Unit 6: micro:bit with Python | **Skill Builder 2: Buttons and Gestures**

In this lesson you will learn about using the micro:bit **buttons and gestures** and then write a program to toss a die and collect the values in a list to be transferred to a data plot.

There are two parts to this lesson:

Part 1: Investigating buttons and gestures

Part 2: Using a button or gesture to generate some data

Objectives:

- Read and process the A and B buttons on the micro:bit
- Observe the difference between **.was** and **.is**
- Transfer data from python to TI-Nspire
- Investigate collected data from the micro:bit
- Use gestures to control the display

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-Nspire lists.

1. The micro:bit has two buttons, labeled A and B, on each side of the display. The python micro:bit module has two *similar* methods for reading the buttons and then performing tasks based on those buttons. First you will test the methods and then write a program that lets you collect data and analyze it elsewhere in the TI-Nspire CX II.

   There is also a 3-axis accelerometer/compass chip on the back of the micro:bit and methods for interpreting micro:bit movement and orientation.

Teacher Tip: On the micro:bit version 2 there is also a 'touch' button above the display (the gold oval). In the module this is referred to as 'Logo Touch' and uses the method **'is_touched()'**. This touch button is not discussed in these lessons but is easy to incorporate into your plan based on what you see in this lesson. Note the difference between the behavior of .is_ and .was_ below…

2. **Part 1: Investigating buttons and gestures**

   Start a new python program in a new document.

   Press the **[home]** key and select **New,** then **Add Python >New…**
   We named the program **buttons_gestures.**

   From **[menu] > More Modules > BBC micro:bit** select the **import** statement at the top of the list:

   from microbit import *

3. Add the **while** loop:

   while get_key()! = 'esc':

   from

   [menu] > More Modules > BBC micro:bit > Commands

   Almost all your micro:bit programs will be designed this way

4. To test button A, add the **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 **[menu] > Built-ins > Control**.

   The *condition* **button_a.was_pressed()** is found on
   [menu] > More Modules> BBC micro:bit > Buttons and Logo Touch

   print( ) is on **[menu] > Built ins > I/O**

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

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

5. You are ready to test the program. Press **[ctrl] [R]** to 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 the button the text will appear as in this image.

   Press **[esc]** and return to the Python Editor.

6. Add another **if** statement to check button B using the condition **button_b.is pressed()**. Note that 'IS' is different than 'WAS'. You will see how they differ soon.

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

Tip: again, pay attention to the indentations!

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

7. Run the program again (press **[ctrl] [R]**). 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***.**

8. **Introducing gestures**. The micro:bit has an electronic *accelerometer* that can measure acceleration forces in three different directions (a 3-axis or 3D accelerometer). In addition to providing numerical values for acceleration in each direction (see image), the micro:bit also provides simple, easy-to understand 'gestures' such as 'face up' and 'face down' that are based on those directional values.

This lesson explores these gestures and demonstrates how to use them in programming your TI-Nspire CX II with micro:bit.

9. Get the gesture value from micro:bit and store it in the variable **g:**

   ♦♦**g = accelerometer.current_gesture()**

   Tip: again, pay attention to the indentations!

   Type ♦♦**g =** and then

   Find **accelerometer.current_gesture()** on

   [menu] > More Modules > BBC micro:bit > Sensors > Gestures >

   Then print the value of **g**:

   ♦♦**print(g)**

   Reminder: **print( )** is on **[menu] > Built ins > I/O**

   Note that these two statements are indented one stop to be part of the **while** *loop but not part of the* **if** *statement above it. Remember that the indentation of each line determines the meaning so BE CAREFUL!*

10. Run the program and watch the calculator display for the various values as you move the micro:bit around in the air. Turn micro:bit over, stand it on each edge, shake, rattle, and roll it. Part of a sample run is shown in this image.

    Some of the available gestures are: **face up, face down, up, down, left, right, shake.** These are returned as string values. Can you see all these gestures on your screen?

11. #comment the last two gesture statements *(use **[ctrl] [T]** to #comment a line)* as shown in the image and add this other gesture function from the same menu:

    ♦♦**if accelerometer.was_gesture("face down"):**
    ♦♦♦♦**print("face down")**

    The gesture methods and the gesture "strings" are all on:

    [menu] > More Modules > BBC micro:bit > Sensors >-Gestures

    Yes, you can simply type the gesture strings but in the **.was_gesture()** *method it must match the menu item exactly.*

    When you run this program, you will notice the change in the output:

    .current_gesture() constantly prints the gesture.
    .was_gesture() only prints when the gesture changes.

12. Buttons and gestures are two ways of getting input from the micro:bit and producing results on either the TI-Nspire CX II screen or the micro:bit display… or both.
The next part of this lesson builds an activity that produces some data using the micro:bit for further investigation on the TI-Nspire CX II.

13. **Part 2:** Let's 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, button B, or a gesture of your choice.
Display the value 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.

14. Add the two statements highlighted as shown in the image:
from random import * and **randint()** are both found on **[menu] > Random**

The rest of the statement ♦♦♦♦**die = randint(1, 6)** is typed in manually and is carefully indented because it is a part of the **if button_a…** block.

Again, be careful about the indentation.

15. After the value of the die has been established, add the statement:
display.show(die)
to show the value of the die on the micro:bit display.

Run the program again. When you press button A you see 'Button A' on the handheld screen and the number 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'.

16. **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…

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-Nspire CX II system for analysis

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

17. The empty list assignment belongs at the start of the program:

tosses = [ ]

The brackets are on the keypad and on **[menu] > Built-ins > Lists**

After the die is determined. it is added to the list with:

tosses.append(die)

.append() is found on **[menu] > Built-ins > Lists**

At the end of the program the final list is stored to a TI-Nspire list:

store_list("tosses", tosses)

the **store_list** function is on **[menu] >BBC micro:bit >Commands**

Note: the two commented lines were removed so that all the code fits on the screen. You can leave those comments in your program.

18. When you run the program now, press button A many times, then press **[esc]** to end the program. Your python program has a list named 'tosses' and now your TI-Nspire also has a list named 'tosses'. These are two separate lists.

Press **[ctrl]-[doc]** or **[ctrl]-[I]** to insert a page and add a **Data & Statistics** app. A bunch of dots will be scattered around the graph. Click the message '*Click to add variable*' at the bottom of the new app and select the list **tosses**. Do you see the screen to the right? Each dot represents one of your die tosses. What information can you gather from this graph? Can you change the graph to a Histogram? Hint: check the app's **[menu]**.

You can modify this program in several ways; for example: show (**print**) how many times you toss the die while you are doing it.

*Note: button B and your gesture have no effect on the die. Try modifying your code to toss a die **only** when you 'shake' the micro:bit.*

Remember to save your document!

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

The program so far with #comments:

```
1.2 1.3 1.4   *SB2   RAD
*test_buttons.py   13/13
from microbit import *
from random import *
tosses = [ ]  # start with empty list
while get_key()!='esc':
    if button_a.was_pressed(): # down and (up to clear)
        print("Button A")
        die = randint(1,6)  # toss a die
        display.show(die)  # on micro:bit
        tosses.append(die)  # add to list
    if button_b.is_pressed():  # down, down, down
        print("Button B")
store_list('tosses', tosses) # store in TI-Nspire
```

19. **Extension:**
    - *Indent* the **store_list()** method so that the list is stored to the TI-Nspire at *every* **button_a** press.
    - **print("done.")** at the end of the program so that you know that the program has finished.
    - #comment the **if buttonB…** code and the **if accelerometer…** code.

```
1.1 1.2   *SB2 - bu...res   RAD
*buttons_gestures.py   15/16
while get_key() != "esc":
    if button_a.was_pressed():
        print("Button A")
        die = randint(1,6)
        display.show(die)
        tosses.append(die)
# if button_b.is_pressed():
#   print("Button B")
# if accelerometer.was_gesture("face down"):
#   print('face down')
    store_list("tosses",tosses)
print('done.')
```

20. Move to the Shell app and press **[ctrl] [4]** to combine the Shell with the Data & Statistics app onto the same page as in the image to the right.

    Run the program by pressing **[ctrl] [R]** on the Shell. Press button A on the micro:bit to begin tossing the die and watch the dot plot grow!

    The plot will display 'No numeric data' *at first because the* **tosses** *list has been emptied by the program but fear not: pressing button A will begin plotting the data.*

21. Instead of printing just 'Button A' at each button press you could **print** entire list of tosses. Where in your code will you **print(tosses)**?