

In this lesson, you will use the `ti_plotlib` to graph a function. That is, you do not need to leave the Python world to create a graph of a function.

Objectives:

- Create a connected graph of plotted points

1. Let's take a look at the **Plotting (x,y) & Text** template when starting a **New** program. It gives not only the necessary import statement, but also a whole demo program.

Notice line 3 of the program contains two list assignments but there's no data in the lists.

```
x = []; y = []
```

In order for the program to run, you need to supply some data in these lists. Enter some numbers inside the brackets such as:

```
x = [1, 2, 3]; y = [1, 2, 3]
```

You can use any numbers, but the lists must have the same number of elements.

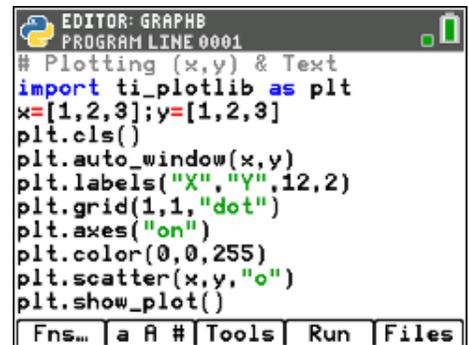
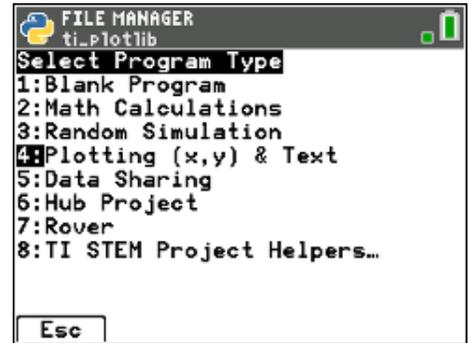
When using the <Setup> functions of `ti_plotlib` (`.cls()`, `.window()`, `.labels()`, `.grid()` and `.axes()`) the *order* in which these statements are written *matters* so they are inserted automatically in the code in the proper order for demonstration.

After entering data in the lists `x` and `y` the program will run.

The last statement,

```
plt.show_plot()
```

causes the plot to remain on the screen until the [clear] key is pressed. Without it, the Shell prompt appears. (Test this by placing a comment sign in front of the statement, and run the program again.)



10 Minutes of Code: Python

TI-84 PLUS CE PYTHON

2. Run the program to see the graph. Not quite what you are expecting? The points are plotted but they are not connected. Let's connect the dots.

3. Press **[clear]** and return to the **<Editor>**. On the next-to-last line, change the word **.scatter** to **.plot** by deleting "scatter" and typing "plot."
plt.scatter(x, y, "o") becomes **plt.plot(x, y, "o")**

*The **plt.plot(xlist, ylist, "mark")** function is on the menu right below **.scatter**, but it is simpler to just edit the line rather than replace it.*

*There is also a function to **plt.plot()** a single point but our **x** and **y** are lists, not numbers.*

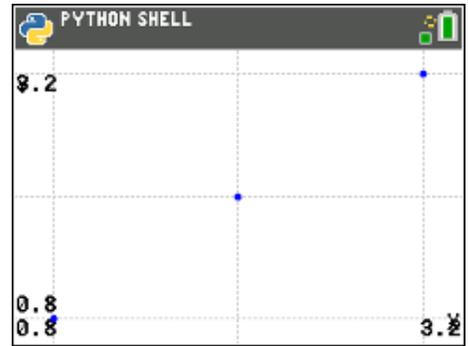
4. Run the program again and see that the three points are now connected with segments.

5. Write a program using **tiplotlib** that will graph any defined function.

Start a new program, AGRAPH (so that it appears near the top of your **Files** list) and use the **<Type> Plotting (x,y) & Text**.

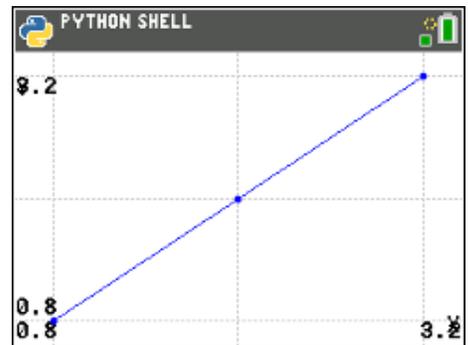
UNIT 5: SKILL BUILDER 3

STUDENT ACTIVITY



```

EDITOR: A PLOT
PROGRAM LINE 0010
# Plotting (x,y) & Text
import tiplotlib as plt
x=[1,2,3];y=[1,2,3]
plt.cls()
plt.auto_window(x,y)
plt.labels("X","Y",12,2)
plt.grid(1,1,"dot")
plt.axes("on")
plt.color(0,0,255)
plt.plot(x,y,"o")_
plt.show_plot()
    
```



```

FILE MANAGER
NEW PROGRAM
Name=AGRAPH

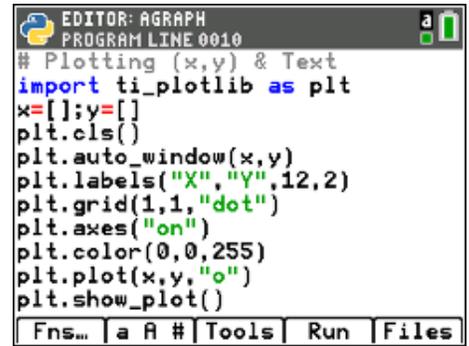
Allowed
- Up to 8 characters
- First character:A-Z
- Remaining characters:A-Z 0-9 _

Plotting (x,y) & Text
    
```

TI-84 PLUS CE PYTHON

STUDENT ACTIVITY

- Change the word “scatter” to “plot” on the next-to-last line as before.



```

EDITOR: AGRAPH
PROGRAM LINE 0010
# Plotting (x,y) & Text
import ti_plotlib as plt
x=[];y=[]
plt.cls()
plt.auto_window(x,y)
plt.labels("X","Y",12,2)
plt.grid(1,1,"dot")
plt.axes("on")
plt.color(0,0,255)
plt.plot(x,y,"o")
plt.show_plot()
Fns... | a A # | Tools | Run | Files
    
```

- Below the **import** statement define a function to graph. This will make it easy to change the function later to graph any function.

We moved the rest of the code off the screen to be less distracting.

Get both **def** and **return** from <Fns...>

Give the function a name (we use **f**) and an argument (we use **x**)

After **return**, write the function expression (we use **x**2**):

```

def f(x):
    ♦♦ return x**2
    
```

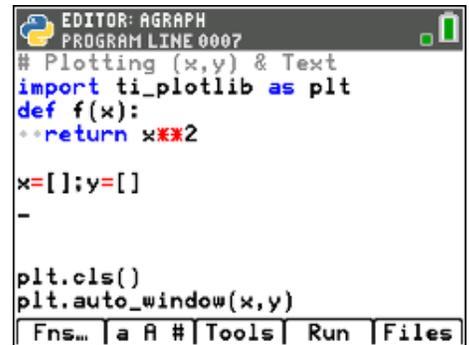
Remember to dedent to the left edge of the screen.



```

EDITOR: AGRAPH
PROGRAM LINE 0005
import ti_plotlib as plt
def f(x):
    ♦♦ return x**2
-
Fns... | a A # | Tools | Run | Files
    
```

- Below the **x=[]; y=[]** statements add some blank lines. Here’s where you will build the two lists one element at a time.



```

EDITOR: AGRAPH
PROGRAM LINE 0007
# Plotting (x,y) & Text
import ti_plotlib as plt
def f(x):
    ♦♦ return x**2

x=[];y=[]
-

plt.cls()
plt.auto_window(x,y)
Fns... | a A # | Tools | Run | Files
    
```

- Write a loop structure that starts at **xmin** and goes up to **xmax** in *small* steps. Since these steps might not be integers a **for** loop is not appropriate because the for loop only allows integer arguments. Use a **while** loop:

```

a = plt.xmin
while a <= plt.xmax:
    ♦♦
    
```

xmin and xmax are found on <Fns...> Modul ti_plotlib Properties



```

EDITOR: AGRAPH
PROGRAM LINE 0009
# Plotting (x,y) & Text
import ti_plotlib as plt
def f(x):
    ♦♦ return x**2

x=[];y=[]
a=plt.xmin
while a<=plt.xmax :
    ♦♦
-
Fns... | a A # | Tools | Run | Files
    
```

10. In the loop body build the two lists x and y using the .append() function.

- ◆◆ x.append(a)
- ◆◆ y.append(f(a))

Recall that .append() is found on <Fns...> List.



```

EDITOR: AGRAPH
PROGRAM LINE 0011
# Plotting (x,y) & Text
import ti_plotlib as plt
def f(x):
    return x**2

x=[];y=[]
a=plt.xmin
while a<=plt.xmax :
    x.append(a)
    y.append(f(a))
    
```

11. Now add a value to the variable a so that it eventually increases to xmax. Let's start with

- ◆◆ a += 1

and see how it looks. You can come back and edit this increment value later to see its effect on the graph.

Recall that a += 1 is the same as a = a + 1.

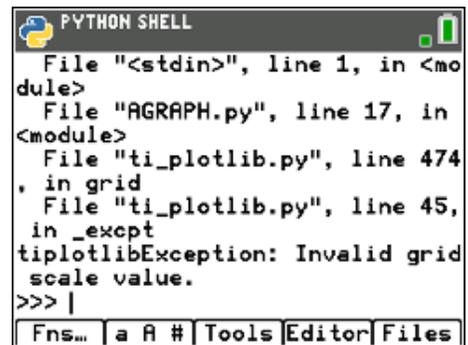


```

EDITOR: AGRAPH
PROGRAM LINE 0011
# Plotting (x,y) & Text
import ti_plotlib as plt
def f(x):
    return x**2

x=[];y=[]
a=plt.xmin
while a<=plt.xmax :
    x.append(a)
    y.append(f(a))
    a+=1
    
```

12. Run the program. You will see an error message, the most important of which is the last statement: **Invalid grid scale value**. This error occurs because **auto_window()** looks at the lists x and y and sets up a window that fits all the data on the screen, but the **.grid()** function cannot make that many grid lines. There are several ways to fix the problem. Try it yourself before looking at the next step.



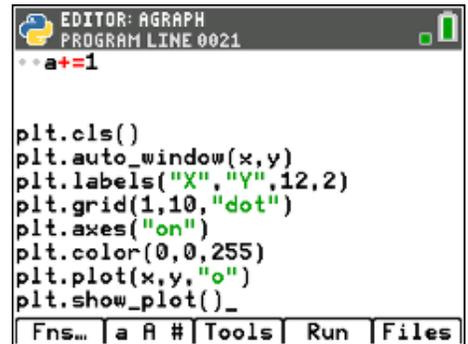
```

PYTHON SHELL
File "<stdin>", line 1, in <module>
File "AGRAPH.py", line 17, in <module>
File "ti_plotlib.py", line 474, in grid
File "ti_plotlib.py", line 45, in _except
tiplotlibException: Invalid grid scale value.
>>> |
    
```

13. Some ways to fix the error:

- Eliminate the .grid() (make it a #comment)
- Change the .grid() values to allow the grid to be drawn
- Set the window and grid by yourself rather than using **auto_window**

We chose to adjust the .grid() values. Can you spot the change?

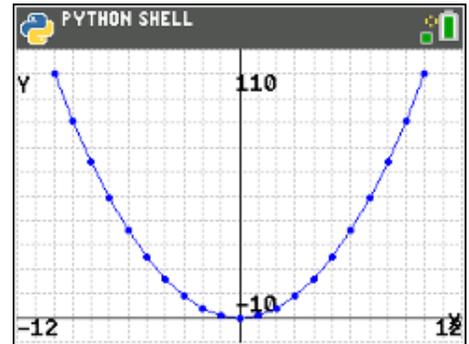


```

EDITOR: AGRAPH
PROGRAM LINE 0021
a+=1

plt.cls()
plt.auto_window(x,y)
plt.labels("X","Y",12,2)
plt.grid(1,10,"dot")
plt.axes("on")
plt.color(0,0,255)
plt.plot(x,y,"o")
plt.show_plot()_
    
```

14. Run the program. Do you see the graph shown here? Something similar? If so, congratulations! Share your success with a friend. You have graphed a function! Now for the extensions.

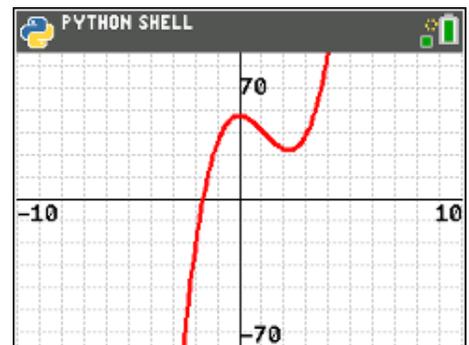


Notes:

- a) If your graph does not appear or quickly disappears be sure that you have the `plt.show_plot()` function at the end of your program. This statement pauses the program until the **[clear]** key is pressed.
- b) If your program is stuck in an “infinite loop” press **[on]** to break the program. Check the **while** loop body to be sure that the variable **a** is being changed. We use `a+=1` at the bottom of the loop. You can use other values than **1** but eventually the variable **a** must eventually exceed **xmax** so that the loop can end.

15. **Extension.** There are a lot of ways to embellish the graph of a function:

- Add Color
- Adjust dot spacing (change the increment)
- Adjust dot size (**o**, **x**, **+** and **.** are the four dot styles)
- Change line thickness (**pen**)
- Use custom window settings
- Change the function
- Graph multiple functions on the same screen



Use the `tiplotlib` module features to enrich the appearance of your graph. A sample is shown here. Once you have the core code, enhancements are easy.

Note: For trigonometric, log, and other special functions, you have to use

from math import *

The trig keys (**[sin]** **[cos]** **[tan]**) all invoke the Trig menu and **[log]**, **[ln]**, **[e^x]**, and **[10^x]** bring up the Python functions as well.