TI-Innovator Rover Explorations

The TI-Innovator Rover can be used to learn simple and complex ideas in computer programming. This suite of activities contains 8 activities. Activity 6 is the only activity that requires completing the previous activity. While they do not need to be completed in succession, each subsequent activity is more complex than the previous.

Objectives:

Programming Objectives:

- Use variables to store information.
- Use selection statements to make decisions.
- Use iteration to repeat code.
- Use functions to modularize code.
- Use lists to store related data.
- Use libraries and library functions.

Key AP Computer Science Principles Standards:

- Represent a value with a variable (AAP-1.A)
- Represent a list or string using a variable (AAP-1.C)
- Write conditional Statements (AAP-2.H)
- Write nested conditionals (AAP-2.I)
- Select appropriate libraries or existing code segments to use in creating new programs (AAP-3 D)
- Write iteration statements (AAP-2.K)
- Write expressions that use list indexing and list procedures. (AAP-2.N)
- Write iteration statements to traverse a list. (AAP-2.O)
- Write statements to call procedures (AAP-3.A)
- Develop procedural abstractions to manage complexity in a program by writing procedures. (AAP-3 C)
- For generating random values, write expressions to generate possible values. (AAP-3 E)

This document contains Activities 1 and 2 of the 8 TI-Innovator Rover activities.

Activity 1: TI-Innovator Rover Basics

Students explore the menu options of the TI-Innovator Rover.
Students will learn how to drive forward and turn left/right.
Students will draw a regular polygon.

Activity 2: TI-Innovator Rover Basics Continued

Students will learn how to change the speed of the rover.
Students will use selection statements to make decisions.

Activity 1: TI-Innovator Rover Basics

Students explore the menu options of the TI-Innovator Rover.
Students will learn how to drive forward and turn left/right.
Students will draw a regular polygon.

1. Create a new program named “rovExploration”.

   Choose Rover Coding for the default type.

   *You can name your project “rovexploration” all in lowercase. However, using capital letters for the E makes it easier to read the name of the file. Often, programmers capitalize the first letter of each new word in a variable name. This form of naming is referred to as “camel case”. It doesn’t change how the program runs; it does however make it easier for the programmer to read.

2. The Rover Coding template automatically imports 5 libraries.

   Let's take a few minutes to discuss some key components in these libraries.

3. ti_rover library:

   Go to Menu > TI Rover

   Notice there are 7 main menu options. Spend a few minutes scrolling through the different options. Record below some of the things you find in the menus.

   a. import ti_rover as rv
      What do you think is the purpose of this command?

   b. Drive
      Explore the options for Drive. What are some commands in this menu?

   c. Inputs
      Explore the options for Inputs. What are some commands in this menu?

d. Outputs
Explore the options for Outputs. What are some commands in this menu?

e. Path
Explore the options for Path. What are some commands in this menu?

f. Settings
Explore the options for Settings. What are some commands in this menu?

g. Commands
Explore the options. What are some commands in this menu?

Teacher Notes:

Answer will vary. Students could share and compare answers in small or large group discussions.

4. Look a little closer at the import lines.
Three of the lines say "import *" at the end.
The other two say "as rv" and "as plt".

You might ask yourself, "Why are they different?".

When you import the math, ti_system, and time libraries you are importing many functions from each library. When using the functions, you will simply type the name. For example, if you need to take the square root of a value, you will use the sqrt() function from the math library. The code could be:
val = sqrt(number).

When you import ti_rover as rv, you create an object named "rv".
When you use commands from this library, they will start with rv.
For example, to move forward the command will be rv.forward(2).
To read and store a distance reading could look like
val = rv.ranger_measurement().

5. The first basic commands you'll explore are:
   forward, backward, left, right.

   Add the following lines to your code:

```
rv.forward(1)
rv.left()
rv.forward(1)
rv.right()
```

6. Connect your calculator to the TI-Innovator Rover.
   Make sure the TI-Innovator Rover is turned on.

   Execute your code.
   If there are any errors, fix your errors.

   Trace the path your TI-Innovator Rover drove below. Make sure to label the line segments with the appropriate distance traveled as well as angle measurements.

7. Modify your code. Draw two connected rectangles with the following dimensions.

8. Comment out the code you used to draw the rectangle above. Put a # symbol in front of each forward, left, or right command. This will allow you to refer to it later on if need be. Above the lines add the following comment:
   #two rectangles

9. You used the command rv.left() and rv.right().
   Without ***parameters***, they draw 90-degree angles.

   With ***parameters*** you can draw angles other than 90 degrees.
   The command rv.left(10) would turn 10 degrees left.

   How would you draw an equilateral triangle?

10. Add five to six lines of code to draw an equilateral triangle.
    Execute your code. Make sure it draws an equilateral triangle.

11. Comment out your triangle code using the # symbol.

12. Add code to draw a pentagon or a hexagon.

Teacher Notes:
Possible Extension Projects:

1.) Write a program to draw your initials in block letters using the TI-Innovator Rover.
If your first initial matches your last initial, draw your first initial and your middle initial.

2.) Write a program to draw the outline of a house. The roof should be either triangular or trapezoidal. The main portion of the house should be square or rectangular.

Activity 2:

Students will learn how to change the speed of the rover.

Students will use selection statements to make decisions.

1. Create a new project named “time_speed”.

   Select “Rover Coding” as the type.

   You named the project time_speed. Using the underscore, _, to join words together for files and variable names is called snake case. It is another method to join words together to create a variable or file name. You could have named the file “timespeed”, but notice using snake case for “time_speed” or camel case for “timeSpeed” makes it easier to read.

2. Add the line:

   rv.forward(1)

   Execute your program. Time how long it took your rover to travel 1 meter.

   Use your program to determine the speed of your rover.

   Currently, the approximate speed of your rover is: $\frac{meter}{seconds}$

   (Fill in the blanks above with the appropriate values.)

3. How far do you think your rover would travel if you told it to drive forward for 7 seconds?

4. Change your program to:

   rv.forward_time(7)

   Execute your program. How far does it travel in 7 seconds?

   Hint:
   Menu > TI_Rover > Drive > Drive with options

5. Let's explore a new way to drive forward.

The command rv.forward_time(time, speed, "unit")
takes three ***parameters***.
The speed specification accepts values from 0.14 m/s to 0.23 m/s.
The unit can be in grid unit / s, m/s, or rev/s.

Replace the line rv.forward_time(7) with the line rv.forward(2, 0.15, "m/s")

The rate the TI-Rover travels could vary based on the surface it travels. If the rover is on thick carpet, it might not travel the same rate or distance as it would if it were to travel on tiled flooring.

Execute your code.

How far did your rover travel?

6. Now that you know how to change the speed of the rover, write code to draw an equilateral triangle with side lengths of 0.5 meters.

Code your project so it takes approximately 30 seconds to draw the triangle. It will take a few seconds for the rover to make each turn.

Execute your code. If it draws the triangle too quickly, slow down your speed. If it draws too slowly, increase the speed.

Teacher Notes:
Answers will vary based on surface.

7. Let's modify your code to let the user determine the speed of the rover.

You will ask the user for two variables: speed and length.
By default, when you get input from the user, the information is stored as a ***string*** variable. That means the computer stores anything you type as characters. It will not treat numbers as numerical values.

When you ask the user for the values, you will use the ***float()*** function to convert the string variables to floating point decimal numbers.

Before your driving commands, type the following:
speed = float(input("Enter a speed: 0.14 to 0.23 m/s "))

8. Add another line to ask the user for the distance in meters to travel.

9. Execute your program.

   Do you have any errors?

   Does the program ask for the two values?

   We need to change a few more things before it actually uses these two variables in the code. What do you need to change?

10. You asked the user for the speed and distance, but you didn't change the code for the rv.forward.

    Modify all your rv.forward commands to rv.forward(dist, speed, "m/s").

    Execute your code. Does it work properly now?

    .

11. The speed can only be between 0.14 and 0.23.

    Execute your program. Try several speed values outside the bounds of 0.14 and 0.23 m/s.

    What happens when you enter values less than 0.14?

    What happens when you enter values more than 0.23?

12. Add some code to your project to prevent these types of mistakes.

    You have three coding choices.
    1. If the user enters invalid values, do nothing.
    2. If the user enters invalid values, set them to default values and preform the task with an error message to the user.
    3. Tell the user the information entered was invalid and request it again until the user enters the appropriate data.

    Which option do you think is better: 1, 2, or 3? Why?

    Can you code any of these options without help? If so, try it out.

13. Let's explore option 1: If the user enters invalid values, do nothing except print "invalid".

    Go to the top of the drive commands, but below the input boxes.

    Insert an if statement.
    Menu > Built-Ins > if..else

14. If the user speed is less than 0.14 or it is more than 0.23 print an error message, otherwise draw the triangle.

    The syntax for this will be:

```
if speed < 0.14 or speed > 0.23:
  print("Invalid")
else:
  rv.forward(dist,speed,"m/s")
  rv.left(120)
  rv.forward(dist,speed,"m/s")
  rv.left(120)
  rv.forward(dist,speed,"m/s")
  rv.left(120)
```

    Notice, the print statement is indented two spaces. The TI-Nspire CX II puts a diamond in place of the indented spaces. This is a useful feature to help you, the coder, with indentation. If the indentation is incorrect, your program will not compile.

15. Execute your program. Enter an invalid value for the speed. Does it print "Invalid" without driving? If you enter a valid, value does it drive a triangular path?

16. Modify your code.

    Modify the input statement for distance. Tell the user the values can only be between 0.5 and 1 meters.

    Modify your if statement so it includes the new settings.

Teacher Notes:

```
# Rover Coding
#================================
import ti_rover as rv
from math import *
import ti_plotlib as plt
from ti_system import *
from time import *
#================================
speed=float(input("Enter a speed: 0.14 to 0.23 m/s "))
dist=float(input("Enter a distance: 0.5 to 1m "))

if speed < 0.14 or speed > 0.23 or dist <0.5 or dist > 1:
  print("Invalid")
else:
  rv.forward_time(dist,speed,"m/s")
  rv.left(120)
  rv.forward_time(dist,speed,"m/s")
  rv.left(120)
  rv.forward_time(dist,speed,"m/s")
  rv.left(120)
```

17. Execute your program.

    Enter an invalid speed but a valid distance. Does it say “Invalid”?

    Enter an invalid distance but a valid speed. Does it say “Invalid”?

    Enter a valid distance and time. Does it draw the triangle?