MERKBLATT
KI IN DER INFORMATIKLEHRE

KOMPETENZZENTRUM
FÜR INNOVATION IN
STUDIUM UND LEHRE

ALLGEMEINE TIPPS

- KI kann die Produktivität eines Programmierers **steigern,** aber **nicht ersetzen.** Kreativität und analytische Fähigkeiten bleiben bei Ihnen.
- Je klarer Ihre Anweisungen, desto nützlicher die KI-Unterstützung. Verwenden Sie **klare und detaillierte Prompts**.
- Nutzen Sie Autokorrektur und intelligente Prompting-Funktionen, um effektiver und effizienter zu programmieren.

1. BEKANNTESTE ANBIETER ZUR INTEGRATION ALS EXTENSION IN VISUAL STUDIO CODE

Verifizierung über die Hochschule für kostenfreie Vollversion (Immatrikulationsbescheinigung oder E-Mail)

- **Codeium** *(kostenfrei Version via Emailanmeldung)* | https://codeium.com
- **GitHub Copilot** *(Vollversion via Verifizierung als Student/ Mitglied einer Bildungseinrichtung)* | https://github.com/features/copilot | https://github.com/education/students
- **Tabnine** *(kostenfrei Version via Emailanmeldung)* | https://www.tabnine.com/
- **Blackbox AI** *(kostenfrei Version via Emailanmeldung)* | https://www.blackbox.ai/
- **CodeGPT** *(kostenfrei Version via Emailanmeldung)* | https://codegpt.co/

2. FUNKTIONSLISTE AM BEISPIEL GITHUB COPILOT

- **Prompting**

- Ist möglich über den *Chatbereich* oder *direkt im Code*
- Konkrete Prompts mit Angaben zur Programmiersprache, den verwendeten Bibliotheken, Datentypen von Eingangsdaten etc. helfen dem Chatbot qualitativ hochwertigen Code zu liefern.
- Beispiel auf der Folgeseite:

Prompt:

„Erstelle in Python eine Klasse Person mit Namen und Geld. Erstelle eine Klasse Autoverkäufer, die von Person erbt und eine Liste mit Autos hat. Jedes Auto hat einen Namen, einen Kilometerstand, eine Marke und einen Geldwert. Importiert werden diese Informationen aus einer JSON-Datei. Zusätzlich verfügt die Klasse über eine Funktion Autoverkaufen bei der ein Person, wenn sie über genug Geld verfügt, ein Auto kaufen kann beim Autoverkäufer. Dokumentiere und kommentiere den gesamten Code.“

Ergebnis:

```python
import json

# Definiert eine Basisklasse für Personen
class Person:
    def __init__(self, name, geld):

# Definiert eine Klasse für Autos
class Auto:
    def __init__(self, name, kilometerstand, marke, geldwert):

# Definiert eine Klasse für Autoverkäufer, die von Person erbt
class Autoverkäufer(Person):
    def __init__(self, name, geld):
        # Ruft den Konstruktor der Basisklasse auf, um den Namen und das Geld zu initialisieren
        super().__init__(name, geld)
        # Initialisiert eine Liste, um Autos zu speichern
        self.autos = []

    def auto_verkaufen(self, kaeufer, auto_name):

    def autos_laden(self, dateipfad):
        # Lädt Auto-Daten aus einer JSON-Datei
        with open(dateipfad, 'r') as file:
            autos_daten = json.load(file)
            # Erstellt Auto-Objekte basierend auf den geladenen Daten und fügt sie zur Liste hinzu
            for auto in autos_daten:
                self.autos.append(Auto(**auto))

# Beispiel zur Verwendung der Klassen
verkaeufer = Autoverkäufer('Hans', 10000)  # Erstellt einen Autoverkäufer
verkaeufer.autos_laden('autos.json')  # Lädt Autos aus einer JSON-Datei
kunde = Person('Maria', 5000)  # Erstellt einen Kunden
verkaeufer.auto_verkaufen(kunde, 'Golf')  # Versucht, ein Auto zu verkaufen
```

Erklärung von Code

- Aufgrund von Variablennamen, Datentypen, Kommentaren und Zusammenhang wird ein erklärender Text mit darauf basierenden Vermutungen verfasst.
- Die Qualität der Erklärung variiert mit der Dokumentation und der Qualität des zu erklärenden Codes.
- Beispiel:

Zeile im Quellcode:

```
[gleichepunkte, nichtgefundenPunkte] = gleicheZielmarken.finden(messReihen[0], messReihen[1])
```

Erklärung des KI-Tools:

Die aktive Auswahl ist eine Zeile Python-Code, die die Methode finden auf einem Objekt namens **gleicheZielmarken** aufruft. Diese Methode erhält zwei Datensätze von Messungen (**messReihen[0]** und **messReihen[1]**) als Argumente. Die Methode ist darauf ausgelegt, diese beiden Messdatensätze zu vergleichen, um übereinstimmende und nicht übereinstimmende Messpunkte zwischen ihnen zu identifizieren. Das Ergebnis dieses Vergleichs wird dann in zwei Variablen aufgeteilt: **gleichePunkte** und **nichtgefundenPunkte**.

[...]

Schließlich gibt die Methode zwei Listen zurück: **gleichePunkte**, die die Listen der übereinstimmenden Messpunkte aus beiden Sätzen enthält, und **nichtgefundenePunkte**, die die Listen der Messpunkte aus jedem Satz enthält, die keine Übereinstimmung gefunden haben. Diese zurückgegebenen Listen werden in die Variablen **gleichePunkte** und **nichtgefundenPunkte** aufgeteilt. Diese Operation kategorisiert effektiv die Messpunkte in solche, die in beiden Messungen konsistent sind, und solche, die einzigartig für jede Messung sind. Dies kann für Anwendungen, die eine präzise Verfolgung von Veränderungen oder Anomalien über die Zeit erfordern, wie in der Robotik oder Qualitätskontrolle in der Fertigung, entscheidend sein.

Dokumentation von Code

- Automatische Erstellung von Code-Dokumentation
- Wichtig in der Zusammenarbeit mit Kolleg*innen oder zur Fertigstellung eines Projektes
- Ist als Vorlage zu verstehen
- Sollte vom Nutzer*innen erweitert und überprüft werden
- Beispiel:

Quellcode (vorher)

```
def transformiereGleichePunkte(punktliste1,punktliste2):

    [gleichepunkte, nichtgefundenPunkte] = gleicheMesspunkte.finden(punktliste1,punktliste2)

    log = []

    [Parameter,log, transformiert,rms_pass_x,rms_pass_y,rms_pass_z] = trafo.Trafo_Helmert_7P_M_fest(
        np.array(gleichepunkte[0]),np.array(gleichepunkte[1]),log)

    return [Parameter, log, transformiert, rms_pass_x, rms_pass_y, rms_pass_z]
```

Diese Zeilen selektieren und den folgenden Prompt ausführen:

`"/doc comment the code and translate documentation and comment into german"`

Quellcode (nachher)

```
def transformiereGleichePunkte(punktliste1, punktliste2):
    """
    Transformiert die gegebenen Punktlisten mithilfe der Helmert-Transformation.

    Args:
        punktliste1 (list): Liste der Punkte im Quellkoordinatensystem.
        punktliste2 (list): Liste der Punkte im Zielkoordinatensystem.

    Returns:
        list: Eine Liste, die die Transformationsparameter, Protokollinformationen,
              transformierte Punkte und RMS (Root Mean Square) Werte für jede Achse enthält.
    """

    # Finde gemeinsame Punkte zwischen den beiden Punktlisten
    [gleichepunkte, nichtgefundenPunkte] = gleicheMesspunkte.finden(punktliste1, punktliste2)

    log = []

    # Führe die Helmert-Transformation mit festen 7 Parametern durch
    [Parameter, log, transformiert, rms_pass_x, rms_pass_y, rms_pass_z] = trafo.Trafo_Helmert_7P_M_fest(
        np.array(gleichepunkte[0]), np.array(gleichepunkte[1]), log)

    return [Parameter, log, transformiert, rms_pass_x, rms_pass_y, rms_pass_z]
```

Autovervollständigung

- Während der Studierende Code schreibt, schlägt das Tool schlägt automatisch das nächste *Zeichen*, *Wort* oder *die gesamte Zeile* vor.
- Vermeidung von *Tipp- oder Klammerfehlern*
- Individuelle Steuerung der Autovervollständigungsvorschläge des Codes über *Kommentaren*
- Die Autovervollständigung orientiert sich an bereits bestehenden Quellcode in der Programmdatei. Vorschläge sind somit zutreffender, wenn eine ähnliche Syntax bereits in Quellcode vorhanden ist.
- Beispiel in TypeScript:

```
122     /**
123      * Returns a list of devices in the Smart Home System.
124      */
125     public getDevices(): SmartHomeDevice[] {
            return this.devices;
```

MERKBLATT
KI IN DER INFORMATIKLEHRE

KOMPETENZZENTRUM
FÜR INNOVATION IN
STUDIUM UND LEHRE

GOOD TO KNOW

- Das Resultat des automatisch **generierten Codes sollte nicht als finale Lösung** angesehen werden, sondern als ein Zwischenschritt zur Arbeitserleichterung, bevor Sie, als Mensch das Ergebnis finalisiert.
- Die generierte **Codequalität** variiert mit der Menge an vorhandenem Code im Internet zum entsprechenden Themenfeld.
- Einzelne Funktionen oder vollständige Bibliotheken können in den Vorschlägen nicht enthalten sein, wenn diese aktueller sind als der **Trainingsstand** des zugrunde liegenden KI-Modells.

DATENSCHUTZ

- Der **Datenschutz** ist oft nur in den Premium-Paketen der jeweiligen Anbieter enthalten. Ihre Eingaben könnten somit für Trainingszwecke der KI verwendet werden und zu einem späteren Zeitpunkt einem anderen Nutzer als Lösung vorgeschlagen werden.
- Die **Quelle** für die exakten Codevorschläge sind unbekannt.
- Automatisch generierter Code kann lizenzierten Code enthalten, sodass **Urheberrechtsverletzungen** nicht ausgeschlossen werden können.

PRÜFUNGSLEISTUNG UND KI

KI-Tools sind im Rahmen von P**rüfungsleistung an der Hochschule** erst einmal verboten, können aber in der Eigenständigkeitserklärung vom Prüfenden erlaubt werden. Für genauere Informationen werfen Sie dazu einen Blick in die Dokumentation zur Nutzung von generativen KI-Tools.

3. ETHIK ASPEKT

- Beachten Sie **Umweltaspekte** bei der Arbeit mit KI. Einer Studie nach hat das Training vom GPT-3 Modell zur Verdunstung von 700.000 Litern sauberen Frischwassers geführt. Dies entspricht dem Jahresverbrauch von etwa 16 Einpersonenhaushalten in Deutschland.
- Für 2027 wird die globale Nachfrage nach sauberem Frischwasser durch KI auf 4,2 bis 6,6 Milliarden Kubikmeter geschätzt, was vier- bis sechsmal dem jährlichen Wasserverbrauch Dänemarks entspricht.

Links:

Weitere Informationen zum Thema KI in der Hochschullehre finden Sie auf unserer Website.

Stand 01.10.2024