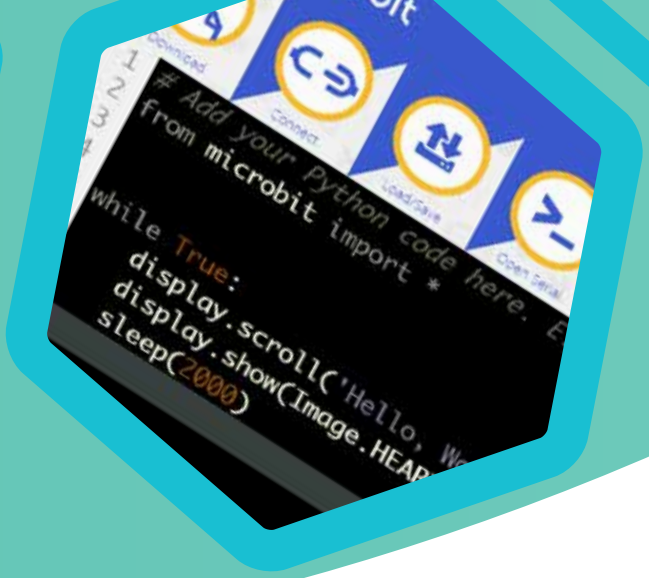
Coding Fundamentals



Micro:bit Python Programming
Communications

Overview
In this lesson, students use the Bluetooth radio capability to send and receive data between micro: bits using Python.

Objectives
- Program the microbit using its Bluetooth radio capability
- Send and receive data between Microbits
- Integrate random numbers into a Microbit program

Materials
- micro:bit and micro-USB cord
- Computer with access to the internet

Approx. Time Required
1-2 hours

Cyber Connections
- **Programming** – Students will program in Python.
- **Hardware and Software** – Students will utilize small electronics and learn how a computer is programmed while using microcontrollers.

CYBER.ORG
THE ACADEMIC INITIATIVE OF THE CYBER INNOVATION CENTER

This content is based upon work supported by the US Department of Homeland Security's Cybersecurity & Infrastructure Security Agency under the Cybersecurity Education Training and Assistance Program (CETAP).

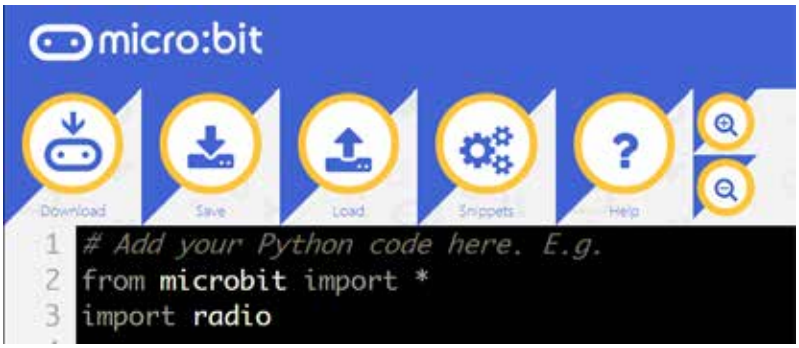
Communications

- So far with the Microbit we have seen a lot of cool capabilities of the little board. It's great to be able to do fun and interesting things as individuals, but we can do even more impressive things when we work together, communicate, and grow together. The designers of the Microbit knew this and included a truly unique addition to the microbit: a way to communicate wirelessly!

  Included with the microbit is a tiny chip has a radio that supports Bluetooth wireless communications. This little radio is much like the one found in most phones, tablets and laptops. You can actually program the Microbit through this Bluetooth link but it is even slower than the drag and drop process. Another use for this Bluetooth radio is the ability for Microbits to talk to one another and exchange data!

- First, we need to learn the new commands to control the Bluetooth radio built into the Microbit. The first and most important is the line **`import radio`**. This line, should be placed at the top of the program along with the required line `from microbit import *`

  See image below:



  Just as the "`from microbit import *`" line is used to import commands that let us interact with the microbit, the **`import radio`** line gives us access to commands to control the Bluetooth radio module. Both the `microbit` and `radio` blocks of code are known as libraries. A ***library*** is a group of ready-made routines or functions that are commonly used in conjunction with an object or device such as the radio. Simply put, someone wrote functions or commands for using the radio and stored them in a library so that others could use the functions without having to write them each and every time we want to interact with the device. The `radio` commands we will use in this lesson will not work without importing the library first because they do not exist in the standard `microbit` library.

- The second line students need to use the radio is a function that turns the radio on. It is necessary in any program in which students plan on using the radio. The function is **`radio.on()`**. In this command **`radio`** tells the microbit what we want to interact with and **`.on()`** is a function of radio, telling the microbit what to ***do*** with the object we want to interact with.

  This line should be typed after the import lines but before the main body of code.

- Finally are the two main commands that the program will use to actually communicate between Microbits:

  First is **`radio.send("message")`** where the word *message* can be replaced by whatever the microbit will actually send. The message must be encased in double quotes.

  The second is **`radio.receive()`** which receives messages from other Microbits. The receiving microbit can then use the data from the message in any way the program sees fit.

- Below is a simple program that displays exactly what is received.

```
from microbit import display, button_a, sleep
import radio

radio.on()

while True:
    if button_a.was_pressed():
        radio.send('A')

    incoming = radio.receive()
    if incoming is not None:
        display.show(str(incoming))
sleep(300)
display.clear()
```

  Note how we altered our "**`from microbit`**" line. Instead of importing ***everything***, which is what the "**`*`**" says to do, we list out only the components from the **`microbit`** code we will make use of. This is because the **`radio`** code takes up a lot of memory and the little Microbit does not have enough room for it all.

  The code works by first turning on the radio. Then checking to see if

button A was pressed. If it was pressed, it sends out "A" over the radio. Whether the button was pressed or not, the next part says to store whatever data was received in the variable `incoming`. A quick check makes sure that `incoming` actually has ***something*** in it. If there's nothing in it, the value of `incoming` is `None`. By using the quick check to see that incoming is ***not*** `None`, we then print whatever value is stored in the variable. We then wait for a moment so the user can read the message on the display. After the pause, we then clear the display and get ready for another incoming message.

- Below is an example from the Microbit official website which uses the keyword "**`flash`**" to tell the other Microbits to flash their screens. Each microbit is listening for this keyword.

  This activity is made particularly fun because it introduces some random behavior. In the import segment, you will see the inclusion of the `random` library. This allows the microbit to "roll the dice". As it is currently written, the Microbit has a 10% chance (1 in 10) of waiting 500ms then sending out "`flash`" to all other microbits in the area.

  This behavior gives a group of the microbits the appearance of a group of fireflies or lightning bugs. In nature, the firefly signals others with phosphorescent glow in its abdomen. Other fireflies signal back with their glow. As more fireflies gather, the signaling glows make for one of Nature's best light shows.

  This activity works best when you have more than 2 or 3 microbits together and with dim lighting. One truly interesting thing is to notice how the number of "echo" `flash` signals bounce around as the number of microbits increases.

  With just 2 microbits, what are the odds of an echo? What are the odds of an echo creating another echo? How does adding a third microbit alter these chances?

  In the code, how can you increase the chances of an echo? How can you ***decrease*** the chances if you have more microbits in a room? How else might the program get modified?

```
import radio
import random
from microbit import display, Image, button_a, sleep

# "flash" animation frames. (Advanced topic for now...)
flash = [Image().invert()*(i/9) for i in range(9, -1,
-1)]

radio.on()

while True:
    if button_a.was_pressed():
        radio.send('flash')

    incoming = radio.receive()
    if incoming == 'flash':
        sleep(random.randint(50, 350))
        display.show(flash, delay=100, wait=False)

        if random.randint(0, 9) == 0:
            sleep(500)
            radio.send('flash')
```