LAB REPORT

**Lab1: Basic Python Programming**

**Môn học:** Lập trình cho kỹ sư mạng máy tính

**Lớp: CNBU103.P31.ATBC**

**THÀNH VIÊN THỰC HIỆN:**

|  |  |  |
| --- | --- | --- |
| **STT** | **Name** | **MSSV** |
| 1 | Đào Minh Châu | 24560025 |

|  |
| --- |
| **Điểm tự đánh giá** |
| **100** |

**ĐÁNH GIÁ KHÁC:**

|  |  |
| --- | --- |
| Tổng thời gian thực hiện |  |
| Phân chia công việc |  |
| Ý kiến *(nếu có)*  + Khó khăn  + Đề xuất, kiến nghị |  |

Contents

[A. REPORT DETAILS 2](#_Toc201434360)

[1. Assignments 1: Python Basics & Control Flow 2](#_Toc201434361)

[a. User Introduction 2](#_Toc201434362)

[b. Even or Odd Checker 3](#_Toc201434363)

[c. Simple Grading System 3](#_Toc201434364)

[d. Multiplication Table 3](#_Toc201434365)

[e. Number Guessing Game 3](#_Toc201434366)

[2. Assignment 2: Data Structures & File I/O 3](#_Toc201434367)

[a. Score: Statistics 3](#_Toc201434368)

[b. Student: Directory 3](#_Toc201434369)

[c. Word Frequency Counter 3](#_Toc201434370)

[d. Tuple: Conversion 3](#_Toc201434371)

[3. Assignment 3: Classes, Objects, and Inheritance 3](#_Toc201434372)

[a. Class: Rectangle 3](#_Toc201434373)

[b. Class: BankAccount 3](#_Toc201434374)

[c. Inheritance: Employee System 3](#_Toc201434375)

[d. Mini Project: Library Management 3](#_Toc201434376)

[B. TÀI LIỆU THAM KHẢO 3](#_Toc201434377)

# REPORT DETAILS

## Assignments 1: Python Basics & Control Flow

### User Introduction

Prompt the user for their name, age, and city. Print a message like:

"Hi Alice, you are 20 years old and live in Hanoi."

name= input("Enter your name:")

age= int(input("Enter your age:"))

while age<0:

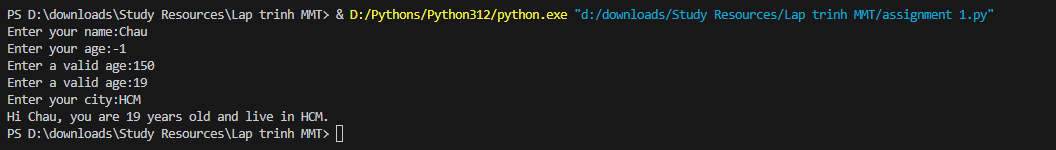
    age=int(input("Enter a valid age:"))

while age>110:

    age=int(input("Enter a valid age:"))

city= input("Enter your city:")

print(f"Hi {name}, you are {age} years old and live in {city}.")



Firstly, we have the name variable to input your name (as string). Using the while loop to confirm that the age is a positive integer and smaller than 110, if it is negative or larger than 101 print a warning and input again, the age is not an integer except ValueError with a warning and re-enter. Finally, input the city and print the requirements.

### Even or Odd Checker

Create a function is\_even(n) that returns True if a number is even. Ask the user for 5 numbers and use the function to check each one.

def is\_even(num):

    if num%2==0:

        print("True!")

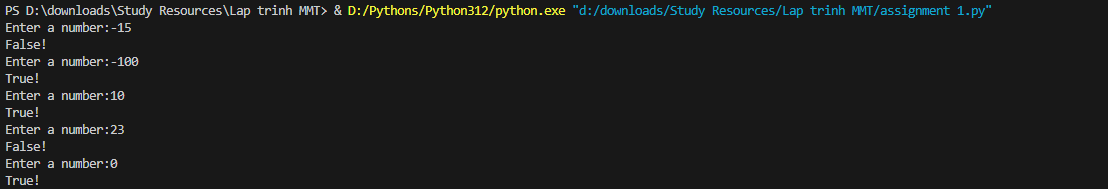
    else:

        print("False!")

for i in range(1,6):

    num= int(input("Enter a number:"))

    is\_even(num)



Create a function to is\_even to call in each loop to check if the value is even or not, if it is even return True and return False if not. Then use the for loop to enter 5 numbers, each loop will check if the number is even or odd.

### Simple Grading System

Write a function that takes a numeric score (0–100) and returns a grade:

A (≥90), B (≥80), C (≥70), D (≥60), F (<60)def grading(n):

score = int(input("Enter your score:"))

while score not in range(0, 101):

    score = int(input("Please enter a valid score (0-100):"))

if score>=90:

    grade="A"

elif score>=80:

    grade="B"

elif score>=70:

    grade="C"

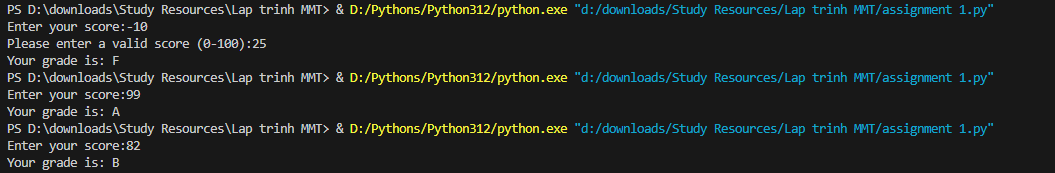
elif score>=60:

    grade="D"

else:

    grade="F"

print(f"Your grade is: {grade}")



Firsly, create a grading loop to check the score and return the specific character A/ B/ C/ D/ F depends on the input. Handling the input error by having a while loop to re-enter the score, if the user enters something that is not an integer, the program will print (“Invalid input. Please enter a number.”). Another case is that if your\_grade is more than 100 or less than 0 (invalid for scoring) then the user has to enter again, only when the value is correct (is an integer, from 0-100), the while loop is broken and print the final grade (call the grading function to get the score).

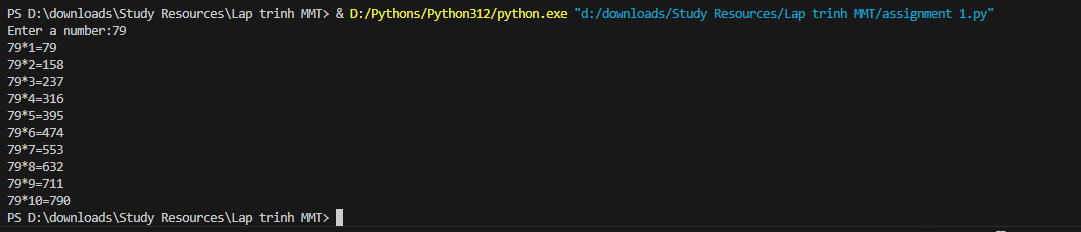
### Multiplication Table

Ask the user for a number and print its multiplication table (1 to 10).

n=int(input("Enter a number:"))

for i in range(1,11):

    print(f"{n}\*{i}={i\*n}")



Firstly, the code ask user to input a number and store it variable n, this function will print out "{n}\*{i}={i\*n}” while looping from 1 to 10, creating a multiplication table.

### Number Guessing Game

Create a game where the user has to guess a number between 1 and 100, with hints for “Too high” or “Too low.”

import random

r = random.randint(1,101)

n=0

while n!=r:

    n=int(input("Guess a number(1,100):"))

    if n>r:

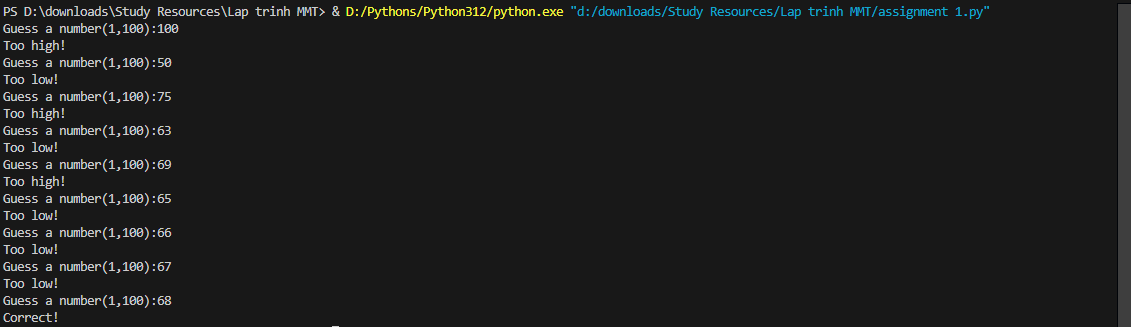
        print("Too high!")

    elif n<r:

        print("Too low!")

    else:

        print("Correct!")



The code above generates a random number between 1 and 100 using random.randint(1, 101). It requires the user to guess a number within this range and tracks the number of valid attempts. In a loop, it takes user input, converts it to an integer, and checks if it's equal to the random number or not. If the guess is outside the 1–100 range, it displays an error and ask the user to provide a valid one. If the guess is too low or too high, it provides feedback and continues the loop. When the correct number is guessed, it congratulates the user, and exits the loop.

Assignment 2: Data Structures & File I/O

### Score: Statistics

Ask the user to enter 5 test scores. Store them in a list and print:

* Highest score
* Average score
* List of all passing scores (≥60)

score = []

for i in range(0,5):

score.append(int(input("Enter a score:")))

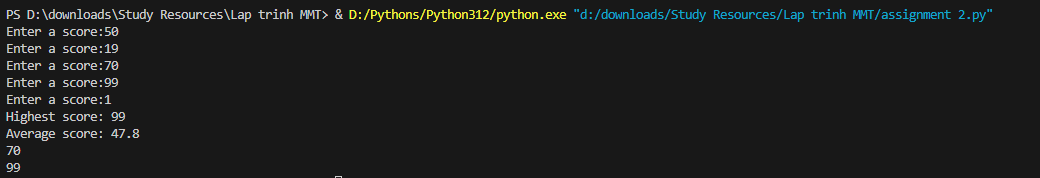
print(f"Highest score: {max(score)}")

print(f"Average score: {sum(score)/len(score)}")

for i in range(len(score)):

if score[i]>=60:

print(f"{score[i]}")



The provided code asks the user to enter five test scores from the user and stores them in a list called “score”. It uses a for loop to iterate five times, prompting the user to input a score for each test. Scores are appended to the list. After collecting all scores, the program calculates the highest score using max(score), computes the average by dividing the sum of scores by the list length (converted to a float), and identifies passing scores (≥60) by iterating through the list. Finally, it prints the highest score, average score, and all passing scores.

### Student: Directory

Input names and ages of 3 students and store them in a dictionary. Display the oldest student

student = {}

for i in range(1,4):

    name = input("Enter a student name:")

    while name in student:

        name=input("Name already exist, enter another one:")

    age = int(input("Enter the student's age:"))

    if age<0:

        age=int(input(("Enter a valid age:")))

    student[name]=age

n=0

m=""

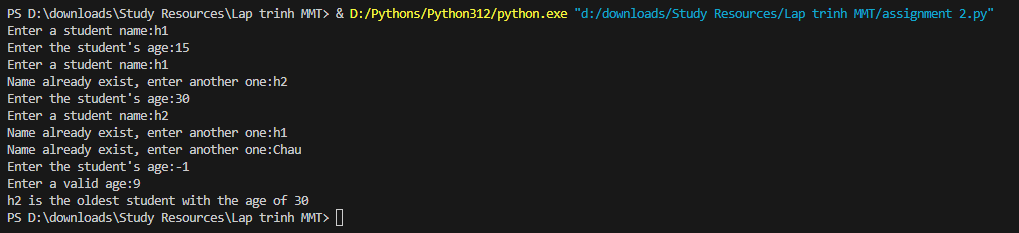
for name, age in student.items():

    if age>n:

        n=age

        m=name

print(f"{m} is the oldest student with the age of {n}")



First, an empty dictionary was initiated for later inputs, then use the for loop to enter the three students’ information (name and age). To be more specific, use the while loop to catch the error input in age (negative age or not an integer value), if the age is correctly input, break the while loop and continue to the next student.An additional code is used to check if the student name has already exist in the dictionary or not, if not existed in the dict, add the newest one into it, if it has already exist, raise an error and require the user to input another one. After those steps, check for the oldest student by using a simple sort loop to find the oldest student then assign the oldest age to the variable “n”, and the student name with the age is assigned to “m”. Finally, return the name and their age using print(f"{m} is the oldest student with the age of {n}")

### Word Frequency Counter

* Read a paragraph from a file named input.txt
* Count how many times each word appears
* Save the result in output.txt as a dictionary

with open("input.txt", "r", encoding=”utf-8”) as file:

    para=file.read()

word=para.split()

dict= {}

for i in range(len(word)):

    if word[i] in dict:

        dict[word[i]] +=1

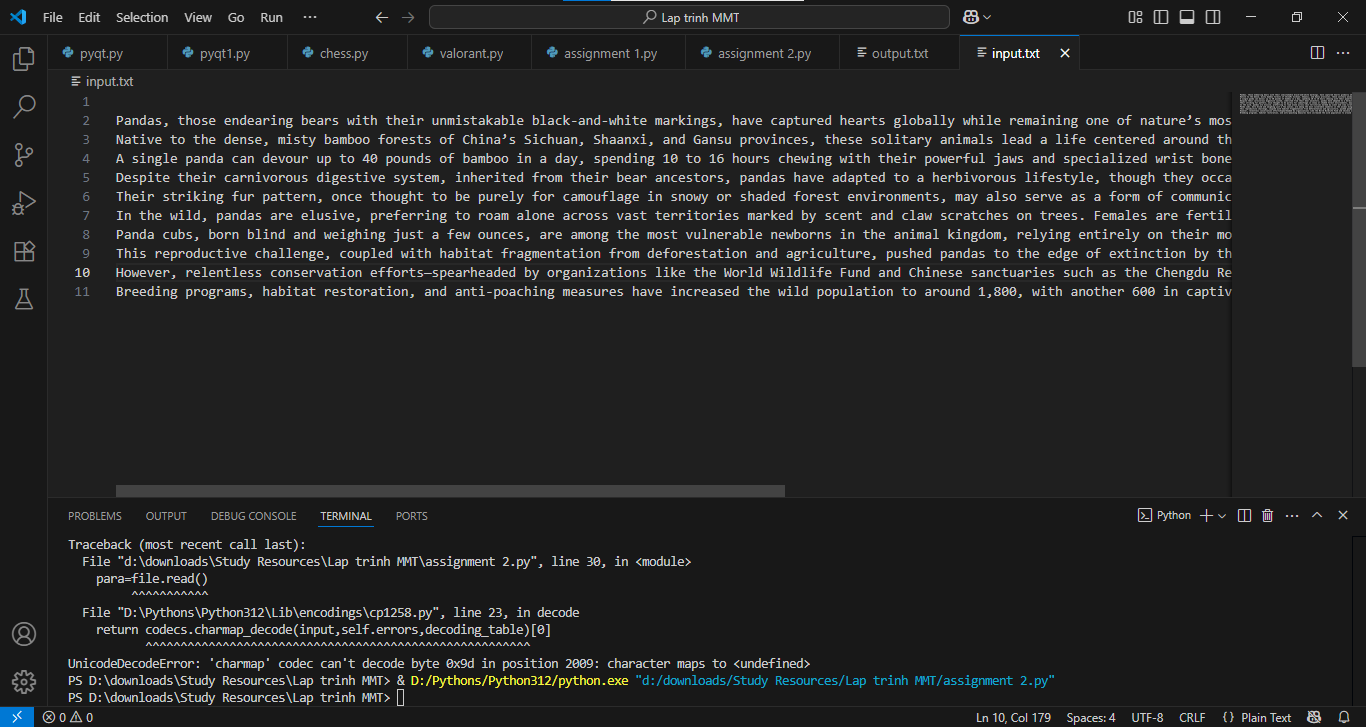
    else:

        dict[word[i]]=1

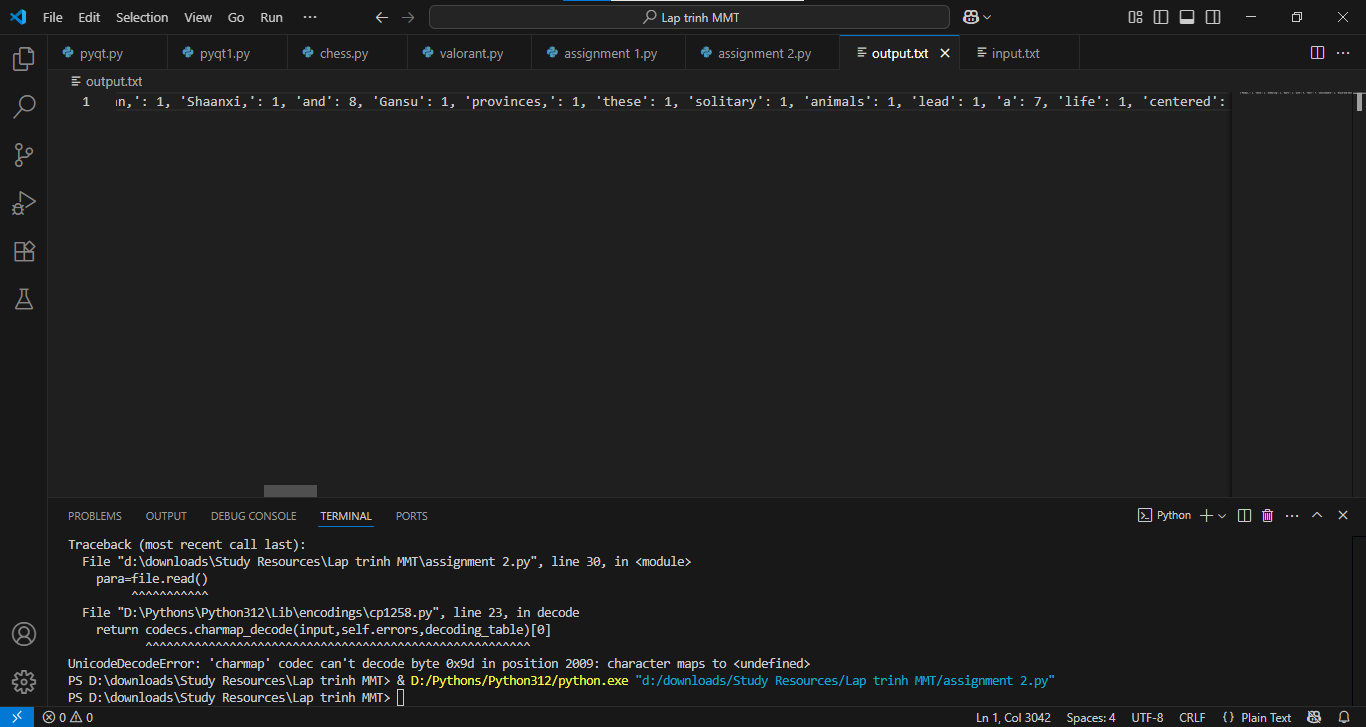
with open("output.txt", "w", encoding=”utf-8”) as file:

    file.write(f"{dict}")

Input file content:



Output:



Having the text as the input for the user to enter the paragraph (is put in the input.txt). After entering the paragraph, create a list named word to split the paragraph into individual words without space. The next step is to count the frequency of each word, if the word is in the list word, plus one for that word value, if it is not in the list word then add the word and set its value to 1. Finally, store the dictionary in output.txt.

### Tuple: Conversion

Given a list of integers, convert it into a tuple and print only the unique even numbers in it

integers= []

n=int(input("Enter the amount of integers in the list:"))

for i in range(0,n):

    integers.append(int(input("Enter the integers:")))

tup=tuple(integers)

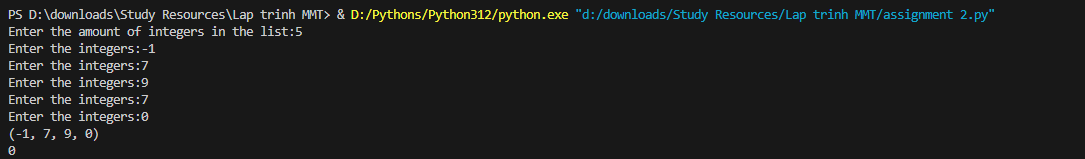
tup1=tuple(dict.fromkeys(tup))

print(tup1)

for i in range(len(tup1)):

    if tup1[i]%2==0:

        print(tup1[i])



Program has a list named integers to store the integers that user inputs, after that, use a for loop to prompt the user to put in the integers. After entering all the numbers, transform the list integers into a tuple. Use the built-in function tup1= tuple(dict.fromkeys(tup)) to extract numbers in the tuple and remove those are repeated in it. Finally check if elements in tup1 if they’re even or not, if it is even then print the number to the terminal, if not then skip to the next one until reach the end of the tuple tup1.

Assignment 3: Classes, Objects, and Inheritance

### Class: Rectangle

Create a class Rectangle with attributes length and width, and methods:

* area() → return area
* perimeter() → return perimeter

class Rectangle:

    def \_\_init\_\_(self, length, width):

        self.length=length

        self.width=width

    def area(self):

        return self.length\*self.width

    def perimeter(self):

        return (self.length+self.width)\*2

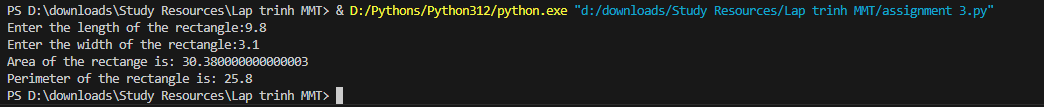
length=float(input("Enter the length of the rectangle:"))

width=float(input("Enter the width of the rectangle:"))

rec=Rectangle(length, width)

print(f"Area of the rectange is: {rec.area()}")

print(f"Perimeter of the rectangle is: {rec.perimeter()}")



Create a class Rectangle to set the width and length of the rectangle, two additional functions are calculating the area and perimeters of the rectangle. After that, asks the user to input the length and width of the rectangle and call all the functions in the Rectangle class for the wanted output.

### Class: BankAccount

* Attributes: account\_number, owner, balance
* Methods:

▪ deposit(amount)

▪ withdraw(amount)

▪ display\_balance()

import random

class BankAccount:

    def \_\_init\_\_(self, account\_number, owner, balance):

        self.account\_number=account\_number

        self.owner=owner

        self.balance=balance

    def deposit(self, amount):

        while amount<=0:

            amount=int(input("Enter a valid amount:"))

        self.balance+=amount

        return self.balance

    def withdraw(self, amount):

        while amount<=0:

            amount=int(input("Enter a valid amount:"))

        if amount>self.balance:

            amount=int(input("Enter a valid amount:"))

        self.balance-=amount

        return self.balance

    def display\_balance(self):

        print(f"Account number: {self.account\_number}")

        print(f"Owner: {self.owner}")

        print(f"Current Balance: {self.balance} VND")

balance=1000000

owner=input("Enter the owner of the account:")

account\_number=19000000+random.randint(0,1000)

print(f"Generated account number:{account\_number}")

BA=BankAccount(account\_number,owner, balance)

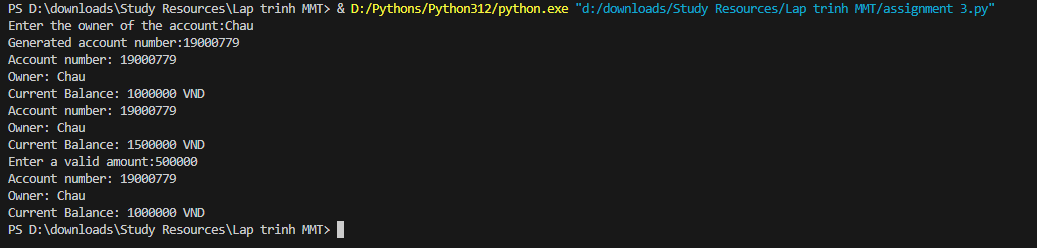
BA.display\_balance()

BA.deposit(500000)

BA.display\_balance()

BA.withdraw(1600000)

BA.display\_balance()



The Bank class manages a bank account with attributes for account number, owner, and balance, which is initialized to 1000000 by default. It includes methods to deposit positive amounts, withdraw valid amounts (non-negative and within balance), set an initial balance via user input, and display the current balance. The program prompts the user for an owner name, and provides a Account number using random.randint(0,1000), creates a Bank object, and sets the initial balance. In the example, I entered my name as Owner, the code automatically initialize the balance to 1000000, and I deposited 500000, after that display the current balance, withdraw 1600000 which is an invalid amount since it is larger than my current balance so it prompted me to enter another number. After that, display the final balance after those transactions.

Inheritance: Employee System

* Base class: Person (name, age)
* Derived class: Employee (employee\_id, position, salary)
* Override display\_info() to show all fields

import random

class Person:

    def \_\_init\_\_(self, name, age):

        self.name=name

        self.age=age

    def display\_info(self):

        print(f"Person's name: {self.name}")

        print(f"Person's age: {self.age}")

class Employee(Person):

    def \_\_init\_\_(self, name, age, employee\_id, position, salary):

        super().\_\_init\_\_(name, age)

        self.employee\_id=employee\_id

        self.position=position

        self.salary=salary

    def display\_info(self):

        print(f"Employee's name: {self.name}")

        print(f"Employee's age: {self.age}")

        print(f"Employee's ID: {self.employee\_id}")

        print(f"Employee's position: {self.position}")

        print(f"Employee's salary: {self.salary}")

employee\_id=19000000+random.randint(0,1000)

me=Person("Chau", 19)

employee=Employee("Chau", 19, employee\_id, "HR", 1000)

me.display\_info()

employee.display\_info()



The Person class defines attributes for name and age, with a display\_info method to print them. The Employee class inherits from Person, adding employee\_id, position, and salary attributes. Its display\_info method overrides the parent’s, calling super().display\_info() to show name and age, then appending employee-specific details. All details are printed via display\_info.

### Mini Project: Library Management

* Classes:
* Book: title, author, ISBN
* Member: name, ID
* Library: list of books, list of members
* Methods:
* Add book/member
* Borrow/return book
* Store all transactions in a file library\_log.txt

class Book:

    def \_\_init\_\_(self, title, author, ISBN,amount):

        self.title = title

        self.author = author

        self.ISBN = ISBN

        self.amount = amount

    def display\_info(self):

        print(f"Title: {self.title}\nAuthor: {self.author}\nISBN: {self.ISBN}\nAmounts: {self.amount}")

class Member:

    def \_\_init\_\_(self, name, id):

        self.name = name

        self.id = id

        self.books\_borrowed = []

    def display\_info(self):

        print(f"Name: {self.name}\nMember ID: {self.id}")

        if self.books\_borrowed:

            print("Borrowed books:",  ", ".join([book.title for book in self.books\_borrowed]))

        else:

            print("No books borrowed")

class Library:

    def \_\_init\_\_(self):

        self.books = []

        self.members = []

        self.history = "library\_log.txt"

    def add\_book(self):

        while True:

            try:

                title=input("\nEnter your new book title: ")

                author=input("Enter the author of the new book: ")

                ISBN=input("Enter the ISBN of your book (13 digits): ")

                amount=int(input("Enter the amount you want to add: "))

                if amount<=0:

                    print("You need to add at least 1 book if you want to donate")

                elif checking\_input(title)==False:

                    print("Title can not be empty")

                elif checking\_input(author)==False:

                    print("Author name can not be empty")

                elif len(ISBN)!=13:

                    print("ISBN must have 13 digits")

                elif len(ISBN)==13:

                    new\_book=Book(title,author,ISBN,amount)

                    if ISBN\_checker(ISBN)==False: #having the same book or same ISBN?

                        for book in self.books: #having same book check

                                if book.title==new\_book.title:

                                    book.amount+=amount

                                    print(f"The library has {book.amount} {book.title} book(s)")

                                    with open(self.history, 'a') as f:

                                        f.write(f"Added {amount} copy(ies) of '{title}' (ISBN: {ISBN})\n")

                                    return

                        print("You can not have the same ISBN for multiple books")

                        continue

                    else:

                        self.books.append(new\_book)

                        print(f"Book '{new\_book.title}' added to the library.")

                        with open(self.history, 'a') as f:

                            f.write(f"Added {amount} copy(ies) of '{title}' (ISBN: {ISBN})\n")

                        break

            except ValueError:

                print("You have to enter a positive amonut of books")

    def borrow\_book(self):

        member\_id = input("\nEnter your member ID (6 digits): ")

        member = None

        for m in self.members:

            if m.id == member\_id:

                member = m

                break

        if not member:

            print("Member not found.")

            return

        book\_title = input("Enter the book title to borrow: ")

        for book in self.books:

            if book.title == book\_title and book.amount > 0:

                member.books\_borrowed.append(book)

                book.amount -= 1

                print(f"{member.name} borrowed '{book.title}'.")

                with open(self.history, 'a') as f:

                    f.write(f"{member.name} borrowed '{book.title}'\n")

                return

        print(f"Book '{book\_title}' is not available.")

    def return\_book(self):

        member\_id = input("\nEnter your member ID: ")

        book\_title = input("Enter the book title to return: ")

        # find the member

        member = None

        for m in self.members:

            if m.id == member\_id:

                member = m

                break

        if not member:

            print("Member not found.")

            return

        # Find the book in member's borrowed list

        for book in member.books\_borrowed:

            if book.title == book\_title:

                member.books\_borrowed.remove(book)

                for library\_book in self.books:

                    if library\_book.title == book\_title:

                        library\_book.amount += 1

                        print(f"'{book\_title}' returned by {member.name}.")

                        with open(self.history, 'a') as f:

                            f.write(f"{member.name} returned '{book\_title}'\n")

                        return

        print(f"{member.name} does not have the book '{book\_title}' borrowed.")

    def add\_member(self):

        while True:

            name = input("\nEnter member name: ")

            if checking\_input(name) == False:

                print("Name can not be empty")

                continue

            member\_id = input("Enter your ID (6 digits): ")

            if not member\_id.isdigit() or len(member\_id) != 6:

                print("ID must have exactly 6 digits (numbers)\n")

                return

            for member in self.members: #Checkingg if id is the same

                if member.id == member\_id:

                    print("This ID is already in use\n")

                    return

            new\_member = Member(name, member\_id)

            self.members.append(new\_member)

            print(f"Member '{name}' added to the library.")

            with open(self.history, 'a') as f:

                f.write(f"Added member '{name}' (ID: {member\_id})\n")

            break

    def display\_library(self):

        print("\nLibrary Books:\n")

        if not self.books:

            print("No books in library")

        for book in self.books:

            book.display\_info()

            print()

        print("Library Members:\n")

        if not self.members:

            print("No members in library")

        for member in self.members:

            member.display\_info()

            print()

def checking\_input(string):

    checker=string.strip()

    if not checker:

        return False

    else:

        return True

def ISBN\_checker(ISBN\_current):

    for book\_current in MyLibrary.books:

        if book\_current.ISBN==ISBN\_current:

            return False

        else:

            return True

MyLibrary=Library()

while True:

    try:

        gen\_number\_of\_books=int(input("\nEnter the number of existing books in the library: "))

        if gen\_number\_of\_books<=0:

            print("You have to enter a positive number of books")

            continue

        break

    except ValueError:

            print("Please enter a positive number of books")

for i in range (gen\_number\_of\_books):

    MyLibrary.add\_book()

while True:

    print("\nWelcome to my library: ")

    print("1. Add a new book")

    print("2. Borrow a book")

    print("3. Return a book")

    print("4. Add a new member")

    print("5. Display current library")

    print("6. Exit")

    try:

        choice = int(input("\nEnter your choice (1-6): "))

        if choice == 1:

            MyLibrary.add\_book()

        elif choice == 2:

            MyLibrary.borrow\_book()

        elif choice == 3:

            MyLibrary.return\_book()

        elif choice == 4:

            MyLibrary.add\_member()

        elif choice == 5:

            MyLibrary.display\_library()

        elif choice == 6:

            print("See you again")

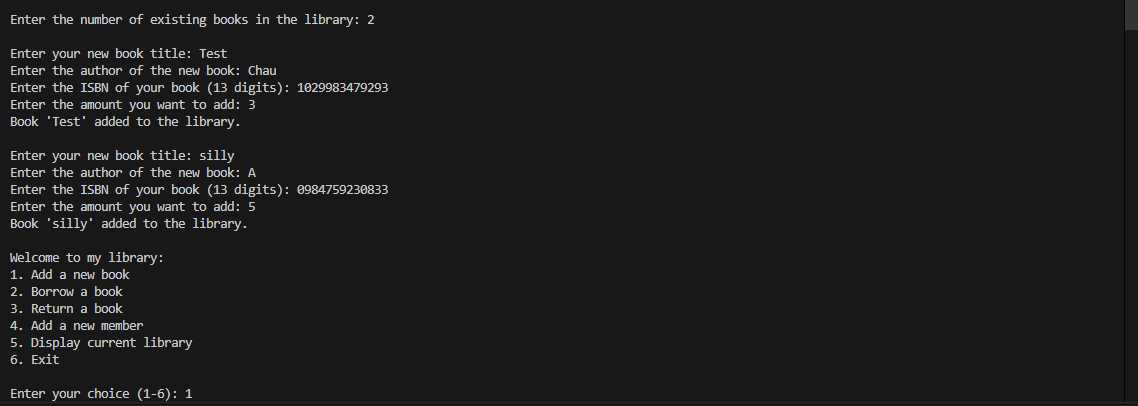
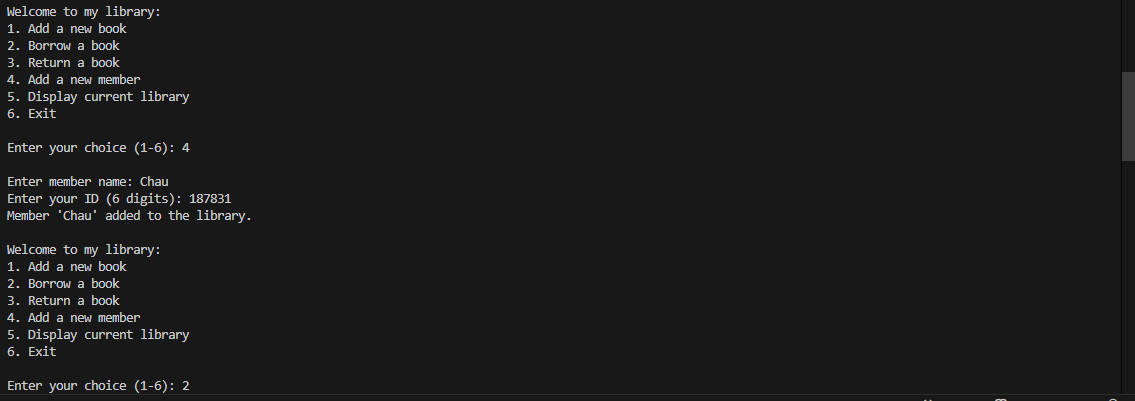
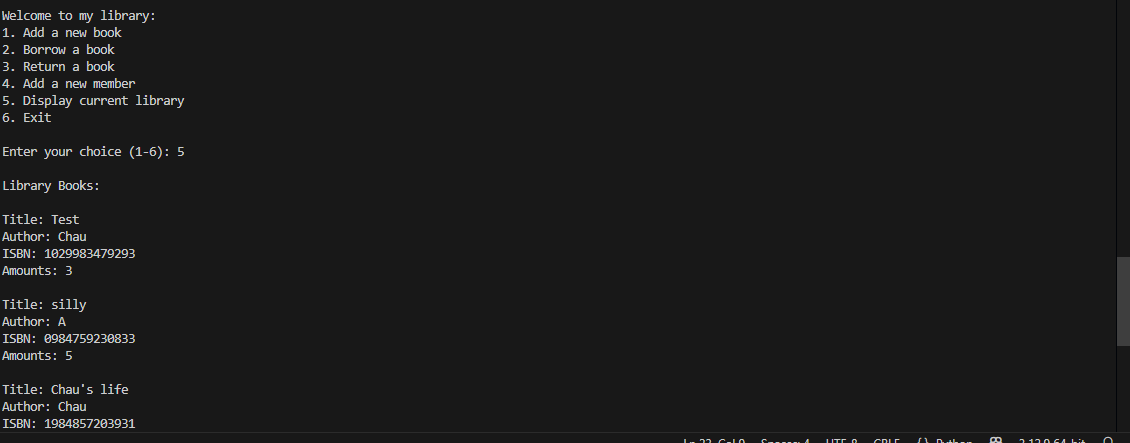
            break

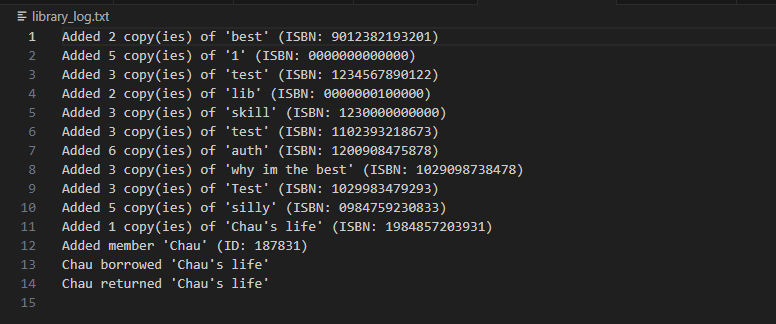
        else:

            print("Invalid choice. Please enter a number between 1 and 6.")

    except ValueError:

        print("Please enter a valid number")



First and foremost, creating a class Book which has the title, author, ISBN (13 digits), and the number of books there is. The class Member has attributes about name, id and a list of borrowed books that they have. Each of these classes also have a display function. Next step is to make a class Library, since the library will be the place where people perform most of the actions, so the library class will hold all the functions. The basic attributes are the books list, member list and a log to register all the actions in this file (library\_log.txt). There are many functions in this class:

* add\_book: uses a while loop to collect a book’s title, author, ISBN (13 digits), and amount from the user. A try-except block catches ValueError for non-integer amount inputs (e.g., entering a string instead of a number), printing "You have to enter a positive amount of books" and prompting the user to re-enter all attributes. The if and elif statements handle logical errors: if amount ≤ 0, it prints "You need to add at least 1 book if you want to donate"; if checking\_input (a function that checks if a string is empty/None, returning False, or valid, returning True) finds an empty title or author, it prints "Title can not be empty" or "Author name can not be empty"; if ISBN isn’t exactly 13 digits, it prints "ISBN must have 13 digits." The ISBN\_checker function ensures no duplicate ISBNs exist; if found, it either adds amount to an existing book with the same title or prints "You can not have the same ISBN for many books" and continues the loop. Valid inputs create a Book object, append it to self.books, log the addition to library\_log.txt, and break the loop.
* borrow\_book: inputs the member\_id (6 digits to add the wanted book to borrow to the book\_borrowed list in that specific member object) and check if the entered member is in the member list of the library, if not, print “Member not found” and exit. The next part is to enter the book title and search for it in the list of books in the library. If found, the book is appended to the member’s books\_borrowed list, the book’s amount is decremented, the transaction is logged in library\_log.txt, and a success message is printed. If the book isn’t available or doesn’t exist, it prints out a notice that the book is not available.
* return\_book: requests a member\_id and book\_title from the user. It searches self.members for a matching member; if not found, it prints "Member not found." and returns to the menu. If the member exists, it checks the member’s books\_borrowed list for the specified book\_title. If the book is found, it’s removed the book from books\_borrowed of the member, the corresponding book’s amount in self.books is incremented, the return is logged in library\_log.txt, and a success message is printed. If the book isn’t in the member’s borrowed list, it returns a warning message that the following user does not borrow the book
* add\_member: uses a while loop to collect a name and the id of a new member. The checking\_input function ensures the name isn’t empty, and continuing the loop if invalid. For member\_id, it checks if the input is exactly 6 digits using isdigit() and length validation; if not, it prints "ID must have exactly 6 digits (numbers)" and exits back to the menu. It then scans self.members for duplicate IDs, returning to the menu if found duplicate. Valid inputs create a Member object, append it to self.members, log the addition in library\_log.txt, and break the loop.
* display\_library: showing all the books in list of books and members in the list of members

Run the program by giving the MyLibrary as a Library class object, the while loop is used to check for any error input while getting the number of books which is already in the library (the library must have books in it already). After that use a for loop to enter the books’ attributes by calling the add\_book function in the Library class. Finally, the while loop activates the menu and allow the user to choose the wanted action.

# TÀI LIỆU THAM KHẢO