Anwender:innen-Dokumentation

computor – Bedienungsanleitung für Lernende, Tutor:innen und Lehrende

computor – netidee-Projekt 8012, TU Graz
Dokumentation der aktuellen Version · Lizenz CC BY 4.0

Überblick **3**
Was ist es 3
Für wen ist es und wem hilft es wodurch 3
Wie funktioniert es 3
Aufbau dieser Dokumentation 3
Teil A: Für Studierende **4**
Kurse 4
Einheiten und Lesematerial 7
Aufgaben 8
Lernen mit Computor 12
Teil B: Für Tutor:innen und Lehrende **14**
Die Tutor-Ansicht 14
Eine Aufgabe bewerten 15
Leitfaden für Lehrende: Kurs-Dashboard 17
Teil C: Kurse und Aufgaben anlegen **22**
Computor Testing Framework – Anleitung 22

Überblick

Was ist es

Die Plattform computor ist eine quelloffene Software für Programmierkurse. Studierende bearbeiten Übungsaufgaben in Visual Studio Code, entweder im Browser auf computor.at oder lokal auf dem eigenen Rechner, und lassen ihre Lösung von automatischen Tests prüfen. Bei Fragen hilft die KI-Tutorin Lena, die so angeleitet ist, dass sie Hinweise statt fertiger Lösungen gibt. Lehrende und Tutor:innen sehen Fortschritt, Testläufe und häufige Fehler ihrer Studierenden und bewerten Abgaben. Diese Anleitung erklärt die Bedienung für alle drei Rollen.

Für wen ist es und wem hilft es wodurch

Studierende erhalten nach jedem Testlauf rasch Rückmeldung, ob ihr Programm das Verlangte leistet. Wer computor außerhalb einer Lehrveranstaltung ausprobieren möchte, kann die drei öffentlichen Kurse der Reihe „Data Science mit Python“ unter computor.at/learn ohne Konto lesen und nach einer Registrierung bearbeiten. Für Tutor:innen und Lehrende entfällt das Einsammeln der Abgaben, die automatischen Tests nehmen ihnen einen Teil der Prüfarbeit ab, und sie können früh erkennen, wo Studierende Schwierigkeiten haben. Lehrende, die eigene Kurse anlegen wollen, finden in Teil C dieser Dokumentation, wie Aufgaben und ihre automatischen Tests beschrieben werden.

Wie funktioniert es

Alle Studierenden erhalten je Kurs eine eigene versionierte Ablage (ein Git-Repository), in der jede Aufgabe einen Ordner bildet. Studierende öffnen eine Aufgabe in der Student-Ansicht von VS Code, schreiben ihren Code und starten einen Testlauf, der den aktuellen Stand hochlädt und auf dem Server prüft; das Testergebnis erscheint direkt neben der Aufgabe. Zur Bewertung reichen Studierende ihre Lösung in einem eigenen Schritt ausdrücklich ein. Tutor:innen öffnen dieselbe Aufgabe in ihrer Tutor-Ansicht, sehen Code, Testergebnisse und Nachrichten und vergeben eine Bewertung. Lehrende finden in der Webanwendung von computor Kursstatistiken und die typischen Fehler je Aufgabe.

Aufbau dieser Dokumentation

Teil A beschreibt die Arbeit als Studierende:r, Teil B die Arbeit als Tutor:in oder Lehrende:r und Teil C das Anlegen von Aufgaben; das vollständige Referenzhandbuch der VS-Code-Erweiterung liegt im Repository. Die Hilfeseiten für Studierende und Tutor:innen sind auch in der Erweiterung selbst über den Befehl **Computor: Willkommen und Hilfe** erreichbar (englische Oberfläche: *Welcome and Help*), den man über die Befehlspalette von VS Code (Strg+Umschalt+P) aufruft. Diese Dokumentation gibt es auch auf Englisch. Die Anleitungen beschreiben die aktuelle Version und werden in den Repositorys computor-vscode, computor-backend und data-science-python gemeinsam mit dem Code gepflegt. Diese Dokumentation steht unter CC BY 4.0.

Teil A: Für Studierende

Kurse

Quelle: https://github.com/computor-org/computor-vscode/blob/main/docs/help/student-course_de.md

Ein Kurs umfasst alles, wofür du in einer Lehrveranstaltung angemeldet bist: ihre Einheiten, ihre Aufgaben und das Repository, in dem deine Arbeit liegt. Er bildet die oberste Zeile in der Ansicht „Kurse" von Computor, und das meiste, was diese Seite beschreibt, gilt für alles, was darunter liegt.

```
📖 Programmieren in der Physik
├── ⚫ Woche 1 – Einführung
│   ├── 🟦 Übung 1
│   └── 🟦 Hausübung 1
└── 🟦 Abschlussprojekt
```

Mit einem Klick auf eine Zeile klappst du sie auf. Die Gliederung legen deine Lehrenden fest; du selbst kannst sie nicht ändern.

Deine Arbeit liegt in einem einzigen Repository

Jede Aufgabe eines Kurses ist ein Ordner in einem **einzigen Repository**, das dir gehört. Das solltest du als Erstes verstehen, weil sich fast alles andere daraus ergibt:

- Wenn du eine Datei speicherst, änderst du sie **nur auf deinem Computer**.
- **Sichern (Commit)** hält deine Änderungen im Repository fest und lädt sie auf den Server hoch. Das ist deine Sicherungskopie, und genau das sehen deine Lehrenden.
- **Abgeben** ist ein eigener, bewusster Schritt, mit dem du eine Aufgabe zur Bewertung einreichst. Ein Commit ist keine Abgabe.

Da das Repository den ganzen Kurs enthält, sichert *Kurs sichern (Commit)* an der Kurszeile in einem Schritt alles, woran du irgendwo im Kurs gearbeitet hast. Die entsprechende Aktion an einer einzelnen Aufgabe tut dasselbe nur für deren Ordner.

Einrichten musst du das Repository normalerweise nie selbst. Es wird angelegt und heruntergeladen, wenn du den Kurs zum ersten Mal öffnest, und erneut, wenn die Erweiterung startet.

Die Markierungen in einer Zeile

Kleine Markierungen in den Zeilen zeigen dir, wie es um deine Arbeit steht. Sie erscheinen an den Aufgaben, an den Einheiten darüber und an der Kurszeile selbst:

Markierung	Bedeutung
●	Du hast Änderungen, die noch nicht gesichert (committet) sind
↑	Gesichert, aber noch nicht auf den Server hochgeladen
⚠	Hoch- oder Herunterladen schlägt fehl — siehe *Anmeldung am Repository reparieren*

An der Kurszeile trägt das `↑` eine Zahl (`↑3`), die angibt, wie viele Commits noch auf das Hochladen warten. Eine Zeile ohne Markierung ist gesichert und hochgeladen, deine Arbeit also in Sicherheit.

Ein `🔔` mit einer Zahl steht für ungelesene Nachrichten.

Aktionen am Kurs

Klicke mit der rechten Maustaste auf die Kurszeile. Die Symbole in der Zeile sind Abkürzungen zu denselben Aktionen, und Einträge, die für deinen Kurs nicht in Frage kommen, werden ausgeblendet, statt angezeigt und dann verweigert zu werden.

Kurs sichern (Commit)

Sichert **deine gesamte** Arbeit im Kurs und lädt sie in einem Schritt hoch. Verwende diese Aktion am Ende einer Arbeitssitzung, bevor du den Arbeitsbereich schließt, oder wann immer du deine Arbeit sicher auf dem Server haben willst.

Repository aus Kursvorlage aktualisieren

Bringt dein Repository auf den Stand des Kursmaterials und **stellt jede Datei wieder her, die du gelöscht hast**.

Selbst ausführen musst du diese Aktion selten: Freigegebene Aktualisierungen kommen von allein an, wenn der Arbeitsbereich startet und wenn die Kursansicht neu geladen wird. Mit dieser Aktion stößt du denselben Abgleich von Hand an, wenn du nicht warten willst.

Dabei geschehen zwei Dinge:

1. Neues und geändertes Material, das deine Lehrenden freigegeben haben, wird in dein Repository übernommen.
2. Jede Datei, die im Kursmaterial vorhanden ist, in deinem Ordner aber **nicht**, wird wiederhergestellt. Dateien, die du noch hast, bleiben genau so, wie sie sind — deine eigene Arbeit wird nie überschrieben.

Mit diesem zweiten Schritt machst du einen Fehler rückgängig. Wenn du eine Datendatei, eine Abbildung oder eine Ausgangsdatei verdorben hast, **lösche sie** und führe diese Aktion aus: Du bekommst eine frische Kopie zurück.

> **Kopiere die Datei zuerst**, wenn du behalten willst, was du geschrieben hast. Rechtsklick auf die Datei → *Datei kopieren…*, danach löschst du das Original.

Nachrichten anzeigen

Kursankündigungen enthalten Neuigkeiten, Termine und Klarstellungen der Lehrenden. Du kannst sie hier lesen; Ankündigungen veröffentlichen dürfen nur Lehrende und Personen mit weitergehenden Rechten.

Mit Frage zum Kurs stellen öffnest du deine Unterhaltung mit dem Lehrteam zu dem Kurs, dessen Hilfe du geöffnet hast. Für eine Frage zu einer bestimmten Übung verwendest du stattdessen *Nachrichten anzeigen* an der betreffenden Aufgabe.

Kursaufgaben exportieren

Packt die Aufgaben, die du auf deinem Rechner hast, in eine einzige ZIP-Datei, damit dir deine Arbeit bleibt, wenn der Kurs endet und dein Zugriff auf das Repository erlischt. Den Aufbau des Archivs wählst du selbst:

- ***Baum*** — bildet die Kursgliederung nach, die Ordner tragen die Titel der Inhalte
- ***Flach*** — ein Ordner je Aufgabe, benannt nach der Kennung ihres Beispiels

Im Browser wird das Archiv in deinem Arbeitsbereich unter `exports/` gespeichert und anschließend auf deinen Computer heruntergeladen. Blockiert dein Browser den Download, verwende die Schaltfläche *Erneut herunterladen* auf der Seite, die sich öffnet, oder klicke die Datei im Explorer mit der rechten Maustaste an und wähle *Herunterladen…* (in englischem VS Code *Download…*). In der Desktop-Version von VS Code wirst du gefragt, wo die Datei gespeichert werden soll.

Außerhalb des Arbeitsbereichs arbeiten

Die Kursseite auf der Website (Kurs öffnen, dann *Check access* unter *Your repository*) gibt dir einen fertigen `git clone`-Befehl zum Einfügen, der dein persönliches Zugriffstoken enthält. Damit erhältst du ein **vollständiges Git-Repository** auf deinem eigenen Computer: Du kannst mit diesem Token aus jedem Git-Programm Aktualisierungen holen und deine Arbeit hochladen. Das Token wirkt wie ein Passwort, also behandle es auch so. Es bleibt eigenständig gültig und ist nicht die Zugangsberechtigung, die der Arbeitsbereich verwaltet.

Das unterscheidet sich von *Kursaufgaben exportieren* oben: Der Export ist eine schlichte ZIP-Datei mit den Dateien, ohne Git, gedacht, um deine Arbeit über den Kurs hinaus aufzubewahren. Der Klon bleibt mit dem Kurs verbunden. So oder so bleibt der Arbeitsbereich der unterstützte Ort zum Arbeiten — getestet und abgegeben wird dort.

Anmeldung am Repository reparieren

Erneuert die Zugangsdaten, die dein Repository verwendet. Setze diese Aktion ein, wenn das Hochladen plötzlich fehlschlägt und die Kurszeile `⚠` zeigt. Deine Dateien bleiben dabei unberührt.

Repository einrichten

Legt dein Repository an und lädt es herunter. **Normalerweise brauchst du das nie** — es geschieht von selbst. Der Eintrag erscheint nur, solange es vom Kurs noch keine lokale Kopie gibt, etwa wenn du in einen anderen Arbeitsordner gewechselt bist oder der erste Versuch fehlgeschlagen ist.

Kursvorlage herunterladen

Für Kurse, deren Material als einfacher Download statt als Repository ausgegeben wird: Lädt das Material als ZIP-Datei herunter oder entpackt es in einen Ordner deiner Wahl. Das Material wird auch automatisch geholt, wenn du einen solchen Kurs zum ersten Mal öffnest; diese Aktion brauchst du, wenn du eine zweite Kopie an einem anderen Ort haben willst.

Wenn etwas nicht stimmt

- **Eine Zeile zeigt** ⚠ — führe *Anmeldung am Repository reparieren* am Kurs aus.
- **Neues Material ist nicht angekommen** — führe *Repository aus Kursvorlage aktualisieren* aus.
- **Du hast etwas gelöscht, das du brauchst** — siehe dieselbe Aktion; gerade das Löschen der Datei sorgt dafür, dass eine frische Kopie zurückkommt.
- **Die Baumansicht wirkt veraltet** — klappe den Kurs zu und wieder auf; dadurch wird er neu vom Server geladen.

Weiter

- Einheiten — wie ein Kurs gegliedert ist, und Lesematerial
- Aufgaben — bearbeiten, testen, abgeben, Dateien

Einheiten und Lesematerial

Quelle: https://github.com/computor-org/computor-vscode/blob/main/docs/help/student-unit_de.md

Eine Einheit fasst zusammengehörige Inhalte innerhalb eines Kurses zusammen, meist eine Woche, ein Thema oder ein Kapitel. Einheiten enthalten Aufgaben, Lesematerial und manchmal weitere Einheiten.

```
📖 Programmieren in der Physik
├── ⚫ Woche 1 – Einführung
│   ├── 📄 Vorlesungsskript
│   ├── 🟦 Übung 1
│   └── 🟦 Hausübung 1
└── ⚫ Woche 2 – Datentypen
```

Eine Einheit wird als Kreis dargestellt, eine Aufgabe als Quadrat und sonstiges Material als Dokument. Die Farbe steht für den **Inhaltstyp**, den deine Lehrenden festgelegt haben — Übung, Hausübung, Prüfung, Projekt —, damit du die Art der Arbeit auf einen Blick erkennst. Welche Farbe was bedeutet, wird für jeden Kurs eigens festgelegt; fährst du mit der Maus darüber, wird dir der Typ mit Namen angezeigt.

Einheiten sind keine eigene Arbeit von dir: An einer Einheitenzeile gibt es nichts zu sichern und nichts abzugeben. Du kannst aber lesen, was zu ihr gehört.

Wählst du den Titel einer Einheit aus, mit der Maus oder mit den Pfeiltasten und anschließend der Eingabetaste, öffnet sich ihre Beschreibung neben deinem Code. Wählst du eine Übung aus, liest du deren Anleitung. Mit dem Pfeil (Chevron) klappst du die untergeordneten Einträge auf oder zu. Beschreibungen verwenden denselben Vorschau-Tab, und das Auswählen einer Einheit richtet kein Repository ein.

Was dir eine Einheit verrät

Fährst du mit der Maus über eine Einheit, siehst du ihren Titel, wie viele Einträge sie enthält, ihren Typ und, falls vorhanden, die Zahl der ungelesenen Nachrichten.

In der Zeile selbst kannst du Folgendes sehen:

- 🔔 3 — ungelesene Nachrichten in dieser Einheit
- ● ↑ ⚠ — den Zustand der Aufgaben darunter, sodass du eine Einheit mit nicht gesicherter Arbeit erkennst, ohne sie aufzuklappen. Die Markierungen bedeuten überall dasselbe; siehe Kurse.

Aktionen an einer Einheit

Beschreibung anzeigen

Öffnet den Text, den deine Lehrenden zu dieser Einheit geschrieben haben, als gestaltete Seite. Der Eintrag erscheint nur, wenn es eine Beschreibung gibt.

Haben deine Lehrenden sie in mehr als einer Sprache verfasst, richtet sich die angezeigte Fassung nach deiner Lesesprache, also nach der Sprache deines Profils oder sonst nach der Anzeigesprache von VS Code; fehlt sie, wird Englisch verwendet. Bei einer Aufgabe kannst du mit **Beschreibung in anderer Sprache anzeigen** die übrigen Fassungen lesen.

Nachrichten anzeigen

Ankündigungen und Diskussionen zu dieser Einheit — Klarstellungen, Hinweise, geänderte Termine. Sie sind enger gefasst als die Nachrichten zum ganzen Kurs, aber weiter als die zu einer einzelnen Aufgabe.

Lesematerial und andere Inhalte

Nicht alles in einem Kurs ist eine Aufgabe. Zeilen, die weder eine Einheit noch eine Aufgabe sind, enthalten Material zum Lesen oder Verwenden: Vorlesungsskripten, Literatur, Daten. Zu ihnen gibt es kein Repository, keine Tests und nichts abzugeben, weshalb sich die Aktionen darauf beschränken, die Beschreibung zu öffnen und Nachrichten zu lesen.

Sieht eine Zeile aus wie etwas, das du abgeben solltest, bietet aber kein *Abgeben* an, dann gehört sie zu diesen Inhalten. Frag in den Kursnachrichten nach, wenn du etwas anderes erwartest; vielleicht bereiten deine Lehrenden sie noch vor.

Verborgene Inhalte

Deine Lehrenden können eine Einheit verbergen, solange sie vorbereitet wird, oder nach einer Prüfung. Eine verborgene Einheit erscheint für dich einfach nicht — du siehst nie eine leere oder ausgegraute Zeile an der Stelle, wo vorher eine war. Dass Inhalte erst im Laufe des Semesters erscheinen, ist normal.

Weiter

- Aufgaben — bearbeiten, testen, abgeben, Dateien
- Kurse — das Repository, Commits und die Markierungen in den Zeilen

Aufgaben

Quelle: https://github.com/computor-org/computor-vscode/blob/main/docs/help/student-assignment_de.md

Eine Aufgabe ist eine Arbeit, die du abgibst: ein Ordner in deinem Kurs-Repository mit der Aufgabenstellung, den Dateien, von denen du ausgehst, und den Dateien, die du schreiben sollst. Diese Seite erklärt, wie du eine Aufgabe bearbeitest und abgibst und was du alles mit den Dateien darin tun kannst.

Was dir die Zeile verrät

```
🟦 Übung 1: Hello World    ● 🔔 1 (2/5) (1/3) 85% 90%
```

Von links nach rechts gelesen:

Teil	Bedeutung
🟦	Der Inhaltstyp, den deine Lehrenden festgelegt haben — Übung, Hausübung, Prüfung. Ein Häkchen oder Kreuz auf dem Symbol zeigt, ob der letzte Test bestanden wurde
● ↑ ⚠	Der Zustand deiner Arbeit: nicht gesichert, nicht hochgeladen, Hochladen schlägt fehl. Siehe Kurse
🔔 1	Ungelesene Nachrichten zu dieser Aufgabe
(2/5)	Testläufe: zwei von fünf erlaubten verbraucht. Nur (2), wenn es keine Obergrenze gibt
(1/3)	Abgaben: eine von drei erlaubten getätigt
85%	Das **Testergebnis** — was die automatischen Tests ergeben haben
90%	Die **Bewertung** — die offizielle Note deiner Lehrenden, sobald bewertet wurde

Die beiden Prozentangaben behalten stets ihren Platz: zuerst das Testergebnis, dann die Bewertung. Hast du eine Bewertung, aber kein Testergebnis, steht an der ersten Stelle `-%`, sodass die zweite Zahl immer die Bewertung ist und nie erraten werden muss.

Fährst du mit der Maus über die Zeile, bekommst du dieselben Angaben in Worten, bei einer Teamaufgabe zusätzlich die Mitglieder deines Teams.

Eine Aufgabe bearbeiten

1. Lies die Aufgabenstellung. Klicke auf den Titel der Übung oder wähle ihn mit den Pfeiltasten aus und drücke die Eingabetaste. Ihre Anleitung öffnet sich neben deinem Code, ohne dass dafür ein Repository eingerichtet wird. Das Vorschausymbol und Rechtsklick → *Anleitung anzeigen* stehen weiterhin zur Verfügung. Mit dem Pfeil (Chevron) klappst du die Dateien der Übung auf, sobald du mit der Arbeit beginnen willst.

2. Schreibe deine Lösung. Klappe die Aufgabe auf, um ihre Dateien zu sehen, und öffne eine davon mit einem Klick. Behalte die Dateien, die die Aufgabenstellung verlangt, unter den verlangten Namen bei — benennst du eine Datei um, nach der die Tests suchen, schlagen sie fehl.

3. Teste. Klicke auf das Becherglas-Symbol oder wähle Rechtsklick → *Aufgabe testen*.

> Ein Test ist kein lokaler Probelauf. Computor speichert deine geöffneten Dateien, sichert sie mit einem Commit, lädt sie hoch und führt die Tests **auf dem Server** aus — ein Testlauf lädt also immer deinen aktuellen Stand hoch und verbraucht immer einen deiner Testläufe, falls die Aufgabe eine Obergrenze hat. Die Ergebnisse erscheinen im Bereich „Testergebnisse"; ein Klick auf einen fehlgeschlagenen Test führt dich zu dem Code, um den es geht.

4. Sichere zwischendurch. Das Commit-Symbol an der Aufgabe speichert diesen Ordner und lädt ihn hoch; *Kurs sichern (Commit)* an der Kurszeile erledigt das für den ganzen Kurs auf einmal. Sichere früh und oft — das ist deine Sicherungskopie, und es kostet nichts. Ein Commit ist keine Abgabe, und niemand bewertet ihn.

5. Gib ab, wenn du so weit bist. Das Upload-Symbol oder Rechtsklick → *Abgeben*. Damit reichst du deine Arbeit bewusst ein; die Zahl der Abgaben ist begrenzt und kann nur steigen.

Computor gibt den Commit ab, der **getestet** wurde, nicht unbedingt deinen neuesten. Unterscheiden sich die beiden — etwa weil der letzte Test nicht laufen konnte —, wirst du darauf hingewiesen, bevor irgendetwas abgegeben wird, sodass deine neueste Arbeit nie stillschweigend bewertet oder stillschweigend übergangen wird.

6. Lies die Rückmeldung. *Testergebnisse anzeigen* für den automatischen Testlauf, *Details anzeigen* für den Verlauf deiner Abgaben und die Bewertung, *Nachrichten anzeigen* für das, was deine Lehrenden oder Tutor:innen geschrieben haben. Der Status *Korrektur nötig* bedeutet, dass Änderungen erwartet werden; *Verbesserung möglich* bedeutet, dass du nachbessern darfst, sofern dir noch Abgaben bleiben.

Testläufe und Abgaben sind begrenzt

Legen deine Lehrenden eine Obergrenze fest, zeigt die Zeile, wie viel dir noch bleibt, und Computor weist dich darauf hin, bevor eine Aktion die Grenze überschreiten würde, nicht erst danach.

Setze Testläufe überlegt ein: Lies die Fehlermeldung, denk über den Code nach und teste erst dann. Spare Abgaben für Arbeit auf, von der du überzeugt bist — sobald die Höchstzahl erreicht ist, kannst du nicht mehr abgeben, und daran ändert auch noch so viel verbleibende Zeit nichts.

Mit Dateien arbeiten

Klappe eine Aufgabe auf, um ihre Dateien zu sehen. Klicke mit der rechten Maustaste auf eine Datei, einen Ordner oder die Aufgabe selbst. Alles, was hier geschieht, geschieht **nur auf deinem Computer** — auf den Server gelangt nichts, bevor du es sicherst (Commit).

Anlegen. *Neue Datei…* und *Neuer Ordner…* an der Aufgabe oder an einem Ordner. Die neue Datei öffnet sich sofort.

Kopieren und Verschieben. *Datei kopieren…* und *Datei verschieben…* bieten nur Ordner innerhalb derselben Aufgabe an, damit Arbeit nicht versehentlich in einer anderen Übung landet. Ist der Name schon vergeben, wirst du gefragt, ob du *Beide behalten* oder *Überschreiben* willst. *Ausschneiden*, *Kopieren* und *Einfügen* sind die Alternative in zwei Schritten; *Einfügen* erscheint, sobald etwas zum Einfügen bereitliegt.

Umbenennen und Duplizieren. *Umbenennen…* vergibt einen neuen Namen; *Duplizieren* legt eine Kopie neben dem Original an.

Löschen. *Datei löschen* und *Ordner löschen*, stets nach einer Rückfrage. Gehörte die Datei zu einer Aufgabe, die du bereits abgegeben hast, sagt dir die Rückfrage das — löschen darfst du sie trotzdem, aber nicht, ohne es zu wissen. Dieselbe Warnung erscheint, wenn du eine solche Datei verschiebst oder umbenennst.

Löschst du eine Datei an anderer Stelle — im Explorer oder mit `rm` in einem Terminal —, lässt sich das nicht vorher abfangen. Computor bemerkt es aber unmittelbar danach und bietet dir, falls die Datei Teil einer Abgabe war, an, deine zuletzt gesicherte Fassung mit einem Klick zurückzuholen.

Pfade. *Pfad kopieren* liefert den vollständigen Pfad, *Relativen Pfad kopieren* den Pfad innerhalb des Repositorys, und *Im Dateimanager anzeigen* zeigt die Datei im Dateimanager — oder, wenn du im Browser arbeitest, im Explorer von VS Code. Verweigert der Browser Computor den Zugriff auf die Zwischenablage, wird der Pfad in einem Feld angezeigt, aus dem du ihn von Hand kopieren kannst.

Was du nicht ändern kannst. Einiges wird für dich verwaltet: `.git` sowie auf der obersten Ebene einer Aufgabe die `README.md`, ihre Übersetzungen und `mediaFiles` — das ist die Aufgabenstellung, die deine Lehrenden pflegen. README-Dateien in Unterordnern darfst du selbst schreiben.

Eine Datei wiederherstellen

Hast du eine Datei verdorben, die zur Aufgabe gehörte — Daten, eine Abbildung, den Ausgangscode —, kannst du dir eine frische Kopie holen:

1. **Kopiere alles, was du behalten willst** (Rechtsklick → *Datei kopieren…*).
2. **Lösche** die Datei.
3. Rechtsklick auf den **Kurs** → *Repository aus Kursvorlage aktualisieren*.

Jede Datei, die im Kursmaterial vorhanden ist, in deinem Ordner aber nicht, wird wiederhergestellt, und Dateien, die du noch hast, bleiben unberührt. Deshalb ist das vorherige Löschen ein notwendiger Schritt und keine Abkürzung: Erst dadurch ist die Wiederherstellung für alles andere, was du geschrieben hast, gefahrlos.

Teamaufgaben

Manche Aufgaben werden von einer Gruppe abgegeben. Du teilst dir mit deinen Teammitgliedern einen Ordner und ein Abgabenkontingent: Jede und jeder von euch kann committen, und jede und jeder kann für alle abgeben. Der Text, der beim Überfahren mit der Maus erscheint, nennt die Mitglieder des Teams.

Sprecht ab, wer wann abgibt. Eine Abgabe, die ein Mitglied verbraucht, ist für alle verbraucht.

Wenn etwas schiefgeht

„Ich habe eine Datei gelöscht, die ich brauche." Siehe Eine Datei wiederherstellen. Hattest du die Datei gesichert, hat Computor dir womöglich auch gleich nach dem Löschen angeboten, deine eigene Fassung wiederherzustellen.

„Die Tests laufen nicht." Die Aufgabe muss lokal eingerichtet sein, und deine Arbeit muss sich hochladen lassen — ein `⚠` in der Zeile bedeutet, dass das Hochladen fehlschlägt; führe deshalb zuerst *Anmeldung am Repository reparieren* am Kurs aus. Startet der Lauf, schlagen aber alle Tests fehl, lies die Ausgabe in „Testergebnisse", bevor du einen weiteren Versuch verbrauchst.

„Die Abgabe ist fehlgeschlagen." Prüfe die Zahl der Abgaben in der Zeile. Liegt es nicht daran, achte auf `⚠` — für eine Abgabe muss deine Arbeit auf dem Server sein.

„Ich habe keine Testläufe mehr." Frag deine Lehrenden in den Kursnachrichten. Bis dahin gilt dein letztes erfolgreiches Ergebnis weiter, und abgeben kannst du trotzdem.

„Meine neueste Arbeit wurde nicht bewertet." Beim Abgeben wird der *getestete* Commit eingereicht. Führe einen Test mit deinem neuesten Commit aus und gib dann erneut ab, sofern dir noch eine Abgabe bleibt.

Weiter

- Kurse — das Repository, Commits und die Markierungen in den Zeilen
- Einheiten — wie ein Kurs gegliedert ist, und Lesematerial

Lernen mit Computor

Quelle: https://github.com/computor-org/data-science-python/blob/main/docs/GETTING_STARTED_de.md

Die drei Kurse der Reihe „Data Science mit Python" sind in der Kursaufteilung aufgeführt. Die Aufgabenstellung einer Übung steht in ihrem Ordner in `content/index_de.md` (englisch: `content/index_en.md`). Um das öffentliche Material zu lesen, brauchst du kein Konto. Musterlösungen für Lehrende und nicht öffentliche Bewertungstests werden getrennt aufbewahrt.

VS Code auf dem Desktop

Installiere Python 3.12 und VS Code. Klone das Repository data-science-python, öffne seinen Ordner und installiere die Erweiterung **Computor** des Herausgebers `computor-org`. Führe in einer virtuellen Python-Umgebung `python -m pip install --require-hashes -r requirements.lock` aus. Kopiere mitgelieferte Vorlagen für Studierende und Eingabedateien, soweit es sie gibt, in einen Arbeitsordner und führe deinen Code lokal aus. Manche Übungen beginnen mit einer leeren Datei. Deine Umgebung prüfst du mit `python -m pytest tests`.

GitHub Codespaces

Erstelle einen Codespace. Dafür brauchst du ein GitHub-Konto, und es gelten das Kontingent und die Abrechnung von GitHub. Die vorbereitete Entwicklungsumgebung des Codespace installiert die fest versionierten Python-Abhängigkeiten und die Computor-Erweiterung. Dein Code läuft in deinem Codespace und nicht auf den Servern von Computor. Ein KI-Modell wird nicht installiert.

Kurse und Lena

Melde dich bei Computor an, um einem Kurs beizutreten, deinen Fortschritt zu speichern, deine Arbeit abzugeben und Lena in den Kursnachrichten um Hinweise oder Rückmeldungen zu bitten. Ein gehosteter Arbeitsbereich ist optional: Wenn dessen Kapazität ausgeschöpft ist, kannst du in VS Code auf dem Desktop oder in Codespaces weiterarbeiten. Lena verarbeitet Fragen mit einem nicht öffentlichen KI-Dienst der TU Graz und kann keinen Code ausführen. Deine Nachrichten und deine abgegebene Arbeit stehen im Kursverlauf deines Kontos; Näheres regelt die Datenschutzerklärung der Plattform.

Um Lena zu fragen, schreibst du in den Nachrichten zu einer Aufgabe `@lena` oder gibst `@` ein und wählst Lena aus den Vorschlägen aus. Auch `@luna`, die ältere Schreibweise, und das kurze `@ai` erreichen dieselbe Tutorin; den Vorschlag auszuwählen ist nicht nötig.

Teil B: Für Tutor:innen und Lehrende

Die Tutor-Ansicht

Quelle: https://github.com/computor-org/computor-vscode/blob/main/docs/help/tutor-course_de.md

Als Tutorin oder Tutor sichtest du, was Studierende abgeben: Du siehst dir ihre Arbeit an, testest sie, bewertest sie und sagst ihnen, was sie ändern sollen. Dafür stellt dir Computor in der Seitenleiste zwei Ansichten bereit:

- **Kurse und Teilnehmende**, in der du den Kurs und die Person auswählst, deren Arbeit du durchsehen willst.
- **Kurse**, die die Kursinhalte dieser Person zeigt und alles bietet, was du brauchst, um eine Aufgabe zu prüfen, zu testen und zu bewerten.

Diese Seite erklärt, wie du dich zurechtfindest und die Arbeit findest, die auf dich wartet. Wie du eine einzelne Aufgabe bearbeitest, beschreibt Eine Aufgabe bewerten.

Eine Person auswählen

Kurse und Teilnehmende führt deine Kurse unter ihrer Organisation und Kursfamilie auf. Klicke auf einen Kurs, um ihn auszuwählen. Danach gilt:

- **Kursgruppen** schränkt die Liste der Teilnehmenden auf eine Gruppe ein. Mit *Alle Gruppen* siehst du alle, mit *Ohne Gruppe* jene, die keiner Gruppe angehören.
- Ein Klick auf eine Person lädt ihre Inhalte in die Ansicht **Kurse**.

Zwei Markierungen hinter dem Namen einer Person zeigen dir, wo Arbeit wartet:

Markierung	Bedeutung
📝 3	Drei abgegebene Aufgaben, die du noch nicht durchgesehen hast
🔔 2	Zwei ungelesene Nachrichten dieser Person

Arbeite die Personen mit einer 📝-Markierung ab; die Markierung verschwindet, sobald du bewertet hast. Sie *ist* deine Aufgabenliste.

Den Inhaltsbaum lesen

Die Ansicht **Kurse** beginnt mit drei Zeilen, die deine Auswahl wiederholen, nämlich Kurs, Gruppe und Person. Darauf folgen die Einheiten und Aufgaben des Kurses in der Reihenfolge, in der auch die Studierenden sie sehen.

Einheiten zeigen, wie viele Einträge sie enthalten, und tragen dieselben Markierungen `[📝]` und `[🔔]` wie ihre Aufgaben, sodass eine zugeklappte Einheit nie wartende Arbeit verbirgt.

Aufgaben sind die Zeilen, mit denen du arbeitest. Die Ecke des Symbols zeigt den Bewertungsstatus, den du gesetzt hast (grün für korrigiert, rot für nötige Korrektur, bernsteinfarben für mögliche Verbesserung); die Markierung darauf zeigt das letzte Testergebnis der Person und ob sie abgegeben hat. Die Zahlen hinter dem Namen sind die

Kontingente der Person: `(2/5)` bedeutet, dass zwei von fünf erlaubten Testläufen verbraucht sind, und das zweite Zahlenpaar zählt die Abgaben. Danach folgen das Testergebnis und die Bewertung in Prozent. Fährst du mit der Maus über die Zeile, erhältst du dieselben Angaben in Worten, einschließlich der Person, die bewertet hat.

Ein Klick auf eine Aufgabe lädt die neueste Abgabe der Person und die Musterlösung herunter und öffnet die Testergebnisse, sofern es welche gibt. Siehe Eine Aufgabe bewerten.

Verborgene Inhalte

Mit `👁 invisible` markierte Zeilen sind Inhalte, die Lehrende vor den Studierenden verborgen haben, meist vorbereitetes Material oder eine abgeschlossene Prüfung. Du siehst diese Zeilen weiterhin und kannst sie auch bewerten: Eine Prüfung wird oft direkt nach ihrem Ende verborgen, und ihre Abgaben brauchen trotzdem noch eine Bewertung.

Nachrichten und Kommentare

Nachrichten anzeigen öffnet bei einer Aufgabe dein Gespräch mit der Person über diese Aufgabe; hier sagst du ihr, was sie verbessern soll. Setzt du den Bewertungsstatus *Korrektur nötig* oder *Verbesserung möglich*, öffnet sich dieses Gespräch von selbst, weil die Person mit dem Urteil nichts anfangen kann, solange du ihr nicht sagst, was sie ändern soll.

Kommentare anzeigen bei einer Person ist etwas anderes: Kommentare sind Notizen über die Person, die nur das Kurspersonal lesen kann. Die Person selbst sieht sie nie. Halte darin alles fest, was die nächste Tutorin, der nächste Tutor oder die Lehrenden wissen sollten.

Weiter

- Eine Aufgabe bewerten: Auschecken, Testen, Vergleichen und Bewerten einer einzelnen Arbeit

Eine Aufgabe bewerten

Quelle: https://github.com/computor-org/computor-vscode/blob/main/docs/help/tutor-assignment_de.md

Eine Aufgabe, eine Person: Diese Seite führt dich durch die Durchsicht einer Abgabe, vom Öffnen bis zur Bewertung und der Nachricht, die dazugehört. Wie du die Person und die Aufgabe findest, beschreibt Die Tutor-Ansicht.

Was unter einer Aufgabe liegt

Klicke auf die Zeile der Aufgabe (oder Rechtsklick → *Auschecken*), und Computor lädt herunter, was du brauchst. Klappst du die Zeile danach auf, siehst du:

- **Abgaben**: die Dateien der neuesten Abgabe der Person. Das ist es, was du bewertest.
- **Musterlösung**: die Lösung zur bereitgestellten Version der Aufgabe, damit du sehen kannst, was erwartet war.
- **Verlauf**: frühere Abgaben, sofern es mehr als eine gibt. Jeder Eintrag lässt sich wie die neueste Abgabe aufklappen, testen und herunterladen.

Auschecken im Kontextmenü fragt nach, bevor es Dateien, die du bereits hast, erneut herunterlädt; ein Klick auf die Zeile holt nur, was noch fehlt.

Die Arbeit durchsehen

Lies zuerst die Aufgabenstellung. Das Vorschausymbol in der Zeile (oder *Anleitung anzeigen*) öffnet die Aufgabenbeschreibung, nach der die Person gearbeitet hat.

Öffne die Dateien. Klappe *Abgaben* auf und klicke auf eine Datei, um sie zu lesen. Um zu sehen, wie sie von der Musterlösung abweicht, klicke mit der rechten Maustaste auf eine Datei der Abgabe → *Mit Musterlösung vergleichen*; dann öffnet sich ein Vergleich mit derselben Datei in der Musterlösung.

Sieh dir das Testergebnis der Person an. Ein Klick auf die Zeile öffnet den letzten eigenen Testlauf der Person im Bereich Testergebnisse. Einträge unter *Verlauf* bieten *Testergebnisse anzeigen* für genau diese Abgabe.

Führe deinen eigenen Test aus. *Test ausführen* auf der Aufgabe oder einer Abgabe verpackt die heruntergeladenen Dateien und lässt die Tests des Kurses darüber laufen. Das ist dein Testlauf, nicht der der Person, und er verbraucht nichts von ihrem Kontingent. Auf *Musterlösung* prüft derselbe Befehl, ob die Musterlösung selbst die Tests noch besteht.

Die Bewertung festlegen

Klicke auf das Bewertungssymbol in der Zeile der Aufgabe oder wähle Rechtsklick → *Bewerten…*. Du wirst nach zwei Dingen gefragt:

1. **Die Bewertung**: eine Zahl zwischen 0.00 und 1.00. Das ist die offizielle Note, die der Person als Prozentwert neben ihrem Testergebnis angezeigt wird.
2. **Der Status**: was als Nächstes geschehen soll:

Status	Bedeutung
corrected	Erledigt. Die Arbeit ist durchgesehen, und die Bewertung bleibt bestehen
correction_necessary	Die Person muss nachbessern und erneut abgeben
improvement_possible	Gut genug, um so zu bleiben, doch die Person darf nachbessern
not_reviewed	Nimmt die Durchsicht zurück, etwa nachdem du die falsche Person bewertet hast

Die Bewertung gilt für die neueste abgegebene Version. Gibt die Person danach erneut ab, erscheint die Aufgabe wieder als nicht durchgesehen.

Nach einem Korrektururteil sag, was zu ändern ist. Wählst du *correction_necessary* oder *improvement_possible*, öffnet sich das Nachrichtengespräch zu dieser Aufgabe von selbst. Die Person sieht nur den Status und den Prozentwert; ohne Nachricht kann sie nicht wissen, was falsch war, also schreib eine, bevor du weitermachst.

Eine mit `👁 invisible` markierte Aufgabe bewertest du genauso. Verborgen heißt nur, dass Studierende den Inhalt nicht mehr sehen können, nicht aber, dass ihre Abgaben keine Bewertung mehr brauchen.

Zusätzliche Versuche gewähren

Hat eine Person keine Testläufe oder Abgaben mehr übrig und bittet um einen weiteren Versuch, klicke mit der rechten Maustaste auf die Aufgabe → *Höchstzahl der Testläufe für diese Person festlegen…* oder *Höchstzahl der Abgaben für diese Person festlegen…*. Die eingegebene Zahl gilt nur für diese Person und nur für diese Aufgabe; lässt du das Feld leer, gilt für sie wieder die normale Grenze der Aufgabe.

Weiter

- Die Tutor-Ansicht: Personen finden, Markierungen, verborgene Inhalte, Nachrichten und Kommentare

Leitfaden für Lehrende: Kurs-Dashboard

Quelle: https://github.com/computor-org/computor-backend/blob/main/docs/teacher-guide_de.md

Dieser Leitfaden richtet sich an Lehrende und Tutor:innen, die in der Computor-Webanwendung die Arbeit ihrer Studierenden verfolgen. Er erklärt, was jede Seite zeigt, woher die Zahlen stammen und was sie nicht aussagen. Technische Vorkenntnisse brauchst du dafür nicht.

Alles auf diesen Seiten wird aus dem berechnet, was die Plattform ohnehin aufzeichnet, wenn Studierende an Aufgaben arbeiten:

- **Testläufe**: Bei jedem Start der automatischen Tests durch Studierende werden der Zeitpunkt, die Punktzahl (0–100 %) und der Testbericht gespeichert.
- **Abgaben**: Wenn Studierende eine Aufgabe offiziell abgeben, wird der Zeitpunkt gespeichert.
- **Bewertungen und Feedback**: Wenn Tutor:innen oder Lehrende eine Abgabe bewerten, werden die Bewertung, der Status, das schriftliche Feedback, die bewertende Person und der Zeitpunkt gespeichert.

Tastenanschläge, Bildschirmzeit oder die im Editor verbrachte Zeit zeichnet die Plattform nicht auf. Was nach „aufgewendeter Zeit" aussieht, ist eine Schätzung und als solche gekennzeichnet.

Wo du die Seiten findest

Öffne einen Kurs und verwende die Seitenleiste:

Eintrag in der Seitenleiste	Wer ihn sieht	Wozu er dient
Lehrende → Bewertung	Lehrende und Personen mit weitergehenden Rechten	Kursstatistik, die Liste der Studierenden mit ihren Auffälligkeiten und die Detailseite jeder einzelnen Person
Lehrende → Typische Fehler	Lehrende und Personen mit weitergehenden Rechten	Die fehlgeschlagenen Tests, die viele Studierende gemeinsam haben
Tutor	Tutor:innen und Personen mit weitergehenden Rechten	Wie viel Arbeit auf eine Prüfung wartet

Eintrag in der Seitenleiste	Wer ihn sieht	Wozu er dient
Tutor → Abgaben	Tutor:innen und Personen mit weitergehenden Rechten	Die Warteschlange der zu prüfenden Abgaben
Tutor → Bewertung	Tutor:innen und Personen mit weitergehenden Rechten	Die Bewertung der Studierenden, eine Person nach der anderen

Tutor:innen sehen nur die Tutor-Seiten. Die Seiten unter „Lehrende", die Daten über den gesamten Kurs zeigen, sind Lehrenden, Maintainer:innen und Eigentümer:innen des Kurses sowie Administrator:innen vorbehalten.

Kursstatistik (Lehrende → Bewertung)

Der obere Teil der Seite fasst den Kurs zusammen:

- **Studierende**, **Durchschnittlicher Fortschritt** und **Abgeschlossen**: Als Fortschritt gilt der Anteil der abgebbaren Aufgaben, die eine Person offiziell abgegeben hat.
- **Gefährdet**: Studierende mit weniger als 25 % Fortschritt.
- **Inaktiv**: Studierende ohne Abgabe in den letzten 14 Tagen.
- **Verteilung des Fortschritts** und das Diagramm zu den Inhaltstypen zeigen, wie sich der Kurs auf die Fortschrittsstufen und die Inhaltstypen verteilt.

Auffälligkeiten

Unter den Diagrammen führt der Bereich **Auffälligkeiten** einfache, fest vorgegebene Regeln auf, die dich auf Studierende hinweisen, die du dir vielleicht ansehen möchtest. Eine Auffälligkeit ist ein Anlass zum Hinsehen, kein Urteil: Jede nennt die Regel, die gegriffen hat, und die Zahlen, die dazu geführt haben. Öffne auf der Seite **So werden Auffälligkeiten berechnet**, um jede Regel mit ihrem Schwellenwert zu sehen.

Auffälligkeit	Wann sie erscheint
Viele fehlgeschlagene Versuche	10 oder mehr Testläufe bei einer Aufgabe, ohne die volle Punktzahl zu erreichen. Solange die volle Punktzahl noch fehlt, erscheint dies als Warnung, sobald sie erreicht wurde, als Information.
Sprung auf volle Punktzahl	Ein Testlauf erreichte 100 % direkt nach einem Lauf mit höchstens 30 % bei derselben Aufgabe. Oft hat die Person einfach den Fehler gefunden; für sich genommen ist das kein Beleg für Abschreiben.
Inaktiv	Kein Testlauf und keine Abgabe in den letzten 14 Tagen oder noch überhaupt keine.
Geringer Fortschritt	Weniger als 25 % der abgebbaren Aufgaben abgegeben.
Abgabe auf den letzten Drücker	Die letzte Abgabe erfolgte weniger als 2 Stunden vor der Frist.
Abgabe nach der Frist	Mindestens eine Abgabe nach der Frist.

Fristen je Aufgabe speichert die Plattform nicht. Die beiden Fristregeln verwenden die **Abgabefrist**, die du im Bereich „Auffälligkeiten" (oder auf der Seite einer Person) einträgst. Sie wird in deinem Browser für diesen Kurs gespeichert und gilt für alle Aufgaben, weshalb du dort die Frist einträgst, die dich gerade interessiert. Ohne Abgabefrist entfallen die beiden Fristregeln.

Die Tabelle der Studierenden hat eine Spalte **Auffälligkeiten**. Fährst du mit der Maus über eine Auffälligkeit, liest du ihre Begründung, und mit einem Häkchen bei **Nur auffällige** listest du nur die betroffenen Studierenden auf. Ein Klick auf eine Person öffnet ihre Detailseite.

Einzelne Studierende (Lehrende → Bewertung → eine Person)

Die Detailseite zeigt von oben nach unten:

1. **Durchschnittliche Bewertung, Gesamtfortschritt und Abgabezahlen.**
2. **Abgabeverlauf**: die Abgaben der Person, über die Zeit aufsummiert, mit dem Startdatum und der Abgabefrist, die du oben auf der Seite festlegst.
3. **Auffälligkeiten**: dieselben Regeln wie oben für diese Person, jede mit ihrer Begründung.
4. **Bearbeitungsverlauf** (in der englischen Oberfläche: *Attempt timeline*): eine Zeile je Aufgabe, an der die Person gearbeitet hat.
    - *Versuche im Zeitverlauf*: Punkte sind Testläufe, nach Punktzahl eingefärbt (grün: volle Punktzahl, bernsteinfarben: mindestens 50 %, rot: unter 50 %); Balken sind Abgaben; eine gestrichelte Linie markiert die Abgabefrist. Alle Zeilen teilen sich eine Zeitachse, sodass du erkennst, wann an welcher Aufgabe gearbeitet wurde.
    - *Läufe* und *Abgaben*: jeweils deren Anzahl.
    - *Erste / letzte Aktivität*: das erste und das letzte Ereignis, sei es ein Testlauf oder eine Abgabe.
    - *Volle Punktzahl erreicht*: wann ein Testlauf erstmals 100 % erreichte, oder „noch nicht".
    - *Beste / letzte*: die beste und die neueste Testpunktzahl.
    - *Tage*: an wie vielen verschiedenen Tagen die Person daran gearbeitet hat.
    - *Mediane Pause*: die typische Zeit zwischen zwei aufeinanderfolgenden Testläufen.
    - *Gesch. aktive Zeit*: eine **Schätzung**. Sie summiert die Pausen zwischen aufeinanderfolgenden Testläufen, die höchstens 30 Minuten auseinanderliegen. Arbeit ohne Testlauf bleibt für die Plattform unsichtbar, weshalb du diesen Wert als Untergrenze lesen solltest und niemals als gemessene Arbeitszeit.
    - *Letzte Abgabe* (nur mit Abgabefrist): wie lange vor oder nach der Frist die letzte Abgabe erfolgte.
5. **Verlauf der Läufe, Abgaben und Rückmeldungen**: alles in zeitlicher Reihenfolge, das Neueste zuerst. Ein *Abgabe-Lauf* ist ein Testlauf zu einer offiziellen Abgabe; eine Zeile *Bewertung* zeigt die Bewertung, ihren Status, das schriftliche Feedback und die Person, die es gegeben hat. Mit den beiden Filtern zeigst du nur eine Aufgabe oder nur eine Art von Einträgen.
6. **Fortschritt im Kursinhalt**: jede Aufgabe mit ihrem Status und der Anzahl der Versuche.

Mit **Drucken** erhältst du die Seite auf Papier oder als PDF, etwa für ein Beratungsgespräch mit der Person.

Typische Fehler (Lehrende → Typische Fehler)

Diese Seite sammelt die fehlgeschlagenen Tests aus den Testläufen der Studierenden und fasst jene zusammen, die aus demselben Grund fehlschlagen. So erkennst du Probleme, die viele Studierende gemeinsam haben, etwa eine missverstandene Aufgabenstellung oder einen fehlenden Hinweis.

Zu jeder Gruppe siehst du die Aufgabe, den fehlgeschlagenen Test und die Fehlermeldung sowie, wie viele **Studierende** betroffen sind und wie viele fehlgeschlagene **Läufe** es gab. Gruppen mit den meisten betroffenen Studierenden stehen oben. Ein Klick auf eine Zeile zeigt bis zu drei Beispiele von verschiedenen Studierenden mit der ursprünglichen Meldung und einem Link zur Seite der jeweiligen Person.

So werden Meldungen gruppiert: Erhalten bleibt nur die erste Zeile, die den Fehler benennt, und Einzelheiten, die sich von Person zu Person unterscheiden, werden durch Platzhalter ersetzt, nämlich Zahlen durch `<n>`, Werte in Anführungszeichen durch `<str>`, Dateipfade durch `<path>` und Speicheradressen durch `<addr>`. Einfache Namen in Anführungszeichen, etwa `'np'` in „name 'np' is not defined”, bleiben erhalten, weil sie meist das eigentliche Problem benennen. Aus „assert 3 == 4” und „assert 7 == 2” wird also die eine Gruppe `assert <n> == <n>`.

Gut zu wissen:

- Es zählen nur Testläufe von Studierenden; die eigenen Testläufe von Lehrenden und Tutor:innen bleiben außen vor.
- Die Seite wertet die neuesten fehlgeschlagenen Testläufe aus, standardmäßig 300. Gibt es mehr, weist ein Hinweis darauf hin; wähle dann eine größere Anzahl oder eine einzelne Aufgabe, um weiter zurückzuschauen.
- *Läufe ohne Bericht* zählt die Läufe, zu denen kein ausführlicher Testbericht gespeichert wurde. Sie können zu keiner Gruppe beitragen.

Tutor-Seiten

Tutor (Übersicht)

Zeigt, wie viele Studierende du hast und wie viele davon unbewertete Abgaben haben, außerdem die Zahl der unbewerteten Abgaben und der ungelesenen Nachrichten, und verlinkt auf die beiden unten beschriebenen Seiten.

Abgaben

Die Warteschlange der zu prüfenden Abgaben umfasst alle Studierenden oder Teams mit Abgaben, standardmäßig nur jene mit unbewerteter Arbeit, wobei die am längsten Wartenden zuerst stehen. Du kannst nach Aufgabe filtern oder das Häkchen bei **Nur unbewertete** entfernen, um alles zu sehen. Ein Klick auf eine Zeile öffnet die Bewertungsseite dieser Person bei der richtigen Aufgabe.

Bewertung

Wähle eine Person aus der Liste aus (mit einem Häkchen bei **Nur mit unbewerteten Abgaben** grenzt du sie ein). Die Seite der Person führt links ihre Aufgaben auf; wählst du eine davon aus, siehst du:

- das letzte Testergebnis und die Zahl der Abgaben und Testläufe,
- ein Formular unter **Letzte Abgabe bewerten**: eine Bewertung in Prozent, einen Status (*Korrigiert*, *Korrektur nötig*, *Verbesserung möglich* oder *Nicht geprüft*) und schriftliches Feedback für die Person,
- die früheren Bewertungen und das Feedback zu dieser Aufgabe,
- die Details des letzten Testlaufs.

Das Feld für die Bewertung ist mit der letzten automatischen Testpunktzahl vorausgefüllt; ändere sie bei Bedarf. Bezieht sich eine Bewertung auf eine ältere Abgabe als die letzte, weist die Seite darauf hin.

Was das Dashboard nicht leistet

- Es beurteilt Studierende nicht automatisch. Auffälligkeiten sind fest vorgegebene, sichtbare Regeln mit genannten Schwellenwerten; eine verborgene Punktzahl gibt es nicht.
- Es misst nicht die Arbeitszeit. Die einzige Zeitangabe ist die als solche gekennzeichnete Schätzung im Bearbeitungsverlauf.
- Fristen für Aufgaben kennt es nur, wenn du eine Abgabefrist einträgst.

Teil C: Kurse und Aufgaben anlegen

Computor Testing Framework – Anleitung

Quelle: https://github.com/computor-org/computor-backend/blob/main/computor-testing/docs/USER_GUIDE_de.md

Diese Anleitung richtet sich an **Aufgabenautor:innen**, die test.yaml-Dateien schreiben und mit dem Framework die Abgaben von Studierenden testen.

Schnellstart

1. Installieren

Installiere die Pakete im Hauptverzeichnis einer lokalen Kopie des Repositorys `computor-backend`, weil `computor-types` nicht auf PyPI liegt:

```
pip install -e ./computor-types -e ./computor-testing
```

2. Umgebung prüfen

```
computor-test check --all
```

3. test.yaml anlegen

```
# Mit der Blocks-CLI eine Ausgangsdatei erzeugen
blocks init -l python -n "My Exercise Tests"

# Oder von Hand anlegen (siehe die Beispiele unten)
```

4. Tests ausführen

```
computor-test python run -t ./student -T test.yaml
```

Verzeichnisstruktur

Dein Aufgabenverzeichnis sollte so aussehen:

```
my-exercise/
├── test.yaml                 # Testdefinitionen (erforderlich)
├── meta.yaml                 # Metadaten (zum Hochladen auf die Plattform erforderlich)
├── content/
│   └── index_en.md           # Aufgabenstellung, eine Datei pro Sprache
├── reference/                # Musterlösung
│   └── solution.py
├── localTests/
│   ├── correctSolution/      # Soll alle Tests bestehen
│   │   └── solution.py
│   └── wrongSolution/        # Soll einige Tests nicht bestehen
│       └── solution.py
└── output/
    └── testSummary.json      # Erzeugte Testergebnisse
```

---

test.yaml schreiben

Grundstruktur

```yaml
name: Exercise Name
description: What this exercise tests
type: python           # python, octave, r, julia, c, fortran, document
version: "1.0"

properties:
  relativeTolerance: 1e-12    # Standardwert für numerische Vergleiche
  absoluteTolerance: 0.0001
  timeout: 180                # Sekunden

  tests:
    - type: exist             # Testsammlung
      name: File Check
      tests:
        - name: solution.py   # Einzelner Test

    - type: variable
      name: Variable Tests
      entryPoint: solution.py
      tests:
        - name: result
          value: 42
```

Vererbung von Eigenschaften

Eigenschaften, die auf der Ebene `properties:` gesetzt sind, dienen als Standardwerte. Eine Testsammlung kann sie überschreiben, und ihre Werte kann wiederum ein einzelner Test überschreiben.

Diese Vererbung gilt in allen Testern für `qualification`, `relativeTolerance`, `absoluteTolerance`, `typeCheck`, `shapeCheck`, `ignoreClass` und `verbosity`, während von `properties:` an die Sammlungen außerdem `timeout`, `storeGraphicsArtifacts` und `competency` weitergegeben werden. `allowedOccuranceRange` und `occuranceType` werden nicht vererbt, weshalb du sie bei jedem Test setzen musst, der sie braucht.

Testtypen

exist – Vorhandensein von Dateien

Dieser Testtyp steht für alle Sprachen zur Verfügung und prüft, ob die geforderten Dateien vorhanden sind.

```
- type: exist
  name: Required Files
  tests:
    - name: solution.py          # Exakter Dateiname
    - name: data/*.csv           # Glob-Muster
    - name: config.json
      allowEmpty: true           # Dateien mit 0 Byte zulassen (Standardwert: false)
```

variable – Vergleich von Variablen

Verfügbar für: Python, Octave, R, Julia

Dieser Testtyp führt ein Skript aus und vergleicht die Variablen, die nach dem Lauf des Skripts vorliegen, mit den erwarteten Werten.

```
- type: variable
  name: Computation Tests
  entryPoint: solution.py          # Auszuführendes Skript
  tests:
    - name: result                 # Name der Variable
      value: 42                    # Erwarteter Wert

    - name: matrix
      value: [[1, 2], [3, 4]]     # Arrays und Matrizen

    - name: pi_approx
      value: 3.14159
      relativeTolerance: 1e-4     # Überschreibt die Standardtoleranz

    - name: greeting
      value: "Hello, World!"
```

Mit Setup- und Teardown-Code

```
- type: variable
  name: Advanced Tests
  entryPoint: solution.py
  setUpCode:
    - "x = 10"                     # Python und Octave: läuft nach dem Einstiegspunkt
    - "y = compute(x)"
  tearDownCode:
    - "cleanup()"                  # Läuft nach setUpCode, bevor die Variablen gelesen
werden
  tests:
    - name: y
      value: 100
```

Mit Eingabe (stdin)

```
- type: variable
  name: Interactive Tests
  entryPoint: solution.py
  inputAnswers:
    - "5"                          # Der erste input()-Aufruf erhält "5"
    - "hello"                      # Der zweite input()-Aufruf erhält "hello"
  tests:
    - name: result
      value: 5
```

stdout – Vergleich der Ausgabe

Verfügbar für: C, Fortran, Python, R, Julia

```
- type: stdout
  name: Output Tests
  entryPoint: main.c
  tests:
    - name: greeting
      qualification: contains      # Prüft, ob die Ausgabe diesen Text enthält
      pattern: "Hello"

    - name: exact
      qualification: matches       # Exakte Übereinstimmung
      value: "Hello, World!\n"

    - name: pattern
      qualification: regexp        # Treffer eines regulären Ausdrucks
      pattern: "Result: \\d+"

    - name: prefix
      qualification: startsWith
      pattern: "Program"

    - name: suffix
      qualification: endsWith
      pattern: "Done."
```

Ohne `qualification` wird die Ausgabe mit `value` verglichen oder, falls kein `value` angegeben ist, mit der Ausgabe der Musterlösung. C und Fortran entfernen vor dem Vergleich den Leerraum am Anfang und Ende der Ausgabe und akzeptieren `value` auch anstelle von `pattern`. Python vergleicht die unveränderte Ausgabe und entnimmt den erwarteten Text bei jeder `qualification`, auch bei `matches`, aus `pattern`. R und Julia unterstützen `contains` und `regexp` (und behandeln `matches` als regulären Ausdruck).

structural – Struktur des Codes

Dieser Testtyp steht für alle Sprachen zur Verfügung und sucht im Quellcode nach Schlüsselwörtern oder Mustern.

```
- type: structural
  name: Code Structure
  file: solution.py                # Zu analysierende Datei
  tests:
    - name: no_eval                # Sprechender Name
      pattern: "\\beval\\b"        # Gesuchter regulärer Ausdruck
      allowedOccuranceRange: [0, 0]  # Darf nicht vorkommen (verboten)

    - name: uses_function
      pattern: "\\bdef\\b"
      allowedOccuranceRange: [1, 0]  # Mindestens einmal, ohne Obergrenze

    - name: for_loop
      pattern: "\\bfor\\b"
      allowedOccuranceRange: [2, 5]  # Muss 2- bis 5-mal vorkommen
```

Hinweis zu `allowedOccuranceRange`: Das Format lautet `[min, max]`:

Wert	Bedeutung
`[0, 0]`	Das Element darf **überhaupt nicht** vorkommen
`[n, 0]` (n > 0)	Mindestens `n` Vorkommen, ohne Obergrenze
`[min, max]`	Zwischen `min` und `max` Vorkommen, einschließlich der Grenzen

Alternativ verlangt `countRequirement: n` eine exakte Anzahl, was allerdings nur gilt, wenn `allowedOccuranceRange` nicht gesetzt ist; ein Untertest ohne beide Angaben wird übersprungen.

In Python wird `name` mit dem Token-Strom abgeglichen, sodass gepunktete Namen wie `np.pi` als Einheit gezählt und Treffer in Kommentaren und Zeichenketten ignoriert werden. Gibst du stattdessen ein `pattern` an, sucht der Test mit einem regulären Ausdruck im unveränderten Dateitext, der Kommentare und Zeichenketten einschließt.

R und Julia zählen `pattern` (oder `name`) auf dieselbe Weise. Octave ignoriert `pattern` und zählt das Schlüsselwort oder die Funktion, die als `name` angegeben ist. C und Fortran wenden den Bereich auf bekannte Schlüsselwörter an, die als `name` angegeben sind (etwa `for` oder `malloc`); bei allen anderen Tests prüfen sie nur, ob `pattern` (oder `name`) im Quellcode vorkommt.

exitcode – Prüfung des Exit-Codes (C, Fortran)

```
- type: exitcode
  name: Exit Code
  entryPoint: main.c
  tests:
    - name: success
      expectedExitCode: 0

    - name: error_case
      expectedExitCode: 1
```

compile – Kompilierungstest (C, Fortran)

```
- type: compile
  name: Compilation
  entryPoint: main.c
  compiler: gcc
  compilerFlags: ["-Wall", "-Wextra", "-Werror"]
  tests:
    - name: compiles_clean         # Soll ohne Fehler kompilieren
```

graphics – Test von Plots (Python, Octave)

Jedes `name` ist ein Ausdruck, der ausgewertet wird, nachdem das Skript gelaufen ist. In Python wird ihm `plt.` vorangestellt, in Octave wird er so ausgewertet, wie er dasteht. Das Ergebnis wird mit `value` oder mit der Musterlösung verglichen.

```
- type: graphics
  name: Plot Tests
  entryPoint: plot_script.py
  tests:
    - name: gcf().axes[0].get_xlabel()
      value: "Time"
    - name: gcf().axes[0].get_title()
      value: "Data Plot"
```

```
- type: graphics
  name: Plot Tests
  entryPoint: plot_script.m
  tests:
    - name: get(gca, 'XLim')
      value: [0, 10]
```

Testtypen für Dokumente

Verfügbar für: Document

```
type: document

properties:
  tests:
    - type: wordcount
      name: Word Count
      file: essay.md
      tests:
        - name: minimum_words
          allowedOccuranceRange: [500, 0]   # Mindestens 500 Wörter

    - type: section
      name: Required Sections
      file: essay.md
      tests:
        - name: Introduction
        - name: Methodology
        - name: Conclusion

    - type: keyword
```

```
      name: Keywords
      file: essay.md
      tests:
        - name: algorithm
          allowedOccuranceRange: [3, 0]     # Mindestens 3 Vorkommen

    - type: linecount
      name: Line Count
      file: essay.md
      tests:
        - name: minimum_lines
          allowedOccuranceRange: [50, 0]
```

Tests vom Erfolg früherer Tests abhängig machen

Du kannst Testsammlungen vom Erfolg vorhergehender Sammlungen abhängig machen:

```
properties:
  tests:
    - type: exist
      name: File Check
      id: file_check              # Vergibt eine ID
      tests:
        - name: solution.py

    - type: variable
      name: Variable Tests
      entryPoint: solution.py
      successDependency: file_check   # Läuft nur, wenn File Check bestanden wurde
      tests:
        - name: result
          value: 42
```

Token-Ersetzung

In Octave-Tests lassen sich Dateien, die frühere `exist`-Tests gefunden haben, im `file` eines `structural`-Tests als `#file_N#` ansprechen (`N` zählt ab 1):

```
- type: exist
  name: Script
  tests:
    - name: "*.m"                 # Beliebige abgegebene .m-Datei

- type: structural
  name: Loops
  file: "#file_1#"                # Wird durch die oben gefundene Datei ersetzt
  tests:
    - name: for
      allowedOccuranceRange: [1, 0]
```

---

meta.yaml

Die Metadaten der Aufgabe sind erforderlich, wenn sie auf die Plattform hochgeladen wird, wo sie „Beispiel” heißt. Der Tester selbst liest nur `properties.testDependencies`.

```
slug: physics.python.hello_world  # Eindeutige, durch Punkte gegliederte Kennung
version: '1.0'
title: Hello World in Python
description: A simple Hello World program
language: en
license: CC BY-NC 4.0
keywords: [python, basics]         # Erscheinen in der Beispielbibliothek als Schlagwörter
category: Basics

authors:
  - name: Your Name
    email: your.email@example.org
    affiliation: Your Institute

properties:
  studentSubmissionFiles:
    - solution.py
  executionBackend:
    language: python
    version: "3.12"
```

`executionBackend` nimmt entweder eine `language` entgegen (python, octave, r, julia, c, cpp, fortran, document, matlab), die mit den Test-Runnern der Installation, also den Diensten, die die Tests ausführen, abgeglichen wird und sich durch eine optionale `version` ergänzen lässt, oder einen `slug`, der genau einen bestimmten Runner festlegt. Die zugehörigen `settings` werden zwar gespeichert, von den Testern aber nicht angewendet; `timeout` und Compileroptionen setzt du in `test.yaml`.

CLI-Referenz

Einheitliche CLI

```
# Tests ausführen
computor-test <language> run -t <student_dir> -T <test.yaml> [-v 0-3]

# Lokal testen (test.yaml im übergeordneten Verzeichnis oder in <solution_dir>)
computor-test <language> local -d <solution_dir>

# Laufzeitumgebung prüfen
computor-test check --all
computor-test check python

# Verfügbare Tester auflisten
computor-test list
```

Blocks-CLI (Erzeugung von test.yaml)

```
# Verfügbare Testtypen auflisten
blocks list -l python

# Neue test.yaml anlegen
blocks init -l python -n "My Tests"

# Testschnipsel erzeugen
blocks generate -l python -t variable -n result

# Vorlagen durchsehen
blocks templates list
blocks templates show "Variable Check" -l python

# Für die VS-Code-Erweiterung exportieren
blocks export ./generated
```

Sandbox-CLI

```
# Backends prüfen
sandbox check -v

# Befehl in der Sandbox ausführen
sandbox run python3 -c "print('hello')"

# Konfiguration anzeigen
sandbox config --show

# Funktion der Sandbox testen
sandbox test
```

Installation von Abhängigkeiten

```
# Alle Abhängigkeiten installieren
deps-installer install -f dependencies.yaml

# Dockerfile erzeugen
deps-installer dockerfile -f dependencies.yaml -o Dockerfile

# Abhängigkeitsdatei validieren
deps-installer validate -f dependencies.yaml
```

Fehlerbehebung

„Runtime not found”

```
# Prüfen, welche Laufzeitumgebungen verfügbar sind
computor-test check --all

# Für R muss jsonlite installiert sein:
Rscript -e "install.packages('jsonlite', repos='https://cloud.r-project.org/')"

# Für Julia muss JSON installiert sein:
julia -e 'import Pkg; Pkg.add("JSON")'
```

Tests bestehen lokal, schlagen auf dem Server aber fehl

- Prüfe die Timeout-Werte, denn der Server kann langsamer sein
- Stelle sicher, dass test.yaml keine fest eingetragenen absoluten Pfade enthält
- Prüfe, ob die Pakete, die dein Code importiert, auf dem Test-Runner der Plattform installiert sind; die Plattform liest `dependencies.yaml` nicht
- Vergewissere dich, dass `executionBackend` in meta.yaml eine Sprache oder einen Runner nennt, die oder den die Installation bereitstellt

„Execution failed” ohne weitere Meldung

Jeder Prozess, der Code von Studierenden ausführt, läuft unter Ressourcenbegrenzungen. `COMPUTOR_JOB_MAX_PROCESSES` (Standardwert 512) begrenzt die Zahl aller Prozesse und Threads deines Benutzerkontos, sodass lokale Läufe auf einem ausgelasteten Arbeitsrechner den Interpreter womöglich gar nicht erst starten können. Setze für lokale Läufe `COMPUTOR_JOB_MAX_PROCESSES=0`.

„Path validation failed”

Das Framework verhindert, dass Pfade aus dem erlaubten Verzeichnis herausführen (Directory Traversal). Achte deshalb auf Folgendes:

- Dateipfade enthalten kein `..`
- Alle Pfade sind relativ zum Wurzelverzeichnis der Tests
- Einstiegspunkte verweisen auf Dateien innerhalb des Verzeichnisses der Abgabe oder der Musterlösung

Ausgabe wird nicht erfasst

Bei interpretierten Sprachen musst du sicherstellen, dass das Einstiegsskript tatsächlich eine Ausgabe erzeugt. Bei kompilierten Sprachen musst du darauf achten, dass `printf`/`cout` die Ausgabe auch wirklich schreibt (verwende `\n` oder `fflush(stdout)`).