Lab 3

```
1. (a) Develop a module called module_ListFunction that includes the following functions:
i. A function to find the maximum value in a given list.
ii. A function to find the minimum value in a given list.
iii. A function to calculate the sum of all elements in a list.
iv. A function to compute the average of the list.
v. A function to determine the median of a list.
```

In [ ]:

```python
from module_ListFunction import Max, Min, Sum, Avg, Median

```

In [17]:

```python

l = [4,2,5,1,9,1,5]

max_val = Max(l)


print("Maximum element in list:", max_val)

```

```
Maximum element in list: 9
```

In [18]:

```python

l = [6,3,6,2,1,9,0]



min_val = Min(l)

print("Minimum element in list:", min_val)


```

```
Minimum element in list: 0
```

In [19]:

```python

l = [8,3,8,3,5,2,6]



sum_val = Sum(l)

print("Sum:", sum_val)


```

```
Sum: 35
```

In [20]:

```python

l = [1,5,3,7,4,6]



avg_val = Avg(l)

print("Average:", avg_val)

```

```
Average: 4.333333333333333
```

In [21]:

```python

l = [2,5,1,7,4,6]



median_val = Median(l)



print("Median:", median_val)

```

```
Median: 4.5
```

```
2. Write a Python program to create a module that performs various set operations.
a. Write a function to add an element to a set, ensuring no errors if the element is already
present.
b. Write a function to remove an element from a set, ensuring no errors if the element is
not present.
c. Write a function to return the union and intersection of two sets, handling empty sets
correctly.
d. Write a function to return the difference S1-S2, handling empty sets correctly.
e. Write a function to check if set S1 is a subset of set S2, handling empty sets correctly.
f. Write a function to find the length of a set without using the len() function.
g. Write a function to compute the symmetric difference of two sets.
h. Write a function to compute the power set of a given set.
i. Write a function to get all unique subsets of a given set.
```

In [1]:

```python
from Set import*
set1 = {1, 2, 3}
set2 = {3, 4, 5}

print("Initial Sets:")
print("Set 1:", set1)
print("Set 2:", set2)

print("\nAdd Operation:")
ADD(4, set1)
print("Set 1 after adding 4:", set1)
ADD(2, set1)  # Should print "Element already exists"

print("\nRemove Operation:")
REMOVE(4, set1)
print("Set 1 after removing 4:", set1)
REMOVE(5, set1)  # Should print "Element doesn't exist"

print("\nUnion:")
UNION(set1, set2)

print("\nIntersection:")
INTERSECTION(set1, set2)

print("\nDifference:")
DIFFERENCE(set1, set2)

print("\nSubset:")
SUBSET({1, 2}, set1)
SUBSET(set1, set2)  # Should print "Is subset: False"

print("\nLength:")
LEN(set1)

print("\nSymmetric Difference:")
SYMMETRIC_DIFFERENCE(set1, set2)

print("\nPower Set:")
Power_Set(set1)

print("\nUnique Subsets:")
unique_subsets(set1)
```

```
Initial Sets:
Set 1: {1, 2, 3}
Set 2: {3, 4, 5}

Add Operation:
Set 1 after adding 4: {1, 2, 3, 4}
Element already exist

Remove Operation:
Set 1 after removing 4: {1, 2, 3}
Element doesn't exist

Union:
{1, 2, 3, 4, 5}

Intersection:
{3}

Difference:
{1, 2}

Subset:
True
False

Length:
3

Symmetric Difference:
{1, 2, 4, 5}

Power Set:
[(), (1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2, 3)]

Unique Subsets:
[(), (1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2, 3)]
```

```
3. Write a program to create functions that can accept multiple dictionaries as arguments using
‘*args’, and perform various operations on them.
(a) Write a function, say, ‘merging_Dict(*args)’ that takes multiple dictionaries and merge
them.
(b) Write a function which can find common keys in multiple dictionaries.

(c) Create a function to invert a dictionary, swapping keys and values. If multiple keys have
the same value, group these keys in a list in the inverted dictionary. Implement and demonstrate
this with examples.
(d)Write a function to find common key-value pairs across multiple dictionaries.
```

In [2]:

```python
def merging_Dict(*args):

    merged_dict = {}
    for d in args:
        if isinstance(d, dict):
            merged_dict.update(d)
        else:
            raise TypeError("All arguments must be dictionaries")
    return merged_dict

def common_keys(*args):

    if not args:
        return set()

    common_keys_set = set(args[0].keys())
    for d in args[1:]:
        if isinstance(d, dict):
            common_keys_set.intersection_update(d.keys())
        else:
            raise TypeError("All arguments must be dictionaries")

    return common_keys_set

def invert_dict(d):

    inverted_dict = {}
    for key, value in d.items():
        if value in inverted_dict:
            inverted_dict[value].append(key)
        else:
            inverted_dict[value] = [key]
    return inverted_dict

def common_key_value_pairs(*args):

    if not args:
        return set()

    common_pairs = set(args[0].items())
    for d in args[1:]:
        if isinstance(d, dict):
            common_pairs.intersection_update(d.items())
        else:
            raise TypeError("All arguments must be dictionaries")

    return common_pairs

# Example usage
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'a': 1, 'b': 2, 'd': 4}
dict3 = {'a': 1, 'b': 2, 'e': 5}

print("Merged Dictionary:", merging_Dict(dict1, dict2, dict3))
print("Common Keys:", common_keys(dict1, dict2, dict3))
print("Inverted Dictionary:", invert_dict({'a': 1, 'b': 2, 'c': 1, 'd': 3}))
print("Common Key-Value Pairs:", common_key_value_pairs(dict1, dict2, dict3))
```

```
Merged Dictionary: {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
Common Keys: {'a', 'b'}
Inverted Dictionary: {1: ['a', 'c'], 2: ['b'], 3: ['d']}
Common Key-Value Pairs: {('b', 2), ('a', 1)}
```

```
4. Create a Python program to efficiently manage and handle a library's collection of books.
Each book in the library is represented with the following attributes: title, author, publisher,
volume, year of publication, and ISBN (International Standard Book Number).
Design and implement a module named LibraryManager.py that includes functions to manage
books in the library. Collect data for 25 recently published books on topics such as Operating
Systems, Data Structures, and Machine Learning using Python, published between 2020 and
2024. Store this information in a dictionary where the key is the ISBN, and the value is another
dictionary containing the book details.
Within the LibraryManager.py module, create functions to:
• Add a book to the library.
• Remove a book from the library by its ISBN.
• Retrieve and display the details of a book using its ISBN.
• Search for books by title or author.
• List all books currently in the library.
• Update the details of an existing book.
• Check if a book is available in the library by its ISBN.
Demonstrate the functionality of your module by adding a few sample books, removing a book,
retrieving the details of a book, searching for books, listing all books, updating book details,
and checking the availability of a book.
```

In [4]:

```
from LibraryManager import*
if __name__ == "__main__":

    add_book('978-0134670958', 'Operating Systems: Three Easy Pieces', 'Remzi
    add_book('978-0262033848', 'Introduction to Algorithms', 'Thomas H. Corme
    add_book('978-0134692881', 'Machine Learning Yearning', 'Andrew Ng', 'Sel


    remove_book('978-0134692881')


    print("\nRetrieve Book Details:")
    get_book_details('978-0134670958')


    print("\nSearch Books by Title:")
    search_books('Algorithms')


    print("\nList All Books:")
    list_all_books()


    update_book('978-0262033848', year=2023)


    print("\nCheck Book Availability:")
    print("Book available:", is_book_available('978-0262033848'))
```

```
Book with this ISBN already exists.
Book with this ISBN already exists.
Book 'Machine Learning Yearning' added successfully.
Book with ISBN 978-0134692881 removed successfully.

Retrieve Book Details:
Title: Operating Systems: Three Easy Pieces
Author: Remzi H. Arpaci-Dusseau
Publisher: Arpaci-Dusseau Books
Volume: 1
Year: 2020
Isbn: 978-0134670958

Search Books by Title:

ISBN: 978-0262033848
Title: Introduction to Algorithms
Author: Thomas H. Cormen
Publisher: The MIT Press
Volume: 3
Year: 2023
Isbn: 978-0262033848

List All Books:

ISBN: 978-0134670958
Title: Operating Systems: Three Easy Pieces
Author: Remzi H. Arpaci-Dusseau
Publisher: Arpaci-Dusseau Books
Volume: 1
Year: 2020
Isbn: 978-0134670958

ISBN: 978-0262033848
Title: Introduction to Algorithms
Author: Thomas H. Cormen
Publisher: The MIT Press
Volume: 3
Year: 2023
Isbn: 978-0262033848
Book with ISBN 978-0262033848 updated successfully.

Check Book Availability:
Book available: True
```

In [ ]:

```
5.Write a Python program to analyze and process weather data for New York Cit
August to 10th July in 2024.
1. Each day’s data includes:
o Date
o Maximum temperature (in Celsius)
o Minimum temperature (in Celsius)
o Humidity (in percentage)
(Hint: Store the data in a list of dictionaries.)
2. Write a function to find the highest and lowest temperatures recorded duri
3. Write a function to determine the number of days with temperatures above 3
4. Write a function to compute the average humidity over the specified period
```

In [7]:

```python
from datetime import datetime, timedelta
import random


def generate_weather_data(date):
    return {
        'date': date.strftime('%Y-%m-%d'),
        'max_temp': random.randint(20, 35),
        'min_temp': random.randint(10, 25),
        'humidity': random.randint(40, 90)
    }


start_date = datetime(2024, 8, 1)
end_date = datetime(2025, 7, 10)

weather_data = {}
current_date = start_date

while current_date <= end_date:
    formatted_date = current_date.strftime('%Y-%m-%d')
    weather_data[formatted_date] = generate_weather_data(current_date)
    current_date += timedelta(days=1)


def find_highest_and_lowest_temperatures(data):
    max_temp = float('-inf')
    min_temp = float('inf')

    for entry in data.values():
        if entry['max_temp'] > max_temp:
            max_temp = entry['max_temp']
        if entry['min_temp'] < min_temp:
            min_temp = entry['min_temp']

    return max_temp, min_temp

def count_days_above_30c(data):
    count = 0

    for entry in data.values():
        if entry['max_temp'] > 30:
            count += 1

    return count

def compute_average_humidity(data):
    if not data:
        return "No data available to compute average humidity."

    total_humidity = 0

    for entry in data.values():
        total_humidity += entry['humidity']

    average_humidity = total_humidity / len(data)
    return average_humidity
```

```python

p
highest_temp, lowest_temp = find_highest_and_lowest_temperatures(weather_data
print(f"Highest Temperature: {highest_temp}°C")
print(f"Lowest Temperature: {lowest_temp}°C")

days_above_30 = count_days_above_30c(weather_data)
print(f"Number of Days with Temperatures Above 30°C: {days_above_30}")

average_humidity = compute_average_humidity(weather_data)
if isinstance(average_humidity, str):
    print(average_humidity)
else:
    print(f"Average Humidity: {average_humidity:.2f}%")

```

```
Highest Temperature: 35°C
Lowest Temperature: 10°C
Number of Days with Temperatures Above 30°C: 110
Average Humidity: 63.80%
```

In [ ]: