AI Coding Assistance Tools Hands-on Project Using GitHub Copilot Student Activity Document

---

Introduction

This document outlines the student activities for the AI Coding Assistance Tools Hands-on Project using GitHub Copilot. The activities will guide you through setting up GitHub Copilot, writing code, debugging errors, and optimizing code using AI assistance. Follow the activities step-by-step to enhance your understanding of how AI can assist in coding.

Student Activities

Activity 1: Setting Up GitHub Copilot (10 minutes)

Objective:

- To install and configure GitHub Copilot in Visual Studio Code (VS Code).

Steps:

1. **Install Visual Studio Code (VS Code)** if not already installed.
2. Open VS Code and go to **Extensions** (`Ctrl+Shift+X`).
3. Search for "GitHub Copilot" and click **Install**.
4. Sign in with your GitHub account to enable the extension.
5. Ensure GitHub Copilot is active by checking settings.

Expected Outcome:

- GitHub Copilot should be enabled and ready to provide suggestions while writing code.

Activity 2: Writing Code with GitHub Copilot (20 minutes)

Objective:

- To use GitHub Copilot's code suggestion feature to speed up the coding process.

Steps:

1. Open a new file in VS Code, for example `example.py`.
2. Start writing a simple function. For example:

```
def calculate_sum(a, b):

# Your code here
```

3. GitHub Copilot will suggest the following:

```
return a + b
```

4. Press `Tab` to accept the suggestion.

Expected Outcome:

- GitHub Copilot should suggest and complete basic code as you type.

Activity 3: Debugging with GitHub Copilot (20 minutes)

Objective:

- To use GitHub Copilot for debugging by fixing an intentional error in the code.

Steps:

1. Modify the function to introduce an error:

```
def calculate_sum(a):

return a + b # Intentional error
```

2. Run the code and observe the error message (e.g., `NameError: name 'b' is not defined`).
3. GitHub Copilot will suggest a fix, such as:

```
def calculate_sum(a, b):

return a + b
```

4. Press `Tab` to accept the fix.

Expected Outcome:

- GitHub Copilot will help identify the error and suggest a solution to correct the code.

Activity 4: Optimizing and Refactoring Code (20 minutes)

Objective:

- To use GitHub Copilot to refactor inefficient code for better performance.

Steps:

1. Write inefficient code, for example:

```
def multiply_numbers(x, y):

result = 0

for i in range(y):

result += x

return result
```

2. GitHub Copilot will suggest a more optimized version of the code:

```
def multiply_numbers(x, y):

return x * y
```

3. Press `Tab` to accept the suggestion.

Expected Outcome:

- GitHub Copilot will refactor the code to a more efficient solution, reducing redundancy.

Activity 5: Review and Reflection (20 minutes)

Objective:

- To review the coding suggestions, debugging help, and optimization provided by GitHub Copilot.

Steps:

1. Review the original code you wrote and compare it to the Copilot-suggested code.
2. Reflect on how Copilot's suggestions improved efficiency and how much time was saved.
3. Identify any patterns in Copilot's suggestions that helped improve your coding.

Expected Outcome:

- A clearer understanding of how GitHub Copilot can help in coding, debugging, and optimization.

Timeline for Completion

- **Setting up GitHub Copilot**: 10 minutes
- **Writing code with Copilot**: 20 minutes
- **Debugging with Copilot**: 20 minutes
- **Optimizing with Copilot**: 20 minutes

- **Reflection and review**: 20 minutes

Conclusion

By completing these activities, students will have practical experience in using GitHub Copilot to assist with writing, debugging, and optimizing code. This project will enhance both your coding efficiency and understanding of AI-assisted coding practices.
