                        'green', 'blue', and 'white'
%        'Thickness'   - If the circle is not filled this specifies how
%                        thick to make the line - Default 0.5
%        "style'       - A string indicating the style of the lines -
%                        Default '-' (Solid)
%                        Other styles are '--' (dashed), ':' (dotted),
%                        '-.' (dash-dotted) Applies to unfilled circles
%                        only

h = Draw_Box(x1, y1, x2, y2)
%
% Draws a box
%
% Inputs
%        x1            - X coordinate of the lower left bounding box
%        y1            - Y coordinate of the lower left bounding box
%        x2            - X coordinate of the upper right bounding box
%        y2            - Y coordinate of the uooer right bounding box
% Optional
%        'color'       - color of the circle - Default black
%                        colors are 'yellow', 'magenta', 'cyan', 'red',
%                        'green', 'blue', and 'white'
%        'Filled'      - A true or false value - Default true
%        'Thickness'   - If the circle is not filled this specifies how
%                        thick to make the line - Default 0.5
%        'Style'       - A string indicating the style of the lines -
%                        Default '-' (Solid) Other styles are '--'
%                        (dashed), ':' (dotted), '-.' (dash-dotted)
%                        Applies to unfilled circles only
%        'Rounded'     - A 2 value array with values between 0 and 1 -
%                        Default [0,0] Indicates how rounded the
corners
%                        are. [0,0] is not rounded, [1, 1] completely
%                        rounded like and elipse
%
% Output
%        h     - Is a handle to the graphics object

h = Draw_Circle(x, y, radius)
%
% Draws a circle
%
```

```
% Inputs
%       x           - X coordinate of the center of the circle
%       y           - Y coordinate of the center of the circle
%       radius      - Radius of the circle
% Optional
%       'Color'     - color of the circle - Default black
%                     colors are 'yellow', 'magenta', 'cyan', 'red',
%                    'green', 'blue', and 'white'
%       'Filled'    - A true or false value - Default true
%       'Thickness' - If the circle is not filled this specifies how
%                     thick to make the line - default 0.5
%       'Style'     - A string indicating the style of the lines -
%                     Default '-' (Solid) Other styles are '--'
%                     (dashed), ':' (dotted), '-.' (dash-dotted)
%                     Applies to unfilled circles only
% Output
%       h     - Is a handle to the graphics object

h = Draw_Ellipse(x1, y1, x2, y2)
%
% Draws an Ellipse
%
% Inputs
%       x1          - X coordinate of the lower left bounding box
%       y1          - Y coordinate of the lower left bounding box
%       x2          - X coordinate of the upper right bounding box
%       y2          - Y coordinate of the upper right bounding box
% Optional
%       'Color'     - color of the circle - Default black
%                     colors are 'yellow', 'magenta', 'cyan', 'red',
%                     'green', 'blue', and 'white'
%       'Filled'    - A true or false value - Default true
%       'Thickness' - If the circle is not filled this specifies how
%                     thick to make the line - Default 0.5
%       'Style'     - A string indicating the style of the lines -
%                     Default '-' (Solid) Other styles are '--'
%                     (dashed), ':' (dotted), '-.' (dash-dotted)
%                     Applies to unfilled circles only
% Output
%       h     - Is a handle to the graphics object
```

```
h = Draw_Arc(x1, y1, x2, y2, start_angle, stop_angle)
%
% Draws an Arc
%
% Inputs
%       x1          - X coordinate of the lower left bounding box
%       y1          - Y coordinate of the lower left bounding box
%       x2          - X coordinate of the upper right bounding box
%       y2          - Y coordinate of the upper right bounding box
%       start_angle - The angle from where the arc is started from
%       stop_angle  - The angle from where the arc is stop
% Optional
%       'Color'     - color of the circle - Default black
%                     colors are 'yellow', 'magenta', 'cyan', 'red',
%                     'green', 'blue', and 'white'
%       'Filled'    - A true or false value - Default true
%       'Rotation'  - is the radians off center in a counter-clockwise
%                     rotation - Default 0
%       'Thickness' - If the circle is not filled this specifies how
%                     thick
%                     to make the line - Default 0.5
%       'Style'     - A string indicating the style of the lines -
%                     Default '-' (Solid) Other styles are '--'
%                     (dashed), ':' (dotted), '-.' (dash-dotted)
%                     Applies to unfilled circles only
% Output
%       h     - Is a handle to the graphics object

h = Draw_Polygon(x,y)
%
% Draws a polygon
%
% Inputs
%       x             - An array of X coordinate for the points in the
%                       polygon
%       y             - An array of Y coordinate for the points in the
%                       polygon
% Optional
%       'Color'       - color of the circle - Default black
%                       colors are 'yellow', 'magenta', 'cyan', 'red',
%                       'green', 'blue', and 'white'
%       'Filled'      - A true or false value - Default true
%       'Thickness'   - If the circle is not filled this specifies how
%                       thick to make the line - Default 0.5
%       'Style'       - A string indicating the style of the lines -
%                       Default '-' (Solid)
%                       Other styles are '--' (dashed), ':' (dotted),
%                       '-.' (dash-dotted)
%                       Applies to unfilled circles only
%
% Output
%       h     - Is a handle to the graphics object

h = Draw_Text(x, y, txt)
%
% Displays text on the graphics window
%
```

```
% Inputs
%        x                    - X coordinate of the lower left corner
%                               for the text unless HorizontalAlignment
%                               is changed
%        y                    - Y coordinate of the lower left corner
%                               for the text unless HorizontalAlignment
%                               is changed
%        txt                  - The text to print
% Optional
%        'Color'              - color of the circle - Default black
%                               colors are 'yellow', 'magenta', 'cyan'
%                               'red', 'green', 'blue', and 'white'
%        'Rotation'           - The rotation of the text - Default 0
%        'HorizontalAlignment'- [ {left} | center | right ]
%        'FontName'           - The name of the text - Default Helvetica
%        'FontSize'           - The size of the text - Default 10
%        'FontAngle'          - [ {normal} | italic | oblique ]
%        'FontWeight'         - [ light | {normal} | demi | bold ]
%        'VerticalAlignment'  - [ top | cap | {middle} | baseline |
%                               bottom ]
% Output
%        h     - Is a handle to the graphics object
```

Getting and Changing Object Value functions

```
Change_Object_Color(h,color)
%
% Changes the color on an object
%
% Inputs
%         h      - Is a handle to the graphics object
%         Color  - color of the circle - Default black
%                  colors are 'yellow', 'magenta', 'cyan', 'red',
%                 'green', 'blue', and 'white'

Delete_Object(h)
%
% Deletes an grphics object and removes it from the window
%
% Inputs
%         h     - Is a handle to the graphics object

Hide_Object(h)
%
% Hides an object on the graphics window but does not delete it
%
% Inputs
%         h     - Is a handle to the graphics window

Move_Object_By(h, xDist, yDist)
%
% Moves an object by an an x and y offset value
%
% Inputs
%         h     - Is a handle to the graphics object
%         xDist - The x distance to move the object by
```

```
%         yDist - The y distance to move the object by

Move_Object_To(h, x, y)
%
% Moves an object to a specific location, not valid for arcs and
%       polygons
%
% Inputs
%        h     - Is a handle to the graphics object
%        x     - X coordinate to move to
%        y     - Y coordinate to move to

h = Select_Object
%
% Returns a handle of the object selected on the graphics window
%
% Inputs
%
% Output
%        h     - Is a handle to the graphics object

Show_Object(h)
%
% If an object was made invible this function will make it visible again
%
% Inputs
%         h     - Is a handle to the graphics window
```

Miscellaneous functions

```
Set_Default_Value(property,value)
%
% Changes the default property values until the grphics window is closed
% and reopened
%
% Properities you can set
%        'Color'              - color of the object - Default black
%                               colors are 'yellow', 'magenta', 'cyan',
%                               'red', 'green', 'blue', and 'white'
%                               or colors are 'k', 'y', 'm', 'c', 'r',
%                               'g', 'b', and 'w'
%                               or [r,g,b], ex [1,1,0] for yellow
%        'Rotation'           - The rotation of the object (Text and
%                               Arcs)- Default 0
%        'Filled'             - A true or false value - Default true
%        'Thickness'          - If the circle is not filled this
%                               specifies how thick to make the line -
%                               Default 0.5
%        'Style'              - A string indicating the style of the
%                               lines - Default '-' (Solid)
%                               Other styles are '--' (dashed), ':'
%                               (dotted), '-.' (dash-dotted)
%                               Applies to unfilled circles only
%        'Rounded'            - A 2 value array with values between 0
%                               and 1 - Default [0,0] Indicates how
%                               rounded the corners are. [0,0] is no
```

```
%                               round, [1, 1] completely rounded like
%                               and elipse
%        'HorizontalAlignment'- [ {left} | center | right ]
%        'FontName'           - The name of the text - Default Helvetica
%        'FontSize'           - The size of the text - Default 10
%        'FontAngle'          - [ {normal} | italic | oblique ]
%        'FontWeight'         - [ light | {normal} | demi | bold ]
%        'VerticalAlignment'  - [ top | cap | {middle} | baseline |
%                               bottom ]

Clear_Graph_Window(h)
%
% Deletes all the object in the grphics window
%
% Inputs
%         h     - Is a handle to the graphics window - Default is
%                 current window

r = Get_Random(rmin,rmax,type)
%
% Calculates an random value
%
% Inputs
%        rmin          - The min value of the random number
%        rmax          - The max number of the random value
%        type          - A string of either 'integer' or 'float' -
%                        Default float
%
% Output
%        r     - A random number of the type and range specified above
```