
Sich wiederholende Aufgaben automatisieren

Hilfe zur Syntax erhalten

Programme testen

Mit KI schneller lernen

Mit KI paarweise programmieren

Kapitel 1

Wie das Coding von KI profitiert

Wenn Sie Programmierer sind oder vielleicht gerade programmieren lernen, kann Ihnen generative künstliche Intelligenz (auch bekannt als *GenKI*) helfen, produktiver zu arbeiten, weniger Fehler zu machen und neue Fähigkeiten und Sprachen schneller zu erlernen. Dieses erste Kapitel soll Ihnen einen allgemeinen Überblick verschaffen. Dabei werden Sie diverse Tools einsetzen, um einen Vorgeschmack auf die verschiedenen Möglichkeiten zu bekommen. Alle angesprochenen Themen werden in späteren Kapiteln ausführlich beschrieben.

Auch wenn es möglich ist, mithilfe von KI funktionierende Computerprogramme zu erstellen, ohne selbst programmieren zu können, rate ich Ihnen dringend davon ab – vor allem, wenn Sie vorhaben, die von Ihnen erstellten Programme produktiv einzusetzen. Generative KI weiß nicht, wie man programmiert. Wenn Sie auch nicht programmieren können, ist die Wahrscheinlichkeit groß, dass der von Ihnen mit KI erstellte Code schwerwiegende Sicherheitsprobleme, Funktionsprobleme oder Schlimmeres aufweist.

Langweilige Aufgaben abschieben

Die aktuellen generativen KI-Modelle können gut für grundlegende und praktische Aufgaben eingesetzt werden. Beispielsweise können sie zur Generierung von grundlegendem Code verwendet werden, den Programmierer liebevoll als »Boilerplate-Code« bezeichnen.



Der Begriff *Boilerplate* stammt aus dem 19. Jahrhundert, als Stahlplatten als Schablonen für den Bau von Dampfkesseln verwendet wurden.

Boilerplate-Code ist der Code, der in jeder Datei oder in jedem Projekt gleich oder fast gleich ist. Niemand schreibt gern Standardcode, aber es bleibt uns nichts anderes übrig, weil der Code, den wir gern schreiben (also die lustigen Sachen), ohne den Standardcode nicht funktioniert.

Langweilige Aufgaben erkennen

Um Aufgaben zu identifizieren, die mit KI automatisiert werden können, denken Sie an die Aufgaben, die Sie in mehreren Projekten wiederholt ausführen. JavaScript-Entwickler könnten beispielsweise ein Element im Browser-Fenster auswählen, etwa eine Schaltfläche, und ihm einen Ereignis-Listener (oder Event-Listener) zuordnen, um eine bestimmte Aktion auszuführen, wenn ein Benutzer mit diesem Element interagiert. Oder etwas Komplexeres, zum Beispiel Code für eine datenbankgestützte Anwendung schreiben, der Aufgaben zum Erstellen, Lesen, Aktualisieren und Löschen (CRUD, Create, Read, Update, Delete) von Datensätzen in der Datenbank übernimmt.

Ein einfaches Beispiel für automatisierbaren Standardcode für Webentwickler und -designer ist die Grundstruktur einer HTML-Seite. Jede gültige HTML-Seite beginnt mit einer Dokumenttyp-Deklaration, gefolgt von einem HTML-Element, dann einem `head`-Element (einschließlich mehrerer interner Elemente wie `title`- und `meta`-Tags) und schließlich einem `body`-Element.

Die Vorlage von KI schreiben lassen

Sie können den Arbeitsaufwand für die Erstellung einer einfachen HTML-Vorlage auf verschiedene Weise minimieren. Beispielsweise könnten Sie ein Codeschnipsel verwenden oder im Internet nach einem Beispiel suchen. Wenn Sie jedoch eine benutzerdefinierte Boilerplate erstellen möchten, die eine gewisse Struktur aufweist und Ihnen einen guten Ausgangspunkt für das Design einer Website bietet, können Sie einfach Ihre KI fragen. Ich habe ChatGPT zum Beispiel den folgenden Prompt übergeben:

Bitte erstelle eine HTML-Vorlage mit einem Stilelement, das Flexbox verwendet, um ein dreispaltiges Layout zu erstellen, mit einer Navigationsleiste oben und einer Fußzeile.

Hier der Code, den ChatGPT erzeugt hat:

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dreispaltiges Layout</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      height: 100vh;
      margin: 0;
```

```

    }
    header, footer {
        background-color: #333;
        color: white;
        text-align: center;
        padding: 1rem;
    }
    .content {
        flex: 1;
        display: flex;
        padding: 1rem;
    }
    .content > div {
        flex: 1;
        padding: 1rem;
    }
    .content > div:not(:last-child) {
        margin-right: 1rem;
    }
    nav ul {
        list-style-type: none;
        padding: 0;
    }
    nav ul li {
        display: inline;
        margin-right: 1rem;
    }
    nav a {
        text-decoration: none;
        color: white;
    }
    footer {
        margin-top: auto;
    }
</style>
</head>
<body>

<header>
    <nav>
        <ul>
            <li><a href="#">Startseite</a></li>
            <li><a href="#">Über uns</a></li>
            <li><a href="#">Produkte</a></li>
            <li><a href="#">Kontakt</a></li>
        </ul>
    </nav>
</header>

```

```
<div class="content">
  <div>Spalte 1</div>
  <div>Spalte 2</div>
  <div>Spalte 3</div>
</div>

<footer>
  © 2024 Ihr Unternehmen
</footer>

</body>
</html>
```

Das für meine Anfrage (auch als *Prompt* bezeichnet) erzeugte HTML stimmt ziemlich gut mit dem überein, was ich gemacht hätte, wenn ich es von Hand geschrieben hätte. Abbildung 1.1 zeigt die resultierende Webseite, wenn ich das generierte HTML-Markup in einer Datei speichere und sie in einem Browser öffne, ohne sie zu verändern.

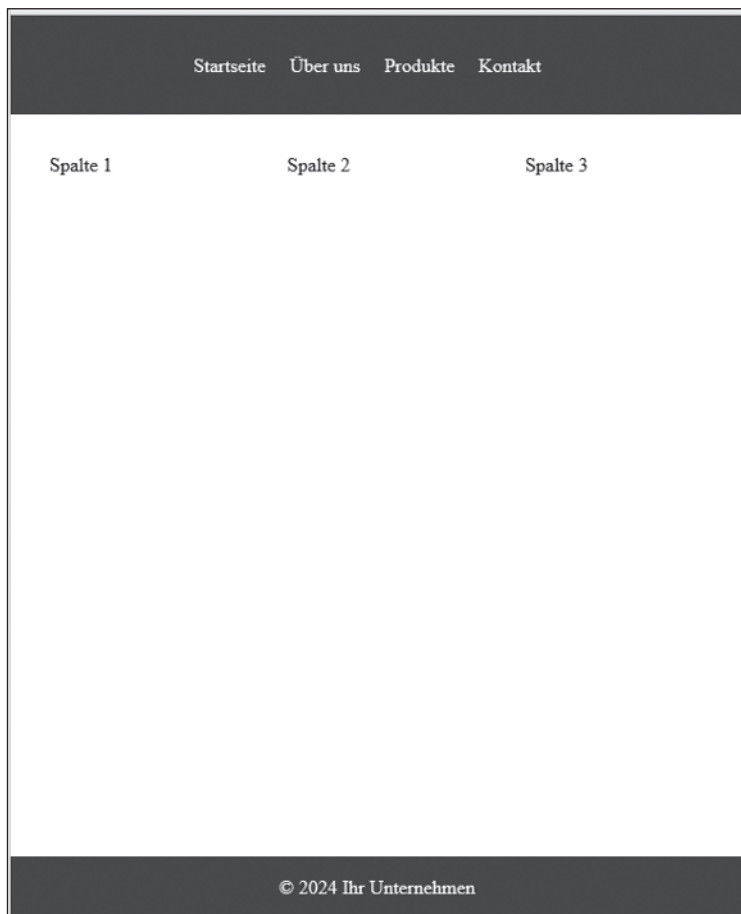


Abbildung 1.1: Eine von ChatGPT generierte HTML-Vorlage © ChatGPT



Sie finden den gesamten in diesem Buch verwendeten Code, einschließlich der in Abbildung 1.1 gezeigten HTML-Vorlage, unter www.wiley-vch.de/ISBN9783527722105.

CRUD mit KI ausarbeiten

Zu den häufigsten Aufgaben in jedem Computerprogramm gehören der Zugriff auf eine Datenquelle und die Entwicklung von Funktionen, die Operationen mit der Datenquelle ausführen. Die grundlegenden Operationen für jede Datenquelle sind das Erstellen eines Datensatzes, das Lesen eines Datensatzes, das Aktualisieren eines Datensatzes und das Löschen eines Datensatzes. Der Sammelbegriff für den Code, der diese Operationen ermöglicht, ist das wunderbar aussagekräftige Akronym CRUD (für Create (Erstellen), Read (Lesen), Update (Aktualisieren), Delete (Löschen) – und außerdem heißt »crud« übersetzt so viel wie »Mist«). Die meisten Menschen schreiben nicht gern CRUD.

In diesem Abschnitt verwenden Sie generative KI, um den Arbeitsaufwand für die Erstellung von CRUD zu verringern. Um mit dieser Übung beginnen zu können, benötigen Sie Zugang zu einer Schnittstelle für das Chatten mit einem generativen KI-Modell wie ChatGPT, Google Gemini oder Microsoft Copilot. Wenn Sie noch kein Konto bei einem dieser Dienste haben, folgen Sie den Anweisungen in Kapitel 4, um sich bei ChatGPT anzumelden.

Wenn Sie dann ein Chat-Fenster vor sich haben, beginnen Sie mit einer kurzen Anfrage an das generative Modell, um zu beobachten, was es tun wird. Wenn der Dienst, den Sie verwenden, Ihre Anfrage nicht erfüllen will oder kann, haben Sie nicht viel Zeit mit der Ausarbeitung einer detaillierten Anfrage verschwendet. Als ich zum Beispiel zu Beginn meiner Arbeit Bing bat, CRUD zu generieren, begann es damit, sagte mir dann aber schnell, ich solle etwas anderes fragen. Mittlerweile ist Bing zu Copilot geworden und hat jetzt Antworten, wie in Abbildung 1.2 gezeigt.

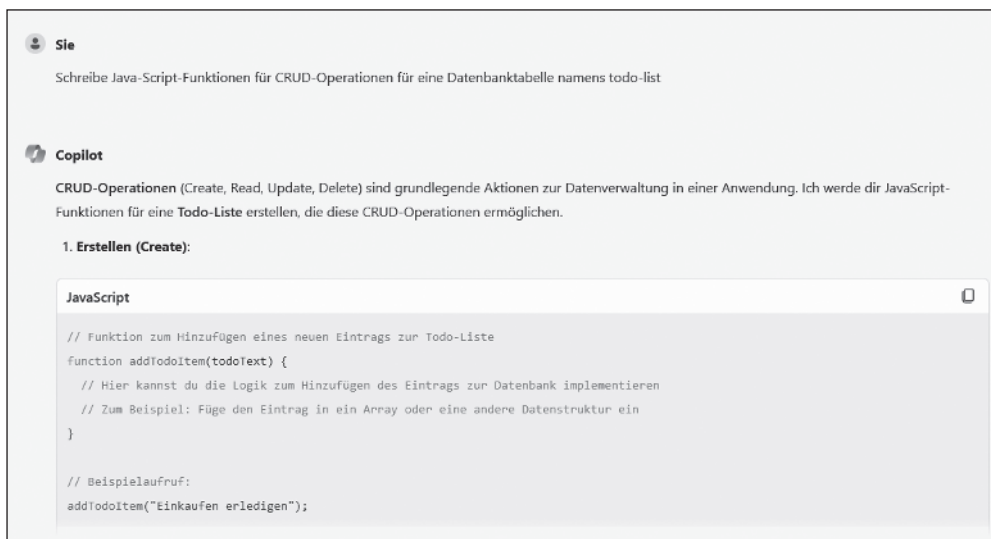


Abbildung 1.2: Hier eine Antwort auf meinen Prompt in Copilot © Microsoft Corporation

Jetzt stellen Sie fest, ob die Antwort vielversprechend ist. Wenn Sie keine Antwort erhalten haben, versuchen Sie es mit einem anderen generativen KI-Modell.

Hier der Anfang der Antwort, die ich auf meinen anfänglichen Prompt (»Schreibe JavaScript-Funktionen für die Durchführung von CRUD-Operationen für eine Datenbanktabelle namens todo-list«) von ChatGPT unter Verwendung des GPT-3.5-Modells erhielt:

```
// Simulierte In-Memory-Datenbank für die To-do-Liste
const todoListDB = [];
// Funktion zum Erstellen eines neuen Code-Elements
function createTodoItem(title, description) {
  const newItem = {
    id: todoListDB.length + 1,
    title,
    description,
    completed: false,
  };
  todoListDB.push(newItem);
  return newItem;
}
```

Die Antwort ist angemessen, aber da ich nicht genügend Informationen über meine gewünschte Anwendung angegeben habe, hat ChatGPT ein Array verwendet, als ob es sich um eine Datenbank handeln würde, was hier wahrscheinlich nicht der Fall ist.

Wenn Sie eine vielversprechende Antwort auf Ihren ersten Prompt erhalten, stellen Sie dem Modell zusätzliche Details bereit. Mein nächster Schritt wäre zum Beispiel, ChatGPT mitzuteilen, dass es sich um eine Node.js-Anwendung handelt, die mit einer MongoDB-Datenbank arbeiten wird und Mongoose verwenden soll.



Der Prozess der Verfeinerung Ihrer Anweisungen an ein GenKI-Modell und die Bereitstellung von Kontext und Beispielen wird *Prompt Engineering* oder *Prompt-Entwicklung* genannt. Kapitel 4 behandelt den Prozess der Prompt-Entwicklung im Detail. Abbildung 1.3 zeigt meinen zweiten Prompt.



Abbildung 1.3: Fügen Sie Kontext hinzu, um eine bessere Antwort zu erhalten. © Microsoft Corporation

Diesmal antwortete ChatGPT mit einer vollständigen Node.js-Anwendung, die sich mit einer lokalen Datenbank verbindet und CRUD-Operationen durchführen kann. Listing 1.1 zeigt den neu generierten Code.

```
const mongoose = require('mongoose');
// Verbindung zu MongoDB
mongoose.connect('mongodb://localhost/todo-list', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
// Definition eines TodoItem-Schemas
const todoItemSchema = new mongoose.Schema({
  title: String,
  description: String,
  completed: Boolean,
});
// Erstellen eines TodoItem-Modells
const TodoItem = mongoose.model('TodoItem', todoItemSchema);
// Funktion zum Erstellen eines neuen To-do-Eintrags
async function createTodoItem(title, description) {
  const newItem = new TodoItem({
    title,
    description,
    completed: false,
  });
  try {
    await newItem.save();
    return newItem;
  } catch (error) {
    throw error;
  }
}
// Funktion zum Lesen aller To-do-Einträge
async function getAllTodoItems() {
  try {
    const todoItems = await TodoItem.find();
    return todoItems;
  } catch (error) {
    throw error;
  }
}
// Funktion zum Lesen eines spezifischen To-do-Eintrags nach ID
async function getTodoItemById(id) {
  try {
    const todoItem = await TodoItem.findById(id);
    return todoItem || null;
  } catch (error) {
    throw error;
  }
}
```

```

    }
  }
  // Funktion zum Aktualisieren eines To-do-Eintrags nach ID
  async function updateTodoItem(id, updatedData) {
    try {
      const updatedItem = await TodoItem.findByIdAndUpdate(id, updatedData,
                                                              { new: true });

      return updatedItem;
    } catch (error) {
      throw error;
    }
  }
}
// Funktion zum Löschen eines To-do-Eintrags nach ID
async function deleteTodoItem(id) {
  try {
    const deletedItem = await TodoItem.findByIdAndRemove(id);
    return deletedItem || null;
  } catch (error) {
    throw error;
  }
}
}

```

Listing 1.1: ChatGPTs zweiter Versuch

Das ist gar nicht so schlecht. Mir gefällt, dass ChatGPT (größtenteils) die neueste JavaScript-Syntax verwendet, mögliche Fehler richtig behandelt und gute Kommentare geschrieben hat. Ich habe von Menschen geschriebenen Code gesehen, der viel schlechter war.

Der eigentliche Test ist jedoch, ob der Code läuft. Nachdem ChatGPT die CRUD-Funktionen generiert hatte, lieferte es einige Beispiele, wie sie verwendet werden können. Diese Beispiele sind in Listing 1.2 dargestellt.

```

// Beispiel für die Verwendung:
(async () => {
  try {
    const newItem = await createTodoItem(
      "Lebensmittel einkaufen", "Milch, Eier und Brot");
    console.log("Neuer Eintrag erstellt", newItem);
    const allItems = await getAllTodoItems();
    console.log("Alle Einträge:", allItems);
    const itemToUpdate = await getTodoItemById(newItem._id);
    if (itemToUpdate) {
      const updatedItem = await updateTodoItem(itemToUpdate._id, {
        completed: true });
      console.log("Eintrag aktualisiert:", updatedItem);
    }
    const deletedItem = await deleteTodoItem(newItem._id);
    console.log("Eintrag gelöscht:", deletedItem);
  }
}
)

```



```

    } catch (error) {
      console.error("Error:", error);
    } finally {
      mongoose.disconnect();
    }
  })());

```

Listing 1.2: Beispiele für die Verwendung des Codes von ChatGPT

Wenn Node.js und MongoDB auf Ihrem Entwicklungsrechner installiert sind, können Sie diesen Code ausprobieren, indem Sie die Listings 1.1 und 1.2 in eine Datei kopieren und diese mit der Erweiterung `.js` speichern.

Bevor Sie die Anwendung ausführen, müssen Sie das Verzeichnis, das die `.js`-Datei enthält, als Node-Paket initialisieren, indem Sie Folgendes in ein Terminalfenster eingeben:

```
npm init -y
```

Dann installieren Sie Mongoose, indem Sie Folgendes eingeben:

```
npm install mongoose
```

Führen Sie dann das Programm aus, indem Sie `node` gefolgt vom Dateinamen eingeben, etwa so:

```
node listing0102.js
```

Abbildung 1.4 zeigt, was passierte, als ich dieses Programm ausführte.



```

nick@chris-mac chapter01 % node listing0103.js
Created item: {
  title: 'Supermarkt',
  description: 'Milch, Eier und Butter',
  completed: false,
  _id: new ObjectId("650c4885564f597926a10ac0"),
  __v: 0
}
All items: [
  {
    _id: new ObjectId("650c4885564f597926a10ac0"),
    title: 'Supermarkt',
    description: 'Milch, Eier und Butter',
    completed: false,
    __v: 0
  }
]
Updated item: {
  _id: new ObjectId("650c4885564f597926a10ac0"),
  title: 'Supermarkt',
  description: 'Milch, Eier und Butter',
  completed: false,
  __v: 0
}

```

Abbildung 1.4: Ausführen meiner Node.js-Anwendung

Um zu überprüfen, ob der Code der KI funktioniert, habe ich den Code, der den erstellten Datensatz löscht, auskommentiert, die Node.js-Anwendung erneut ausgeführt, dann die Mongo-Shell gestartet und mir den Inhalt der Collection `todo-list` angesehen, wie in Abbildung 1.5 dargestellt.

```
test> use todo-list
switched to db todo-list
todo-list> show collections
todoitems
todo-list> db.todoitems.find()
[
  {
    _id: ObjectId("650c49f3acefaa817b939047"),
    title: 'Supermarkt',
    description: 'Milch, Eier und Butter',
    completed: true,
    __v: 0
  }
]
todo-list> █
```

Abbildung 1.5: Anzeige des Inhalts der Collection in MongoDB

Hilfe bei der Syntax

Ein großer Teil der Arbeit beim Coding besteht darin, sich Regeln zu merken oder nachzuschlagen, die die Struktur einer Programmiersprache, auch *Syntax* genannt, definieren. Jede Sprache oder Code-Bibliothek hat ihre eigene Art, Dinge zu tun. Sobald Sie die Grundlagen der Funktionsweise einer Programmiersprache kennen (zum Beispiel das Erstellen einer Funktion, die Verwendung grundlegender Operatoren und das Schreiben von Schleifen), müssen Sie wissen, welche eingebauten Funktionen in Ihrer Umgebung (sei es ein Browser oder ein mobiles Betriebssystem) verfügbar sind, und welche Parameter und Datentypen sie erwarten. Das ist eine Menge, die man sich merken muss, und kein Programmierer, den ich je getroffen habe, kann sich alles merken, was es über eine Programmiersprache zu wissen gibt, geschweige denn über mehrere Programmiersprachen. Mithilfe der GenKI-Tools können Sie sofort auf das gesammelte Wissen von Millionen von Programmierern zugreifen.

Vielleicht fragen Sie sich jetzt: »Aber ist es ethisch vertretbar, dass sich KI den Code aller Benutzer auf diese Weise zunutze macht?« Dieses Thema wird heftig diskutiert und ist Gegenstand mindestens eines Gerichtsverfahrens. Ich gehe in diesem Buch immer wieder auf rechtliche und ethische Fragen im Zusammenhang mit GenKI ein.

Hören Sie auf, sich triviale Details merken zu wollen

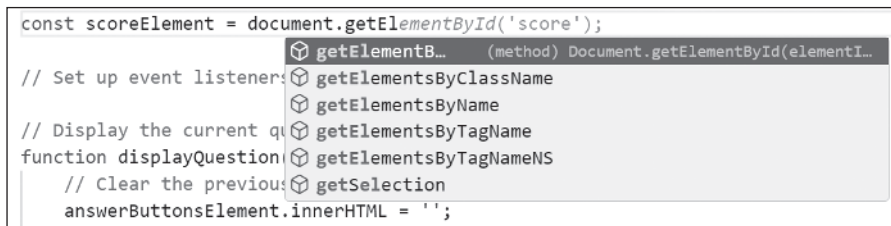
Wenn ich Programmieren unterrichte, stellen mir meine Studierenden oft Fragen zur Syntax und zu Anwendungsprogrammierschnittstellen (APIs) – und nicht dazu, wie etwas funktioniert. Wenn ich eine Frage zur Syntax bekomme, beantworte ich sie, falls ich das, ohne nachzuschlagen, kann, andernfalls lege ich den Studierenden nahe, einfach zu googeln. Mit der Zeit und der Erfahrung wird das Erinnern an die Syntax zum Selbstläufer.



Wenn man Software schreibt, sollte man unbedingt wissen, wie und wo man nach Antworten suchen muss. Und in den meisten Fällen ist der beste Ort dafür eine Suchmaschine. Da Suchmaschinen maschinelles Lernen einsetzen, um die besten Ergebnisse für die Beantwortung von Suchanfragen zu ermitteln, nutzen wir somit seit einiger Zeit KI für die Programmierung.

Hinweise, wie es im Code weitergeht

Eine der ältesten Formen des computergestützten Codings ist die Code-Vervollständigung. Microsoft führte 1996 mit IntelliSense ihre Implementierung davon in Visual Studio ein. Solche Tools arbeiten, indem sie Funktionen und Methoden vorschlagen, die teilweise mit dem übereinstimmen, was Sie gerade eingegeben haben (siehe Abbildung 1.6). Herkömmliche Code-Vervollständigungsverfahren verwenden keine GenKI und ihre Vorschläge können oft frustrierend falsch sein. Wenn Sie jedoch Hilfe bei der Syntax oder der Rechtschreibung benötigen (oder die Namen von Funktionen nicht vollständig eingeben möchten), ist die Code-Vervollständigung nützlich.



```
const scoreElement = document.getElementById('score');

// Set up event listener:
// Display the current question:
function displayQuestion() {
  // Clear the previous answer:
  answerButtonsElement.innerHTML = '';
}
```

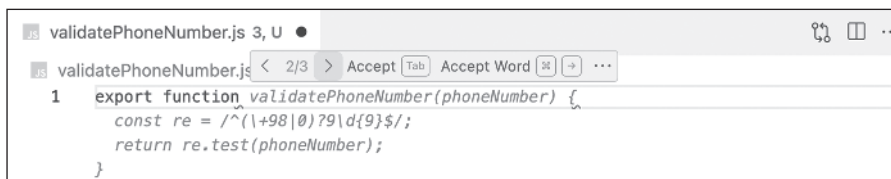
Suggestions shown: `getElementById`, `getElementsByClassName`, `getElementsByName`, `getElementsByTagName`, `getElementsByTagNameNS`, `getSelection`.

Abbildung 1.6: Die Code-Vervollständigung ist oft hilfreich. © Microsoft Corporation

Generative KI bringt die Code-Vervollständigung auf die nächste Stufe, indem sie Vorschläge auf der Grundlage ihres Trainings macht. Wenn sie in Ihre IDE integriert sind, können Tools wie Copilot von GitHub oder CodeWhisperer von Amazon ganze Anweisungen oder Funktionen vorschlagen, statt nur einzelne Funktionsaufrufe.

GenKI-Modelle, die auf großen Code-Datensätzen trainiert wurden, können mehrere Vorschläge machen, die darauf basieren, was andere Programmierer geschrieben haben – Bibliotheken, Klassen und Funktionen, die Sie in die aktuelle Datei importiert haben; und sogar andere Dateien, die in Ihrer IDE oder in Ihrem Code-Repository geöffnet sind.

Abbildung 1.7 zeigt einen Vorschlag von Copilot auf der Grundlage der Tatsache, dass ich meine Datei `validatePhoneNumber.js` genannt habe.



```
validatePhoneNumber.js 3, U
1 export function validatePhoneNumber(phoneNumber) {
  const re = /^(+98|0)?9\d{9}$/;
  return re.test(phoneNumber);
}
```

Abbildung 1.7: Die von Copilot vorgeschlagene Funktion zur Überprüfung von Telefonnummern © Microsoft Corporation



Leider ist die vorgeschlagene Funktion in diesem Fall für die meisten Menschen eher nutzlos, da sie nur bei einer Telefonnummer, die mit 98 beginnt, der Landesvorwahl für den Iran, den richtigen Wert liefert.

Allerdings kann mehr Kontext das Modell dazu bringen, eine bessere Antwort zu geben. Als ich einen Kommentar abgab, in dem ich beschrieb, was ich zu tun versuchte, lieferte das Modell einen viel besseren Vorschlag, wie in Abbildung 1.8 gezeigt. Dieser Vorschlag ist zwar nicht perfekt und viel zu ausführlich, aber er bringt mich zumindest näher an das heran, wonach ich suche.

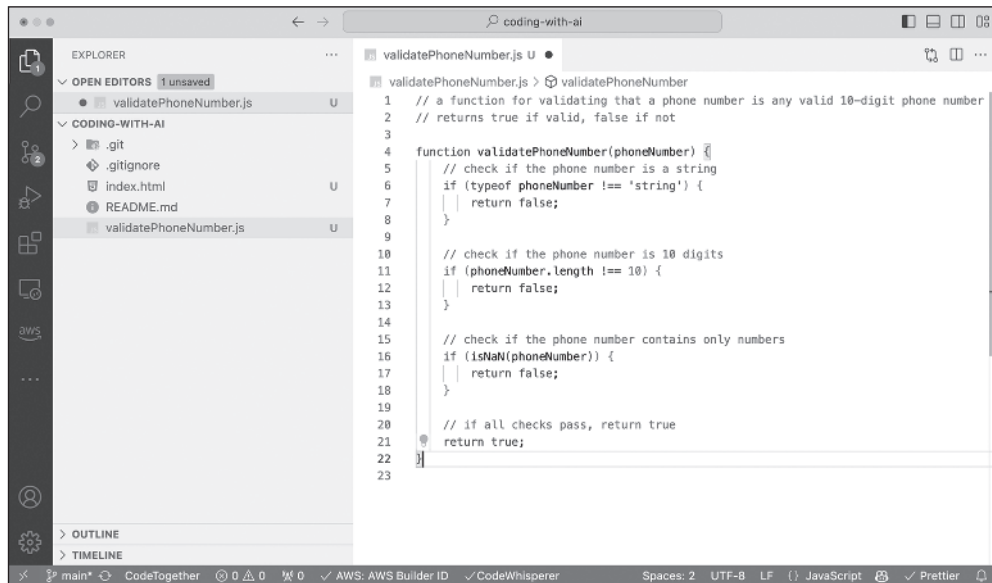


Abbildung 1.8: GenKI-Modelle schneiden besser ab, wenn sie Kontext erhalten. © Microsoft Corporation

In Teil II erfahren Sie mehr darüber, wie Sie GenKI-Tools zum Schreiben, Formatieren, Übersetzen und Optimieren Ihres Codes verwenden können.

Unterstützung bei der Syntax

Ähnlich wie Code-Hinweisfunktionen in IDEs kann GenKI Ihnen helfen, die Syntax für Funktionen oder Methoden zu lernen oder sie sich zu merken. Copilot gibt Ihnen beispielsweise während der Eingabe Hinweise, die Sie entweder ignorieren können, indem Sie mit der Eingabe fortfahren, oder akzeptieren, indem Sie die Tabulatortaste drücken.

Ich habe zum Beispiel Schwierigkeiten, mir die Reihenfolge der Argumente zu merken, die der Methode `Array.reduce` übergeben werden müssen. Abbildung 1.9 zeigt einen Copilot-Hinweis zum Starten der Methode.

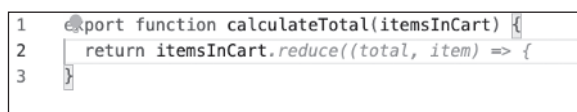


Abbildung 1.9: Abruf der Syntaxunterstützung von Copilot

Sie können auch GenKI-Chat-Oberflächen verwenden, um neue Syntax zu lernen. Denken Sie jedoch daran, dass das von Ihnen verwendete Modell möglicherweise auf Daten trainiert wurde, die es vor der Funktion oder Methode gab, für die Sie Hilfe suchen. Der Stichtag für die Trainingsdaten von GPT-3 ist zum Beispiel September 2021, sodass das Modell keine Kenntnisse über Ereignisse oder Technologien hat, die nach diesem Datum entwickelt wurden. Fragte ich also GPT-3 nach einer Funktion von JavaScript, die 2022 eingeführt wurde, beispielsweise die Kennzeichnung von privaten Methoden durch ein #, teilte es mir mit, dass es keine bestimmte Kennzeichnung gab – weil es sie noch nicht kannte.

GPT-4 hingegen hat einen neueren Trainingsdaten-Stichtag und kann auch das Web durchsuchen. Abbildung 1.10 zeigt die korrekte Antwort von ChatGPT auf die Frage – weil das Modell GPT-4 entsprechend trainiert wurde.

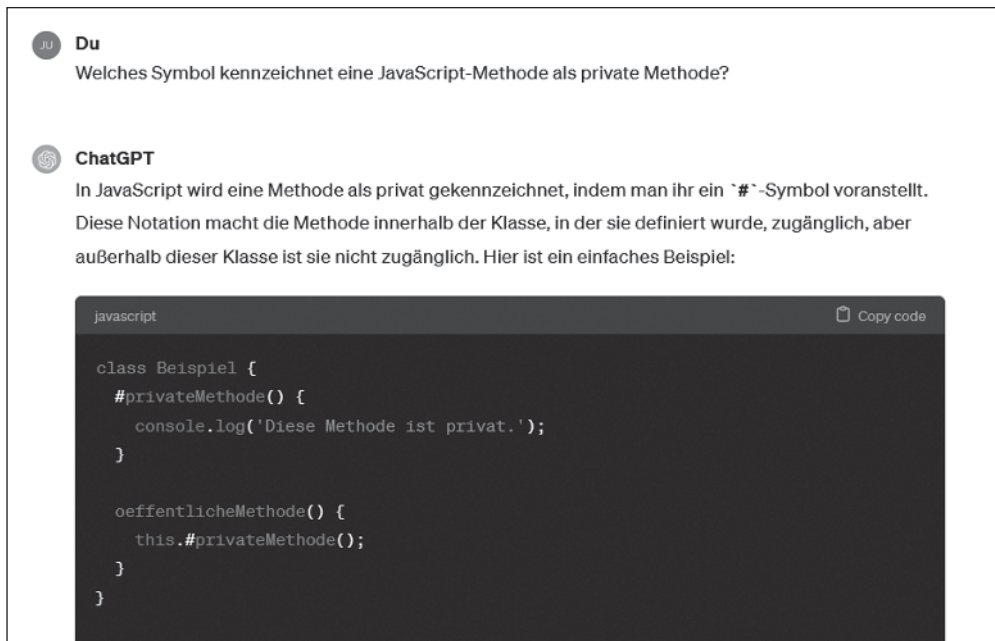


Abbildung 1.10: GPT-4 erzeugt eine korrekte Antwort auf die Frage nach einer neuen Syntax.
© ChatGPT



Denken Sie auch daran, dass ein GenKI-Modell als Antwort auf eine Anfrage etwas erfinden kann, das zwar plausibel klingt, aber falsch ist. Die Neigung von ChatGPT, Fakten und Personen zu erfinden, ist legendär. Man bezeichnet dies auch als »halluzinieren«. Wenn es jedoch um etwas geht, das regelbasiert ist wie eine Computersprache, sind die Fakten meist richtig. Misstrauen Sie grundsätzlich jedem KI-generierten Code. Überprüfen Sie ihn immer.

Abbildung 1.11 zeigt die Antwort von ChatGPT 3.5 auf meine Frage nach einer nicht existierenden JavaScript-Methode.

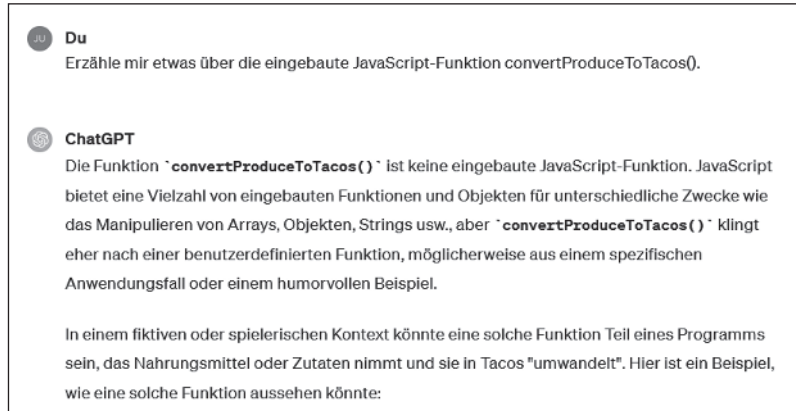


Abbildung 1.11: ChatGPT ist in der Regel korrekt, was die Grundlagen der Syntax von Programmiersprachen angeht. © ChatGPT

Linting mit KI

Linter sind Werkzeuge, die Programmierfehler, Bugs und Stilprobleme aufspüren. Der technische Name für die Arbeit von Lintern ist »statische Codeanalyse«. Der Teil *statisch* im Begriff *statische Codeanalyse* bezieht sich auf die Tatsache, dass diese Tools den Code überprüfen, ohne ihn zu kompilieren oder auszuführen. Die Verwendung eines Linters kann Ihnen helfen, die Qualität Ihres Codes zu verbessern.

Da die meisten GenKI-Tools (zum Zeitpunkt der Drucklegung dieses Buches) nicht in der Lage sind, den von Ihnen geschriebenen Code zu kompilieren und auszuführen, verwenden Sie immer ein maschinelles Lernmodell als Linter, wenn Sie es auffordern, nach Fehlern oder fehlerhaftem Stil in Ihrem Code zu suchen.

Fehlerhaften Code mit statischer Codeanalyse erkennen

Um einen KI-Chatbot als Linter zu verwenden, können Sie dem Modell Ihren Code übergeben und es fragen, was damit nicht stimmt. Da GenKI-Modelle mit einer großen Menge an funktionierendem Code trainiert wurden, sind sie in der Regel ziemlich gut darin, Tippfehler, Ungereimtheiten und Code zu finden, der nicht richtig aussieht.

Schreiben Sie einfach etwas wie »Was stimmt mit diesem Code nicht?« und fügen Sie dann den Code ein, der nicht funktioniert. Abbildung 1.12 zeigt die Antwort von Microsoft Copilot auf meine Frage zu einer Funktion mit mehreren Tippfehlern und Beispielen für schlechten Code.

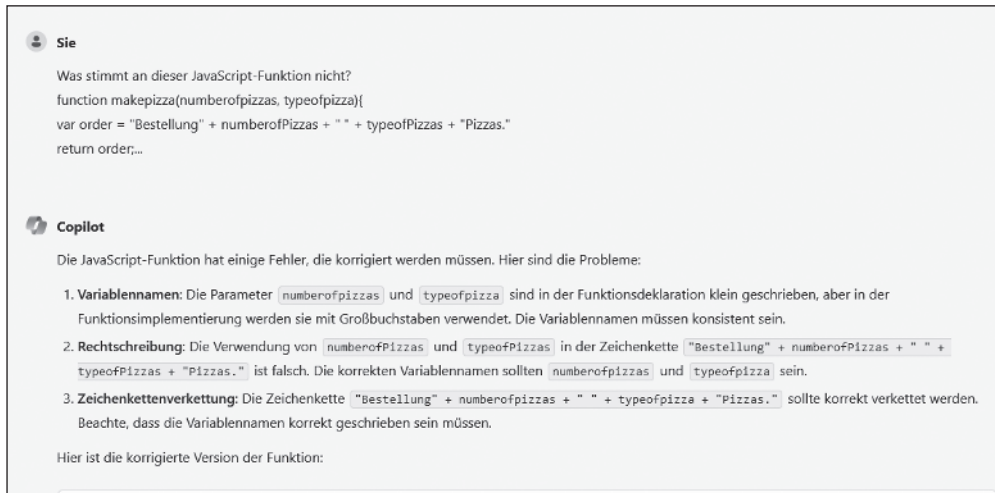


Abbildung 1.12: Verwendung von Microsoft Copilot als Linter © Microsoft Corporation

Integration von KI in die statische Codeanalyse

Da Programmiersprachen strenge Regeln haben, müssen Linter nicht unbedingt KI einsetzen, um schlechten Code zu erkennen. Linting-Tools, die KI nutzen, können jedoch Funktionen bereitstellen, die mit Standard-Code-Lintern nicht möglich sind, wie zum Beispiel:

- ✓ Detaillierte Erklärungen in natürlicher Sprache, was mit Ihrem Code nicht in Ordnung ist
- ✓ Definition neuer Regeln unter Verwendung natürlicher Sprache
- ✓ Korrektur von problematischem Code oder Refactoring von problematischem Code oder beides

Viele nicht KI-gestützte Linter können bestimmte Probleme mit Ihrem Code automatisch beheben, und die Definition neuer Regeln ist im Allgemeinen nicht schwierig. Das Potenzial für die Bereitstellung detaillierter Beschreibungen und die Verbesserung Ihres Codes ist vielversprechend.

Einige Tools fügen vorhandenen Lintern KI hinzu. `eslint-ai` beispielsweise (<https://github.com/iamando/eslint-ai>) ist ein Open-Source-Projekt, das GPT verwendet, um die vom beliebtesten JavaScript-Linter ESLint zurückgegebenen Fehler zu korrigieren.

Für die Verwendung von `eslint-ai` benötigen Sie ein Konto und einen API-Schlüssel von OpenAI, und die Verwendung des Tools kann zu OpenAI-Gebühren führen. GitHub Copilot und andere Tools enthalten jedoch Funktionen zum Bereinigen, Korrigieren und Verbessern Ihres Codes als Teil ihrer Standardabonnements. Kapitel 7 behandelt Linting und Debugging mit KI im Detail.

Der Einsatz von KI als Tutor

Die Frage, ob generative KI in der Ausbildung eingesetzt werden sollte, wird heftig diskutiert. Einerseits können KI-Chatbots und KI-gestützte Suchmaschinen oft maßgeschneiderte und genaue Antworten auf Fragen geben, die herkömmliche Suchmaschinen nicht geben können. Andererseits kann es für einen Programmieranfänger verlockend sein, sich auf den von der KI generierten Code zu verlassen, statt Erfahrungen zu sammeln, indem er sich endlose Stunden lang mit dem Programmieren abmüht, was die traditionelle Art und Weise ist, wie Menschen das Programmieren lernen (oder das Schreiben oder irgendetwas anderes). In diesem Abschnitt gehe ich auf einige der Vor- und Nachteile des Einsatzes von KI beim Programmierenlernen ein.

Das Potenzial von KI beim Lernen

KI kann ein nützliches Werkzeug für jemanden sein, der gerade programmieren lernt. So wie Suchmaschinen, Online-Tutorials und Programmierbücher heute sowohl von neuen als auch von erfahrenen Programmierern genutzt werden, werden KI-Chatbots und Programmierassistenten wahrscheinlich bald als völlig normale und unverzichtbare Tools angesehen.

Egal, ob Sie mit älteren Technologien (wie Büchern oder einem menschlichen Lehrer) oder dem neuesten GenKI-Modell lernen, es gibt keinen Ersatz für das Sammeln von Erfahrungen durch das Schreiben von Code oder die Interaktion mit erfahreneren Programmierern.



Wenn Sie einen KI-Chatbot zum Programmierenlernen einsetzen, stellen Sie die richtigen Fragen und seien Sie skeptisch gegenüber seinen Antworten.

Potenzielle Fallstricke vermeiden

GenKI-Modelle und die Chatbots, die sie nutzen, wissen nicht, wie man programmiert. Alles, was sie tun, ist es, auf der Grundlage ihres Trainings die Zahlen zu berechnen und Ihnen das nächste wahrscheinlichste Wort zu nennen. Selbst mit dieser scheinbar einfachen Funktion sind große Sprachmodelle wie GPT-4 oft erstaunlich genau und klingen menschlich.



Obwohl Anstrengungen unternommen werden, um GenKI-Modelle dazu zu bringen, sich richtig auszudrücken, wenn sie Zweifel an ihren Antworten haben, sind die heutigen Modelle äußerst zuversichtlich in Bezug auf ihre Antworten, selbst wenn das, was sie sagen, völlig falsch ist. Verlassen Sie sich niemals vollständig auf ein GenKI-Modell. Sie sollten jede Codeausgabe, die Sie erhalten, immer testen und verifizieren, insbesondere bevor Sie sie in einer Produktionsumgebung verwenden.

Paar-Programmierung mit KI

Die *Paar-Programmierung* ist eine Technik der Softwareentwicklung, bei der zwei Programmierer gemeinsam an einem Computer arbeiten. Bei der Paar-Programmierung fungiert eine Person als Driver und übernimmt die gesamte Eingabe, während die andere Person

die Aufgabe eines Navigators übernimmt. Im Idealfall sind beide Programmierer gleich gut ausgebildet und wechseln je nach Bedarf die Rolle des Navigators und des Drivers, um die Stärken des jeweils anderen zu nutzen. Die Paar-Programmierung funktioniert jedoch auch gut, wenn einer der Programmierer erfahrener ist (Experte/Anfänger) oder wenn beide Programmierer unerfahren sind (Anfänger/Anfänger).

Paarweises Programmieren hilft den Teammitgliedern, Wissen auszutauschen und zu lernen, zusammenzuarbeiten, und führt zu weniger Fehlern und besserem Code.

Überblick über verschiedene Arten der Paar-Programmierung

Je nach den Fähigkeiten der Programmierer können verschiedene Varianten der Paar-Programmierung eingesetzt werden:

- ✓ **Driver-Navigator:** Diese Art der Paar-Programmierung ist die häufigste. Bei Driver-Navigator kümmert sich der Driver um die Eingabe, während der Navigator das Gesamtbild im Auge behält und auf Fehler des Drivers achtet.
- ✓ **Rücksitz-Navigator:** Bei diesem Stil tippt ebenfalls der Driver, aber der Navigator übernimmt eine aktivere Rolle und diktiert Anweisungen, zum Beispiel wann eine Datei oder Methode erstellt oder wie eine Variable benannt werden soll. Dieser Stil funktioniert am besten, wenn der Navigator ein erfahrener Programmierer ist.
- ✓ **Tour-Guide:** Beim Tour-Guide-Stil ist der Driver der erfahrene Programmierer. Er übernimmt die Eingabe und erklärt dem Navigator bei jedem Schritt, was er tut und warum.
- ✓ **Ping-Pong:** Der Ping-Pong-Stil ist für die testorientierte Entwicklung zu zweit gedacht. Die erste Person schreibt einen Codeabschnitt, um zu überprüfen, ob eine Funktion wie erwartet funktioniert (einen *Test*). Der zweite Programmierer schreibt den Code, um den Test erfolgreich durchzuführen. Dann schreibt der zweite Programmierer einen neuen Test und der erste Programmierer schreibt den Code so, dass er funktioniert. Für diese Vorgehensweise sind in der Regel zwei erfahrene Entwickler erforderlich.

Die Vor- und Nachteile der Paar-Programmierung mit KI verstehen

Bei der KI-Paar-Programmierung sind Sie der Navigator, der die Richtung vorgibt und das strategische Denken übernimmt. Sie vermitteln der KI das Ziel des Projekts durch Kommentare und Code, den Sie schreiben. Während Sie tippen, schlägt der KI-Navigator Codeschnipsel und Codeblöcke vor. Sie müssen bei jedem Vorschlag entscheiden, ob Sie ihn annehmen, Ihre eigene Lösung schreiben oder Ihren KI-Assistenten bitten, es noch einmal zu versuchen.

Hier einige der Vorteile der Paar-Programmierung mit einem KI-Partner:

- ✓ Sie (der Programmierer) müssen weniger Zeit damit verbringen, die Syntax nachzuschlagen und sich wiederholenden oder Boilerplate-Code einzugeben.
- ✓ Der KI-Assistent ist bereit, wann immer Sie es sind.
- ✓ Der KI-Assistent ist schnell.
- ✓ Das GenKI-Modell, das hinter dem Assistenten steht, ist in vielen verschiedenen Programmiersprachen und Programmierstilen geschult, was Ihnen Zugang zu Lösungen verschafft, die Sie sonst vielleicht nicht in Betracht gezogen hätten.

Die Nachteile der Paar-Programmierung mit KI können folgende sein:

- ✓ Teammitglieder, die einzeln mit einem KI-Partner zusammenarbeiten, profitieren nicht von den Vorteilen der traditionellen Paar-Programmierung, die der Wissensaustausch mit sich bringt.
- ✓ Der von der KI vorgeschlagene Code ist möglicherweise nicht genau oder nicht auf dem neuesten Stand der Syntax oder des Coding-Stils.
- ✓ Der von der KI vorgeschlagene Code kann Sicherheitslücken oder andere Arten von Problemen enthalten, die ein menschlicher Programmierpartner leicht erkennen würde.



Die Paar-Programmierung mit KI funktioniert am besten für Programmierer, die ihre Sprache beherrschen und Erfahrung im Schreiben von Code ohne KI haben. Während Sie programmieren, denken Sie daran, dass Ihr Partner (das GenKI-Modell) selbstbewusste Aussagen trifft, aber keine Ahnung vom Programmieren hat.

KI-Paar-Programmierungssitzung

In diesem Abschnitt arbeiten Sie mit einem KI-Paar-Programmierer zusammen, um ein interaktives webbasiertes Quizspiel zu entwickeln. Für diese Übung benötigen Sie Zugang zu GitHub Copilot.

Copilot installieren

Wenn die Copilot-Erweiterung nicht in Ihrem Code-Editor installiert ist, folgen Sie diesen Schritten, um sie zu installieren und sich für eine kostenlose Testversion von Copilot anzumelden:

1. Öffnen Sie Visual Studio Code.

Wenn Visual Studio Code nicht installiert ist, können Sie es unter <https://code.visualstudio.com> herunterladen.

2. Klicken Sie auf das Erweiterungssymbol in der linken Seitenleiste von Visual Studio Code und suchen Sie nach Copilot, wie in Abbildung 1.13 dargestellt.

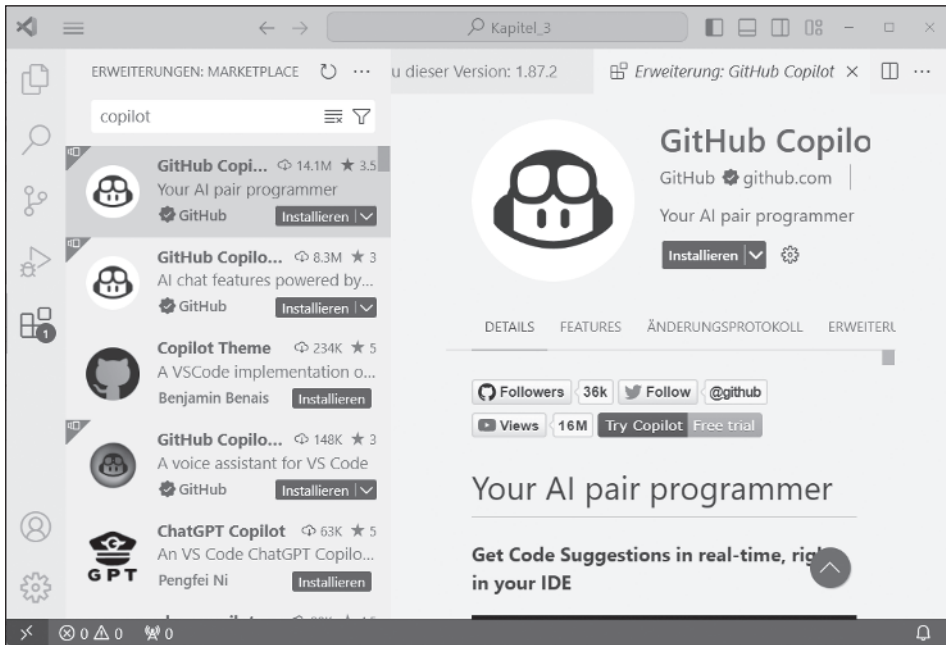


Abbildung 1.13: Suche nach der Copilot-Erweiterung © Microsoft Corporation

3. Installieren Sie die Copilot-Erweiterung.

Beachten Sie, dass die Copilot-Chat-Erweiterung automatisch installiert wird, wenn Sie Copilot installieren.

4. Rufen Sie in Ihrem Browser die Seite <https://github.com> auf und melden Sie sich an.

Wenn Sie noch kein Konto haben, erstellen Sie eines und melden Sie sich dann an. Um Copilot verwenden zu können, benötigen Sie ein GitHub-Konto.

5. Klicken Sie in dem Fenster, das nach der Installation von Copilot in Visual Studio Code angezeigt wird, auf SIGN INTO GITHUB.

Wenn das Fenster nicht geöffnet ist, klicken Sie auf das Copilot-Symbol in der unteren rechten Ecke von Visual Studio Code.

6. Durchlaufen Sie die Dialogfelder, die angezeigt werden, um Visual Studio Code Zugriff auf Ihr GitHub-Konto zu geben.

Wenn Sie GitHub und Visual Studio Code verknüpft haben, zeigt Copilot eine Meldung an, die besagt, dass Sie keinen Zugriff auf Copilot haben.

7. Klicken Sie auf den Link, um zu GitHub zu gehen und sich für eine kostenlose 30-Tage-Testversion von Copilot anzumelden.



Sie müssen ein Copilot-Abonnement haben, um Copilot nutzen zu können. Eine kostenlose Testversion ist unter <https://github.com/features/copilot#pricing> verfügbar. Lehrkräfte und Studierende haben über GitHub Global Campus unter <https://education.github.com/> kostenlosen Zugang zu Copilot.



Wenn Sie sich für eine Copilot-Testversion anmelden, müssen Sie Zahlungsinformationen eingeben. Die Testversion wird nach 30 Tagen automatisch in ein kostenpflichtiges Abonnement umgewandelt, sofern Sie sie nicht kündigen.

Sobald Sie sich bei Copilot angemeldet haben, ist das Copilot-Symbol in der unteren rechten Ecke von Visual Studio Code aktiv. Wenn Sie auf das Symbol klicken, werden oben in VS Code mehrere Optionen angezeigt, darunter auch das Deaktivieren von Copilot, wodurch die Vorschläge des Modells ausgeschaltet werden, bis Sie erneut auf das Symbol klicken und es wieder aktivieren.

Herzlichen Glückwunsch! Sie haben sich bei Copilot angemeldet und sind bereit, ein Quizspiel zu erstellen.

Prompting zur Erstellung eines Quizspiels

In diesem Abschnitt verwenden Sie Copilot, um ein Quizspiel zu erstellen, das jeweils eine Multiple-Choice-Frage anzeigt und dann zufällig eine neue Frage auswählt, wenn der Benutzer eine Antwort auswählt. Denken Sie über die Struktur der Anwendung nach, die Sie erstellen möchten. Welche anderen Details können Sie sich vorstellen, die wichtig sind, um sicherzustellen, dass Copilot brauchbare Auswahlmöglichkeiten bietet?

In den folgenden Schritten sehen Sie, wie die Übung bei mir ablief. Ihre Erfahrung wird anders sein:

1. Erstellen Sie eine Datei namens `index.html` und geben Sie einen HTML-Kommentar ein.

Hier der Kommentar, den ich geschrieben habe:

```
<!-- Homepage für ein einfaches Quizspiel über Maschinenlernen -->
```

Als Antwort auf diesen Prompt schlägt Copilot eine DOCTYPE-Deklaration für eine HTML5-Datei vor, was korrekt ist.

2. Drücken Sie die `↵`, um den Vorschlag von Copilot zu akzeptieren, und drücken Sie dann die `↵`, um zur nächsten Zeile zu gehen und den nächsten Vorschlag zu bewerten.

3. Akzeptieren Sie die Vorschläge von Copilot so lange, bis er Ihnen einen Vorschlag macht, mit dem Sie nicht einverstanden sind.

Copilot schlug eine einfache HTML-Vorlage vor, die ein Formular enthält, das an ein PHP-Skript weitergeleitet wird. Das ist nicht das, was ich mir vorgestellt habe, also

muss ich zusätzliche Informationen für meine ursprüngliche Eingabeaufforderung bereitstellen. Ich habe meinen Prompt wie folgt geändert:

```
<!-- Homepage für ein Quizspiel über maschinelles Lernen. Dieses Spiel zeigt eine Multiple-Choice-Frage mit JavaScript an, die Daten aus data/trivia.js abrufen. Die möglichen Antworten werden mit Optionsfeldern auf der linken Seite angezeigt. Wenn ein Optionsfeld angeklickt wird, wird dem Benutzer eine neue Zufallsfrage angezeigt. -->
```

Mit dem zusätzlichen Kontext war Copilot in der Lage, etwas zu erstellen, das meinen Vorstellungen viel näherkam:

```
<!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="Frage"></div>
      <div id="Antworten"></div>
      <div id="Score"></div>
    </main>
    <footer>
      <p>&copy; Chris Minnick</p>
    </footer>
  </div>
  <script src="js/trivia.js"></script>
</html>
```

4. Fügen Sie `type="module"` zum `<script>`-Tag hinzu, um Copilot mitzuteilen, dass Sie eine Import-Anweisung in der Datei `trivia.js` verwenden werden.
5. Erstellen Sie drei Ordner mit den Namen `js`, `css` und `data`.
6. Erstellen Sie im Ordner `data` eine Datei namens `triviaQuestions.js`.
7. Schreiben Sie einen Kommentar am Anfang von `triviaQuestions.js`, der beschreibt, was die Datei enthalten soll.

Das habe ich geschrieben:

```
// Ein Modul mit einem Array mit 10 Objekten
// mit Quizfragen und Antworten.
```

8. Wenn Ihr Prompt so wie der meine nicht den erwarteten Inhalt erzeugt hat, gehen Sie wie folgt vor:

- Klicken Sie auf das Chat-Symbol links in Visual Studio Code, um die Copilot-Chat-Oberfläche zu öffnen.

Sie können für diesen Schritt auch ChatGPT verwenden.

- Geben Sie in der Chat-Oberfläche den folgenden Prompt ein:

Gib mir ein JavaScript-Array mit 10 Multiple-Choice-Quizfragen zum maschinellen Lernen.

Als Antwort auf diesen Prompt erhielt ich von Copilot Chat eine ordnungsgemäß formatierte und angemessene Reihe von Fragen und Antworten. Hier die ersten paar Fragen:

```
const triviaQuestions = [
  {
    question: "Wie heißt der Algorithmus, der allgemein für überwachtes Lernen benutzt wird?",
    choices: ["Entscheidungsbaum", "K-Means", "Naive Bayes", "Zufallswald"],
    answer: "Entscheidungsbaum"
  },
  {
    question: "Wie heißt der Algorithmus, der allgemein für nicht überwachtes Lernen benutzt wird?",
    choices: ["K-Means", "Entscheidungsbaum", "Naive Bayes", "Zufallswald"],
    answer: "K-Means"
  },
  {
    question: "Wie heißt der Algorithmus, der allgemein für verstärkendes Lernen benutzt wird?",
    choices: ["Q-Learning", "K-Means", "Naive Bayes", "Zufallswald"],
    answer: "Q-Learning"
  },
  ...
]
```

9. Kopieren Sie das erzeugte Array und fügen Sie es in Ihre Datendatei ein.

10. Sie werden das Array in Ihre JavaScript-Datei importieren, also fügen Sie das Schlüsselwort export vor const ein, etwa so:

```
export const triviaQuestions = [
```

11. Erstellen Sie eine neue Datei im Ordner `js` mit dem Namen `trivia.js`.



Stellen Sie sicher, dass Sie `triviaQuestions.js` und `index.html` geöffnet lassen, während Sie an `trivia.js` arbeiten. Copilot verwendet Dateien, die Sie geöffnet haben, als Kontext für die Datei, an der Sie gerade arbeiten.

12. Schreiben Sie einen Kommentar am Anfang von `trivia.js`, der beschreibt, was es tun soll.

Hier der Kommentar, den ich geschrieben habe:

```
/* JavaScript für das Quizspiel. Dieses Skript wird von der Datei index.html geladen und zeigt Fragen und mögliche Antworten an, aus denen die Benutzer wählen können. Das Spiel zeigt eine neue Zufallsfrage an, wenn der Benutzer auf ein Optionsfeld klickt, um eine Antwort auszuwählen, ebenso wie den Überblick über den Punktestand des Benutzers. */
```

13. Schreiben Sie unmittelbar nach dem Kommentar eine JavaScript-Import-Anweisung, um die Fragedaten zu importieren.

Unabhängig davon, ob Copilot herausfindet, was Sie tun und Ihnen hilft oder nicht, sollte die Importanweisung wie folgt aussehen:

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

14. Drücken Sie die `(↵)` und akzeptieren Sie die von Copilot vorgeschlagenen Variablen.

Schließlich wird Copilot eine Funktion vorschlagen.



Akzeptieren Sie die vorgeschlagene Funktion nicht sofort. Wenn Copilot keine Vorschläge macht, versuchen Sie, eine Leerzeile einzufügen. Beobachten Sie dann das Copilot-Symbol in der unteren rechten Ecke. Es sollte sich zu drehen beginnen, und nach ein paar Sekunden sehen Sie einen Vorschlag, wie Sie mit dem Schreiben des Codes beginnen können.

15. Bewegen Sie den Mauszeiger über den Funktionsvorschlag, um das Copilot-Menü anzuzeigen, in dem mehrere mögliche Vorschläge aufgelistet sein können, wie in Abbildung 1.14 dargestellt.

16. Wenn Ihnen einer der Vorschläge gefällt, akzeptieren Sie ihn. Wenn nicht, versuchen Sie, Ihren Kommentar zu verfeinern, indem Sie mehr Informationen darüber bereitstellen, was Sie wollen.

17. Setzen Sie diesen Prozess des Annehmens von Vorschlägen, des Schreibens von Code und des Verwendens von Kommentaren als Kontext fort, bis Sie etwas haben, das funktionieren könnte.

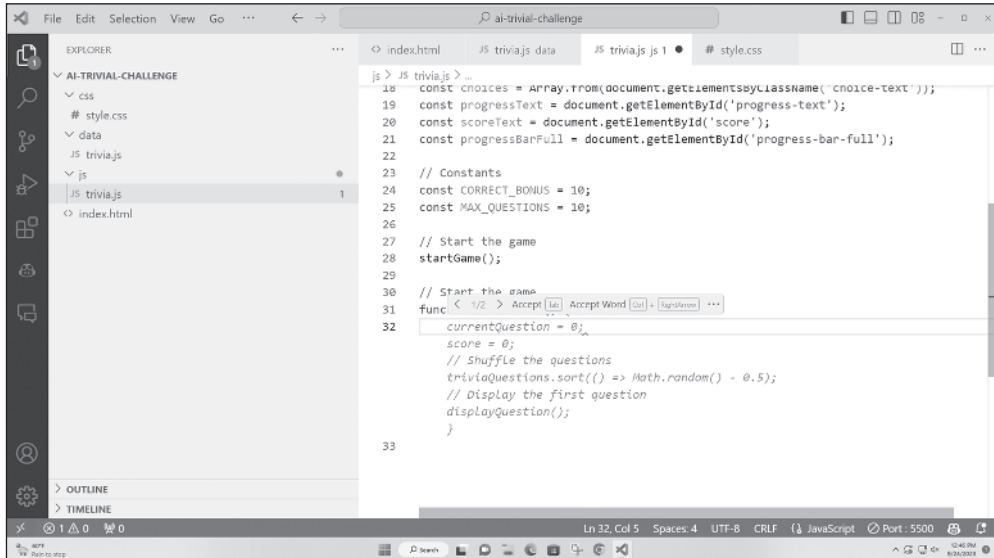


Abbildung 1.14: Anzeige des Copilot-Menüs und mehrerer Vorschlagsoptionen © Microsoft Corporation

Jetzt ist es an der Zeit, eine Vorschau Ihrer Anwendung anzusehen:

1. **Klicken Sie auf das Erweiterungssymbol links neben VS Code (siehe Abbildung 1.13) und verwenden Sie das Suchfeld, um die Live-Server-Erweiterung zu finden.**
2. **Klicken Sie auf die Schaltfläche INSTALL (Installieren) unter der Live-Server-Erweiterung.**

Diese Erweiterung öffnet HTML-Dateien mithilfe eines Entwicklungsservers.

3. **Klicken Sie im Datei-Explorer von Visual Studio Code mit der rechten Maustaste auf index.html und wählen Sie MIT LIVE SERVER ÖFFNEN.**

Ihre Anwendung wird in Ihrem Standard-Webbrowser geöffnet.

Abbildung 1.15 zeigt das (grobe und unfertige) Spiel, das ich mit der Hilfe von Copilot erstellt habe. Bei meinem zweiten Versuch benötigte ich dafür etwa 20 Minuten.

Wenn Ihre Anwendung nicht funktioniert, versuchen Sie die Fehlersuche mithilfe von Copilot Chat.

Unabhängig davon, ob Sie eine brauchbare – oder sogar gute – Anwendung erhalten haben, sollten Sie sich die folgenden Fragen stellen:

- ✓ War die Paar-Programmierung mit KI einfacher oder schwieriger als die Paar-Programmierung mit einem anderen Programmierer?

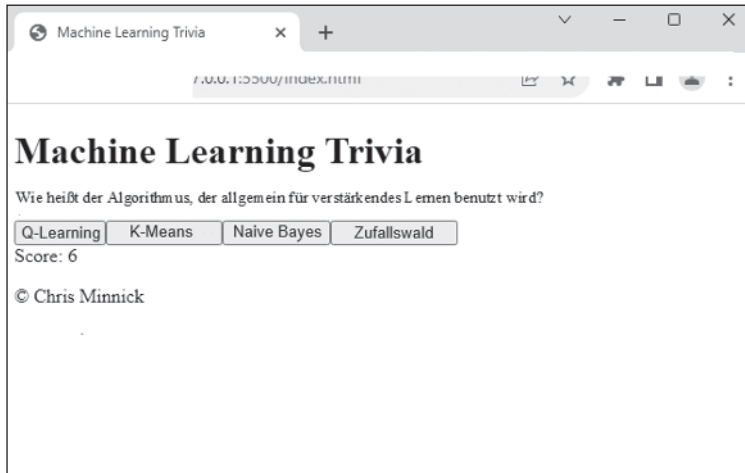


Abbildung 1.15: Ein einigermaßen funktionierendes Quizspiel

- ✓ Waren Sie von den Vorschlägen von Copilot überrascht (positiv oder negativ)?
- ✓ Wie würden Sie Ihre Herangehensweise an die Paar-Programmierung mit KI ändern, wenn Sie diese Übung wiederholen würden?

In Teil II erfahren Sie mehr darüber, wie Sie mit GenKI bessere Ergebnisse erzielen können.

