**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.1**

**Title: Employee Class**

**Theory:** The code is written to display employee data using employee class, using two different methods, get and print.

**Code:**

class Employee:

def \_\_init\_\_(self, emp\_id, emp\_name, emp\_salary):

self.emp\_id = emp\_id

self.emp\_name = emp\_name

self.emp\_salary = emp\_salary

def get\_employee\_details(self):

return f"Employee ID: {self.emp\_id}\nEmployee Name: {self.emp\_name}\nEmployee Salary: {self.emp\_salary}"

def print\_employee\_details(self):

print(self.get\_employee\_details())

employee1 = Employee(emp\_id=1011, emp\_name="John Doe", emp\_salary=250000)

details = employee1.get\_employee\_details()

print("Employee Details got:\n", details)

print("\nEmployee Details print: ")

employee1.print\_employee\_details()

id1=int(input("\nEnter your id: "))

name=input("Enter your name: ")

sal=float(input('Enter your salary amout: '))

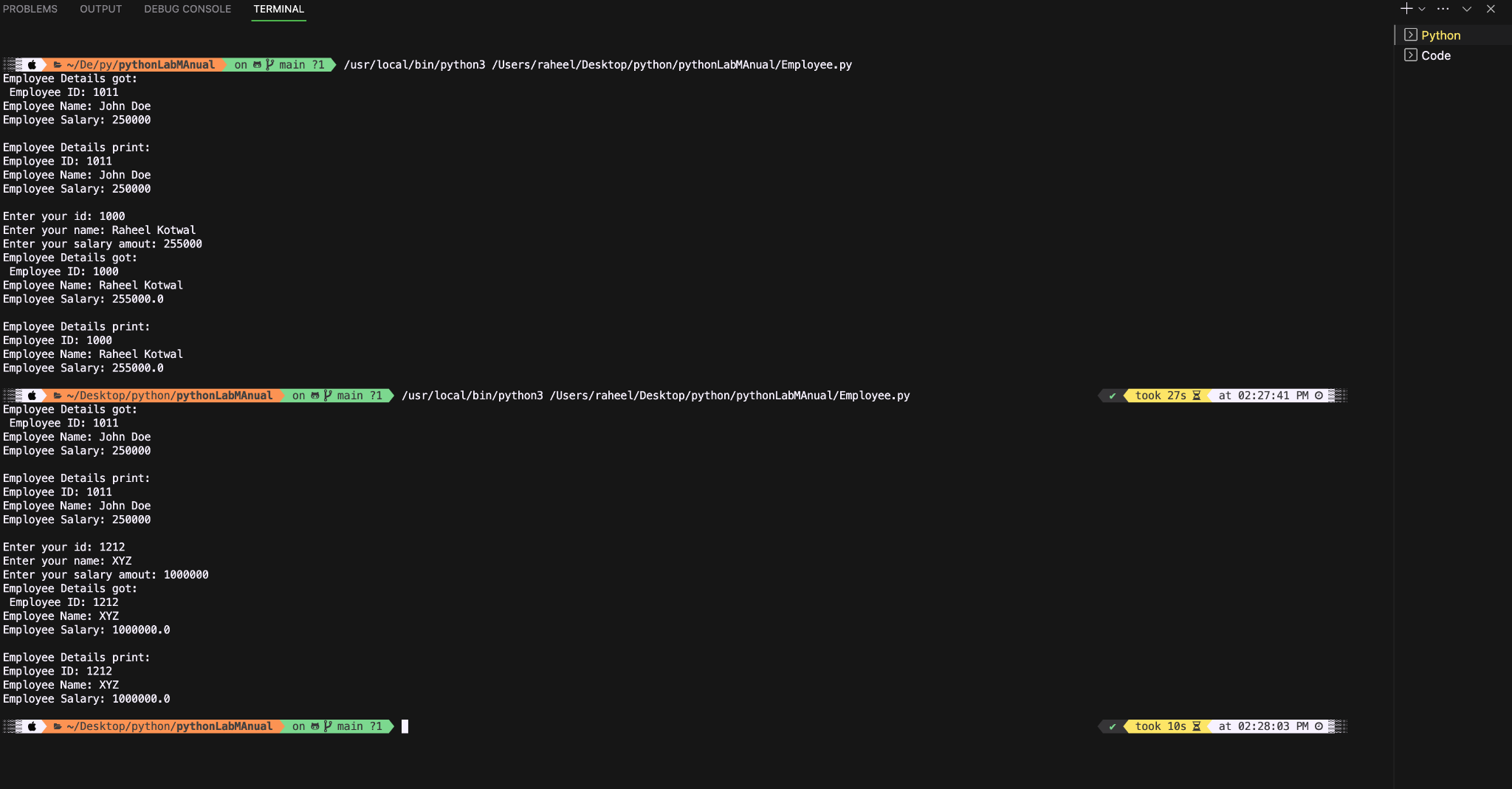
employeeUser = Employee(emp\_id=id1, emp\_name=name, emp\_salary=sal)

print("Employee Details got:\n", employeeUser.get\_employee\_details())

print("\nEmployee Details print: ")

employeeUser.print\_employee\_details()

**Output:(screenshot):**



**Conclusion:**

**The code works perfectly as seen above using two cases. The salary is taken as a float value correctly and displayed as such too.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.2**

**Title: Positional and Keyword Based arguments**

**Theory:**  The code below uses a function to demonstrate the positional and keyword based arguments.

**Code:**

def get\_user\_info(\*args, \*\*kwargs):

if len(args) >= 3:

name = args[0]

email = args[1]

age = args[2]

additional\_info = kwargs.get('more\_info', '')

print("\nUser Information:")

print(f"Name: {name}")

print(f"Email: {email}")

print(f"Age: {age}")

print(f"Additional Info: {additional\_info}")

get\_user\_info('John Doe', 'john@example.com', 30, more\_info='Loves fishing')

#first three parameters are taken as positional arguements and 4th is taken as keywordBased argument

name1=input("Enter name: ")

email1=input("Enter email: ")

get\_user\_info(name1, email1, 30, more\_info='no info')

**Output:(screenshot):**



**Conclusion:**

**This code allows flexibility in function calls, enhancing readability**

**and accommodating various input scenario.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.3**

**Title:** Class, Object and Constructor

**Theory:** The code uses a class to admit students, track their department distribution, and display the list of students along with a summary of the department-wise admission count.

**Code:**

class Student:

students = []

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

self.name = name

self.department = department

self.count\_students(department)

self.students.append(self)

def count\_students(self, department):

if department not in self.departments\_count:

self.departments\_count[department] = 1

else:

self.departments\_count[department] += 1

def display\_students(self):

print("\nList of Admitted Students:")

for student in self.students:

print(f"Name: {student.name}, Department: {student.department}")

@classmethod

def display\_summary(cls):

print("\nStudent Admission Summary:")

for department, count in cls.departments\_count.items():

print(f"{department.capitalize()}: {count} students")

departments\_count = {}

students = []

num\_students = int(input("Enter the number of students to admit: "))

for i in range(num\_students):

name = input(f"Enter the name of student {i + 1}: ")

department = input(f"Enter the department of student {i + 1} (pgdm/btech): ").lower()

while department not in ['pgdm', 'btech']:

print("Invalid department. Please enter 'pgdm' or 'btech'.")

department = input(f"Enter the department of student {i + 1} (pgdm/btech): ").lower()

student = Student(name, department)

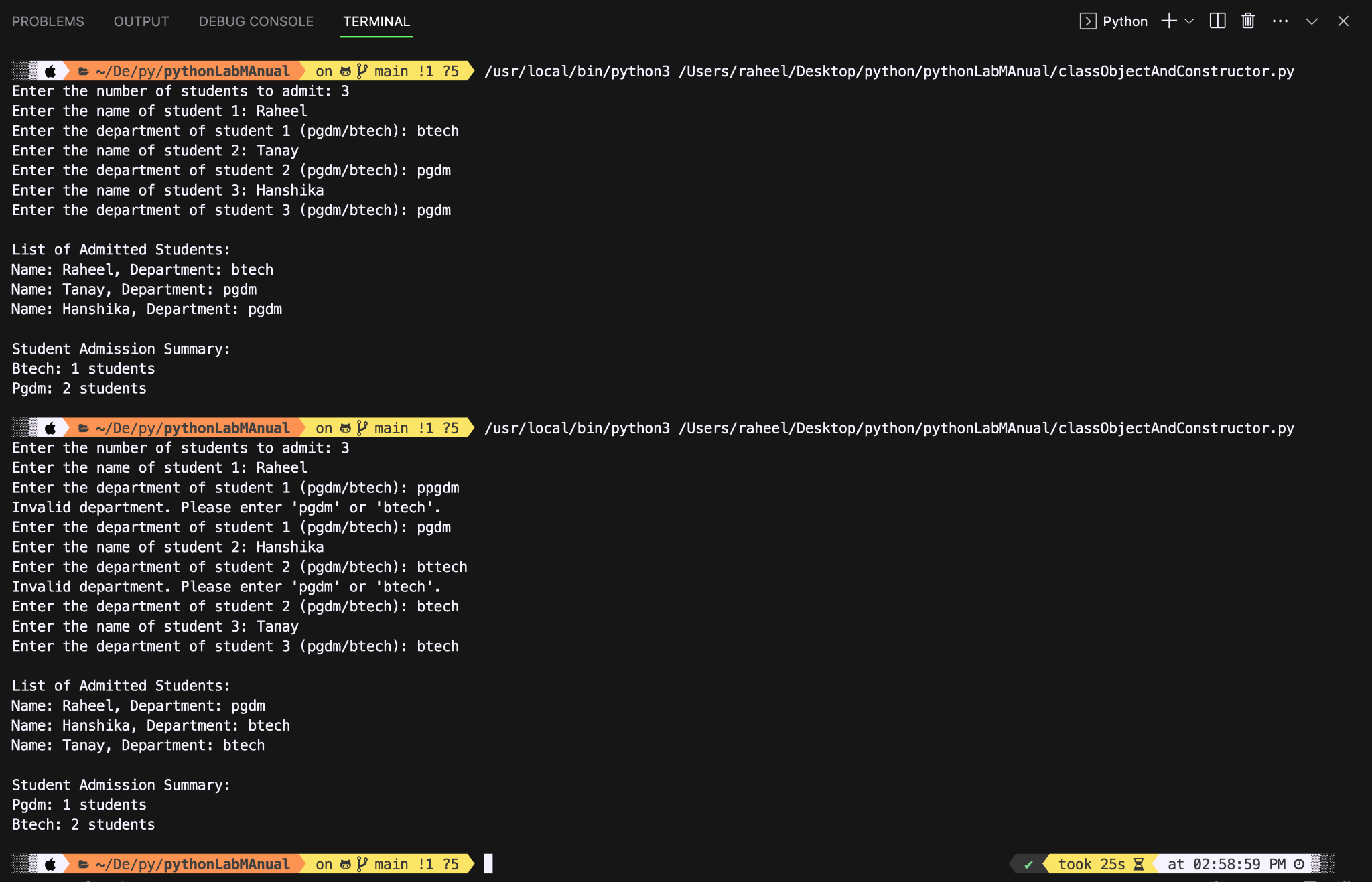
students.append(student)

for student in students:

student.display\_students()

Student.display\_summary()

**Output:(screenshot):**



**Conclusion:**

**The code successfully creates a constructor and works with user input as well.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.4**

**Title: Cube from a given list**

**Theory:** The code displays all products one by one and the user has the choice to select those number of items..

**Code:**

class Store:

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

self.products = {'A': {'name': 'Pen', 'price': 10}, 'B': {'name': 'Bud', 'price': 15}, 'C': {'name': 'Eraser', 'price': 20}}

def display\_menu(self):

print("\nProduct Menu:")

for code, product\_info in self.products.items():

print(f"{code}: {product\_info['name']} - ₹{product\_info['price']}")

def generate\_bill(self, quantities):

total\_amount = 0

print("\nBill:")

for code, quantity in quantities.items():

if code in self.products:

product\_info = self.products[code]

item\_total = quantity \* product\_info['price']

total\_amount += item\_total

print(f"{product\_info['name']} ({quantity} units) - ₹{item\_total}")

print("\nTotal Amount: ₹{:.2f}".format(total\_amount)) #searched how to...

store = Store()

store.display\_menu()

quantities = {}

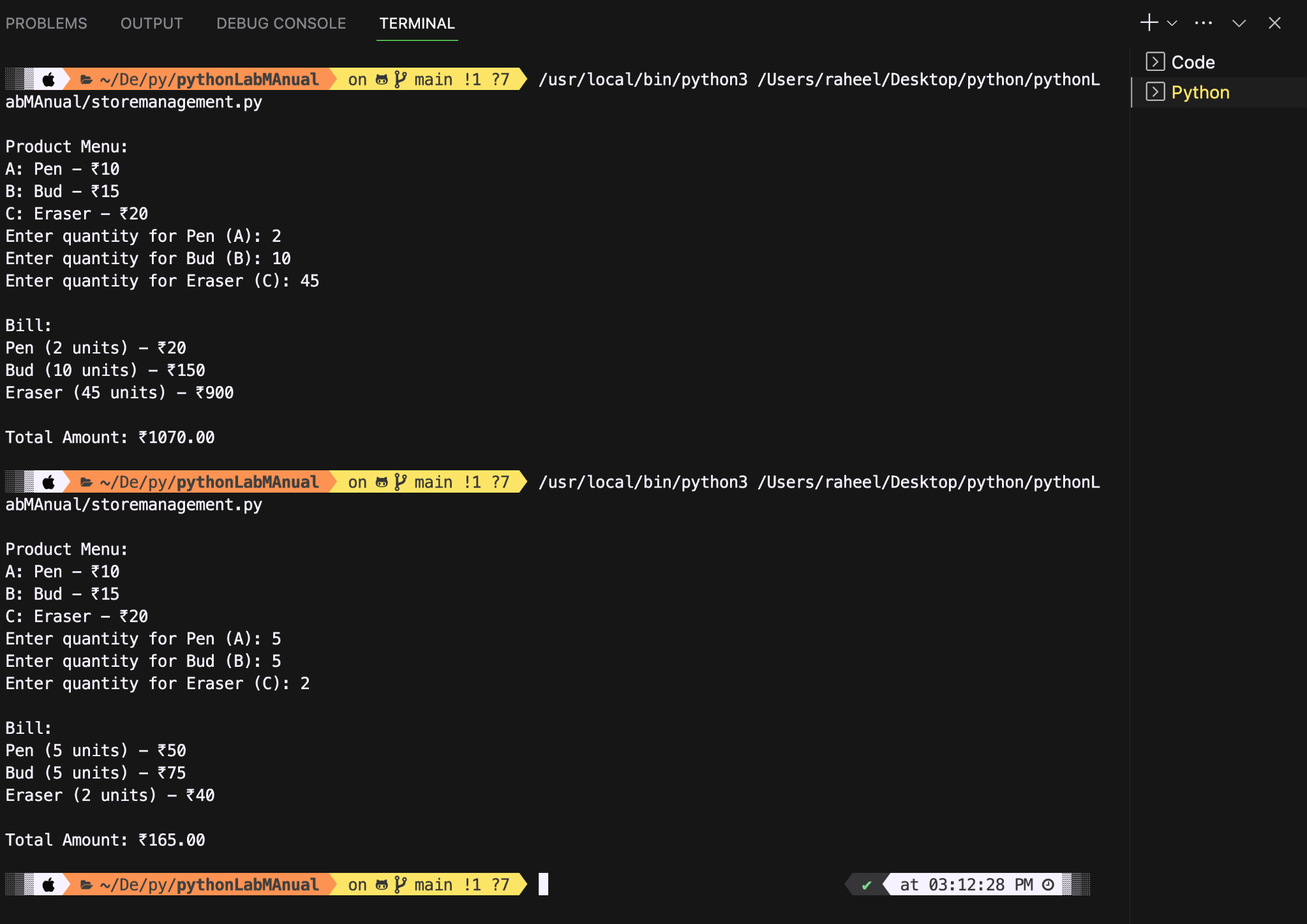
for code in store.products:

quantity = int(input(f"Enter quantity for {store.products[code]['name']} ({code}): "))

quantities[code] = quantity

store.generate\_bill(quantities)

**Output:(screenshot):**



**Conclusion:**

**The code successfully runs based on the user input where if user doesn’t want a certain product they can simply enter quantity as 0.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.5**

**Title: Single Inheritance**

**Theory:** The program uses single inheritance to demonstrate addition of two numbers.

**Code:**

class addition:

def add1(self):

print(self.a+self.b)

class values(addition):

def getvalues(self):

self.a=int(input("enter the value of a : "))

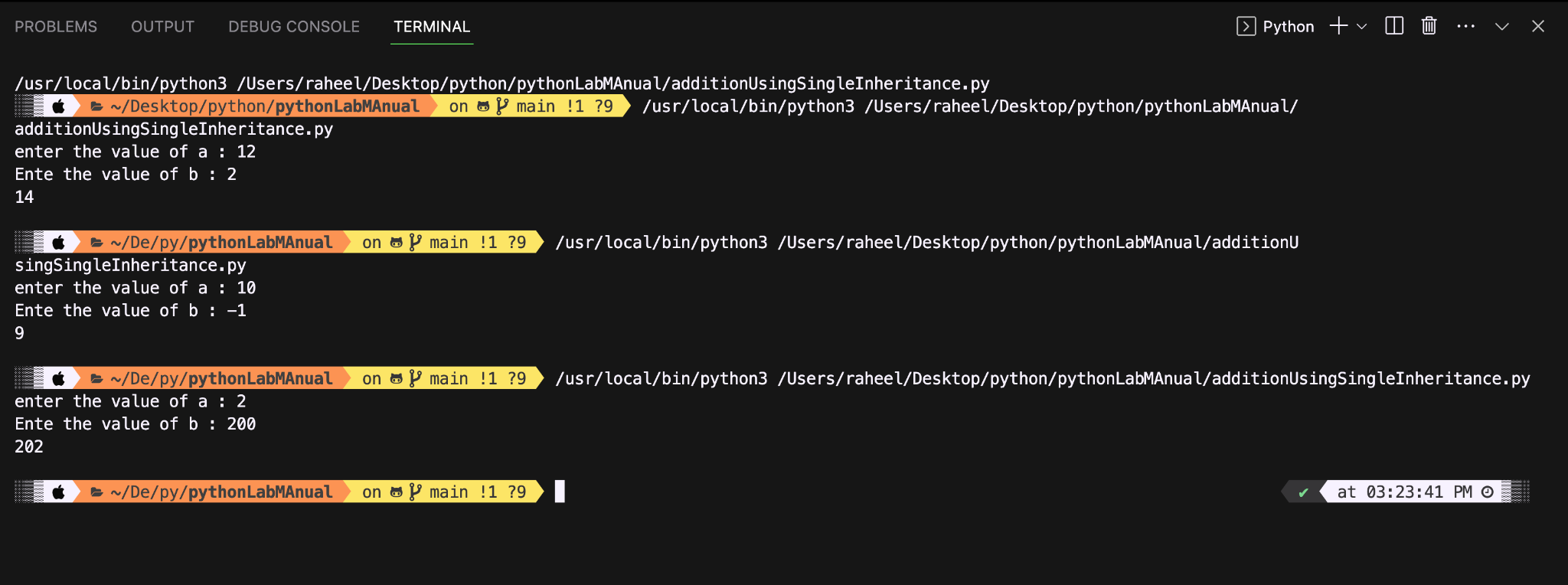
self.b=int(input("Ente the value of b : "))

c=values()

c.getvalues()

c.add1()

**Output:(screenshot):**

****

**Conclusion:**

The code successfully demonstrates single inheritance and returns the sum of two numbers where the child class takes values from the user, and the parent class returns the sum.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.6**

**Title: Multiple Inheritance**

**Theory:** The following code demonstrates multiple inheritance using base classes LU and ITM and derived class as Courses.

**Code:**

class ITM:

subjects=["Hotel management","BTECH CSE","Design","Business"]

teachers=["Sheetal ma'am","Sumit sir","Nidhi ma'am","Sai sir"]

duration=["90 days","100 days","80 days","30 days"]

class LU:

subjects=["AI/ML","AR/VR","Cloud Computing","Full Stack Developer"]

teachers=["Sai Sir","Swapnil Sir","Vaibhav sir","Rohini ma'am"]

duration=["60 days","30 days","50 days","55 days"]

class Courses(LU,ITM):

print("Lets Upgrade courses:\n")

luSubs=LU.subjects

for i in range(len(luSubs)):

print(i+1,LU.subjects[i],"by",LU.teachers[i],"for",LU.duration[i])

lucourses=[]

subNum=int(input("How many subjects you want to select: "))

if subNum>4:

print("Invalid number of subjects")

else:

for i in range(subNum):

courseNum=int(input("Select your interested course: "))

lucourses.append(courseNum)

print("\nITM courses are:\n")

itmSubs=ITM.subjects

for i in range(len(itmSubs)):

print(i+1,ITM.subjects[i],"by",ITM.teachers[i],"for",ITM.duration[i])

itmCourses=[]

subNum=int(input("How many subjects you want to select: "))

if subNum>4:

print("Invalid number of subjects")

else:

for i in range(subNum):

courseNum=int(input("Select your interested course: "))

itmCourses.append(courseNum)

print("\nYour selected courses are:\n")

for i in lucourses:

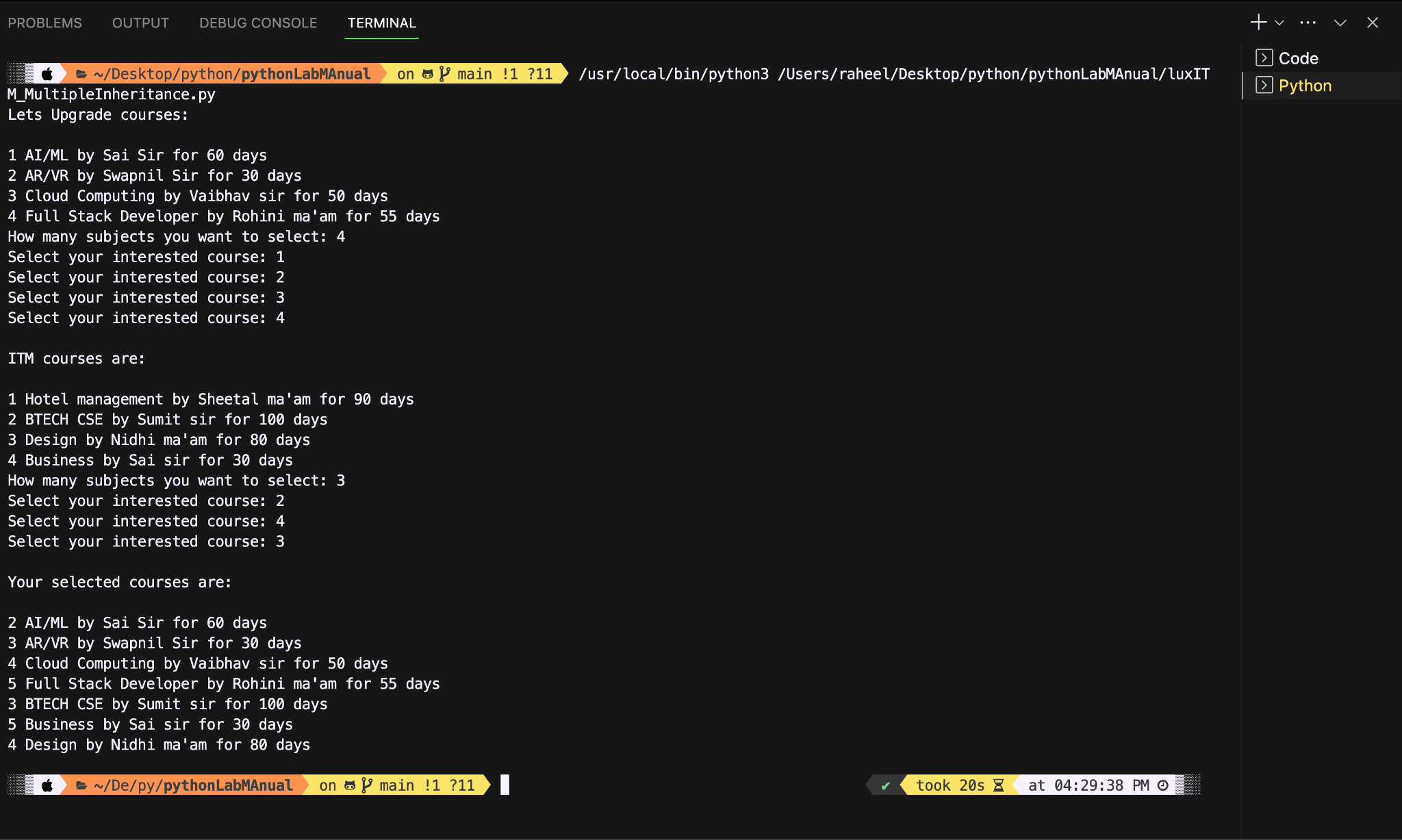
print(i+1,LU.subjects[i-1],"by",LU.teachers[i-1],"for",LU.duration[i-1])

for i in itmCourses:

print(i+1,ITM.subjects[i-1],"by",ITM.teachers[i-1],"for",ITM.duration[i-1])

obj=Courses()

**Output:(screenshot):**

****

**Conclusion:**

The program successfully demonstrates multiple inheritance and also works with logically sound based on user input.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.7**

**Title: Property Inheritance (Multilevel Inheritance)**

**Theory:** The code below demonstrates multilevel inheritance using the example of property inheritance.

**Code:**

**class GrandFather:**

**def \_\_init\_\_(self):**

**self.name = " GrandFather"**

**self.\_assets = 1500000**

**class Father(GrandFather):**

**def \_\_init\_\_(self):**

**super().\_\_init\_\_()**

**self.name = " Father" + self.name**

**self.\_inharitedAssets = self.\_assets**

**self.\_purchasedAssets = 200000**

**self.\_totalAssets = self.\_inharitedAssets + self.\_purchasedAssets**

**class Child(Father):**

**def \_\_init\_\_(self, name, assets):**

**super().\_\_init\_\_()**

**self.name = name + " " + self.name**

**self.\_\_inharitedAssets = self.\_totalAssets**

**self.\_\_purchasedAssets = assets**

**self.\_totalAssets = self.\_\_inharitedAssets + self.\_\_purchasedAssets**

**def displayData(self):**

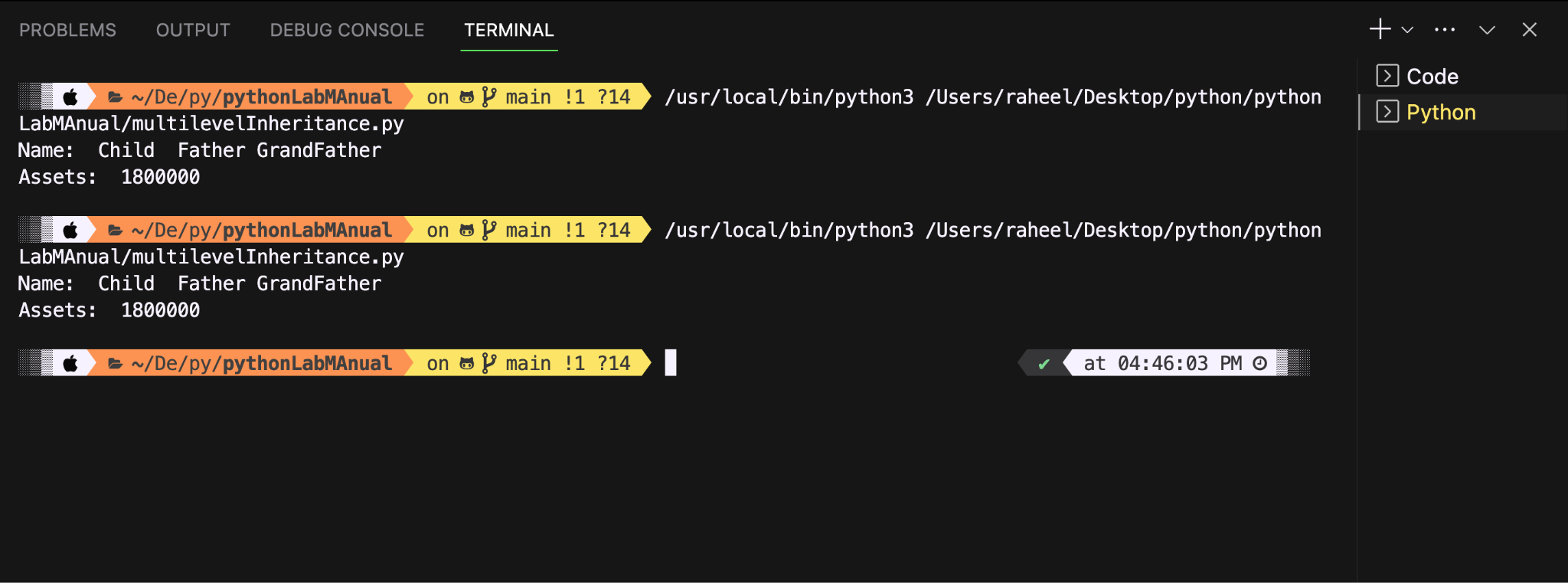
**print("Name: ", self.name)**

**print("Assets: ", self.\_totalAssets)**

**obj = Child("Child",100000)**

**obj.displayData()**

**Output:(screenshot):**

****

**Conclusion:**

Multilevel Inheritance is shown properly, super keyword used perfectly for constructor.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 4.8**

**Title: Library Management**

**Theory:** This code demonstrates a library management system properly, using the inheritance concept.

**Code:**

**class LibraryItem:**

**library\_items = []**

**def \_\_init\_\_(self, item\_type, item\_name, item\_id, item\_count=0):**

**self.item\_type = item\_type**

**self.item\_name = item\_name**

**self.item\_id = item\_id**

**self.item\_count = item\_count**

**item\_data = {**

**'item\_type': self.item\_type,**

**'item\_name': self.item\_name,**

**'item\_id': self.item\_id,**

**'item\_count': self.item\_count**

**}**

**if self.item\_type == 'book':**

**self.author\_name = input("Enter author's name for the book: ")**

**item\_data['author\_name'] = self.author\_name**

**elif self.item\_type == 'journal':**

**self.publisher\_name = input("Enter publisher's name for the journal: ")**

**item\_data['publisher\_name'] = self.publisher\_name**

**elif self.item\_type == 'dvd':**

**self.director\_name = input("Enter director's name for the DVD: ")**

**item\_data['director\_name'] = self.director\_name**

**else:**

**raise ValueError("Invalid item\_type. Please retry with 'book', 'journal', or 'dvd'.")**

**self.library\_items.append(item\_data)**

**def display\_info(self):**

**for item\_data in self.library\_items:**

**print("Information about library items:")**

**for key, value in item\_data.items():**

**print(f"{key}: {value}")**

**print("\n")**

**def check\_out(self, item\_name):**

**for item\_data in self.library\_items:**

**if item\_data['item\_name'] == item\_name:**

**if item\_data['item\_count'] > 0:**

**print(f"Checking out {item\_name}")**

**item\_data['item\_count'] -= 1**

**else:**

**print(f"All copies of {item\_name} are checked out.")**

**return**

**print(f"{item\_name} not found in the library.")**

**def check\_in(self, item\_name):**

**for item\_data in self.library\_items:**

**if item\_data['item\_name'] == item\_name:**

**print(f"Checking in {item\_name}")**

**item\_data['item\_count'] += 1**

**return**

**self.\_\_init\_\_(self.item\_type, item\_name, len(self.library\_items) + 1, item\_count=1)**

**class Book(LibraryItem):**

**def \_\_init\_\_(self, item\_name, item\_id):**

**super().\_\_init\_\_('book', item\_name, item\_id)**

**class DVD(LibraryItem):**

**def \_\_init\_\_(self, item\_name, item\_id):**

**super().\_\_init\_\_('dvd', item\_name, item\_id)**

**class Journal(LibraryItem):**

**def \_\_init\_\_(self, item\_name, item\_id):**

**super().\_\_init\_\_('journal', item\_name, item\_id)**

**book = Book('The Great Gatsby', 1) #item\_name and item\_id**

**dvd = DVD('Inception', 2)**

**journal = Journal('Scientific American', 3)**

**book.check\_in('The Great Gatsby') #plus 1 to item\_count**

**book.check\_in('The Great Gatsby')**

**book.check\_in('The Great Gatsby')**

**book.check\_in('The Great Gatsby')**

**book.check\_in('The Great Gatsby')**

**book.display\_info()**

**book.check\_out('The Great Gatsby')**

**book.display\_info()**

**book.display\_info()**

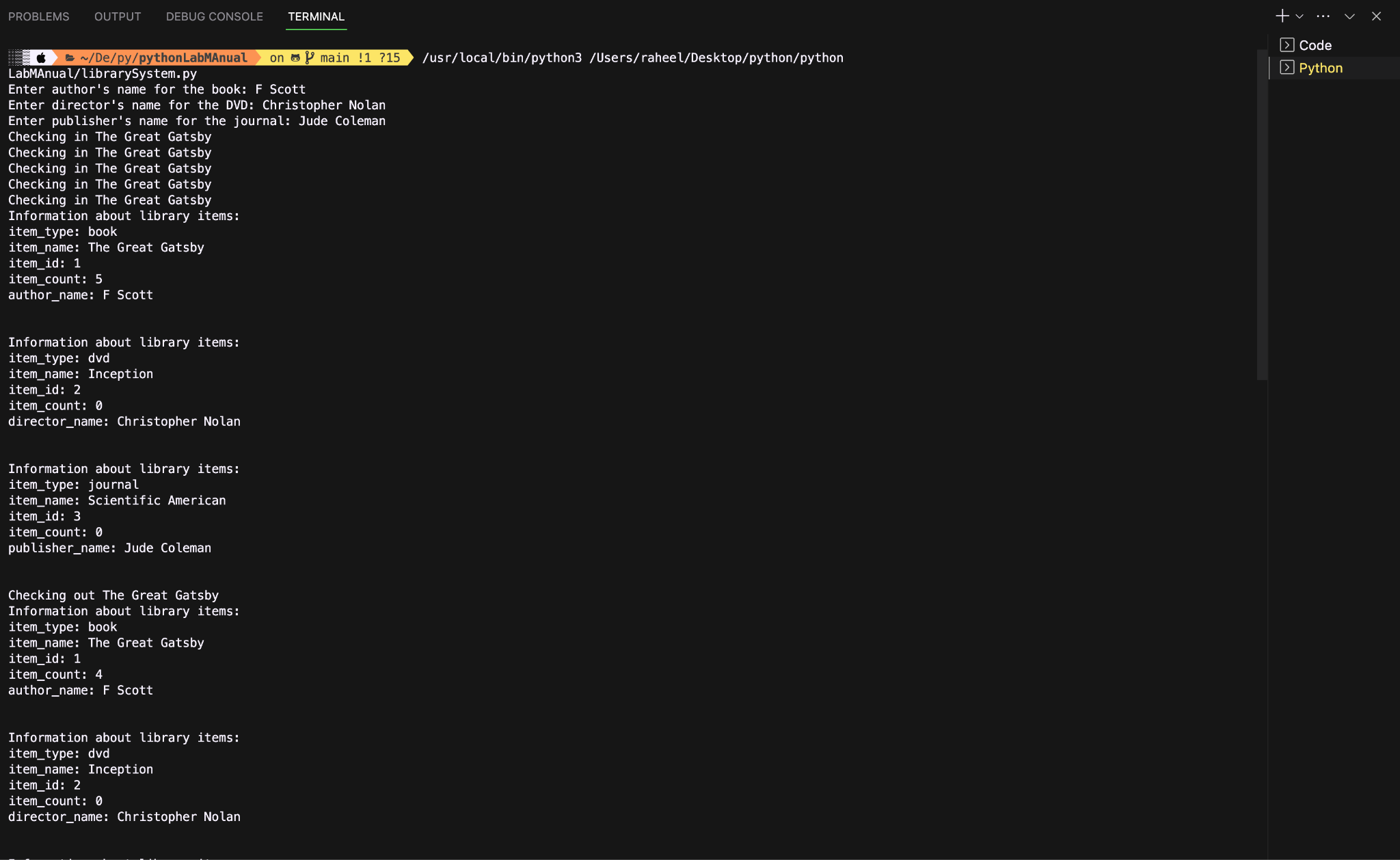
**dvd.check\_in('Inception')**

**dvd.display\_info()**

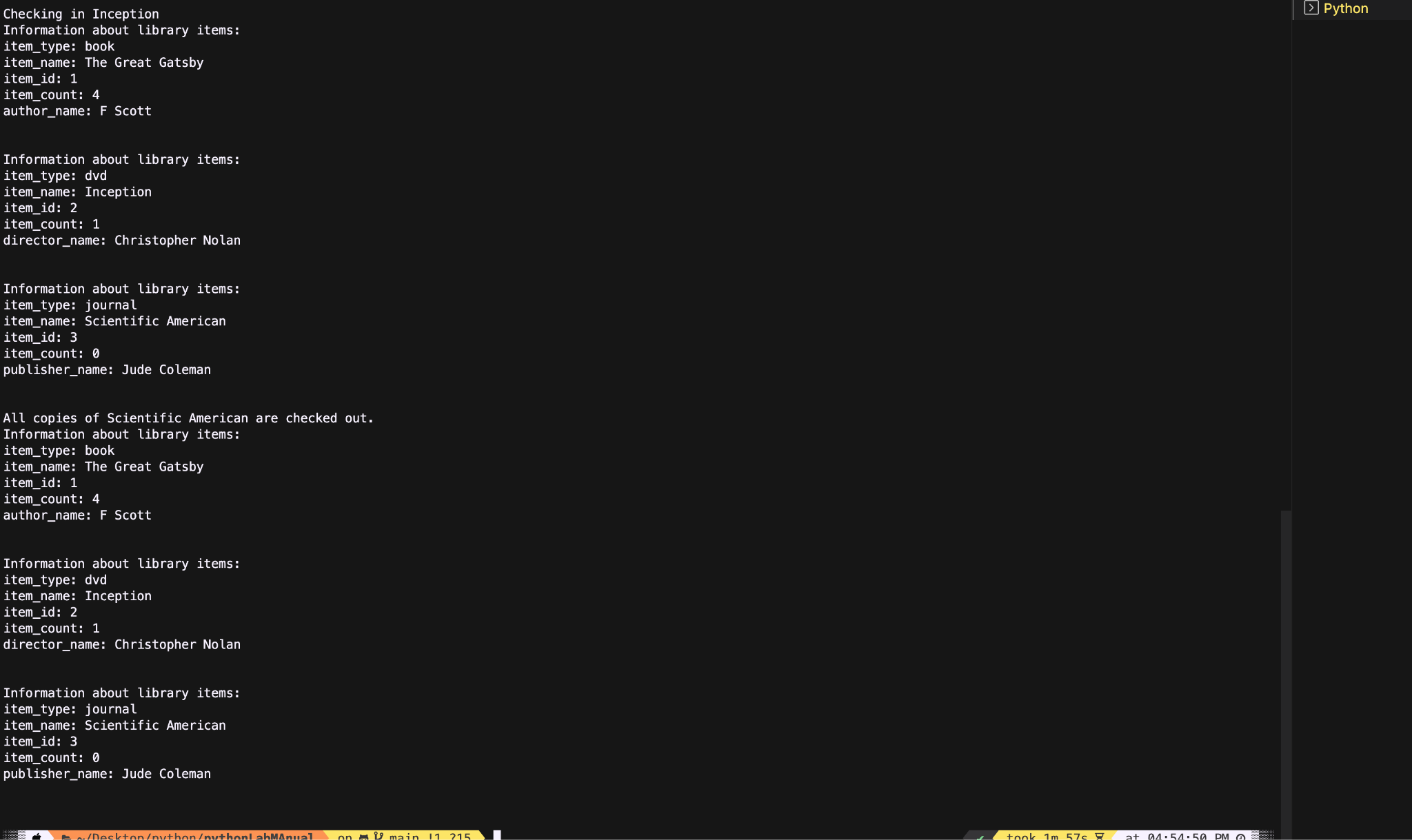
**journal.check\_out('Scientific American')**

**journal.display\_info()**

**Output:(screenshot):**

****

****

****

**Conclusion:**

The library management system shows each step of the collection process. Thus it has a long output divided by 3 different screenshots. The code does not take user input for checking in or out any particular item from the library.