Cohort Session 3, Week 4
Internet of Things

Objectives
1. Read and write data to the Firebase database with Python.
2. Make use of Firebase to establish connection between eBot to Raspberry Pi.

Please work on this lab in a group of **two or three**. Be sure to email your partner all the modified code, printouts and data. You may have to use them during your exams.

You are required to follow the instructions on **Firebase Set Up Guide** before starting work on this project.

1 Equipment & Software

Download the software for this project from **1D Project website**.

Each group should have:

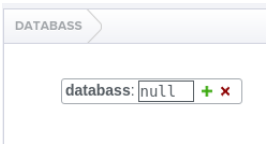
1. An eBot with charging cable.
2. A Raspberry Pi.
3. A cobbler.
4. An LCD touch screen.
5. A wireless keyboard.
6. A wireless mouse.
7. Four push buttons.
8. A few jumper wires.
9. `wk4_firebasebasics.py`, which contains basic code that reads data from Firebase.
10. `wk4_ebot.py` and `wk4_raspberrypi.py`, which you would use as template to write your code.
11. A Firebase account with a database and a secret token.

2 Adding Data Manually

Task
Adding data from Firebase dashboard.

Instructions:

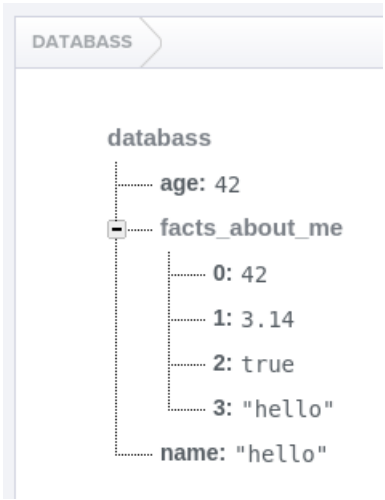
1. If you have not set up a Firebase account, please do so before proceeding. Refer to the instructions on **Firebase Set Up Guide**.
2. Open up your database dashboard in your internet browser by navigating to: https://[database name].firebaseio.com
3. Add data into your database from the dashboard.
   a. To add a new data, hover your cursor to where it says: [database name]: null.
   b. Click on the green plus icon.



   c. Key in the name of the data and the value and press add.
   d. Add the following name, value pair (You may copy and paste directly into the dashboard):

name	value
`age`	`42`
`facts_about_me`	`[42, 3.14, true, "hello"]`
`name`	`"hello"`

   e. Verify that your database has the following:



3 Reading from Database Using Python

> Task
> Retrieve data from Firebase using Python.

Instructions:

You should have created a token for your database. If you have not done so, refer to the instructions on **Firebase Set Up Guide**.

Open up `wk4_firebasebasics.py`:

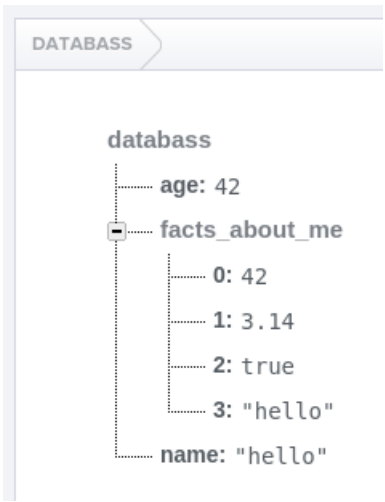
```
3  url = "Replace me" # URL to Firebase database
4  token = "Replace me" # unique token used for authentication
```

Replace the url and token in `wk4_firebasebasics.py` with yours. For example:

```
# These are dummy url and token!
url = "https://databasename.firebaseio.com"
token = "Gv7pntgzatWZSsOknJ61W2Zr5O5i0S1L4ayrLxwj"
```

```
9  firebase = firebase.FirebaseApplication(url, token)
10
11 print "Reading from my database."
12 print firebase.get('/age') # get the value from the node age
```

Line 9 creates an object that has the functions `put` and `get`. The `get(node)` function allows us to get data from the firebase database through Python. It takes in an input `node`, that specifies the node that we want to retrieve our data from. In order to understand what node is, we first need to know the data structure that is being used for the data storage.



From the figure, we can see a hierarchical structure. At the top, we have `databass`, which is the ancestor of every object in the database. An object such as `databass` is a node in the structure. `age`, `name` and `facts_about_me` are all nodes. However, `42`, `"hello"`, are not nodes. They are the values of the node `age` and `name`, respectively.

A node that is the ancestor of every other node is called a root node. We call this hierarchical structure a tree. It then makes sense that databass is the root, since it is the 'fundamental' for all other item in the database.

While `databass` is a root node, it is also a parent node. More specifically, it is the parent of the nodes `age`, `name` and `facts_about_me`. Conversely, these three nodes are the children of `databass`.

One more term that is also important is leaf. A node that has no children is a leaf node. Both `age` and `name` are leaf nodes. However, `facts_about_me` is not. This is important to note because in Firebase, only leaf nodes can have a value that is either a number (int or float), string or boolean.

This is a basic overview of the tree structure used in the Firebase database. We can now apply what we have learnt to some practical use.

Recall that the `get(node)` function retrieves the value of the node specified in the argument. In order to retrieve the value of the root node, we use the following:

```
firebase.get('/')
```

`'/'` is the root node. The node `age`, `facts_about_me` and `name` are respresented as:

node	representation
`age`	`'/age'`
`name`	`'/name'`
`facts_about_me`	`'/facts_about_me'`

If we want to get the value of a child node, the general form is `'/parent/child'`, where parent can itself be of the form `'parentof-parent/parent'`. As an example, since the node `facts_about_me` has children nodes: `0`, `1`, `2` and `3` (they are leaf nodes too), we can retrieve the leaf node `0` by:

```
firebase.get('/facts_about_me/0') # returns 42
```

Modify the `wk4_firebasebasics.py` so that you retrieve from the firebase database the value of the remaining data. The expected output of the program should be:

```
Reading from my database:
42
hello
[42, 3.14, True, u'hello']
```

4 Writing to Database Using Python

Task
Writing data to Firebase using Python.

Instructions:

1. Modify the `wk4_firebasebasics.py` to add data into the database.
   a. Modify the `wk4_firebasebasics.py` from the previous task to write the following into the database:

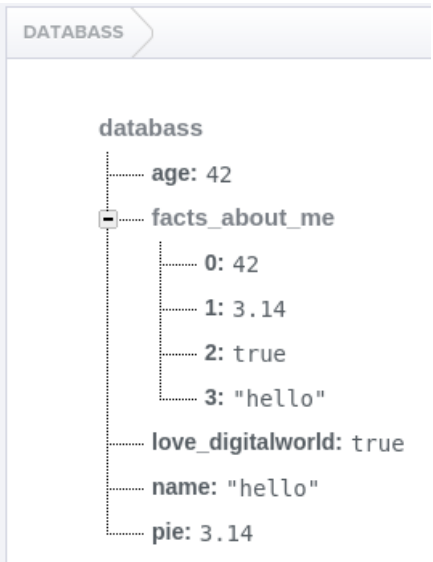
name	value
`love_digitalworld`	`True`
`pie`	`3.14`

   b. In order to add a data into an existing node in the database, you can use the `firebase.put` function. For example, if you want to add a node `pie` that has value `3.14` to the root node, you can use:

```
firebase.put('/', 'pie', 3.14)
```

   The `put` function takes in three arguments: `firebase.put(root, node, data)`. The function adds a node specified by the argument `node` with value `data` to the `root`. This makes `root` the parent of `node`.

2. Run the program and verify that your dashboard has been updated as such:



5 Move the Robot with Raspberry Pi

Task
Use the Firebase database to communicate between the eBot and Raspberry Pi.

Use the `wk4_ebot.py` file in your laptop and the `wk4_raspberrypi.py` file in your Raspberry Pi as template to write the code for this checkoff.

On the Raspberry Pi side, you should wire up four push buttons to the GPIO pins. Three of the buttons will be used to 'issue' movement commands to the eBot. The last button is used to confirm the commands and then write them to firebase. On the eBot side, the Python program should read the commands from the firebase and execute them.

More specifically:

- The Raspberry Pi should have four push buttons
  1. left rotate button: issue command to cause the eBot to rotate counter-clockwise on the spot for 1 second.
  2. forward button: issue command to cause the eBot to move forward for 1 second.
  3. right rotate button: issue command to cause the eBot to rotate clockwise on the spot for 1 second.
  4. ok button: confirms all the commands and write the commands to Firebase.
- Your program on the Raspberry Pi should use a Python list to store the movement commands selected by the push buttons before the ok button has been pressed. Once the ok button has been pressed, the program on the Raspberry Pi should send the Python list over to Firebase. On the other end, the program for the eBot should read the list of movement commands from the Firebase and execute each command in sequential order for exactly **1 second**.

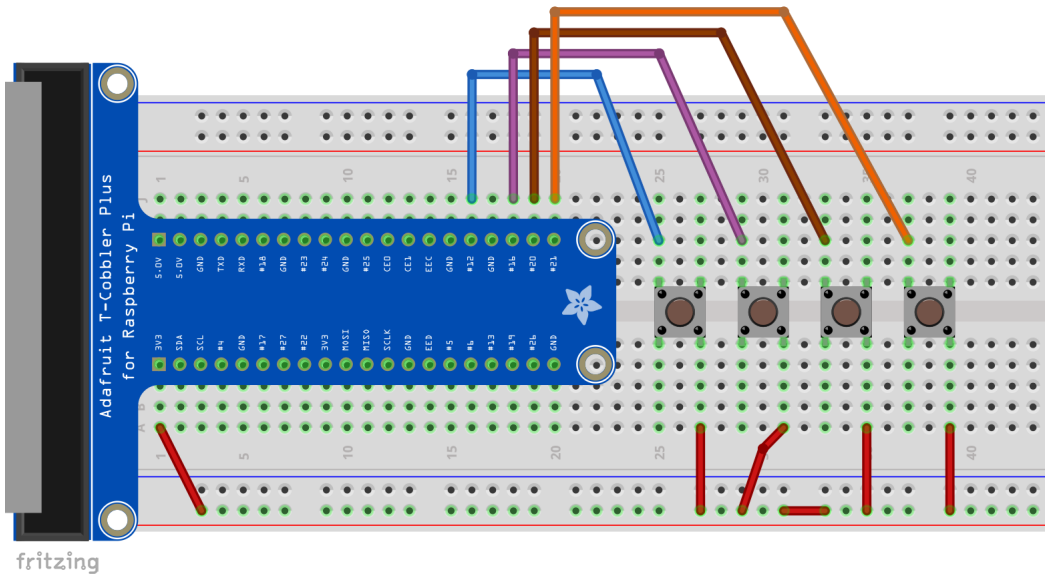
For example, if the following buttons are pressed in the given order:

1. left button
2. left button
3. forward button
4. right button
5. right button
6. ok button

The eBot will begin executing the movement commands after the ok button has been pressed. It will:

1. rotate counter-clockwise for 2 seconds,
2. move forward (in the direction that it is facing) for 1 second,
3. rotate clockwise for 2 seconds and stop.

You may refer to the following figure to wire up the the buttons.



Note

If you have already completed the code and tested it, you may find that there are instances where a single button press is being read by your program as several presses instead. When a button is pressed, it does not simply go from opened switch to closed. Instead, there may be bouncing effects, causing your program to think that the button was pressed multiple times in a very short span of time.

A simple workaround to address this issue is to introduce a small delay after a button press has been detected. You may need to import the `time` module to make use of the `sleep` function to cause the delay. Note that the delay should not be set too high (more than a second). Otherwise for those of you who have fast fingers would find that the program is not reading the number of presses correctly. Conversely, you would not want to set the delay too low (less than 0.1 seconds) as the bouncing effect may span more than 0.1 seconds and cause your program to still read a single press as multiple presses.

WARNING!
The state of the GPIOs will be retained after the program terminates. That is, if any GPIO output was HIGH at the moment the program terminates, it would remain as HIGH even after the program terminates.

Consequently, it is possible to accidentally damage the Raspberry Pi by connecting GPIOs that was HIGH directly to ground. In order to 'reset' the GPIOs, you can use gpiocleanup.py (found at **Courseware website**) and run it in your Raspberry Pi. The program sets every GPIO as input. As such, they would not get damaged even if they are connected directly to either HIGH or LOW.

Checkoff 1
Explain and demonstrate the working program to an instructor. Your program should:

1. make use of list to store the movement commands.
2. use a loop to traverse the list and execute each command sequentially.
3. use Firebase to read and write the list.
4. attempt to address the button bouncing issue.