Student name: Hoang Long Tran

Student ID: s223128143

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

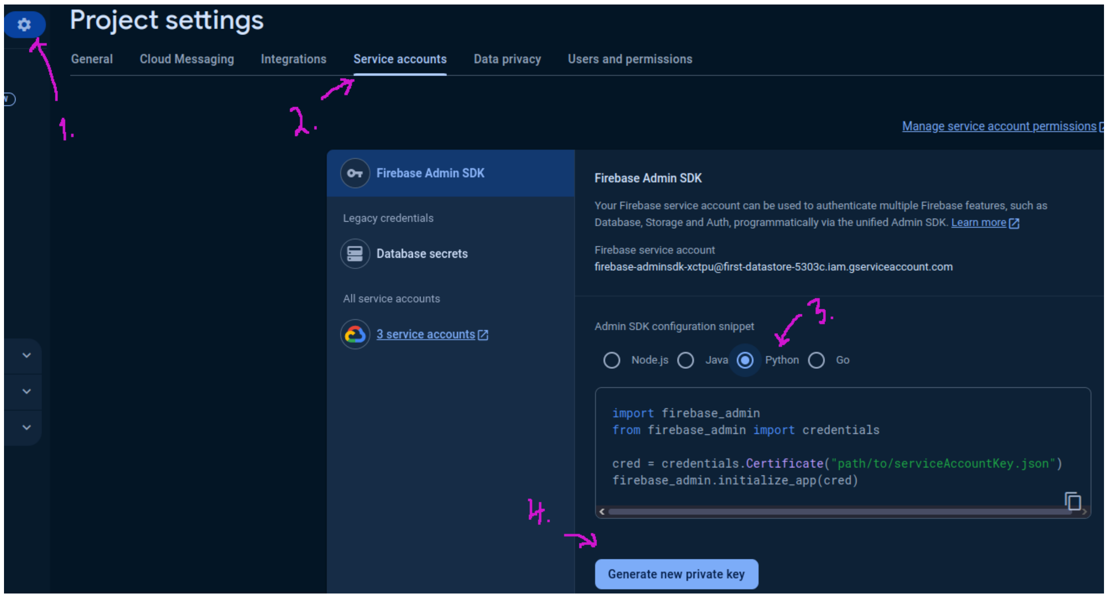
Steps

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.



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

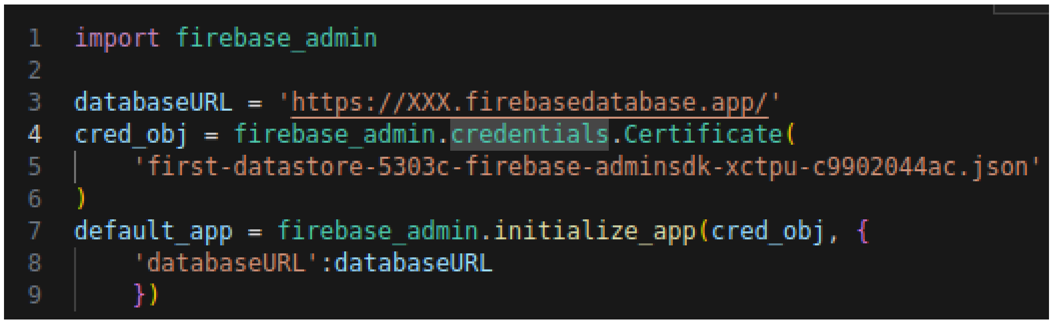
Firebase service account
firebase-adminsdk-xctpu@first-datastore-5303c.iam.gserviceaccount.com

Admin SDK configuration snippet

Node.js Java [✓ Python] Go

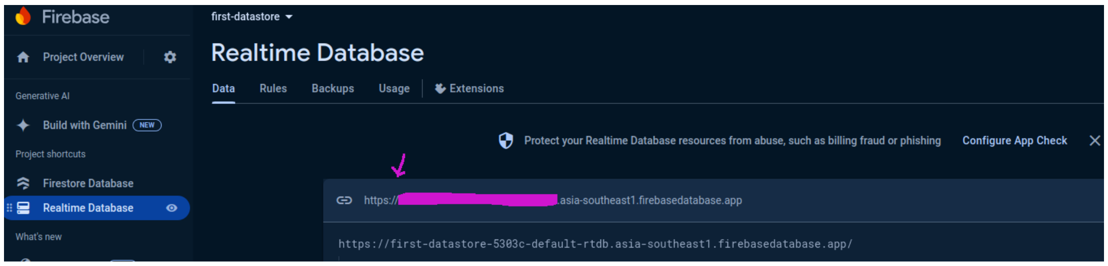
```
import firebase_admin
from firebase_admin import credentials

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).



```
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.



Realtime Database

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

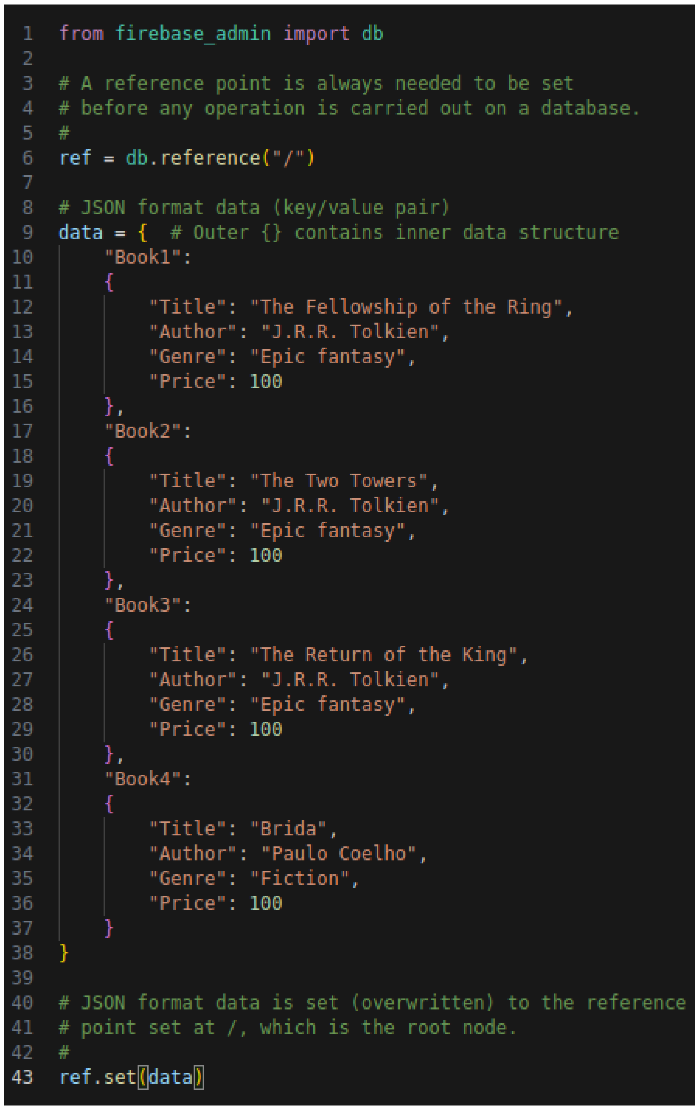
https://[redacted].asia-southeast1.firebasedatabase.app

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**:

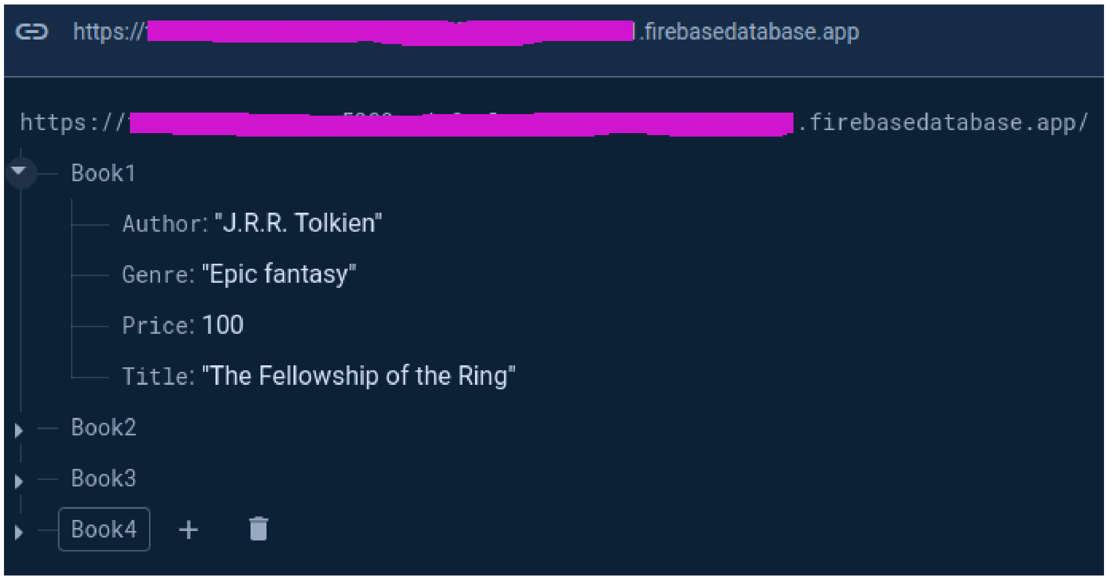
We set the reference to the root of the database (or we could also set it to a key value or child key value). Data needs to be in JSON format as below.



```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)
```

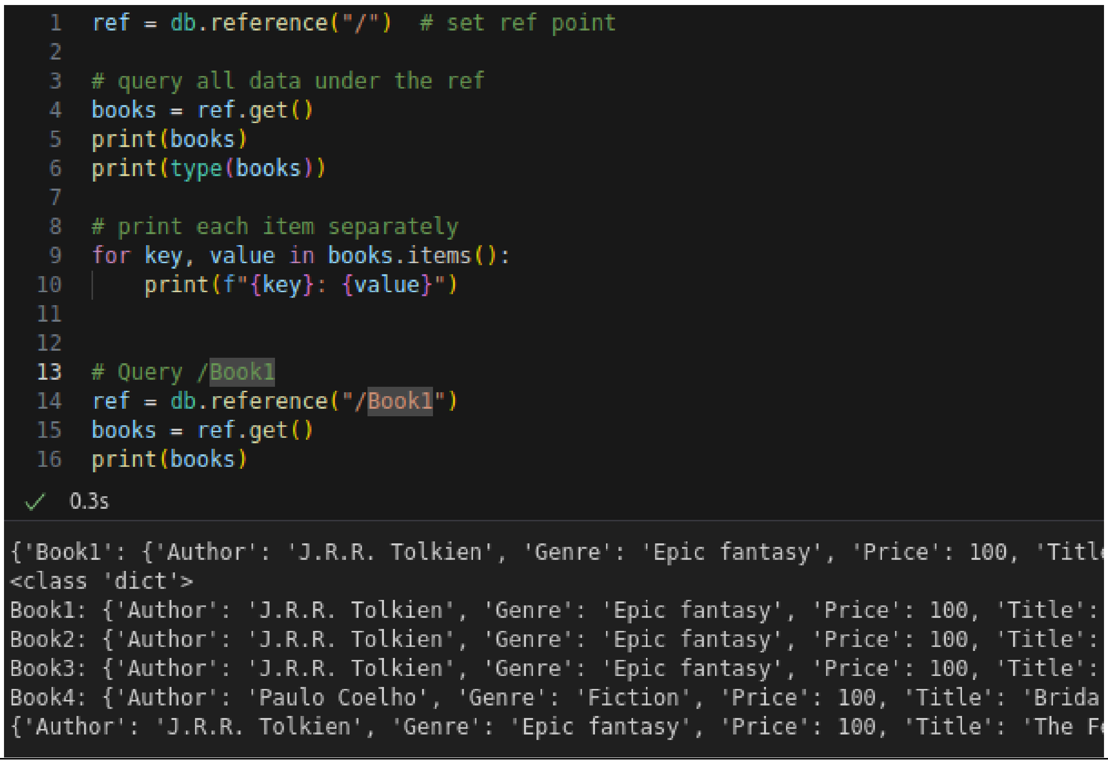
A reference point always needed to be set where the data read/write will take place. In the code above, the reference point is set at the root of the NoSQL Document, where consider the database is a JSON tree and / is the root node of the tree). The set() function writes (overwrites) data at the set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



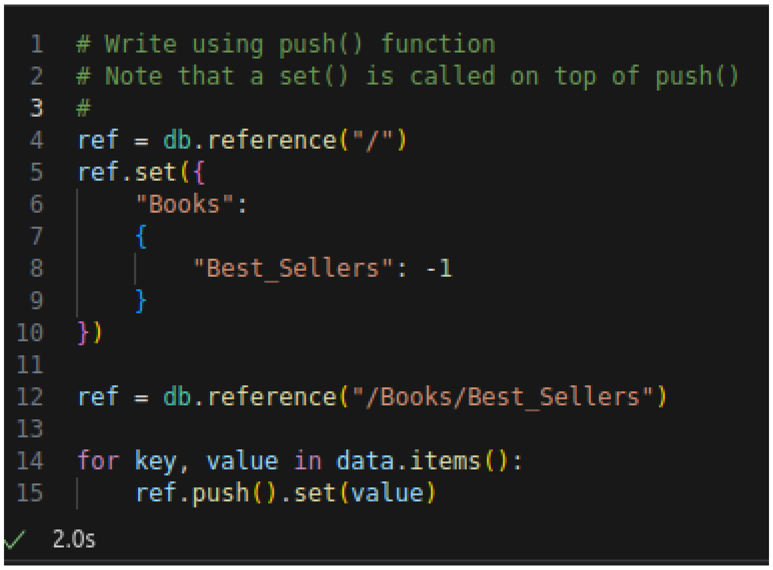
```python
ref = db.reference("/")  # set ref point

# 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().

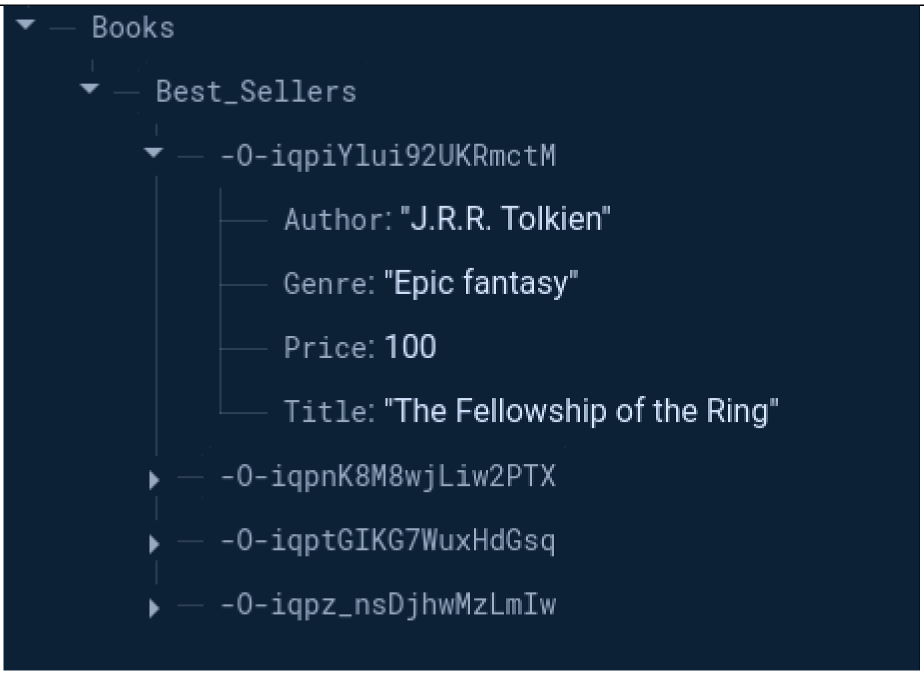
7 **Write to database Using the push() Function**:
The push() function saves data under a *unique system generated key*. This is different than set() where you set the keys such as Book1, Book2, Book3 and Book4 under which the content (author, genre, price and title) appears. Let’s try to push the same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

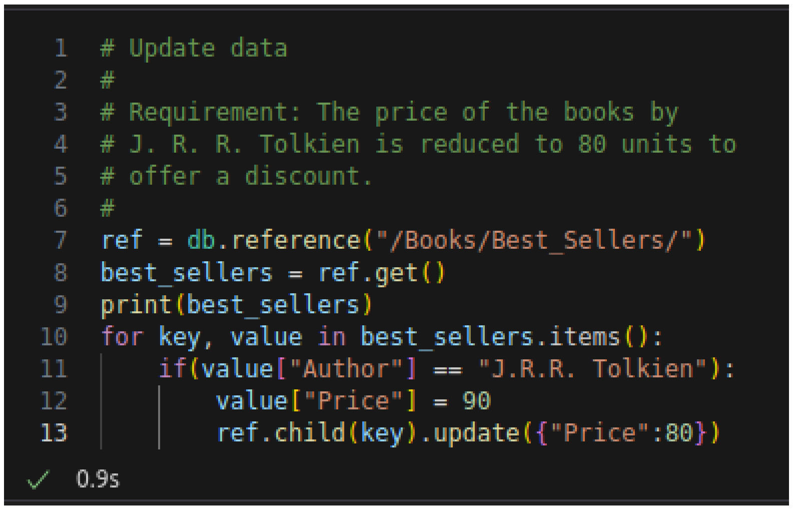
The output will reset the previous data set in / node. The current data is shown below.



```
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
```

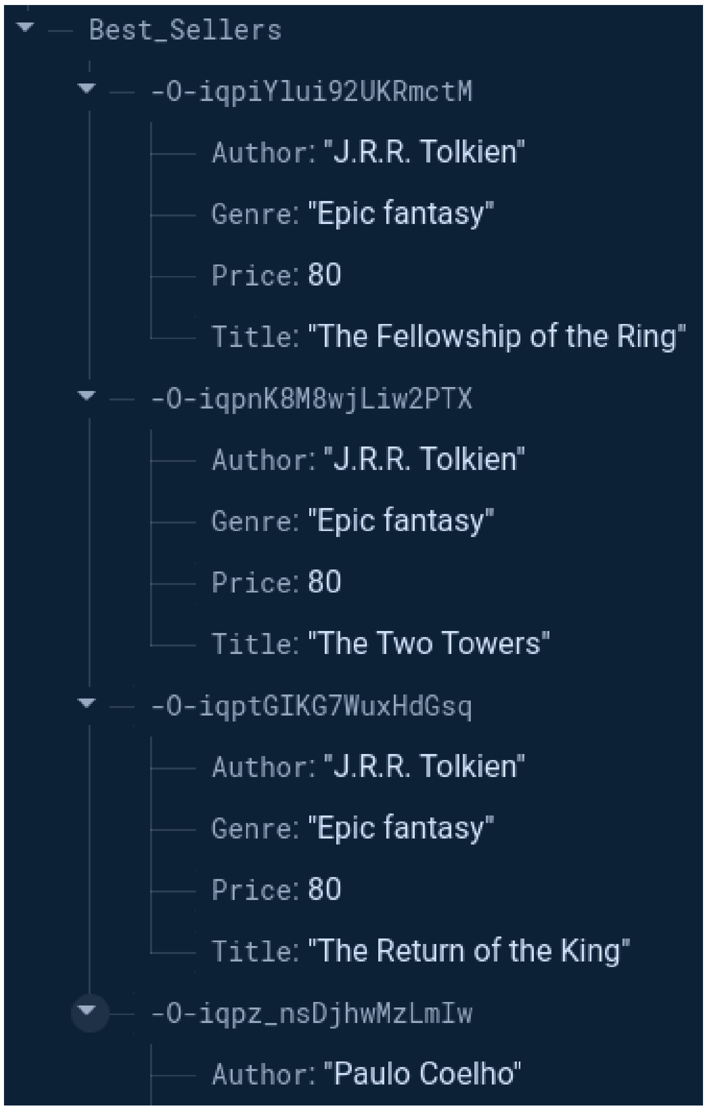
As you can see, under /Books/Best_Sellers there are 4 nodes where the node head (or node ID) is a randomly generated key which is due to the use of push() function. When data key does not matter, the use of push() function desirable.

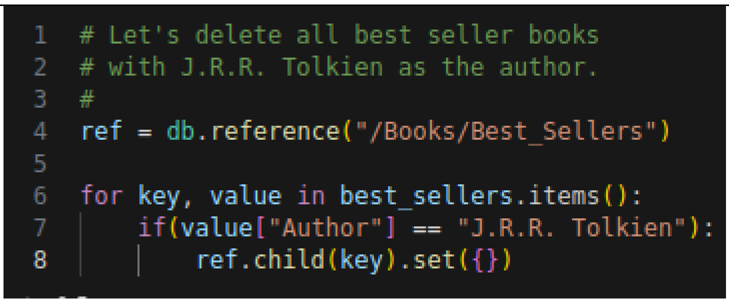
8 **Update data**:
Let’s say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.



```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

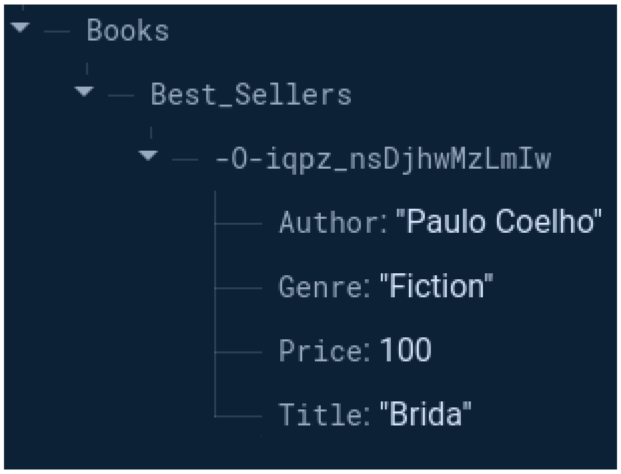
As you can see, the author name is compared and the new price is set in the best_sellers dictionary and finally, an update() function is called on the ref, however, the current ref is a ‘/Books/Best_Sellers/’, so we need to locate the

	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.



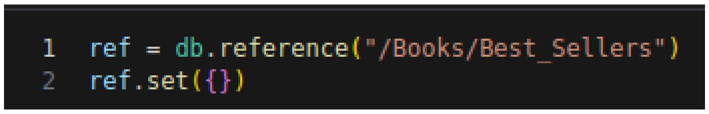
```python
# Let's delete all best seller books
# with J.R.R. Tolkien as the author.
#
ref = db.reference("/Books/Best_Sellers")

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.

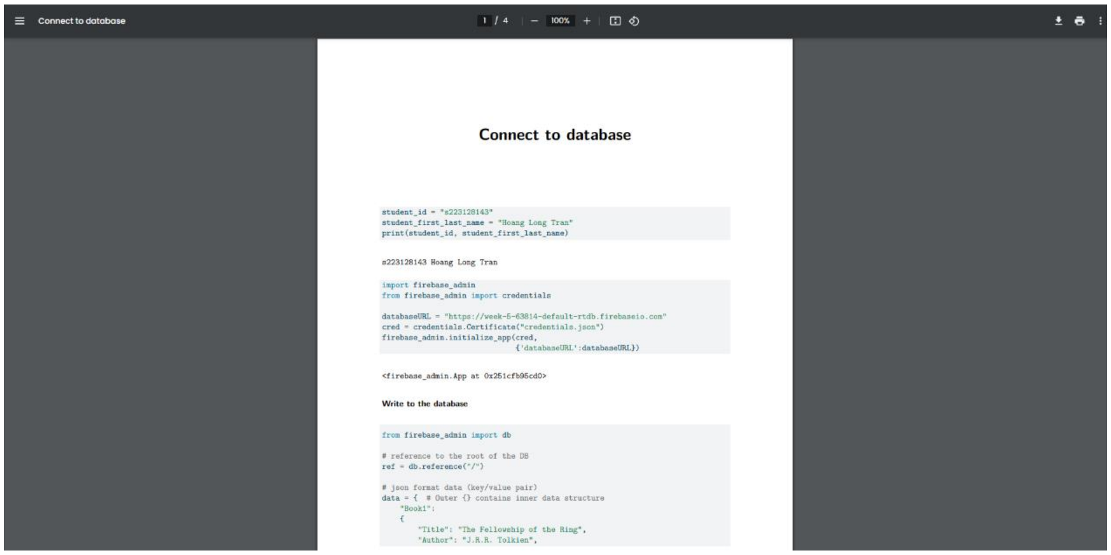


```python
ref = db.reference("/Books/Best_Sellers")
ref.set({})
```

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.



Connect to database

```
student_id = "s223128143"
student_first_last_name = "Hoang Long Tran"
print(student_id, student_first_last_name)
```

```
s223128143 Hoang Long Tran
```

```
import firebase_admin
from firebase_admin import credentials

databaseURL = "https://week-5-63814-default-rtdb.firebaseio.com"
cred = credentials.Certificate("credentials.json")
firebase_admin.initialize_app(cred,
                              {'databaseURL':databaseURL})
```

```
<firebase_admin.App at 0x251cfb95cd0>
```

Write to the database

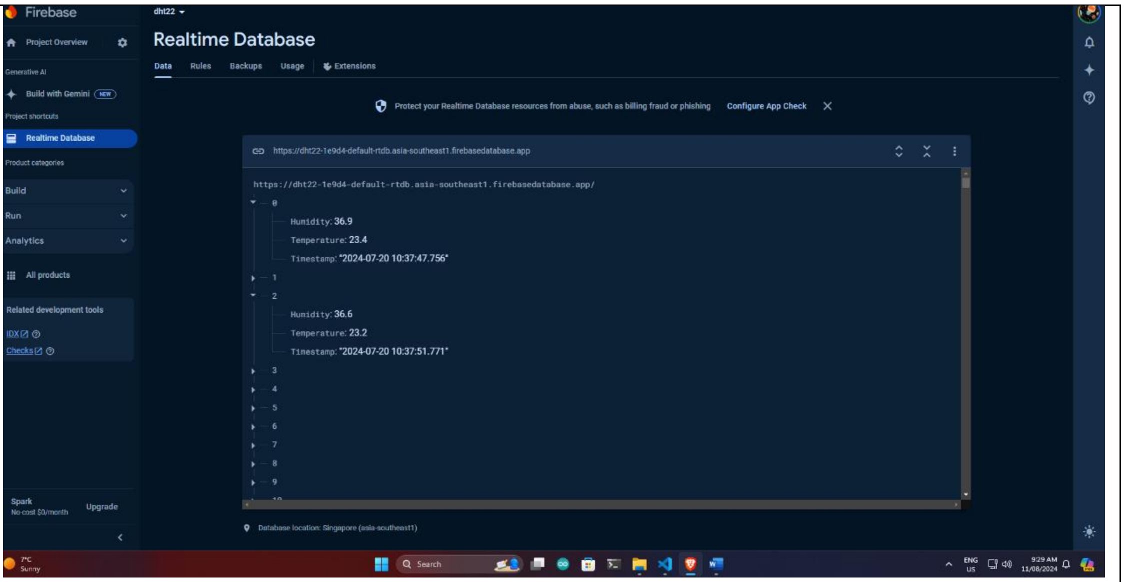
```
from firebase_admin import db

# reference to the root of the DB
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",
```

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: Since the JSON format is similar to that of a dictionary, I will explain in dictionary terms. There will be keys for each sensor, in each key there will be values of that sensor.

11 Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here.

Answer: I am going to reuse my week 2 dht22 data.

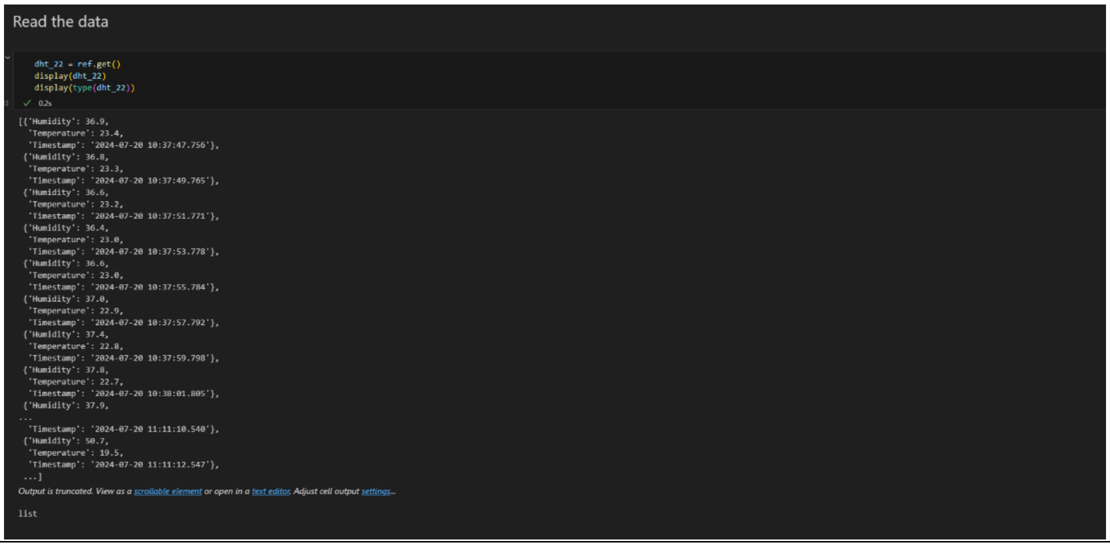


Realtime Database

https://dht22-1e9d4-default-rtdb.asia-southeast1.firebasedatabase.app/

```
▾ 0
    Humidity: 36.9
    Temperature: 23.4
    Timestamp: "2024-07-20 10:37:47.756"
▸ 1
▾ 2
    Humidity: 36.6
    Temperature: 23.2
    Timestamp: "2024-07-20 10:37:51.771"
▸ 3
▸ 4
▸ 5
▸ 6
▸ 7
▸ 8
▸ 9
```

Database location: Singapore (asia-southeast1)



Read the data

```python
dht_22 = ref.get()
display(dht_22)
display(type(dht_22))
```

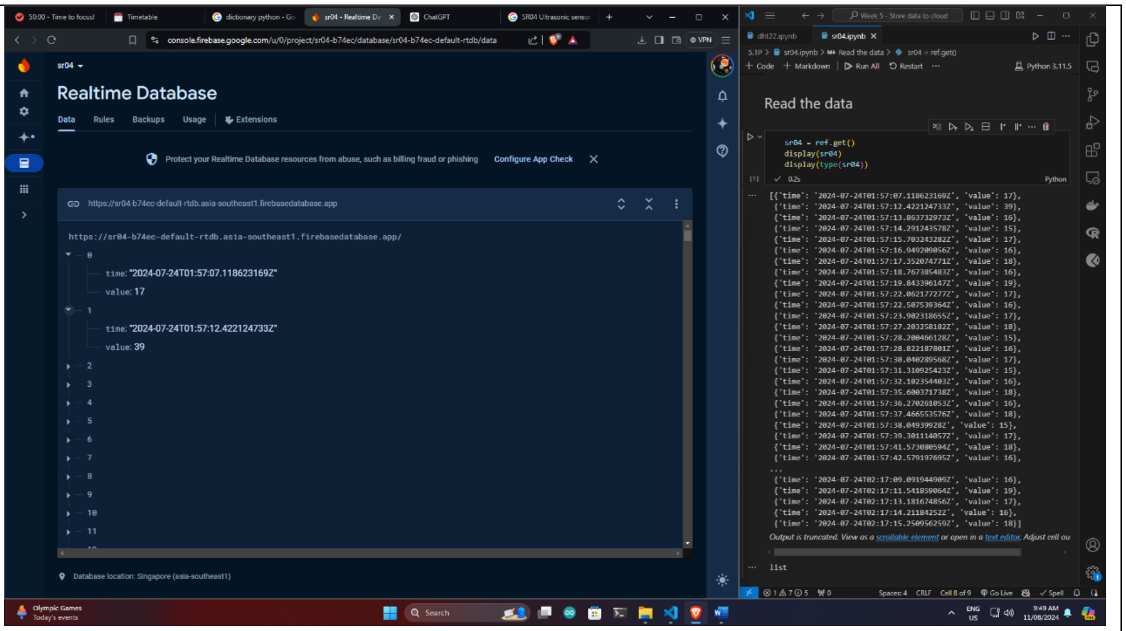
✓ 0.2s

```
[{'Humidity': 36.9,
  'Temperature': 23.4,
  'Timestamp': '2024-07-20 10:37:47.756'},
 {'Humidity': 36.8,
  'Temperature': 23.3,
  'Timestamp': '2024-07-20 10:37:49.765'},
 {'Humidity': 36.6,
  'Temperature': 23.2,
  'Timestamp': '2024-07-20 10:37:51.771'},
 {'Humidity': 36.4,
  'Temperature': 23.0,
  'Timestamp': '2024-07-20 10:37:53.778'},
 {'Humidity': 36.6,
  'Temperature': 23.0,
  'Timestamp': '2024-07-20 10:37:55.784'},
 {'Humidity': 37.0,
  'Temperature': 22.9,
  'Timestamp': '2024-07-20 10:37:57.792'},
 {'Humidity': 37.4,
  'Temperature': 22.8,
  'Timestamp': '2024-07-20 10:37:59.798'},
 {'Humidity': 37.8,
  'Temperature': 22.7,
  'Timestamp': '2024-07-20 10:38:01.805'},
 {'Humidity': 37.9,
...
  'Timestamp': '2024-07-20 11:11:10.540'},
 {'Humidity': 50.7,
  'Temperature': 19.5,
  'Timestamp': '2024-07-20 11:11:12.547'},
 ...]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
list
```

12	Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here. Answer: I will reuse the SR04 data from week 3



Realtime Database

Data Rules Backups Usage Extensions

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

```
https://sr04-b74ec-default-rtdb.asia-southeast1.firebasedatabase.app/
▾ 0
    time: "2024-07-24T01:57:07.118623169Z"
    value: 17
▾ 1
    time: "2024-07-24T01:57:12.422124733Z"
    value: 39
▸ 2
▸ 3
▸ 4
▸ 5
▸ 6
▸ 7
▸ 8
▸ 9
▸ 10
▸ 11
```

Database location: Singapore (asia-southeast1)

Read the data

```python
sr04 = ref.get()
display(sr04)
display(type(sr04))
```

```
[{'time': '2024-07-24T01:57:07.118623169Z', 'value': 17},
 {'time': '2024-07-24T01:57:12.422124733Z', 'value': 39},
 {'time': '2024-07-24T01:57:13.863732973Z', 'value': 16},
 {'time': '2024-07-24T01:57:14.291243578Z', 'value': 15},
 {'time': '2024-07-24T01:57:15.703243282Z', 'value': 17},
 {'time': '2024-07-24T01:57:16.949209056Z', 'value': 16},
 {'time': '2024-07-24T01:57:17.352074771Z', 'value': 18},
 {'time': '2024-07-24T01:57:18.767385483Z', 'value': 16},
 {'time': '2024-07-24T01:57:19.843396147Z', 'value': 19},
 {'time': '2024-07-24T01:57:22.062177277Z', 'value': 17},
 {'time': '2024-07-24T01:57:22.507539364Z', 'value': 16},
 {'time': '2024-07-24T01:57:23.902318655Z', 'value': 17},
 {'time': '2024-07-24T01:57:27.203258182Z', 'value': 18},
 {'time': '2024-07-24T01:57:28.200466128Z', 'value': 15},
 {'time': '2024-07-24T01:57:28.822187801Z', 'value': 16},
 {'time': '2024-07-24T01:57:30.040289568Z', 'value': 17},
 {'time': '2024-07-24T01:57:31.310925423Z', 'value': 15},
 {'time': '2024-07-24T01:57:32.102354403Z', 'value': 16},
 {'time': '2024-07-24T01:57:35.600371738Z', 'value': 18},
 {'time': '2024-07-24T01:57:36.270261053Z', 'value': 16},
 {'time': '2024-07-24T01:57:37.466553576Z', 'value': 18},
 {'time': '2024-07-24T01:57:38.04939928Z', 'value': 15},
 {'time': '2024-07-24T01:57:39.301114057Z', 'value': 17},
 {'time': '2024-07-24T01:57:41.573080594Z', 'value': 18},
 {'time': '2024-07-24T01:57:42.579197695Z', 'value': 16},
...
 {'time': '2024-07-24T02:17:09.091944909Z', 'value': 16},
 {'time': '2024-07-24T02:17:11.541859064Z', 'value': 19},
 {'time': '2024-07-24T02:17:13.181674856Z', 'value': 17},
 {'time': '2024-07-24T02:17:14.21184252Z', 'value': 16},
 {'time': '2024-07-24T02:17:15.250956259Z', 'value': 18}]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell ou

list
```

13

<span style="color:red">Question</span>: 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 idea of database event is to handle events without needing to update client code. There are 2 levels to handle Realtime database events. First one being to listen for specific operations like only write, create update or delete events. The second is to listen for any change of any kind to a reference.

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.

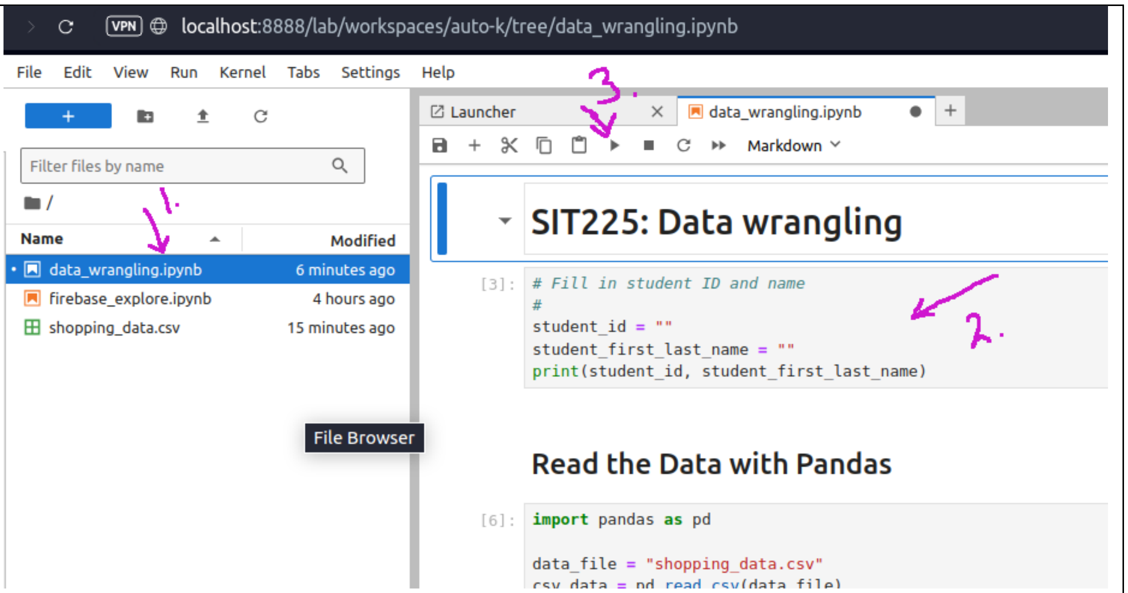
Software Required

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).

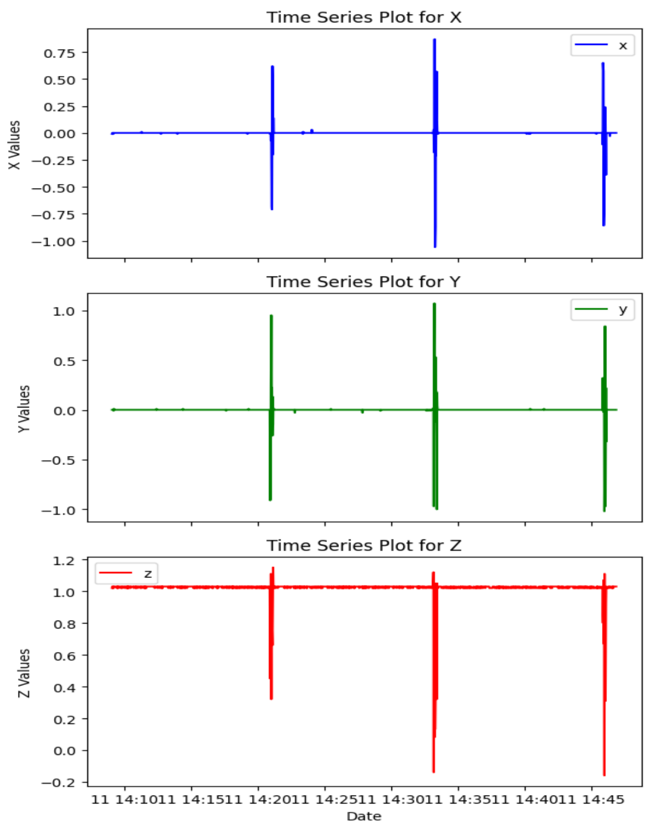
2	Question: Run each cell to produce output. Follow instructions in the notebook to complete codes in some of the cells. Convert the notebook to PDF from menu File > Save and Export Notebook As > PDF. Convert this activity sheet to PDF and merge with the notebook PDF. Answer: There is no answer to write 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, I can, I can modify the header of the csv, I can drop some columns if I feel I don't need it, I can convert the csv to json file to upload to the firebase. There are many things I can do with Pandas, and it depends on the application.
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: Handle missing values is when we decide what to do with values that are missing. One simplest method is just to drop it, granted if we have sufficient data. Other basic methods are to fill those missing data with mean and median. The choice between mean and median depends on our data. Mean or average is suitable for normally distributed data but are sensitive to outliers. Median or the middle value, suitable for skewed data but its less sensitive to small changes. The are other advanced techniques which use machine learning, but that will be beyond the scope of this task.

Pass Task: Store data to cloud

Q2.

I am simulating an earthquake at 10 minutes interval. I recorded the data for 37 minutes and here is the graph showing the outcome.

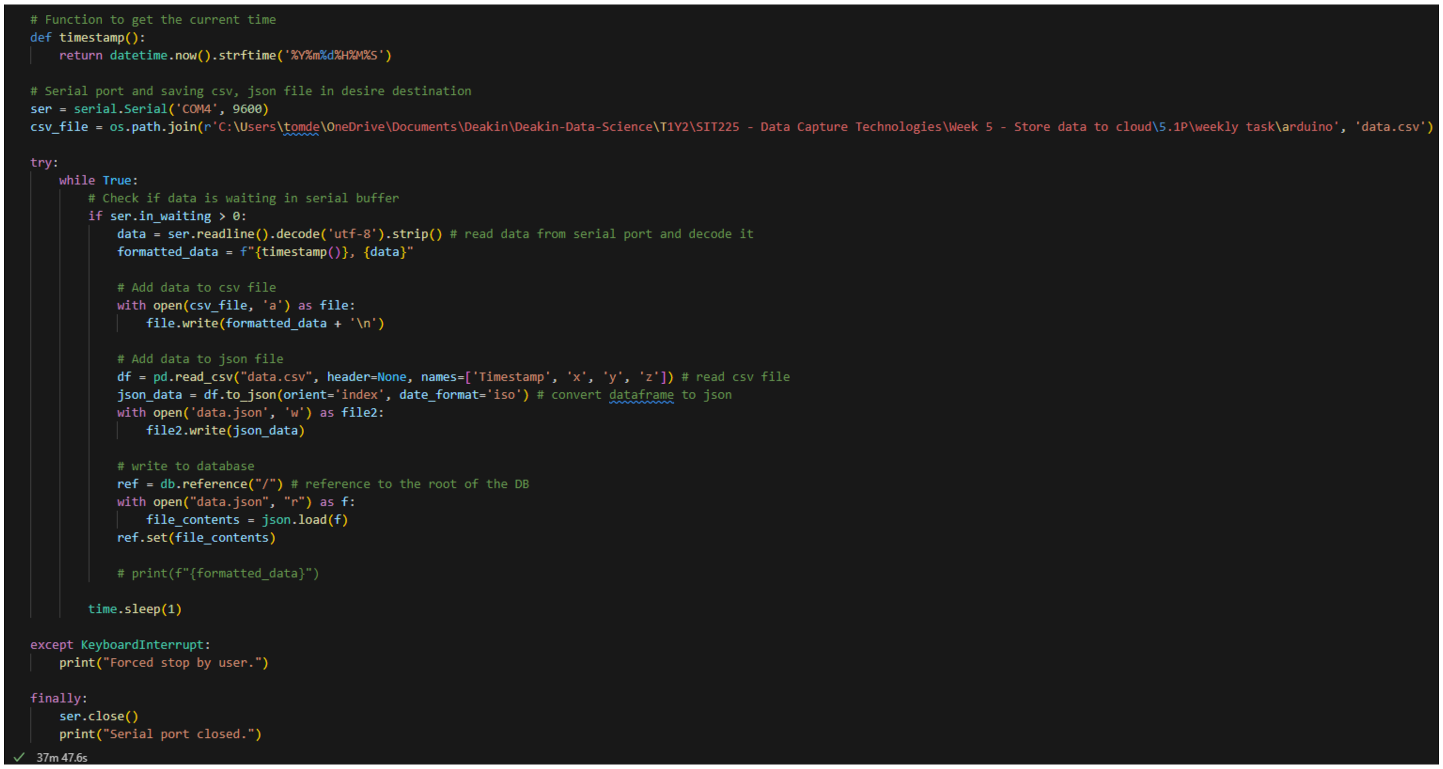


This graph shows the 3 axis or 3 rotations for every time the sensor is detecting large rotations.

Q3.

```
#include <Arduino_LSM6DS3.h>

float x, y, z;

void setup() {
  Serial.begin(9600); // set baud rate
  while (!Serial);  // wait for port to init
//  Serial.println("Started");

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

//  Serial.println(
//    "Accelerometer sample rate = "
//    + String(IMU.accelerationSampleRate()) + " Hz");
}

void loop() {
  // read accelero data
  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);
  }

  Serial.println(String(x) + "," + String(y) + "," + String(z));
  delay(1000); // delay 1s
}
```

The code above is the Arduino code for reading the rotations of each axis and sending it through the Serial.



```python
# Function to get the current time
def timestamp():
    return datetime.now().strftime('%Y%m%d%H%M%S')

# Serial port and saving csv, json file in desire destination
ser = serial.Serial('COM4', 9600)
csv_file = os.path.join(r'C:\Users\tomde\OneDrive\Documents\Deakin\Deakin-Data-Science\T1Y2\SIT225 - Data Capture Technologies\Week 5 - Store data to cloud\5.1P\weekly task\arduino', 'data.csv')

try:
    while True:
        # Check if data is waiting in serial buffer
        if ser.in_waiting > 0:
            data = ser.readline().decode('utf-8').strip() # read data from serial port and decode it
            formatted_data = f"{timestamp()}, {data}"

            # Add data to csv file
            with open(csv_file, 'a') as file:
                file.write(formatted_data + '\n')

            # Add data to json file
            df = pd.read_csv("data.csv", header=None, names=['Timestamp', 'x', 'y', 'z']) # read csv file
            json_data = df.to_json(orient='index', date_format='iso') # convert dataframe to json
            with open('data.json', 'w') as file2:
                file2.write(json_data)

            # write to database
            ref = db.reference("/") # reference to the root of the DB
            with open("data.json", "r") as f:
                file_contents = json.load(f)
            ref.set(file_contents)

            # print(f"{formatted_data}")

        time.sleep(1)

except KeyboardInterrupt:
    print("Forced stop by user.")

finally:
    ser.close()
    print("Serial port closed.")
✓ 37m 47.6s
```

This code runs continuously for 37 minutes and 47 seconds. It reads the data (x,y,z) from the Serial, then saves it into a csv and json file along with the timestamp. Using the json file, I continuously upload the data to the Firebase database.

Q4.

https://www.youtube.com/watch?v=eUz0PqXwnzU

Q5.

https://github.com/tomadonna1/SIT225_2024T2