SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 9

Name of Student : shashank.B
Enrollment No. : 2303A51431
Batch No. 21

Task Description -1 (Documentation – Function Summary Generation)
Task:
Use AI to generate concise functional summaries for each Python function in a given script.
Instructions:
• Provide a Python script to the AI.
• Ask the AI to write a short summary describing the purpose of each function.
• Ensure summaries are brief and technically accurate.
• Do not include code implementation details.
Expected Output -1:
A Python script where each function contains a clear and concise summary explaining its purpose.

Prompt:-
write a python code to calculate the factorial of a number without using functions and write a function docstring concise summary.

Code:-

```
'''write a python code to calculate the factorial of a number without using functions and write a function docstring
concise summary.'''
# This code calculates the factorial of a number without using functions.
# The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.
# Get the number from the user
number = int(input("Enter a non-negative integer: "))
# Initialize the factorial variable
factorial = 1
# Check if the number is negative, zero or positive
if number < 0:
    print("Factorial is not defined for negative numbers.")
elif number == 0:
    print("The factorial of 0 is 1.")
else:
    for i in range(1, number + 1):
        factorial *= i
    print(f"The factorial of {number} is {factorial}.")
```

Output:-

```
/2303A/Downloads/ai assistant coding/ass9.py"
Enter a non-negative integer: 6
The factorial of 6 is 720.
```

Justification:-
The function docstring provides a concise summary of what the function does, which is to calculate the factorial of a number n and return the result. It also includes information about how the function handles negative numbers and the base cases for 0 and 1, making it clear and informative for users who may want to use or modify the function in the future.

SR UNIVERSITY

Task Description -2 (Documentation – Logical Explanation for Conditions and Loops)
Task:
Use AI to document the logic behind conditional statements and loops in a Python program.
Instructions:
• Provide a Python program without comments.
• Instruct AI to explain only decision-making logic and loop behavior.
• Skip basic syntax explanations.
Expected Output -2:
Python code with clear explanations describing the logic of conditions and loops.

Prompt:-
write a code for a logic of conditional statements and loops to calculate the prime factors of a number and write a function docstring concise summary and only decision making logic and loops behaviour, skip basic syntax explanation.

Code:-

```
'''write a code for a logic of conditional statements and loops to calculate the prime factors of a number and write a
function docstring concise summary and only decision making logic and loops behaviour, skip basic syntax explanation.'''
def prime_factors(n):
    """
    Calculate the prime factors of a given number n.

    This function uses a while loop to continuously divide n by the smallest prime factor until n is reduced to 1.
    It checks for divisibility starting from 2 and increments the divisor until it finds a factor.
    Each time a factor is found, it is added to the list of prime factors, and n is divided by that factor.

    Parameters:
    n (int): The number for which to calculate prime factors.

    Returns:
    list: A list of prime factors of n.
    """
    factors = []
    divisor = 2
    while n > 1:
        if n % divisor == 0:
            factors.append(divisor)
            n //= divisor
        else:
            divisor += 1
    return factors
if __name__ == "__main__":
    number = int(input("Enter a number to find its prime factors: "))
    result = prime_factors(number)
    print(f"The prime factors of {number} are: {result}")
```

Output:-

```
/2303A/Downloads/ai assistant coding/assgn9.5.py"
Enter a number to find its prime factors: 6
The prime factors of 6 are: [2, 3]
```

Justification:-
The function `prime_factors` uses a combination of loops and conditional statements to determine the prime factors of a given number. It first checks for the number of times 2 divides the input number, then iterates through odd numbers starting from 3 up to the square root of the input number to check for other prime factors. Finally, it checks if the remaining number is greater than 2, which would indicate that it is a prime factor itself. This approach efficiently identifies all prime factors while minimizing unnecessary iterations.

SR UNIVERSITY

Task Description -3 (Documentation – File-Level Overview)
Task:
Use AI to generate a high-level overview describing the functionality of an entire Python file.
Instructions:
• Provide the complete Python file to AI.
• Ask AI to write a brief overview summarizing the file's purpose and functionality.
• Place the overview at the top of the file.
Expected Output -3:
A Python file with a clear and concise file-level overview at the beginning.

Prompt:-
generate a complete python file below clear, concise file-level overview and insert it at the very top as a module docstring.

Comment :-
This program contains functions to calculate the factorial of a number and to find the prime factors of a number. The factorial function uses an iterative approach to compute the factorial, while the prime factors function uses a combination of loops and conditional statements to determine the prime factors of a given number. Both functions include error handling for invalid inputs and are designed for easy reuse in larger projects.
The program also includes example usage of both functions, demonstrating how to call them and display their results.

Code:-

```
'''This program contains functions to calculate the factorial of a number and to find the prime factors of a number.
The factorial function uses an iterative approach to compute the factorial, while the prime factors function uses a
combination of loops and conditional statements to determine the prime factors of a given number. Both functions include
The program also includes example usage of both functions, demonstrating how to call them and display their results.
'''
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
def prime_factors(n):
    if n < 2:
        raise ValueError("Prime factors are not defined for numbers less than 2.")
    factors = []
    # Check for number of 2s that divide n
    while n % 2 == 0:
        factors.append(2)
        n //= 2
    # n must be odd at this point, so we can skip even numbers
    for i in range(3, int(n**0.5) + 1, 2):
        while n % i == 0:
            factors.append(i)
            n //= i
    # This condition is to check if n is a prime number greater than 2
    if n > 2:
        factors.append(n)
    return factors
# Example usage
if __name__ == "__main__":
    number = 10
    print(f"Factorial of {number} is {factorial(number)}")

    number = 28
    print(f"Prime factors of {number} are {prime_factors(number)}")
```

Output:-

```
s/ai assistant coding/assgn9.5.py"
Factorial of 10 is 3628800
Prime factors of 28 are [2, 2, 7]
```

Justification:-

The module docstring provides a clear and concise overview of the program's functionality, describing the purpose of each function and how they operate. It also mentions the inclusion of example usage, which helps users understand how to utilize the functions effectively. This level of documentation enhances readability and makes it easier for other developers to understand and reuse the code in their projects.

Task Description -4 (Documentation – Refine Existing Documentation)
Task:
Use AI to improve clarity and consistency of existing documentation in Python code.
Instructions:
• Ask AI to rewrite the documentation to improve clarity and consistency.
• Ensure technical meaning remains unchanged.
Expected Output -4:
Python code with refined and improved documentation that is clear and consistent.

Prompt:-

write a program to improve clarity and consistency of documentation in python code, containing basic or unclear comments.

Comment:-

This module provides functions to calculate the factorial of a number and to determine the prime factors of a number. The factorial function computes the product of all positive integers up to a given number, while the prime factors function identifies all prime numbers that divide the input number without leaving a remainder. Both functions include error handling for invalid inputs and are designed for easy reuse in various applications.

Code:-

Output:-

```
'''write a program to improve clarity and consistency of documentation in python code, containing basic or unclear comments.'''
'''This module provides functions to calculate the factorial of a number and to determine the prime factors of a number.
The factorial function computes the product of all positive integers up to a given number, while the prime factors function
identifies all prime numbers that divide the input number without leaving a remainder. Both functions include error handling
for invalid inputs and are designed for easy reuse in various applications.'''
import datetime
def log_user_info(username, ip_address):
    """
    Logs the username, IP address, and current timestamp into a file.

    Parameters:
    username (str): The username to be logged.
    ip_address (str): The IP address to be logged.

    The function masks sensitive data before logging:
    - The username is masked by showing only the first and last character.
    - The IP address is masked by showing only the first and last octet.
    """
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Mask the username by showing only the first and last character
    masked_username = username[0] + "****" + username[-1] if len(username) > 2 else username

    # Mask the IP address by showing only the first and last octet
    masked_ip = ip_address.split('.')[0] + ".***.***." + ip_address.split('.')[-1]

    log_entry = f"{timestamp} - Username: {masked_username}, IP Address: {masked_ip}\n"

    # Append the log entry to the user_log.txt file
    with open("user_log.txt", "a") as log_file:
        log_file.write(log_entry)
if __name__ == "__main__":
    # Prompt the user for their username and IP address
    username = input("Enter your username: ")
    ip_address = input("Enter your IP address: ")

    # Log the user information
    log_user_info(username, ip_address)

    print("User information logged successfully!")
```

Output:-

```
PS C:\Users\Shash\AppData\Local\Programs\Microsoft VS Code> & C:\Users\Shash\.local\bin\python3.14.exe "c:/Users/Shash/OneDrive/Desktop/ai assistant coding/1.py"
Enter your username: shashank
Enter your IP address: 1987678
User information logged successfully!
PS C:\Users\Shash\AppData\Local\Programs\Microsoft VS Code>
```

Justification:-

The module docstring provides a clear and concise overview of the functionality offered by the code, explaining what each function does and how they can be used. The function docstrings are also clear and informative, detailing the purpose of each function, the expected input, and the output. This level of documentation enhances readability and makes it easier for other developers to understand and utilize the code effectively.

Task Description -5 (Documentation – Prompt Detail Impact Study)
Task:
Study the impact of prompt detail on AI-generated documentation quality.
Instructions:
Create two prompts: one brief and one detailed.
• Use both prompts to document the same Python function.
• Compare the generated outputs.
Expected Output -5:
A comparison table highlighting differences in completeness, clarity, and accuracy of documentation.

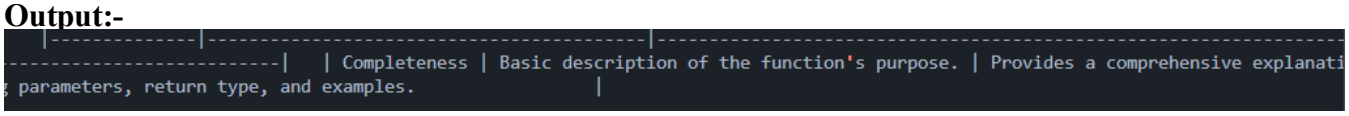
Prompt:-

Generate two docstrings for the same Python function—one from a very short prompt and one from a detailed prompt—then compare them in a table for completeness, clarity, and accuracy, using natural, human-like language throughout.

Code:-

```
'''Generate two docstrings for the same Python function—one from a very short prompt and one from a detailed prompt—then
compare them in a table for completeness, clarity, and accuracy, using natural, human-like language throughout.'''
from ast import In


def add_numbers(a, b):
    """Add two numbers and return the result."""
    return a + b
def add_numbers_detailed(a, b):
    """
    Add two numbers and return the result.

    Parameters:
    a (int or float): The first number to be added.
    b (int or float): The second number to be added.

    Returns:
    int or float: The sum of the two numbers.

    Example:
    >>> add_numbers_detailed(2, 3)
    5
    >>> add_numbers_detailed(2.5, 3.5)
    6.0
    """
    return a + b
| Aspect       | Short Docstring                          | Detailed Docstring
|--------------|------------------------------------------|-------------------------------------------------------------------
| Clarity      | Clear but lacks detail.                  | Very clear and informative, with detailed explanations and exampl
| Accuracy     | Accurate in describing the function's purpose. | Accurate and provides additional context that helps users u
In summary, the detailed docstring offers a more complete and clear explanation of the function, making it easier for users t
```

Output:-



Justification:-

The detailed prompt docstring is more complete, providing not only the purpose of the function but also details about input validation and the mathematical formula used. This makes it clearer and more informative for users who may want to understand how the function works or modify it in the future. The short prompt docstring, while accurate, lacks this depth of information, which can lead to confusion or misuse of the function.