SIT225 Data Capture Technologies

Pass Task: Store data to cloud

Overview

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.

Hardware Required

i. Arduino Nano 33 IoT device, ii. USB cable, iii. LSM6DS3 module on the Arduino Nano 33 IoT for Gyroscope data.

Software Required

Python 3.

Pre-requisites: You must do the following before this task

Week 5 activities in the unit site.

Task Objective

In this week, you have learned how to use Firebase real-time database and perform operations such as add/update/query data and listen to live changes to your data. In this task, you will need to record Gyroscope readings from the built-in module of Arduino Nano board and pass data to a Python script through Serial communication (or through Arduino IoT Cloud variable synchronisation with Python). The Python script upload data to Firebase real-time database as soon as it receives any. After enough data is stored to capture interesting patterns, for no less than 10 minutes, query the database for data, format it to save in a CSV file and analyse the data. *You may reuse this data next week, so keep your CSV file handy*.

a. ww

Submission details

Q1. Perform week 5 activities mentioned in the unit site and produce outputs.

Student name:

Student ID:

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

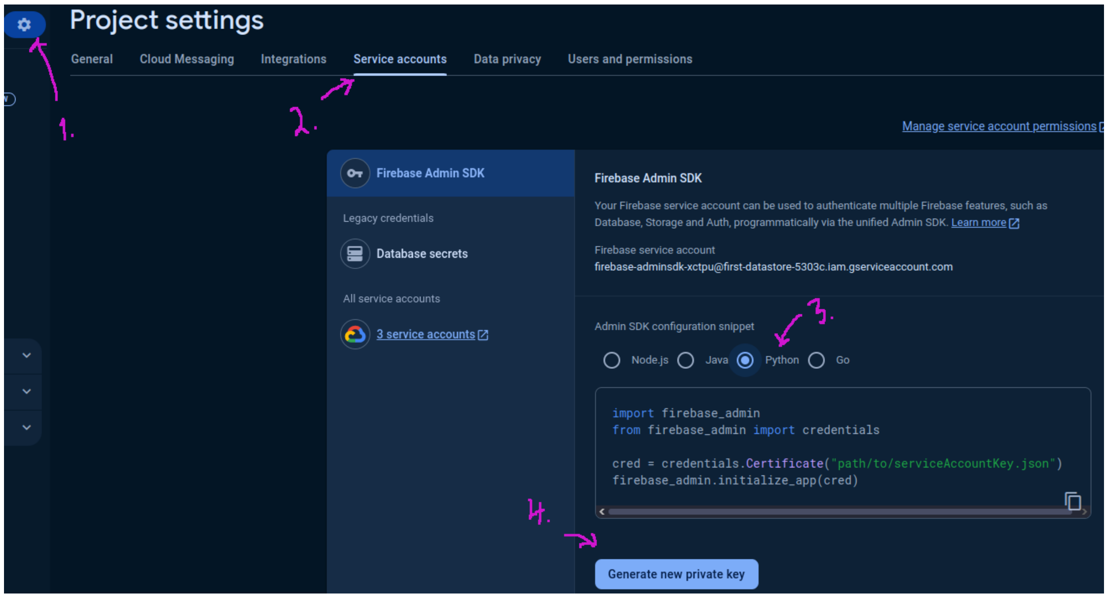
No hardware is required.

Software Required

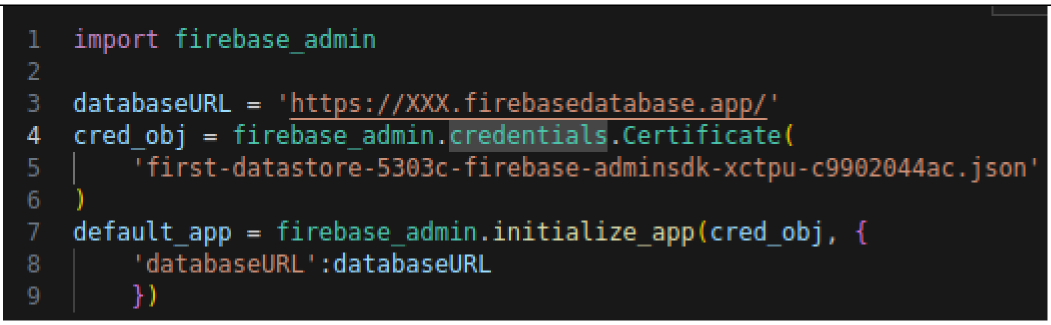
Firebase Realtime database
Python 3

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**:

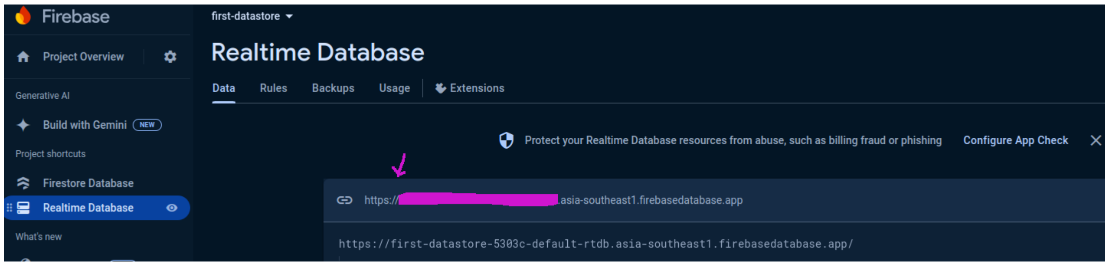
	We will be using Admin Database API, which is available in *firebase_admin* library. Use the below command in the command line to install. You can follow a Firebase tutorial here (https://www.freecodecamp.org/news/how-to-get-started-with-firebase-using-python).
	
	`$ pip install firebase_admin`
	
	Firebase 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.
	
	 Project settings
	General
	Cloud Messaging
	Integrations
	Service accounts
	Data privacy
	Users and permissions
	1.
	2.
	Manage service account permissions
	Firebase Admin SDK
	Legacy credentials
	Database secrets
	All service accounts
	3 service accounts
	3.
	4.
	Generate new private key
	Firebase Admin SDK
	Your Firebase service account can be used to authenticate multiple Firebase features, such as Database, Storage and Auth, programmatically via the unified Admin SDK. Learn more
	Firebase service account
	firebase-adminsdk-xctpu@first-datastore-5303c.iam.gserviceaccount.com
	Admin SDK configuration snippet
	Node.js Java [✓ Python] Go
	`import firebase_admin`
	`from firebase_admin import credentials`
	
	`cred = credentials.Certificate("path/to/serviceAccountKey.json")`
	`firebase_admin.initialize_app(cred)`

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).



```python
import firebase_admin

databaseURL = 'https://XXX.firebasedatabase.app/'
cred_obj = firebase_admin.credentials.Certificate(
    'first-datastore-5303c-firebase-adminsdk-xctpu-c9902044ac.json'
)
default_app = firebase_admin.initialize_app(cred_obj, {
    'databaseURL':databaseURL
    })
```

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.



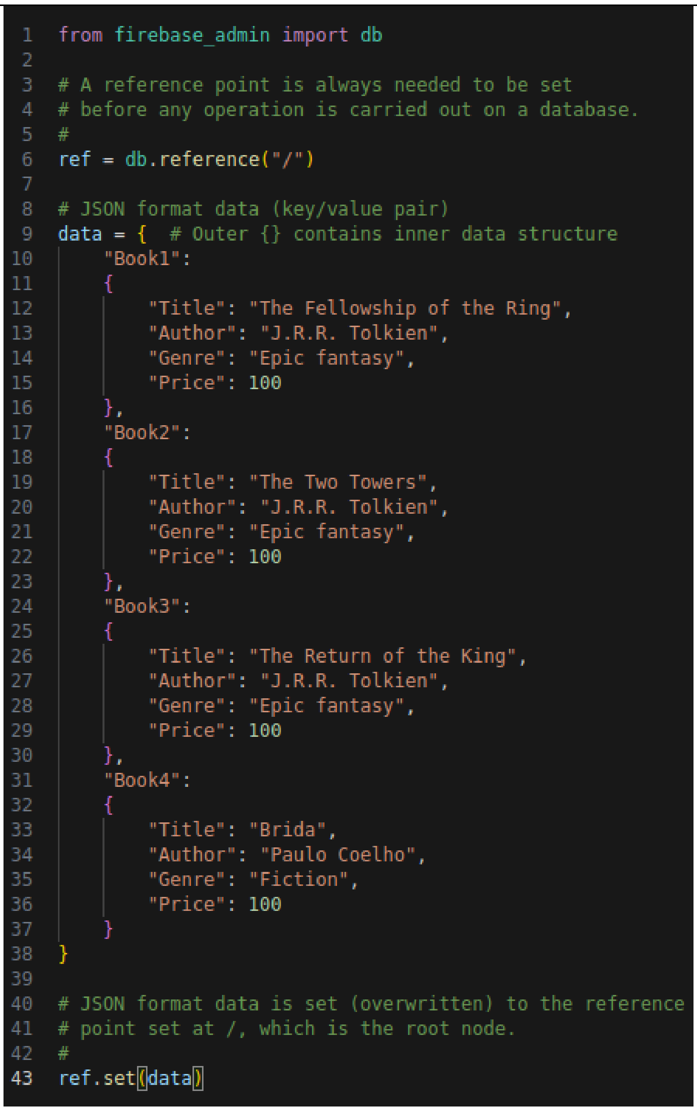
Protect your Realtime Database resources from abuse, such as billing fraud or phishing

https://[redacted].asia-southeast1.firebasedatabase.app

```
https://first-datastore-5303c-default-rtdb.asia-southeast1.firebasedatabase.app/
```

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

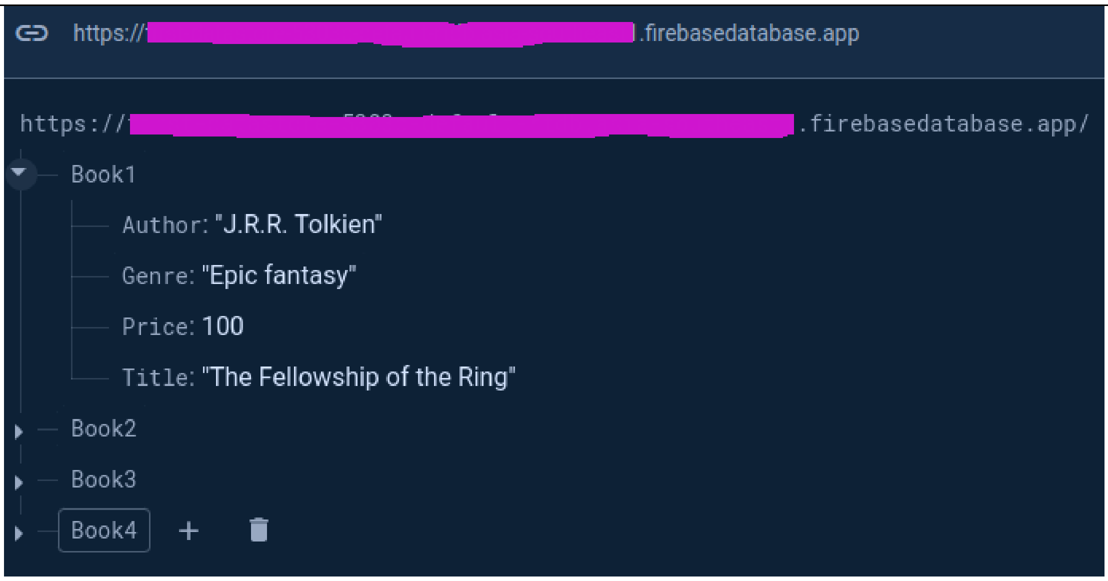
5 **Write to database Using the set() Function**:
We set the reference to the root of the database (or we could also set it to a key value or child key value). Data needs to be in JSON format as below.



```python
from firebase_admin import db

# A reference point is always needed to be set
# before any operation is carried out on a database.
#
ref = db.reference("/")

# JSON format data (key/value pair)
data = {  # Outer {} contains inner data structure
    "Book1":
    {
        "Title": "The Fellowship of the Ring",
        "Author": "J.R.R. Tolkien",
        "Genre": "Epic fantasy",
        "Price": 100
    },
    "Book2":
    {
        "Title": "The Two Towers",
        "Author": "J.R.R. Tolkien",
        "Genre": "Epic fantasy",
        "Price": 100
    },
    "Book3":
    {
        "Title": "The Return of the King",
        "Author": "J.R.R. Tolkien",
        "Genre": "Epic fantasy",
        "Price": 100
    },
    "Book4":
    {
        "Title": "Brida",
        "Author": "Paulo Coelho",
        "Genre": "Fiction",
        "Price": 100
    }
}

# JSON format data is set (overwritten) to the reference
# point set at /, which is the root node.
#
ref.set(data)
```

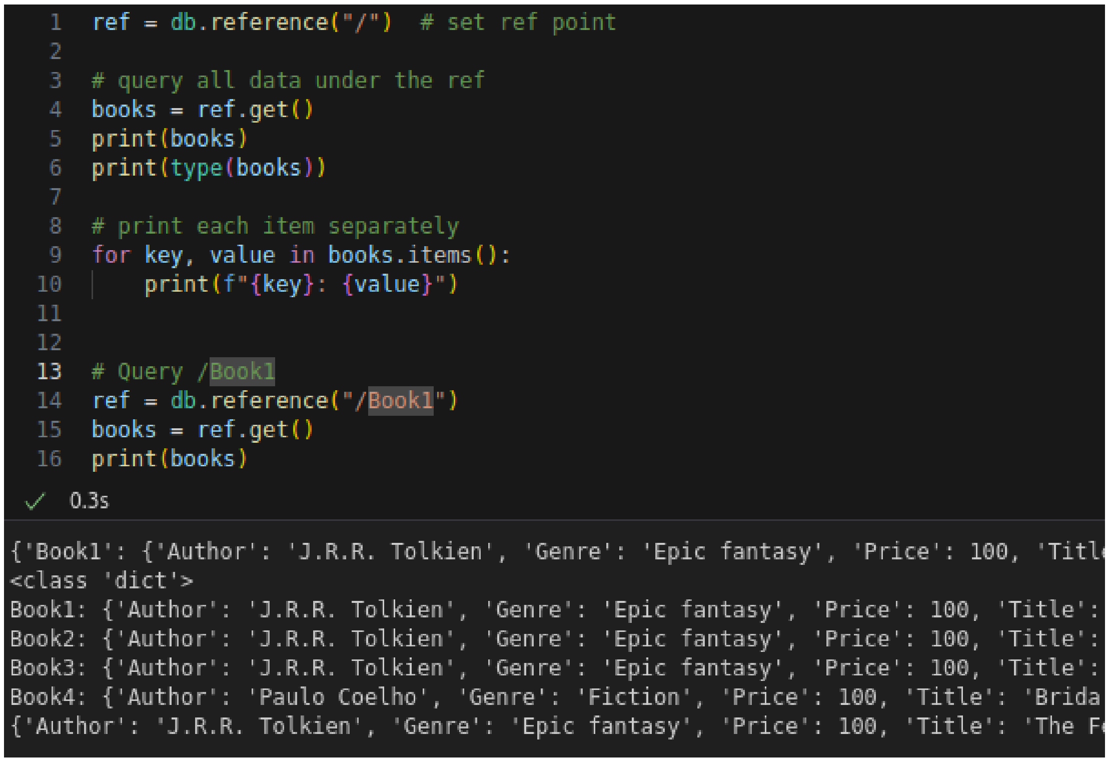
A reference point always needed to be set where the data read/write will take place. In the code above, the reference point is set at the root of the NoSQL Document, where consider the database is a JSON tree and / is the root node of the tree). The set() function writes (overwrites) data at the set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

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



```
https://[redacted].firebasedatabase.app

https://[redacted].firebasedatabase.app/
▾ Book1
    Author: "J.R.R. Tolkien"
    Genre: "Epic fantasy"
    Price: 100
    Title: "The Fellowship of the Ring"
▸ Book2
▸ Book3
▸ Book4  +  
```

6 **Read data using get() function**:
Data can be read using get() function on the reference set beforehand, as shown below.



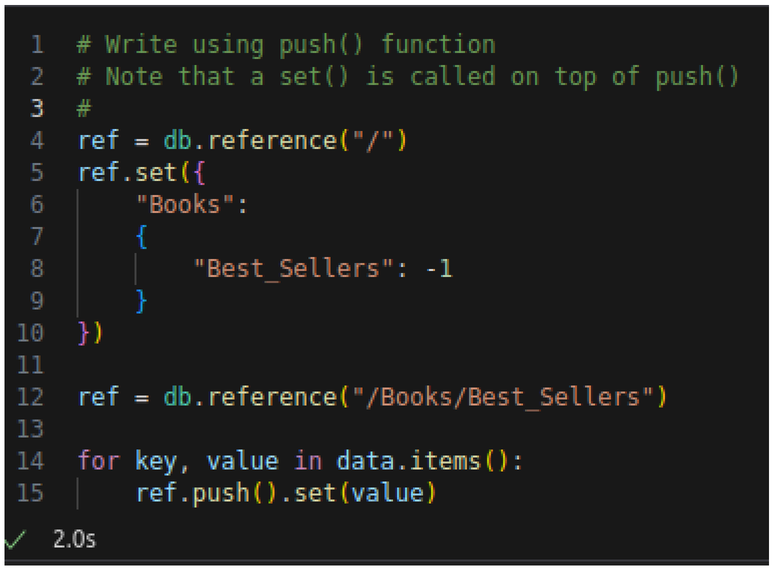
```python
ref = db.reference("/")  # set ref point

# query all data under the ref
books = ref.get()
print(books)
print(type(books))

# print each item separately
for key, value in books.items():
    print(f"{key}: {value}")


# Query /Book1
ref = db.reference("/Book1")
books = ref.get()
print(books)
```

✓ 0.3s

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

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().

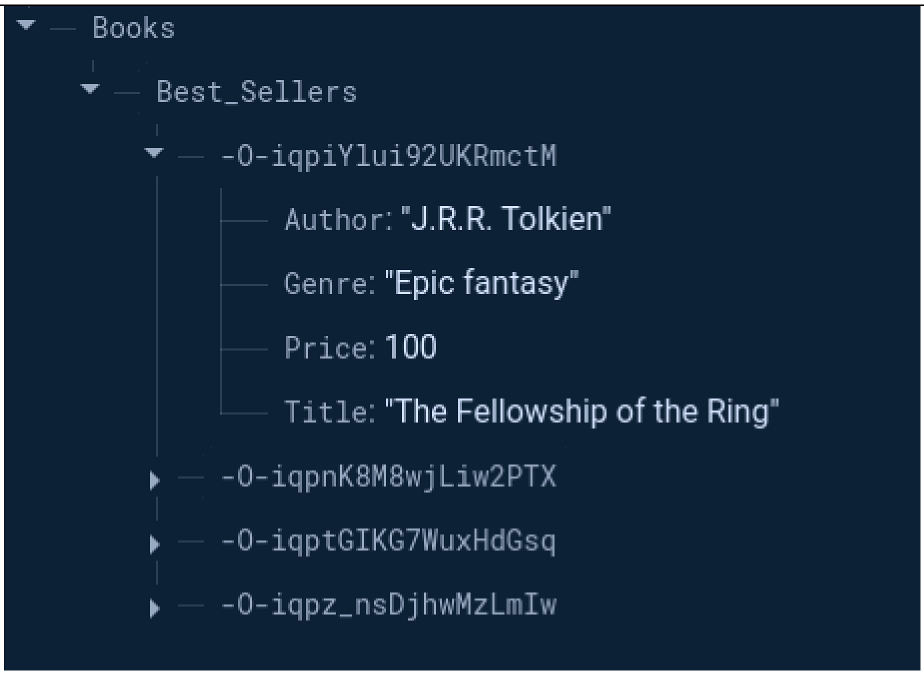
7 **Write to database Using the push() Function**:
The push() function saves data under a *unique system generated key*. This is different than set() where you set the keys such as Book1, Book2, Book3 and Book4 under which the content (author, genre, price and title) appears. Let's try to push the same 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.



```python
# Write using push() function
# Note that a set() is called on top of push()
#
ref = db.reference("/")
ref.set({
    "Books":
    {
        "Best_Sellers": -1
    }
})

ref = db.reference("/Books/Best_Sellers")

for key, value in data.items():
    ref.push().set(value)
```

```
✓  2.0s
```

The output will reset the previous data set in / node. The current data is shown below.

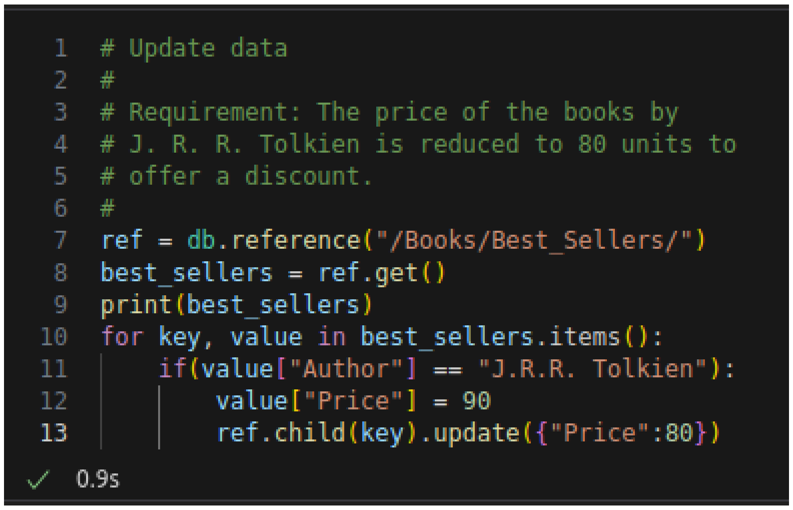


```
▾ — Books
    ▾ — Best_Sellers
        ▾ — -O-iqpiYlui92UKRmctM
                — Author: "J.R.R. Tolkien"
                — Genre: "Epic fantasy"
                — Price: 100
                — Title: "The Fellowship of the Ring"
        ▸ — -O-iqpnK8M8wjLiw2PTX
        ▸ — -O-iqptGIKG7WuxHdGsq
        ▸ — -O-iqpz_nsDjhwMzLmIw
```

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 desirable.

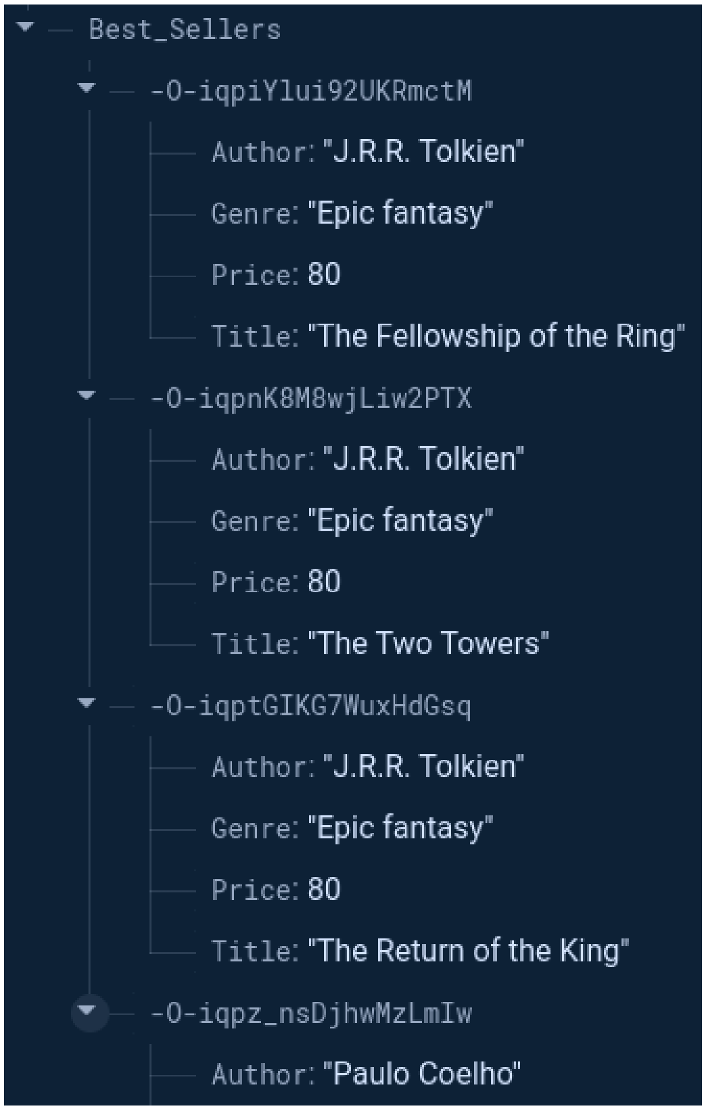
8 **Update data**:

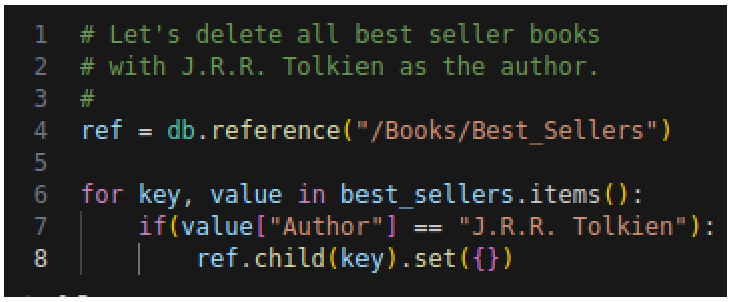
Let's say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.



```python
# Update data
#
# Requirement: The price of the books by
# J. R. R. Tolkien is reduced to 80 units to
# offer a discount.
#
ref = db.reference("/Books/Best_Sellers/")
best_sellers = ref.get()
print(best_sellers)
for key, value in best_sellers.items():
    if(value["Author"] == "J.R.R. Tolkien"):
        value["Price"] = 90
        ref.child(key).update({"Price":80})
```

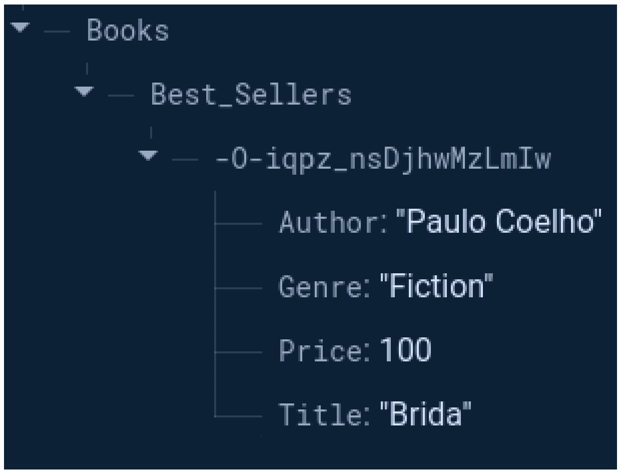
✓ 0.9s

	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.  Best_Sellers -O-iqpiYlui92UKRmctM Author: "J.R.R. Tolkien" Genre: "Epic fantasy" Price: 80 Title: "The Fellowship of the Ring" -O-iqpnK8M8wjLiw2PTX Author: "J.R.R. Tolkien" Genre: "Epic fantasy" Price: 80 Title: "The Two Towers" -O-iqptGIKG7WuxHdGsq Author: "J.R.R. Tolkien" Genre: "Epic fantasy" Price: 80 Title: "The Return of the King" -O-iqpz_nsDjhwMzLmIw Author: "Paulo Coelho"
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.



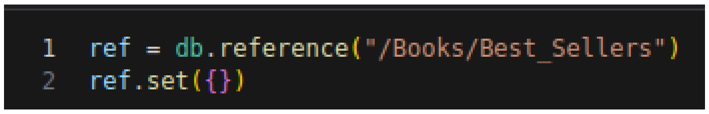
```python
# Let's delete all best seller books
# with J.R.R. Tolkien as the author.
#
ref = db.reference("/Books/Best_Sellers")

for key, value in best_sellers.items():
    if(value["Author"] == "J.R.R. Tolkien"):
        ref.child(key).set({})
```

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



```
Books
  Best_Sellers
    -O-iqpz_nsDjhwMzLmIw
      Author: "Paulo Coelho"
      Genre: "Fiction"
      Price: 100
      Title: "Brida"
```

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



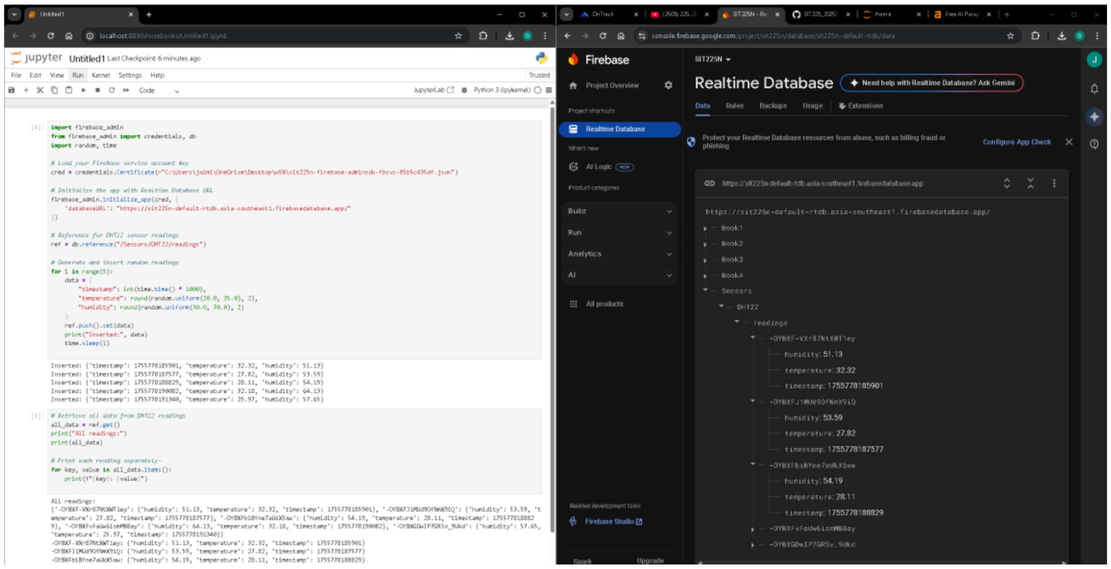
```python
ref = db.reference("/Books/Best_Sellers")
ref.set({})
```

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: **I will organize the data so that every sensor is saved under sensors/{sensor_id}, including a metadata section (sensor_name, type, location, units) and a readings section that contains all recorded values.** **Each reading will feature a shared timestamp (Unix ms) along with only the variables pertinent to that sensor — for instance, temperature and humidity for DHT22, or x, y, and z values for an accelerometer.** **To facilitate continuous logging, new entries will be appended under readings using Firebase push IDs, which prevents key collisions and naturally arranges the data chronologically.** **This structure reduces redundant metadata, accommodates various sensor types, simplifies time-based queries (latest values or by specific range), and guarantees efficient storage and retrieval in Firebase.**
11	Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here. Answer: 

```
[4]: import firebase_admin
from firebase_admin import credentials, db
import random, time

# Load your Firebase service account key
cred = credentials.Certificate(r"C:\Users\jalml\OneDrive\Desktop\w6N\sit225n-firebase-adminsdk-fbsvc-85b5c435df.json")

# Initialize the app with Realtime Database URL
firebase_admin.initialize_app(cred, {
    'databaseURL': "https://sit225n-default-rtdb.asia-southeast1.firebasedatabase.app/"
})

# Reference for DHT22 sensor readings
ref = db.reference("/Sensors/DHT22/readings")

# Generate and insert random readings
for i in range(5):
    data = {
        "timestamp": int(time.time() * 1000),
        "temperature": round(random.uniform(20.0, 35.0), 2),
        "humidity": round(random.uniform(30.0, 70.0), 2)
    }
    ref.push().set(data)
    print("Inserted:", data)
    time.sleep(1)
```

```
Inserted: {'timestamp': 1755778185901, 'temperature': 32.32, 'humidity': 51.13}
Inserted: {'timestamp': 1755778187577, 'temperature': 27.82, 'humidity': 53.59}
Inserted: {'timestamp': 1755778188829, 'temperature': 28.11, 'humidity': 54.19}
Inserted: {'timestamp': 1755778190082, 'temperature': 32.18, 'humidity': 64.13}
Inserted: {'timestamp': 1755778191340, 'temperature': 25.97, 'humidity': 57.65}
```

```
[5]: # Retrieve all data from DHT22 readings
all_data = ref.get()
print("All readings:")
print(all_data)

# Print each reading separately-
for key, value in all_data.items():
    print(f"{key}: {value}")
```

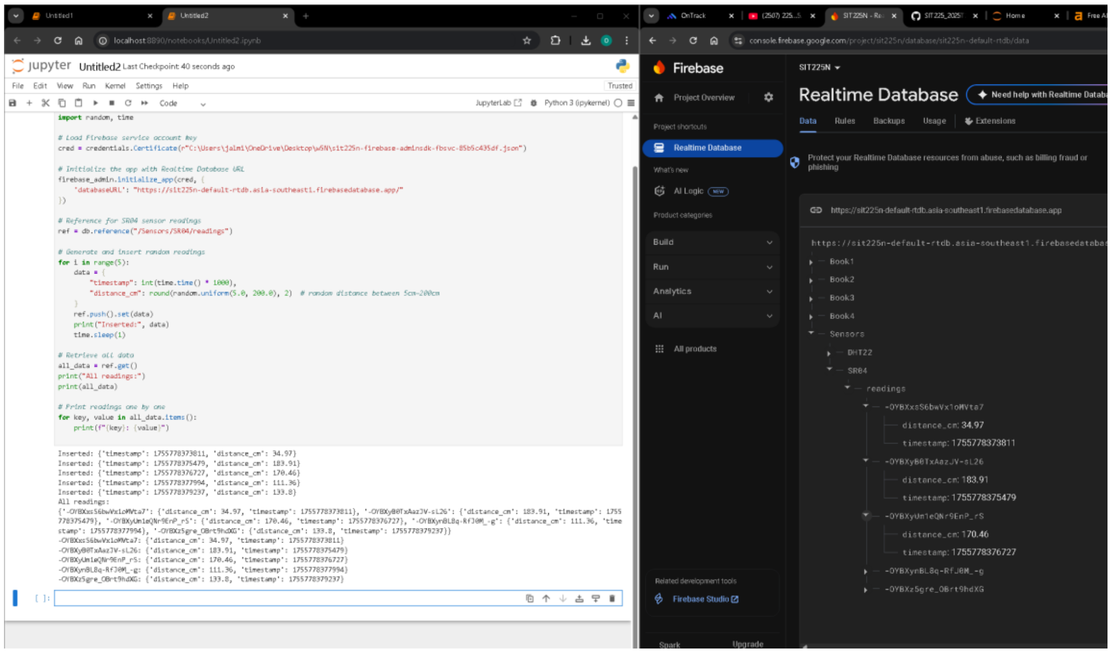
```
All readings:
{'-OYBXF-VXr87NtXWT1ey': {'humidity': 51.13, 'temperature': 32.32, 'timestamp': 1755778185901}, '-OYBXFJ1MUd9OfNmX91Q': {'humidity': 53.59, 't
emperature': 27.82, 'timestamp': 1755778187577}, '-OYBXFbiBYne7aUkX5xw': {'humidity': 54.19, 'temperature': 28.11, 'timestamp': 175577818882
9}, '-OYBXFvFaUw6ismMB8ay': {'humidity': 64.13, 'temperature': 32.18, 'timestamp': 1755778190082}, '-OYBXGDwIP7GRSv_9Ukd': {'humidity': 57.65,
'temperature': 25.97, 'timestamp': 1755778191340}}
-OYBXF-VXr87NtXWT1ey: {'humidity': 51.13, 'temperature': 32.32, 'timestamp': 1755778185901}
-OYBXFJ1MUd9OfNmX91Q: {'humidity': 53.59, 'temperature': 27.82, 'timestamp': 1755778187577}
-OYBXFbiBYne7aUkX5xw: {'humidity': 54.19, 'temperature': 28.11, 'timestamp': 1755778188829}
```

```
https://sit225n-default-rtdb.asia-southeast1.firebasedatabase.app/
  Book1
  Book2
  Book3
  Book4
  Sensors
    DHT22
      readings
        -OYBXF-VXr87NtXWT1ey
          humidity: 51.13
          temperature: 32.32
          timestamp: 1755778185901
        -OYBXFJ1MUd9OfNmX9iQ
          humidity: 53.59
          temperature: 27.82
          timestamp: 1755778187577
        -OYBXFbiBYne7aUkX5xw
          humidity: 54.19
          temperature: 28.11
          timestamp: 1755778188829
        -OYBXFvFaUw6ismMB8ay
        -OYBXGDwIP7GRSv_9Ukd
```

jupyter Untitled1 Last Checkpoint 6 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Python 3 (ipykernel)
Firebase
Realtime Database
Need help with Realtime Database? Ask Gemini
Data Rules Backups Usage Extensions
Protect your Realtime Database resources from abuse, such as billing fraud or phishing
Configure App Check

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

Answer:



```
import random, time

# Load Firebase service account key
cred = credentials.Certificate(r"C:\Users\jalmi\OneDrive\Desktop\w5N\sit225n-firebase-adminsdk-fbsvc-85b5c435df.json")

# Initialize the app with Realtime Database URL
firebase_admin.initialize_app(cred, {
    'databaseURL': "https://sit225n-default-rtdb.asia-southeast1.firebasedatabase.app/"
})

# Reference for SR04 sensor readings
ref = db.reference("/Sensors/SR04/readings")

# Generate and insert random readings
for i in range(5):
    data = {
        "timestamp": int(time.time() * 1000),
        "distance_cm": round(random.uniform(5.0, 200.0), 2)  # random distance between 5cm-200cm
    }
    ref.push().set(data)
    print("Inserted:", data)
    time.sleep(1)

# Retrieve all data
all_data = ref.get()
print("All readings:")
print(all_data)

# Print readings one by one
for key, value in all_data.items():
    print(f"{key}: {value}")
```

```
Inserted: {'timestamp': 1755778373811, 'distance_cm': 34.97}
Inserted: {'timestamp': 1755778375479, 'distance_cm': 183.91}
Inserted: {'timestamp': 1755778376727, 'distance_cm': 170.46}
Inserted: {'timestamp': 1755778377994, 'distance_cm': 111.36}
Inserted: {'timestamp': 1755778379237, 'distance_cm': 133.8}
All readings:
{'-OYBXxsS6bwVx1oMVta7': {'distance_cm': 34.97, 'timestamp': 1755778373811}, '-OYBXyB0TxAazJV-sL26': {'distance_cm': 183.91, 'timestamp': 1755
778375479}, '-OYBXyUm1eQNr9EnP_rS': {'distance_cm': 170.46, 'timestamp': 1755778376727}, '-OYBXynBL8q-RfJ0M_-g': {'distance_cm': 111.36, 'time
stamp': 1755778377994}, '-OYBXz5gre_OBrt9hdXG': {'distance_cm': 133.8, 'timestamp': 1755778379237}}
-OYBXxsS6bwVx1oMVta7: {'distance_cm': 34.97, 'timestamp': 1755778373811}
-OYBXyB0TxAazJV-sL26: {'distance_cm': 183.91, 'timestamp': 1755778375479}
-OYBXyUm1eQNr9EnP_rS: {'distance_cm': 170.46, 'timestamp': 1755778376727}
-OYBXynBL8q-RfJ0M_-g: {'distance_cm': 111.36, 'timestamp': 1755778377994}
-OYBXz5gre_OBrt9hdXG: {'distance_cm': 133.8, 'timestamp': 1755778379237}
```

```
https://sit225n-default-rtdb.asia-southeast1.firebasedatabas
  Book1
  Book2
  Book3
  Book4
  Sensors
    DHT22
    SR04
      readings
        -OYBXxsS6bwVx1oMVta7
          distance_cm: 34.97
          timestamp: 1755778373811
        -OYBXyB0TxAazJV-sL26
          distance_cm: 183.91
          timestamp: 1755778375479
        -OYBXyUm1eQNr9EnP_rS
          distance_cm: 170.46
          timestamp: 1755778376727
        -OYBXynBL8q-RfJ0M_-g
        -OYBXz5gre_OBrt9hdXG
```

13 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.

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.

Answer: The Firebase Realtime Database activates events whenever data is created, updated, deleted, or written. By utilizing Cloud Functions (Gen 2), you can link triggers like onValueCreated, onValueUpdated, onValueDeleted, and onValueWritten to a specific path (for instance, /sensors/{sensorId}/readings/{pushId}), and the function will run with each modification. The handler will obtain a snapshot for create/delete events or before/after snapshots for write/update events, enabling you to validate, compute aggregates, or send alerts without altering client code. You can use path wildcards (such as {sensorId}) to handle multiple sensors, and it's wise to deploy the function in the same region as the database to reduce latency.

	In Python, the Admin SDK allows secure read/write operations but does not support realtime listeners. A monitoring application can either (1) use REST streaming (SSE) to follow a path in near-realtime, or (2) periodically check for recent readings. A suggested method is to let Cloud Functions normalize updates (for example, writing to /latest/{sensorId} or /events) and have the Python dashboard keep an eye on that single path through SSE or light polling.

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.

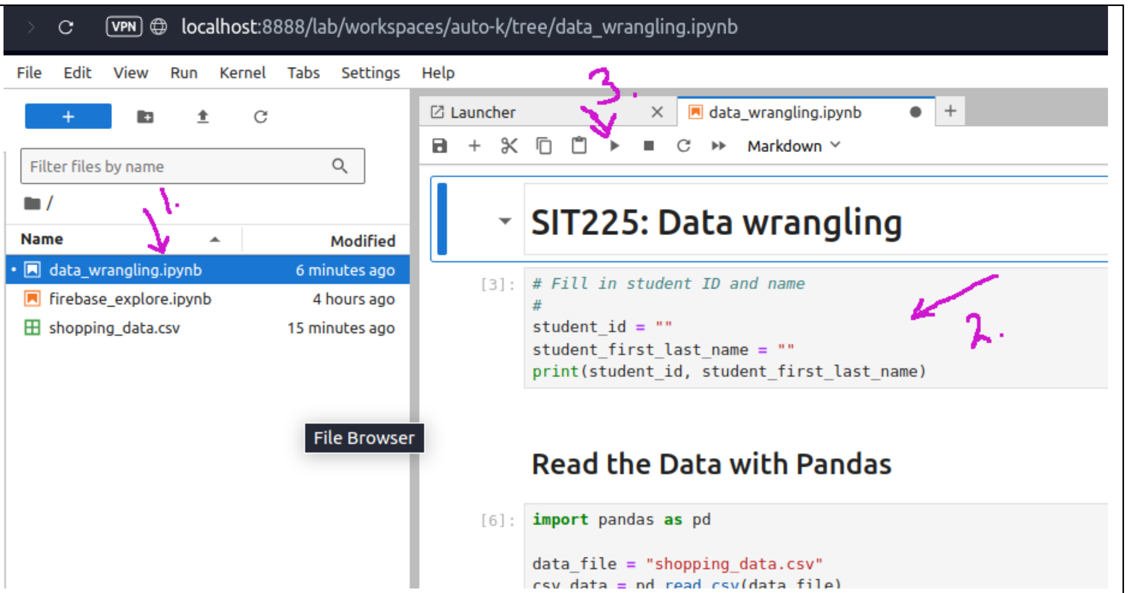
Software Required

Python 3
Pandas Python library

Steps

Step	Action

1	Install Pandas using the command below. Most likely you already have Pandas installed if you have installed Python using Anaconda disribution (https://www.anaconda.com/download).
	
	`$ pip install pandas`
	
	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:
	
	`$ jupyter lab`
	
	This will open Python Jupyter Notebook where in the left panel you can see the files (labeled as 1 in figure).

	 localhost:8888/lab/workspaces/auto-k/tree/data_wrangling.ipynb
	File Edit View Run Kernel Tabs Settings Help
	Filter files by name
	Name Modified
	data_wrangling.ipynb 6 minutes ago
	firebase_explore.ipynb 4 hours ago
	shopping_data.csv 15 minutes ago
	File Browser
	Launcher
	data_wrangling.ipynb
	Markdown
	1.
	2.
	3.SIT225: Data wrangling
	[3]: `# Fill in student ID and name`
	`#`
	`student_id = ""`
	`student_first_last_name = ""`
	`print(student_id, student_first_last_name)`
	Read the Data with Pandas
	[6]: `import pandas as pd`
	`data_file = "shopping_data.csv"`
	`csv_data = pd.read_csv(data_file)`
	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).

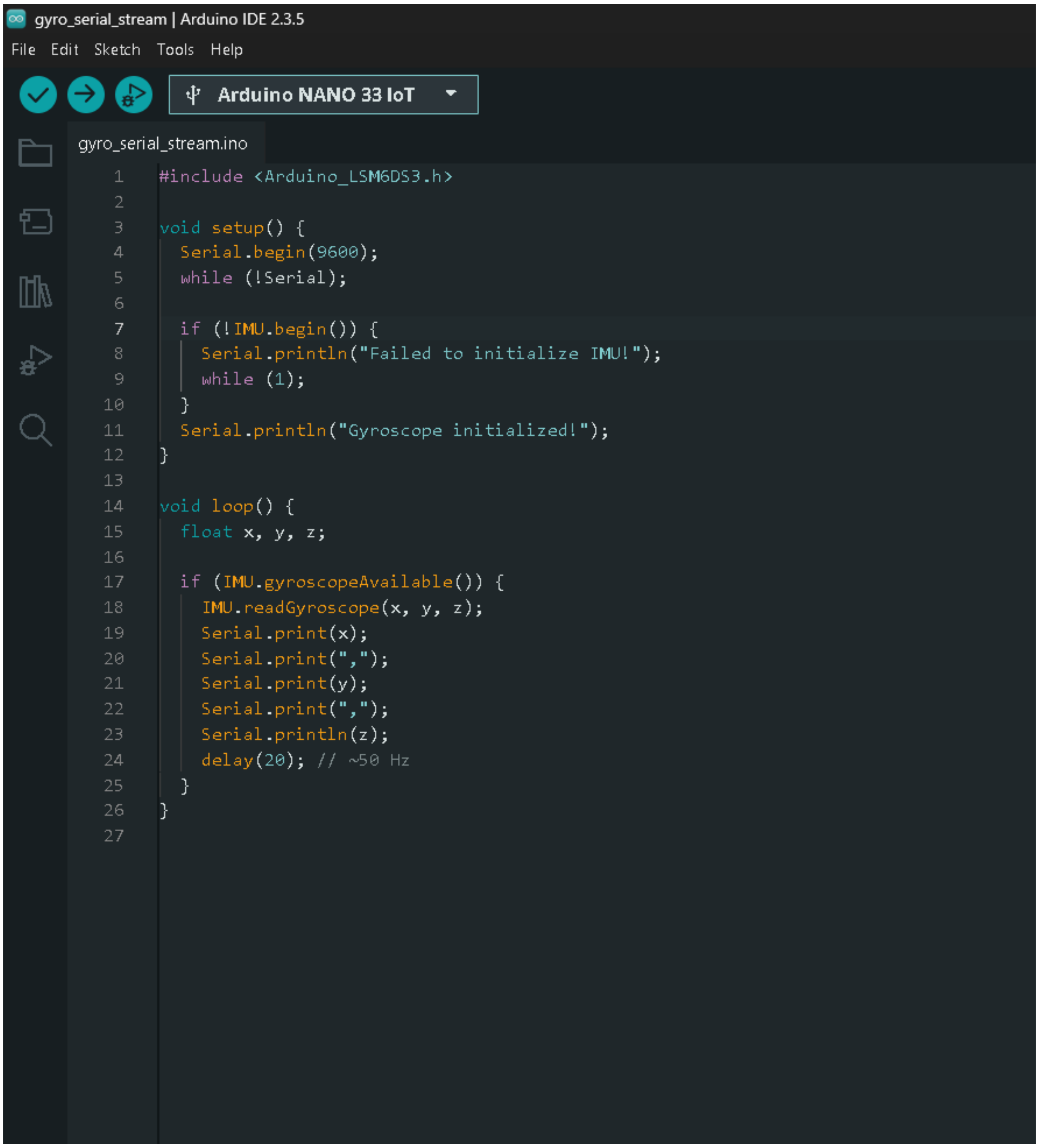
2	Question: Run each cell to produce output. Follow instructions in the notebook to complete codes in some of the cells. Convert the notebook to PDF from menu File > Save and Export Notebook As > PDF. Convert this activity sheet to PDF and merge with the notebook PDF. Answer: There is no answer to write here. You have to answer in the Jupyter Notebook.
3	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? Answer: Yes, Pandas can easily handle sensor CSV files with read_csv(). The main tweaks I would make are cleaning and preparing the data: converting timestamps into proper datetime format and using them as the index, standardizing column names and units (e.g., temperature, humidity, distance_cm), handling missing values with fill or interpolation, dropping duplicates, and filtering unrealistic readings such as humidity over 100% or negative distance values.

4	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. Answer: I checked out the "Handling Missing Value" section and took it to mean "first spot the NaNs, then pick a fill that fits the data." If there are just a few missing rows and they seem random, I'd just get rid of them; for numeric columns, I'd use the mean or median (going with the median if the data is skewed), and for categorical data, I'd go for the mode. When it comes to sensor time-series data, I'd use forward or backward filling or time-based interpolation for short gaps. Plus, if the features are linked, a simple model like KNN could work well. For example, with DHT22 temperature, I'd fill in a short gap with interpolation, for a skewed variable like Income, I'd pick the median, and for Gender, I'd choose the mode—then I'd make sure to note down the changes for clarity.

Q2. State the hypothesis you can think of out of your data. Show the graph created from the sensor data, analyse it and describe if there are any interesting patterns you can observe. Justify if your hypothesis holds, at what level; if not, then what might be the reason?

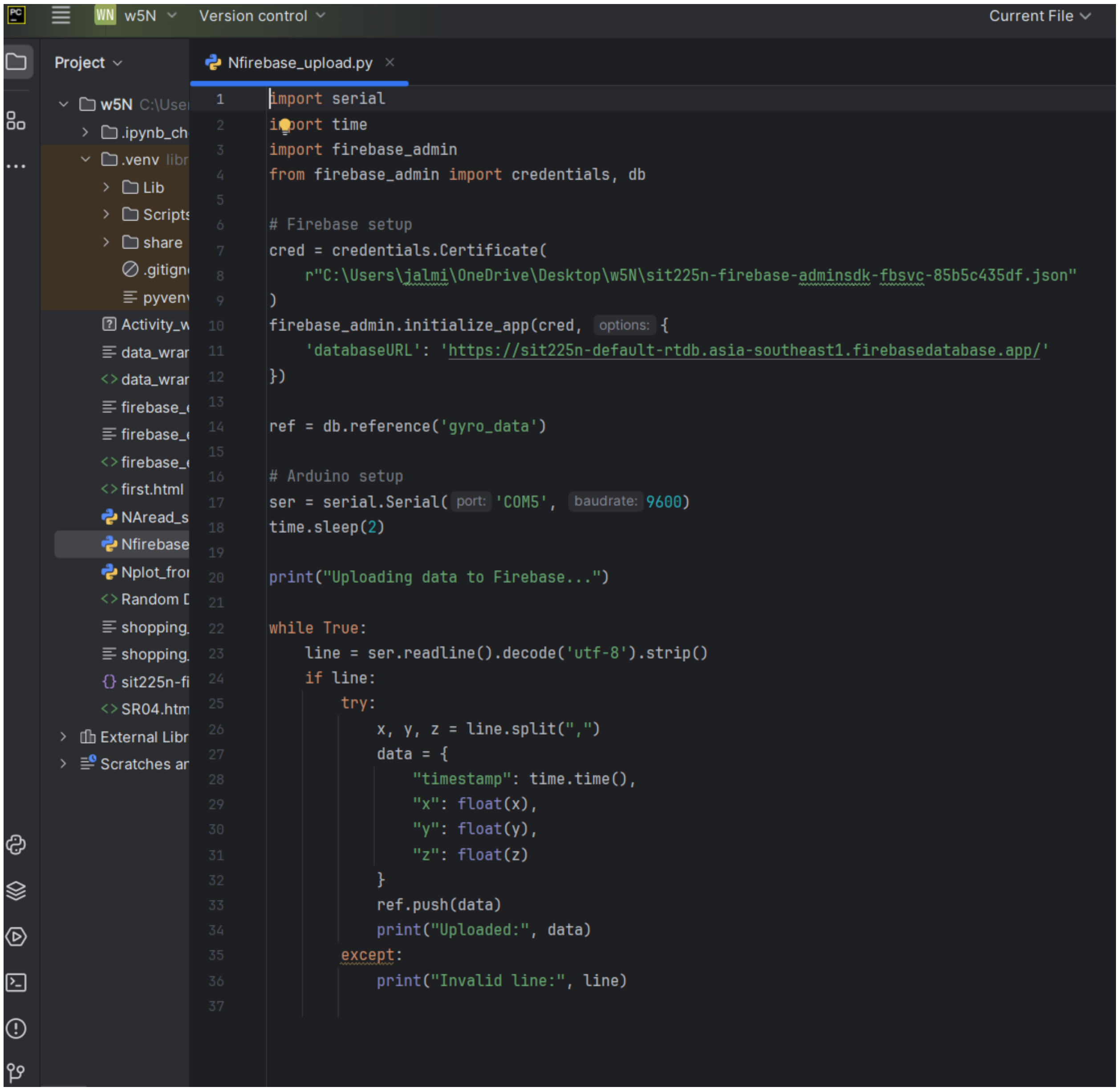
My hypothesis was that the gyroscope readings on the X, Y, and Z axes would stay close to zero when the Arduino board was stationary, but would show clear spikes whenever the board was rotated. From the graphs I collected, the results supported this idea because the values stayed stable with only small fluctuations when the board was not moving, while sharp peaks appeared in the axis that matched the direction of rotation. I also noticed that sometimes the other axes showed minor changes, which could be due to the way the sensor is aligned. Overall, the hypothesis holds true at a general level because the gyroscope correctly detected both stillness and movement, but the data was not perfectly stable. This is most likely caused by normal sensor noise, drift over time, or small calibration errors.

Q3. Paste Python and Arduino sketch and explain program steps.


gyro_serial_stream | Arduino IDE 2.3.5
File Edit Sketch Tools Help
Arduino NANO 33 IoT
gyro_serial_stream.ino

```
#include <Arduino_LSM6DS3.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);

  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.println("Gyroscope initialized!");
}

void loop() {
  float x, y, z;

  if (IMU.gyroscopeAvailable()) {
    IMU.readGyroscope(x, y, z);
    Serial.print(x);
    Serial.print(",");
    Serial.print(y);
    Serial.print(",");
    Serial.println(z);
    delay(20); // ~50 Hz
  }
}
```

The Arduino sketch starts by including the Arduino_LSM6DS3 library and initializing the gyroscope sensor inside the setup() function. If the sensor is successfully detected, it prints a confirmation message; otherwise, it displays an error. In the loop(), the Arduino continuously checks if gyroscope data is available, reads the values for the X, Y, and Z axes, and then sends them over the serial port in a comma-separated format with a small delay to maintain a stable reading rate.



```python
import serial
import time
import firebase_admin
from firebase_admin import credentials, db

# Firebase setup
cred = credentials.Certificate(
    r"C:\Users\jalmi\OneDrive\Desktop\w5N\sit225n-firebase-adminsdk-fbsvc-85b5c435df.json"
)
firebase_admin.initialize_app(cred, options: {
    'databaseURL': 'https://sit225n-default-rtdb.asia-southeast1.firebasedatabase.app/'
})

ref = db.reference('gyro_data')

# Arduino setup
ser = serial.Serial( port: 'COM5',  baudrate: 9600)
time.sleep(2)

print("Uploading data to Firebase...")

while True:
    line = ser.readline().decode('utf-8').strip()
    if line:
        try:
            x, y, z = line.split(",")
            data = {
                "timestamp": time.time(),
                "x": float(x),
                "y": float(y),
                "z": float(z)
            }
            ref.push(data)
            print("Uploaded:", data)
        except:
            print("Invalid line:", line)
```

On the Python side, the script uses the serial library to connect to the Arduino through the correct COM port at 9600 baud rate. It then continuously reads the incoming serial data, splits each line into X, Y, and Z values, and stores them in lists along with timestamps. The program uses matplotlib to update live plots, showing the changes in each axis in real time. This way, the

Arduino is responsible for capturing and transmitting the raw sensor data, while the Python program handles storing and visualizing it.

Q4. Create a video in Panopto/CloudDeakin showing your program execution, data collection, data upload to Firebase and graph output, share the video link here.

Q5. Create a subdirectory 'week-5' under directory 'SIT225_<YYYY>T2' in your drive where you copy the Python script file, Arduino sketch file, data file and the generated graphs. Commit and push changes to GitHub. Include the link to your repository here with a GitHub page screenshot of weekly folder content. A tutor may try to access your GitHub link, if necessary. Give access to your tutor by adding the tutor's email address as a collaborator of your private repository.

Instructions

Consolidate outputs following the submission details above into a single PDF file.

Submit your work

When you are ready, login to OnTrack and submit your pdf which consolidates all the items mentioned in the submission detail section above. Remember to save and backup your work.

Complete your work

After your submission, your OnTrack reviewer (tutor) will review your submission and give you feedback in about 5 business days. Your reviewer may further ask you some questions on the weekly topics and/or about your submissions. You are required to address your OnTrack reviewer's questions as a form of task discussion. Please frequently login to OnTrack for the task ***Discuss/Demonstrate*** or ***Resubmit*** equivalent to fix your work (if needed) based on the feedback to get your task signed as ***Complete***.