

Student name:Tien Phat Nguyen

Student ID:223171213

SIT225: Data Capture Technologies

Activity 5.1: Firebase Realtime database

The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real-time. Data is stored as JSON and synchronized in real-time to every connected client. In this activity, you will set up and perform operations such as queries and updates on the database using Python programming language.

Hardware Required

No hardware is required.

Software Required

Firebase Realtime database
Python 3

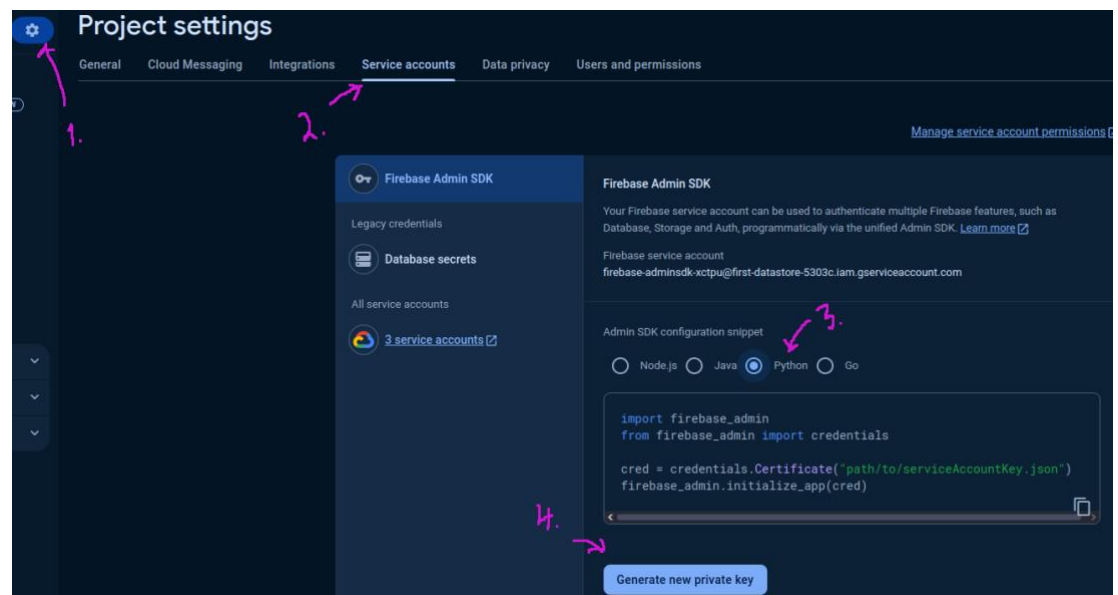
Steps

Step	Action
1	Create an Account: First, you will need to create an account in the Firebase console, follow instructions in the official Firebase document (https://firebase.google.com/docs/database/rest/start).
2	Create a Database: Follow the above Firebase document to create a database. When you click on Create Database, you have to specify the location of the database and the security rules. Two rules are available – locked mode and test mode; since we will be using the database for reading, writing, and editing, we choose test mode.
3	Setup Python library for Firebase access: We will be using Admin Database API, which is available in <i>firebase_admin</i> library. Use the below command in the command line to install. You can

follow a Firebase tutorial here (<https://www.freecodecamp.org/news/how-to-get-started-with-firebase-using-python>).

```
$ pip install firebase_admin
```

Firebase will allow access to Firebase server APIs from Google Service Accounts. To authenticate the Service Account, we require a private key in JSON format. To generate the key, go to project settings, click Generate new private key, download the file, and place it in your current folder where you will create your Python script.



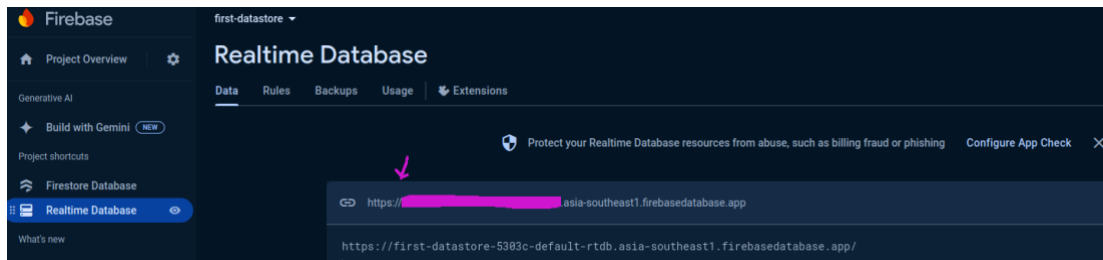
4

Connect to Firebase using Python version of Admin Database API:

A credential object needs to be created to initialise the Python library which can be done using the Python code below. Python notebook can be downloaded here (https://github.com/deakin-deep-dreamer/sit225/blob/main/week_5/firebase_explore.ipynb).

```
1 import firebase_admin
2
3 databaseURL = 'https://XXX.firebaseiodatabase.app/'
4 cred_obj = firebase_admin.credentials.Certificate(
5 | 'first-datastore-5303c-firebase-adminsdk-xctpu-c9902044ac.json'
6 | )
7 default_app = firebase_admin.initialize_app(cred_obj, {
8 | 'databaseURL': databaseURL
9 | })
```

The databaseURL is a web address to reach your Firebase database that you have created in step 2. This URL can be found in the Data tab of Realtime Database.



If you compile the code snippet above, it should do with no error.

5

Write to database Using the set() Function:

We set the reference to the root of the database (or we could also set it to a key value or child key value). Data needs to be in JSON format as below.

```

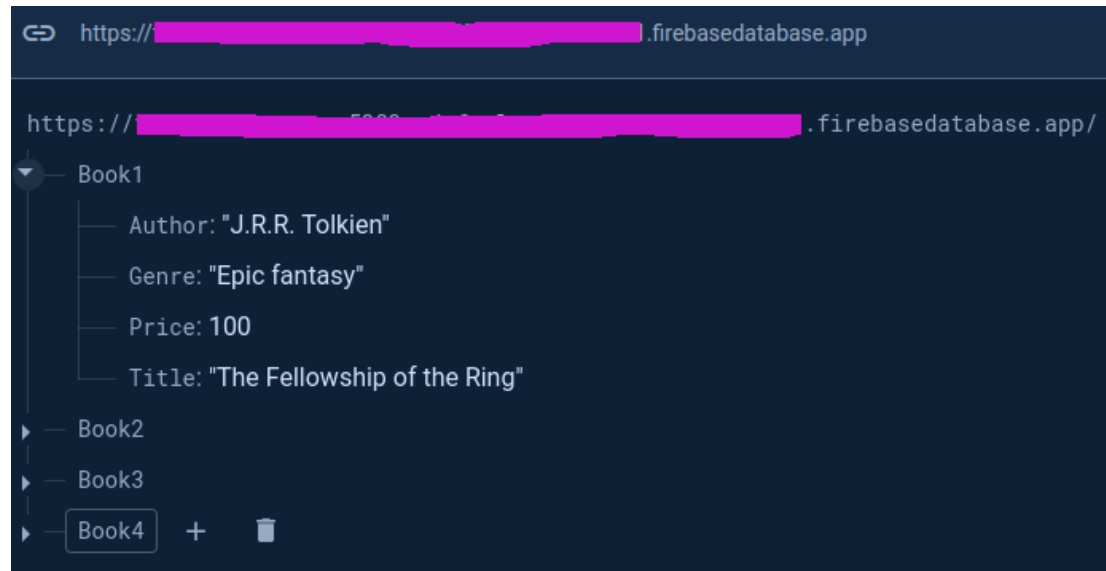
1  from firebase_admin import db
2
3  # A reference point is always needed to be set
4  # before any operation is carried out on a database.
5  #
6  ref = db.reference("/")
7
8  # JSON format data (key/value pair)
9  data = { # Outer {} contains inner data structure
10     "Book1":
11         {
12             "Title": "The Fellowship of the Ring",
13             "Author": "J.R.R. Tolkien",
14             "Genre": "Epic fantasy",
15             "Price": 100
16         },
17     "Book2":
18         {
19             "Title": "The Two Towers",
20             "Author": "J.R.R. Tolkien",
21             "Genre": "Epic fantasy",
22             "Price": 100
23         },
24     "Book3":
25         {
26             "Title": "The Return of the King",
27             "Author": "J.R.R. Tolkien",
28             "Genre": "Epic fantasy",
29             "Price": 100
30         },
31     "Book4":
32         {
33             "Title": "Brida",
34             "Author": "Paulo Coelho",
35             "Genre": "Fiction",
36             "Price": 100
37         }
38 }
39
40 # JSON format data is set (overwritten) to the reference
41 # point set at /, which is the root node.
42 #
43 ref.set(data)

```

A reference point always needed to be set where the data read/write will take place. In the code above, the reference point is set at the root of the NoSQL Document, where consider the database is a JSON tree and / is the root node

of the tree). The set() function writes (overwrites) data at the set reference point.

You can visualise the data in the Firebase console as below -



6 Read data using get() function:

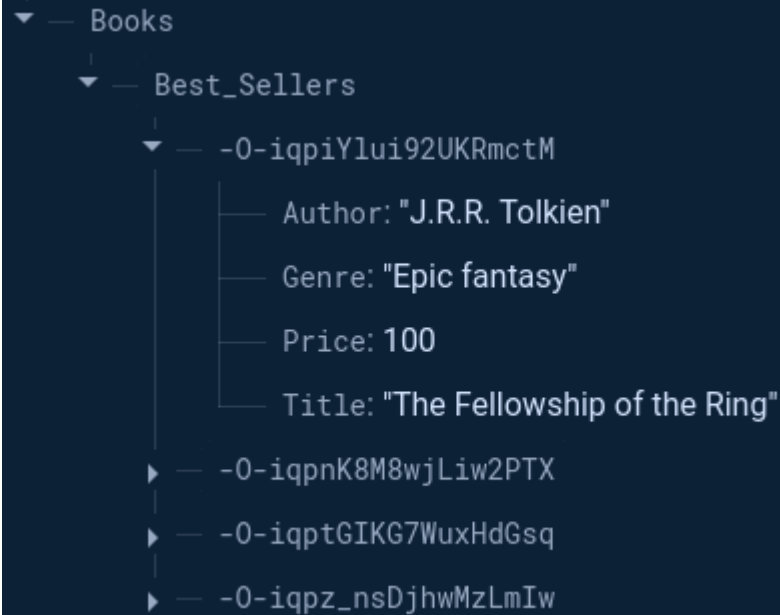
Data can be read using get() function on the reference set beforehand, as shown below.

```
1  ref = db.reference("/") # set ref point
2
3  # query all data under the ref
4  books = ref.get()
5  print(books)
6  print(type(books))
7
8  # print each item separately
9  for key, value in books.items():
10 |     print(f"{key}: {value}")
11
12
13 # Query /Book1
14 ref = db.reference("/Book1")
15 books = ref.get()
16 print(books)
```

✓ 0.3s

```
{'Book1': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'},
'Book2': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'},
'Book3': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'},
'Book4': {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Price': 100, 'Title': 'Brida'}}
<class 'dict'>
Book1: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'}
Book2: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'}
Book3: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'}
Book4: {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Price': 100, 'Title': 'Brida'}
```

	Consider the reference set in line 1 and the output compared to the reference set at line 14 and the bottom output line to understand the use of db.reference() and ref.get().
7	<p>Write to database Using the push() Function:</p> <p>The push() function saves data under a <i>unique system generated key</i>. This is different than set() where you set the keys such as Book1, Book2, Book3 and Book4 under which the content (author, genre, price and title) appears. Let's try to push the same data in the root reference. Note that since we already has data under root / symbol, setting (or pushing) in the same reference point will eventually rewrite the original data.</p> <pre>1 # Write using push() function 2 # Note that a set() is called on top of push() 3 # 4 ref = db.reference("/") 5 ref.set({ 6 "Books": 7 { 8 "Best_Sellers": -1 9 } 10 }) 11 12 ref = db.reference("/Books/Best_Sellers") 13 14 for key, value in data.items(): 15 ref.push().set(value)</pre> <p>✓ 2.0s</p> <p>The output will reset the previous data set in / node. The current data is shown below.</p>



As you can see, under /Books/Best_Sellers there are 4 nodes where the node head (or node ID) is a randomly generated key which is due to the use of push() function. When data key does not matter, the use of push() function is desirable.

8

Update data:

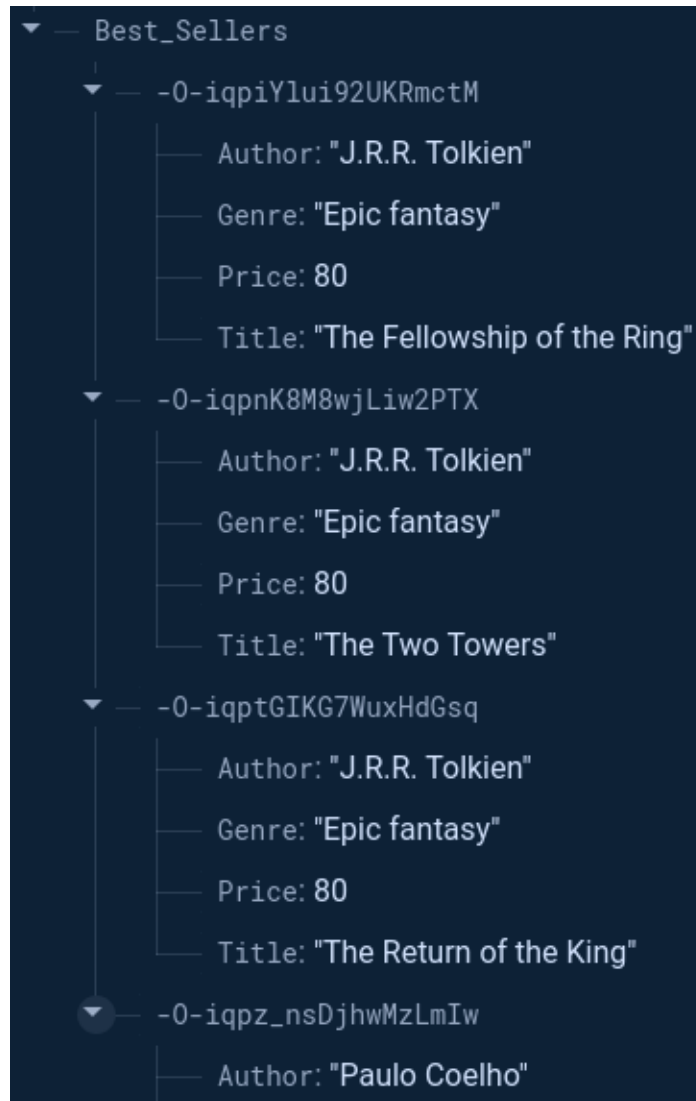
Let's say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.

```
1 # Update data
2 #
3 # Requirement: The price of the books by
4 # J. R. R. Tolkien is reduced to 80 units to
5 # offer a discount.
6 #
7 ref = db.reference("/Books/Best_Sellers/")
8 best_sellers = ref.get()
9 print(best_sellers)
10 for key, value in best_sellers.items():
11     if(value["Author"] == "J.R.R. Tolkien"):
12         value["Price"] = 90
13         ref.child(key).update({"Price":80})
```

✓ 0.9s

As you can see, the author name is compared and the new price is set in the best_sellers dictionary and finally, an update() function is called on the ref, however, the current ref is a '/Books/Best_Sellers/', so we need to locate the

child under the ref node, so `ref.child(key)` is used in line 13. The output is shown below with a discounted price.



9

Delete data:

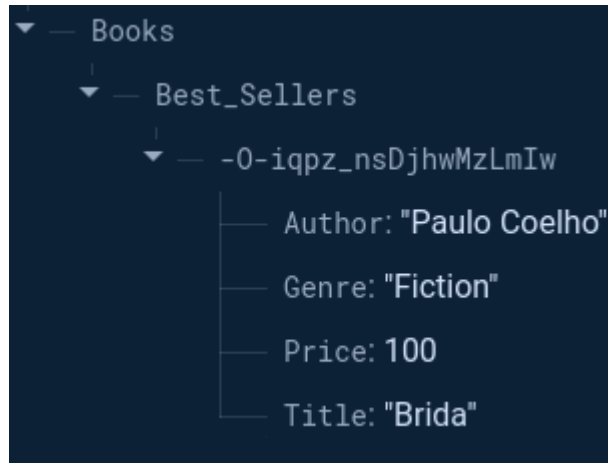
Let's delete all bestseller books with J.R.R. Tolkien as the author. You can locate the node using `db.reference()` (line 4) and then locate specific record (for loop in line 6) and calling `set()` with empty data `{}` as a parameter, such as `set({})`. The particular child under the ref needs to be located first by using `ref.child(key)`, otherwise, the ref node will be removed – **BE CAREFUL**.

```

1 # Let's delete all best seller books
2 # with J.R.R. Tolkien as the author.
3 #
4 ref = db.reference("/Books/Best_Sellers")
5
6 for key, value in best_sellers.items():
7     if(value["Author"] == "J.R.R. Tolkien"):
8         ref.child(key).set({})

```

This keeps only the other author data, as shown below.



If ref.child() not used, as shown the code below, all data will be removed.

```

1 ref = db.reference("/Books/Best_Sellers")
2 ref.set({})

```

Now in Firebase console you will see no data exists.

10 **Question:** Run all the cells in the Notebook you have downloaded in Step 4, fill in the student information at the top cell of the Notebook. Convert the Notebook to PDF and merge with this activity sheet PDF.

Answer: Convert the Notebook to PDF and merge with this activity sheet PDF.

10 **Question:** Create a sensor data structure for DHT22 sensor which contains attributes such as sensor_name, timestamp, temperature and humidity. Remember there will be other sensors with different sensor variables such as DHT22 has 2 variables, accelerometer sensor has 3. For each such sensor, you will need to gather data over time. Discuss how you are going to handle multiple data values in JSON format? Justify your design.

Answer:

	 <pre> sensor_data_structure = { "sensors": { "DHT22_001": { "sensor_type": "DHT22", "location": "Room_A", "readings": { } }, "SR04_001": { "sensor_type": "SR04_Ultrasonic", "location": "Door_A", "readings": { } } } } dht22_reading = { "timestamp": "2025-08-19T10:30:00Z", "temperature": 25.6, "humidity": 60.2, "sensor_id": "DHT22_001" } sr04_reading = { "timestamp": "2025-08-19T10:30:00Z", "distance": 150.5, "sensor_id": "SR04_001" } </pre> <p>I choose an array of readings so it's easy to append new values and iterate over them later in code, and each sensor keeps its own list. We can query data by sensor or by time, and adding new sensor types with different variables just means adding fields in each reading without changing the structure.</p>
11	<p>Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here.</p> <p>Answer:</p>

	<pre> ... 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:03.990093', 'temperature': 19.0, 'humidity': 75.8, 'sensor_id': 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:04.292875', 'temperature': 23.7, 'humidity': 48.2, 'sensor_id': 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:04.598769', 'temperature': 29.3, 'humidity': 56.8, 'sensor_id': 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:04.908346', 'temperature': 19.7, 'humidity': 63.6, 'sensor_id': 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:05.189601', 'temperature': 23.7, 'humidity': 61.1, 'sensor_id': 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:05.489806', 'temperature': 32.6, 'humidity': 34.9, 'sensor_id': 'DHT22_001'} Inserted: {'timestamp': '2025-08-19T13:44:05.815660', 'temperature': 32.6, 'humidity': 59.9, 'sensor_id': 'DHT22_001'} === All DHT22 Readings === Key: -0Y-QJfImE-Mm0cj0j0, Data: {'humidity': 33.9, 'sensor_id': 'DHT22_001', 'temperature': 29.4, 'timestamp': '2025-08-19T13:44:02.963446'} Key: -0Y-QJKmMLfNIiz13yKd, Data: {'humidity': 50.5, 'sensor_id': 'DHT22_001', 'temperature': 27.8, 'timestamp': '2025-08-19T13:44:03.313964'} Key: -0Y-QJQQzVwWfLLko2nW, Data: {'humidity': 77.7, 'sensor_id': 'DHT22_001', 'temperature': 20.0, 'timestamp': '2025-08-19T13:44:03.680461'} Key: -0Y-QJV7D7V13JCOCVnn, Data: {'humidity': 75.8, 'sensor_id': 'DHT22_001', 'temperature': 19.0, 'timestamp': '2025-08-19T13:44:03.990093'} Key: -0Y-QJZxRchpfUK-2jSv, Data: {'humidity': 48.2, 'sensor_id': 'DHT22_001', 'temperature': 23.7, 'timestamp': '2025-08-19T13:44:04.292875'} Key: -0Y-QJdg4I3zVHvkmkqd, Data: {'humidity': 56.8, 'sensor_id': 'DHT22_001', 'temperature': 29.3, 'timestamp': '2025-08-19T13:44:04.598769'} Key: -0Y-QJiY-dEbopr8PekN, Data: {'humidity': 63.6, 'sensor_id': 'DHT22_001', 'temperature': 19.7, </pre>
--	--

12 **Question:** Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here.

Answer:

```

... 'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:27.951278', 'distance': 375.6, 'sensor_id': 'SR04_001', 'unit':
'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:28.245117', 'distance': 397.7, 'sensor_id': 'SR04_001', 'unit':
'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:28.542956', 'distance': 351.1, 'sensor_id': 'SR04_001', 'unit':
'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:28.858855', 'distance': 384.1, 'sensor_id': 'SR04_001', 'unit':
'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:29.186648', 'distance': 229.7, 'sensor_id': 'SR04_001', 'unit':
'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:29.552591', 'distance': 307.7, 'sensor_id': 'SR04_001', 'unit':
'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:29.880580', 'distance': 21.5, 'sensor_id': 'SR04_001', 'unit': 'cm'}
Inserted: {'timestamp': '2025-08-19T13:44:30.184672', 'distance': 335.9, 'sensor_id': 'SR04_001', 'unit':
'cm'}

=== All SR04 Readings ===
Key: -0Y-QP5r-qh-uexeCyuj, Data: {'distance': 165.4, 'sensor_id': 'SR04_001', 'timestamp': '2025-08-
19T13:44:26.936162', 'unit': 'cm'}
Key: -0Y-QPE95990gJn00YfD, Data: {'distance': 337.5, 'sensor_id': 'SR04_001', 'timestamp': '2025-08-
19T13:44:27.472001', 'unit': 'cm'}
Key: -0Y-QPLgdyPUT2HmInG, Data: {'distance': 375.6, 'sensor_id': 'SR04_001', 'timestamp': '2025-08-
19T13:44:27.951278', 'unit': 'cm'}
Key: -0Y-QPQDAWKZCpCzj7li, Data: {'distance': 397.7, 'sensor_id': 'SR04_001', 'timestamp': '2025-08-
19T13:44:28.245117', 'unit': 'cm'}
Key: -0Y-QPUon5PL7s11k79b, Data: {'distance': 351.1, 'sensor_id': 'SR04_001', 'timestamp': '2025-08-
19T13:44:28.542956', 'unit': 'cm'}

```

13	<p>Question: Firebase Realtime database generates events on data operations. You can refer to section 'Handling Realtime Database events' in the document (https://firebase.google.com/docs/functions/database-events?gen=2nd). Discuss in the active learning session and summarise the idea of database events and how it is handled using Python SDK.</p> <p>Note that these events are useful when your sensors (from Arduino script) store data directly to Firebase Realtime database and you would like to track data update actions from a central Python application such as a monitoring dashboard.</p> <p>Answer: <Your answer></p> <p>Firebase Realtime Database can fire events like <code>on_create</code>, <code>on_update</code>, <code>on_delete</code> whenever data changes, and with the Python SDK I can define functions that subscribe to these events. <code>ref.listen</code> registers a callback that receives an event object with <code>event_type</code> and <code>data</code> property holding the new or changed value. This lets a Python dashboard track Arduino sensor updates in real time, so I can plot charts or trigger alerts as soon as new readings arrive.</p>

Activity 5.2: Data wrangling

Data wrangling is the process of converting raw data into a usable form. The process includes collecting, processing, analyzing, and tidying the raw data so that it can be easily read and analyzed. In this activity, you will use the common library in python, "pandas".

Hardware Required

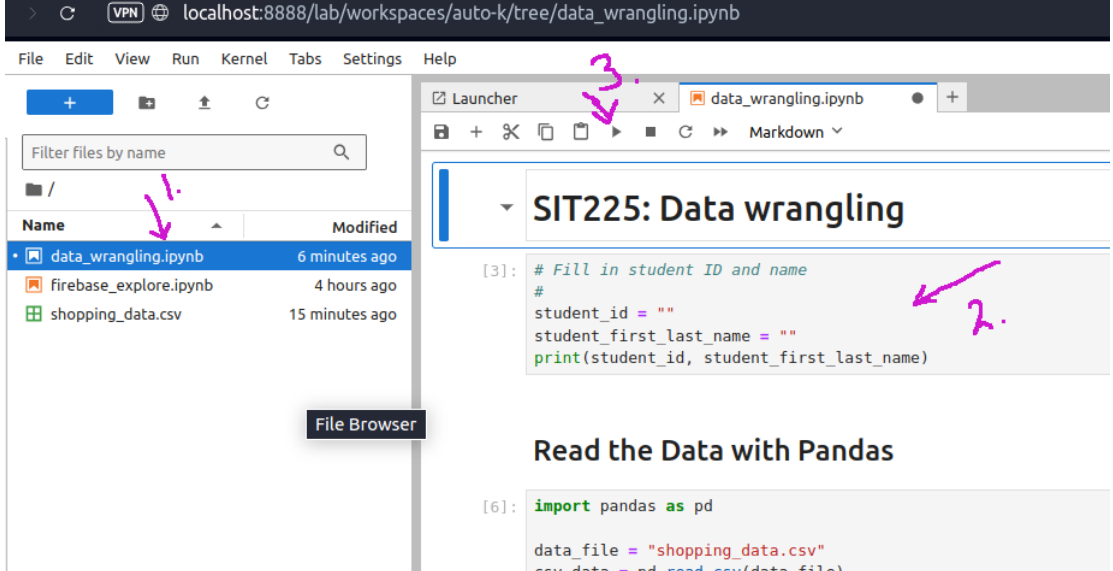
No hardware is required.

Software Required

Python 3
Pandas Python library

Steps

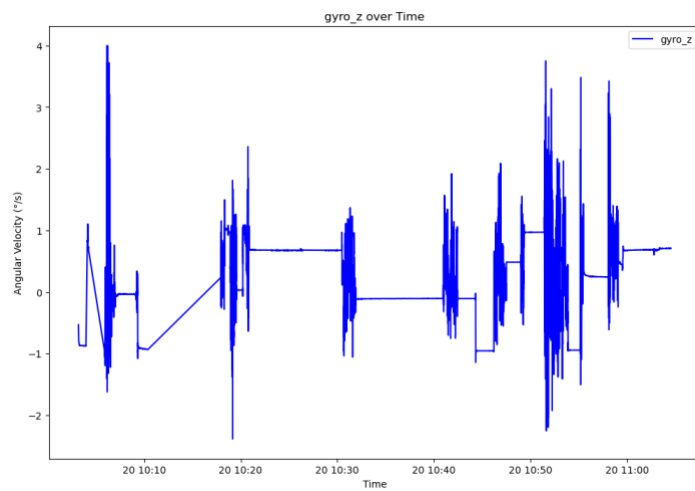
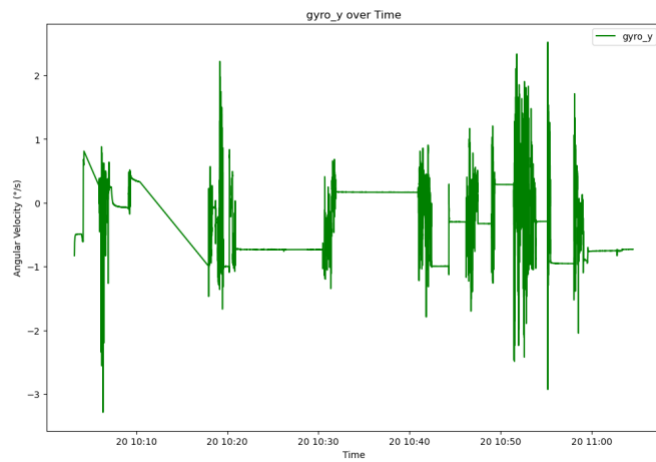
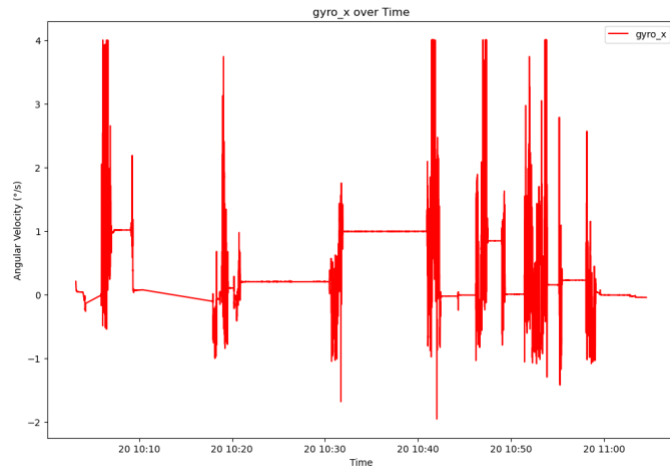
Step	Action
1	<p>Install Pandas using the command below. Most likely you already have Pandas installed if you have installed Python using Anaconda distribution (https://www.anaconda.com/download).</p> <pre>\$ pip install pandas</pre> <p>A Python notebook is shared in the GitHub link (https://github.com/deakin-deep-dreamer/sit225/tree/main/week_5). There will be a data_wrangling.ipynb, shopping_data.csv and shopping_data_missingvalue.csv files among others. Download the week_5 folder in your computer, open a command prompt in that folder, and write the command below in the command line:</p> <pre>\$ jupyter lab</pre> <p>This will open Python Jupyter Notebook where in the left panel you can see the files (labeled as 1 in figure).</p>

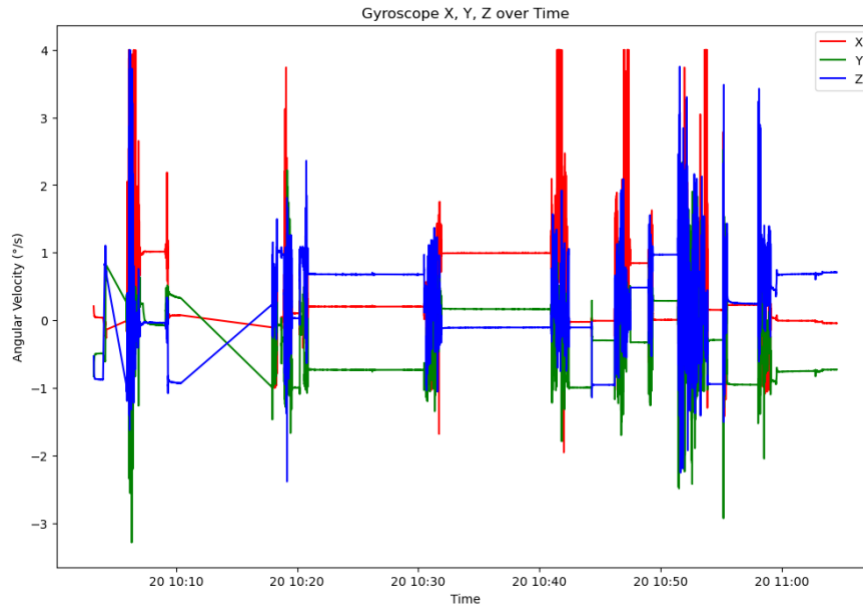
	 <p>Each cell contains Python code (labeled as 2 in figure), you can run a cell by clicking on the cell, so the cursor appears in that cell and then click on the play button at the top of the panel (labeled as 3 in the figure).</p>
2	<p>Question: Run each cell to produce output. Follow instructions in the notebook to complete codes in some of the cells. Convert the notebook to PDF from menu File > Save and Export Notebook As > PDF. Convert this activity sheet to PDF and merge with the notebook PDF.</p> <p>Answer: There is no answer to write here. You have to answer in the Jupyter Notebook.</p>
3	<p>Question: Once you went through the cells in the Notebook, you now have a basic understanding of data wrangling. Pandas are a powerful tool and can be used for reading CSV data. Can you use Pandas in reading sensor CSV data that you generated earlier? Describe if any modification you think necessary?</p> <p>Answer: I use a command like <code>df = pd.read_csv('sensor_data.csv', parse_dates=['timestamp'])</code> which tells the computer that timestamp column contains dates and times. I make the timestamp as main index which makes it much easier to create graphs or group your data by time periods later</p>
4	<p>Question: What do you understand of the Notebook section called Handling Missing Value? Discuss in group and briefly summarise different missing value imputation methods and their applicability on different data conditions.</p>

Answer: <Your answer>

Handling missing values in pandas means first you detect them then you can drop rows or columns with `df.dropna()`, or fill gaps with simple strategies: use `df.fillna(df.mean())` if the missingness is random and numeric, or `df.fillna(method='ffill')` when readings are sequential and you assume last value persists.

Pass Task





Q2: The experiment involves a 30-minute test with periods of staying still, slow turns, and regular spinning movements of 45 degrees that happen 1 to 2 times each second. During the spinning movements, the rotation speed data should show a wave pattern on two different measurement directions. The main spinning direction should show a clear up and down pattern, while the other directions should stay nearly flat unless the device is tilted. This matches how the measuring device works because it records how fast something spins in three different directions. The test results support our prediction, though small differences might happen because of tiny measurement mistakes or timing issues during recording.

According to the LSM6DS3 datasheet, the gyroscope supports multiple sample rates including 12.5, 26, 52, 104, 208, 416, 833, and 1666 Hz. For most hand movement applications, we decided to keep this 9600 baud rate as it provides good accuracy for detecting motion. However, we set the data reading interval to 50 milliseconds (equivalent to 20 Hz sampling) to reduce the load on both serial communication and Firebase storage. This 20 Hz frequency is sufficient to capture hand rotation movements, which typically occur below 10 Hz, while also saving bandwidth and reducing data storage costs.


```

#include "thingProperties.h"
#include <Arduino_LSM6DS3.h>

// Timing variables
unsigned long previousMillis = 0;
const unsigned long interval = 50;

void setup() {
  Serial.begin(9600);
  delay(1500);

  // initialize sensor
  if (!IMU.begin()) {
    Serial.println("Failed to initialize LSM6DS3 IMU!");
    while (1);
  }

  Serial.print("LSM6DS3 Accelerometer sample rate = ")
  Serial.print(IMU.accelerationSampleRate());
  Serial.println(" Hz");

  // initialize arduino cloud properties
  initProperties();

  // connect to arduino iot cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  Serial.println("LSM6DS3 Cloud Logger Started!");
}

```

The sketch starts serial communication, waits briefly, then initializes the LSM6DS3 IMU; if initialization fails, it prints an error and stops, otherwise it prints the current accelerometer sample rate in hertz to the Serial Monitor. Next, it loads Arduino IoT Cloud properties, establishes a cloud connection, sets the debug level, prints connection diagnostics, and confirms that the cloud logger has started.

```

void loop() {
  // maintain cloud connection
  ArduinoCloud.update();

  unsigned long currentMillis = millis();

  // read sensor data at specified interval
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    float x, y, z;

    // check if new acceleration data is available
    if (IMU.accelerationAvailable()) {

      IMU.readAcceleration(x, y, z);

      // update cloud variables
      accel_x = x;
      accel_y = y;
      accel_z = z;

      // print to Serial Monitor for debugging
      Serial.print("X=");
      Serial.print(x, 3);
      Serial.print(", Y=");
      Serial.print(y, 3);
      Serial.print(", Z=");
      Serial.println(z, 3);
    }
  }
}

```

The sketch keeps the device connected to the cloud by calling `ArduinoCloud.update()` every loop, then uses `millis()` with `previousMillis` and `interval` to perform timing. At each interval, it checks if new accelerometer data is available, reads `x`, `y`, `z`, updates the cloud variables, and prints the values to the Serial Monitor for debugging.

```

import serial
import time
import json
import firebase_admin
from firebase_admin import credentials, db

databaseURL = 'https://iot-db-27d22-default-rtdb.asia-southeast1.firebaseio.com/'
cred_obj = firebase_admin.credentials.Certificate(
    'iot-db-27d22-firebase-adminsdk-fbsvc-43adb56347.json'
)
default_app = firebase_admin.initialize_app(cred_obj, {
    'databaseURL': databaseURL
})

ref = db.reference('/gyroscope')

ser = serial.Serial('/dev/cu.usbmodem11101', 9600, timeout=1)
time.sleep(2)

try:
    while True:
        line = ser.readline().decode().strip()
        # print("Received:", line)
        parts = line.split(',')
        if len(parts) != 3:
            continue
        x = float(parts[0].split('=')[1])
        y = float(parts[1].split('=')[1])
        z = float(parts[2].split('=')[1])
        timestamp = time.time()
        data = {
            'timestamp': timestamp,
            'gyro_x': x,
            'gyro_y': y,
            'gyro_z': z
        }
        ref.push(data)
        print("Pushed:", data)
except KeyboardInterrupt:
    ser.close()
    print("Stopped data collection.")

```

The script loads Firebase Admin credentials, initializes the app with the databaseURL, and obtains a reference to the /gyroscope path in the Realtime Database. It opens the serial port at /dev/cu.usbmodem11101 with 9600 baud and a 1-second timeout, then reads each incoming line safely using readline in a continuous loop. For each valid line, it parses gyro x, y, and z values, adds a timestamp, and pushes the record as a new child under the database reference until interrupted, after which the serial port is closed.

```

import csv
from firebase_admin import db

ref = db.reference('/gyroscope')
all_data = ref.get()

with open('gyro_data.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['timestamp', 'gyro_x', 'gyro_y', 'gyro_z'])
    for key, sample in all_data.items():
        writer.writerow([
            sample.get('timestamp'),
            sample.get('gyro_x'),
            sample.get('gyro_y'),
            sample.get('gyro_z')
        ])
print("Data saved to gyro_data.csv")

```

The program connects to Firebase Realtime Database, creates a reference to the '/gyroscope' path, and uses the get() method to read all the data at that location into Python. It opens a file named gyro_data.csv in write mode, creates a CSV writer, and writes a header row for the columns timestamp, gyro_x, gyro_y, and gyro_z. Then it loops through each record from the database and writes those fields as CSV rows, and finally prints a message confirming the save is complete.

```

import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('gyro_data.csv')
df = df.dropna()

df = df[pd.to_numeric(df['gyro_x'], errors='coerce').notnull()]
df.to_csv('gyro_data_clean.csv', index=False)

df = pd.read_csv('gyro_data_clean.csv', parse_dates=['timestamp'])
df['ts'] = pd.to_datetime(df['timestamp'], unit='s')

```

The program reads the gyro_data.csv file into a DataFrame, drops any rows with missing values, and keeps only rows where the gyro_x column can be safely converted to numeric values before saving this cleaned result to gyro_data_clean.csv. Next, it loads the cleaned file, creates a datetime column in seconds.

```
for var, color in zip(['gyro_x', 'gyro_y', 'gyro_z'], ['r', 'g', 'b']):
    plt.figure(figsize=(12, 8))
    plt.plot(df['ts'], df[var], label=var, color=color)
    plt.legend()
    plt.title(f"{var} over Time")
    plt.xlabel("Time"); plt.ylabel("Angular Velocity (°/s)")
    plt.show()

# Đồ thị chung
plt.figure(figsize=(12, 8))
plt.plot(df['ts'], df['gyro_x'], 'r', label='X')
plt.plot(df['ts'], df['gyro_y'], 'g', label='Y')
plt.plot(df['ts'], df['gyro_z'], 'b', label='Z')
plt.title("Gyroscope X, Y, Z over Time")
plt.xlabel("Time"); plt.ylabel("Angular Velocity (°/s)")
plt.legend()
plt.show()
```

The code first creates three separate plots for gyro_x, gyro_y, and gyro_z over time using plt.plot,. It then makes a combined plot that draws all three lines on the same axes with different colors.

Video: <https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=83f78379-857e-4202-a778-b3490088a88e>

Github:

https://github.com/tienphatnguyendev/SIT_225_2025_T2/tree/main/week-5

