

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 5 and 6 of the 8 total TI-Innovator Rover activities.

Activity 5: Regular Polygons Revisited

Students will use loops to repeat code.

Activity 6: Quiz Race 2 Slope

Students will use the random integer function from the random library to generate random integers.

Students will use loops to repeat code.

Students will use selection statements to make decisions.

Students will use the eval() function to evaluate mathematical expressions from input.

Students will use the fabs() function to trouble shoot roundoff errors.

Students will fix a runtime error.



Activity 5: Regular Polygons Revisited

Students will use loops to repeat code.

- Activity 4: Regular Polygons used the following selection to draw regular triangles, squares, pentagons, and hexagons.

```

if length <=0:
    print("Invalid side length")
elif sides<3 or sides > 6:
    print("Invalid polygon choice")
elif sides==3:
    rv.forward(length)
    rv.left(120)
    rv.forward(length)
    rv.left(120)
    rv.forward(length)
elif sides==4:
    rv.forward(length)
    rv.left(90)
    rv.forward(length)
    rv.left(90)
    rv.forward(length)
    rv.left(90)
    rv.forward(length)
elif sides==5:
    rv.forward(length)
    rv.left(72)
    rv.forward(length)
    rv.left(72)
    rv.forward(length)
    rv.left(72)
    rv.forward(length)
    rv.left(72)
    rv.forward(length)
else:
    rv.forward(length)
    rv.left(60)
    rv.forward(length)
    rv.left(60)
    rv.forward(length)
    rv.left(60)
    rv.forward(length)
    rv.left(60)
    rv.forward(length)
    rv.left(60)
    rv.forward(length)

```

That was A LOT of repetitive code. How could you code more efficiently? Let's look for a pattern. Complete the table below.

Number of Sides	# rv.forward commands	Turn Angle
3	3	120
4	4	90
5	5	72
6		
7		
8		
N		



2. Look back over your table in step 1. Notice, the number of sides match the number of times the TI-Innovator Rover drives forward. You could use a loop to repeat this action.

The code

```
for c in range(sides):  
    rv.forward(length)
```

will repeat the `rv.forward(length)` a total number of **sides** times.

If you use the relationship

$$\text{turn angle} = \frac{360}{\text{sides}}$$

The code

```
for c in range(sides):  
    rv.forward(length)  
    rv.left(360/sides)
```

will replace ALL the lines of code it took to draw the triangle, square, pentagon, and hexagon. It also lets you draw ANY regular polygon.

3. Create a new program named “regPoly2”.
Choose Rover Coding for the default type.
4. Ask the user for the number of sides. Store the value as an *integer* in a variable named **sides**.
5. Ask the user for the length of a side. Store the value as a *double* in a variable named **length**.
6. If the user enters an integer value less than 3 for the number of sides, display a message that says “Invalid”. Otherwise, add the three lines of code for the *for loop* in step 2 to draw the regular polygon.
7. Execute your code. Debug any errors you find.

Activity 6: Quiz Race 2 Slope

Students will use the random integer function from the random library to generate random integers.

Students will use loops to repeat code.

Students will use selection statements to make decisions.

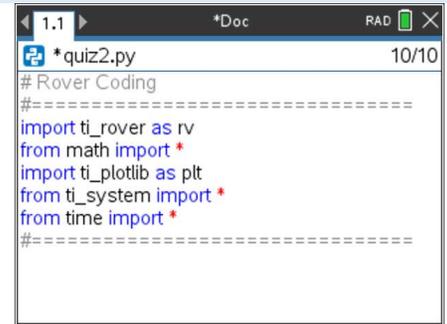
Students will use the eval() function to evaluate mathematical expressions from input.

Students will use the fabs() function to troubleshoot roundoff errors.

Students will fix a runtime error.

1. Create a new program named “quiz2”.

Choose Rover Coding for the default type.

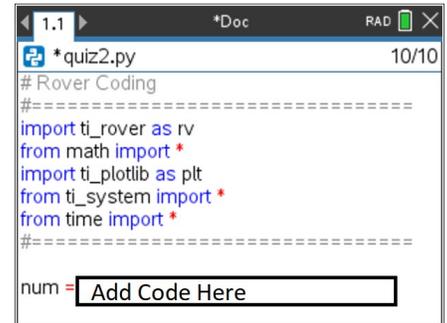


```

1.1 *Doc RAD 10/10
*quiz2.py
# Rover Coding
#-----
import ti_rover as rv
from math import *
import ti_plottlib as plt
from ti_system import *
from time import *
#-----
    
```

2. This first version of the game will ask the user for the number of practice questions.

Since this number will be a value used for calculations, store it as a number. Should you store the value as an integer or as a float?



```

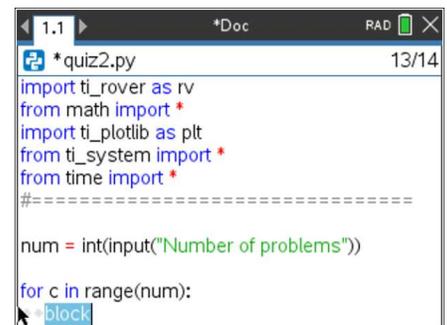
1.1 *Doc RAD 10/10
*quiz2.py
# Rover Coding
#-----
import ti_rover as rv
from math import *
import ti_plottlib as plt
from ti_system import *
from time import *
#-----
num = 
    
```

3. You will use a **for loop** to ask num random questions.

Add the line:

for c in range(num):

Notice the next line is *indented* two spaces. On the TI-Nspire CX II, two diamonds illustrate the indentation.



```

1.1 *Doc RAD 13/14
*quiz2.py
import ti_rover as rv
from math import *
import ti_plottlib as plt
from ti_system import *
from time import *
#-----
num = int(input("Number of problems"))
for c in range(num):
    block
    
```



4. Generate and display two random integers x_1 and y_1 . These values should be integers between -10 and 10.

```
1.1 *quiz2.py 15/15
import ti_plottlib as plt
from ti_system import *
from time import *
#-----
num = int(input("Number of problems"))

for c in range(num):
    x1=randint(-10,10)
    y1=randint(-10,10)
    print(x1,y1)
```

5. Run your program on the TI-Innovator Rover.

You get the following **runtime error**:

`NameError: name 'randint' isn't defined`

Why did you get this error?

```
1.1 1.2 *Doc 11/11
Python Shell
>>>#Running quiz2.py
>>>from quiz2 import *
Number of problems 5
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  File "C:\Users\bbyer\AppData\Roaming\Texas Instruments\TI-Nspire CX CAS Premium Teacher Software\python\doc3\quiz2.py", line 13, in <module>
NameError: name 'randint' isn't defined
>>>
```



- To fix the runtime error, add the random library to the top of your code.

```
from random import *
```

```

1.1 1.2 *Doc RAD 9/16
*quiz2.py
# Rover Coding
#-----
import ti_rover as rv
from math import *
import ti_plotlib as plt
from ti_system import *
from time import *
from random import *
#-----

num = int(input("Number of problems "))

```

- Execute your code.

Enter 5 for the number of values.

Did you get five random pairs of numbers between -10 and 10?

Sample Run:

```

1.1 1.2 *Doc RAD 9/9
Python Shell
>>>#Running quiz2.py
>>>from quiz2 import *
Number of problems 5
-7 0
-1 -10
1 -3
-2 -5
-5 3
>>>|

```

- Remove the print statement.

Generate random integer values between -10 and 10 for x2 and y2.

```

1.1 1.2 *Doc RAD 18/18
*quiz2.py
from random import *
#-----

num = int(input("Number of problems "))

for c in range(num):
    x1=randint(-10,10)
    y1=randint(-10,10)
    x2=
    y2=

```

- What is the formula for calculating slope?

There is potential for a **runtime error** when the program is running. Can you think of a scenario where calculating the slope would cause a runtime error?

10. Remember: $\text{slope} = \frac{y_2 - y_1}{x_2 - x_1}$

If x_1 equals x_2 you will divide by zero. This will cause a runtime error.

To avoid this, you could add the lines:

```
if x1 == x2:
    x1= randint(-10,10)
```

However, there is a chance, x_1 could again match x_2 .
What is the probability x_1 would match x_2 again?

11. To avoid this problem, use a while statement.

While x_1 matches x_2 , regenerate x_1 .

```
num = int(input("Number of problems: "))
for c in range(num):
    x1=randint(-10,10)
    y1=randint(-10,10)
    x2=randint(-10,10)
    y2=randint(-10,10)
    while x1 == x2:
        x1=randint(-10,10)
    # Add Code
```

12. Ask the user for the slope. Make sure you display both points.

Store the value as a float. Make sure you use the eval() function so the user can enter the slope as a fraction.

****Hint:** To display string and integer variables, don't forget str().
For example "(" + str(x) + ", "

```
num = int(input("Number of problems: "))
for c in range(num):
    x1=randint(-10,10)
    y1=randint(-10,10)
    x2=randint(-10,10)
    y2=randint(-10,10)
    while x1 == x2:
        x1=randint(-10,10)
    slope=eval(input("Find the slope between (" + str(x1) + "," + str(y1) + ") and (" + str(x2) + "," + str(y2) + "): "))
    print("Slope: " + str(slope))
```

13. Run your code.

Enter 3 for the number of problems.

Does it generate a random pair of integers then wait for you to enter a slope?

Does it repeat this process two more times?

If you have any errors, fix them.

```
>>>#Running quiz2.py
>>>from quiz2 import *
Number of problems: 3
Find the slope between (10,-10) and (0,4): 14/10
Find the slope between (6,2) and (5,7): 5/-1
Find the slope between (2,-5) and (-4,-6): -1/-
>>>
```

A sample run is provided on the right. Your points will be different since the generator is selecting random integers. The circled values are user input to advance to the next line.



14. If the user's slope equals the slope between the two points, the rover should drive forward. Otherwise, it should move back 0.25 meters and display the correct slope.

You could write:

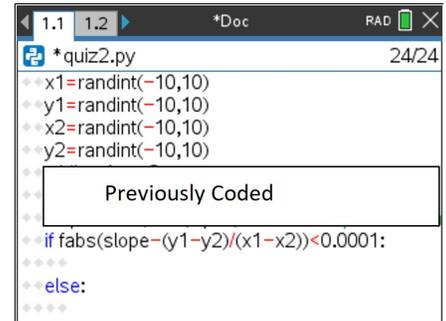
```
if slope == (y1-y2)/(x1-2):
    #move forward
else:
    #move backward
    #display the correct answer
```

However, a **roundoff error** might erroneously evaluate to false. Give an example of a slope that might lead to a **roundoff error**.

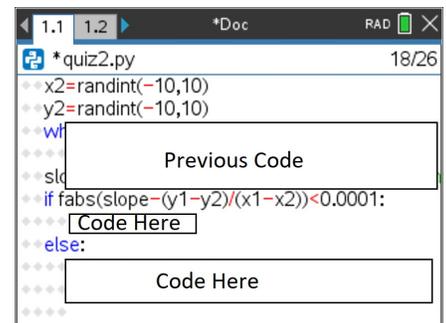
- 15.

16. To avoid a roundoff error, you will write the code:

```
if fabs(slope - (y1-y2)/(x1-x2)) < 0.0001:
    #move forward
else:
    #move backward
    Display the correct slope.
```



17. Add the move forward / move backward code.



18. Execute your code.

Try three problems. Answer at least one correctly and one incorrectly.

Fix any errors.

A sample run is displayed on the right

The first problem was answered correctly. The TI-Innovator Rover moved forward 0.5 meters.

The second question was answered incorrectly. The TI-Innovator Rover moved backwards 0.25 meters. The correct answer, $-1\frac{1}{4}$ was displayed.

The last question was answered correctly. The TI-Innovator Rover moved forward 0.5 meters.

```

Python Shell 8/8
>>>#Running quiz2.py
>>>from quiz2 import *
Number of problems: 3
Find the slope between (2,-4) and (9,-10): -6/7
Find the slope between (-10,-4) and (-6,7): 1/2
Slope was: -11 / -4
Find the slope between (-10,6) and (9,9): 3/19
>>>
    
```

19. You're now ready for a race! Challenge another group to a TI-Innovator Rover slope race. Can you answer all the questions correctly before another group?

20. Modify your game.

Instead of asking for the "Number of problems", ask for the "Race Length".

21. Create a new variable named distance that starts at 0.

22. Change the "for c in range(num):"

to a while loop. While the distance traveled is less than the number entered, generate new questions.

23. Lastly, if correct, add 0.5 to distance, otherwise subtract 0.25.

24. Execute your program. Answer at least one question incorrectly. Does the program terminate when you have traveled the desired distance?