**SAVITRIBAI PHULE PUNE UNIVERSITY  
(Formely University of Pune)**

**JSPM’s JAYAWANTRAO SAWANT COLLEGE OF**

**ENGINEERING, PUNE-33**

**M.C.A. - I**

**PYTHON LAB ASSIGNMENT**

**Semester 1**

|  |  |
| --- | --- |
| **Name of Student:** | **Manish Shetty** |
| **Roll No.** | **(H) 2560** |
| **Email:** | **manishnshetty77@gmail.com** |
| **Phone:** | **8208618905** |

**Subject Teacher/ GFM HOD**

**Academic Year 2025-27**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Assignment -2** | **Date** | **Sign** |
| 1. | 2.1 Function Basics – Scope, Nested Function, Non-local Statements  Case Study-1: Student Grading System |  |  |
| 2. | 2.2 Built-in Functions  Case Study-2: E-commerce Order Summary |  |  |
| 3. | 2.3 Topic: Types of Functions, Anonymous Function  Case Study-3: Student Data Management |  |  |
| 4. | 2.4 Decorators and Generators  Case Study-4: Banking Transactions |  |  |
| 5. | 2.5 Topic: Modules – Basic Usage, Creating & Importing Modules  Case Study-5: Scientific Calculator |  |  |
| 6. | 2.6 Importing Functions and Variables from Different Modules  Case Study-6: Travel Expense Calculator |  |  |
| 7. | 2.7 Python Built-in Modules – math, random, datetime  Case Study-7: Lucky Draw Contest |  |  |
| 8. | 2.8 Package: Import Basics  Case Study-8: Library Management System |  |  |
| 9. | 2.9**:** Python Namespace Packages  Case Study-9: University Departments System |  |  |
| 10. | 2.10 User Defined Modules and Packages  Case Study-10: Hospital Management System |  |  |
| 11. | 2.11.1 Avoiding Code Break using Exception Handling  **Case Study**: Online Payment Processing  2.11.2 Safeguarding File Operation using Exception Handling  **Case Study**: Student Record File System  2.11.3 Handling Multiple and User Defined Exceptions  **Case Study**: ATM Withdrawal  2.11.4 Handling and Helping Developer with Error Code  **Case Study**: Web Application Debugging  2.11.5 Programming using Exception Handling  **Case Study**: Web Application Debugging |  |  |

**Assignment No-2**

**Subject:** Python Programming (PP) **Subject Teacher:** Prof. Leena Deshmukh HOD

**Topic:** Python Lab Assignment-2

**Name:** Manish Narayan Shetty **Roll No:** (H25)60 **Class:** MCA – Semester-I

**College Name:** JSPM’s Jayawantrao Sawant College of Engineering (JSCOE)

**Submitted To:** **Submitted Date:**

**Case Study-1: Student Grading System** **Topic: Function Basics – Scope, Nested Function, Non-local Statements**

* Create a function calculate\_grade() that takes marks as input.
* Inside it, define a nested function grade\_category() that assigns a grade.
* Use a nonlocal variable to update and print remarks from the nested function (e.g., “Pass with Distinction”).

**Program/Code:**

def calculate\_grade(marks):

remarks = "No remarks available."

def grade\_category():

nonlocal remarks

if marks >= 75:

grade = 'A'

remarks = "Pass with Distiniction"

elif marks >= 60:

grade = 'B'

remarks = "First Class"

elif marks >= 50:

grade = 'C'

remarks = "Second Class"

elif marks >= 40:

grade = 'D'

remarks = "Pass"

else:

grade = 'F'

remarks = "Fail"

return grade

final\_grade = grade\_category()

print("Marks: ",marks)

print("Grade: ",final\_grade)

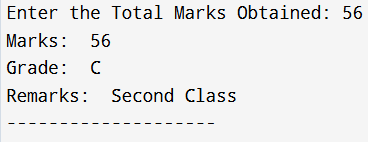
print("Remarks: ",remarks)

print("-" \* 20)

marks = int(input("Enter the Total Marks Obtained: "))

calculate\_grade(marks)

**Output:**



**Case Study-2:** **E-commerce Order Summary**  **Topic: Built-in Functions**

* Use built-in functions like len(), max(), min(), sum(), sorted() to:
* Find the total number of orders.
* Calculate the highest, lowest, and average order value.
* Sort the order values in ascending order.

**Program/Code:**

def summarize\_orders(orders):

print("--- E-commerce Order Summary ---")

if not orders:

print("No orders to summarize.")

return

total\_orders = len(orders)

print("Total number of orders: ",total\_orders)

highest\_order = max(orders)

lowest\_order = min(orders)

print("Highest order value: ",highest\_order)

print("Lowest order value: ",lowest\_order)

total\_value = sum(orders)

average\_order = total\_value / total\_orders

print("Average order value: ",average\_order)

sorted\_orders = sorted(orders)

print("\nOrders sorted (ascending):")

value = [values for values in sorted\_orders]

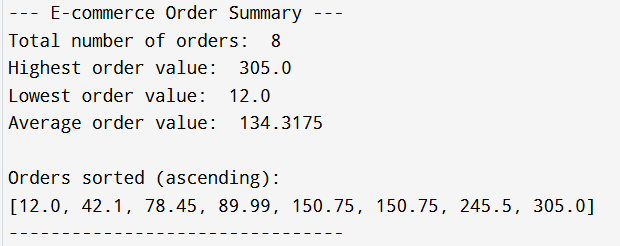
print(value)

print("-" \* 32)

order\_values = [150.75, 89.99, 245.50, 42.10, 150.75, 12.00, 305.00, 78.45]

summarize\_orders(order\_values)

**Output:**

****

**Case Study-3: Employee Salary Slip**  **Topic: Types of Functions, Anonymous Function (lambda)**

* Define a normal function to calculate HRA, DA, PF.
* Define a lambda function to calculate net salary.
* Show difference between user-defined and lambda function usage.

**Program/Code:**

def calculate\_allowances\_deductions(basic\_salary):

hra\_percent = 0.30 # 30% of basic

da\_percent = 0.20 # 20% of basic

pf\_percent = 0.12 # 12% of basic

hra = basic\_salary \* hra\_percent

da = basic\_salary \* da\_percent

pf = basic\_salary \* pf\_percent

return hra, da, pf

employee\_basic\_salary = int(input("Enter Employee Salary: "))

hra\_amount, da\_amount, pf\_amount = calculate\_allowances\_deductions(employee\_basic\_salary)

calculate\_net\_salary = lambda basic, hra, da, pf: basic + hra + da - pf

net\_salary = calculate\_net\_salary(employee\_basic\_salary, hra\_amount, da\_amount, pf\_amount)

print("--- Employee Salary Slip ---")

print("Basic Salary : ",employee\_basic\_salary)

print("HRA (30%) : ",hra\_amount)

print("DA (20%) : ",da\_amount)

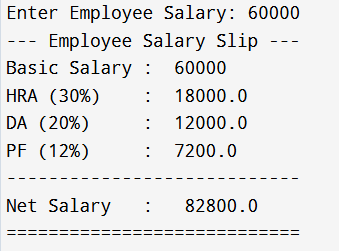
print("PF (12%) : ",pf\_amount)

print("-" \* 28)

print("Net Salary : ",net\_salary)

print("=" \* 28)

**Output:**

****

**Case Study 4: Online Banking System**  **Topic: Decorators and Generators**

* Create a decorator @authenticate\_user to check if a user is logged in before performing a transaction.
* Implement a generator function transaction\_history() that yields last 5 transactions of a user.

**Program/Code:**

# --- Decorator to check authentication ---

def authenticate\_user(func):

def wrapper(user, \*args, \*\*kwargs):

if not user.get("is\_logged\_in"):

print("Access denied. Please log in first.")

return

return func(user, \*args, \*\*kwargs)

return wrapper

# --- Generator function for last 5 transactions ---

def transaction\_history(transactions):

# Yield last 5 transactions

for txn in transactions[-5:]:

yield txn

# --- Function to perform a transaction (uses decorator) ---

@authenticate\_user

def perform\_transaction(user, amount):

print("Transaction of ₹ ”,amount,” successful for user: “,user['name'])

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

# User data

user = {"name": "Manish", "is\_logged\_in": True}

# Sample transaction list

transactions = [500, -200, 1000, -150, 300, -400, 700]

# Perform a transaction

perform\_transaction(user, 250)

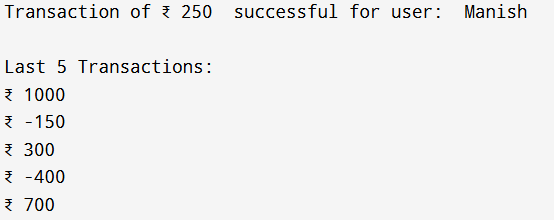
# Display last 5 transactions using generator

print("\nLast 5 Transactions:")

for txn in transaction\_history(transactions):

print("₹”, txn)

**Output:**

****

**Case Study 5: Scientific Calculator Topic: Modules – Basic Usage, Creating & Importing Modules**

* Create a module calculator.py with functions add, subtract, multiply, divide.
* Import the module in another file main.py and perform user-chosen operations.

**Program/Code:**

**Calculator.py (Module) Code:**

def add(a,b):

return a + b

def sub(a,b):

return a - b

def mul(a,b):

return a \* b

def div(a,b):

if b == 0:

print("ERROR: Cannot divide by Zero!")

retursn a / b

**main.py Code:**

import calculator as cal

print("-"\*25)

print("1. ADD\n2. Substract\n3. Multiply\n4. Divivde\n")

choice = int(input("Choose Operation from above by selecting the Option Number: "))

num1 = float(input("Enter Value 1: "))

num2 = float(input("Enter Value 2: "))

if choice == 1:

print("Addition of ",num1," & ",num2," is: ",cal.add(num1,num2))

elif choice == 2:

print("Substraction of ",num1," & ",num2," is: ",cal.sub(num1,num2))

elif choice == 3:

print("Multiplication of ",num1," & ",num2," is: ",cal.mul(num1,num2))

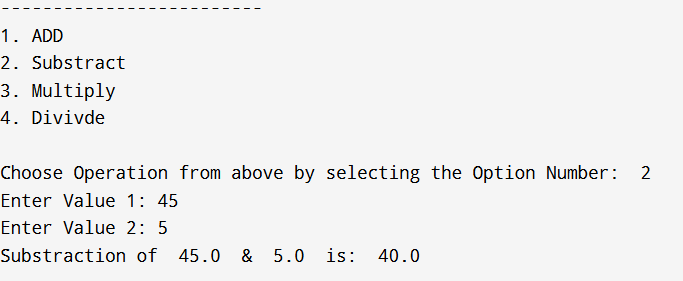
elif choice == 4:

print("Division of ",num1," & ",num2," is: ",cal.div(num1,num2))

else:

print("Enter Valid Choice!")

**Output:**

****

**Case Study 6: Travel Expense Calculator**

**Topic: Importing Functions and Variables from Different Modules**

* Create two modules:
* transport.py → contains bus/train/flight fare functions.
* stay.py → contains hotel/food expense functions.
* Import specific functions into main.py to calculate total trip cost.

**Program/Code:**

**Transport.py code:**

def get\_bus\_fare(distance\_km):

return distance\_km \* 0.5 # $0.5 per km

def get\_train\_fare(distance\_km):

rate = 0.75

return distance\_km \* rate

def get\_flight\_fare(destination):

fares = {'Pune': 5300, 'Mumbai': 3450, 'Mangalore': 7350}

return fares.get(destination, 500) # Default fare if not found

**stay.py Code:**

def calculate\_hotel\_cost(nights):

nightly\_rate = 2000

return nights \* nightly\_rate

def calculate\_food\_expense(days):

avg\_daily\_cost = 500

return days \* avg\_daily\_cost

**main.py Code:**

import transport as t

from stay import calculate\_hotel\_cost, calculate\_food\_expense

print("Enter your Travel and stay Details: ")

dest = input("Enter Destination Name: ")

distance = int(input("Enter the Total Distance from your Current Location to destination in km: "))

mode = input("Enter the mode of your transport: ")

nights = int(input("Enter Total no of nights you will stay: "))

days = int(input("Enter Total no of Days you will stay: "))

travel\_cost = 0

if mode == "flight":

travel\_cost = t.get\_flight\_fare(dest)

elif mode == "bus":

travel\_cost = t.get\_bus\_fare(distance)

elif mode == "train":

travel\_cost = t.get\_train\_fare(distance)

else:

print("Enter Correct mode of Transport!")

hotel\_cost = calculate\_hotel\_cost(nights)

food\_cost = calculate\_food\_expense(days)

total\_trip\_cost = travel\_cost + hotel\_cost + food\_cost

# --- Display Results ---

print("Travel Cost to ",dest," : ",travel\_cost)

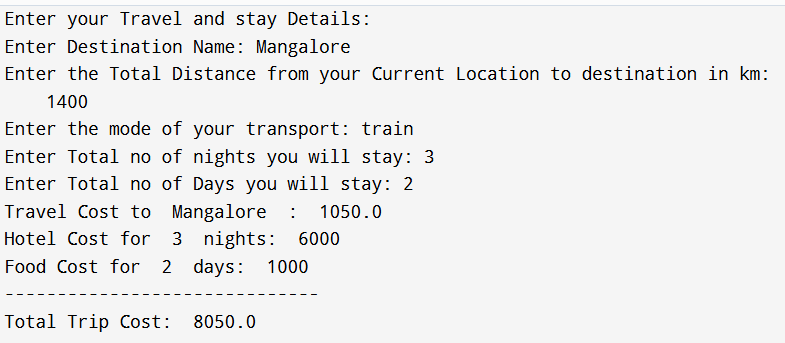
print("Hotel Cost for ",nights," nights: ",hotel\_cost)

print("Food Cost for ",days," days: ",food\_cost)

print("-" \* 30)

print("Total Trip Cost: ",total\_trip\_cost)

**Output:**

****

**Case Study 7: Lucky Draw Contest Topic: Python Built-in Modules – math, random, datetime**

* Use random to pick 5 winners from a list of participants.
* Use datetime to print the date & time of draw.
* Use math to calculate prize money distribution with percentage-based bonus.

**Program/Code:**

import random

import datetime

import math

participants = ["Manish", "Ganesh", "Deepak", "Kaushik", "Rohan", "Kaushal", "Shrinivas", "Maithili", "Rutuja"]

total\_prize\_pool = 50000

print("--- Lucky Draw Contest ---")

num\_winners = 5

winners = random.sample(participants, num\_winners)

print("The ",num\_winners," Winners are: ",winners)

now = datetime.datetime.now()

print("Draw Date & Time: ",now.strftime('%Y-%m-%d %H:%M:%S'))

base\_prize = total\_prize\_pool / num\_winners

bonus\_percentages = [0.40, 0.30, 0.10, 0.10, 0.10]

prizes = []

print("\n--- Prize Distribution ---")

for i, winner in enumerate(winners):

bonus\_amount = base\_prize \* bonus\_percentages[i]

total\_prize = math.ceil(base\_prize + bonus\_amount)

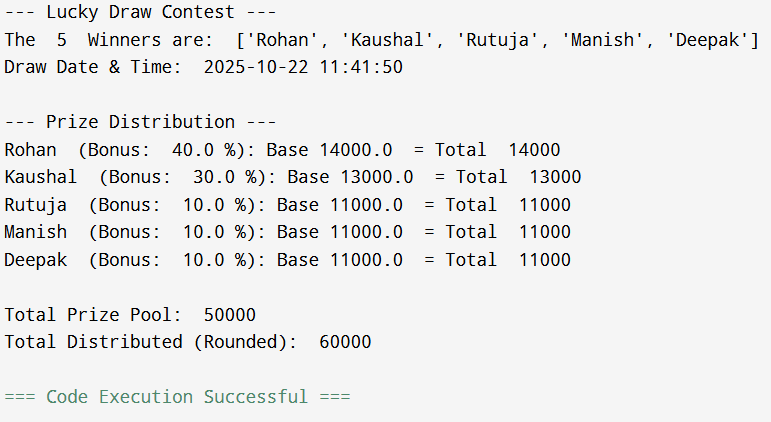
prizes.append(total\_prize)

print(winner," (Bonus: ",bonus\_percentages[i]\*100,"%): Base", base\_prize + bonus\_amount," = Total ",total\_prize)

print("\nTotal Prize Pool: ",total\_prize\_pool)

print("Total Distributed (Rounded): ",sum(prizes))

**Output:**

****

**Case Study 8: Library Management System Topic: Package: Import Basics**

* Create a package library with modules:
* books.py → add, remove, search books.
* members.py → register and update members.
* Import these modules inside main.py to simulate library operations.

**Program/Code:**

**# library/books.py**

book\_list = []

def add\_book(title, author):

book = {"title": title, "author": author, "status": "Available"}

book\_list.append(book)

print("Book added: ",title," by ",author)

def remove\_book(title):

global book\_list

original\_len = len(book\_list)

book\_list = [b for b in book\_list if b['title'].lower() != title.lower()]

if len(book\_list) < original\_len:

print("Book removed: ",title)

else:

print("Book not found: ",title)

def search\_book(title):

results = [b for b in book\_list if title.lower() in b['title'].lower()]

print("--- Search Results for ",title," ---")

if results:

for book in results:

print("Title: ",book['title']," Author: ",book['author']," Status: ",book['status'])

else:

print("No books found matching the search criteria.")

**# library/members.py**

member\_list = {}

member\_id\_counter = 1001

def register\_member(name):

global member\_id\_counter

member\_list[member\_id\_counter] = {"name": name, "status": "Active"}

print("Member registered: ",name," with ID ",member\_id\_counter)

member\_id\_counter += 1

return member\_id\_counter - 1

def update\_member\_status(member\_id, new\_status):

if member\_id in member\_list:

member\_list[member\_id]['status'] = new\_status

print("Member ",member\_id," status updated to ",new\_status)

else:

print("Error: Member ID ",member\_id," not found.")

def get\_member\_details(member\_id):

return member\_list.get(member\_id, "Member not found.")

**main.py Code:**

import library.books

import library.members

def simulate\_library\_operations():

print("\n--- Welcome to the Library System ---")

library.books.add\_book("Python Crash Course", "Eric Matthes")

library.books.add\_book("The Lord of the Rings", "J.R.R. Tolkien")

library.members.register\_member("System Admin")

print("\n(System initialized with a few books and one member.)")

print("-" \* 40)

while True:

print("\n--- Main Menu ---")

print("1. Add a Book")

print("2. Search for a Book")

print("3. Register a New Member")

print("4. Update Member Status (e.g., Suspended)")

print("5. Display All Books and Members")

print("6. Exit")

choice = input("Enter your choice (1-6): ").strip()

if choice == '1':

title = input("Enter the book title: ").strip()

author = input("Enter the author's name: ").strip()

if title and author:

library.books.add\_book(title, author)

else:

print("Title and Author cannot be empty.")

elif choice == '2':

search\_term = input("Enter title/keyword to search: ").strip()

library.books.search\_book(search\_term)

elif choice == '3':

name = input("Enter the new member's name: ").strip()

if name:

library.members.register\_member(name)

else:

print("Member name cannot be empty.")

elif choice == '4':

if not library.members.member\_list:

print("No members registered yet.")

continue

print("\nCurrent Members:")

for mid, details in library.members.member\_list.items():

print(" ID: ",mid," - Name: ",details['name']," (Status: ",details['status'],")")

try:

member\_id = int(input("Enter the Member ID to update: ").strip())

new\_status = input("Enter the new status (e.g., Active, Suspended, Lost): ").strip()

library.members.update\_member\_status(member\_id, new\_status)

except ValueError:

print("Invalid input. Please enter a valid number for the Member ID.")

elif choice == '5':

print("\n" + "=" \* 40)

print("--- Current Library Status ---")

print("\nAll current books:")

if library.books.book\_list:

for b in library.books.book\_list:

print(" Title: ",b['title']," | Author: ",b['author']," | Status: ",b['status'])

else:

print(" (No books in the library.)")

print("\nAll current members:")

if library.members.member\_list:

for mid, details in library.members.member\_list.items():

print(" ID: ",mid," | Name: ",details['name']," | Status: ",details['status'])

else:

print(" (No members registered.)")

print("=" \* 40)

elif choice == '6':

print("\nThank you for using the Library Management System. Goodbye! ")

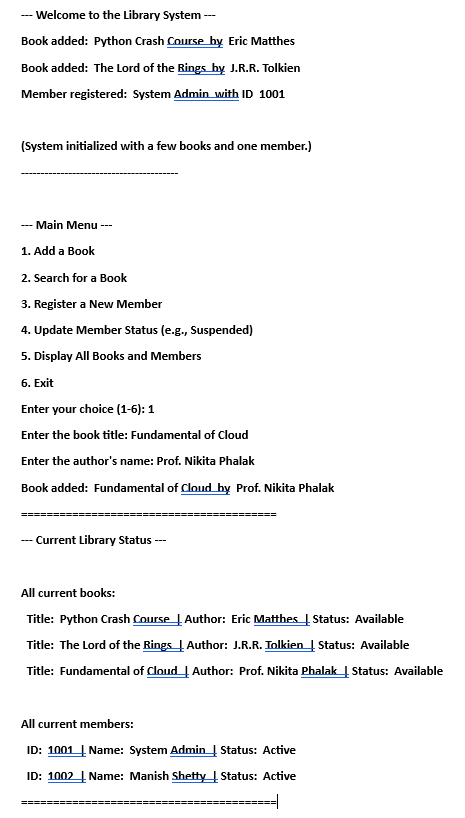
break

else:

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

simulate\_library\_operations()

**Output:**

****

**Case Study 9: University Departments System Topic: Python Namespace Packages**

* Create a namespace package university with sub-packages:
* science → subjects list.
* arts → subjects list.
* commerce → subjects list.
* Import and display subjects from different namespaces.

**Program/Code:**

**# science/subjects.py**

SCIENCE\_SUBJECTS = [

"Physics",

"Chemistry",

"Biology",

"Mathematics",

"Computer Science"

]

def get\_science\_subjects():

return SCIENCE\_SUBJECTS

**# arts/subjects.py**

ARTS\_SUBJECTS = [

"History",

"Literature",

"Philosophy",

"Sociology",

"Visual Arts"

]

def get\_arts\_subjects():

return ARTS\_SUBJECTS

**# commerce/subjects.py**

COMMERCE\_SUBJECTS = [

"Accounting",

"Economics",

"Business Studies",

"Finance",

"Marketing"

]

def get\_commerce\_subjects():

return COMMERCE\_SUBJECTS

**main.py**

import Science.subjects as science\_sub

import Arts.subjects as arts\_sub

import commerce.subjects as commerce\_sub

print("--- University Department Subjects ---")

# Import and display subjects from different namespaces

science\_list = science\_sub.get\_science\_subjects()

print("\nScience Subjects:")

for subject in science\_list:

print("- ",subject)

arts\_list = arts\_sub.get\_arts\_subjects()

print("\nArts Subjects:")

for subject in arts\_list:

print("- ",subject)

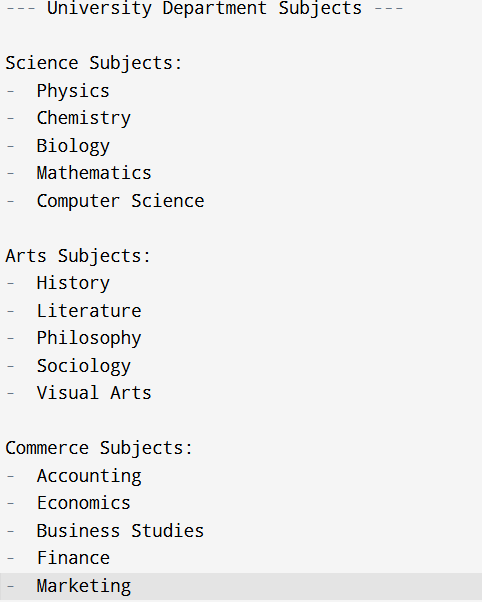
commerce\_list = commerce\_sub.get\_commerce\_subjects()

print("\nCommerce Subjects:")

for subject in commerce\_list:

print("- ",subject)

**Output:**

****

**Case Study 10: Hospital Management System Topic: User Defined Modules and Packages**

* Create a package hospital with modules:
  + patient.py → store patient details.
  + doctor.py → store doctor details.
  + appointment.py → book appointments.
* Import and combine all to simulate hospital operations.

**Program/Code:**

**# hospital/patient.py**

PATIENTS = {}

p\_id\_counter = 101

def register\_patient(name, age, condition):

global p\_id\_counter

PATIENTS[p\_id\_counter] = {"name": name, "age": age, "condition": condition}

print("Patient ",name," registered with ID P",p\_id\_counter,".")

p\_id\_counter += 1

return p\_id\_counter - 1

def get\_patient\_details(patient\_id):

return PATIENTS.get(patient\_id)

**# hospital/doctor.py**

DOCTORS = {}

d\_id\_counter = 201

def add\_doctor(name, specialty):

global d\_id\_counter

DOCTORS[d\_id\_counter] = {"name": name, "specialty": specialty}

print("Doctor ",name," (",specialty,") added with ID D",d\_id\_counter,".")

d\_id\_counter += 1

return d\_id\_counter - 1

def get\_doctor\_details(doctor\_id):

return DOCTORS.get(doctor\_id)

**# hospital/appointment.py**

APPOINTMENTS = []

def book\_appointment(patient\_id, doctor\_id, date\_time):

if not isinstance(patient\_id, int) or not isinstance(doctor\_id, int):

print("Error: Invalid ID type for booking.")

return False

appointment = {

"patient\_id": patient\_id,

"doctor\_id": doctor\_id,

"date\_time": date\_time

}

APPOINTMENTS.append(appointment)

print("Appointment booked for P",patient\_id," with D",doctor\_id," on ",date\_time,".")

return True

def get\_appointments\_for\_doctor(doctor\_id):

return [app for app in APPOINTMENTS if app['doctor\_id'] == doctor\_id]

**# main.py**

import hospital

print("--- Hospital Management System Simulation ---")

p\_id\_jane = hospital.register\_patient("Jane Doe", 35, "Fever")

p\_id\_john = hospital.register\_patient("John Smith", 60, "Routine Checkup")

print("-" \* 20)

d\_id\_dr\_sam = hospital.add\_doctor("Dr. Sam Wilson", "Cardiology")

d\_id\_dr\_lily = hospital.add\_doctor("Dr. Lily Chen", "General Practice")

print("-" \* 20)

hospital.book\_appointment(p\_id\_jane, d\_id\_dr\_lily, "2025-10-25 10:00")

hospital.book\_appointment(p\_id\_john, d\_id\_dr\_sam, "2025-10-25 11:30")

hospital.book\_appointment(p\_id\_jane, d\_id\_dr\_lily, "2025-10-26 09:00") # Jane's second appointment

print("-" \* 20)

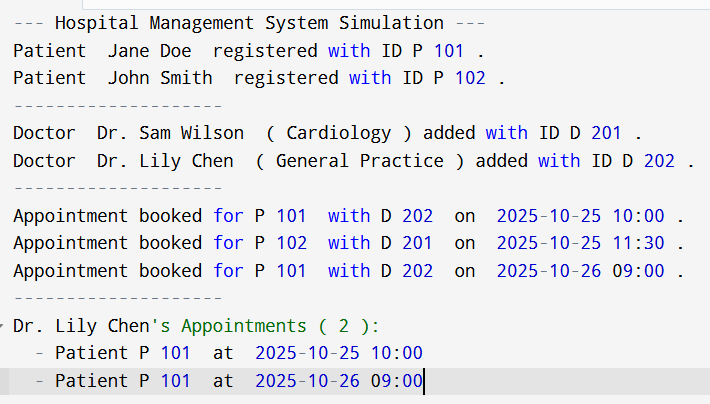
dr\_lily\_schedule = hospital.get\_appointments\_for\_doctor(d\_id\_dr\_lily)

print("Dr. Lily Chen's Appointments (",len(dr\_lily\_schedule),"):")

for app in dr\_lily\_schedule:

print(" - Patient P",app['patient\_id']," at ",app['date\_time'])

**Output:**

****

**Case Study 11.1: Online Payment Processing Topic: Exception Handling**

* Take payment input from user.
* Handle cases where user enters invalid amount (string instead of number) using try except.

**Program/Code:**

def process\_payment():

print("--- Online Payment Processing ---")

try:

amount\_str = input("Enter payment amount (e.g., 50.00): ")

amount = float(amount\_str)

if amount <= 0:

print("Payment failed: Amount must be positive.")

else:

print("Payment of ",amount," processed successfully.")

except ValueError:

print("Payment failed: Invalid amount entered.")

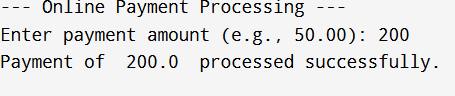
print("Please enter a valid numeric value (e.g., 50.00).")

except Exception as e:

print("An unexpected error occurred: ",e)

process\_payment()

**Output:**

****

**Case Study 11.2: Student Record File System**

* Try to open a file students.txt.
* If the file is missing, handle exception and create a new file.

**Program/Code:**

import os

FILE\_NAME = "students.txt"

def safeguard\_file\_operation():

print("--- Safeguarding File: ",FILE\_NAME," ---")

try:

with open(FILE\_NAME, 'r') as file:

print("File '",FILE\_NAME,"' found and opened successfully.")

content = file.read().strip()

if content:

print("Content preview:\n" + content[:100] + ("..." if len(content) > 100 else ""))

else:

print("File is empty.")

except FileNotFoundError:

print("File '",FILE\_NAME,"' is missing. Creating a new file...")

try:

with open(FILE\_NAME, 'w') as file:

file.write("Student Records:\n")

file.write("Name, Age, Grade\n")

print("New file '",FILE\_NAME,"' created successfully.")

except IOError as e:

print("Error creating file: ",e)

except Exception as e:

print("An unexpected error occurred: ",e)

if os.path.exists(FILE\_NAME):

os.remove(FILE\_NAME)

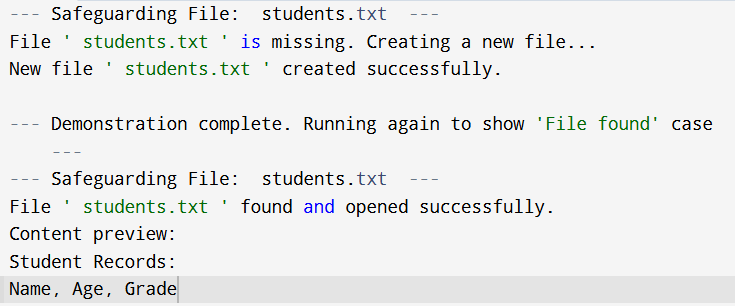
print("(Cleaned up previous '",FILE\_NAME,"' for fresh demo)")

safeguard\_file\_operation()

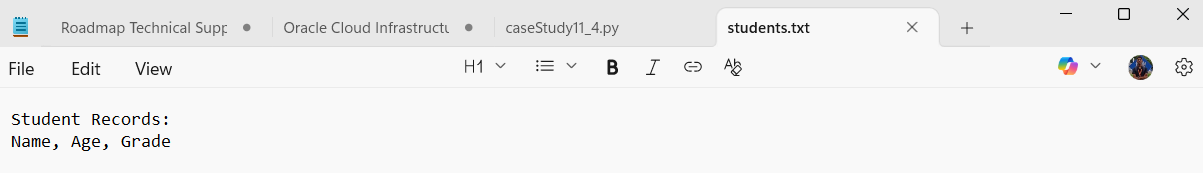
print("\n--- Demonstration complete. Running again to show 'File found' case ---")

safeguard\_file\_operation()

**Output:**

****

**Students.txt file:**

****

**Case Study 11.3: ATM Withdrawal**

* Raise a user-defined exception if withdrawal amount > balance.
* Handle multiple exceptions like: o Invalid input type. o Amount not multiple of 100.

**Program/Code:**

class InsufficientFundsError(Exception):

def \_\_init\_\_(self, requested, balance):

self.requested = requested

self.balance = balance

super().\_\_init\_\_("Withdrawal amount (",requested,") exceeds available balance (",balance,").")

class NotMultipleOfHundredError(Exception):

def \_\_init\_\_(self, amount):

self.amount = amount

super().\_\_init\_\_("Withdrawal amount (",amount,") must be a multiple of 100.")

def atm\_withdrawal(current\_balance):

print("\n--- ATM Withdrawal Simulation ---")

print("Current Balance: ",current\_balance)

try:

amount\_str = input("Enter withdrawal amount (multiple of 100): $")

withdrawal\_amount = int(amount\_str)

if withdrawal\_amount % 100 != 0:

raise NotMultipleOfHundredError(withdrawal\_amount)

if withdrawal\_amount > current\_balance:

raise InsufficientFundsError(withdrawal\_amount, current\_balance)

new\_balance = current\_balance - withdrawal\_amount

print("Transaction successful! Withdrew ",withdrawal\_amount)

print("New Balance: ",new\_balance)

except ValueError:

print("Error: Invalid input. Please enter a whole number.")

except NotMultipleOfHundredError as e:

print("Error",e)

except InsufficientFundsError as e:

print("Transaction failed: ",e)

print("Please enter a lower amount.")

except Exception as e:

print(f"An unexpected system error occurred: {e}")

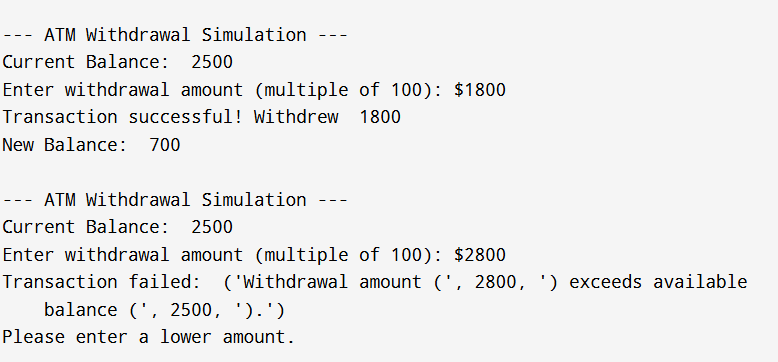
atm\_withdrawal(current\_balance=2500) # Valid transaction

atm\_withdrawal(current\_balance=2500) # Test InsufficientFundsError (e.g., enter 3000)

atm\_withdrawal(current\_balance=2500) # Test NotMultipleOfHundredError (e.g., enter 250)

atm\_withdrawal(current\_balance=2500) # Test ValueError (e.g., enter 'five hundred')

**Output:**

****

**Case Study 11.4: Web Application Debugging**

* Create a function that logs exceptions into an error file error\_log.txt.
* Along with the exception, store an error code and timestamp.

**Program/Code:**

import datetime

ERROR\_LOG\_FILE = "error\_log.txt"

def log\_exception(error\_code, exception\_message):

timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

log\_entry = f"[{timestamp}] CODE: {error\_code} | EXCEPTION: {exception\_message}\n"

try:

with open(ERROR\_LOG\_FILE, 'a') as f:

f.write(log\_entry)

print(f"Error logged successfully with code {error\_code}.")

except Exception as e:

print(f"CRITICAL ERROR: Could not write to log file: {e}")

def process\_user\_data(data):

print(f"\n--- Processing data: {data} ---")

try:

user\_id = data['user\_id']

username = data['username']

result = 100 / data['settings']['divisor']

print(f"Data processed for User {user\_id} ({username}). Result: {result}")

except KeyError as e:

error\_msg = f"Missing dictionary key: {e}"

log\_exception(error\_code="4001\_MISSING\_KEY", exception\_message=error\_msg)

print("Processing failed. Check error log.")

except ZeroDivisionError as e:

error\_msg = str(e)

log\_exception(error\_code="5002\_ZERO\_DIV", exception\_message=error\_msg)

print("Processing failed. Check error log.")

except Exception as e:

error\_msg = str(e)

log\_exception(error\_code="9999\_UNEXPECTED", exception\_message=error\_msg)

print("Processing failed. Check error log.")

# Test Case 1: Missing 'username' key (KeyError)

process\_user\_data({'user\_id': 123, 'settings': {'divisor': 1}})

# Test Case 2: Division by zero (ZeroDivisionError)

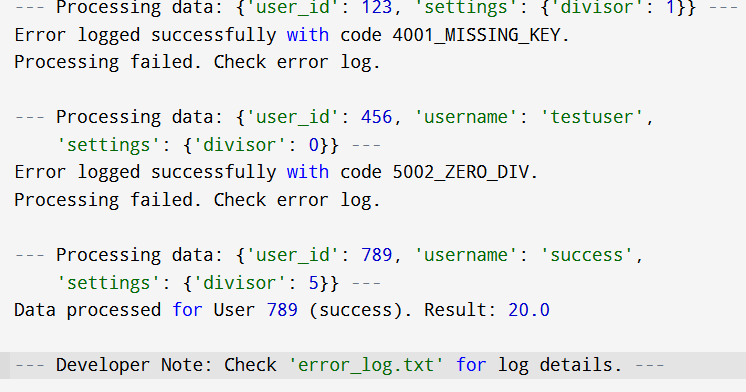
process\_user\_data({'user\_id': 456, 'username': 'testuser', 'settings': {'divisor': 0}})

# Test Case 3: Successful execution

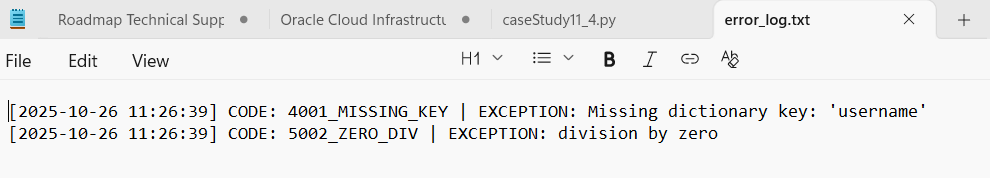
process\_user\_data({'user\_id': 789, 'username': 'success', 'settings': {'divisor': 5}})

print(f"\n--- Developer Note: Check '{ERROR\_LOG\_FILE}' for log details. ---")

**Output:**

****

**Errorlog.txt File Created:**

****

**Case Study 11.5: Airline Ticket Reservation**

* Implement ticket booking where:
* If seats are full → raise an exception.
* If user enters invalid passenger data → handle gracefully.
* Always close the file/logging resource using finally.

**Program/Code:**

class SeatsFullError(Exception):

def \_\_init\_\_(self, flight\_id):

self.flight\_id = flight\_id

super().\_\_init\_\_("Flight flight\_id is fully booked. No seats available.")

MAX\_SEATS = 2

BOOKED\_SEATS = 0

LOG\_FILE = "reservation\_log.txt"

def book\_ticket(flight\_id, passenger\_data):

global BOOKED\_SEATS

file\_resource = None

print(f"\n--- Attempting to book for {flight\_id} ---")

try:

if BOOKED\_SEATS >= MAX\_SEATS:

raise SeatsFullError(flight\_id)

if not isinstance(passenger\_data.get('name'), str) or len(passenger\_data['name']) < 2:

print("Booking failed: Invalid or missing passenger name.")

return

if not isinstance(passenger\_data.get('age'), int) or passenger\_data['age'] <= 0:

print("Booking failed: Invalid passenger age.")

return

BOOKED\_SEATS += 1

file\_resource = open(LOG\_FILE, 'a')

log\_entry = f"BOOKED: Flight {flight\_id}, Seat {BOOKED\_SEATS}, Passenger: {passenger\_data['name']}\n"

file\_resource.write(log\_entry)

print(f"Booking successful! Seat {BOOKED\_SEATS} on {flight\_id} for {passenger\_data['name']}.")

except SeatsFullError as e:

print(f"Booking failed: {e}")

except Exception as e:

print(f"An unexpected error occurred during booking: {e}")

finally:

if file\_resource:

file\_resource.close()

print("--- Booking attempt finished. ---")

import os

if os.path.exists(LOG\_FILE):

os.remove(LOG\_FILE)

# Test Case 1: Successful booking

book\_ticket("AI101", {'name': 'Alice Smith', 'age': 30})

# Test Case 2: Successful booking (max seats = 2)

book\_ticket("AI101", {'name': 'Bob Johnson', 'age': 45})

# Test Case 3: Seats full (Raises SeatsFullError)

book\_ticket("AI101", {'name': 'Charlie Brown', 'age': 22})

# Test Case 4: Invalid passenger data (Handled gracefully with return)

book\_ticket("AI102", {'name': '', 'age': 25})

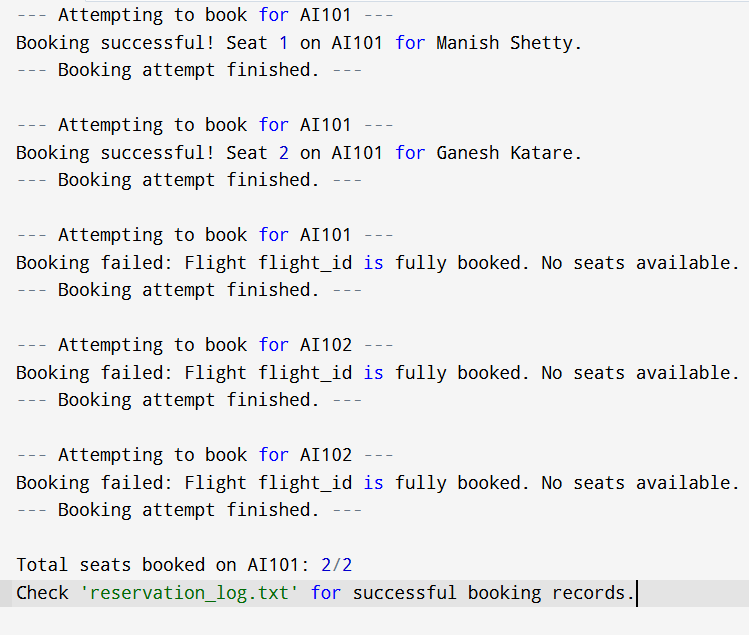
# Test Case 5: Invalid passenger data (Handled gracefully with return)

book\_ticket("AI102", {'name': 'Eve', 'age': 'twenty'})

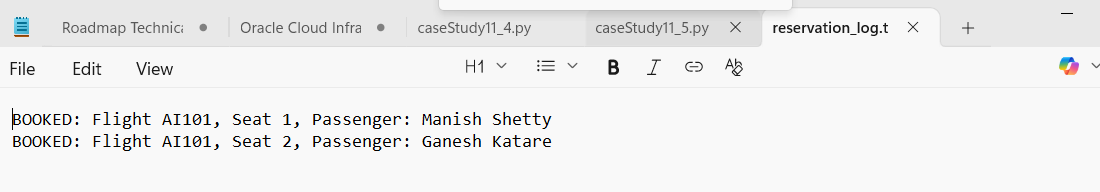
print(f"\nTotal seats booked on AI101: {BOOKED\_SEATS}/{MAX\_SEATS}")

print(f"Check '{LOG\_FILE}' for successful booking records.")

**Output:**

****

**Reservation\_log.txt file:**

****