WEEK 1:

1. Write a program to demonstrate different number data types in Python.

```
# Different number data types

# Integer

a = 20

print(type(a))


# Float

b = 21.23

print(type(b))


# Boolean

c = True

d = False

print(type(c))

print(type(d))


# String

e = "Hello World"

print(type(e))


# Complex

f = 3 + 4j

print(type(f))


# List
```

```
list1 = ["apple","banana", "watermelon"]

print(type(list1))


# Tuple

tup = ("apple","banana", "watermelon")

print(type(tup))


# Dictionary

dict1 = {"name":"SK", "age":19}

print(type(dict1))
```

OUTPUT:

```
<class 'int'>

<class 'float'>

<class 'bool'>

<class 'bool'>

<class 'str'>

<class 'complex'>

<class 'list'>

<class 'tuple'>

<class 'dict'>
```

2. Write a python program to Read input data from the Keyboard and perform different Arithmetic Operations on the numbers.

```
# To Perform different operations on the numbers taken as input from the user

num1 = float(input("Enter a number:\n"))

num2 = float(input("Enter another number:\n"))
```

```
# Addition

addition =  num1 + num2

print(f"The sum of the two numbers is {addition}")


# Subtraction

subtraction = num1 - num2

print(f"The difference between the two numbers is {subtraction}")


# Multiplication

multiply = num1 * num2

print(f"The product of the two numbers is {multiply}")


# Division

divide = num1 / num2

print(f"The division of the two numbers leads to the answer {divide} ")


# Floor division

f_divide = num1 // num2

print(f"The floor division of the two numbers leads to the answer {f_divide}")


# Exponential to the power of 2

expo = float(input("Enter one number which must be used as the power of prev. two numbers: \n"))

expo1 = num1 ** expo

expo2 = num2 ** expo

print(f"The result of first entered number is {expo1} and that of the second number is {expo2}")


# Modulo operator
```

```
rem = float(input("Enter a number to get the remainder of prev. two numbers when divided with this number: \n"))

modulo1 = num1 % rem

modulo2 = num2 % rem

print(f"The remainder of the first number when divided with {rem} is {modulo1} and that of the second number is {modulo2}")
```

OUPUT:

```
Enter a number:

5

Enter another number:

10

The sum of the two numbers is 15.0

The difference between the two numbers is -5.0

The product of the two numbers is 50.0

The division of the two numbers leads to the answer 0.5

The floor division of the two numbers leads to the answer 0.0

Enter one number which must be used as the power of prev. two numbers:

2

The result of first entered number is 25.0 and that of the second number is 100.0

Enter a number to get the remainder of prev. two numbers when divided with this number:

3

The remainder of the first number when divided with 3.0 is 2.0 and that of the second number is 1.0
```

3. Write a python program to declare variables and display types of respective variables.

```
# To declare variables and display types of those variables


var1 = 10

print(f"The type of variable var 1 is {type(var1)} ")
```

```
var2 = 10.3568

print(f"The type of variable var 2 is {type(var2)} ")


var3 = "Namaste"

print(f"The type of variable var 3 is {type(var3)} ")


var4 = True

print(f"The type of variable var 4 is {type(var4)} ")


var5 = 2 +  5j

print(f"The type of variable var 5 is {type(var5)} ")


var6 = ["hello","world"]

print(f"The type of variable var 6 is {type(var6)} ")


var7 = ("hello", "world")

print(f"The type of variable var 7 is {type(var7)} ")


var8 = {"name":"raju", "age":50}

print(f"The type of variable var 8 is {type(var8)} ")
```

SAMPLE OUTPUT:

```
The type of variable var 1 is <class 'int'>

The type of variable var 2 is <class 'float'>

The type of variable var 3 is <class 'str'>

The type of variable var 4 is <class 'bool'>
```

The type of variable var 5 is <class 'complex'>

The type of variable var 6 is <class 'list'>

The type of variable var 7 is <class 'tuple'>

The type of variable var 8 is <class 'dict'>

4. Write a python program to convert integer type to float and vice versa.

```
# Converting float to integer and vice versa


float1 = float(input("Enter a floating type of number:\n"))

int1 = int(input("Enter an integer type of number:\n"))


conv_float = float(int1)

conv_int = int(float1)


print(f"The floating number when converted to integer becomes {conv_int}")

print(f"The integer number when converted to float becomes {conv_float}")
```

OUTPUT:

Enter a floating type of number:

32.1

Enter an integer type of number:

45

The floating number when converted to integer becomes 32

The integer number when converted to float becomes 45.0

5. Write a python program to print the current date in the following format "Sun May 29 02:26:23 IST 2017".

```
# Printing the date

from datetime import datetime
```

```
import pytz


utc = datetime.now(pytz.utc)




ist = utc.astimezone(pytz.timezone('Asia/Kolkata'))


formatted_date = ist.strftime('%a %b %d %H:%M:%S IST %Y')


print(formatted_date)
```

OUTPUT:

```
Thu Sep 05 17:11:22 IST 2024
```

6. Write a program to read the following Employee data from the keyboard and print that data.
Employee Details:
a. Enter Employee No [Data type: int]
b. Enter Employee Name [Data type: string]
c. Enter Employee Salary [Data type: int]
d. Enter Employee Address [Data type: string]
e. Employee Married ?[True|False]: [Data type: Boolean]

```
# Employee Data

emp_no = int(input("Enter Employee No: "))

emp_name = input("Enter Employee Name: ")

emp_sal = float(input("Enter Employee Salary: "))

emp_add = input("Enter Employee Address: ")

emp_mar = eval(input("Employee Married? [True/False]: "))




print(f"Employee No: {emp_no}")
```

```
print(f"Employee Name: {emp_name}")

print(f"Employee Salary: {emp_sal}")

print(f"Employee Address: {emp_add}")

print(f"Employee Married: {emp_mar}")
```

OUTPUT:

```
Enter Employee No: 101

Enter Employee Name: MAYANK

Enter Employee Salary: 900000

Enter Employee Address: UDUPI

Employee Married? [True/False]: True

Employee No: 101

Employee Name: MAYANK

Employee Salary: 900000.0

Employee Address: UDUPI

Employee Married: True
```

WEEK2:

1. Write a Python program to find the longest increasing subsequence from a given list of numbers.

```
def longest_increasing_subsequence(arr):
    n = len(arr)
    lis = [1] * n  # Initialize LIS values for all indexes as 1

    # Compute optimized LIS values in a bottom-up manner
    for i in range(1, n):
        for j in range(0, i):
```

```
            if arr[i] > arr[j] and lis[i] < lis[j] + 1:
                lis[i] = lis[j] + 1

    # Find the maximum value in lis[]
    maximum = max(lis)

    # Reconstruct the longest increasing subsequence
    lis_sequence = []
    current_length = maximum
    for i in range(n - 1, -1, -1):
        if lis[i] == current_length:
            lis_sequence.append(arr[i])
            current_length -= 1

    lis_sequence.reverse()  # The sequence is constructed in reverse order

    return lis_sequence

# Example usage
arr = [10, 22, 9, 33, 21, 50, 41, 60]
print("Longest Increasing Subsequence is:", longest_increasing_subsequence(arr))
```

OUTPUT:

Longest Increasing Subsequence is: [10, 22, 33, 41, 60]

2. Create a Python script to generate a list that contains 25 elements and display the frequency of each item in a list

To generate a list and find the frequency of each item in the list

```
def CountFrequency(my_list):

    freq = {}
    for item in my_list:
        if (item in freq):
            freq[item] +=1
        else:
            freq[item] = 1

    for key,value in freq.items():
        print("%d : %d " % (key,value))


n = int(input("Enter the no of items in the list: \n"))
lis = []
i = 0
print("Enter the items of the list:")
while i < n:
    lis_item = lis.append(int(input()))
    i +=1

CountFrequency(lis)
```

OUTPUT:

```
Enter the no of items in the list:
25
```

Enter the items of the list:

1

2

3

4

5

6

7

8

9

1

2

3

4

5

6

7

8

9

1

2

3

4

5

6

7

1 : 3

2 : 3

3 : 3

4 : 3

5 : 3

6 : 3

7 : 3

8 : 2

9 : 2

3. Develop a Python program that constructs a list of 15 strings. It should then determine the count of strings in this list that have a minimum length of two characters and also start and end with identical characters. You can choose any specific list of strings for this task.

```
def count_strings_with_identical_ends(strings):
    count = 0
    for s in strings:
        if len(s) >= 2 and s[0].lower() == s[-1].lower():  # Added .lower() to handle case sensitivity
            count += 1
    return count


strings = [
    'Ava',
    'Ben',
    'Civic',
    'David',
    'Eve',
    'Felicity',
    'Gog',
    'Hannah',
```

```
    'I',
    'Jill',
    'Kayak',
    'Liam',
    'Madam',
    'Nina',
    'Otto'
]

count = count_strings_with_identical_ends(strings)
print("Count of strings with min length of 2 and identical start and end characters:", count)
```

OUTPUT:

Count of strings with min length of 2 and identical start and end characters: 9

4. Develop a Python script that generates a list with 15 items and then eliminates any duplicates from that list.

```
# To create a random list of 15 elements and delete the duplicate elements from that list

import random

def generate_random_list(size, lower_bound, upper_bound):
    return [random.randint(lower_bound, upper_bound) for _ in range(size)]

def remove_duplicates(input_list):
    return list(set(input_list))
```

```
list_size = 15

lower_bound = 1

upper_bound = 100



random_list = generate_random_list(list_size, lower_bound, upper_bound)

print("Original list with possible duplicates:")

print(random_list)



unique_list = remove_duplicates(random_list)

print("\nList after removing duplicates:")

print(unique_list)
```

OUTPUT:

```
Original list with possible duplicates:

[48, 70, 24, 52, 67, 34, 68, 30, 24, 19, 83, 39, 54, 91, 86]


List after removing duplicates:

[34, 67, 68, 70, 39, 48, 19, 52, 83, 54, 86, 24, 91, 30]
```

5. Develop a Python script that builds a list with 15 elements. This script will reposition the items in the list by doing a circular right shift. The number of positions shifted will be based on a user-specified value.

```
# To  reposition the items in the list by doing a circular right shift. The number of positions shifted will be based on a user-specified value.



import random
```

```
def circular_right_shift(lst, positions):
    n = len(lst)
    positions = positions % n
    return lst[-positions:] + lst[:-positions]


def main():


    lis = []
    n =  15
    i = 0
    while i < n :
        lis.append(int(random.randint(0, 100)))
        i += 1
    print("Original list:", lis)



    while True:
        try:
            positions = int(input("Enter the number of positions to shift (integer): "))
            if positions < 0:
                print("Please enter a non-negative integer.")
            else:
                break
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
```

```
    shifted_list = circular_right_shift(lis, positions)



    print("List after shifting:", shifted_list)


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

OUTPUT:

Original list: [42, 6, 5, 91, 11, 49, 64, 65, 22, 60, 2, 99, 4, 3, 89]

Enter the number of positions to shift (integer): 2

List after shifting: [3, 89, 42, 6, 5, 91, 11, 49, 64, 65, 22, 60, 2, 99, 4]

WEEK 3:

1. Write a program that uses a 'while' loop along with a found flag to search through a list of powers of 2. Your program should identify the value corresponding to 2 raised to the fifth power (32).

```
powerof2=[2**i for i in range(10)]

found = False

index=0

target_value=32


while index < len(powerof2):
    if powerof2[index] == target_value:
        found = True
        break
    index += 1
```

```
if found :
    print(f"The value {target_value} was found at index {index}.")
else:
    print(f"The value {target_value} was not found in the list.")
```

OUTPUT:

The value 32 was found at index 5.

2. Write a program to calculate the distance covered by a robot after a series of movements on a plane. A robot starts at the origin point (0,0) in a 2D plane.
It can move in four directions: UP, DOWN, LEFT, and RIGHT.
The robot's movements are defined as follows:
a. UP - 5
b. DOWN - 3
c. LEFT - 3
d. RIGHT - 2
The number following each direction indicates the number of steps taken in that direction.
Implement a program that:
a. Tracks the robot's position after the given sequence of movements.
b. Calculates the distance between the robot's final position and the origin (0,0).
c. If the calculated distance is a floating-point number, round it to the nearest integer.

Constraints:
a. You are not allowed to use any external packages or libraries.
b. Only basic Python functionalities should be used.

```
movement_distances = {
    'UP' : 5,
    'DOWN' : 3,
    'LEFT' : 3,
    'RIGHT' : 2
```

```
}

x,y = 0,0

movements = [

    ('UP',2),
    ('RIGHT',3),
    ('DOWN',1),
    ('LEFT',1)
]

for direction , steps in  movements:
    if direction == 'UP':
        y+= movement_distances['UP']*steps
    elif direction == 'DOWN' :
        y-=movement_distances['DOWN']*steps
    elif direction == 'LEFT' :
        x-=movement_distances['LEFT']*steps
    elif direction == 'RIGHT':
        x += movement_distances["RIGHT"]*steps



distance = (x**2 + y**2) ** 0.5

distance_rounded = round(distance)
```

```
print(f"Final Position: ({x},{y}")

print(f"Distance from origin (rounded): {distance_rounded}")
```

OUTPUT:

Final Position: (3,7

Distance from origin (rounded): 8

3. Write a python program that can accept two strings as input and print the string with maximum length in console. If two strings have the same length, then the program should print all strings in one line.

```
string1 = input("Enter the first string:")

string2 = input("Enter the second string")


if len(string1) > len(string2):
    print(string1)
elif len(string1) < len(string2):
    print(string2)
else:
    print(f"{string1} {string2}")
```

OUTPUT:

Enter the first string:Mayank

Enter the second stringAmishi

Mayank Amishi

4. Write a program that computes the net amount in a bank account based on a series of transactions provided as input. Instructions: a. Your program should accept a transaction log as input, where each transaction is either a deposit or a withdrawal. b. The transaction log format is as follows: c. D 100 indicates a deposit of 100 units. d. W 200 indicates a withdrawal of 200 units. e. The program should calculate the net amount in the bank account after processing all transactions.

```
net_amount = 0

print("Enter transactions in the format 'D 100' for deposits or 'W 200' for withdrawals.")
print("Type 'done' to finish.")

while True:
    transaction = input("Enter transaction: ")

    if transaction.lower() == 'done':
        break

    parts = transaction.split()

    if len(parts) != 2:
        print("Invalid format. Please enter in the format 'D 100' or 'W 200'.")
        continue

    action, amount_str = parts

    try:
        amount = float(amount_str)
        if action == 'D':
            net_amount += amount
        elif action == 'W':
            net_amount -= amount
        else:
```

```
            print("Invalid action. Use 'D' for deposit and 'W' for withdrawal.")
    except ValueError:
        print("Invalid amount. Please enter a valid number.")

print(f"Net amount in the account: ${net_amount:.2f}")
```

OUTPUT:

Enter transaction: D 1000

Enter transaction: W 200

Enter transaction: Done

Net amount in the account: $1600.00

5. Write a python program to implement the binary search which searches an item in a sorted list. (Do not use any Packages)

```
def binary_search(sorted_list, target):
    left = 0
    right = len(sorted_list) - 1

    while left <= right:
        mid = (left + right) // 2
        mid_value = sorted_list[mid]

        if mid_value == target:
            return mid
        elif mid_value < target:
            left = mid + 1
        else:
            right = mid - 1
```

```
    return -1

if __name__ == "__main__":
    sorted_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
    target = int(input("Enter the value to search for: "))

    result = binary_search(sorted_list, target)

    if result != -1:
        print(f"Value {target} found at index {result}.")
    else:
        print(f"Value {target} not found in the list.")
```

OUTPUT:

Enter the value to search for: 5

Value 5 found at index 2.

6. Write a Python 3 program that demonstrates how to calculate the summation of all possible combinations of elements in a list of tuples. Use nested for loops to iterate through the tuples and generate the combinations.

```
def calculate_combinations_sum(tuples_list):
    if not tuples_list:
        return 0

    total_sum = 0

    num_combinations = 1
```

```
    for t in tuples_list:
        num_combinations *= len(t)

    def generate_combinations(tuples_list, current_combination, index):
        nonlocal total_sum

        if index == len(tuples_list):
            total_sum += sum(current_combination)
            return

        for element in tuples_list[index]:
            generate_combinations(tuples_list, current_combination + [element], index + 1)

    generate_combinations(tuples_list, [], 0)

    return total_sum

if __name__ == "__main__":
    tuples_list = [(1, 2), (3, 4), (5, 6)]

    result = calculate_combinations_sum(tuples_list)
    print(f"Total sum of all possible combinations: {result}")
```

OUTPUT:

Total sum of all possible combinations: 84

7. Write a Python3 program that demonstrates how to convert a tuple into a list, where each element in the list is the succeeding element of the original tuple.

```
def convert_tuple_to_succeeding_list(tup):
    return [tup[i + 1] if i + 1 < len(tup) else None for i in range(len(tup))]

# Example usage
if __name__ == "__main__":
    tup = (1, 2, 3, 4, 5)
    result = convert_tuple_to_succeeding_list(tup)
    print(result)
```

OUTPUT:

[2, 3, 4, 5, None]

WEEK 4:

Exercise 1: Customer Directory Management System
Objective
The goal of this case study is to develop a Python program that manages customer information for a telecommunications company. This involves creating a function to handle customer data, extracting relevant information, and printing the data in various formats.

Requirements
1. Function Definition:
A. Define a function named tel_directory that takes a list of dictionaries representing customer information. Each dictionary contains:
a. customer_id (Unique identifier for the customer)
b. customer_name (Name of the customer)
c. Subscription_type (Type of subscription: "prepaid" or "postpaid")

2. Data Input:
B. Input at least 10 customer records using the tel_directory function.

3. Data Extraction:
C. Extract data from the list of dictionaries and create a list of lists. Each list should be structured based on different key combinations.

4. Output:
D. Print the extracted data in three different combinations of key fields:
a. Combination 1: A list containing [customer_id, customer_name, Subscription_type] b. Combination 2: A list containing [customer_id, customer_name]
c. Combination 3: A list containing [customer_name, Subscription_type]

Exercise 2: Enhancing the Customer Directory Management System

Add a new function named Search_Customer to the existing Customer Directory Management System.
This function should:
1. Function Purpose:
A. Search for a customer by their name within the directory.
2. Function Details:
A. The function should take the customer_name as input.
B. If the customer is found in the directory, display their information.
C. If the customer is not found, print a message indicating that the customer is not in the directory.

Exercise 3: Enhancing the Customer Directory Management System

Add a new function named Search_subscription to the existing Customer Directory Management System. This function should:

1. Function Purpose:
A. Search for customers based on their subscription type.
2. Function Details:
A. The function should take the Subscription_type ("prepaid" or "postpaid") as input.
B. Display the information of all customers who have the specified subscription type.
C. If no customers have the given subscription type, print appropriate message.

```
def tel_directory(customers):

    combo1 = [[cust['customer_id'], cust['customer_name'], cust['subscription_type']] for cust in customers]

    combo2 = [[cust['customer_id'], cust['customer_name']] for cust in customers]

    combo3 = [[cust['customer_name'], cust['subscription_type']] for cust in customers]


    print("Combination 1: [customer_id, customer_name, subscription_type]")

    print(combo1)
```

```
    print("Combination 2: [customer_id, customer_name]")
    print(combo2)

    print("Combination 3: [customer_name, subscription_type]")
    print(combo3)

def Search_Customer(customers, customer_name):
    found = False
    for cust in customers:
        if cust['customer_name'].lower() == customer_name.lower():
            print(f"Customer Found: ID: {cust['customer_id']}, Name: {cust['customer_name']},
Subscription Type: {cust['subscription_type']}")
            found = True
            break
    if not found:
        print("Customer not found in the directory.")

def Search_subscription(customers, subscription_type):
    found = False
    for cust in customers:
        if cust['subscription_type'].lower() == subscription_type.lower():
            print(f"Customer Found: ID: {cust['customer_id']}, Name: {cust['customer_name']},
Subscription Type: {cust['subscription_type']}")
            found = True
    if not found:
        print("No customers found with the specified subscription type.")
```

```
def main():
    customers = []

    num_customers = int(input("Enter the number of customers to add: "))
    for _ in range(num_customers):
        customer_id = input("Enter customer ID: ")
        customer_name = input("Enter customer name: ")
        subscription_type = input("Enter subscription type: ")

        # Adding customer information to the list
        customers.append({
            'customer_id': customer_id,
            'customer_name': customer_name,
            'subscription_type': subscription_type
        })

    # Display the different combinations
    tel_directory(customers)

    # Search for a customer by name
    search_name = input("Enter customer name to search: ")
    Search_Customer(customers, search_name)

    # Search for customers by subscription type
    search_subscription = input("Enter subscription type to search: ")
    Search_subscription(customers, search_subscription)
```

```
if __name__ == "__main__":

    main()
```

OUTPUT:

```
Enter the number of customers to add:  2
Enter customer ID:  101
Enter customer name:  MAYANK
Enter subscription type:  prepaid
Enter customer ID:  102
Enter customer name:  Nishant
Enter subscription type:  postpaid
Combination 1: [customer_id, customer_name, subscription_type]
[['101', 'MAYANK', 'prepaid'], ['102', 'Nishant', 'postpaid']]
Combination 2: [customer_id, customer_name]
[['101', 'MAYANK'], ['102', 'Nishant']]
Combination 3: [customer_name, subscription_type]
[['MAYANK', 'prepaid'], ['Nishant', 'postpaid']]
Enter customer name to search:  MAYANK
Customer Found: ID: 101, Name: MAYANK, Subscription Type: prepaid
Enter subscription type to search:  prepaid
Customer Found: ID: 101, Name: MAYANK, Subscription Type: prepaid
```

[ ]: