Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2025/26

Übungsblatt 5
Version 1.0

20 Punkte

Ausgabe:	14.01.2026, ca. 12:00 Uhr
Abgabestart:	21.01.2026, 12:00 Uhr
Abgabefrist:	29.01.2026, 06:00 Uhr

Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt jederzeit (auch nachträglich) zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Beachten Sie darüber hinaus die Richtlinien der Fakultät zum Verwenden von Generativer KI [1].

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich jederzeit zum Ausschluss der Erfolgskontrolle führen. Dies bedeutet ausdrücklich, dass auch nachträglich die Punktzahl reduziert werden kann.

Kommunikation und aktuelle Informationen

In unseren *FAQs*[2] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im Wiki[3].

In den *ILIAS-Foren* oder auf *Artemis* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki
[2] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ
[3] https://sdq.kastel.kit.edu/programmieren/

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[4] einsehen.

Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Übungsblatt 5 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 21*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io`, `java.util`, `java.util.regex`, `java.util.stream`, `java.util.function`, `java.time` und `java.time.format`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 140 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln ein.

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Orientieren Sie sich zudem an den Bewertungskriterien im Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.

[4] `https://artemis.cs.kit.edu/`

- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 21.01.2026, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 29.01.2026, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

Wiederverwendung von Lösungen

Falls Sie für die Bearbeitung der Abschlussaufgaben oder Übungsblätter Beispiellösungen aus diesem Semester wiederverwenden, *müssen* Sie in die entsprechenden Klassen "Programmieren-Team" ins Autor-Tag eintragen. Dies ist nötig, um die Checkstyle-Kriterien zu erfüllen.

Aufgabe A: Procrastinot

In dieser Aufgabe soll eine TODO-Anwendung erstellt werden, welche die Planung und Verwaltung von Aufgaben, Termine oder Aktivitäten ermöglicht. Die Anwendung ermöglicht es Benutzern, Aufgaben zu erstellen, zu bearbeiten, zu priorisieren und zu verfolgen. Sie bietet Funktionen wie zum Beispiel das Zuweisen von Prioritäten oder das Erstellen von Unteraufgaben. Die TODO-Anwendung dient als Hilfsmittel zur persönlichen oder beruflichen Organisation, um Aufgaben effizient zu verwalten und den Überblick zu behalten.

A.1 Konzepte

Die folgenden Konzepte sind Teil der TODO-Anwendung und müssen für die Nutzer implementiert werden. Achten Sie besonders auf eine sinnvolle und saubere Modellierung. Da Sie in einer zukünftigen Erweiterung planen, die textuelle Schnittstelle der Anwendung durch eine grafische Oberfläche zu ersetzen, müssen Sie zudem auf die Trennung der Belange achten!

A.1.1 Aufgaben

Aufgaben sind die zentralen Elemente der Anwendung. Eine Aufgabe ist zu Beginn offen und kann dann durch den Nutzer als erledigt markiert werden. Nutzer können Aufgaben anlegen, als erledigt markieren, löschen und wiederherstellen (siehe Abbildung A.1). Aufgaben können auch andere Aufgaben als Unteraufgaben hinzugefügt werden. Auch die Unteraufgaben können dementsprechend wieder Unteraufgaben beinhalten.

Eine Aufgabe hat eine eindeutige Identität und einen Namen (doppelte Namen sind also erlaubt). Zudem kann eine Aufgabe jeweils optional eine Frist, eine Priorität und beliebig viele Schlagworte haben.

Wird eine Aufgabe erstellt, ist sie zu Beginn keiner Liste und keiner anderen Aufgabe zugewiesen. Dies kann dann der Benutzer jedoch später nach Belieben tun. Aufgaben, die keiner anderen Aufgabe als Unteraufgabe zugewiesen sind, sind *übergeordnete* Aufgaben.

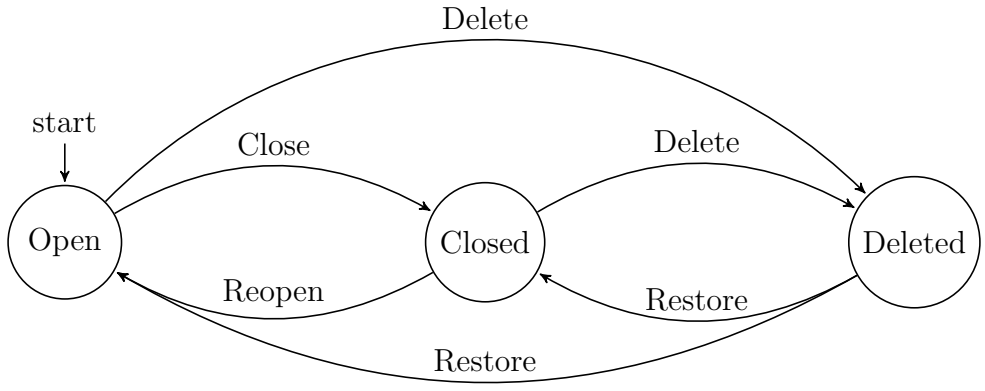

Abbildung A.1: Mögliche Zustände einer Aufgabe.

Aufgaben werden auf der Konsole zeilenweise ausgegeben:

Beispielinteraktion

```
- [ ] Paper [HI]: (diss, research, other) --> 2023-12-30
- [x] Name [MD]: (other)
- [x] Name [LO]: --> 2023-09-05
- [x] FixBug: (coding) --> 2023-09-05
```

Ist kein Schlagwort vorhanden, werden die runden Klammern nicht angezeigt. Ist keine Frist vorhanden, wird auch der Pfeil weggelassen. Fehlt sowohl Frist, als auch Schlagworte, wird nach der Priorität nichts mehr ausgegeben. Ist die Aufgabe erledigt, wird sie durch die Zeichenkette `[x]` gekennzeichnet, ansonsten durch die Zeichenkette `[ ]`.

Wichtig: Bei der Ausgabe werden die übergeordneten Aufgaben direkt ausgegeben und alle anderen Aufgaben, nachdem ihre Elternausgaben ausgegeben wurden. Jede Unteraufgabe ist zwei Leerzeichen tiefer eingerückt als die entsprechende Elternaufgabe:

Beispielinteraktion

```
- [ ] Parent [MD]: (coding)
  - [ ] Child1 [MD]: (issue)
    - [ ] ChildofChild1 [MD]: (issue)
  - [ ] Child2 [MD]: (bug)
```

Werden mehrere Aufgaben ausgegeben, findet eine Sortierung nach zwei Kriterien statt. Zuerst wird absteigend nach der Priorität sortiert (`HI` >`MD` > `LO` > keine Priorität). Danach wird nach Einfügereihenfolge der Aufgaben sortiert. Übergeordnete Aufgaben werden direkt nach diesen Kriterien sortiert ausgegeben. Zudem werden jeweils alle Unteraufgaben einer Aufgabe auch nach diesen Kriterien sortiert ausgegeben.

Beispielinteraktion

```
- [ ] Parent1 [HI]
  - [ ] Child1 [LO]
  - [ ] Child1
- [ ] Parent1 [MD]
- [ ] Parent1 [LO]
```

A.1.2 Listen

Listen dienen der Gruppierung von Aufgaben. Eine Liste hat einen eindeutigen Namen und kann beliebig viele Aufgaben enthalten. Die Aufgaben der Liste stehen aber abgesehen von der Reihenfolge in keiner weiteren Relation. Wird eine Liste erstellt, ist die zuerst leer. Dann müssen der Liste Aufgaben hinzugefügt werden. Auch Listen können, wie Aufgaben, mit Schlagwörtern markiert werden.

A.1.3 Priorität

Die Priorität markiert den Grad der Dringlichkeit oder Wichtigkeit einer Aufgabe. Sie ermöglicht es den Benutzern, ihre Aufgaben nach ihrer Bedeutung zu ordnen und zu priorisieren. Es sollen drei Dringlichkeitsstufen unterstützt werden: Hoch, mittel und niedrig.

A.1.4 Schlagworte

Ein Schlagwort ist ein eindeutiger Name, die einer Aufgabe zugewiesen wird, um sie zu kennzeichnen oder bestimmten Kategorien zuzuordnen. Auf diese Weise können Benutzer Aufgaben mit ähnlichen Eigenschaften oder Themen leicht filtern, organisieren oder suchen.

A.1.5 Fristen

Eine Frist bezieht sich auf ein Datum, bis zu der eine Aufgabe abgeschlossen sein soll. Es ermöglicht den Benutzern, ein bestimmtes Limit für die Erledigung einer Aufgabe festzulegen und eine zeitliche Orientierung zu haben.

A.2 Platzhalter

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen (`<` und `>`) für Platzhalter stehen, welche bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine spitzen Klammern.

`<name>`: eine Zeichenkette die keine Zeilenumbrüche, oder Leerzeichen enthält.

`<date>`: eine Datumsangabe als Zeichenkette im Format JJJJ-MM-TT, z.B. `2023-05-24`.

`<priority>`: die textuelle Repräsentation der Priorität (`HI`, `MD` oder `LO`) oder ein leere Zeichenkette falls keine Priorität vergeben werden soll.

`<id>`: eine eindeutige Ganzzahl die nur für eine Aufgabe vergeben wird (beginnend bei 1 und dann aufsteigend).

`<tag>`: eine eindeutige, nichtleere Zeichenkette die nur aus Buchstaben (A-Z, a-z) und Zahlen besteht.

`<list>`: eine eindeutige, nichtleere Zeichenkette die nur aus Buchstaben (A-Z, a-z) besteht.

A.3 Befehle

Nach dem Start nimmt Ihr Programm über die Konsole die nachfolgenden Befehle unter Verwendung der Standardeingabe entgegen, welche im Folgenden näher spezifiziert werden. Alle Befehle werden auf dem aktuellen Zustand des Programms ausgeführt. Nach Abarbeitung einer Eingabe wartet Ihr Programm auf weitere Eingaben, bis das Programm irgendwann durch die Eingabe der Zeichenfolge `quit` beendet wird.

Achten Sie darauf, dass durch die Ausführung der folgenden Befehle die gegebene Spezifikation nicht verletzt werden und geben Sie in diesen Fällen immer eine aussagekräftige Fehlermeldung aus. Wenn die Benutzereingabe nicht dem vorgegebenen Format entspricht, ist auch eine Fehlermeldung auszugeben. Nach der Ausgabe einer Fehlermeldung soll das Programm wie erwartet fortfahren und wieder auf die nächste Eingabe warten. Jede Fehlermeldung muss mit `ERROR:` gefolgt von einem Leerzeichen beginnen und darf keine Zeilenumbrüche enthalten. Den weiteren Text der Fehlermeldung dürfen Sie frei wählen, er sollte jedoch sinnvoll sein.

A.3.1 Der `add`-Befehl

Dieser Befehl fügt eine neue übergeordnete Aufgabe hinzu.

Eingabeformat: `add <name> [priority] [date]`

Ausgabeformat: Im Erfolgsfall wird `added <id>: <name>` ausgegeben.

Wichtig: Hierbei sind die durch `[ ]`-Platzhalter markierten Argumente optional, das heißt der Befehl ist, unabhängig davon, ob die Argumente gesetzt sind, valide. So kann der Befehl zum Beispiel ohne Priorität aufgerufen werden, aber auch ohne Priorität und ohne Datum.

A.3.2 Der `add-list`-Befehl

Dieser Befehl fügt eine neue, leere Liste hinzu.

Eingabeformat: `add-list <list>`

Ausgabeformat: Im Erfolgsfall wird `added <list>` ausgegeben.

A.3.3 Der `tag`-Befehl für Aufgaben

Dieser Befehl markiert eine Aufgabe mit einem Schlagwort. Der Befehl ist nur valide, falls die Aufgabe mit diesem Schlagwort bisher noch nicht markiert wurde.

Eingabeformat: `tag <id> <tag>`

Ausgabeformat: Im Erfolgsfall wird `tagged <name> with <tag>` ausgegeben.

A.3.4 Der `tag`-Befehl für Listen

Dieser Befehl markiert eine Liste mit einem Schlagwort. Der Befehl ist nur valide, falls die Liste mit diesem Schlagwort bisher noch nicht markiert wurde.

Eingabeformat: `tag <list> <tag>`

Ausgabeformat: Im Erfolgsfall wird `tagged <list> with <tag>` ausgegeben.

A.3.5 Der `assign`-Befehl für Aufgaben

Dieser Befehl fügt einer Aufgabe eine andere Aufgabe als Unteraufgabe hinzu. War die Aufgabe, welche als Unteraufgabe hinzugefügt wird, bisher Unteraufgabe einer anderen Aufgabe, wird sie dort entfernt. Eine Aufgabe kann also immer nur bei einer anderen Aufgabe untergeordnet sein.

Eingabeformat: `assign <id> <id>` (die erste ID ist die Aufgabe, welche der zweiten hinzugefügt wird)

Ausgabeformat: Im Erfolgsfall wird `assigned <name> to <name>` ausgegeben.

Dabei ist die erste ID und der erste Name für die Unteraufgabe und die zweite ID bzw. der zweite Name für die Elternaufgabe.

A.3.6 Der `assign`-Befehl für Listen

Dieser Befehl fügt einer Liste eine Aufgabe hinzu. Die Aufgabe darf dabei nicht in der Liste schon enthalten sein. Hat die Aufgabe Unteraufgaben, zählen auch diese nach dem Hinzufügen als Teil der Liste. Eine Aufgabe kann Teil von mehreren Listen sein.

Eingabeformat: `assign <id> <list>`

Ausgabeformat: Im Erfolgsfall wird `assigned <name> to <list>` ausgegeben.

A.3.7 Der `toggle`-Befehl

Dieser Befehl markiert eine Aufgabe als erledigt, falls sie noch nicht als erledigt markiert wurde. Dabei werden auch alle Unteraufgabe (direkt und indirekt) als erledigt markiert. Falls die Aufgabe bereits zuvor als erledigt markiert wurde, wird die Markierung bei der Aufgabe und allen Unteraufgaben (direkt und indirekt) entfernt.

Eingabeformat: `toggle <id>`

Ausgabeformat: Im Erfolgsfall wird `toggled <name> and <no> subtasks` ausgegeben.

`no` ist hierbei die Anzahl der direkten und indirekten Unteraufgaben der Aufgabe.

A.3.8 Der `change-date`-Befehl

Dieser setzt bzw. ändert die Frist einer Aufgabe. Eine bestehende Frist wird dabei gegebenenfalls verworfen.

Eingabeformat: `change-date <id> <date>`

Ausgabeformat: Im Erfolgsfall wird `changed <name> to <date>` ausgegeben.

A.3.9 Der `change-priority`-Befehl

Dieser setzt bzw. ändert die Priorität einer Aufgabe. Eine bestehende Priorität wird dabei gegebenenfalls verworfen. Sollte keine Priorität übergeben werden, so entfernt der Befehl die Priorität der Aufgabe.

Eingabeformat: `change-priority <id> [priority]`

Ausgabeformat: Im Erfolgsfall wird `changed <name> to <priority>` ausgegeben. Sollte keine Priorität angeben worden sein, so wird im Erfolgsfall `changed <name> to NONE` ausgegeben.

A.3.10 Der `delete`-Befehl

Dieser Befehl löscht eine Aufgabe. Dabei werden auch alle Unteraufgabe (direkt und indirekt) gelöscht

Eingabeformat: `delete <id>`

Ausgabeformat: Im Erfolgsfall wird `deleted <name> and <no> subtasks` ausgegeben.

`no` ist hierbei die Anzahl der direkten und indirekten Unteraufgaben der gelöschten Aufgabe.

A.3.11 Der `restore`-Befehl

Dieser Befehl stellt eine gelöschte Aufgabe wieder her. Dabei werden auch alle gelöschten Unteraufgaben (direkt und indirekt) wiederhergestellt, egal wann diese gelöscht wurden. Nach dem Wiederherstellen hat die wiederhergestellte Aufgabe dann ihre ursprüngliche Zugehörigkeit, falls es sich um eine Unteraufgabe handelt und die übergeordnete Aufgabe nicht gelöscht wurde. Auch die Zugehörigkeit zu Listen wird wiederhergestellt. Falls die übergeordnete Aufgabe gelöscht wurde, ist die Aufgabe nach dem Wiederherstellen eine übergeordnete Aufgabe.

Eingabeformat: `restore <id>`

Ausgabeformat: Im Erfolgsfall wird `restored <name> and <no> subtasks` ausgegeben.

`no` ist hierbei die Anzahl der direkten und indirekten Unteraufgaben der wiederhergestellten Aufgabe.

A.3.12 Der `show`-Befehl

Dieser Befehl zeigt eine beliebige Aufgabe (inklusive aller direkten und indirekten Unteraufgaben) an.

Eingabeformat: `show <id>`

Ausgabeformat: Im Erfolgsfall wird die Aufgabe wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.13 Der `todo`-Befehl

Dieser Befehl zeigt alle Aufgaben an, welche noch nicht als erledigt markiert wurden. Hat eine Aufgabe, die nicht erledigt wurde, (direkte oder indirekte) Unteraufgaben, welche bereits erledigt wurden, werden diese Unteraufgaben dann nicht angezeigt.

Eingabeformat: `todo`

Ausgabeformat: Im Erfolgsfall werden alle offenen Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.14 Der `list`-Befehl

Dieser Befehl zeigt alle Aufgaben (inklusive aller direkten und indirekten Unteraufgaben) an, welche Teil einer bestimmten Liste sind. Dabei werden Unteraufgaben nicht erneut gelistet, falls sie bereits durch eine übergeordnete Aufgabe in der Ausgabe enthalten sind.

Eingabeformat: `list <list>`

Ausgabeformat: Im Erfolgsfall werden die Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.15 Der `tagged-with`-Befehl

Dieser Befehl zeigt alle Aufgaben (inklusive aller direkten und indirekten Unteraufgaben) an, welche mit einem bestimmten Schlagwort versehen wurden. Dabei werden Unteraufgaben, welche den Tag enthalten, nicht erneut gelistet, falls sie bereits durch eine übergeordnete Aufgabe in der Ausgabe enthalten sind.

Eingabeformat: `tagged-with <tag>`

Ausgabeformat: Im Erfolgsfall werden die Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.16 Der `find`-Befehl

Dieser Befehl zeigt alle Aufgaben (inklusive aller direkten und indirekten Unteraufgaben) an, bei denen der Name eine gewünschte Zeichenkette enthält. Der Name muss also nicht exakt der Zeichenkette entsprechen, die Zeichenkette muss lediglich darin vorkommen. Dabei werden Unteraufgaben nicht erneut gelistet, falls sie bereits durch eine übergeordnete Aufgabe in der Ausgabe enthalten sind.

Eingabeformat: `find <name>` (`<name>` ist hier die gewünschte Zeichenkette)

Ausgabeformat: Im Erfolgsfall werden die Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.17 Der `upcoming`-Befehl

Dieser Befehl zeigt alle Aufgaben (inklusive aller direkten und indirekten Unteraufgaben) an, bei denen die Frist innerhalb der nächsten 7 Tage nach einem bestimmten Datum fällig ist. Es zählt der spezifizierte Tag und die nächsten sechs folgenden Tage. Dabei werden Unteraufgaben nicht erneut gelistet, falls sie bereits durch eine übergeordnete Aufgabe in der Ausgabe enthalten sind.

Eingabeformat: `upcoming <date>`

Ausgabeformat: Im Erfolgsfall werden die fälligen Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.18 Der `before`-Befehl

Dieser Befehl zeigt alle Aufgaben (inklusive aller direkten und indirekten Unteraufgaben) an, bei denen die Frist innerhalb vor oder an einem bestimmen Datum liegt. Dabei zählen also auch Aufgabe, die direkt am Tag der Datumsangabe fällig sind. Dabei werden Unteraufgaben nicht erneut gelistet, falls sie bereits durch eine übergeordnete Aufgabe in der Ausgabe enthalten sind.

Eingabeformat: `before <date>`

Ausgabeformat: Im Erfolgsfall werden die Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.19 Der `between`-Befehl

Dieser Befehl zeigt alle Aufgaben (inklusive aller direkten und indirekten Unteraufgaben) an, bei denen die Frist zwischen zwei Datumsangaben liegt. Dabei zählen auch Aufgabe, die direkt am Tag der Datumsangaben fällig sind. Dabei werden Unteraufgaben nicht erneut gelistet, falls sie bereits durch eine übergeordnete Aufgabe in der Ausgabe enthalten sind.

Eingabeformat: `between <date> <date>`

Ausgabeformat: Im Erfolgsfall werden die Aufgaben wie in Unterunterabschnitt A.1.1 beschrieben ausgegeben.

A.3.20 Der `duplicates`-Befehl

Dieser Befehl findet für alle Aufgaben Duplikate, wobei zwei Aufgaben genau dann Duplikate sind, wenn beide folgenden Bedingungen erfüllt sind:

1. der Name von beiden ist identisch
2. die Frist ist identisch, falls beide eine Frist haben ODER maximal eine Aufgabe eine Frist hat.

Eingabeformat: `duplicates`

Ausgabeformat: Im Erfolgsfall wird `Found <no> duplicates: <dup>` ausgegeben. Hierbei ist `<no>` die Anzahl der Aufgaben, die mindestens ein Duplikat haben und `dup` die Liste der Identitäten aller Aufgaben, die mindestens ein Duplikat haben (aufsteigend sortiert).

Beispiel: `Found 3 duplicates: 2, 8, 17`

A.4 Beispielinteraktion

Der Beispielablauf zeigt die Verwendung nach Start des Programmes. Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Nutzereingaben sind mit dem Zeichen `>` gekennzeichnet. Die Zeichen `%>` (Prozent-Zeichen und Größer-Zeichen gefolgt von einem Leerzeichen) stellt die Kommandozeile dar. Der Name des Programms dient nur als Beispiel und ist nicht vorgeschrieben.

Beispielinteraktion

```
1  >add Final1 HI 2023-07-31
2  added 1: Final1
3  >add complete_commands MD 2023-07-26
4  added 2: complete_commands
5  >add command_add
6  added 3: command_add
7  >toggle 3
8  toggled command_add and 0 subtasks
9  >add command_assign MD
10 added 4: command_assign
11 >add command_show 2023-07-25
12 added 5: command_show
13 >assign 3 2
14 assigned command_add to complete_commands
15 >toggle 4
16 toggled command_assign and 0 subtasks
17 >assign 4 2
18 assigned command_assign to complete_commands
19 >add-list timeCommands
20 added timeCommands
```

Beispielinteraktion

```
>assign 3 timeCommands
assigned command_add to timeCommands
>add command_change-date
added 6: command_change-date
>assign 6 2
assigned command_change-date to complete_commands
>assign 6 timeCommands
assigned command_change-date to timeCommands
>tag 2 incomplete
tagged complete_commands with incomplete
>add check_for_leap_year 2024-02-29
added 7: check_for_leap_year
>assign 7 3
assigned check_for_leap_year to command_add
>assign 2 1
assigned complete_commands to Final1
>show 1
- [ ] Final1 [HI]: --> 2023-07-31
  - [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
    - [x] command_assign [MD]
    - [x] command_add
      - [ ] check_for_leap_year: --> 2024-02-29
    - [ ] command_change-date
>toggle 5
toggled command_show and 0 subtasks
>change-date 6 2023-07-27
changed command_change-date to 2023-07-27
>toggle 6
toggled command_change-date and 0 subtasks
>upcoming 2023-07-26
- [ ] Final1 [HI]: --> 2023-07-31
  - [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
    - [x] command_assign [MD]
    - [x] command_add
      - [ ] check_for_leap_year: --> 2024-02-29
    - [x] command_change-date: --> 2023-07-27
>add variations_of_optional_parameters LO
added 8: variations_of_optional_parameters
>assign 8 3
assigned variations_of_optional_parameters to command_add
>list timeCommands
- [x] command_add
  - [ ] variations_of_optional_parameters [LO]
  - [ ] check_for_leap_year: --> 2024-02-29
- [x] command_change-date: --> 2023-07-27
>delete 3
deleted command_add and 2 subtasks
>list timeCommands
- [x] command_change-date: --> 2023-07-27
>show 1
```

Beispielinteraktion

```
- [ ] Final1 [HI]: --> 2023-07-31
  - [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
    - [x] command_assign [MD]
    - [x] command_change-date: --> 2023-07-27
>restore 3
restored command_add and 2 subtasks
>tagged-with incomplete
- [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
  - [x] command_assign [MD]
  - [x] command_change-date: --> 2023-07-27
  - [x] command_add
    - [ ] variations_of_optional_parameters [LO]
    - [ ] check_for_leap_year: --> 2024-02-29
>toggle 7
toggled check_for_leap_year and 0 subtasks
>todo
- [ ] Final1 [HI]: --> 2023-07-31
  - [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
    - [x] command_add
      - [ ] variations_of_optional_parameters [LO]
>add check_for_leap_year
added 9: check_for_leap_year
>assign 9 6
assigned check_for_leap_year to command_change-date
>duplicates
Found 2 duplicates: 7, 9
>before 2023-07-30
- [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
  - [x] command_assign [MD]
  - [x] command_change-date: --> 2023-07-27
    - [ ] check_for_leap_year
  - [x] command_add
    - [ ] variations_of_optional_parameters [LO]
    - [x] check_for_leap_year: --> 2024-02-29
- [x] command_show: --> 2023-07-25
>change-priority 8 HI
changed variations_of_optional_parameters to HI
>tag timeCommands confusing
tagged timeCommands with confusing
>find s
- [ ] complete_commands [MD]: (incomplete) --> 2023-07-26
  - [x] command_assign [MD]
  - [x] command_change-date: --> 2023-07-27
    - [ ] check_for_leap_year
  - [x] command_add
    - [ ] variations_of_optional_parameters [HI]
    - [x] check_for_leap_year: --> 2024-02-29
- [x] command_show: --> 2023-07-25
>toggle 1
toggled Final1 and 7 subtasks
```