Lab-9.2

Course : AI Assisted Coding

Topic:- Documentation Generation: Automatic Documentation and Code Comments

Student Details:

Name : Mohammed Sabir

Hall Ticket No : 2303A51506

Batch : 22

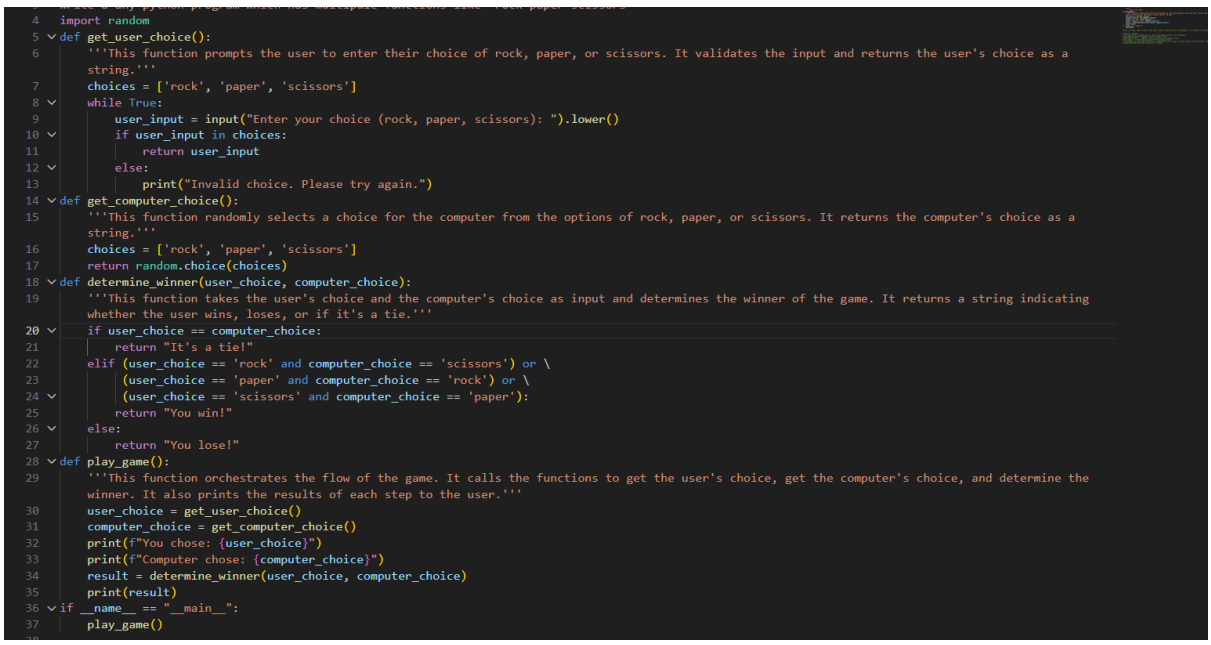
Task-01

Final Optimal Prompt:

include detailed sumaarie for every funtion in the program

write a any python program which has multipule functions like rock paper scissors

Code Screenshot :



```python
import random
def get_user_choice():
    '''This function prompts the user to enter their choice of rock, paper, or scissors. It validates the input and returns the user's choice as a string.'''
    choices = ['rock', 'paper', 'scissors']
    while True:
        user_input = input("Enter your choice (rock, paper, scissors): ").lower()
        if user_input in choices:
            return user_input
        else:
            print("Invalid choice. Please try again.")
def get_computer_choice():
    '''This function randomly selects a choice for the computer from the options of rock, paper, or scissors. It returns the computer's choice as a string.'''
    choices = ['rock', 'paper', 'scissors']
    return random.choice(choices)
def determine_winner(user_choice, computer_choice):
    '''This function takes the user's choice and the computer's choice as input and determines the winner of the game. It returns a string indicating whether the user wins, loses, or if it's a tie.'''
    if user_choice == computer_choice:
        return "It's a tie!"
    elif (user_choice == 'rock' and computer_choice == 'scissors') or \
         (user_choice == 'paper' and computer_choice == 'rock') or \
         (user_choice == 'scissors' and computer_choice == 'paper'):
        return "You win!"
    else:
        return "You lose!"
def play_game():
    '''This function orchestrates the flow of the game. It calls the functions to get the user's choice, get the computer's choice, and determine the winner. It also prints the results of each step to the user.'''
    user_choice = get_user_choice()
    computer_choice = get_computer_choice()
    print(f"You chose: {user_choice}")
    print(f"Computer chose: {computer_choice}")
    result = determine_winner(user_choice, computer_choice)
    print(result)
if __name__ == "__main__":
    play_game()
```

Output Screenshot:

Summary:- # This is a Rock, Paper, Scissors game where a player competes against the computer. The program is well-documented with four main functions:

Function Purpose

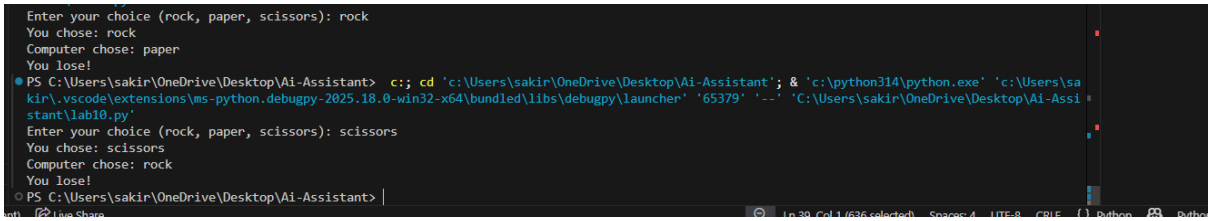
get_user_choice() Prompts user for input (rock/paper/scissors) with validation

get_computer_choice() Randomly selects computer's move

determine_winner() Compares choices and returns win/lose/tie result

play_game() Orchestrates the game flow and displays results

The program runs when executed directly, prompting the user to make a choice, displays both selections, and announces the winner. Each function includes detailed docstrings explaining its purpose



Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program implements a simple Rock, Paper, Scissors game using functions and modular design. It uses the random module to generate the computer's choice randomly from the available options. The get_user_choice() function ensures valid input by repeatedly prompting the user until a correct choice is entered. The determine_winner() function contains the game logic, comparing user and computer choices to decide the result. The play_game() function coordinates the overall flow of the game and displays results. Overall, the program demonstrates good use of functions, input validation, conditional logic, and basic game development structure in Python.

Task-02

Final Optimal Prompt:

write any python program which has conditional statements and loops and include detailed documentation for each function in the program

Code Screenshot :

```python
def calculate_factorial(n):
    '''This function calculates the factorial of a given non-negative integer n using a loop. It returns the
    factorial value. If n is negative, it raises a ValueError.'''
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    factorial = 1
    for i in range(1, n + 1):
        factorial *= i
    return factorial
def main():
    '''This is the main function that prompts the user to enter a non-negative integer, calls the calculate_factorial
    function to compute the factorial, and prints the result. It also handles invalid input by catching exceptions.'''
    user_input = input("Enter a non-negative integer to calculate its factorial: ")
    try:
        number = int(user_input)
        result = calculate_factorial(number)
        print(f"The factorial of {number} is {result}.")
    except ValueError as e:
        print(e)
if __name__ == "__main__":
    main()
```

Output Screenshot:

This program calculates the factorial of a non-negative integer entered by the user. It consists of two main functions:

Function Purpose

calculate_factorial(n) Calculates factorial using a loop and handles negative input

main() Handles user input, calls the factorial function, and manages exceptions

The program prompts the user for input, computes the factorial, and displays the result. If the user enters a negative number or invalid input, it provides an appropriate error message. Each function is documented with detailed docstrings explaining its functionality.

```
led\libs\debugpy\launcher' '51399' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab10.py'
Enter a non-negative integer to calculate its factorial: 5
The factorial of 5 is 120.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\deb
py\launcher' '51417' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab10.py'
Enter a non-negative integer to calculate its factorial: -64
Factorial is not defined for negative numbers.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program calculates the factorial of a non-negative integer using an iterative approach. The calculate_factorial() function first checks whether the input number is negative and raises a ValueError if so, ensuring proper validation. It then initializes a variable factorial to 1 and multiplies it sequentially by numbers from 1 to n using a loop. The main() function

handles user interaction by taking input, converting it to an integer, and managing errors using a try-except block. Docstrings are included for documentation. Overall, the program demonstrates good use of loops, exception handling, and structured programming practices.

Task-03

Final Optimal Prompt:

prime_checker.py

Write a brief high-level overview summarizing the purpose and functionality of this entire Python file

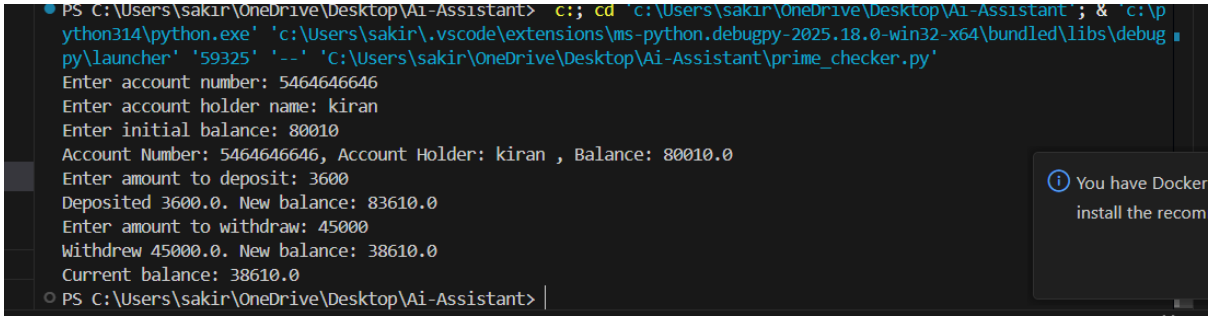
---

Code Screenshot :

```python

class BankAccount:
    def __init__(self, account_number, account_holder, balance):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"Deposited {float(amount)}. New balance: {self.balance}"
        else:
            return "Deposit amount must be positive."

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds."
        elif amount <= 0:
            return "Withdrawal amount must be positive."
        else:
            self.balance -= amount
            return f"Withdrew {float(amount)}. New balance: {self.balance}"

    def get_balance(self):
        return self.balance

    def get_account_info(self):
        return f"Account Number: {self.account_number}, Account Holder: {self.account_holder}, Balance: {self.balance}"
def main():
    account_number = input("Enter account number: ")
    account_holder = input("Enter account holder name: ")
    initial_balance = float(input("Enter i (variable) account_holder: str

    account = BankAccount(account_number, account_holder, initial_balance)

    print(account.get_account_info())

    deposit_amount = float(input("Enter amount to deposit: "))
    print(account.deposit(deposit_amount))

    withdraw_amount = float(input("Enter amount to withdraw: "))
    print(account.withdraw(withdraw_amount))

    print(f"Current balance: {account.get_balance()}")
if __name__ == "__main__":
    main()
    # Assert statements for testing
    test_account = BankAccount("123456789", "John Doe", 1000.0)
    assert test_account.get_balance() == 1000.0
    assert test_account.deposit(500) == "Deposited 500.0. New balance: 1500.0"
    assert test_account.withdraw(200) == "Withdrew 200.0. New balance: 1300.0"
    assert test_account.withdraw(1500) == "Insufficient funds."
    assert test_account.deposit(-100) == "Deposit amount must be positive."
    assert test_account.withdraw(-50) == "Withdrawal amount must be positive."

```

Output Screenshot:

The prime_checker.py file defines a BankAccount class that simulates basic banking operations such as depositing, withdrawing, and checking the balance of an account. The class includes methods for handling deposits and withdrawals with appropriate validations, as well as methods to retrieve account information and balance. The main function prompts the user to create a bank account, perform transactions, and displays the results. Additionally, the file includes assert statements for testing the functionality of the BankAccount class to ensure it behaves as expected under various conditions.



Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program demonstrates a well-structured implementation of a BankAccount class using Object-Oriented Programming principles. It stores account details such as account number, account holder name, and balance as instance variables. The deposit() and withdraw() methods include proper validation to ensure positive transaction amounts and prevent overdrawing beyond the available balance. The program updates and returns the balance after each valid transaction. The main() function handles user interaction, while assert statements verify correctness through testing. Overall, the code shows good encapsulation, validation logic, user input handling, and practical application of OOP concepts in Python.

Task-04

Final Optimal Prompt:

lecture9.py

Rewrite and improve the documentation of this Python code to make it clear, professional, and consistent. Do not change the functionality.

Code Screenshot :

```python
'''
def is_prime(n):
    """Check if a number is prime.

    A prime number is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers.
    The function first checks if the number is less than 2, in which case it returns False.
    Then it iterates from 2 to the square root of n (inclusive) and checks if n is divisible by any of those numbers.
    If n is divisible by any of those numbers, it returns False, indicating that n is not prime.
    If n is not divisible by any of those numbers, it returns True, indicating that n is prime.

    Parameters:
    n (int): The number to check for primality.

    Returns:
    bool: True if n is prime, False otherwise.
    """
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def main():
    '''The main function prompts the user to enter a number and checks if it is prime using the is_prime function. It handles the case where the user input is not a valid integer by catching a ValueError and printing an appropriate message.'''
    try:
        num = int(input("Enter a number: "))
        if is_prime(num):
            print(f"{num} is a prime number.")
        else:
            print(f"{num} is not a prime number.")
    except ValueError:
        print("Please enter a valid integer.")
if __name__ == "__main__":
    main()
```

Output Screenshot:

```
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  & 'c:\python314\python.exe' 'c:\Users\sakir\.vscode\extensions
\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65105' '--' 'C:\Users\sakir\OneDrive\Desk
top\Ai-Assistant\lab10.py'
Enter a number: 85
85 is not a prime number.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\p
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '56198' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab10.py'
Enter a number: 11
11 is a prime number.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program checks whether a given number is prime using an efficient algorithm. The is_prime() function first eliminates numbers less than 2, since prime numbers must be greater than 1. It then checks divisibility from 2 up to the square root of the number, which improves efficiency compared to checking all numbers up to n-1. If any divisor is found, the function returns False; otherwise, it returns True. The main() function handles user input and uses a try-except block to manage invalid entries. Overall, the program demonstrates good optimization, proper validation, and clear documentation practices.

Tack-05

Final Optimal Prompt:

write a python program simple interest with discountand inclued documentation

write a python program of simple interest with discount and make fully professional documentation for each function in the program. The program should prompt the user for input and handle exceptions appropriately.

Code Screenshot :

```python
def calculate_simple_interest(principal, rate, time):
    '''This function calculates the simple interest based on the formula: Simple Interest = (Principal * Rate * Time) / 100. It takes three parameters: principal (the initial amount of money), rate (the annual interest rate in percentage), and time (the time period in years). It returns the calculated simple interest.'''
    return (principal * rate * time) / 100
def apply_discount(amount, discount_percentage):
    '''This function applies a discount to a given amount. It takes two parameters: amount (the original amount) and discount_percentage (the percentage of the discount to be applied). It calculates the discount amount and returns the final amount after applying the discount.'''
    discount_amount = (amount * discount_percentage) / 100
    return amount - discount_amount
def main():
    '''The main function prompts the user to enter the principal amount, interest rate, time period, and discount percentage. It then calculates the simple interest using the calculate_simple_interest function, applies the discount using the apply_discount function, and prints the final amount after applying the discount. It also handles invalid input by catching exceptions.'''
    try:
        principal = float(input("Enter the principal amount: "))
        rate = float(input("Enter the annual interest rate (in %): "))
        time = float(input("Enter the time period (in years): "))
        discount_percentage = float(input("Enter the discount percentage: "))

        simple_interest = calculate_simple_interest(principal, rate, time)
        final_amount = apply_discount(simple_interest, discount_percentage)

        print(f"The simple interest is: {simple_interest:.2f}")
        print(f"The final amount after applying discount is: {final_amount:.2f}")
    except ValueError:
        print("Please enter valid numerical values.")
if __name__ == "__main__":
    main()
```

```python
def calculate_simple_interest(principal, rate, time):
    '''Calculate the simple interest based on the formula: Simple Interest = (Principal * Rate * Time) / 100.

    Parameters:
    principal (float): The initial amount of money.
    rate (float): The annual interest rate in percentage.
    time (float): The time period in years.

    Returns:
    float: The calculated simple interest.
    '''
    return (principal * rate * time) / 100
def apply_discount(amount, discount_percentage):
    '''Apply a discount to a given amount.

    Parameters:
    amount (float): The original amount before discount.
    discount_percentage (float): The percentage of the discount to be applied.

    Returns:
    float: The final amount after applying the discount.
    '''
    discount_amount = (amount * discount_percentage) / 100
    return amount - discount_amount
def main():
    '''The main function prompts the user to enter the principal amount, interest rate, time period, and discount percentage. It then calculates the simple interest using the calculate_simple_interest function, applies the discount using the apply_discount function, and prints the final amount after applying the discount. It also handles invalid input by catching exceptions and providing appropriate error messages.'''
    try:
        principal = float(input("Enter the principal amount: "))
        rate = float(input("Enter the annual interest rate (in %): "))
        time = float(input("Enter the time period (in years): "))
        discount_percentage = float(input("Enter the discount percentage: "))

        simple_interest = calculate_simple_interest(principal, rate, time)
        final_amount = apply_discount(simple_interest, discount_percentage)

        print(f"The simple interest is: {simple_interest:.2f}")
        print(f"The final amount after applying discount is: {final_amount:.2f}")
    except ValueError:
        print("Please enter valid numerical values.")
if __name__ == "__main__":
    main()
```

Output Screenshot:

```
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\python314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65502' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab10.py'
Enter the principal amount: 5000
Enter the annual interest rate (in %): 5
Enter the time period (in years): 2
Enter the discount percentage: 50
The simple interest is: 500.00
The final amount after applying discount is: 250.00
Enter the principal amount: 8000
Enter the annual interest rate (in %): 3
Enter the time period (in years): 30
Enter the discount percentage: 8
The simple interest is: 7200.00
The final amount after applying discount is: 6624.00
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program calculates simple interest and then applies a discount to the calculated interest amount. It is well-structured using separate functions for modularity and clarity. The calculate_simple_interest() function computes interest using the standard formula, while the apply_discount() function calculates and subtracts the discount from the given amount. Each function includes professional documentation describing parameters and return values,

improving readability and maintainability. The main() function handles user interaction and uses exception handling to manage invalid numeric inputs. Overall, the program demonstrates good functional decomposition, proper documentation practices, input validation, and clear financial calculation logic.