One Day Python Programming Workshop (part of the SIS Computing Festival) on 10th May 2019

(version: 9/5/2019 pm)

Conducted by Mok Heng Ngee (hnmok@smu.edu.sg)

TAs: Nigel Poon (nigel.poon.2018@sis.smu.edu.sg) & Christine Tan (jttan.2018@sis.smu.edu.sg)

Venue: SOE/SOSS SAS Lab (level 5, Room 5013)

Time: 9:30am – 5pm

Audience: Potential students who have been offered a place in IS/CS/SMT for Aug 2019 (not necessarily matriculated)

Overview:

Unit	Name	Objective(s)	Duration/mins
1	Getting ready	• Install Mu Editor • Get familiar with the MB • Write "hello world!" and flash it to the MB	30
2	Basic display	• Use **display.set_pixel** to change the display, and **sleep()** to insert a pause • Learn how to read error messages using Mu's Checker • Understand sequential program flow	30
3	While True loops	• Use a permanent loop to repeat a block of statements indefinitely • Understand program flow using loops • Understand indentation to demarcate a block of code • Additional stuff: generate a random integer	30
4	Conditionals (if/else)	• Use **if**, **else**, **elif** to direct program flow • Understand "booleans" as True or False • Use **and** and **or** keywords in Boolean expressions • Use **button_a.is_pressed()**, **button_b.is_pressed()** to perform some basic logic handling	30
5	Variables	• Understand that a variable stores a value. 3 types are covered: strings, integers and booleans • Basic arithmetic operators (+-*/) • Assignment: x = x + 1 • += and -= operators for integers	60

		• >, >=, <, <=, ==, != operators for Booleans • Demonstrate **while** with boolean conditions (e.g. while x < 10:)	
6	for loops	• Understand **for** loops using **range**: **for i in range(0, 10)**: • Compare **while** and **for** loops	30
7	Nested for loops	• Show examples of nested for loops. • Emphasize on indentation	30
8	Mixing loops and conditionals	• Show examples of loops with conditionals	30
9	Optional fun stuff: Music and speech	• Learn how to connect the MB to a speaker • Use **music.play** and **speech.say**	-
10	Optional fun stuff: Communicating using radio	• Use **radio.config(channel=?)** to set a channel • Use **radio.send()**, **radio.receive()** to transmit and receive a string respectively • Show an example of a simple message sender and message receiver	-

Code examples & exercises:

1: Getting ready

Notes:

- Do not touch the micro:bit (MB) with wet hands. Please handle with TLC.

```
# 1_1.py
from microbit import *

display.show("A")
display.show("Hello World!")
display.show("8")              # A string is passed in

display.scroll("Hello World!", delay=150, loop=False)
display.show(Image.HAPPY)
```

Notes:

- Python is a case-sensitive language. This will cause a problem:

```
Display.show(image.HAPPY)
```

- It is OK to omit the spaces after commas, but by convention, do keep a space after each comma, and don't keep a space before the opening round bracket.
- A # symbol denotes a comment. Text after the # will be ignored. Comments help code readers understand what is happening.

- A “string” is a series of characters demarcated by quotation marks. E.g. `"Hello World!"` is a string. `"8"` is a string, while 8 is not (8 is an integer). You may use single quotation marks as well.

Ex1a: Try scrolling a string (text) with loop set to True (capital “T” for “True”), and other delay values.

e.g. : `display.scroll("Hello World!", delay=150, loop=True)`

What happens?

Ex1b: Try displaying (show or scroll) other text, and displaying other predefined images. See https://microbit-micropython.readthedocs.io/en/v1.0.1/image.html (Attributes section) for other pre-defined images that you can use.

2: Basic display

The MB comes with a simple 5x5 set of pixels with adjustable brightness (0 for no light, to 9 for max brightness). You can switch individual pixels on and off like this:

```
display.set_pixel(<x>, <y>, <brightness>)
```

where x and y are integers that represent the coordinates of the pixel, and brightness is an integer from 0 to 9.

e.g.:

```
display.set_pixel(0, 4, 9) # turns on upper-rightmost pixel to max
```

This statement turns the same light off:

```
display.set_pixel(0, 4, 0)
```

This statement clears the whole display (turns off every pixel):

```
display.clear()
```

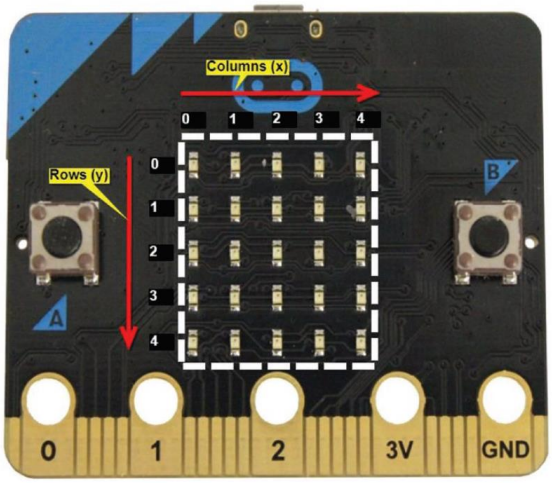


Figure 2-1. *Built-in LED display consists of columns and rows*

Diagram from p.40 of Seneviratne (Beginning BBC microbit)

This instruction pauses the program for a specified number of milliseconds:

```
sleep(<time in milliseconds>)   # sleep(1000) pauses for 1 sec
```

e.g.

```
sleep(500)   # pauses for half a second
```

```
# 2_1.py
from microbit import *

display.set_pixel(0, 0, 9)
sleep(1000)
display.set_pixel(1, 1, 9)
sleep(1000)
display.set_pixel(2, 2, 9)
sleep(1000)
display.set_pixel(3, 3, 9)
sleep(1000)
display.set_pixel(4, 4, 9)
sleep(1000)
```

Note:

- A program consists of multiple statements. Each statement is an instruction. Statements run in sequence, starting from the first one. The program ends when it reaches the last statement.

Ex2a: Write a simple program called **ex2a.py** that displays the first row of pixels one at a time from left to right. Each pixel lights up after a short interval. Once the whole row is lit up, your program then turns the pixels off one at a time until the whole display is cleared. If you have time, you may want to experiment with different brightness of the pixels as well.

Ex2b: Try inserting errors in your code (syntax errors, capitalization errors etc) and examine the error messages generated on the MB when you flash your program to the MB, and when you use Mu's Checker. Mu's Checker displays "reminders" (e.g. such as missing whitespaces), not only errors.

e.g. you can try a statement with an invalid value for x, y or brightness and observe the error. Does Mu's checker flag this as a problem?

```
display.set_pixel(0, 10, 10)
```

3: While True loops

Only "while True" is covered in this unit. "while <boolean condition>" will be covered in a later unit.

```
# 3_1.py
from microbit import *

while True:
    display.show(Image.HAPPY)
    sleep(500)
    display.show(Image.HEART)
    sleep(500)
```

```
# 3_2.py blinking light
from microbit import *

while True:
    display.set_pixel(0, 0, 9)
    sleep(1000)
    display.set_pixel(0, 0, 0)
    sleep(1000)
```

Notes:

- The colon in the while statement is important
- Using a loop, it is possible to "jump back" to a statement at the top. We call a "white True" loop a "permanent loop", and the program never ends since the loop continues indefinitely.
- Indentation is very important in Python: it is used to demarcate a "block" of code that is in a loop. What happens if code is not properly indented? We use 4 spaces (or a single tab) for indentation.

Ex3a: Write a program called **ex3a.py** that displays a happy face, waits for a short while, then displays sad face, waits for a short while, and then displays a heart. The whole cycle repeats again indefinitely.

Ex3b: Write a program called **ex3b.py** that repeats what **ex2a.py** does in a permanent loop.

Ex3c: Write a program called **ex3c.py** that displays a clock face timer. This is what it does:

- Display Image.CLOCK12
- Pause for 1sec
- Display Image.CLOCK1
- Pause for 1 sec
- Display Image.CLOCK2
- …

Something extra:

Here is how to generate a random integer in Python:

```
random.randint(0, 4)  # generates a random int from 0-4(inclusive)
```

Remember to insert this statement at the top of your program:

```
import random
```

```
# 3_3.py
from microbit import *
import random

while True:
    display.set_pixel(random.randint(0, 4), 0, 9)
    sleep(100)
    display.clear()
```

The program above lights up one of the pixels on the first row during each iteration.

Ex3d: Write a program called **ex3d.py** that lights up one pixel on the whole display randomly (on any row and column), pauses for a while, then clears the display. It then repeats the whole cycle indefinitely.

4: Conditionals

Possible usages of conditionals:

(1) If only

```
if <Boolean expression>:
      <statements>
```

(2) If/else

```
if <Boolean expression>:
      <statements>
else:
      <statements>
```

(3) If/elif (elif = else if)

```
if <Boolean expression>:
      <statements>
elif <Boolean condition>:
      <statements>
```

(4) If/elif/else

```
if <Boolean expression>:
      <statements>
elif <Boolean condition>:
      <statements>
else:
      <statements>
```

A boolean expression is something that evaluates to **True** or **False** (more about this in a later unit).

In this unit, we will learn about conditionals using the 2 push buttons on your MB.

Try using the 2 push-buttons: button A & button B:

- **`button_a.is_pressed()`** `evaluates to True or False depending on whether button A is pressed.`
- **`button_b.is_pressed()`** `does the same for button B.`

The round brackets are important. Load the next program to the MB. Press button A quickly after pressing the reset button on the MB and see if the happy face appears.

```
# 4_1.py
from microbit import *

if button_a.is_pressed():
    display.show(Image.HAPPY)
else:
    display.show(Image.SAD)
```

It can be quite difficult to get the happy face because the program runs too quickly. You can insert a sleep statement so that you get some time before the button press check happens:

```
# 4_2.py
```

```
from microbit import *

sleep(1000) # 1 sec
if button_a.is_pressed():
    display.show(Image.HAPPY)
else:
    display.show(Image.SAD)
```

Here is a slight improvement to the program. Try and understand what it does:

```
# 4_3.py
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

Notes:

- Focus on the program flow: not every statement will get a chance to execute this time.
- The indentation for if/else blocks is very important. You can have multiple statements in each block.
- Unfortunately, the following statement works (notice the missing brackets):

```
from microbit import *

while True:
    if button_a.is_pressed:   # <-- this works!!! But always evaluates to
True
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

Ex4a: Write a program called **ex4a.py** that shows a silly face (Image.SILLY) when it starts running. When button A is pressed, the silly face becomes an "A". When button B is pressed, the silly face becomes a "B". Once the button is released, the silly face appears again and the program waits for the next button press.

You can also use the **and** and **or** keywords in Boolean expressions. For example, the following expressions may evaluate to **True** or **False**:

- **`button_a.is_pressed() and button_b.is_pressed()`**
- **`button_a.is_pressed() or button_b.is_pressed()`**

The "truth table" for AND and OR is quite straightforward:

A	B	A AND B	A OR B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Example:

```
    if button_a.is_pressed() and button_b.is_pressed(): # both buttons
pressed simultaneously
        display.scroll("AB")
```

Ex4b: Modify **ex4a.py** to create **ex4b.py**. The difference here is that if both buttons are pressed simultaneously, display something else on the screen to indicate that.

5: Variables

A variable stores a value:

- A variable may store an integer, a string, a Boolean (i.e. True or False), or pretty much anything else actually.
- As the program runs, it is possible to change this value stored in that variable.
- Every variable has a name; use alphabets, digits and underscores in your variable names (e.g. my_counter, x, random_value, secret3)
- Variable names cannot be reserved words (e.g. **if**, **else**, **elif**, **while**, **True**…)

Use the assignment operator (**=**) to store a value to a variable. Here's how you can store the value **3** into a variable called **x**:

```
x = 3
```

Here is how you can store the value **100** into a variable called counter:

```
counter = 100
```

To reassign a new value to counter, just use the assignment operator again to reassign a new value to that variable:

```
counter = 101
```

Remember that at any time, a variable can only store ONE value (the latest assigned value).

```
# 5_1.py
from microbit import *
```

```
counter = 1
display.show(counter)
sleep(1000)

counter = 3
display.show(counter)
sleep(1000)

counter = 9
display.show(counter)
```

Here is a good place to introduce the basic arithmetic operators for addition, subtraction, multiplication and division: + - * /

Try this:

```
# 5_2.py
from microbit import *

counter = 1
while True:
    display.show(counter)
    counter = counter + 1
    sleep(1000)
```

What does : **counter = counter + 1** mean?

The statement **counter += 1** is equivalent to **counter = counter + 1**

Here is another example that does terminate after a while:

```
# 5_3.py
from microbit import *

counter = 1
while counter < 10:
    display.show(counter)
    counter += 1
    sleep(1000)
```

The expression **counter < 10** evaluates to True or False, depending on the current value stored in counter. This is called a Boolean expression. The following are examples of Boolean expressions:

- `counter > 3 # bigger than`
- `x < 40      # smaller than`
- `i >= 9      # bigger than or equals`
- `i <= 9      # smaller than or equals`
- `i == 10     # equals`
- `i != 3      # not equals`

Note the difference between the assignment operator (=) and the equality operator (==). They work differently!

Ex5a: Write a program called **ex5a.py** that displays the value 9. The display changes to 8, 7, 6… all the way to 0 at intervals of 1 second. It then displays "end", and terminates.

The expression **`button_a.get_presses()`** evaluates to the number of times button A has been pressed quickly (an integer). This program will show how many times you can press button A within 2 seconds after the program starts:

```
# 5_4.py
from microbit import *

sleep(2000)
no_presses = button_a.get_presses()
display.scroll(no_presses)
```

Ex5b: Write a simple game called **ex5b.py** that checks the number of times you can press button A within 1 second. If you can press fewer than 4 times within 1 second, the program shows a SAD face. If you can press 4-5 times, it shows a SILLY face. If you can press 6 or more times, it shows a HAPPY face. Hint: You will have to use a variable to store the number of button presses and check the variable's value using if/elif/else to determine which face to display.

Study the following program and understand what it does:

```
# 5_5.py
from microbit import *

while True:
    if button_a.is_pressed():
        display.scroll("A")
    elif button_b.is_pressed():
        display.scroll("B")
    else:
        display.show(Image.SAD)
```

Ex5c: Write a program called **ex5c.py** that displays a counter that starts from 0. Every time button A is pressed, the counter increases by 1. Every time button B is pressed, the counter is decreased by 1. You may extend your program so that when both buttons are pressed simultaneously, the counter is reset to 0.

Remember random number generation?

Ex5d: Write a program called **ex5d.py** that displays a vertical (or horizontal line) on the display on a randomly selected row (or column), clears the display after a pause, and then repeats.

Ex5e: Modify **ex5d.py** to create **ex5e.py**. The difference is that in ex5e, the program randomly displays either a vertical or a horizontal line during each round.

6: for Loops

We have seen the **while** loop. There is another way to loop in Python: by using the **for** keyword.

Syntax:

```
    for <variable> in range(<start>, <end>):   # includes start, but
excludes end
```

The variable here is also known as the "loop counter". During the first iteration (or loop), the variable will be assigned the value of start. During the second iteration, variable will become (start+1), during the 3rd iteration, variable will be (start+2), and so on. During the last iteration, variable will be (end-1).

E.g.:

```
    for i in range(0, 10):   # 0, 1, 2… 9 (excludes 10)
```

During the first iteration (or loop), i will store 0. During the 2nd iteration, i will store 1, and so on. During the 10th, and last iteration, i will store 9.

Simple example:

```
# 6_1.py
from microbit import *

for i in range(3, 9):
    display.show(i)
    sleep(500)
```

More examples:

```
# 6_2.py
from microbit import *

for x in range(0, 5):
    display.set_pixel(x, 0, 9)
    sleep(200)
```

```
# 6_3.py
from microbit import *

for x in range(0, 5):
    display.set_pixel(x, 0, 9)
    display.set_pixel(x, 1, 9)
```

```
    display.set_pixel(x, 2, 9)
    display.set_pixel(x, 3, 9)
    display.set_pixel(x, 4, 9)
    sleep(200)
```

Ex6a: Write a program called **ex6a.py** that uses a for-loop to light up rows of pixels starting from row 0 and ending at row 4.

Ex6b: Write a program called **ex6b.py** that uses a for-loop to light up the following pixels in order (with a pause in between): (0, 0), (1, 1), (2, 2), (3, 3), (4, 4)

Ex6c: Modify **ex6a.py** to make **ex6c.py**. In ex6c, use a for-loop to light up rows of pixels starting from row 4 and ending at row 0. Think about how you can do this.

Fast students: check out for loops with a step by trying these:

```
for i in range(0, 10, 2):      # step is 2
    display.scroll(i)
```

```
for i in range(0, 10, 3):      # step is 3
    display.scroll(i)
```

```
for i in range(10, 0, -1):      # step is -1
    display.scroll(i)
```

```
for i in range(10, 0, -2):      # step is -2
    display.scroll(i)
```

Comparing for and while loops. The following programs function the same way:

```
# 6_4.py
from microbit import *

for i in range(3, 10):
    display.show(i)
    sleep(500)
```

Using while:

```
# 6_5.py
from microbit import *

i = 3
while i < 10:   # can also use while i <= 9
    display.show(i)
```

```
    sleep(500)
    i += 1
```

Important difference between loop counters in for-loops and while-loops:

- In a for-loop, if you change the value of the loop counter, the value will be "reset" to the "correct" value during the next iteration. For example:

```
for i in range(3, 10):
    display.show(i)
    i += 1   # <-- does not affect the value of i during the next
iteration
    sleep(500)
```

7: Nested for loops

A nested for-loop is a for-loop within an external for-loop.

6_3.py from the previous unit can be changed to this:

```
# 7_1.py
from microbit import *

for x in range(0, 5):
    for y in range(0, 5):
        display.set_pixel(x, y, 9)
    sleep(200)  # <-- is it significant how this statement is indented?
```

Let's study the program above carefully.

8: Mixing loops and conditionals

In this unit, we will attempt some problems that require knowledge of everything that you have learnt so far.

Study this program and guess what it does:

```
# 8_1.py
from microbit import *

counter = 1
while True:
    if button_a.is_pressed():
        while counter < 10:
            display.show(counter)
            counter += 1
            sleep(500)
    else:
```

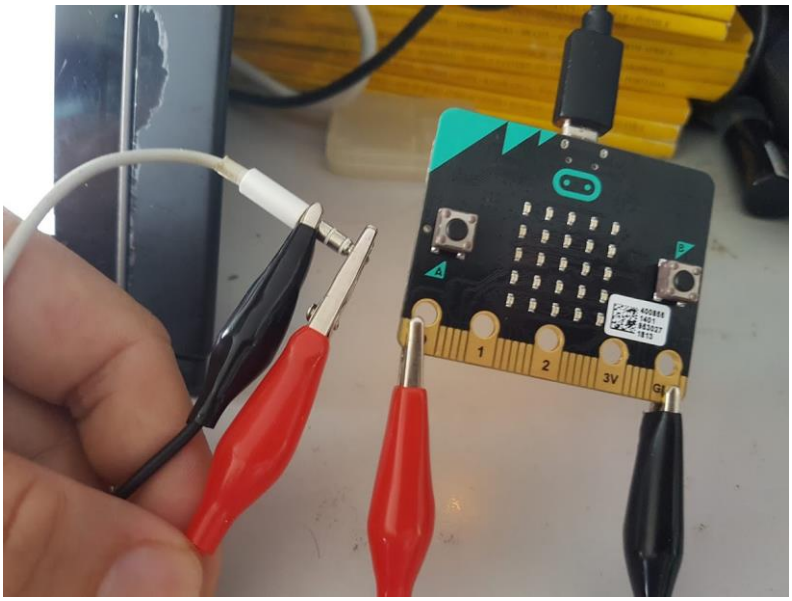
```
        display.show(Image.HAPPY)
        counter = 1  # reset
    sleep(2000)
```

Ex8a: Write a reaction game called **ex8a.py** which checks which of two players has a faster reaction time. Each player is allocated a button (player A to button A; player B to button B). When the game starts, the display shows a sad face. After a random amount of time (say, between 2-10 seconds), the sad face becomes a happy face. The first player who presses his button when the happy face appears wins. The display shows either "A" or "B" to indicate the winner.

9: Optional fun stuff: Music & speech

Collect two crocodile clips from your instructor.

Here is how you should attach wired audio speakers (or ear phones) to the MB using crocodile clips. The "0" on the MB to the tip of the audio jack, and the "GND" (ground) to the base of the audio jack.



Examples of how to produce voice/sound. Try the statements individually:

```
# 9_1.py
from microbit import *
import music
import speech

music.play(music.BIRTHDAY)

tune = ["C", "D", "E", "F", "G", "A", "B"]
music.play(tune)
```

```
speech.say("hello world!")
```

Check out https://microbit-micropython.readthedocs.io/en/v1.0.1/tutorials/music.html

---

10: Optional fun stuff: Communicating Using Radio

One MB can send a message wirelessly to another MB via proprietary radio. Radios can be configured to "listen" or "broadcast" at a specific channel (There are 84 channels for the MB: channel 0 through 83).

Partner with a friend, because you will need two MBs to try this. One MB should be the designated message sender, and the other will be the message receiver. First, agree on a channel (0 to 83, inclusive). Both the sender and receiver must be configured to the same channel.

This is the sender's code to be flashed to one MB. Try to understand what it does.

```
# 10_1_sender.py
from microbit import *
import radio

radio.config(channel = 0)  # <-- Decide on a channel and change the value
here. Default channel is 7
radio.on()
number = 1

while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
        message = str(number)
        radio.send(message)
        number += 1
        sleep(2000)
    else:
        display.show(Image.SAD)
```

Notes about the sender program:

- You can only pass in a string to **radio.send()**. This works: **radio.send("88")**. You cannot pass in an integer, so this causes an error: **radio.send(8)**. In other words, you can only send strings as messages.
- To convert an integer into a string, you can do this: str(<integer>). So the following statement converts 8 into a string "8", and stores it in the variable **message**:

  ```
  message = str(8)
  ```
- In the sender code above, the reason why **number** stores an integer (instead of a string) is because the value needs to be increased by one during each iteration of the loop. You cannot add 1 to a string. The integer is then converted into a string before it gets sent out.

This is the receiver code, to be flashed to the second MB.

```
# 10_1_receiver.py
from microbit import *
import radio

radio.config(channel = 0)  # ← change to same channel as sender
radio.on()

while True:
    incoming = radio.receive()
    if incoming != None:
        display.scroll(incoming)
        sleep(2000)
    else:
        display.show(Image.SAD)
```

Notes about the receiver program:

- **radio.receive()** evaluates to either **None** or a string (the message) if it receives a message at the current channel.

Notes about using radio:

- Turn off the radio (**radio.off()**) when not in use because it consumes power from the external battery pack.
- You can actually set the "power" of the radio transmission from 0 (weak) to 7 (strong). The default power is set to 6. The higher the power, the more power is required, and the further the signals can be transmitted. This is how to set the power and channel:
  `radio.config(power = 7, channel = 8)`

Ex10a: Write a sender that uses radio to send a secret number to a receiver. This secret number is basically the number of button A presses. Write a receiver that displays this secret number when it gets it.

Some other project ideas (the examples provided at this website uses the drag-and-drop interface to create a MB program):

- https://makecode.microbit.org/