Python Refresher for AI

Practical Application of AI-Powered Code Generation Tools

Quick Recap

- Functions allow code reusability and modularity by defining reusable blocks of code.
- Object-Oriented Programming (OOP) helps structure programs using classes and objects.
- File handling enables reading, writing, and managing files efficiently in Python.
- Error handling uses try-except blocks to manage and prevent runtime errors.

Engage and Think

You're working on a complex coding project with tight deadlines. A colleague suggests using an AI-powered tool like GitHub Copilot to speed up development. However, you're unsure whether AI-generated code can be trusted for accuracy, efficiency, and security.

Have you ever used AI for coding? If so, what challenges or benefits did you encounter?
If not, what would you expect from an AI-powered coding tool?

Learning Objectives

By the end of this lesson, you will be able to:

- Analyze the role and impact of AI in software development to understand its benefits, challenges, and applications
- Configure AI-powered coding tools like GitHub Copilot to enhance coding efficiency and streamline development workflows
- Construct effective AI prompts to generate high-quality, context-aware code aligned with project requirements
- Evaluate AI-generated code to identify errors, optimize performance, and ensure security and maintainability
- Analyze ethical and legal considerations in AI-assisted coding to make responsible decisions regarding ownership, security, and compliance

Introduction to AI-Powered Code Generation

AI-Powered Code Generation

This helps developers write, optimize, and debug code, making the software development process faster and more efficient.

AI-Powered Code Generation

What is AI-powered coding?

AI-driven tools assist developers by generating, optimizing, and refining code using machine learning. This reduces manual effort and enhances efficiency.

How it works

AI analyzes code structures, predicts the next logical sequence, and suggests improvements based on recognized patterns and best practices.

Why it matters

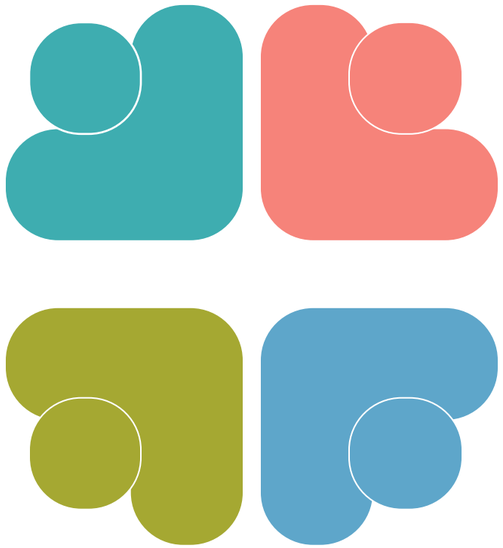
AI accelerates development by automating repetitive tasks, reducing errors and enabling faster debugging, thus boosting productivity.

Why Use AI for Code Generation?

Here are the key benefits of using AI for code generation:

Speed
AI-powered coding significantly reduces development time by offering instant code suggestions.

Productivity impact
AI enables developers to spend more time solving problems and less time on mundane coding tasks.



Efficiency
AI minimizes syntax errors, enforces best practices, and enhances code readability.

Automation
AI assists in writing boilerplate code, repetitive functions, and debugging common issues.

How AI Generates Code

The following points outline how AI-powered code generation streamlines development:

- Generates entire functions from simple comments or partial input
- Processes code as tokens and predicts the next logical sequence
- Learns from open-source repositories, software documentation, and curated datasets
- Uses machine learning models to analyze and predict code patterns

AI Code-Generation Tools

The following AI code-generation tools provide diverse functionalities, catering to different programming needs, from code completion to security analysis and optimization.

GitHub Copilot

Offers real-time, context-aware code suggestions inside various IDEs

tabnine

Tabnine

Provides AI-powered autocompletion that adapts to individual coding styles

Amazon Q Developer

Offers AI-driven code suggestions optimized for AWS cloud development

Quick Check

Which of the following best describes how AI generates code?

A. AI randomly writes code based on past examples

B. AI follows strict predefined templates without variation

C. AI analyzes code patterns, predicts the next logical sequence, and suggests optimizations

D. AI relies solely on user-provided complete code without making predictions

Overview of GitHub Copilot

Introduction to GitHub Copilot

GitHub Copilot is a tool that uses AI to help developers write code more quickly and minimize errors. Below are its key features and advantages:

Real-time code suggestions

Provides real-time, context-aware code snippets as developers write

Multi-language support

Works with various programming languages and frameworks

Context-aware assistance

Analyzes previous lines of code to provide relevant suggestions

Productivity boost

Reduces repetitive coding tasks and enhances development speed

Requirements for Using GitHub Copilot in Various IDEs

GitHub Copilot allows developers to incorporate AI-driven assistance into their workflow. Below are the key system requirements and compatible platforms for installation:

Supported IDEs

Works with VS Code, JetBrains, Neovim, and other popular IDEs

GitHub account requirement

Requires an active GitHub subscription to enable Copilot

Installation process

Involves installing the Copilot extension and signing into GitHub

Configuration settings

Customizable options allow users to optimize AI-generated suggestions

Demo: Installing GitHub Copilot in VS Code

Duration: 05 minutes

Overview:

This demo will guide you through the process of installing GitHub Copilot in Visual Studio Code (VS Code) to enable AI-powered code suggestions. You will learn how to access the VS Code marketplace, install the GitHub Copilot extension, and sign in with a GitHub account. By the end of this demo, you will have GitHub Copilot successfully installed and ready to assist in writing code.

DEMONSTRATION

Configuration Settings for Optimal AI Assistance

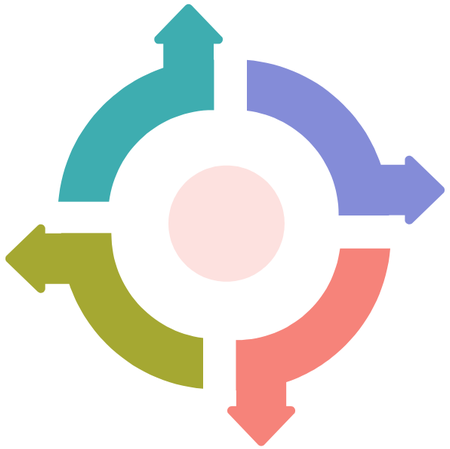
Once GitHub Copilot is installed, configuring its settings ensures accurate, relevant, and secure AI-generated suggestions. Below are the key settings developers should customize:

Language-specific tuning

Adjusts settings to improve AI suggestions for different programming languages

Security and privacy controls

Manages AI interactions to prevent unauthorized access or insecure code recommendations



File-type preferences

Enables or disables Copilot for specific project files to maintain control over code suggestions

Coding style adaptation

Learns from developer inputs to refine and personalize suggestions

Demo: Customizing Copilot Settings in VS Code

Duration: 05 minutes

Overview:

This demo will showcase how to customize GitHub Copilot settings in Visual Studio Code (VS Code) to tailor AI-generated code suggestions to your workflow. You will learn how to adjust Copilot's behavior, enable or disable inline suggestions, and configure settings for specific programming languages. By the end of this demo, you will have a personalized Copilot experience that enhances productivity and code quality.

DEMONSTRATION

Quick Check

Which of the following steps must be completed to successfully install GitHub Copilot in VS Code?

A. Download and install the GitHub Copilot extension from the VS Code marketplace

B. Write a custom configuration file before enabling GitHub Copilot

C. Use GitHub Copilot without signing into a GitHub account

D. Install additional third-party plugins for Copilot to work

Effective AI Prompts

Introduction to AI Prompts

AI prompts serve as input instructions that help AI generate relevant code. A well-structured prompt improves the quality of AI-generated code, making it more useful and aligned with project requirements.

AI Prompts: Insights

Below are the key insights on AI prompts and their impact on AI-generated outputs:

Impact on code quality	Role of structured prompts	Consequences of poor prompts	Best practices for effective prompts
AI prompts influence the accuracy and efficiency of AI-generated code.	Structured prompts help AI understand coding context and produce meaningful outputs.	Poorly written prompts can lead to incorrect or incomplete code suggestions.	Best practices include using clear instructions, defining constraints, and refining prompts iteratively.

Effective Prompting Techniques for Coding

Writing clear and structured prompts helps AI generate high-quality, accurate code. Below are key techniques for improving AI-generated outputs:

Use precise instructions

Specify what AI should generate to reduce ambiguity

Provide examples

Show AI the expected output format to improve context and accuracy

Break down complex tasks

Divide multi-step logic into smaller, manageable prompts for better results

Iterate and refine

Test and adjust prompts continuously to enhance AI-generated responses

Exploration of Various Prompt Styles

The effectiveness of AI-generated code depends on how prompts are structured. Below are different prompt styles and their impact on code output:

Inline comments

Add comments within the code to guide AI's understanding

Docstring-based prompts

Use structured docstrings to define function behavior before AI generates code

Contextual hints

Provide relevant details about expected inputs, outputs, or performance constraints

Partial code completion

Writes part of the code and lets AI predict the rest

Examples of AI Prompt Styles

Different AI prompt styles influence the quality of AI-generated code. Below are examples of various prompt styles and how they guide AI assistance:

Inline comments

```
# Create a function to calculate factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Impact: AI understands the purpose of the function and suggests an optimized implementation.

Docstring-based prompts

```
def find_maximum(numbers):
    """
    This function takes a list of numbers and returns the maximu
    Args:
        numbers (list): A list of numerical values.
    Returns:
        int: The maximum number in the list.
    """
```

Impact: AI generates a complete function by understanding the function behavior from the docstring.

Examples of AI Prompt Styles

Different AI prompt styles influence the quality of AI-generated code. Below are examples of various prompt styles and how they guide AI assistance.

Contextual hints

```
# Define a function to reverse a string
# The function should handle both uppercase and lowercase
```

Impact: AI incorporates additional logic based on the provided context.

Partial code completion

```
def is_prime(n):
    for i in range(2, n):
```

Impact: AI predicts the missing logic to complete the function.

Demo: Comparing AI Responses to Different Prompt Styles

Duration: 10 minutes

Overview:

This demo will explore how different prompt styles impact AI-generated responses. You will compare various prompting techniques, such as inline comments, docstring-based prompts, contextual hints, and partial code completion, to see how AI interprets and generates code. By the end of this demo, you will understand how to craft effective prompts for better AI-assisted coding.

DEMONSTRATION

Quick Check

What is the primary purpose of an AI prompt?

A. To write code automatically without developer input

B. To guide AI in generating meaningful and relevant code

C. To replace traditional coding practices completely

D. To test AI’s ability to understand human language

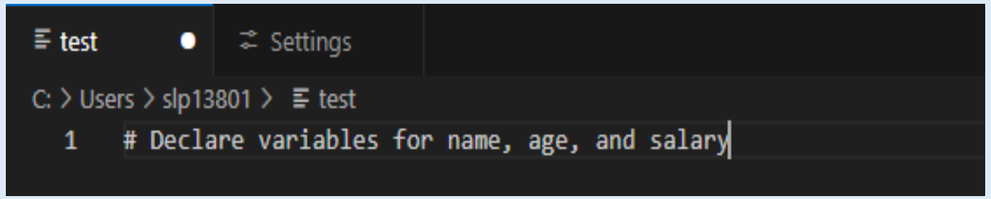
Basic Python Operations with GitHub Copilot

Copilot for Variable Declarations and Data Types

It creates code from written comments, allowing them to instruct the AI with directives within the code.

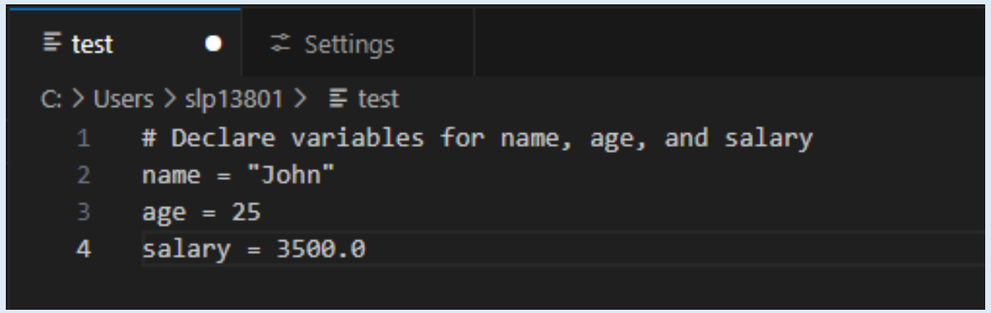
Example: Basic variable declarations

1. Open a new Python file (Ctrl + N or Cmd + N)
2. Type a comment describing the variables you want



3. Press **Enter** and wait for Copilot to generate the full code
4. Press **Tab** to accept the suggestion

Output:

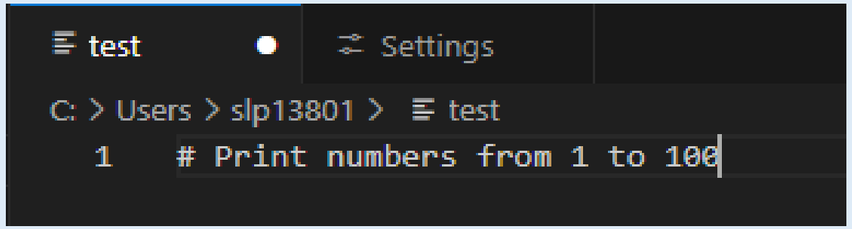


Copilot for Loops

It generates loops automatically based on comments or incomplete code.

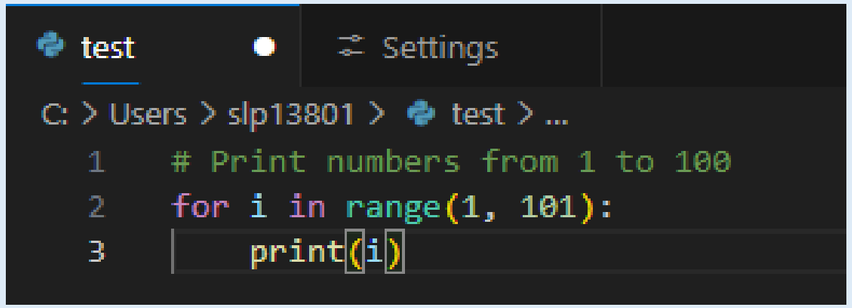
Example: Creating a *for* loop

1. Type a comment describing the loop you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

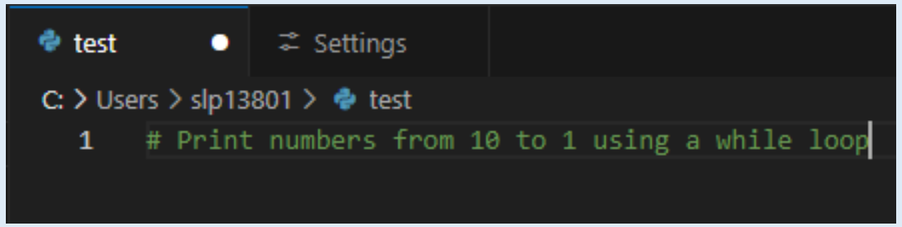


Copilot for Loops

Copilot can generate loops automatically based on comments or incomplete code.

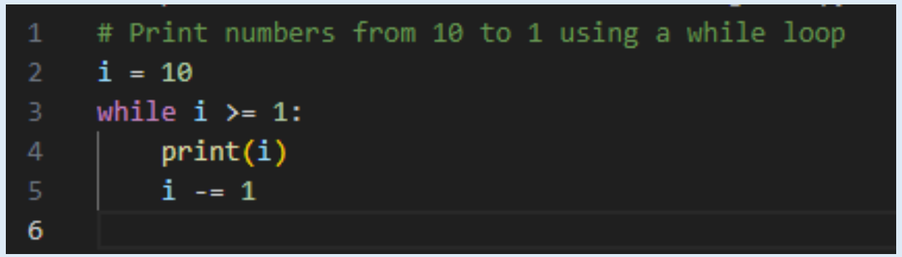
Example: Creating a *while* loop

1. Type a comment describing the loop you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

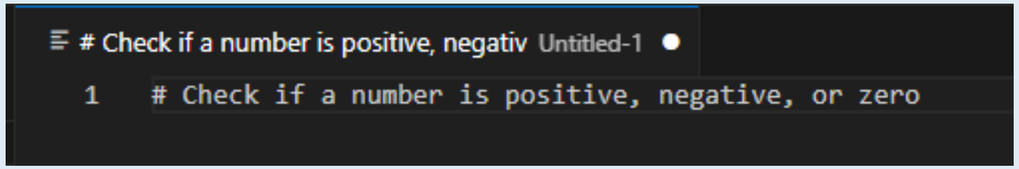


Copilot for Conditionals

It helps generate if-else conditions based on descriptions.

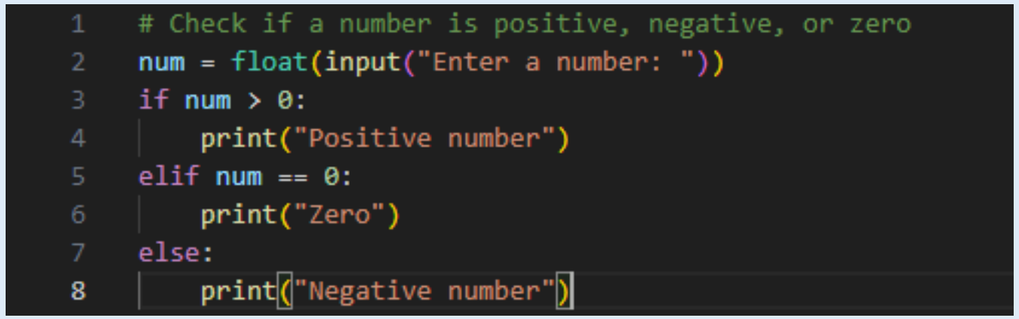
Example: Creating an *if-else* condition

1. Type a comment describing the loop you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

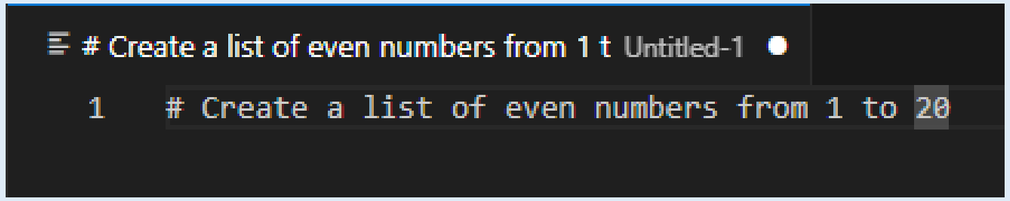


Copilot for Lists, Dictionaries, and Tuples

It generates lists, dictionaries, and tuples when provided with a descriptive comment.

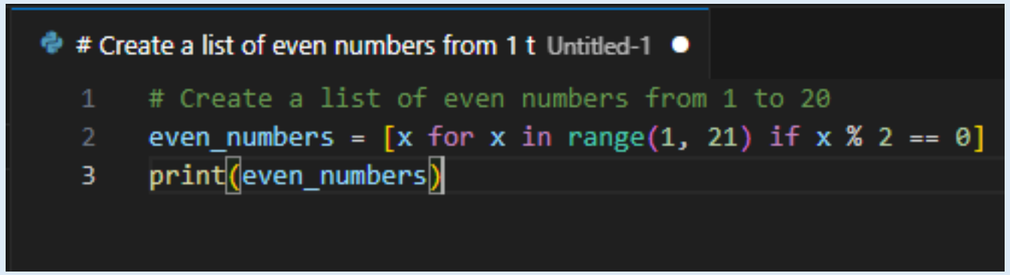
Example: Creating a list

1. Type a comment describing the list you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

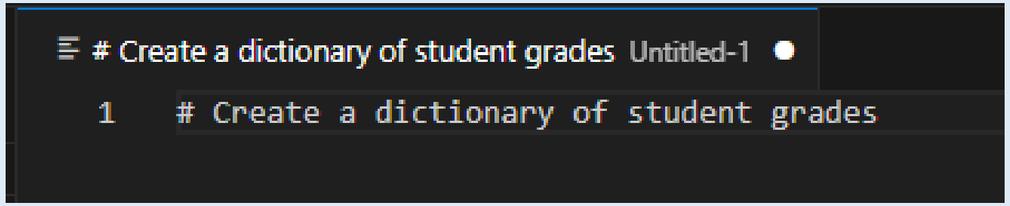


Copilot for Lists, Dictionaries, and Tuples

It generates lists, dictionaries, and tuples when provided with a descriptive comment.

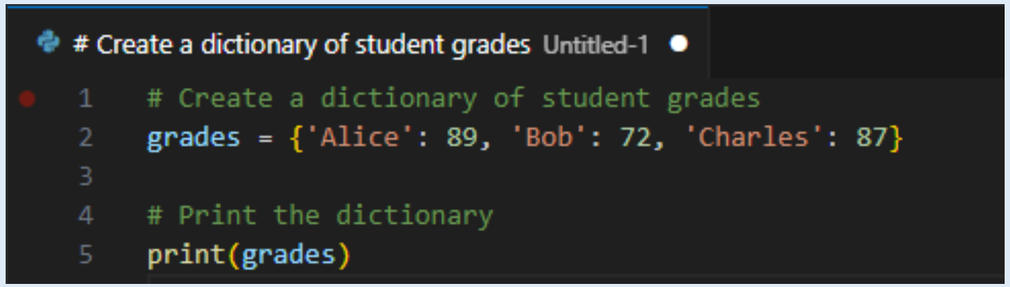
Example: Creating a dictionary

1. Type a comment describing the dictionary you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

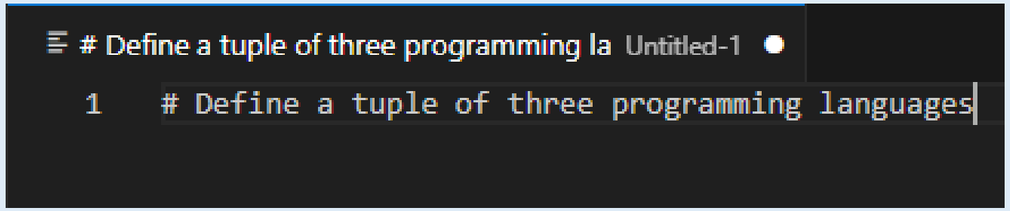


Copilot for Lists, Dictionaries, and Tuples

It generates lists, dictionaries, and tuples when provided with a descriptive comment.

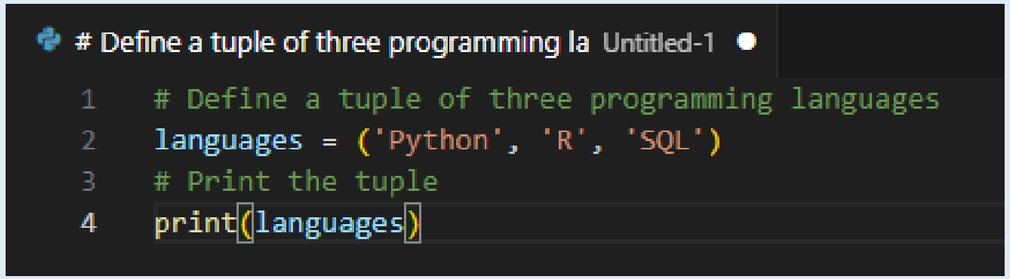
Example: Creating a tuple

1. Type a comment describing the tuple you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:



Quick Check

How does GitHub Copilot assist with writing basic Python operations?

A. It automatically completes code based on context and learned patterns.

B. It modifies Python’s syntax to create a new programming language.

C. It replaces the need for developers to understand Python fundamentals.

D. It restricts users to access predefined code templates without customization.

Copilot for Generating Functions

Function Structure in Python

Here is the breakdown of how functions are structured in Python:

Function syntax in Python

```
def function_name(parameters):
    """This is a docstring explaining the
function."""
    # This is an inline comment explaining
a specific line of code.
    return output
```

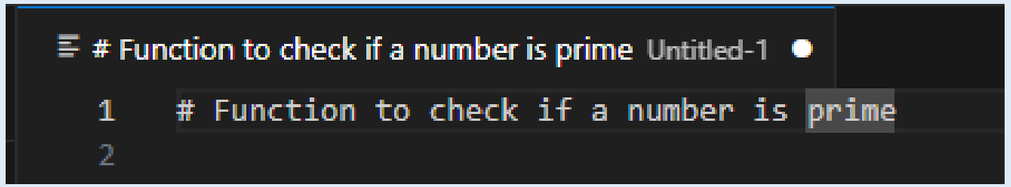
- **def** → Defines the function
- **parameters** → Pass optional arguments to the function
- **"""Docstring"""** → Describes what the function does
- **Inline Comments (#)** → Explain specific lines of code
- **return** → Outputs a result

Method 1: Generating a Function Using a Comment

It generates function logic based on a descriptive comment.

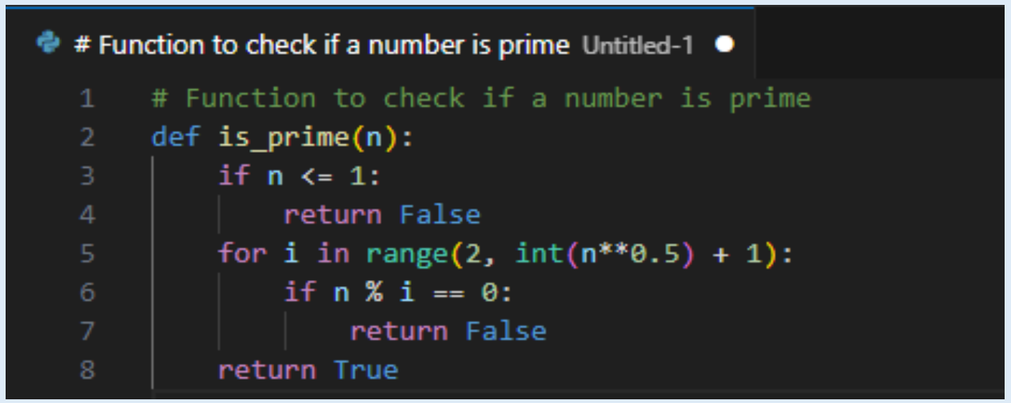
Example: Prime number checker

1. Type a comment describing the function you want to create



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

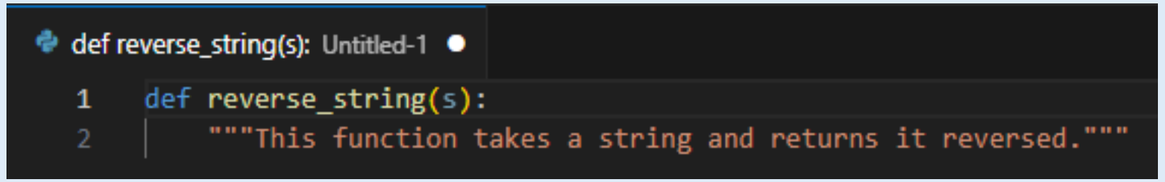


Method 2: Generating a Function Using a Docstring

It generates function logic when you start with a docstring.

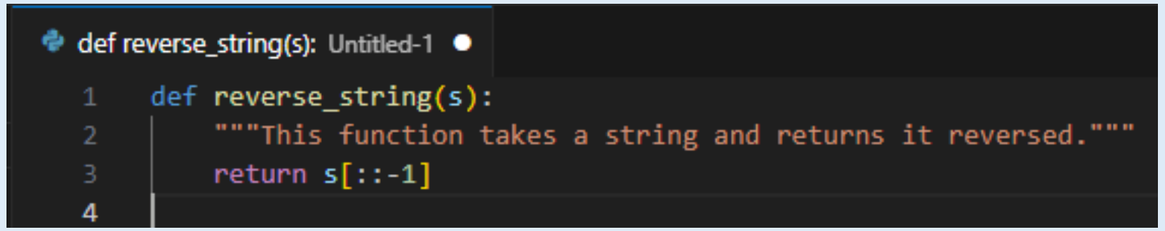
Example: String reversal

1. Type the function definition followed by a docstring



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:



Best Practices for Writing Functions with Copilot

Use comments before writing code

Helps Copilot generate better functions

Manually verify output

Prevents errors in AI-generated code

Use Copilot for repetitive tasks

Speeds up development

Demo: Creating a Function Using GitHub Copilot

Duration: 05 minutes

Overview:

This demo will guide learners in creating a Python function with the help of GitHub Copilot. Copilot will assist in auto-completing function definitions, suggesting parameters, and optimizing logic based on the function's intended purpose. By the end of this demo, learners will understand how to leverage AI-powered suggestions to write efficient and well-structured functions.

DEMONSTRATION

Quick Check

What will happen if you define a Python function with a meaningful name but without any comments or docstrings while using GitHub Copilot?

A. Copilot will generate an accurate function body with complete logic.

B. Copilot may generate a function, but it could lack important details or correct logic.

C. Copilot will ignore the function definition and provide no suggestions.

D. Copilot will automatically fetch a function from the internet and insert it.

GitHub Copilot for Object-Oriented Programming (OOP)

OOP Structure in Python

This is how Python's Object-Oriented Programming (OOP) works:

OOP syntax in Python

```
class ClassName:
    def __init__(self, attribute1, attribute2):
        self.attribute1 = attribute1
        self.attribute2 = attribute2

    def method_name(self):
        # Function logic
        return something
```

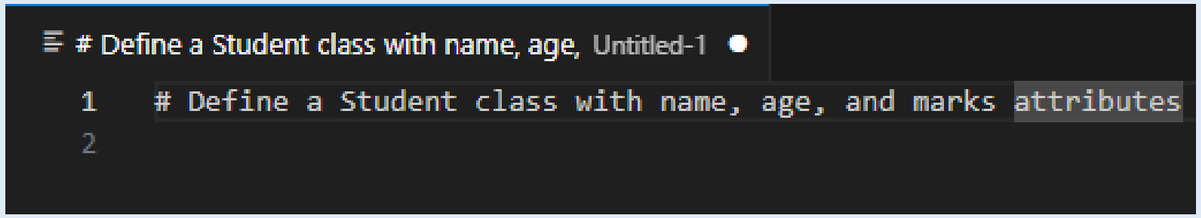
- **Class** → A blueprint for creating objects
- **Object** → An instance of a class
- **Attributes** → Variables that store data in a class
- **Methods** → Functions defined inside a class to operate on its data
- **Constructor (__init__)** → Initializes objects with default values

Basic Class with Copilot

Copilot automatically completes class structures when provided with a descriptive comment.

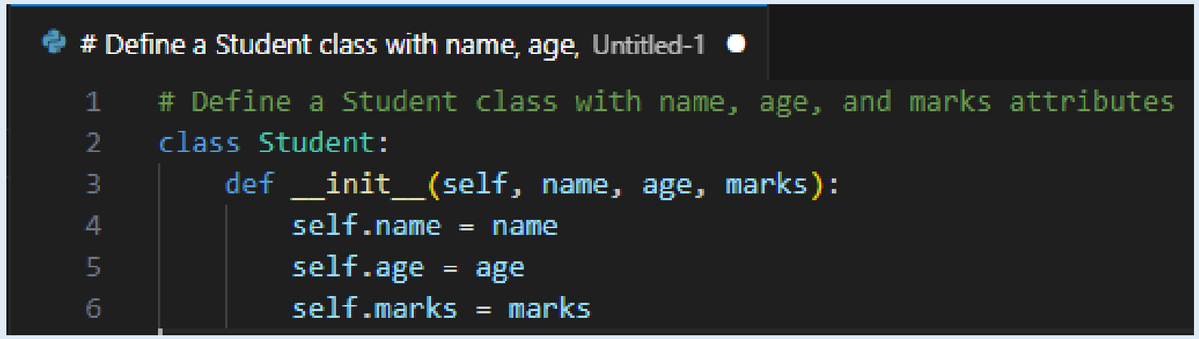
Example: Creating a student class

1. Type a comment describing the class



2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:



Addition of Methods to a Class Using Copilot

It generates class methods when prompted.

Example: Adding a method to calculate grades

1. Type a comment inside the class we have created earlier

```
8   # Method to calculate the grade based on marks
9
```

2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

```
8   # Method to calculate the grade based on marks
9       def calculate_grade(self):
10          if self.marks >= 80:
11              return 'A'
12          elif self.marks >= 60:
13              return 'B'
14          elif self.marks >= 40:
15              return 'C'
16          else:
17              return 'F'
```

Creating Objects Using Copilot

Copilot helps to create objects after defining the class.

Example: Instantiating and using the Student class

1. Type a comment to create an object

```
19    # Create a Student object and print their grade
20
```

2. Press **Enter** and wait for Copilot to generate the full code
3. Press **Tab** to accept the suggestion

Output:

```
19    # Create a Student object and print their grade
20    student1 = Student('John', 20, 90)
21    print(student1.calculate_grade())
```

Best Practices for OOP with GitHub Copilot

1

Write class comments before code

Helps Copilot generate more relevant structures

2

Use docstrings inside methods

Improves readability and documentation

3

Verify class output manually

Prevents errors in the code and ensures the logic is correct

4

Leverage Copilot for repetitive patterns

Saves time when defining multiple classes

Quick Check

How does GitHub Copilot assist in writing Object-Oriented Programming (OOP) code in Python?

A. It automatically generates class structures, methods, and attributes based on context.

B. It rewrites Python's OOP principles to create a new programming paradigm.

C. It removes the need for defining constructors and instance variables manually.

D. It restricts developers from modifying generated class methods.

Debugging and Refining AI-Generated Code

Introduction to AI-Generated Errors and Hallucinations

AI-generated code is powerful but often produces errors or hallucinations that must be carefully reviewed. Below are key reasons why understanding AI errors is essential:

AI models generate code based on patterns, which can result in unexpected mistakes.

AI lacks true reasoning and may misinterpret logic or requirements.

AI may generate generic or inefficient solutions that need optimization.

Common AI-Generated Errors and Hallucinations

AI-generated code often introduces various issues that need careful debugging. Below are the most common types of errors:

Syntax errors

It may suggest incorrect or outdated syntax.

Logical errors

It may misunderstand requirements, producing flawed logic.

Hardcoded or inefficient solutions

AI-generated code might not be scalable.

Real-World Example: Misinterpretation of Logic by AI

When AI misinterprets logic, it may produce incorrect calculations or results. Below is an example of such an issue:

Problem: AI generates incorrect logic for a simple sum of even numbers function.

AI-suggested code:

```
def sum_even_numbers(n):
    return sum(n for n in range(n) if n % 2 == 0)
```

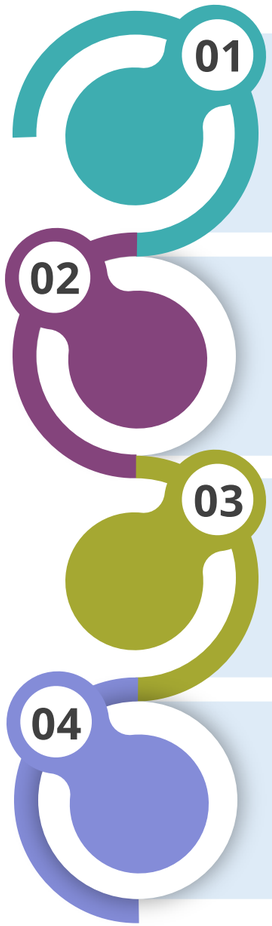
Corrected code:

```
def sum_even_numbers(n):
    return sum(i for i in range(1, n+1) if i % 2 == 0)
```

Key fix: The AI excluded the upper bound when interpreting the specified range, leading to inaccurate results.

Debugging AI-Suggested Code: Step-by-Step Approach

Debugging AI-generated code requires a systematic approach to identify and resolve the errors. Follow these steps systematically:



Understand the code

Read and analyze the AI-suggested logic

Run test cases

Check if expected and actual outputs match

Add print statements

Trace variables and execution flow

Use debugging tools

Utilize breakpoints and step-through debugging

Example: Fixing Incorrect AI-Generated Function Logic

Problem: AI generates a function to check if a number is prime, but the logic is flawed.

AI-suggested code:

```
def is_prime(n):
    return all(n % i != 0 for i in range(2, n))
```

Issues:

- The function incorrectly checks divisibility up to n, instead of up to sqrt(n).
- It does not handle edge cases like n < 2.

Example: Fixing Incorrect AI-Generated Function Logic

Problem: AI generates a function to check if a number is prime, but the logic is flawed.

Debugging steps:

1. Review AI-generated logic: The AI checks divisibility but uses an incorrect range
2. Identify missing conditions: Numbers less than 2 should return False
3. Optimize efficiency: The prime-check should only iterate up to sqrt(n)
4. Implement fixes and test the function

Corrected code:

```
def is_prime(n):
    if n < 2:
        return False
    return all(n % i != 0 for i in range(2, int(n**0.5) + 1))
```

Demo: Debugging AI-Generated Code Using GitHub Copilot

Duration: 05 minutes

Overview:

This demo will guide learners through the process of identifying and fixing errors in AI-generated code using GitHub Copilot. Learners will explore common issues such as syntax errors, logical errors, and inefficiencies, and use debugging techniques like print statements, error messages, and test cases to refine AI-suggested code. By the end of this demo, learners will be able to validate, debug, and optimize AI-generated code for accuracy and efficiency.

DEMONSTRATION

Quick Check

You're using GitHub Copilot to generate code for a project. The code works, but you want to refine it for better maintainability.
Which is the best practice?

A. Adding meaningful comments and docstrings to explain the code

B. Hardcoding values to prevent unexpected inputs

C. Avoiding test cases if the AI-generated code runs without errors

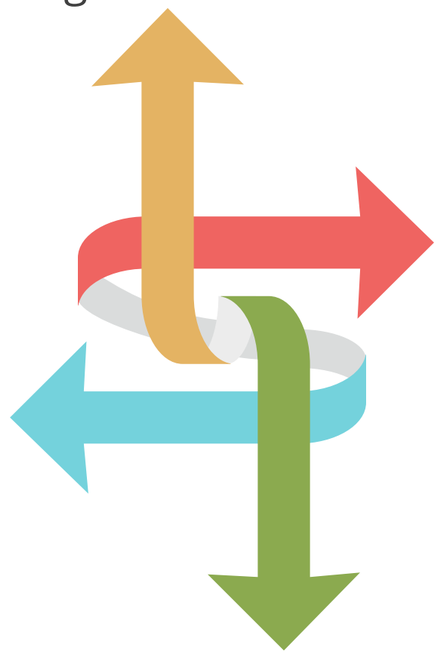
D. Keeping AI-generated code unchanged to maintain its originality

Ethical and Legal Considerations in AI Code Generation

Introduction to Ethics in AI Code Generation

AI-generated code is transforming software development, but it raises critical ethical and legal questions. Below are key reasons why ethics in AI-assisted coding is essential:

Ensures fairness and reduces bias in AI-generated solutions



Clarifies ownership and licensing of AI-assisted code

Encourages responsible AI usage by following best practices

Prevents security vulnerabilities that could arise from AI-generated code

AI Bias and Fairness in Code Suggestions

Bias in AI-generated code can lead to unintended consequences. Below are key factors contributing to bias:

Training data limitations

AI models learn from existing data, which may contain biased patterns.

Algorithmic bias

Some AI models prioritize efficiency over fairness, leading to biased suggestions. Bias can arise from training objectives, data representation, or model design.

Lack of context awareness

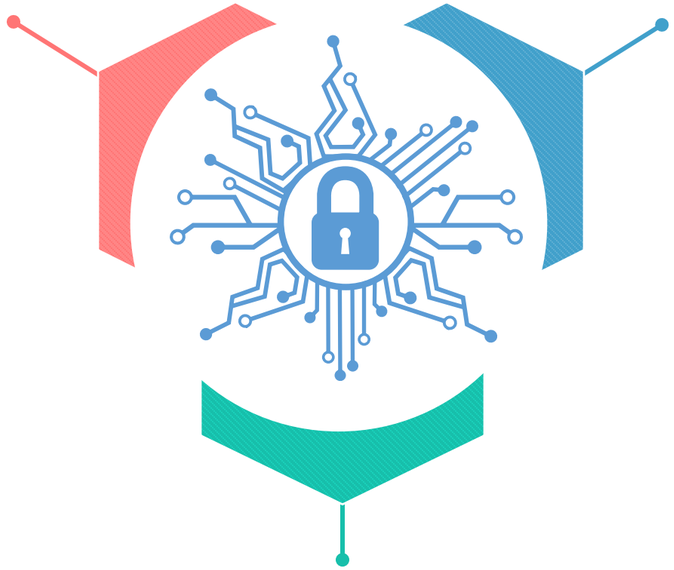
AI lacks social and ethical awareness, making it unable to evaluate the fairness or implications of its generated code.

Security and Privacy Challenges in AI-Generated Code

The legal status of AI-generated code is still evolving. Below are key ownership considerations:

Unvalidated inputs

AI-generated code may not sanitize user input, creating security risks.



Data leaks

AI-generated code may expose sensitive data like API keys or credentials.

Weak security

AI-generated authentication may allow unauthorized access.

Best Practices for Responsible AI-Assisted Coding

To mitigate ethical and legal risks, follow these best practices:

- Review AI-generated code manually to detect and correct biases
- Use ethical AI tools that comply with industry standards

- Validate code security by running security checks on AI-assisted outputs
- Adhere to licensing rules when incorporating AI-generated code into projects

Quick Check

Which of the following is most important when evaluating the ethical impact of AI-generated code?

A. Ensuring the AI-generated code follows industry coding standards

B. Checking if AI-generated code is free from bias, security risks, and legal conflicts

C. Using AI-generated code only for non-commercial projects

D. Measuring the speed at which AI generates the code

Guided Practice

Overview

Duration: 20 minutes

In this guided practice, you will build a simple number guessing game using Python and GitHub Copilot in VS Code. You will use inline comments and docstrings to guide Copilot in generating structured and optimized code. By the end, you will have a working game where the player tries to guess a randomly generated number within a set number of attempts.

GUIDED PRACTICE

Key Takeaways

- AI-powered coding tools like GitHub Copilot assist in writing, optimizing, and debugging code, enhancing efficiency and reducing manual effort.
- Effective AI prompting techniques improve code generation by providing clear, structured, and contextual instructions.
- Evaluation of AI-generated code ensures accuracy, security, and maintainability, requiring manual review and testing.
- Ethical and legal considerations in AI-assisted coding highlight the importance of responsible usage, bias detection, and adherence to licensing rules.

Q&A