ChatGPT

Setting up micro:bit MicroPython development with Visual Studio Code

1. Prepare your PC and micro:bit

1. **Install Python** – use the latest **CPython 3.x** from the official website and **add it to your PATH**. The micro:bit extensions for VS Code rely on Python being available [1].
2. **Install micro:bit MicroPython firmware** – make sure your micro:bit has a recent MicroPython firmware loaded. You can flash a hex from the online Python editor or use the `uflash` tool (`pip install uflash`) to embed your script into the MicroPython hex and copy it to the board [2]. When plugged in via a USB data cable the micro:bit should appear as a storage device [3].
3. **Connect the micro:bit** using a **USB data cable** (not charge-only). On Windows it appears as a COM port; on macOS it appears as a `/dev/cu.*` device; and on Linux as `/dev/ttyUSB*` [4]. Make a note of the port for the REPL later.

2. Configure Visual Studio Code for MicroPython

Visual Studio Code (VS Code) can flash MicroPython scripts to the micro:bit and give you error feedback, but you must install the correct extensions and workspace settings.

micro:bit extension (Statped)

The **Statped micro:bit extension** integrates MicroPython development into VS Code. It provides type hints, flashes your script, and reads runtime errors via the REPL [1].

Steps to set it up:

1. **Create a folder or workspace** in VS Code [1].
2. Install Microsoft's **Python** and **Pylance** extensions (Ctrl + Shift + X, search "Python") [1].
3. **Search for and install "microbit" by Statped** in the Extensions view [1].
4. **Restart VS Code**. Open the **Command Palette** (Ctrl + Shift + P) and run `micro:bit Prepare`. This step installs stub files and environment settings into your workspace, including `.microbit-stubs`, `.vscode` settings and `.env` [5]. Restart VS Code again when prompted [1].
5. **Write your MicroPython script** for the micro:bit. The extension adds type hints and will highlight syntax errors [6].
6. **Flash the script** by pressing **Ctrl + F5** or running `micro:bit: Flash` from the Command Palette [1]. VS Code copies your `.py` file to the micro:bit's `main.py` and resets it.
7. **Read runtime errors** with `micro:bit: Read micro:bit (REPL)` (Ctrl + Alt + F5). Before running this command, set the COM port via `micro:bit: Set COM port` if needed [7].

Shortcut commands

- `micro:bit Prepare`: install workspace settings and stub files [1].

- `micro:bit Flash` (Ctrl + F5): flash current script [8].
- `micro:bit Set COM port`: set the serial port for REPL [7].
- `micro:bit Read micro:bit (REPL)` (Ctrl + Alt + F5): read errors and output from the micro:bit [7].

microbit-explorer extension

The **Microbit Explorer** extension offers a graphical file explorer for the micro:bit. It can send `.py` files, strip comments, update firmware, display the pinout, and upload/download/delete files on the board [9]. Requirements:

- MicroPython must already be flashed to the micro:bit and you must have a folder/workspace open [10].
- The Python and Pylance extensions must be installed [10].

Use the coloured micro:bit button that appears on `.py` files to send the script to `main.py`, or right-click files in the explorer to upload them with or without comments [11]. The extension adds a **Micro:bit** output panel for status and error messages [12].

3. Using the MicroPython REPL

Accessing the REPL (Read-Evaluate-Print Loop) lets you run commands interactively and view runtime output. You can access the REPL in several ways:

1. **Python Editor or WebUSB** – flash a program, click **Open Serial**, then send **Ctrl + C** to enter REPL [13]. WebUSB works with Chrome and micro:bits with firmware v0249 or above [14].
2. **Serial terminal programs** – after identifying the micro:bit's COM port (Device Manager on Windows, `ls /dev/cu.*` on macOS, or `dmesg | tail` on Linux) [15], open the port at **115200 bps**, 8 data bits, no parity, 1 stop bit using PuTTY, Tera Term, `screen`, or similar [16]. Press **Ctrl + C** to interrupt the running program and drop into the REPL.

4. Serial communication and MIDI notes

For your MIDI project you need the micro:bit to send raw MIDI bytes via serial. Key points from the documentation:

- The micro:bit's `uart` object is used for serial communication. Calling `uart.init(baudrate, bits, parity, stop, tx, rx)` initialises UART [17]. Leaving `tx` and `rx` unset uses the internal USB serial lines [18].
- **The REPL and UART share the same hardware**. Initialising UART disables the USB console; restore it afterwards by calling `uart.init(115200)` without pins [19]. This is important when debugging.
- `uart.write(buf)` transmits data; `buf` can be a string or bytes. For example, `uart.init(tx=pin0, rx=pin1); uart.write(b'Test')` writes bytes on external pins [20].
- When wiring an external device, **cross the TX and RX lines** and connect grounds [21].

To send MIDI notes to a computer via the micro:bit's USB serial, set `uart.init(baudrate=115200)` at the start of your script. Then send three-byte MIDI messages (status, note, velocity) using `uart.write(bytes([status, note, velocity]))`. The status byte is `0x90 | channel` for **Note On** and `0x80 | channel` for **Note Off**. See the micro:bit docs for more on using `uart` [17].

Example: sending a Note On/Off when pin P1 is high

```
from microbit import *

# calibrate the accelerometer when button A is pressed
origin_x = origin_y = origin_z = 0

THRESHOLD = 200
current_note = Note.C

# initialise UART at 115200 baud on internal USB (disables REPL until reset)
uart.init(baudrate=115200)

# helper to convert a micro:bit Note enum to MIDI note number
def note_to_midi(n):
    return {Note.C:60, Note.D:62, Note.E:64, Note.F:65, Note.G:67, Note.A:69,
Note.B:71}.get(n, 60)

# state for edge detection
last_gate = False
last_sent_note = note_to_midi(current_note)

while True:
    # update current_note based on accelerometer offsets (same logic as your
original code)
    dx = accelerometer.get_x() - origin_x
    dy = accelerometer.get_y() - origin_y
    if dx > THRESHOLD:
        current_note = Note.D
        display.show('R')
    elif dx < -THRESHOLD:
        current_note = Note.E
        display.show('L')
    elif dy > THRESHOLD:
        current_note = Note.F
        display.show('D')
    elif dy < -THRESHOLD:
        current_note = Note.G
        display.show('U')
    elif abs(dx) < THRESHOLD and abs(dy) < THRESHOLD:
        current_note = Note.A
        display.show('O')

    note_num = note_to_midi(current_note)

    # gate logic based on digital pin P1 (wire HIGH => play)
    gate = pin1.read_digital() == 1

    # retrigger if note changed while gate is high
    if gate and note_num != last_sent_note and last_gate:
        uart.write(bytes([0x80 | (MIDI_CH-1), last_sent_note & 0x7F, 0]))  #
```

```
Note Off
        uart.write(bytes([0x90 | (MIDI_CH-1), note_num & 0x7F,
MIDI_VELOCITY]))  # Note On
        last_sent_note = note_num

    # rising edge – send Note On
    if gate and not last_gate:
        uart.write(bytes([0x90 | (MIDI_CH-1), note_num & 0x7F,
MIDI_VELOCITY]))
        last_sent_note = note_num

    # falling edge – send Note Off
    if not gate and last_gate:
        uart.write(bytes([0x80 | (MIDI_CH-1), last_sent_note & 0x7F, 0]))

    last_gate = gate
    sleep(20)
```

This code replicates your original logic (calibrating the accelerometer, changing notes based on motion, and using pin P1 as a gate) but uses `uart.write` to send MIDI bytes instead of playing tones. Replace `MIDI_CH` and `MIDI_VELOCITY` with the desired MIDI channel and velocity.

5. REPL and debugging tips

- **Always close any terminal or REPL connection before flashing**; otherwise the COM port is busy and flashing will fail. If VS Code shows "Access is denied," close the REPL and try again.
- If you initialise UART on external pins and then the REPL stops working, call `uart.init(115200)` without the `tx` / `rx` parameters to restore the USB console [19].
- Use `print()` to send debugging messages to the REPL; this uses the USB serial even if you are not using the REPL. In the Python editor or in VS Code, use **Read micro:bit (REPL)** to view these messages.
- The `microbit.scale()` function can be handy for mapping accelerometer values (−2000 to 2000 mg) to a range like 0–127 for MIDI velocity [22].

6. Recommendations for your project

- **Test your MIDI messages with a serial monitor** before connecting to FL Studio. Send a Note On (e.g., `0x90 0x3C 0x64`) followed by a Note Off to ensure the micro:bit sends the correct bytes.
- **Use a buffer** for smoothing accelerometer data if your note mapping is jittery. A simple moving average over the last 10 readings stabilises the value.
- When ready to connect to FL Studio, use **Hairless MIDI – Serial Bridge** with the baud rate set to **115200**, and route it to a virtual MIDI port (loopMIDI on Windows) as you did for Arduino. Then enable that port in FL Studio's MIDI settings.

This report summarises how to set up MicroPython development for the micro:bit in Visual Studio Code, important notes from the MicroPython documentation (REPL access and UART usage), and provides a corrected approach to sending MIDI notes over serial from the micro:bit.

---

[1] [5] [6] [7] [8] [9] [10] [11] [12] micro:bit - Visual Studio Marketplace
https://marketplace.visualstudio.com/items

[2] [3] uFlash — uFlash 1.2.4 documentation
https://uflash.readthedocs.io/en/latest/

[4] [13] [14] [15] [16] Accessing the REPL — BBC micro:bit MicroPython 2 documentation
https://microbit-micropython.readthedocs.io/en/v2-docs/devguide/repl.html

[17] [18] [19] [21] UART — BBC micro:bit MicroPython 2 documentation
https://microbit-micropython.readthedocs.io/en/v2-docs/uart.html

[20] Using UART on the micro:bit
https://www.fredscave.com/microbit-module/07mm-uart.html

[22] Microbit Module — BBC micro:bit MicroPython 2 documentation
https://microbit-micropython.readthedocs.io/en/v2-docs/microbit.html