PYTHON PROGRAMMING: ROBOTICS FOUNDATION COURSE

FOR YOUNG INNOVATORS

KASHIF BAIG

Python Programming: Robotics Foundation Course

First Edition 2026

About the Author

Kashif Baig is an IT professional with more than 30 years of industry experience spanning enterprise software development, data warehousing, analytics and reporting.

Kashif is also the author of Arduino Web Development: Pushing the Limits and has a long-standing passion for innovation, embedded systems, automation, and practical technology education.

Readers can find out more about the author at https://www.linkedin.com/in/kashifbaig

Contents

1 Introduction

This course has been designed for students who have already acquired a basic understanding of Python programming and are ready to develop their skills further through a series of practical and progressively challenging projects. Students should be familiar and comfortable with most of the topics covered in the warmup exercises of section 2. The projects in section 3 provide opportunities to apply fundamental programming concepts while introducing more advanced techniques that are commonly used in real-world software development.

To make the learning experience both interesting and engaging, many of the projects incorporate simple interactions with sensors and actuators. Sensors are devices that report physical measurements, such as light, sound or temperature. Actuators are devices that perform physical actions, such as displays, beepers or motors. Projects using these devices help demonstrate how software can be used to monitor, control, and respond to events in the physical world. By working with tangible inputs and outputs, students can see the immediate effects of their code and gain a deeper appreciation of how programming is used in a wide range of technological systems.

The sensors and actuators that students will be writing programs for are part of a readily available All-in-One starter kit for Arduino. Students are not required to have this equipment as it is made available during the online supervised practical sessions.

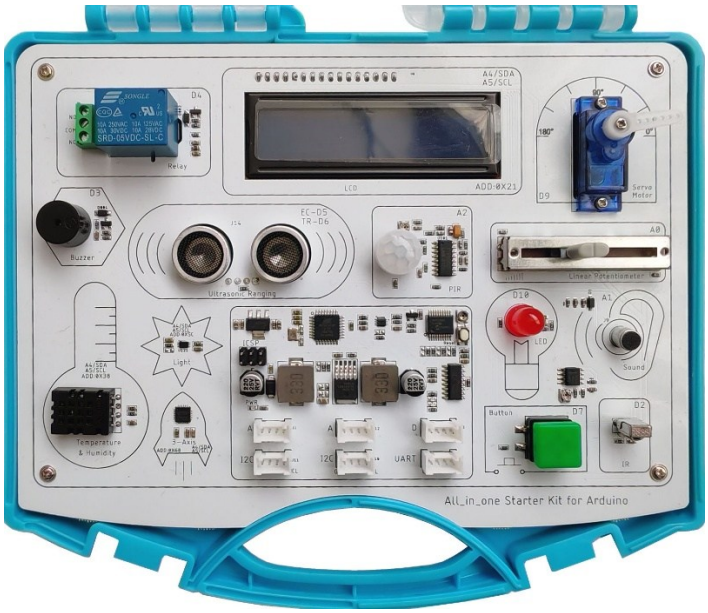

Figure 1-1 Sensors and actuators in the All-in-One Starter Kit for Arduino

The projects begin with straightforward requirements designed to lay the foundations of the course. As students progress through the course, the challenges become gradually more demanding, furthering concepts such as loops, if-then-else decision making, string manipulation, reading and processing text files, array searching and sorting techniques, and the use of SQL for data retrieval. Projects typically build upon knowledge and skills developed in previous activities, encouraging a structured and incremental approach to learning. This is often in the form of additional requirements to be implemented once a project's core objectives have been achieved.

Students will encounter a variety of programming tasks throughout this course. Depending on the objectives of a particular project, they may be required to write Python code from scratch, modify existing code, or adapt solutions developed in earlier projects. This variety reflects the types of tasks programmers commonly perform in professional environments and helps develop problem-solving, debugging, and code interpretation skills.

To support independent learning, some projects may require students to consult online documentation, programming references, or instructional resources. Video demonstrations are also provided to illustrate the required program behaviour or expected outcomes. Developing the ability to locate, interpret, and apply technical information is an important skill for all programmers and forms part of the learning process.

Students are expected to complete one or more projects in electronic format before attending supervised practical sessions. During these sessions, students will implement their proposed Python solutions and test their programs in a practical environment. The tutor will review progress, provide guidance where necessary, and offer feedback on both the approach and the final implementation. This combination of preparation, practical application, and tutor support is intended to strengthen understanding and build confidence in the development of effective programming solutions.

2 Python Warm-Up Exercises

2.1 Introduction

Before starting the main projects of the course in section 3, it is useful to review some core Python programming concepts.

The following warm-up exercises are designed to help the student practise the programming techniques that will be used throughout the course. The exercises gradually increase in difficulty and cover topics commonly found within GCSE-level Computing courses.

Some questions require the student to write Python code, while others use multiple-choice or short-answer formats.

2.2 Warm-Up Exercises

2.2.1 Variables and Assignment

Practise creating variables and assigning values.

1. Write Python statements to create the following variables:

Variable	Value
`name`	`"Sam"`
`age`	`15`
`height`	`1.72`

2. What will be displayed when the following code is run?

```
age = 15
print(age)
```

A. "age"
B. 15
C. "15"
D. Error

3. What value will x contain?

```
x = 10
x = x + 5
```

A. 5
B. 10
C. 15
D. 20

2.2.2 Outputting to the Console

Practise displaying information using `print()`.

4. Write a Python statement that displays:

```
Hello World
```

5. What will be displayed when the following code is run?

```
name = "Alex"
print("Hello", name)
```

A. Hello Alex
B. name
C. Hello
D. Error

6. Correct the erroneous statement:

```
print(The quick brown fox)
```

2.2.3 Arithmetic Operators

Practise mathematical calculations.

7. Given the value of `x` is `10` at the start of each row, complete the table.

Expression	Result
`x += 5`	?
`x --`	?
`x ++`	?
`x /= 2`	?
`x =10 % 3`	?

8. What does the `%` operator do?

A. Division
B. Multiplication
C. Remainder after division
D. Power

9. Write Python statements to create a variable named radius and calculate and print the area of a circle.

2.2.4 String Manipulation

Practise working with text strings.

10. What is the datatype of the value stored in `message`?

```
message = "integer"
```

11. What will be displayed when the following code is run?

```
word = "Code"
print(word[0])
```

A. C
B. o
C. d
D. Code

12. Write code that joins:

```
"Hello"
and
"World"
```

to create:

```
Hello World
```

2.2.5 Selection (if Statements)

Practise making decisions.

13. Replace the underscored parts to complete the code.

```
temperature = 28
if temperature __ 25:
    _______
```

The program should display:

```
Hot
```

14. What will be displayed when the following code is run?

```
age = 17
if age >= 16:
    print("Can apply")
```

A. Nothing
B. Can apply
C. 16
D. Error

15. Write an if-else statement that displays:

"Merit" if a score is 75 or higher.
"Pass" if a score is 60 or higher, but less than 75.
"Fail" otherwise.

2.2.6 Repetition Using Loops

Practise repeating instructions.

16. How many times will the loop execute?

```
for i in range(5):
    print(i)
```

A. 4
B. 5
C. 6
D. Infinite

17. What will be displayed when the following code is run?

```
for i in range(3):
    print("Hi")
```

18. Write a loop that displays:

2
4
6
8

2.2.7 While Loops

Practise condition-controlled repetition.

19. What will be displayed when the following code is run?

```
count = 1
while count <= 3:
    print(count)
    count += 1
```

20. Why might the following loop cause a problem?

```
while True:
    print("Hello")
```

21. Write a loop that displays numbers from 10 down to 1.

2.2.8 Using Lists

Practise storing multiple values.

22. Explain what is stored in `colours`:

```
colours = ["red", "green", "blue"]
```

23. What will be displayed when the following code is run?

```
colours = ["red", "green", "blue"]
print(colours[1])
```

A. red
B. green
C. blue
D. Error

24. Write a Python statement to add "yellow" to the end of the list in question 23.

2.2.9 Searching a List

Practise searching for data.

25. Given the list below:

```
numbers = [5, 8, 2, 7, 9]
```

Write a loop that displays every value.

26. Which value will be found first when searching for the value `12`?

```
numbers = [4, 8, 12, 16]
```

27. Write code that determines whether the value 7 exists in the list of question 26.

2.2.10Counting

Practise using counters.

28. What is the final value of `count` when the code below is run?

```
count = 0
for i in range(4):
    count += 1
```

29. Write code that counts how many numbers in the following list are greater than `10`.

```
numbers = [5, 11, 7, 18, 12]
```

30. What is the purpose of a counter variable?

A. Store text
B. Repeat loops

C. Count occurrences of something
D. Store colours

2.2.11 Functions

Practise creating reusable code.

31. Define a function named `display_name()` that displays:

```
Python Student
```

32. What is the purpose of a function?

A. Store data
B. Group reusable instructions
C. Create loops
D. Create variables

33. Complete the function:

```
def square(number):
    return __________
```

2.2.12 File Handling

Practise reading text files. Assume the file `fruit.txt` contains:

```
Apple
Banana
Orange
Lime
```

34. What does the following statement do?

```
file = open("fruit.txt")
```

35. What will be stored in `line` after:

```
line = file.readline()
```

A. Apple
B. Banana
C. Lime
D. Entire file

36. Why should files be closed after use?

2.2.13 Multi-dimensional Lists

Practice processing structured data.

Given:

```
students = [
    ["Alice", 14],
    ["Ben", 15],
    ["Chris", 16]
]
```

37. What does the following statement return?

```
students[0][0]
```

38. What does this next statement return?

```
students[1][1]
```

39. Write code that displays all student names.

3 Programming Projects Using Sensors and Actuators

3.1 Basic Structure of the Python Projects

Most of the projects in this section are based on the Python program template below:

```
from app_config import *

try:
    # Connect to All-in-One kit
    all_in_one_kit.Connect()
    print("Press Esc to stop program.")

    # Main loop. Keep looping until Escape key pressed
    while not escape_pressed():
        pass

finally:
    all_in_one_kit.Close()
```

The template imports essential libraries (which declare variables for sensors and actuators), and performs the tasks required to communicate with the All-in-One Starter Kit for Arduino. It provides a structure that can be extended to implement different solutions. Once connected, the program enters a continuous loop until the Escape (Esc) key is pressed on the keyboard, at which point the program disconnects from the All-in-One starter kit and ends. Variables needed for the duration of the program may be defined at a suitable point before the `while` loop, and (unless stated otherwise) any logic that needs to run repeatedly should be within the `while` loop.

Note, students are not required to have the aforementioned All-in-One starter kit, as they will have remote access to necessary equipment during the online sessions.

3.2 Example with Solution 1 – Blink an LED Every Second

Modify the above program template so that an LED (Light Emitting Diode) repeatedly turns on and off at one-second intervals. The LED should continue blinking until the Escape key is pressed.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=z8jO9ulyihkXaGLE&t=4

The variable `led` of type Switch is provided for controlling the state of the LED:

Method / Property	Description	Example
led.On()	Turns the LED on.	`led.On()`
led.Off()	Turns the LED off.	`led.Off()`
led.OnForDuration(duration)	Turns the LED on for duration specified in seconds.	`led.OnForDuration(0.5)`
led.IsOn	Returns True if the LED is currently on, otherwise returns False.	`if led.IsOn:`
not led.IsOn	Returns True if the LED is currently off.	`if not led.IsOn:`

Table 3-1 Methods and properties for setting and testing the state of an LED.

Use the Python `time.sleep()` function to temporarily pause program execution for a specified number of seconds between LED blink states:

Statement	Delay
`time.sleep(1)`	Pauses the program for 1 second.
`time.sleep(2)`	Pauses the program for 2 seconds.
`time.sleep(0.5)`	Pauses the program for half a second.

Table 3-2 Python function for pausing program execution for a specified period of time.

3.2.1 Summary of Expected Outcome

When the program runs:

- The LED should switch on.
- One second later it should switch off.
- One second later it should switch on again.
- This process should continue repeatedly until the Escape key is pressed.

3.2.2 Solution for *Blink an LED Every Second*

```
from app_config import *

try:
    # Connect to All-in-One kit
    all_in_one_kit.Connect()
    print("Press Esc to stop program.")

    while not escape_pressed():
        # Alternate the state of the LED
        if not led.IsOn:
            led.On()
        else:
            led.Off()

        # Pause for 1 second
        time.sleep(1)

finally:
    all_in_one_kit.Close()
```

3.3 Example with Solution 2- Control LED Blink Speed Using the Potentiometer

Modify the previous LED blinking solution so that the blink speed is controlled by the position of the sliding potentiometer (slider) on the All-in-One starter kit.

The program should calculate a suitable delay value based on the slider position and use that value to determine how long the program pauses between LED state changes.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=SX96rcm2mP7ZAVDm&t=13

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.3.1 Reading the Position of the Sliding Potentiometer

The provided variable named `slider` of type AnalogInput represents the sliding potentiometer (slider) fitted to the All-in-One starter kit. The slider works in the same way as a control knob, except that the slider has a linear action.

The current position of the slider can be obtained using the `Value` property. When the slider is positioned at one end, the `Value` property reports 0. When positioned at the other end, the `Value` property reports 1023. When the slider is positioned halfway, the value reported is 512.

Property	Description	Possible Values
`slider.Value`	Returns the current position of the potentiometer.	Range from 0 to 1023

Table 3-3 Reading the sliding potentiometer value using variable 'slider'.

3.3.2 Calculating the Blink Interval

The LED should blink slowly when the slider is near its minimum position and blink more quickly as the slider is moved towards its maximum position.

To achieve this, create a helper function named `blink_interval()` that calculates and returns a delay value based on the current slider position.

The returned value should:

- Be approximately between 0.1 and 1.

The returned value must then be supplied to the `time.sleep()` function. As the slider position changes, the blinking speed should change without restarting the program.

3.3.3 Summary of Expected Outcome

When the program runs, the expected behaviour should be as described in the table below:

Slider Position	Expected Behaviour
Low value reported	LED blinks slowly.
High value reported	LED blinks rapidly.
Any value in between	LED blink rate change corresponds accordingly

Table 3-4 Expected outcome of LED when slider position changes.

3.3.4 Solution for *Control LED Blink Speed Using the Potentiometer*

```
# Blinks LED with interval controlled by the potentiometer slider and helper function.

from app_config import *

# Helper function to calculate blink interval using slider position.
def blink_interval()->int:
    ''' Returns an interval value expressed in seconds using the slider position.'''
    return 1/(slider.Value/100 + 1)

try:
    # Connect to All-in-One kit
    all_in_one_kit.Connect()
    print("Press Esc to stop program.")

    while not escape_pressed():
        # Alternate the state of the LED
        if not led.IsOn:
```

```
            led.On()
        else:
            led.Off()

        # Pause for duration determined by slider
        time.sleep(blink_interval())

finally:
    all_in_one_kit.Close()
```

3.4 Example with Solution 3 – Use a Push Button to Start and Stop LED Blinking

Adapt the previous LED blinking program so that a push button is used to control whether the LED blinks.

The program should start with blinking disabled. Each time the button is pressed, the blinking state should toggle between:

- Blinking enabled
- Blinking disabled

When blinking is disabled, the LED must remain off.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=PfPg1yUCs2ZsbrO5&t=31

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.4.1 Detecting Button Presses

The All-in-One starter kit includes a push button that can be used to trigger actions within a program. The provided helper function `button_pressed()` can be used to detect when the button has been pressed.

Function	Description	Return Value

`button_pressed()`	Checks whether a button press has been detected. Only the downward press is detected, returning True. False is returned otherwise.
	
	Example:
	`if button_pressed():`
	`  print("Button pressed")`
		`True` or `False`

Table 3-5 Description and usage of the button_pressed() helper function.

3.4.2 Keeping Track of the Blink Enabled State

A useful programming technique for keeping track of an enabled/disabled state is to alternate the value of a Boolean variable, which can be set to either True or False. Introduce and use a suitably named Boolean variable to keep track of the blink state.

Hint	Description
`blink_on = False`	Creates a variable to store the current blinking state.
`if blink_on:`	Executes code only when blinking is enabled.
`blink_on = not blink_on`	Reverses the current state of the variable.

Table 3-6 Hint for using a Boolean variable for managing the enabled/disabled blink state.

3.4.3 Summary of Expected Outcome

When the program runs, the expected behaviour should be as described in the table below:

Action	Expected Result
Program starts	LED is off and not blinking.
First button press	Blinking starts.
Move slider	Blink speed changes.
Second button press	Blinking stops and LED turns off.
Additional button presses	Blinking continues to toggle on and off.

Table 3-7 Expected outcome when push button is used for controlling LED blink state.

3.4.4 Solution for *Use a Push Button to Start and Stop LED Blinking*

```
# Blinks LED with interval controlled by the potentiometer slider. A push button is used
# for turning the blinking on or off.

from app_config import *

try:
    # Connect to All-in-One kit
    all_in_one_kit.Connect()
    print("Press Esc to stop program.")

    # Variable to control the blinking
    blink_on = False

    while not escape_pressed():
        if blink_on:
            # Alternate the state of the LED
            if not led.IsOn:
                led.On()
            else:
                led.Off()

            # Pause for duration determined by slider
            time.sleep(1/(slider.Value/100 + 1))

        if button_pressed():
            blink_on = not blink_on
            led.Off()
finally:
    all_in_one_kit.Close()
```

3.5 Simulate an Automatic Night Light

Before proceeding any further, please be sure to have thoroughly reviewed sections 3.1 to 3.4

This project simulates the operation of an automatic night light that switches on when a room becomes dark.

Create a program that automatically turns on an LED when the ambient light level falls below a threshold value. The threshold must be adjustable using the sliding potentiometer. As the slider is moved, the light level at which the LED turns on should change. Use the Python template in section *3.1 Basic Structure of the Python Projects* as a starting point.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=2ea7oWlIFVf9Yo-l&t=45

This project introduces sensor initialisation, sensor readings, threshold-based decision making, and a practical real-world control application while reinforcing the use of variables and if-else statements.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.5.1 Obtaining Ambient Light Measurements

The All-in-One starter kit includes a light sensor that can measure ambient light levels. The sensor is represented by the provided variable `light_meter` of type LightMeter. Before the sensor can be used, it must be enabled after connection is made to the All-in-One starter kit.

Command	Description
`light_meter.Enable()`	Enables the light sensor.
`all_in_one_kit.WaitUntilSensorsReady(light_meter)`	Waits until the sensor is ready to provide readings.

Table 3-8 Enabling and waiting for light sensor to become ready.

The current light level can then be obtained using the `LuxValue` property.

Property	Description	Example
`light_meter.LuxValue`	Returns the current ambient light level from 0 to 50000.	`ambient_light = light_meter.LuxValue`

Table 3-9 Using the variable light_meter to read reported LUX values.

Lux (LUX) is a unit used to measure illumination.

Environment	Typical Lux Value
Bright sunlight	10,000+
Well-lit classroom	300–500
Dim room	50–100
Near darkness	Less than 10

Table 3-10 LUX values that correspond to different environments.

In the example project of section 3.3, the slider was used to control the blink speed of an LED. In this project, the slider must be used to set the ambient light threshold. The slider value should be converted into a threshold value that can be compared with the current Lux reading.

The slider position can be obtained using `slider.Value` which reports a value from 0 to 1023.

3.5.2 Outline of Solution Design

Using the Python template in section 3.1:

1. Enable the light sensor after connecting to the All in One kit.
2. Wait until the sensor is ready to provide readings.
3. Inside the main loop, read the current ambient light level from the light sensor.
4. Use the slider position to determine a light threshold value.
5. Compare the ambient light level with the threshold.
6. Turn the LED on when the ambient light level is below the threshold.
7. Turn the LED off when the ambient light level is equal to or above the threshold.

Refer to section 3.2 for methods for controlling the LED.

3.5.3 Summary of Expected Outcome

When the program runs, the expected behaviour should be as described in the table below:

Condition	Expected Result
Ambient light below threshold	LED turns on.
Ambient light above threshold	LED turns off.
Slider moved upwards	LED turns on at higher light levels.
Slider moved downwards	LED turns on only in darker conditions.

Table 3-11 Expected outcome of automatic night light program.

3.5.4 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution to:

1. Print the current Lux reading and calculated threshold value to the computer console.
2. Count the number of times the LED changes state and display the count on the console when the program ends.

3.6 Passive Infra-Red (PIR) Security Light

Extend the Automatic Night Light program (section 3.5) so that the LED only turns on when:

1. The ambient light level is below a threshold set by the slider.
2. Motion is detected by a PIR (Passive Infra-Red) sensor.

This simulates the behaviour of a typical security light commonly found outside homes and buildings.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=f1yvmFBTS0YqBg7p&t=66

This project reinforces sensor integration, Boolean logic, compound conditions using `and`, and the design of simple automated control systems that respond to multiple environmental inputs.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.6.1 How to Detect Motion

A PIR security light uses two conditions to determine whether the light should switch on:

Condition	Requirement
Darkness	The ambient light level must be below a specified threshold.
Motion	Movement must be detected by the PIR sensor.

Table 3-12 Conditions for switching on a PIR security light.

The PIR sensor is represented by the provided variable `motion_detected` of type DigitalInput, and the current state of the PIR sensor can be obtained using its `Value` property.

Property	Description	Possible Values
`motion_detected.Value`	Indicates whether motion has recently been detected.	Remains `True` for a few seconds after motion has been detected, `False` otherwise

Table 3-13 Determining when the PIR sensor has detected motion.

3.6.2 Outline of Solution Design

Based on the solution from the Automatic Night Light project (section 3.5):

1. Enable the light sensor and wait for it to become ready.
2. Inside the main loop:
3. Read the ambient light level.
4. Read the slider position to determine a threshold value.
5. Read the state of the PIR sensor using `motion_detected.Value`.
6. Determine whether:
   - Motion has been detected.
   - The ambient light level is below the threshold.
7. Turn the LED on only when both conditions are true.
8. Turn the LED off when either condition becomes false.

Refer to section 3.2 for methods for controlling the LED.

3.6.3 Summary of Expected Outcome

When the program runs, the expected behaviour should be as described in the table below:

Ambient Light	Motion Detected	LED State
Bright	No	Off
Bright	Yes	Off
Dark	No	Off
Dark	Yes	On

Table 3-14 Expected outcome of PIR security light program.

3.6.4 Additional Requirements

Once the core objectives above have been achieved, modify the solution design (section 3.6.2) to switch on the LED for a longer period if motion has been detected within 2 seconds of the LED having switched off. In this scenario, the LED is intended to stay on for longer rather than switch on and off in quick succession for motion detected at shorter intervals.

Currently, the LED on duration is determined by how long `motion_detected.Value` returns `True` (which is approximately 3 seconds). This will no longer be the case with the new design.

Investigate an approach for controlling the LED on duration in this program without using Python function `time.sleep()`.

3.7 Sound Level Meter

Use the Python template in section 3.1 to create a program that displays the current sound level as a horizontal bar graph on the LCD display. The bar graph should grow and shrink in response to the loudness of sounds detected by the sound sensor. When the sound level becomes very loud, the LED should briefly flash to indicate that a high sound level has been detected.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=X6d6g2VRHkNijyVd&t=90

This project reinforces loops, conditional statements, string manipulation, sensor input, LCD output, scaling numeric values, and simple user-interface design using a character display.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.7.1 Measuring the Sound Level

The All-in-One starter kit includes a sound sensor that can detect changes in sound level. The current sound level can be obtained using the `Value` property of variable `sound_sensor` (of type AnalogInput).

Property	Description	Range
`sound_sensor.Value`	Returns the current sound level measurement.	0 to 1023

Table 3-15 Using variable sound_sensor to read sound levels.

Typical values are show in the table below.

Sound Level	Example Reading
Quiet room	20–50
Normal conversation	100–300
Loud sounds	700–1023

Table 3-16 Typical sound values obtained from the sound sensor.

3.7.2 Printing Text to the LCD Display

The All-in-One starter kit includes a Liquid Crystal Display (LCD) with 16 columns and 2 rows for displaying text. The display is represented by the provided variable `display` (of type DisplayLcd) and uses the method `PrintAt` for displaying text.

Method	Description
`display.PrintAt(column, row, text)`	Displays text at a specified column and row position on the LCD. At most the first 16 characters of the text string will be displayed. Example: `display.PrintAt(0, 0, "Hello")`
`display.Clear()`	Clears the LCD.

Table 3-17 Displaying text on the LCD display using the PrintAt method.

3.7.3 Creating a Bar Graph Using Text

A simple bar graph can be created using repeated characters, for example using the “>” character. As the sound level increases, more “>” characters can be displayed. Because the LCD contains 16

columns, the completed string should always contain exactly 16 characters. Spaces should be added after the “>” characters so that the string always fills the display width.

A `for` loop can be used to repeatedly build the display string. For example:

```
for i in range(0,16):
```

This loop executes 16 times. During each iteration, the program can decide whether to add a “>” or a space character to give the effect of a bar graph that represents the sound volume.

3.7.4 Additional LED Control Methods

Previous projects used `led.On()` and `led.Off()` for controlling the state of the LED. The method `OnForDuration()` offers a convenient way to turn the LED on for specified time before automatically turning it off:

```
led.OnForDuration(1)    # LED remains on for 1 second

led.OnForDuration(0.5)  # LED remains on for half a second.
```

This method can be used to briefly flash the LED when a loud sound is detected.

3.7.5 Outline of Solution Design

Within the main loop of the Python template in section 3.1:

1. Read the current sound level from the sound sensor.
2. Scale the sound level into a value between 0 and 16.
3. Create an empty string that will be used as the sound level indicator.
4. Use a `for` loop to build a 16-character string:
   - Add a > character for positions within the measured sound level of step 2.
   - Add a space character for the remaining positions.
5. Display the completed string on the LCD.
6. If the sound level exceeds a chosen threshold, briefly flash the LED.

3.7.6 Summary of Expected Outcome

When the program runs, as the sound level changes, the bar graph should expand and contract to reflect the current sound intensity.

Sound Level	LCD Display
Quiet	Few > characters shown.
Moderate	Approximately half the display filled.
Loud	Most of the display filled.
Very loud	All 16 positions filled and LED flashes.

Table 3-18 Expected outcome of sound level meter program.

3.7.7 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution to show on the bar graph the highest sound level detected during the last 5 seconds.

3.8 Sonar-Based Movement Detector

Create a program that uses the ultrasonic sonar sensor to detect when an object moves closer to the sensor. When an object moves at least 2 centimetres closer than its previous measured position:

1. The LED should briefly switch on.
2. The relay should briefly activate.

This project simulates a simple proximity alarm system that responds when movement towards the sensor is detected.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=8gEX_o0tBfyqoyEO&t=104

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.8.1 Ultrasonic Distance Measurement

The All-in-One starter kit includes an ultrasonic sonar module that measures the distance between the sensor and an object. The sonar sensor works by:

1. Emitting an ultrasonic pulse.
2. Waiting for the echo to return.
3. Calculating the distance based on the echo travel time.

The ultrasonic sensor is represented by the provided variable `sonar` (of type Sonar) which has several methods and properties for performing measurements.

Method / Property	Description
`sonar.Ping()`	Sends an ultrasonic pulse.
`sonar.DistanceAcquired`	Indicates whether a measurement is available by returning `True` or `False`.
`sonar.GetDistance()`	Returns the measured distance in centimetres, or -1 if no measurement is available. The datatype is integer.

Table 3-19 Methods and properties for obtaining measurements using the sonar object.

Distance measurements are not available immediately after a ping is sent. The program must first request a measurement as long as a distance has not yet been acquired:

```
sonar.Ping()
```

The program can then check whether the measurement is ready:

```
if sonar.DistanceAcquired:
```

When the measurement becomes available:

```
distance = sonar.GetDistance()
```

returns the measured distance in centimetres.

3.8.2 Detecting Movement

A single distance reading cannot indicate whether an object is moving. To detect movement, the program must compare:

- The most recent distance measurement.
- The previous distance measurement.

To determine whether an object has moved closer:

```
previous_distance - current_distance
```

can be calculated. If the result is positive, the object is closer than before.

3.8.3 Outline of Solution Design

Starting with the Python template in section 3.1:

1. Create two variables to store consecutive sonar distance measurements.
2. Request a new distance measurement using `sonar.Ping()`.
3. If a measurement has been acquired.
   - Store the new distance reading.
4. Compare the current distance with the previous distance.
5. Determine whether the object has moved at least 2 cm closer to the sensor.
6. If this condition is met:
   - Activate the LED for one second.
   - Activate the relay for one second.
7. Pause for a fraction of a second.
8. Continue monitoring until the Escape key is pressed.

Refer to section 3.2 for switching on the LED, and section 3.11.2 for switching on the relay.

3.8.4 Summary of Expected Outcome

When the program is run, the LED and relay should only activate when an object has moved at least 2 cm closer than its previous measured position.

3.8.5 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution so that:

- LED remains on for the duration an object is within 10 cm of the sensor
- The slider is used to adjust the movement change distance, which is currently set to 2 cm. Refer to section 3.3.1 for reading the position of the slider.

3.9 Automatic Windscreen Wiper

Create a program that simulates an automatic vehicle windscreen wiper. The program should monitor a rain sensor and use a servo motor to perform a wiper action when moisture is detected. The speed of the wiper should increase as the amount of moisture increases.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=sfwPcORRuPuC3eJb&t=117

To make the program easier to maintain and reuse, the servo sweep action must be implemented in a function named `wipe_screen()`.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.9.1 Detecting the Moisture Level

A rain sensor provides a measurement that indicates the amount of moisture detected. The current sensor reading can be obtained using the `Value` property of variable `rain_sensor` (of type AnalogInput).

Property	Description	Range
`rain_sensor.Value`	Returns the current moisture reading.	`0 – 229` - No moisture detected `230 – 500` - Light moisture detected `501 –1023` - Heavy moisture detected

Table 3-20 Detecting the level of moisture using variable rain_sensor.

Higher values indicate more moisture. These ranges can be used to determine the wiper speed.

3.9.2 Using a Servo Motor as the Screen Wiper

The All-in-One starter kit includes a 180 degree servo motor that will be used to simulate the movement of a windscreen wiper. The servo is represented by the variable `servo` of type Servo. The servo motor can be positioned at any angle between 0 and 180 degrees at a configurable speed.

Method	Description
`servo.SetPosition(angle)`	Moves the servo to the specified angle, between 0 to 180 degrees.
`servo.SetSpeed(speed)`	Sets the movement speed of the servo between 1 to 10. 1 slowest, 10 fastest. A 180 degree sweep takes approximately 0.8 seconds at speed 6, and 0.4 seconds at speed 8.

Table 3-21 Setting the position and movement speed of the servo motor.

The servo motor takes a certain amount of time to move from one position to another, depending on the configured speed. Use the `time.sleep()` function to pause program execution for a suitable time to allow servo movement completion.

3.9.3 Create a Function to Simplify Program Design

A function allows a section of code to be written once and reused many times. For this project, create a function named: `wipe_screen(speed)`.

The function should:

1. Accept a speed parameter.
2. Adjust the servo speed according to the parameter supplied.
3. Move the servo from one side of the windscreen to the other (i.e. 0 degrees to 180 degrees).
4. Return the servo to its home position (0 degrees).

This function represents one complete wipe of the windscreen.

3.9.4 Outline of Solution Design

Starting with the Python template of section 3.1:

1. Create a function named `wipe_screen(speed)`.
2. Inside the function:
   - Set an appropriate servo speed.
   - Move the servo through a full sweep.
   - Return the servo to its starting (home) position.
   - Allow sufficient time for each movement to complete.
3. Initialise the servo in its home position before entering the main loop.
4. Inside the main loop:
   - Read the current rain sensor value.
   - Determine the amount of moisture present.
   - Perform a slow wipe (using `wipe_screen`) when light moisture is detected.
   - Perform a fast wipe (using `wipe_screen`) when heavy moisture is detected.
   - Perform no action when little or no moisture is detected.
5. Continue running until the Escape key is pressed.

3.9.5 Summary of Expected Outcome

When the program runs, the expected behaviour should be as described in the table below:

Rain Sensor Reading	Wiper Behaviour
`0 – 229`	No wiper movement.
`230 – 500`	Slow full sweep.
`501 –1023`	Fast full sweep.

Table 3-22 Expected outcome of automatic windsceen wiper program.

As additional moisture is detected, the wiper should operate more frequently at the higher speed because each wipe cycle takes less time to complete.

3.9.6 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution so that the wiper has a brief interval between wipe cycles when operating at a low moisture level, simulating intermittent wiping.

Investigate how to maintain and use a running average of the last five rain sensor readings to allow the windscreen wiper to respond only to sustained presence of moisture.

3.10 Using the IR Remote Control

Create a program that detects button presses from an Infra-Red (IR) remote control and displays the corresponding button name on the computer console. The program must use a lookup table (list or array) to translate received IR button codes into human-readable button names. If a button code is received that does not exist in the lookup table, the program should report that the code is unmapped.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=WSsUJ4dC6-e0YMTS&t=133

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.10.1 Reading IR Remote Control Button Presses

Infra-Red remote controls transmit a unique numerical code for each button pressed. The IR receiver built into the All-in-One starter kit can detect these codes and make them available to a Python program. However, the received code is only a number. To make the input more meaningful, the code should be translated into a button name.

The provided function `get_ir_code()` attempts to read a button press from the IR receiver. The function returns an integer value, where a positive value indicates a button press, and -1 indicates no button press.

3.10.2 Creating and Searching a Lookup Table

A lookup table is a collection of related values that can be searched. For this project, create a list containing tuples, in this case pairs of values:

```
(code, name)
```

where:

- `code` is the integer IR button code.
- `name` is the corresponding button name string.

This example list maps IR codes for the digit buttons to their string equivalent:

```
[(22,"0"), (12,"1"), (24,"2"), (94,"3"), (8, "4"), (28,"5"),
(90,"6"), (66,"7"), (82,"8"), (74,"9")]
```

A lookup table allows the program to convert a received IR code into a button name. A `for` loop can be used to examine each item in a list.

Example:

```
for (code, name) in lookup_table:
```

Each iteration retrieves one pair of values from the list. The program can then compare the received IR code with the stored code.

Example:

```
if ir_code == code:
```

If a match is found:

1. Store the button name.
2. Display the button name.
3. Use the `break` statement to immediately exit the loop

Before the `for` loop, initialise the variable holding the button name to an empty string, i.e. `""`. If the search completes and the variable still equals `""`, then no matching button code exists in the lookup table.

3.10.3 Outline of Solution Design

Begin with the Python template of section 3.1. Outside of the main loop, create a lookup table that maps IR button codes to button names (use the list example in the previous section 3.10.2).

Within the main loop:

1. Use `get_ir_code()` to obtain the received code.
2. Ignore negative values because they indicate that no button has been pressed.
3. When a positive code is received:
   - Search the lookup table.
   - Determine whether the code exists.
4. If the code is found:
   - Display the button code.
   - Display the corresponding button name.
5. If the code is not found:
   - Display a message indicating that the code is unmapped.
6. Continue monitoring for button presses until the Escape key is pressed.

3.10.4 Summary of Expected Outcome

When the program runs and buttons are pressed on the IR remote control, the output on the computer console should resemble the following:

```
IR code 22 = "0"
IR code 12 = "1"
IR code 94 = "3"
```

If a button is pressed whose code is not present in the lookup table:

```
45 not mapped.
```

is an example of what should be displayed.

3.10.5 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution to use a dictionary instead of a list, and use its `get()` method instead of using a `for` loop for obtaining the button name from its code.

3.11 Electronic PIN Code Entry System

Extend the solution to the IR remote control decoder program (section 3.10) to create a simple electronic PIN code entry system.

The system should allow a user to enter a four-digit PIN code using the IR remote control. As digits are entered, the LCD display should show # characters instead of the actual digits. When the correct PIN code is entered, an LED and relay should be activated for a short period of time.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=5bF8KX7Pvq2nae5F&t=155

This project demonstrates how electronic access control systems, alarm keypads, and door entry systems validate user credentials before granting access.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.11.1 Building on the Previous Project

In the previous project, a program was required that:

- Detected IR remote button presses.
- Translated IR button codes into button names (i.e. the text/symbols on the button face).
- Displayed the button names on the console.

In this project, the button names will be used to build a PIN code entered by the user. Instead of simply displaying the button names, the program must:

1. Collect entered digits.
2. Build a four-digit code.
3. Compare the entered code against a stored PIN.
4. Activate an output if the PIN is correct.

3.11.2 Activating a Relay Switch

The variable `relay` represents the relay switch included the All-in-One starter kit, and is controlled the same way as the LED used in earlier projects (refer to section 3.2).

Method	Description
`relay.On()`	Turns the relay on.
`relay.Off()`	Turns the relay off.
`relay.OnForDuration(seconds)`	Turns the relay on for a specified period before automatically turning it off.

Table 3-23 Methods to control state of the relay.

In a real security system, the relay could be connected to:

- An electronic door lock
- A magnetic lock
- A gate controller
- An alarm system

For this project, the relay simply simulates an access control output.

3.11.3 Processing Entered PIN Codes

The valid PIN code can be stored in a string variable.

Example:

```
pass_key = "1234"
```

This represents the valid four-digit code that the user must enter. A second string variable can be used to store the digits entered by the user.

Example:

```
code_entered = ""
```

Initially, no digits have been entered. The string can be built one character at a time using the + operator whenever a button has been pressed on the IR remote control.

The `len()` function returns the number of characters in a string and can be used to determine when the user has entered four digits.

```
len(code_entered)
```

Value of code_entered	Result Using len()
""	0
"3"	1
"1234"	4

Table 3-24 Determining the length of a string using the len() function.

When the same number of digits as the `pass_key` have been entered, `code_entered` is compared with `pass_key` using the == operator, which returns `True` only when both strings contain the same characters.

3.11.4 Hiding the Entered PIN

Real security systems do not normally display entered PIN digits. Instead, they display a masking character such as #. In Python, the multiplication operator * can be used to generate repeated characters that have the same length as the code entered.

Example:

```
"#" * 4
```

Produces:

```
####
```

3.11.5 Using Date and Time Values

The program should automatically clear entered PIN codes if the user stops entering digits after a certain period. The following statement stores the current time:

```
last_button_time = datetime.now()
```

The stored time can later be compared with the current time to determine how many seconds have passed since the last button press:

```
if (datetime.now() - last_button_time).total_seconds() > 2.0:
```

This allows the system to automatically reset after a period of inactivity.

3.11.6Outline of Solution Design

Starting with the solution from the previous IR Remote Control project (section 3.10):

1. Use the original IR code lookup table (section 3.10.2).
2. Create a variable named `pass_key` containing a four-digit PIN code.
3. Create a variable named `code_entered` to store the digits entered by the user.
4. When an IR remote button is pressed:
   - Append the button name to the entered code.
   - Display a matching number of # characters on the LCD.
   - Store the datetime of button press
5. When four digits have been entered:
   - Compare the entered code with the stored PIN.
   - If the PIN is correct:
     - Switch on the relay for a short period.
     - Switch on the LED for the same period.
   - Reset the entered code to empty string.
   - Clear the LCD display.
6. If no button has been pressed for two seconds and there is a partially entered code:
   - Reset the entered code to empty string.
   - Clear the LCD display.
7. Continue monitoring the IR remote until the Escape key is pressed.

Refer to section 3.2 for methods for controlling the LED (which are the same for the `relay`) and section 3.7.2 for printing text to the LCD display.

3.11.7Summary of Expected Outcome

As buttons are pressed on the IR remote control, a # symbol is shown for each key press on the LCD. If the entered code matches the stored PIN, the LED and relay are temporarily activated, and the LCD is cleared. If by the fourth button press an invalid code has been entered, or if there has been no key press for short period of time, the LCD is cleared.

3.11.8Additional Requirements

Many real security systems provide feedback when an incorrect code is entered. Once the core objectives above have been achieved, investigate how to implement a lockout period if there have been three successive incorrect PIN entries.

3.12 Scrolling Message of the Day

Create a program that reads a message from a text file and scrolls it across the LCD display of the All-in-One starter kit.

The text file contains a different message for each day of the week. The program must determine the current day, retrieve the appropriate message from the file, and display it as a scrolling message on the LCD.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=-bZ38nNim5YbHkLt&t=179

This project introduces file handling, reading data from text files, searching records, and string splitting and slicing.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.12.1 Reading the Message File

The message file for use by the program is named `message_file.txt` and contains one message per line.

Each line consists of two fields separated by the pipe (|) character. There is no header row in the file.

Field	Description
Field 1	Day of week number (0–6)
Field 2	Message text

Table 3-25 Pipe symbol delimited text message file, with one line for each day of the week.

Example file contents:

```
0|Welcome to Monday
1|Tuesday is project day
2|Remember to complete Project 12
3|Thursday coding challenge
4|Happy Friday
5|Enjoy your weekend
6|Prepare for the week ahead
```

A text file can be opened using the following Python code:

```
with open("message_file.txt") as file:
```

Lines can then be read one at a time.

Example:

```
line = file.readline()
```

Reading the file one line at a time allows the program to stop searching as soon as the required message is found.

The current day of the week can be determined using:

```
datetime.today().weekday()
```

The returned value ranges from 0 to 6, where 0 represents Monday, 1 represents Tuesday, and so on.

The program must search the file for the line containing the matching day number. Since each line in the file contains both the day number and the message text separated by a pipe symbol (|), the line must be split in to two parts, with the first part being the day of week number, and the second part being the message for the day. This can be achieved using:

```
day_of_week, message = line.split('|')
```

3.12.2 Creating a Scrolling Effect

The LCD can only display 16 characters at a time. Refer to section 3.7.2 for displaying text to the LCD.

To create a scrolling effect:

1. Add spaces to the beginning and end of the message string.
2. Extract and display a 16-character section of the string.
3. Increment the extract starting position one character at a time.

Example:

```
message = " " * 15 + message + " " * 16
```

A loop can be used to increment the starting position of the message section to extract.

Example:

```
for i in range(0, len(message)-15):
```

Each iteration can be used to display a different section of the message, creating the illusion of scrolling movement:

```
message[i:i+16]
```

This returns a substring that is exactly 16 characters long, starting from position `i`.

A short delay is required between display updates:

```
time.sleep(0.15)
```

This pauses the program for 150 milliseconds.

3.12.3 Outline of Solution Design

Starting with the Python template in section 3.1:

1. Delete the lines for the `while` loop.
2. Create a variable to store the message text.
3. Open the file `message_file.txt`.
   - Read the file one line at a time.
   - Extract day of week and message fields and assign to variables.
   - Determine whether the line corresponds to the current day of the week.
   - When a matching day is found:
     - Stop searching the file.
4. Add space padding to both ends of the message.
5. Use a loop to extract a scrolling 16-character window across the message.
   - Update the LCD display with extracted string.
   - Pause briefly between updates.

- Allow the program to terminate if the Escape key is pressed.

6. When LCD scrolling is complete, pause briefly, then clear the LCD.

3.12.4Summary of Expected Outcome

When the program is run, a message corresponding to the current day of the week is read from the text file, and scrolled one character at a time across the LCD.

3.12.5Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution such that a message is randomly chosen from the text file and scrolled on the LCD.

Given SQL database table `MessageOfTheDay`, with columns `DayOfWeek` (type `INT`) and `Message` (type VARCHAR), write the SQL to select

1. The message for a chosen day of the week.
2. All messages for weekend days.

DayOfWeek	Message
0	Welcome to Monday
1	Tuesday is project day
2	Remember to complete Project 12
3	Thursday coding challenge
4	Happy Friday
5	Enjoy your weekend
6	Prepare for the week ahead

Table 3-26 SQL database table MessageOfTheDay holding messages corresponding to a day of the week.

3.13 Simulate a Morse Code Message Transmitter

Create a program that converts a text message into Morse code and transmits it using the beeper on the All-in-One starter kit.

As each character is transmitted:

- The corresponding Morse code should be sounded using short and long beeps.
- The message should gradually appear on the LCD display.
- When the displayed text exceeds the width of the LCD, the display should scroll automatically.
- Spaces between words should be represented by a short pause.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=wpvTxCCRxggrfbGM&t=197

This project simulates a simple Morse code transmitter similar to those used in early radio communication systems.

Review the information in the following sub-sections to assist in achieving the project objectives.

3.13.1 Background Information

Morse code represents letters using combinations of dots and dashes when in written form, and short and longer beepers when expressed in audible form. For this project, a Python dictionary named `LETTER_TO_MORSE` has already been provided.

```
LETTER_TO_MORSE = {
    "a":".-", "b":"-...", "c":"-.-.", "d":"-..",
    "e":".", "f":"..-.", "g":"--.", "h":"....",
    "i":"..", "j":".---", "k":"-.-", "l":".-..",
    "m":"--", "n":"-.", "o":"---", "p":".--.",
    "q":"--.-", "r":".-.", "s":"...", "t":"-",
    "u":"..-", "v":"...-", "w":".--", "x":"-..-",
    "y":"-.--", "z":"--.."
}
```

The dictionary maps letters to their Morse code representation, and can be searched using the `get()` method to obtain the Morse code equivalent of an alphabetical letter:

```
morse_code = LETTER_TO_MORSE.get(letter, None)
```

In the above example if a letter is not found in the dictionary, `None` is returned.

3.13.2 How to Sound the Beeper

The All-in-One starter kit includes a beeper that can generate audible tones. The beeper is represented by the variable `beeper` of type Trigger. A beep can be generated using:

```
beeper.Pulse(duration)
```

where the parameter `duration` is the length of the beep in milliseconds. For this project, a suitable duration for the short beep is 60 milliseconds, and 110 milliseconds for the long beep.

3.13.3 Simply the Program Design

To simplify the design of the program, create a function named `sound_beeps(letter)`.

The function should:

1. Accept a single letter as a parameter.
2. Look up the Morse code representation in the dictionary.
3. Generate a sequence of beeps corresponding to the dots and dashes.
4. Ignore any letters that do not exist in the dictionary.

This function will be called once for every letter in the message.

3.13.4 Creating a Scrolling Display

As characters are transmitted, the LCD should display the message progress. A variable can be used to accumulate the characters already transmitted.

Example:

```
message_progress = message_progress + letter
```

To ensure the display never exceeds 16 characters:

```
message_progress = message_progress[-16:]
```

This keeps only the rightmost 16 characters and can be used to create a scrolling effect as more characters of the message are added.

A `for` loop can be used to process each character in a message.

Example:

```
for letter in message:
```

During each iteration:

1. Display the character on the LCD.
2. Determine whether it is a space or a letter.
3. Transmit the Morse code if letter, briefly pause if space.

3.13.5 Outline of Solution Design

Starting with the Python template in section 3.1, and the supplied `LETTER_TO_MORSE` dictionary (section 3.13.1):

1. Delete the lines for the `while` loop
2. Define a function named `sound_beeps(letter)`.
3. Inside the function:
   - Look up the Morse code for the supplied letter.
   - Return immediately if the letter is not found.
   - Generate a short beep for each dot.
   - Generate a long beep for each dash.
   - Add a short pause between Morse code symbols.
4. Create a text message variable with the message to transmit.
5. Create a variable to store the scrolling display text.
6. Process the message one character at a time.
7. Update the LCD display after each character.
8. Keep only the most recent 16 characters for display.
9. When a space character is encountered:

- Pause briefly.
- Do not transmit any Morse code.

10. For all other characters:
    - Call `sound_beeps()` to transmit the Morse code.
11. Allow the program to terminate if the Escape key is pressed.
12. Clear the LCD display when transmission is complete.

Refer to section 3.7.2 for printing text to the LCD.

3.13.6Summary of Expected Outcome

As the program runs:

1. The letters appear one by one on the LCD.
2. Each letter is converted to a Morse code string.
3. The beeper generates a sequence of short and long tones for each letter.
4. A brief pause occurs between words.

3.13.7Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution to:

1. Add support for numbers using Morse code.
2. Treat uppercase and lowercase letters the same.
3. Lengthen the pause between the Morse code beeps and accompany with LED flashes. Refer to section 3.2 for using the LED.

3.14 Automatic Cooling Fan

Create a program that simulates an automatic cooling fan.

The program should continuously monitor the temperature by reading values from a temperature sensor. If the temperature rises above a specified threshold and remains above that threshold for more than three seconds, the fan should automatically switch on.

The fan speed should increase as the temperature increases, providing a simple form of proportional temperature control.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=VSu1IIowCVkXTKFs&t=215

This project introduces timing, proportional control, range mapping, and controlling the speed of a DC motor.

Review the information in the following sub-sections to assist in achieving the project objectives.

3.14.1 Automatic Temperature Control

Many electrical devices contain cooling fans that prevent components from overheating.

Rather than simply switching the fan fully on or off, many systems adjust the fan speed according to the measured temperature.

For example:

Temperature	Fan Speed
Below threshold	Off
Slightly above threshold	Slow
Moderately above threshold	Medium
Well above threshold	Fast

Table 3-27 An example of how fan speeds are adjusted according to measured temperature.

A proportional fan speed produces quieter operation and reduces power consumption.

3.14.2 Reading the Temperature

The helper function `get_temperature()` is provided which returns the current temperature as a floating-point value in degrees Celsius.

Example:

```
temperature = get_temperature()
```

assigns the current temperature to the variable `temperature`. A floating point value is of a data type that represents real numbers with fractional parts.

Example values returned by the function `get_temperature()`:

- 24.6
- 27.3
- 31.8
- 35.1

3.14.3 Driving the Fan Motor

The cooling fan is driven by a DC motor represented by the variable `fan`, which is of type Motor. The following methods are available.

Method1	Description
`fan.SetAcceleration(seconds)`	Sets the time taken for the fan to accelerate from 0% to full speed.
`fan.Drive(speed)`	Sets the fan speed as a percentage of full speed.

Table 3-28 Methods available for controlling the fan motor.

Examples:

Statement	Result
`fan.Drive(0)`	Fan stopped
`fan.Drive(30)`	Fan runs at 30% speed
`fan.Drive(60)`	Fan runs at 60% speed
`fan.Drive(100)`	Fan runs at full speed

Table 3-29 Examples of driving the fan motor at different speeds.

Using acceleration produces a smoother change in fan speed.

Example:

```
fan.SetAcceleration(2)
```

This specifies that when the fan is driven, it should accelerate at a rate that will take approximately two seconds to run from stationary to full speed.

3.14.4 Delaying Fan Activation

A cooling fan should not switch on immediately if the temperature briefly rises above the threshold. Instead, the program should record the time when the threshold is first exceeded and only activate the fan if the temperature remains above the threshold for more than three seconds.

The current time can be obtained using:

```
start_time = datetime.now()
```

The elapsed time can then be calculated. Example:

```
(datetime.now() - start_time).total_seconds()
```

This returns the number of seconds that have passed since `start_time`.

3.14.5 Mapping Temperature to Fan Speed

The helper function `map_range()` is provided to convert a value from one range of values into another.

The function accepts five parameters:

```
map_range(value, source_min, source_max, target_min, target_max)
```

For this project:

- the `value` is the current temperature

- the source range is the temperature range, starting from the threshold
- the target range is the fan speed range.

For example, if the threshold is 27°C the temperature range 27°C to 37°C could be mapped to a fan speed range of 33% to 70%. The `value` parameter will be treated as clamped if outside of the source range.

The program should use `map_range()` to calculate the appropriate fan speed automatically given the current temperature.

3.14.6 Outline of Solution Design

Starting with the Python template in section 3.1:

1. Define suitable constants for the minimum and maximum fan speeds.
2. Define a temperature threshold above which cooling should begin.
3. Configure the fan acceleration.
4. Monitor the temperature using `get_temperature()`.
5. Detect when the temperature first rises above the threshold.
6. Record the time at which the threshold was exceeded.
7. Reset the timer if the temperature falls back below the threshold.
8. If the temperature remains above the threshold for more than three seconds:
   - Calculate an appropriate fan speed using `map_range()`.
   - Drive the fan at the calculated speed.
   - Display THold Exceeded on the second row of the LCD.
9. Otherwise:
   - Stop the fan.
   - Clear the second row of the LCD.
10. Display the current temperature on the first row of the LCD.
11. Pause briefly and continue running until the Escape key is pressed.

Refer to section 3.7.2 for using the LCD.

3.14.7 Summary of Expected Behaviour

Assume the temperature threshold is 27°C and temperature range 27°C to 37°C is mapped to a fan speed range of 33% to 70%. When the program is run, the behaviour should be as follows:

Temperature	Time Above Threshold	Fan
26.5°C	-	Off
27.8°C	2 seconds	Off
28.1°C	3.5 seconds	Running near lower limit of fan speed range
31.5°C	5 seconds	Running faster
36.0°C	8 seconds	Running near upper limit of fan speed range
26.8°C	-	Off

Table 3-30 Expected behaviour of automatic cooling fan program.

The fan should only operate after the threshold has been exceeded continuously for more than three seconds.

3.14.8 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution to:

1. Display the current fan speed on the LCD.
2. Display the elapsed time since the temperature exceeded the threshold on the second row of the LCD (instead of THold Exceeded).
3. Allow the slider control to adjust the temperature threshold.

3.15 Rubik's Cube Colour Counter

Create a program that uses a colour sensor to count how many times each Rubik's Cube colour is detected.

As the colour sensor is moved across the faces of a Rubik's Cube, the program should:

- Detect the current colour.
- Ignore colours that are not recognised.
- Detect when the colour changes from one cube square to another.
- Count the number of times each Rubik's Cube colour is encountered.
- Display the final totals when the program ends.

This project introduces data collection, counting, list searching, and simple statistical analysis.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=9RdgsPXcWmwMVn8R&t=247

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.15.1 Detecting Colours Using a Sensor

A low-cost colour sensor can be connected to the All-in-One starter kit. When an object is placed closely in front of the sensor, various light measurements are reported by it from which a determination can be made about the object's colour.

The variable `colour_sensor` represents the colour sensor and is of type ColourSensor. Before the sensor can be used, it must be enabled.

Command	Description
`colour_sensor.Enable()`	Enables the colour sensor.
`all_in_one_kit.WaitUntilSensorsReady(colour_sensor)`	Waits until the sensor is ready to provide readings.

Table 3-31 Enabling the colour sensor when it is connected to the All-in-One starter kit.

The program must wait until the sensor is ready before attempting to detect colours.

The helper function `detected_colour()` is provided to identify the colour visible to the sensor. The value returned is the colour name represented by a text string, e.g. `"orange"`. If no colour, or an unrecognised colour is detected, the function returns None, which in this case represents an invalid colour.

Why Detect Colour Changes? Suppose the colour sensor remains over the same cube square for several seconds. Without additional logic, the same colour could be counted hundreds of times. To avoid erroneous multiple counts, the program should only increment a colour count when the detected colour changes. To be able to do this, the program must store the previous valid colour and the current valid colour, compare the two, and only increment the count if the colours are different.

3.15.2 Using a List for Storing and Searching

The colour counts should be stored in a two-dimensional list.

Example:

```
colour_counts = [
```

```
    ["white", 0],
    ["red", 0],
    ["orange", 0],
    ["yellow", 0],
    ["green", 0],
    ["blue", 0]
]
```

A loop can be used to search for the detected colour.

Example:

```
for i in range(len(colour_counts)):
```

The colour name can then be compared with the detected colour.

Example:

```
if current_colour == colour_counts[i][0]:
```

If a match is found, the count can be increased using:

```
colour_counts[i][1] = colour_counts[i][1] + 1
```

At this point the `for` loop can be exited.

3.15.3 Outline of Solution Design

Starting with the Python template in section 3.1:

1. Create a list variable containing all six Rubik's Cube colours and a count for each colour.
2. Enable the colour sensor.
3. Wait until the sensor is ready.
4. Create a variable to store the previously detected colour.
5. Use `detected_colour()` to determine the current colour.
6. Ignore readings that return None.
7. Detect when the current colour differs from the previous colour.
8. When a new colour is detected:
   - Display the colour name.
   - Find the colour in the list.
   - Increment its count.
   - Assign the new colour to the variable holding the previous colour
9. Continue monitoring until the Escape key is pressed.
10. After the program finishes:
    - Display all colours and their counts.

3.15.4 Summary of Expected Outcome

When the program is run, as the sensor is moved over the faces of the Rubik's cube, each time a colour change is detected, the count of the current colour is incremented. When the Escape key is pressed, the colour counts are displayed on the computer console, e.g.

```
Colour counts
------------------
white : 0
red : 3
orange : 0
```

```
yellow : 1
green : 4
blue : 2
```

3.15.5 Additional Requirements

Once the core objectives above have been achieved, modify a copy of the solution to implement or use a sorting algorithm to display the colour counts in decreasing order. For any counts that are the same, the colours should be in alphabetical order.

3.16 Digital Spirit Level

Create a program that uses the Motion Processing Unit (MPU) sensor to measure the angle of incline of the All-in-One starter kit and display the result on the LCD.

The program should:

- Calibrate the sensor when the program starts.
- Measure the current incline angle.
- Apply a calibration offset to improve accuracy.
- Display the measured angle on the LCD.
- Display a graphical level indicator that behaves like a spirit level.

🎓 View online demonstration: https://youtu.be/7gXj41NGkIg?si=MKS3CDIT_GwSx6eM&t=284

This project introduces sensor calibration, averaging measurements, numerical processing, and graphical representation of sensor data.

Review the background information in the following sub-sections to assist in achieving the project objectives.

3.16.1 The Purpose of a Spirit Level

What is a Spirit Level? A spirit level is a tool used to determine whether a surface is horizontal. A traditional spirit level contains a liquid-filled tube with a bubble that moves as the tool is tilted.

In this project, the LCD display will simulate a spirit level by displaying a bar that changes position according to the measured angle.

3.16.2 Reading from the MPU Sensor

The All-in-One starter kit includes an MPU6050 motion sensor that can be used to determine the angle of incline. Before the sensor can be used, it must be enabled.

Command	Description
`mpu_sensor.Enable()`	Enables the MPU sensor.
`all_in_one_kit.WaitUntilSensorsReady(mpu_sensor)`	Waits until the sensor is ready.

Table 3-32 Enabling the MPU sensor which is part of the All-in-One starter kit.

The program must wait until the sensor is ready before attempting to read measurements.

For reading the incline angle, a helper function is provided: `incline_angle()`. This function returns the current incline angle of the sensor.

Example:

```
angle = incline_angle()
```

The returned value is a floating-point number.

- The value 0.0 indicates the sensor is on a perfectly level surface.
- A positive value indicates a tilt in one direction.
- A negative value indicates a tilt in the opposite direction.
- A larger value indicates a steeper incline.

Real-world sensors rarely produce perfectly stable readings. Even when the sensor is completely still, successive measurements may vary slightly. These small variations are known as sensor noise.

3.16.3 Why Calibration is Required

The sensor may not report exactly zero degrees when placed on a perfectly flat surface, a cause of which may be imprecise sensor orientation at time of installation. Sometimes ambient temperatures can cause values to seem erroneous. To compensate for this, the program should perform a simple calibration.

One way to calibrate the sensor is to take several readings and calculate their average:

```
angle_offset = 0
```

Accumulate several readings in a loop, remembering to include a small pause between each reading:

```
angle_offset += incline_angle()
```

After collecting ten readings, divide by 10:

```
angle_offset /= 10
```

The resulting value represents the average sensor offset. This value can then be subtracted from future readings.

Example:

```
corrected_angle = incline_angle() - angle_offset
```

This makes the displayed angle more accurate.

3.16.4 Displaying the Angle

The measured angle should be displayed centred on the first row of the LCD (refer to section 3.7.2 for using the LCD).

Example:

```
display.PrintAt(4, 0, f"{angle:.1f}")
```

Example output:

```
2.3
```

The format specification:

```
{angle:.1f}
```

rounds the value to one decimal place.

3.16.5 Creating a Level Indicator

The second row of the LCD should display a simple graphical indicator. The indicator consists of dash (-) characters.

When the angle is positive, dash characters should appear to the left of the display centre:

```
----|
```

When the angle is negative, dash characters should appear to the right of the display centre:

```
|----
```

When the angle is close to level, the row should show a blank line.

The number of dash characters should increase as the angle increases. This creates a simple visual representation of the current tilt.

In Python, a repeated character string can be created using multiplication.

Example:

```
"-" * 5
```

produces:

```
-----
```

The functions `rjust()` and `ljust()` can be used to provide padding on either side of the level indicator.

3.16.6 Outline of Solution Design

Starting with the Python template in section 3.1:

1. Enable the MPU sensor after connecting the All-in-One starter kit.
2. Wait until the MPU sensor is ready.
3. Perform a calibration routine by:
   - Taking ten incline measurements, pausing between each reading.
   - Calculating their average.
   - Storing the result as an offset value.
4. In the main loop, measure the current incline angle.
5. Subtract the offset from each reading.
6. Display the corrected angle on the first row of the LCD.
7. Create and display a level indicator bar on the second row.
8. Pause briefly.
9. Continue monitoring until the Escape key is pressed.

3.16.7 Summary of Expected Outcome

Before running the program, the All-in-One starter kit must be placed on a level surface. When the program is run, it will take a moment to calibrate the sensor. A value close to 0.0 should be displayed on the first row of the LCD. The second row of the LCD should be blank. As the kit is titled in one direction, the tilt angle is displayed on the first row, and level indicator displayed on one side of the second row. When the kit is titled in the opposite direction, a negated angle and accompanying level indicator on the opposite side is shown.

3.16.8 Reducing Sensor Noise and Spikes

While the program is running, the observer may notice that the displayed angle occasionally jumps suddenly, even when the sensor is stationary. These spikes occur because real sensors are affected by electrical noise and measurement inaccuracies.

Once the core objectives above have been achieved, research and separately adapt both of the following techniques to reduce these unwanted spikes.

3.16.8.1 Exponential Smoothing

Combine the previous filtered value with the latest reading.

Example:

```
filtered_angle = filtered_angle * 0.8 + new_angle * 0.2
```

This creates a gradually responsive display without requiring a long history of stored values.

3.16.8.2 Median Filter

Keep a history of 5 most recent readings and sort them. Use the middle value as the noise-filtered value. This technique is known as median filtering and is used for filtering out sudden spikes in sensor readings.

4 Where Do You Want to go Next?

Congratulations!

You have reached the end of this course and completed a wide range of practical Python programming projects that interact with an assortment sensors and actuators. You have done so using the Robo-Tx robotics framework in conjunction with the All-in-One starter kit for Arduino.

Along the way, you have developed far more than just an understanding of Python syntax. You have learnt how software can interact with the physical world by reading information from sensors, processing data, making decisions, and controlling electronic devices.

Each project has introduced new programming techniques while reinforcing concepts from previous assignments. By gradually building your knowledge, you have developed the skills required to design larger and more sophisticated Python programs.

More importantly, you have experienced the complete development process—planning a solution, writing code, testing it with real hardware, debugging problems, and refining your implementation. These are the same skills used by professional software developers and robotics engineers every day.

Programming Concepts That Were Covered

Throughout this course you have re-enforced knowledge and experience in many of the core programming concepts used in GCSE and A-Level Computer Science, including:

- Variables and data types
- Mathematical calculations
- Selection using if, elif and else
- Repetition using for and while loops
- Functions and reusable code
- String manipulation
- Lists and dictionaries
- Searching and counting algorithms
- Reading data from text files
- Event-driven programming
- Sensor calibration
- Working with noisy sensor data
- Debugging and testing techniques
- Designing clear, maintainable Python programs

More importantly, you have learnt how these programming concepts are applied in real engineering applications.

Looking Beyond This Course

The Robo-Tx framework has been designed for much more than programming exercises.

It provides a simple yet powerful bridge between Python and Arduino-based electronics, making it possible to develop increasingly capable robotic systems without first needing to learn C++ programming for Arduino.

The same programming techniques you have used throughout this course can be applied to many exciting robotics projects, including:

- Line-following robots
- Obstacle avoidance robots
- Robotic arms
- Solar tracking systems
- Environmental monitoring systems
- Home automation projects
- Security and surveillance systems
- Smart sensor networks
- Internet-connected devices

As you continue your programming journey, you will discover that these larger projects are simply combinations of the techniques you have already learnt.

Build Your Own Robot

Although this course has provided remote access to robotics learning hardware during your practical sessions, you may now wish to build your own robotics platform.

Arduino-compatible electronics, motors, sensors and robot chassis kits are inexpensive and widely available, making them ideal for experimenting at home.

Using the Robo-Tx framework, you can continue programming these devices in Python while applying everything you have learnt throughout this course. A growing collection of demonstration projects built using the Robo-Tx framework is available in the *Python for Robotics Simplified* GitHub repository:

https://github.com/kashif-baig/Python-for-Robotics-Simplified

There you will find links for guidance on the software, firmware and tools needed to begin developing your own projects. Remember that building robots, even small ones, can be dangerous, so be sure to have the supervision of an experienced responsible adult.

The Journey Has Only Just Begun

You have taken an important step from writing simple Python programs to creating software that interacts with the physical world.

The skills you have developed form an excellent foundation for future study in:

- Computer Science
- Software Engineering
- Robotics
- Artificial Intelligence
- Embedded Systems
- Mechatronics
- Electronics
- Automation and Control Systems

Whether your next project is a line-following robot, an autonomous vehicle, a robotic arm, or an idea entirely of your own, you now have the programming knowledge and practical experience to begin turning those ideas into reality.

Keep experimenting. Keep learning. Keep building. The most exciting project you will create is the one you decide to invent next.