Unit 6: micro:bit | **Skill Builder 2: The Buttons**

In this lesson you will learn about using the micro:bit **buttons** and then extend the program to toss a die and collect the values in a list. Finally, transfer the list to the CE environment to create a data plot.

There are three parts to this lesson:

Part 1: Investigate the button functions
Part 2: Use a button to generate some data
Part 3: store data in a list using buttons and transfer the list to the TI-84 Plus CE.

Objectives:

- Read and process the A and B buttons on the micro:bit
- Observe the difference between **.was** and **.is**
- Investigate collected data from the micro:bit
- Transfer data from python to TI-84 Plus CE

Teacher Tip: As in Skill Builder 1, this lesson is designed from the inside out. Code is not introduced sequentially, but developed to first focus on the button and gesture features of the micro:bit and then finish with making a connection between python lists and TI-84 lists.

Python variables are not calculator variables. A python list must be 'transferred' to the CE environment to analyze the date in the list. Similarly, a calculator list (either built in (L1..L6) or named) can be transferred to a Python program for use in the code. There are two functions in the ti_system module that accomplish these transfers.

1. The micro:bit has two buttons labeled **A** and **B** on each side of the display. The Python micro:bit buttons module has two *similar* methods for reading each button and then performing tasks based on those actions. First you will test these methods then write a program that lets you collect data and analyze it elsewhere in the TI-84 Plus CE.

Teacher Tip: On the micro:bit version 2 there is also a 'touchable' button called '**pin_logo**' above the display (the gold oval with two dots). In the buttons module this is referred to as 'Logo Touch' and uses one method, **'.is_touched()'** that is listed along with the buttons functions but only in the version 2 micro:bit buttons module. This touch button is not discussed in these lessons but is easy to incorporate into your plans based on what you see in this lesson. (There is no '**.was_touched()**' function.)

Note the difference between the behavior of **.is_** and **.was_** below…

1. **Part 1: Investigating buttons and gestures**
   Start a new python program called BUTTONS.

   Add the ti_system modules to your code using [math] ti_system:

   from ti_system import *

   From **<Fns…> Modul** select **<Add-On> microbit** to get the statement:

   from microbit import *

   Then start a loop:

   while not escape():

   ♦ ♦

   The special **while** loop can be found under **[math] ti_system…** or **<Fns…> Modul ti_system…**

   Important: *The **microbit** import must come after the **ti_system** import statement!*

   Handy Tip: Replicate this file as MBSTART and use it as a template for all other micro:bit programs. Just replicate the file using the name of a new program!

2. To use the micro:bit buttons A and B, you must first **import** the buttons module.
   Place your cursor below all existing import statements at the top of your code (on a blank line).
   Press **[math]** and select **Micro:bit…**. Choose **Buttons and Touch Logo**. This inserts the import statement **from mb_butns import ***
   It also adds a new item to the **<Fns…> Modul** (or **[math]**) menu below **Micro:bit…**.

3. To test button A, in the **while** loop body add the following **if** **:** structure:

```
if button_a.was_pressed():
    print("button A")
```

__if__ is indented to be part of the __while__ loop and __print( )__ is indented even more to be part of the __if__ block. Remember that proper indentation is very important in Python. The wrong indentation can cause syntax errors or improper execution of your code. Note the light gray diamond symbols (♦ ♦) that indicate the indentation spacing.

if is found on **<Fns…> Ctl** and automatically adds leading spaces below it for further indentation of the block.

The *condition* **button_a.was_pressed()** is found on the menu item:
[math] buttons and touch…

Notice that there is a **button A** *and a* **button B** *sub-menu at the top of the screen (not shown). If using version 2 modules, there is also a* **pin_logo** *sub menu.*

Select **.was_pressed()** under the **button A** menu.
Remember to leave the colon at the end of the **if** statement**:**
print( ) is on **<Fns…> I/O**

Type the text **"button A"** inside the **print( )** function.

Note: **.is_pressed()** *will also be discussed later.*

4. **<Run>** the program. It looks like nothing is happening. Press and release button A on the micro:bit. You will see 'Button A' appear on the calculator screen. Each time you press and release the button the text will appear as in this image.

Reminder: if you think your program is stuck in an infinite loop press and hold the **[on]** key *on your calculator to 'break' the program.*

Press **[clear]** to end the while loop (and the program) and then return to the Python **<Editor>**.

5. Add another **if** statement to check button B using the condition **button_b.is_pressed()**. You will see how **.is_** and **.was_** differ soon.

```
    if button_b.is_pressed():
        print("Button B")
```

Tip: again, pay attention to the indentations!

Teacher Tip: The two functions behave differently because there are programming circumstances where the choice between the two different behaviors will be valuable.

6. **<Run>** the program again. Try both buttons A and B.
Tap each button *and* **press-n-hold** each button.
You will see 'Button B' repeatedly displayed as long as button B is held down, but not 'Button A'. There is a difference between **.was_pressed()** (which needs a release of the button to be reset) and **.is_pressed()** which just checks to see if the button is down *at the very moment* that the statement is processed.

Note: if you tap button B quickly, the program may not display Button B *since the button is not down at the* <u>*very moment*</u> *the* **if** *statement is being processed.*

Teacher Tip:
.was_pressed() requires a full 'down-up' action to detect individual presses (clicks).

.was_pressed() detects a down-up event which can happen even when the button was clicked at <u>some other time</u> in the execution of the loop. The button must be released for another click to occur. The micro:bit 'remembers' that the button 'was pressed'.

.is_pressed() is like an 'is_down' event. Some event-driven programming languages have a similar function such as 'mouse_down'. **.is_pressed()** only produces **True** if the button is <u>down</u> at the exact moment that the statement is processed.

*In the rest of this lesson the code for button B is ignored***.**

7. **Part 2:** Let's use a button to toss a die (a cube numbered 1..6 on each face). When **button A** is pressed, assign a random integer from 1 to 6 to a variable. You can use 'button A was pressed' or 'button a is pressed'. Try both to see the difference.
Display the value of the die on the micro:bit only. Try it yourself before looking at the next step. We will use the current program and add code to simulate the die toss.

Can you determine what number is on the bottom of the pictured die?

Teacher Tip:
Answer: the bottom value is 3. The opposite faces of a die always add up to 7.

8. Add the two statements highlighted as shown in the image:
from random import * and **randint(,)** are both found on **[math] random…**

The variable **die** and the arguments ♦♦♦♦**die = randint(1, 6)** are typed in manually. Notice that the statement is indented to be part of the **if button_a…** block.

Again, be careful about the indentation.

9. Now note these two new highlighted statements in the image again. After the value of the **die** has been established, we would like to display it on the micro:bit.

First, in order to use the micro:bit display features, you must **import** the display module (the top statement). Get it from **[math] Micro:bit…**. Then add the statement:

♦♦♦♦**display.show(die)**

below the **die=** statement to show the value of the die on the micro:bit.

Micro:bit display commands are found under **[math] display…**

<Run> the program again. When you press button A you see 'Button A' on the handheld screen and the die value on the micro:bit display changes… but not *every* time! Sometimes the random number selected is the same as the last number and… that's OK. The presses are 'independent events'.

10. **Part 3: Collecting data:** Tossing all those dice with just a button press is nice, but for further study it would be helpful to store all those values so that you can interpret the data: which number occurs most often? What is the average number? and so on.

In the next step, you will add statements to the program to:
- Create an empty list
- Add (.**append**) the die value to the list
- **Store** the **list** from python to the TI-84 Plus CE system for analysis

Each of these three tasks translate into statements that are placed in special places in the program. Try it yourself before proceeding to the next step.

11. Begin with an empty list. The empty list assignment belongs at the start of the program, *before* the **while** loop:

tosses = [ ]

The variable name, tosses, is typed in. The square brackets are found on the keypad, on **<a A #>,** on **[list]** (**[2nd] [stat]**) and on **<Fns…> List**.

After the die value is determined, it is added to the list with the statement:

tosses.append(die)

.append() is found on **<Fns…> List**

At the end of the program, *after* the **while** loop ends, the final list is stored to a TI-84 list:

store_list("TOSS", tosses)

the **store_list** function is on **[math] ti_system…**

Note that this statement is not indented at all so that it is not inside the **while** *loop but is only executed once at the end of the program once* **[clear]** *is pressed. This function stores the Python list* **tosses** *over to the CE list* **TOSS**. *The CE list "***TOSS***" must be UPPERCASE and less than 6 characters.*

***Again, pay attention to the indentations, especially* **store_list()** *which is not indented at all so that it is not part of the* **while** *loop.*

Note: the ***display.show( )*** *statement has been modified to include the optional* ***delay=*** *and* ***wait=*** *arguments as explained in the previous lesson.*

Note: You may need to add a **sleep(100)** *statement to the* **while** *loop (just before the* **if** *statement) if button A does not respond.*

Teacher Tip:
The lesson uses a named list, TOSS in the TI-84 Plus CE. You can also use one of the six built-in lists L_1..L_6 by using just the *number* of the list in the quotes: **store_list("1", tosses)** stores the python list tosses in the TI-84 list L_1.

12. Button B is not used (yet). Can you use button B for something special here?
    When you run the program now:
    - if you used .**was_pressed**, press and release button A many times (You should see 'button A' on the calculator screen and numbers on the micro:bit display).
    - if you use **.is_pressed**, hold button A down.

    Press **[clear]** to end the **while** loop. Your python program had a list named 'tosses' and now your TI-84 CE also has a list named 'LTOSS'. These are two separate lists in two separate environments.

    Quit Python and set up a **[stat plot]** histogram of **TOSS**. You can also perform a **1-var Stats** analysis. Do you notice a pattern? Try more tosses*!

    Note: due to memory constraints, the* **store_list( ) *function is limited to lists of 100 elements maximum.*

Teacher Tip: Button B could be used as a 'reset' button, clearing the data and starting over with an empty list.

The program could also *count* the button presses, report the count on the calculator screen and limit the count to 100 tosses.

Changing **was_pressed** to **is_pressed** will make the program run faster just by holding the button down for a long time.