Block 04 Übung: Datenstrukturen

Introduction to Programming, Frühjahrssemester 2025
Author: Giuseppe Accaputo

In dieser Übung hast Du die Gelegenheit, das in den ersten vier Vorlesungen Gelernte praktisch anzuwenden. Deine Aufgabe ist es, ein bestimmtes Programm zu implementieren, das Dir dabei hilft, die Konzepte und Techniken, die Du bisher studiert hast, zu vertiefen und zu festigen. Während der Umsetzung dieser Aufgabe ist es essenziell, dass Du Deine Gedankengänge sorgfältig dokumentierst.

Inhaltsverzeichnis

Aufgabe: Implementation einer TODO-Liste

In dieser Aufgabe entwickelst Du ein benutzerfreundliches Python-Programm, das als TODO-Liste dient.

Abgabe 1: Dokumentation von Gedankengängen

Dokumentiere deine Überlegungen bereits vor und während der Programmierung, nicht erst nachher. Ich möchte verstehen, wie du an die Aufgabe herangegangen bist und welche Entscheidungen du getroffen hast, bevor du den ersten Code geschrieben hast. Das

Dokumentieren deiner Gedankengänge vor Beginn der Programmierung ist ein wesentlicher Teil dieser Aufgabe (siehe Abschnitt *Planugsphase (vor dem Coding)* **- fehlt dieser Teil, führt das zu einem erheblichen Punktabzug.**

Bei jeder verwendeten Quelle (Internetseiten, Bücher, KI-Tools wie ChatGPT usw.) musst du nicht nur die Quelle angeben, sondern auch erklären, was du dabei gelernt hast und warum du diese Quelle benötigt hast. Dies gilt besonders für Chatbots und KI-Tools – erkläre, für welchen spezifischen Teil der Aufgabe du Hilfe gesucht hast und wie die erhaltene Information dir geholfen hat, das Problem besser zu verstehen oder zu lösen.

Planungsphase (vor dem Coding)

- Beschreibe deinen gedanklichen Prozess zur Lösung des Problems. Wie hast du das Problem zunächst strukturiert/zerlegt?
- Welche Datenstrukturen und Algorithmen hast du in Betracht gezogen und warum?
- Skizziere deine initiale Lösungsstrategie (eventuell mit einem Pseudocode oder Flussdiagramm)
- Welche Teile der Aufgabe hast du als besonders herausfordernd angesehen und warum?

Implementierungsphase

- Beschreibe wichtige Entscheidungen während der Programmierung. Warum hast du bestimmte Ansätze gewählt?
- Auf welche Probleme bist du gestossen und wie hast du diese gelöst?
- Hast du deine ursprüngliche Strategie angepasst? Warum?
- Welche alternativen Lösungswege hast du verworfen und aus welchen Gründen?

Reflexion

- Was hast du aus diesem Projekt gelernt?
- Was würdest du beim nächsten Mal anders machen?
- Gibt es Teile deiner Lösung, die du noch verbessern könntest? Welche?

Diese Art der Dokumentation wird nicht nur deine Fähigkeit zur Selbstreflexion verbessern, sondern dir auch helfen, deine Programmierfähigkeiten kontinuierlich zu entwickeln.

Verwende für die Dokumentation deiner Gedankengänge maximal zwei A4 Seiten.

Abgabe 2: Implementation des TODO-Listen-Programms

Dein TODO-Listen-Programm sollte folgende Aktionen mittels einer einfachen Text-Schnittstelle ermöglichen:

1. **Hinzufügen einer Aufgabe (Befehl h):**
   - Fordere eine Beschreibung für jede neue Aufgabe an.
   - Stelle sicher, dass die Beschreibung nicht leer ist. Falls der Benutzer eine leere Aufgabe versucht hinzuzufügen, gib eine Fehlermeldung aus.
2. **Anzeigen aller Aufgaben (Befehl a):**

- Liste alle vorhandenen Aufgaben mit ihrer Beschreibung und ihrem Status (offen oder erledigt) auf.
- Gib eine Nachricht aus, wenn keine Aufgaben vorhanden sind.

3. **Markieren einer Aufgabe als erledigt (Befehl `m`):**
   - Ermögliche es, eine Aufgabe durch Eingabe ihrer Nummer als erledigt zu markieren. Die Nummerierung der Aufgaben beginnt dabei bei 1, d.h. die erste Aufgabe hat in der Anzeige die Nummer 1. Möchte der Benutzer die Aufgabe 1 als erledigt markieren, so muss 1 eingegeben werden.
   - Prüfe, ob die eingegebene Nummer einer existierenden Aufgabe entspricht. Falls nicht, informiere den Benutzer, dass keine Aufgabe unter der eingegebenen Nummer gefunden werden kann.
4. **Löschen einer Aufgabe (Befehl `l`):**
   - Erlaube das Löschen einer Aufgabe durch Eingabe ihrer Nummer. Die Nummerierung der Aufgaben beginnt dabei bei 1, d.h. die erste Aufgabe hat in der Anzeige die Nummer 1. Möchte der Benutzer die Aufgabe 1 als erledigt markieren, so muss 1 eingegeben werden.
   - Prüfe, ob die eingegebene Nummer einer existierenden Aufgabe entspricht. Falls nicht, informiere den Benutzer, dass keine Aufgabe unter der eingegebenen Nummer gefunden werden kann.
5. **Beenden des Programms (Befehl `b`):**
   - Beende das Programm durch die Eingabe von `b`.

Das Programm fordert den Benutzer wiederholt zur Eingabe von Aktionen auf, bis dieser entscheidet, das Programm mit dem Befehl **`b`** zu beenden. Sollte ein ungültiger Befehl eingegeben werden, zeigt das Programm eine Fehlermeldung an und bittet den Benutzer, einen gültigen Befehl erneut einzugeben. Wie Du die Eingaben aus der Konsole und Ausgaben auf der Konsole im Programm gestalten solltest entnimmst Du bitte aus den Beispielabläufen des Programms im Abschnitt *Beispielablauf*.

Deinen Code darfst Du in der mitgelieferten Datei ***`todo_list.py`*** an der markierten Stelle implementieren. Bitte passe den existierenden Code nicht an und verwende in Deiner Lösung die beiden vorgegebenen Listen **`descriptions`** und **`status`**.

Die beiden Listen **`descriptions`** und **`status`** werden verwendet, um Informationen über Aufgaben in Deinem TODO-Listen-Programm zu speichern. Hier ist eine Erläuterung ihrer Rollen und des Zusammenhangs zwischen ihnen:

- **Liste `descriptions`**: Diese Liste speichert die Beschreibungen der Aufgaben. Jeder Eintrag in dieser Liste ist ein String, der angibt, was zu tun ist. Beispielsweise könnte ein Eintrag **`"Einkaufen gehen"`** oder **`"Python üben"`** sein. Diese Liste dient als Hauptinformationsquelle darüber, welche Aufgaben der Benutzer geplant hat.
- **Liste `status`**: Diese Liste speichert den Status jeder Aufgabe, wobei jeder Eintrag ein Boolean-Wert ist. Ein **`False`** bedeutet, dass die Aufgabe an dieser Position in der

Liste `descriptions` noch nicht erledigt ist, und ein `True` zeigt an, dass die Aufgabe abgeschlossen wurde.

Die beiden Listen sind eng miteinander verbunden und arbeiten parallel zueinander. Für jeden Eintrag in der Liste `descriptions` gibt es einen korrespondierenden Eintrag in der Liste `status` am gleichen Index. Dies bedeutet, dass der Status einer Aufgabe durch Zugriff auf den `status`-Eintrag am gleichen Index wie die Aufgabenbeschreibung in der Liste `descriptions` abgefragt oder geändert werden kann.

Beispiel: Angenommen, die Liste `descriptions` enthält die Einträge `["Einkaufen gehen", "Python üben"]`. Die entsprechende Liste `status` könnte dann `[False, False]` enthalten, was bedeutet, dass beide Aufgaben noch nicht erledigt sind. Wenn die Aufgabe `"Einkaufen gehen"` erledigt wird, ändert man den ersten Eintrag in der Liste `status` auf `True`, was nun `[True, False]` ergibt. Wenn eine neue Aufgabe hinzugefügt wird, so wird die Liste `descriptions` um diese erweitert und die Liste `status` wird parallel um einen weiteren Zustand erweitert, welcher dem Zustand der neu hinzugefügten Aufgabe repräsentiert.

Diese Struktur ermöglicht es dem Programm, effizient den Überblick über Aufgaben und deren Erfüllungsstatus zu halten basierend auf den Themen, welche wir in den ersten vier Vorlesungen angeschaut haben.

Beispielablauf

Nachfolgend sind einige Beispielabläufe des Programms auf der Konsole dargestellt. Eingabeaufforderungen erscheinen in `blau`, Benutzereingaben in ***grün*** und die Ausgaben des Programms in **schwarz**. Die `grau` dargestellten Zahlen neben den Beispielen sind Zeilennummern, die nur zur Orientierung dienen und nicht Teil der tatsächlichen Konsolenausgabe sind.

Hinzufügen und anzeigen mehrerer Aufgaben

```
1 Wähle eine Aktion (h, a, m, l, b): h
2  > Beschreibung der Aufgabe: Gedanken zur Implementation machen
3 Wähle eine Aktion (h, a, m, l, b): h
4  > Beschreibung der Aufgabe: TODO-Liste implementieren
5 Wähle eine Aktion (h, a, m, l, b): a
6  1: Gedanken zur Implementation machen (Status: Offen)
7  2: TODO-Liste implementieren (Status: Offen)
```

Markieren einer Aufgabe als erledigt

```
1 Wähle eine Aktion (h, a, m, l, b): a
2  1: Gedanken zur Implementation machen (Status: Offen)
3  2: TODO-Liste implementieren (Status: Offen)
4 Wähle eine Aktion (h, a, m, l, b): m
5  > Aufgabennummer markieren: 0
6  * Fehler: Ungültige Aufgabennummer.
```

```
Wähle eine Aktion (h, a, m, l, b): m
 > Aufgabennummer markieren: 3
 * Warnung: Aufgabennummer nicht gefunden.
Wähle eine Aktion (h, a, m, l, b): m
 > Aufgabennummer markieren: 1
 * Aufgabe 1 wurde als erledigt markiert.
Wähle eine Aktion (h, a, m, l, b): a
 1: Gedanken zur Implementation machen (Status: Erledigt)
 2: TODO-Liste implementieren (Status: Offen)
```

Löschen einer Aufgabe

```
Wähle eine Aktion (h, a, m, l, b): a
  1: Gedanken zur Implementation machen (Status: Erledigt)
  2: TODO-Liste implementieren (Status: Offen)
Wähle eine Aktion (h, a, m, l, b): l
 > Aufgabennummer löschen: 0
 * Fehler: Ungültige Aufgabennummer.
Wähle eine Aktion (h, a, m, l, b): l
 > Aufgabennummer löschen: 3
 * Fehler: Ungültige Aufgabennummer.
Wähle eine Aktion (h, a, m, l, b): l
 > Aufgabennummer löschen: 2
 > Aufgabe 2 wurde gelöscht.
Wähle eine Aktion (h, a, m, l, b): a
 1: Gedanken zur Implementation machen (Status: Erledigt)
```

Eingabe eines ungültigen Befehls

```
Wähle eine Aktion (h, a, m, l, b): a
 1: Gedanken zur Implementation machen (Status: Offen)
 2: TODO-Liste implementieren (Status: Offen)
Wähle eine Aktion (h, a, m, l, b): k
 * Fehler: Ungültige Aktion. Bitte erneut versuchen.
Wähle eine Aktion (h, a, m, l, b): z
 * Fehler: Ungültige Aktion. Bitte erneut versuchen.
Wähle eine Aktion (h, a, m, l, b): 3
 * Fehler: Ungültige Aktion. Bitte erneut versuchen.
Wähle eine Aktion (h, a, m, l, b): a
 1: Gedanken zur Implementation machen (Status: Erledigt)
 2: TODO-Liste implementieren (Status: Offen)
```

Beenden des Programms

```
Wähle eine Aktion (h, a, m, l, b): a
 1: Gedanken zur Implementation machen (Status: Erledigt)
Wähle eine Aktion (h, a, m, l, b): b
 * Auf wiedersehen!
```

Tipps zur Implementation

- Bitte beachte, dass während der Programmausführung alle Aufgaben nummeriert sind, beginnend mit der Nummer `1`. Dies gilt sowohl für die Anzeige aller Aufgaben als auch für das Markieren und Löschen einer Aufgabe. Wenn Du beispielsweise die dritte Aufgabe löschen möchtest, so muss eine `3` eingegeben werden.
- Die Methode `strip()` eines String-Objektes entfernt führende und abschliessende Leerzeichen von einem String, wodurch sie nützlich ist, um leere Eingaben oder solche, die nur aus Leerzeichen bestehen, in Benutzereingaben zu erkennen und zu behandeln. Der Ausdruck `"  Hallo, Welt!  ".strip()` evaluiert zum String `"Hallo, Welt!"`; der Ausdruck `"  ".strip()` (bestehend aus nur Leerzeichen) evaluiert hingegen zum leeren String `""`. Überlegt euch, wie ihr die `strip()`-Methode (in Kombination mit einer weiteren Funktion) verwenden könnt, um leere Eingaben beim Hinzufügen von Aufgaben abzufangen, z.B. wenn der Benutzer sich entscheidet, für den Aufgabentitel eine Eingabe bestehend aus nur Leerzeichen zu verwenden.
- Um die Aufgaben in Deiner Liste mit einer Nummerierung anzuzeigen, kannst Du die `range()`-Funktion zusammen mit der `len()`-Funktion verwenden, um durch die Liste zu iterieren. Dies erlaubt es Dir, sowohl den Index als auch den Inhalt jedes Eintrags während der Iteration zu verwenden. Denke daran, dass `range()` bei `0` beginnt, also möglicherweise eine Anpassung nötig ist, um die Nummerierung benutzerfreundlicher zu gestalten (beginnend bei `1`).

Bewertungskriterien

Abgabe 1: Dokumentation von Gedankengängen

- Die Dokumentation spiegelt den tatsächlichen Problemlösungsprozess wider, nicht nur eine nachträgliche Erklärung des Codes
- Die Gedankengänge sind klar und logisch strukturiert
- Die Überlegungen zur Programmstruktur sind nachvollziehbar
- Alle Quellen (inkl. KI-Tools wie ChatGPT) sind korrekt zitiert
- Der Text ist verständlich formuliert und frei von gravierenden Rechtschreibfehlern

Don'ts - Was zu vermeiden ist

- **Nur den fertigen Code beschreiben oder kommentieren**
  ❌ "Diese Funktion prüft, ob die Eingabe gültig ist."
  Stattdessen: Erkläre die Entscheidungen hinter deiner Implementierung.
- **Nachträgliche Begründungen erfinden**
  ❌ "Natürlich habe ich von Anfang an Listen verwendet, weil ich wusste, dass sie am besten passen."
  Stattdessen: Dokumentiere ehrlich deinen Denkprozess und eventuelle Kursänderungen.
- **Oberflächliche oder generische Aussagen treffen**
  ❌ "Ich habe einen guten Algorithmus verwendet."
  Stattdessen: Erläutere konkret, warum du bestimmte Ansätze gewählt hast.
- **Fehler und Probleme verschweigen**
  ❌ "Alles hat auf Anhieb funktioniert."
  Stattdessen: Beschreibe auch Sackgassen und wie du aus ihnen gelernt hast.
- **Quellen nicht angeben**
  ❌ Codeblöcke aus dem Internet kopieren ohne Quellenangabe und Rechtfertigung.
  Stattdessen: Gib ehrlich an, welche externen Ressourcen du genutzt hast. Fehlende oder unvollständige Quellenangaben können zu wesentlichen Punktabzügen führen, da ich diese Quellen in den meisten Fällen identifizieren kann.
- **Nur das Ergebnis beschreiben, nicht den Weg dorthin**
  ❌ "Mein Programm kann Aufgaben hinzufügen, anzeigen, markieren und löschen."
  Stattdessen: Erkläre deinen schrittweisen Entwicklungsprozess.
- **Zentrale Entscheidungen nicht erklären**
  ❌ Keine Erklärung, warum du bestimmte Datenstrukturen gewählt hast.
  Stattdessen: Begründe wichtige Designentscheidungen.
- **Vorgefertigte Lösungen ohne eigene Überlegungen übernehmen**
  ❌ "Ich habe die Lösung aus dem Tutorial XYZ verwendet."
  Stattdessen: Zeige, wie du externe Ressourcen an deine speziellen Anforderungen angepasst hast.
- **Zu technischer Jargon ohne Erklärung**
  ❌ "Ich habe Polymorphismus implementiert, um die Aufgabe zu lösen."
  Stattdessen: Erkläre Konzepte in eigenen Worten, die deinem aktuellen Wissensstand entsprechen.

- **Zu wenig Detail bei kritischen Entscheidungen**
  ❌ "Ich habe mich für eine Schleife entschieden."
  Stattdessen: Erkläre, warum du dich zwischen verschiedenen Schleifentypen für eine bestimmte entschieden hast.

Abgabe 2: Implementation des TODO-Listen-Programms

- Sind alle erforderlichen Aktionen (hinzufügen, anzeigen, markieren, löschen, beenden) implementiert?
- Funktionieren alle implementierten Aktionen wie erwartet? Werden Fehler und ungültige Eingaben korrekt gehandhabt?
- Ist der Code gut lesbar, i.e., wurden jeweils passende Variablennamen definiert?

Quellenangabe und Verwendung von Chatbots

- Bitte lies die ehrenwörtliche Erklärung sorgfältig durch.
- Um Quellenangaben hinzuzufügen, verwende das Quellenverwaltungs-Feature in Word. Eine Anleitung dazu findest du hier: **https://support.microsoft.com/de-de/office/add-or-change-sources-citations-and-bibliographies-159264ec-0a8a-4e9e-acf7-21faa9c371c2**.
- Wenn Du ein Chatbot (z.B. ChatGPT) verwendest, musst Du den Chat-Verlauf freigeben und zum Quellenverzeichnis hinzufügen, und in einem Abschnitt der Quellenangabe begründen, warum ChatGPT genutzt und was dabei gelernt wurde.
- Im Code musst du zusätzlich die Stellen kennzeichnen, für die ChatGPT zur Beantwortung herangezogen wurde, und sie mit einem Link zum Chat-Verlauf versehen. Der entsprechende Link muss auch im Quellenverzeichnis aufgeführt werden, einschliesslich einer Rechtfertigung für die Verwendung von ChatGPT, wie im vorherigen Punkt beschrieben.
- Bei jeder verwendeten Quelle (Internetseiten, Bücher, KI-Tools wie ChatGPT usw.) musst du nicht nur die Quelle angeben, sondern auch erklären, was du dabei gelernt hast und warum du diese Quelle benötigt hast. Dies gilt besonders für Chatbots und KI-Tools – erkläre, für welchen spezifischen Teil der Aufgabe du Hilfe gesucht hast und wie die erhaltene Information dir geholfen hat, das Problem besser zu verstehen oder zu lösen.