Hello! After the last worksheet, you should now have a fully assembled robot. In this worksheet, I am going to walk you through the creation of a test program so you can make sure everything works as intended. Before beginning, connect to your Pi on a PC using VNC Viewer.

Thonny IDE

Thonny is the name of the IDE (integrated development environment) that comes preinstalled with Raspberry Pi OS. You can see how to access it in Figure 1.

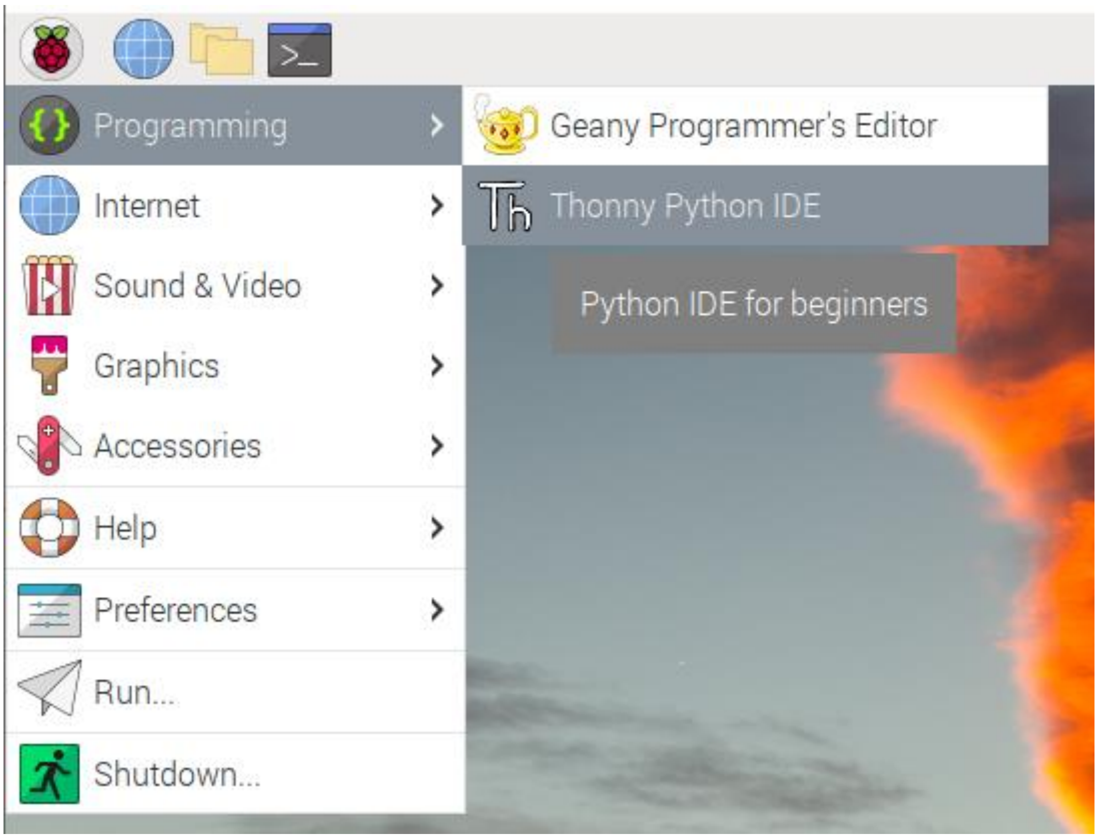
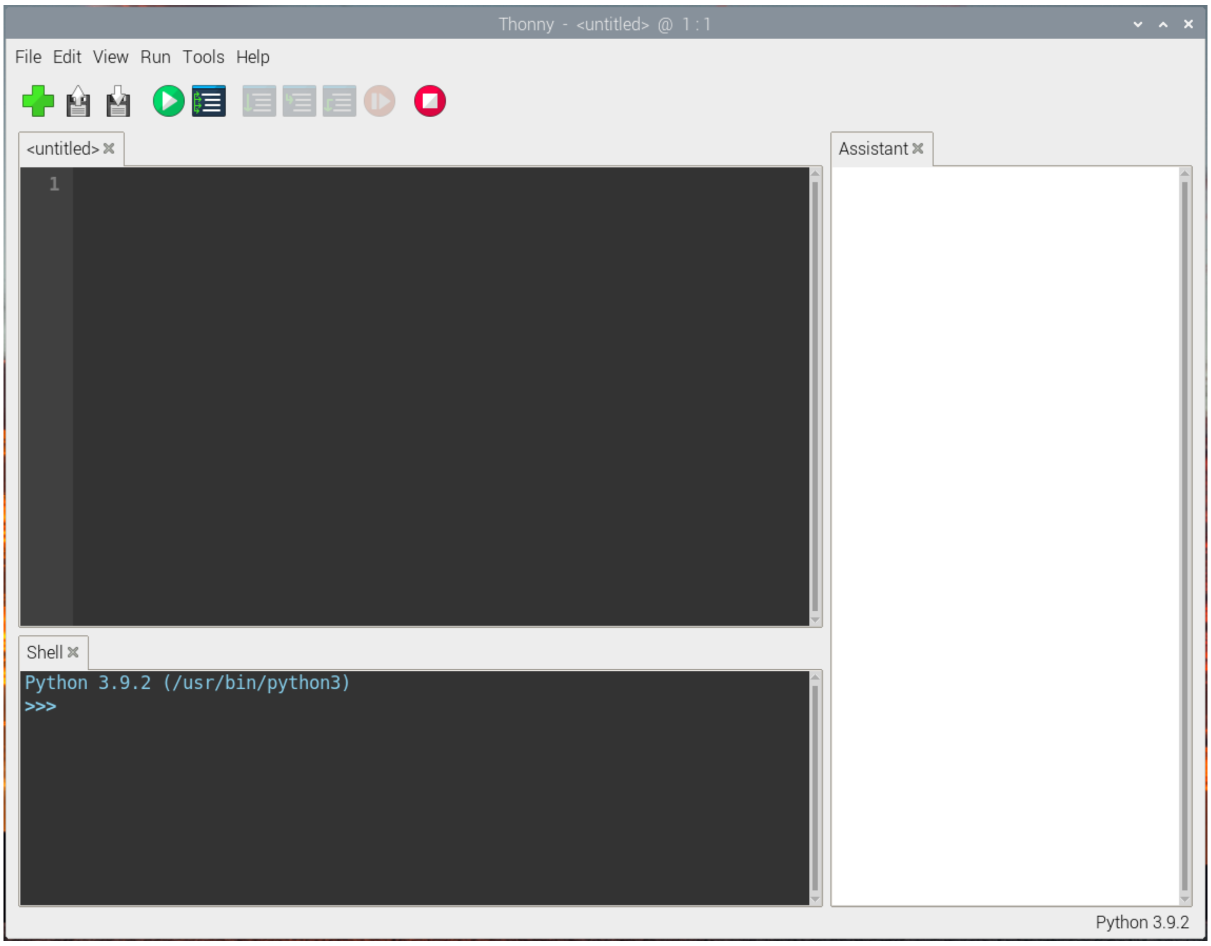


Figure 1 Accessing Thonny Python IDE

Once opened, you will be able to see an empty section to write your code that is currently untitled. Below there will be a shell which will show your program output and allow for input. To the right there will be an Assistant tab that will offer hints about how you can fix your program if you run into any exceptions. This is all shown in Figure 2. So you are able to see more code on screen in my examples, I have closed the Assistant tab but I would recommend that you keep yours open in case you need any help.



```
Python 3.9.2 (/usr/bin/python3)
>>>
```

Figure 2 New session of Thonny IDE

Now that your IDE is open, save the file and give it a name. I recommend “robot_test.py”. Adding “.py” to the end of a file tells the computer that the file contains Python code. After this, it’s time to get coding!

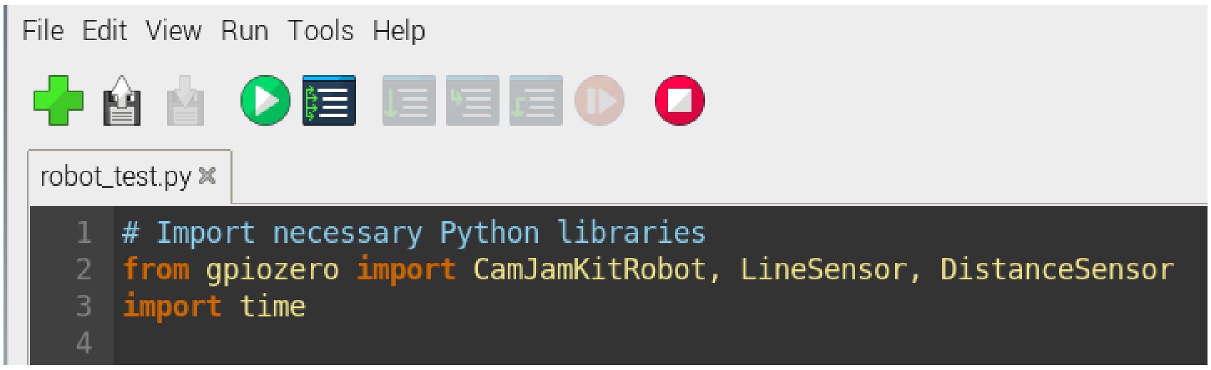
Code Libraries

To function, our robot is going to require the help of certain Python libraries. A code library is just a collection of prewritten code that you can import into your program and use. There are many libraries, some official and some not, that are available. Part of being a good programmer is knowing how to apply existing code to solve your own problems. This saves time writing functionality yourself and makes your code more readable. This is how the professionals do it!

For our test program, we are going to be using two libraries: GPIO Zero and Time.

Importing Libraries

The two libraries you will be using come preinstalled with the Raspberry Pi so you do not need to worry about where you can find them.



```python
# Import necessary Python libraries
from gpiozero import CamJamKitRobot, LineSensor, DistanceSensor
import time
```

Figure 3 Importing Python libraries

As the GPIO Zero library is quite large, we are only importing what we need from it as shown in Figure 3. Importing "CamJamKitRobot" will allow us to control the robot's movements. "LineSensor" and "DistanceSensor" will allow us to use the sensors you have installed on your Pi. GPIO stands for "General Purpose Input/Output" and will connect our code to the pins on your Raspberry Pi where you attached your motor controller.

Time is a smaller library. Whilst we are only going to be using one function, "sleep", I have imported the whole library to show you how this is still legal Python code.

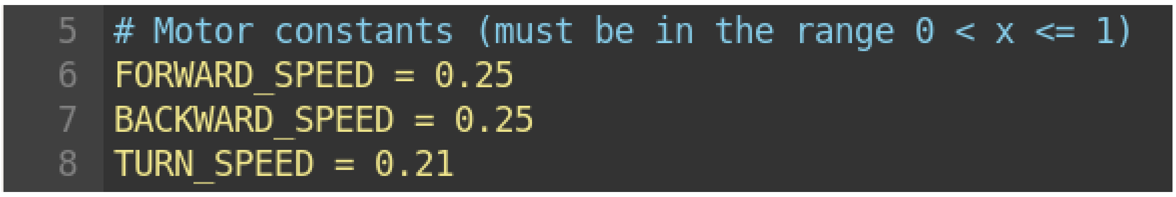
If you want to read more about these libraries, you can follow the links below to their documentation:

- GPIO Zero: gpiozero — GPIO Zero 1.6.2 Documentation (Nuttall & Jones, n.d.)
- Time: time — Time access and conversions — Python 3.10.4 documentation (Python, n.d.)

Next, we are going to declare our programs constants.

<u>Constants</u>

Within programming, it is bad practice to have a lot of repeated values that represent the same thing in your code. This is because anyone reading it will struggle to make any changes without breaking the code as making one change will lead to having to make many other changes. It can be easy to miss things when you have a lot of lines of code. We solve this problem using constants, like those declared in Figure 4.

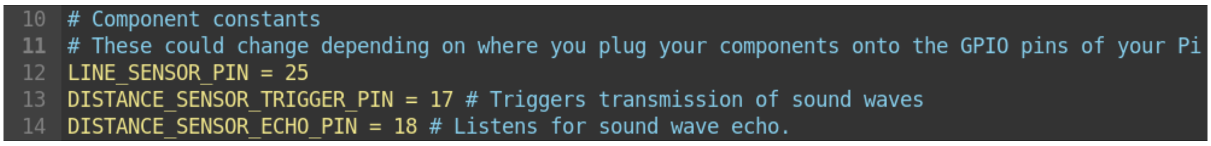


```python
# Motor constants (must be in the range 0 < x <= 1)
FORWARD_SPEED = 0.25
BACKWARD_SPEED = 0.25
TURN_SPEED = 0.21
```

Figure 4 Declaring Constants

I can use these constants as many times as I like within my code and if I want to make changes, I can do so where they are declared. Constants could also have the same value but represent different things. In this test program, I want "FORWARD_SPEED" to have the same value as "REVERSE_SPEED" however this may not always be the case. If I had just written "0.25" everywhere, it would be hard to know which speed I was referring to, making the code harder to change.

It is convention in Python to capitalise your constants and replace spaces with underscores "_". Python does not officially support constants like a lot of other programming languages do, meaning it is important you clearly label them and do not change their values later in the code (Python Tutorial, n.d.). Whilst this is legal Python as it would not return any errors, you may be adding bugs to your program without realising. You ***MUST*** only change constants where you have declared them. You are allowed to use constants in calculations, just remember not to write the answer back to the constant itself.



```python
# Component constants
# These could change depending on where you plug your components onto the GPIO pins of your Pi
LINE_SENSOR_PIN = 25
DISTANCE_SENSOR_TRIGGER_PIN = 17 # Triggers transmission of sound waves
DISTANCE_SENSOR_ECHO_PIN = 18 # Listens for sound wave echo.
```

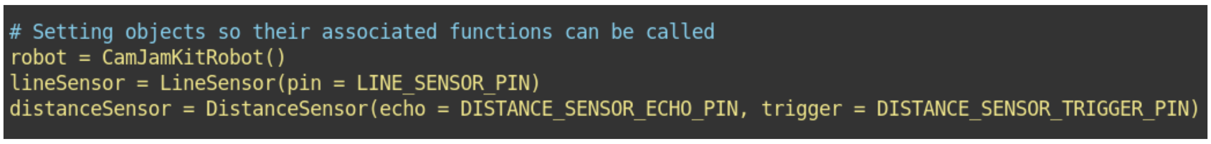
Figure 5 Component Constants

Figure 5 shows how we can set the constants for our components. You'll remember that when building your robot that you had to plug the jumper wires into specific slots. Take a look at the motor controller to double check you have your wires in the right slots, then declare them as constants.

Now we are going to see how the constants declared in Figure 5 are used to create objects.

Objects

Objects in code are instances of a class. A class is a collection of variables and functions. I like to think of them like sweets. You may buy a bag of sweets that has many different kinds in them. Whilst they may have different colours or flavours, they are still all sweets. In programming, the class would be called "sweet" and the object would be created by setting certain values at creation, like "colour" or "flavour". You could then name the object something like "colaBottle" or "gummy_bear". Using objects allows programmers to interact with different instances of the same class separately.



```python
# Setting objects so their associated functions can be called
robot = CamJamKitRobot()
lineSensor = LineSensor(pin = LINE_SENSOR_PIN)
distanceSensor = DistanceSensor(echo = DISTANCE_SENSOR_ECHO_PIN, trigger = DISTANCE_SENSOR_TRIGGER_PIN)
```

Figure 6 Creating Objects

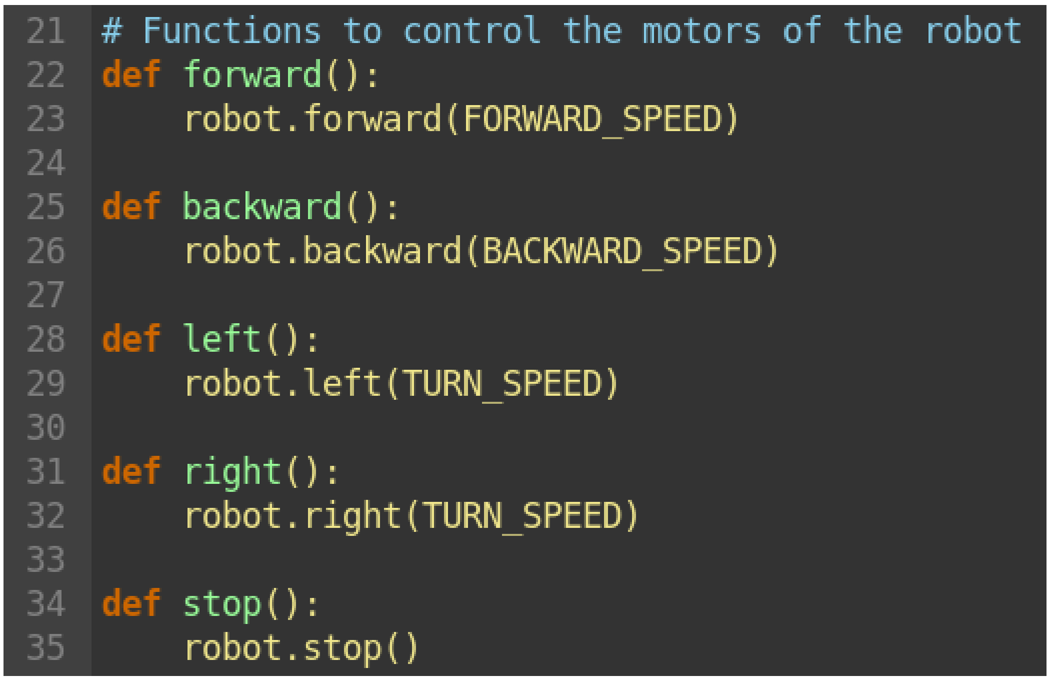
In Figure 6, you can see how I have created "lineSensor" and "distanceSensor" objects from their classes. You could name these whatever you like but it is always best to have descriptive names. In the case of our robot, we only have one distance sensor and one line sensor, making them difficult to confuse. However, if we had multiple sensors, it would be important to be able to differentiate them and interact with them separately. That is why we use objects.

If you want to read more about the classes we are making the objects from, see the following links:

- Distance Sensor: 13. API - Input Devices — GPIO Zero 1.6.2 Documentation (Nuttall & Jones, n.d.)
- Line Sensor: 13. API - Input Devices — GPIO Zero 1.6.2 Documentation (Nuttall & Jones, n.d.)
- CamJamKitRobot: 16. API - Boards and Accessories — GPIO Zero 1.6.2 Documentation (Nuttall & Jones, n.d.)

Creating Functions

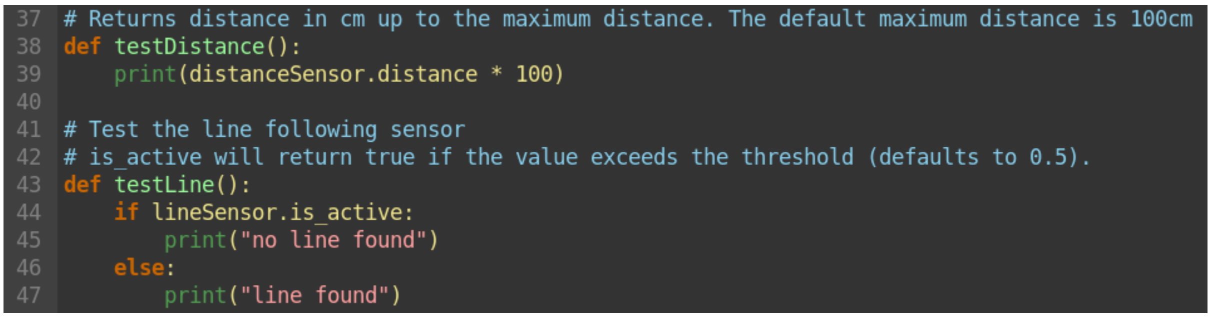
Everything is ready for us to start using the constants and objects we have declared in our code. Firstly, we are going to create some functions to control the motors.



```
# Functions to control the motors of the robot
def forward():
    robot.forward(FORWARD_SPEED)

def backward():
    robot.backward(BACKWARD_SPEED)

def left():
    robot.left(TURN_SPEED)

def right():
    robot.right(TURN_SPEED)

def stop():
    robot.stop()
```

Figure 7 Create Motor Functions

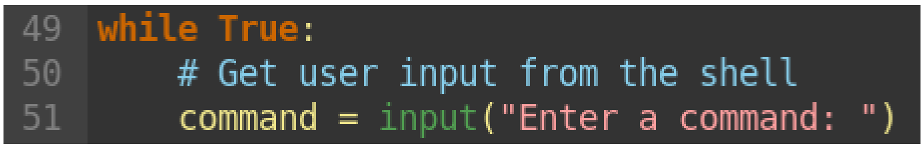
If you have read the documentation for “CamJamKitRobot”, you may have noticed that it is an extension of the “Robot” class. This means that we can use the functions available in both classes for our “robot” object. Follow this link to see all the available functions for the “Robot” class: 16. API - Boards and Accessories — GPIO Zero 1.6.2 Documentation (Nuttall & Jones, n.d.).



```python
# Returns distance in cm up to the maximum distance. The default maximum distance is 100cm
def testDistance():
    print(distanceSensor.distance * 100)

# Test the line following sensor
# is_active will return true if the value exceeds the threshold (defaults to 0.5).
def testLine():
    if lineSensor.is_active:
        print("no line found")
    else:
        print("line found")
```

Figure 8 Sensor Test Functions

The functions in Figure 7 will make your robot move when called, making it easy to see if your robot is working as intended. However, sensors require you to output something to the shell to see if they are functioning correctly. That is why we create functions to output test results to the Shell in the IDE (Figure 8).



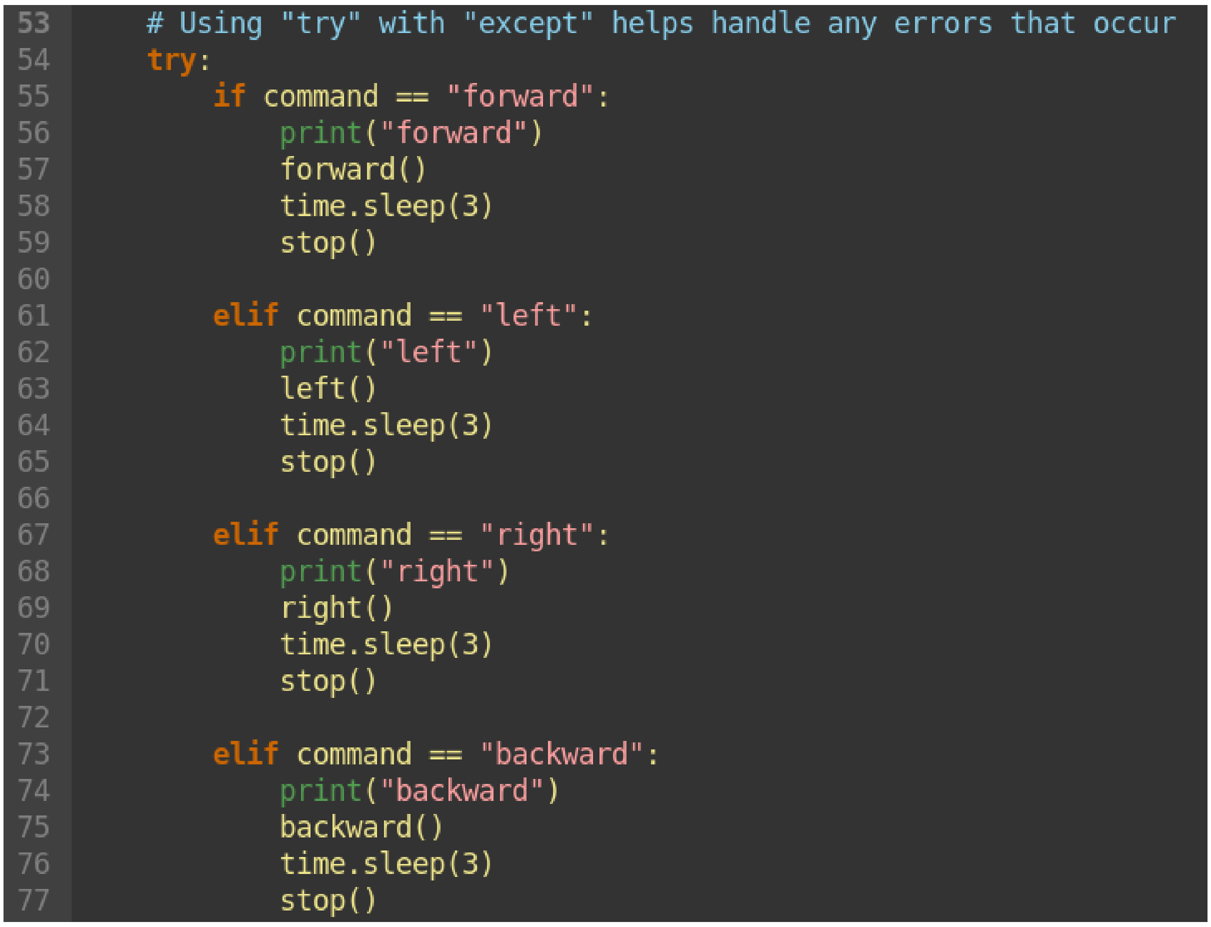
```python
while True:
    # Get user input from the shell
    command = input("Enter a command: ")
```

Figure 9 Getting user input

Finally, we are going to program code that allows us to take user input and call the functions we have created so we can test our robot.

<u>User Input & Calling Functions</u>

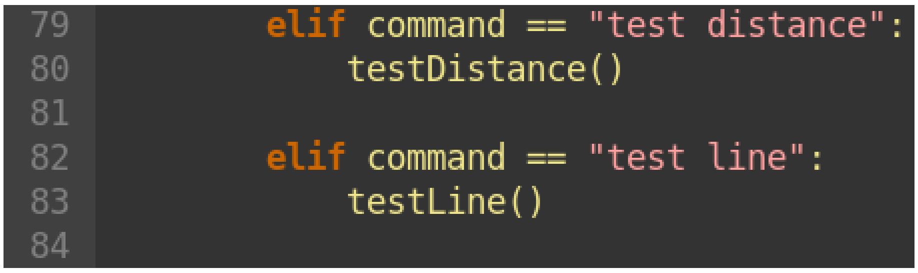
Figure 9 shows how we can use a “while True” loop to keep asking for the user to enter a command. This loop is infinite so if a user keeps typing commands, the program will keep taking this input, do something with it, and ask for another command. Any type of user input will often be contained in a loop like this in case the user needs to enter many commands.



```python
    # Using "try" with "except" helps handle any errors that occur
    try:
        if command == "forward":
            print("forward")
            forward()
            time.sleep(3)
            stop()

        elif command == "left":
            print("left")
            left()
            time.sleep(3)
            stop()

        elif command == "right":
            print("right")
            right()
            time.sleep(3)
            stop()

        elif command == "backward":
            print("backward")
            backward()
            time.sleep(3)
            stop()
```

Figure 10 Calling Motor Functions from user input

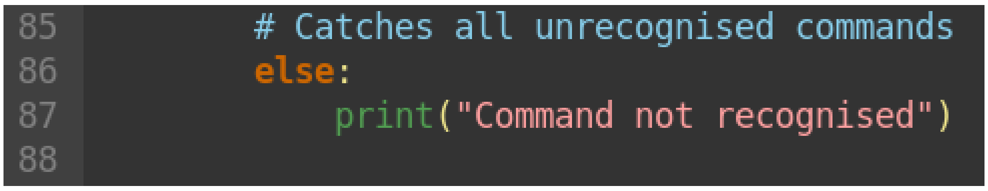
Programming what to do with the user input from the Shell is important, without user input, how will the user test the parts of the robot they want to? That is why we use a selection of “if” and “elif” (else if) statements (Figure 10) to see what value the user has passed to the “command” variable created in Figure 9. The “sleep” function from “time” is being used to ensure that each motor command stays active for three seconds before being stopped. We wrap the code checking the user input in a “try” block so we can pair it with an “except” block later in the code to stop the program.



```python
        elif command == "test distance":
            testDistance()

        elif command == "test line":
            testLine()
```

Figure 11 Calling Sensor Functions

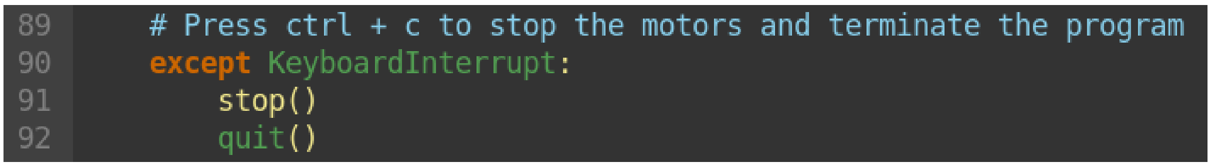
Figure 11 shows how the sensor functions are called in the same way as the motor functions, by comparing the text typed by the user in the Shell to what is on the right-hand side of “==”.



```python
        # Catches all unrecognised commands
        else:
            print("Command not recognised")

```

Figure 12 Command not recognised

To catch any other input from the Shell that doesn't match our "if" and "elif" statements, we can use an "else" (Figure 12). In our case, this will print to the Shell the text "Command not recognised". As all these checks for user input are in a loop, after each one is complete, the program will return to the start of the loop and print "Enter a command:" so the user can repeat the process again.



```python
    # Press ctrl + c to stop the motors and terminate the program
    except KeyboardInterrupt:
        stop()
        quit()
```

Figure 13 Terminate program through user input

To end the program, we have the "except" part of our "try". Typing "KeyboardInterrupt" after "except" means that if the user presses ctrl + c on their keyboard, the rest of the code in Figure 13 will run. This will stop the robot motors if they are still moving and then quit the program. You must stop the motors at the end of the program as just quitting whilst the motors are still running will result in them being stuck on. This happens because even if the code quits, the computer doesn't know to stop sending signals to move the motors unless you tell it to. If the motors remain on after quitting this way, run the program again and they should stop.

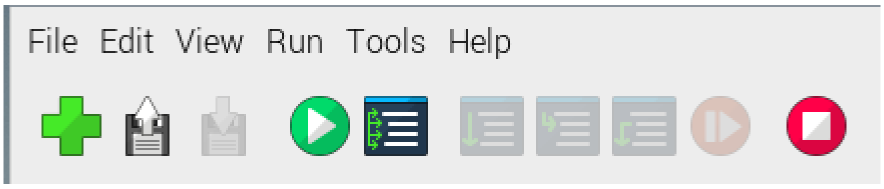
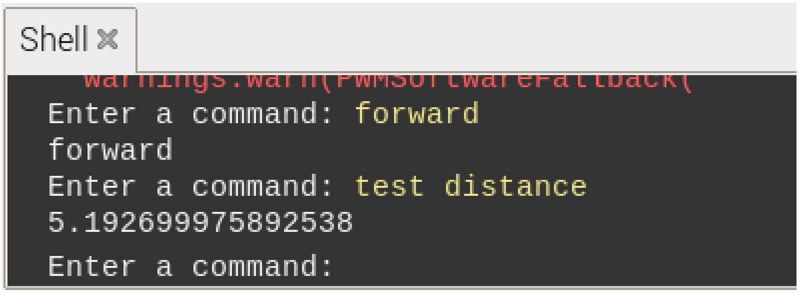


Figure 14 Thonny Tool Bar

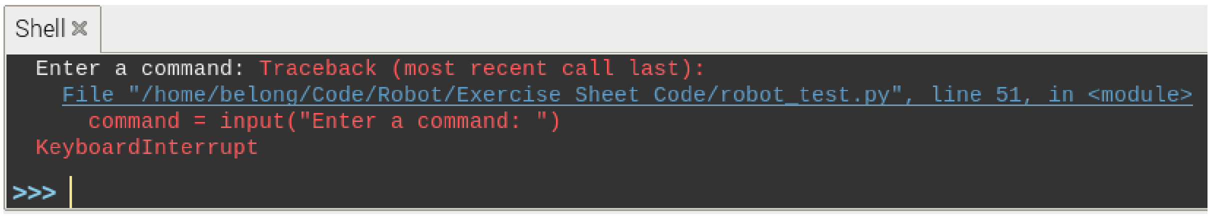
When you are ready to run your code, you can click the green play button in the IDE's toolbar (Figure 14). If your program takes user input, you will be able to type into the Shell. You can also stop your code by clicking the red stop button however if you have been using your motors, I wouldn't recommend this.



```
    warnings.warn(PWMSoftwareFallback(
  Enter a command: forward
  forward
  Enter a command: test distance
  5.192699975892538
  Enter a command:
```

Figure 15 Program Output Example

Figure 15 gives an example of program output based on user input. If the user decides to quit the program using ctrl + c, the Shell will output the contents of Figure 16. To learn more about Python Exceptions such as "KeyboardInterrupt", see the following link: [Python Try Except (w3schools.com)](https://www.w3schools.com) (W3Schools, n.d.)



```
  Enter a command: Traceback (most recent call last):
    File "/home/belong/Code/Robot/Exercise Sheet Code/robot_test.py", line 51, in <module>
      command = input("Enter a command: ")
  KeyboardInterrupt
>>> 
```

Figure 16 KeyboardInterrupt Exception

<u>Testing your robot</u>

When using the program you just created, if you notice your motors are running in the wrong direction, this can be easily fixed. Disconnect your robot from its power supplies and swap the wires that come from the motors and lead into the motor controller around in their two slots. This should fix your motor issue. You may need to do this to both motors. Unfortunately, there is no way to check your motors are wired correctly without testing them yourself.

If you are having problems with your sensors, disconnect your robot from its power supplies and check your wiring is correct on both the breadboard and motor controller. The line sensor is quite short range so make sure it is as close to the ground as possible.

<u>Challenge</u>

- Try printing the available commands to the Shell so a user knows without looking at your code what commands they can use.

END OF WORKSHEET 2

Reference List

Nuttall, B. & Jones, D., n.d. *gpiozero.* [Online]
Available at: https://gpiozero.readthedocs.io/en/stable/#
[Accessed 20 04 2022].

Python Tutorial, n.d. *Python Constants.* [Online]
Available at: https://www.pythontutorial.net/python-basics/python-constants/
[Accessed 20 04 2022].

Python, n.d. *time — Time access and conversions.* [Online]
Available at: https://docs.python.org/3/library/time.html
[Accessed 20 04 2022].

W3Schools, n.d. *Python Try Except.* [Online]
Available at: https://www.w3schools.com/python/python_try_except.asp
[Accessed 20 04 2022].