ITLPA701: PYTHON AND FUNDAMENTALS OF AI

LEARNING UNIT 4–DEVELOP PYTHON SCRIPT

Mr. Etienne NTAMBARA

ntambaraienne94@gmail.com

Assistant Lecturer in ICT Department

Rwanda Polytechnic, IPRC-HUYE

Homepage: https://94etienne.github.io/profile/

Learning hours: 15

IPRC HUYE

Learning Outcome 4.1: Perform File handling

4.1.1. Practice to read file

File handling allows programs to **open, read, write, and delete** files on a system.

Before reading a file, ensure it exists and that you have the necessary read permissions.

- Below, we will explore these operations with clear examples in **Python**.

4.1.1.1 Open file

- In Python, use the **open()** function to open a file. The default mode is "**r**" (read mode).

Example: file = open("example.txt", "r") # Opens the file in read mode

4.1.1.1 Reading a File permission

Once the file is opened, you can read its contents using different methods:

```python
readcontent.py > ...
1    # Read the entire file content
2    with open("example.txt", "r") as file:
3        content = file.read()
4        print(content)
5
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS D:\RP_TEACHING_JOB\teaching_notes_etienne\Python\practice\unit4> python readcontent.py
Helle LEVEL 7 IT from RP-HUYE COLLEGE
PS D:\RP_TEACHING_JOB\teaching_notes_etienne\Python\practice\unit4>
```

Other ways to read a file

1. Read one line at a time:

```python
with open("example.txt", "r") as file:
    line = file.readline()  # Reads one line at a time
    print(line)
```

2. Read file as a list of lines:

```python
with open("example.txt", "r") as file:
    lines = file.readlines()  # Returns a list of lines
    print(lines)
```

4.1.2. Practice to write/create file

You can create and write to a file using "w" (write mode) or "a" (append mode).

1. Creating a New File

If the file does not exist, "w" mode creates a new file.

```python
with open("newfile.txt", "w") as file:
    file.write("This is a new file.\n")
```

2. Writing to an Existing File

Overwrite content ("w" mode)

3. Append new content ("a" mode)

```python
with open("newfile.txt", "a") as file:
    file.write("This text will be added to the existing file.\n")
```

```python
with open("newfile.txt", "w") as file:
    file.write("Overwritten content.\n")
```

4.1.3. Pactice to delete file

Deleting Files and Folders: Python provides the **os** and **shutil** modules to handle file deletion.

1. **Removing a File:** Use **os.remove()** to delete a file:

2. **Checking if a File Exists:**

Before deleting a file, check if it exists:

```python
import os

file_path = "example.txt"

if os.path.exists(file_path):
    print("File exists.")
else:
    print("File does not exist.")
```

```python
import os

if os.path.exists("newfile.txt"):
    os.remove("newfile.txt")
    print("File deleted successfully.")
else:
    print("File does not exist.")
```

3. Deleting a Folder

1. Remove an empty folder using **os.rmdir()**

```python
os.rmdir("empty_folder")  # The folder must be empty
```

2. Remove a folder with files using **shutil.rmtree()**

```python
import shutil

shutil.rmtree("folder_with_files")  # Deletes folder and all contents
```

Learning Outcome 4.2: Determine Python library

- When working with data science, machine learning, and numerical computing in Python, certain libraries make tasks easier and more efficient.

- **The most important libraries**:

1. Numpy
2. Pandas
3. MultplotLib
4. SciPy
5. Scikit-Learn

1. Numpy

Purpose

NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

Features:

✓ Supports N-dimensional arrays (**ndarray**).

✓ Provides mathematical and statistical functions.

✓ Used for linear algebra operations.

✓ Efficient memory handling.

1. Numpy

Interact with numpy need to install it via cmd or jupyter notebook by:

Pip install numpy

```
C:\Users\user>pip install numpy
```

Example of numpy python code:

```python
import numpy as np

# Creating an array
arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)

# Performing operations
print("Mean:", np.mean(arr))
print("Standard Deviation:", np.std(arr))
print("Square root of elements:", np.sqrt(arr))
```

2. Pandas (Data Manipulation and Analysis)

Purpose:

Pandas is used for data manipulation and analysis. It provides data structures like Series (1D) and DataFrame (2D) to handle structured data easily.

Features:

DataFrame: A table-like structure to store and manipulate data.

Handles missing values easily.

Supports filtering, grouping, and merging datasets.

Works well with NumPy and visualization libraries.

2. Pandas (Data Manipulation and Analysis)

Example: Creating and manipulating a DataFrame

```python
import pandas as pd

# Creating a DataFrame
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [24, 27, 22],
    "Score": [85, 90, 88]
}

df = pd.DataFrame(data)
print(df)

# Filtering students with Score > 85
filtered_df = df[df["Score"] > 85]
print("Students with score > 85:\n", filtered_df)
```

```
      Name  Age  Score
0    Alice   24     85
1      Bob   27     90
2  Charlie   22     88
Students with score > 85:
      Name  Age  Score
1      Bob   27     90
2  Charlie   22     88
```

3. MultplotLib

- **Purpose:**
  Matplotlib is a library for creating static, animated, and interactive visualizations in Python.

- **Features:**

- Allows customization of plots.

- Supports different chart types (line, bar, scatter, histogram, etc.).

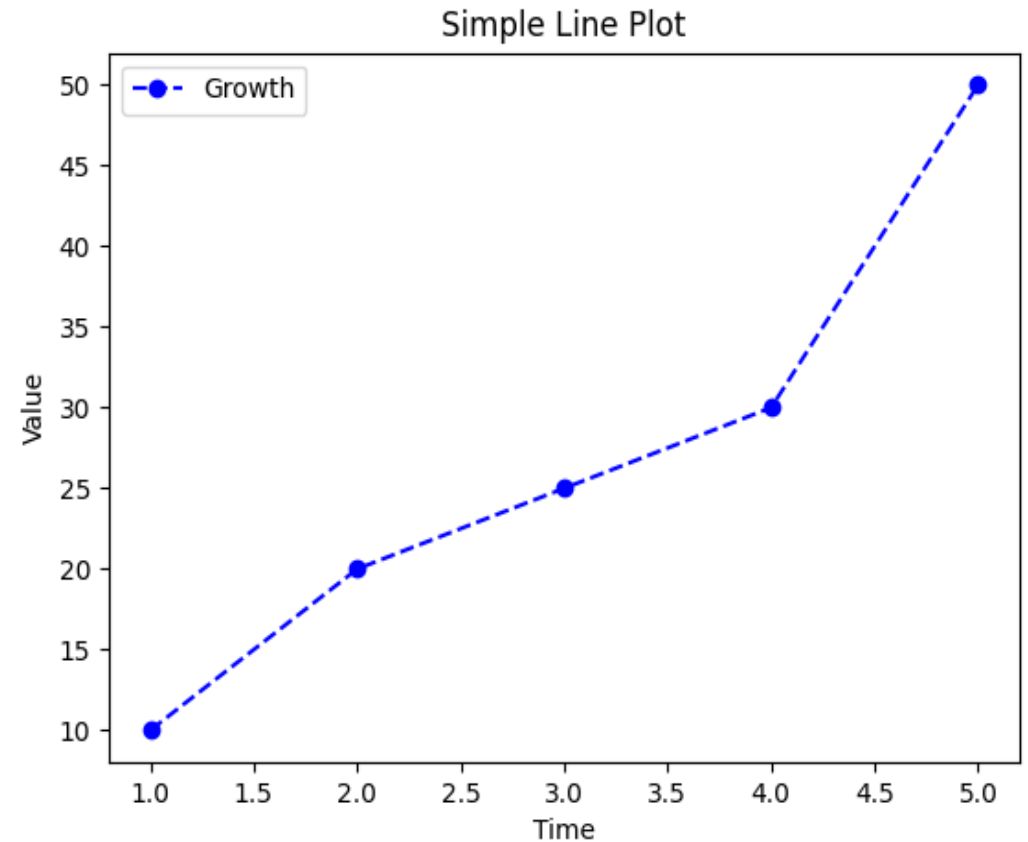- Works well with NumPy and Pandas.

3. MultplotLib

- **Example:** Plotting a simple line graph

```python
import matplotlib.pyplot as plt

# Sample data

x = [1, 2, 3, 4, 5]

y = [10, 20, 25, 30, 50]


# Creating a line plot

plt.plot(x, y, marker='o', linestyle='--', color='b', label="Growth")

plt.xlabel("Time")

plt.ylabel("Value")

plt.title("Simple Line Plot")

plt.legend()

plt.show()
```

4. SciPy (Scientific Computing)

Purpose:
SciPy builds on NumPy and provides additional functionalities for scientific computing, including optimization, integration, statistics, and signal processing.

Features:

- Supports optimization (finding minima/maxima).

- Provides statistical functions.

- Includes numerical integration and differential equations solvers.

4. SciPy (Scientific Computing)

- **Example:** Finding the minimum of a function using SciPy

```python
from scipy.optimize import minimize
import numpy as np


# Function to minimize
def func(x):
    return (x - 3) ** 2 + 4


# Finding the minimum
result = minimize(func, x0=np.array([0]))  # Initial guess at x=0


# Formatting the output to 2 decimal places
print("Optimal value of x: {:.2f}".format(result.x[0]))
print("Minimum function value: {:.2f}".format(result.fun))
```

```
Optimal value of x: 3.00
Minimum function value: 4.00
```

5. Scikit-Learn (Machine Learning Library)

Purpose:
Scikit-Learn is a powerful library for machine learning. It provides tools for supervised and unsupervised learning, model selection, and evaluation.

Features:

- Supports classification, regression, and clustering.

- Provides preprocessing and feature selection tools.

- Works well with NumPy and Pandas

Example: Training a simple classification model

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
# Sample dataset
X = [[1], [2], [3], [4], [5]]
y = [0, 0, 1, 1, 1]  # Labels (0 or 1)
# Scaling the features (important for Logistic Regression)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Training a logistic regression model
model = LogisticRegression(solver='liblinear')  # You can try different solvers like 'lbfgs', 'saga'
model.fit(X_scaled, y)
# Making predictions
y_pred = model.predict(X_scaled)
# Evaluating the model
accuracy = accuracy_score(y, y_pred)
print(f"Accuracy: {accuracy}")
```

✓ 0.0s

Accuracy: 1.0

Learning Outcome 4.3: Interact with database

When working with MySQL in Python, we use the **mysql-connector-python** library to establish connections and perform database operations.

- Below are the steps to interact with MySQL using Python, along with examples.

Dounload MSQL: [MySQL :: Download MySQL Installer](#)

Step by step: https://youtu.be/uj4OYk5nKCg?si=RLzrV1LtJz2Zo4xu

4.3.1 Python Mysql commands

1. **Install Driver**

Before connecting Python to MySQL, we need to install the MySQL Connector.

```
pip install mysql-connector-python
```

2. Test MySQL Connector

Once installed, we can test if the MySQL connector is working.

```python
import mysql.connector

print("MySQL Connector is working!")
```

If the script runs without errors, the connector is working fine.

3. Create Connection

To interact with a MySQL database, we first need to establish a connection.

```python
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password"
)

if conn.is_connected():
    print("Connected to MySQL successfully!")

conn.close()
```

4. Create Database

We can create a new database using SQL commands in Python.

```python
import mysql.connector

conn = mysql.connector.connect(host="localhost", user="root", password="password")
cursor = conn.cursor()
cursor.execute("CREATE DATABASE university_db")
print("Database created successfully!")

cursor.close()
conn.close()
```

5. Create Table

After creating the database, we create tables to store information.

```python
conn = mysql.connector.connect(host="localhost", user="root", password="password", database="university_db")
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    age INT,
    major VARCHAR(100)
)
""")
print("Table created successfully!")

cursor.close()
conn.close()
```

6. Insert Data

We insert records into the table using the INSERT INTO statement.

```python
cursor = conn.cursor()
sql = "INSERT INTO students (name, age, major) VALUES (%s, %s, %s)"
values = ("Alice", 22, "Computer Science")
cursor.execute(sql, values)
conn.commit()  # Save changes


print(cursor.rowcount, "record inserted.")


cursor.close()
conn.close()
```

7. Select Data

To fetch data from the database, we use the **SELECT** statement.

```python
cursor = conn.cursor()
cursor.execute("SELECT * FROM students")
result = cursor.fetchall()


for row in result:
    print(row)


cursor.close()
conn.close()
```

8. Delete Data

We can remove specific records using the **DELETE** statement.

```python
cursor = conn.cursor()

cursor.execute("DELETE FROM students WHERE name = 'Alice'")

conn.commit()


print(cursor.rowcount, "record(s) deleted.")


cursor.close()

conn.close()
```

9. Where Condition

To filter records, we use the **WHERE** clause.

```python
cursor = conn.cursor()
cursor.execute("SELECT * FROM students WHERE age > 20")
result = cursor.fetchall()


for row in result:
    print(row)


cursor.close()
conn.close()
```

10. Order By

To sort results, we use the **ORDER BY** clause.

```python
cursor = conn.cursor()
cursor.execute("SELECT * FROM students ORDER BY name ASC")
result = cursor.fetchall()

for row in result:
    print(row)

cursor.close()
conn.close()
```

11. Drop Table

To delete a table completely, we use **DROP TABLE**.

```python
cursor = conn.cursor()
cursor.execute("DROP TABLE students")
print("Table dropped successfully!")

cursor.close()
conn.close()
```

12. Update Data

To modify existing records, we use the **UPDATE** statement

```python
cursor = conn.cursor()
cursor.execute("UPDATE students SET age = 23 WHERE name = 'Alice'")
conn.commit()


print(cursor.rowcount, "record(s) updated.")


cursor.close()
conn.close()
```

12. Update Data

```python
# UPDATE TABLE
try:
    # BEFORE UPDATE
    cursor.execute(f"SELECT * FROM {table} WHERE id = 5")
    result = cursor.fetchall()
    for row in result:
        print(row)
    print(f"---------" * 10)
    # DURING UPDATED
    cursor.execute(f"UPDATE {table} SET age = 33 WHERE id = 5")
    conn.commit()# save changes
    print(f"{cursor.rowcount}, Record with ID=5 Updated Successfully!")
    # AFTER UPDATE
    print(f"---------" * 10)
    # Display Updated data
    cursor.execute(f"SELECT * FROM {table} WHERE id = 5")
    result = cursor.fetchall()
    for row in result:
        print(row)
except:
    print(f"Failed to Update a Record from {table} Table")
```

✓ 0.0s

```
(5, 'KALISA Emmy', 35, 'Civil Engineeriing')
------------------------------------------------------------
1, Record with ID=5 Updated Successfully!
------------------------------------------------------------
(5, 'KALISA Emmy', 33, 'Civil Engineeriing')
```

13. Limit Results

To restrict the number of rows returned, we use **LIMIT**.

```python
cursor = conn.cursor()
cursor.execute("SELECT * FROM students LIMIT 2")
result = cursor.fetchall()

for row in result:
    print(row)

cursor.close()
conn.close()
```

14. Join Tables

To combine data from multiple tables, we use the **JOIN** clause.

```python
cursor = conn.cursor()
cursor.execute("""
SELECT students.name, courses.course_name
FROM students
JOIN courses ON students.id = courses.student_id
""")
result = cursor.fetchall()


for row in result:
    print(row)


cursor.close()
conn.close()
```

14. Join Tables

```
------before JOIN----------before JOIN----------before JOIN----


------STUDENTS DATA----------STUDENTS DATA----------STUDENTS DATA----


(1, 'UMUHOZA Ketia', 30, 'IT')
(2, 'UWIMANA MUHIRE', 22, 'CS')
(5, 'KALISA Emmy', 33, 'Civil Engineeriing')
(7, 'Abimana Afssa', 35, 'Electronics')
(8, 'KALISA Emmy', 30, 'Civil Engineeriing')
(9, 'KALISA Emmy', 30, 'Civil Engineeriing')
(10, 'KALISA Emmy', 30, 'Civil Engineeriing')


------COURSES DATA----------COURSES DATA----------COURSES DATA----


(1, 'Python', 7)
(2, 'Advanced Web', 1)


------AFTER JOIN----------AFTER JOIN----------AFTER JOIN----


('Abimana Afssa', 35, 'Electronics', 'Python')
('UMUHOZA Ketia', 30, 'IT', 'Advanced Web')
```

Conclusion

These Python MySQL commands allow us to interact with a database efficiently. By mastering these operations, students can develop database-driven applications in Python.

4.3.2 MongoDB

MongoDB is a NoSQL **"Not Only SQL"** database that stores data in **JSON-like documents** with **schema flexibility**, allowing for dynamic and scalable data management. Unlike traditional relational databases, MongoDB uses **collections** instead of tables and **documents** instead of rows.

1. Creating a Database

MongoDB is a NoSQL **"Not Only SQL"** database that stores data in **JSON-like documents** with **schema flexibility**, allowing for dynamic and scalable data management. Unlike traditional relational databases, MongoDB uses **collections** instead of tables and **documents** instead of rows.

Python provides the **pymongo** library to interact with MongoDB.

Installing pymongo

To work with MongoDB in Python, install the **pymongo** package:

```
pip install pymongo
```

Connecting to MongoDB

Before performing operations, we need to establish a connection to MongoDB.

```python
import pymongo

# Connect to the MongoDB server
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create or access a database
db = client["university"]
```

```python
import pymongo

# Connect to MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Access the database
db = client["university"]

# Create a collection and insert a sample document
level = db["class"]
level.insert_one({"name": "Level7", "department": "IT"})

# Now check the database and collection names
print(client.list_database_names())  # "university" should now appear
print(db.list_collection_names())  # "class" should now appear
```

✓ 0.0s

['admin', 'config', 'kigali', 'local', 'university']
['students', 'class']

1. Create a Database

MongoDB automatically creates a database when a collection (table) is added.

```python
# Creating a database (it will only be created when we add a collection)
db = client["university"]

print("Database created successfully!")
```

2. Create a Collection (Table)

In MongoDB, a collection is equivalent to a table in relational databases.

```python
# Creating a collection (table)
students = db["students"]

print("Collection created successfully!")
```

3. Insert Data into Collection

MongoDB stores data in JSON-like documents.

```python
# Insert a single document (record)
student_data = {
    "name": "John Doe",
    "age": 21,
    "department": "Computer Science"
}
students.insert_one(student_data)

# Insert multiple documents
students.insert_many([
    {"name": "Alice", "age": 22, "department": "AI"},
    {"name": "Bob", "age": 20, "department": "Data Science"}
])
print("Data inserted successfully!")
```

Now we can access all our db and collection from mongoDb app

RP_HUYE

- ▶ admin
- ▶ config
- ▼ country
  - 📁 **students**
- ▶ kigali
- ▶ local
- ▶ university

⊕ ADD DATA ▾ **⬀ EXPORT DATA ▾** **✎ UPDATE** **🗑 DELETE**

```
_id: ObjectId('679b92c516d428a97f5f47c1')
name : "John Doe"
age : 21
department : "Computer Science"
```

```
_id: ObjectId('679b92c516d428a97f5f47c2')
name : "Alice"
age : 22
department : "AI"
```

```
_id: ObjectId('679b92c516d428a97f5f47c3')
name : "Bob"
age : 20
department : "Data Science"
```

4. Select (Retrieve) Data

Retrieve documents from a collection.

```python
# Retrieve all documents
for student in students.find():
    print(student)


# Retrieve specific fields
for student in students.find({}, {"_id": 0, "name": 1, "department": 1}):
    print(student)
```

```
{'_id': ObjectId('679b92c516d428a97f5f47c1'), 'name': 'John Doe', 'age': 21, 'department': 'Computer Science'}
{'_id': ObjectId('679b92c516d428a97f5f47c2'), 'name': 'Alice', 'age': 22, 'department': 'AI'}
{'_id': ObjectId('679b92c516d428a97f5f47c3'), 'name': 'Bob', 'age': 20, 'department': 'Data Science'}
{'_id': ObjectId('679b948d16d428a97f5f47cf'), 'name': 'John Doe', 'age': 21, 'department': 'Computer Science'}
{'_id': ObjectId('679b948d16d428a97f5f47d0'), 'name': 'Alice', 'age': 22, 'department': 'AI'}
{'_id': ObjectId('679b948d16d428a97f5f47d1'), 'name': 'Bob', 'age': 20, 'department': 'Data Science'}
{'name': 'John Doe', 'department': 'Computer Science'}
{'name': 'Alice', 'department': 'AI'}
{'name': 'Bob', 'department': 'Data Science'}
{'name': 'John Doe', 'department': 'Computer Science'}
{'name': 'Alice', 'department': 'AI'}
{'name': 'Bob', 'department': 'Data Science'}
```

5. Delete Data

Delete specific records using conditions.

```python
# Delete a specific document
students.delete_one({"name": "Alice"})


# Delete multiple documents
students.delete_many({"department": "Data Science"})


# Delete all documents
students.delete_many({})
```

6. Using Where Condition

Query documents using conditions.

```python
# Find students older than 20
for student in students.find({"age": {"$gt": 20}}):
    print(student)
```

```python
# Find students whose age is less than 20
for student in students.find({"age": {"$lt": 22}}):
    print(student)
```

```python
# Find students whose age is equal to 22
for student in students.find({"age": {"$eq": 22}}):
    print(student)
```

✓ 0.0s

```python
# Find students whose age is less than or equal to 20
for student in students.find({"age": {"$lte": 20}}):
    print(student)
```

```python
# Find students whose age is greater than or equal to 20
for student in students.find({"age": {"$gte": 20}}):
    print(student)
```

7. Order By

Sort the results in ascending or descending order.

```python
# Sort by age in ascending order
for student in students.find().sort("age", 1):
    print(student)


# Sort by age in descending order
for student in students.find().sort("age", -1):
    print(student)
```

8. Drop a Collection (Table)

Remove a collection from the database. If you have only one collection then once you drop it, it will automatically drop its database.

```python
# Drop a collection
students.drop()
print("Collection dropped successfully!")
```

9. Update Data

Modify existing records.

```
    # Update a single document
    students.update_one({"name": "John Doe"}, {"$set": {"age": 23}})

  ✓  0.0s

UpdateResult({'n': 1, 'nModified': 1, 'ok': 1.0, 'updatedExisting': True}, acknowledged=True)


    # Update multiple documents
    students.update_many({"department": "Computer Science"}, {"$set": {"department": "CS"}})
  ✓  0.0s

UpdateResult({'n': 1, 'nModified': 1, 'ok': 1.0, 'updatedExisting': True}, acknowledged=True)
```

10. Limit Results

Restrict the number of documents returned.

```python
# LIMITS
# Retrieve only 2 documents
for student in students.find().limit(2):
    print(student)
```

✓ 0.0s

```
{'_id': ObjectId('679b9add4fd493a63bec0caf'), 'name': 'John Doe', 'age': 23, 'department': 'CS'}
{'_id': ObjectId('679b9ade4fd493a63bec0cb0'), 'name': 'Alice', 'age': 22, 'department': 'AI'}
```

11. Join in MongoDB

MongoDB does not support SQL-style joins natively, but it provides **$lookup** for joining collections.

```python
# JOIN
# Creating another collection for demonstration
courses = db["courses"]
courses.insert_many([
    {"student_name": "John Doe", "course": "Database Systems"},
    {"student_name": "Alice", "course": "Machine Learning"}
])

# Aggregation with $lookup (Joining students with courses)
result = students.aggregate([
    {
        "$lookup": {
            "from": "courses",
            "localField": "name",
            "foreignField": "student_name",
            "as": "enrolled_courses"
        }
    }
])

for doc in result:
    print(doc)
```

✓ 0.0s   Python

```
{'_id': ObjectId('679b9add4fd493a63bec0caf'), 'name': 'John Doe', 'age': 23, 'department': 'CS', 'enrolled_courses': []}
{'_id': ObjectId('679b9ade4fd493a63bec0cb0'), 'name': 'Alice', 'age': 22, 'department': 'AI', 'enrolled_courses': []}
{'_id': ObjectId('679b9ade4fd493a63bec0cb1'), 'name': 'Bob', 'age': 20, 'department': 'Data Science', 'enrolled_courses': []}
```

Summary

Operation	MongoDB Equivalent
Create Database	`client["db_name"]`
Create Table (Collection)	`db["collection_name"]`
Insert	`insert_one()`, `insert_many()`
Select	`find()`
Delete	`delete_one()`, `delete_many()`
Where Condition	`find({"field": "value"})`
Order By	`sort("field", 1 or -1)`
Drop Table	`collection.drop()`
Update	`update_one()`, `update_many()`
Limit	`find().limit(n)`
Join	`$lookup`

Formative Assessment 2. Next class