Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_07\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Design a class Song that internally stores all the metadata of a song.Implement an abstraction function in the Song class that returns a dictionary with only the song title, artist, and duration.Demonstrate the functionality by creating a few Song objects and displaying their simplified views. |
| 2 | Design a class Vehicle that internally stores all the specifications of a vehicle.Implement an abstraction function in the Vehicle class that returns a dictionary with only the vehicle's make, model, and year.Demonstrate the functionality by creating a few Vehicle objects and displaying their quick overviews. |
| 3 | Design a class Product that internally stores all the attributes of a product.Implement an abstraction function in the Product class that returns a dictionary with only the product name, price, and image.Demonstrate the functionality by creating a few Product objects and displaying their listings. |
| 4 | Design a class Book that stores the ID, title, author, and number of copies available.Implement rep invariants to ensure that:The book ID is unique.The number of copies available is never negative.Demonstrate the functionality by creating a few Book objects and trying to set negative copies or duplicate IDs. |
| 5 | Design a class Student that stores the student number, name, and a list of enrolled courses.Implement rep invariants to ensure that:The student number is unique.A student is not enrolled in the same course more than once.  Demonstrate the functionality by creating a few Student objects and trying to enroll them in the same course multiple times or assign duplicate student numbers. |

Submitted On

8-11-2023

(Date: DD/MM/YY)

**Task 1:** Create a program that recursively searches for files with a specific extension in a given directory.

**Solution:**

import os

def search\_files(directory, extension):

matching\_files = []

for item in os.listdir(directory):

item\_path = os.path.join(directory, item)

if os.path.isfile(item\_path) and item.endswith(extension):

matching\_files.append(item\_path)

elif os.path.isdir(item\_path):

matching\_files.extend(search\_files(item\_path, extension))

return matching\_files

directory\_path = "E:"

file\_extension = ".txt"

found\_files = search\_files(directory\_path, file\_extension)

if found\_files:

print("Found matching files:")

for file\_path in found\_files:

print(file\_path)else:

print("No matching files found.")

**Task 2:** Design a class Vehicle that internally stores all the specifications of a vehicle.

Implement an abstraction function in the Vehicle class that returns a dictionary with only the vehicle's make, model, and year.

Demonstrate the functionality by creating a few Vehicle objects and displaying their quick overviews.

**Solution:**

class Vehicle:

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

self.make = make

self.model = model

self.year = year

self.color = color

def get\_specifications(self):

return {

"Make": self.make,

"Model": self.model,

"Year": self.year,

"Color": self.color}

def get\_quick\_overview(self):

return {

"Make": self.make,

"Model": self.model,

"Year": self.year}

vehicle1 = Vehicle("Toyota", "Camry", 2022, "Silver")

vehicle2 = Vehicle("Ford", "Mustang", 2020, "Red")

vehicle3 = Vehicle("Honda", "Civic", 2021, "Blue")

print("Vehicle 1 Overview:", vehicle1.get\_quick\_overview())

print("Vehicle 2 Overview:", vehicle2.get\_quick\_overview())

print("Vehicle 3 Overview:", vehicle3.get\_quick\_overview())

**Output:**

A screenshot of a computer program

Description automatically generated

**Task 3:** Design a class Product that internally stores all the attributes of a product.Implement an abstraction function in the Product class that returns a dictionary with only the product name, price, and image.Demonstrate the functionality by creating a few Product objects and displaying their listings.

**Solution:**

class Product:

def \_\_init\_\_(self, name, price, image\_url, description):

self.name = name

self.price = price

self.image\_url = image\_url

self.description = description

def get\_attributes(self):

return {

"Name": self.name,

"Price": self.price,

"Image": self.image\_url,

"Description": self.description}

def get\_listing(self):

return {

"Name": self.name,

"Price": self.price,

"Image": self.image\_url}

product1 = Product("Laptop", 1200.0, "laptop\_image.jpg", "Powerful laptop for all your computing needs")

product2 = Product("Smartphone", 599.99, "phone\_image.jpg", "Feature-packed smartphone with a high-quality camera")

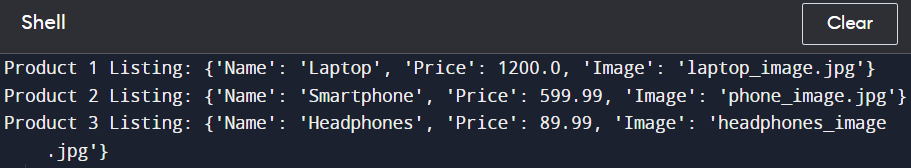
product3 = Product("Headphones", 89.99, "headphones\_image.jpg", "Wireless headphones with noise-canceling technology")

print("Product 1 Listing:", product1.get\_listing())

print("Product 2 Listing:", product2.get\_listing())

print("Product 3 Listing:", product3.get\_listing())

**Output:**



**Task 4:** Design a class Book that stores the ID, title, author, and number of copies available.Implement rep invariants to ensure that:The book ID is unique.The number of copies available is never negative.Demonstrate the functionality by creating a few Book objects and trying to set negative copies or duplicate IDs.

**Solution:**

class Book:

def \_\_init\_\_(self, book\_id, title, author, copies\_available):

self.book\_id = book\_id

self.title = title

self.author = author

self.copies\_available = copies\_available

assert self.is\_unique\_id(), "Book ID must be unique."

assert self.copies\_available >= 0, "Number of copies available cannot be negative."

def is\_unique\_id(self):

for book in Book.instances:

if book.book\_id == self.book\_id:

return False

return True

def set\_copies\_available(self, new\_copies):

assert new\_copies >= 0, "Number of copies available cannot be negative."

self.copies\_available = new\_copies

Book.instances = []

book1 = Book(1, "Introduction to Python", "John Smith", 10)

book2 = Book(2, "Data Structures in Java", "Jane Doe", 5)

book3 = Book(3, "Machine Learning Basics", "Alice Johnson", 15)

print("Book 1:", book1.\_\_dict\_\_)

print("Book 2:", book2.\_\_dict\_\_)

print("Book 3:", book3.\_\_dict\_\_)

try:

book1.set\_copies\_available(-5)

except AssertionError as e:

print("Error:", e)

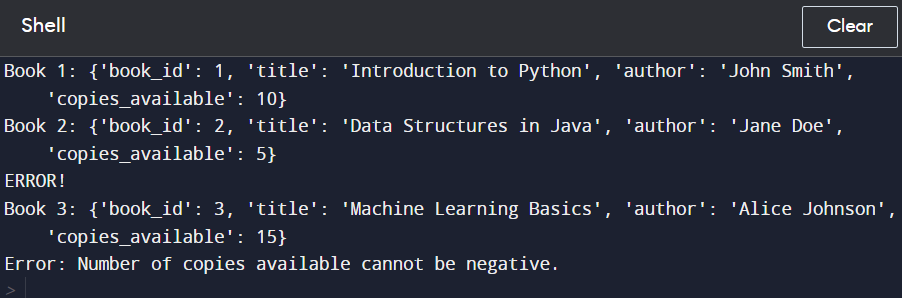
try:

duplicate\_book = Book(1, "Duplicate Book", "Duplicate Author", 8)

except AssertionError as e:

print("Error:", e)

**Output:**



**Task 5:** Design a class Student that stores the student number, name, and a list of enrolled courses.Implement rep invariants to ensure that:The student number is unique.A student is not enrolled in the same course more than once.Demonstrate the functionality by creating a few Student objects and trying to enroll them in the same course multiple times or assign duplicate student numbers.

**Solution:**

class Student:

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

self.student\_number = student\_number

self.name = name

self.enrolled\_courses = []

def is\_unique\_student\_number(self):

for student in Student.instances:

if student.student\_number == self.student\_number:

return False

return True

def enroll\_in\_course(self, course):

self.enrolled\_courses.append(course)

Student.instances = [] # List to store instances of Student for checking uniqueness

student1 = Student(101, "John Doe")

student2 = Student(102, "Jane Smith")

student3 = Student(103, "Alice Johnson")

print("Student 1:", student1.\_\_dict\_\_)

print("Student 2:", student2.\_\_dict\_\_)

print("Student 3:", student3.\_\_dict\_\_)

student1.enroll\_in\_course("Mathematics")

student1.enroll\_in\_course("Physics")

student2.enroll\_in\_course("Computer Science")

student3.enroll\_in\_course("Chemistry")

try:

student2.enroll\_in\_course("Mathematics")

except AssertionError as e:

print("Error:", e)

try:

duplicate\_student = Student(101, "Duplicate Student")

except AssertionError as e:

print("Error:", e)

**Output:**

