Lab Exam Practice

1. Your task is to create a simple calculator(+,-,*,/). Choose two programming languages (Python, Java, JavaScript), assess their suitability, and pick one. Implement the solution in your selected language and provide explanatory comments on why you opted for that specific language based on any two criteria.

```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def prod(a,b):
    return a * b
f=add(2,3)
print(f)
```

2 . WAP that uses the function described below to let the users enter country name and its capital and see a neatly formatted sentence and if users input ‘q’ at any time to quit. Function should combine the country name and capital name with a space in between to complete a full sentence, and then capitalize the first letters and returns the full sentence.(Muscat Is Capital Of Oman)

```
def get_sentence(x, y):
   s = y   +  "  is capital of "    + x
   return s.title()
print("Enter 'q' at any time to quit.")
while True:
   cname= input("\nPlease give country name: ")
   if cname == 'q':
      break
   capname = input("Please give  its capital: ")
   if capname == 'q':
      break
   formatted_s = get_sentence(cname, capname)
   print("\tNeatly formatted name: " + formatted_s+ '.' )
```

3. Write a python program to create a dictionary containing five languages and the country having one of these as an official language. One key-value pair might be 'Oman': 'Arabic'.
Use a loop to print a sentence about each language, such as Arabic is the official language of Oman.

```
languages = {
'Arabic': 'Oman',
'Hindi': 'India',
'Urdu': 'Pakistan',
}
for language, country in languages.items():
   print("The " + language.title() + " is official language " + country.title() + ".")
```

4.Implement program that should:
Generate or simulate a set of sales data for a period (e.g., 30 days).
Calculate and display the average sales over this period.
Determine and display the day with the maximum sales.
Identify and display the most common sales amount during this period.

```
import random
from collections import Counter
def generate_sales_data():
   return [(f"2024-03-{day}", random.randint(500, 10000), random.randint(10, 200)) for day in range(1, 31)]

def main():
   sales_data = generate_sales_data()
   avg_sales = sum(day[1] for day in sales_data) / len(sales_data)
   max_sales = max(sales_data, key=lambda x: x[1])
   most_common_sales = Counter(day[1] for day in sales_data).most_common(1)[0][0]

   print(f"Average Daily Sales: OMR {avg_sales:.2f}")
   print(f"Day with Highest Sales: {max_sales[0]} with OMR {max_sales[1]}")
   print(f"Most Common Sales Amount: OMR {most_common_sales}")

if __name__ == "__main__":
   main()
```

5. Choose two programming languages (Python, Java, JavaScript), assess their suitability, and pick one. Implement the solution in your selected language and provide explanatory comments on why you opted for that specific language based on any two criteria.
Implement program that should:
Generate or simulate a set of weather data for a period (e.g., 30 days).
Calculate and display the average temp over this period.
Determine and display the day with the maximum temp.
Identify and display the most common temp during this period.

```
import random
from collections import Counter

def generate_weather_data():
    weather_conditions = ['Sunny', 'Rainy', 'Cloudy', 'Windy', 'Snowy']
    return [(f"2024-03-{day}", random.randint(-10, 35), round(random.uniform(0, 20)),
random.choice(weather_conditions)) for day in range(1, 31)]

def main():
    weather_data = generate_weather_data()
    avg_temp = sum(day[1] for day in weather_data) / len(weather_data)
    max_temp = max(weather_data, key=lambda x: x[1])
    most_common_temp = Counter(day[1] for day in weather_data).most_common(1)[0][0]

    print(f"Average Temperature: {avg_temp:.2f}°C")
    print(f"Day with Max temp: {max_temp[0]} with {max_temp[1]} °C")
    print(f"Most Common temperature: {most_common_temp } °C")

if __name__ == "__main__":
    main()
```

6.Design a Python class StudInfo with the following specifications:

Methods:
a. __init__(self): Initializes an empty dictionary to store student information.

b. add_student(self, name, ID): Adds a student to the information system with the provided name and ID. Display a confirmation message.

c. get_name(self, ID): Retrieves the name associated with the given ID. If the student exists, return the name; otherwise, return "Student not found."

Additional Requirement:

Implement the main() function to create an instance of StudInfo and provide a user-friendly menu to interact with the student information system. The menu should include options to add a student, get a student's name by ID, and exit the program.

```
class StudInfo:
    def __init__(self):
        self.students = {}

    def add_student(self, name, ID):
        self.students[ID] = name
        print(f"Student {name} added with ID {ID}.")

    def get_name(self, ID):
        if ID in self.students:
            return f"Name: {self.students[ID]}"
        else:
            return "ID not found."

def main():
    info = StudInfo()

    while True:
        print("\n1. Add Student\n2. Get Student Name by ID\n3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            name = input("Enter student name: ")
            ID = input("Enter student ID: ")
            info.add_student(name, ID)
        elif choice == '2':
            ID = input("Enter student ID: ")
            print(info.get_name(ID))
        elif choice == '3':
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

7. Design a Python class EmpRecord with the following specifications:

Methods:
a. __init__(self): Initializes an empty dictionary to store employee records.

b. add_employee(self, name, emp_id): Adds an employee to the record system with the provided name and employee ID. Display a confirmation message.

c. get_employee_name(self, emp_id): Retrieves the name associated with the given employee ID. If the employee exists, return the name; otherwise, return "Employee not found."

Additional Requirement:

Implement the main() function to create an instance of EmpRecord and provide a user-friendly menu to interact with the employee record system. The menu should include options to add an employee, get an employee's name by ID, and exit the program.

```
class EmpRecord:
    def __init__(self):
        self.employees = {}

    def add_employee(self, name, emp_id):
        self.employees[emp_id] = name
        print(f"Employee {name} added with ID {emp_id}.")

    def get_employee_name(self, emp_id):
        if emp_id in self.employees:
            return f"Name: {self.employees[emp_id]}"
        else:
            return "Employee not found."

def main():
    record = EmpRecord()

    while True:
        print("\n1. Add Employee\n2. Get Employee Name by ID\n3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            name = input("Enter employee name: ")
            emp_id = input("Enter employee ID: ")
            record.add_employee(name, emp_id)
        elif choice == '2':
```

```
            emp_id = input("Enter employee ID: ")
            print(record.get_employee_name(emp_id))
        elif choice == '3':
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

8. Design a Python class BookCatalog with the following specifications:

Methods:
a. __init__(self): Initializes an empty dictionary to store book details.

b. add_book(self, title, isbn): Adds a book to the catalog with the provided title and ISBN. Display a confirmation message.

c. get_book_title(self, isbn): Retrieves the title associated with the given ISBN. If the book exists in the catalog, return the title; otherwise, return "Book not found."

Additional Requirement:

Implement the main() function to create an instance of BookCatalog and provide a user-friendly menu to interact with the book catalog system. The menu should include options to add a book, get a book's title by ISBN, and exit the program.

```
class BookCatalog:
    def __init__(self):
        self.books = {}

    def add_book(self, title, isbn):
        self.books[isbn] = title
        print(f"Book '{title}' added with ISBN {isbn}.")

    def get_book_title(self, isbn):
        if isbn in self.books:
            return f"Title: {self.books[isbn]}"
        else:
            return "Book not found."

def main():
    catalog = BookCatalog()

    while True:
```

```
        print("\n1. Add Book\n2. Get Book Title by ISBN\n3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            title = input("Enter book title: ")
            isbn = input("Enter book ISBN: ")
            catalog.add_book(title, isbn)
        elif choice == '2':
            isbn = input("Enter book ISBN: ")
            print(catalog.get_book_title(isbn))
        elif choice == '3':
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```