F-04 - GitHub Copilot Ask Mode

SUMMARY

In this lesson, we explored how GitHub Copilot can significantly streamline the process of generating unit tests and documentation, and how we can interact with Copilot using the **Ask Mode**. We focused on two primary inline actions: generating unit tests and creating documentation for code. These tasks, typically tedious and time-consuming for developers, can be automated using Copilot, increasing productivity and focusing on more complex code development.

To generate **unit tests**, we followed these steps:

1. Right-click the function or section of code that requires testing.
2. Select **Copilot** and choose **Generate Tests**.
3. Upon first use, choose a testing framework, such as **Pytest**.
4. Copilot creates a new file with the prefix `test_` and populates it with relevant unit tests.

For generating **documentation**:

1. Highlight the code or file needing documentation.
2. Go to **Copilot** and select **Generate Docs**.
3. Copilot inserts descriptive doc strings, including parameter descriptions, following common standards and making complex logic more understandable.

The **Ask Mode** was used to further explore Copilot's capabilities:

1. **Fixing Errors**: Highlight an error, use **Control-I** or Editor Inline Chat, and ask Copilot to fix it.
2. **Modifying Code**: Use **Control-I** to ask Copilot to make specific code modifications or add new features.
3. **Creating Scripts**: Use **Control-I** to command Copilot to create new scripts, like fetching Bitcoin prices.

WHAT WE LEARNED

- Generating unit tests with GitHub Copilot.
- Creating comprehensive code documentation using Copilot.
- Using Ask Mode for error fixing and code modifications.
- Commanding Copilot to quickly create new scripts or functionalities.

HOW WE CAN APPLY IT

- Automate routine unit testing tasks, freeing time for more complex coding tasks.
- Enhance codebases with detailed documentation for better team collaboration.
- Use Ask Mode to handle and fix errors without deep-diving into issues manually.
- Rapidly prototype new features or scripts by leveraging Copilot's coding capabilities.

TIPS AND TRICKS

- Ensure your code is well-commented; clear logic makes Copilot more effective in generating accurate tests and documentation.
- Familiarize yourself with shortcut keys like **Control-I** to make interactions with Copilot seamless.
- Regularly review and adjust the generated code to fit specific project needs.
- Use Copilot for brainstorming different approaches to solve coding challenges.

EXAMPLES

Unit Test Generation Example

```
def convert_temperature(fahrenheit):
    # Convert fahrenheit to celsius
    return (fahrenheit - 32) * 5.0/9.0

# Generated test file: test_temperature.py
def test_convert_temperature():
    assert convert_temperature(32) == 0
    assert convert_temperature(212) == 100
    assert convert_temperature(-40) == -40
```

Documentation Generation Example

```
def complex_business_logic(age):
    """
    Determines the category based on age.

    :param age: Age of the individual
    :return: Category string
    """
    if age < 25:
        return "Youth"
    elif 25 <= age < 40:
        return "Adult"
    else:
        return "Senior"
```

Script Creation Example

```
# Command issued: "Create a script that fetches the latest price of Bitcoin"
import requests

def fetch_bitcoin_price():
    """
    Fetches the latest Bitcoin price from an API.

    :return: Current Bitcoin price
    """
    response = requests.get('https://api.coindesk.com/v1/bpi/currentprice.json')
    data = response.json()
    return data['bpi']['USD']['rate_float']

print(fetch_bitcoin_price())
```

In conclusion, leveraging GitHub Copilot for these actions not only saves time but also enhances the quality and maintainability of our code. By utilizing the **Ask Mode**, we gain a powerful tool to quickly

resolve coding challenges and adapt to new requirements.