Student name: Anish Bansal

Student ID: 221435713

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

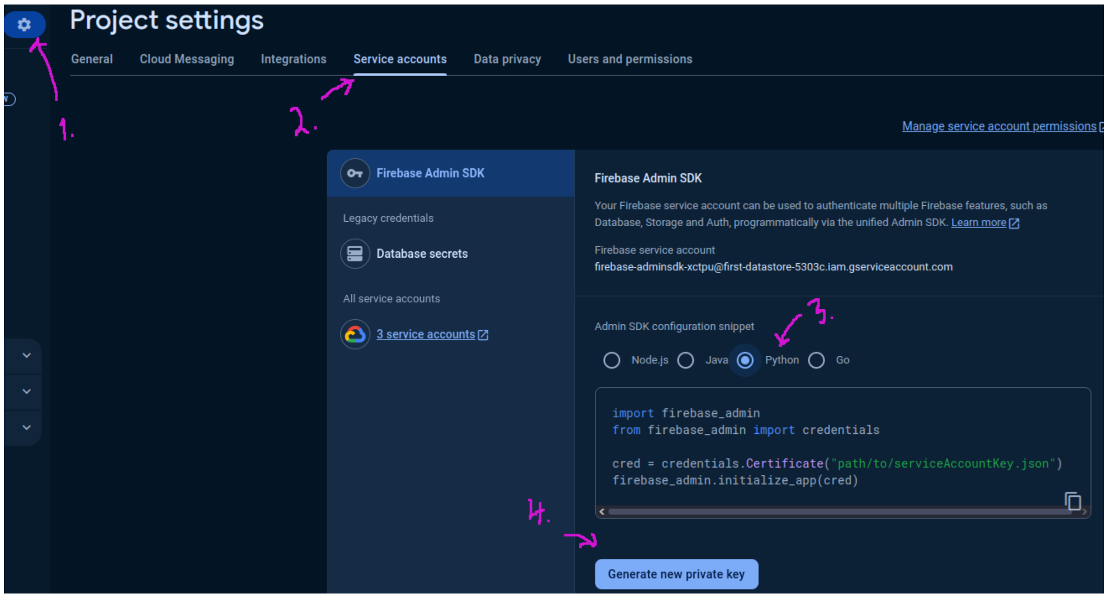
Steps

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

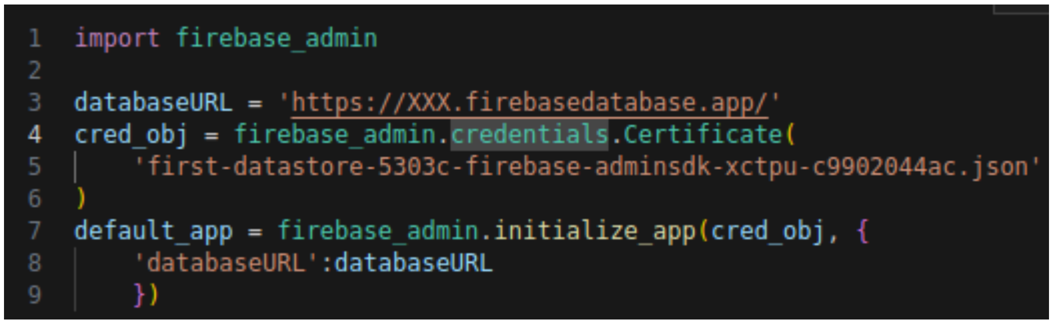
Firebase service account
firebase-adminsdk-xctpu@first-datastore-5303c.iam.gserviceaccount.com

Admin SDK configuration snippet

Node.js Java [✓ Python] Go

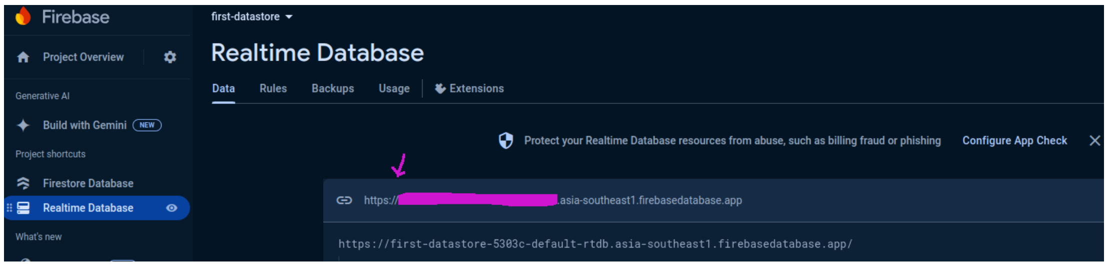
```python
import firebase_admin
from firebase_admin import credentials

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.



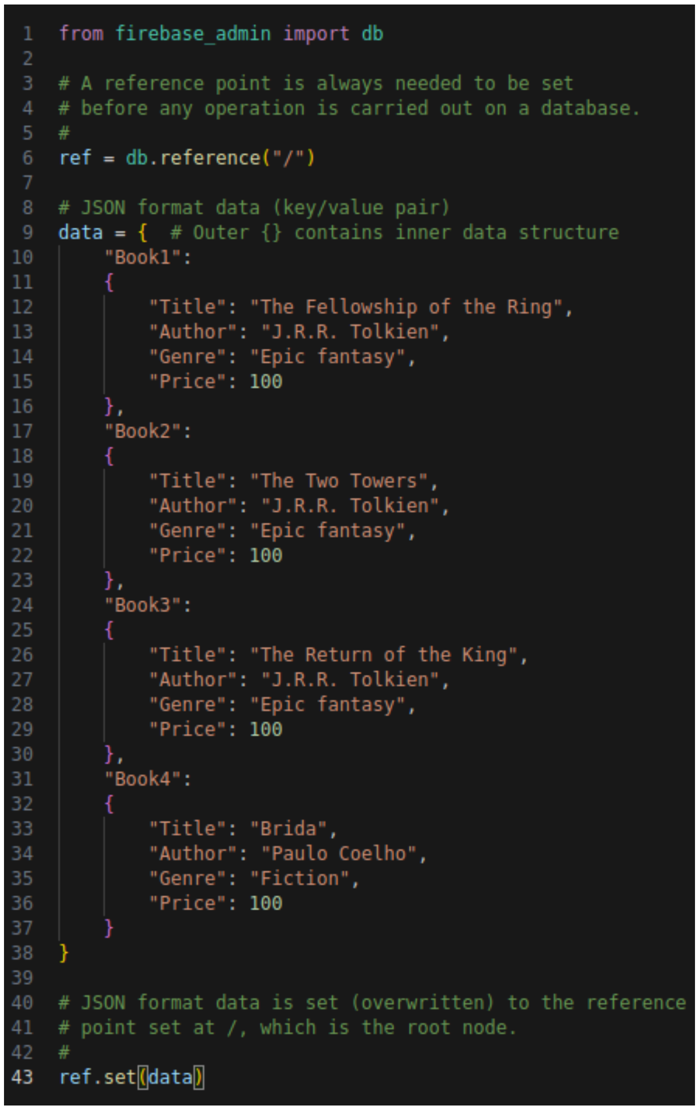
Protect your Realtime Database resources from abuse, such as billing fraud or phishing

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.

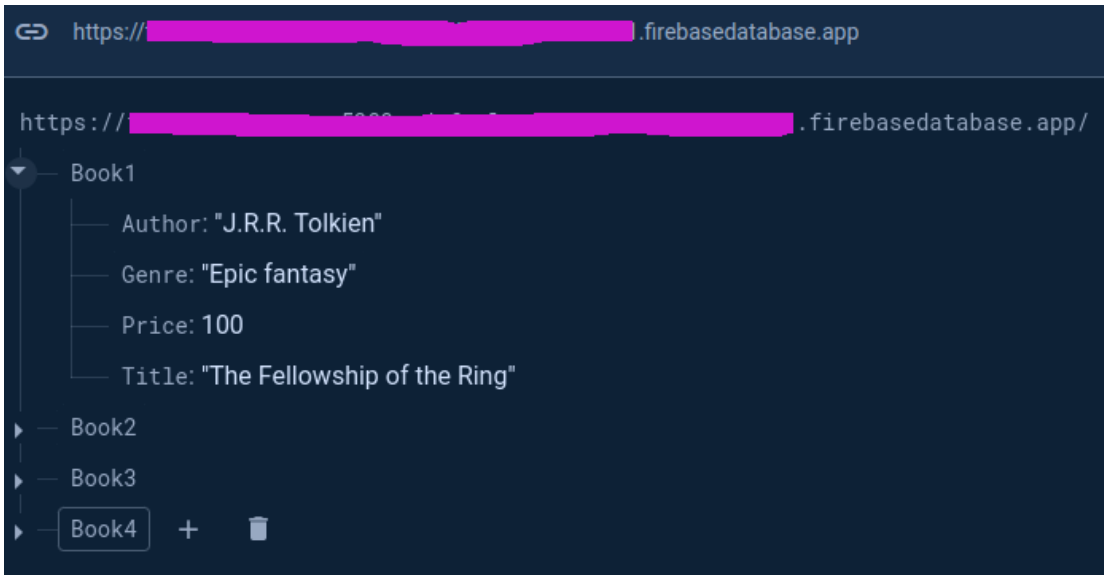
5 **Write to database Using the set() Function**:
We set the reference to the root of the database (or we could also set it to a key value or child key value). Data needs to be in JSON format as below.



```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)
```

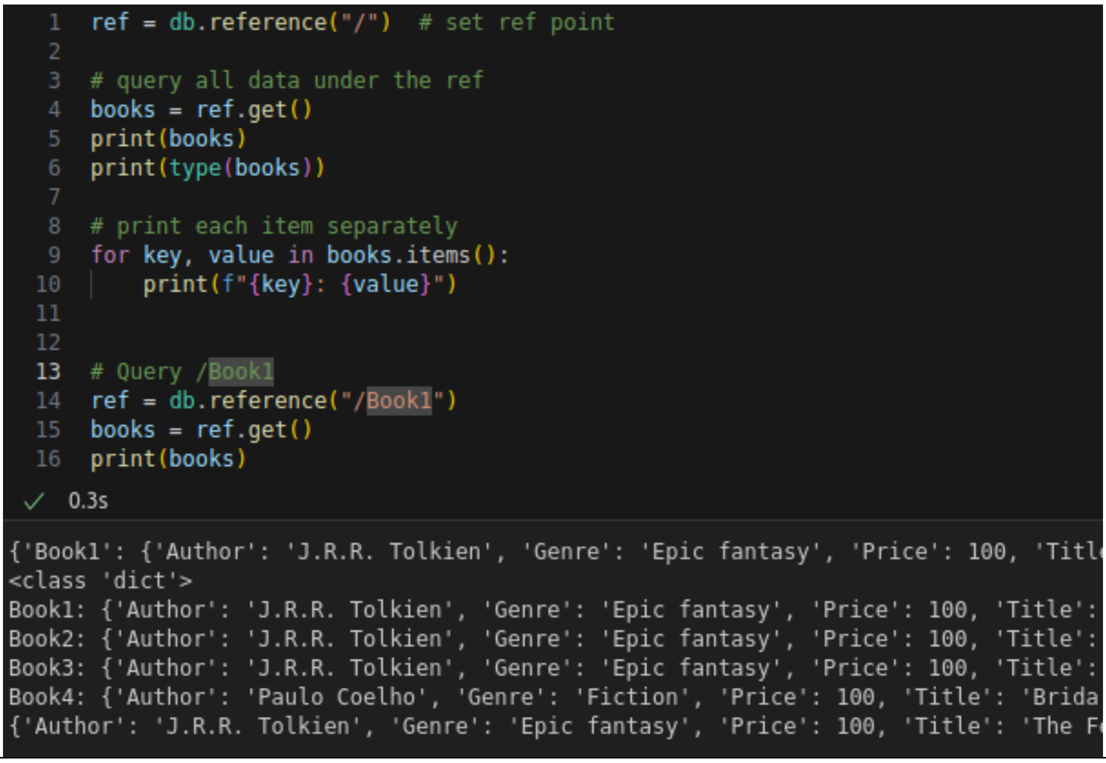
A reference point always needed to be set where the data read/write will take place. In the code above, the reference point is set at the root of the NoSQL Document, where consider the database is a JSON tree and / is the root node of the tree). The set() function writes (overwrites) data at the set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



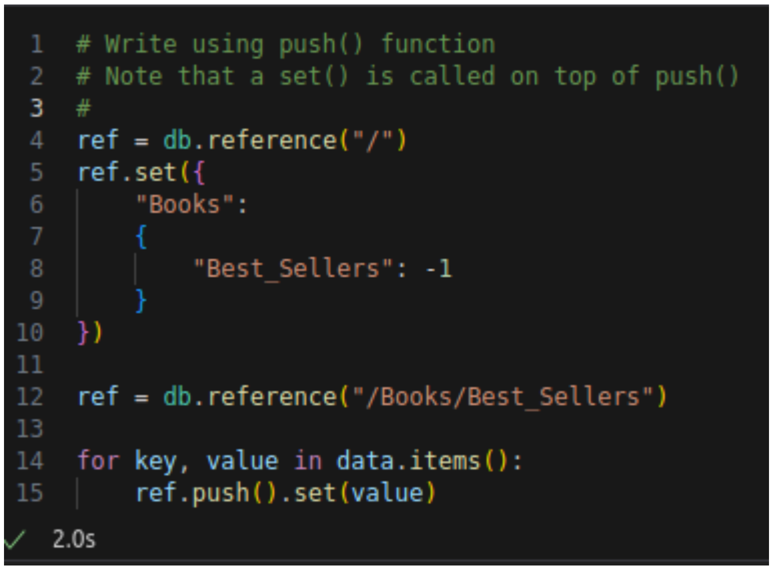
```python
ref = db.reference("/")  # set ref point

# 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
```

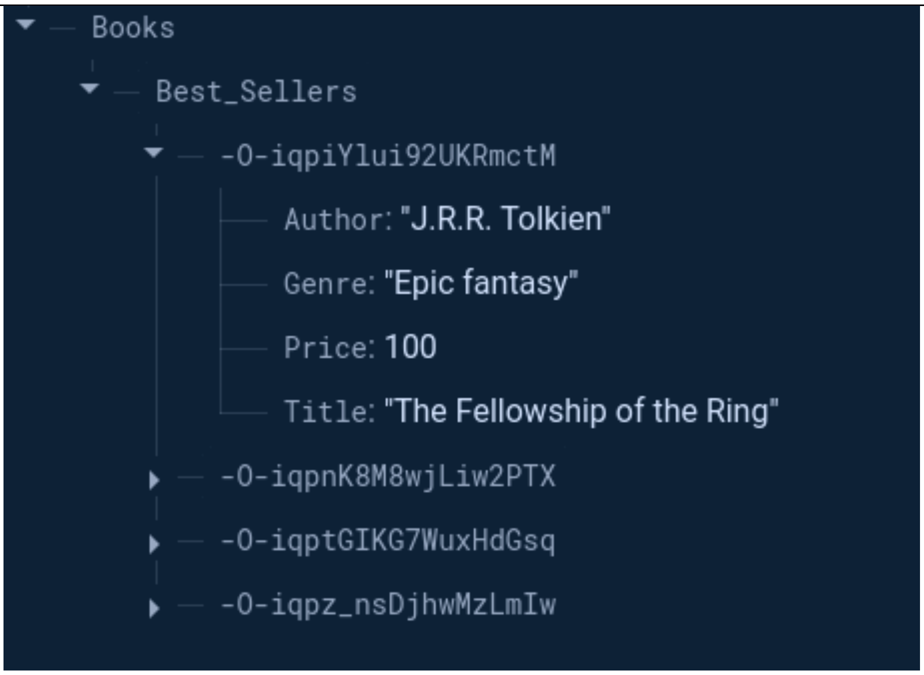
Consider the reference set in line 1 and the output compared to the reference set at line 14 and the bottom output line to understand the use of db.reference() and ref.get().

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.



```
# 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
```

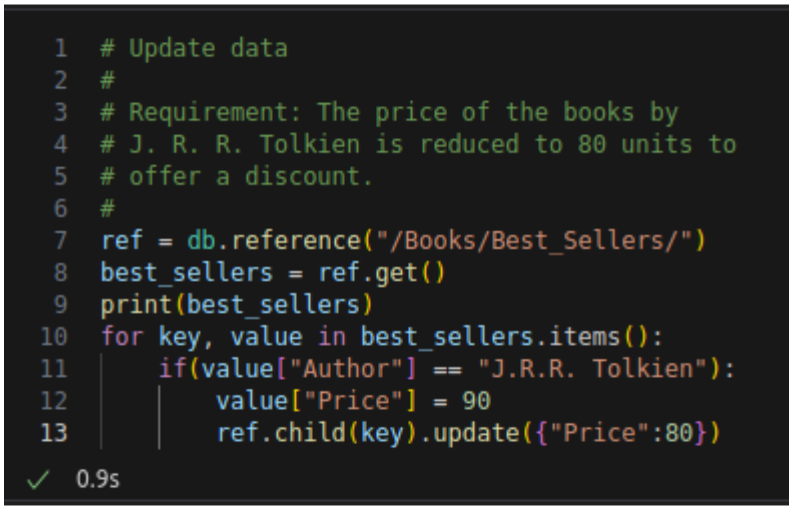
The output will reset the previous data set in / node. The current data is shown below.



```
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.

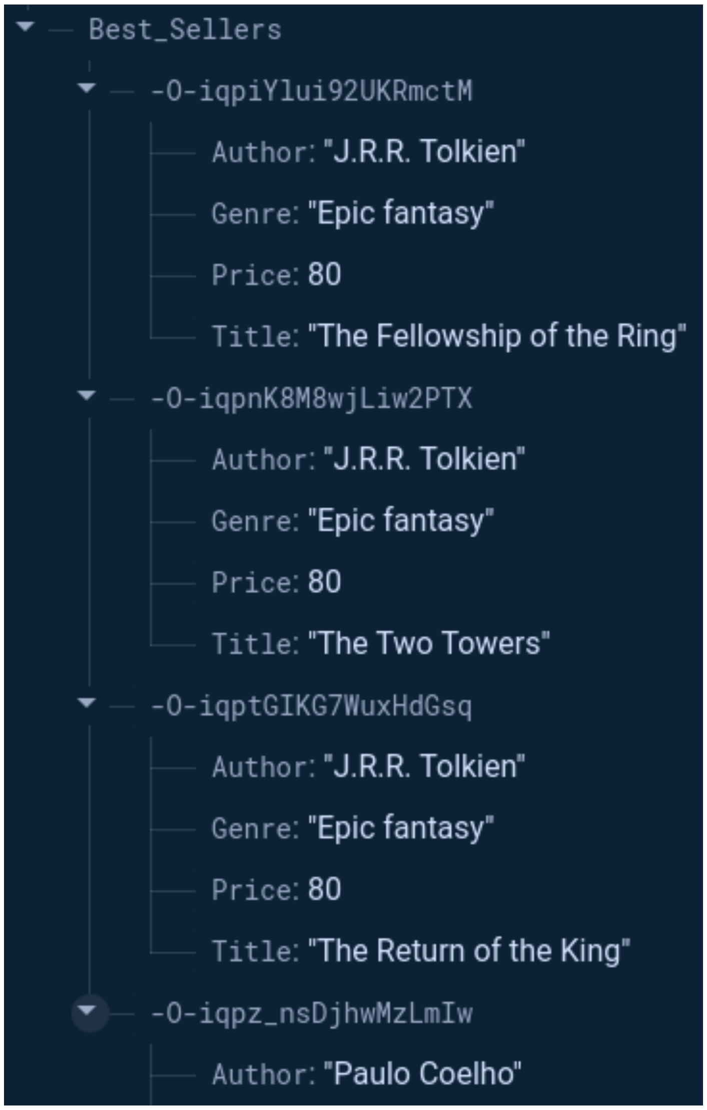
8 **Update data**:
Let's say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.



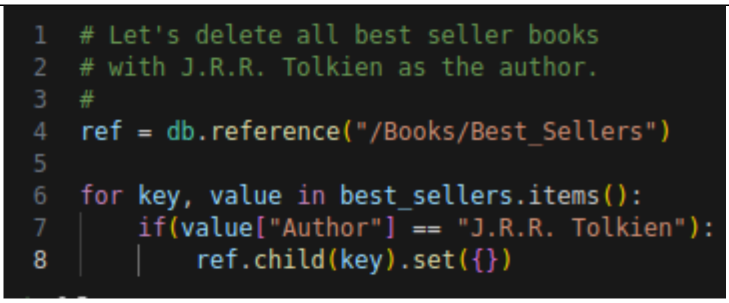
```python
# Update data
#
# Requirement: The price of the books by
# J. R. R. Tolkien is reduced to 80 units to
# offer a discount.
#
ref = db.reference("/Books/Best_Sellers/")
best_sellers = ref.get()
print(best_sellers)
for key, value in best_sellers.items():
    if(value["Author"] == "J.R.R. Tolkien"):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

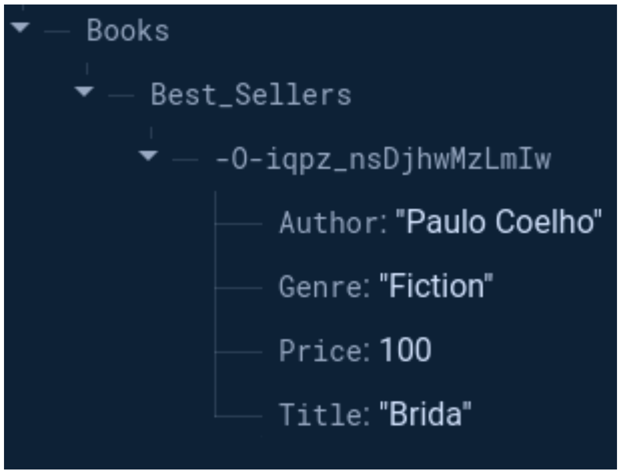
	child under the ref node, so ref.child(key) is used in line 13. The output is shown below with a discounted price. 
9	**Delete data**: Let's delete all bestseller books with J.R.R. Tolkien as the author. You can locate the node using db.reference() (line 4) and then locate specific record (for loop in 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.

```
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"
```



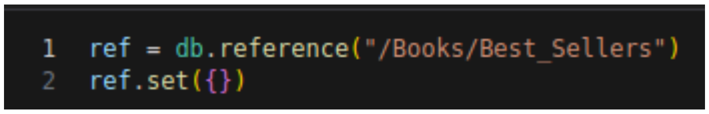
```python
# Let's delete all best seller books
# with J.R.R. Tolkien as the author.
#
ref = db.reference("/Books/Best_Sellers")

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: <Your answer> For a DHT22 sensor, which measures temperature and humidity, a JSON structure can be designed to include the following:

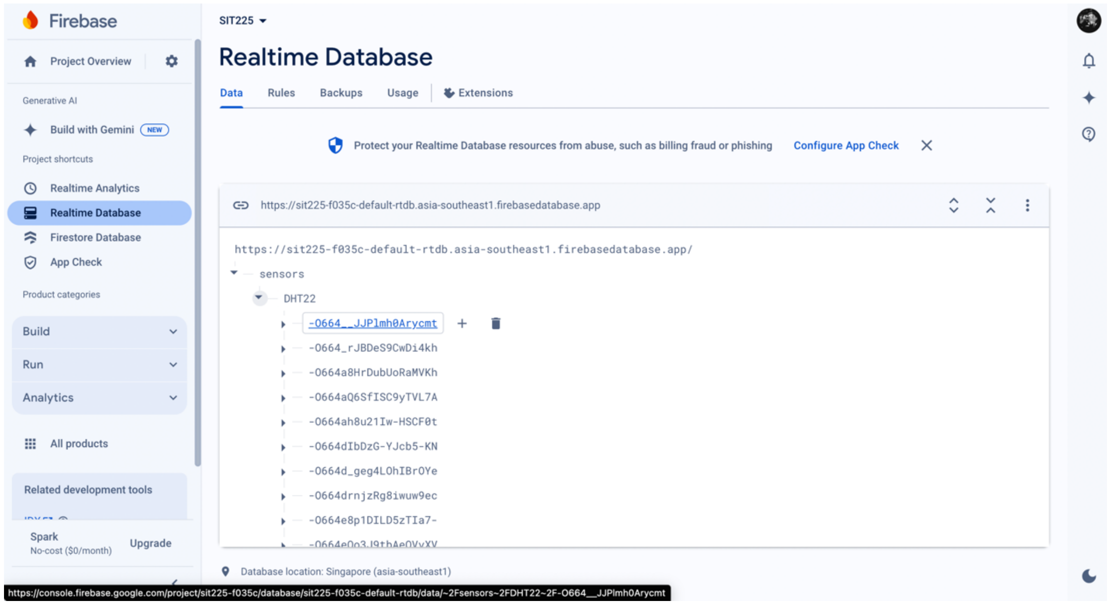
```
{
   "sensor_name": "DHT22",
   "timestamp": "20240903181118",
   "data": {
      "temperature": 24.5,
      "humidity": 55.2
   }
}
```

Justification:

- **Modularity**: This JSON structure allows you to easily add more sensor data or handle multiple sensors. Each sensor has a unique sensor_name and timestamp.
- **Compatibility**: This format is flexible and works well with other sensors like accelerometers, which have multiple variables (x, y, z).
- **Efficiency**: Storing sensor data in a consistent way helps with querying and storing large amounts of time-series data.

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

Answer: <Your answer>



Realtime Database

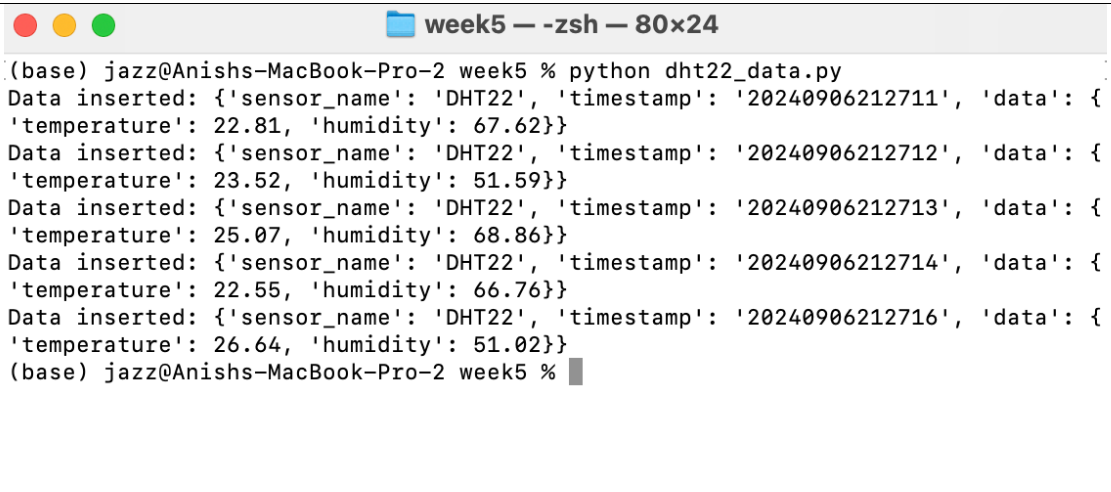
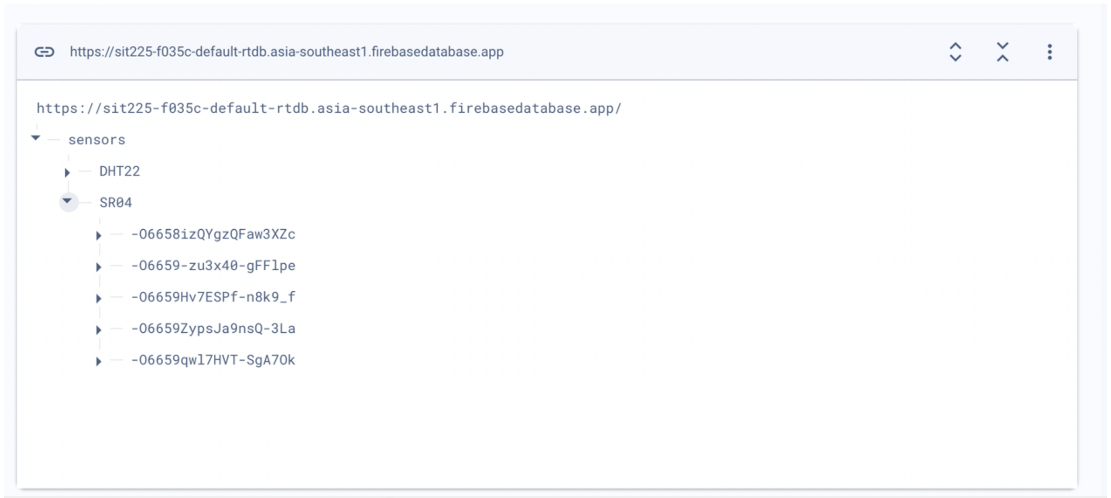
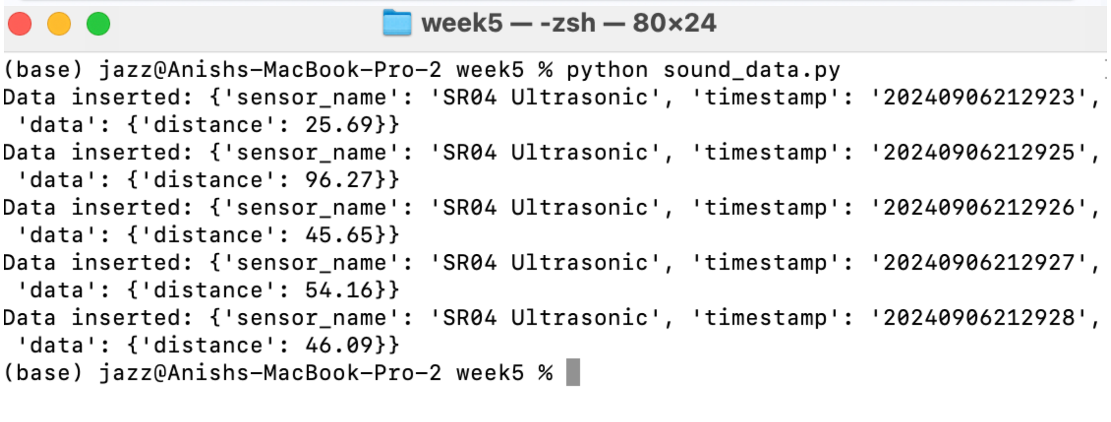
Protect your Realtime Database resources from abuse, such as billing fraud or phishing

https://sit225-f035c-default-rtdb.asia-southeast1.firebasedatabase.app

```
https://sit225-f035c-default-rtdb.asia-southeast1.firebasedatabase.app/
sensors
   DHT22
      -O664__JJPlmh0Arycmt
      -O664_rJBDeS9CwDi4kh
      -O664a8HrDubUoRaMVKh
      -O664aQ6SfISC9yTVL7A
      -O664ah8u21Iw-HSCF0t
      -O664dIbDzG-YJcb5-KN
      -O664d_geg4LOhIBrOYe
      -O664drnjzRg8iwuw9ec
      -O664e8p1DILD5zTIa7-
      -O664eOo3J9thAeOVxXV
```

Database location: Singapore (asia-southeast1)

https://console.firebase.google.com/project/sit225-f035c/database/sit225-f035c-default-rtdb/data/~2Fsensors~2FDHT22~2F-O664__JJPlmh0Arycmt

	
12	Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here. Answer: <Your answer>  
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-

```
(base) jazz@Anishs-MacBook-Pro-2 week5 % python dht22_data.py
Data inserted: {'sensor_name': 'DHT22', 'timestamp': '20240906212711', 'data': {
'temperature': 22.81, 'humidity': 67.62}}
Data inserted: {'sensor_name': 'DHT22', 'timestamp': '20240906212712', 'data': {
'temperature': 23.52, 'humidity': 51.59}}
Data inserted: {'sensor_name': 'DHT22', 'timestamp': '20240906212713', 'data': {
'temperature': 25.07, 'humidity': 68.86}}
Data inserted: {'sensor_name': 'DHT22', 'timestamp': '20240906212714', 'data': {
'temperature': 22.55, 'humidity': 66.76}}
Data inserted: {'sensor_name': 'DHT22', 'timestamp': '20240906212716', 'data': {
'temperature': 26.64, 'humidity': 51.02}}
(base) jazz@Anishs-MacBook-Pro-2 week5 %
```

```
https://sit225-f035c-default-rtdb.asia-southeast1.firebasedatabase.app/
sensors
  DHT22
  SR04
    -O6658izQYgzQFaw3XZc
    -O6659-zu3x40-gFFlpe
    -O6659Hv7ESPf-n8k9_f
    -O6659ZypsJa9nsQ-3La
    -O6659qwl7HVT-SgA7Ok
```

```
(base) jazz@Anishs-MacBook-Pro-2 week5 % python sound_data.py
Data inserted: {'sensor_name': 'SR04 Ultrasonic', 'timestamp': '20240906212923',
 'data': {'distance': 25.69}}
Data inserted: {'sensor_name': 'SR04 Ultrasonic', 'timestamp': '20240906212925',
 'data': {'distance': 96.27}}
Data inserted: {'sensor_name': 'SR04 Ultrasonic', 'timestamp': '20240906212926',
 'data': {'distance': 45.65}}
Data inserted: {'sensor_name': 'SR04 Ultrasonic', 'timestamp': '20240906212927',
 'data': {'distance': 54.16}}
Data inserted: {'sensor_name': 'SR04 Ultrasonic', 'timestamp': '20240906212928',
 'data': {'distance': 46.09}}
(base) jazz@Anishs-MacBook-Pro-2 week5 %
```

	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: <Your answer> Firebase Realtime Database provides event-driven functionality that allows tracking changes to data in real-time. These events are particularly useful in IoT scenarios where sensors, like DHT22 or ultrasonic sensors, send data directly to the database. The key events generated include:
	1. **onCreate** – Triggered when new data is added.
	2. **onUpdate** – Triggered when existing data is modified.
	3. **onDelete** – Triggered when data is removed.
	4. **onWrite** – A general event that captures any write operation, including create, update, and delete.
	These events allow for the continuous monitoring of changes in the database and are essential for applications like real-time dashboards or monitoring systems.
	Using Python, you can handle these events with the Firebase Admin SDK. By setting a reference to a part of the database, you can listen for changes and act upon them. Here’s an example:
	
	import firebase_admin
	from firebase_admin import credentials, db
	
	# Initialize Firebase app with credentials
	cred = credentials.Certificate('path_to_your_firebase_admin_sdk_key.json')
	firebase_admin.initialize_app(cred, {
	    'databaseURL': 'https://your-database-name.firebaseio.com/'
	})
	
	# Reference to a part of the database
	ref = db.reference('sensors/DHT22')
	
	# Listener function to handle database events
	def listener(event):
	    print(f"Event type: {event.event_type}") # 'put' or 'patch'
	    print(f"Path: {event.path}") # Location of the change
	    print(f"Data: {event.data}") # Updated data
	
	# Start listening for changes
	ref.listen(listener)

	This code sets up a listener for changes at the /sensors/DHT22 path, printing details whenever data is created, updated, or deleted. The listener function captures event types and data, which can be used to update a dashboard or trigger alerts in real time. In summary, Firebase events make it easy to build real-time applications by tracking database changes as they happen, and Python SDK offers a flexible way to integrate this functionality into a broader monitoring system or dashboard application.

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.

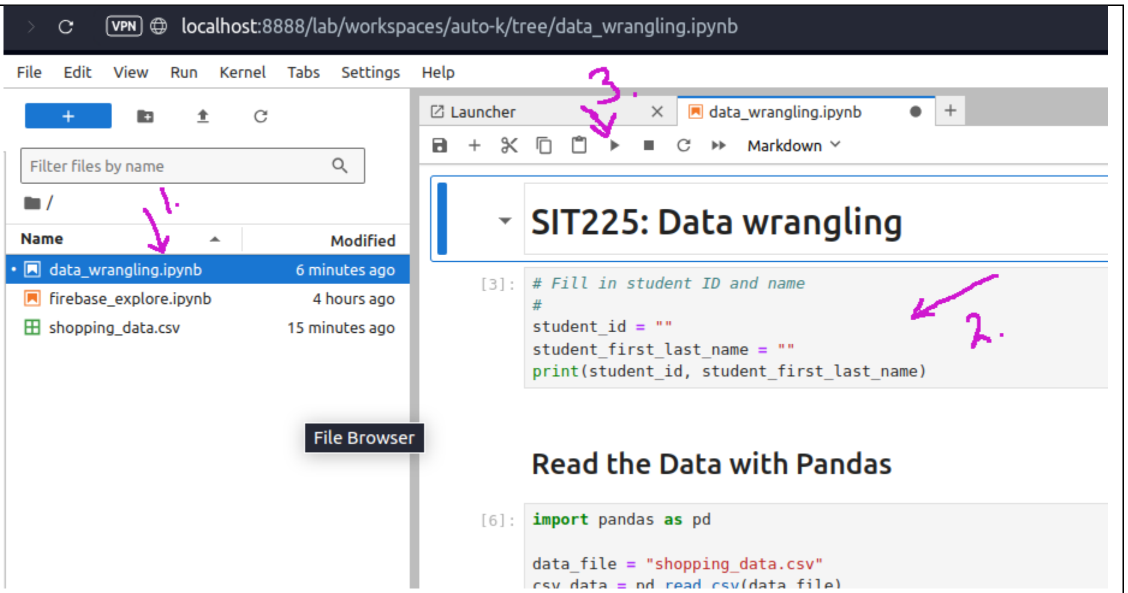
Software Required

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
	Launcher
	data_wrangling.ipynb
	Markdown
	File Browser
	1.
	2.
	3.
	Name / Modified
	data_wrangling.ipynb 6 minutes ago
	firebase_explore.ipynb 4 hours ago
	shopping_data.csv 15 minutes ago
	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: <Your answer>Yes, Pandas can be used to read the sensor CSV data generated earlier. Using Pandas, I can read the CSV data efficiently and perform data wrangling operations such as cleaning, manipulation, and analysis. For example, if I have temperature and humidity sensor data (such as from a DHT22 sensor), I would load the CSV file using pd.read_csv(). If the data requires modification, I can handle missing values, filter outliers, and convert timestamps into a datetime format. Additionally, I can create visualizations like line graphs to observe trends over time. Pandas makes it easier to process and modify sensor data to suit different analysis requirements. In case the sensor data contains missing values or incorrectly

	formatted values, I can modify the CSV file by cleaning and imputing missing data using functions like fillna() or interpolate()
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: <Your answer>In the "Handling Missing Value" section of the notebook, the main focus is on methods to handle missing data, which is common in real-world datasets. Different imputation methods are used depending on the nature of the missing data. The common methods include: • **Deleting Data:** Rows or columns with missing data can be removed. This is simple but may lead to loss of valuable data. • **Imputation Methods:** These include replacing missing values with the mean, median, or mode. This is suitable for continuous numerical data, especially when the data distribution has minimal outliers. • **Linear Interpolation:** This method is used for time series data, where missing values are filled by linearly interpolating between known values. • **Multiple Imputation:** In this method, missing values are filled by considering the relationships between various variables in the dataset, creating several possible datasets, and averaging the results. These methods are applicable depending on the dataset. For example, mean imputation is useful for sensor data with stable readings over time, while linear interpolation works well with time series sensor data like temperature or humidity readings.

1. Data Collection Justification:

I chose to collect gyroscope data for 30 minutes to ensure a sufficient amount of data to observe meaningful patterns. Since gyroscopes measure rotational movement, I aimed to capture a range of activities, including slow and fast rotations, as well as stationary periods, to observe the sensor's sensitivity and accuracy over time. The choice of 30 minutes was based on capturing enough variation in the data to identify trends.

2. Data Rate of the Gyroscope Module:

The data rate of the LSM6DS3 gyroscope module can be configured to operate at different speeds, ranging from 12.5 Hz to 1.66 kHz. For this experiment, a sampling rate of 100 Hz was chosen, which is frequent enough to capture fine movements but not so high that it would overwhelm the system or generate excessive data. This balance ensures accurate movement detection while maintaining manageable data output.

3. Observing Data Changing Patterns in Plots

Single Variable Plot Observation (X, Y, Z):

When observing each axis (X, Y, Z) separately, there are fluctuations and noticeable changes over time. These changes represent the sensor's movements. For example, repeating peaks and valleys can indicate a consistent movement, such as rotation along that particular axis. In cases of sudden, steep increases or decreases, it can reflect sharp rotations or changes in the gyroscope's orientation.

Combined Plot (X, Y, Z):

By observing the combined plot of X, Y, and Z values, it's possible to detect more complex patterns in motion. If all three axes display a similar peak or change simultaneously, this suggests that the movement equally affects all three dimensions (e.g., a shake or tilt). Additionally, when one axis increases while another decreases, it can indicate rotational movement in space.

Hypothesis and Analysis:

The hypothesis is that when the gyroscope is stationary, there will be little to no variation in the X, Y, or Z-axis data. However, when it is in motion, the sensor data will reflect consistent fluctuations and peaks depending on the intensity and direction of the movement.

Based on the collected gyroscope data, the hypothesis holds true for consistent movement, as the X, Y, and Z variables show predictable patterns and peaks over time when motion occurs. In periods of rest, there is little to no significant variation in the data. For sudden movements, there are visible spikes in the graphs, confirming that the sensor captures changes effectively.

If the data shows irregular or unpredictable patterns, external interference or noise may have affected the sensor, leading to outliers or unexpected results.

Arduino Sketch:

1. **Purpose**: The Arduino sketch is designed to read real-time data from the built-in gyroscope sensor (LSM6DS3) on the Arduino Nano 33 IoT board. It captures the rotational velocity data along the X, Y, and Z axes.
2. **Reading Data**: The gyroscope continuously measures rotational velocity, which is obtained through the IMU.readGyroscope(x, y, z) function in the loop. Each reading gives the angular velocity in degrees per second for each axis (X, Y, Z).
3. **Serial Communication**: Once the gyroscope data is captured, the values of X, Y, and Z are sent to the connected Python script over serial communication using the Serial.print function. The values are formatted in such a way that the Python script can easily parse and process them.
4. **Loop Execution**: The loop ensures that data is sent continuously, and the sensor readings keep updating as long as the program is running.

Python Script:

1. **Purpose**: The Python script runs in parallel to receive the gyroscope data from the Arduino over a serial connection and upload it to Firebase in real-time. It also records the system's timestamp at the moment of data reception.
2. **Serial Communication Setup**: The script uses the pyserial library to open the serial port (matching the one used by the Arduino) and listens for incoming data. This is configured using serial.Serial(port, baudrate).
3. **Receiving Data**: The script reads the X, Y, and Z gyroscope values sent by the Arduino. For each set of values received, it appends a timestamp (retrieved using Python’s time module) to the data to capture the exact moment the data was received.
4. **JSON Message**: After receiving the gyroscope readings and the timestamp, the data is converted into a JSON object, containing keys for timestamp, x, y, and z.
5. **Uploading to Firebase**: The JSON data is uploaded to Firebase Realtime Database using the Firebase Admin SDK. Each data point is stored under a unique key (using push), ensuring that every sample has a timestamp and the corresponding X, Y, Z values.

6. **Continuous Loop**: The script continues running, receiving data and uploading it to Firebase until the user manually stops the process. The loop ensures the system keeps updating Firebase in real-time with the latest gyroscope readings.

Firebase Query and CSV Generation:

1. **Querying Firebase**: After enough data is collected, the Python script queries Firebase to retrieve the stored gyroscope data. The Firebase Admin SDK is used to retrieve the data, which is returned as a JSON array.
2. **Converting Data to CSV**: Once the data is retrieved, it is processed using Python's Pandas library. Each JSON object (representing a gyroscope sample) is converted into a CSV row with columns for timestamp, X, Y, and Z values.
3. **Data Cleaning**: During the CSV creation, any missing or non-numerical data is either removed or handled accordingly to ensure the final dataset is clean and ready for analysis.
4. **Storing Data in CSV**: The cleaned and formatted data is saved as a CSV file using Pandas, which allows for easy storage and future use for analysis.