**DATA ENGINEERING PYTHON ASSIGNMENT 01**

**# Question 1: List Comprehensions**

**# a. Generate a list of squares of numbers from 1 to 10**

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

print("Squares:", squares)

**# b. Extract vowels from the string "hello world"**

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

print("Vowels:", vowels)

**# Question 2: Dictionary Comprehension**

**# a. Create a dictionary where keys are numbers 1 to 5 and values are cubes of keys**

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

print("Cube Dictionary:", cube\_dict)

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

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

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

print("First Letter Dictionary:", first\_letter\_dict)

**# Question 3: Lambda Functions**

**# a. Lambda function to add 10 to a number and apply to 5**

add\_ten = lambda x: x + 10

result\_add\_ten = add\_ten(5)

print("Result of adding 10 to 5:", result\_add\_ten)

**# b. Lambda function to check if a number is even and apply to 4**

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

result\_is\_even = is\_even(4)

print("Is 4 even?", result\_is\_even)

**# Question 4: Filter Function**

**# a. Filter out odd numbers from [1, 2, 3, ..., 10]**

even\_numbers = list(filter(lambda x: x % 2 == 0, range(1, 11)))

print("Even numbers:", even\_numbers)

**# b. Remove strings shorter than 4 characters from ["cat", "dog", "elephant", "rat"]**

filtered\_words = list(filter(lambda x: len(x) >= 4, ["cat", "dog", "elephant", "rat"]))

print("Filtered words:", filtered\_words)

**# Question 5: Map Function**

**# a. Double all numbers in [1, 2, 3, 4, 5]**

doubled\_numbers = list(map(lambda x: x \* 2, [1, 2, 3, 4, 5]))

print("Doubled numbers:", doubled\_numbers)

**# b. Convert integers [1, 2, 3] to their string representations**

str\_representations = list(map(lambda x: str(x), [1, 2, 3]))

print("String representations:", str\_representations)

**# Question 6: Reduce Function**

from functools import reduce

**# a. Find product of numbers in [1, 2, 3, 4, 5]**

product = reduce(lambda x, y: x \* y, [1, 2, 3, 4, 5])

print("Product of numbers:", product)

**# b. Concatenate strings ["Hello", "World", "from", "Python"] into one string**

concatenated\_string = reduce(lambda x, y: x + " " + y, ["Hello", "World", "from", "Python"])

print("Concatenated string:", concatenated\_string)

**# Question 7: Generator Functions**

**# a. Generate first 5 even numbers**

def generate\_even\_numbers():

num = 2

count = 0

while count < 5:

yield num

num += 2

count += 1

generated\_evens = list(generate\_even\_numbers())

print("Generated even numbers:", generated\_evens)

**# b. Generate Fibonacci sequence up to the 10th number**

def fibonacci\_sequence():

a, b = 0, 1

count = 0

fibonacci\_list = []

while count < 10:

fibonacci\_list.append(a)

a, b = b, a + b

count += 1

return fibonacci\_list

fibonacci\_numbers = fibonacci\_sequence()

print("Fibonacci sequence:", fibonacci\_numbers)

**# Question 8: Handling Exceptions**

**# a. Divide two numbers handling division by zero**

def divide\_numbers(a, b):

try:

result = a / b

print(f"Division result: {result}")

except ZeroDivisionError:

print("Division by zero is not allowed.")

divide\_numbers(10, 2)

**# b. Read a file handling file not found exception**

def read\_file(file\_path):

try:

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

content = f.read()

print("File content read successfully.")

return content

except FileNotFoundError:

print("File not found.")

read\_file("test.txt")

**# Question 9: File I/O**

**# a. Write "Hello, World!" to a file and read it back**

def write\_and\_read\_file():

with open("test.txt", 'w') as f:

f.write("Hello, World!")

print("Content written to file.")

with open("test.txt", 'r') as f:

content = f.read()

print("File content read successfully.")

return content

write\_and\_read\_file()

**# b. Append "Goodbye!" to an existing file and read its content**

def append\_and\_read\_file():

with open("test.txt", 'a') as f:

f.write("Goodbye!")

print("Content appended to file.")

with open("test.txt", 'r') as f:

content = f.read()

print("File content read successfully.")

return content

append\_and\_read\_file()

**# Question 10: Regular Expressions**

**# a. Find all words starting with 'a' in "apple and banana are amazing"**

words\_starting\_with\_a = re.findall(r'\ba\w+', "apple and banana are amazing")

print("Words starting with 'a':", words\_starting\_with\_a)

**# b. Extract all email addresses from "contact us at email@example.com or admin@site.org"**

email\_addresses = re.findall(r'\b[\w.-]+@[a-zA-Z-]+\.[a-zA-Z.]{2,6}\b', "contact us at email@example.com or admin@site.org")

print("Email addresses found:", email\_addresses)

**# Question 11: Classes and Objects**

**# a. Define a Person class and instantiate an object**

class Person:

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

self.name = name

self.age = age

person\_obj = Person("Alice", 30)

print("Person object created:", person\_obj.name, person\_obj.age)

**# b. Define a Car class and instantiate an object**

class Car:

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

self.make = make

self.model = model

self.year = year

car\_obj = Car("Toyota", "Camry", 2022)

print("Car object created:", car\_obj.make, car\_obj.model, car\_obj.year)

**# Question 12: Inheritance**

**# a,b. Define an Employee class inheriting from Person with salary attribute,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

employee\_obj = Employee("Alice", 30, 50000)

print("Employee attributes:")

print("Name:", employee\_obj.name)

print("Age:", employee\_obj.age)

print("Salary:", employee\_obj.salary)

manager\_obj = Manager("Bob", 35, 80000, "HR")

print("\nManager attributes:")

print("Name:", manager\_obj.name)

print("Age:", manager\_obj.age)

print("Salary:", manager\_obj.salary)

print("Department:", manager\_obj.department)

**# Question 13: Static Methods**

**# a. Add a static method to Person class to return a greeting message**

class Person:

@staticmethod

def greet():

return "Hello, there!"

print("Static method greeting:", Person.greet())

**# b. Add a static method to MathUtils class to return the square of a number**

**class MathUtils:**

@staticmethod

def square(num):

return num \*\* 2

print("Square of 5 using static method:", MathUtils.square(5))

**# Question 14: Class Methods**

**# a. Add a class method to Person class to return a Person instance with default values**

class Person:

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

self.name = name

self.age = age

@classmethod

def default(cls):

return cls("Unknown", 0) # Default values for name and age

default\_person = Person.default()

print("Default Person:")

print("Name:", default\_person.name)

print("Age:", default\_person.age)

**#b Add a class method to the Book class that creates a new Book instance from a given title**

class Book:

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

self.title = title

self.author = author

@classmethod

def create(cls, title, author):

return cls(title, author) # Creating a new instance of Book with given title and author

# Example usage:

new\_book = Book.create("Python Programming", "Gud")

print("New Book:")

print("Title:", new\_book.title)

print("Author:", new\_book.author)

**# Question 15: Magic Methods**

**# a. Override \_\_str\_\_ method in 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\_obj = Person("David", 40)

print(person\_obj) # Output: Person: David, Age: 40

**# b. Override \_\_add\_\_ method in 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)

vec1 = Vector(1, 2)

vec2 = Vector(3, 4)

result\_vec = vec1 + vec2

print("Vector addition result:", result\_vec.x, result\_vec.y)

**# Question 16: Context Managers**

**# a. Create a context manager to print "Entering" and "Exiting"**

**from contextlib import contextmanager**

@contextmanager

def custom\_context():

print("Entering")

yield

print("Exiting")

with custom\_context():

print("Inside context")

**# b. Create a context manager to temporarily change current directory**

import os

@contextmanager

def change\_directory(directory):

current\_dir = os.getcwd()

try:

os.chdir(directory)

yield

finally:

os.chdir(current\_dir)

# Example usage:

with change\_directory("/tmp"):

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

**# Question 17: Iterators**

**# a. Create a custom iterator to return numbers from 1 to 5**

class CountIterator:

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

self.count = 0

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

return self

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

if self.count >= 5:

raise StopIteration

self.count += 1

return self.count

count\_iter = CountIterator()

print("Custom iterator numbers:", list(count\_iter))

**# b. Create a custom iterator to return characters from a string one by one**

class CharIterator:

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

result = self.string[self.index]

self.index += 1

return result

char\_iter = CharIterator("Hello")

print("Custom iterator characters:", list(char\_iter))

**# Question 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!")

# File: main.py

from my\_module import greet

# Calling the greet function from my\_module

greet()

# b

**# Question 19: Date and Time**

**# a. Get current date and time in "YYYY-MM-DD HH:MM" format**

**from datetime import datetime**

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

print("Current date and time:", current\_datetime)

**# b. Calculate days between two given dates**

**# Question 20: Collections**

**# a. Use Counter class to count characters in "abracadabra"**

**from collections import Counter**

char\_counts = Counter("abracadabra")

print("Character counts:", char\_counts)

**# b. Use defaultdict class to create dictionary with default value of 0 for new keys**

from collections import defaultdict

default\_dict = defaultdict(int)

print("Default dictionary example:", dict(default\_dict))