GIT COPILOT

01 July 2025 17:08

What is GitHub Copilot?

GitHub Copilot is an AI tool that helps you write code faster and smarter.
It’s like having a super-intelligent coding partner sitting next to you.

💡 What Can Copilot Do?

- ✍️ Writes code for you – Just type a comment or a few lines, and it will suggest the rest.
- 💡 Suggests better solutions – It helps you improve your code.
- 🧪 Helps with testing – It can generate test cases.
- 🧹 Refactors code – Makes your code cleaner and better organized.
- 🧠 Explains code – It can explain what complex code is doing in simple language.

🧑‍💻 What is Agent Mode?

This is a new powerful feature in Copilot:

- It can understand your goal and do multi-step tasks.
- It acts more like a smart teammate than just a code helper.
- For example, you can say:
  👉 *“Create a login page and connect it to a database”*
  ➤ Copilot will generate and even improve that whole setup.

💬 How Does It Work?

- You give it natural language prompts (like English instructions).
- Copilot turns those into working code.
- This saves time and reduces the stress of figuring everything out yourself.

◯ Using GitHub Copilot with JavaScript

- What is Prompt Engineering?
- **Prompt Engineering** means **writing good instructions** for AI so it gives you the **right answer**.

- Example-Instead of saying:"Write code"
- You can say:
- "Write a Python function that adds two numbers and returns the result."
- ✅ **4 Ss of Prompt Engineering**
- These are **4 tips** to help you write better prompts, especially when using tools like **GitHub Copilot**:
-
- 1. 🧽 **Specific**
- **Be clear and detailed.**
  Tell the AI *exactly* what you want.
- 🔸 Instead of:
- "Write a function"
- 🔹 Do this:
- "Write a JavaScript function that takes two numbers and returns their sum."
-
- 2. 🎯 **Structured**
- **Organize your prompt** in a logical way—like explaining what you want, what the input is, and what the output should be.
- 🔸 Example:
- "Create a Python function that checks if a number is even. Input: integer. Output: True or False."
- This helps the AI **understand the task better**.
-
- 3. 📚 **Stepped**
- **Break down complex tasks** into smaller steps.
- 🔸 Instead of asking:
- "Build a shopping cart"
- 🔹 Ask step-by-step:
- "1. Create a class for Product.
- Add a method to calculate total price.
- Create a Cart class that holds multiple products."

- This way, Copilot can build code piece by piece — **more accurately**.
-
- 4. 🧩 **Simple**
- Keep your prompt **simple and easy to understand**.
  Avoid vague or overly technical language unless needed.
- 🔸 Use clear terms like:
- "Generate a function to sort a list of numbers in descending order."

- What does "Provide enough context" mean?
- When you're writing code with **GitHub Copilot**, it's like having an AI coding assistant sitting next to you.

  But just like a human, Copilot also needs **clarity** about what you want.

  The more details you give, the **better suggestions** it will give.

Example: No Context (Bad)

```
function calculate() {
  // ...
}
```

This is wrong

Copilot will guess — but it might not know **what to calculate**, so the suggestions may be **wrong or generic**.

Example: With Context (Good)

```
javascript
CopyEdit
// This function calculates total price after tax for an online shopping cart.
// It receives price and tax rate, and returns final price.
function calculateTotal(price, taxRate) {
  // ...
}
```

- Now Copilot knows:
- You’re calculating price
- It includes tax
- What the inputs are

Assert and Iterate – What does it mean?

When you're using **GitHub Copilot**, your first prompt (comment or instruction) might not give the **perfect code** — and **that’s totally fine!**

The trick is to:

1. **Assert** (check) the output.
2. If it’s not correct, **improve your comment** (make it clearer).
3. **Try again** — this is called **iterating**.

Think of Copilot as a coding buddy you're chatting with. If they don’t understand your request the first time, you just explain it better the second time.

Example:

Let’s say you write:

```
// Convert Celsius to Fahrenheit
```

Copilot might give you something like:

```
function convert(temp) {
  return temp * 1.8 + 32;
}
```

✅ This is good!

But maybe you want:

- Input in Celsius
- Output clearly labeled
- Error handling for wrong input

So you **improve your comment**:

```
// Function to convert Celsius to Fahrenheit. Takes a number, returns a string like "25°C is 77°F".
```

Now Copilot gives a **better version** — that’s called **iteration**!

How Copilot Learns From Prompts

Copilot is trained on millions of examples of code.

But to understand your specific need, it looks at:

- Your comments
- Your variable names
- Your existing code

You can help it by:

- Writing good comments
- Giving examples
- Improving unclear prompts

What is Zero-Shot Learning?

- It means Copilot generates code without examples, based only on your comment.
- Ex-

```
// Convert Celsius to Fahrenheit
```

Even without seeing sample code, Copilot can guess the right logic because it's trained on similar patterns before.

What is One-shot learning?

➡️ Imagine you teach someone a new thing by showing just one example.
Example:
You say:
"Look, this is how we convert Celsius to Fahrenheit in code."

```
function celsiusToFahrenheit(c) {

  return (c * 9/5) + 32;
}
```

Then you ask Copilot:

"Now write a function to convert Fahrenheit to Celsius."

✅ Copilot learns from just that one example and gives a smart answer.

What is Few-shot learning?

➡️ You show a few examples (2-3 or more) to give Copilot a better idea.

👶 *Example:*

You give 3 examples:

```
sayHello(8); // "Good morning"
sayHello(14); // "Good afternoon"
sayHello(20); // "Good evening"
```

Then you ask:

"What will sayHello(23) return?"