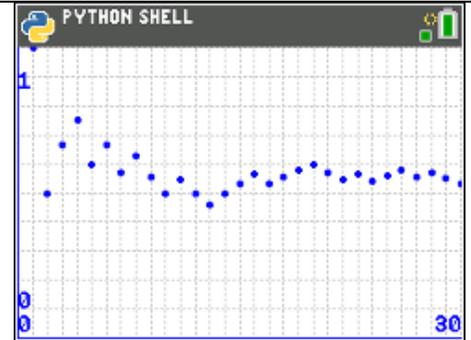




This activity introduces the **ti\_plotLib** module that is used for plotting points and graphing data sets (scatter plots).

0. Included in the TI-84 Plus CE Python system, the **ti\_plotLib** module contains statements and functions used for plotting data sets (pairs of lists), individual points, lines, and drawing text on a graph screen. This activity introduces the module through a coin-tossing (percent heads) simulation.
  
1. The **ti\_plotlib Setup** menu contains functions that *prepare* the graph screen:
  - cls()** – clear the screen
  - grid()** – set the grid scale values – style chosen from a sub-menu
  - window()**, **auto-window()** – set the viewing window
  - axes()** – style of axes (or off) mode chosen from a sub-menu
  - labels()** – label the axes
  - title()** – place a title at the top of the graph
  - show\_plot()** – pause while showing the plot. Press [clear] to exit
  
2. The **ti\_plotLib Draw** menu (many have sub-menus):
  - color()** – set the plot color
  - scatter()** – a scatterplot of the lists *with connecting segments*
  - plot()** – a scatterplot of either two lists (dots only) or plot a point
  - line()** – draw a line between two points
  - lin\_reg()** – show a least squares regression line for the two lists
  - pen()** – set the pen size and style for lines
  - text\_at()** – draw “text” at row number



```

EDITOR: PLTA2
ti_plotlib module
Setup Draw Properties
1:import ti_plotlib as plt
2:cls()          clear screen
3:grid(xsc1,yscl,"style")
4>window(xmin,xmax,ymin,ymax)
5:auto_window(xlist,ylist)
6:axes("mode")
7:labels("xlabel","ylabel",x,y)
8:title("title")
9:show_plot()   display>[clear]
  
```

```

EDITOR: PLTA
ti_plotlib module
Setup Draw Properties
1:color(r,g,b)      0-255
2:cls()          clear screen
3:show_plot()   display>[clear]
4:scatter(xlist,ylist,"mark")
5:plot(xlist,ylist,"mark")
6:plot(x,y,"mark")
7:line(x1,y1,x2,y2,"mode")
8:lin_reg(xlist,ylist,"disp")
9:pen("size","style")
0:text_at(row,"text","align")
  
```



3. **Coin Tossing:** When tossing a fair coin, *approximately* half of the tosses will be heads. If you toss the coin just a few times the likelihood that there will be an equal number of heads and tails is small. But, as the number of tosses grows, this ratio improves towards our expectation.

This activity creates an interactive simulation of coin tosses and displays a growing scatter plot of the percentage of the outcomes that are heads.



4. Begin a new Python program (we called it 'PLTA') and select the **<Type> Plotting (x,y) & Text** template from the dropdown list when naming the program. This template provides the unique import statement:

**import ti\_plotlib as plt**

along with an *incomplete* 'demo' program. The lists **x** and **y** do not contain any data. But all the **plt.** functions you will need are already in the code!

This type of **import** statement requires that all **ti\_plotlib** functions be preceded by the *alternate* name **plt**. In Python-ese this is called '*aliasing*' the module (give it a different, usually shorter name). When selecting **ti\_plotlib** functions from the menus they will include this name at the beginning of the function as seen in the statements shown here.

*Note: the Setup statements, if used, should be listed in this order since each one paints the canvas (screen) over the previous one: clear the screen, set the window, draw the grid, then draw the axes.*

```

EDITOR: PLTA
PROGRAM LINE 0011
# 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.scatter(x,y,"o")
plt.show_plot()
Fns... | a A # | Tools | Run | Files

```

5. The main program consists of a loop that ends when **[clear]** is pressed. Usually this is done with the statement: **while not escape( ):** but this program needs to also pause until a key is pressed to toss more coins when you are ready. Near the top of the program, below the two list assignments, assign the variable **k** the result of **wait\_key( ):**

**k = wait\_key( )**

and write the **while** loop that ends when **k** equals **9** by writing:

**while k != 9:**

*because the wait\_key( ) function returns the value 9 when the [clear] key is pressed.*

```

EDITOR: PLTA
PROGRAM LINE 0008
x=[];y=[]
k=wait_key()
while k!=9:
  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.scatter(x,y,"o")
  plt.show_plot()
Fns... | a A # | Tools | Run | Files

```

Indent all the **plt.** statements except the last one to become part of the loop body.

Add **from random import \*** at the top of your program since we'll be



'tossing a coin' using the `randint()` function. (*not shown*)

- 6. The plot will show the toss number along the x-axis and the *percentage of heads* tossed (as a decimal) on the y-axis. The percentage will be a number between 0 and 1.

Initialize two *counting* variables:

**t = h = 0**

before the **while** loop.

**t** counts the number of **tosses**

**h** counts the number of **heads**

- 7. In the **while** loop body, turn your attention to coin tossing and recording data. At each keypress, toss the coin 10 times using a **for** loop:

**for i in range(10):**

**t += 1** # count the toss

**x += [t]** # append the count to the list x

**h += randint(0,1)** # toss coin and add 1 if head

**y += [h / t]** # append the ratio to the list y

Note: `x += [t]` (shorthand for `x = x + [t]`) is the same as `x.append(t)`.

Find the square brackets on [2<sup>nd</sup>] [stat] or on <a A #>

```

EDITOR: PLTA
PROGRAM LINE 0009
import ti_plotlib as plt
from ti_system import *
from random import *

x=[];y=[]

t=h=0

k=wait_key()
while k!=9:
    plt.cls()

```

```

EDITOR: PLTA
PROGRAM LINE 0016
x=[];y=[]
t=h=0
k=wait_key()
while k!=9:
    for i in range(10):
        t+=1
        x+= [t]
        h+=randint(0,1)
        y+= [h/t]
    -

```



8. It's time to address the plot functions:

**#comment** the functions `.auto_window()`, `.labels()`, and `.grid()`.

press [2<sup>nd</sup>] [3] for the **#comment** symbol (#)

Add a custom window setting:

**plt.window(0, t, 0, 1)**

that changes the window to suit the number of tosses **t**

Find `plt.window( , , )` on [math] ti\\_plotlib... Setup

Note that all the **plt.** functions are still part of the while loop (indented\_ except for the last one, `plt.show_plot( )`, which simply pauses the program until the **[clear]** key is pressed.

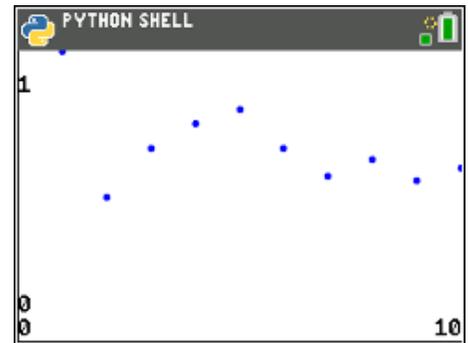
The last statement in the **while** loop is the `wait_key()` function to pause and wait for another keypress:

**k = wait\_key( )**

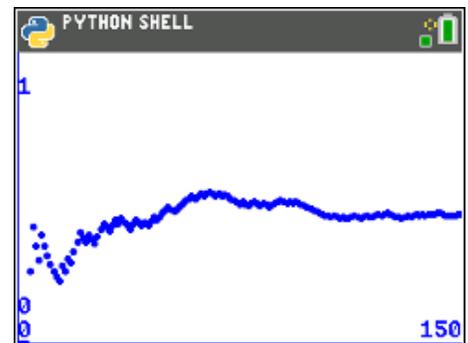
9. **<Run>** the program and press a key. You will see the first 10 dots on the graph. Each represents the percentage of the number of heads after each toss. The x-axis ranges from 0 to 10 and the y-axis ranges from 0 to 1.

```
EDITOR: PLTA
PROGRAM LINE 0019
**
plt.cls()
plt.auto_window(x,y)
plt.window(0,t,0,1)
plt.labels("X","Y",12,2)
plt.grid(1,1,"dot")
plt.axes("on")
plt.color(0,0,255)
plt.scatter(x,y,"o")
k=wait_key()
plt.show_plot()
Fns... | a A # | Tools | Run | Files
```

10. Press a key several more times. The while loop adds 10 tosses for each keypress *and* updates the plot to show more tosses. The number in the lower right corner (150 in this image) is the total number of tosses thus far.



Notice the pattern in the plot: The dots added on the right get closer to the vertical center of the screen which represents the value  $y=0.5$ . This is what you expect from tossing a coin a large number of times: the number of heads is approximately 50% (0.5 or  $\frac{1}{2}$ ) of the tosses.



To end the program, press the **[clear]** key *twice*: once to end the **while** loop and once to end the `plt.show_plot( )` function at the bottom of the program.



11. If you continue to press a key to toss more coins, eventually you will encounter the **MemoryError** indicated here. The Python App runs out of memory because the lists get too large.

#### Challenge:

You can overcome the memory limit either by:

- Plot fewer data points (say, every 10 tosses instead of every one). You will still run out of memory but at a larger (10x) total.
- Only plot the *last 100* data points. You won't run out of memory. But you will have to adjust the viewing window so that the plot still fills the screen.

```
PYTHON SHELL

Traceback (most recent call last
):
  File "<stdin>", line 1, in <mo
dule>
  File "PLTA.py", line 15, in <m
odule>
MemoryError: memory allocation f
ailed, allocating 1492 bytes
>>>
```



### Teacher Tip: Sample code

```
# Plotting (x,y) & Text
import tiplotlib as plt
from ti_system import *
from random import *

x=[];y=[]
t=h=0
k=wait_key()
while k!=9:
    for i in range(10):
        t+=1
        x+=[t]
        h+=randint(0,1)
        y+=[h/t]

plt.cls()
#plt.auto_window(x,y)
plt.window(0,t,0,1)
#plt.labels("X","Y",12,2)
#plt.grid(1,1,"dot")
plt.axes("on")
plt.color(0,0,255)
plt.scatter(x,y,"o")
k=wait_key()
plt.show_plot()
```