# F-18 - Custom Instructions in GitHub Copilot

## SUMMARY

In this lesson, we explored the powerful feature of **Custom Instructions** in GitHub Copilot, which allows us to customize the responses provided by the AI to fit our preferences and project standards. This includes using preferred libraries, coding conventions, or even whimsical styles like pirate or Shakespearean language.

To begin using Custom Instructions, we must create a `.github` folder in our project directory and add a `CopilotInstructions.md` file within it. Here, we can specify various directives for GitHub Copilot to follow. It's important to keep these instructions straightforward and concise to avoid confusion during AI processing.

### Steps to Activate Custom Instructions:

1. **Create a `.github` directory:** This will store Custom Instructions and related markdown files.
2. **Add `CopilotInstructions.md`:** Inside this file, you can specify instructions, such as "use camel case for variables" or "respond in bullet points."
3. **Use GitHub Copilot in a code file:** Observe how the AI's responses conform to the set instructions.

By using this feature effectively, we can ensure consistency across our codebase and project documentation.

## WHAT WE LEARNED

- Customizing GitHub Copilot's responses with specific instructions.
- Creating and organizing a `CopilotInstructions.md` file.
- Guidelines for writing effective custom instructions.
- Examples of directives, such as using specific languages or coding styles.

## HOW WE CAN APPLY IT

- **Team Standardization:** Enforce consistent coding styles across team members.
- **Project Conventions:** Automatically adhere to project-specific language and framework preferences.
- **Documentation:** Ensure detailed explanations and annotations in generated code.
- **Testing Automation:** Mandate inclusion of unit tests for every function.

## TIPS AND TRICKS

- **Keep Instructions Concise:** Overly complex instructions can lead to confusion.
- **Segment Instructions by Language:** Use different files for different programming languages to avoid clutter.
- **Leverage External Files:** Reference other instruction files as needed.
- **Testing Changes:** Regularly test to ensure instructions are behaving as expected.

## EXAMPLES

Below are some example code snippets based on our lesson, showcasing the use of custom instructions.

### Python Server Example

```
# CopilotInstructions.md
- Respond like a pirate.
- Use camel case for variables in Python.
```

```
# hello_world.py
# Example Prompt: "Create a server in Python"
# Copilot Response:
def startServer():
    from http.server import HTTPServer, BaseHTTPRequestHandler

    class HelloWorldHandler(BaseHTTPRequestHandler):
        def do_GET(self):
            self.send_response(200)
            self.send_header('Content-type', 'text/plain')
            self.end_headers()
            self.wfile.write(b"Ahoy! Welcome to our ship, matey!")

    server_address = ('', 8000)
    httpd = HTTPServer(server_address, HelloWorldHandler)
    httpd.serve_forever()
```

### JavaScript Variable Conventions

```
# CopilotInstructions.md
- When programming in JavaScript, use the conventions mentioned in JavaScript.txt.
```

```
// JavaScript.txt
- All variable names must be only two letters.
- All variable names must start with var.



// Example Prompt: "Create a script in JavaScript that converts weights"
function wtConverter(wt) {
    var no = document.getElementById("weight").value;
    var nt = no * 2.20462;
    document.getElementById("output").innerHTML = nt + " pounds";
}
```

Utilizing these methods will provide a tailored experience with GitHub Copilot, making it a more integral part of our development workflow.