# AI Assisted Coding

ASSIGNMENT 5.1 AND 6

NAME: G.V.N HASINI

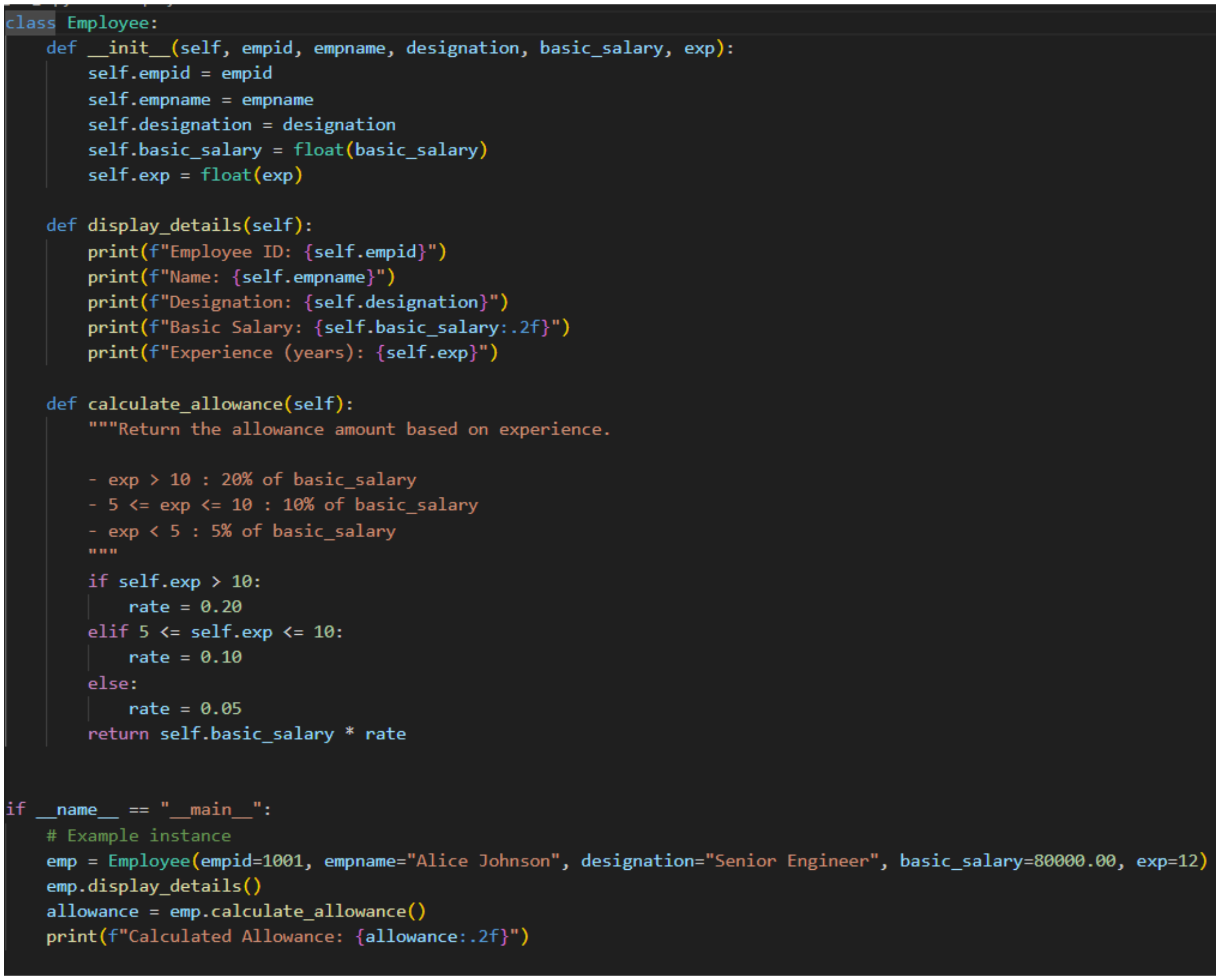
HT.NO: 2303A51099

Task 1:

Employee Data: Create Python code that defines a class named `Employee` with the following attributes: `empid`, `empname`, `designation`, `basic_salary`, and `exp`. Implement a method `display_details()` to print all employee details. Implement another method `calculate_allowance()` to determine additional allowance based on experience:

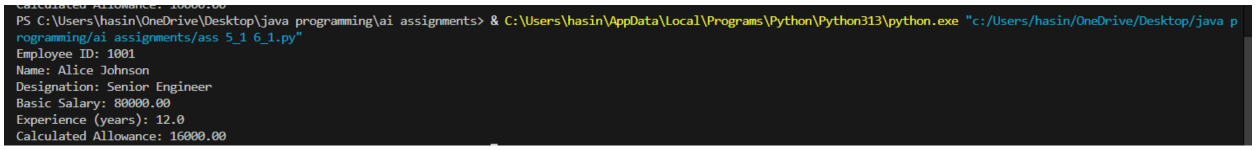
- If `exp > 10 years` → allowance = 20% of `basic_salary`
- If `5 ≤ exp ≤ 10 years` → allowance = 10% of `basic_salary`
- If `exp < 5 years` → allowance = 5% of `basic_salary`

Finally, create at least one instance of the `Employee` class, call the `display_details()` method, and print the calculated allowance.



```python
class Employee:
    def __init__(self, empid, empname, designation, basic_salary, exp):
        self.empid = empid
        self.empname = empname
        self.designation = designation
        self.basic_salary = float(basic_salary)
        self.exp = float(exp)

    def display_details(self):
        print(f"Employee ID: {self.empid}")
        print(f"Name: {self.empname}")
        print(f"Designation: {self.designation}")
        print(f"Basic Salary: {self.basic_salary:.2f}")
        print(f"Experience (years): {self.exp}")

    def calculate_allowance(self):
        """Return the allowance amount based on experience.

        - exp > 10 : 20% of basic_salary
        - 5 <= exp <= 10 : 10% of basic_salary
        - exp < 5 : 5% of basic_salary
        """
        if self.exp > 10:
            rate = 0.20
        elif 5 <= self.exp <= 10:
            rate = 0.10
        else:
            rate = 0.05
        return self.basic_salary * rate


if __name__ == "__main__":
    # Example instance
    emp = Employee(empid=1001, empname="Alice Johnson", designation="Senior Engineer", basic_salary=80000.00, exp=12)
    emp.display_details()
    allowance = emp.calculate_allowance()
    print(f"Calculated Allowance: {allowance:.2f}")
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Employee ID: 1001
Name: Alice Johnson
Designation: Senior Engineer
Basic Salary: 80000.00
Experience (years): 12.0
Calculated Allowance: 16000.00
```

Task 2:

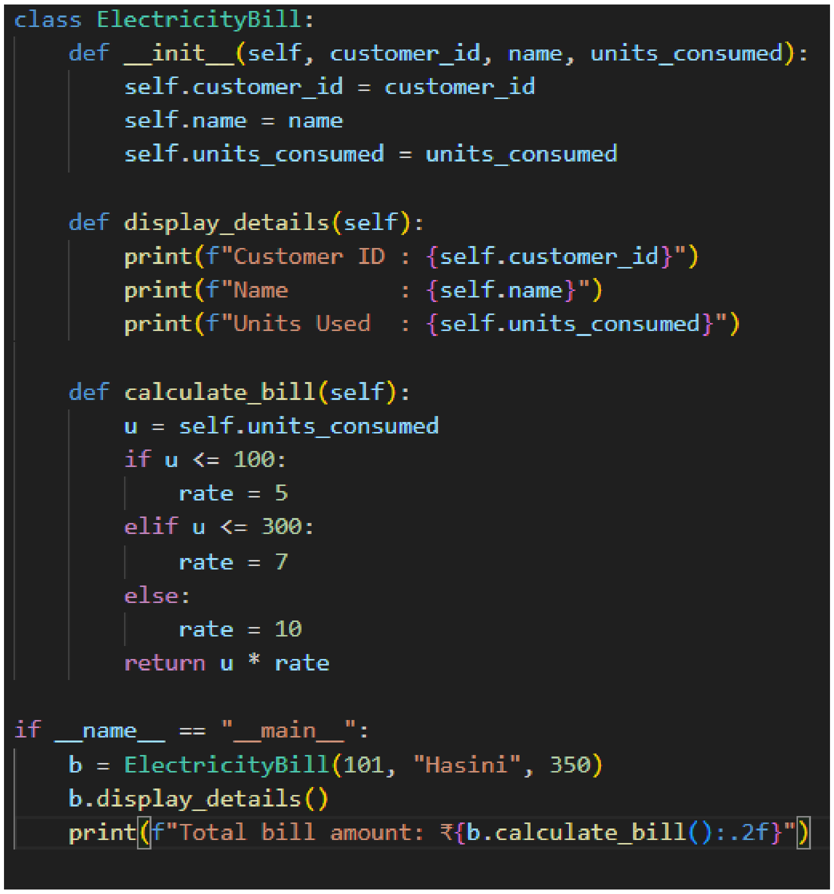
Electricity Bill Calculation- Create Python code that defines a class named `ElectricityBill` with attributes: `customer_id`, `name`, and `units_consumed`. Implement a method `display_details()` to print customer details, and a method `calculate_bill()` where:

- Units ≤ 100 → ₹5 per unit

- 101 to 300 units → ₹7 per unit

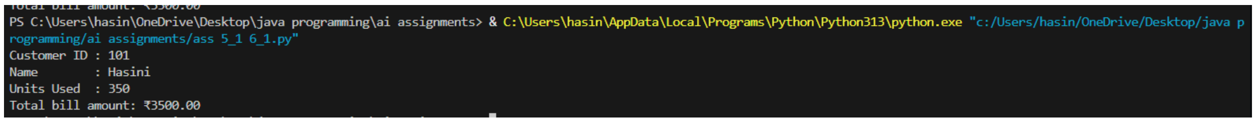
- More than 300 units → ₹10 per unit

Create a bill object, display details, and print the total bill amount.



```python
class ElectricityBill:
    def __init__(self, customer_id, name, units_consumed):
        self.customer_id = customer_id
        self.name = name
        self.units_consumed = units_consumed

    def display_details(self):
        print(f"Customer ID : {self.customer_id}")
        print(f"Name        : {self.name}")
        print(f"Units Used  : {self.units_consumed}")

    def calculate_bill(self):
        u = self.units_consumed
        if u <= 100:
            rate = 5
        elif u <= 300:
            rate = 7
        else:
            rate = 10
        return u * rate

if __name__ == "__main__":
    b = ElectricityBill(101, "Hasini", 350)
    b.display_details()
    print(f"Total bill amount: ₹{b.calculate_bill():.2f}")
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Customer ID : 101
Name        : Hasini
Units Used  : 350
Total bill amount: ₹3500.00
```

Task 3:

Product Discount Calculation- Create Python code that defines a

class named `Product` with attributes: `product_id`, `product_name`,

`price`, and `category`. Implement a method `display_details()` to print product details. Implement another method

`calculate_discount()` where:

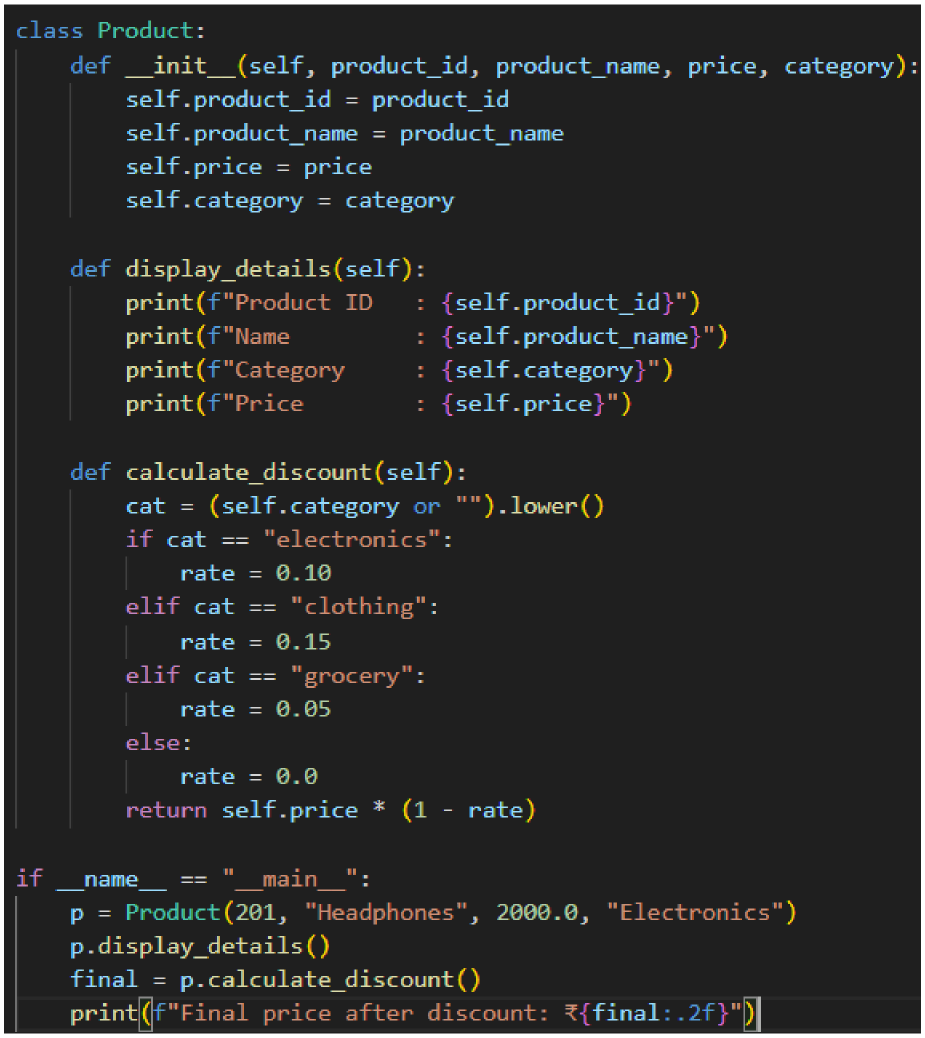
- Electronics → 10% discount

- Clothing → 15% discount

- Grocery → 5% discount

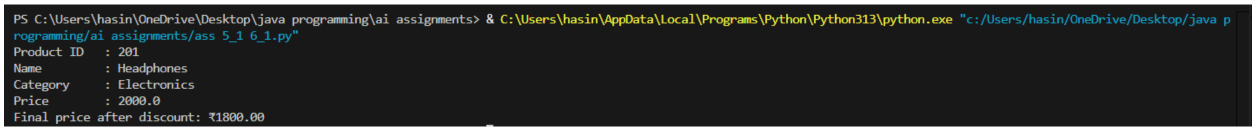
Create at least one product object, display details, and print the final

price after discount.



```python
class Product:
    def __init__(self, product_id, product_name, price, category):
        self.product_id = product_id
        self.product_name = product_name
        self.price = price
        self.category = category

    def display_details(self):
        print(f"Product ID   : {self.product_id}")
        print(f"Name         : {self.product_name}")
        print(f"Category     : {self.category}")
        print(f"Price        : {self.price}")

    def calculate_discount(self):
        cat = (self.category or "").lower()
        if cat == "electronics":
            rate = 0.10
        elif cat == "clothing":
            rate = 0.15
        elif cat == "grocery":
            rate = 0.05
        else:
            rate = 0.0
        return self.price * (1 - rate)

if __name__ == "__main__":
    p = Product(201, "Headphones", 2000.0, "Electronics")
    p.display_details()
    final = p.calculate_discount()
    print(f"Final price after discount: ₹{final:.2f}")
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Product ID   : 201
Name         : Headphones
Category     : Electronics
Price        : 2000.0
Final price after discount: ₹1800.00
```

Task 4:

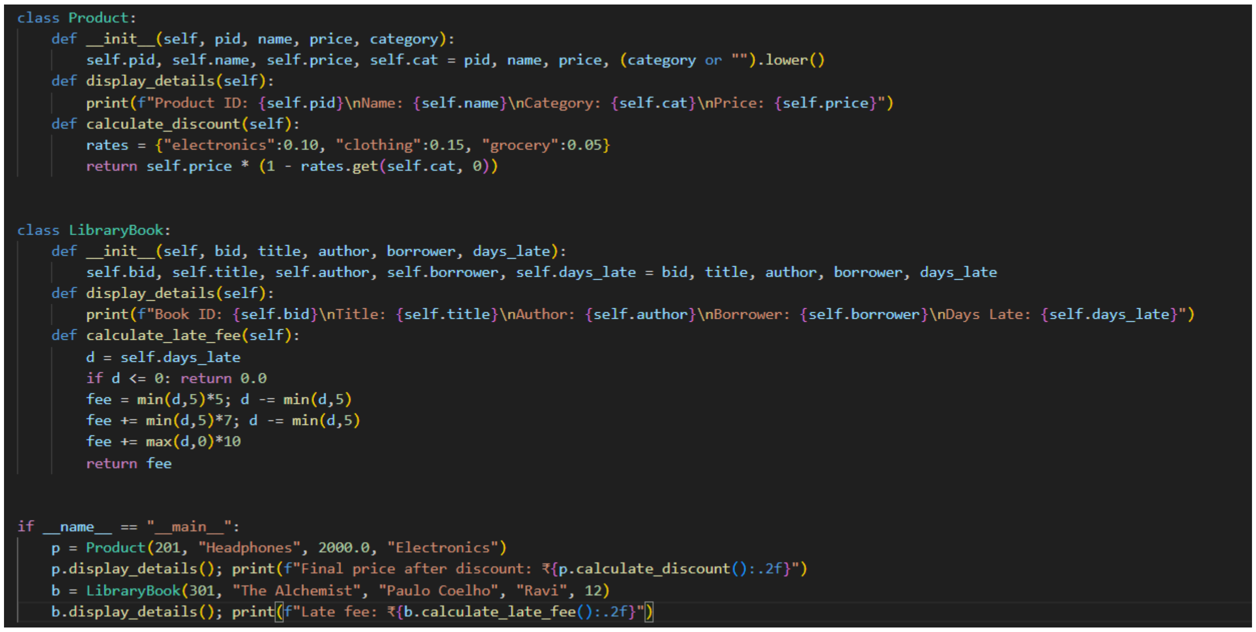
Book Late Fee Calculation- Create Python code that defines a class named `LibraryBook` with attributes: `book_id`, `title`, `author`, `borrower`, and `days_late`. Implement a method `display_details()` to print book details, and a method `calculate_late_fee()` where:

- Days late ≤ 5 → ₹5 per day

- 6 to 10 days late → ₹7 per day

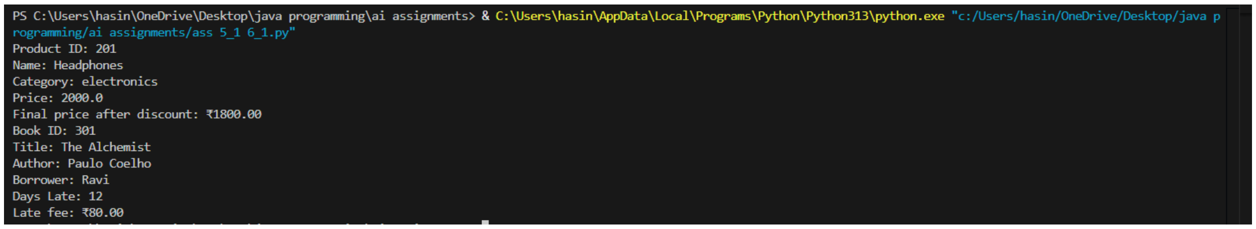
- More than 10 days late → ₹10 per day

Create a book object, display details, and print the late fee.



```python
class Product:
    def __init__(self, pid, name, price, category):
        self.pid, self.name, self.price, self.cat = pid, name, price, (category or "").lower()
    def display_details(self):
        print(f"Product ID: {self.pid}\nName: {self.name}\nCategory: {self.cat}\nPrice: {self.price}")
    def calculate_discount(self):
        rates = {"electronics":0.10, "clothing":0.15, "grocery":0.05}
        return self.price * (1 - rates.get(self.cat, 0))


class LibraryBook:
    def __init__(self, bid, title, author, borrower, days_late):
        self.bid, self.title, self.author, self.borrower, self.days_late = bid, title, author, borrower, days_late
    def display_details(self):
        print(f"Book ID: {self.bid}\nTitle: {self.title}\nAuthor: {self.author}\nBorrower: {self.borrower}\nDays Late: {self.days_late}")
    def calculate_late_fee(self):
        d = self.days_late
        if d <= 0: return 0.0
        fee = min(d,5)*5; d -= min(d,5)
        fee += min(d,5)*7; d -= min(d,5)
        fee += max(d,0)*10
        return fee


if __name__ == "__main__":
    p = Product(201, "Headphones", 2000.0, "Electronics")
    p.display_details(); print(f"Final price after discount: ₹{p.calculate_discount():.2f}")
    b = LibraryBook(301, "The Alchemist", "Paulo Coelho", "Ravi", 12)
    b.display_details(); print(f"Late fee: ₹{b.calculate_late_fee():.2f}")
```

OUTPUT:



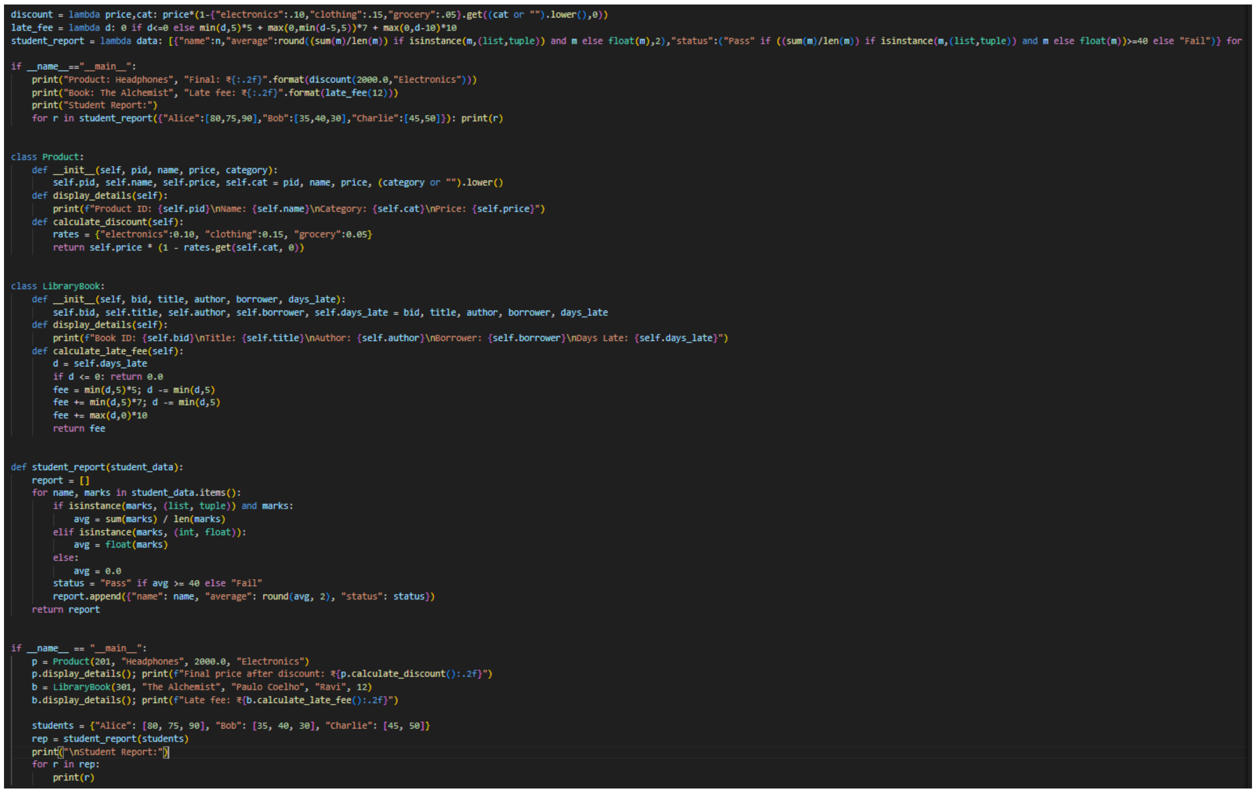
```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Product ID: 201
Name: Headphones
Category: electronics
Price: 2000.0
Final price after discount: ₹1800.00
Book ID: 301
Title: The Alchemist
Author: Paulo Coelho
Borrower: Ravi
Days Late: 12
Late fee: ₹80.00
```

Task 5:

Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing student names and their marks. The function should:

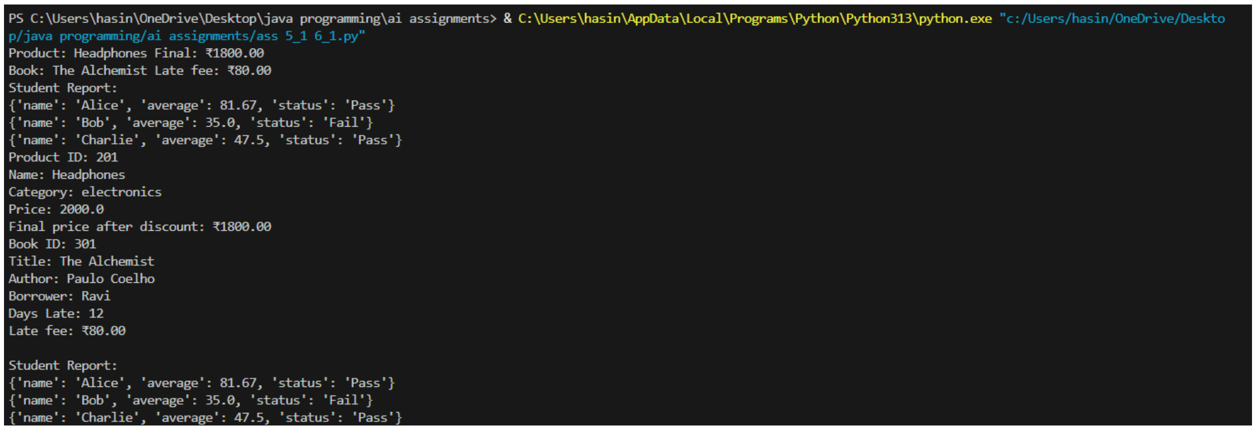
- Calculate the average score for each student
- Determine pass/fail status (pass ≥ 40)
- Return a summary report as a list of dictionaries

Use Copilot suggestions as you build the function and format the output.



```python
discount = lambda price,cat: price*(1-{"electronics":.10,"clothing":.15,"grocery":.05}.get((cat or "").lower(),0))
late_fee = lambda d: 0 if d<=0 else min(d,5)*5 + max(0,min(d-5,5))*7 + max(0,d-10)*10
student_report = lambda data: [{"name":n,"average":round((sum(m)/len(m)) if isinstance(m,(list,tuple)) and m else float(m),2),"status":("Pass" if ((sum(m)/len(m)) if isinstance(m,(list,tuple)) and m else float(m))>=40 else "Fail")} for

if __name__=="__main__":
    print("Product: Headphones", "Final: ₹{:.2f}".format(discount(2000.0,"Electronics")))
    print("Book: The Alchemist", "Late fee: ₹{:.2f}".format(late_fee(12)))
    print("Student Report:")
    for r in student_report({"Alice":[80,75,90],"Bob":[35,40,30],"Charlie":[45,50]}): print(r)


class Product:
    def __init__(self, pid, name, price, category):
        self.pid, self.name, self.price, self.cat = pid, name, price, (category or "").lower()
    def display_details(self):
        print(f"Product ID: {self.pid}\nName: {self.name}\nCategory: {self.cat}\nPrice: {self.price}")
    def calculate_discount(self):
        rates = {"electronics":0.10, "clothing":0.15, "grocery":0.05}
        return self.price * (1 - rates.get(self.cat, 0))


class LibraryBook:
    def __init__(self, bid, title, author, borrower, days_late):
        self.bid, self.title, self.author, self.borrower, self.days_late = bid, title, author, borrower, days_late
    def display_details(self):
        print(f"Book ID: {self.bid}\nTitle: {self.title}\nAuthor: {self.author}\nBorrower: {self.borrower}\nDays Late: {self.days_late}")
    def calculate_late_fee(self):
        d = self.days_late
        if d <= 0: return 0.0
        fee = min(d,5)*5; d -= min(d,5)
        fee += min(d,5)*7; d -= min(d,5)
        fee += max(d,0)*10
        return fee


def student_report(student_data):
    report = []
    for name, marks in student_data.items():
        if isinstance(marks, (list, tuple)) and marks:
            avg = sum(marks) / len(marks)
        elif isinstance(marks, (int, float)):
            avg = float(marks)
        else:
            avg = 0.0
        status = "Pass" if avg >= 40 else "Fail"
        report.append({"name": name, "average": round(avg, 2), "status": status})
    return report


if __name__ == "__main__":
    p = Product(201, "Headphones", 2000.0, "Electronics")
    p.display_details(); print(f"Final price after discount: ₹{p.calculate_discount():.2f}")
    b = LibraryBook(301, "The Alchemist", "Paulo Coelho", "Ravi", 12)
    b.display_details(); print(f"Late fee: ₹{b.calculate_late_fee():.2f}")

    students = {"Alice": [80, 75, 90], "Bob": [35, 40, 30], "Charlie": [45, 50]}
    rep = student_report(students)
    print("\nStudent Report:")
    for r in rep:
        print(r)
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Deskto
p/java programming/ai assignments/ass 5_1 6_1.py"
Product: Headphones Final: ₹1800.00
Book: The Alchemist Late fee: ₹80.00
Student Report:
{'name': 'Alice', 'average': 81.67, 'status': 'Pass'}
{'name': 'Bob', 'average': 35.0, 'status': 'Fail'}
{'name': 'Charlie', 'average': 47.5, 'status': 'Pass'}
Product ID: 201
Name: Headphones
Category: electronics
Price: 2000.0
Final price after discount: ₹1800.00
Book ID: 301
Title: The Alchemist
Author: Paulo Coelho
Borrower: Ravi
Days Late: 12
Late fee: ₹80.00

Student Report:
{'name': 'Alice', 'average': 81.67, 'status': 'Pass'}
{'name': 'Bob', 'average': 35.0, 'status': 'Fail'}
{'name': 'Charlie', 'average': 47.5, 'status': 'Pass'}
```

Task 6:

Taxi Fare Calculation-Create Python code that defines a class named `TaxiRide` with attributes: `ride_id`, `driver_name`, `distance_km`, and `waiting_time_min`. Implement a method `display_details()` to print ride details, and a method `calculate_fare()` where:

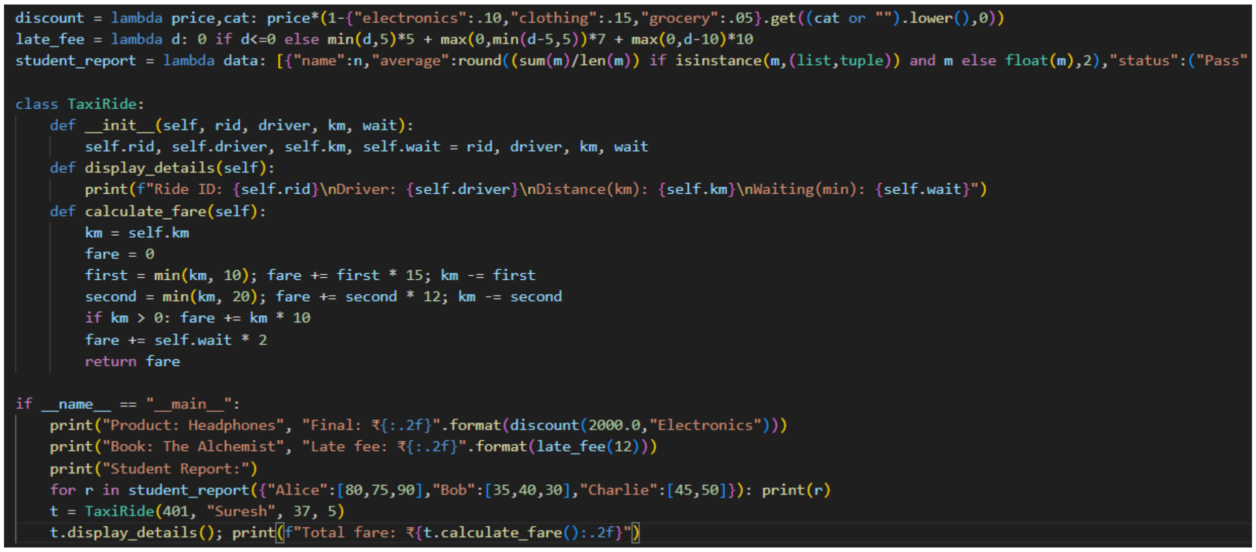
- ₹15 per km for the first 10 km

- ₹12 per km for the next 20 km

- ₹10 per km above 30 km

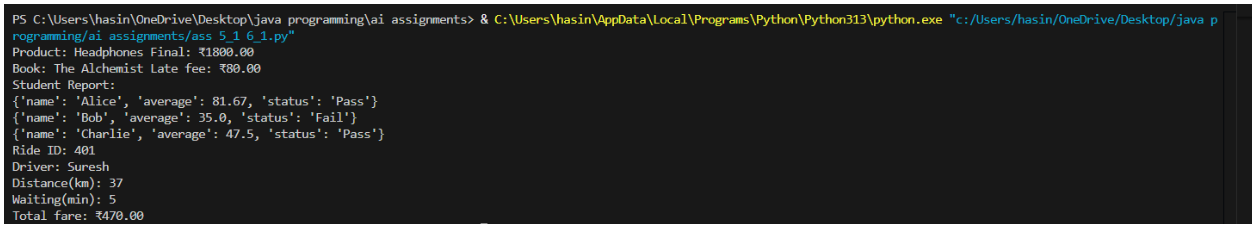
- Waiting charge: ₹2 per minute

Create a ride object, display details, and print the total fare.



```python
discount = lambda price,cat: price*(1-{"electronics":.10,"clothing":.15,"grocery":.05}.get((cat or "").lower(),0))
late_fee = lambda d: 0 if d<=0 else min(d,5)*5 + max(0,min(d-5,5))*7 + max(0,d-10)*10
student_report = lambda data: [{"name":n,"average":round((sum(m)/len(m)) if isinstance(m,(list,tuple)) and m else float(m),2),"status":("Pass"

class TaxiRide:
    def __init__(self, rid, driver, km, wait):
        self.rid, self.driver, self.km, self.wait = rid, driver, km, wait
    def display_details(self):
        print(f"Ride ID: {self.rid}\nDriver: {self.driver}\nDistance(km): {self.km}\nWaiting(min): {self.wait}")
    def calculate_fare(self):
        km = self.km
        fare = 0
        first = min(km, 10); fare += first * 15; km -= first
        second = min(km, 20); fare += second * 12; km -= second
        if km > 0: fare += km * 10
        fare += self.wait * 2
        return fare

if __name__ == "__main__":
    print("Product: Headphones", "Final: ₹{:.2f}".format(discount(2000.0,"Electronics")))
    print("Book: The Alchemist", "Late fee: ₹{:.2f}".format(late_fee(12)))
    print("Student Report:")
    for r in student_report({"Alice":[80,75,90],"Bob":[35,40,30],"Charlie":[45,50]}): print(r)
    t = TaxiRide(401, "Suresh", 37, 5)
    t.display_details(); print(f"Total fare: ₹{t.calculate_fare():.2f}")
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Product: Headphones Final: ₹1800.00
Book: The Alchemist Late fee: ₹80.00
Student Report:
{'name': 'Alice', 'average': 81.67, 'status': 'Pass'}
{'name': 'Bob', 'average': 35.0, 'status': 'Fail'}
{'name': 'Charlie', 'average': 47.5, 'status': 'Pass'}
Ride ID: 401
Driver: Suresh
Distance(km): 37
Waiting(min): 5
Total fare: ₹470.00
```

Task 7:

Statistics Subject Performance - Create a Python function

`statistics_subject(scores_list)` that accepts a list of 60 student scores

and computes key performance statistics. The function should return the following:

- Highest score in the class

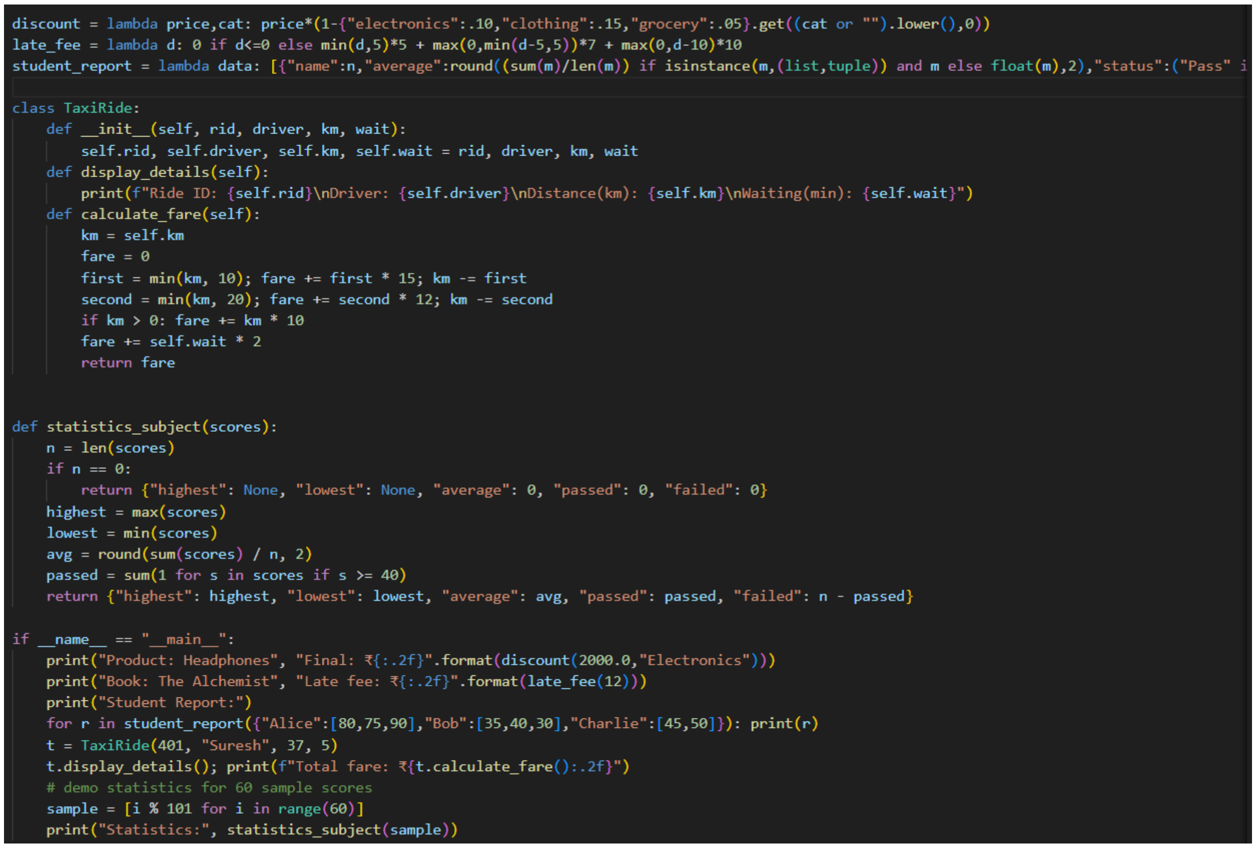
- Lowest score in the class

- Class average score

- Number of students passed (score ≥ 40)

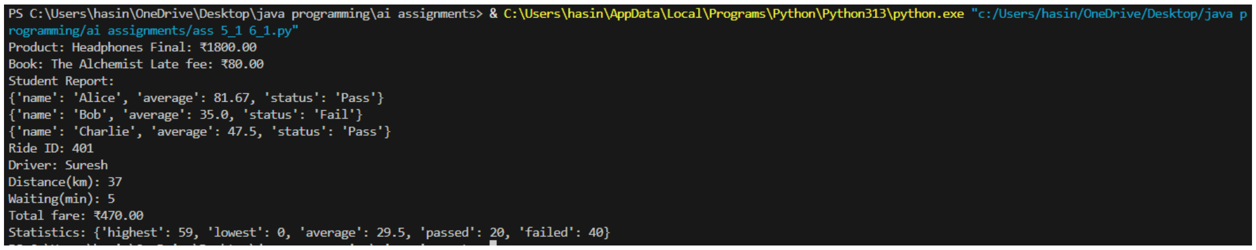
- Number of students failed (score < 40)

Allow Copilot to assist with aggregations and logic



```python
discount = lambda price,cat: price*(1-{"electronics":.10,"clothing":.15,"grocery":.05}.get((cat or "").lower(),0))
late_fee = lambda d: 0 if d<=0 else min(d,5)*5 + max(0,min(d-5,5))*7 + max(0,d-10)*10
student_report = lambda data: [{"name":n,"average":round((sum(m)/len(m)) if isinstance(m,(list,tuple)) and m else float(m),2),"status":("Pass" i

class TaxiRide:
    def __init__(self, rid, driver, km, wait):
        self.rid, self.driver, self.km, self.wait = rid, driver, km, wait
    def display_details(self):
        print(f"Ride ID: {self.rid}\nDriver: {self.driver}\nDistance(km): {self.km}\nWaiting(min): {self.wait}")
    def calculate_fare(self):
        km = self.km
        fare = 0
        first = min(km, 10); fare += first * 15; km -= first
        second = min(km, 20); fare += second * 12; km -= second
        if km > 0: fare += km * 10
        fare += self.wait * 2
        return fare


def statistics_subject(scores):
    n = len(scores)
    if n == 0:
        return {"highest": None, "lowest": None, "average": 0, "passed": 0, "failed": 0}
    highest = max(scores)
    lowest = min(scores)
    avg = round(sum(scores) / n, 2)
    passed = sum(1 for s in scores if s >= 40)
    return {"highest": highest, "lowest": lowest, "average": avg, "passed": passed, "failed": n - passed}

if __name__ == "__main__":
    print("Product: Headphones", "Final: ₹{:.2f}".format(discount(2000.0,"Electronics")))
    print("Book: The Alchemist", "Late fee: ₹{:.2f}".format(late_fee(12)))
    print("Student Report:")
    for r in student_report({"Alice":[80,75,90],"Bob":[35,40,30],"Charlie":[45,50]}): print(r)
    t = TaxiRide(401, "Suresh", 37, 5)
    t.display_details(); print(f"Total fare: ₹{t.calculate_fare():.2f}")
    # demo statistics for 60 sample scores
    sample = [i % 101 for i in range(60)]
    print("Statistics:", statistics_subject(sample))
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Product: Headphones Final: ₹1800.00
Book: The Alchemist Late fee: ₹80.00
Student Report:
{'name': 'Alice', 'average': 81.67, 'status': 'Pass'}
{'name': 'Bob', 'average': 35.0, 'status': 'Fail'}
{'name': 'Charlie', 'average': 47.5, 'status': 'Pass'}
Ride ID: 401
Driver: Suresh
Distance(km): 37
Waiting(min): 5
Total fare: ₹470.00
Statistics: {'highest': 59, 'lowest': 0, 'average': 29.5, 'passed': 20, 'failed': 40}
```

Task Description #8 (Transparency in Algorithm Optimization)

Task: Use AI to generate two solutions for checking prime numbers:

• Naive approach(basic)

• Optimized approach

Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

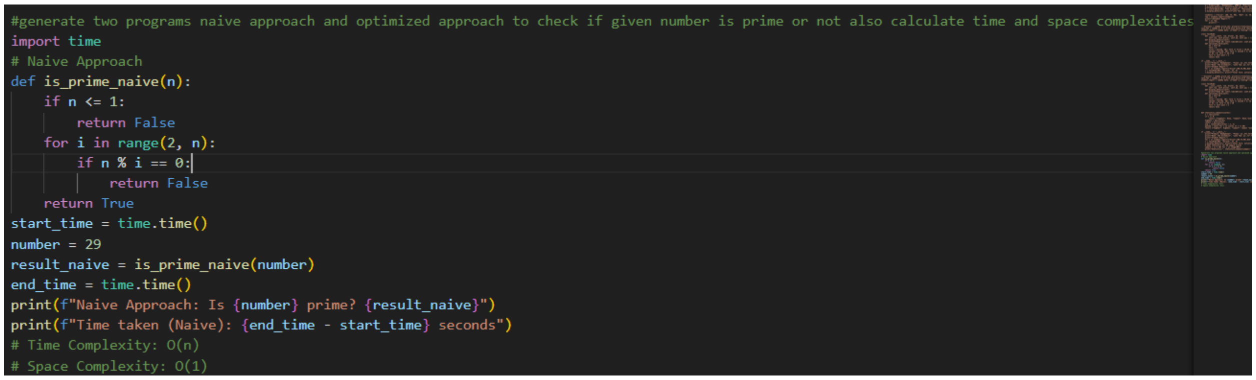
Expected Output:

• Code for both methods.

• Transparent explanation of time complexity.

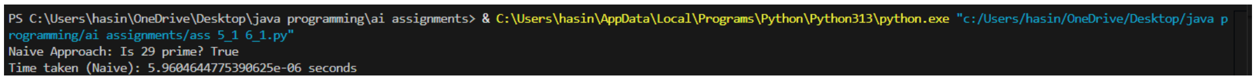
• Comparison highlighting efficiency improvements.

## NAIVE APPROACH:



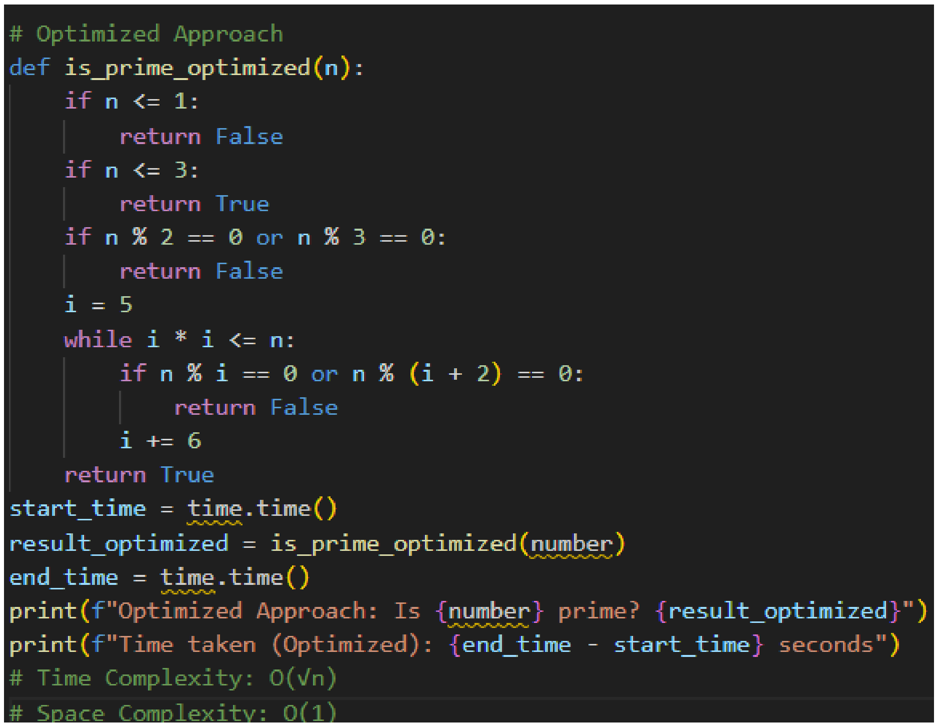
```python
#generate two programs naive approach and optimized approach to check if given number is prime or not also calculate time and space complexities
import time
# Naive Approach
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
start_time = time.time()
number = 29
result_naive = is_prime_naive(number)
end_time = time.time()
print(f"Naive Approach: Is {number} prime? {result_naive}")
print(f"Time taken (Naive): {end_time - start_time} seconds")
# Time Complexity: O(n)
# Space Complexity: O(1)
```

OUTPUT:



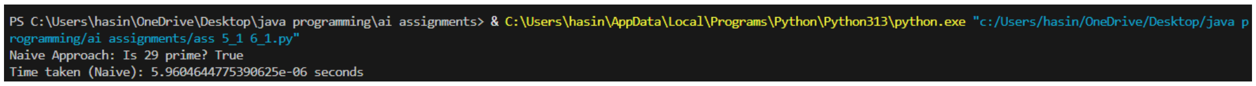
```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Naive Approach: Is 29 prime? True
Time taken (Naive): 5.9604644775390625e-06 seconds
```

## OPTIMIZED APPROACH:



```python
# Optimized Approach
def is_prime_optimized(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
start_time = time.time()
result_optimized = is_prime_optimized(number)
end_time = time.time()
print(f"Optimized Approach: Is {number} prime? {result_optimized}")
print(f"Time taken (Optimized): {end_time - start_time} seconds")
# Time Complexity: O(√n)
# Space Complexity: O(1)
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Naive Approach: Is 29 prime? True
Time taken (Naive): 5.9604644775390625e-06 seconds
```

## Explanation:

- A number does not need to be checked beyond its square root.
- Even numbers greater than 2 are skipped.
- Only odd divisors are checked, reducing iterations.

## Time Complexity:

- **O(√n)** — significantly fewer checks compared to naive method.

## Comparision and highlighting efficiency improvement:

| **Method** | **Time Complexity** | **Efficiency** |
|---|---|---|
| Naive | O(n) | Slower for large numbers |
| Optimized | O(√n) | Faster and scalable |

Task Description #9 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate

Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.

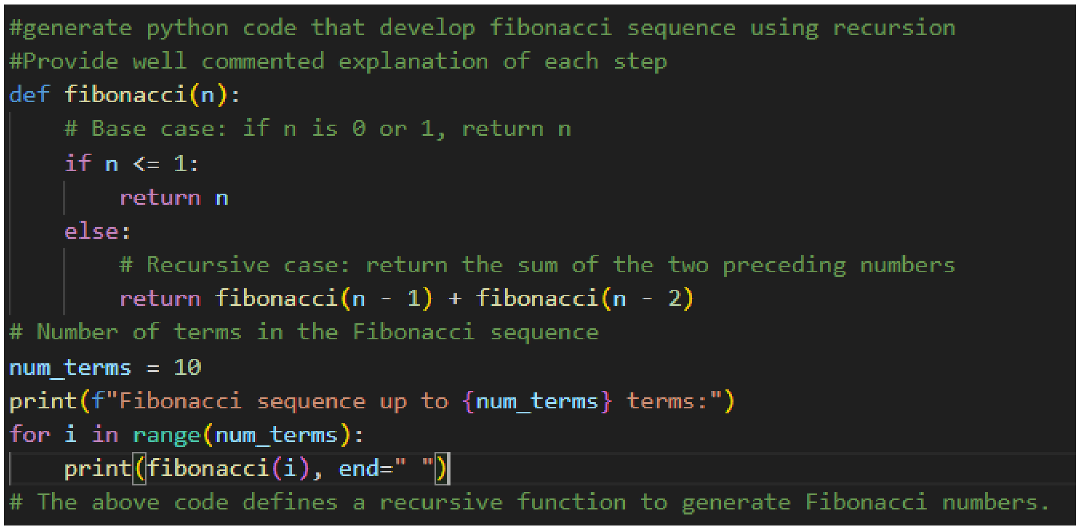
2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.

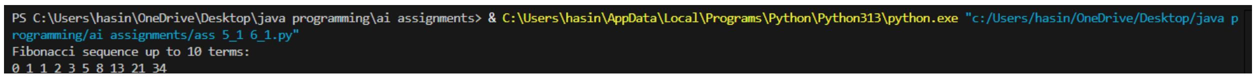
• Clear explanation of how recursion works.

• Verification that explanation matches actual execution.



```python
#generate python code that develop fibonacci sequence using recursion
#Provide well commented explanation of each step
def fibonacci(n):
    # Base case: if n is 0 or 1, return n
    if n <= 1:
        return n
    else:
        # Recursive case: return the sum of the two preceding numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Number of terms in the Fibonacci sequence
num_terms = 10
print(f"Fibonacci sequence up to {num_terms} terms:")
for i in range(num_terms):
    print(fibonacci(i), end=" ")
# The above code defines a recursive function to generate Fibonacci numbers.
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
Fibonacci sequence up to 10 terms:
0 1 1 2 3 5 8 13 21 34
```

## Explanation Of Recursion:

**What recursion is:**
Recursion is a technique where a function **calls itself** to solve a smaller version of the same problem.

In this function:

- `fibonacci(n)` depends on:
    - `fibonacci(n-1)`
    - `fibonacci(n-2)`

## Verification With Actual Execution:

```
Example: fibonacci(4)

fibonacci(4)
= fibonacci(3) + fibonacci(2)
= (fibonacci(2) + fibonacci(1)) + (fibonacci(1) + fibonacci(0))
= (1 + 1) + (1 + 0)
= 3


This matches the Fibonacci sequence:
 0, 1, 1, 2, 3
```

Task Description #10 (Transparency in Error Handling)

Task: Use AI to generate a Python program that reads a file and

processes data.

Prompt:

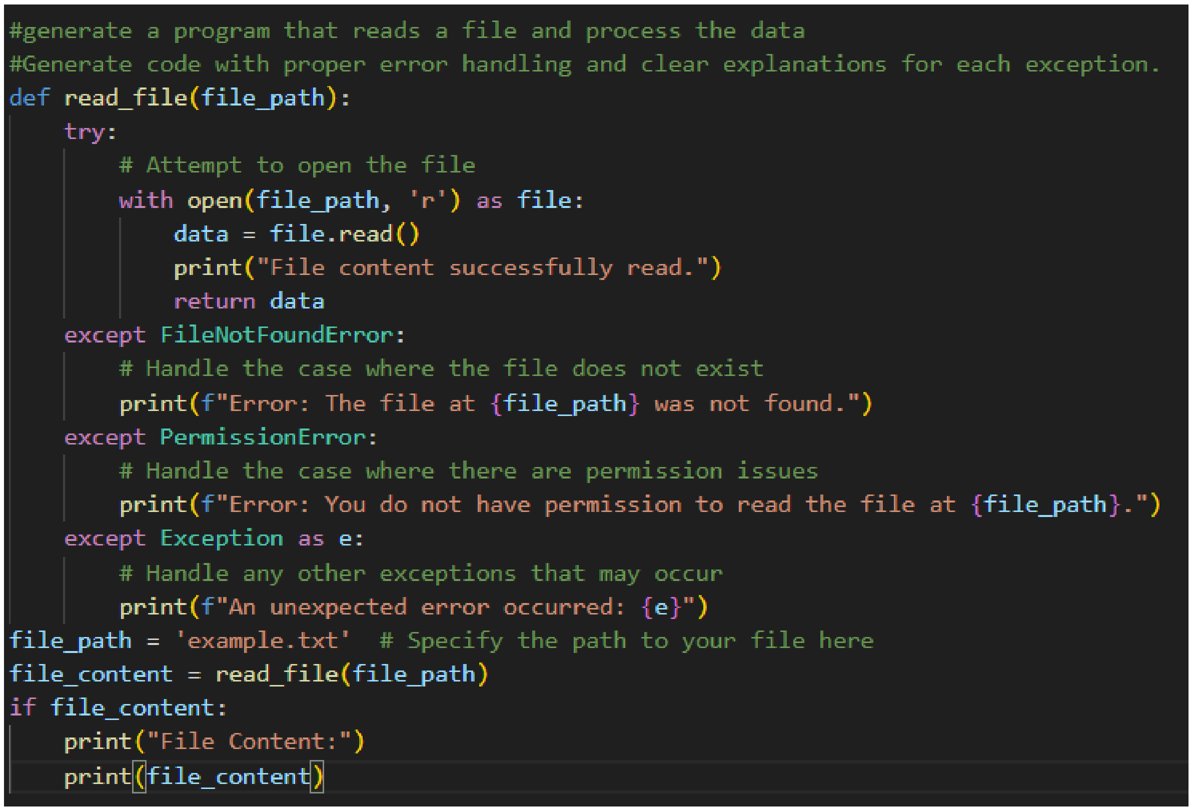
“Generate code with proper error handling and clear explanations for each exception."

Expected Output:

• Code with meaningful exception handling.

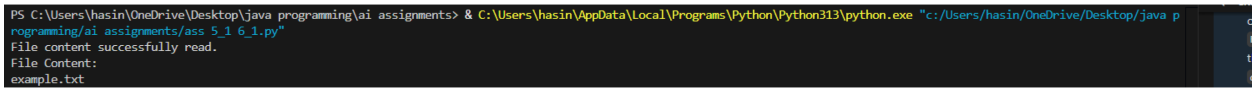
• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior.



```python
#generate a program that reads a file and process the data
#Generate code with proper error handling and clear explanations for each exception.
def read_file(file_path):
    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            data = file.read()
            print("File content successfully read.")
            return data
    except FileNotFoundError:
        # Handle the case where the file does not exist
        print(f"Error: The file at {file_path} was not found.")
    except PermissionError:
        # Handle the case where there are permission issues
        print(f"Error: You do not have permission to read the file at {file_path}.")
    except Exception as e:
        # Handle any other exceptions that may occur
        print(f"An unexpected error occurred: {e}")
file_path = 'example.txt'  # Specify the path to your file here
file_content = read_file(file_path)
if file_content:
    print("File Content:")
    print(file_content)
```

OUTPUT:



```
PS C:\Users\hasin\OneDrive\Desktop\java programming\ai assignments> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java p
rogramming/ai assignments/ass 5_1 6_1.py"
File content successfully read.
File Content:
example.txt
```

## Explanation of Error Scenarios

- **FileNotFoundError**
  Occurs when the specified file does not exist at the given path.

- **PermissionError**
  Occurs when the file exists but the program does not have permission to read it.
- **Exception**
  Catches any other unexpected error that may occur during execution.

## Runtime Behavior Validation

- When the file exists and is readable → `try` block runs, `else` executes.
- When the file is missing → `FileNotFoundError` block executes.
- When access is denied → `PermissionError` block executes.
- In all cases → `finally` block executes.