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))

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}")

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)} ")

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}")

5. Write a python program to print the current date in the following format

“Sun May 29 02:26:23 IST 2017”.

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)

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]

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}")

Week 2

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

of numbers.

# Longest subsequence from a given list of numbers

def lis(arr):

n = len(arr)

lis = [1]\*n

for i in range(1,n):

for j in range(0,i):

if arr[i] > arr[j]:

lis[i] = max(lis[i], lis[j]+1)

return max(lis)

arr = [10, 22, 9, 33, 21, 50, 41, 60]

print("Longest increasing subsequence of the list is", lis(arr))

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 list of 25 elements and find the number of frequency in the list

import random

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))

lis = []

i = 0

while i < 25:

lis\_items = lis.append(random.randint(1, 100))

i +=1

CountFrequency(lis)

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.

# To determine the strings with minimum 2 characters and also same start and end character from a list of 15 strings

def count\_strings\_with\_identical\_ends(strings):

count = 0

for s in strings:

if len(s) >= 2 and s[0] == s[-1]:

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)

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)

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()

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).

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.

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.

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.

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):

low = 0

high = len(sorted\_list) - 1

while low <= high:

mid = (low + high) // 2

if sorted\_list[mid] == target:

return mid

elif sorted\_list[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1

def main():

sorted\_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

target = int(input("Enter the number to search for: "))

index = binary\_search(sorted\_list, target)

if index != -1:

print(f"Element {target} found at index {index}.")

else:

print(f"Element {target} not found in the list.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

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.

from itertools import product

def sum\_of\_combinations(tuple\_list):

# Ensure that there is at least one tuple in the list

if not tuple\_list:

return 0

# Generate all possible combinations of elements from the tuples

combinations = product(\*tuple\_list)

# Calculate the summation of each combination

total\_sum = 0

for combination in combinations:

total\_sum += sum(combination)

return total\_sum

# Example usage

tuple\_list = [(1, 2), (3, 4), (5, 6)]

result = sum\_of\_combinations(tuple\_list)

print(f"The summation of all possible combinations is: {result}")

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 tuple\_to\_succeeding\_list(input\_tuple):

succeeding\_list = []

# Iterate through the tuple except for the last element

for i in range(len(input\_tuple) - 1):

# Append the succeeding element to the list

succeeding\_list.append(input\_tuple[i + 1])

return succeeding\_list

def main():

# Example tuple

example\_tuple = (10, 20, 30, 40, 50)

# Convert the tuple to the list of succeeding elements

result\_list = tuple\_to\_succeeding\_list(example\_tuple)

# Print the result

print(f"Original tuple: {example\_tuple}")

print(f"List of succeeding elements: {result\_list}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

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):

combination1 **=** [[customer['customer\_id'], customer['customer\_name'], customer['Subscription\_type']] **for** customer **in** customers]

combination2 **=** [[customer['customer\_id'], customer['customer\_name']] **for** customer **in** customers]

combination3 **=** [[customer['customer\_name'], customer['Subscription\_type']] **for** customer **in** customers]

print("Combination 1 (ID, Name, Subscription):")

**for** entry **in** combination1:

print(entry)

print("\nCombination 2 (ID, Name):")

**for** entry **in** combination2:

print(entry)

print("\nCombination 3 (Name, Subscription):")

**for** entry **in** combination3:

print(entry)

**def** collect\_data():

customers **=** []

**for** i **in** range(3):

print(f"\nEnter details for customer {i**+**1}:")

customer\_id **=** input("Customer ID: ")

customer\_name **=** input("Customer Name: ")

subscription\_type **=** input("Subscription Type (prepaid/postpaid): ")

**while** subscription\_type **not** **in** ['prepaid', 'postpaid']:

print("Invalid subscription type. Please enter 'prepaid' or 'postpaid'.")

subscription\_type **=** input("Subscription Type (prepaid/postpaid): ")

customers**.**append({

'customer\_id': customer\_id,

'customer\_name': customer\_name,

'Subscription\_type': subscription\_type

})

tel\_directory(customers)

search\_customer\_interface(customers)

search\_subscription\_interface(customers)

**def** Search\_Customer(customers, customer\_name):

found **=** **False**

**for** customer **in** customers:

**if** customer['customer\_name']**.**lower() **==** customer\_name**.**lower():

print("\nCustomer found:")

print(f"Customer ID: {customer['customer\_id']}")

print(f"Customer Name: {customer['customer\_name']}")

print(f"Subscription Type: {customer['Subscription\_type']}")

found **=** **True**

**break**

**if** **not** found:

print("Customer not found in the directory.")

**def** search\_customer\_interface(customers):

customer\_name **=** input("Enter the customer name to search: ")

Search\_Customer(customers, customer\_name)

**def** Search\_subscription(customers, subscription\_type):

found **=** **False**

**for** customer **in** customers:

**if** customer['Subscription\_type']**.**lower() **==** subscription\_type**.**lower():

print("\nCustomer found:")

print(f"Customer ID: {customer['customer\_id']}")

print(f"Customer Name: {customer['customer\_name']}")

print(f"Subscription Type: {customer['Subscription\_type']}")

found **=** **True**

**if** **not** found:

print(f"No customers found with the subscription type '{subscription\_type}'.")

**def** search\_subscription\_interface(customers):

subscription\_type **=** input("Enter the subscription type to search (prepaid/postpaid): ")

**while** subscription\_type **not** **in** ['prepaid', 'postpaid']:

print("Invalid subscription type. Please enter 'prepaid' or 'postpaid'.")

subscription\_type **=** input("Enter the subscription type to search (prepaid/postpaid): ")

Search\_subscription(customers, subscription\_type)

*# Run the collection of data*

collect\_data()