**Lab Exercise 5- Slash (/) Commands in GitHub Copilot Chat**

**Lab Exercise: Using Slash Commands in GitHub Copilot Chat**

**Objective:**

Learn how to use **slash ( / ) commands** in **GitHub Copilot Chat** to generate code, debug, explain, and optimize efficiently.

* **Use /explain** to understand complex code.
* **Use /fix** to debug errors quickly.
* **Use /tests** to generate test cases.
* **Use /optimize** to improve code efficiency.
* **Use /generate** to create new features faster.
* **Use /docstring** to document functions easily.

**Prerequisites:**

* **GitHub Copilot Chat** enabled in **VS Code**.
* Basic knowledge of programming (Python/JavaScript).

**Exercise Steps**

**1. Using /explain to Understand Code**

**Task: Get an Explanation of a Code Snippet**

1. Copy and paste this code in **Copilot Chat** and type:

/explain

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

1. Observe how **Copilot Chat explains** the function step by step.

**Expected Outcome:**

Copilot will provide a detailed explanation of how the recursive function calculates the factorial of a number.

**2. Using /fix to Debug Code**

**Task: Fix a Buggy Function**

1. Copy and paste this **buggy code** into Copilot Chat and type:

/fix

def add\_numbers(a, b):

return a + c # This contains an error

1. Copilot will suggest a **correction**.

**Expected Outcome:**

Copilot will identify that c is undefined and correct the code to:

def add\_numbers(a, b):

return a + b

**3. Using /tests to Generate Unit Tests**

**Task: Create Tests for a Function**

1. Paste the following function in **Copilot Chat** and type:

/tests

def is\_even(n):

return n % 2 == 0

1. Copilot will generate test cases.

**Expected Outcome:**

Copilot will provide test cases using pytest or unittest:

import unittest

class TestIsEven(unittest.TestCase):

def test\_is\_even(self):

self.assertTrue(is\_even(2))

self.assertTrue(is\_even(0))

self.assertFalse(is\_even(3))

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**4. Using /optimize to Improve Performance**

**Task: Optimize an Inefficient Function**

1. Copy and paste this inefficient function in **Copilot Chat** and type:

/optimize

def find\_max(numbers):

max\_num = numbers[0]

for num in numbers:

if num > max\_num:

max\_num = num

return max\_num

1. Copilot will suggest an optimized version.

**Expected Outcome:**

Copilot will simplify it using Python’s built-in max():

def find\_max(numbers):

return max(numbers)

**5. Using /generate to Create New Code**

**Task: Generate a Flask API with One Endpoint**

1. In **Copilot Chat**, type:

/generate a simple Flask API with a single GET endpoint that returns "Hello, World!"

1. Accept and run the generated code.

**Expected Outcome:**

Copilot will generate a basic Flask API:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return "Hello, World!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**6. Using /docstring to Add Documentation**

**Task: Generate a Docstring for a Function**

1. Copy and paste this function in **Copilot Chat** and type:

/docstring

def multiply(a, b):

return a \* b

1. Copilot will generate proper documentation.

**Expected Outcome:**

Copilot will return:

def multiply(a, b):

"""

Multiply two numbers.

Args:

a (int or float): The first number.

b (int or float): The second number.

Returns:

int or float: The product of a and b.

"""

return a \* b

**Conclusion**

* **Use /explain** to understand complex code.
* **Use /fix** to debug errors quickly.
* **Use /tests** to generate test cases.
* **Use /optimize** to improve code efficiency.
* **Use /generate** to create new features faster.
* **Use /docstring** to document functions easily.

By following this lab, you'll **develop features faster** using **slash commands** in GitHub Copilot Chat!