**1. List Comprehensions**

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

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

print(squares)

#Output: [1,4,9,16,25,36,49,64,81,100]

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

vowels = [char for char in "hello world" if char in 'aeiou']

print(vowels) # ['e', 'o', 'o']

**2. Dictionary Comprehension**

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

**of the keys.**

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

print(cube\_dict) #{1: 1, 2: 8, 3: 27, 4: 64, 5: 125}

**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.**

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

first\_letter\_dict = {word[0]: word for word in words}

print(first\_letter\_dict) #{'a': 'apple', 'b': 'banana', 'c': 'cherry'}

**3. Lambda Functions**

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

**number 5.**

add\_ten = lambda x: x + 10

result = add\_ten(5)

print(result) #15

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

is\_even = lambda x: x % 2 == 0

result = is\_even(4)

print(result) #True

**4. Filter Function**

**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]

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

print(filtered\_numbers) #[2, 4, 6, 8, 10]

**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"]

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

print(filtered\_words) #['elephant']

**5. Map Function**

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

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

doubled\_numbers = list(map(lambda x: x \* 2, numbers))

print(doubled\_numbers) #[2, 4, 6, 8, 10]

**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(lambda x: str(x), numbers))

print(string\_numbers) #['1', '2', '3']

**6. Reduce Function**

**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) #120

**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) #Hello World from Python

**7. Generator Functions**

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

def first\_five\_evens():

count = 0

num = 0

while count < 5:

if num % 2 == 0:

yield num

count += 1

num += 1

# Using the generator function

evens = list(first\_five\_evens())

print(evens) #[0, 2, 4, 6, 8]

**b. Create a generator function that yields numbers in the Fibonacci sequence up to the**

**10th number.**

def fibonacci\_sequence():

a, b = 0, 1

count = 0

while count < 10:

yield a

a, b = b, a + b

count += 1

# Using the generator function

fibonacci\_numbers = list(fibonacci\_sequence())

print(fibonacci\_numbers) #[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

**8. Handling Exceptions**

**a. Write a function that takes two numbers and returns their division, handling the division**

**by zero exception.**

def divide\_numbers(a, b):

try:

result = a / b

return result

except ZeroDivisionError:

return "Division by zero is not allowed."

# Example

print(divide\_numbers(10, 2)) # Output: 5.0

print(divide\_numbers(10, 0)) # Output: Division by zero is not allowed.

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

def read\_file(filename):

try:

with open(filename, 'r') as file:

content = file.read()

return content

except FileNotFoundError:

return f"File '{filename}' not found."

# Example usage

print(read\_file('some sample.txt')) # actual file path should be used here

**9. File I/O**

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

# Writing to file

with open('output.txt', 'w') as file:

file.write("Hello, World!")

# Reading from file

with open('output.txt', 'r') as file:

content = file.read()

print(content)

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

**and prints the file content.**

# Appends the file

with open('output.txt', 'a') as file:

file.write("\nGoodbye!")

# Reading from file

with open('output.txt', 'r') as file:

content = file.read()

print(content)

**10. Regular Expressions**

**a. Use regular expressions to find all the words starting with 'a' in the string "apple and**

**banana are amazing".**

import re

text = "apple and banana are amazing"

words\_with\_a = re.findall(r'\b[aA]\w+', text)

print(words\_with\_a) #['apple', 'and', 'are', 'amazing']

**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"

emails = re.findall(r'\b[\w.-]+@[a-zA-Z.-]+\.\w+\b', text)

print(emails) #['email@example.com', 'admin@site.org']

**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

person1=Person("Lokesh",23)

print(f"Name:{person1.name},Age: {person1.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

car1 = Car("Audi", "Cameo", 2024)

print(f"Make: {car1.make}, Model: {car1.model}, Year: {car1.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

employee1 = Employee("Ram", 23, 40000)

print(f"Name: {employee1.name}, Age: {employee1.age}, Salary: ${employee1.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 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

class Manager(Employee):

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

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

self.department = department

manager1 = Manager("Swamy", 30, 80000, "Operations")

print(f"Name: {manager1.name}, Age: {manager1.age}, Salary: ${manager1.salary}, Department: {manager1.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

@staticmethod

def greet():

return "Hello, nice to meet you!"

print(Person.greet())

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

class MathUtils:

@staticmethod

def square(x):

return x \*\* 2

print(MathUtils.square(5))

**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

@classmethod

def create\_default(cls):

return cls("Unknown", 0)

default\_person = Person.create\_default()

print(f"Default Person - Name: {default\_person.name}, 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

@classmethod

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

return cls(title, author)

book = Book.from\_title\_author("Python Fundamentals", "Lokesh")

print(f"Book Title: {book.title}, 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: {self.name}, Age: {self.age}"

person1 = Person("Lokesh", 23)

print(person1)

**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):

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

vector1 = Vector(3, 8)

vector2 = Vector(6, 4)

result\_vector = vector1 + vector2

print(f"Result Vector - x: {result\_vector.x}, y: {result\_vector.y}")

**16. Context Managers**

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

**when exiting the context.**

from contextlib import contextmanager

@contextmanager

def custom\_context():

print("Entering")

yield

print("Exiting")

with custom\_context():

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

@contextmanager

def change\_directory(new\_path):

current\_path = os.getcwd()

try:

os.chdir(new\_path)

yield

finally:

os.chdir(current\_path)

# Using the context manager

with change\_directory('/tmp'):

print("Current working directory:", os.getcwd())

# After exiting the context, the original working directory is restored

print("Current working directory:", os.getcwd())

**17. Iterators**

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

class NumberIterator:

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

self.current = 1

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

return self

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

if self.current > 5:

raise StopIteration

else:

self.current += 1

return self.current - 1

# Using the custom iterator

numbers = NumberIterator()

for num in numbers:

print(num)

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

class StringIterator:

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

self.string = string

self.index = 0

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

return self

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

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

raise StopIteration

else:

self.index += 1

return self.string[self.index - 1]

# Using the custom iterator

text = "Lokesh Ram"

characters = StringIterator(text)

for char in characters:

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.**

# my\_module.py

def greet():

print("Hello, World!")

# Using my\_module

import my\_module

my\_module.greet() # Output: Hello, World!

**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.**

# utils.py

def square(x):

return x \*\* 2

# Using utils from my\_package

from my\_package.utils import square

result = square(5)

print(result) # Output: 25

**19. Date and Time**

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

from datetime import datetime

current\_datetime = datetime.now()

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

print(formatted\_datetime)

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

**from datetime import datetime**

date\_str1 = "2023-07-01"

date\_str2 = "2024-07-15"

date1 = datetime.strptime(date\_str1, "%Y-%m-%d")

date2 = datetime.strptime(date\_str2, "%Y-%m-%d")

delta = date2 - date1

print(f"Number of days between {date\_str1} and {date\_str2}: {delta.days}") #Output:380

**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"

char\_counts = Counter(text)

print(char\_counts)

**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

# Using defaultdict with int (returns 0 for missing keys)

d = defaultdict(int)

print(d['key1']) # Output: 0

# Using defaultdict with list (returns empty list for missing keys)

d = defaultdict(list)

print(d['key2']) # Output: []