Writing Your First Function With GitHub Copilot

To best understand the future of software development, try programming this factorial calculator with GitHub Copilot!

In software development, there is a constant demand for the quick delivery of efficient code. With the recent creation of GitHub Copilot, a cutting-edge AI-driven coding assistant, developers can now code better innovations more easily and quickly than ever before. To show-off just how easy Copilot can make programming, this project will walk through crafting a factorial calculator. Factorials serve as a great example of a simple problem that has many applications in statistics and algorithm design, such as counting the many arrangements of a group of objects. Even if it has been decades since your last statistics class, this functionality can still be easily built by leaving the specifics of these mathematical properties to Copilot.

Setting Up GitHub Copilot

Before diving into this new way of coding, it's crucial to be properly set up. For an in-depth guide on setting up Visual Studio Code, we recommend reviewing our GitHub Copilot setup article. For those more familiar with the process, here is a high-level overview of what is needed to follow along with this project:

1. **GitHub Account:** If you don't have one, sign up here.
2. **GitHub Copilot Subscription:** Copilot requires a subscription to use. This can be purchased with the option of a free trial here.
3. **Visual Studio Code (VSCode):** While Copilot can work in a variety of code editors, it integrates best with VSCode. Download it here.
4. **GitHub Copilot Extension:** Once you're in VSCode, head over to the left extensions sidebar and search for "GitHub Copilot". Install this official extension and log in to your GitHub account when prompted.

Working with GitHub Copilot

With the setup complete, the easiest way to get started with Copilot is just to start typing away. While you code as usual, Copilot will make suggestions to speed up your development and proactively prevent errors. More precisely, this loop looks like this:

1. **Start Typing:** As you begin typing in VSCode, Copilot will automatically provide suggestions.

2. **Accept Suggestions:** Press Tab to accept the auto-completion if Copilot correctly understood your intention.
3. **Ask Copilot:** If you're unsure about how to implement the next feature, just write a comment describing the problem in a comment, such as `// How do I...`. Copilot will then attempt to give you the answer as another code suggestion in your text editor.
4. **Continue Coding:** Keep the momentum going, coding the functions easiest to implement for you, and letting Copilot take the lead when feeling stuck.

Having Copilot Write the Factorial Function

Now that we know the workflow of working with GitHub Copilot, let's kick things off with the backbone of our calculator: the factorial function.

Factorials are easy to calculate. They are the product of all positive integers less than or equal to the given number. For example, the factorial of five, written as `5!`, is `5 × 4 × 3 × 2 × 1 = 120`.

To get started, create a new Python file called `factorial_calculator.py`. Then, have GitHub Copilot help write the function to calculate the factorial:

1. Start by typing `def factorial(n):` in your Python file.
2. GitHub Copilot should automatically suggest code similar to the following:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

1. Press Tab to accept the suggestion.

Voila! In no time at all, you've got your function!

Do keep in mind that while the above example is the most common way to calculate a factorial, it is also possible for Copilot to recommend a different solution entirely. For example, instead of recursively multiplying each integer, a loop could be used to calculate the factorial iteratively. This may offer better performance in certain cases and highlights why general technical knowledge is still important even when leaving the coding to Copilot.

Prompting Copilot to Build User Input for the Function

User input is a great example of boilerplate code that is often written very similarly regardless of the application. With Copilot, not only is this precious time saved for the

programmer, but edge cases that may be glossed over until they cause errors can be proactively prevented. To start making the calculator interactive:

1. Begin by typing `def get_factorial_input():`.
2. Wait for Copilot's suggestion, which should be along the lines of:

```
def get_factorial_input():
    """
    Prompts the user for a non-negative integer and returns it.
    Keeps prompting until a valid non-negative integer is provided.
    """
    while True:
        try:
            # Get input from the user
            num = int(input("Enter a non-negative integer for factorial computation: "))

            # Check if the number is non-negative
            if num >= 0:
                return num
            else:
                print("Please enter a non-negative integer.")
        except ValueError:
            print("Invalid input. Please enter a non-negative integer.")
```

1. Accept the suggestion by pressing Tab.

Now, combine the two functions by asking the user for input and then calculating the factorial:

```
number = get_factorial_input()
print(f"The factorial of {number} is {factorial(number)}")
```

And just like that, not only has input been built for calculating factorials, but an edge case that may not have crossed the mind of most programmers has also been solved! This is the power of GitHub Copilot! Faster software development with fewer errors from having analyzed millions of lines of code.

Write Tests for the Code with Copilot

As we just saw, there may be many surprise use cases for our code. To ensure that our code works as expected, we can ask Copilot to write some common test cases for us.

1. Start with `def test_factorial():`.
2. Copilot might suggest:

```
def test_factorial():
    assert factorial(5) == 120
    assert factorial(0) == 1
    assert factorial(7) == 5040
    print("All tests passed!")
```

1. Accept the suggestion and run the tests with `test_factorial()`.

It is important to verify that these test cases work as expected. While Copilot will likely get many things right, there is a chance that it may misinterpret the intended functionality of the program. This again highlights that while Copilot can very much

improve the speed, and possibly the quality, of software developers, it cannot replace them. A human will always be required to verify that the code matches the real-world goals it was originally set out to achieve.

Continuing Your Programming Journey with Copilot

GitHub Copilot isn't just a tool; it's a revolution in how people develop software. It seamlessly integrates AI into the programmer's workflow, making coding more accessible and efficient. By following this guide, not only did you craft a nifty factorial calculator, but you also took a leap into the future of coding. Use it well, and happy programming!