GitHub Copilot Hands-on Lab for Advanced Learners

A guide to explore the features and capabilities of GitHub Copilot for experienced developers

Introduction

GitHub Copilot is an AI-powered pair programmer that helps you write code faster and better. It can suggest whole lines or entire functions for you, based on the context of your code and the comments you write. It can also help you find relevant documentation, examples, and tests for your code.

In this hands-on lab, you will learn how to use GitHub Copilot for advanced coding tasks, such as refactoring, debugging, testing, and optimizing your code. You will also learn how to customize GitHub Copilot to suit your preferences and workflows.

Prerequisites

- A GitHub account and a GitHub repository with some code in it
- A Visual Studio Code editor with the GitHub Copilot extension installed
- A basic understanding of GitHub Copilot and its features
- A working knowledge of the programming language you are using

Lab Overview

This lab consists of four exercises, each focusing on a different aspect of GitHub Copilot. You can complete them in any order, but we recommend following the sequence below:

- Exercise 1: Refactoring code with GitHub Copilot
- Exercise 2: Debugging code with GitHub Copilot
- Exercise 3: Testing code with GitHub Copilot
- Exercise 4: Optimizing code with GitHub Copilot

For each exercise, you will be given a code snippet and a task to perform. You will use GitHub Copilot to help you complete the task, and compare the results with the expected output. You will also learn some tips and tricks to make the most of GitHub Copilot.

Exercise 1: Refactoring code with GitHub Copilot

Refactoring is the process of improving the structure and readability of your code, without changing its functionality. GitHub Copilot can help you refactor your code by suggesting alternative ways to write your code, such as using different syntax, variables, or functions.

In this exercise, you will use GitHub Copilot to refactor a code snippet that calculates the factorial of a number. The code snippet is written in Python, but you can use any programming language you prefer.

```
# This is a code snippet that calculates the factorial of a number

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n * factorial(n-1)
```

The task is to refactor the code snippet to make it more concise and readable, using GitHub Copilot.

Follow these steps to complete the task:

1. Open your Visual Studio Code editor and create a new file.
2. Copy and paste the code snippet into the file.
3. Write a comment above the code snippet, describing what you want to do. For example, you can write "# Refactor this code to use a loop instead of recursion".
4. Press Ctrl+Enter to trigger GitHub Copilot. You should see a suggestion from GitHub Copilot that refactors the code snippet according to your comment.
5. Review the suggestion and accept it if you are satisfied, or press Ctrl+Enter again to see more suggestions.
6. Compare the refactored code with the original code and verify that they produce the same output.

The expected output is a code snippet that calculates the factorial of a number using a loop instead of recursion. For example, GitHub Copilot might suggest the following code:

```
# Refactor this code to use a loop instead of recursion

def factorial(n):

result = 1

for i in range(1, n+1):

result *= i

return result
```

Some tips and tricks to use GitHub Copilot for refactoring code are:

- Be specific and clear in your comments, as GitHub Copilot will use them to generate suggestions.
- Use natural language to describe your intent, rather than technical terms.
- Use the feedback mechanism to rate the suggestions and help GitHub Copilot improve.
- Use the documentation and examples features to learn more about the code suggestions and how they work.

Exercise 2: Debugging code with GitHub Copilot

Debugging is the process of finding and fixing errors in your code. GitHub Copilot can help you debug your code by suggesting possible causes and solutions for the errors, as well as providing relevant documentation and examples.

In this exercise, you will use GitHub Copilot to debug a code snippet that converts a temperature from Celsius to Fahrenheit. The code snippet is written in JavaScript, but you can use any programming language you prefer.

// This is a code snippet that converts a temperature from Celsius to Fahrenheit

function celsiusToFahrenheit(celsius) {

let fahrenheit = celsius * (9 / 5) + 32;

return fahrenheit;

}

// Test the function with some inputs

[URL](celsiusToFahrenheit(0)); // Expected output: 32

[URL](celsiusToFahrenheit(100)); // Expected output: 212

[URL](celsiusToFahrenheit(-40)); // Expected output: -40

The task is to debug the code snippet and fix the error that causes the wrong output for the last input.

Follow these steps to complete the task:

1. Open your Visual Studio Code editor and create a new file.
2. Copy and paste the code snippet into the file.
3. Run the code and observe the output. You should see that the last output is incorrect, as it should be -40 instead of 40.
4. Write a comment above the code snippet, describing the error and asking for help. For example, you can write "// Why is the output for -40 wrong? How can I fix it?"
5. Press Ctrl+Enter to trigger GitHub Copilot. You should see a suggestion from GitHub Copilot that explains the error and provides a solution.
6. Review the suggestion and accept it if you are satisfied, or press Ctrl+Enter again to see more suggestions.
7. Run the code again and verify that the output is correct.

The expected output is a code snippet that converts a temperature from Celsius to Fahrenheit correctly, with a comment that explains the error and the solution. For example, GitHub Copilot might suggest the following code:

```
// Why is the output for -40 wrong? How can I fix it?

// The error is due to the operator precedence in JavaScript. The multiplication is performed before the addition, so the expression celsius * (9 / 5) + 32 is equivalent to (celsius * 9) / 5 + 32. When celsius is -40, this results in -40 * 9 / 5 + 32, which is 40.

// To fix it, you need to add parentheses around the addition, so that it is performed before the multiplication. The expression should be celsius * ((9 / 5) + 32).

function celsiusToFahrenheit(celsius) {

let fahrenheit = celsius * ((9 / 5) + 32);

return fahrenheit;

}

// Test the function with some inputs

[URL](celsiusToFahrenheit(0)); // Expected output: 32

[URL](celsiusToFahrenheit(100)); // Expected output: 212

[URL](celsiusToFahrenheit(-40)); // Expected output: -40
```

Some tips and tricks to use GitHub Copilot for debugging code are:

- Be descriptive and concise in your comments, as GitHub Copilot will use them to generate suggestions.
- Use questions or statements to express your problem, rather than commands or requests.
- Use the feedback mechanism to rate the suggestions and help GitHub Copilot improve.
- Use the documentation and examples features to learn more about the code suggestions and how they work.

Exercise 3: Testing code with GitHub Copilot

Testing is the process of verifying that your code works as expected and meets the requirements. GitHub Copilot can help you test your code by suggesting test cases, assertions, and frameworks for your code, as well as providing relevant documentation and examples.

In this exercise, you will use GitHub Copilot to test a code snippet that checks if a string is a palindrome. A palindrome is a word, phrase, or sequence that reads the same backward as forward, such as "racecar" or "madam". The code snippet is written in Java, but you can use any programming language you prefer.

```
// This is a code snippet that checks if a string is a palindrome

public class Palindrome {

// A method that returns true if a string is a palindrome, and false otherwise
```

```
public static boolean isPalindrome(String str) {

// Convert the string to lower case and remove any spaces or punctuation

str = [URL]().replaceAll("[\\s\\p{Punct}]", "");

// Compare the string with its reversed version

return [URL](new StringBuilder(str).reverse().toString());

}

}
```

The task is to test the code snippet and write a unit test that verifies the functionality of the isPalindrome method, using GitHub Copilot.

Follow these steps to complete the task:

1. Open your Visual Studio Code editor and create a new file.
2. Copy and paste the code snippet into the file.
3. Write a comment below the code snippet, describing what you want to do. For example, you can write "// Write a unit test for the isPalindrome method using JUnit".
4. Press Ctrl+Enter to trigger GitHub Copilot. You should see a suggestion from GitHub Copilot that writes a unit test for the isPalindrome method using JUnit.
5. Review the suggestion and accept it if you are satisfied, or press Ctrl+Enter again to see more suggestions.
6. Run the unit test and verify that it passes.

The expected output is a code snippet that writes a unit test for the isPalindrome method using JUnit. For example, GitHub Copilot might suggest the following code:

```
// Write a unit test for the isPalindrome method using JUnit

import static [URL].*;

import [URL];

public class PalindromeTest {

// A test case that checks if the isPalindrome method returns true for palindromes

@Test

public void testIsPalindromeTrue() {

assertTrue([URL]("racecar"));

assertTrue([URL]("Madam, I'm Adam"));

assertTrue([URL]("A man, a plan, a canal: Panama"));

}

// A test case that checks if the isPalindrome method returns false for non-palindromes

@Test
```

```
public void testIsPalindromeFalse() {

assertFalse([URL]("hello"));

assertFalse([URL]("GitHub Copilot"));

assertFalse([URL]("This is not a palindrome"));

}

}
```

Some tips and tricks to use GitHub Copilot for testing code are:

- Be specific and clear in your comments, as GitHub Copilot will use them to generate suggestions.
- Use the name of the testing framework and the method you want to test in your comment.
- Use the feedback mechanism to rate the suggestions and help GitHub Copilot improve.
- Use the documentation and examples features to learn more about the code suggestions and how they work.

Exercise 4: Optimizing code with GitHub Copilot

Optimizing is the process of improving the performance and efficiency of your code, without compromising its functionality or readability. GitHub Copilot can help you optimize your code by suggesting ways to reduce the time and space complexity of your code, as well as providing relevant documentation and examples.

In this exercise, you will use GitHub Copilot to optimize a code snippet that sorts an array of numbers using bubble sort. Bubble sort is a simple sorting algorithm that repeatedly swaps adjacent elements that are out of order, until the array is sorted. The code snippet is written in C, but you can use any programming language you prefer.

```
// This is a code snippet that sorts an array of numbers using bubble sort

#include <[URL]>

// A function that sorts an array of numbers using bubble sort

void bubbleSort(int arr[], int n) {

// Loop through the array n-1 times

for (int i = 0; i < n-1; i++) {

// Loop through the array from 0 to n-i-1

for (int j = 0; j < n-i-1; j++) {

// If the current element is greater than the next element, swap them

if (arr[j] > arr[j+1]) {

int temp = arr[j];
```

```
arr[j] = arr[j+1];
arr[j+1] = temp;
```