CALL REQUESTS:

#import requests

#response=requests.get("https://dummyjson.com/products/1")

#print(response.json())

import pandas as pd

#Creating a DataFrame from a dictionary with Indian names

data = {

"Name": ["Amit", "Priya", "Vikram", "Neha", "Ravi"],

"Age": [25, 30, 35, 48, 45],

"City": ["Mumbai", "Delhi", "Bangalore", "Chennai", "Pune"]}

df =pd.DataFrame(data)

#print(df)

#Accessing a single column

#print(df['Name'])

#Accessing multiple columns

# print(df[['Name','Age']])

#Accessing rows using index

#print(df.iloc[0]) #First row

#print(df.iloc[0:3]) #multiple rows

#Filtering rows based on conditions

#filtered\_df=df[df['Age']>30]

#print(filtered\_df)

df['Salary']=[50000,70000,65000,86000,94000]

#print(df)

df\_renamed=df.rename(columns={'Name':'Fullname','Age':'Years'})

#print(df\_renamed)

df\_dropped=df.drop(columns=['City'])

#print(df\_dropped)

#drop the row by index

df\_dropped\_row=df.drop(index=2)

#print(df\_dropped\_row)

#Create a new column 'Seniority' based on the Age

df['Seniority'] = df['Age'].apply(lambda x: 'Senior' if x >= 35 else 'Junior')

#print(df)

df\_grouped=df.groupby("City")['Salary'].mean()

#print(df\_grouped)

# Apply a custom function to the 'Salary' column to add a 10% bonus

#def add\_bonus (salary):

#return salary \* 1.10

#df['Salary\_With\_Bonus'] = df ['Salary'].apply(add\_bonus)

#print(df)

#Create another DataFrame

#df\_new = pd.DataFrame({

#"Name": ["Amit", "Priya", "Ravi"],

#"Bonus": [5000, 6000, 7000]

#})

# Merge based on the 'Name' column

#df\_merged = pd.merge(df, df\_new, on="Name", how="left")

#print(df\_merged)

df\_new = pd.DataFrame({

"Name": ["Sonia", "Rahul"],

"Age": [29, 31],

"City": ["Kolkata", "Hyderabad"],

"Salary": [58000, 63000]

})

# Concatenate the two DataFrames

df\_concat = pd.concat([df,df\_new], ignore\_index=True)

#print(df\_concat)

filter\_df=df[df['Salary']>50000]

print(filter\_df)

filtering\_df = df[df['Name'].str.startswith('A')]

print(filtering\_df)

CREATE FILE EXERCISE:

#Exercise 1

import json

data={'name':'John Doe','age':30,'city':'New York','skills':['Python','Machine Learning','Data Analysis']}

with open(r"C:\Training\data1.json", "w") as file:

json.dump(data,file)

with open("C:/Training/data1.json", "r") as file:

loaded\_data = json.load(file)

print(loaded\_data)

#Exercise 2

import json

profile = {

"name": "Jane Smith",

"age": 28,

"city": "Los Angeles",

"hobbies": ["Photography", "Traveling", "Reading"]

}

with open('profile.json', 'w') as file:

json.dump(profile, file)

print("Profile saved to profile.json")

#Exercise 3

import csv

import json

students\_list = []

with open('students.csv','r') as file:

reader = csv.DictReader(file)

for row in reader:

students\_list.append(row)

print(students\_list)

with open('students.json', 'w') as file:

json.dump(students\_list, file)

print("Data saved to students.json")

#Exercise 4

import json

import csv

with open('data.json', 'r') as json\_file:

data = json.load(json\_file)

if isinstance(data, list) and all(isinstance(item, dict) for item in data):

headers = data[0].keys()

with open('data.csv', 'w', newline='') as csv\_file:

writer = csv.DictWriter(csv\_file, fieldnames=headers)

writer.writeheader()

writer.writerows(data)

print("Data successfully converted to data.csv")

else:

print("Error: JSON data is not in the expected format.")

#Exercise 5

books: [

{"title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "year": 1925},

{"title": "War and Peace", "author": "Leo Tolstoy", "year": 1869},

{"title": "The Catcher in the Rye", "author": "J.D. Salinger", "year": 1951}

]

import json

with open('books.json', 'r') as file:

data = json.load(file)

books = data.get('books', [])

for book in books:

print(book.get('title', 'No title available'))

INHERITANCE:

class Animal: #def-method, init-constructor

def \_\_init\_\_(self,name): #self is to refer the object within the class

self.name=name

def speak(self):

pass

class Dog(Animal):

def speak(self):

return 'Woofff!'

class Cat(Animal):

def speak(self):

return 'Meowww!'

dog=Dog('Buddy')

print(f"{dog.name} says:{dog.speak()}")

cat=Cat("Shitzu")

print(f"{cat.name} says:{cat.speak()}")

class Payment:

def \_\_init\_\_(self,name):

self.name=name

def get\_Payment(self):

pass

class Gpay(Payment):

def get\_Payment(self):

return "Speaking to Google Payment Gateway"

class PhonePay(Payment):

def get\_Payment(self):

return "Speaking to Phonepay Gateway"

class AmazonPay(Payment):

def get\_Payment(self):

return "Speaking to Amazonpay Gateway"

gpay= Gpay('GooglePay')

ppay=PhonePay("PhonePe")

amPay=AmazonPay("AmazonPay")

print(f"{gpay.name} says: {gpay.get\_Payment()}")

print(f"{ppay.name} says: {ppay.get\_Payment()}")

print(f"{amPay.name} says: {amPay.get\_Payment()}")

SET:

#Creating a set of fruits

fruits = {"apple", "banana", "orange"}

#Adding an element to the set

fruits.add("grape")

#Removing an element from the set

fruits.remove("banana")

#Checking if an element is in the set

print("Is 'apple' in the set?", "apple" in fruits)

print("Is 'banana' in the set?", "banana" in fruits)

#Length of the set

set\_length = len(fruits)

print("Number of elements in the set:", set\_length)

# Looping through the set

print("Set elements:")

for fruit in fruits:

print(fruit)

TUPLE:

colors=('Red','Green','Blue')

#Accessing Elements

print('First color: ',colors[0])

print('Last color: ',colors[-1])

#Length of the tuple

tuple\_length = len(colors)

print("Length of the tuple:", tuple\_length)

# Looping through the tuple

print("Tuple elements:")

for color in colors:

print(color)

LIST OF DICTIONARIES:

#List of dictionaries representing students' information

students = [

{"name": "Alice", "age": 20, "grade": "A"},

{"name": "Bob", "age": 22, "grade": "B"},

{"name": "Charlie", "age": 21, "grade": "C"},

{"name": "David", "age": 23, "grade": "B"}]

# Accessing and manipulating individual records

print(students[0]["name"]) # Accessing the name of the first student

students[1]["age"] = 24 #Modifying the age of the second student

#Adding a new student record to the list

new\_student = {"name": "Eva", "age": 19, "grade": "A"}

students.append(new\_student)

# Iterating through the list of students

for student in students:

print(f"Name: {student['name']}, Age: {student['age']}, Grade: {student['grade']}")

DICTIONARY EXERCISE:

#Exercise1

#1

person={

'name':'Alice',

'age':25,

'city':'New York'

}

#2

print(person)

#Exercise2

#1

print(person['city'])

#Exercise3

#1

person['email']='alice@example.com'

#2

person['age']=26

#3

print(person)

#Exercise4

#1

del person['city']

#2

print(person)

#Exercise5

#1

if 'email' in person:

print('The email key is present')

else:

print('Key not found')

#2

if 'phone' in person:

print('The phone key is present')

else:

print('Key not found')

#Exercise6

#1

for key, value in person.items():

print(f"{key}: {value}")

#2

for key in person.keys():

print(key)

#3

for value in person.values():

print(value)

#Exercise7

#1

employees = {

101: {"name": "Bob", "job": "Engineer"},

102: {"name": "Sue", "job": "Designer"},

103: {"name": "Tom", "job": "Manager"}

}

#2

print(employees[102])

#3

employees[104]={'name':'Linda','job':'HR'}

#4

print(employees)

#Exercise8

#1

squared\_numbers={z\*\*2 for z in range(1,6)}

#2

print(squared\_numbers)

#Exercise9

#1

dict1 = {"a": 1, "b": 2}

dict2 = {"c": 3, "d": 4}

#2

merged\_dict= dict1 | dict2

print(merged\_dict)

#Exercise10

#1

letters={"a": 1, "b": 2, "c": 3}

value=letters.get('b')

print(value)

#2

letters={"a": 1, "b": 2, "c": 3}

value=letters.get('d',0)

print(value)

#Exercise11

#1

keys = ["name", "age", "city"]

values = ["Eve", 29, "San Francisco"]

#2

dictionary = dict(zip(keys, values))

#3

print(dictionary)

#Exercise12

#1

sentence = "the quick brown fox jumps over the lazy dog the fox"

words = sentence.split()

word\_count = {}

for word in words:

word\_count[word] = word\_count.get(word, 0) + 1

#2

print(word\_count)

STRING UTILS:

def capitalize\_first\_letter(sentence):

words = sentence.split()

capitalized\_words = [word.capitalize() for word in words]

capitalized\_sentence = ' '.join(capitalized\_words)

return capitalized\_sentence

def reverse\_string(s):

reversed\_s = s[::-1]

return reversed\_s

def count\_vowels(s):

vowels='aeiouAEIOU'

count = 0

for char in s:

if char in vowels:

count += 1

return count

def is\_palindrome(s):

cleaned\_s = s.replace(" ", "").lower()

return cleaned\_s == cleaned\_s[::-1]