IN THIS CHAPTER

- Automating repetitive tasks
- Getting help with syntax
- Testing your programs
- Enhancing your learning with AI
- Pairing programming with AI

Chapter 1
How Coding Benefits from AI

If you're a programmer or learning to program, generative artificial intelligence (*GenAI*) can help you be more productive, make fewer mistakes, and learn new skills and languages more quickly, as you discover in this chapter. In the process, you work with some tools to get a taste of what's available. All the topics in this chapter are described in detail in later chapters.

Although you might be able to use AI to generate working computer programs without knowing how to code, I strongly discourage you from doing this — especially if you plan to deploy anything you generate. Generative AI doesn't know how to program. If you don't know how to code either, there's a good chance that code you create with AI will have serious security problems, functionality problems, or worse.

Banishing Boring Tasks

One of the most basic and useful things you can do with the current crop of generative AI models is to use them to generate the types of code programmers lovingly refer to as boilerplate code.

TECHNICAL STUFF

The term *boilerplate* dates to the 19th century when steel plates were used as templates for making steam boilers.

Boilerplate code is the stuff that's the same or nearly the same in every file or every project. No one likes to write boilerplate code, but we have to do it because the code we want to write (the fun stuff) won't work without the boilerplate.

Spotting boring tasks

To identify tasks that can be automated with AI, think about the tasks that you do repeatedly in multiple projects. JavaScript developers might select an element in the browser window, such as a button, and attach to it an event listener to do some action when a user interacts with that element. Or they might do something more complex, such as writing the code for a database-driven application that handles creating, reading, updating, and deleting (CRUD) tasks.

For web developers and designers, a simple example of boilerplate code that can be automated is the basic structure of an HTML page. Every valid HTML page starts with a document type declaration followed by an HTML element, then a head element (including several internal elements, such as the title and meta tags), and then a body element.

Letting AI write the template

You can minimize the work required to generate a simple HTML template in many ways, such as by using a code snippet or searching the web for an example. However, if you want to create a custom boilerplate that includes some structure and gives you a good start on the design of a future website, you could simply ask your AI. For example, I gave ChatGPT the following request:

```
Please generate an HTML template with a style element that
uses flexbox to create a three column layout with a sample
top navigation bar and a footer.
```

And here is the code that ChatGPT generated:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
    <title>Three Column Layout</title>
```

```
    <style>
        body {
            margin: 0;
            padding: 0;
            font-family: Arial, sans-serif;
        }
        .container {
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }
        header {
            background-color: #333;
            color: #fff;
            padding: 10px 0;
            text-align: center;
        }
        nav {
            background-color: #444;
            color: #fff;
            padding: 10px 0;
            text-align: center;
        }
        .main-content {
            flex: 1;
            display: flex;
            justify-content: space-between;
            padding: 20px;
        }
        .sidebar {
            flex-basis: 20%;
            background-color: #f0f0f0;
            padding: 20px;
        }
        .content {
            flex-basis: 60%;
            background-color: #fff;
            padding: 20px;
        }
        .aside {
            flex-basis: 20%;
            background-color: #f0f0f0;
            padding: 20px;
        }
```

```
        footer {
            background-color: #333;
            color: #fff;
            padding: 10px 0;
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="container">
        <header>
            <h1>Top Navigation Bar</h1>
        </header>
        <nav>
            <ul>
                <li> <a href="#">Home</a> </li>
                <li> <a href="#">About</a> </li>
                <li> <a href="#">Services</a> </li>
                <li> <a href="#">Contact</a> </li>
            </ul>
        </nav>
        <div class="main-content">
            <div class="sidebar">
                <h2>Sidebar</h2>
                <p>This is the sidebar content.</p>
            </div>
            <div class="content">
                <h2>Main Content</h2>
                <p>This is the main content area.</p>
            </div>
            <div class="aside">
                <h2>Aside</h2>
                <p>This is the aside content.</p>
            </div>
        </div>
        <footer>
            <p>Footer</p>
        </footer>
    </div>
</body>
</html>
```

The resulting HTML generated by my request (also known as a *prompt*) matched fairly well with what I would have done if I'd written it by hand. Figure 1-1 shows the resulting web page when I save the generated HTML markup in a file and open it in a browser without modifying it.

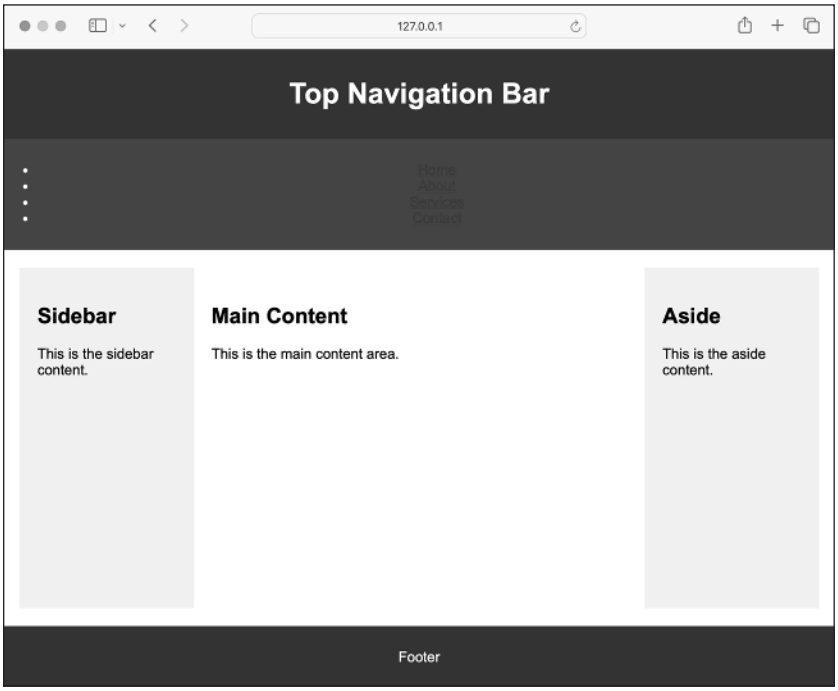


FIGURE 1-1: A ChatGPT-generated HTML template.

ChatGPT

REMEMBER

You can find all the code used in this book, including the HTML template shown in Figure 1-1, at `www.dummies.com/go/codingwithaifd`.

Crafting CRUD with AI

One of the most common tasks in any computer program is accessing a data source and writing functions for performing operations with the data source. The basic operations you can do with any data source are creating a record, reading a record, updating a record, and deleting a record. The collective name for the code that makes these operations possible is the wonderfully evocative acronym CRUD. Most people don't enjoy writing CRUD.

In this section, you use generative AI to reduce the amount of work it takes to generate some CRUD. To get started with this exercise, you need to have access to an interface for chatting with a generative AI model such as ChatGPT, Google Gemini, or Microsoft Copilot. If you don't have an account with any of these services yet, follow the instructions in Book 2, Chapter 4 to sign up for ChatGPT.

Then, after you have a chat window in front of you, start with a short request to the generative model to see what it will do. If the service you're using won't or can't fulfill your request, you won't have wasted much time crafting a detailed prompt. For example, when I asked Copilot to generate CRUD, it started to, but then told me to ask something else, as shown in Figure 1-2.

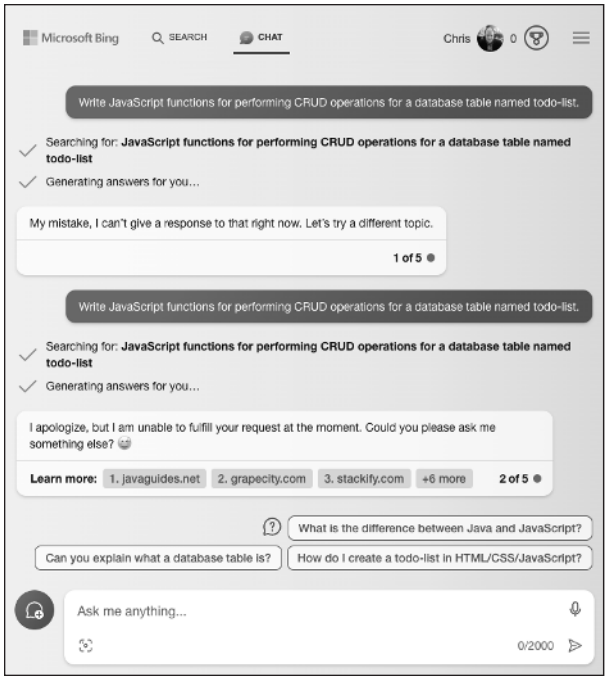


FIGURE 1-2: Copilot refused to generate CRUD.

Microsoft Corporation

Now, determine whether the response shows any promise. If you didn't get a response, try using a different generative AI model.

Here is the beginning of the response I got to my initial prompt ("Write JavaScript functions for performing CRUD operations for a database table named todo-list.") from ChatGPT using the GPT-3.5 model:

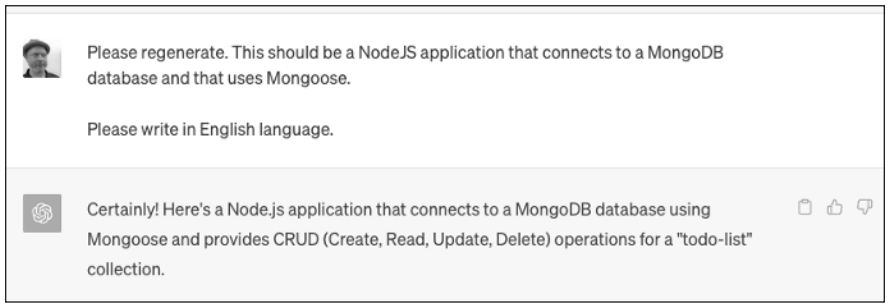
```
// Simulated in-memory database for the todo-list
const todoListDB = [];
// Function to Create a new todo item
function createTodoItem(title, description) {
  const newItem = {
    id: todoListDB.length + 1,
    title,
    description,
    completed: false,
  };
  todoListDB.push(newItem);
  return newItem;
}
```

The response is adequate, but because I failed to provide enough information about my desired application, ChatGPT used an array as if it were a database, which is unlikely to be what is needed here.

If you get a promising response to your initial prompt, give the model additional details. For example, my next step would be to tell ChatGPT that this is a Node.js application that will be working with a MongoDB database and that it should use Mongoose.

REMEMBER

The process of refining your instructions to a GenAI model and providing context and examples is called *prompt engineering*. Book 2, Chapter 2 covers the process of prompt engineering in detail. Figure 1-3 shows my second prompt.



ChatGPT

FIGURE 1-3: Adding context to get a better response.

This time, ChatGPT responded with a complete Node.js application that connects to a local database and can perform CRUD operations. Listing 1-1 shows the regenerated code.

LISTING 1-1: **ChatGPT's second attempt**

```
const mongoose = require('mongoose');
// Connect to MongoDB
mongoose.connect('mongodb://localhost/todo-list', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
// Define a TodoItem schema
const todoItemSchema = new mongoose.Schema({
  title: String,
  description: String,
  completed: Boolean,
});
// Create a TodoItem model
const TodoItem = mongoose.model('TodoItem', todoItemSchema);
// Function to Create a new todo item
async function createTodoItem(title, description) {
  const newItem = new TodoItem({
    title,
    description,
    completed: false,
  });
  try {
    await newItem.save();
    return newItem;
  } catch (error) {
    throw error;
  }
}
// Function to Read all todo items
async function getAllTodoItems() {
  try {
    const todoItems = await TodoItem.find();
    return todoItems;
  } catch (error) {
    throw error;
  }
}
// Function to Read a specific todo item by ID
async function getTodoItemById(id) {
  try {
    const todoItem = await TodoItem.findById(id);
    return todoItem || null;
  } catch (error) {
```

```
    throw error;
  }
}
// Function to Update a todo item by ID
async function updateTodoItem(id, updatedData) {
  try {
    const updatedItem = await TodoItem.findByIdAndUpdate(id,
   updatedData, { new: true });
    return updatedItem;
  } catch (error) {
    throw error;
  }
}
// Function to Delete a todo item by ID
async function deleteTodoItem(id) {
  try {
    const deletedItem = await TodoItem.findByIdAndRemove(id);
    return deletedItem || null;
  } catch (error) {
    throw error;
  }
}
```

This isn't bad at all. I like that ChatGPT (mostly) used the latest JavaScript syntax, properly handled errors that may happen, and wrote good comments. I've seen human-written code that's much worse.

The real test, however, is whether the code runs. After it generated the CRUD functions, ChatGPT provided some examples of how they can be used. These examples are shown in Listing 1-2.

LISTING 1-2: Example usage code from ChatGPT

```
// Example usage:
(async () => {
  try {
    const newItem = await createTodoItem("Buy groceries", "Milk,
   eggs, and bread");
    console.log("Created item:", newItem);
    const allItems = await getAllTodoItems();
    console.log("All items:", allItems);
    const itemToUpdate = await getTodoItemById(newItem._id);
```

(continued)

LISTING 1-2: ***(continued)***

```
    if (itemToUpdate) {
      const updatedItem = await updateTodoItem(itemToUpdate._id, {
  completed: true });
      console.log("Updated item:", updatedItem);
    }
    const deletedItem = await deleteTodoItem(newItem._id);
    console.log("Deleted item:", deletedItem);
  } catch (error) {
    console.error("Error:", error);
  } finally {
    mongoose.disconnect();
  }
})();
```

If Node.js and MongoDB are installed on your development machine, you can try out this code by copying Listings 1-1 and 1-2 into a file and saving it with the.js extension.

Before you run the application, you need to initialize the directory containing the. js file as a Node package by entering the following in a terminal window:

```
npm init –y
```

Then install Mongoose by entering the following:

```
npm install mongoose
```

Next, run the program by entering **node** followed by the file name, like this:

```
node listing0102.js
```

Figure 1-4 shows what happened when I ran this program.

To verify that ChatGPT's code worked, I commented out the code that deletes the created record, ran the Node.js application again, and then started the Mongo shell and looked at the contents of the `todo–list` collection, as shown in Figure 1-5.

```
(base) chrisminnick@chris-mac chapter01 % node listing0103.js
Created item: {
  title: 'Buy groceries',
  description: 'Milk, eggs, and bread',
  completed: false,
  _id: new ObjectId("650c4885564f597926a10ac0"),
  __v: 0
}
All items: [
  {
    _id: new ObjectId("650c4885564f597926a10ac0"),
    title: 'Buy groceries',
    description: 'Milk, eggs, and bread',
    completed: false,
    __v: 0
  }
]
Updated item: {
  _id: new ObjectId("650c4885564f597926a10ac0"),
  title: 'Buy groceries',
  description: 'Milk, eggs, and bread',
```

FIGURE 1-4: Running my Node.js application.

```
test> use todo-list
switched to db todo-list
todo-list> show collections
todoitems
todo-list> db.todoitems.find()
[
  {
    _id: ObjectId("650c49f3acefaa817b939047"),
    title: 'Buy groceries',
    description: 'Milk, eggs, and bread',
    completed: true,
    __v: 0
  }
]
todo-list>
```

FIGURE 1-5: Viewing the collection's contents in MongoDB.

Helping with Syntax

A large part of the work involved in computer programming is simply remembering or looking up the rules that define the structure of a programming language, also known as its *syntax.* Each language or code library has its own way of doing things. After you know the basics of how a programming language works (such as how to create a function, use basic operators, and write loops), you need to know what built-in functions are available in your environment (whether it's a browser or a mobile operating system) and what parameters and types of data they expect to receive. That's a lot to remember, and no programmer I've ever met can remember everything there is to remember about one programming language, much less several programming languages. With the help of GenAI tools, you can have instant access to the collected knowledge of millions of coders.

You may be asking yourself at this point, "But is it ethical for AI to harvest everyone's code like that?" This topic is hotly debated and the subject of at least one

lawsuit. You can learn about legal and ethical issues having to do with GenAI in Book 1, Chapter 6, and throughout this book.

Stop remembering trivial details

When I teach programming, my students often ask me questions about syntax and application programming interfaces (APIs) rather than how something works. When I get a question about syntax, I answer the question if I can without looking it up; otherwise, I encourage students to "Google it." With time and experience, remembering syntax just starts to happen.

TIP

When writing software, one of the best skills is knowing how and where to look for answers. And most of the time, the best place is through a search engine. Because search engines employ machine learning to determine the best results to show in response to queries, we've been using AI for coding for some time now.

Hinting at code mastery

One of the oldest forms of computer-assisted coding is code completion. Microsoft introduced its implementation of code completion, IntelliSense, in Visual Studio in 1996. These types of tools work by suggesting functions and methods that partially match something you've started typing, as shown in Figure 1-6. Traditional code completion functionality doesn't employ GenAI, and its suggestions can often be frustratingly incorrect. However, if you need help with the syntax or spelling (or don't want to type the full names of functions), code completion is useful.

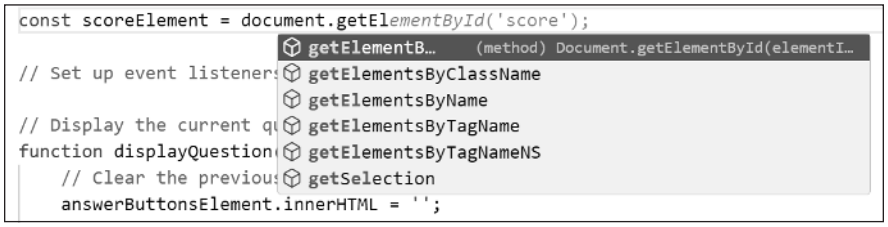


FIGURE 1-6: Code completion is often helpful.

Microsoft Corporation

Generative AI takes code completion to the next level by offering suggestions based on its training. When integrated into your IDE, tools such as GitHub Copilot or Amazon CodeWhisperer can suggest entire statements or functions, rather than just single function calls.

GenAI models trained on large datasets of code can offer multiple suggestions based on what other programmers have written, libraries, classes, and functions

you've imported into the current file, and even other files that are open in your IDE or in your code repository.

Figure 1-7 shows a suggestion from Copilot based on the fact that I named my file validatePhoneNumber.js.

FIGURE 1-7: Copilot's suggested phone number validation function.

Microsoft Corporation

WARNING

Unfortunately, in this instance, the suggested function is worse than useless to most people because it returns `true` only when provided with a phone number starting with 98, which is the country code for Iran.

However, more context can coax the model into returning a better response. When I provided a comment describing what I was trying to do, the model returned a much better suggestion, as shown in Figure 1-8. While not perfect and far too wordy, at least this suggestion gets me closer to what I'm looking for.

In Part 2, you learn much more about how to use GenAI tools to write, format, translate, and optimize your code.

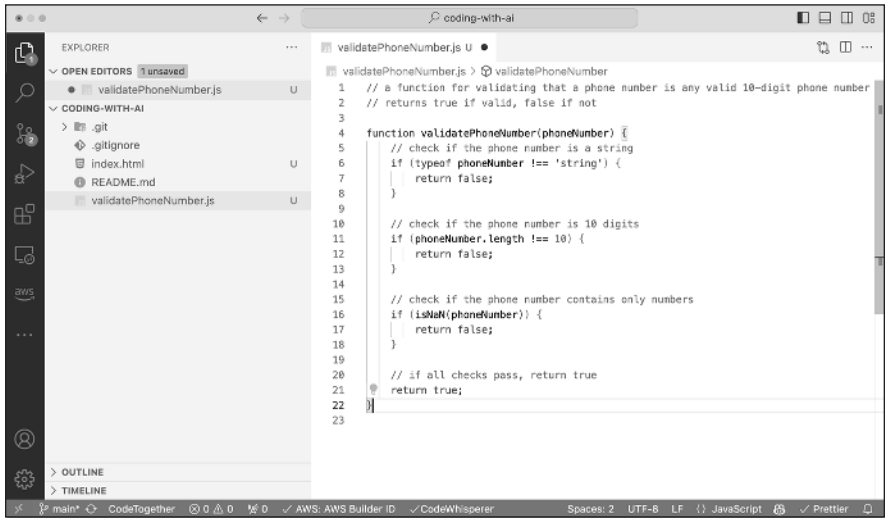


FIGURE 1-8: GenAI models do better when given context.

Microsoft Corporation

Adapting to new syntax

Like code hint features in IDEs, GenAI can help you learn or remember the syntax for functions or methods. For example, Github Copilot will give you hints as you type; you can ignore these by continuing to type or accept them by pressing tab.

For example, I have trouble remembering the order of the arguments to pass to the `Array.reduce` method. Figure 1-9 shows a Copilot hint for starting the method.

```
1   export function calculateTotal(itemsInCart) {
2     return itemsInCart.reduce((total, item) => {
3   }
```

FIGURE 1-9: Getting syntax support from Copilot.

You can also use GenAI chat interfaces to learn about new syntax. However, keep in mind that the model you're using may have been trained on data that predates the existence of the function or method you're seeking help with. For example, GPT-3's training data cutoff date is September 2021, so the model has no knowledge of events or technologies developed after that date. Figure 1-10 shows what happened when I asked GPT-3 about a feature of JavaScript introduced in 2022.

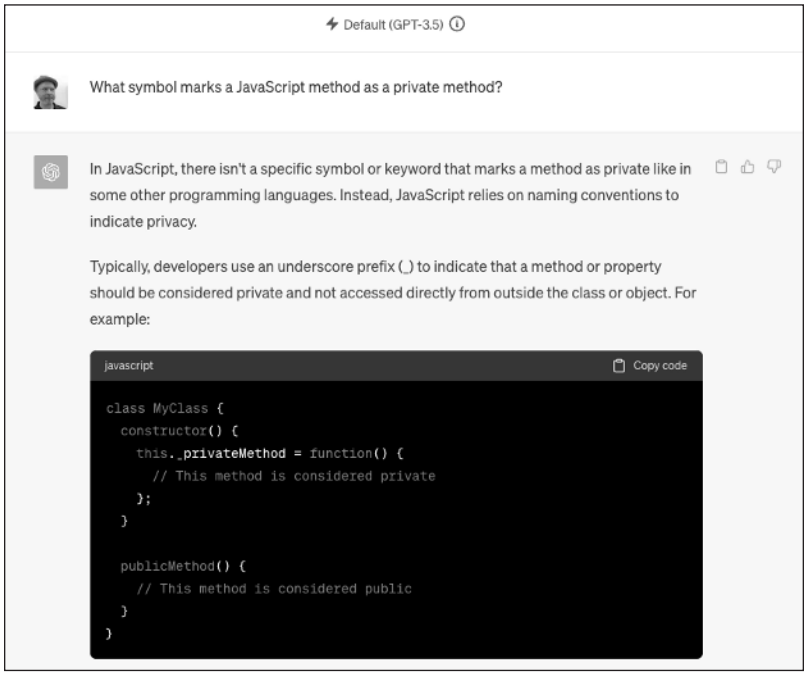
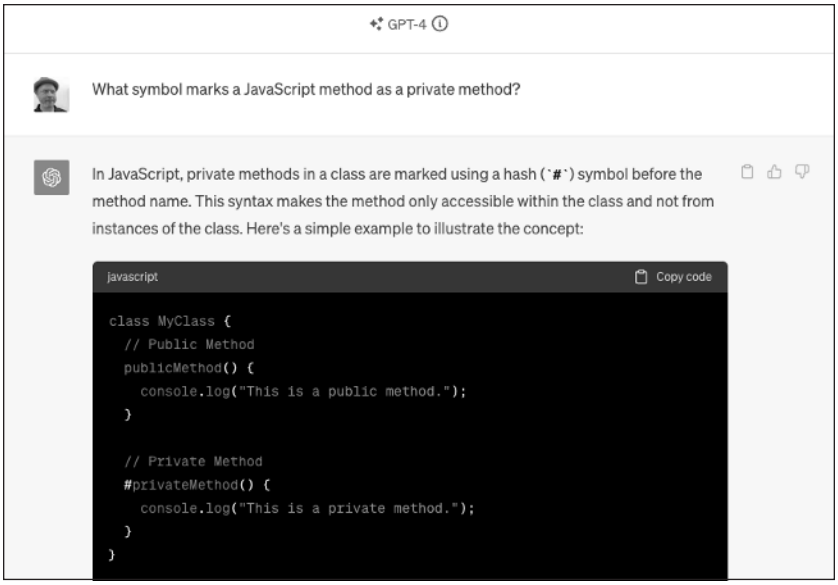


FIGURE 1-10: GPT-3 doesn't know about recent additions to JavaScript.

ChatGPT

GPT-4, on the other hand, has a more recent training data cutoff. Also, ChatGPT has the ability to search the web while using GPT-4. Figure 1-11 shows ChatGPT's response to the same question but with the model set to GPT-4.



ChatGPT

FIGURE 1-11: GPT-4 generates a correct answer when asked about new syntax.

WARNING

Also, remember that, in response to a query, a GenAI model may just invent something that sounds plausible but is incorrect. ChatGPT's tendency to make up facts and people is legendary. However, in cases in which something is as rule-based as a computer language, it usually gets the facts right. As a rule, have a certain amount of distrust of any AI-generated code. Always verify.

Figure 1-12 shows ChatGPT 3.5's response to my question about a non-existent JavaScript method.

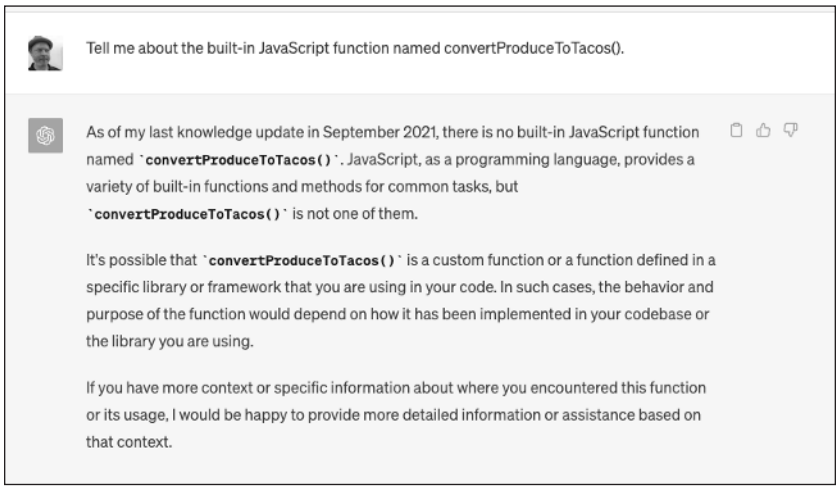


FIGURE 1-12: ChatGPT is usually correct about programming language syntax basics.

ChatGPT

Linting with AI

Linters are tools that flag programming errors, bugs, and style issues. The technical name for the job that linters do is static code analysis. The *static* part of *static code analysis* refers to the fact that these tools check the code without compiling or running it. Using a linter can help you improve the quality of your code.

Since most GenAI tools are (at the time of this writing) incapable of compiling and running the code you write, anytime you prompt a machine learning model to look for errors or bad style in your code, you're using it as a linter.

Detecting bad code with static code analysis

To use an AI chatbot as a linter, you can prompt the model with your code and ask it what's wrong with it. Since GenAI models have been trained on a large quantity of working code, they're generally pretty good at finding typos, inconsistencies, and code that doesn't look right.

Simply write something like "What's wrong with this code?" and then paste in the code that's not working. Figure 1-13 shows Google Gemini's response to my question about a function with several typos and examples of bad coding style.

FIGURE 1-13: Using Bard as a linter.

Microsoft Corporation

Integrating AI with static code analysis

Because programming languages have strict rules, linters don't necessarily need to use AI to detect bad code. However, linting tools that make use of AI can provide functionality that's not possible with standard code linters, such as

- Detailed natural language explanations of what's wrong with your code
- Defining new rules using natural language
- Fixing problematic code or refactoring problematic code or both

Many linters that aren't AI-enhanced can automatically fix certain kinds of problems with your code, and defining new rules generally isn't difficult. The potential for providing detailed descriptions as well as improving your code is promising.

Several tools add AI to existing linters. For example, eslint-ai (available at `https://github.com/iamando/eslint-ai`) is an open-source project that uses GPT-3 to enhance the errors returned by the most popular JavaScript linter, ESLint.

Using eslint-ai requires you to have an account and an API key from OpenAI, and using the tool may result in OpenAI charges. However, GitHub Copilot and other tools include features for cleaning, fixing, and improving your code as part of their standard subscriptions.

Using AI as a Tutor

The question of whether generative AI should be used in education is hotly debated. On the one hand, AI chatbots and AI-enhanced search engines can often provide customized and accurate answers to questions that traditional search engines can't. On the other hand, it may be tempting for a new coder to rely on code generated by AI rather than on gaining experience through struggling with coding for endless hours, which is the traditional way people learn to program (or to write, or anything else for that matter). In this section, I touch on some of the pros and cons of using AI to learn to code.

Studying AI's potential in education

AI can be a useful tool for someone who is learning to code. Just as search engines, online tutorials, and coding books are used today by both new and experienced programmers, AI chatbots and coding assistants will soon be seen as normal and essential tools.

Whether you're learning from older technologies (such as books or a human instructor) or the latest GenAI model, there's no substitute for gaining experience through writing code or from interacting with more experienced programmers.

TIP

When using an AI chatbot to learn to code, ask the right questions and be skeptical of its answers.

Avoiding potential pitfalls

GenAI models and the chatbots that make use of them don't know how to code. All they do is crunch the numbers, based on their training, and tell you the next most likely word. Even with this seemingly simple functionality, large language models such as GPT-4 are often surprisingly accurate and human-sounding.

WARNING

Although efforts are underway to make GenAI models properly express themselves when they have doubts about their answers, today's models are supremely confident in their answers, even when what they say is completely wrong. Never fully trust a GenAI model. You should always test and verify any code output you get, especially before using it in a production environment.

Pairing Up with AI

Pair programming is a software development technique in which two programmers team up at one computer. In pair programming, one person acts as the driver and handles all the typing, while the other acts as the navigator. Ideally, both programmers are equally skilled and switch roles between navigator and driver as needed to take advantage of each person's strengths. However, pair programming also works well when one of the programmers is more experienced (known as expert-novice) or when both programmers are inexperienced (novice-novice).

Pair programming helps team members share knowledge and learn to work together, and it leads to fewer mistakes and better code.

Overview of pair programming styles

Depending on the skill levels of the programmers, several different variations of pair programming might be used:

- **Driver-navigator:** This style of pair programming is the most common. In driver-navigator, the driver handles the typing while the navigator looks at the big picture and keeps an eye out for mistakes being made by the driver.
- **Backseat navigator:** In this style, the driver still does the typing, and the navigator takes a more active role and dictates instructions, such as when to create a file or method or what to name a variable. This style works best when the navigator is a more experienced programmer.
- **Tour guide:** In the tour guide style, the driver is the expert programmer. They handle the typing and explain to the navigator at every step what they're doing and why.
- **Ping-pong:** The ping-pong style is designed for test-driven development as a pair. The first person writes a piece of code designed to verify that a feature works as expected (a *test*). The second programmer writes the code to make the test pass. Then the second programmer writes a new test, and the first programmer writes the code to make it work. This style usually requires two expert developers.

Understanding the pros and cons of pair programming with AI

In AI pair programming, you're the navigator who sets the direction and does the strategic thinking. You communicate the project's goal to the AI through comments and code that you write. As you type, the AI navigator suggests snippets

and code blocks. With each suggestion, you have to decide whether to accept the suggestion, write your own solution, or ask your AI assistant to try again.

Following are some of the benefits of pair programming with an AI partner:

- You (the coder) can spend less time looking up syntax and typing repetitive or boilerplate code.
- The AI assistant is available whenever you are.
- The AI assistant is fast.
- The GenAI model behind the assistant is trained on many different programming languages and programming styles, potentially giving you access to solutions you might not have otherwise considered.

The cons of pair programming with AI may include the following:

- Team members, each working individually with an AI partner, don't get the knowledge-sharing benefits of traditional pair programming.
- AI-suggested code may not be accurate or up-to-date with the latest syntax or coding styles.
- AI-suggested code may contain security flaws or other types of issues that a human coding partner would easily spot.

WARNING

Pair programming with AI works best for coders who know their language and have experience writing code without the use of AI. As you're coding, remember that your partner (the GenAI model) speaks confidently but doesn't know anything about programming.

AI pair programming session

In this section, you work with an AI pair programmer to develop an interactive web-based trivia game. For this exercise, you need access to GitHub Copilot.

Installing Copilot

If the Copilot extension isn't installed in your code editor, follow these steps to install it and sign up for a Copilot free trial:

1. **Open Visual Studio Code.**

   If Visual Studio Code isn't installed, you can download it at `https://code.visualstudio.com`.

2. **Click the extensions icon in the left sidebar of Visual Studio Code and search for Copilot, as shown in Figure 1-14.**

3. **Install the Copilot extension.**

   Note that the Copilot Chat extension is installed automatically when you install Copilot.

4. **In your browser, go to `https://github.com` and sign in.**

   If you don't have an account, create one and then sign in. To use Copilot, you need a GitHub account.

5. **In the window that appears in Visual Studio Code after you installed Copilot, click Sign In to GitHub.**

   If the window isn't open, click the Copilot icon in the lower-right corner of Visual Studio Code.

6. **Walk through the dialog boxes that appear to give Visual Studio Code access to your GitHub account.**

   When you've linked GitHub and Visual Studio Code, Copilot displays a message saying that you don't have access to Copilot.

7. **Click the link to go to GitHub and sign up for a 30-day free trial of Copilot.**

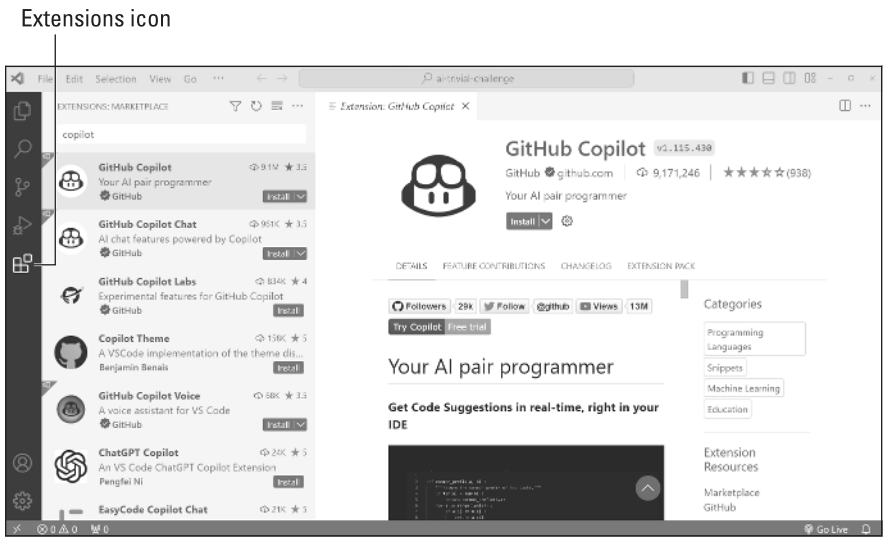


FIGURE 1-14: Searching for the Copilot extension.

Microsoft Corporation

TIP

You must have a Copilot subscription to use Copilot. A free trial is available at https://github.com/features/copilot#pricing. Educators and students have free access to Copilot through GitHub Global Campus at https://education.github.com/.

WARNING

Signing up for a Copilot trial requires you to enter payment information, and the trial converts to a paid subscription automatically after 30 days unless you cancel it.

After you've signed up with Copilot, the Copilot icon in the lower-right corner of Visual Studio Code is active. If you click the icon, several options appear at the top of VS Code, including Deactivate Copilot, which turns off suggestions from the model until you click the icon again and reactivate it.

Congratulations! You've signed up with Copilot and are ready to create a trivia game.

Using prompting to create a trivia game

In this section, you use Copilot to help you build a trivia game that displays one multiple-choice question at a time and then randomly picks a new question when the user selects an answer. Think about the structure of the application you want to build. What other details can you imagine will be important to make sure that Copilot offers usable selections?

In the following steps, you see how the exercise went for me. Your experience will be different:

1. **Create a file named index.html and type an HTML comment.**

   Here's the comment I wrote:

   ```
   <!-- Homepage for a trivia game about machine learning -->
   ```

   In response to this prompt, Copilot suggests a DOCTYPE declaration for an HTML5 file, which is correct.

2. **Press the tab key to accept Copilot's suggestion and then press Enter to go to the next line and evaluate the next suggestion.**

3. **Continue accepting Copilot's suggestions until it provides one you don't want.**

   Copilot suggested a basic HTML template containing a form that submits to a PHP script. This isn't what I envisioned, so I need to provide additional information for my initial prompt. I revised my prompt as follows:

```
<!--
Homepage for trivia game about Machine Learning.
This game will display one multiple-choice question at
a time, using JavaScript that gets data
from data/trivia.js.
Possible answers will be displayed with radio buttons
to their left. When a radio button is clicked, the
user will be shown a new random question.
-->
```

With the additional context, Copilot was able to create something that much more closely resembling what I envisioned:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Machine Learning Trivia</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Machine Learning Trivia</h1>
        </header>
        <main>
            <div id="question"></div>
            <div id="answers"></div>
            <div id="score"></div>
        </main>
        <footer>
            <p>&copy; Chris Minnick</p>
        </footer>
    </div>
    <script src="js/trivia.js"></script>
</html>
```

4. **Add `type="module"` to the `<script>` tag to tell Copilot that you'll be using an `import` statement inside the trivia.js file.**
5. **Create three folders named js, css, and data.**
6. **Inside the data folder, create a file named triviaQuestions.js.**

7. **Write a comment at the beginning of triviaQuestions.js that describes what the file should contain.**

   This is what I wrote:

```
// A module containing an array of 10 objects
// containing quiz questions and answers.
```

8. **If your prompt, like mine, didn't generate the content you expected, do the following:**

   a. *Click the Chat icon in the left panel of Visual Studio code to open the Copilot Chat interface.*

   You can also use ChatGPT for this step.

   b. *In the Chat interface, enter the following prompt:*

```
Give me a JavaScript array containing 10 multiple-choice
  trivia questions about machine learning.
```

   In response to this prompt, Copilot Chat gave me a properly formatted and appropriate array of questions and answers. The first few questions follow:

```
const triviaQuestions = [
  {
    question: "What is the name of the algorithm that is
   commonly used for supervised learning?",
    choices: ["Decision Tree", "K-Means", "Naive Bayes",
   "Random Forest"],
    answer: "Decision Tree"
  },
  {
    question: "What is the name of the algorithm that is
   commonly used for unsupervised learning?",
    choices: ["K-Means", "Decision Tree", "Naive Bayes",
   "Random Forest"],
    answer: "K-Means"
  },
  {
    question: "What is the name of the algorithm that is
   commonly used for reinforcement learning?",
    choices: ["Q-Learning", "K-Means", "Naive Bayes",
   "Random Forest"],
    answer: "Q-Learning"
  },

...
];
```

9. **Copy the generated array and paste it into your data file.**

10. **You'll be importing the array into your JavaScript file, so add the `export` keyword before `const`, like this:**

```
export const triviaQuestions = [
```

11. **Create a new file in the js folder named trivia.js.**

REMEMBER

Make sure that you keep triviaQuestions.js and index.html open while you're working on trivia.js. Copilot uses files you have open as context for the one you're working on.

12. **Write a comment at the beginning of trivia.js describing what it should do.**

Here's the comment I wrote:

```
/* JavaScript for the Trivia Game.
This script is loaded by the index.html file
and will display questions and possible answers
that users can select from. The game will display
a new random question when the user clicks a radio
button to choose an answer and keep track of the
user's score. */
```

13. **Immediately following the comment, start a JavaScript `import` statement to import the question data.**

Whether Copilot figures out what you're doing and helps you or not, the import statement should look like this:

```
import {triviaQuestions} from '../data/triviaQuestions.js';
```

14. **Press Enter and accept the variables that Copilot suggests.**

Eventually, Copilot will suggest a function.

TIP

Don't accept the suggested function right away. If Copilot isn't making any suggestions, try inserting a blank line. After that, look at the Copilot icon in the lower-right corner. It should start spinning, and after a few seconds you'll see a suggestion for how to start writing the code.

15. **Hover your mouse pointer over the function suggestion to display the Copilot menu, which might list multiple possible suggestions, as shown in Figure 1-15**.

16. **If you like one of the suggestions, accept it. If not, try refining your comment to provide more information about what you want.**

17. **Continue this process of accepting suggestions, writing code, and using comments to provide context until you have something that might work.**

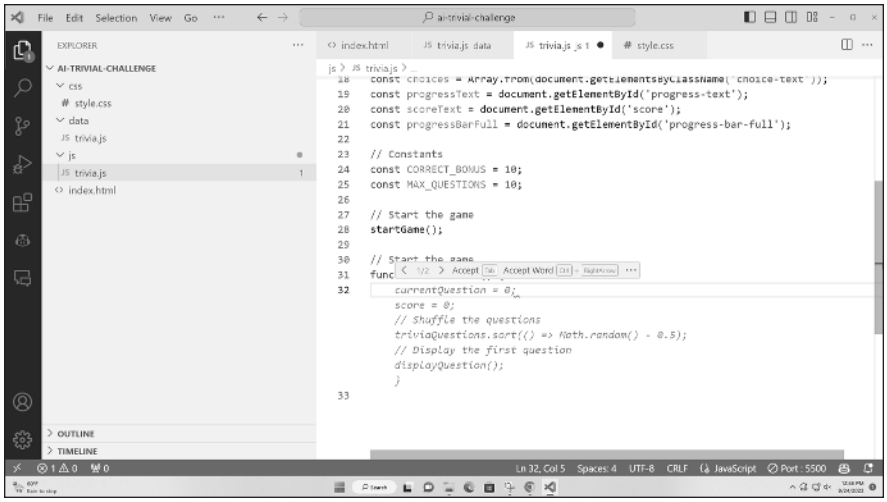


FIGURE 1-15: Viewing the Copilot menu and multiple suggestion options.

Microsoft Corporation

Now it's time to preview your application:

1. **Click the extensions icon to the left of VSCode (labeled in Figure 1-14) and use the Search box to find the Live Server extension.**
2. **Click the Install button under the Live Server extension.**

   This extension opens HTML files using a development server.
3. **In Visual Studio Code's File Explorer, right-click index.html and select Open with Live Server.**

   Your application opens in your default web browser.

Figure 1-16 shows the (rough and unready) game that I created with Copilot's help. The process took me around 20 minutes on my second attempt.

If your application doesn't work, try debugging with the help of Copilot Chat.

Whether or not you ended up with a usable — or even good — application, ask yourself the following questions:

- Was pair programming with AI easier or more difficult than pair programming with another coder?
- Were you surprised (positively or negatively) by the suggestions offered by Copilot?
- How would you change your approach to pair programming with AI if you were to repeat this exercise?

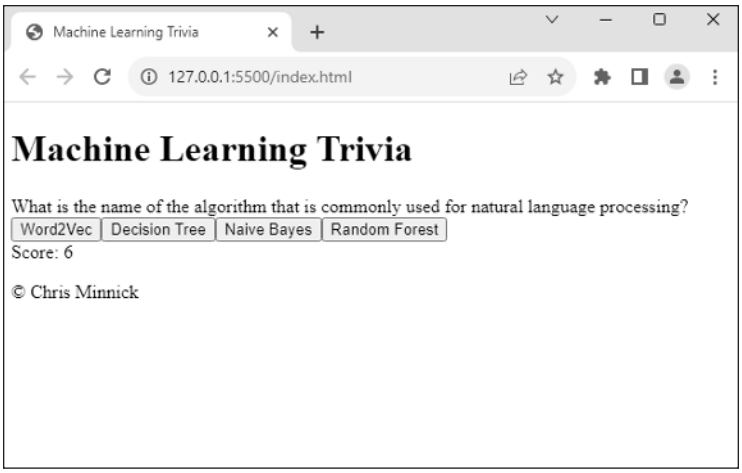


FIGURE 1-16: A somewhat functional trivia game.

In Chapter 3 of this book, you learn much more about how to get better results from GenAI. In the next chapter of this book (Chapter 2), you find out about some of the tools that are available for coding with AI.