UNIVERSITY OF BIRMINGHAM DUBAI

LM Software Workshop 1 (34153, 34182, 34168, 36990)
Lab Exercise Sheet
Week 8 Introduction to Python Classes and Objects

Question 1.a.

Create a class named as Airplane with the following attributes and methods

a) Attributes: Fuel, seatingcapacity, company, engineType
b) Methods: boarding() takes a number as a number of passengers, fueling() takes a number as a number for number of litters of fuel
c) Create objects for more than one companies and call their corresponding methods for different scenarios

Question 1.b. (book project 9.1)

Add three methods to the **Student** class that compare two **Student** objects. One method should test for equality. A second method should test for less than. The third method should test for greater than or equal to. In each case, the method returns the result of the comparison of the two students' names. Include a **main** function that tests all of the comparison operators.

You can use the following in-built methods:

__lt__: Less than

__le__: Less than or equal to

__eq__: Equal to

__ne__: Not equal to

__gt__: Greater than

__ge__: Greater than or equal to

```
"""
File: student.py
Resources to manage a student's name and test scores.
"""
class Student(object):
    """Represents a student."""

def  __init__(self, name, number):
     """Constructor creates a Student with the given
     name and number of scores and sets all scores
     to 0."""
     self.name = name
     self.scores = []
     for count in range(number):
         self.scores.append(0)

def getName(self):
    """Returns the student's name."""
    return self.name

def setScore(self, i, score):
    """Resets the ith score, counting from 1."""
    self.scores[i - 1] = score

def getScore(self, i):
    """Returns the ith score, counting from 1."""
    return self.scores[i - 1]
```

```
def getAverage(self):
    """Returns the average score."""
    return sum(self.scores) / len(self.scores)

def getHighScore(self):
    """Returns the highest score."""
    return max(self.scores)

def __str__(self):
    """Returns the string representation of the
    student."""
    return "Name: " + self.name + "\nScores: " + \
           " ".join(map(str, self.scores))
```

Question 2. (book project 9.2)

This project assumes that you have completed Project 1. Place several **Student** objects into a list and shuffle it. Then run the **sort** method with this list and display all of the students' information.

Use `random.shuffle()and lyst.sort() in-built methods`

Question 3 [Challenge Question] (book project 9.5)

The **Doctor** program described in Chapter 5 combines the data model of a doctor and the operations for handling user interaction. Restructure this program according to the model/view pattern so that these areas of responsibility are assigned to separate sets of classes. The program should include a **Doctor** class with an interface that allows one to obtain a greeting, a signoff message, and a reply to a patient's string. The

rest of the program, in a separate main program module, handles the user's interactions with the **Doctor** object. You may develop either a terminal-based user interface or a GUI. The chapter 5 program is as follows:

```
"""
File: doctor.py
Project 5.10
Conducts an interactive session of nondirective
psychotherapy.
Adds a history list of earlier patient sentences,
which can
be chosen for replies to shift the conversation to
an earlier topic.
"""
import random
history = []
hedges = ("Please tell me more.",
          "Many of my patients tell me the same
thing.",
          "Please coninue.")

qualifiers = ("Why do you say that ",
              "You seem to think that ",
              "Can you explain why ")

replacements = {"I":"you", "me":"you",
"my":"your",
                "we":"you", "us":"you",
"mine":"yours",
                "you":"I", "your":"my",
"yours":"mine"}

def reply(sentence):
    """Implements three different reply
strategies."""
    probability = random.randint(1, 5)
    if probability in (1, 2):
        # Just hedge
        answer = random.choice(hedges)
    elif probability == 3 and len(history) > 3:
```

```
            # Go back to an earlier topic
            answer = "Earlier you said that " + \

changePerson(random.choice(history))
        else:
            # Transform the current input
            answer = random.choice(qualifiers) +
changePerson(sentence)
        # Always add the current sentence to the
history list
        history.append(sentence)
        return answer

def changePerson(sentence):
        """Replaces first person pronouns with second
person
        pronouns."""
        words = sentence.split()
        replyWords = []
        for word in words:
            replyWords.append(replacements.get(word,
word))
        return " ".join(replyWords)

def main():
        """Handles the interaction between patient and
doctor."""
        print("Good morning, I hope you are well
today.")
        print("What can I do for you?")
        while True:
            sentence = input("\n>> ")
            if sentence.upper() == "QUIT":
                print("Have a nice day!")
                break
            print(reply(sentence))

# The entry point for program execution
if __name__ == "__main__":
        main()
```

TASKS from the CANVAS lab practice part section

Task-1

You have been given the below code that throw some errors. You need to identify these errors and fix the code

```
class Pizza:
        def __int__(crust_type, toppings, sauce, diameter):
                self.crust_type = crust_type
                self.toppings = toppings
                sauce = self.sauce
                self.diameter = diameter

class Patron:
    def __init_(self, name, age, email, favorite_pizza):
        self.name = name
        age = self.age
        self.email = email
        self.favorite_pizza = favorite_pizza
Pie:
    _init_(self, type, cost, is_vegan):
        type = self.type
        self.cost = cost
        self.is_vegan = is_vegan
```

Task-2

Building a Movie Class with Unique IDs

Objective: Understand the implementation of class and instance attributes, and how to manage unique identifiers for each class instance in Python.

Task Description:
Define a class named **Movie**.
Within the class, create a class attribute named **id_counter** initialized to 0. This attribute will track the next available ID for new movie instances.
Define the _ ***init*** _ method to accept **title** and **rating** as parameters. Additionally, the _ ***init*** _ method should automatically set an **id** instance attribute by incrementing **Movie.id_counter** and assign this new value to the **id**.

Ensure that each new **Movie** instance has a unique **id** by incrementing **Movie.id_counter** each time a new instance is created.
Inside the **Movie** class, define a method info that prints out the movie's ID, title, and rating in a readable format.
Test your class:
Create an instance of the **Movie** class with the title "Inception" and a rating of 8.8. Create another instance of the **Movie** class with the title "The Matrix" and a rating of 8.7. Call the **info** method on both instances to display their information.

Expected Output: When calling the **info** method on your **Movie** instances, the output should display the ID, title, and rating of each movie, like so:

ID: 1, Title: Inception, Rating: 8.8ID: 2, Title: The Matrix, Rating: 8.7

Guidelines:
Remember that the **id_counter** should be incremented only within the _ *init* _ method. Do not forget to use the **self** keyword when working with instance attributes. Use the **@classmethod** decorator if you decide to create a class method to increment the **id_counter**. Provide comments in your code to explain the purpose of each method and attribute.

Take home question

Task-3

You have been given the uncompleted code below for an HR system that includes payroll functionality. Your task is to complete the classes by including the necessary class attributes. First, you need to analyze the **process_payroll** function located at the bottom of the code below. From that code, you need to determine the attributes that each class must contain. After a bit of thinking, you have determined that these attributes are shared by all instances of the classes, hence they should be class attribute. Please assign realistic values to these attributes.
Note: Within the **process_payroll** function, class attributes are accessed using the syntax ..

```
# Classes ========================================================================

class HRManager:

    # Add the class attributes

    def __init__(self, name, age, department, phone):
        self.name = name
        self.age = age
        self.department = department
        self.phone = phone

class Employee:

    # Add the class attributes

    def __init__(self, name, age, address, phone, department):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone
        self.department = department
```

```
# Program ========================================================================

# Function that prints the monthly salary of each employee
# and the total payroll expense for the company
def process_payroll(employees):

    total_payroll_expense = 0
    print("\n========= Welcome to the HR Payroll System =========\n")

    # Iterate over the list of instances to calculate
    # and display the monthly salary for each employee,
    # and add the total payroll expense
    for emp in employees:
        monthly_salary = emp.annual_salary / 12
        print(f"{emp.name.capitalize()}'s monthly salary is: ${monthly_salary:.2f}")
        total_payroll_expense += monthly_salary

    # Print the total payroll expense for the month
    print("\nThe total payroll expense for the month is: $", total_payroll_expense)

# Instances (employees)
jack = HRManager("jack", 50, "Human Resources", "555-321-4567")
isabel = Employee("isabel", 28, "123 Maple St", "555-654-1234", "Marketing")

# List of employee instances
employees = [jack, isabel]

# Call the function, passing in the list of employee instances
process_payroll(employees)
```