**Virtual Pi2Go Programming: Lists**

**AIM:** After completing this worksheet you should be able to use list data structures in your programs. In particular you should be able to access elements in a list, add items to a list, iterate over a list and join two lists together and extract a sub-list from a list.

**You Need:** To complete this worksheet you need to have a virtual Initio simulator (see WS1), understand how to control the robot’s motors (WS3), be able to use files to store Programs (WS5), be able to compare numbers (WS12) and strings(WS14), and use Python control structures (WS7-10).

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

As the name suggests a Python list stores a list of things. For instance, we could have a list called actions of strings for each action the robot can take:

actions = ['forward', 'backward', 'left', 'right', 'stop']

We can access an element in a list by a number starting from zero. So, actions[0], is the string ‘forward’.

Consider the following program:

import simclient.simrobot as pi2go

import time

actions = ['forward', 'backward', 'left', 'right', 'stop']

initio.init()

element = 0

while element < 5:

action = actions[element]

if (action == 'forward'):

pi2go.forward(10)

time.sleep(3)

elif (action == 'backward'):

pi2go.reverse(10)

time.sleep(3)

elif (action == 'left'):

pi2go.spinLeft(10)

time.sleep(3)

elif (action == 'right'):

pi2go.spinRight(10)

pi2go.sleep(3)

else:

pi2go.stop()

element = element + 1

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

Instead of using while and a counter to examine every element in the actions list, we can instead use for. for x in list: selects each element of list in turn and assigns it to the variable x. So, for instance to select each element of actions in turn and assign them to the variable action we could write:

for action in actions:

**Exercise 1:** Rewrite the above program so that it uses for instead of while and doesn’t use the variable element anywhere.

Write your program below:

Lists can take elements of any type, for instance we can mix strings and numbers:

actions = ['forward', 3, 'backward', 2, 'left', 1, 'right', 5, 'stop']

**Exercise 2:** Write a program using the above list where the robot performs the action that comes first in the list for the number of seconds given second in the list, then the action that comes third for the number of seconds that comes fourth and so on.

Write your program below:

**Useful List Functions:**

The following are useful functions on lists

*list.*append(*item*) adds *item* to the end of a list

>>> actions = ['forward', 'backward', 'left', 'right', 'stop']

>>> actions.append(‘new’)

>>> print(actions)

['forward', 'backward', 'left', 'right', 'stop', 'new']

len(*list)* len returns the length of *list.* So, for instance:

>>> actions = ['forward', 'backward', 'left', 'right', 'stop']

>>> num\_actions = len(actions)

>>> print(num\_actions)

5

*list.*remove(*item)* removes *item* from *list.*

>>> actions = ['forward', 'backward', 'left', 'right', 'stop']

>>> actions.remove(‘left’)

['forward', 'backward', 'right', ‘stop’]

*item* in *list* returns True if *item* is in *list* and False otherwise

>>> actions = ['forward', 'backward', 'left', 'right', 'stop']

>>> ‘backward’ in actions

True

>>> ‘other’ in actions

False

Use these functions to write the following programs:

**Exercise 3:** Write a program which starts the robot moving and takes readings from the ultrasonic sensor every 3 seconds. When it has 30 readings, it stops and prints out a list of all the readings (use append and len).

**Hint:** An empty list is written []

**Exercise 4:**  Write a program which tries each of the robot’s actions in a loop. If, after executing the action for 3 seconds, the robot is closer than 50 to an obstacle then the program removes that action and carries on trying the other actions, until there are no actions left (use remove).

**Testing Hint:** You may need to move obstacles around the robot to test that this is working.

**Exercise 5:** Create a program containing two lists, one of all actions and one of “permitted” actions. The program should try each of the robots actions in a loop. If the action is permitted then the robot should execute the action, otherwise it should print out a message that the action isn’t permitted.

**Lists within lists**

In Python a list may contain another list. For instance, you could create a list with three items containing an action the robot has taken and the values from the distance sensor before and after executing that action for three seconds. You could then store these lists all together in a larger list.

**Exercise 6:** Write a data logging program that loops through each of the robot’s actions. It takes a distance reading before each action and after each action and creates a list containing lists of three items (an action, a before sensor reading, and an after sensor reading). Once it has tried every action, it should print out the list.



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/).