Turtle Graphics

Turtle graphics refers to a means of controlling a graphical entity (a “turtle”) in a graphics window with x,y coordinates. A turtle can be told to draw lines as it travels, therefore having the ability to create various graphical designs. Turtle graphics was first developed for a language named Logo in the 1960s for teaching children how to program. Remnants of Logo still exist today.

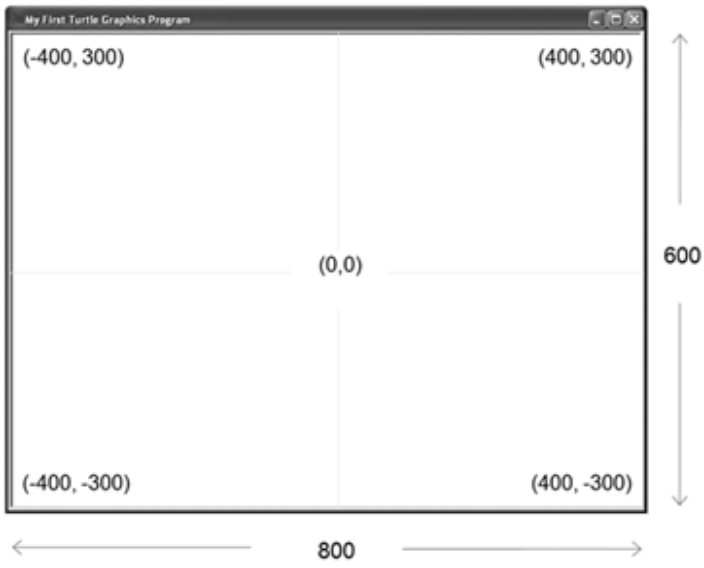
Python provides the capability of turtle graphics in the `turtle` Python standard library module. There may be more than one turtle on the screen at once. Each turtle is represented by a distinct object. Thus, each can be individually controlled by the methods available for turtle objects. You are being introduced to turtle graphics for two reasons - first, to provide a means of better understanding objects in programming, and second, to have some fun!

Creating a Turtle Graphics Window

The first step in the use of turtle graphics is the creation of a turtle graphics window (a *turtle screen*). The figure below shows how to create a turtle screen of a certain size with an appropriate title bar.

```
import turtle

# set window size
turtle.setup(800, 600)

# get reference to turtle window
window = turtle.Screen()

# set window title bar
window.title('My First Turtle Graphics Program')
```

Assuming that the `import turtle` form of import is used, each of the turtle graphics methods must be called in the form `turtle.`*`methodname`*. The first method called, `setup`, creates a graphics window of the specified size (in pixels). In this case, a window of size 800 pixels wide by 600 pixels high is created. The center point of the window is at coordinate `(0,0)`. Thus, x-coordinate values to the right of the center point are positive values, and those to the left are negative values. Similarly, y-coordinate values above the center point are positive values, and those below are negative values. The top-left, top-right, bottom-left, and bottom-left coordinates for a window of size `(800, 600)` are as shown in Figure 6-18. A turtle graphics window in Python is also an object. Therefore, to set the title of this window, we need the reference to this object. This is done by call to method `Screen`.



The background color of the turtle window can be changed from the default white background color. This is done using method `bgcolor`,

```
window = turtle.Screen()
window.bgcolor('blue')
```

The “Default” Turtle

A “turtle” is an entity in a turtle graphics window that can be controlled in various ways. Like the graphics window, turtles are objects. A “default” turtle is created when the setup method is called. The reference to this turtle object can be obtained by,

```
the_turtle = turtle.getturtle()
```

A call to `getturtle` returns the reference to the default turtle and causes it to appear on the screen. The initial position of all turtles is the center of the screen at coordinate (0,0), as shown below:

The default turtle shape is an arrowhead. (The size of the turtle shape was enlarged from its default size for clarity.) A turtle's shape can be set to basic geometric shapes, or even made from a provided image file.

Fundamental Turtle Attributes and Behavior

Recall that objects have both attributes and behavior. Turtle objects have three fundamental attributes: position, heading (orientation), and pen attributes. We discuss each of these attributes next.

Absolute Positioning

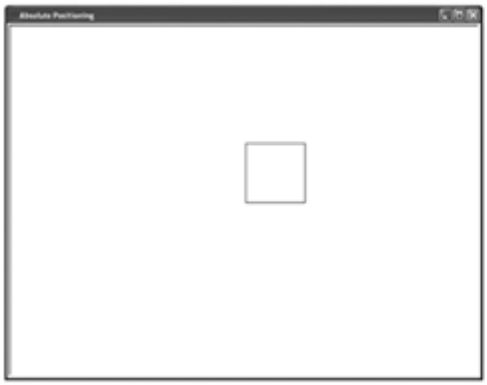
Method `position` returns a turtle's current position. For newly created turtles, this returns the tuple (0, 0). A turtle's position can be changed using *absolute positioning* by moving the turtle to a specific x,y coordinate location by use of method `setposition`. An example of this is shown here:

```
# set window title
window = turtle.Screen()
window.title('Absolute Positioning')

# get default turtle and hide
the_turtle = turtle.getturtle()
the_turtle.hideturtle()

# create square (absolute positioning)
the_turtle.setposition(100, 0)
the_turtle.setposition(100, 100)
the_turtle.setposition(0, 100)
the_turtle.setposition(0, 0)

# exit on close window
turtle.exitonclick()
```

The turtle is made invisible by a call to method `hideturtle`. Since newly created turtles are positioned at coordinates `(0, 0)`, the square will be displayed near the middle of the turtle window. To draw the square, the turtle is first positioned at coordinates `(100, 0)`, 100 pixels to the right of its current position. Since the turtle's pen is down, a line will be drawn from location `(0, 0)` to location `(100, 0)`. The turtle is then positioned at coordinates `(100, 100)`, which draws a line from the bottom-right corner to the top-right corner of the square. Positioning the turtle to coordinates `(0, 100)` draws a line from the top-right corner to the top-left corner. Finally, positioning the turtle back to coordinates `(0, 0)`draws the final line from the top-left corner to the bottom-left corner.

Turtle Heading and Relative Positioning

A turtle's position can also be changed through relative positioning. In this case, the location that a turtle moves to is determined by its second fundamental attribute, its heading. A newly created turtle's heading is to the right, at 0 degrees. A turtle with heading 90 degrees moves up; with a heading 180 degrees moves left; and with a heading 270 degrees moves down. A turtle's heading can be changed by turning the turtle a given number of degrees left, `left(90)`, or right, `right(90)`. The `forward` method moves a turtle in the direction that it is currently heading. An example of relative positioning is shown below:



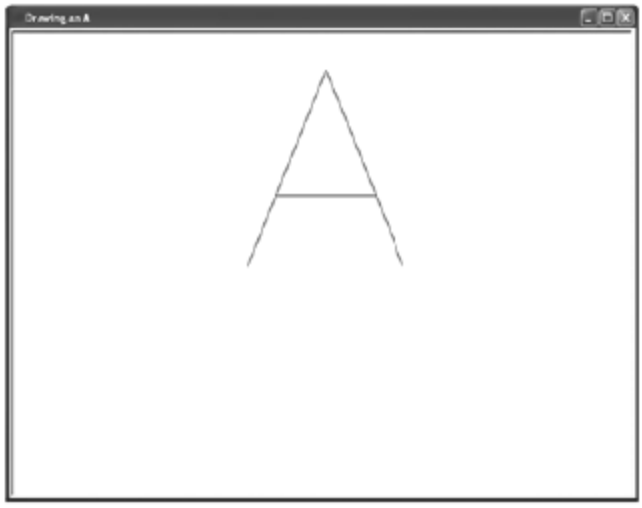
```
# set window title
window = turtle.Screen()
window.title('Relative Positioning')

# get default turtle and hide
the_turtle = turtle.getturtle()
the_turtle.hideturtle()

# create box (relative positioning)
the_turtle.forward(100)
the_turtle.left(90)
the_turtle.forward(100)
the_turtle.left(90)
the_turtle.forward(100)
the_turtle.left(90)
the_turtle.forward(100)

# exit on close window
turtle.exitonclick()
```

In this example, the turtle is controlled using relative positioning, drawing the same square as in the figure above. Since turtles are initially positioned at coordinates `(0, 0)` with an initial heading of 0 degrees, the first step is to move the turtle forward 100 pixels. That draws the bottom line of the square. The turtle is then turned left 90 degrees and again moved forward 100 pixels. This draws the line of the right side of the square. These steps continue until the turtle arrives back at the original coordinates `(0, 0)`, completing the square.

Methods `left` and `right` change a turtle's heading relative to its current heading. A turtle's heading can also be set to a specific heading by use of method `setheading`: `the_turtle.set- heading(90)`. In addition, method `heading` can be used to determine a turtle's current heading.

Pen Attributes

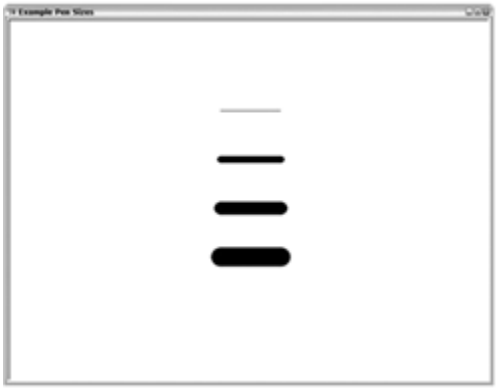
The pen attribute of a turtle object is related to its drawing capabilities. The most fundamental of these attributes is whether the pen is currently "up" or "down," controlled by methods `penup()` and `pendown()`. When the pen attribute value is "up," the turtle can be moved to another location without lines being drawn. This is especially needed when drawing graphical images with disconnected segments. Example use of these methods is given in the figure on the next page.



```
turtle.setup(800, 600)
window = turtle.Screen()
window.title('Drawing an A')

the_turtle = turtle.getturtle()
the_turtle.hideturtle()
the_turtle.penup()
the_turtle.setposition(-100, 0)
the_turtle.pendown()
the_turtle.setposition(0, 250)
the_turtle.setposition(100, 0)
the_turtle.penup()
the_turtle.setposition(-64, 90)
the_turtle.pendown()
the_turtle.setposition(64, 90)

turtle.exitonclick()
```

In this example, the turtle is hidden so that only the needed lines appear. Since the initial location of the turtle is at coordinate `(0, 0)`, the pen is set to “up” so that the position of the turtle can be set to `(-100, 0)` without a line being drawn as it moves. This puts the turtle at the bottom of the left side of the letter. The pen is then set to “down” and the turtle is moved to coordinate `(0, 250)`, drawing as it moves. This therefore draws a line from the bottom of the left side to the top of the “A.” The turtle is then moved (with its pen still down) to the location of the bottom of the right side of the letter, coordinate `(100, 0)`. To cross the “A,” the pen is again set to “up” and the turtle is moved to the location of the left end of the crossing line, coordinate `(-64, 90)`. The pen is then set to “down” and moved to the end of the crossing line, at coordinate `(64, 90)`, to finish the letter.

The pen size of a turtle determines the width of the lines drawn when the pen attribute is “down.” The pensize method is used to control this: `the_turtle.pensize(5)`. The width is given in pixels, and is limited only by the size of the turtle screen. Example pen sizes are shown here:



The pen color can also be selected by use of the `pencolor` method: `the_turtle.pencolor('blue')`. The name of any common color can be used, for example `'white'`, `'red'`, `'blue'`, `'green'`, `'yellow'`, `'gray'`, and `'black'`. Colors can also be specified in RGB (red/green/blue) component values. These values can be specified in the range 0–255 if the color mode attribute of the turtle window is set as given below,

```
turtle.colormode(255)
the_turtle.pencolor(238, 130, 238) # violet
```

This provides a means for a full spectrum of colors to be displayed.

Part II - Additional Turtle Attributes

In addition to the fundamental turtle attributes already discussed, we provide details on other attributes of a turtle that may be controlled. This includes whether the turtle is visible or not, the size (both demonstrated above), shape, and fill color of the turtle, the turtle's speed, and the tilt of the turtle. We will discuss each of these attributes next.

Turtle Visibility

As we saw, a turtle's visibility can be controlled by use of methods `hideturtle()` and `showturtle()` (in which an invisible turtle can still draw on the screen). There are various reasons for doing this. A turtle may be made invisible while being repositioned on the screen. In gaming, a turtle might be made invisible when it meets its "demise."

Turtle Size

The size of a turtle shape can be controlled with methods resizemode and turtlesize as shown here.

```
# set to allow user to change turtle size
the_turtle.resizemode('user')

# set a new turtle size
the_turtle.turtlesize(3, 3)
```

The first instruction sets the resize attribute of a turtle to '`user`'. This allows the user (programmer) to change the size of the turtle by use of method `turtlesize`. Otherwise, calls

to `turtlesize` will have no effect. The call to method `turtlesize` in the figure is passed two parameters. The first is used to change the *width* of the shape (perpendicular to its orientation), and the second changes its *length* (parallel to its orientation). Each value provides a factor by which the size is to be changed. Thus, `the_turtle.turtlesize(3, 3)`stretches both the width and length of the current turtle shape by a factor of 3. (A third parameter can also be added that determines the thickness of the shape's outline.)

There are two other values that method `resizemode` may be set to. An argument value of `'auto'` causes the size of the turtle to change with changes in the pen size, whereas a value of `'noresize'` causes the turtle shape to remain the same size.

Turtle Shape

There are a number of ways that a turtle's shape (and fill color) may be defined to something other than the default shape (the arrowhead) and fill color (black). First, a turtle may be assigned one of the following provided shapes: `'arrow'`, `'turtle'`, `'circle'`, `'square'`, `'triangle'`, and `'classic'` (the default arrowhead shape), as shown below:

The shape and fill colors are set by use of the `shape` and `fillcolor` methods,

```
the_turtle.shape('circle')
the_turtle.fillcolor('white')
```

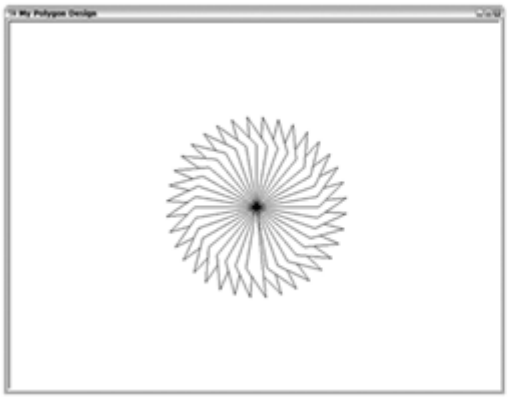
New shapes may be created and registered with (added to) the turtle screen's shape dictionary. One way of creating a new is shape by providing a set of coordinates denoting a polygon, as shown here:

```
turtle.setup(800, 600)
window = turtle.Screen()
window.title('My Polygon')
the_turtle = turtle.getturtle()

turtle.register_shape('mypolygon',
((0, 0), (100, 0), (140, 40)))

the_turtle.shape('mypolygon')
the_turtle.fillcolor('white')
```

In the figure, method `register_shape` is used to register the new turtle shape with the name `mypolygon`. The new shape is provided by the tuple of coordinates in the second argument. These coordinates define the polygon shown in the figure. Once the new shape is defined, a

turtle can be set to that shape by calling the `shape` method with the desired shape's name. The `fillcolor` method is then called to make the fill color of the polygon white (with the edges remaining black). It is also possible to create turtle shapes composed of various individual polygons called *compound shapes*.

The creation of this polygon may not seem too exciting, but the orientation of a turtle can be changed. In addition, a turtle is able to *stamp* its shape on the screen, which remains there even after the turtle is repositioned (or relocated). That means that we can create all sorts interesting graphic patterns by appropriately repositioning the turtle, as shown below:



```
turtle.setup(800, 600)
window = turtle.Screen()
window.title('My Polygon Design')
the_turtle = turtle.getturtle()

turtle.register_shape('mypolygon',
((0, 0), (100, 0), (140, 40)))
the_turtle.shape('mypolygon')
the_turtle.fillcolor('white')

for angle in range(0, 360, 10):
    the_turtle.setheading(angle)
    the_turtle.stamp()
```

Only a few lines of code are needed to generate this design. The for loop in the figure iterates variable angle over the complete range of degrees, 0 to 360, by increments of 10 degrees. Within the loop the turtle's heading is set to the current angle, and the stamp() method is called to stamp the polygon shape at the turtle's current position. By varying the shape of the polygon and the angles that the turtle is set to, a wide range of such designs may be produced.

Another way that a turtle shape can be created is by use of an image. The image file used must be a "gif file" (with file extension .gif). The name of the file is then registered and the shape of the turtle set to the registered name,

```
register_shape('image1.gif')
the_turtle.shape('image1.gif')
```

Turtle Speed

At times, you may want to control the speed at which a turtle moves. A turtle's speed can be set to a range of speed values from 0 to 10, with a "normal" speed being around 6. To set the speed of the turtle, the speed method is used,

`the_turtle.speed(6)`. The following speed values can be set using a descriptive rather than a numeric value,

```
10: 'fast'   6: 'normal'  3: 'slow'    1: 'slowest' 0: 'fastest'
```

Thus, a normal speed can also be set by `the_turtle.speed('normal')`. When using the turtle for line drawing only, the turtle will move more quickly if it is made invisible (by use of the `hideturtle` method).

Creating Multiple Turtles

So far, we have seen examples in which there is only one turtle object, the default turtle created with a turtle window. However, it is possible to create and control any number of turtle objects. To create a new turtle, the `Turtle()` method is used,

```
turtle1 = turtle.Turtle()
turtle2 = turtle.Turtle()
```
etc.

By storing turtle objects in a list, any number of turtles may be maintained,

```
turtles 5 []

turtles.append(turtle.Turtle())
turtles.append(turtle.Turtle()) etc.
```

Concepts and Procedures

1. A turtle screen is an 800-pixel wide by 600-pixel high graphics window. (TRUE/FALSE).

2. The three main attributes of a turtle object are_____ ,_____, and _____.

3. A turtle can be moved using either_____ or_____ positioning.

4. A turtle can only draw lines when it is not hidden. (TRUE/FALSE)

5. A turtle shape is limited to an arrow, turtle, circle, square, triangle, or classic (default) shape. (TRUE/FALSE)

6. What attribute of a turtle determines the size of the lines it draw?

a) Pen size
b) Turtle size

7. A turtle can draw in one of seven colors. (TRUE/FALSE)

8. A turtle can leave an imprint of its shape on the screen by use of the _____ method.

9. In order to create a new turtle object, the_____ method is called.

Problem Solving

1. Give a set of instructions to create a turtle window of size 400 pixels wide and 600 pixels high, with a title of `'Turtle Graphics Window'`.

2. Give a set of instructions that gets the default turtle and sets it to an actual turtle shape.

3. For each of the following method calls on turtle `the_turtle`, indicate in what part of the screen the turtle will be placed relative to the center of the screen.

a) `the_turtle.setposition(0, 0)`
b) `the_turtle.setposition(-100, 0)`
c) `the_turtle.setposition(-50, 0)`
d) `the_turtle.setposition(0, -50)`

4. For the following method calls on turtle `the_turtle`, describe the shape that will be drawn.

```
the_turtle.penup()
the_turtle.setposition(-100, 0)
the_turtle.pendown()
the_turtle.setposition(100, 0)
the_turtle.setposition(100, 50)
```

```
the_turtle.setposition(-100, 50)
the_turtle.setposition(-100, 0)
```

3. What color line will be drawn in the following?

```
turtle.colormode(255)
the_turtle.pencolor(128, 0, 0)
the_turtle.pendown()
the_turtle.forward(100)
```

4. What will be displayed by the following turtle actions?

```
the_turtle.pendown()
the_turtle.showturtle()
the_turtle.forward(25)
the_turtle.penup()
the_turtle.hide_turtle()
the_turtle.forward(25)
the_turtle.pendown()
the_turtle.showturtle()
the_turtle.forward(25)
```

Programming Problems

1. Give a set of instructions for controlling the turtle to draw a line from the top-left corner of the screen to the bottom-right corner, and from the top-right corner to the bottom-left corner, thereby making a big X on the screen. There should be no other lines drawn on the screen.

2. Using relative positioning, give a set of instructions for controlling the turtle to draw an isosceles triangle on the screen (that is, a triangle with two equal-length sides).

3. Give a set of instructions for controlling the turtle to draw the letter W using relative positioning.

4. Give a set of instructions for controlling the turtle to create three concentric circles, each of different color and line width.

5. Give a set of instructions that sets the turtle to an actual turtle shape, and moves it from the bottom of the screen towards the top, getting smaller as it moves along.

6. Give a set of instructions that moves the turtle with an actual turtle shape from the bottom of the screen toward the top, changing its fill color when it crosses the x axis of the grid coordinates.

7. Give a set of instructions to create your own polygon shape and create an interesting design with it.

8. Give a set of instructions so that the turtle initially moves slowly around the edge of the screen, then moves faster and faster as it goes around.

9. Give a set of instructions to create two turtle objects each with circle shape that move to various locations of the turtle screen, each stamping their circle shape of varying sizes and colors.