# PROG8421 - Programming for Big Data

**Assignment 8&9**

**Group 1 Members:**

|  |  |
| --- | --- |
| **Name: Sudhan Shrestha** | **Name: Jose Encarnacion** |
| **Student ID: 8889436** | **Student ID: 8982860** |

**Question-1. Create an object-oriented program that allows you to enter data for customers and employees.**

**A close-up of a contact list

Description automatically generated**

**Specifications**

* **Create a Person class that provides attributes for first name, last name, and email address. This class should provide a property or method that returns the person’s full name.**
* **Create a Customer class that inherits the Person class. This class should add an attribute for a customer number.**
* **Create an Employee class that inherits the Person class. This class should add an attribute for a social security number (SSN).**
* **The program should create a Customer or Employee object from the data entered by the user, and it should use this object to display the data to the user. To do that, the program can use the isinstance() function to check whether an object is a Customer or Employee object.**

**Code**

from typing import \*

# Question-1. Create an object-oriented program that allows you to enter data for customers and  employees.

# Create a Person class that provides attributes for first name, last name, and email address.

# This class  should provide a property or method that returns the person’s full name.

class Person:

    """

    Person Class: This base class has attributes for first name, last name, and email.

    It also includes a property full\_name that returns the person's full name.

    """

    def \_\_init\_\_(self, first\_name: str, last\_name: str, email: str) -> None:

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.email = email

    @property

    def full\_name(self) -> str:

        return f"{self.first\_name} {self.last\_name}"

# Create a Customer class that inherits the Person class. This class should add an attribute for a  customer number.

class Customer(Person):

    """

    Customer Class: Inherits from Person and adds a customer\_number attribute.

    """

    def \_\_init\_\_(self, first\_name: str, last\_name: str, email: str, customer\_number: int) -> None:

        super().\_\_init\_\_(first\_name, last\_name, email)

        self.customer\_number = customer\_number

#Create an Employee class that inherits the Person class. This class should add an attribute for a social  security number (SSN).

class Employee(Person):

    """

    Employee Class: Inherits from Person and adds an ssn attribute.

    """

    def \_\_init\_\_(self, first\_name: str, last\_name: str, email: str, ssn: str) -> None:

        super().\_\_init\_\_(first\_name, last\_name, email)

        self.ssn = ssn

def main():

    print()

    print("Customer/Employee Data Entry")

    while True:

        role = input("Customer or employee? (c/e): ").strip().lower()

        if role not in ['c', 'e']:

            print("Invalid input. Please enter 'c' for customer or 'e' for employee.")

            continue

        print()

        print("DATA ENTRY")

        first\_name = input("First name: ").strip()

        last\_name = input("Last name: ").strip()

        email = input("Email: ").strip()

        if role == 'c':

            customer\_number = input("Number: ").strip()

            person = Customer(first\_name, last\_name, email, customer\_number)

            print()

            print("CUSTOMER")

        else:

            ssn = input("SSN: ").strip()

            person = Employee(first\_name, last\_name, email, ssn)

            print()

            print("EMPLOYEE")

        print(f"Name: {person.full\_name}")

        print(f"Email: {person.email}")

        if isinstance(person, Customer):

            print(f"Number: {person.customer\_number}")

        else:

            print(f"SSN: {person.ssn}")

        while True:

            print()

            continue\_entry = input("Continue? (y/n): ").strip().lower()

            if continue\_entry == 'y':

                break

            elif continue\_entry == 'n':

                print("Bye!")

                exit()

            else:

                print("Invalid input. Please enter 'y' or 'n'.")

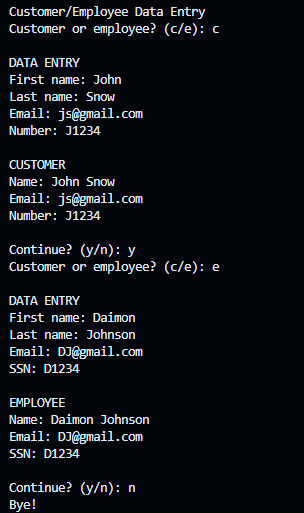
if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Comment**

The class Person is a base class that stores the persons information like first\_name, last\_name and email, here a property full\_name exits that restores the persons full name, Customer class exits which inherits the Person class and has an additional attributes customer\_number. Employee class inherits from the Person class as well and has an additional attribute social security number as ssn. The main function handles the major of the validation and logics from input, choosing the roles for the person as well as displaying the output.

**Screenshots**

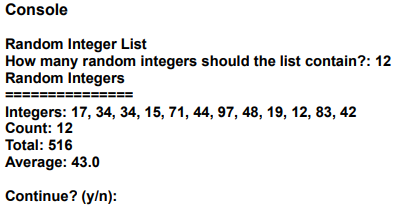
****

**Error Handling**

**A screenshot of a computer

Description automatically generated**

**Question-2. Create an object-oriented program that uses a custom list object to automatically generate and work with a series of random integers.**

****

**Specifications**

* **Create a RandomIntList class that inherits the list class. This class should allow a programmer to create a list of random integers from 1 to 100 by writing a single line of code. For example, a programmer should be able to create a custom list that stores 12 random integers with this line of code:**
  + **int\_list = RandomIntList(12)**
* **To do that, you can use the self-keyword to access the list superclass like this:** 
  + **self.append(rand\_int)**
* **The RandomIntList class should contain methods or properties for getting the count, average, and total of the numbers in the list. In addition, it should contain a \_\_str\_\_method for displaying a commaseparated list of integers as shown above.**
* **The program should use the RandomIntList class to generate the list of random integers, display the list, and get the summary data (count, total, and average).**
* **The program should make sure the integer entered by the user is valid.**

**Code**

from typing import \*

import random

#  Question-2. Create an object-oriented program that uses a custom list object to automatically

# generate and work with a series of random integers.

# • Create a RandomIntList class that inherits the list class.

# This class should allow a programmer to create  a list of random integers from 1 to 100 by writing a single line of code.

# For example, a programmer  should be able to create a custom list that stores 12 random integers with

# this line of code: int\_list = RandomIntList(12)

class RandomIntList(list):

    """

    Class RandomIntList Inherits from list class

    """

    def \_\_init\_\_(self, count) -> None:

        super().\_\_init\_\_()

        for \_ in range(count):

            self.append(random.randint(1, 100))

    @property

    def count(self):

        return len(self)

    @property

    def total(self):

        return sum(self)

    @property

    def average(self):

        return self.total / self.count if self.count > 0 else 0

    def \_\_str\_\_(self):

        return ', '.join(map(str, self))

def main():

    print()

    print("Random Integer List")

    while True:

        try:

            num\_integers = int(input("How many random integers should the list contain?: "))

            if num\_integers <= 0:

                raise ValueError("Number of integers must be positive.")

        except ValueError as e:

            print(f"Invalid input: {e}")

            continue

        int\_list = RandomIntList(num\_integers)

        print("\nRandom Integers")

        print("===============")

        print(f"Integers: {int\_list}")

        print(f"Count: {int\_list.count}")

        print(f"Total: {int\_list.total}")

        print(f"Average: {int\_list.average:.1f}")

        while True:

            cont = input("Continue? (y/n): ").strip().lower()

            if cont == 'y':

                break

            elif cont == 'n':

                exit()

            else:

                print("Invalid input. Please enter 'y' or 'n'.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Comment**

The RandomIntList class is inherited from the list, which is an inbuilt class from python class, where we initialize a list of random integers between 1 and 100 based on the count provided, we also define three property count, total and average which does calculation as their names suggest. In the main function we validate user input and display user output, this also handles the creation of random list values using the RandomIntList.

**Screenshots**

**A computer screen shot of a black and white screen

Description automatically generated**

**Error handling:**

**A computer screen shot of a black background with white text

Description automatically generated**

**Question-3. Enhance the Product Viewer program so it provides one more type of product: a music album. When you enter the product number for a music album, it should print the data to the console like this:**

**A white background with black text

Description automatically generated**

**Open and test the program**

1. **In PyCharm, open the objects.py and product\_viewer.py files provided to you in eConestoga.**
2. **Review the code and run the program to make sure it works correctly. Note that the Movie class displays the format, which is DVD, as part of the name of the product. Improve the Movie and Book classes**
3. **In the Movie class, add an attribute named format that stores the format of the product. For example, the format could be DVD, streaming, and so on.**
4. **In the product\_viewer module, modify the code that creates the Movie object so it stores “DVD” as the format attribute instead of appending this data to the end of the name attribute. Then, modify the code that displays the Movie object so it displays the format on a separate line, after the year of the movie.**
5. **Repeat steps 3 and 4 for the Book class. You can use “Hardcover” as the format for the book. Or, if you prefer, you can specify a different type of book format such as “Paperback” or “ebook”. Add an Album class**
6. **In the objects module, add a class named Album that inherits the Product class. The Album class should add two attributes: one for storing the artist and another for storing the format.**
7. **In the product\_viewer.py file, modify the code that creates the objects so it includes a fourth object, an Album object. This object should contain the data for a music album that you like. Then, add code that displays the Album object as shown at the beginning of this exercise. Add a Media class**
8. **In the objects module, add a class named Media that inherits the Product class. This class should add a format attribute to the Product class.**
9. **Modify the Movie, Book, and Album classes so they inherit the Media class, not the Product class. This should create a class hierarchy that looks like this: Product**

**Media**

**Movie**

**Book**

**Album**

1. **In the product\_viewer.py file, modify the code that displays a product so it only displays the format attribute for Media objects. Note how this reduces code duplication.**

**Code**

***objects.py***

from dataclasses import dataclass

@dataclass

class Product:

    """

    Product is a base class with three attributes:

    name: a string representing the product's name.

    price: a float representing the product's price.

    discountPercent: an integer representing the discount percentage.

    """

    name:str = ""

    price:float = 0.0

    discountPercent:int = 0

    def getDiscountAmount(self):

        """Calculates the discount amount based on the price and discount percent."""

        return self.price \* self.discountPercent / 100

    def getDiscountPrice(self):

        """Returns the price after applying the discount."""

        return self.price - self.getDiscountAmount()

    def getDescription(self):

        """Returns the product's name."""

        return self.name

@dataclass

class Media(Product):

     format:str =""

@dataclass

class Book(Media):

    """

    Book is a subclass of Product with an additional attribute:

    author: a string representing the book's author.

    """

    author:str = ""

    def getDescription(self):

        """Returns a string that includes the book's name and author."""

        return f"{Product.getDescription(self)} by {self.author}"

@dataclass

class Movie(Media):

    """

    Movie is a subclass of Product with an additional attribute:

    year: an integer representing the movie's release year.

    movieFormat: an string representing the format of the movie

    """

    year:int = 0

    def getDescription(self):

        """Returns a string that includes the movie's name and release year."""

        return f"{Product.getDescription(self)} ({self.year})"

@dataclass

class Album(Media):

    """

    Album is a subclass of Product with an additional attribute:

    artist: an string representing the artist.

    albumFormat: an string representing the album format

    """

    artist:str =""

    def getDescription(self):

        """Returns a string that includes the Album's artist and format."""

        return f"{Product.getDescription(self)} ({self.artist})"

***q3.py***

from typing import \*

from objects import Product, Book, Movie, Album, Media

import os

# Question-3. Enhance the Product Viewer program so it provides one more type of product:

# a music  album. When you enter the product number for a music album,

# it should print the data to the console like  this:

# Enter product number: 4

# PRODUCT DATA

# Name: Rubber Soul

# Artist: The Beatles

# Format: CD

# Discount price: 10.00

def show\_products(products):

    print("PRODUCTS")

    for i, product in enumerate(products, start=1):

        print(f"{i}. {product.getDescription()}")

    print()

def show\_product(product):

    w=18

    print("PRODUCT DATA")

    print(f"{'Name:':{w}}{product.name}")

    if isinstance(product, Book):

        print(f"{'Author:':{w}}{product.author}")

    if isinstance(product, Movie):

        print(f"{'Year:':{w}}{product.year}")

    if isinstance(product, Album):

        print(f"{'Author:':{w}}{product.artist}")

    if isinstance(product, Media):

        print(f"{'Format:':{w}}{product.format}")

    print(f"{'Discount price:':{w}}{product.getDiscountPrice():.2f}")

    print()

def main():

    print()

    print("The Product Viewer program")

    print()

    products = (Product("Stanley 13 Ounce Wood Hammer", 12.99, 62),

                Book(name="The Big Short",price=15.95, discountPercent=34, author="Michael Lewis",format="ebook"),

                Movie(name="The Holy Grail", price=14.99, discountPercent=68, year=1975, format="DVD"),

                Album(name="Flesh Of My Fleshblood Of My Blood", artist="DMX", price=15.99, discountPercent=18, format="DVD"),

                Album(name="Grateful", artist="DJ Khaled", price=15.99, discountPercent=0, format="DVD"),

                Album(name="Strange Clouds ", artist="B.o.B", price=15.99, discountPercent=18,  format="Streaming"),

                Album(name="COMMON - ELECTRIC CIRCUS ", artist="COMMON", price=5.99, discountPercent=1,  format="DVD"),)

    show\_products(products)

    choice = "y"

    while choice.lower() == "y":

        try:

            number = int(input("Enter product number: "))

            if number < 1 or number > len(products):

                raise ValueError("Product number out of range.")

            print()

            product = products[number-1]

            show\_product(product)

        except ValueError as ve:

            print(f"Invalid input: {ve}. Please enter a valid product number.")

            continue

        while True:

            choice = input("View another product? (y/n): ").strip().lower()

            if choice in ('y', 'n'):

                break

            else:

                print("Invalid input. Please enter 'y' for yes or 'n' for no.")

        print()

    print("Bye!")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Comment**

This program has two files, we defined them as objects.py and q4.py referring to question 4. The objects.py file holds our classes, Product, Media, Book, Movie and Album class. The Product class is inherited by Media Class and this Media class is then inherited by the rest of the other class Book, Movie and Album. The product class as addition functions getDiscountAmmount, getDiscountPrice and getDescription function, this function performs tasks as described in their name.

The q3.py file consists of a processing part of the program, where show\_products display the product information whereas the main function consists of the input information with logic for validation and displaying the output.

**Screenshot**

**A screen shot of a computer

Description automatically generated**

**Error Handling**

**A screenshot of a computer program

Description automatically generated**

**Question-4. Work with a Book object that uses an Authors object to store one or more Author objects.**

**Open and test the program**

1. **In PyCharm, open the objects.py and authors\_tester.py files provided in eConestoga**
2. **Review the code and note how the Book object uses an Authors object to store one or more Authors.**
3. **Run the code to see how it works. At this point, it doesn’t display the book or author information correctly, but you’ll fix that later in this exercise. Improve the Author, Authors, and Book classes**
4. **In the Author class, add a \_\_str\_\_() method that returns the first and last name of the author, separated by a space.**
5. **In the Authors class, add a \_\_str\_\_() method that returns the name of each author, separating multiple authors with a comma and a space.**
6. **In the Book class, add a \_\_str\_\_() method that returns the title of the book, followed by the word “by” and one or more authors, with each author separated by a comma.**
7. **Run the authors\_tester module again. This time, it should display the correct data for the book and author information. Make sure the program works correctly for a single author**
8. **In the authors\_tester module, comment out the statement that adds the second author. Then, run the module again. This should work, but the last line says “Authors” where it should say “Author”.**
9. **Modify the code so it uses the count property of the Authors object to display the correct label for the author or authors depending on the number of authors for the book. Define and use an iterator for the Authors object**
10. **In the Authors class, add the \_\_iter\_\_() method that makes it possible to use a for statement to loop through all Author objects stored in the Authors object.**
11. **In the authors\_tester module, add a for statement that loops through each author in the Authors object and prints each author to the console. When you’re done, running this module should display this data to the console:**

**A white background with black text

Description automatically generated**

**Code**

***AuthorObjectsNew.py***

from dataclasses import dataclass, field

from typing import List, Iterator

@dataclass

class Author:

    firstName: str = ""

    lastName: str = ""

    def \_\_str\_\_(self) -> str:

        return f"{self.firstName} {self.lastName}"

class Authors:

    def \_\_init\_\_(self):

        self.\_list: List[Author] = []

    def add(self, author: Author):

        self.\_list.append(author)

    @property

    def count(self) -> int:

        return len(self.\_list)

    def \_\_str\_\_(self) -> str:

        return ', '.join(str(author) for author in self.\_list)

    def \_\_iter\_\_(self) -> Iterator[Author]:

        return iter(self.\_list)

@dataclass

class Book:

    title: str = ""

    authors: Authors = field(default\_factory=Authors)

    def \_\_str\_\_(self) -> str:

        return f"{self.title} by {self.authors}"

***q4.py***

from AuthorObjectsNew import Book, Author, Authors

import os

def main():

    print()

    print("The Authors Tester program")

    print()

    author1 = Author("Mark", "Twain")

    author2 = Author("Charles", "Warner")

    authors = Authors()

    authors.add(author1)

    authors.add(author2)

    book = Book("The Gilded Age", authors)

    # display the book data

    print("BOOK DATA - SINGLE LINE")

    print(book)

    print()

    print("BOOK DATA - MULTIPLE LINES")

    print("Title:   ", book.title)

    print("Author:  " if authors.count == 1 else "Authors: ", book.authors)

    print("\nAUTHORS")

    for author in authors:

        print(author)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Comment**

The program also consists of two files AuthorObjectsNew.py and q5.py similarly. The AuthorObjectsNew.py consists of all the namely namely Author and Book which uses @dataclass decorator. Autor class has two attributes firstName and lastName and a \_\_str\_\_ method which returns this name as a full name this class also has \_\_iter\_\_ method to allow iteration overt the list of authors, while Book class has title and authors attribute and returns these attributes in the format “title by authors”.

The q4.py file consists of a main function which takes the data as input, uses the classes we created to create an instance of it and displays the output in the needed formats that is the book data in both single line and multiple formats, here we also iterate over authors to pint the authors’ name.

**Screenshot**

**A screenshot of a computer

Description automatically generated**