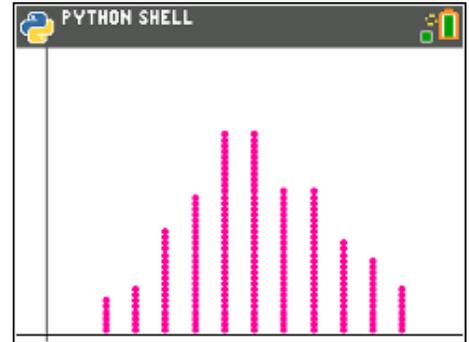
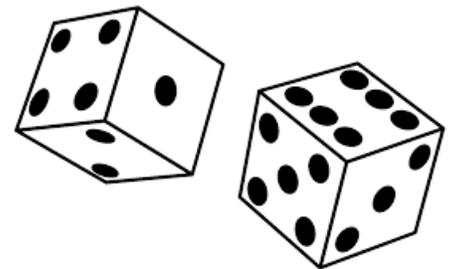


1. This activity is a compilation of the micro:bit skills you learned in the previous activities: write a program that uses a button to collect some data and display the data on the micro:bit, show a growing dot plot of the collected data and then store the lists as TI-84 Plus CE lists for further study and analysis.



2. The micro:bit can act as a 2-dice roller and display two values (sequentially). The calculator can display a growing dot plot of the **sums** recorded *at the same time*. Write a program that conducts the dice tossing experiment and collects the data. At the end of the program store the data into CE lists for further study and analysis on the calculator. Let's get started...



3. Start a new program called DICEPAIR.

You will need a whole bunch of Python tools for this program so there are quite a few imports that will take place.

- **ti\_system** for `escape( )`
- **random** for `randint( , )`
- **ti\_plotlib** for the Python graphics
- **microbit** for the micro:bit
- **mb\_disp** for the micro:bit display
- **mb\_butns** for the micro:bit buttons



```
EDITOR: DICEPAIR
PROGRAM LINE 0008

from ti_system import *
from random import *
import ti_plotlib as plt
from microbit import *
from mb_disp import *
from mb_butns import *
```

A screenshot of a code editor window titled 'EDITOR: DICEPAIR' with 'PROGRAM LINE 0008' at the top. The window contains the following Python code:

```
from ti_system import *
from random import *
import ti_plotlib as plt
from microbit import *
from mb_disp import *
from mb_butns import *
```

The editor has a menu bar at the bottom with options: 'Fns...', 'a A #', 'Tools', 'Run', and 'Files'.

*Tip: as you develop a program from scratch you will find that you need more import statements than you originally thought. It's fine to go back and add them as needed.*



# 10 Minutes of Code: Python Modules

## TI-84 PLUS CE PYTHON

- Use button **A** to toss the dice and button **B** to 'reset' the data collection and start over. It will be convenient then to have a 'setup' subroutine that can be used in different parts of the program to set up the plot screen in the calculator.

Start your code with a **def setup( )**: function.

```

EDITOR: DICEPAIR
PROGRAM LINE 0010

from ti_system import *
from random import *
import ti_plotlib as plt
from microbit import *
from mb_disp import *
from mb_butns import *

def setup():
  +-
  -
  
```

- This function performs the setup commands for plotting as indicated by the plt prefix. Find these commands on **[math]** **ti\_plotlib**.

*All these statements are indented to form the function body. Remember that the order of these setup commands is important and they should be used in the order that they appear on the menu. The color statement is on the **Draw** menu. We chose black axes and purple dots.*

You may add more statements to this function later.

```

EDITOR: DICEPAIR
PROGRAM LINE 0015

from mb_disp import *
from mb_butns import *

def setup():
  +-
  -
  plt.cls()
  +-
  plt.window(-1,14,-1,50)
  +-
  plt.color(0,0,0)
  +-
  plt.axes("axes")
  +-
  plt.color(255,0,155)
  
```

- Begin the main program using a *#comment* as an indicator.

Create two lists:

```

ttls=[0,1,2,3,4,5,6,7,8,10,11,12]
sums=[0]*13
  
```

ttls[0] and ttls[1] are not possible sums but act as 'placeholders' so that the indexes match the element contents. When the sum of the dice is 2 we will increment element sums[2].

*Note: To produce a scatter plot we need two lists. **ttls** is the first list and **sums** is the second list.*

Now 'call' the **setup( )** function you defined above to prepare the calculator screen for plotting.

```

EDITOR: DICEPAIR
PROGRAM LINE 0020

+-
-
plt.color(0,0,0)
+-
plt.axes("axes")
+-
plt.color(255,0,155)

# MAIN:.....
ttls=[0,1,2,3,4,5,6,7,8,9,10,11,
      12]
sums=[0]*13
setup()
  
```



7. Begin with the usual...

```

while not escape():
    ♦♦ if button_a.is_pressed():

```

Why did we choose `.is_` instead of `.was_` ? Think about the difference, or just wait and see...

Remember, pressing button A

- tosses the dice
- displays them on the micro:bit
- calculates the total
- increments (adds 1 to) the proper element of `sums`
- produces a scatter plot.

Try writing the code for button A now.

8. Here is most of the code for button A. One feature is missing: displaying the two dice on the micro:bit. That is left as a project for you.

Hint: The command

```
display.show(<variable>, delay=400, wait=True)
```

displays the value of the `<variable>` on the micro:bit. You will need to display two values (each die). Good luck!

The statement `plt.scatter(...)` creates a scatterplot on the calculator screen.

9. You can test your program as written so far (even without the `display` statements). It should show a scatter plot on the calculator screen as you *hold down* button A (fast). That's why we chose `.is_`.

If we used `.was_` then we would have to click and release button A to make each toss.

But wait... there's more!

Press **[clear]** to end the program.

```

EDITOR: DICEPAIR
PROGRAM LINE 0022

# MAIN:.....
tts=[0,1,2,3,4,5,6,7,8,9,10,11,12]
sums=[0]*13
setup()

while not escape():
  ♦♦ if button_a.is_pressed():

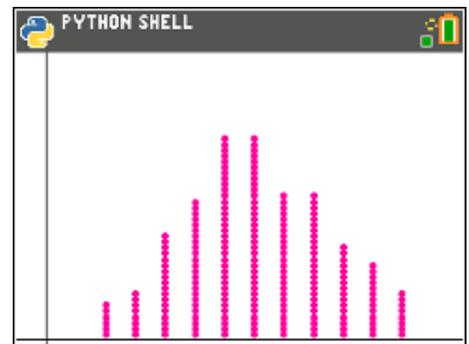
```

```

EDITOR: DICEPAIR
PROGRAM LINE 0025

while not escape():
  ♦♦ if button_a.is_pressed():
    ♦♦♦ d1=randint(1,6)
    ♦♦♦ d2=randint(1,6)
    ♦♦♦ t=d1+d2
    ♦♦♦ sums[t]+=1
    ♦♦♦ plt.scatter(tts,sums,"o")

```





# 10 Minutes of Code: Python Modules

TI-84 PLUS CE PYTHON

MICRO:BIT : PAIR-O-DICE

10. There are two more tasks:

(in addition to making the micro:bit display the values of the dice under button A)

- Button B should act as a 'reset' button: clearing the screen by calling **setup()** and set the **sums** list to all 0's again. That's only two statements in the **if button\_b...** block as shown.  
*Note the use of the **setup()** function again.*
- The final two lists should be exported to the CE for further study. You need two **store\_list( , )** statements (*incomplete as shown*), one for each list in the program.

*Pay attention to the indentation.*

*Note: we used **.was\_** instead of **.is\_** for button B. Can you feel the difference?*

11. After you have stored the two lists by pressing **[clear]** to end the loop and the program, quit the Python app (**[2<sup>nd</sup>] [quit]**) and set up a scatter plot of **TTLS**, **SUMS** - the CE lists are all CAPS - as shown in this image.

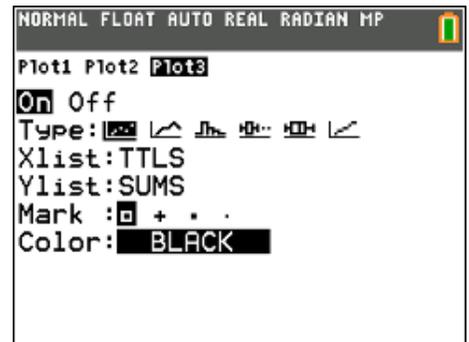
```

EDITOR: DICEPAIR
PROGRAM LINE 0033
= True)
t=d1+d2
sums[t]+=1
plt.scatter(ttls,sums,"o")

if button_b.was_pressed():
sums=[0]*13
setup()

store_list("",)
store_list("",)

```



12. What **mathematical model** best approximates this (long range) data?  
*Hint: It is NOT a regression!*

