  
  
**Assignment Cover Sheet**

|  |  |
| --- | --- |
| **Subject Code: CSIT110** |  |
| **Subject Name: Fundamental Programming with Python** |  |
| **Submission Type: Online** |  |
| **Assignment Title: Assignment 1** |  |
| **Student Name: Mudethir Abdulhaq Mohammed Elhassan, Zahra Hossaini** |  |
| **Student Number: 6686345, 6331282** |  |
| **Student Phone/Mobile No.: 0501855175, 0563700760** |  |
| **Student E-mail: mb094, zh173** | @uowmail.edu.au |
| **Lecturer Name: Abdellatif Tchantchane** |  |
| **Due Date: 10/08/2022** |  |
| **Date Submitted: 06/08/2022** |  |

|  |  |
| --- | --- |
| **PLAGIARISM:** The penalty for deliberate plagiarism is FAILURE in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. UOWD has a strong policy against plagiarism.  The University of Wollongong in Dubai also endorses a policy of non-discriminatory language practice and presentation.  **PLEASE NOTE:**STUDENTS MUST RETAIN A COPY OF ANY WORK SUBMITTED | **DECLARATION:** I/We certify that this is entirely my/our own work, except where I/we have given fully-documented references to the work of others, and that the material contained in this document has not previously been submitted for assessment in any formal course of study. I/we understand the definition and consequences of plagiarism.  **Signature of Student: Mudethir Abdulhaq Mohammed Elhassan, Zahra Hossaini** |

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Optional Marks:** | | **Comments:** | |

https://my.uowdubai.ac.ae/images/scissors.gif

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Lecturer Assignment Receipt**(To be filled in by student and retained by Lecturer upon return of assignment) | | | **Subject:** | **Assignment Title:** | | **Student Name:** | **Student Number:** | | **Due Date:** | **Date Submitted:** | | **Signature of Student:** | | |

https://my.uowdubai.ac.ae/images/scissors.gif

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Student Assignment Receipt** (To be filled in and retained by Student upon submission of assignment) | | | **Subject:** | **Assignment Title:** | | **Student Name:** | **Student Number:** | | **Due Date:** | **Date Submitted:** | | **Signature of Lecturer** | | |

Table of Contents

[Part A 3](#_Toc110718663)

[Code: 3](#_Toc110718664)

[**Main.py:** 3](#_Toc110718665)

[**Logic.py:** 6](#_Toc110718666)

[Screenshots: 10](#_Toc110718667)

[**Add an employee:** 11](#_Toc110718668)

[**Remove an employee:** 12](#_Toc110718669)

[**View list of employees :** 12](#_Toc110718670)

[**Change employee’s salary:** 13](#_Toc110718671)

[**Change employee’s section:** 13](#_Toc110718672)

[**View Salaries Average:** 13](#_Toc110718673)

[**View Sections Summary:** 14](#_Toc110718674)

[**Exit:** 14](#_Toc110718675)

[Part B 15](#_Toc110718676)

[Code: 15](#_Toc110718677)

[Test Cases: 15](#_Toc110718678)

# Part A

## Code:

### **Main.py:**

import logic

def addEmployee():

    #while given id is not available

    idExists = True

    while idExists is True :

        try:

            id = int(input("\nKindly enter id of new employee containing 6 digits:\n"))

            #Check to accept only ID of 6 digits

            if len(str(id)) != 6:

                print("ID not accepted.")

            #Check if id already exists

            pos = 0

            for emp in logic.Employees:

                if id == emp.id:

                    print("ID already Exists.")

                    idExists = True

                    break

                pos += 1

            #Given id is new

            idExists = False

            logic.addEmployee(id)

        except ValueError:

            print("Enter an ID containing integers only.")

def removeEmployee():

    empPos = logic.getID("\nEnter ID of Employee to Remove:\n")

    logic.removeEmployee(empPos)

def checkSalary():

    empPos = logic.getID("Enter ID of Employee to change his/her Salary:")

    new\_salary = 0

    while True:

        try:

            new\_salary = int(input("Enter new Salary:"))

            break

        except ValueError:

            print("Enter a salary of digits only")

    #if both inputs are valid

    logic.changeSalary(empPos, new\_salary)

def checkSection():

    empPos = logic.getID("Enter ID of Employee to change his/her Section:")

    new\_section = logic.getSection("Enter new Section:")

    #if both inputs are valid

    logic.changeSection(empPos, new\_section)

def displaySalariesAVG():

    avgs = logic.salariesAverage()

    print("\nAdmin Average Salary:", str(int(avgs[0])),

          "\nIT Average Salary:   ", str(int(avgs[1])),

          "\nHR Average Salary:   ", str(int(avgs[2])),

          "\nSales Average Salary:", str(int(avgs[3])),

          "\nAccounting Average Salary: ", str(int(avgs[4])))

def displaySummary():

    #get number of employees, number of male and females per section

    Admin, IT, HR, Sales, Accounting = logic.sectionsSummary()

    #list of average salary in all sections (as integers)

    averages = [ int(average) for average in logic.salariesAverage()]

    #calculate total average salary of all sections

    sectionsAverage = 0

    for average in averages:

        sectionsAverage += average

    sectionsAverage = int(sectionsAverage / len(averages))

    #display table

    print("\t\tAdmin\t|\tIT\t|\tHR\t|\tSales\t|\tAccounting")

    print("NO of Males\t  {0}\t\t{1}\t\t{2}\t\t {3}\t\t    {4}".format(Admin[0], IT[0], HR[0], Sales[0], Accounting[0]))

    print("NO of Females\t  {0}\t\t{1}\t\t{2}\t\t {3}\t\t    {4}".format(Admin[1], IT[1], HR[1], Sales[1], Accounting[1]))

    print("Salary Average\t  {0}\t\t{1}\t\t{2}\t\t {3}\t\t    {4}".format(averages[0], averages[1], averages[2], averages[3], averages[4]))

    print("Total Employees\t  {0}\t\t{1}\t\t{2}\t\t {3}\t\t    {4}".format(Admin[2], IT[2], HR[2], Sales[2], Accounting[2]))

    print("Total company employees:", str(Admin[2]+IT[2]+HR[2]+Sales[2]+Accounting[2]))

    print("Total average of sections salaries:", str(sectionsAverage))

#User Menu

option = 0

while True:

    print("\n\n=======================================")

    print("Select an option: ")

    print("\t1. Add an employee\n\t2. Remove an employee\n\t3. View list of employees\n\t4. Change employee's salary\n\t5. Change employee's section\n\t6. View Salaries average\n\t7. View Sections summary\n\t8. Exit")

    print("=======================================")

    option = int(input("input: "))

    if option == 1:

        addEmployee()

    elif option == 2:

        removeEmployee()

    elif option == 3:

        logic.viewEmployeeList()

    elif option == 4:

        checkSalary()

    elif option == 5:

        checkSection()

    elif option == 6:

        displaySalariesAVG()

    elif option == 7:

        displaySummary()

    elif option == 8:

        print("Exiting Menu")

        print("Thank you for using the program")

        break

    else:

        print("Choose one of the provided options")

### **Logic.py:**

class Employee:

    def \_\_init\_\_(self, name, id, salary, section, hoursPerWork, male):

        self.name = name

        self.id = id

        self.salary = salary

        self.section = section

        self.hoursPerWork = hoursPerWork

        self.male = male

Employees = [

    Employee("James", 782136, 70000, "Admin", 100, True),

    Employee("Mark", 554723, 55000, "Admin", 90, True),

    Employee("Ali", 116573, 10000, "IT", 40, True),

    Employee("Mohammed", 873654, 20000, "IT", 60, True),

    Employee("Fatima", 445236, 30000, "HR", 84, False),

    Employee("Khalid", 229475, 15000, "Sales", 56, True),

    Employee("Ayesha", 792643, 60000, "IT", 93, False),

    Employee("Maria", 888130, 20000, "Accounting", 55, False),

    Employee("Saad", 913746, 35000, "HR", 67, True),

    Employee("Riya", 546293, 45000, "Accounting", 50, False),

    Employee("Rahul", 232903, 50000, "Sales", 61, True),

    Employee("Jazlyn", 446839, 65000, "Sales", 75, False)

]

sections = ['Admin','IT','Sales','HR','Accounting']

def addEmployee(id):

    name = input("Enter Name of new Employee:\n")

    salary = int(input("Enter Salary of new Employee:\n"))

    section = getSection("Enter Section of new Employee:\n")

    hoursPerWork = int(input("Enter Hours per Work of new Employee:\n"))

    male = ""

    #Get a valid gender input

    while True:

        gender = input("Enter Gender of new Employee ['F' for female, 'M' for male]:")

        if gender == "M".casefold():

            male = True

            break

        elif gender == "F".casefold():

            male = False

            break

        else:

            print("Kindly enter F or M")

    #add employee to the list

    Employees.append(Employee(name,id,salary,section,hoursPerWork, male))

    print("Employee added successfully.\n")

def removeEmployee(empPos):

    Employees.pop(empPos)

    print("Employee Removed successfully.")

def viewEmployeeList():

    print("\nEmployees List:\n")

    for obj in Employees:

        print(obj.name,obj.id,obj.salary,obj.section,obj.hoursPerWork,obj.male)

def changeSalary(empPos, new\_salary):

    Employees[empPos].salary = new\_salary

    print("\nEmployee Salary has been updated Successfully.\n")

def changeSection(empPos,new\_section):

    Employees[empPos].section = new\_section

    print("\nEmployee Section has been updated Successfully.\n")

def salariesAverage():

    Admin\_average = 0; Admin\_count = 0

    IT\_average = 0; IT\_count = 0

    HR\_average = 0; HR\_count = 0

    Sales\_average = 0; Sales\_count = 0

    Accounting\_average = 0; Accounting\_count = 0

    #iterate through the employees list and calculate salary average and number of employees per section respectively

    for employee in Employees:

        if employee.section == "Admin":

            Admin\_average += employee.salary

            Admin\_count += 1

        elif employee.section == "IT":

            IT\_average += employee.salary

            IT\_count += 1

        elif employee.section == "HR":

            HR\_average += employee.salary

            HR\_count += 1

        elif employee.section == "Sales":

            Sales\_average += employee.salary

            Sales\_count += 1

        else:

            #Accounting

            Accounting\_average = employee.salary

            Accounting\_count += 1

    #get total average for each section

    Admin\_average = Admin\_average / Admin\_count

    IT\_average = IT\_average / IT\_count

    HR\_average = HR\_average / HR\_count

    Sales\_average = Sales\_average / Sales\_count

    Accounting\_average = Accounting\_average / Accounting\_count

    return [Admin\_average, IT\_average, HR\_average, Sales\_average, Accounting\_average]

def sectionsSummary():

    Admin\_male = 0; Admin\_female = 0

    IT\_male = 0; IT\_female = 0

    HR\_male = 0; HR\_female = 0

    Sales\_male = 0; Sales\_female = 0

    Accounting\_male = 0; Accounting\_female = 0

    for employee in Employees:

        if employee.section == "Admin":

            if employee.male is True:

                Admin\_male += 1

            else:

                Admin\_female += 1

        elif employee.section == "IT":

            if employee.male is True:

                IT\_male += 1

            else:

                IT\_female += 1

        elif employee.section == "HR":

            if employee.male is True:

                HR\_male += 1

            else:

                HR\_female += 1

        elif employee.section == "Sales":

            if employee.male is True:

                Sales\_male += 1

            else:

                Sales\_female += 1

        else:

            #Accounting

            if employee.male is True:

                Accounting\_male += 1

            else:

                Accounting\_female += 1

    #tuples for each section: numbers of males, numbers of females, number of employees per section

    Admin = (Admin\_male, Admin\_female, Admin\_male+Admin\_female)

    IT = (IT\_male, IT\_female, IT\_male+IT\_female)

    HR = (HR\_male, HR\_female, HR\_male+HR\_female)

    Sales = (Sales\_male, Sales\_female, Sales\_male+Sales\_female)

    Accounting = (Accounting\_male, Accounting\_female, Accounting\_male+Accounting\_female)

    return [Admin, IT, HR, Sales, Accounting]

def getID(message):

    matchingID = False

    empPos = 0

    #Keep asking for an existing id

    while matchingID is False:

            try:

                id = int(input(message))

                pos = 0

                for emp in Employees:

                    if id == emp.id:

                        matchingID = True

                        empPos = pos

                        break

                    pos += 1

                if matchingID is False:

                    print("ID not found.")

            except ValueError:

                print("Enter an ID containing integers only.")

    #return pos of employee in the list

    return empPos

def getSection(message):

    new\_section = ""

    matchingSection = False

    while matchingSection is False:

        new\_section = input(message)

        for section in sections:

            if new\_section == section:

                matchingSection = True

        if matchingSection is False:

            print("Section does not exist.")

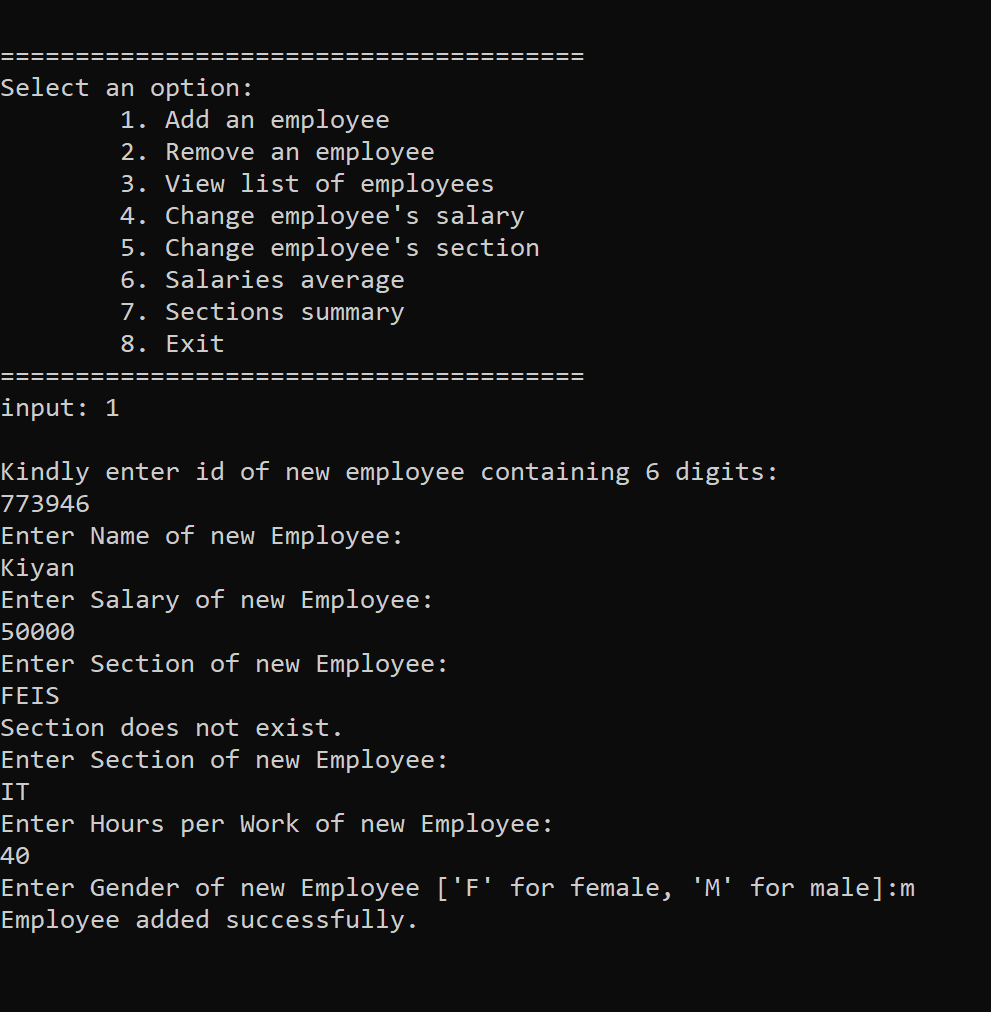
    return new\_section

## Screenshots:

Text

Description automatically generated

### **Add an employee:**



### **Remove an employee:**

Graphical user interface, text, application

Description automatically generated

### **View list of employees :**

Text

Description automatically generated

### **Change employee’s salary:**

Text

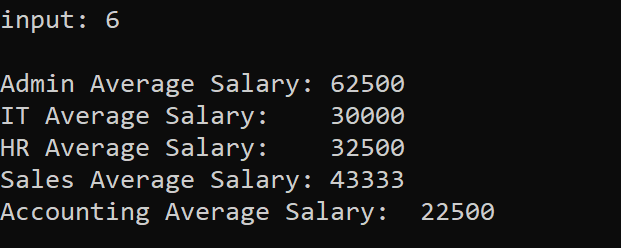
Description automatically generated

### **Change employee’s section:**

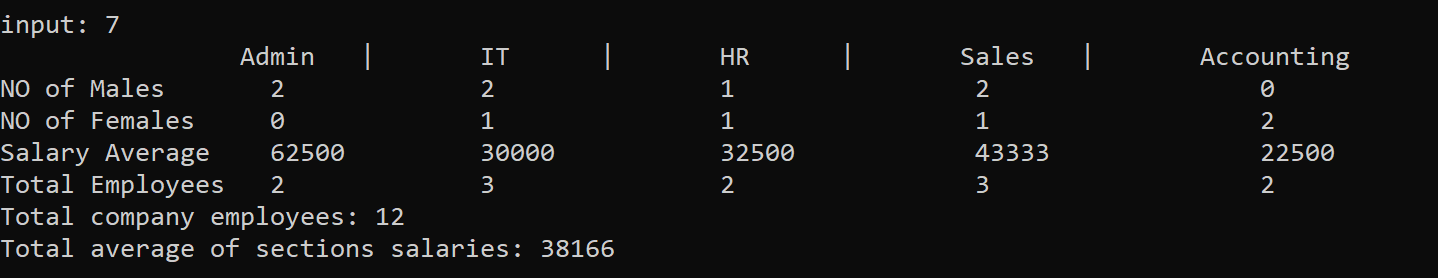
Text

Description automatically generated

### **View Salaries Average:**



### **View Sections Summary:**



### **Exit:**

Text

Description automatically generated

# Part B

## Code:

### **Main.py:**

import logic

import time

import datetime

import os

def delay(seconds, message=""):

    start = time.time()

    duration = 0

    halfway = False

    while int(duration) != seconds:

        duration = int(time.time() - start)

        if halfway is False:

            if duration == 30:

                halfway = True

                print(message)

def register():

    while True:

        username = input("Enter your username: ")

        #check username length

        if len(username) < 4:

            print("Enter a username of minimum 4 characters")

        elif logic.checkExistingUsername(username) is False:

            #if username is available, store credentials

            randomPassword = logic.passwordGenerator()

            encryptedPassword = logic.encrypted(randomPassword)

            dateOfRegistration = str(datetime.datetime.now())

            latestLogin = dateOfRegistration

            logic.storeCredentials(username, encryptedPassword, dateOfRegistration, latestLogin)

            #display password

            print("Your password is:", randomPassword, " |  (This password is shown for one minute)")

            delay(60, "30 seconds passed")

            print("One minute passed. Going back to menu.")

            delay(3)

            os.system("cls" if os.name == "nt" else "clear") #clear screen

            break

def login():

    while True:

        username = input("Enter username:")

        value = logic.checkExistingUsername(username) #if not false, value is an integer

        if value is not False:

            while True:

                password = input("Enter password:")

                recordIndex = logic.login(value,password) #if not false, recordIndex is an integer

                if recordIndex is not False:

                    print("Login successful. \nWelcome to our program!")

                    #display record

                    record = logic.returnRecord(recordIndex)

                    username, encryptedPassword, reg\_date, latestLogin = record.split(";")

                    print("Username:", username)

                    print("Your Password:", password)

                    print("Registration Date:", reg\_date)

                    print("Latest Login:", latestLogin)

                    #get current login date

                    loginDate = str(datetime.datetime.now())

                    #update user record

                    logic.updateRecord(recordIndex, loginDate)

                    break

                elif logic.login(value,password) is False:

                    print("Wrong password!")

            break

        elif logic.checkExistingUsername(username) is False:

            print("Username does not exist!")

#User Menu

option = 0

while True:

    print("\n\n=======================================")

    print("Select an option: ")

    print("\t1. Register\n\t2. Login\n\t3. Exit")

    print("=======================================")

    try:

        option = int(input("input: "))

        if option == 1:

            register()

        elif option == 2:

            login()

            pass

        elif option == 3:

            print("Exiting Menu")

            print("Thank you for using the program")

            break

        else:

            print("Choose one of the provided options")

    except ValueError:

        print("Enter the number of the option you want")

### **Logic.py:**

import random

#create database file if non-existent

try:

    file = open("database.txt", "r")

    file.close()

except FileNotFoundError:

    #create file

    file = open("database.txt", "w")

    file.close()

#generate password with range of capital letters and small letters and some symbols

def passwordGenerator():

    password = ""

    for x in range(4):

        password += chr(random.randrange(64, 126))

    return password

def encrypted(password):

    encrypted = ""

    for char in password:

        # if shifted char is between the valid password range

        if ord(char) + 7 <= 126:

            encrypted += chr(ord(char) + 7)

        else:

            #add last char from the valid password range

            encrypted += chr(126)

    return encrypted

def storeCredentials(username, password, dateOfRegistration, latestLogin):

    file = open("database.txt", "a")

    file.write(username+";"+password+";"+dateOfRegistration+";"+latestLogin+"\n")

    file.close()

def checkExistingUsername(username):

    file = open("database.txt", "r")

    i = 0

    for line in file:

        #extract username from each user record

        savedUsername = line.split(";")[0]

        if username == savedUsername:

            return i  #username already exists

        i += 1

    return False #username does not exist

def updateRecord(index, latestLogin):

    file1 = open("database.txt",'r')

    records = file1.readlines()

    record = records[index].split(";")

    #update latestLogin element

    record[3] = latestLogin

    #update records

    records[index] = record[0] + ";" + record[1] + ";" + record[2] + ";" + record[3] + "\n"

    file1.close()

    #update database file

    file1 = open("database.txt", "w")

    for record in records:

        file1.write(record)

    file.close()

def returnRecord(index):

    file1 = open("database.txt",'r')

    record = file1.readlines()[index]

    file1.close()

    return record

def login(index,password):

    record = returnRecord(index)

    username, encryptedPassword, reg\_date, latestLogin = record.split(";")

    if encrypted(password) == encryptedPassword:

        return index

    return False

## Test Cases:

### **Register user 1:**

**Before one minute:**

Graphical user interface, text

Description automatically generated

**After one minute:**

Text

Description automatically generated

Text

Description automatically generated

Register user 2:

Before one minute:

Text

Description automatically generated

After one minute:

Text

Description automatically generated

Text

Description automatically generated

Register user 3:

Before one minute:

Text

Description automatically generated

**After one minute:**

Text

Description automatically generated

Text

Description automatically generated

**Users Data stored in Database.txt file (with encrypted passwords, login date and registration date):**

Text

Description automatically generated

### **User 1 Login (with correct password):**

Text

Description automatically generated

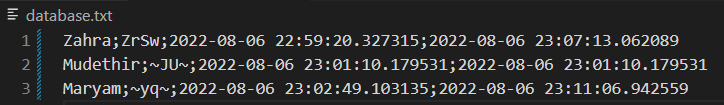
Text

Description automatically generated

### **User 3 Login (with correct password):**

Text

Description automatically generated



### **User 1 Login (with WRONG password):**

Text

Description automatically generated

Text

Description automatically generated

### **User 1 Login again (with CORRECT password):**

Text

Description automatically generated

Text

Description automatically generated

### **Exiting Menu:**

Text

Description automatically generated