**Codedokumentation: 3 x 3 Matrixtaschenrechner**

Dies ist die Dokumentation des Informatikprojekts von Moritz Wagner und Jonas Heselschwerdt. In diesem Dokument erläutern wir die Funktionsweise unseres Programms und gehen dabei auch auf Gedankengänge beim Entwurf und die mathematischen Grundlagen der einzelnen Berechnungen ein.

**Was macht unser Programm?**

Unser Python- Programm ist ein Matrixtaschenrechner der als Nutzeroberfläche (nachfolgend „UI“ genannt, kurz für „User Interface“) eine .txt- Datei und die Konsole von Visual Studio Code verwendet. Grundidee dahinter ist, dass der Nutzer in die .txt- Datei seine Matrizen schreibt und danach in der Konsole einen Rechenbefehl gibt. Das Programm gibt dem Nutzer dann über dieselbe .txt- Datei die Lösungen aus, sowie Informationen über was genau gemacht wurde. Das Programm arbeitet daher nach dem „EVA“ Prinzip (**E**ingabe - **V**erarbeitung – **A**usgabe). Das Ganze findet in einer Schleife statt, so dass der Nutzer mit den Lösungen danach weiter rechnen kann.

Grundidee hinter der Verwendung einer .txt- Datei für die Ausgabe und Teile der Eingabe war es, dass man hierdurch mehr Platz wie in der Konsole für Texte hat und der Nutzer daher ein übersichtlicheres Programm erhält, welches intuitiv und vor allem schnell zu bedienen ist.

**Der Grundaufbau**

Wie bereits erklärt läuft das Hauptprogramm in einer Schleife ab, in welcher auf die verschiedenen Funktionen und Variablen des Programms zugegriffen wird.

Diese Funktionen sind:

* **refreshUI()** : Eine Funktion welche für die Ausgabe der Lösungen der Berechnungen, sowie die Autoskalierung der Matrizen verantwortlich ist.
* **getMatrix()** : Eine Funktion die auf die .txt- Datei zugreift und die darin eingetippten Matrizen in zweidimensionale Arrays verwandelt.
* **getUserInput()** : Eine Funktion die den Nutzer abfragt welche Berechnungen er ausführen möchte. Diese Befehle werden auf Falscheingaben überprüft und der Nutzer wird gegebenenfalls zur Korrektur von Eingaben aufgefordert.
* **calculate()** : Diese Funktion startet je nach dem welcher Operator gewählt wurde eine der 5 Rechenfunktionen, diese sind:
  + **addition()** : Addiert zwei Matrizen
  + **subtraction()** : Bildet die Differenz zweier Matrizen
  + **multiplication()** : Multipliziert zwei Matrizen
  + **transpose()** : Transponiert eine Matrix
  + **invert()** : Invertiert eine Matrix
* **determinant() :** Auch eine Rechenfunktion, wird aber nicht von **calculate(),** sondern separat aufgerufen, bildet die Determinanten der Matrizen

Neben diesen Funktionen werden im Hauptprogramm auch einige Variablen genutzt, diese sind:

* **matrix\_A, matrix\_B, matrix\_C, matrix\_Ans und matrix\_Sol:** global definierte zweidimensionale 3x3 Arrays die über getMatrix() gefüllt werden. Beinhalten das, was der Nutzer eingegeben hat.
* **matrix\_Operand\_1 und matrix\_Operand\_2:** Zweidimensionale 3x3 Arrays in die zwischenzeitlich die Operanden gespeichert werden.
* **calculated und matrix\_saved\_in:** Strings, die speichern was berechnet wurde und wo das Ergebnis gespeichert wurde
* **help\_menue, output\_file und template:** Dateinamen der Vorlagen und der Output .txt- Datei
* **det\_A, det\_B, det\_C, det\_Ans und det\_Sol:** Floats, die die Werte der einzelnen Determinanten speichern.
* **userInput() :** Ein Array, dass 4 Strings beinhaltet, zum Speichern der vom Nutzer gewünschten Berechnung.

Das Hauptprogramm startet indem die Variable **userInput** gefüllt wird mithilfe der Funktion **getUserInput()**. Danach werden die eingetippten Matrizen mithilfe von **getMatrix()** gespeichert. Es werden nun, abhängig von der gewünschten Berechnung die richtigen Matrizen in **matrix\_Operand\_1** und **matrix\_Operand\_2** kopiert. Das Übergeben **von matrix\_Operand\_1, matrix\_Operand\_2** und **operator** (entspricht **userInput[0]**) an die Funktion **calculate()** sorgt nun dafür dass die richtige Berechnung ausgeführt wird. Das Ergebnis wird immer in **matrix\_Sol** gespeichert und anschließend in eine vom Nutzer spezifizierte Matrix kopiert, standardmäßig aber in **matrix\_Ans**, es sei denn der Nutzer gibt eine der drei anderen Matrizen als Ziel an. Nun werden noch die 5 Variablen der einzelnen Determinanten geändert mithilfe von **determinant()** und die Strings **calculated** und **matrix\_saved\_in** geändert. Zu guter Letzt wird **refreshUI()** aufgerufen um dem Nutzer die Ausgabe zu präsentieren, in der Konsole gibt es außerdem eine Rückmeldung darüber, dass die Berechnung erfolgreich ausgeführt wurde. Die While- Schleife des Hauptprogramms startet nun erneut.

**Die einzelnen Funktionen im Detail**

**getUserInput()**

getUserInput() beinhaltet eine While- Schleife die so lange ausgeführt wird, bis der Nutzer alle Eingaben korrekt gemacht hat, somit kann das Programm durch Falscheingaben nicht zum Absturz gebracht werden. Der Nutzer hat zum Start der Schleife auch die Möglichkeit ein Hilfsmenü aufzurufen. Dies geschieht durch die Eingabe von „?“. Daraufhin wird gespeichert, was der Nutzer bereits eingegeben hat und die UI wird durch das Hilfsmenü ersetzt. Durch das Drücken der Enter- Taste kann das Hilfsmenü wieder verlassen werden und der ursprüngliche Stand der UI wird wiederhergestellt. Die Funktion gibt eine Liste mit 4 Strings aus, die klar definieren, welche Operation ausgeführt werden soll. Wenn die Operation nur einen Operanden enthält (wenn transponiert oder invertiert wird), wird der zweite Operand nicht abgefragt.

**getMatrix()**

getMatrix() öffnet zunächst Output\_File.txt für Lesezwecke, dies ist einfach möglich wenn diese Datei im selben Ordner liegt wie das Python Programm. Es wird nun fest definiert in welchen Zeilen die Matrizen gesucht werden müssen. In den betroffenen Zeilen werden alle Style- Elemente entfernt und anschließend die Elemente darin getrennt durch „|“. Diese Elemente werden nun durch einen Typecast in Floats umgewandelt. Sollte der Nutzer eine Falscheingabe gemacht haben, wird dies als eine „0.0“ interpretiert, um ein Absturz des Programmes später zu verhindern. Die Floats werden nun in ein 3x3 Array hinzugefügt, welches die Matrix darstellt und von der Funktion ausgegeben wird.

Hinweis: Es existiert streng genommen in diesen Arrays ein Element [1][3] (Zeile 2, Spalte 4) welches durch die Determinante entsteht, die neben der Matrix angezeigt wird. Dieses Element beeinträchtigt die Rechenalgorithmen nicht, es wird nicht verwendet und ist durch den dabeistehenden Text immer der Float „0.0“.

**addition() und subtraction()**

Diese Funktionen funktionieren sehr einfach und sind fast identisch, durch zwei for– Schleifen werden alle positionsgleichen Elemente der Operanden addiert / subtrahiert. Die Laufindizes der for- Schleifen werden hierbei benutzt, um auf die 3x3 Arrays zuzugreifen.

**multiplication()**

Es werden 3 for- Schleifen verwendet um 9-mal die „i-te“ Zeile mit der „j-ten“ Spalte skalar zu multiplizieren und es wird anschließend das Ergebnis an der „i-ten“ Zeile und „j-ten“ Spalte der matrix\_Sol gespeichert.

**transpose()**

Mit zwei for- Schleifen werden hier die Elemente an der Stelle [ i ][ j ] an die Stelle [ j ] [ i ] gebracht. Dies entspricht mathematisch einer Spiegelung an der Hauptdiagonalen.

**invert()**

Der hier genutzte Weg um an die Inverse zu kommen ist eine geschickte Umformung der adjunkten Matrix, die in folgender Abbildung dargestellt ist:

Ein Bild, das Text, Handschrift, Schrift, Entwurf enthält.

KI-generierte Inhalte können fehlerhaft sein.

(Quelle: <https://de.wikipedia.org/wiki/Adjunkte>)

Die letzte Darstellungsweise der Adjunkten wird im Programm für die Berechnung verwendet. Die einzelnen Elemente der Inversen können durch Multiplikationen, Subtraktionen und Divisionen einfach berechnet werden.

Das multiplizieren der Adjunkten mit dem Kehrwert der Determinante, die ebenfalls in invert() gebildet wird (durch Aufrufen von determinant() ) ergibt die Inverse Matrix.

Ist die Determinante = 0 wird ein Fehler ausgegeben und keine Berechnung angestellt. Die Inverse existiert in diesem Fall nicht.

**determinant()**

In dieser Funktion werden die Determinanten aller 5 gespeicherten Matrizen berechnet. Mathematische Grundlage für die Berechnung ist hierbei der Satz von Sarrus, die Funktion gibt die Liste det zurück welche alle 5 Determinanten beinhaltet.

**refreshUI()**

Hier findet zuallererst die Ausgabe der berechneten Matrizen statt, diese werden hier autoskaliert. Die Autoskalierung sorgt dafür, dass alle angezeigten Zahlen (werden umgewandelt in Strings durch einen Typecast) aus 5 Zeichen bestehen (inklusive Dezimalpunkt, negatives Vorzeichen nicht miteinbezogen). Bei positiven Zahlen und Zahlen mit 4 Ziffern vor dem Komma wird durch das geschickte Setzen von Leerzeichen vor und hinter der Zahl dafür gesorgt, dass diese mit kleineren oder negativen Zahlen in derselben „Spalte“ starten. Die Autoskalierung funktioniert für Zahlen von -9999 bis 9999. Umso größer die Zahl ist, umso mehr Nachkommastellen werden „abgeschnitten“.

Anschließend werden noch die in Template.txt enthaltenen Platzhalter für die Determinanten, die Rechenoperation und die Zielmatrix für die Lösung überschrieben. Template.txt dient daher als Vorlage, die modifiziert und in Output\_File.txt transferiert wird.

**Fazit**

Wir sind mit der Endversion des Programms sehr zufrieden, das einzige Ziel, welches wir nicht umsetzen konnten, war das Berechnen der Eigenwerte, dies hat sich als zu komplex herausgestellt und hätte vermutlich dafür gesorgt, dass die maximale Codezeilenanzahl von 500 Zeilen überschritten wird. Wir hätten hierfür entweder komplexe numerische Verfahren benötigt oder eine Polynomdivision ausführen müssen, was beides den Rahmen des Programms gesprengt hätte.

Weitere Funktionen, die man hinzufügen könnte, wären zum Beispiel:

* Erweiterung auf Matrizen die keine 3 x 3 Matrizen sind
* Skalarmultiplikation
* Die komplette Eingabe findet in Output\_File.txt statt, ohne die Konsole von Visual Studio Code zu verwenden.

Wir sind außerdem damit zufrieden, dass unser Programm laut unserem Kenntnisstand Absturzsicher ist, in unseren Tests stürzte das Programm nicht ab, egal inwiefern der Nutzer Falscheingaben in der Konsole oder in Output\_File.txt macht.

In diesem Dokument wurde grob erklärt was unser Programm und die darin enthaltenen Funktionen machen, für weitere Informationen verweisen wir auf die Kommentare im Programmcode und auf README.md, sowie das Hilfsmenü, welches ins Programm integriert ist.