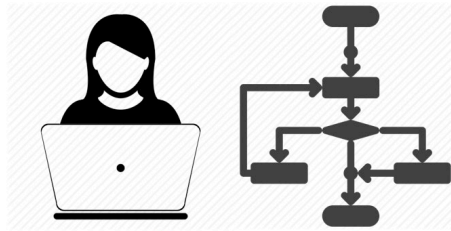


SWE 321

Object Oriented Programming – Lab



جامعة زايد
ZAYED UNIVERSITY

Revision History

Version	Semester	Author(s)	Reviewed by	Comments
1.00	Spring 2018	Abdallah T., Andrew L., Eleana K., Emad B., and Sujith M.	Zakaria M., and John G.	Course is created to introduce CTI students to the basics of object-oriented design and programming with Python programming language.
2.00	Fall 2018	Afshan P.	Sujith M.	Updated exercises to emphasize basic concepts taught in SWE 225 and include Appendix section to introduce text file handling. Realigned and added exercises to match course delivery.
3.0	Spring 2020	Afshan P. Tarannum P.	Sujith M.	Updated labs with new exercises. Also, changed the labs to suit the restructuring of SWE 320 and included exercises for chapter on GUI.
4.0	Fall 2021	Afshan P. Tarannum P.	Sujith M.	Updated the labs with new exercises. Included sample exam questions and homework for students to review.

Table of Contents

1	Introduction	1
1.1	Grading Scheme.....	1
1.2	Submission Guidelines	1
1.2.1	Blackboard Portfolio.....	1
1.2.2	Submissions Deadlines.....	3
1.2.3	Code Documentation	4
1.2.4	Zayed University Honor Code.....	4
1.2.5	Plagiarism Policy	4
2	Lab 1 - Review of Basic Programming Structures	6
2.1	Assignment 1	6
2.2	Assignment 2.....	8
3	Lab 2 - UML Class Diagrams	10
3.1	Assignment 1	10
3.2	Assignment 2.....	11
4	Lab 3 - Introduction to Python Classes	12
4.1	Assignment 1	12
4.2	Assignment 2.....	13
5	Lab 4 - Inheritance.....	14
5.1	Assignment 1	14
5.2	Assignment 2.....	16
6	Lab 5 - Class Association.....	17
6.1	Assignment 1	17
6.2	Assignment 2.....	17
7	Lab 6 - Polymorphism	19
7.1	Assignment 1	19
8	Lab 7 - Exception Handling.....	21
8.1	Assignment 1	21
9	Lab 8 - Graphical User Interface (GUI)	23
9.1	Assignment 1	23
10	Appendix	25
10.1	Text File Reading and Writing.....	25
10.2	UML Tool	26
10.3	Sample Questions	28
10.4	Grading Rubric	32
10.5	Grading Key	33

1 Introduction

This lab manual guides CTI students to complete the weekly assignments of SWE-321 Object Oriented Programming (OOP) Lab course. These assignments provide practical exercises for the concepts and principles taught in SWE-320 OOP (co-requisite course). All assignments in this document should be completed in the sequence that it is presented in to ensure proper learning gradient.

1.1 Grading Scheme

The completion and submission of all assignments in this lab manual contributes to 70% of the total course grade. The remaining 30% is for the project component. The rubric to grade each assignment and the Grading Key are provided in the Appendix section of this document.

The lab assignments are graded by the instructor and the feedback is provided to the student via Blackboard. The onus is on the student to ensure that their individual lab portfolio is updated on a timely basis with the working solutions of the assignments.

1.2 Submission Guidelines

The solutions to all the assignments in this lab manual are submitted to a Blackboard portfolio that is created under each student's Blackboard profile. The portfolio should be organized in the same order of weekly assignments that is presented in this document.

The due dates for submissions are defined by the course instructor. Delayed submission could lead to reduction of scores as indicated in the rubric. Ensure to provide UML diagrams, code in Python, sufficient documentation, screenshots of output, and other content to help in the assessment of the submitted work.

1.2.1 Blackboard Portfolio

To create a Blackboard portfolio, follow these steps (see figures below).

1. Login to Blackboard
2. *Open Global Navigation Menu* by clicking on your name on the top-right corner.
3. Select *Tools* and then click on *Portfolios* (Tools->Portfolios)
4. Click on Create Portfolio

My Portfolios
Portfolios offer a means to demonstrate formative and/or summative progress and achievement. The My Portfolios Page contains additional portfolios, modify, share and delete existing portfolios.

Create Portfolio

Sort by:

Displaying 1 to

GENERAL INFORMATION

* Title: SWE 321 - OOP Lab

Description:

Character count: 0

Available: ☒

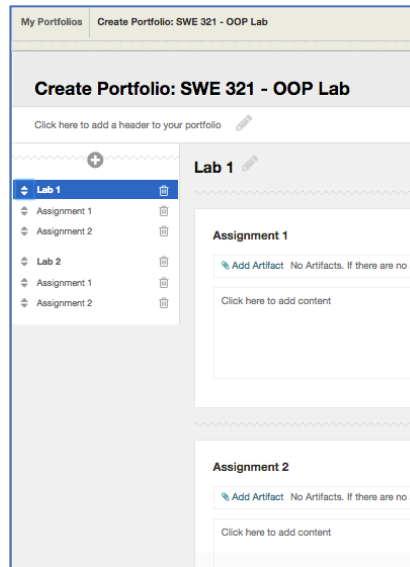
Comments are Private: ☐
If checked, all comments will be hidden from users who can view the Portfolio.

5. Under General Information

- Title:** SWE 321 – OOP Lab
- Check the *Available* option
- Uncheck the Comments are Private option

6. Organize the portfolio as shown below.

Follow the organization of this document's Table on Contents to organize the portfolio pages.



7. The portfolio is updated with the solutions to the required assignments.
8. Updating the portfolio **DOES NOT MEAN SUBMISSION**. Please ensure that your portfolio is submitted as advised by the instructor on a regular basis for assessment. The instructor will **NOT SEE and CANNOT EVALUATE** the portfolio unless it is submitted to the instructor.

1.2.2 Submissions Deadlines

This is a **suggestion for submission** deadlines that the lab instructor can use for Spring/Fall.

<ul style="list-style-type: none"> • Lab 1 at the end of Week 2 • Lab 2 at the end of Week 4 • Lab 3 at the end of Week 6 • Lab 4 at the end of Week 7 	<ul style="list-style-type: none"> • Lab 5 at the end of Week 9 • Lab 6 at the end of Week 10 • Lab 7 at the end of Week 11 • Lab 8 at the end of Week 12
--	---

Evaluation is completed by instructor at the end of each submission deadline. Each submission is a cumulative update of the previous submission. All assignments in lab manual would be the final submission. Homework questions are provided for students to practice the topics after class hours.

A separate submission of the **final project is for Weeks 13-15** which is not assessed as part of the weekly lab assignments.

1.2.3 Code Documentation

Python code that is submitted as solutions for the assignments must contain proper documentation. Suggested header documentations are provided for programs and functions. All programs and all functions must contain sufficient documentation to explain their intent and use.

- Program Header **Suggestion:**

```
# Program/Class Name:
# Author(s):
# Date of Creation:
```

- Class *doc_string* must be included for all class definitions.
- Function Header **Suggestion:**

```
# Function Description:
# Parameters:
# Return Type:
```

1.2.4 Zayed University Honor Code

“In the Name of God Most Gracious Most Merciful”

As a student of the University that carries the name of the beloved and revered father of the nation, the late Sheikh Zayed Bin Sultan Al Nahayan (may his soul rest in eternal peace),

I pledge to:

- Demonstrate the virtues of honesty, respect and fairness
- Adhere to the highest standards of personal moral conduct
- Refrain from any and all forms of academic dishonesty
- Present a positive image of myself by acting with maturity and honor
- Take responsibility for my actions and do my part to maintain a community of trust
- Dedicate myself to the achievement of the University’s excellence

I promise to honor Sheikh Zayed and to preserve his legacy by following the example set by the wise and beloved father of the United Arab Emirates

1.2.5 Plagiarism Policy

Plagiarism is presenting someone else’s work or ideas as if it were one’s own. Examples of plagiarism include the following:

- Copying another person's work either word for word or making some changes but keeping the structure, much of the language, and main ideas the same. Even if the work is not published, it should be treated as someone else's work and not one's own work.
- Buying, borrowing, or otherwise obtaining and handing in a paper, project or course assignment as if it were one's own.
- Turning in someone else's paper as if it were one's own is strictly prohibited, even if the paper is enclosed in quotation marks. A large part of a paper cannot simply be quotations.
- Allowing someone else to edit, rewrite or make substantial changes in one's work and turning it in as if one had done it all, without acknowledging the other person's contribution and without prior permission of the instructor.
- Using someone else's words or ideas without crediting that person.
 - If a student uses someone else's words, he must identify them by putting quotation marks around them and citing the source.
 - If a student downloads a picture from the Internet, he must cite the source of the picture.
 - If a student paraphrase someone's work, he must specify the source of the statement.
 - Every book, magazine, or internet site used in a paper must be identified in the bibliography.
- At any time, if a student thinks he may have unknowingly plagiarized someone's work, he should discuss it with his instructor before turning in the assignment.

2 Lab 1 – Review of Basic Programming Structures

The assignments for this week provide a review of the topics in SWE-225 *Introduction to Programming and Problem Solving* (pre-requisite course). The objective here, is to review basic programming constructs, like the use of primitive data types, operators, sequential logic, branching with selection, repetitions, the use of functions, and creating collection of data using lists.

2.1 Assignment 1

Write a Python program with the following **functions**. Solution to the first question is given as a sample.

- All input values are to be taken from the user.
 - Ensure to check for possible error cases like mismatch of data type and input out of range, for every function.
 - Include test cases for the functions with different parameters to ensure all working conditions.
1. **greatestOfFour (w, x, y, z)** - Function to find the greatest of four numbers w, x, y, and z, and return it. (*Answer to this is given below as an example*)

```
# Function Name: greatestOfFour
# Desc: Finds greatest of 4 numbers
# Parameters: num1 - int, num2 - int, num3 - int, num4 - int
# Return: int

def greatestOfFour(num1, num2, num3, num4):
    if num1>num2 and num1>num3 and num1>num4:
        return num1
    elif num2>num3 and num2 >num4:
        return num2
    elif num3>num4:
        return num3
    else:
        return num4

# Read Input
print("Enter four numbers to check the greatest: ")
num1 = input("1st num: ")
num2 = input("2nd num: ")
num3 = input("3rd num: ")
num4 = input("4th num: ")

# Call function for Process and Output
print("Greatest of 4 numbers: ")
print(greatestOfFour(int(num1), int(num2), int(num3), int(num4)))
```

Output:

```

/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6
Enter four numbers to check the greatest:
1st num: 13
2nd num: 45
3rd num: 234
4th num: 89
Greatest of 4 numbers:
234
Process finished with exit code 0

```

2. **reverseSentence(sentence)** - Function that reverses a given sentence given by user.
3. **listOdd(n1, n2)** – Function to return list of all the odd numbers between two numbers.
4. **calculateTotalSalary(month_sal, no_of_years)** - Function computes the total salary based on monthly salary and number of years. If the number of years is greater than 5 years, an increment of 5% is added to the total salary from 6th year.
5. **listMax(numList)** – Function to find the maximum from a list of numbers.
6. **profit_or_loss(cost_price,selling_price)** - Function to calculate actual profit or loss based on cost price and selling price using the formula, Profit = (Selling Price - Cost Price) and Loss = (Cost Price - Selling Price).
7. **printEveIndexChar(str)** - Function that displays only those characters which are present at an even index number in a given String.
8. **rental_car_cost(rate_per_day, no_of_days)** - Function that calculates the cost of renting the car based on the rental rate per day and number of days the car is rented. If you rent the car for 7 or more days, you get \$50 off your total cost. Alternatively, if you rent the car for 3 or more days, you get \$20 off your total. You cannot get both of the above discounts at the same time.
9. **average(list)** - Function to find the average of all the numbers in the given list.
10. **square_of_numbers()** -function to create and print a list where the values are square of numbers between 1 and 30 (both included).

Homework:

1. **degreeToFahrenheit(degree_celcius)** - Function that asks the user to input data in Degree Celsius (C) and convert it to Fahrenheit (F) using the formula, $F = (C * 5/9) + 32$.
2. **countVowels(sentence)** – Function that returns the count of vowels in a *sentence*. Check for both upper-case and lower-case alphabets.
3. **factorial(num)** - Function that computes the factorial of a number entered. Example: factorial of 5 = $5 * 4 * 3 * 2 * 1$.

4. **commonData(list1, list2)** - Function that takes two lists and returns true if they have at least one common element.
5. **sortList(nList, order)** – Function that sorts a list *nList*, either in descending or ascending order.
6. **string_countupperlower(sentence)** - Function that accepts a string and calculates the number of uppercase letters and lowercase letters
7. **sortList(nList, order)** – Function that sorts a list *nList*, either in descending or ascending order.

2.2 Assignment 2

1. Write a Python program to read and print an entire text file.
2. Write a Python program to read a file line by line and store it into a list.
3. Write a Python program to find the longest word in a text file.
4. Write a Python program that reads a text file called “sample.txt”. A sample of the file is provided below, you may add more details to the file. Your program must read the data from the file and count the number of words in the file

sample.txt:

I like Python Programming

5. Write a Python program that reads a text file called “employee.txt”. A sample of the file is provided below, you may add more details to the file. Each row has an employee name and respective salary. Your program must read the data from the file and identify the person with the highest salary in the list. Sample file is given below:

salary.txt:

John 41500
 Mary 43500
 Ahmed 31000
 Tom 55300
 Peter 38600
 Samantha 36900

6. Write a Python program asking for a username and password as shown below:

Login:

Username: John
 Password: abc@123

The Python program should check if the user exists in a text file named **user.txt**. If the user exists, it should welcome the user by printing “Welcome <username>”, if the user does not exist, it should print “Invalid User”. The sample text file is given below:

user.txt:

```
John abc@123
Tom pqr@789
Mary xyz@456
```

Homework:

1. Write a Python program that reads a text file called student.txt. The text file contains: student_ID, student_name and total_marks for each student. Your program should read each student details from the file, calculate the average of total marks and display the student details with highest marks and lowest marks. The program should output the report into a file called output.txt. A sample of source file and output file is given below where the values represent **student_ID**, **student_name**, and **total_marks**.

bill.txt:

20181867	Alia	400
20197541	Shamma	421
20183245	Zainab	375
20199876	Noora	456
20185432	Hamda	350
20179807	Fatma	478

output.txt:

Class Report

There are total of 6 students

The average of total marks is 413

Fatma scored the highest marks as 478

Hamda scored the lowest marks as 350

3 Lab 2 – UML Class Diagrams

The assignments for this week reviews Object Oriented design with UML notations to describe classes, attributes/data, behavior/functions, and class relationships like Association, Aggregation, Composition, and Inheritance. See Appendix A to download a tool to draw UML class diagrams. The student may use any available tool to complete the assignments.

3.1 Assignment 1

1. Design and draw a UML class diagram with attributes and behaviour to represent a class called **ZUStudent**. Ensure to select at-least 5 attributes and related behaviours.
2. Design and draw a UML class diagram with attributes and behaviour to represent a class called **Employee**. Ensure to select at-least 5 attributes and related behaviours.
3. Design and draw a UML class diagram with attributes and behaviour to represent a class called **Book**. Ensure to select at-least 5 attributes and related behaviours.

Homework:

1. For UML class diagram given below (Figure 1), identify the classes and their relationships and multiplicity. Based on your understanding of the class diagram in Figure 1, write a paragraph explaining all classes and their relationships.

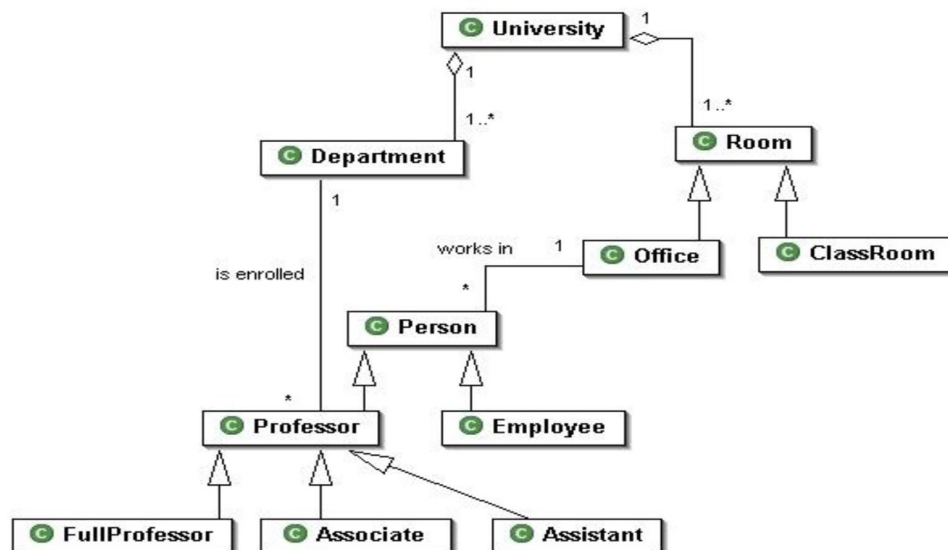


Figure 1- Class Diagram 3.1

3.2 Assignment 2

1. Design and draw UML class diagram with attributes, behavior, and class relationships to represent the **hotel reservation system**. A hotel system manages information about rooms, reservations, customers, and customer billing. A customer can make reservations, change, or cancel reservations through the hotel website. The website also maintains information about rooms which includes-Room price, Maximum occupancy, Type of room (single, double, twin, executive, suite) and room availability status. Before reservation the customer must check room availability status. If a room is available, the customer enters basic information to the system and receives a confirmation number from the web site. A desk clerk checks in a customer with only a prior reservation, change the checkout date, and check out the customer. A room is assigned to the customer at check-in time and a customer billing record is created at that time. When a customer checks out, the desk clerk prints the bill. A customer can pay the bill by cash, check, or credit card.

Homework:

1. Design and draw UML class diagram with attributes, behavior, and class relationships for the following scenario. Model a system for an **airline management system** that manages flights and pilots. An airline operates flights. Each airline has an ID. Each flight has an ID, a departure airport and an arrival airport. An airport as a unique identifier. Each flight has a pilot and a co-pilot, and it uses an aircraft of a certain type. A flight also has a departure time and an arrival time. An airline owns a set of aircrafts of different types. An aircraft can be in a working state or it can be under repair. In a moment an aircraft can, be landed or airborne. A company has a set of pilots: each pilot has an experience level:1 is minimum, 3 is maximum. A type of aircraft may need several pilots, with a different role (e.g.: captain, co-pilot, navigator).

4 Lab 3 – Introduction to Python Classes

The assignments for this week introduce the creation of classes in Python. The exercises help to understand the syntax and semantics for converting the UML class diagrams into Python code.

4.1 Assignment 1

1. Create a Python class named **Employee** and initialize it with the attributes `emp_name`, `emp_id`, `emp_age` and `salary`. Include the constructors and other required methods to set and get the class attributes. Create instances of the class and test all the associated methods.
2. From the previous assignment in Lab 2, Assignment 1, and Question 1, where a **Student** was modeled using UML class diagram, create a Python class for Student with respective attributes and methods. Create instances of the class and test all the associated methods.
3. Write a Python class named **Product** which has attributes `product_id`, `product_name`, number of items. Include the constructor and other required methods to set and get the class attributes. The class should have 2 additional methods to add and remove products from the inventory. If the number of items for a product in the inventory is less than zero, the user should be notified with a print message stating, “Product is out of stock”. Create 2 instances of the class and display all its associated attributes.
4. Create a Python class named **BankAccount**. Your class supports the following methods.

```
class BankAccount:
    """Bank Account protected by a pin number."""

    def __init__(self, pin):
        #Initial account balance is 0 and pin is 'pin'.

    def deposit(self, pin, amount):
        #Increment balance by amount and return new balance.

    def withdraw(self, pin, amount):
        #Decrement balance by amount and return amount withdrawn.

    def get_balance(self, pin):
        #Return account balance.

    def change_pin(self, oldpin, newpin):
        #Change pin from old pin to new pin.
```

4.2 Assignment 2

1. Design a Python class named **Weather** to analyze weather data. The class should have attributes such as `daily_low_temperature` and `daily_high_temperature`. The class must also include the methods to do the following:
 - a. Ask the user to enter the daily low and daily high temperature of 5 days.
 - b. Find and display the lowest and highest temperatures of all days.
 - c. Calculate and display the average temperature for each day by using the formula:
 - d. Display the 5 average temperatures in descending order.

2. Write a Python program that creates a class which represents an **Employee** in an organization. The class includes a function that reads a text file called `employee_details.txt`. A sample of the file is provided below. Each row in the file corresponds to employee id, employee name, number of years employed and salary. Also include the following functions to process the content read from the file.
 - a. `getData()`: This method reads the data from a file and stores the data as a list.
 - b. `totalSalary()`: This method calculates the total salary for each employee. The method should add an incentive of 3% to the total salary if the number of years worked by the employee is greater than 5 years.
 - c. `whoishighestTotalSalary()` and `whoislowestTotalSalary()`: These methods calculate the highest and lowest total salary and display the respective employee names.
 - d. `sortEmployeeBySalary()`: Sort the employee list in the ascending order of their salaries.

Sample input file: **employee_details.txt**

```
E001, Hasan A Jassim, 9, 8587
E002, Smith John Krane, 8, 6770
E003, Malik Nathan, 7, 8052
E004, P. S. Gupta, 3, 9598
E005, Tony Knot Blair, 5, 9170
E006, Ahmed Khan, 7, 8188
```


5 Lab 4 – Inheritance

The assignments for this week provide hands-on experience to understand and design the relationship between classes called, Inheritance. Inheritance relationship between classes is usually identified with “is-a”, “type-of”, or “kind-of” statements within the requirement statement.

5.1 Assignment 1

1. Write a Python program that represent the following scenario. An employee’s details are name, annual salary, hired date that gives the year, month, and day that the employee got hired, and a social security number. An HourlyEmployee and a SalariedEmployee are two types of employees. The difference between HourlyEmployee and SalariedEmployee is the way the salary is calculated. For HourlyEmployee the salary is calculated based on the number of hours an employee had worked and hourly rate while for the SalariedEmployee a fixed amount is given as the salary every month. Create the base class and subclasses to represent the given scenario. Create a separate test module where instances of the class are created, and the methods are tested by giving the required details and printing the details for each class.

2. An automobile company wants to store the information of different types of vehicles. Create a class named **Vehicle** with attributes: mileage and price. Types of vehicles are:
 - a. **Car** with attributes to store ownership cost, warranty (by years), seating capacity and fuel type (diesel or petrol).
 - b. **Bike** with attributes to store the number of cylinders, number of gears, cooling type (air, liquid or oil), wheel type (alloys or spokes) and fuel tank size (in inches).
 Two types of cars are: **Sedan** and **SUV**, each having attributes to store the model type. Create a separate test module where instances of the class are created, and the methods are tested by giving the required details and printing the details for each class.

3. Create a generic superclass Person, with subclasses and attributes as shown below:

```
class Person:
    Attributes: self.name, self.surname, self.phone_number

class Student(Person):
```

```

        Attributes: self.studentid, self.student_type=(UNDERGRADUATE,
        POSTGRADUATE), self.studentCGPA

class StaffMember(Person):
    Attributes: self.staffid, self.employment_type=(PERMANENT,
    TEMPORARY), self.salary

class Lecturer(StaffMember):
    Attributes: self.highest_degree, self.years_of_experience,
    self.course_taught

```

Implement the classes in Python. Create a separate test module where 2 instances of each subclass are created. Test the methods by displaying their information.

Homework:

1. Consider the following Python code and answer the following questions.
 - a. Identify the parent and child classes here?
 - b. What is the output of the code? (Try figuring it out without running the program)
 - c. What do we need to do so that self.data for class D is initialized to an integer? Write down the code that needs to be added or changed.

```

class A(object):
    def __init__(self):
        print('A')
        self.data = None

class B(A):
    def __init__(self):
        print('B')
        A.__init__(self)
        self.data = None

class C(A):
    def __init__(self):
        print('C')
        A.__init__(self)
        self.data = None

class D(B, C):
    def __init__(self):
        print('D')
        B.__init__(self)
        C.__init__(self)
        self.data = None

d = D()

```

5.2 Assignment 2

1. Create a class **Taxi** which has attributes name, mileage and seating capacity. The fare charge of any vehicle is seating capacity * 100. Create a class **TaxiVan** that inherits from the Taxi class. If Taxi is TaxiVan instance, we need to add an extra 10% on full fare as a maintenance charge. So, the total fare for the TaxiVan instance will be the final amount = total fare + 10% of the total fare. Create a separate test module where instances of the class are created, and the methods are tested by giving the required details and printing the details for each class.
2. A **hospital** is a very busy place with a lot of patients and also a lot of employees to keep the system running perfectly. All people in the hospital have basic information like title, name, gender, address, and phone number. Employees of the hospital have extra information like employee-id and salary. For every 25 employees, there is a manager who is also an employee, but gets an allowance of 7% more than the salary. There are also doctors and nurses who are employees in the hospital. Doctors and nurses are assigned to a specific department, like dental, pediatric, cardiac, and so on. Then, there are two types of patients in the hospital, the In-Patients and the Out-Patients, while both have all the basic information, the In-Patients have a room number where they are admitted.

You are required to draw the **UML Class diagram** to represent different types of people in the hospital. Identify the base class, subclasses and the relationships between them. Implement the classes in Python. Create a separate test module where instances of the class are created, and the methods are tested.

6 Lab 5 – Class Association

The assignments for this week provide hands-on experience to understand and design the relationship between classes called, Association. The two types of Associations, Aggregation and Composition, are also seen as part of the exercises.

6.1 Assignment 1

1. Write a Python program to represent given statement. Create the necessary classes and relationships between these classes.

A student borrows books from a library.

Examples of outputs that should be produced after running the program:

- a. Student Alia has borrowed 2 books and the details are as follows:
 - i. Book-1 with title “Python Programming” and Author “Jason Rees”
 - ii. Book-2 with title “Networking” and Author “Tanenbaum”
 - b. Student Ahmad has borrowed 1 book and the details are as follows:
 - i. Book -1 with title “Database” and Author “Korth”
2. Write a Python program to manage league matches. Different teams can participate in a league. A player belongs to a team and a team can play many games in a league. For each team the league wants to track the name of the team and the number of players in each team. The league also wants to track the number of games the team has played, and the total numbers of points scored across all games played. During each game a team can score no points, or a number of points represented by a whole number. Create a separate module to test the classes created with at-least 5 teams and display the matches played and the respective scores.

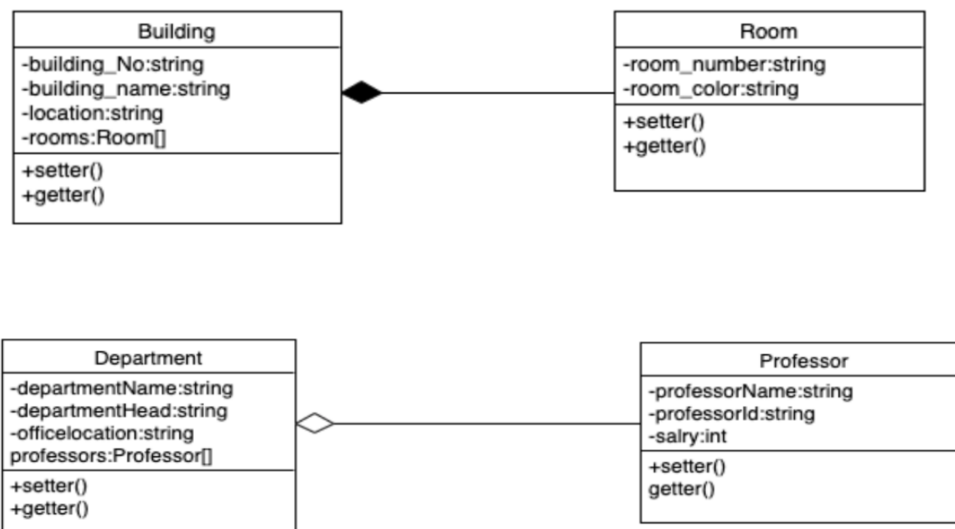
6.2 Assignment 2

1. Customers place orders for an item in the catalog. The customer needs to specify the name and address to which the ordered items will be shipped. When the order is placed, the date of order is recorded along with the current status of the order. For a given order, total price is computed along with the total weight of the items. An order can have several items and

each item could be ordered in large quantities. The details of the order includes the weight and price of each item type. Every item is detailed with its price and weight along with the description of the item.

Draw a UML diagram by identifying all the classes and relations between them. Convert UML to a python module that has all the identified classes, attributes and methods. Create a separate test module where the instances of the classes are created, and the associated methods are tested.

2. Analyze the given UML diagrams. Identify classes and relations. Convert each of the UML to python code by creating required classes and representing appropriate relations. Test each of the created classes.



Homework:

1. Write a Python Program to manage **Student Course Registration**. Each Student has a name and student number and can register for a list of courses belonging to a Department. Each Department has a department name and a department code and has a list of courses which the student can register from. Each Course has a course description, course code, credit and the department to which it belongs. You are required to draw the **UML class diagram and create the classes**, related attributes and methods to represent this scenario

7 Lab 6 – Polymorphism

The assignments for this week provide hands-on experience to understand Polymorphism in Python.

7.1 Assignment 1

1. Consider the following Python code for class `Coordinate` that attempts to simulate a point (spot) in the 2-D scheme of dimensions. Class `Coordinate` has 2 attributes `x` and `y`.

```
class Coord:

    def __init__(self, y=0, z=0):
        self.y = y
        self.z = z

    def __str__(self):
        return "({0},{1})".format(self.y, self.z)

    def __add__(self, other):
        y = self.y + other.y
        z = self.z + other.z
        return Coord(y, z)

c1 = Coord(1, 2)
c2 = Coord(2, 3)
print(c1+c2)
```

In the above code, when two instances of the class `Coord` i.e. `c1` and `c2` are added with the `+` operator we see that the built-in function `__add__` is called. Here, the `+` operator was overloaded. Include the `__mul__` and `__sub__` function in the above code using the below table and test it.

Operator	Method
+	<code>__add__(self, other)</code>
-	<code>__sub__(self, other)</code>
*	<code>__mul__(self, other)</code>
//	<code>__floordiv__(self, other)</code>
/	<code>__truediv__(self, other)</code>
%	<code>__mod__(self, other)</code>
**	<code>__pow__(self, other[, modulo])</code>

2. Write a Python class called **MyList** that has one attribute which is a list (**pList**). Include the following methods to do the listed functions.
 - `__init__`: to initialize `pList`
 - `__add__`: to join two lists
 - `__mul__`: to update `pList` with *n* copies of itself
 - `__repr__`: to print the content of the list, each element on a separate line

3. For the Python code given below:

```
class Rectangle:
    def __init__(self, l,w):
        self.length = l
        self.width=w

    def area(self):
        return self.length*self.width

    def __add__(self, other):
        new_length=self.length+other.length
        new_width=self.width+other.width
        return Rectangle(new_length,new_width)
```

- Create 2 instances of class Rectangle, r1 and r2.
- Create r3 which is a sum of r1 and r2 and print the area of r3
- Check if r3 is greater than r2 using operator overloading [gt (self, other)]. Similarly, check if r1 is less than r2 using operator overloading [lt (self, other)]
- Override the __str__ function to print the length and width of each Rectangle with a description.

Homework:

1. For the python code given below

```
class Vehicle:

    def __init__(self, fare):

        self.fare = fare
```

- Create two instances of Vehicle class-bus and car.
- Use built-in function __add__ appropriately to return sum of fare of bus and car.
- Similarly use built-in method __gt__(self, other), __lt__(self, other) to check if bus fare is greater or less than car fare.
- Override the __str__ function to print the fare of each instance of Vehicle class with a description

8 Lab 7 - Exception Handling

The assignments for this week provide hands-on experience to understand Exception Handling in Python.

8.1 Assignment 1

- For each of the predefined exception class listed below, describe for what reasons the exception is triggered.

Exception	Description
AssertionError	
ZeroDivisionError	
NameError	
MemoryError	
ValueError	
IndexError	
KeyError	
ImportError	
SyntaxError	
FileNotFoundError	

- Write a function in python that prompts the user to enter length in feet and outputs the equivalent length in centimeters. If the user enters a negative number or a non-digit number, your program should throw and handle an appropriate exception and prompt the user to enter the number again.
- Consider the code given below:
 - Test the above code and make the program crash. Identify and list for what test cases the program would crash.
 - Modify the above functions to make them more robust to erroneous input/data by handling each function with appropriate Exceptions such as: ZeroDivisonError , ValueError, IndexError, TypeError, and FileNotFoundError.
 - Test the modified code for all the above-mentioned scenarios.


```

def example1():
    age = int(input("Please enter your age: "))
    print("I see that you are %d years old." % age)

def example2():
    for i in range(3):
        dividend = float(input("Please enter the dividend: "))
        divisor = float(input("Please enter the divisor: "))
        quotient = dividend / divisor
        print(quotient)

def print_list_element(thelist, index):
    print(thelist[index])

def printUpperFile(fileName):
    file = open(fileName, "r")
    for line in file:
        print(line.upper())
    file.close()

def main():
    example1()
    example2()
    L = [10, 3, 5, 6, 9, 3]
    print_list_element(L,6)
    print_list_element([10, 3, 5, 6, "NA", 3],2)
    printUpperFile("doesNotExistY.txt")
    printUpperFile("Test.txt")
# Test Code
main()

```

4. Create a class **Customer** in Python that takes user input for customer name, email address, mobile number and age for a Movie Ticket Booking Application. The class has a method to checkAge(age) which checks a customer's age. The program should raise an exception if the customer's age is less than 18 years of age. Test the class with inputs given by the user and check if all the exceptions are raised appropriately.
5. Define a class **SavingAccount** (accNo, name, balance). The class should have withdraw(), deposit () and viewBalance() methods. The minimum balance for the account must be AED 500, your program should raise a user defined exception stating "Insufficient Funds" when your balance is not sufficient.

9 Lab 8 – Graphical User Interface (GUI)

The assignments for this week are to understand the use of tkinter modules to create GUI in Python.

9.1 Assignment 1

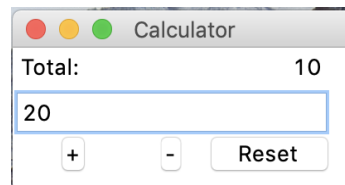
1. Create the following window using the tkinter module in Python. The button Close closes the window, and the button Print prints a greeting “Hello Python!” on the console.



2. Create the following window using the tkinter module in Python. The button *Quit* closes the window, and the button *Show* prints the first name and last name and email entered in the Entry boxes.



3. Create a simple calculator in Python which will allow the user to enter a number in the entry field, and either add it to or subtract it from a running total, using the “+” and “-” buttons. The reset button will allow the user to reset the total back to zero. If the entered value is not a number, then raise an appropriate exception in a MessageBox.



4. Create the following Registration Form using the Tkinter module in Python. The button Submit writes all the details entered in a text file. All fields are mandatory, if the user does not choose a country, gender or select a programming language when the Submit button

is clicked, then popup a message box asking the user to complete the missing information before saving it to the file.

5. Create the following window using the tkinter module in Python. All fields are mandatory, the user must use a radio button, check boxes and Process Selection button to make selections and process the order. All fields are mandatory, if the user does not choose Topping, Pizza size or Pizza Type then popup a message box asking the user to complete the missing information. Use a text area to display the customer's order and the amount due.

10 Appendix

This section has additional information for the OOP-Lab.

10.1 Text File Reading and Writing

In Python, text files are a sequence of lines. Each line ends with a special character, called the EOL or End of Line character (this represent the enter key ↵). It ends the current line and tells the interpreter a new line has begun.

To open a file for writing or reading, you must use the Python built-in function: **open()**. The **open()** function, returns a **file object** which is a handler for the file. The file object or handler has a number of functions associated with it, to help handle the file i.e. to read or write to the file.

So, for example, to open a file called “height.txt”, the Python syntax would look like this:

```
foHeights = open("height.txt", "r")
```

Here, “r” stands for “read”, which is the mode to open the file with. So, the code above is asking Python to open the file named “*height.txt*” in the read mode i.e. to read from the file and not to write to the file. The variable *foHeights* is the file object which handles the file henceforth in the program. Don’t forget to create the file “*height.txt*” before running the above line of code!

Of course, we could open a file in different modes. The modes are:

- ‘r’ – Read mode which is used when the file is only being read
- ‘w’ – Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)
- ‘a’ – Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end
- ‘r+’ – Special read and write mode, which is used to handle both actions when working with a file

To read and print the lines in file, try this code:

```
foHeights = open("height.txt", "r")
for line in foHeights:
    print(line)
```

The above code uses the for loop to loop through each line of the file and prints it out.

Now, let’s look at how to make a copy of the file.

We would need the **first file object** for the file we **read from** and the **second file object** for the file we have to **write to**. Each needs to be opened in the appropriate mode. Then, we read through every line in the first file and write it to the second file. Here's the code.

```
foHeights = open("height.txt", "r")
fwHeights = open("copyHeight.txt", "w")

for line in foHeights:
    fwHeights.write(line)
```

File object **foHeights** is for the file that we read from, and file object **fwHeights** is the file object for the file that we write to.

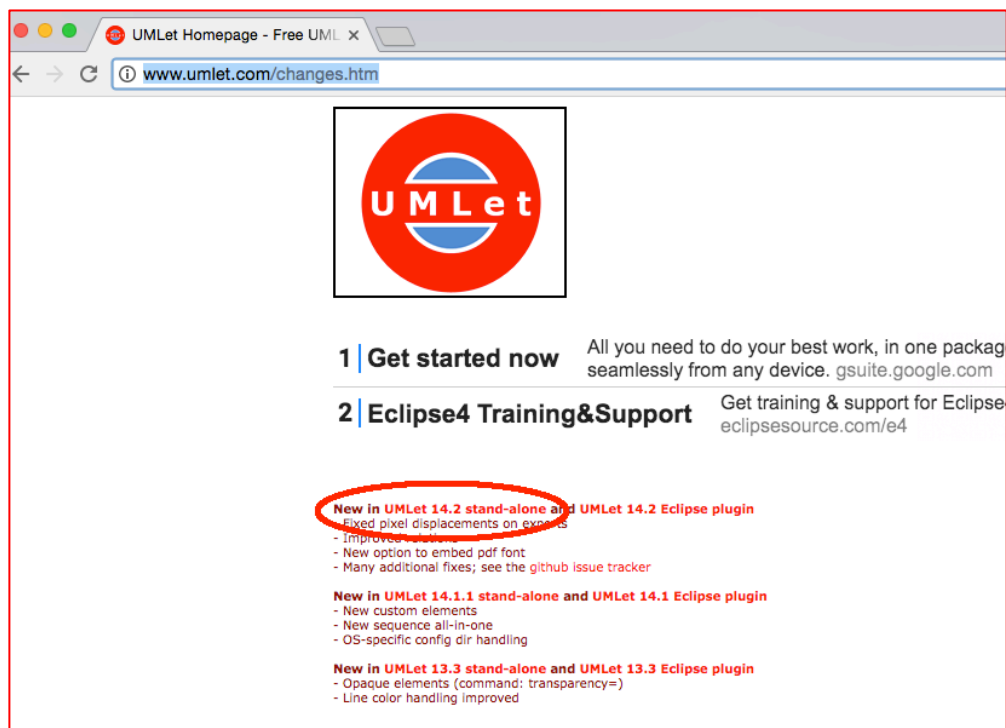
The **for** loop reads every line from **foHeights** and the *write()* function of **fwHeights** in the loop writes every line to the new file. Try it out!

While reading the lines from a file in the loop, the program can make various decisions and also process the content of the file using the usual programming techniques.

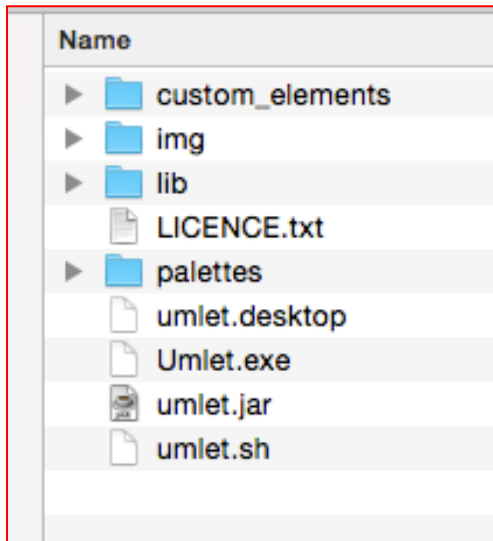
10.2 UML Tool

Download the UMLET tool to draw UML diagrams. Any tool that helps to draw UML diagrams can be used, this is only suggestion.

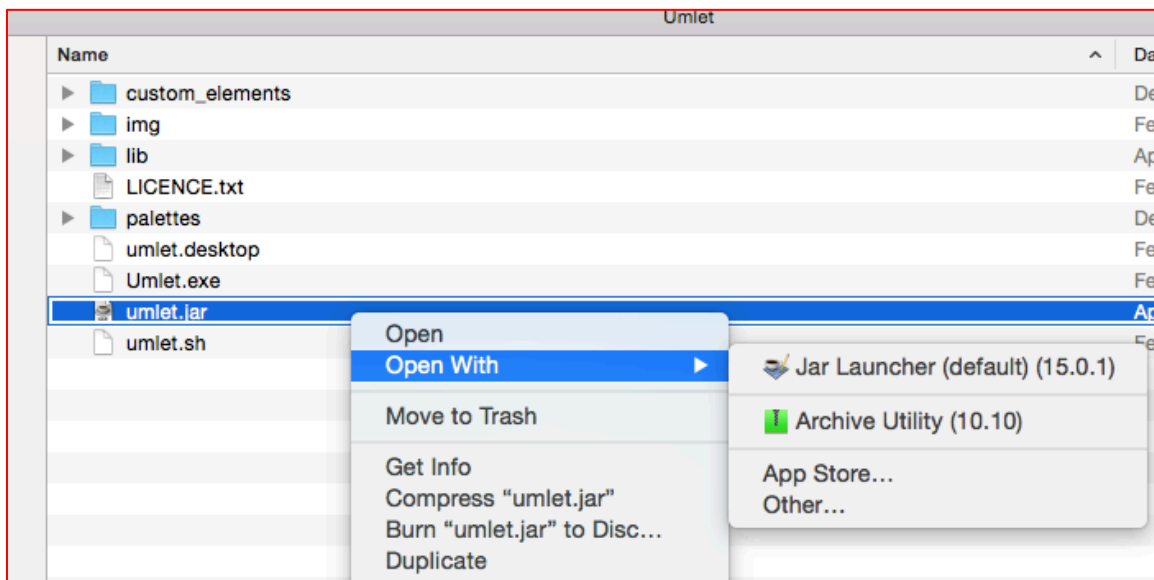
1. Navigate to <http://www.umlet.com/changes.htm>



2. Download the latest **UMLet Standalone** tool as a zip file.
3. Unzip the zipped file to see the **Umlet** folder
4. The folder content is shown below:



5. Use the appropriate startup file depending on the OS of your computer.
 - a. For Windows: Umlet.exe
 - b. For Unix: umlet.sh
 - c. For Others like Mac: umlet.jar (run with Jar Launcher, as shown below)



10.3 Sample Questions

1. What is the output of the following program?

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def mySub(self, other):
        x = self.x - other.x
        y = self.y - other.y
        return Point(x,y)

p1 = Point(3, 4)
p2 = Point(1, 2)
result = p2.mySub(p1)
print(result.__getattribute__("x"))
```

2. Choose the best option.

The **+** sign next to an attribute in UML, denotes the _____ class member visibility.

- ☐ A. private
- ☐ B. public
- ☐ C. protected
- ☐ D. package

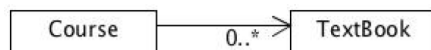
3. Choose the best option.

The State of an Object is determined by changes to _____.

- ☐ A. Functions
- ☐ B. Value of Attributes
- ☐ C. Classes
- ☐ D. Methods in Classes

4. Choose the best option.

What does the UML Class diagram represent?



- ☐ A. Aggregation - Many textbooks in a Course
- ☐ B. Binary Association - Course has textbook
- ☐ C. Unary Association - Course has zero or more textbook.
- ☐ D. Unary Association - Textbook has zero or more courses.

5. Briefly describe in your own words two reasons why object-oriented programming is more preferable to structured or unstructured programming.

6. What is the output of the following program?

```
class Person:
    def __init__(self, id):
        self.id = id
ma = Person(1001)
ma.__dict__['age'] = 49
print(ma.age - len(ma.__dict__))
```

7. What is the output of the following program?

```
class Demo:
    def __init__(self):
        self.x = 1
    def change(self):
        return self.x

class Demo_derived(Demo):
    def change(self):
        self.x = self.x + 10
        return self.x
def main():
    obj = Demo_derived()
    print(obj.change())
main()
```

8. What is the output of the following program?

```
class Parent:
    def __init__(self, param1):
        self.v1 = 12 - param1

class Child(Parent):
    def __init__(self, param):
        self.v1 = param
        self.v2 = param

obj = Child(12)
print(obj.v2, obj.v1)
```

9. What is the output of the following program?

```
try:
    if '950' != 950:
        raise AssertionError
    else:
        print("no error has occurred")
except:
    print("an exception has happened")
else:
    print("No error at all")
finally:
    print("End")
```


10. Choose the best option.

What is getattr() used for?

- ☐ A. To access the attribute of the object
- ☐ B. To delete an attribute
- ☐ C. To check if an attribute exists or not
- ☐ D. To set an attribute

11. Choose the best option.

When we add two underscore (__) before a class attribute, it becomes _____.

- ☐ A. private
- ☐ B. public
- ☐ C. protected
- ☐ D. package

12. Choose the best option.

Spaghetti Code is the possible consequence of using an unstructured or structured programming paradigm for building large software.

- ☐ True
- ☐ False

13. Write an Object-Oriented Python program for the description given below.

The class Tablet has five attributes to define its name, memory size in GB, basic price, years of warranty, and a Boolean indicating whether it has a protective case or not. Implement the function getPrice based on the logic, for each year of warranty, add 50 AED. Further, if the tablet has a case, add 200 AED.

The following test case reveals the use of the class. Define a Python class to match the test

```
#Test: Do NOT change this code
t1 = Tablet()
t1.setName("iPad 5")
t1.setBasicPrice(700)
t1.setMemorySize(64)
t1.setWarrantyYears(2)
t1.setCase(True)
print(t1.getPrice()) # calculate and show the final price of the tablet

#defines name, price, memory size, warranty, and case respectively.
t2 = Tablet("Amazon Fire X",400,128,3,False)
print(t2.getWarrantyYears())
# shows all information about the tablet including price
print(t2)
```

14. Write an Object-Oriented Python program for the description given below. Classes represent Apartment, Building, and Villa. Building and Villa inherit from the class Asset.

The class Asset has a name, address, and revenue as attributes where revenue is a float value. A Villa is an Asset that has rent and maintenance, both are float values. A Building is an Asset that has a number of apartments. An Apartment has rent and maintenance, both are float values.

All the revenue and rent attributes are private. Each class has a revenue() method to calculate the monthly revenue as shown below. Your program needs to calculate the revenue from Buildings and Villas and display the Asset's name, address, and total revenue. Test the program to display the details of all assets. If a particular asset does not generate any revenue, an appropriate exception is thrown to indicate the same.

Formulas:

- Revenue of Villa = rent – maintenance
- Revenue of Building = (number of apartments x apartment rent) – (maintenance x number of apartments)

The student is required to identify classes, related attributes, and behaviors. Students are free to add as many attributes as required. The appropriate access modifiers (private, public, or protected) must be used while implementing class relationships. Proper documentation of code is mandatory. The student must also test the code, by creating at-least three objects. All classes must have a display function, and the program must have the functionality to print the current state of the objects

15. Write an Object-Oriented Python program for the description given below.

Write a Python class to represent the salary of an Employee. The salary has two components Basic and Allowance. The class has functions to add each component to an employee's salary. The class also has a function to compare the salaries of two employees, which checks if two employees have the same total salary. When comparing salaries, the salary components may be different but if the total salary is NOT the same, then the function returns an appropriate error or else returns the total salary. Complete the Python code given below.

The student is required to identify classes, related attributes, and behaviors. Students are free to add as many attributes as required. The appropriate access modifiers (private, public, or protected) must be used while implementing class relationships. Proper documentation of code is mandatory. The student must also test the code, by creating at-least three objects. All classes must have a display function, and the program must have the functionality to print the current state of the objects.




```
# Sample code to test the EmpSalary class
e1 = EmpSalary() # First Employee
e1.addBasic(8000)
e2 = EmpSalary(8000, 800) # Second Employee

try:
    print(e1.compareSalary(e2))
except AssertionError:
    print("Salaries are not same")
else:
    print("Salaries are same")
```






10.4 Grading Rubric

The reader is advised to go through the following content to understand assessment criteria for the lab assignments. The assignment grading rubrics are provided here and is used to assess the submitted assignments.

Rubric for evaluating submissions with UML Diagrams

Criteria	Levels of Achievement				
	Exemplary (A)	Accomplished (A-/B+/B)	Developing (B-/C+/C)	Beginning (C-/D+/D)	0 (F)
DOCUMENTATION  Weight 20.00%	90 to 100 % Excellent readability.	80 to 89 % Very good readability.	70 to 79 % Good readability.	60 to 69 % Submission is not clear and readability is poor.	0 to 59 % No documentation included.
UML Notations  Weight 60.00%	90 to 100 % The solution fully meets the below criteria: All UML notations are correctly used.	80 to 89 % The solution mostly meets the below criteria: All UML notations are correctly used.	70 to 79 % The solution somewhat meets the below criteria: All UML notations are correctly used.	60 to 69 % The solution partially meets the below criteria: All UML notations are correctly used.	0 to 59 % The solution fails to meet the below criteria: All UML notations are correctly used.
SUBMISSION QUALITY  Weight 20.00%	90 to 100 % Submission was on time.	80 to 89 % Submission is 1 day late	70 to 79 % Submission is 2 days late	60 to 69 % Submission is 3 days late	0 to 59 % Submission is very late

Rubric for evaluating submissions with Python Code

Criteria	Levels of Achievement				
	Exemplary (A)	Accomplished (A-/B+/B)	Developing (B-/C+/C)	Beginning (C-/D+/D)	0 (F)
DOCUMENTATION  Weight 10.00%	90 to 100 % Excellent readability. Comments are sufficient and Indentation are correct.	80 to 89 % Very good readability. Comments are sufficient and Indentation are correct	70 to 79 % Good readability. Comments are included and Indentation are correct	60 to 69 % Code is not readable and not well documented.	0 to 59 % No documentation included.
INPUT STATEMENTS  Weight 20.00%	90 to 100 % The solution fully meets the below criteria: Input statements are correct according to specs.	80 to 89 % The solution mostly meets the below criteria: Input statements are correct according to specs.	70 to 79 % The solution somewhat meets the below criteria: Input statements are correct according to specs.	60 to 69 % The solution partially meets the below criteria: Input statements are correct according to specs.	0 to 59 % The solution fails to meet the below criteria: Input statements are correct according to specs.
OUTPUT STATEMENTS  Weight 30.00%	90 to 100 % The solution fully meets the below criteria: Output statements are correct as required	80 to 89 % The solution mostly meets the below criteria: Output statements are correct as required	70 to 79 % The solution somewhat meets the below criteria: Output statements are correct as required	60 to 69 % The solution partially meets the below criteria: Output statements are correct as required	0 to 59 % The solution fails to meet the below criteria: Output statements are correct as required
PROGRAM LOGIC  Weight 30.00%	90 to 100 % The solution fully meets the below criteria: Program logic is correct.	80 to 89 % The solution mostly meets the below criteria: Program logic is correct.	70 to 79 % The solution somewhat meets the below criteria: Program logic is correct.	60 to 69 % The solution partially meets the below criteria: Program logic is correct.	0 to 59 % The solution fail to meet the below criteria: Program logic is correct.
SUBMISSION QUALITY  Weight 10.00%	90 to 100 % The solution fully meets the below criteria: Submission was on time.	80 to 89 % The solution mostly meets the below criteria: Submission is 1 day late	70 to 79 % The solution mostly meets the below criteria: Submission is 2 days late	60 to 69 % The solution mostly meets the below criteria: Submission is 3 days late	0 to 59 % The solution mostly meets the below criteria: Submission is very late

10.5 Grading Key

Based on Zayed University policy ACA-ADM-11, the grade points and percentage ranges are assigned to each letter grade as follows:

Percentage	Grade	Grade Points	
90-100%	A	4.000	The highest academic grade possible. This grade is not automatically given to a student who ranks highest in the course, but is reserved for accomplishment that is truly distinctive and demonstrably outstanding. It represents a superior mastery of course material and is a grade that demands a very high degree of understanding as well as originality or creativity appropriate to the nature of the course. The grade usually indicates that the student works independently with unusual effectiveness and often takes the initiative in seeking new knowledge outside the requirements of the course.
87-89%	A-	3.700	
84-86%	B+	3.300	
80-83%	B	3.000	Denotes achievement considerably above acceptable standards. Good mastery of course materials is evident, and student performance demonstrates a degree of originality, creativity, or both. The grade usually indicates that the student works fairly well independently and often demonstrates initiative.
77-79%	B-	2.700	
74-76%	C+	2.300	
70-73%	C	2.000	Indicates an appropriate level of competency in the course's basic learning outcomes. It is the grade that may be expected of a student with an average level of performance who gives to the work a reasonable amount of time and effort. This grade implies understanding of the content of the course, acceptable mastery of course material and learning outcomes, and completion of all requirements. The student must have a minimum cumulative GPA of 2.0 (C) to earn a baccalaureate degree from Zayed University.
67-69%	C-	1.700	
64-66%	D+	1.300	
60-63%	D	1.000	Denotes a limited understanding of the subject matter, meeting only the minimum requirement for passing the course. It signifies work that in quality or quantity falls below the average acceptable standard for passing the course. Performance is deficient in analysis, synthesis, and critical expression. There is little evidence of originality or creativity.
0-59%	F	0	Indicates inadequate or unsatisfactory attainment, serious deficiency in understanding of course material, or failure to complete the requirements of the course.