1.

a. Write a list comprehension that generates a list of squares of the numbers from 1 to 10.

l=[x\*\*2 for x in range(1,11)]

print(l)

b. Write a list comprehension to extract all the vowels from the string "hello world".

l=[i for i in 'hello world' if i in 'aeiou']

print(l)

2.

a. Create a dictionary where the keys are numbers from 1 to 5 and the values are the cubes of the keys.

l = {x: x\*\*3 for x in range(1, 6)}

print(l)

b. Create a dictionary where the keys are the first letters of the words in the list ["apple", "banana", "cherry"] and the values are the words themselves.

w = ["apple", "banana", "cherry"]

w\_dict = {w[0]: w for w in w}

print(w\_dict)

3.

a. Write a lambda function that adds 10 to a given number and use it to add 10 to the number 5.

add= lambda x:x+10

res=add(5)

print(res)

b. Write a lambda function that checks if a number is even and use it to test the number 4

evenodd= lambda x:'even' if x%2==0 else 'odd'

res=evenodd(4)

print(res)

4.

a. Use the filter function to filter out the odd numbers from the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even\_numbers)

b. Use the filter function to remove all strings shorter than 4 characters from the list ["cat", "dog", "elephant", "rat"].

words = ["cat", "dog", "elephant", "rat"]

filter\_words = list(filter(lambda x: len(x) >= 4, words))

print(filter\_words)

5.

a. Use the map function to double all the numbers in the list [1, 2, 3, 4, 5].

l=[1, 2, 3, 4, 5]

double=list(map(lambda x : 2\*x ,l))

print(double)

b. Use the map function to convert a list of integers [1, 2, 3] to their corresponding string representations.

numbers = [1, 2, 3]

string\_numbers = list(map(str, numbers))

print(string\_numbers)

6.

a. Use the reduce function to find the product of the numbers in the list [1, 2, 3, 4, 5].

from functools import reduce

numbers = [1, 2, 3, 4, 5]

product = reduce(lambda x, y: x \* y, numbers)

print(product)

b. Use the reduce function to concatenate a list of strings ["Hello", "World", "from", "Python"] into a single string.

from functools import reduce

strings = ["Hello", "World", "from", "Python"]

concatenated\_string = reduce(lambda x, y: x + " " + y, strings)

print(concatenated\_string)

7.

a. Create a generator function that yields the first 5 even numbers.

def generate\_even\_numbers():

count = 0

num = 2

while count < 5:

yield num

num += 2

count += 1

for i in generate\_even\_numbers():

print(i)

b. Create a generator function that yields numbers in the Fibonacci sequence up to the 10th number.

def fibonacci\_generator(n):

a, b = 0, 1

count = 0

while count < n:

yield a

a, b = b, a + b

count += 1

for number in fibonacci\_generator(10):

print(number)

8.

a. Write a function that takes two numbers and returns their division, handling the division by zero exception.

def safe\_divide(numerator, denominator):

try:

result = numerator / denominator

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

return result

print(safe\_divide(10, 2))

print(safe\_divide(10, 0))

b. Write a function that reads a file and handles the file not found exception.

def read\_file(file\_path):

try:

with open(file\_path, 'r') as file:

content = file.read()

return content

except FileNotFoundError:

return f"Error: The file '{file\_path}' was not found."

print(read\_file('existing\_file.txt'))

print(read\_file('non\_existent\_file.txt'))

9. File I/O

a. Write a program that writes the string "Hello, World!" to a file and then reads it back.

file\_name = "example.txt"

with open(file\_name, 'w') as file:

file.write("Hello, World!")

with open(file\_name, 'r') as file:

content = file.read()

print(content)

b. Write a program that appends the string "Goodbye!" to an existing ϐile and then reads and prints the file content.

file\_name = "example.txt"

with open(file\_name, 'a') as file:

file.write("Goodbye!")

with open(file\_name, 'r') as file:

content = file.read()

print(content)

10.

a. Use regular expressions to ϐind all the words starting with 'a' in the string "apple and banana are amazing".

import re

text = "apple and banana are amazing"

pattern = r'\ba\w\*'

words\_starting\_with\_a = re.findall(pattern, text)

print(words\_starting\_with\_a)

b. Use regular expressions to extract all the email addresses from the string "contact us at email@example.com or admin@site.org".

import re

text = "contact us at email@example.com or admin@site.org"

pattern = r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

emails = re.findall(pattern, text)

print(emails)

11. Classes and Objects a. Create a class Person with attributes name and age. Instantiate an object of this class and print its attributes.

class Person:

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

self.name = name

self.age = age

person = Person(name="Alice", age=30)

print("Name: {person.name}")

print("Age: {person.age}")

b. Create a class Car with attributes make, model, and year. Instantiate an object of this class and print its attributes.

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

car = Car(make="Toyota", model="Corolla", year=2020)

print(f"Make: {car.make}")

print(f"Model: {car.model}")

print(f"Year: {car.year}")

12. Inheritance a. Create a class Employee that inherits from the Person class and adds an attribute salary. Instantiate an object of Employee and print all its attributes.

class Person:

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

self.name = name

self.age = age

class Employee(Person):

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

super().\_\_init\_\_(name, age)

self.salary = salary

employee = Employee(name="Alice", age=30, salary=50000)

print(f"Name: {employee.name}")

print(f"Age: {employee.age}")

print(f"Salary: {employee.salary}")

b. Create a class Manager that inherits from Employee and adds an attribute department. Instantiate an object of Manager and print all its attributes.

class Manager(Employee):

def \_\_init\_\_(self, name, age, salary, department):

super().\_\_init\_\_(name, age, salary)

self.department = department

manager = Manager(name="Bob", age=45, salary=70000, department="Sales")

print(f"Name: {manager.name}")

print(f"Age: {manager.age}")

print(f"Salary: {manager.salary}")

print(f"Department: {manager.department}")

13. Static Methods

a. Add a static method to the Person class that returns a greeting message.

class Person:

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

self.name = name

self.age = age

def get\_greeting():

return "Hello! Welcome to our platform."

print(Person.get\_greeting())

b. Add a static method to the MathUtils class that returns the square of a given number.

class MathUtils:

def square(number):

return number \* number

print(MathUtils.square(5)) # Output: 25

14. Class Methods

a. Add a class method to the Person class that returns a new instance of Person with a default name and age.

class Person:

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

self.name = name

self.age = age

def default\_person(cls):

return cls(name="John Doe", age=30)

default\_person = Person.default\_person()

print(f"Name: {default\_person.name}")

print(f"Age: {default\_person.age}")

b. Add a class method to the Book class that creates a new Book instance from a given title and author

class Book:

def \_\_init\_\_(self, title, author):

self.title = title

self.author = author

def from\_title\_author(cls, title, author):

return cls(title=title, author=author)

book = Book.from\_title\_author(title="1984", author="George Orwell")

print(f"Title: {book.title}")

print(f"Author: {book.author}")

15. Magic Methods

a. Override the \_\_str\_\_ method in the Person class to return a formatted string.

class Person:

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

self.name = name

self.age = age

def \_\_str\_\_(self):

return f"Person(name={self.name}, age={self.age})"

person = Person(name="Alice", age=30)

print(person)

b. Override the \_\_add\_\_ method in a Vector class to add two vectors together.

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, other):

if isinstance(other, Vector):

return Vector(self.x + other.x, self.y + other.y)

return NotImplemented

def \_\_repr\_\_(self):

return f"Vector(x={self.x}, y={self.y})"

# Example usage:

v1 = Vector(1, 2)

v2 = Vector(3, 4)

result = v1 + v2

print(result)

16. Context Managers

a. Create a context manager that prints "Entering" when entering the context and "Exiting" when exiting the context.

class SimpleContextManager:

def \_\_enter\_\_(self):

print("Entering")

return self

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

print("Exiting")

with SimpleContextManager():

print("Inside the context")

b. Create a context manager that temporarily changes the current working directory and restores it back when exiting the context

import os

from contextlib import contextmanager

def change\_directory(new\_dir):

old\_dir = os.getcwd() # Save the current working directory

os.chdir(new\_dir) # Change to the new directory

try:

yield

finally:

os.chdir(old\_dir) # Restore the old directory

with change\_directory('/path/to/new/directory'):

print("Inside the new directory")

# Perform operations in the new directory

print("Back to the original directory")

17. Iterators

a. Create a custom iterator that returns numbers from 1 to 5.

class NumberIterator:

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

self.current = 1

self.end = 5

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.current > self.end:

raise StopIteration

number = self.current

self.current += 1

return number

# Example usage:

for number in NumberIterator():

print(number)

b. Create a custom iterator that returns the characters in a string one by one.

class StringIterator:

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

self.text = text

self.index = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.index >= len(self.text):

raise StopIteration

char = self.text[self.index]

self.index += 1

return char

for char in StringIterator("hello"):

print(char)

18. Modules and Packages

a. Create a simple module my\_module.py with a function greet that prints "Hello, World!". Import and use this function in another script.

def greet():

print("Hello, World!")

import my\_module

my\_module.greet()

b. Create a package my\_package with a module utils.py that contains a function square that returns the square of a number. Import and use this function in another script.

my\_package/

\_\_init\_\_.py

utils.py

def square(number):

return number \* number

from my\_package.utils import square

print(square(5))

19. Date and Time

a. Write a program that gets the current date and time and prints it in the format "YYYYMM-DD HH:MM".

from datetime import datetime

now = datetime.now()

formatted\_time = now.strftime("%Y-%m-%d %H:%M")

print(formatted\_time)

b. Write a program that calculates the number of days between two given dates.

from datetime import datetime

date\_format = "%Y-%m-%d"

date1 = datetime.strptime("2024-01-01", date\_format)

date2 = datetime.strptime("2024-07-23", date\_format)

difference = (date2 - date1).days

print(f"Number of days between the dates: {difference}")

20. Collections

a. Use the collections.Counter class to count the occurrences of each character in the string "abracadabra".

from collections import Counter

text = "abracadabra"

counter = Counter(text)

print(counter)

b. Use the collections.defaultdict class to create a dictionary that returns a default value of 0 for any new keys

from collections import defaultdict

default\_dict = defaultdict(int)

default\_dict['apple'] += 1

default\_dict['banana'] += 2

print(default\_dict['apple']) # Output: 1

print(default\_dict['banana']) # Output: 2

print(default\_dict['cherry']) # Output: 0 (default value for new keys)