Skill Builder 1: Moving Forward

Overview:

Students will write a program on their calculator to control the two motors on Rover. They are challenged to spin the motors fast, slow, clockwise and counter-clockwise. Next students are challenged to control the motors to drive in a straight line. As the final challenge, students are asked to drive a distance and then stop as close as possible to a barrier without touching that barrier.

Goals:

Students will:

1. learn to create, open, close, edit and run programs from the editor.
2. write a TI Python program that controls the speed and direction of the motors.
3. write a program that controls how far forward Rover will drive.
4. use proportional thinking, or pattern recognition, or linear modeling to predict how far Rover will travel based on their measurements.

Background:

Rover will move forward when a TI Python program rv.forward() function sends the forward command from the calculator to the TI-Innovator Hub connected to Rover. When the Hub interprets the command, it turns on Rover's motors. The longer the motors are turned on, the further Rover goes. When the motors spin slowly, Rover goes a short distance in a given amount of time. If the motors spin quickly, Rover goes a greater distance in the same amount of time. The distance Rover travels in a given amount of time is called **speed**. The speed of rover can be calculated by dividing the distance traveled by the amount of time. While automobile speed is calculated in miles per hour, Rover speed is in meters per second.

Statement	Example	Behavior
import module_name as name_space	`import ti_rover as rv`	Required for all TI Rover Python programs. Imports the ti_rover module into the Python program. The module provides the methods for controlling the Rover. Sets the current position of the RV as the origin and the heading as 0 degrees measured from the x-axis. The import ti_rover as rv statement is available from the Fns>Modul>ti_rover>Drive menu.
rv.motor_right(+/- power/speed, time) rv.motor_left(+/- power/speed, time)	rv.motor.left(75,5)	Left motor spins slowly clockwise for 5 seconds Motor power values range from 0 (off) to 255 (maximum). The sign of the power value determines the direction of the motor. Positive values spin clockwise and negative values spin counter-clockwise. Motor functions are available from the Fns>Modul>ti_rover>I/O>Outputs menu.
rv.motors("left wheel direction", left wheel power, "right wheel direction", right wheel power, time)	rv.motors("ccw",200,"cw",200, 10)	Both motors spin and Rover moves forward for 10 seconds The rv.motors() function send signals to control both left and right motors at the same time. Motor power values range from 0 (off) to 255 (maximum). Wheel directions are "CCW" (counter-clockwise) and "CW" (clockwise). Motor functions are available from the Fns>Modul>ti_rover>I/O>Outputs menu.
rv.forward(distance, "unit")	rv.forward(1.2, "m")	Rover drives forward 1.2 M at default speed of .20 M/S ** The rv.forward(distance,"unit") function is found on the Rover Drive with Options menu. Unit options are "grid units" (10 centimeters is the default), "m" (meters) and "revs" (wheel revolutions). rv.forward(distance, "unit") is found toward the bottom of the Fns>Modul>ti_rover>Drive menu.
rv.forward_time(time duration in seconds)	rv.forward_time(4.5)	Rover drives forward for 4.5 seconds at default speed of .20 M/S ** rv.forward(time) is found toward the bottom of the Fns>Modul>ti_rover>Drive menu.

Skill Builder 1: Moving Forward

rv.backward(distance, "unit")	rv.backward(1.2, "m")	Rover drives backward 1.2 M at default speed of .20 M/S ** Unit options are "grid units" (10 centimeters is the default), "m" (meters) and "revs" (wheel revolutions). rv.backward(time) is found toward the bottom of the Fns>Modul>ti_rover>Drive menu.

See the Rover module section beginning on page 26 of the Python Programming for the TI-84 Plus CE Python Graphing Calculator Guidebook for more programming information.
* The speed of the Rover will vary from the stated values depending on the floor surface. Some surfaces cause Rover to move more slowly. If accuracy is important, the speed should be measured by a method similar to the one in this activity

Setup:

Students may work in groups of two or three. Choose an area to work that has at least 2 meters of clear uniform floor space. Carpeted flooring is less desirable than tile. If needed, driving mats may be used for a driving surface. For the 2nd and 3rd challenges, a driving lane is setup using tape or cones.

Materials

- Miniature traffic cones
- Masking tape
- Drive mats or hard, flat, clean surfaces
- TI-84 Plus CE graphing calculator with On-Ramp to Robotics Unit 1 Student TI-84 Plus CE Python program files loaded for use in the Python app.
- Calculator unit-to-unit cable (USB mini A to USB mini B cable)
- TI-Innovator™ Hub
- TI-Innovator™ Rover.
- Student "Challenge" cards have been provided and can be printed/ cut into individual task cards, and distributed to students (optional)

Student Activity:

Sit in small groups with your calculator and supplies for this activity. Practice the guidance modeled by your teacher.

Teacher Activity:

Review and introduce the calculator, Hub, and Rover commands needed for this activity. In preparation for the coding on this activity, refer to Meet the Rover with Geometry Challenges (download the TI-84 Plus CE Python PDF), Unit 1 Getting Started with Python Skill Builder 1 of the 10 minutes of code activities and Unit 4 Rover's Driving Features Skill Builder 1 of the 10 minutes of code with TI-Innovator activities.

- Download the student and teacher files from the project web page.
- Use CE Connect to download the student Python files to the handhelds. The student files include the necessary module import statements and some additional scaffolding of the activity.
- Open the student Python program file for each challenge in the Python Editor. Have the students enter the necessary statements to drive the challenge.
- Attach Rover.

Skill Builder 1: Moving Forward

THE ON-RAMP TO ROBOTICS

Challenge 1: Write a program that makes the left wheel spin slowly clockwise for 5 seconds. Now modify your program to make the wheel spin faster and then in the opposite direction. Try to make the wheel turn for a longer time. Try the other wheel. Be sure to flip Rover on its back before running your program.

Guidance during challenge 1:

- Introduce the TI Python editor and how to run a program to students.
  1. Basic navigation on the calculator.
  2. Saving and opening files.
  3. Editing new and existing programs.
  4. Running programs.
  5. Editing program features.
  6. See the example program for challenge 1 below.
- The wheel-spinning activity is intended to be an introduction to Rover programming. Students should work with a partner and encourage them to explore how to control each motor's direction and speed. Students should remove their calculator from the Rover and place Rover on its back to complete challenge 1, be sure the calculator is plugged into Rover and Rover's power switch is set to "ON".
- Discuss with students what clockwise means and the frame of reference for that designation. Ask them to consider the Earth's rotation. Is it clockwise (CW) or counter-clockwise (CCW)? Is that designation using a frame of reference viewed from the North or South Pole? Is there an absolute frame of reference? Or is it a relative choice? The wheels of rover are designated as CW or CCW based on looking at the wheel from outside Rover and along the motor shaft. Demonstrate to students for a particular object's rotation if it is designated as a CW motion, observing from the opposite direction along the axis of rotation, the same motion will be designated as CCW. Ask students what direction each wheel should rotate in order to go forward?
- For students who finish quickly, have them place a piece of tape on the wheel and count the number of rotations for the time interval. Challenge them to calculate the revolutions per minute (RPM) of the wheel.
- Motor functions are available from the Fns>Modul>ti_rover>I/O>Outputs menu.

Program: ORSB1C1T

```
import ti_rover as rv
rv.motor_left(100,5)
```

Challenge 2: Write a program to make the Rover drive a straight path down the lane that is setup in your classroom. Do you think you could backup down the lane?

Guidance during challenge 2:

- This challenge is a variation of the prior challenge. Students will need to control both motors at the same time. The trick is, since the motors are mounted in opposite directions, the frames of reference for the two motors are opposite. As a result, to move forward, the left motor must turn in the CCW while the right must turn a CW direction. Do not offer this to the students, let them discover it instead.

- In addition, each motor is manufactured with variation; as a result, the motors will spin at different rates for the same power setting. This causes Rover to pull either left or right of a straight line depending on the unit. When this happens, ask the students which wheel must compensate by turning faster to straighten out the pulling? Then ask them what parameter in the motor command needs to be adjusted to change the speed of that wheel. Let students discover the correct answer.
- Fortunately, all of this is taken care of in more advanced ROVER commands, such as rv.forward(). Since both motors are required to turn concurrently to go forward, the individual motor commands will not work since they are executed sequentially. Instead use the MOTORS command which will set both motors at the same time. You could allow your students to discover the need for this new command. See example program for Challenge 2 below.
- Motor functions are available from the Fns>Modul>ti_rover>I/O>Outputs menu.

Program: ORSB1C2T

```
import ti_rover as rv
rv.motors("ccw",200,"cw",200,20)
```

Challenge 3: Write a program to drive your Rover straight down the challenge lane and make it stop as close to the final target as possible without making contact. You will use the rv.forward(distance,"unit").

Predict how long it will take your Rover to reach the mark. Use rv.forward(time) to drive your Rover.

Can your Rover be the closest?

Guidance during challenge 3:

- In the previous challenges, students experienced the difficulty of getting their Rover to drive a straight line. In this challenge, students will use the rv.forward(distance,"unit) function found toward the bottom of the Fns>Modul>ti_rover>Drive menu, which makes use of the built-in rotary encoders on Rover to automatically drive a straight line.
- Students are asked to make a prediction of the time needed to drive their Rover and stop at a predetermined distance. Discuss with your students the idea of prediction using patterns, proportional reasoning or linear extrapolation.
- In order to accomplish this challenge, students will need to run some tests of the Rover using the rv.forward(distance,"unit") function. Have the students record the times required to travel three or four uniform distances and then set the challenge distance beyond the last measured distance (see example on the following page)
- Students should program the distances into the rv.forward(distance,"unit") function and then use a stopwatch to time the drive path. They should record the ordered pairs of time and distance on paper. Students will then use that data to predict the time needed for Rover to drive the challenge distance using the rv.forward(time) function.
- There are several approaches students could use to make this prediction. They could use the skill of identifying a pattern in their data table and extend the pattern to the challenge distance. Alternatively, they could use proportional thinking, or they could plot a graph of time vs. position and extrapolate a linear model. The teacher is encouraged to let the student decide. The teacher could ask questions to direct a student who is having trouble arriving at a schema for solving the problem.

- Below are example programs and possible solutions.

Program: ORSB1C3T Drive for a distance with unknown time:

```
import ti_rover as rv
rv.forward(.4,"m")
```

Program: ORSB1C4T Challenge 3 Part 2: Drive for a time to for a measured distance

```
import ti_rover as rv
rv.forward_time(11)
```

Identifying Patterns:

$$\Delta t_1 = 4.85 - 2.43 = 2.42$$

$$\Delta t_2 = 9.68 - 7.17 = 2.51$$

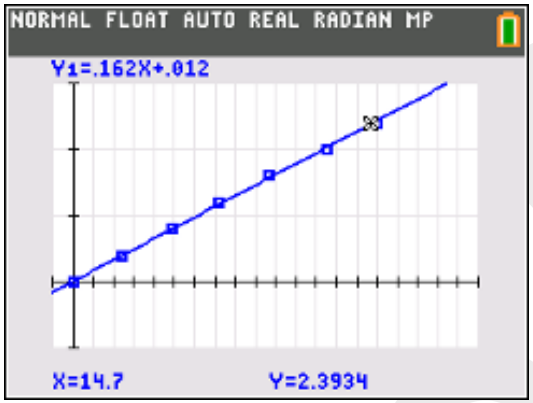
$$t = 12.5 + 2.47 = 14.97$$

Time (sec)	Distance (M)
0	0
2.43	.4
4.85	.8
7.17	1.2
9.68	1.6
12.5	2.0
~15	2.4

Proportional Reasoning:

$$\frac{12.5}{2.0} = \frac{X}{2.4},\ X = 2.4 \times \frac{12.5}{2.0},\ X = 15$$

Graphing linear model:



$$2.4 = .162X + .012$$

$$X = \frac{2.4 - .012}{.163}$$

$$X = 14.7$$