TI plotLib **Sequences**

This activity demonstrates plotting mathematical sequences using the **ti_plotLib** module in two different ways: by plotting individual points and by plotting lists.

---

0. **Part 1:** Begin a Python program using the '**Plotting (x,y) & Text**' template from the **'Type:'** dropdown list. Our program is called **PLTB**.

   Let's start by graphing both an arith**metic** (linear) and a geo**metric** (exponential) sequence in one graph.

1. As explained in the Getting Started activity and worth repeating here, this template provides the unique **import** statement:

   import ti_plotlib as plt

   along with a *nearly* complete set of programming functions.
   This type of **import** statement requires that all **ti_plotlib** functions be preceded by the alternate module name **plt.** This is called '*aliasing*' the module (give it a different, shorter name). When selecting **ti_plotlib** function from the menus, they will include this name at the beginning of the function as seen here in the first three functions provided.
   Note: the statements are placed in a particular order. First set the window, label the axes, draw the grid, draw the axes. The plot the data.

2. We will not use the **auto_window()** function. Instead, use four 'window variables' to set the **plt.window()** because you might need these values elsewhere in your code.

   x0, x1, y0, y1 = -5, 30, -5, 100

   We're placing the origin in the lower left corner of the screen because we will be plotting points in the first quadrant.

   Having the four window variables assigned on a single line makes them easier to locate & edit and saves vertical space in your program.

   The **plt.grid((1, 1, "dotted")** values can be edited as needed. Change the y-scale value from 1 to to **10**. There are limitations on the window and these grid values *may* interfere with the run of your program.

3. Plot both an arith**metic** and a geo**metric** sequence in the same graph to compare them. Use a **for** loop that makes use of the window variable:

   for n in range(x1):

   Note: change the loop variable from **i** *to* **n**.
   In the loop body, first assign two variables the value of an arith**metic**/linear term like **3n** and a geometric/exponential term like $\mathbf{2^n}$

   ```
   a = 3*n      # arithmetic
   b = 2**n     # geometric
   ```

4. Plot each term using a different plot style ("mark"):

   ```
   plt.plot(n, a, "+")
   plt.plot(n, b, "o")
   ```

   Get **plt.plot(**x, y, "mark"**)** from **[math] ti_plotLib… > Draw**

   *Note: There are two **plt.plot( )** functions on the menu. One is for plotting lists and the other is for plotting points.*
   The plot "mark" is chosen from a sub-menu and can be changed (by hand) and is limited to ".", "+", "x", or "o".

5. **<Run>** the program and you should see the two plots. One is the graph of points on a line and the other quickly disappears off the top of the screen. Which is which?

6. Try changing the window settings to see different views of these plots. Depending on your window values you may encounter the error seen here: We changed xmax (**x1**) to 100 and got this 'invalid grid scale' error because the grid lines would be too close together.
   Either:
   - change the **plt.grid**( ) *x-scale* value to a larger number (like 10) or
   - **#comment** the **plt.grid( )** function to hide the grid.

7. After fixing the grid issue, running the program now should produce a graph like the one shown here. Notice how the geometric sequence grows so quickly compared to the arith**metic** sequence. How high is each sequence at the right edge of the screen?

8. In the next section you will use most of this program to plot a different sequence, make a copy of it: Select **<Files>**, point to this file and select **<Manage>** **Replicate Program…** and provide a name for the new program. We use **PLTB2**

9. **Part 2: The Collatz Conjecture**

The final sequence in this activity is *based on* the Collatz Conjecture which is also discussed in the **TI Codes Python** materials.

The Collatz Conjecture goes like this:

0. **Start with any counting number** (we will use **input( )**).
1. **If it is even, then divide it by 2**
2. **Otherwise, multiply it by 3 and add 1**

Repeat steps 2 and 3 with the new number.

The Conjecture (guess) is that all numbers eventually will become 1 but *this has not been proven… yet*.

Starting with the number **37**, we get the sequence shown to the right in the Python Shell. Let's **plot** some Collatz sequences like this one.

10. This program will differ greatly from the previous one as we will save the plotting till the end using two lists.
    Press **[enter]** a few times after the **import** statement to make room for some new code to generate some data.

    Write the first two statements shown:

    print("Collatz Conjecture")
    n = int(input("Number? "))

    Remember that **int()** *is on the* **<Fns…> Type**
    and **input()** *is on* **<Fns… > I/O**

11. Initialize a counter variable **c** to be **0**.

    c = 0

    Create two lists:
    xs is the list of counters, starting with **0**.
    ys is the list of terms in the Collatz sequence for n, starting with **n**.

    xs = [c]
    ys = [n]

    We're going to make a big assumption here: eventually the sequence will reach 1. After all, it has never *not* happened, right?

    while n > 1:

12. Write the **if… else** structure that processes the Collatz algorithm:

```
while n>1:
  if n % 2 == 0:  #    % is 'mod'
    n //= 2       # integer division
  else:
    n = 3 * n + 1
```

Note: use integer division (//) to ensure all values are integers and there are no rounding issues.

13. Still in the loop: increment the counter and add (**append**) the counter **c** and the number **n** to their associated lists. Note the use of 'addition' here to *append* a list to another list:

```
c += 1
xs += [c]   # same as xs.append(c)
ys += [n]
```

Note that these statements are part of the **while** *loop but not part of the* **else:** *block and are indented accordingly.*

14. When the **while** loop ends the sequence is complete so we're ready to plot the data.

Much of the code for plotting is already in the program but needs some editing to suit our project.
We can use the **plt.auto_window()** function but use the arguments (**xs**, **ys**) that we created above. **#comment** the other window variables and function.

15. The next three statements control the labels, grid and axes.

Plot the lists using either

plt.plot(xs, ys, "o") will plot the lists with segments connecting the dots.

or :

plt.scatter(xs, ys, "o") to plot just the points without the connections.

16. Run the program. At the **input** prompt enter a positive integer. The plot displays the unique progression of the Collatz sequence of numbers for your entered number and the *xmax* value on the screen is a little more than the number of steps it took to reach 1. Your starting value is the point on the y-axis (it was 37 for this image).

If your graph does not remain on the screen, use **plt.show_plot()** *after plotting.*
Note: For some numbers (like 125) you might have a grid scale issue, so just comment the **.grid( )** *statement or find a scale that works.*

17. To display the initial value on the graph use:

    plt.text_at(row, "text", "align")

    found on **[menu] TI PlotLib > Draw**

    row is a number from 1 to 13 that you enter.

    "text" is your text (or a string variable) to display

    "align" can be "left", "center", or "right" selected from a pop-up menu.

    Since **n** is changed in the Collatz sequence generation, to display the starting number for your sequence add a statement right after the **input** statement that stores the value of **n** as a string:

    t = str(n)

    str() is found on **[menu] Built-Ins > Type** and use that **t** variable in the statement

    plt.text_at(3, t, "center")

    '99' at the top center of this screen is the result of the **plt.text_at()** function.

18. Each starting value has a unique sequence. Some take quite a while to reach 1. Can you find a number that does not reach 1? If so, you will be famous!

    Challenge: for each Natural number (up to an entered number) plot the *length* (number of terms) of its Collatz sequence.

    For many other interesting integer sequences see https://oeis.org/, the **Online Encyclopedia of Integer Sequences**.

Teacher Tip:

Part 1 sample code:

```
import ti_plotlib as plt
x=[];y=[]
plt.cls()
x0,x1,y0,y1=-5,200,-5,100
plt.window(x0,x1,y0,y1)
plt.labels("X","Y",12,2)
plt.grid(10,10,"dot")
plt.axes("on")

for n in range(x1):
  a=3*n
  b=2**n
  plt.plot(n,a,"+")
  plt.plot(n,b,"o")
```

Part 2 sample code:

```
import ti_plotlib as plt
print("Collatz Conjecture")
n=int(input("number? "))
t=str(n)
c=0
xs=[c]
ys=[n]
while n>1:
  if n%2==0:
    n//=2
  else:
    n=3*n+1
  c+=1
  xs+=[c]
  ys+=[n]
plt.cls()
plt.auto_window(xs,ys)
# plt.labels("X","Y",12,2)
plt.axes("on")
plt.text_at(3,t,"center")
plt.plot(xs,ys,"o")
plt.show_plot()
```

©2022 Texas Instruments Incorporated