**ASSIGNMENT 3**

**NAME : GUNESH G BHAT**

**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)

-------------------------------------------------------------------------------------------------------------------

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

vowel = [char for char in "hello world" if char in "aeiou"]

print(vowel)

-------------------------------------------------------------------------------------------------------------------

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

print(cube\_dictionary)

-------------------------------------------------------------------------------------------------------------------

**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)

-------------------------------------------------------------------------------------------------------------------

**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)

-------------------------------------------------------------------------------------------------------------------

**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)

**-------------------------------------------------------------------------------------------------------------------**

**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)

**-------------------------------------------------------------------------------------------------------------------**

**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)

**-------------------------------------------------------------------------------------------------------------------**

**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)

**-------------------------------------------------------------------------------------------------------------------**

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

**representations.**

numbers = [1, 2, 3]

number\_strings = list(map(lambda x: str(x), numbers))

print(number\_strings)

**-------------------------------------------------------------------------------------------------------------------**

**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

def multiply(x, y):

return x \* y

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

product = reduce(multiply, 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

def concatenate\_strings(x, y):

return x + " " + y

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

concatenated\_string = reduce(concatenate\_strings, strings)

print(concatenated\_string)

**-------------------------------------------------------------------------------------------------------------------**

**7. Generator Functions**

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

def first\_five\_even\_numbers():

num = 0

count = 0

while count < 5:

if num % 2 == 0:

yield num

count += 1

num += 1

even = list(first\_five\_even\_numbers())

print(even)

**-------------------------------------------------------------------------------------------------------------------**

**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

fibonacci = list(fibonacci\_sequence())

print(fibonacci)

**-------------------------------------------------------------------------------------------------------------------**

**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

except ZeroDivisionError:

result = float('inf')

return result

# Example

print(divide\_numbers(10, 2))

print(divide\_numbers(5, 0))

print(divide\_numbers(-8, 4))

print(divide\_numbers(0, 10))

**-------------------------------------------------------------------------------------------------------------------**

**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()

except FileNotFoundError:

content = "File not found"

return content

**-------------------------------------------------------------------------------------------------------------------**

**9. File I/O**

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

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

file.write("Hello, World!")

with open('hello.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.**

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

file.write("\nGoodbye!")

with open('hello.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\_starting\_with\_a = re.findall(r'\b[aA]\w+', 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"

emails = re.findall(r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', 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("Julie", 40)

print(person.name, 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("Hyundai", "Creta", 2017)

print(car.make, car.model, 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("Lokesh", 23, 40000)

print(employee.name)

print(employee.age)

print(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 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

manager = Manager("Karthik", 23, 80000, "Full stack")

print(manager.name)

print(manager.age)

print(manager.salary)

print(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

@staticmethod

def greet():

return "Hello!"

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("Anonymous", 0)

default\_person = Person.create\_default()

print(default\_person.name, 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("Java Programming ", "Balaguruswamy")

print(book.title, 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}"

person = Person("Thomas", 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):

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

v1 = Vector(1, 2)

v2 = Vector(3, 4)

result = v1 + v2

print(result.x, result.y)

**-------------------------------------------------------------------------------------------------------------------**

**16. Context Managers**

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

class MyContextManager:

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

print("Entering")

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

print("Exiting")

with MyContextManager():

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

class ChangeDirContext:

def \_\_init\_\_(self, new\_dir):

self.new\_dir = new\_dir

self.saved\_dir = None

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

self.saved\_dir = os.getcwd()

os.chdir(self.new\_dir)

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

os.chdir(self.saved\_dir)

with ChangeDirContext('/tmp'):

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

print("Current directory (after context):", os.getcwd())

**-------------------------------------------------------------------------------------------------------------------**

**17. Iterators**

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

class NumIterator:

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

self.current = 0

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

return self

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

self.current += 1

if self.current > 5:

raise StopIteration

return self.current

numbers = NumIterator()

for num in numbers:

print(num)

**-------------------------------------------------------------------------------------------------------------------**

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

class StringIter:

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

result = self.text[self.index]

self.index += 1

return result

text\_iter = StringIter("Hello")

for char in text\_iter:

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.**

# File: my\_module.py

def greet():

print("Hello, World!")

# Usage in another script:

from my\_module import greet

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.**

# File: my\_package/utils.py

def square(x):

return x \*\* 2

# Usage in another script:

from my\_package.utils import square

result = square(5)

print(result)

**-------------------------------------------------------------------------------------------------------------------**

**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\_datetime = now.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 = "2023-01-01"

date\_str2 = "2023-12-31"

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

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

delta = date2 - date1

print(delta.days)

**-------------------------------------------------------------------------------------------------------------------**

**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\_count = Counter(text)

print(char\_count)

**-------------------------------------------------------------------------------------------------------------------**

**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\_dictionary = defaultdict(int)

default\_dictionary['a'] = 1

print(default\_dictionary['a'])

print(default\_dictionary['b'])

**-------------------------------------------------------------------------------------------------------------------**