**Initio Programming: Debugging**

**AIM:** After completing this worksheet you should be able to identify problems in programs based on error messages and use the IDLE debugger.

**You Need:** To complete this worksheet you need to have an Initio (see WS1) and a virtual initio is also recommended (see Initio Simulator, WS1), and to be able to use files to store Programs (WS5). You also need to know the commands to operate the Initio motors and sensors (WS3 & WS4). You should be able to use If statements (WS7), while loops (WS8 & WS10), variables, numbers (WS12) and strings (WS13) in Python programs.

You have written a number of programs by now and so are probably already familiar with some of the error messages you might get.

Consider the following program. It contains an error:

import robohat as initio

initio.init()

direction = input(["Which way would you like the robot to move? (F, B, L, R)"])

while(direction != "S"):

if (direction == "F"):

initio.forward()

elif (direction == "B"):

initio.reverse(10)

elif (direction == "L"):

initio.spinLeft(10)

elif (direction == "R"):

initio.spinRight(10)

direction = input(["Which way would you like the robot to move next? (F, B, L, R, S)"])

initio.stop()

If you run this program and type F as input you should see the following error message:

Traceback (most recent call last):

File "/Users/louisedennis/PiRovers/pirover\_simulator/examples/tmp2.py", line 12, in <module>

initio.forward()

TypeError: forward() missing 1 required positional argument: 'speed'

Read the error message carefully.

**Question 1:** On what line has the error occurred?

**Question 2:** What does the error message say is missing from forward() ?

An *argument* is something that goes in the brackets. You can fix this program by putting something in the brackets for forward().

Correct the program and check that it works. In order to do this, you might want to use the Initio Simulator before you use the Initio, itself (you will need to change the first line in each program to import simclient.simrobot instead of robohat). This will save the Initio batteries, allow other people to use the robot if you are sharing, and it will be quicker to use the simulator where you don’t have to connect and disconnect cables. In many programming situations with robots it is useful to have a simulator in order to develop and debug programs and only work with the actual robot once your program works on the simulator.

**Debuggers**

Sometimes problems in programs can not be found simply by looking for a syntax error. Many programmers use print statements to help them understand what their program is doing and what the values of variables may be but there is a better way to do this and that is by using a *debugger.* There are a lot of different debuggers and each one works differently key features however are

* **Code stepping** a debugger will let you execute a program one command at a time. This will let you see what is happening at each stage of the program, which branches of if statements are being used and similar things.
* **Breakpoints** in long programs simply stepping through every instruction can get tiresome, particularly if you know that the error doesn’t occur until later in the program. Debuggers will let you set a *breakpoint* in your code and then execute the program until the breakpoint is reach at which point you can start stepping through the code.
* **Inspecting Variables** Debuggers let you see what the values of variables are which can help you figure out what is going wrong.

We are going to look at the debugger that comes with IDLE. Again, we recommend you use the debugger with the Initio Simulator in order to get the program right and then check the resulting program on the actual robot, though you can use the debugger there if you want.

To start the debugger, you should click on the **Debug** menu in IDLE’s Python Shell window and select **Debugger.** A window a bit like the following should appear and the words [DEBUG ON] will appear in the Shell window.

A screenshot of a cell phone

Description automatically generated

The Debugger Window

Run the Python Program you have been working on. You will notice that it doesn’t run, but a lot of information appears in the Debugger window such as \_\_file\_\_ showing the file name. The five buttons **Go, Step, Over, Out** and **Quit** are available.

**Question 3:** Click on **Go**. What happens?

Stop the program either by selecting S when prompted by the program.

Now run the program again and this time click **Over** instead. **Over** lets you step through each line in the program (we will discuss the use of **Step**) in a later tutorial.

In the top part of the debugger window you can see the line of code that the debugger is currently at – for instance

* ‘\_\_main\_\_’.<module>(), line 9: while(direction != “S”):

Is at line 9 of the program – the command while(direction != “S”):

The bottom part of the debugger window contains values of variables like direction.

Step through the code entering all the different letters F, B, L, R, S as prompted by the program.

**Question 4:** How many times do you have to click **Over**?

**Question 5:** Run the program again and click **Over** a couple of times and then click **Go.** What happens?

Lastly, we will look at how to set a **breakpoint.** In the program right click on the line

elif (direction == "L"):

and select **Set Breakpoint**. The line should go yellow.

**Question 6:** Now run the program and click **Go.** Enter R when prompted by the program. What happens?

**Question 7:** What line have you stopped at?

**Question 8:** What is the value of direction?



University of Liverpool, 2019

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