**Programming in Python with the Pi2Go Simulator: Part 1**

**Chapter 1: Getting Started**



**AIM:** After completing this chapter you should be able to start the virtual Pi2Go simulator and change the simulation environment for the virtual robot by adding and removing boxes, lines and light sources.

**Before you start:** You need to have installed our simulator somewhere. It should be in a folder called pirover\_simulator. You will find the simulator in this folder. It is called pysim.py (Windows and Linux) or pysimosx.py (Macs).

To start you need to run the simulator. You can run it either from the *command line* or from IDLE.

**From the Command Line (Windows and Linux):** Type python3 pysim.py then press return.

**From the Command Line (Macs):** Type python3 pysimosx.py then press return.

**From IDLE:** Start IDLE. A window will open and you should see something like:

Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)

[Clang 6.0 (clang-600.0.57)] on darwin

Type "help", "copyright", "credits" or "license()" for more information.

>>>

Open pysim.py (pysimosx.py on Macs) from the File Menu.

This file will open in a new IDLE window, including a new menu item **Run**.

From **Run** select **Run Module.**

You should see:

**A picture containing screenshot

Description automatically generated**

From here you can start a simulation for either the Initio robot (the top image) or the Pi2Go robot (the bottom image) in a number of settings (the “World Files” on the left).

Select the Pi2Go robot and the default\_world.xml.

Then click on *Start Simulation*

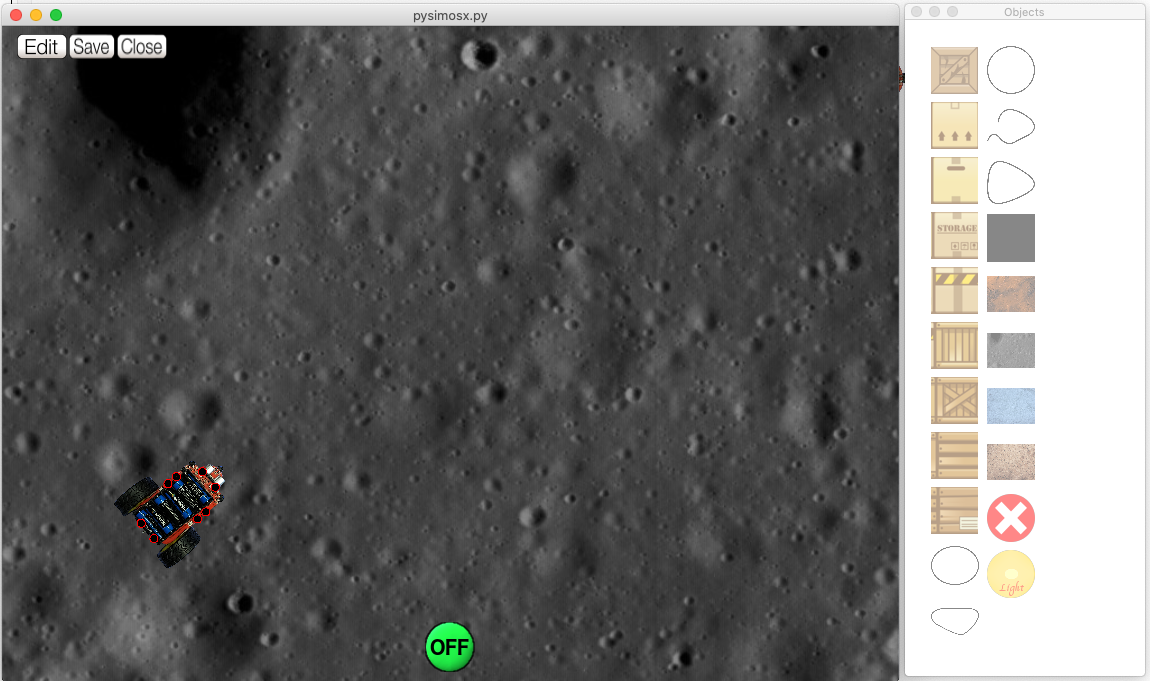
Once the simulation has started, you can stop it by clicking *Close* in the Pi2Go window.

You can exit the simulator by clicking *Quit.*

**Opening the Edit Window**

You can change the environment your virtual Pi2Go encounters by using objects from the Edit window.

Click on *Edit* in the Pi2Go Simulator. A second window, the Objects window, should open as shown in the screenshot below



The Objects Window contains 8 blocks, 5 lines, a Black Square, 4 backgrounds, a Delete button and a Light source.

1. Select the Objects Window
2. Click on the top block
3. Select the Simulation Window
4. Right click somewhere.

**Question 1:** What happens?

1. Select the Objects Window
2. Click on the Delete Button
3. Select the Simulation Window
4. Right click on the Block



**Question 2:** What happens?

1. Select the Objects Window
2. Click on the Black Square
3. Select the Simulation Window
4. Right click somewhere

(the Black Square should appear)

1. Left click on the Black Square in the Simulation Window

and use the mouse to drag it around.

**Question 3:** If you drag the Black Square to the same space as the robot does it appear over or under the robot?

Add a block object and drag it around with the mouse.

**Question 4:** What happens when the block is dragged over the robot?

Try adding and removing and moving:

1. Lines
2. Backgrounds
3. Light Source

**Question 5:** Can backgrounds be moved by clicking and dragging? YES/NO



**Question 6:** How do you move the ray of light from the Light Source?

When you have altered a world, you can save it by clicking *Save* but you will need to change its name.

When you want to exit the simulator, select click *Close* in the Pi2Go world window, followed by *Quit* in the simulator window.

**Chapter 2: Command Lines**



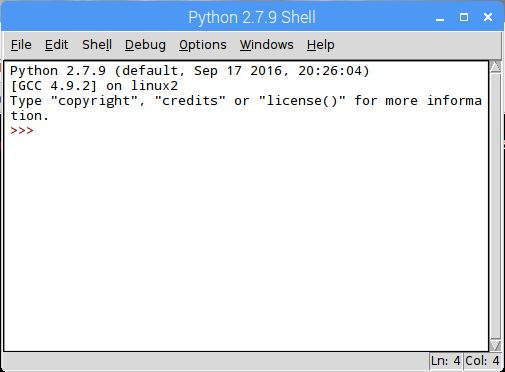
**AIM:** After completing this chapter you should be able to start and stop the IDLE integrated development environment and use simple commands.

**You Need:** To complete this worksheet you need to have access to IDLE.

You should have the virtual simulator running and have selected the Pi2Go robot and the default\_world.xml world (WS1).

For simple programming tasks we can use the *Python Command Line Interpreter*. We are going to use this from within an *integrated development environment* called IDLE.

Start IDLE (if you already have IDLE running for the simulator you should start **a *new IDLE window***for programming the virtual robot).



**3 signs (>>>) are called the prompt**

The Python Command Line Interpreter in IDLE

You should now be in the Python Interpreter which should look like the above. You DO NOT need to type the “prompt” (**>>>**) before any python commands.

**print(“Hello World”)**

followed by Return.



**Question 1:** What happens?

We will now try to import a module for our Pi2Go robot.

Type

**import simclient.simrobot as pi2go**

followed by Return.

**IMPORTANT:** if you get an error at this point that says

ModuleNotFoundError: No module named 'simclient.simrobot'

Then you need to add the simulator to your PYTHONPATH. **To do this see the box at the end of this worksheet.**

open the file setup\_programming.py using the File menu (you will find setup\_programming.py in the pirover\_simulator folder). Once this file is open select **Run Module** from the **Run** menu.

This will print the path to the simulator directory. You can ignore this for now, but will need it when you start using files for your programs.

Then try

**import simclient.simrobot as pi2go**

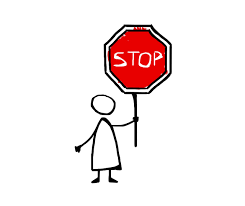
followed by Return.

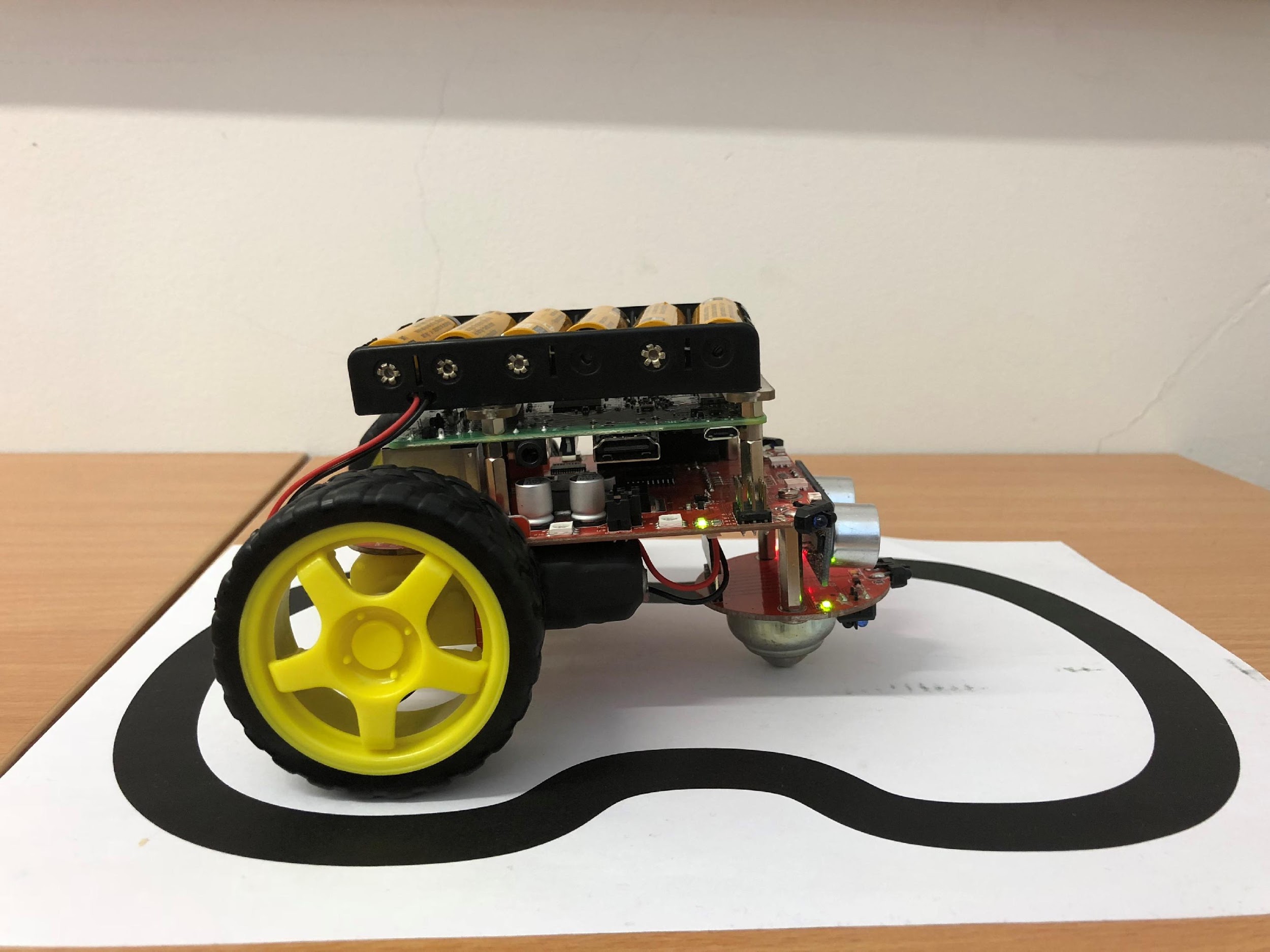
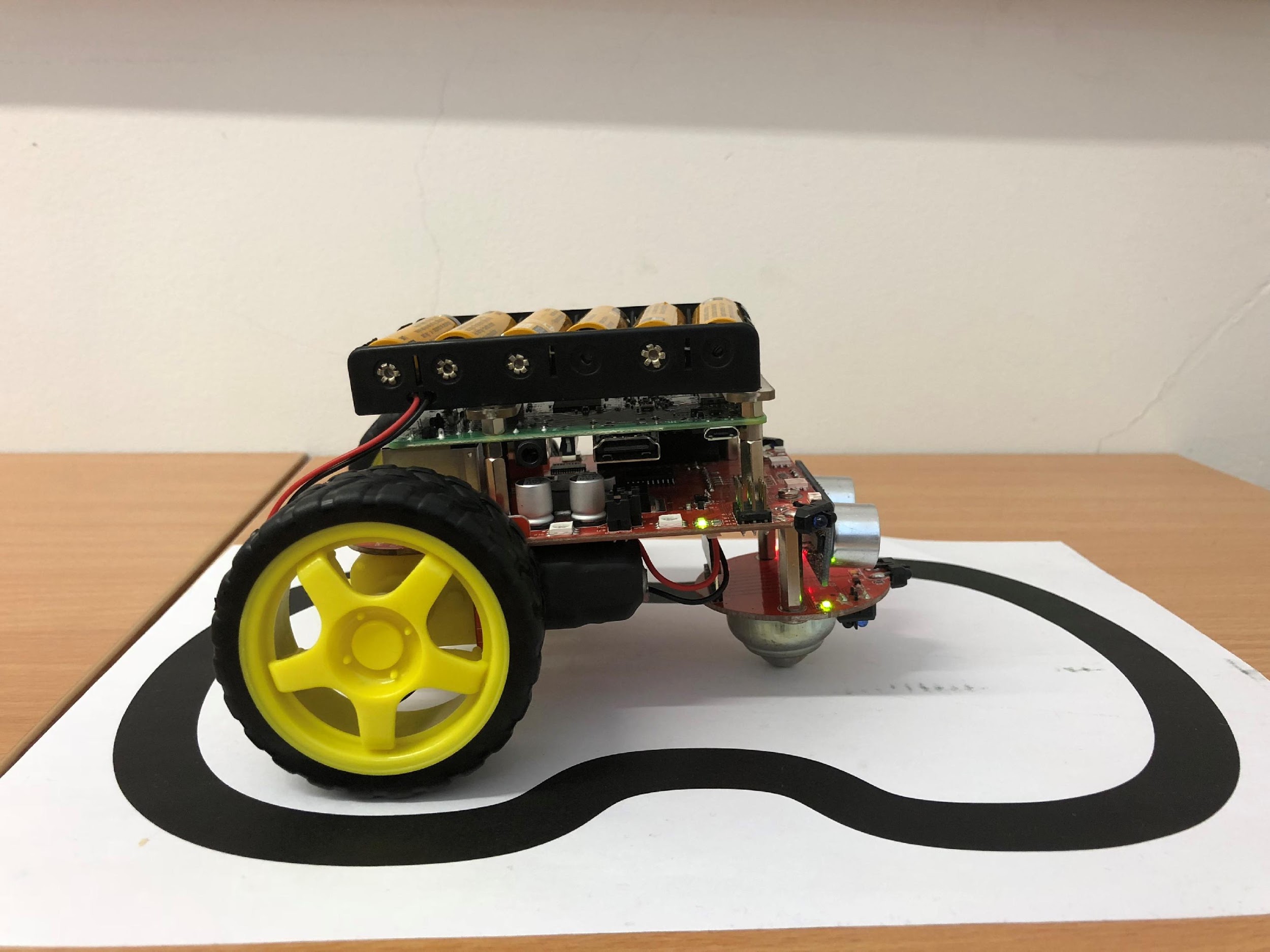
Now type the following commands one after another, each followed by Return.

**pi2go.init()**

**MOVE**

**MOVE**

**pi2go.forward(10)**

**pi2go.stop()**



**Question 2:** What happens?

You can use *alt-p* (pressing the Alt key and P at the same time) to scroll back through previous commands.

Try replaying some commands using *alt-p*

You will learn more about controlling the Pi2Go robot from the Python Command Line in chapter 3.

If you have finished working with your robot type:

**pi2go.cleanup()**

Otherwise carry on to chapter 3.

**Chapter 3:**

**Python Statements and Pi2Go Commands**



**AIM:** After completing this chapter you should be able to control your Pi2Go simulation using simple statements issued at the Python Command Line and be able to explain what a statement in a programming language is.

**You Need:** To complete this worksheet you need to have a virtual Pi2Go simulator (see Chapter 1) and understand how to control the virtual Pi2Go from the IDLE command line (see Chapter 2).

**If the simulator isn’t already running:**

Start it (see Chapter 1) and select the Pi2Go robot and default\_world.xml

**Now open a *new* IDLE window**.

**Remember:** You can scroll back through commands in IDLE by typing *alt-p*

In order to control your robot simulation you need to *initialise it properly.* You did this as part of Worksheet 2.

To initialise your simulation you type

>> import simclient.simrobot as pi2go

>> pi2go.init()

If you have not already done this from worksheet 2. Do it now.

Now you can use simple commands to control your simulated robot. Try typing the following:

>> pi2go.forward(20)

>> pi2go.stop()



**Question 1:** What happens?

Each of these commands pi2go.init(), pi2go.forward(20), pi2go.stop() is a *python statement.* Statements are the basic commands that are used to build up programs. You have several commands available to you for operating the Pi2Go robot.

These commands are described in the box.

**IMPORTANT:** Note that parts in *italics* in the box are inputs to the commands which you have to select. So, for **pi2go.spinLeft(***speed***)** you have to replace *speed* with a number between 0 and 100. For instance, **pi2go.spinLeft(30)**

**pi2go.stop()**

**pi2go.forward(***speed***)**

where *speed* is a number and 0 <= *speed* <= 100

**pi2go.reverse(***speed***)**

where *speed* is a number and 0 <= *speed* <= 100

**pi2go.spinLeft(***speed***)**

where *speed* is a number and 0 <= *speed* <= 100

**pi2go.spinRight(***speed***)**

where *speed* is a number and 0 <= *speed* <= 100

**pi2go.turnForward(***leftSpeed, rightSpeed*

where *leftSpeed* and *rightSpeed* are numbers and

0 <= *leftSpeed, rightSpeed* <= 100

**pi2go.turnReverse(***leftSpeed, rightSpeed***)**

where *leftSpeed* and *rightSpeed* are numbers and

0 <= *leftSpeed, rightSpeed* <= 100

**pi2go.go(***leftSpeed, rightSpeed***)**

where *leftSpeed* and *rightSpeed* are numbers and

-100 <= *leftSpeed, rightSpeed* <= 100

**pi2go.go(***speed***)**

where -100 <= *speed* <= 100

**pi2go.setLED(***LED, Red, Green, Blue***)**

where *LED, Red, Green* and *Blue* are numbers and

0 <= *LED* <=3 and 0 <= *Red, Green, Blue* <= 4095

**pi2go.setAllLEDs(***Red, Green, Blue***)**

where *Red, Green* and *Blue* are numbers and 0 <= *Red, Green, Blue* <= 4095

Try typing the following:

>> pi2go.setLED(3, 0, 0, 1000)



**Question 2:** What happens?

pi2go.setLED controls two *light emitting diodes (LED)*. It takes four inputs: the number for the LED (0 is front, 1 is right, 2 is back and 3 is left) , followed by numbers for the amount of Red, Green and Blue light to display. So this command gets the right LED to shine quite a lot of blue light and no red or green light.

Try typing the following:

>> pi2go.setAllLEDs(1000, 0, 0)



**Question 3:** What happens?

**Question 4:** Try six of the other commands from the table. What do they do?

**Command Result**

.

**Chapter 4: Sensors**



**AIM:** After completing this chapter you should be able to describe the sensors on the Pi2Go simulation and use Python to get information from them.

**You Need:** To complete this worksheet you need to have to have a virtual Pi2Go simulator (see Chapter 1), and understand how to control it from the IDLE command line (see Chapter 2).

**If the simulator isn’t already running:** Start it (see Chapter 1) and select the Pi2Go robot and default\_world.xml

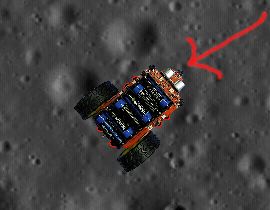
Now open a new IDLE window.

A Pi2Go robot has four different kinds of sensors:

1. **Ultrasonic:** It has an ultrasonic sensor which uses sound and echoes to measure distances.
2. **Infrared:** It has several infrared sensors that measure how much infrared light is reflected. These can be used both to measure distances (like with the ultrasonic sensor) and to measure colour – for instance a black surface reflects less infra-red light than a white surface.
3. **Light:** The Pi2Go also has several light sensors which detect the intensity of light shining on them.
4. **Switch:** Finally the Pi2Go has a switch which can detect whether it is pressed or not.

These sensors are all simulated on your virtual Pi2Go and you can use Python to get values from the sensors.

**The Ultrasonic Sensor**



The Ultrasonic Sensor

Locate the ultrasonic sensor on your virtual Pi2Go. There is one command you can use to get information from the ultrasonic sensor. This is:

>>> pi2go.getDistance()

pi2go.getDistance() returns the distance in cm to the nearest reflecting object. A value of zero means it does not detect an object

Type pi2go.getDistance()



**Question 1:** What value do you get?

Let us put something in front of the virtual Pi2Go.

Type E anywhere in the Simulator Window to open the Objects window and place one of the blocks in front of the Pi2Go’s distance sensor.

Try getting the distance using the Python command.

**Question 2:** What value do you get?

Now try moving the object closer to the sensor using the mouse to click and drag it.

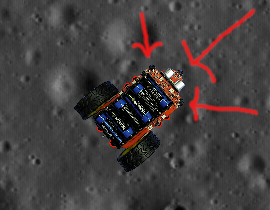


**Question 3:** What value do you get?

**Infrared Sensors**

The Pi2Go’s infrared sensors are organised into two groups: sensors for detecting obstacles and sensors for detecting lines.

There are three sensors for detecting obstacles grouped at the front of the robot:



The locations of the InfraRed Obstacle Sensors

There are four commands for getting information from the obstacle sensors:

>>> pi2go.irLeft()

returns 1 (meaning **True**) if the Left IR Obstacle sensor detects an obstacle

and 0 (meaning **False**) otherwise

>>> pi2go.irRight()

returns 1 (meaning **True**) if the Right IR Obstacle sensor detects an obstacle

and 0 (meaning **False**)otherwise

>>> pi2go.irCentre()

returns 1 (meaning **True**) if the Centre IR Obstacle sensor detects an obstacle

and 0 (meaning **False**) otherwise

>>> pi2go.irAll()

returns 1 (meaning **True**) if any of the Obstacle sensors detect an obstacle

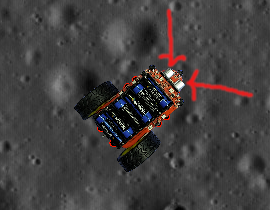
and 0 (meaning **False**) otherwise

Using the Objects Window place blocks so that pi2go.irAll() returns 1.

**Question 4:** How many blocks did you need?

The Pi2Go’s other infrared sensors are used to detect how dark the colour of the surface the robot stands on is. These are referred to as “line sensors” since their primary use is for detecting black lines on white surfaces.

There are two line sensors.



The Line Sensors

There are two command for getting information from the line sensors:

>>> pi2go.irLeftLine()

returns the state of Left IR Line sensor

>>> pi2go.irRightLine()

returns the state of Right IR Line sensor

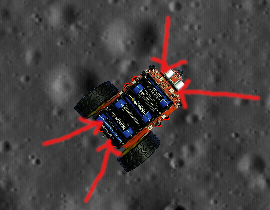
In the Objects Window select the Black Square and place it under the front of the robot.

**Question 5:** What do the Python commands return if the Pi2Go is over the black square?

**Question 6:** What do they return if the Pi2Go is over the background?

**Light Sensors**

There are four light sensors: two at the front and two at the back.



Front and back light sensors.

There are five commands for getting data from the Pi2Go light sensors.

>>> pi2go.getLight(0)

returns the value 0..1023 for the the sensor numbered 0.

Try getting values for sensors 1, 2 and 3 as well.

>>> pi2go.getLightFL()

returns the value 0..1023 for Front-Left light sensor

>>> pi2go.getLightFR()

returns the value 0..1023 for Front-Right light sensor

>>> pi2go.getLightBL()

returns the value 0..1023 for Back-Left light sensor

>>> pi2go.getLightBR()

returns the value 0..1023 for Back-Right light sensor

Put a light source into the world.

Check the values returned by one of the sensors when

1. the light is in front of the sensor shining towards it,
2. the light is in front of the sensor shining away from it,
3. the light is on the other side of the robot from the sensor shining towards it

**Question 7:** What value is returned?

Case A Case B Case C

**Switch**

The switch is on the side of the real Pi2Go.

In the Pi2Go simulator, you will find it displayed as a button at the bottom of the virtual world window.



The Pi2Go Switch

There is one command for getting data from the Pi2Go Switch:

>>> pi2go.getSwitch();

returns True when the switch is pressed and False when it isn’t.

**Try using the switch now.**

**Chapter 5: Worlds**



**AIM:** This chapter provides additional activities for investigating the Pi2Go robot sensors. It assumes familiarity with the material in chapters 1-4.

**Exercise 1**: So far, we have used our Pi2Go simulation only in default\_world.xml. For this exercise we will look at a different world.

If the simulator is currently running, you need exit default\_world.xml (by clicking on close) and disconnect your virtual Pi2Go robot by typing pi2go.cleanup() in the IDLE window where you have been doing the previous exercises.

Open maze1.xml. In the IDLE window where you have been doing the exercises, reconnect your virtual initio by typing pi2go.init();

**Question 1:** What are the values returned by the ultrasonic sensor and the three infrared distance sensors in this world?

Drive the Pi2Go using the motor commands to another part of the world and take new values for the ultrasonic and infrared sensors.



**Question 2:** What happens if the Pi2Go hits one of the blocks?

**Exercise 2:** Put a light source into the world. Notice that the light source shines over the boxes.

What is the value returned by pi2go.getLight(0) and pi2go.getLightFL()?



**Question 3:** Why are these the same?

**Question 4:** Give another example of two light sensor commands that will give the same answer?

**Exercise 3:**

Close maze1.xml and disconnect your virtual Pi2Go robot by typing pi2go.cleanup() in the IDLE window where you have been doing the exercises.

Open line\_following.xml. In the IDLE window where you have been doing the exercises, reconnect your virtual initio by typing pi2go.init()



**Question 5:** What are the values returned by the two infrared line sensors in this world?

Move the robot backwards and forwards until you find a spot where the two infrared line sensors both return 0.

Note that this means the simulators sensors are not in quite the right place.

**Chapter 6: Programs in Files**



**AIM:** After completing this chapter you should be able to write a Python program using IDLE’s program editor and execute it in IDLE.

**You Need:** To complete this worksheet you need to have a virtual Pi2Go Simulator (see chapter 1).

**If the simulator isn’t already running:** Start it (see chapter 1) and select the Pi2Go robot and default\_world.xml

Now open a new IDLE window.

You can control your virtual Pi2Go by writing commands at the Python command line. However, for longer programs, or programs you wish to run several times this can become tedious. To get around this you can write a Python program in a file.

We are going to consider the following Python program

import simclient.simrobot as pi2go

pi2go.init()

pi2go.forward(10)

pi2go.setAllLEDs(2000, 2000, 2000)

pi2go.stop()

To write this program we are going to use the editor that comes with IDLE.

Select **File > New File** in IDLE.

Type the program above into this file (make sure you spell everything correctly).

You can save the file using the **File** menu. We suggest you create a folder for your Python programs.

You can execute your program by selecting **Run > Run Module**



**Question 1:** Try this now. What happens?

If you are lucky this will have run smoothly.

**IMPORTANT:** if you get an error at this point that says

ModuleNotFoundError: No module named 'simclient.simrobot'

Then you need to add the simulator to the python path called by the file.  **O**pen the file setup\_programming.py using the File menu (you will find setup\_programming.py in the pirover\_simulator folder). Once this file is open select **Run Module** from the **Run** menu.

This will print out the path to the pirover\_simulator.

At the top of your file you now need to add the lines

import sys

sys.path.append(*path\_to\_simulator*)

Where you replace *path\_to\_simulator* with the path that was printed when you ran setup\_programming.py

If the file doesn’t run smoothly (and you don’t have a path problem as described in the box above), you should have received an error message and will need to check your program for spelling errors.

**Exercise 1:** Modify your program so that it sets all the LEDs to 2000 first and then sets them back to 0 after the Pi2Go has stopped.

**Chapter 7: The Time Module**



**AIM:** After completing this chapter you should be able to use commands from Python’s Time module to delay execution of commands on the Pi2Go. You should also be able to explain how to import a module into Python.

**You Need:** To complete this chapter you need to have a virtual Pi2Go Simulation (see chapter 1), and to be able to use files to store Programs (see chapter 5). You also need to know the commands to operate the virtual Pi2Go motors (see chapter 3).

**If the simulator isn’t already running:** Start it (see chapter 1) and select the Pi2Go robot and default\_world.xml

Now open a new IDLE window.

When we program, we don’t always want to write everything the program should do from scratch. When someone has already solved a problem, for instance delaying the execution of a command, then we would like to be able to use their solution. This is done by importing *modules* (in some languages called libraries) of pre-programmed commands into our program.

You are already using a module in your programs. The statement import simclient.simrobot as pi2go at the start of all your programs is importing the simulation module which contains the commands for operating your virtual Pi2Go robot, and renaming this module pi2go.

In this worksheet we will use the time module to delay the execution of some of those commands.

Create a file containing the following program and execute it:

import simclient.simrobot as pi2go, time

pi2go.init()

pi2go.forward(10)

time.sleep(10)

pi2go.stop()



**Question 1:** What happens?

**The Sleep Command**

The command time.sleep(*seconds*) makes the program pause for the number of seconds before executing the next command.

**Exercise 1:** Adapt the program to make the robot move forward for 20 seconds before stopping.

**Question 2:** What did you change?

**Exercise 2:** Write a program to make the robot wait for 30 seconds, move forward for 10 seconds, turn for 10 seconds and then stop.

Write your program below:

.

**Chapter 8: Simple Programs**



**AIM:** This chapter provides additional simple programs for you to write for your virtual Pi2Go. It assumes familiarity chapters 1-7.

**Exercise 1**: Write a program that will spin the robot for 2 seconds, then take a distance sensor reading and print it to the screen and stop.

To print the value of the sensor use: print(pi2go.getDistance())

**Exercise 2**: Write a program which will make the robot’s LEDs flash blue for 5 seconds, then green for 5 seconds and then red for 5 seconds.

**Exercise 3**: Place a Light Source in the world. Write a program so that your robot performs a full turn on the spot, printing out the light measurements from each light sensor after roughly each quarter turn. Some experimentation will be needed to work out how long a quarter turn takes for your virtual robot.

**Exercise 4:** Write a program where the robot moves forward changing speed every 10 seconds for a total of 30 seconds.

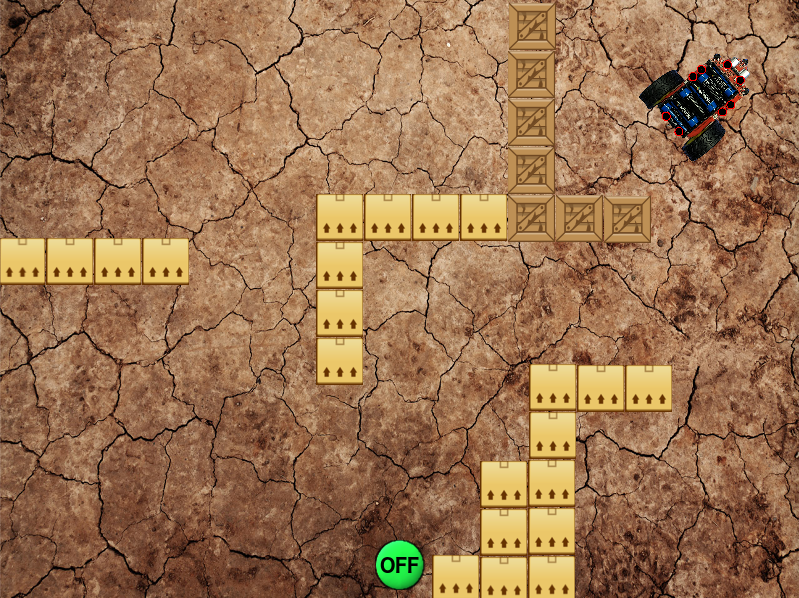
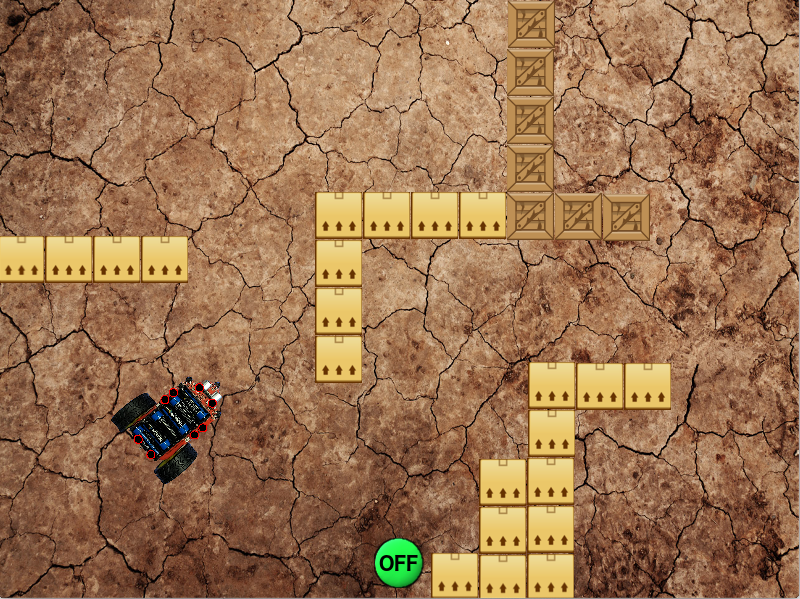
**Exercise 5**: Pick four colours for the LEDs (we will call the colours A, B, C, D). Write a program so that the robot displays colour A at the front, colour B on the left, colour C at the back, and colour D on the right. It moves forward for 10 seconds. Then it rotates the colours (so A is now on the left, B at the back and so on). Then it spins on the spot for 10 seconds. Then it rotates the colours again. Then it reverses for 10 seconds. Then it rotates the colours again. Then it stops.

**Exercise 6**: Exit the simulator (by pressing Q) and disconnect your virtual Pi2Go robot by typing pi2go.cleanup() in the IDLE window where you have been doing the previous exercises.

Now re-start the simulator and select the world maze1.xml.

In the IDLE window where you have been doing the exercises, reconnect your virtual robot by typing pi2go.init();

Write a program that will move your robot from the start position to the top right hand corner (without you clicking and dragging the robot). To do this you will need to experiment to find out how long it needs to move forward in each stage and how long it needs to turn for in each stage.



Move the robot from the position on the left to the position on the right.

**Chapter 9: If**



**AIM:** After completing this chapter you should be able to use **if**, **elif** and **else** statements in Python to create more complex programs.

**You Need:** To complete this chapter you need to have a virtual Pi2Go simulation (see chapter 1), and to be able to use files to store programs (see chapter 6). You also need to know the commands to operate the virtual Pi2Go motors and sensors, and the time.sleep command (see chapters 3, 4 and 6).

**If the simulator isn’t already running:** Start it and select the Pi2Go robot and default\_world.xml**.** Now open a new IDLE window.

We want programs to be more flexible than simply following a series of instructions. We might want them to do different things in different circumstances. For instance, we might want our Pi2Go robot to move backwards if there is an obstacle in front of it and forwards otherwise.

The following program will do this:

import simclient.simrobot as pi2go, time

pi2go.init()

if (pi2go.irCentre()):

print(“Obstacle!”)

pi2go.reverse(10)

else:

pi2go.forward(10)

time.sleep(10)

pi2go.stop()

There are a few things to note here. Python uses spaces to tell when something is part of a block of code inside an if statement. This feature is unique to Python and is not used in many other programming languages.

So in the program above, Python knows that it should print “Obstacle!” and then execute pi2go.reverse(10) if there is an obstacle detected by the centre sensor, because these statements follow after the if and are indented.

Similarly, it knows that pi2go.forward(10) should be executed if there is no obstacle, because it follows after the else and is indented.

Note also that we have colons after the if and the else.

**Question 1:** Create a file containing this program and execute it. What happens?

**Exercise 1:** Describe how you would test your program to make sure it was working correctly both when there was an obstacle and when there wasn’t.

**Question 2:** Perform your tests. Is your program working correctly? YES/NO

**Exercise 2:** Write a program using an if statement that will turn in one direction if there is an obstacle on the right and turn in the other if there is an obstacle on the left.

*Hint:* To do this you may want to use elif which means (else if) - e.g.

elif (pi2go.irLeft()):

instead of else:

**Question 3:** How many cases do you need to consider to test your program?

Does the description of the exercise tell you what the program should do in all cases? YES/NO

**Exercise 3:** Modify the program so that the robot reverses for 10 seconds if there is an obstacle and then turns for 10 seconds.

If there isn’t an obstacle the robot moves forward.

**Chapter 10: While Loops**



**AIM:** After completing this worksheet you should be able to use while loops in Python programs.

**You Need:** To complete this worksheet you need to have a virtual Pi2Go (see chapter 1), and to be able to use files to store programs (see chapter 6). You also need to know the commands to operate the Pi2Go motors and sensors (see chapters 3 & 4).

**If the simulator isn’t already running:** Start it and select the Pi2Go robot and default\_world.xml**.** Now open a new IDLE window.

A while loop allows a Python program to continue executing some statement multiple times *while* something remains true. For instance, we might want our Pi2Go to keep reversing while there is an obstacle.

import simclient.simrobot as pi2go

pi2go.init()

pi2go.reverse(10)

while (pi2go.irCentre()):

print(“Reversing”)

pi2go.stop()

Notice that we start reversing *before* we execute the while loop. This is because once you start your Pi2Go moving, it carries on doing that until it is told to do something else. So we are simply using the while loop prevent the program moving on to the stop command, until the obstacle is no longer detected.



**Question 1:** Create a file containing this program and execute it. What happens?

**Question 2:** What sort of tests would you need to perform to check the program was working properly in all cases?

**Chapter 11: Using Logic in Conditions**



**AIM:** After completing this chapter you should be able to use logic to construct more complex conditions for if and while statements.

**You Need:** To complete this chapter you need to have a virtual Pi2Go simulator (see chapter 1), and be able to use files to store programs (see chapter 6). You also need to know the commands to operate the Pi2Go motors, LEDs and sensors (see chapters 3 & 4) and how to use if and while statements in Python (see chapters 9 & 10).

**If the simulator isn’t already running:** Start it and select the Pi2Go robot and default\_world.xml**.** Now open a new IDLE window.

**Problem:** When you program a real Pi2Go robot it has to be connected to a keyboard and monitor which is very cumbersome. Ideally you want to have time after the program has started running to disconnect everything, before the robot starts to move.

Consider the program below

import simclient.simrobot as pi2go, time

pi2go.init()

while not (pi2go.getSwitch()):

print("Waiting")

pi2go.setAllLEDs(4095, 4095, 4095)

time.sleep(1)

pi2go.setAllLEDs(0, 0, 0)

**Note:** On the real robot the switch is ON when pressed down and OFF when not pressed down. In the simulator you click on the switch to turn it ON and then click on it again to turn it OFF.



**Question 1:** What does this program to do?

**Question 2:** Execute the program. Did it do what you expected? YES/NO.

**Exercise 1:**  Modify the program by adding a second while loop so that the LEDs stay on until the switch is switched on a second time.

**Logic in conditions:** You can use *logic* to make the conditions of your Python while and if statements more flexible. The main *logical operators* are not, and and or.

We can use these to build up logical expressions so, for instance:

|  |  |
| --- | --- |
| pi2go.getSwitch() | returns True if the switch is on |
| not (pi2go.getSwitch()) | returns True if the switch is off |
| (pi2go.getSwitch() or pi2go.irCentre()) | returns True if the switch is on *or* there is an obstacle in the centre |
| (pi2go.getSwitch() and pi2go.irCentre()) | returns True if the switch is on *and* there is an obstacle in the centre |

**Exercise 2:** Modify your program again so that if the switch is on or there is an obstacle then the Pi2Go reverses. Then when the switch is turned on (a second time if it was pressed the first time), the Pi2Go stops moving.

What expression are you using in the condition (if statement)?

You can make conditions more and more complex by composing the logical operators.

**Exercise 3:** Modify your program again so that if the switch is on and there is not an obstacle then the Pi2Go moves forward. Then when the switch is turned on a second time, Pi2Go stops moving.



What expression are you using in the condition?

.

**Chapter 12: More Loops**

**AIM:** After completing this chapter you should be able to use break, continue and while True: in programs

**You Need:** To complete this worksheet you need to have a virtual Pi2Go simulator (see chapter 1), and to be able to use files to store Programs (chapter 6). You also need to know the commands to operate the Pi2Go motors and sensors (chapters 3 & WS4). You should be able to use If statements (chapter 9) and while loops (chapter 10) in Python programs.

**If the simulator isn’t already running: Start the Simulator, Select the Pi2Go Simulation and default\_world.xml, then start IDLE (open a *new IDLE window* if you have used IDLE to start the simulator).**

Loops are fundamental to programming. However, there are a few constructs that make them much more flexible and easy to use.

**Break:** The command break is used to “break out” of a **while** loop. It is normally used inside an **if** statement and will stop the loop without executing anything that comes after it.

Consider the following program

import simclient.simrobot as pi2go

import time

pi2go.init()

while (not pi2go.irLeft()):

if (pi2go.irRight()):

break

pi2go.forward(10)

time.sleep(10)

pi2go.stop()

**Question 1:** What does it do?

**Continue:** Continue is used to mean start executing the loop back from the beginning, skipping the rest of the code. An obvious use of continue is when you simply want a loop to execute until something happens.

Consider the following program

import simclient.simrobot as pi2go

import time

pi2go.init()

while (not pi2go.irLeft()):

continue

pi2go.forward(10)

while (not pi2go.irRight()):

continue

pi2go.stop()

**Question 2:** What does it do?

**While True:** Lastly we can use the construct while True: in order to create a loop that just keeps executing until someone presses Control-C to exit the program or a **break** statement is used.

**Exercise 1:** Write a program using while True: and break. Where the robot starts moving the left IR sensor detects an obstacle and then it stops.

**Exercise 2:** Write a program using break, continue, elif and while True that does the following.

1. If there is something in front of both the right and left IR sensors the program and robot stop.
2. If there is something in front of the right sensor the robot should spin left and print “spinning to avoid obstacle”
3. If there is something in front of the left sensor the robot should spin right and print “spinning to avoid obstacle”
4. Otherwise the robot should move forward and print nothing.
5. Use only one print statement.

**Chapter 13: Logic and Control**



**AIM:** This chapter provides additional exercises using logic, if statements and while loops for you to write for your virtual Pi2Go. It assumes familiarity with chapters 1-12.

**Exercise 1**: Write a program that will light up the front LEDs if there is an obstacle in front of the Pi2Go, and the left side LEDs if there is an obstacle on the left and the right side LEDs if there is an obstacle on the right.

**Exercise 2**:  Write a program that behaves as follows: When the switch is pressed the program starts printing out the value from the ultrasonic distance sensor. When the switch is pressed again it stops printing the distance out.

**HINT:** You may to use the time module to insert a small delay after the switch is pressed for the first time.

**Exercise 3**: Extend your program from exercise 1 so it keeps changing which LEDs light up as obstacles appear and disappear.

**Exercise 4:** Write a program that will make the Pi2Go reverse if the switch is pressed and there is an obstacle in front of it. Once the obstacle disappears it should stop reversing.

**Exercise 5**: Write a program that will make the Pi2Go move forward if the switch is pressed and there is no obstacle in front of it. Once an obstacle appears it should stop moving.

**Exercise 6**: Adapt your programs from exercises 4 and 5 so that the robot reverses until there is no obstacle and then moves forward until there is an obstacle and keeps repeating this until the switch is released.

.

**Chapter 14: Obstacle Avoidance**



**AIM:** After completing this chapter you should be able to integrate your Python programming knowledge to create obstacle avoidance behaviour in a virtual Pi2Go robot.

**You Need:** To complete this worksheet you need to have a virtual Pi2Go simulator (see chapter 1), and to be able to use files to store Programs (chapter 6). You also need to know the commands to operate the Pi2Go motors, LEDs and sensors (chapters 3 & 4). You should be able to use If statements (chapter 9) and while loops (chapter 10) in Python programs.

**If the simulator isn’t already running: Start the Simulator, Select the Pi2Go Simulation and default\_world.xml, then start IDLE (open a *new IDLE window* if you have used IDLE to start the simulator).**

**Challenge:** Implement obstacle avoidance behaviour on a Pi2Go robot.

**Designing the program:** Before you can implement a behaviour you should try to define what that behaviour involves. In this challenge we will define obstacle avoidance behaviour to mean: “The Pi2Go should move forwards while there is no obstacle and should turn (either right or left) when there is an obstacle”.

We can illustrate this with a flow chart

Is there an obstacle?

Turn

Forward

Yes

No

**Question 1:** What sensor and motion commands do you expect to use for this challenge?

You can go ahead and attempt to write the program now – or you can follow the suggested steps below.

**Step 1:** Write a python program containing a while loop, which will make your Pi2Go move forward while there is no obstacle and then stop when an obstacle appears. Test your program works.

**Step 2:** Extend your program so that you have a Python program that will move forward while there is no obstacle, then will turn while there is an obstacle, and then stops. Test your program works.

**Step 3:** Now you want to extend the program so instead of going around one while loop (while there is no obstacle) and then a second (while there is an obstacle) and then stopping, it instead repeats going around the two while loops. You can do this with a third while loop that contains the other two and which starts with while True:

**Question 2:** Write out how you are going to test that your program works.

**Question 3:** Execute your program. Does it work? YES/NO

**Extending the Program:** While you have successfully implemented obstacle avoidance behaviour, the behaviour of your program isn’t as nice as you might like. For instance, you have to use Ctrl-C to stop the program running and then stop the Pi2Go at the Python command line.

**Exercise 1:** Extend the program so you can stop and start it easily using the switch.

.

**Exercise 2**: Write a program that will follow a line. You should use the same techniques you used to design and develop your obstacle avoidance program and use **line\_following.xml** world to test the program. Your program design should involve the robot moving forward when its line sensors are either side of the line and turning left or right as appropriate if one of the line sensors detects the line.

You may have to move the robot to a suitable starting position.



University of Liverpool, 2020

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).