**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Session: \_\_\_\_\_**

**Programming II**

**Lab Exercise 3.8.2024**

**Introduction to Cyber Security**

Getting Started – Using Thonny to Program the Micro:Bit

1. Plug the Micro:Bit a USB port
2. Start Thonny and select MicroPython (BBC Micro:Bit) on the lower right corner.

Graphical user interface, application, Word

Description automatically generated

1. Click the Stop button if you received the above message. This should allow the MicroPython interpreter to start.

Graphical user interface, application, Word

Description automatically generated

1. Select the File|Open Menu and select This Computer.

Graphical user interface, application, PowerPoint

Description automatically generated

1. Navigate to the folder where braille.py is located and select that file to open.

A picture containing graphical user interface

Description automatically generated

1. Now run your program.

A picture containing graphical user interface

Description automatically generated

Important Notes:

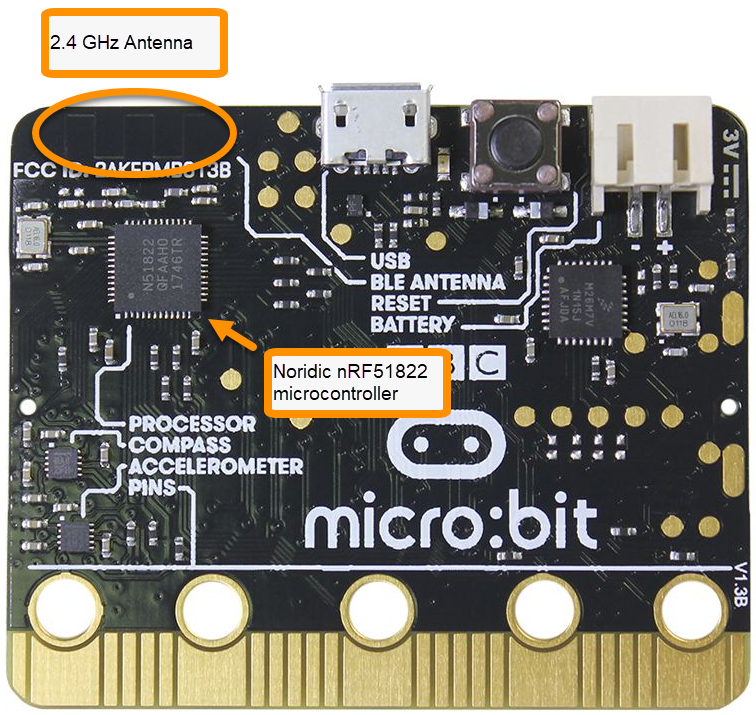
1. All Python programs can be found in today’s Starter Code folder. They are complete so you do not need to type them in.
2. All source code specifies channel 7 for Micro:Bit communication. You will have to modify this. In order to have two micro:bit’s communicate, they must be on the same channel. Allowable channel numbers are 0 – 83.
3. Part 1 and 2 require 2 Micro:Bit’s. You can do these in pairs. If you are going to work individually, you will need to make sure Thonny is configured to allow multiple instances to be run. This can be done by going to the Tools|Options menu and uncheck the Allow Only Single Thonny Instance checkbox.

Part One: Introducing the micro:bit radio

The micro:bit module’s main microcontroller is called the Nordic nRF51822 system on a chip (SOC).  It’s the processor that runs your MicroPython scripts, but it also has a built-in radio.  In MicroPython, this radio can work with either the Bluetooth or the Nordic Gazell communication protocols.

Bluetooth makes devices like wireless earbuds, mice, and keyboards pair with devices like computers, tablets and phones.  After pairing, the two devices work much like their “wired” counterparts.

Nordic’s Gazell protocol allows more than just two micro:bit modules to be part of the conversation, and it also takes fewer memory resources, making room for more statements in your scripts.



# Experiment #1 Make micro:bit Radios Send and Receive

In this activity, a script will make one micro:bit repeatedly send messages through its radio.  Another micro:bit will run a different script that receives those radio messages and prints them to its terminal.

This block diagram shows how the one-direction application will work.  The sender micro:bit prints its activity to the terminal, but that’s just so you can verify that its script is running.  The important part is that the sender micro:bit transmits radio messages to a receiver micro:bit.  That receiver micro:bit prints the messages it receives to its own terminal, where you can check and verify the radio data exchange.

Graphical user interface

Description automatically generated

**#send\_radio\_test\_message.py**

from microbit import \*

import radio

radio.on()

radio.config(channel = 7)

sleep(1000)

print("micro:bit radio sender")

while True:

message = "Testing 123..."

print("Send: ", message)

radio.send(message)

print("Done!")

sleep(2000)

**# receive\_radio\_test\_message.py**

from microbit import \*

import radio

radio.on()

radio.config(channel = 7)

sleep(1000)

print("micro:bit radio receiver")

while True:

message = radio.receive()

if message is not None:

print("Receive: ", message)

1. Write a paragraph on what you learned from this experiment.

# Experiment #2 Bidirectional Texts

With unidirectional RF text working, the next step is bidirectional texting. Each micro:bit transceiver (transmitter/receiver) will take input that you type into its terminal and send it through the radio.  Each micro:bit will also receive radio messages and print them to the terminal.

IMPORTANT: The Send: prompt will not appear until you click inside the terminal and

start typing.

Graphical user interface

Description automatically generated

**#terminal\_chat\_through\_microbits.py**

**#For Transeiver A**

from microbit import \*

import radio

radio.on()

radio.config(channel=7)

sleep(1000)

print("micro:bit transceiver A")

print("Type messages, press enter to send.")

print("Received messages will also be displayed.")

while True:

if uart.any():

tx = input("Send: ")

radio.send(tx)

message = radio.receive()

if message is not None:

print("Receive: ", message)

**#terminal\_chat\_through\_microbits.py**

**#For Transceiver B**

from microbit import \*

import radio

radio.on()

radio.config(channel=7)

sleep(1000)

print("micro:bit transceiver B")

print("Type messages, press enter to send.")

print("Received messages will also be displayed.")

while True:

if uart.any():

tx = input("Send: ")

radio.send(tx)

message = radio.receive()

if message is not None:

print("Receive: ", message)

# Write a paragraph on what you learned from this experiment

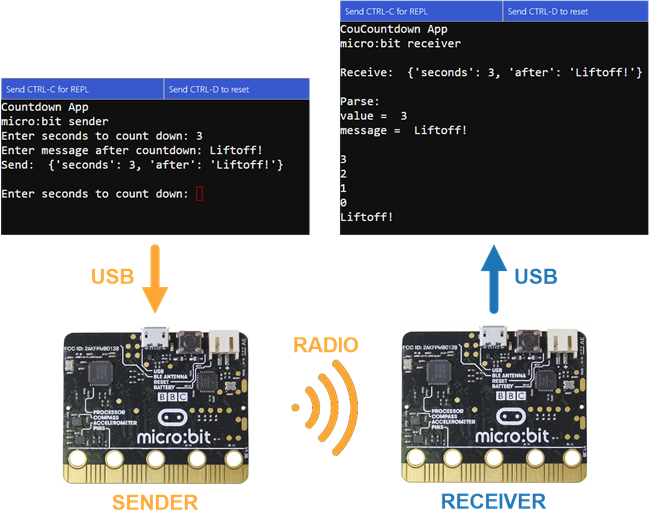
# Can you improve this terminal chat program?

# Part 2: Radio Data

# Experiment #1 Radio Data Packets

Devices typically use packets to exchange data.  Packets can contain multiple pieces of information, including headers, trailers, sequence numbers, checksums, and other features to make communication between apps and over the web robust.

In this tutorial’s example, you will enter data into a terminal connected to the sender micro:bit.  It will packetize the start number and ending message before sending it over the radio.  The receiver micro:bit will unpack the data and use it to perform the countdown.



**# countdown\_sender.py**

from microbit import \*

import radio

radio.on()

radio.config(channel=7,length=50)

sleep(1000)

print("Countdown App")

print("micro:bit sender")

while True:

text = input("Enter countdown start: ")

value = int(text)

message = input("Enter message after countdown: ")

dictionary = { }

dictionary['start'] = value

dictionary['after'] = message

packet = str(dictionary)

print("Send: ", packet)

radio.send(packet)

print()

**# countdown\_receiver.py**

from microbit import \*

import radio

radio.on()

radio.config(channel=7,length=50)

sleep(1000)

print("Countdown App")

print("micro:bit receiver\n")

while True:

packet = radio.receive()

if packet is not None:

print("Receive: ", packet)

print()

print("Parse: ")

dictionary = eval(packet)

value = dictionary['start']

message = dictionary['after']

print("value = ", value)

print("message = ", message, "\n")

while value >= 0:

print(value)

sleep(1000)

value = value - 1

print(message)

print()

1. Write a paragraph on what you learned from this experiment.

# Experiment #2 Radio Data Packets

Now we will modify the sender and receiver for the countdown program.

**# countdown\_sender\_mod.py**

from microbit import \*

import radio

radio.on()

radio.config(channel=7,length=50)

sleep(1000)

print("Countdown App")

print("micro:bit sender")

while True:

text = input("Enter countdown start: ")

value = int(text)

text = input("Enter ms time between counts: ")

ms = int(text)

message = input("Enter message after countdown: ")

dictionary = { }

dictionary['start'] = value

dictionary['time'] = ms

dictionary['after'] = message

packet = str(dictionary)

print("Send: ", packet)

radio.send(packet)

print()

**# countdown\_receiver\_mod.py**

from microbit import \*

import radio

radio.on()

radio.config(channel=7,length=50)

sleep(1000)

print("Countdown App")

print("micro:bit receiver\n")

while True:

packet = radio.receive()

if packet is not None:

print("Receive: ", packet)

print()

print("Parse: ")

dictionary = eval(packet)

value = dictionary['start']

ms = dictionary['time']

message = dictionary['after']

print("value = ", value)

print("ms = ", ms)

print("message = ", message, "\n")

while value >= 0:

print(value)

sleep(ms)

value = value - 1

print(message)

print()

1. Write a paragraph on what you learned from this experiment.

**Part 3: Encryption**

# Encryption Intro

Diagram

Description automatically generated

### What it's about

Encryption is the process of encoding information into an alternate form that is unintelligible.  Decryption is the process of decoding that alternate form to get the original information.

Encryption is used to protect private information as it is transmitted through channels that can be monitored, like radio communication and the Internet.  Most web sites, apps, and servers communicate over secure channels, meaning that they encrypt the data before sending and decrypt after receiving.

In this tutorial, you will start with some basic encryption concepts using one of the oldest and most simple forms of encryption, the Caesar cipher.  Above is a picture of the Caesar cipher with a key of 5 applied to the first two characters in “HELLO”.  The encrypted version of the H E in HELLO is M J.  Can you complete the last three characters in the ciphertext?

Encrypt Letters with a Caesar Cipher Script

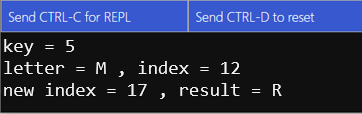
Alright, are you ready to write a script that does what you just did by hand?

Before getting started, think about this set of steps as a conversion between by-hand to pseudo code for programming.

1. Store the key in a variable
2. Store the plaintext letter you are starting with in a variable
3. Create an alphabet
4. Find the index of that plaintext letter in the alphabet
5. Add the key to the index
6. Find and store the ciphertext letter at the new index

# Experiment #1 Encrypt a Single Letter

This next script will encrypt individual letters with the Caesar cipher.  As written, it encrypts the letter M with a key of 5.



**# caesar\_encrypt\_letter.py**

from microbit import \*

sleep(1000)

key = 5

letter = "M"

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

index = alpha.find(letter)

print("key =", key)

print("letter =", letter, ", index =", index)

index = index + key

index = index % 26

result = alpha[index]

print("new index =", index, ", result =", result)

print()

1. Write a paragraph on what you learned from this experiment.

### **Experiment #2 Caesar Cipher on Terminal-Entered Characters**

In this example, the terminal will prompt you for a key and a letter, and then display the Caesar cipher result.  Previously, the encryption key was hard coded as 5.  With this script, you can enter 5, or 13 for ROT-13, or any other value you decide to use.  Next, enter the letter to encrypt, and the script displays the ciphertext result in the terminal.

Text

Description automatically generated

**# caesar\_terminal\_letters.py**

from microbit import \*

sleep(1000)

print("Set your keyboard to CAPS LOCK.")

print()

while True:

text = input("Enter key: ")

key = int(text)

letter = input("Enter a letter: ")

letter = letter.upper()

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

index = alpha.find(letter)

index = index + key

index = index % 26

result = alpha[index]

print("result =", result)

print()

1. Write a paragraph on what you learned from this experiment.

# Experiment #3 Apply Caesar Cipher to Words from Terminal

This script places the Caesar cipher in a loop.  This will allow you to type entire words for encrypting/decrypting!  It still also works with single characters if that’s all you want to encrypt.

Text

Description automatically generated

**# caesar\_terminal\_words.py**

from microbit import \*

sleep(1000)

print("Set your keyboard to CAPS LOCK.")

print()

while True:

text = input("Enter key: ")

key = int(text)

word = input("Enter character(s) in A...Z range: ")

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

result = ""

for letter in word:

letter = letter.upper()

index = ( alpha.find(letter) + key ) % 26

result = result + alpha[index]

print("result:", result)

print()

1. Write a paragraph on what you learned from this experiment.

# Experiment #4 Caesar Cipher in a Function

Why move the Caesar cipher routine to a function?  One advantage would be that you can swap it out with other, better encryption functions, or even function/method calls to a module.  As an example, in the next activity, you will replace the **caesar** function with another one called **ascii\_shift**.  After the function swap, your script will only need one line changed!

**# caesar\_cipher\_function.py**

from microbit import \*

''' Function converts plaintext to ciphertext using key '''

def caesar(key, word):

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

result = ""

for letter in word:

letter = letter.upper()

index = ( alpha.find(letter) + key ) % 26

result = result + alpha[index]

return result

sleep(1000)

print("Set your keyboard to CAPS LOCK.")

print()

while True:

text = input("Enter key: ")

key = int(text)

letters = input("Enter character(s) in A...Z range: ")

result = caesar(key, letters)

print("result:", result)

print()

1. Write a paragraph on what you learned from this experiment.

### **Experiment #5 ASCII Shift Cipher**

The Caesar cipher works well as an introduction to ciphers, but it’s not overly practical.  With only 25 keys and every word separated by a space, it’s definitely one of the easiest ciphers to crack.  The Caesar cipher is also not a very good fit for encrypting radio data since CAPS LOCK letters with no other characters or spaces would make a script to send an encrypted version of this dictionary really difficult:

**{'start' : 3, 'after' : 'Liftoff! '}**

The ASCII Shift Cipher works on all printable characters, including spaces, so that dictionary string would be no problem to encrypt and decrypt with ASCII Shift.  Although it’s still considered very weak in the encryption world, 93 different keys is still more secure than 25.

Text

Description automatically generated

**# ascii\_shift\_cipher.py**

from microbit import \*

''' Function converts plaintext to ciphertext using key '''

def ascii\_shift(key, text):

result = ""

for letter in text:

ascii = ( ord(letter) + key - 32 ) % 94 + 32

result = result + chr(ascii)

return result

''' Script starts from here... '''

sleep(1000)

while True:

text = input("Enter key: ")

key = int(text)

text = input("Enter printable character(s): ")

result = ascii\_shift(key, text)

print("result:", result)

print()

1. Write a paragraph on what you learned from this experiment.

# Experiment #6 Substitution Ciphers

Another encryption example is the substitution cipher.  With a substitution cipher, each character in an alphabet maps to a cryptabet with different characters in the same position.  The simplest example of this is the Atbash or reverse-alphabet cipher.

Alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ  
Cryptabet: ZYXWVUTSRQPONMLKJIHGFEDCBA

Graphical user interface, application

Description automatically generated with medium confidence

**# substitution\_cipher\_atbash.py**

from microbit import \*

# Atbash cipher.

def atbash(text):

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

crypta = "ZYXWVUTSRQPONMLKJIHGFEDCBA"

result = ""

for letter in text:

letter = letter.upper()

index = alpha.find(letter)

result = result + crypta[index]

return result

# The script starts executing statements from here.

sleep(1000)

print("Set your keyboard to CAPS LOCK.")

print()

while True:

plaintext = input("Enter a CAPS LOCK string: ")

result = atbash(plaintext)

print("result =", result)

1. Write a paragraph on what you learned from this experiment.