Receiver Notes and Hints

python.microbit.org/v/3/

Start by importing all the modules / dependencies you will need to complete the task; you want to use the `radio` module.

Enable the radio on the micro:bit.

> Reference > Radio > On and Off

The radio channel needs to be set to the same as the micro:bit transmitting the message.

> Reference > Radio > Groups

The group number could be set to ANY integer number between 0 and 255 - I will not tell you which it is. You have two options here:

1. Manually try each possible value (boring)
2. Use a `for` loop to get the computer to try each value for you!

If you can, try to define a function called `find_radio_group` to *encapsulate* this process. This separation of concerns will help both reusability and readability.

> Reference > Functions > Procedures

An example of what this could look like written in Psuedo-code:

```
define function find_radio_group, no parameters
    for number from 0 to 255
        set group to number
        receive message
        if message is present
            break from loop (and function)
```

This procedure would iterate through each integer value from 0 to 255. It would then set the radio group to that number. If it receives a message with content it will break from the loop, and the radio group will be set to the correct value.

It may also be worth displaying the current loop value to the micro:bit's LED panel...

> Reference > Display > Show

Next comes the control loop. The control loop is the `while True` loop that will run infinetly whilst the device is powered.

This is where you want to call your `find_radio_group` procedure.

> Reference > Functions

Then, attempt to `receive` a message. If you dont receive a message with content, `continue` to the next iteration of the `while` loop.

> The keyword `continue` can be used here to skip the remaining code in the loop and move on to the next iteration to try again.
> You could add an attempt counter here to ensure the program doesn't run forever.

If you have received a message, output it to the console and / or to the LED panel.

> Reference > Display > Scroll

This section of the code should look like:

```
while true
    call find_radio_group
    receive message
    if message is empty
        continue to next iteration
    output message
```

Extension

> This extension assumes some level of confidence in Python. It can also be completed on paper if preferred.

What does `"ifmmp xpsme"` mean?
The message is encrypted using a Ceasar Shift!

A Ceasar Shift Decoder can be written here as a function. Unlike the `find_radio_group()` procedure you wrote earlier, a function takes in arguments.

```
def function_name(arg1, arg2):
    ...
```

For a Ceasar Shift we have two arguments to pass in:

- The Encrypted Message
- The Magnitude of the Shift

> `"a"` --> `"b"` would have a shift magnitude of `1`.

In psuedo-code this function would look like:

```
define 'decode' with parameters 'encrypted' and 'shift'
    initialise empty string called 'decoded'
    initialise string 'alphabet'
    initialise string 'numbers'
```

```
    for character in encrypted message
        if character is in the alphabet
            find position of the character in 'alphabet'
            add the letter at index '(value - shift) MOD 26' to 'decoded'
        else if character is in the 'number' string
            find position of the character in 'numbers'
            add the letter at index '(value - shift) MOD 10' to 'decoded'
        else
            # assume value is punctuation, doesn't need to be shifted.
            add the character to 'decoded'
```

```
alphabet = "abcdefghijklmnopqrstuvwxyz"
digits = "0123456789"
```

> Information on the MOD / Modulus operator can be found below.

You then need to call the `decode` function from the control loop.

As you don't know the value of the shift, use a `for` loop to iterate through each possibility (0 to 26).

```
for shift in range(0, 26):
    decrypted_message = decode(message, shift)
    print(decrypted_message)
```

MOD

In Python, the Modulus operator (`%`) returns the remainder of a division.

```
37 / 26 = 1.42... # Division
37 // 26 = 1      # Integer (Whole Number) Division
37 % 26 = 11      # Modulus
```

```
100 % 9 = 1
# Because 100 / 9 = 11 with a remainder of 1.
```