Programming micro:bit in Python with MatDeck

MatDeck supports the use of a micro:bit with Python; similar to the Guide published at https://microbit.org/get-started/user-guide/python/. In this document, you can find short examples of code and procedure that performs quick and practical projects.

Flash Python code to micro:bit device

The Python code for a micro:bit can be written directly within a MatDeck document, as shown below. The Python block is obtained by inserting a code block, which is found in the Programming tab, under Text Code; or can be added using Ctrl+i on a keyboard. The code editor is switched from MatDeck C++ style script to Python using the Python Block button in the Programming tab, or Ctrl+k on the keyboard. All the following examples and any Python code can also be done in the Python IDE, which can be opened by selecting the arrow on the New icon in the File tab and clicking New Python.

The Python code for a micro:bit should start with ***from microbit import ****. This import is used to define the necessary functions and objects. The import line is also used as an indicator, which tells that the current code block should be deployed and flashed to a micro:bit device.

When the Python code is ready, the user can flash .HEX files onto a micro:bit directly from the document. The micro:bit has to be connected to the PC. The flashing is done by selecting the Deploy option from the Programming tab. If there are several code blocks in the document, the user must select the block which should be deployed, and after that select the Deploy button from the Programming tab. The selection must include all lines between the *#py* line, and ### line. In the following example, the lines ranging from 2 to 10 should be selected for deployment. After deployment, check the micro:bit, it should display the output.

Example 1: Images

As well as showing images such as the HEART icon, you can use lots more built-in images with Python. For example, the HAPPY, DUCK or GHOST icons. You'll find a list of all the built-in images in the micro:bit Python reference guide, https://microbit-micropython.readthedocs.io/en/latest/tutorials/images.html .

```
1  #py
2  from microbit import *
3  #Put python code below this line
4  while True:
5      display.show(Image.HAPPY)
6      sleep(1000)
7      display.show(Image.DUCK)
8      sleep(1000)
9      display.show(Image.GHOST)
10     sleep(1000)
11 ###
```

You need to be very precise when coding in text-based languages, Python is no exception. The spaces at the start of lines 5, 6, 7, 8 and 10 are important. They are called indentations, made from four space characters (or one press of the TAB key.)

You'll notice that Python programs often use a *while True:* statement. This is an infinite loop like the forever block in MakeCode or Scratch. The instructions indented after the *while True:* form part of the loop, your micro:bit will keep carrying out these instructions as long as it's on.

Any instructions after the *while True:*, that are not indented, won't be carried out until the loop has finished.

Example 2: Making your own image

You can also make your own pictures, as seen in the code below. Try different numbers from 0 to 9 to make each LED darker or brighter:

```
#py
from microbit import *
#Put python code below this line
display.show(Image('00300:'
                   '03630:'
                   '36963:'
                   '03630:'
                   '00300'))
###
```

Example 3: Buttons

In this example, we use the infinite *while True:* loop to keep checking if a button has been pressed.

```
#py
from microbit import *
while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    if button_b.is_pressed():
        display.show(Image.SAD)
```

Sensors

The micro:bit has multiple built-in sensors which you can access with Python, this allows you to interact with the world around you.

Example 4: Accelerometer - Gestures

The micro:bit has a built-in accelerometer input sensor that measures physical forces. You can use it to program your micro:bit to respond to when it has been moved in a certain way, such as when you shake it, drop it, turn it on its side, face down or face up. These movements are called gestures.

```
#py
from microbit import *
while True:
    if accelerometer.was_gesture('shake'):
        display.show(Image.SILLY)
        sleep(2000)
    if accelerometer.was_gesture('face up'):
        display.show(Image.HAPPY)
    if accelerometer.was_gesture('left'):
        display.show('<')
    if accelerometer.was_gesture('right'):
        display.show('>')
###
```

Example 5: Accelerometer - Data

It is also possible to get a more accurate reading of the forces in 3 dimensions using the micro:bit's accelerometer. The program below works as a kind of spirit level; showing a dash if it's level, or showing arrows to tell you which way it's leaning, if it's not flat on your desk. It does this by just measuring the forces in the x-axis:

```
#py
from microbit import *
while True:
    reading = accelerometer.get_x()
    if reading > 20:
        display.show(">")
    elif reading < -20:
        display.show("<")
    else: display.show("-")
###
```

Example 6: Temperature - Thermometer

The micro:bit's processor contains a temperature sensor that can be used in your programs, it gives a useful approximation of the temperature around the micro:bit. This program acts as a thermometer and shows how warm or cold the micro:bit is in °C when a user press button A:

```
#py
from microbit import *
while True:
    if button_a.was_pressed():
        display.scroll(temperature())
###
```

Example 7: Temperature - Min Max Thermometer

The task is to track the highest and lowest temperatures by leaving this program running on a micro:bit. The program uses the temperature sensor inside the micro:bit's CPU (central processing unit) to measure the temperature in °C (Celsius). This program keeps track of the lowest and highest temperatures recorded by using 3 variables: *currentTemp*, which is the current temperature reading; *max*, which is the maximum temperature recorded and *min*, which is the minimum temperature recorded.

- At the start of the program, the variables are all set to the same value and an infinite (forever) loop ensures that every two seconds it takes a reading, and the program compares the current temperature with the *max* and *min* variables.
- If the current temperature is less than (<) the value stored in the *min* variable, it changes the *min* variable to be the same as the current temperature.
- If the current temperature is greater than (>) the *max* variable's value, it changes the *max* variable to be the same as the current temperature.
- The program also flashes a dot on the LED display every time the infinite loop runs so that you know it's still working.
- To show the temperature recordings, press button A to show the minimum and button B to show the maximum temperatures recorded.
- You can leave the micro:bit running for 24 hours, record the maximum and minimum temperatures and plot them on a chart at the same time every day and then reset it daily.

```
#py
from microbit import *
currentTemp = temperature()
max = currentTemp
min = currentTemp
while True:
    display.show('.')
    currentTemp = temperature()
    if currentTemp < min:
        min = currentTemp
    if currentTemp > max:
        max = currentTemp
    if button_a.was_pressed():
        display.scroll(min)
    if button_b.was_pressed():
        display.scroll(max)
    sleep(1000)
    display.clear()
    sleep(1000)
###
```

Example 8: Light - Night-light

The LED display on the front of your micro:bit can detect light, acting as a sensor input as well as an output. Try this simple night-light project: shine a light on your micro:bit, then cover it or turn off the lights and you should see the display light up.

- This program uses selection to sense if the light shining on the micro:bit falls below a certain level – less than (<) 100. If it is dark, it lights up the micro:bit display, otherwise, it clears the screen so the LEDs are dark.
- You may need to adjust the threshold number (100) depending on the lighting conditions you are in.

```
#py
from microbit import *
while True:
    if display.read_light_level() < 100:
        display.show(Image(
        "99999:"
        "99999:"
        "99999:"
        "99999:"
        "99999"))
    else:
        display.clear()
    sleep(2000)
###
```

Example 9: Compass - North

This simple compass shows you which way is North. The micro:bit has a compass sensor called a magnetometer that measures magnetic fields. It can sense the Earth's magnetic field and therefore you can use it as a compass.

- When you first use the micro:bit compass you will have to calibrate it. A little game appears on the screen where you have to tilt the micro:bit to light up every LED, then you're ready to go.
- The program uses an infinite (forever) loop to constantly take compass readings and store them in a variable called *'bearing*. It then uses selection: an *if… else* statement to show N for North if the bearing is greater (>) than 315 degrees or less than (<) 45 on the LED display.
- This means that it will show you where North is when your micro:bit is pointing in roughly the right direction.

```
#py
from microbit import *
compass.calibrate()
while True:
    bearing = compass.heading()
    if bearing < 45 or bearing > 315:
        display.show('N')
    else:
        display.show(' ')
###
```

Sound

Example 10: Make some noise

To make sounds with a micro:bit, you have to attach headphones or a speaker. Clip the tip of the headphone plug to pin 0 on the micro:bit. Clip the longer part of the headphone plug to the GND pin on the micro:bit. Use this program to make your micro:bit play one of its built-in tunes when you press button A. The gold pins on the bottom of the micro:bit are used for inputs and outputs. Here we use pin 0 as an output. The micro:bit sends pulses of electrical signals from pin 0 when it plays the tune. The headphones must also be connected to the GND pin on the micro:bit to complete the electrical circuit.

```
#py
from microbit import *
import music
while True:
    if button_a.was_pressed():
        music.play(music.NYAN)
###
```

Example 11: Clap-o-meter

The program measures how long a round of applause - or any loud sounds - lasts with this timer. The timer uses the microphone on the new micro:bit.

- At the start of the program, the threshold for triggering a loud sound event is set. Use bigger numbers for louder sounds and smaller numbers for smaller sounds. You can use any number from 0 to 255.
- A variable called *start* is set to 0. This is used for tracking when the loud sound begins.
- When the microphone detects a loud sound, the *start* variable is then set to the micro:bit's current running time and an icon is shown on the LED display, so you know the timer has started.
- *Running time* is a measure of how long your micro:bit has been running your program in milliseconds (thousandths of a second).
- When the loud sound stops, a quiet sound event is triggered.
- If there has already been a loud event, and the timer has started, the *start* variable will have a value greater than (>) 0. In this case, a variable called *time* is set as: the new current *running time* minus the start time. This tells us how long the loud sound lasted.

- Because the time is measured in milliseconds, the program multiplies it by 1000 to convert it to seconds and then displays it on the LED display.

```
#py
from microbit import *
microphone.set_threshold(SoundEvent.LOUD, 150)
start = 0
while True:
    if microphone.was_event(SoundEvent.LOUD):
        start = running_time()
        display.show(Image.TARGET)
    if microphone.was_event(SoundEvent.QUIET):
        if start > 0:
            time = running_time() - start
            start = 0
            display.clear()
            sleep(100)
            display.scroll(time / 1000)
###
```