|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TITLE  Engineering Work Experience Software Activity: Decrypting Hidden Transmissions with the BBC micro:bit | | | Document Number:  Revision:  Change Number: | TODO  1  TODO | |
|  | | | | | |
|  |  | | |  |  |
| |  | | --- | |  | |  | | This document/file does not contain any export controlled technical data (NSR) | | | |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | | |  |
|  |
|  | | |  |
|  | | |
| THIS DOCUMENT IS THE PROPERTY OF COLLINS AEROSPACE AND CONTAINS CONFIDENTIAL AND/OR PROPRIETARY INFORMATION. YOU MAY NOT POSSESS, USE, COPY OR DISCLOSE THIS DOCUMENT OR ANY INFORMATION IN IT, FOR ANY PURPOSE, INCLUDING WITHOUT LIMITATION, TO DESIGN, MANUFACTURE OR REPAIR PARTS, OR OBTAIN ANY GOVERNMENT APPROVAL TO DO SO, WITHOUT COLLINS AEROSPACE’S EXPRESS WRITTEN PERMISSION. NEITHER RECEIPT NOR POSSESSION OF THIS DOCUMENT ALONE, FROM ANY SOURCE, CONSTITUTES SUCH PERMISSION. POSSESSION, USE, COPYING OR DISCLOSURE BY ANYONE WITHOUT COLLINS AEROSPACE’S EXPRESS WRITTEN PERMISSION IS NOT AUTHORIZED AND MAY RESULT IN CRIMINAL AND/OR CIVIL LIABILITY. | | |
|  |
| Author: | Corey Richardson | | |
| Date: | 29/06/23 | |  |
|  | | | |

This page is intentionally left blank

Copyright Statements

Document number: TODO

Entitled: Engineering Work Experience Software Activity: Decrypting Hidden Transmissions with the BBC micro:bit.

This is an unpublished work created in 2025, any copyright in which vests in Atlantic Inertial Systems Limited. All rights reserved.

The information contained in this document is made available in confidence.

Public authorities are prohibited from releasing the information unless its release would not constitute an actionable breach of confidence. Public authorities should contact Atlantic Inertial Systems Limited to determine the current releasability of the information.

[5 USC 552(b)(4) and 18 USC 1905]/ [Sections 41 and 43 of the Freedom of Information Act 2000] are applicable.

UK Origin

Any enquiries relating to this document or its contents should be addressed in the first instance to:

Atlantic Inertial Systems Limited,  
Clittaford Road,  
Southway,  
Plymouth,  
Devon  
PL6 6DE

Telephone: (01752) 695695 International: +44 1752 695695  
Fax: (01752) 695500 International: +44 1752 695500

Atlantic Inertial Systems Limited is part of Collins Aerospace Systems.

Change History

| Revision | Change no | Date | Author | Summary of Change |
| --- | --- | --- | --- | --- |
| 1 | TODO | 29/06/2023 | C. Richardson | First Issue |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |



CONTENTS

Section Page

[Glossary 7](#_Toc138945116)

[1 Introduction 8](#_Toc138945117)

[1.1 About this document 8](#_Toc138945118)

[2 Prior and Prerequisite Knowledge 14](#_Toc138945119)

[2.1 Portfolio 14](#_Toc138945120)

[2.2 Indexing Into a List 15](#_Toc138945121)

[2.3 String .find() Method 15](#_Toc138945122)

[2.4 Python Structure Guide: Indentation 16](#_Toc138945123)

[2.5 Flashing Code to the BBC micro:bit 17](#_Toc138945124)

[3 Task 1: Decrypting a Secret Transmission 18](#_Toc138945125)

[3.1 Template 18](#_Toc138945126)

[3.2 Importing Modules 18](#_Toc138945127)

[3.3 Receiving a Message 18](#_Toc138945128)

[3.4 Display the Message 19](#_Toc138945129)

[3.5 Writing a Function 19](#_Toc138945130)

[3.6 Maintainability and Comments 20](#_Toc138945131)

[3.7 Decoding the Message 21](#_Toc138945132)

[3.8 Pseudocode Structure 22](#_Toc138945133)

[4 Task 2: Manual Vs. Software Decryption (Extension) 25](#_Toc138945134)

[4.1 Caesar Shift Decoder Function 25](#_Toc138945135)

[4.2 User Input 26](#_Toc138945136)

[5 Solution Code 29](#_Toc138945137)

[5.1 Task 1 29](#_Toc138945138)

[5.2 Task 2 (Extension) 30](#_Toc138945139)

[6 Transmitting Script 31](#_Toc138945140)

[7 Language Comparison: Python vs C# 37](#_Toc138945141)

[7.1 Dynamically Typed Languages 45](#_Toc138945142)

[7.1.1 Advantages 49](#_Toc138945143)

[7.1.2 Disadvantages 52](#_Toc138945144)

[7.1.3 Caesar Shift decode() Function - Python 57](#_Toc138945145)

[7.2 Statically Typed Languages 61](#_Toc138945146)

[7.2.1 Advantages 65](#_Toc138945147)

[7.2.2 Disadvantages 69](#_Toc138945148)

[7.2.3 Caesar Shift Decode() Function – C# 74](#_Toc138945149)

Tables

No table of figures entries found.

Figures

[Figure 1: BBC micro:bit Python Editor 8](#_Toc138945150)

[Figure 2: Indexing Into a List 15](#_Toc138945151)

[Figure 3: String .find() Method 15](#_Toc138945152)

[Figure 4: Template Code 18](#_Toc138945153)

[Figure 5: Display the Message Pseudocode 19](#_Toc138945154)

[Figure 6: Python Function Template 19](#_Toc138945155)

[Figure 7: Example of Good Use of Comments 20](#_Toc138945156)

[Figure 8: Example of Bad Use of Comments 20](#_Toc138945157)

[Figure 9: Task 1 Pseudocode 24](#_Toc138945158)

[Figure 10: Caesar Shift Decoder Function Pseudocode 25](#_Toc138945159)

[Figure 11: Functions as Arguments 25](#_Toc138945160)

[Figure 12: User Input Pseudocode 26](#_Toc138945161)

[Figure 13: Task 1 Solution Code 29](#_Toc138945162)

[Figure 14: Task 2 (Extension) Solution Code 30](#_Toc138945163)

[Figure 15: Transmitter Code 36](#_Toc138945164)

[Figure 16: Declaring an Integer Variable - Python 48](#_Toc138945165)

[Figure 17: Caesar Shift Decode() Function - Python 58](#_Toc138945166)

[Figure 18: Declaring an Integer Variable - C# 64](#_Toc138945167)

[Figure 19: Caesar Shift Decode() Function - C# 75](#_Toc138945168)

Glossary

|  |  |
| --- | --- |
|  |  |
| BBC  LED  NASA JPL | British Broadcasting Corporation  Light Emitting Diode  National Aeronautics and Space Administration Jet Propulsion Laboratory |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Introduction

## About this document

This document will guide your through a Software and Electronics activity to complete on the BBC micro:bit platform, as part of your Work Experience week.

The task can be completed using either the Microsoft MakeCode block-based programming language, or the preferred Python.

Online code editors are provided on the BBC micro:bit website at [microbit.org/code/](https://microbit.org/code/).

|  |
| --- |
| Figure 11: BBC micro:bit Python Editor |

*At this stage, you should have already completed the* ***Meet your micro:bit*** *exercise project.*

The contents of this project have been adapted from **NASA JPL’s ‘Coding a Radio Message for Space’** project (<https://www.jpl.nasa.gov/edu/teach/activity/coding-a-radio-message-for-space/>) and has been split into two tasks.

The aim of this project is to provide you with experience in embedded programming using Python/MicroPython and to teach “Computational Thinking”, logic and control flow. Computational Thinking is essential to the development of computer applications but it can also be used to support problem solving across all disciplines, including maths, science and humanities.

# Prior and Pre-requisite Knowledge

This section provides you with the key skills, or “pre-requisite” knowledge you need to progress. Please complete the knowledge capture before moving on.

## Knowledge Capture

Please fill in this table to assess your prior programming knowledge. Then, once you have completed the task revisit this section and update accordingly.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 11: Knowledge Capture Table   |  |  |  | | --- | --- | --- | | Process | Prior Knowledge | After Task | | Using a micro:bit | Close with solid fill | Checkmark with solid fill | | ­­­­­Importing Modules |  |  | | if Statements |  |  | | if-elif-else Statements |  |  | | while Loops |  |  | | for Loops |  |  | | continue Keyword |  |  | | Functions and Procedures |  |  | | User Input |  |  | |

|  |
| --- |
|  |

## Indexing Into a List

Use square brackets after the list name to index into a list.

Python uses zero-indexing for lists.

|  |
| --- |
| Figure 22: Indexing Into a List |

## String .find() Method

The **.find()** method finds the first occurrence of the specified value.

The **.find()** method returns `-1` if the value is not found.

|  |
| --- |
| Figure 33: String .find() Method |

## Python Structure Guide: Indentation

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a *block* of code. Indentation refers to the spaces at the beginning of a code line.

In this code, Python can see that the **print** statement *belongs to* the **if** statement and will only be run if the *condition* **5 > 2** is **True**.

The second **print** statement isn't part of the block, and so will run even if the **if** statement condition returns **False**.

|  |
| --- |
| Figure 44: Good Indentation Example |

Python will give you an error if you skip the indentation:

|  |
| --- |
| Figure 55: No Indentation Example |

Traditionally, 4 spaces are used as indentation, however you can use as many as you prefer to **as long as you are consistent**. You must use the same number of spaces in the same block of code, otherwise Python will give you an error.

|  |
| --- |
| Figure 66: Inconsistent Indentation Example |

## Flashing Code to the BBC micro:bit

To flash your code, you need to connect the micro:bit to your computer using the Micro-USB cable provided. Then, in the web-based code editor, click **Send to micro:bit** and follow the on screen instructions.

This can also be done wirelessly, using a mobile app, however this is not covered in this guide.

# Task 1: Decrypting a Secret Transmission

Your mission is to intercept the secret message being transmitted. You will need to:

* Conduct a “Digital Treasure Hunt” to solve clues and detect the transmission
* Use the clues to reveal the cipher
* Decipher the message

The subsequent headings are here to guide you through the process.

## Template

Getting started: here is the code you need:

|  |
| --- |
| Figure 77: Template Code |

## Importing Modules

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

To 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. For now, set this to any number between 0 and 255.

**Reference > Radio > Groups**

## Receiving a Message

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

Remove the ellipsis placeholder. Similar to using three dots in English to omit content, you can use the ellipsis in Python as a placeholder for unwritten code (Looks a bit like this ‘…’)

When a button is pressed, attempt to **receive** a message. Assign this to button **B**.

**Reference > Buttons > Button was pressed**

**Reference > Radio > Receive a message**

If you don’t receive a message with content, **continue** to the next iteration of the **while** loop.

If you don't receive a message with content, `continue` to the next iteration of the `while` loop.

The *logical operator* **not** can be used here to determine if the variable has no value: **if not variable\_name:**.

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.

## Display the Message

If you have received a message, output it to the console and/or to the micro: bit’s LED panel.

**Reference > Display > Scroll**

Written in *pseudocode*, this section of the code will look like:

|  |
| --- |
| Figure 88: Display the Message Pseudocode |

## Writing a Function

A function is a "chunk" of code that you can use over and over again, rather than writing it out multiple times. Functions enable programmers to break down or *decompose* a problem into smaller chunks, each of which performs a particular task. The basic structure for a function in Python is as so:

|  |
| --- |
| Figure 99: Python Function Template |

**Reference > Functions**

Upon testing you find that the **scroll** method doesn't give you enough time to record the message being output by the micro:bit. Replace this line with a function call to **display\_message**, passing in the **message** as an argument.

If you tried to run the program now, an *error* would occur as we have not yet provided a *definition* for this function!

Return towards the start of your program and define the function you have just called.

In this function, *iterate* through each **letter** in **message** using a **for** loop, and use the **display.show()** method to output this letter to the LED panel.

Remember here that **message** is a *string*!

**Reference > Display > Show**

**Reference > Loops > For Loops > Letters**

Use the micro:bit's **sleep(ms)** function to create a delay between showing each character, using an appropriate delay as the argument.

You may also want to use the **display.clear()** function after each letter to ensure there is a clear distinction between repeating letters.

## Maintainability and Comments

Commenting in Python is the inclusion of short descriptions along with the code to increase its readability. A developer uses them to record their thought process while writing the code. It explains the basic logic behind why a particular line of code was written.

In Python, comments are written using a hash symbol, **#**.

For example, a *good* example of how to comment your code would be:

|  |
| --- |
| Figure 1010: Example of Good Use of Comments |

Notice how not every line has a comment. Avoid self-explanatory comments such as:

|  |
| --- |
| Figure 1111: Example of Bad Use of Comments |

Add some *descriptive* comments to your code.

This would help you or another developer understand why the code was written in a certain way if you had to come back to this project in a number of years’ time when the original processes have been forgotten.

## Digital Treasure Hunt to Decode the Message

Follow the program flow and determine the output of the code provided. This should be done on paper, and not written into the script.

Here are some of the key things you may want to hunt for... (Use the internet)

* **[::-1]**
* **[0:3]**
* **[-1]**
* **range(a, b, c)**
* **.join([a, b])**
* **//**



### Clues…

|  |
| --- |
| Figure 1212: do\_something function |
| What is the effect of this function in Figure 1212?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Figure 1313: do\_something\_else function |
| What is the effect of Figure 1313?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Figure 1414: value from for loop |
| What is the Value from Figure 1414 ?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Figure 1515: value // 10 function |
| What is the Value from Figure 1515?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Figure 1616: Solve the do\_something\_else function |
| Radio Group = *\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Shift = *\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Encrypted Message: \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_  Decrypted Message: \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ |

## Pseudocode Structure

Pseudocode is a plain language, description of steps needed for an algorithm. It often follows structural conventions of the programming language, but is intended to be human-readable.

Software Engineers often use pseudocode to help plan and structure their code.

Figure 1717 provides an example of this task, in pseudocode form.

|  |
| --- |
| Figure 1717: Task 1 Pseudocode |

# Task 2: Manual Vs. Software Decryption (Extension)

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

## Caesar Shift Decoder Function

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

Whilst it is certainly possible, it would become very tedious if you had to keep decrypting these messages yourself. What if the messages you needed to decrypt were longer, or you had multiple to decrypt? You could use an *automation script* to do this for you.

A Caesar Shift Decoder can be written here as a function. For a Caesar Shift we have two arguments to pass in:

* The Encrypted Message
* The Magnitude of the Shift

In pseudocode this function would look like Figure 1818:

|  |
| --- |
| Figure 1818: Caesar Shift Decoder Function Pseudocode |

*Constants* are unchanging variables. They are written in all capital letters with underscores separating words. Examples include **MAX\_OVERFLOW** and **TOTAL**.

You should then call the **decode** function from the control loop as an argument of **display\_message**. Add this to the area where you receive the message when a button is pressed.

Functions can be arguments to other *functions* and *procedures*.

|  |
| --- |
| Figure 1919: Functions as Arguments |

## User Input

Add functionality to set the magnitude of the **shift** using the micro:bit. This could be done with button presses which increment a *counter variable*. Assign this to button **A**. Don't allow this value to be greater than 9.

|  |
| --- |
| Figure 2020: User Input Pseudocode |

Encrypted Message:

\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_

Decrypted Message:

 \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_

**NOTE: The following page contains the solution code, do not turn over until you have finished the task!**

###### Coding Solutions

This section contains the solutions to the software tasks. **These should only been reviewed upon completion of the task, or if the task leader allows.**

Task 1

This is the suggested solution to task 1.

|  |
| --- |
| Figure 2121: Task 1 Solution Code |

Task 2 – Extension task

This is the suggested solution to task 2.

|  |
| --- |
| Figure 2222: Task 2 (Extension) Solution Code |







Transmission Code

For the micro:bit to receive a message, there must also be a micro:bit in range transmitting a message.

This is the suggested code to flash onto a micro:bit, being sure to set the **SECRET** and **SHIFT** values to the values calculated in 3.7.

|  |
| --- |
| Figure 2323: Transmitter Code |



###### Language Comparisons

(pronounced C Sharp)

of the subsequent sections between4Inspect them both, and see if you can spot the differences.

Dynamically Typed Languages

In dynamically typed languages, type checking takes place at runtime or execution time. This means that variables are checked against types only when the program is executing. There is no requirement to explicitly state the data type during variable declarations.

Python is a dynamically typed language. To declare an integer (whole number) variable called **counter** you would write:

|  |
| --- |
| Figure 2424: Declaring an Integer Variable - Python |

Advantages:

* It tends to reduce unnecessary, duplication and repetition of code.
* Its code can be used *polymorphically* without programmer decoration.

Disadvantages:

* More errors are detected at runtime and in the final code.
* It tends to prohibit compilation and yields poor performance of code.

Caesar Shift decode() function – Python

An example of the Caesar Shift decode function, created in Python:

|  |
| --- |
| Figure 2525: Caesar Shift Decode() Function - Python |

Statically Typed Languages

In statically typed programming languages, type checking occurs at compile time. At compile time, source code in a specific programming language is converted to a machine-readable format. This means that *before* source code is compiled, the type associated with every variable must be known. The programmer is required to explicitly state the data type of each variable when the variable is being declared.

C# is a statically typed language. To declare an integer (whole number) variable called **counter** you would write:

|  |
| --- |
| Figure 2626: Declaring an Integer Variable - C# |

Advantages:

Disadvantages

* It can lead to verbose type declarations.
* It can be slower to write code.

Caesar Shift decode() function – C#

This is a suggested coding solution to the Caesar Shift, using C#.



Figure 2727: Caesar Shift decode() function - C#















This is the final page of this document