Putt-Putt with Python!

In this lesson, you will create a golf game using the Python Editor. Your game code will randomly generate a location for the golf ball and the hole. To take a shot, the user will enter a linear equation in point-slope form that models the path for the shot. The program will plot the golf ball, hole and shot.

Objectives:

Programming Objectives:

- Use the input function and a variable to collect and store data from a user
- Use the randint() function to generate random integers.
- Use the plot library to plot points
- Use the plot library to plot a line segment

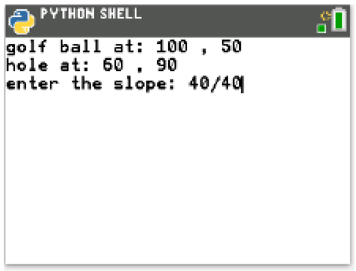
Math Objectives:

- Use the point slope form of a line to plot a line segment

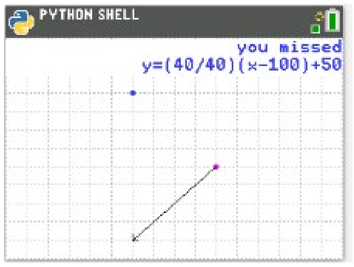
Math Course Connections: This activity is recommended for Pre-Algebra or Algebra 1**.**

In this project, you will create a golf game. The golf game gives the user two random points, one for the golf ball and one for the golf hole. The user will enter a value for the slope. The program will plot the equation of the line through the golf ball x1,y1 using the slope.

Incorrect Shot

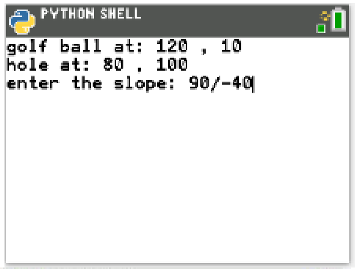


```
golf ball at: 100 , 50
hole at: 60 , 90
enter the slope: 40/40
```

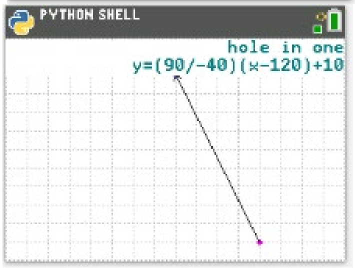


```
you missed
y=(40/40)(x-100)+50
```

Correct Shot



```
golf ball at: 120 , 10
hole at: 80 , 100
enter the slope: 90/-40
```



```
hole in one
y=(90/-40)(x-120)+10
```

1. To create the golfing game, you will need to plot two points, the golf ball and the hole. First, brainstorm the information you will need to plot the ball and hole.
    1.)

    2.)

    3.)

    4.)

    5.)

2. If possible, compare your list to a classmate's list.

   What do you have in common?

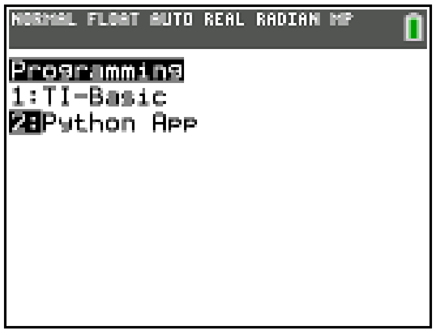
   Are there items you forgot?

3. The first step will be to create a python document.

   Open up the python editor:
   [prgm]

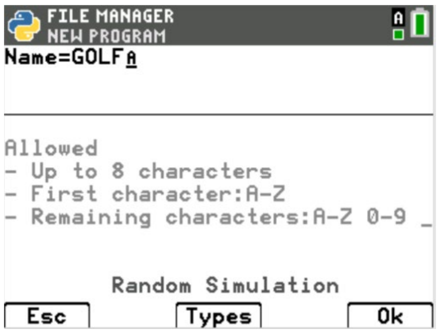
   [2] Python APP

   [zoom] New



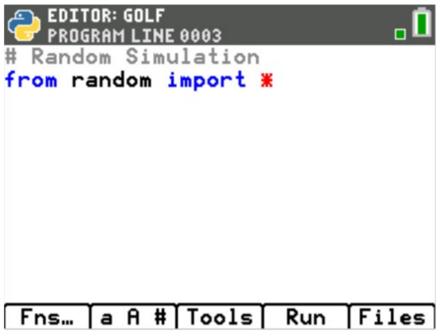
   Create a new python project named "GOLF".
   Select "Random Simulation" from the types menu.

   [graph] Ok



```
Name=GOLF

Allowed
- Up to 8 characters
- First character:A-Z
- Remaining characters:A-Z 0-9 _

Random Simulation
```

   Verify your editor has the same lines of code as the picture to the right.

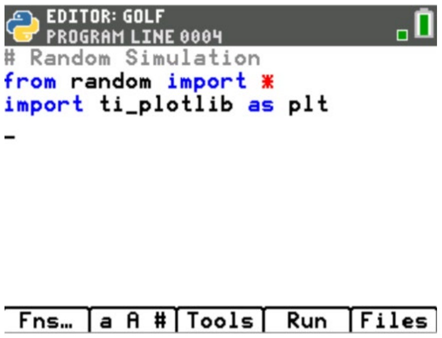


```
# Random Simulation
from random import *
```

4. The random library contains the randint function for generating random integer golf ball locations.

   You will need one more library, the plot library.
   This library contains the plotting functions.

   Place your cursor on the line below the **from random import ***
   Fns → Modul → ti_plotlib → import ti_plotlib as plt



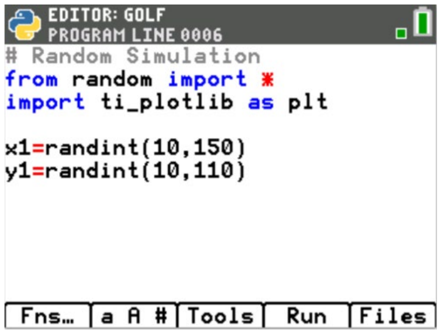
```
EDITOR: GOLF
PROGRAM LINE 0004
# Random Simulation
from random import *
import ti_plotlib as plt
_
```

5. ***You need an x value and a y value to plot a golf ball location.***

   The plot window will have a domain of [0,160] and a range of [0,120]. To keep the point on the page, generate random x1 and y1 values within the domain and range.

   The screen on the right generates x1 values between [10 and 150] and y1 values between [10,110].

   **randint can be found under the menu.
   Fns → Modul → Random → randint



```
EDITOR: GOLF
PROGRAM LINE 0006
# Random Simulation
from random import *
import ti_plotlib as plt

x1=randint(10,150)
y1=randint(10,110)
```

6. Add a print line to print the golf ball location to the screen.
   Fns → I/O→ print

   print(“golf ball at: ”, x1, “,”, y1)

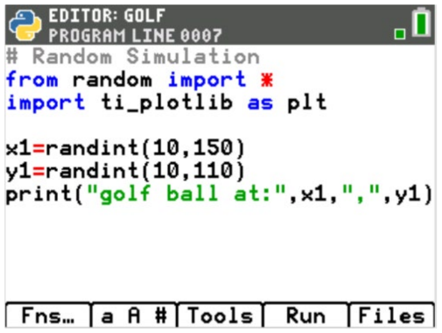
   Notice the string “golf ball at: ” and “,” are in green.
   All string values are displayed in green.

   Execute your code: Run

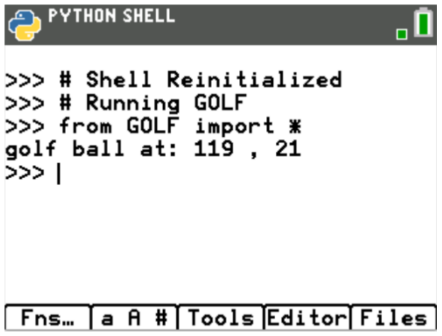
   The golf ball location should appear on the screen.
   The example at the right generated a point at 119,21.

   Execute the code a few more times.
   Verify the golf ball is placed at a different place each time.

   Tech Tip: The menu [a A #] obtained by pressing the [window] might make it easier to type your text “golf ball at”.



```
EDITOR: GOLF
PROGRAM LINE 0007
# Random Simulation
from random import *
import ti_plotlib as plt

x1=randint(10,150)
y1=randint(10,110)
print("golf ball at:",x1,",",y1)
```



```
PYTHON SHELL

>>> # Shell Reinitialized
>>> # Running GOLF
>>> from GOLF import *
golf ball at: 119 , 21
>>> |
```

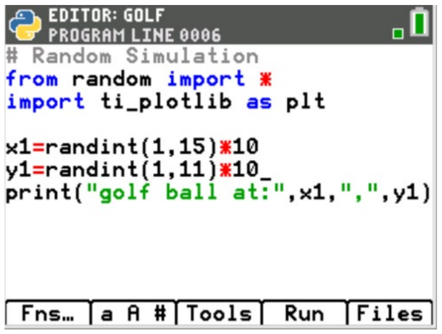
Teacher Tip: This lesson uses double quotes for string values. Python allows for both double and single quotes. However, if a “string” starts with double quotes it must end with double quotes. Likewise, if it starts with a single quote is must end with a single quote.

7. You may choose to keep x1 and y1 as any integer between [10,150] and [10,110] or you may choose to make them multiples of 10 for easier linear calculations when making your golf swing.

   If you modify your code to:

   x1 = randint(1,15)*10
   y1 = randint(1,11)*10

   your golf ball coordinates will always be multiples of 10.



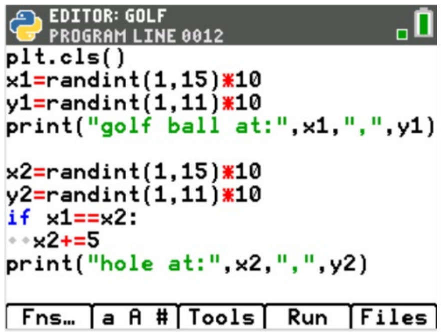
```
# Random Simulation
from random import *
import ti_plotlib as plt

x1=randint(1,15)*10
y1=randint(1,11)*10_
print("golf ball at:",x1,",",y1)
```

8. Add in three lines of code to generate x2, y2 and print the location of the golf hole.

   Tech Tip: The menu [Tools] obtained by pressing the [zoom] button will allow you to copy and paste lines. You might find it easier to modify a line rather than add the entire line.

9. There is a 1/15 chance the x1 and x2 values will match, creating an undefined slope. To fix this problem, add the lines:
   if x1 == x2:
   x2 += 5

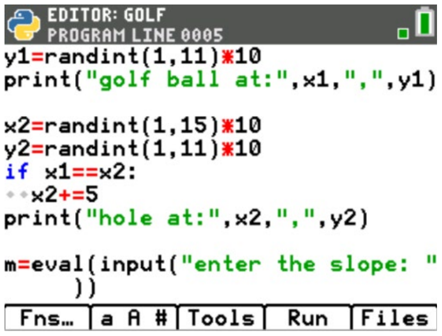
   This will ensure the points don’t match.



```
plt.cls()
x1=randint(1,15)*10
y1=randint(1,11)*10
print("golf ball at:",x1,",",y1)

x2=randint(1,15)*10
y2=randint(1,11)*10
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)
```

10. The user will enter the slope as a fraction. To store the fraction as an evaluated float value, use the eval function with the input function.
    Fns → I/O → eval
    Fns → I/O → input

    m = eval(input(“enter the slope: ”))



```
y1=randint(1,11)*10
print("golf ball at:",x1,",",y1)

x2=randint(1,15)*10
y2=randint(1,11)*10
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)

m=eval(input("enter the slope: "
     ))
```

11. Execute the code: Run

    The program first displays the coordinates for the golf ball and the hole. Next, it asks the user to input the slope.

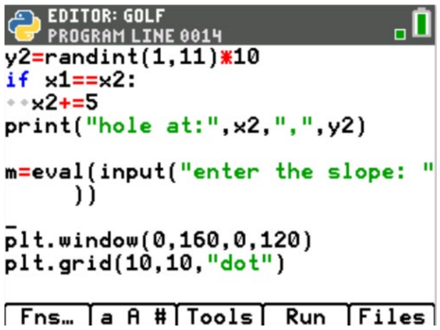
    The lines of code are executed sequentially, meaning in the order they appear on the screen. Before you added the input line, the two print statements appeared.

    After the user enters a value for the slope and presses enter, the program terminates.

12. Now, you will set up the plot window. The domain will be [0,160]. The range will be [0,120]. Because the window is so large, we will set the grid scale to count by 10.

    Add the two lines
    plt.window(0,160,0,120)
    plt.grid(10,10,"dot")

    *Both commands can be found using the path
    Fns → Module → ti_plotlib



```
y2=randint(1,11)*10
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)
m=eval(input("enter the slope: "
     ))
plt.window(0,160,0,120)
plt.grid(10,10,"dot")
```

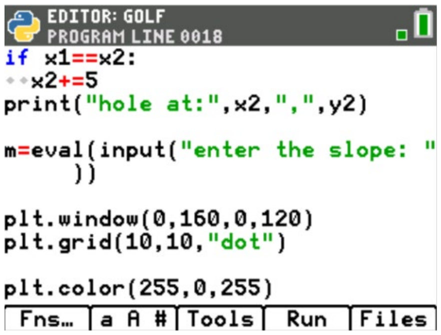
13. You can plot a point in any color imaginable.
    The command from the plot library is color(red, green, blue).

    The values for each color parameter can be any integer from 0-255.

    Example colors:

red	color(255,0,0)
green	color(0,255,0)
magenta	color(255, 0, 255)
plum	color(221,160,221)

    The color function can be found using:
    Fns → Module → TI-PlotLib → Draw → color



```
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)
m=eval(input("enter the slope: "
     ))
plt.window(0,160,0,120)
plt.grid(10,10,"dot")
plt.color(255,0,255)
```

Choose any values you like for red, green and blue as long as they are integers between 0-255.

14. To plot the point, use the plt.plot (x, y, mark) command.
    Fns → Modul → TI-PlotLib → Draw → plot

    Use your random integer variables x1 and y1 for the coordinates.

    plt.plot(x1, y1, “o”)

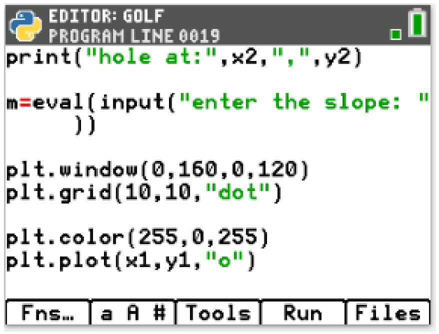
    Execute the code. Run

    Notice the order.
    1. Display ball and hole location
       Request slope
    2. After the slope is entered graph the ball
    3. Immediately clear the graph

    Sequentially, the program executes the code. Don’t worry that it immediately clears the graph, we will address this on the next step.

    Run the code a few more times.
    Does the code generate a new golf ball location each time?

    Are you satisfied with the color of your golf ball?
    If not, experiment with various values for red, green and blue.

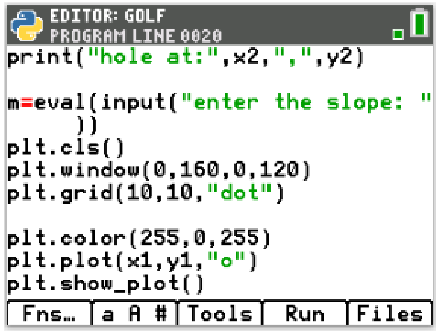


```
print("hole at:",x2,",",y2)

m=eval(input("enter the slope: "
      ))

plt.window(0,160,0,120)
plt.grid(10,10,"dot")

plt.color(255,0,255)
plt.plot(x1,y1,"o")
```

15. To clear the print and input lines, add
    plt.cls()
    after the slope input line.

    To make the plot stay on the screen until the user presses [clear], add the line plt.show_plot().

    Both commands can be found in
    Fns → Modul → TI-PlotLib



```
print("hole at:",x2,",",y2)

m=eval(input("enter the slope: "
      ))
plt.cls()
plt.window(0,160,0,120)
plt.grid(10,10,"dot")

plt.color(255,0,255)
plt.plot(x1,y1,"o")
plt.show_plot()
```

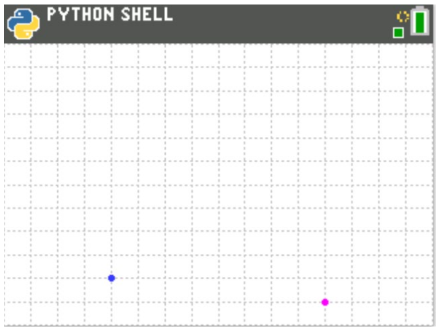
16. Execute the code: Run

    After entering the slope, does the screen clear before it graphs?
    Does the graph with the plotted ball stay until you press [clear]?

17. Before the plt.show_plot() add two lines.

    1. Change the plot color to any color you like.
    2. Plot the coordinates x2,y2 to represent the golf hole.

    Execute your code. Verify the golf ball and the golf hole appear at different locations and have different colors.



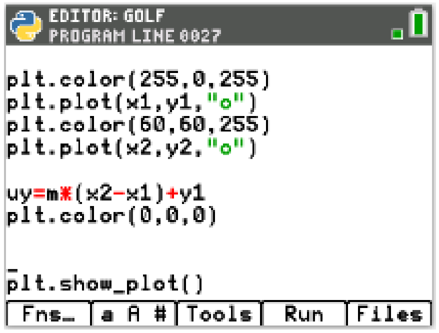
18. The last step is to graph an equation using the user's slope, the golf ball coordinate x1,y1 and the hole's x2 value.

    Recall, y = m(x – x1) + y1.

    Use m, x2, x1, and y1 to calculate the user's y value called uy

    uy = m*(x2 – x1) + y1

    Change the plot color. The example to the right changes the color to black.



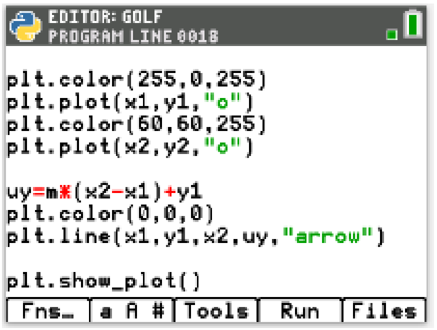
```
plt.color(255,0,255)
plt.plot(x1,y1,"o")
plt.color(60,60,255)
plt.plot(x2,y2,"o")

uy=m*(x2-x1)+y1
plt.color(0,0,0)


plt.show_plot()
```

19. Plot the line.

    The command is plt.line(x1, y1, x2, y2, “mode”)
    Fns → Modul → **ti-plotlib** → **draw** → **line**

    plt.line(x1, y1, x2, uy, “arrow”)

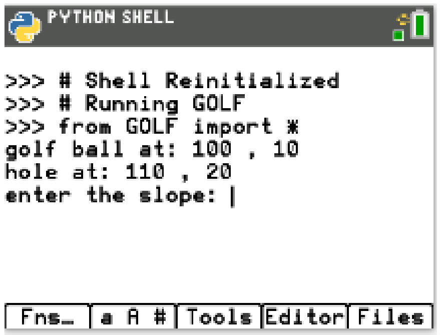
    **Make sure you plotted uy NOT y2.
    You want the line to reflect the user's slope.



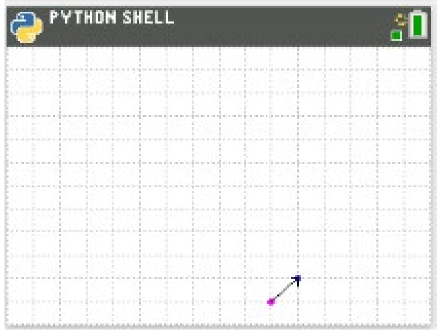
```
plt.color(255,0,255)
plt.plot(x1,y1,"o")
plt.color(60,60,255)
plt.plot(x2,y2,"o")

uy=m*(x2-x1)+y1
plt.color(0,0,0)
plt.line(x1,y1,x2,uy,"arrow")

plt.show_plot()
```

20. *Try your game out.*

If you enter the correct slope, do you make the shot?
The example at the top is an example of a correct shot.

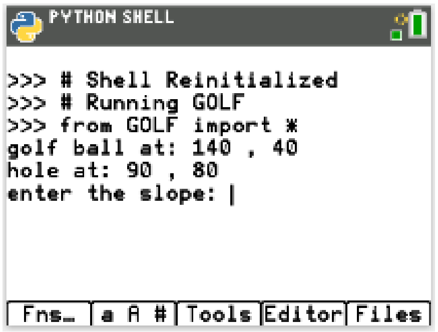


```
>>> # Shell Reinitialized
>>> # Running GOLF
>>> from GOLF import *
golf ball at: 100 , 10
hole at: 110 , 20
enter the slope: |
```

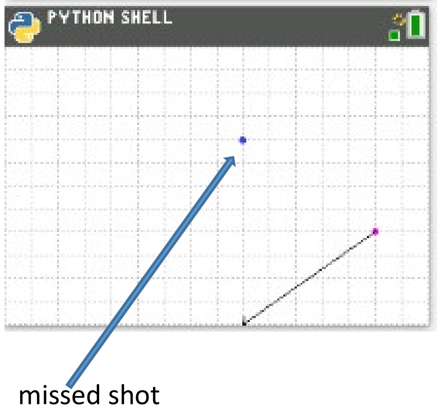


If you enter the incorrect slope, do you miss the shot?

The example at the bottom is an example of an incorrect shot

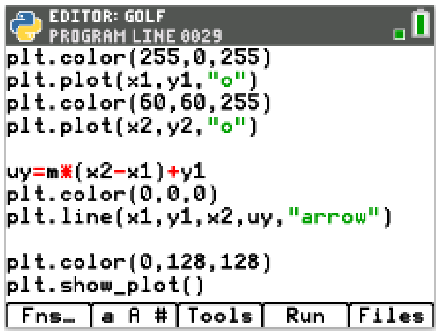


```
>>> # Shell Reinitialized
>>> # Running GOLF
>>> from GOLF import *
golf ball at: 140 , 40
hole at: 90 , 80
enter the slope: |
```



21. Shouldn't the program tell the user if the shot was a "hole in one" or a "miss"?

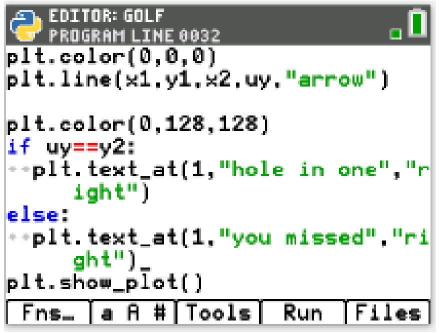
    First, change the pen color above the plt.show_pot.



```
plt.color(255,0,255)
plt.plot(x1,y1,"o")
plt.color(60,60,255)
plt.plot(x2,y2,"o")

uy=m*(x2-x1)+y1
plt.color(0,0,0)
plt.line(x1,y1,x2,uy,"arrow")

plt.color(0,128,128)
plt.show_plot()
```

22. Now check to see if the calculated value uy equals y2.

```
if uy == y2:
  plt.text_at(1, "hole in one", "right")
else:
  plt.text_at(1,"you missed","right")
```



```
plt.color(0,0,0)
plt.line(x1,y1,x2,uy,"arrow")

plt.color(0,128,128)
if uy==y2:
  plt.text_at(1,"hole in one","right")
else:
  plt.text_at(1,"you missed","right")
plt.show_plot()
```

*plt.text_at can be found in the Menu using:

Fns → Module → ti-plotlib → draw → text_at

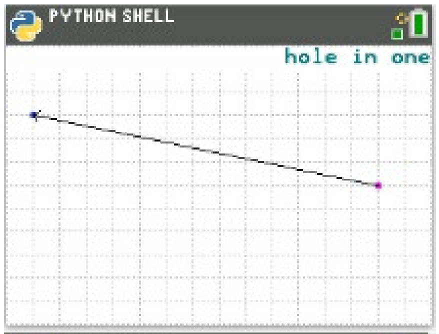
*if..else.. can be found in the Menu using:

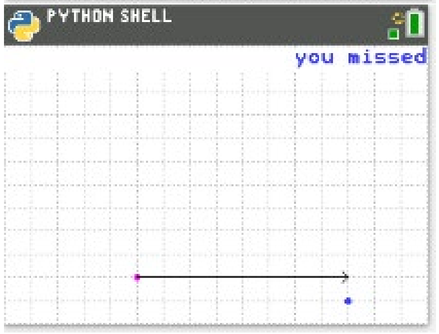
Fns → Ctl → if..else..

23. Execute your program many times.

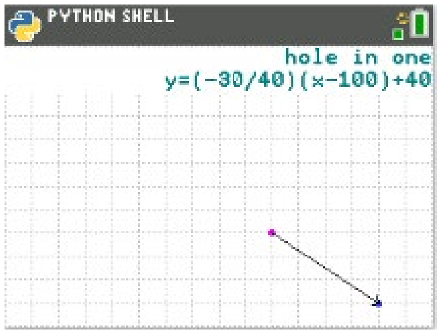
    If you answer a question correctly, does it say "hole in one"?
    If incorrect, does it say "you missed"?

    Can you modify the code to display "you missed"
    and "hole in one" in different colors?





24. **Extension:** Add text to display the equation of the line segment



```
hole in one
y=(-30/40)(x-100)+40
```

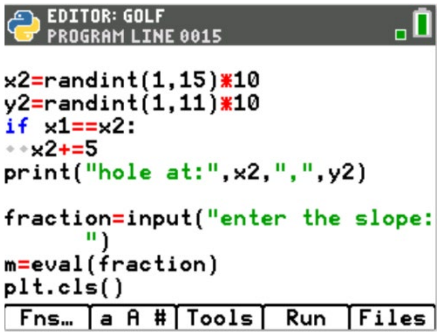
25. You need to store the fraction as text before the program evaluates the numeric approximation.

    Modify, the line:

    m = eval(input(“enter the slope:”))

    to

    fraction = input(“enter the slope:”)
    m = eval(fraction)



```
x2=randint(1,15)*10
y2=randint(1,11)*10
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)

fraction=input("enter the slope:
   ")
m=eval(fraction)
plt.cls()
```

26. To print the equation in the form $y = m(x – x_1) + y_1$, you will need to combine words such as “y=” with variable values.

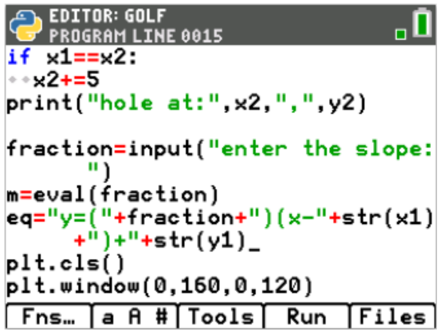
    In coding, letters, numbers, and symbols are called string values. When you put quotes around string values the python editor changes the font color to green.

    To concatenate, or add together, strings and variables use the plus symbol. In algebra, you combine like terms. In Python, you cannot combine a string with a numeric variable unless you *type cast* the numeric value to a string using the function str().

    For example, **“y=(” + fraction** is a valid concatenation because the variable fraction is already a string storing the text entered from the user.

    However, to combine **“)x-” + x1** is not valid because x1 is a float variable containing a numeric value. To combine these values using *type casting* you would enter **“)x-” + str(x1)**.

    To store the entire equation as one string variable named eq type
    eq = “y=(” + fraction + “)(x-” + str(x1) + “)+” + str(y1)



```
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)

fraction=input("enter the slope:
   ")
m=eval(fraction)
eq="y=("+fraction+")(x-"+str(x1)
   +")+"+str(y1)_
plt.cls()
plt.window(0,160,0,120)
```

Tech Tip: Remember the [a A #] menu under the [window] button is helpful for typing text.

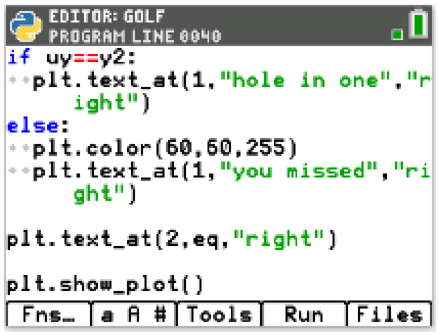
27. To display the equation you will use the text_at() command. The text_at command has three parameters, or values it needs to function.

    plt.text_at(row, text, alignment)

    The row values range from 1 at the top to 13 at the bottom of the screen. The text is a string value so it is typed using quotes unless using a variable. The alignment can be left, center or right.

    If you type plt.text_at(2, eq, “right”) your equation will appear in the top right corner of the screen on the second line.

    Decide where you want to display the equation, then add the last line of the code.

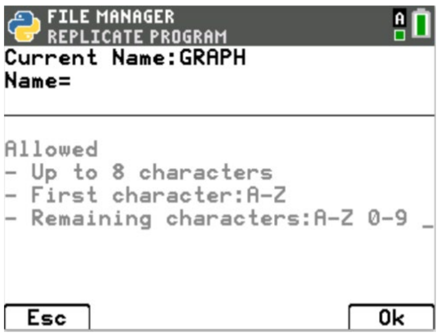


```
if uy==y2:
  plt.text_at(1,"hole in one","right")
else:
  plt.color(60,60,255)
  plt.text_at(1,"you missed","right")

plt.text_at(2,eq,"right")

plt.show_plot()
```

28. **Challenge:**

    Modify your code to ask the user for the slope AND the y-intercept.

    Use the equation y = mx + b to calculate the user’s y value (uy) using m, b, x2.



```
Current Name:GRAPH
Name=

Allowed
- Up to 8 characters
- First character:A-Z
- Remaining characters:A-Z 0-9 _
```

**To save this file and make a new copy
Select the Files Menu from either the Python Shell or Editor.

Move the Arrow Next to the file “Golf”

Select Manage

Replicate Program

Notice, the File Manager says
Current Name: GRAPH
Name =

Give the new file a name such as GRAPH2, then select Ok.

```
# Random Simulation
from random import *
import ti_plotlib as plt
plt.cls()
x1=randint(1,15)*10
y1=randint(1,11)*10
print("golf ball at:",x1,",",y1)
x2=randint(1,15)*10
y2=randint(1,11)*10
if x1==x2:
  x2+=5
print("hole at:",x2,",",y2)

fraction=input("enter the slope: ")
m=eval(fraction)

#altered eq code for the challenge
eq = "y=(" + fraction + ")x + " + str(b)
#new code for the challenge
b=eval(input("enter the y-intercept: "))

plt.cls()
plt.window(0,160,0,120)
plt.grid(10,10,"dot")

plt.color(255,0,255)
plt.plot(x1,y1,"o")
plt.color(60,60,255)
plt.plot(x2,y2,"o")

#altered uy code for the challenge
uy=m*x2+b

plt.color(0,0,0)
plt.line(x1,y1,x2,uy,"arrow")
plt.color(0,128,128)
if uy==y2:
  plt.text_at(1,"hole in one","right")
else:
  plt.color(60,60,255)
  plt.text_at(1,"you missed","right")
plt.text_at(2,eq,"right")
plt.show_plot()
```