Name: T.Phanindra

H.NO: 2303A51703

Batch: 24

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.

Code:

```
class Employee:
    def __init__(self,empid,empname,designation,salary,experience):
        self.empid=empid
        self.empname=empname
        self.designation=designation
        self.salary=salary
        self.experience=experience
    def display_details(self):
        print(f"Employee ID: {self.empid}")
        print(f"Employee Name: {self.empname}")
        print(f"Designation: {self.designation}")
        print(f"Salary: {self.salary}")
```

```python
        print(f"Experience: {self.experience} years")
    def caclculate_allowance(self):
        if self.experience >10:
            allowance = 0.20 * self.salary
        elif 5<=self.experience <=10:
            allowance = 0.10 * self.salary
        else:
            allowance = 0.05 * self.salary
        print(f"Allowance: {allowance}")
        print(f"Total Salary including Allowance: {self.salary + allowance}")
emp1 = Employee(101, "Alice Smith", "Manager", 80000,12)
emp1.display_details()
emp1.caclculate_allowance()
```

output: Employee ID: 101

Employee Name: Alice Smith

Designation: Manager

Salary: 80000

Experience: 12 years

Allowance: 16000.0

Total Salary including Allowance: 96000.0

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.

Code:

```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 calculate_bill(self):
        if self.units_consumed <= 100:
            bill_amount = self.units_consumed * 5
        elif 101<=self.units_consumed <= 300:
            bill_amount =self.units_consumed  * 7
        else:
            bill_amount = self.units_consumed * 10
        return bill_amount
    def display_bill(self):
        bill_amount = self.calculate_bill()
        print(f"Customer ID: {self.customer_id}")
        print(f"Customer Name: {self.name}")
        print(f"Units Consumed: {self.units_consumed}")
        print(f"Total Bill Amount: {bill_amount} INR")
# Example usage
obj1=ElectricityBill("C001","Alice",150)
obj1.display_bill()
obj2=ElectricityBill("C002","Bob",350)
obj2.display_bill()
```

output:

Customer ID: C001

Customer Name: Alice

Units Consumed: 150

Total Bill Amount: 1050 INR

Customer ID: C002

Customer Name: Bob

Units Consumed: 350

Total Bill Amount: 3500 INR

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.

Code:

```
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"Product Name: {self.product_name}")
        print(f"Price: {self.price}")
        print(f"Category: {self.category}")
    def calculate_discount(self):
```

```python
        if self.category.lower() == "electronics":

            discount = 0.10 * self.price

        elif self.category.lower() == "clothing":

            discount = 0.15 * self.price

        else:

            discount = 0.05 * self.price

        return discount
```

 # Example usage

product1 = Product(101, "Smartphone", 500, "Electronics")

product1.display_details()

discount1 = product1.calculate_discount()

print(f"Discount on {product1.product_name}: {discount1}")

print("\n")

product2 = Product(202, "Jeans", 80, "Clothing")

product2.display_details()

output:

Product ID: 101

Product Name: Smartphone

Price: 500

Category: Electronics

Discount on Smartphone: 50.0

Product ID: 202

Product Name: Jeans

Price: 80

Category: Clothing

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.

Code:

```
class LibraryBook:
    def __init__(self,book_id,title,author,borrower,days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.borrower = borrower
        self.days_late = days_late
    def display_details(self):
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Borrower: {self.borrower}")
        print(f"Days Late: {self.days_late}")
    def calculate_late_fee(self):
        if self.days_late<= 5:
            fee=self.days_late * 5
        elif 6 <= self.days_late <= 10:
            fee=self.days_late * 7
        else:
            fee=self.days_late * 10
        return fee
```

```
# Example usage
book = LibraryBook(book_id=101, title="The Great Gatsby", author="F. Scott Fitzgerald",
borrower="Alice", days_late=8)

book.display_details()

late_fee = book.calculate_late_fee()

print(f"Late Fee: ${late_fee}")
```

output: Book ID: 101

Title: The Great Gatsby

Author: F. Scott Fitzgerald

Borrower: Alice

Days Late: 8

Late Fee: $56

Task 5: 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.

Code:

```
def student_report(student_marks):
    report=[]
    for name,marks in student_marks.items():
        avg_marks=sum(student_marks.values())/len(student_marks)
        if avg_marks>=40:
            status="Pass"
        else:
            status="Fail"
```

```
        report.append({"name":name,"Average Marks":avg_marks,"Status":status})
    return report
student_marks={"nikhil":85,"ram":78,"Sam":65,"phani":45}
report=student_report(student_marks)
for student in report:
    print(student)
```

output:

c:/Users/nikhi/OneDrive/Attachments/Desktop/AI_assist_coding/pract5.1.py

{'name': 'nikhil', 'Average Marks': 68.25, 'Status': 'Pass'}

{'name': 'ram', 'Average Marks': 68.25, 'Status': 'Pass'}

{'name': 'Sam', 'Average Marks': 68.25, 'Status': 'Pass'}

{'name': 'phani', 'Average Marks': 68.25, '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.

Code:

```
class TaxiRide:
    def __init__(self, ride_id, driver_name, distance_km, waiting_time_min):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.waiting_time_min = waiting_time_min
```

```python
    def display_details(self):
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver Name: {self.driver_name}")
        print(f"Distance (km): {self.distance_km}")
        print(f"Waiting Time (min): {self.waiting_time_min}")


    def calculate_fare(self):
        if self.distance_km <= 10:
            fare = self.distance_km * 15
        elif 11 <= self.distance_km <= 30:
            fare = (10 * 15) + (self.distance_km - 10) * 12
        else:
            fare = (10 * 15) + (20 * 12) + (self.distance_km - 30) * 10


        fare += self.waiting_time_min * 2
        return fare


ride = TaxiRide(501, "nikhilyams", 25, 10)

ride.display_details()

fare = ride.calculate_fare()

print(f"Total Fare: {fare}")
```

Output:

Ride ID: 501

Driver Name: nikhilyams

Distance (km): 25

Waiting Time (min): 10

Total Fare: 350

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

Code:
```python
def statistics_subject(score_list):
    total = sum(score_list)
    average = total / len(score_list)
    highest = max(score_list)
    lowest = min(score_list)
    passed = 0
    failed = 0
    for i in score_list:
        if i >= 40:
            passed += 1
        else:
            failed += 1
    print(f"Number of Students Passed: {passed}")
    print(f"Number of Students Failed: {failed}")
    return {
        "average": average,
        "highest": highest,
        "lowest": lowest
```

```
    }
scores = [
    28, 49, 33, 72, 15, 60, 95, 40, 53, 81, 22, 47, 68, 79,34, 91, 44, 58, 73, 38, 66, 84, 29, 50, 77, 92, 41,
    36, 65, 80, 54, 87, 30, 69, 45, 71, 39, 83, 59, 74
]
stats = statistics_subject(scores)
print(stats)
```

Output: Number of Students Passed: 30

Number of Students Failed: 10

{'average': 57.775, 'highest': 95, 'lowest': 15}

Task 8: 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.

Code: #Generate Python code for two prime-checking methods and explain how the optimized version improves performance.well commented"

```python
# Method 1: Basic Prime Checking
def is_prime_basic(n):
    """Check if a number is prime using basic method."""
    if n <= 1:
        return False
    for i in range(2, n):
```

```python
        if n % i == 0:
            return False
    return True

# Method 2: Optimized Prime Checking

def is_prime_optimized(n):
    """Check if a number is prime using optimized method."""
    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

# Example usage

number = 29

print(f"Is {number} prime? {is_prime_basic(number)}")

print(f"Is {number} prime? {is_prime_optimized(number)}")

# Explanation of Optimization:

# The optimized method reduces the number of checks needed to determine if a number is prime.

# Instead of checking all numbers up to n, it only checks up to the square root of n.

# It also skips even numbers and multiples of 3 after initial checks, significantly improving performance for larger numbers.

#Generate Python code that extracts specific information from a nested dictionary representing student data.
```

Output: Is 29 prime? True

Is 29 prime? True

Task 9:

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.

Code: #generate a recursive function to calculate fibonacci numbers with clear comments and explain base case and recursive calls

```
def fibonacci(n):

    """

    This function returns the nth Fibonacci number using recursion.


    Base Case:

    If n is 0, return 0.

    If n is 1, return 1.


    Recursive Case:

    For n greater than 1, return the sum of the two preceding Fibonacci numbers:
```

```python
        fibonacci(n-1) + fibonacci(n-2)
    """

    if n == 0:
        return 0  # Base case: the 0th Fibonacci number is 0
    elif n == 1:
        return 1  # Base case: the 1st Fibonacci number is 1
    else:
        # Recursive case: sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage
num = 10
print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
```

Output: The 10th Fibonacci number is: 55

Task 10:

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.

Code: #Generate code with proper error handling and clear explanations for each exception.

Generate a python code that reads a file and process data with well commented

```python
def read_and_process_file(file_path):
    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            data = file.readlines()

        # Process the data (for example, converting to uppercase)
        processed_data = [line.strip().upper() for line in data]

        return processed_data

    except FileNotFoundError:
        # Handle the case where the file does not exist
        return "Error: The file was not found. Please check the file path."

    except IOError:
        # Handle other I/O errors
        return "Error: An I/O error occurred while trying to read the file."

    except Exception as e:
        # Handle any other exceptions that may occur
        return f"An unexpected error occurred: {str(e)}"
# Example usage
file_path = 'example.txt'
result = read_and_process_file(file_path)
if isinstance(result, list):
    for line in result:
        print(line)
else:
```

```
    print(result)  # Print the error message if an error occurred
```

Output: Error: The file was not found. Please check the file path.