Unit 6: Coordinates with Rover — Skill Builder 1: Drive and Plot

In this lesson, you will learn about the Rover's coordinate system and how to drive to a particular point represented by an ordered pair of numbers.

Objectives:

- Understand the Rover's coordinate system and initial position and heading
- Make the Rover move to a certain point on the coordinate plane
- Plot Rover's points on the TI-84 Plus CE screen

The Rover has a 'built-in' coordinate system just like a Cartesian graphing system. When you **import ti_rover as rv**, the Rover's position on the coordinate grid is set to (0,0) and its heading is 0 degrees (pointing toward the positive x-axis, or 'east' on a map).

Teacher Tip: Rover also has a polar coordinate system!
Students might want to use a marker in the Rover to plot the points, but the marker will draw lines between the points, too.

In addition to Rover's coordinate grid, we can use the on-screen coordinate system that is available from the **ti_plotlib** module which is also imported into the Rover Coding template for just this reason. If you use these statements from **menu [math]> ti_plotLib**, you will get the screen to the right when you run:

```
import ti_plotlib as plt
plt.cls()
plt.grid(1,1,"dashed")
plt.window(-10,10,-7,7)
plt.axes("on")
plt.show_plot()
```

As we tell Rover to move around to points on the floor, we will also be plotting those points on the screen.

Teacher Tip: The Rover's coordinate unit is the same as the driving unit: 10cm.
The grid unit can be changed using **rv.grid_m_unit(scale_value)** found on the **ti_rover… Commands** menu.
The default scale_value is 0.1 m/unit (10cm).
To make the grid unit one inch use the scale_value 0.0254.

1. Start a new Python program using the Rover template and add the statement

   import ti_plotlib as plt

   found on **[math] ti_plotlib…**

2. Add the setup statements described earlier:

   plt.cls()
   plt.grid(1,1,"dashed")
   plt.window(-10,10,-7,7)
   plt.axes("on")

   and add the statement to plot Rover's starting position at the origin:

   plt.plot(0,0,"o")

   found on **[math] ti_plotlib Draw**. You will select the *dot type* from an additional sub-menu.

   Be sure to select the **plot** statement **5: plot(x, y, "mark")** and not 6: plot(xlist, ylist, "mark").

3. Depending on the space that you have, make Rover drive to a point in each of the four quadrants using **rv.to_xy()** found on **menu > TI Rover > Drive**.

   rv.to_xy(1,1)
   rv.to_xy(-1,1)
   rv.to_xy(-1,-1)
   rv.to_xy(1,-1)

You do not have to use **1**s and you don't even have to use the same numbers. But you do have to make sure that Rover visits all four quadrants.

Try your program now. Notice that Rover turns directly toward the position of the next point before moving there.

Teacher Tip: Rover might 'wiggle' a little when lining up for the move to the next point. This is Rover's gyroscope fine-tuning the heading before driving.

4. Add **plt.plot(x,y,"mark")** statements to your program to plot the points on the screen *right after* Rover reaches them. Try it now.
   Did the program perform as you expected?

Teacher Tip: Plotting on the screen is fast and easy using the **ti_plotlib** tools.

5. *Answer: No!* The points on the screen are plotted almost immediately when the program runs, and it takes Rover some time to drive to all four points.
   How do you sync the plotting with the driving?

6. Use the Rover function **rv.wait_until_done()** found on **[math] ti_rover > Commands** to pause the *program* while Rover is moving. You will need one of these functions for each driving point and order of the statements in the program does matter.

   Try it now. Where will you place those wait functions?

Teacher Tip: The **wait_until_done()** functions belong after the **rv_to_xy()** and before the **plt.plot()** for each point**:**

7. At the end of your program, send Rover to the 'home' position, (0, 0). Also use the statement

   rv.to_angle(0, "degrees")

   found on **[math] ti_over…** to have Rover point in its original direction ('east'). It's nice to put the toys away, right?

   And finally, to keep the graphic display on the screen add the statement

   plt.show_plot()

   which waits for the **[clear]** key to be pressed before returning to the Python Shell prompt.