

2.1.1 Definition Computational Thinking

Bis heute gibt es verschiedene Versionen wie Computational Thinking definiert wird und es wurde kein Konsens über eine gemeinsam vereinbarte Definition gefunden. Der Begriff wurde 1996 von Seymour Papert kreiert und durch Jeanette Wing in 2006 populär gemacht. Wing und Papert definieren Computational Thinking als die

«Fähigkeit, ein Problem in kleinere Probleme zu zerlegen, die deterministisch gelöst werden können.» – Wing (2006) & Papert (1996)

Beim CT geht es spezifisch um die Fähigkeit, ein Problem in kleine Teilprobleme zu zerlegen, die wiederum eindeutig gelöst werden können, das Ergebnis führt zu einem Algorithmus. Es umfasst das Finden von kreativen Lösungskonzepten, und dass Verstehen dieser Konzepte, so dass sie einer Maschine oder einem anderen Menschen beigebracht werden oder von ihnen ausgeführt werden können.

«Die **Denkprozesse**, die bei der Formulierung von **Problemen** und deren **Lösungen** ablaufen, so dass die Lösungen in einer Form dargestellt werden, die von einem informationsverarbeitenden Agenten effektiv durchgeführt werden kann.» – Cuny, Snyder, Wing (2011)

2.1.2 Was ist Computational Thinking, und was ist es nicht

Wing (2006) grenzt Computational Thinking wie folgt ab:

- CT ist Konzeptualisierung und nicht Programmierung: Wie ein Informatiker zu denken, bedeutet mehr als zu Programmieren und Computerprogramme zu schreiben. Es erfordert ein Denken auf mehreren Abstraktionsebenen und das analytische angehen von Problemen.
- CT ist eine Art und Weise, wie Menschen denken und Probleme lösen. Es beabsichtigt nicht Menschen dazu zu bringen wie Computer zu denken.
- Der Fokus von CT liegt auf Ideen und Prozessen, nicht auf physischen Software- oder Hardware Artefakten, die aus CT entstehen können. Es geht um Konzepte mit deren Hilfe wir Probleme angehen und lösen, unser tägliches Leben meistern und mit anderen Menschen kommunizieren und interagieren.
- CT ist für jeden und überall, nicht nur für Informatiker.

2.2 Die Hauptbestandteile von Computational Thinking

Die vier Hauptbestandteile von Computational Thinking sind:

- Dekomposition,
- Mustererkennung,
- Abstraktion, und
- Algorithmen.

In den folgenden Kapiteln stellen wir diese Bestandteile genauer vor.

2.2.1 Dekomposition

Dekomposition erlaubt es, ein Problem in kleinere Teilprobleme herunterzubrechen, so dass man sich auf das Lösen dieser kleineren und einfacheren Teilprobleme fokussieren kann. Dies ist vor allem bei grösseren, komplexeren Problemen sehr wichtig, denn nur mit Zerlegung ist es möglich eine Anwendung zu schreiben die, wie zum Beispiel die Google Suche, Millionen von Schritten und Anweisungen beinhaltet für die Lösung.

Der grosse Vorteil der Zerlegung ist, dass die Komplexität eines Problems reduziert wird. Das Zerlegen des Problems in kleinere Teile bedeutet, dass jedes selbstständig untersucht und gelöst werden kann. Zudem ist es einfacher einen Überblick über das gesamte Problem zu wahren.

Es gibt unterschiedliche Methoden zur Zerlegung eines Problems in Teilprobleme. Eine davon ist die Problemlösung durch Zielreduktion. Dies ist eine Methode, die eine Zerlegung in kleinere, aber sonst gleiche Teile erlaubt. Beispielsweise wird das Problem halbiert. Das führt dann zu zwei gleich grossen Teilen, die kleiner und leichter lösbar sind als das Ausgangsproblem.

Eine weitere Methode ist die rekursive Problemlösung mit Zielreduktion. Dabei wird die Teilung von einem Problem wiederholt angewendet. Ein Beispiel dafür ist, wenn ein Name im Telefonbuch gesucht wird. Erst wird das Telefonbuch in der Mitte aufgeschlagen und geprüft ob der gesuchte Namen vor oder nach der aufgeschlagenen Seite aufgeführt ist. Im nächsten Schritt beschränken wir uns dann auf den Teil, in dem der Name vorkommen sollte. Wir haben nun ein ähnliches, aber kleineres Problem: wir suchen nun den Namen nur noch in der Hälfte des Telefonbuches. Dafür wird das Buch nochmals in der Hälfte dieses einen Teiles, in dem der Name vorkommt, aufgeschlagen, und so weiter bis wir den Namen gefunden haben.

2.2.2 Mustererkennung

Ein weiterer zentraler Bestandteil von CT ist die Mustererkennung. Dabei werden Gemeinsamkeiten oder Ähnlichkeiten zwischen den einzelnen Teilproblemen sowie auch zwischen Problemen identifiziert. Dies ermöglicht eine Vereinfachung eines komplexen Problems durch die Wiederverwendung der gleichen Lösung für gleiche oder ähnliche Probleme. Hat man beispielsweise bereits ein Problem gelöst, das Gemeinsamkeiten mit dem aktuellen Problem aufweist, kann auf die bereits existierende Lösung zurückgegriffen werden. Man muss somit nicht immer wieder von Neuem beginnen und kann Erfahrungen und Wissen in die aktuelle Problemlösung mit einbeziehen.

2.2.3 Abstraktion

Ein weiterer wichtiger Bestandteil von CT ist es die wichtigen Aspekte eines Problems zu identifizieren und die unwichtigen Aspekte wegzulassen. Diese Abstraktion—also der Fokus auf relevante Eigenschaften und das Verbergen der irrelevanten Details—erlaubt es uns die Komplexität besser zu bewältigen und erleichtert die Lösung eines Problems.

Es gibt unterschiedliche Anwendungen von Abstraktion, beispielsweise die Abstraktion der Steuerung oder die Datenabstraktion. Bei der Abstraktion der Steuerung werden Anweisungen so gruppiert, dass sich neue Anweisungen ergeben, die grössere Schritte darstellen. In Folgen, können wir diese grösseren Schritte verwenden um ein Problem zu lösen und die Details der notwendigen Einzelschritte verbergen. Bei der Datenabstraktion werden irrelevante Details der Daten verborgen, wie zum Beispiel die Speicherung der Daten oder die genaue Repräsentation.

2.2.4 Algorithmus

CT ist ein Denkprozess um Probleme zu lösen und anzugehen. Die daraus resultierenden Lösungen sind meist Algorithmen. Ein Algorithmus ist eine Reihe von Anweisungen oder eine Beschreibung aus mehreren Anweisungen, um etwas zu erreichen:

«Ein Algorithmus ist eine endliche Folge wohldefinierter Anweisungen, typischerweise zur Lösung einer Klasse von Problemen.»

«[...]. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten. [...] Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.» – Leiserson et al. (2010)

Aus dieser Definition sind folgende Eigenschaften eines Algorithmus ableitbar:

- Ein Algorithmus ist ein endlicher Text in eindeutiger Sprache der eine Abfolge von Schritten festlegt. Algorithmen können zur Ausführung in einem Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert werden. Die Abfolge von Schritten erzeugen aus der Eingabe eine Ausgabe.
- Die Repräsentation und der Wertebereich der Eingabe und Ausgabe/Ergebnisse müssen eindeutig definiert sein.
- Jeder Schritt des Algorithmus muss tatsächlich ausführbar sein.
- Die Schritte sind Basisoperationen, d.h. sie sind vom Computer oder Menschen direkt ausführbar, oder sie sind selbst wieder durch einen Algorithmus definiert.

Wenn für die Problemlösung ein Algorithmus vorhanden ist, sollte jeder das Problem lösen können, ohne den Algorithmus davor zu kennen oder verstehen zu müssen, was dieser im Detail macht. Solange die Schritte genau befolgt werden, wird der Algorithmus zur richtigen Lösung führen. Somit ist das Ziel eines Algorithmus, dass eine erstellte Beschreibung aus einfachen Anweisungen von einer Maschine oder jemandem anderem ausgeführt werden kann, um eine bestimmte Problemstellung zu lösen. Daraus folgt auch dass eine Maschine jede Variation des Problems lösen kann, indem sie einfach mechanisch den Anweisungen folgt. Denn das ist alles, was Computer tun: Sie folgen Algorithmen, die von Menschen geschrieben wurden.

Kriterien von Algorithmen

Es gibt drei Kriterien, die Algorithmen erfüllen müssen um von einem Computer ausgeführt werden zu können (Lustenberger, 2008): Maschinentauglichkeit, Allgemeinheit und Korrektheit.

Das erste Kriterium ist die Maschinentauglichkeit: ein Algorithmus muss von einer einfachen Maschine ausgeführt werden können. Dass heisst, der Algorithmus muss aus einer Beschreibung von einfachen Anweisungen bestehen, die von einer Maschine gelesen werden kann und die Maschine muss basierend auf der Beschreibung die beschriebenen Tätigkeiten ausführen können.

Das zweite Kriterium ist die Allgemeinheit: ein Algorithmus muss auf alle Problemfälle des zu lösenden Problems anwendbar sein. Beispielsweise wenn man einen Algorithmus zur Sortierung von Zahlen entwickelt, muss dieser für beliebige Zahlen und eine beliebige Anzahl von Zahlen anwendbar sein.

Das letzte wichtige Kriterium ist die Korrektheit: ein Algorithmus muss für alle Fälle eines gegebenen Problems in endlicher Zeit die richtige Lösung liefern.

2.3 Weitere Fertigkeiten von Computational Thinking

Computational Thinking bringt unterschiedliche Fertigkeiten zusammen, die vereint eine leistungsfähige Art des Denkens ergeben. Zu diesen Fähigkeiten gehören Modellierung, wissenschaftliches und logisches Denken, Kreativität, Heuristiken, Menschen zu verstehen und die Evaluation von Algorithmen (Curzon & McOwen 2018).

Computer Modellierung

Bei der Computer Modellierung wird ein Algorithmus erstellt, der die Realität simuliert, d.h. etwas aus der realen Welt wird in einer virtuellen Welt abgebildet. Dadurch können Experimente durchgeführt werden, die beispielsweise bei einer realen Durchführung zu lange dauern oder nicht möglich sind. Ein Beispiel für eine Computermodellierung ist eine Wettervorhersage: Indem man die reale Welt im Computer modelliert, kann man Vorhersagen darüber treffen, was in der Realität geschehen wird, etwa ob es morgen in einer bestimmten Region regnen wird oder nicht.

Wissenschaftliches Denken

Wissenschaftliches Denken ist das strukturierte und systematische Vorgehen für Erklärungen und um Vorhersagen zu treffen. Dies bedeutet, dass man eine Fragestellung so angeht, dass man Erkenntnisse gründlich recherchiert, diese systematisch durcharbeitet und versucht, die vorliegenden Erkenntnisse zu verstehen, zu bewerten und daraus Schlüsse für die eigene Fragestellung zu ziehen. Zum Schluss wird eine Hypothese aufgestellt, die dann durch weitere Daten überprüft wird.

Logisches Denken

Zum Computational Thinking gehört auch das logische Denken. Es erlaubt uns Zusammenhänge zu erschliessen und Muster zu erkennen. Es ist ein folgerichtiges und schlüssiges Denken und führt dazu, dass alle Eventualitäten sowie Ausnahmefälle berücksichtigt und Details beachtet werden.

Kreativität

Eine weitere Fertigkeit ist die Kreativität. Probleme zu lösen und Ideen zu entwickeln sowie auch deren Konzepte zur Umsetzung in der Realität auszuarbeiten ist ein kreativer Prozess. Lösungen für ein Problem können sehr kreativ sein und alte oder neue Probleme innovativ lösen.

Heuristik

In Algorithmen kommen häufig Heuristiken ins Spiel, vor allem da es nicht immer möglich ist die allerbeste Lösung in absehbarer Zeit zu finden. Deshalb gibt es heuristische Algorithmen, welche die bestmögliche, vernünftige Lösung in angemessener Zeit finden.

Menschen verstehen

Beim Computational Thinking ist es auch wichtig auf den Menschen zu achten und Menschenkenntnis einfließen zu lassen. Insbesondere da Algorithmen oft Menschen unterstützen, ist es wichtig die Stärken und Schwächen von Menschen zu verstehen und in der Technologie, resp. dem Algorithmus, zu berücksichtigen.

Menschen verstehen – Post Completion Error

Das folgende Rätsel stammt von Curzon & McOwen (2018).

Aufgabe:

Eine Bäuerin ist mit ihrem Schäferhund Maxi, der sie immer begleitet, auf dem Weg zum nächsten Dorf. Um dorthin zu gelangen, muss sie einen reissenden Fluss überqueren. Dafür hat eine Erfinderin, die auf der Dorfseite des Flusses wohnt, eine Vorrichtung erfunden, die die Flussüberquerung ermöglicht. Sie besteht aus einem Seil, Rollen und einem Sitz, der an dem Seil hängt und Platz für eine Person bietet. Die Bewohner des Ortes haben vereinbart, dass der Sitz immer auf der Seite des Dorfes zu lassen, wo die Erfinderin wohnt. Als die Bäuerin zum Fluss kommt, zieht sie den Sitz mit dem Seil zu sich herüber, setzt sich hinein, nimmt Maxi auf den Arm, zieht sich selbst auf die andere Seite und setzt ihren Weg zum Dorf fort. Eines Tages kauft sie eine Henne und einen Sack Reis. Als sie auf dem Heimweg zum Fluss kommt, stellt sie fest, dass sie nur ein Ding mit sich hinübernehmen kann, wenn sie den Sitz verwenden möchte. Deshalb benötigt sie mehrere Fahrten. Dabei gibt es folgendes Problem:

Wenn sie die Henne mit dem Getreide auf einer Seite allein lässt, wird die Henne das Getreide fressen. Lässt sie Hund Maxi und die Henne auf einer Seite zusammen, wird der Hund die Henne jagen. Maxi frisst kein Getreide, deshalb kann er mit dem Getreide allein gelassen werden.

Schreiben Sie einen Algorithmus um alle drei Sachen auf die andere Flussseite zu bringen, ohne dass etwas gefressen oder gejagt wird.

Lösung:

1. Die Bäuerin fährt mit der Henne auf die andere Seite.

2. Die Bäuerin kehrt alleine zurück.
 3. Die Bäuerin fährt mit dem Hund auf die andere Seite.
 4. Die Bäuerin kehrt mit der Henne zurück.
 5. Die Bäuerin fährt mit dem Getreide auf die andere Seite.
 6. Die Bäuerin kehrt alleine zurück.
 7. Die Bäuerin fährt mit der Henne auf die andere Seite.
- 8. Die Bäuerin schickt den leeren Sitz zum anderen Ufer zurück.**

Die meisten Lösungen für dieses Problem, vielleicht auch Ihre, enden bei Schritt 7. Bei der Lösung dieser Aufgabe konzentrieren wir uns oft sehr stark auf das Ziel die einzelnen Dinge sicher über den Fluss zu bringen. Da wir Menschen ein begrenztes Kurzzeitgedächtnis haben, kommt es öfters vor, dass die Erfüllung eines Teilziels, das zum korrekten Abschluss des Hauptziels, vergessen geht. In diesem Beispiel ist der letzte Schritt um das Problem zu lösen, dass die Bäuerin den leeren Sitz an das andere Ufer zurück sendet. Insbesondere wenn Menschen bereits viele Informationen im Kurzzeitgedächtnis haben, passiert es, dass wir diesen Schritt vergessen. Dieser Fehler wird auch Post-Completion-Error genannt. Dabei handelt es sich um Fehler, bei denen Menschen den letzten Schritt, respektive das letzte Teilziel vergessen, um ein Problem vollständig zu lösen. Wenn wir Menschenkenntnisse mit in unsere Algorithmen einfließen lassen, können wir solche Fehler vermeiden und Menschen besser unterstützen.

Ein Beispiel für einen Algorithmus der einen Post-Completion Fehler vermeidet ist bei Bankautomaten anzutreffen. Die meisten Bankautomaten geben dem Nutzer erst Geld aus, wenn die Karte entnommen wurde. Bei Automaten die zuerst das Geld ausgeben, kommt es häufiger vor, dass Menschen die Karte am Ende vergessen.

Evaluation

Die Evaluation stellt sicher, dass der Algorithmus funktioniert und eine den Zweck erfüllt. Dabei ist es auch wichtig zu überprüfen, ob der Algorithmus den Anforderungen entspricht, die das Problem beschreiben. Neben diesen Aspekten können auch weitere Aspekte wie die Benutzerfreundlichkeit oder Leistungsfähigkeit des Algorithmus evaluiert werden. Es ist wichtig, den Algorithmus nicht nur am Schluss, wenn er fertig entwickelt wurde, zu evaluieren, sondern den Algorithmus über den ganzen Prozess hinweg kontinuierlich zu evaluieren.

2.4 Aufgaben

Begabtenförderung

Jährlich erhält eine Stiftung zur Förderung eines Studiums mehrere hundert Bewerbungen, von denen sie 20 mit einem Stipendium fördert. Entwerfen Sie einen Algorithmus, der eine Rangliste der Bewerber*innen und derer CVs basierend auf den nachfolgenden Kriterien erstellt, und die besten 20 auswählt.

Förderungskriterien:

- Notenschnitt von mindestens 5.3 (Matura, Berufsmatura)
- Alter von höchstens 28 Jahren
- Kenntnisse von mindestens zwei Amtssprachen
- Gesellschaftliches Engagement, breite Interessen, intellektuelle Neugier und Kreativität

Tipp:

Um eine Rangliste zu erstellen, benutze ein Punktesystem dass basierend auf gewissen Charakteristiken die Bewerber*innen bewertet. Zum Beispiel könnte es pro gespieltem Musikinstrument +1 Punkt geben usw..

Meetingplanung

Für ein Seminar gibt es drei Gruppen die mit dem Teaching Assistant ein wöchentliches Meeting am Donnerstag vereinbaren sollen. Jede Gruppe schreibt dem TA eine E-Mail mit drei passenden Zeitfenstern mit Priorisierung.

Entwerfen Sie einen Algorithmus, der die Meetings für den TA plant so dass es keine Konflikte gibt.

Beispiele für Daten, die der TA von den Studenten erhält:

- Gruppe 1 reicht folgende Termine ein:
 - 1. Priorität: 9 - 10 Uhr
 - 2. Priorität: 10 - 11 Uhr
 - 3. Priorität: 11 - 12 Uhr
- Gruppe 2 reicht folgende Termine ein:

- 1. Priorität: 10 - 11 Uhr
 - 2. Priorität: 11 - 12 Uhr
 - 3. Priorität: 13 - 14 Uhr
- Gruppe 1 reicht folgende Termine ein:
- 1. Priorität: 9 - 10 Uhr
 - 2. Priorität: 11 - 12 Uhr
 - 3. Priorität: 16 - 17 Uhr

2.5 Lösungen

Begabtenförderung

Um den Algorithmus zur Auswahl der Kandidaten zu erstellen wird das Problem zuerst in kleinere Probleme zerlegt. Anschliessend werden Gemeinsamkeiten zwischen den Teilproblemen identifiziert und die Daten abstrahiert. Zum Schluss wird der Algorithmus definiert.

Dekomposition

Das Problem lässt sich in folgende Teilprobleme unterteilen:

- Überprüfe Förderungsfähigkeit der Bewerber*in:
- Alter von höchstens 28 Jahren
- Kenntnisse von mindestens zwei Amtssprachen
- Notenschnitt des Maturitätsabschluss von mindestens 5.3
- Bewerte Bewerber*in (CVs) basierend auf gesellschaftlichem Engagement, breiten Interessen, intellektueller Neugier und Kreativität
- Erstelle Rangliste und wähle Top 20 aus

Abstraktion & Muster

Die Bewerbungen werden alle nach gleichem Schema/Ablauf bearbeitet:

- Überprüfung der Förderungsfähigkeit ist gleich für alle Bewerber*in
- Bewertung sollte für Bewerber*innen gleich ablaufen

Bei der Erstellung des Algorithmus legt man den Fokus auf die für das Problem relevanten Eigenschaften der Bewerber*innen. Diese sind Charakteristiken, die für die Überprüfung der Aufnahmekriterien notwendig sind, dazu gehören unter anderem Alter, Sprachkenntnisse, Notendurchschnitt, Anzahl gesellschaftlicher Engagements, Musikinstrumente.

Algorithmus

Durch die Zerkleinerung des Problems, sowie Mustererkennung und Abstraktion kann der Algorithmus einfacher hergeleitet werden. Der Algorithmus wählt die Top 20 Kandidaten aus den Bewerbungen aus. Dazu wird für jede eingegangene Bewerbung zuerst überprüft, ob der/die Kandidat*in förderfähig ist und die Aufnahmekriterien erfüllt. Anschliessend wird für jede/jeden Kandidat*in bewertet indem eine Punktzahl berechnet wird.

1. Für jede*n Bewerber*in
 1. Prüfe ob Bewerber*in förderfähig ist: Alter <= 28 Jahre && Anzahl Sprachen >= 2 Amtssprachen && Notenschnitt >= 5.3
 2. Bewerte CV / Berechne Punktezahl
 1. +1 Punkt pro gesellschaftlichen Engagement (sozialer Verein, etc.)
 2. +1 Punkt pro Musikinstrument
 3. +1 Punkt pro extracurricularem Projekt
 4. + ((Notenschnitt – 5.3) * 10) Punkte // +1 pro 0.1 im Notenschnitt über 5.3
 5. + (28 – Alter) Punkte // je schneller zum Studium desto besser
 2. Erstelle Rangliste basierend auf Bewertungen für jeden Bewerber*in
 3. Wähle die Top 20 der Rangliste aus

('//' bedeutet Kommentar, d. h. ist keine ausführbare Aktion im Algorithmus)

Meetingplanung

Ein möglicher Algorithmus, der die Termine für den TA plant kann wie folgt aussehen:

1. Gehe durch alle Gruppen nach Zeitpunkt der Email (first come, first serve)
 - 1.1 Für aktuelle Gruppe, gehe durch die 3 Zeitfenster nach Priorisierung
 - 1.1.1 Ist das aktuelle Zeitfenster im TA Kalender frei?
Ja: weise dieses Zeitfenster zu und gehe zur nächsten Gruppe in Schritt 1
Nein: gehe zum nächsten Zeitfenster und wiederhole Schritt 1.1.1
2. Sende zugewiesene Zeitfenster an Studierende

2.6 Neutralität von Algorithmen

Algorithmen sind nicht zwingend neutral, d.h. dass sie aus verschiedenen Gründen bestimmte Personengruppen oder Minderheiten diskriminieren. Denn Algorithmen werden von Menschen entworfen und basieren auf Daten, die unsere Realität widerspiegeln. Der Hauptgrund liegt darin, dass Menschen einen Bias haben (können).

Bias ist ein Überbegriff für Vieles, von Vorurteilen über Verzerrungen in Entscheidungen bis hin zur Förderung oder Vernachlässigung bestimmter gesellschaftlicher Gruppen. Ein Bias ist nicht unbedingt gewollt oder überhaupt bewusst und man unterscheidet generell zwei Arten von Bias: bewusster und unbewusster Bias. Beim **bewussten Bias** weiss die Person, dass sie einen Bias, also zum Beispiel Vorurteile, hat. Beim **unbewussten Bias** kennt die Person ihre Vorurteile/Verzerrungen nicht.

Ein Beispiel des unbewussten Bias ist der Affinity Bias, oder auch Similarity Bias. Bei diesem Bias erfahren Menschen eine Tendenz sich mit Leuten verbunden zu fühlen, die einen ähnlichen Hintergrund/Interessen oder Erfahrungen haben wie sie selbst. Dies kann beispielsweise zu Implikation bezüglich Diversität am Arbeitsplatz haben, denn man wählt verstärkt die Bewerber aus die einem ähnlich sind.

Die **Bias-Blindheit** ist die Tendenz von Menschen zu glauben, dass sie selbst keinen Bias haben. Es gibt Tests wie den «**Implicit Association Test**¹» den man durchführen kann um sich selbst auf gewissen unbewussten Bias zu testen.

Bei Algorithmen kann Bias auch häufig vorkommen. Man spricht dabei von einem **algorithmischen Bias**:

«*Algorithmischer Bias beschreibt systematische und wiederholbare Fehler in einem Computersystem, die unfaire Ergebnisse erzeugen, wie zum Beispiel die Privilegierung einer willkürlichen Gruppe von Benutzern gegenüber anderen.*»

¹<http://www.understandingprejudice.org/iat/>

In der Theorie können Systeme dabei helfen, Diskriminierung sichtbar zu machen und abzubauen. Aber mit zunehmender Entscheidungsübernahme von Algorithmen über das menschliche Leben, wird die Problematik von algorithmischen Verzerrungen verstärkt. Ein **Grund** für algorithmischen Bias kann der/die Ersteller*in/Benutzer*in selbst sein, denn Algorithmen agieren nicht unabhängig von Menschen, die sie erstellen oder einsetzen. Dies führt dazu, dass die implizite Werte und damit auch der bewusste als auch der unbewusste Bias der Entwickler*innen im Algorithmus anzutreffen ist. Des weiteren sind vor allem datenbasierte Algorithmen nur so gut wie die Daten, mit denen sie arbeiten. Ein historischer Bias in Daten kann zu struktureller Benachteiligung führen, z. B. im Bereich Recruiting: wenn bei Bewerbungen eine Vorauswahl durch ein Computer Programm getroffen wird, das dann zum Beispiel automatisch Personen über 60 Jahre oder mit einem ausländischen Nachnamen aussortiert. Viele Modelle, die heutzutage zum Beispiel bei Suchmaschinen oder anderen Applikationen im Hintergrund eingesetzt werden, werden häufig auf grossen existierenden Datenmengen, wie allen Wikipedia Einträgen trainiert. Dabei kann ein inhärenter Bias auftreten, denn diese Einträge werden auch vorwiegend von bestimmten Menschengruppen geschrieben die zum Beispiel den Zugang und die Zeit dafür haben.

Aufgabe 1 Bias im Algorithmus

Diese Aufgabe basiert auf dem erstellten Algorithmus zur Auswahl der Top 20 Kandidaten, die eine Begabtenförderung erhalten. Die besten Kandidaten werden aus einer Rangliste ausgelesen, die sich aus einer Punktzahl von verschiedenen Kriterien, wie z. B. Sprachkenntnisse, gesellschaftliches Engagement, intellektuelle Neugier, zusammensetzt.

Schauen Sie Sich Ihren Algorithmus an und überlegen Sie, ob es möglich wäre, dass ein Bias im Algorithmus vorliegt. Falls ja, wo liegt dieser Bias vor?

Aufgabe 2 Bias im Algorithmus

Unter den Bewerbern gibt es folgende zwei Kandidaten:

Anna K.	Felix M.
<ul style="list-style-type: none"> • 24 Jahre alt • Mexikanische Staatsbürgerschaft • seit mehreren Jahren in der Schweiz wohnhaft • Notendurchschnitt von 5.7 bei der Matura für Erwachsene • Alleinerziehende Mutter, Unterstützung ihrer Familie in Mexiko 	<ul style="list-style-type: none"> • 20 Jahre alt • Schweizer Staatsbürgerschaft • wohnhaft in der Schweiz • Notendurchschnitt von 5.4 in der Kantonsschule • Mitglied im Orchester und Schachclub • Spielt E-Gitarre, Kontrabass und Violine • Freiwilliger Helfer bei Ponte-Sano

Gehen Sie den Algorithmus für beide Kandidaten durch und schauen Sie, wie viele Punkte jeder der beiden erhält. Überlegen Sie, ob dies gerechtfertigt ist und/oder ob der Algorithmus angepasst werden sollte, und falls ja, wie.

Lösung zur Aufgabe 1 Bias im Algorithmus

Ja, dieser Algorithmus enthält einen Bias. Basierend auf unseren Erwartungen haben wir Annahmen getroffen, die unbewusst und ohne Absicht eine gewisse Personengruppe benachteiligt.

Lösung zur Aufgabe 2 Bias im Algorithmus

Felix erhält insgesamt 6 Punkte, Anna hingegen nur 4 Punkte. Obwohl Anna eine gute Maturanote hat, hat sie neben ihren familiären Pflichten nicht viel Zeit oder auch nicht das Geld um sich gesellschaftlich in Vereinen zu engagieren und/oder Musikinstrumente zu spielen.

Referenzen

- Angwin, J., Larson, J., Mattu, S. & Kirchner, L. (2016) Machine Bias. ProPublica. Verfügbar unter <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing> (letzter Zugriff 11. Juni 2020)
- Balkow C., Eckardt I (2019) Denkimpuls Digitale Ethik: Bias in algorithmischen Systemen – Erläuterungen, Beispiele und Thesen. Initiative D21 e. V. Verfügbar unter Denkimpuls Digitale Ethik: Bias in algorithmischen Systemen (letzter Zugriff 11. Juni 2020)
- Bock L., Welle B. (2014). You don't know what you don't know: How our unconscious minds undermine the workplace. Official Blog Google. Verfügbar unter <https://googleblog.blogspot.com/2014/09/you-dont-know-what-you-dont-know-how.html> (letzter Zugriff 11. Juni 2020)
- Chalmers J & Watts T. (2014). Kids should code: why ‘computational thinking’ needs to be taught in schools. The Guardian. Abgerufen von <https://www.theguardian.com/commentisfree/2014/dec/19/kids-should-code-why-computational-thinking-needs-to-be-taught-in-schools> (letzter Zugriff am 9. Juni 2020)
- Curzon, P., & McOwan, P. W. (2018). Computational Thinking: Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln. Springer-Verlag.
- Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in <http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>.
- Dvorsky G. (2014). The 10 Algorithms that dominate our world. Gizmodo. Abgerufen von <https://io9.gizmodo.com/the-10-algorithms-that-dominate-our-world-1580110464> (letzter Zugriff am 09. Juni 2020)
- Lustenberger B. (2018). Ein einfacher Sortieralgorithmus, Leitprogrammatische Unterrichtsunterlagen von Bruno Lustenberger.
- Papert, S. (1996). An exploration in the space of mathematics educations. IJ Computers for Math. Learning, 1(1), 95-123.
- Tan J. (2015). Computational thinking an essential skill for next-generation. Today Online. Abgerufen von <https://www.todayonline.com>

/singapore/computational-thinking-essential-skill-next-generation
(letzter Zugriff am 9. Juni 2020)

- Schwendimann B. A. (2019). Computational Thinking – where do Swiss school stand today?. Digital Switzerland. Abgerufen von <https://digitalswitzerland.com/2019/06/12/computational-thinking-where-do-swiss-schools-stand-today/> (letzter Zugriff am 9. Juni 2020)
- Social Psychology Network. Implicit Association Test. Verfubar unter <http://www.understandingprejudice.org/iat/> (letzter Zugriff am 11. Juni 2020)
- Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.

© Digital Society Initiative

2.7 Folien



**Universität
Zürich**
UZH

Digital Society Initiative



Studium Digitale

Kursbaustein 8: Computational Thinking (CT)

Lektion 1: Einführung zu Computational Thinking

Prof. Thomas Fritz, Institut für Informatik

Lernziele

Am Ende dieser Vorlesung wissen Sie ...

- wie Algorithmen das tägliche Leben beeinflussen,
- was Computational Thinking (CT) bedeutet,
- wieso CT eine wichtige Fähigkeit ist.

Algorithmen verändern die Welt

Algorithmen sind praktisch überall und nicht wegzudenken

Einflussreiche Algorithmen:

- Google Suche 
- Facebook Newsfeed 
- Produkt-Empfehlungen  

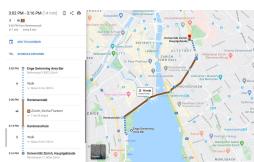


Computational Thinking



Reales Problem

Computational
Thinking



Algorithmus & Lösung

Computational Thinking – Definition

«Fähigkeit, ein Problem in kleinere Probleme zu zerlegen, die deterministisch gelöst werden können.» – Wing (2006) & Papert (1996)

«Die **Denkprozesse**, die bei der Formulierung von **Problemen** und deren **Lösungen** ablaufen, so dass die Lösungen in einer Form dargestellt werden, die von einem informationsverarbeitenden Agenten effektiv durchgeführt werden kann.» – Cuny, Snyder, Wing (2011)

Computational Thinking – Beispiel



ZVV Route zur UZH

1. Finde nahegelegene ZVV Haltestellen
2. Schaue nach welche Linien Richtung UZH fahren
3. Fährt eine direkt zur UZH?
4.



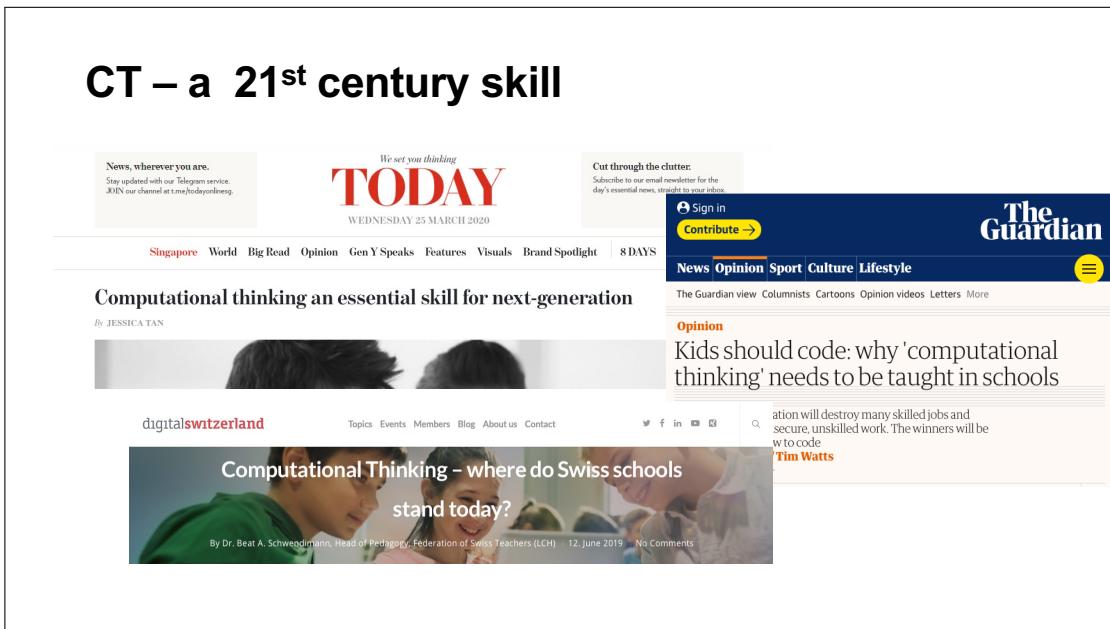
**Algorithmen sind
überall, nicht nur
in Software**

«**Computational Thinking** ist eine Art und Weise, wie Menschen Probleme lösen; es ist nicht der Versuch, Menschen dazu zu bringen, wie Computer zu denken. Computer sind stumpfsinnig und langweilig; Menschen sind klug und einfallsreich. Wir Menschen machen Computer spannend. »

- Wing (2006)

Was ist CT und was ist es nicht?

- Konzeptualisierung **nicht** Programmierung
- Fokus auf Ideen & Prozessen, **nicht** Software & Hardware
- Für jeden, **nicht** nur für Informatiker*innen



Übersicht CT Baustein

- Einführung & Definition
- Hauptbestandteile von CT
- Fall-Beispiele
- Neutralität von Algorithmen

Referenzen

- Chalmers J & Watts T. (2014). Kids should code: why 'computational thinking' needs to be taught in schools. *The Guardian*. Abgerufen von <https://www.theguardian.com/commentisfree/2014/dec/19/kids-should-code-why-computational-thinking-needs-to-be-taught-in-schools> (letzter Zugriff am 9. Juni 2020)
- Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. *Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf*.
- Dvorsky G. (2014). The 10 Algorithms that dominate our world. *Gizmodo*. Abgerufen von <https://io9.gizmodo.com/the-10-algorithms-that-dominate-our-world-1580110464> (letzter Zugriff am 09. Juni 2020)
- Papert, S. (1996). An exploration in the space of mathematics educations. *IJ Computers for Math. Learning*, 1(1), 95-123.
- Tan J. (2015). Computational thinking an essential skill for next-generation. *Today Online*. Abgerufen von <https://www.todayonline.com/singapore/computational-thinking-essential-skill-next-generation> (letzter Zugriff am 9. Juni 2020)
- Schwendimann B. A. (2019). Computational Thinking – where do Swiss school stand today?. *Digital Switzerland*. Abgerufen von <https://digitalswitzerland.com/2019/06/12/computational-thinking-where-do-swiss-schools-stand-today/> (letzter Zugriff am 9. Juni 2020)
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative



Universität
Zürich UZH

Digital Society Initiative



Studium Digitale

Kursbaustein 8: Computational Thinking

Lektion 2: Hauptbestandteile von Computational Thinking – Teil 1

Prof. Thomas Fritz, Institut für Informatik

Lernziele

Am Ende dieser Vorlesung können Sie ...

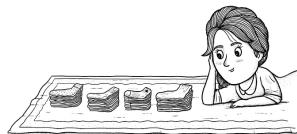
- erklären was die vier Hauptbestandteile von CT sind,
- Mustererkennung, Dekomposition und Abstraktion auf ein gegebenes Problem anwenden.

Beispiel aus dem Alltag – Wäsche zusammenlegen



CT

- Sortieren
 - Farbe
 - Grösse
- Zusammenlegen

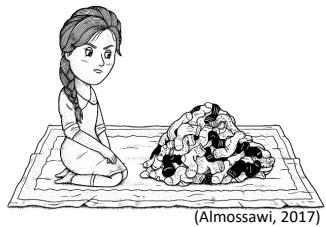


(Almossawi, 2017)



(Almossawi, 2017)

Dekomposition: sortieren, zusammenlegen



(Almossawi, 2017)

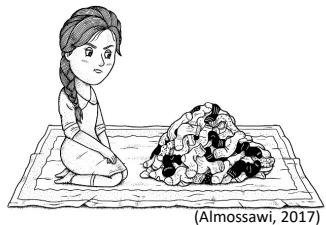
Dekomposition
Mustererkennung



(Almossawi, 2017)

Dekomposition
Mustererkennung
Abstraktion
Fokussieren auf das
Wesentliche / Wichtigste





(Almossawi, 2017)

Dekomposition

Mustererkennung

Abstraktion

Algorithmus: Ablauf festlegen

Spielt eine grosse Rolle

Hauptkomponenten von Computational Thinking

- Dekomposition
- Mustererkennung
- Abstraktion
- *Algorithmus*

Methoden der Dekomposition

- Zielreduktion (z.B. Problem halbieren)
- Rekursive Zielreduktion: wiederholte Anwendung der Problemzerlegung



Beispiel: Dekomposition

Schreiben Sie einen Aufsatz über ein selbstgewähltes Thema im Bereich Deep Learning

Teilprobleme / Schritte:

- Suche und Festlegung auf Thema
- Tiefere Recherche zum Thema
- Strukturieren/Aufteilen des Aufsatzes: Einleitung, Hauptteil, etc.
- Korrekturlesen
- ...



Dekomposition

Erstellung einer Smartphone Applikation “Buch2Go” für den Handel von secondhand Büchern

Aufgabe: Nehmen Sie Sich ein paar Minuten Zeit und überlegen Sie, wie Sie vorgehen würden, um dieses komplexe Problem zu lösen, und in welche Teilprobleme man es zerlegen kann.



Dekomposition

Erstellung einer Smartphone Applikation “Buch2Go” für den Handel von secondhand Büchern

- ⌚ Budget?
- ⚙️ Funktionalitäten?
Erfassen, Suchen, Kaufen, ...
- 👤 Zielpublikum?
- taboola Marketing?
- 📅 Zeitplan?

Hauptkomponenten von Computational Thinking

- Dekomposition
- **Mustererkennung**
- Abstraktion
- *Algorithmus*

Mustererkennung

Ähnlichkeiten & Gemeinsamkeiten zwischen und / oder innerhalb von Problemen identifizieren

- Vereinfachung der Problemlösung durch Wiederverwendung der gleichen Lösung



Beispiel: Mustererkennung

Entwerfe Design von zwei Küchen

Gemeinsame Muster

- Herd
- Kühlschrank
- Waschbecken
- Arbeitsfläche
- ...



Mustererkennung



“Buch2Go” Funktionalität: Erfassen von verschiedenen Büchern die zum Verkauf oder Tausch angeboten werden

Aufgabe:

Nehmen Sie Sich ein paar Minuten Zeit und überlegen Sie ob es bei dieser Funktionalität gewisse Muster / Ähnlichkeiten / Gemeinsamkeiten gibt.



Mustererkennung

“Buch2Go” Funktionalität: Erfassen von verschiedenen Büchern die zum Verkauf oder Tausch angeboten werden

Ablauf Erfassung eines Buches für alle Benutzer und alle Bücher gleich:

- Titel, ISBN, Zustand des Buches und Beschreibung angeben
- Preis festlegen
- Optional Foto hochladen



Mustererkennung

Buch2Go-App: **zusätzlich zur Smartphone App wird eine Web App erstellt**

Aufgabe: Gibt es Gemeinsamkeiten zwischen den zwei Problemen: Smartphone App und Web App?



Mustererkennung

Buch2Go-App: zusätzlich zur Smartphone App wird eine Web App erstellt

Gemeinsame Teilprobleme: Funktionalität, Marketing, Zielgruppe, etc.
→ Wiederverwendung von Lösungen



Hauptkomponenten von Computational Thinking

- Dekomposition
- Mustererkennung
- **Abstraktion**
- *Algorithmus*

Abstraktion & Datenrepräsentation

Einteilung in wichtige und unwichtige Eigenschaften des Problems;
Fokus auf relevante Aspekte

Arten

- Abstraktion der Steuerung
- Datenabstraktion



Datenabstraktion

Buch2Go-App – Buch



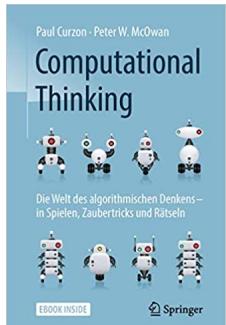
Aufgabe: Welche Eigenschaften eines Buches sind relevant bei der Erfassung in der App?



Datenabstraktion

Buch2Go-App – Buch





The book cover for "Computational Thinking" by Paul Curzon and Peter W. McOwan. It features a blue background with white text. The title "Computational Thinking" is prominently displayed in large letters. Below the title, it says "Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln". There are several small robot icons at the bottom. The Springer logo is in the bottom right corner.

Produktinformation

Taschenbuch: 243 Seiten
Verlag: Springer; Auflage: 1. Aufl. 2018 (25. Juli 2018)
Sprache: Deutsch
ISBN-10: 3662567733
ISBN-13: 978-3662567739
Größe und/oder Gewicht: 19,3 x 1,3 x 22,9 cm



Datenabstraktion

Buch2Go-App – Buch



Relevante Eigenschaften

- ✓ Autor*in
- ✓ Titel
- ✓ ISBN

...

Irrelevante Eigenschaften

- ✗ Farbe des Covers
- ✗ Geburtsort Autor*in
- ✗ Schriftgrösse

...

Hauptkomponenten von Computational Thinking

- Dekomposition
- Mustererkennung
- Abstraktion
- *Algorithmus*

Referenzen

Almossawi, A. (2017). *Bad Choices: How Algorithms Can Help You Think Smarter and Live Happier*. Penguin.

Curzon, P., & McOwan, P. W. (2018). *Computational Thinking: Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln*. Springer-Verlag.

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative



**Universität
Zürich^{UZH}**
Digital Society Initiative

Studium Digitale

Kursbaustein 8: Computational Thinking

Lektion 2: Hauptbestandteile von Computational Thinking – Teil 2

Prof. Thomas Fritz, Institut für Informatik

Lernziele

Am Ende dieser Vorlesung können Sie ...

- den Algorithmus-begriff erklären,
- Algorithmen für ein Problem entwerfen und in Pseudocode erfassen,
- Algorithmen nachvollziehen,
- die Korrektheit von Algorithmen bewerten.

Pfannkuchen Rezept

Zutaten für 4 Portionen

400 g Mehl

750 ml Milch

...

Zubereitung (1h30)

1. Trenne Eiweiss vom Eigelb
2. Mehl, Milch, Eigelb und Prise Salz verrühren

...

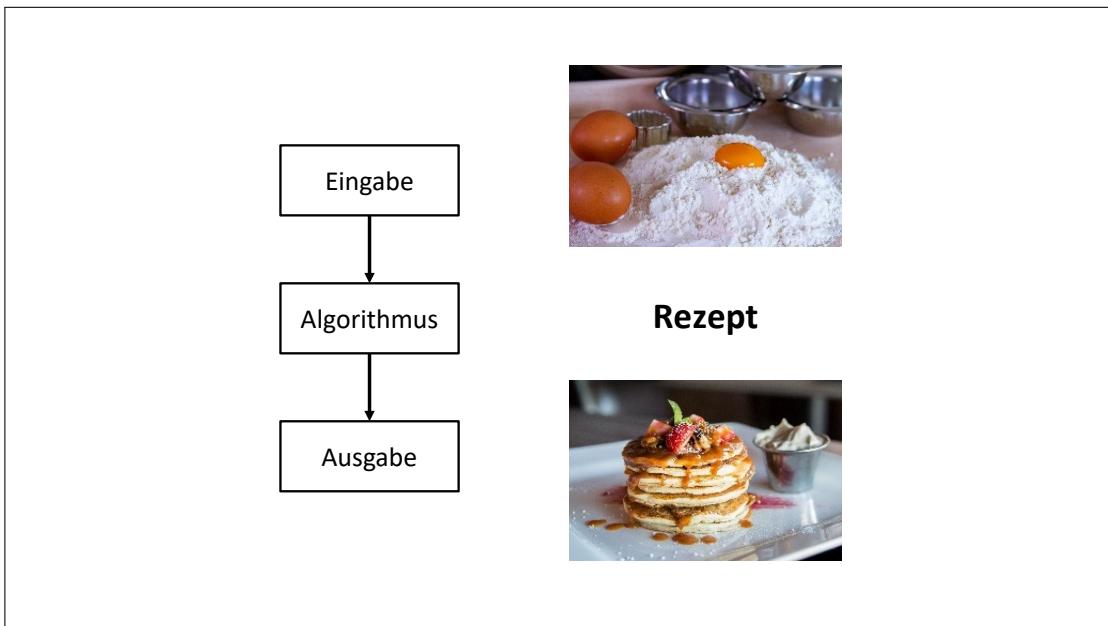


Algorithmus – Definitionen

«Ein Algorithmus ist eine endliche Folge wohldefinierter Anweisungen, typischerweise zur Lösung einer Klasse von Problemen.»

«[...]. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten. [...] Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.»

– Leiserson et al. (2010)



Beispiel: Algorithmus

Erstelle einen Algorithmus für den Kauf eines Zug-Tickets von Zürich nach Bern in der SBB-App.

SBB Ticket

1. Auf Zeitplan tippen
2. Felder «Von», «Nach», «Datum» ausfüllen
3. Zugverbindung auswählen
4. Ticket kaufen

Kriterien von Algorithmen

- Maschinentauglichkeit
- Allgemeinheit
- Korrektheit

Können Algorithmen jedes Problem lösen?

Nein, Halte-Problem ist ein ungelöstes Problem



Algorithmus



Kauf eines Buches in “Buch2Go”

Aufgabe: Nehmen Sie Sich ein paar Minuten Zeit und erstellen Sie einen Algorithmus für den Kauf eines Buches in der Buch2Go-App. Nehmen Sie an, der Benutzer ist bereits registriert.



Algorithmus



Kauf eines Buches in “Buch2Go”

1. User meldet sich an
2. Eingabe des Titels im Suchtext und suchen drücken
3. Buch auswählen und in den Warenkorb legen
4. Kauf abschliessen



Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?

100 Kürbisse liegen nebeneinander

1. Fange von ganz links (1. Kürbis) an
 - a) Vergleiche den aktuellen Kürbis mit seinem rechten Nachbarn
 - b) Ist der Kürbis grösser als sein Nachbar, dann vertausche sie
 - c) Gehe zum rechten der zwei Kürbisse und wiederhole Schritte a)
bis c)
2. Wiederhole den 1. Schritt 99 mal



Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?



4 Kürbisse als Beispiel

1. Fange von ganz links (1. Kürbis) an



Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?



1. Iteration

- a) Vergleiche den **aktuellen** Kürbis mit seinem **rechten** Nachbarn
- b) Ist der Kürbis grösser als sein Nachbar, dann vertausche sie



Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?



1. Iteration

- a) Vergleiche den aktuellen Kürbis mit seinem rechten Nachbarn
- b) Ist der Kürbis grösser als sein Nachbar, dann vertausche sie

Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?

1. Iteration

c) Gehe zum rechten der zwei Kürbisse (blau) und wiederhole Schritte a) bis c)

Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?

1. Iteration

linker Kürbis > rechter Kürbis? Nein
→ nicht tauschen, gehe an die Position des rechten Kürbis (rot)

Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?

1. Iteration

The diagram shows four pumpkins in a row. The first two are highlighted with blue boxes, and the last two with orange boxes. A black arrow points from the blue box to the orange box, indicating a swap between the first and second pumpkins. Below the row, the pumpkins are shown again, with the first two swapped positions highlighted by their original colors.

linker Kürbis > rechter Kürbis? Ja → vertausche sie

Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?

1. Iteration

The diagram shows the state of the four pumpkins after the first iteration. The first two pumpkins are highlighted with blue boxes, and the last two with orange boxes. A black arrow points from the blue box to the orange box, indicating a swap between the first and second pumpkins. Below the row, the pumpkins are shown again, with the first two swapped positions highlighted by their original colors.

Ende erster Iteration.
Starte die nächste Iteration wieder beim 1. Kürbis ganz links ...



Algorithmus

Aufgabe:
Was bewirkt/beschreibt der folgende Algorithmus?

Pseudocode Bubble Sort

```
BubbleSort(Array array)
    for i=0 to array.length - 2 {
        for j=0 to array.length - 2 {
            if array[j] > array[j+1]
                swap(array[j], array[j+1])
        }
    }
```

Referenzen

Curzon, P., & McOwan, P. W. (2018). *Computational Thinking: Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln*. Springer-Verlag.

Leiserson, C. E., Rivest, R., & Stein, C. (2010). *Algorithmen-Eine Einführung*. Oldenbourg Verlag, München

Lustenberger Bruno, Ein einfacher Sortieralgorithmus, Leitprogrammatische Unterrichtsunterlagen von Bruno Lustenberger, EDUCETH Das Bildungsportal der ETH Zürich, verfügbar unter [https://ethz.ch/content/dam/ethz/special-interest/dual/educeth-dam/documents/Unterrichtsmaterialien/informatik/Einfacher%20Sortieralgorithmus%20\(Leitprogrammatische%20Unterrichtsunterlagen\)/Lustenberger_SlectionSort_jun_08.pdf](https://ethz.ch/content/dam/ethz/special-interest/dual/educeth-dam/documents/Unterrichtsmaterialien/informatik/Einfacher%20Sortieralgorithmus%20(Leitprogrammatische%20Unterrichtsunterlagen)/Lustenberger_SlectionSort_jun_08.pdf) (letzter Zugriff am 9. Juni 2020)

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative

Lernziele

Am Ende dieser Vorlesung ...

- kennen Sie sieben weitere Fertigkeiten für gutes CT,
- können Sie die Fertigkeiten an einem Beispiel erklären / anwenden.

Beispiel aus dem Alltag

Im Auto durch die Stadt navigieren



↓ CT



CT-Fertigkeiten

- **Modellierung**



CT-Fertigkeiten

- Modellierung
- **Wissenschaftliches Denken**



CT-Fertigkeiten

- Modellierung
- Wissenschaftliches Denken
- **Logisches Denken**



CT-Fertigkeiten

- Modellierung
- Wissenschaftliches Denken
- Logisches Denken
- **Kreativität**



CT-Fertigkeiten

- Modellierung
- Wissenschaftliches Denken
- Logisches Denken
- Kreativität
- **Menschenkenntnis**



Beispiel: Menschenkenntnis

Algorithmus zum sicheren Flussüberqueren

Beispiel aus Curzon & McOwan (2018)

Eine Bäuerin ist mit ihrem Schäferhund Maxi, der sie immer begleitet, auf dem Weg zum nächsten Dorf. Um dorthin zu gelangen, muss sie einen reisenden Fluss überqueren. Dafür hat eine Erfinderin, die auf der Dorfseite des Flusses wohnt, eine Vorrichtung erfunden, die die Flussüberquerung ermöglicht. Sie besteht aus einem Seil, Rollen und einem Sitz, der an einem Seil hängt und Platz für eine Person bietet. Die Bewohner des Ortes haben vereinbart, den Sitz immer auf der Seite des Dorfes zu lassen, wo die Erfinderin wohnt.



Beispiel: Menschenkenntnis

Algorithmus zum sicheren Flussüberqueren

Beispiel aus Curzon & McOwan (2018)

Als die Bäuerin zum Fluss kommt, zieht sie den Sitz mit dem Seil zu sich herüber, setzt sich hinein, nimmt Maxi auf den Arm, zieht sich selbst auf die andere Seite und setzt ihren Weg zum Dorf fort. Eines Tages kauft sie eine Henne und einen Sack Reis. Als sie auf dem Heimweg zum Fluss kommt, stellt sie fest, dass sie nur ein Ding mit sich hinübernehmen kann, wenn sie den Sitz verwenden möchte. Deshalb benötigt sie mehrere Fahrten. Dabei gibt es folgendes Problem:



Beispiel: Menschenkenntnis

Algorithmus zum sicheren Flussüberqueren

Beispiel aus Curzon & McOwan (2018)

Wenn sie die Henne mit dem Getreide auf einer Seite allein lässt, wird die Henne das Getreide fressen. Lässt sie Hund Maxi und die Henne auf einer Seite zusammen, wird der Hund die Henne jagen. Maxi frisst kein Getreide, deshalb kann er mit dem Getreide allein gelassen werden.

Aufgabe:

Schreiben Sie einen Algorithmus um alle drei Sachen auf die andere Flussseite zu bringen, ohne dass etwas gefressen oder gejagt wird.
Beachten Sie: Sowohl der Hund und die Henne als auch die Henne und das Getreide voneinander getrennt gehalten werden müssen.



Beispiel: Menschenkenntnis

Algorithmus zum sicheren Flussüberqueren

Beispiel aus Curzon & McOwan (2018)

1. Sie fährt mit der Henne auf die andere Seite.
2. Sie kehrt alleine zurück.
3. Sie fährt mit dem Hund auf die andere Seite.
4. Sie kehrt mit der Henne zurück.
5. Sie fährt mit dem Getreide auf die andere Seite.
6. Sie kehrt alleine zurück.
7. Sie fährt mit der Henne auf die andere Seite.



Beispiel: Menschenkenntnis

Algorithmus zum sicheren Flussüberqueren

Beispiel aus Curzon & McOwan (2018)

1. Sie fährt mit der Henne auf die andere Seite.
2. Sie kehrt alleine zurück.
3. Sie fährt mit dem Hund auf die andere Seite.
4. Sie kehrt mit der Henne zurück.
5. Sie fährt mit dem Getreide auf die andere Seite.
6. Sie kehrt alleine zurück.
7. Sie fährt mit der Henne auf die andere Seite.
- 8. Sie schickt den leeren Sitz zum anderen Ufer zurück.**

Menschen verstehen

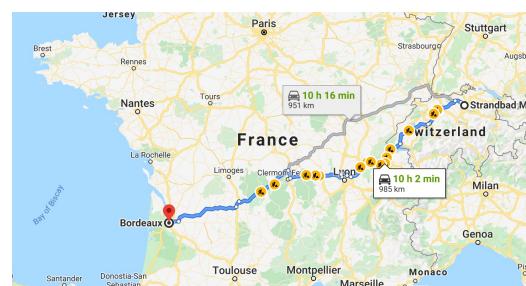
Post-Completion Error

Weiteres Beispiel: Geldautomat



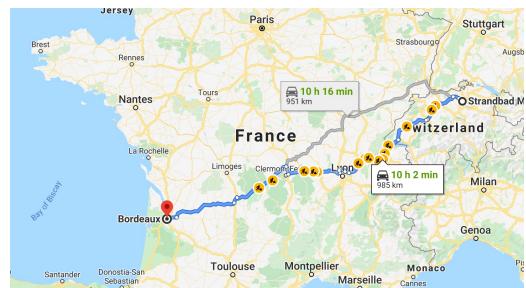
CT-Fertigkeiten

- Modellierung
- Wissenschaftliches Denken
- Logisches Denken
- Kreativität
- **Menschenkenntnis**



CT-Fertigkeiten

- Modellierung
- Wissenschaftliches Denken
- Logisches Denken
- Kreativität
- Menschenkenntnis
- **Heuristiken**



Weitere Aspekte von CT

- Computermodellierung
- Wissenschaftliches Denken
- Logisches Denken
- Kreativität
- Menschenkenntnis
- Heuristiken
- **Evaluation**

Evaluation

Überprüfung auf

- Funktionale Richtigkeit
- Zweckerfüllung
- Leistungsfähigkeit
- Benutzerfreundlichkeit/Bedienbarkeit
- ...

Nicht nur am Ende, sondern kontinuierlich



Dekomposition
Mustererkennung
Abstraktion
Algorithmus

Modellierung
Wissenschaftliches Denken
Logisches Denken
Kreativität
Menschenkenntnis
Heuristiken
Evaluation

Referenzen

Curzon, P., & McOwan, P. W. (2018). *Computational Thinking: Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln.* Springer-Verlag.

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative

Lernziele

Am Ende dieser Vorlesung

- Sind Sie in der Lage Computational Thinking auf ein Problem anzuwenden und einen Algorithmus zu erstellen der das Problem löst.



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Jährlich erhält eine Stiftung zur Förderung eines Studiums mehrere hundert Bewerbungen, von denen sie 20 mit einem Stipendium fördert.

Aufgabe (ca. 20 Minuten):

Entwerfen Sie einen Algorithmus, der ein Rangliste der Bewerber*innen und derer CVs basierend auf den nachfolgenden Kriterien erstellt, und die besten 20 auswählt.



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Förderungskriterien:

- Notenschnitt von mindestens 5.3 (Matura, Berufsmatura)
- Alter von höchstens 28 Jahren
- Kenntnisse von mindestens zwei Amtssprachen
- Gesellschaftliches Engagement, breite Interessen, intellektuelle Neugier und Kreativität



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Tipp:

Um eine Rangliste zu erstellen, benutzen Sie ein Punktesystem, dass Bewerber*innen basierend auf gewissen Charakteristiken bewertet. Zum Beispiel könnte es pro gespieltem Musikinstrument +1 Punkt geben, usw..



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Dekomposition

- Überprüfe Förderungsfähigkeit der Bewerber*innen:
 - Alter <= 28 Jahre; Kenntnisse >= 2 Amtssprachen; Notenschnitt >= 5.3
- Bewerte Bewerber*in (CVs) basierend auf gesellschaftlichem Engagement, breiten Interessen, intellektueller Neugier und Kreativität
- Erstelle Rangliste und wähle Top 20 aus



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Abstraktion & Muster

- Bewerbungen werden alle nach gleichem Schema/Ablauf bearbeitet
 - Überprüfung der Förderungsfähigkeit ist gleich für alle Bewerber*in
 - Bewertung sollte auch gleich ablaufen für alle Bewerber*in
- Fokus auf relevante Charakteristika: Alter, Sprachkenntnisse, Note, Anzahl gesellschaftlicher Engagements, Musikinstrumente,...



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Algorithmus

1. Für jeden Bewerber*in
 1. Prüfe ob Bewerber*in förderfähig ist: Alter <= 28 Jahre && #Sprachen >= 2 Amtssprachen && Notenschnitt >= 5.3
 2. Bewerte CV / Berechne Punktezahl



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

1.2 Bewerte CV / Berechne Punktezahl

1. +1 Punkt pro gesellschaftlichen Engagement (sozialer Verein, etc.)
2. +1 Punkt pro Musikinstrument
3. +1 Punkt pro extracurricularem Projekt
4. + ((Notenschnitt – 5.3) * 10) Punkte // +1 pro 0.1 im Notenschnitt über 5.3
5. + (28 – Alter) Punkte // je schneller zum Studium desto besser



Begabtenförderung



Erstellen eines Ranglisten & Auswahl Algorithmus

Algorithmus

1. Für jeden Bewerber*in
 1. Prüfe ob Bewerber*in förderfähig ist: Alter <= 28 Jahre && #Sprachen >= 2 Amtssprachen && Notenschnitt >= 5.3
 2. Bewerte CV / Berechne Punktezahl
 3. Erstelle Rangliste basierend auf Bewertungen für jeden Bewerber*in
2. Wähle die Top 20 der Rangliste aus



Meetingplanung



Erstelle Algorithmus zur zeitlichen Planung von Meetings

Für ein Seminar gibt es drei Gruppen die mit dem Teaching Assistant ein wöchentliches Meeting am Donnerstag vereinbaren sollen. Jede Gruppe schreibt dem TA eine E-Mail mit drei passenden Zeitfenstern mit Priorisierung.

Aufgabe (ca. 20 Minuten):

Entwerfe einen Algorithmus, der die Meetings für den TA plant so dass es keine Konflikte gibt.



Meetingplanung

Erstelle Algorithmus zur zeitlichen Planung von Meetings

Icon: Calendar with two people.

Beispiel Daten für TA:

- **G1:** 09-10(1), 10-11(2), 11-12 (3)
- **G2:** 10-11(1), 11-12 (2), 13-14 (3)
- **G3:** 09-10(1), 11-12 (2), 16-17 (3)



Meetingplanung

Erstelle Algorithmus zur zeitlichen Planung von Meetings

Icon: Calendar with two people.

Algorithmus

1. Gehe durch alle Gruppen nach Zeitpunkt der Email (first come first serve)
 1. Für aktuelle Gruppe, gehe durch die 3 Zeitfenster nach Priorisierung
 1. Ist das aktuelle Zeitfenster im TA Kalender frei? [Schritt 1.1.1]
Ja: weise Zeitfenster zu und gehe zur nächsten Gruppe in Schritt 1
Nein: gehe zum nächsten Zeitfenster und wiederhole Schritt 1.1.1
 2. Sende zugewiesene Zeitfenster an Studenten



Meetingplanung

Erstelle Algorithmus zur zeitlichen Planung von Meetings



- Ist der Algorithmus TA- oder Studenten-freundlich?
- Was passiert wenn es mehr als 3 Gruppen sind?

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative

Lernziele

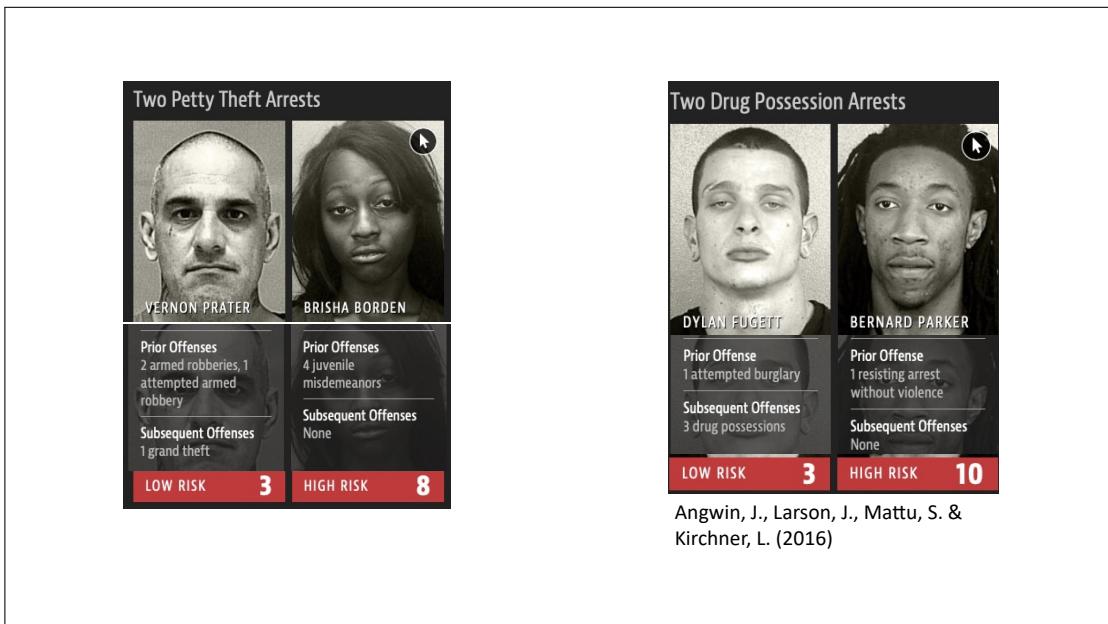
Am Ende dieser Vorlesung ...

- wissen Sie was algorithmischer Bias ist.
- Können Sie gewissen Bias in Algorithmen erkennen.

Beispiel: COMPAS Software

«*There's software used across the country to predict future criminals. And it's biased against blacks.*»

-- Angwin, J., Larson, J., Mattu, S. & Kirchner, L. (2016) *Machine Bias*.



Bias

- Vorurteile
- Verzerrungen
- Vernachlässigung gesellschaftlicher Gruppen

Arten von Bias

Bewusster Bias

- Person weiss, dass sie z.B. Vorurteile hat, beeinflusst ist, etc.

Unbewusster Bias

- Bias ist Person selber nicht bewusst, z.B. Name oder Affinity Bias

Implicit Association Test <http://www.understandingprejudice.org/iat/>

Algorithmischer Bias – Definition

«Algorithmischer Bias beschreibt systematische und wiederholbare Fehler in einem Computersystem, die unfaire Ergebnisse erzeugen, wie zum Beispiel die Privilegierung einer willkürlichen Gruppe von Benutzern gegenüber anderen.»

Gründe für algorithmischer Bias

- Algorithmen wider-spiegeln impliziten Werte der Menschen, die beim Erstellen beteiligt sind
- Daten auf denen Algorithmus basiert (z. B. im Recruiting)



Begabtenförderung & Bias

Erstellen eines Ranglisten & Auswahl Algorithmus

Recap

Entwerfe einen Algorithmus, der ein Rangliste der Bewerber*innen (CVs) basierend auf Kriterien erstellt, und die besten 20 auswählt.

Kriterien:

Notenschnitt, Alter, Sprachkenntnisse, gesellschaftliches Engagement, breite Interessen, intellektuelle Neugier und Kreativität



Begabtenförderung & Bias

Erstellen eines Ranglisten & Auswahl Algorithmus

Aufgabe

- Wäre es möglich dass ein Bias im Algorithmus vorliegt?
Falls ja, wo?



Algorithmischer Bias

Erstellen eines Ranglisten & Auswahl Algorithmus

2 Kandidaten, wer wird besser bewertet und ist dies fair?

<p>Anna K.</p> <ul style="list-style-type: none">- 24 J, mexikanische Staatsbürgerschaft, seit mehreren Jahren wohnhaft in der Schweiz- 5.7 (Matura für Erwachsene)- Alleinerziehende Mutter, Unterstützung der Familie in Mexiko	<p>Felix M.</p> <ul style="list-style-type: none">- 20 J, schweizer Staatsbürgerschaft, wohnhaft in CH- 5.4 (Kantonsschule)- Mitglied im Orchester&Schachclub- Spielt E-Gitarre, Kontrabass&Violine- Freiwilliger Helfer bei Pontesano
---	--



Algorithmischer Bias

Erstellen eines Ranglisten & Auswahl Algorithmus



Algorithmus

1. Für jeden Bewerber*in
 1. Prüfe ob Bewerber*in förderfähig ist: Alter <= 28 Jahre && #Sprachen >= 2 Amtssprachen && Notenschnitt >= 5.3
 2. Bewerte CV / Berechne Punktezahl
 1. +1 Punkt pro gesellschaftlichen Engagement (sozialer Verein, etc.)
 2. +1 Punkt pro Musikinstrument
 3. +1 Punkt pro extracurricularem Projekt
 4. + ((Notenschnitt – 5.3) * 10) Punkte // +1 pro 0.1 im Notenschnitt über 5.3
 5. + (28 – Alter) Punkte // je schneller zum Studium desto besser
 2. Erstelle Rangliste basierend auf Bewertungen für jeden Bewerber*in
 3. Wähle die Top 20 der Rangliste aus



Algorithmischer Bias

Erstellen eines Ranglisten & Auswahl Algorithmus

2 Kandidaten, wer wird besser bewertet und ist dies fair?

Anna K.	Felix M.
<ul style="list-style-type: none"> - 24 J, mexikanische Staatsbürgerschaft, seit mehreren Jahren wohnhaft in der Schweiz - 5.7 (Matura für Erwachsene) - Alleinerziehende Mutter, Unterstützung der Familie in Mexiko 	<ul style="list-style-type: none"> - 20 J, schweizer Staatbürgerschaft, wohnhaft in CH - 5.4 (Kantonsschule) - Mitglied im Orchester&Schachclub - Spielt E-Gitarre,Kontrabass&Violine - Freiwilliger Helfer bei Pontesano

Algorithmen verändern die Welt

Computational Thinking
kreieren und verstehen von
Algorithmen

FACEBOOK **amazon**
NETFLIX **Google**



Referenzen und zum Weiterlesen

Angwin, J., Larson, J., Mattu, S. & Kirchner, L. (2016) Machine Bias. *ProPublica*. Verfügbar unter <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing> (letzter Zugriff 11. Juni 2020)

Balkow C., Eckardt I (2019) Denkimpuls Digitale Ethik: Bias in algorithmischen Systemen – Erläuterungen, Beispiele und Thesen. *Initiative D21 e. V.* Verfügbar unter Denkimpuls Digitale Ethik: Bias in algorithmischen Systemen (letzter Zugriff 11. Juni 2020)

Social Psychology Network. Implicit Association Test. Verfügbar unter <http://www.understandingprejudice.org/iat/> (letzter Zugriff am 11. Juni 2020)

Bock L., Welle B. (2014). You don't know what you don't know: How our unconscious minds undermine the workplace. *Official Blog Google*. Verfügbar unter <https://googleblog.blogspot.com/2014/09/you-dont-know-what-you-dont-know-how.html> (letzter Zugriff 11. Juni 2020)

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative

3 Programmierung

PROF DR. THOMAS FRITZ UND CLAUDIA VOGEL

Studium Digitale Kursbaustein 9 Begleit-Skript

Dieses Skript bereitet die Inhalte des Kursbaustein Einführung in Programmierung schriftlich auf. Die Inhalte sind nicht identisch mit den Videos; sie wurden angepasst für die Lesbarkeit als Dokument.

3.1 Einführung in die Programmierung

Software ist heutzutage überall anzutreffen, und bestimmt in vielen Bereichen auch Teile unseres Lebens. Die Kosten für die Software eines Autos sind heute oft teurer als der Motor des Autos, und wer benutzt heute noch keine Email oder Messaging Applikation um mit Freunden zu kommunizieren. Im Kursbaustein zu Computational Thinking haben wir schon erwähnt dass Algorithmen überall sind, aber um etwas zu bewirken, müssen wir Algorithmen in Software / Programme überführen die von Computern ausgeführt werden können.

Beim Programmieren geht es darum Ideen und Lösungen zu einer Abfolge von Anweisungen zu transformieren, die dann vom Computer ausgeführt werden können. Dass heisst, Programme sagen dem Computer was genau gemacht werden soll und sie müssen dazu in einer Sprache geschrieben werden, die vom Computer verstanden wird. Doch Programmierung ist mehr als nur das Schreiben von Programmen in einer Programmiersprache. Zum Programmieren gehört es auch gute Lösungen zu Problemen zu finden. Wurde eine Lösung oder ein Algorithmus gefunden, wird dieser mit einer Programmiersprache formal ausgedrückt, so dass der Computer es ausführen kann.

3.2 Programmiersprachen

Eine Programmiersprache ist eine formale Sprache und dient wie die natürliche Sprache der Kommunikation, in diesem Fall der Kommunikation mit dem Computer. Um Computer zu steuern, ist es notwendig, ihnen klar verständliche Befehle zu geben. Da Computer auf ihrer untersten Ebene auf die Verarbeitung von 0en und 1en beschränkt sind und eine Folgen von 0en und 1en für Menschen schwer verständlich ist, wurden über die Jahre verschiedenste Programmiersprachen entwickelt um das Programmieren zu erleichtern.

Das Artefakt, das durch das Programmieren entsteht nennt sich Quellcode, Quelltext, Programmcode oder auch einfach nur Code.

3.2.1 Drei Generationen der Programmiersprachen (Peroni 2018, Boles 1999)

Programmiersprachen können grob in drei Generationen unterteilt werden:

1. Generation: Maschinensprachen

Maschinensprache ist eine Reihe von Befehlen, die direkt vom Prozessor (CPU: Central Processing Unit) eines Computers ausgeführt werden können, dass heisst, die Programmiersprach wurde sehr stark von der Hardware beeinflusst. Die Programme in Maschinensprache werden in der Regel binär kodiert, d.h. in einer Sequenz von 0 und 1 ausgedrückt. Diese Sprache ist nur begrenzt von Menschen lesbar und führt zu erheblichen Schwierigkeiten bei der Wartung und Fehlerbeseitigung.

2. Generation: Assemblersprachen

Die 2. Generation von Programmiersprachen hat eine Abstraktionsebene über dem Maschinencode eingeführt, die sogenannten Assemblersprachen. In Assemblersprachen werden die in der Maschinensprache verwendeten Zahlenfolgen durch lesbare Abkürzungen ersetzt. Dadurch kann man Programme schreiben, die für Menschen leichter lesbar und verständlicher sind. Ein sogenannter Assembler übersetzt dann wie ein Dolmetscher das Programm von Assembler-sprache in Maschinensprache. Dadurch wurden die Vorteile der hohen Ausführungs geschwindigkeit und des geringen Speicherbedarfs von Programmen beibehalten.

3. Generation: Höhere Programmiersprachen

Bei der 3. Generation handelt es sich um höhere Programmiersprachen. Höhere Programmiersprachen sind Sprachen die sich durch eine starke Abstraktion von der Maschinensprache auszeichnen und näher an der natürlichen Sprache von Menschen sind. Eine höhere Programmiersprache kann natürlich sprachliche Wörter für spezifische Konstrukte verwenden, um für den Menschen leicht benutzbar und verständlich zu sein. Im Allgemeinen gilt, dass die Programmiersprache umso verständlicher ist, je stärker die Abstraktion von der Maschinensprache gegeben ist. Die Programme, die in einer höheren Programmiersprache geschrieben sind, müssen zunächst in eine maschinenlesbare Form, beispielsweise binär in 0en und 1en, übersetzt werden, bevor sie vom Computer ausgeführt werden können.

3.2.2 Assembler, Compiler & Interpreter

Für die Übersetzung von Code in Maschinensprache wird je nach Sprache ein Assembler, Compiler, Interpreter oder eine Kombination davon verwendet. Assembler werden für die Übersetzung von Assemblersprachen genutzt. Compiler bzw. Interpreter oder eine Kombination aus beiden, werden für höhere Programmiersprachen eingesetzt. Der Hauptunterschied zwischen Compiler und Interpreter liegt darin, dass ein Compiler **vor** der Ausführung eines Programmes den gesamten Quellcode übersetzt, wohingegen der Interpreter **während** der Ausführung die benötigten Teile übersetzt.

Assembler

Ein Assembler transformiert ein Programm in Assemblersprache in ein Programm in Maschinensprache.

Compiler

Ein Compiler wandelt Quellcode aus einer höheren Programmiersprache in Maschinensprache um, er übersetzt also das gesamte Programm von einer Programmiersprache in Maschinensprache. Der Code wird vollständig übersetzt, bevor das Programm vom Computer ausgeführt wird. Deshalb benötigt er vergleichsweise mehr Zeit und Ressourcen als ein Interpreter. Sobald das fertige Programm läuft, kann es oft jedoch effizienter als interpretierte Programme ausgeführt werden, da alle Anweisungen bereits vollständig in Maschinensprache übersetzt wurden. Ein Compiler erzeugt bei vielen Sprachen plattformabhängigen Maschinencode, d. h. der erzeugte Maschinencode kann nur auf bestimmten Betriebssystemen ausgeführt werden. Reine Compiler-Sprachen sind zum Beispiel C und C++.

Interpreter

Ein Interpreter verarbeitet den Code eines Programms zur Laufzeit. Dazu geht der Interpreter Zeile für Zeile vor: Eine Anweisung wird eingelesen, analysiert und sofort ausgeführt. Dann geht es mit der nächsten Anweisung weiter. Ein Interpreter erzeugt keine Datei in Maschinensprache, die man mehrmals ausführen könnte. Da jede Anweisung bei jeder Ausführung des Programms einzeln verarbeitet wird, können interpretierte Programme langsamer als kompilierte Sprachen sein, allerdings gibt es heute auch häufig Kombinationen wie Just-In-Time-Compilation. Python oder Perl sind Beispiele für Programmiersprachen, die einen Interpreter verwenden.

3.2.3 Definition von Programmiersprachen

Programmiersprachen sind sehr exakte, künstliche Sprachen zur Formulierung von Programmen. Sie dürfen keine Mehrdeutigkeiten bei der Programmerstellung zulassen, damit der Computer das Programm auch korrekt ausführen kann. Bei der Definition einer Programmiersprache muss ihre Lexikalk, Syntax und Semantik definiert werden:

- Lexikalk: sie definiert die gültigen Zeichen bzw. Wörter, aus denen Programme der Programmiersprache zusammengesetzt sein dürfen.
- Syntax: sie definiert den korrekten Aufbau der Sätze aus gültigen Zeichen, d. h. sie legt fest, in welcher Reihenfolge lexikalisch korrekte Zeichen im Programm auftreten dürfen.
- Semantik: sie definiert die Bedeutung syntaktisch korrekter Sätze, d.h. sie beschreibt, was passiert, wenn beispielsweise bestimmte Anweisungen ausgeführt werden.

3.2.4 Programmierparadigmen

Bevor ein Programm erstellt wird, wird entschieden mit welcher Programmiersprache es umgesetzt wird. Es gibt viele verschiedene höhere Programmiersprachen, die es erlauben Ideen und Lösungen auf unterschiedliche Art und Weise auszudrücken und zu lösen. Jede weist ihre eigenen Vor- und Nachteile auf. Generell kann man Programmiersprachen nach Paradigmen kategorisieren. Bei einem Programmierparadigma geht es um den grundlegenden Stil, in dem ein Programm geschrieben wird. Es beschreibt welche Prinzipien und Denkmuster angewandt und welche Herangehensweisen genutzt werden. Programmierparadigmen sind beispielsweise die deklarative, imperitative, objektorientierte, funktionale, oder logische Programmierung. Dabei gibt es keine exakte Definitionen der einzelnen Paradigmen und gewisse Programmiersprachen können auch mehreren Paradigmen gleichzeitig zugewiesen werden.

Je nach zu lösendem Problem eignen sich manche Programmierparadigmen besser als andere, denn sie bestimmen wie wir unsere Ideen in Code ausdrücken können. Wenn wir zum Beispiel ein Programm schreiben wollen bei dem wir dem Computer eine genaue Abfolge von Instruktionen geben wollen die er abarbeiten soll, dann benutzen wir oft eine imperative Programmiersprache. Wenn wir dagegen eher die reale Welt ausdrücken wollen, in der es

Objekte gibt die verschieden Verhalten haben, dann benutzen wir eher eine objektorientiert Programmiersprache.

Deklarative Programmiersprachen legen den Fokus auf die Beschreibung des gewünschten Ergebnisses anstatt die Arbeitsschritte aufzulisten. Das *funktionale* Programmierparadigma stammt aus der Mathematik, genauer aus der Theorie der Funktionen. Das zentrale Element dieses Paradigma ist die Funktion, im traditionellen mathematischen Sinne $y=f(x)$. Alle Elemente können als Funktion aufgefasst und der Code kann durch aneinander gereihte Funktionsaufrufe ausgeführt werden. Die *imperative* Programmierung ist dadurch charakterisiert, dass Programmcode aus einer Abfolge von Anweisungen oder auch Instruktionen zur Steuerung besteht. Die Idee ist dass alle Anweisungen Schritt für Schritt nach dem Prinzip «erledige zuerst das, dann das, etc.» abgearbeitet werden, d.h. der Fokus ist hier auf dem «wie» anstelle von dem «was» (deklarativ).

Objektorientierte Programmiersprachen haben in den letzten Jahren viel an Popularität gewonnen. Objektorientierung erlaubt es uns die reale Welt als Objekte zu repräsentieren die miteinander interagieren können und gewisse Eigenschaften haben. Dies ist häufig sehr nah daran, wie Menschen die reale Welt beschreiben.

3.3 Sequenzen und Variablen

Obwohl sich Programmiersprachen in vielen Aspekte unterscheiden können, gibt es gewisse grundlegende Programmierkonstrukte die in sehr vielen Sprachen anzutreffen sind. Zwei der grundlegenden logischen Strukturen in der Programmierung sind Anweisungen und Sequenzen.

3.3.1 Anweisung und Sequenzen

Eine **Anweisung** (Statement, Befehl, usw.) ist eine elementare Einheit im Programm und steht für einen einzelnen Abarbeitungsschritt im Algorithmus. Eine Anweisung wird auch häufig Statement, Kommando oder Befehl genannt. Meistens entspricht eine Anweisung auch einer Programmierzeile im Code aber die Syntax unterscheidet sich je nach Programmiersprache.

Eine Abfolge von Anweisungen, die nacheinander ausgeführt werden, nennt man **Sequenz**. Eine Sequenz kann beliebig viele Anweisungen enthalten, sie werden der Reihe nach von oben nach unten ausgeführt, aber es können keine Anweisungen übersprungen / ausgelassen werden.

3.3.2 Variablen

Während das Programm läuft, müssen oft viele verschiedene Werte (zwischen-)gespeichert werden. Dieses Speichern von einfachen Werten ist mit **Variablen** möglich. Variablen erlauben es gewisse Wert zu speichern und den gespeicherten Wert zu einem späteren Zeitpunkt auszulesen oder zu verändern. Bevor eine Variable verwendet werden kann, muss sie deklariert, d.h. erstellt werden. Die Deklaration definiert den Namen der Variable und je nach Programmiersprache auch deren Datentyp. Wenn eine Variable deklariert wird, merkt sich der Compiler/Interpreter den Namen und reserviert Speicherplatz für diese. Nach oder während der Erstellung kann in dieser Variable ein Wert gespeichert werden. Erstmaliges Speichern/Zuweisen eines Wertes nennt man Initialisierung. Erst nach der Initialisierung hat die Variable einen Wert gespeichert und kann im fortlaufenden Programm verwendet werden. D.h. man kann über den Namen der Variable auf den Wert der Variable zugreifen und diesen auslesen. Die Deklaration und Initialisierung können miteinander oder nacheinander erfolgen. Je nach Datentyp kann der Wert der Variable verändert werden, respektive kann der Variable auch ein neuer Wert zugewiesen werden.

3.3.3 Datentypen

Die zu verarbeitenden Daten, resp. die Variablen, können von unterschiedlichen Datentypen sein. Ein Datentyp beschreibt eine Menge von Datenobjekten, die alle die gleiche Struktur haben und mit denen die gleichen Operationen ausgeführt werden können. Also zum Beispiel kann ich alle Datenobjekte des Datentyps Ganze Zahlen addieren, multiplizieren oder subtrahieren. Beispiele für Datentypen sind Ganze Zahlen, Gleitkommazahlen, Zeichen(-ketten), Listen, Wahrheitswerte. Diese Datentypen können sich je nach Programmiersprache unterscheiden.

3.3.4 Fallunterscheidung

Die Fallunterscheidung ermöglicht es alternative Abläufe zu haben und so sequentielle Abläufe zu steuern. In einer Fallunterscheidung wird eine Frage gestellt, die entweder mit Ja (wahr/true) oder Nein (falsch/false) beantwortet werden kann. Je nach Antwort wählt das Programm eine von zwei Handlungsabläufen. Diese Fragenstruktur braucht einen Eingabewert, eine sogenannte Variable, die überprüft/ausgewertet wird und entweder eine Bedingung erfüllt (true) oder nicht (false).

Es gibt die einseitige und zweiseitige Fallunterscheidung. Bei einer einseitigen Fallunterscheidung werden die angegebenen Anweisungen ausgeführt, falls die Bedingung erfüllt ist. Falls die Bedingung nicht erfüllt ist, geschieht nichts. Bei der zweiseitigen Fallunterscheidung werden für beide Fälle Anweisungen angegeben und je nach Auswertung die Anweisungen des einen oder des anderen Falles ausgeführt.

3.3.5 Schleifen

Schleifen ermöglichen es einen Block von Anweisungen wiederholt auszuführen. Dabei müssen die Anweisungen nur einmal geschrieben werden und die Schleife führt diese Anweisung dann mehrmals aus, wobei spezifiziert werden kann wie oft die Anweisungen wiederholt werden sollen.

In diesem Kurs schauen wir uns zwei verschiedene Schleifentypen an:

- zählerkontrollierte Schleifen, und
- bedingungskontrollierte Schleifen.

Zählerkontrollierte Schleifen

Eine zählerkontrollierte Schleife ist ein Programmkonstrukt bei dem die Anzahl der Wiederholungen (oft spezifiziert in einer Zählervariable) von vornherein feststeht. Die Anweisungen im Schleifenkörper werden dann genau so oft ausgeführt wie durch die Anzahl vorgegeben ist.

Bedingungskontrollierte Schleifen

Bei der bedingungskontrollierten Schleife wird der Block von Anweisungen im Schleifenkörper solange wiederholt ausgeführt bis die angegebene Bedingung wahr ist. Die tatsächliche Anzahl der Iterationen ist bei der bedingungskontrollierten Schleife erst zur Ausführungszeit bekannt und wird nicht beim Programmieren von vornherein definiert.

Endlos-Schleife

Eine Endlosschleife ist eine Schleife, die sich unendlich oft wiederholt. Eine Endlosschleife entsteht, wenn eine Abbruchbedingung bei einer bedingungskontrollierten Schleife fehlerhaft ist und immer als wahr ausgewertet wird. Normalerweise führt eine Endlosschleife zu einem Fehler im Programm und sollte vermieden werden.

3.3.6 Funktionen

Wird ein Stück Code oft wiederholt oder wird es an verschiedenen Stellen eines Programms eingesetzt, kann es in eine Funktion ausgelagert werden. Dann wird jeweils die Funktion aufgerufen wenn der Code der Funktion ausgeführt werden soll. Um die Funktion aufrufen zu können, wird die Funktion mit einem Namen definiert. Der Funktion können gewisse Werte übergeben werden, die man Parameter nennt, beispielsweise wenn die Funktion mehrmals aufgerufen wird, aber unterschiedliche Werte verwenden soll.

Funktionen sind ein wichtiges Konstrukt um «modular» zu programmieren und Probleme in kleinere Teilprobleme zu zerlegen. Für jedes Teilproblem kann eine eigene Funktion programmiert werden und dass gesamte Problem kann dann gelöst werden in dem die Funktionen für die Teilprobleme ausgeführt werden.

3.4 Python

Häufig verwendete höhere Programmiersprachen für Computerprogramme oder Applikationen sind Sprachen wie Python, Java oder JavaScript. Python wird zum Beispiel für grosse Teile von Dropbox, Spotify, Netflix und Youtube eingesetzt.

Die bereits vorgestellten Programmierkonstrukte—Anweisungen, Sequenzen, Fallunterscheidungen, Schleifen, und Funktionen—sind grundlegende Konstrukte, die bei fast jeder höheren Programmiersprache anzutreffen sind. Im Folgenden erläutern wir wie diese Konstrukte in Python programmiert werden.

Um direkt in Python zu programmieren und die folgenden Code Snippets auch gleich auszuprobieren, können Sie zum Beispiel den online Python Editor von repl.it direkt im Browser benutzen (<https://repl.it/languages/python3>).

3.4.1 Anweisungen und Sequenzen

Eine Zeile Code in Python entspricht einer Anweisung. Ein Python-Programm setzt sich aus mehreren Anweisungen zusammen, die dann von oben nach unten abgearbeitet werden und eine Sequenz darstellen.

Beispiel für Sequenzen Das folgende Programm besteht aus drei Anweisungen. Es fragt den Benutzer nach seinem Namen, speichert den Wert in der Variable *name* und gibt dann “Hallo <name>” aus.

1. `print("Wie heisst du?")`
2. `name = input()`
3. `print("Hallo" + name)`

3.4.2 Fallunterscheidung

Die ein- und zweiseitige Fallunterscheidungen lassen sich durch das If-Else-Konstrukt in Python umsetzen. Bei der einseitigen Fallunterscheidung wird einfach der Else Zweig weggelassen.

Die einseitige Fallunterscheidung:

```
if (Bedingung):  
... Anweisungsblock falls Bedingung wahr ...
```

Die zweiseitige Fallunterscheidung:

```
if (Bedingung):  
... Anweisungsblock falls Bedingung wahr ...  
else:  
... Anweisungsblock falls Bedingung falsch ...
```

Beispiel für eine einseitige Fallunterscheidung:

1. `a = 10`
2. `if (a<12):`
3. `print("Zahl ist zu klein")`

Wichtig hier ist der Einzug / die Einrückung bei der dritten Zeile, in diesem Fall vor dem print-Statement. Dieser Einzug definiert, dass der Anweisungsblock (oder in diesem Fall die print Anweisung) zum if gehört und nur ausgeführt wird, falls die Bedingung ($a < 12$) erfüllt ist.

Aufgabe für eine zweiseitige Fallunterscheidung Erstelle eine Variable und initialisiere diese mit 10. Programmiere eine Fallunterscheidung die abfragt ob der Wert der Variable kleiner als 12 ist, und falls ja gib aus, dass die Zahl zu klein ist. Ansonsten gib aus, dass die Zahl gross genug ist.

Lösung zur zweiseitigen Fallunterscheidung

1. $a = 10$
2. `if (a < 12):`
3. `print("Zahl ist zu klein.")`
4. `else:`
5. `print("Zahl ist gross genug")`

3.4.3 Schleifen

In Python gibt es unterschiedliche Schleifenkonstrukte. Im Folgenden betrachten wir wieder zählerkontrollierte und bedingungskontrollierte Schleifen.

Zählerkontrollierte Schleifen Die zählerkontrollierte Schleife wird in Python mit einer sogenannten For-Loop umgesetzt. Dabei kann mit `range(start, stop)` eine Zahlensequenz/-liste von start nach stop erstellt werden (wobei stop nicht eingeschlossen ist in der Zahlensequenz) über die dann in der For-Loop iteriert wird.

Ein Beispiel für eine zählerkontrollierte Schleife, die die Zahlen von 1 bis 10 in die Konsole schreibt kann dann wie folgt programmiert werden:

`for i in range(1, 11):`

`print(i)`

`range(1,11)` erstellt dabei eine Zahlenliste von 1 bis 10.

3.4.4 Bedingungskontrollierte Schleifen

Die bedingungskontrollierte Schleife kann durch einen While-Loop in Python programmiert werden.

`while (Bedingung):`

... Anweisungsblock falls Bedingung wahr ...

3.4.5 Funktionen

Werden Anweisungsblöcke mehrmals wiederholt oder an unterschiedlichen Stellen im Programm verwendet, lohnt es sich diese in Funktionen auszulagern. Dies erleichtert die Wartbarkeit des Programms und erhöht die Effizienz beim Programmieren. Die Funktion kann Parameter, d.h. Werte von ausserhalb der Funktion entgegennehmen. Es können eine beliebige Anzahl Parameter definiert werden. Doch beim Aufruf sollte darauf geachtet werden, die Parameter korrekt zu übergeben, d. h. die Anzahl und Reihenfolge beachten.

In Python wird eine Funktion mit dem Schlüsselwort “def” erstellt:

```
def funktion1OhneParameter():
... Anweisungsblock der Funktion ...
def funktion2MitParameter (parameter1, parameter2):
... Anweisungsblock der Funktion ...
```

Im Programm kann die Funktion über ihren Namen und mit den Parametern, falls definiert, aufgerufen werden:

```
funktion1OhneParameter ()
oder
funktion2MitParameter (parameter1, parameter2)
```

Referenzen

- Balzert, H. (1999). Lehrbuch Grundlagen der Informatik: Konzepte und Notationen in UML, Java und C++; Algorithmik und Software-Technik; Anwendungen. Spektrum Akad. Verlag.
- Boles, D. (1999). Programmieren spielend gelernt mit dem Java-Hamster-Modell (Vol. 2). Teubner.
- Harper, Robert (1 May 2017). “What, if anything, is a programming-paradigm?”. FifteenEightyFour. Cambridge University Press
- Wagenknecht, C. (2004). Programmierparadigmen: eine Einführung auf der Grundlage von Scheme. Teubner Verlag.

3.5 Folien



**Universität
Zürich**
UZH

Digital Society Initiative



Studium Digitale

Kursbaustein 9: Programmieren.

Lektion 1: Einführung in die Programmierung

Prof. Thomas Fritz, Institut für Informatik

Lernziele

Am Ende dieser Lektion wissen Sie ...

- was Programmieren ist,
- wieso Programmieren wichtig ist.



*«Software is eating
the world»*

- Marc Andreessen



Beispiel aus dem Alltag – Zugverbindung finden

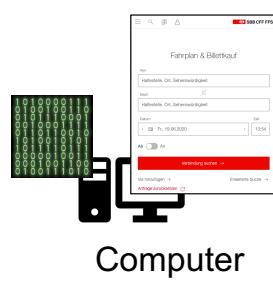


Beispiel aus dem Alltag – Zugverbindung finden

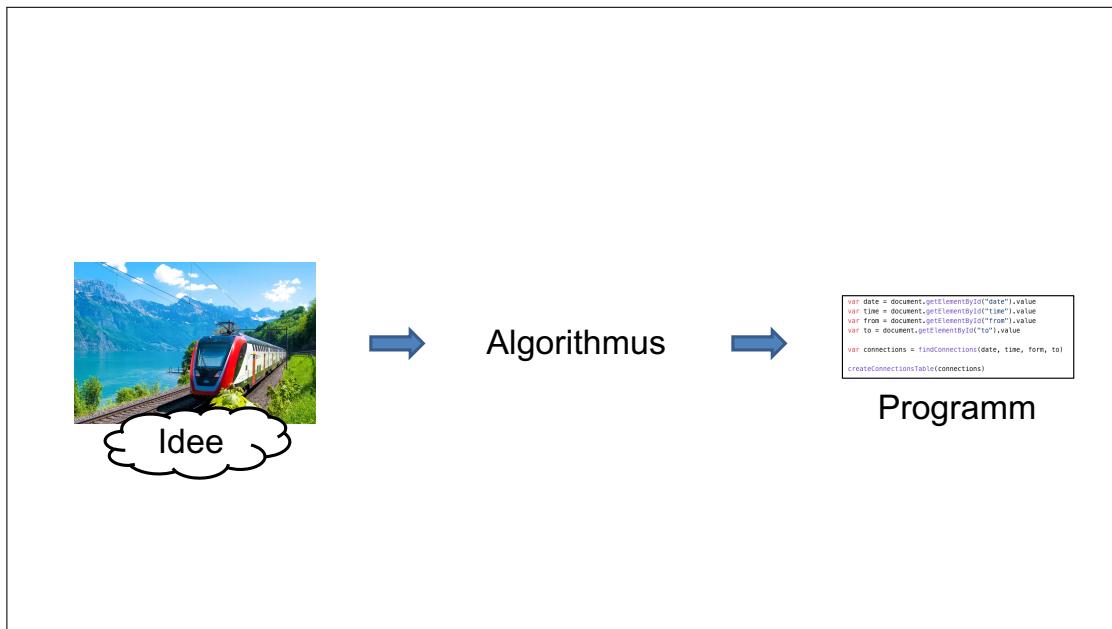


```
var date = document.getElementById("date").value  
var time = document.getElementById("time").value  
var from = document.getElementById("from").value  
var to = document.getElementById("to").value  
  
var connections = findConnections(date, time, from, to)  
createConnectionsTable(connections)
```

Programm



Programmieren



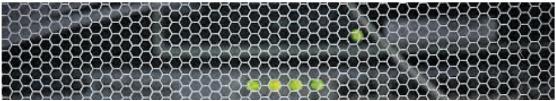
BILDUNG UND GESELLSCHAFT

Montag, 3. Juni 2013 · Nr. 125
Neue Zürcher Zeitung

Programmieren oder programmiert werden

Die Grundlagen der Informatik sollten an den Schulen in einem eigenständigen, obligatorischen Fach unterrichtet werden

tionsgesellschaft leute mit Informatik- „Menschen, die en und Grenzen der m Informationsverar- eurteilen wissen. mer Stundenlang am



komplexität, die Grenzen der Automatisierbarkeit, Sicherheit, Kommunikation, Modellbildung und Problemlösungstechnik. Es geht nicht darum, aus Gymnasiasten Informatikexperten zu machen, sondern ihnen Einblicke zu vermitteln, die zur verantwortungsvollen Nutzung und zur Vermeidung grösseren

Software / Programm



Lime-Tretroller werfen Fahrer ab

Lime hat seine elektrischen Tretroller in der Schweiz stillgelegt, nachdem es möglicherweise durch einen Softwarefehler dazu kam, dass die kleinen Fahrzeuge während der Fahrt unvermittelt stoppten.

Verunglücktes Tesla-Auto fuhr per Autopilot

Vor etwa einer Woche kam der Fahrer eines Tesla-Fahrzeugs bei einem Unfall in Kalifornien ums Leben. Ermittler konzentrieren sich nun auf den Autopiloten.

Grundlegender Softwarefehler in der Boeing 737 Max gefunden

Tests der US-Luftfahrtbehörde decken weitere Fehlfunktionen auf, die erneute Verzögerungen für die Rückkehr des Flugzeugtyps bedeuten.

Software / Programm

Lime-Tretroller werfen Fahrer ab

Lime hat seine elektrischen Tretroller in der Schweiz stillgelegt, nachdem es möglicherweise durch einen Softwarefehler dazu kam, dass die kleinen Fahrzeuge während der Fahrt unvermittelt stoppten.

Verunglücktes Tesla-Auto fuhr per Autopilot

Vor etwa einer Woche kam der Fahrer eines Tesla-Fahrzeugs bei einem Unfall in Kalifornien ums Leben. Ermittler konzentrieren sich nun auf den Autopiloten.

Übersicht Programmieren

- Einführung
- Programmiersprachen
- grundlegende Programmierkonstrukte
- Scratch-Einführung
- Übungen

Referenzen

Stefan Betschon (2013). Porgrammieren oder porgrammiert werden. NZZ. Abgerufen von http://fit-in-it.ch/sites/default/files/small_box/nzz_zehnder_2013-06-03.pdf (letzter Zugriff am 17. Juni 2020)

Andreas Donath (2019). Lime-Tretroller werfen Fahrer ab. golem.de. Abgerufen von <https://www.golem.de/news/softwarefehler-lime-tretroller-werfen-fahrer-ab-1901-138694.html> (letzter Zugriff am 17. Juni 2020)

Reuters (2018). Verunglücktes Tesla-Auto fuhr per Autopilot. NZZ. Abgerufen von <https://www.nzz.ch/mobilitaet/verungluecktes-tesla-auto-fuhr-per-autopilot-id.1371004> (letzter Zugriff am 17. Juni 2020).

awp/dpa (2018). Autopilot beschleunigte Tesla vor tödlichem Crash. NZZ. Abgerufen von <https://www.nzz.ch/panorama/toedlicher-tesla-unfall-autopilot-beschleunigte-vor-crash-id.1393013> (letzter Zugriff am 17. Juni 2020).

Andreas Spaeth (2019). Grundlegender Softwarefehler in der Boeing 737 Max gefunden. NZZ. Abgerufen von <https://www.nzz.ch/mobilitaet/luftfahrt/boeing-737-max-grundlegender-softwarefehler-gefunden-id.1499955?reduced=true> (letzter Zugriff am 17. Juni 2020)

Referenten

Prof. Thomas Fritz & Claudia Vogel
Institut für Informatik

© Universität Zürich
Digital Society Initiative



Universität
Zürich UZH

Digital Society Initiative



Studium Digitale

Kursbaustein 9: Einführung in Programmierung

Lektion 2: Programmiersprachen

Prof. Thomas Fritz, Institut für Informatik

Lernziele

Am Ende dieser Lektion ...

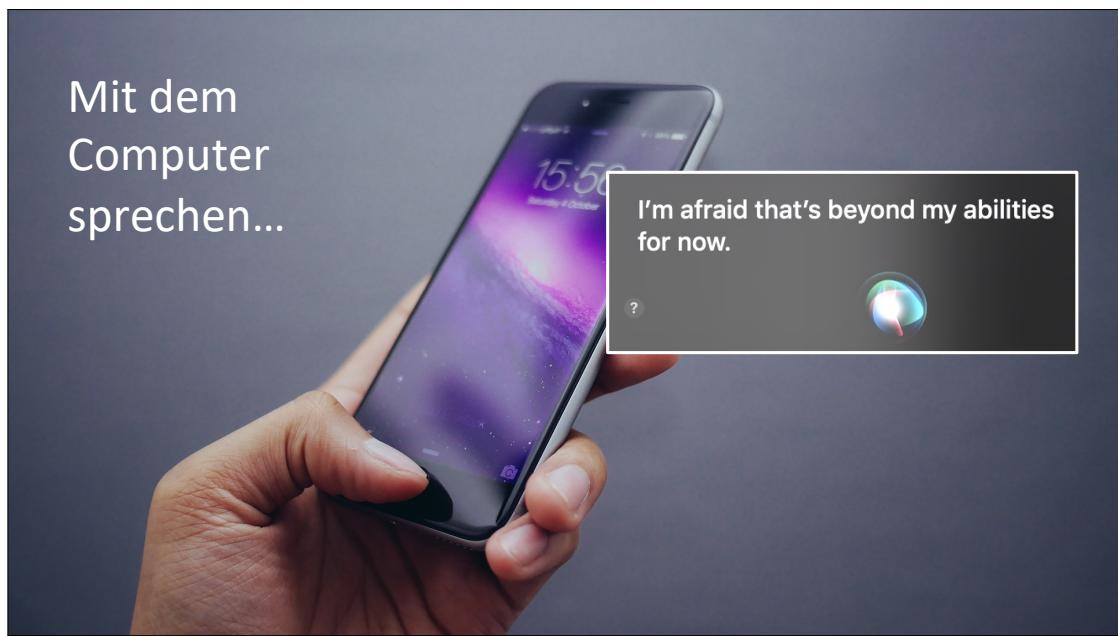
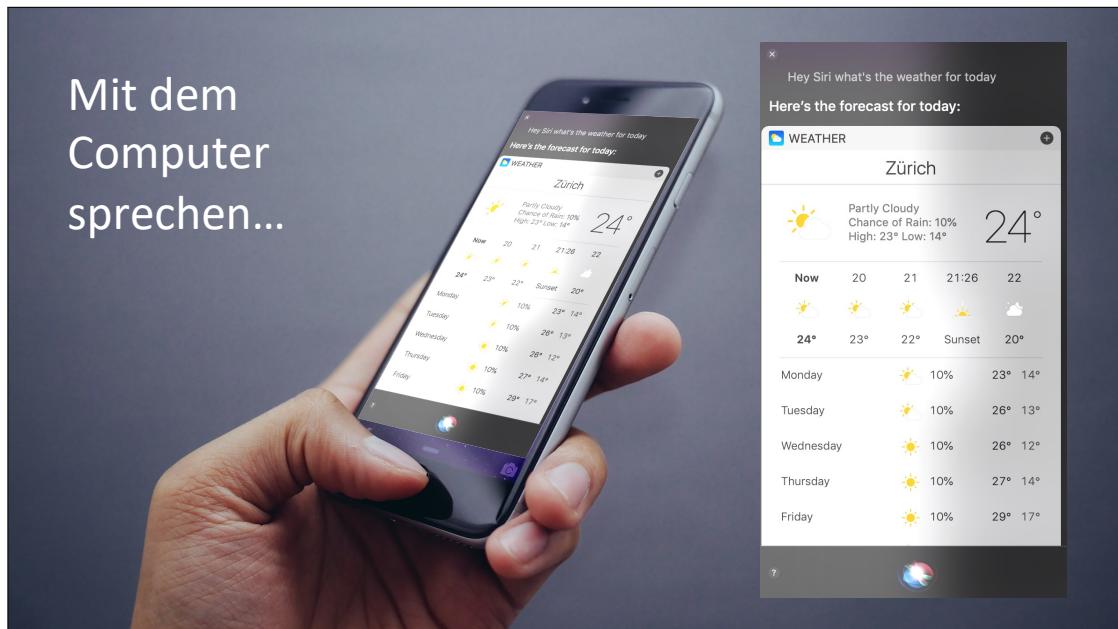
- wissen Sie was eine Programmiersprache ist und wie sie definiert ist,
- kennen Sie die drei Generationen der Programmiersprachen,
- verstehen Sie was Programmierparadigmen sind und können die vier vorgestellten Paradigmen an Hand eines Beispiels erklären und erkennen.

Wie sprechen Sie mit einem Computer?



Mit dem
Computer
sprechen...





3 Generationen der Programmiersprachen

Höhere Programmiersprache

Assemblersprache

Maschinensprache

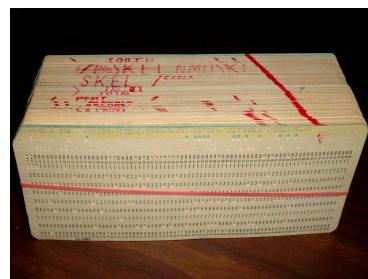


Computer

Generation Maschinensprache



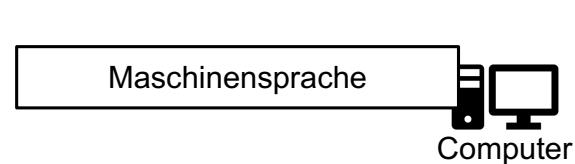
“Programmieren”



Lochkarten Programm

Bild von Arnold Reinhold, CC BY-SA 3.0
<https://creativecommons.org/licenses/by-sa/3.0/>

Generationen der Programmiersprachen



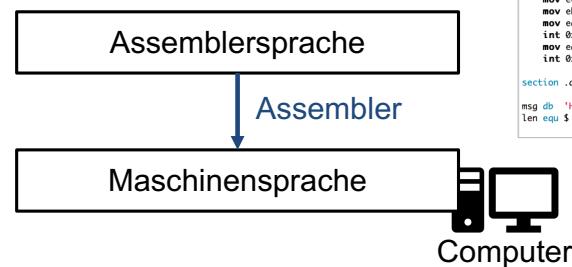
Maschinencode

```

01011111110010001111111101000...
00011011001010010111111001000110
111010000001101100101011111110
010100010101110010010111111111
11001000110000000000000000000000100
00001101100000101000000000000000
000000000000000000000000000000000000
000000000000000000000000000000000000
11111111110010001111111010000001
101100101001011111100100011111110
1000000110110010101100101000000000000
000000000000000000000000000000000000
000000000000000000000000000000000000
010001100000000000000000000000000000
010011000000000000000000000000000000
010101100000000000000000000000000000

```

Generationen der Programmiersprachen



Assemblercode

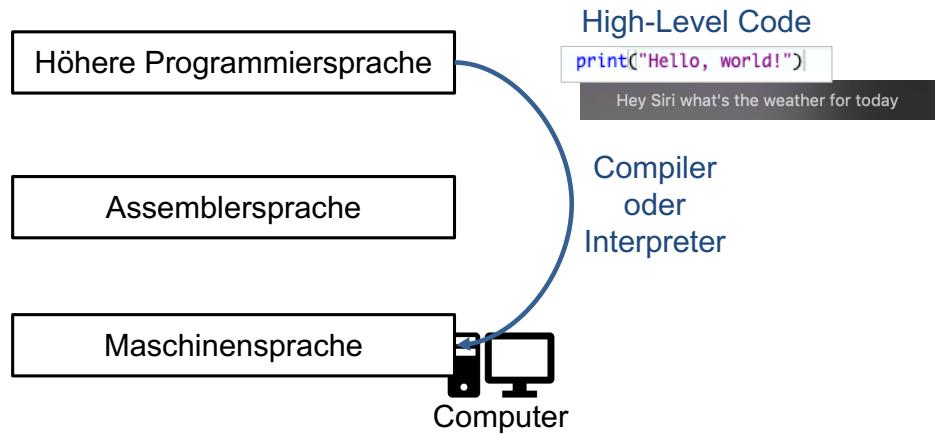
```

section .text
global _start           ;must be declared for using gcc
_start:
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int $0x80
    mov eax, 1
    int $0x80

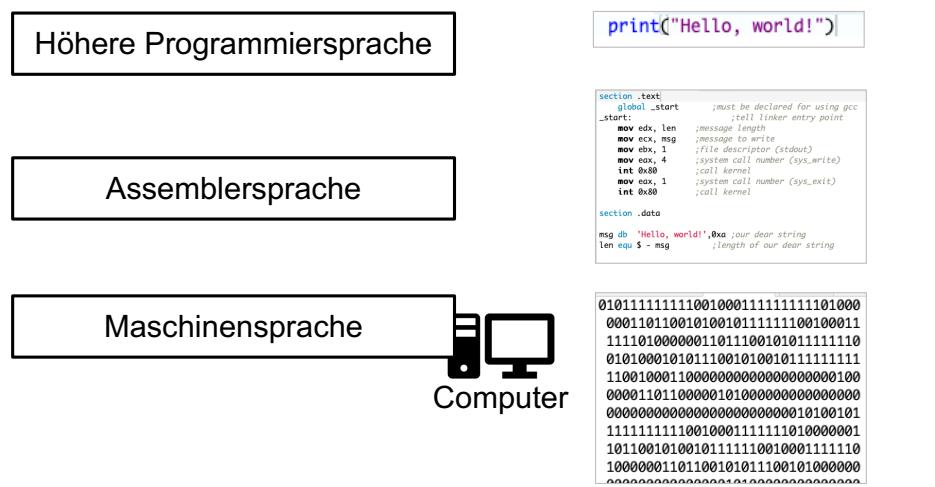
section .data
msg db 'Hello, world!',0xa ;our dear string
len equ $ - msg           ;length of our dear string

```

Generationen der Programmiersprachen



Generationen der Programmiersprachen



Beispiel: SBB App für Zugverbindungen

☰ 🔎 🚅 🚶

Fahrplan & Billettkauf

Von

Haltestelle, Ort, Sehenswürdigkeit

Nach

Haltestelle, Ort, Sehenswürdigkeit

Datum	Zeit
< ⏪ Fr., 19.06.2020 ⏩ >	13:54

Ab An

Verbindung suchen →

Via hinzufügen + Erweiterte Suche →

Anfrage zurücksetzen ⏪

HTML

```
var date = document.getElementById("date").nodeValue  
var time = document.getElementById("time").nodeValue  
var from = document.getElementById("from").nodeValue  
var to = document.getElementById("to").nodeValue  
  
var connections = findConnections(date, time, from, to)  
  
createConnectionsTable(connections)
```

Javascript

Definition Programmiersprache

- Lexikalik
 - Syntax
 - Semantik

Definition Programmiersprache

Lexikalik – definiert gültigen Zeichen

A-Z, a-z, 0-9, =, <, >, ...

```
var date* = document.getElementById("date").value
var time0 = document.getElementById("time").value
var from_ = document.getElementById("from").value
var to$ = document.getElementById("to").value

var connections = findConnections(date, time, form, to)
```

Definition Programmiersprache

Syntax – definiert korrekten Aufbau der Sätze

```
var date + time = document.getElementById("date").value
var time = document.getElementById("time").value
var from = document.getElementById("from").value
var to = document.getElementById("to").value

var connections = findConnections(date, time, form, to)

createConnectionsTable(connections)
```

Definition Programmiersprache

Semantik – definiert Bedeutung semantisch korrekter Sätze

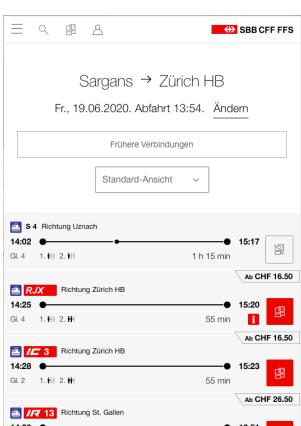
```
var date = document.getElementById("date").nodeValue  
var time = document.getElementById("time").nodeValue  
var from = document.getElementById("from").nodeValue  
var to = document.getElementById("to").nodeValue  
  
var connections = findConnections(date, time, from, to)  
  
createConnectionsTable(connections)
```

Definition Programmiersprache

Semantik – definiert Bedeutung semantisch korrekter Sätze

```
var number1 = 2  
var number2 = 4  
var sum = number1 + number2  
  
var divBy0 = sum / 0
```

... wenn alles korrekt



... wenn nicht

```
error: unknown: Unexpected token, expected ";" (1:9)
> 1 | var date + time = document.getElementById("date")
|           ^
2 | var time = document.getElementById("time").value
3 | var from = document.getElementById("from").value
4 | var to = document.getElementById("to").value
```



Programmiersprache Paradigmen

«Paradigma bezeichnet eine fundamentale Denkungsart, einen Denkstil oder ein Denkmuster»

-- Wagenknecht 2004

Programmierparadigmen

- Deklarative Programmierung
- Funktionale Programmierung
- Imperative Programmierung
- Objekt-orientierte Programmierung
- ...

Programmierparadigmen

Deklarativ

- «Was?»
- Beschreibung des gewünschten Endergebnisses
- z.B. SQL, Oz, ...

Programmierparadigmen

Deklarativ

"Gib mir alle
Zugverbindungen von
Sargans nach Zürich am
19.06.2020 zwischen
13:00 und 14:00 Uhr"

WAS?



Datenbank: Speichert
alle Zugverbindungen

Id	Time	From	To
1	13:28	Sargans	Zürich
2	13:31	Sargans	Chur
...			

Programmierparadigmen

Funktional

- Stützt auf mathematischen Grundlagen
- Fokus auf Funktionen
 - $y = f(x)$
- z.B. Scheme, SML, Lisp, Haskell, ...

Programmierparadigmen

Funktional

Hinfahrt	
Sargans → Zürich HB	
2. Klasse	
1x Streckenbillett, Halbtax-Abo	①
Gültig: Fr., 19.06.2020	
bis Sa., 20.06.2020 05:00	Ab CHF 16.50
Rückfahrt	
Zürich HB → Sargans	
2. Klasse	
1x Streckenbillett, Halbtax-Abo	①
Gültig: So., 21.06.2020	
bis Mo., 22.06.2020 05:00	Ab CHF 16.50
Gesamtpreis	CHF 33.00

Berechne
Gesamtpreis

```
fun sum [] = 0
| sum (x::xs) = x + sum xs
```

Programmierparadigmen

Imperativ

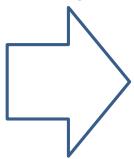
- «Wie?»
- Genauer Ablauf: «Zuerst mach das, dann das...»
- z.B. C, PHP, Python, ...

Programmierparadigmen

Imperativ

Hinfahrt	
Sargans	→ Zürich HB
2. Klasse	
1x Streckenbillett, Halbtax-Abo	
Gültig: Fr., 19.06.2020	Ab CHF 16.50
bis Sa., 20.06.2020 05:00	
Rückfahrt	
Zürich HB	→ Sargans
2. Klasse	
1x Streckenbillett, Halbtax-Abo	
Gültig: So., 21.06.2020	Ab CHF 16.50
bis Mo., 22.06.2020 05:00	
Gesamtpreis	CHF 33.00

Berechne
Gesamtpreis



```
var total = 0;  
for (connection in bookedConnections) {  
    total = total + connection.price  
}
```

Programmierparadigmen

Objektorientiert

- Fokus auf Objekte: Eigenschaften, Manipulation, und Kommunikation von und zwischen Objekten
- z.B. Java, Smalltalk

Programmierparadigmen

Objektorientiert

Hinfahrt	
Sargans	→ Zürich HB
2. Klasse	
1x Streckenbillett, Halbtax-Abo	
Gültig: Fr., 19.06.2020	①
bis Sa., 20.06.2020 05:00	
	Ab CHF 16.50
Rückfahrt	
Zürich HB	→ Sargans
2. Klasse	
1x Streckenbillett, Halbtax-Abo	
Gültig: So., 21.06.2020	①
bis Mo., 22.06.2020 05:00	
	Ab CHF 16.50
Gesamtpreis	CHF 33.00

Berechne
Gesamtpreis



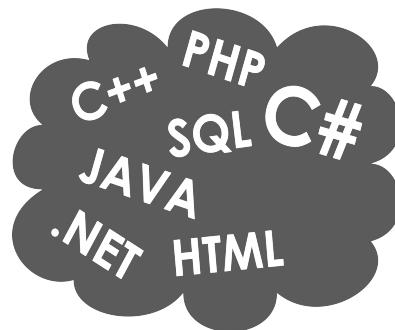
```
class Connection {
    private Location from, to;
    private float cost;

    public float getCost() {
        return cost;
    }
    ...
}
```

```
class ShoppingCart {
    private List<Connection> allConns;

    public float getTotalCost() {
        float totalCost;
        for (Connection con : allConns) {
            totalCost = con.getCost() + totalCost;
        }
        return totalCost;
    }
    ...
}
```

Programmiersprachen



Referenzen

- Balzert, H. (1999). *Lehrbuch Grundlagen der Informatik: Konzepte und Notationen in UML, Java und C++; Algorithmik und Software-Technik; Anwendungen*. Spektrum Akad. Verlag.
- Wagenknecht, C. (2004). *Programmierparadigmen: eine Einführung auf der Grundlage von Scheme*. Teubner Verlag.

Referentin

Prof. Thomas Fritz
Institut für Informatik

© Universität Zürich
Digital Society Initiative



Universität
Zürich UZH

Digital Society Initiative



Studium Digitale

Kursbaustein 9: Einführung in Programmierung

Lektion 3: Sequenzen und Variablen

Prof. Thomas Fritz, Institut für Informatik

Lernziele

Am Ende dieser Lektion ...

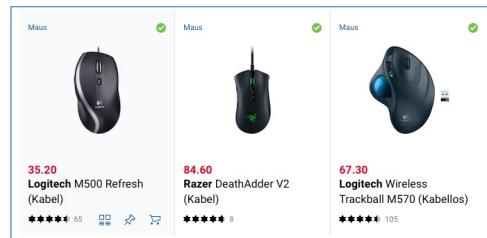
- wissen Sie was ein Datentyp ist,
- verstehen Sie was eine Variable ist und wie man sie nutzt,
- wissen Sie was eine Sequenz ist und wie man sie anwendet.

Online Bestellung der Home-Office Einrichtung: Tastatur, Maus, Monitor

1. Produkt suchen



2. Produkt auswählen



Online Bestellung der Home-Office Einrichtung: Tastatur, Maus, Monitor

3. Produkt in den Warenkorb legen

In den Warenkorb

4. Zur Kasse & Bestellung absenden

Online Bestellung der Home-Office Einrichtung: Tastatur, Maus, Monitor

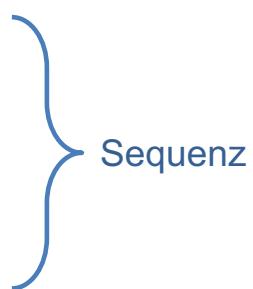
1. Produkt suchen
 2. Produkt auswählen
 3. Produkt in Warenkorb legen
 4. Zur Kasse & Bestellung absenden
- ← Anweisung

Anweisung

- elementare Einheit im Programm
- steht für einen einzelnen Abarbeitungsschritt im Algorithmus
- auch Statement, Kommando, oder Befehl genannt

Online Bestellung der Home-Office Einrichtung: Tastatur, Maus, Monitor

1. Produkt suchen
2. Produkt auswählen
3. Produkt in Warenkorb legen
4. Zur Kasse & Bestellung absenden

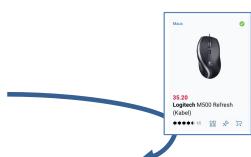


Sequenz

- Abfolge von Anweisungen
- sequenzielle Ausführung in der gegebenen Reihenfolge von oben nach unten
- kein Überspringen von Anweisungen

Online Bestellung der Home-Office Einrichtung: Tastatur, Maus, Monitor

1. Produkt suchen
2. Produkt auswählen
3. Produkt in Warenkorb legen
4. Zur Kasse & Bestellung absenden



Variablen: Leerer Warenkorb

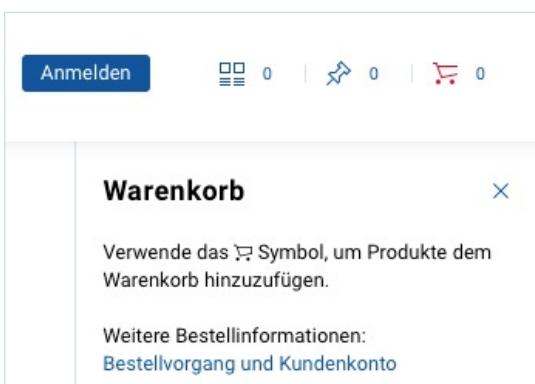
A screenshot of a web-based shopping cart interface. At the top, there is a navigation bar with a 'Anmelden' button, a user icon (0), a search icon (0), and a shopping cart icon (0). Below this is a large button labeled 'Warenkorb' with an 'X' icon. Inside the cart area, there is a message: 'Verwende das ✦ Symbol, um Produkte dem Warenkorb hinzuzufügen.' Below this message, there is a link: 'Weitere Bestellinformationen: Bestellvorgang und Kundenkonto'.

```
var shoppingCartSize;
```

Variable

- speichert einfache Werte in Programmen
- erlaubt auf den gespeicherten Wert zuzugreifen

Variablen: Leerer Warenkorb



```
var shoppingCartSize;
```

Deklaration

'var' erstellt neue Variable

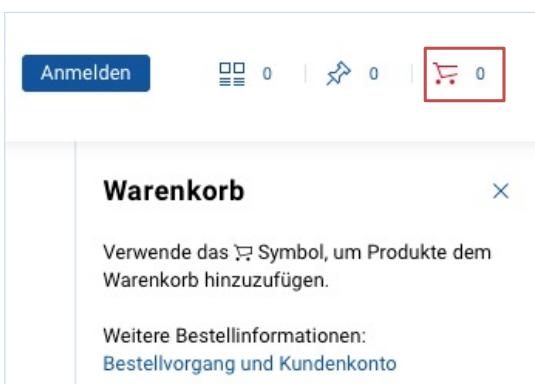
Verwendung von Variablen

Deklaration

- erstellt Variable
- definiert Typ und Namen der Variable

```
var shoppingCartSize;
```

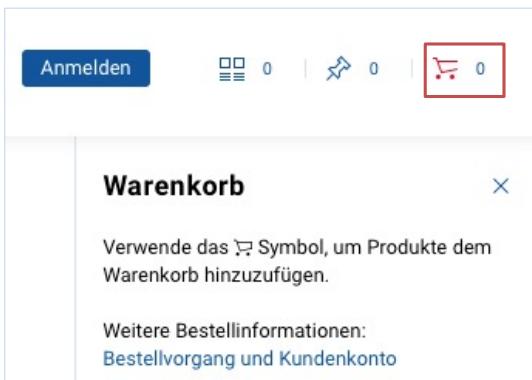
Variablen: Leerer Warenkorb



```
var shoppingCartSize = 0;
```

Deklaration & Initialisierung

Variablen: Leerer Warenkorb



```
var shoppingCartSize;
```

Deklaration

```
shoppingCartSize = 0
```

Initialisierung

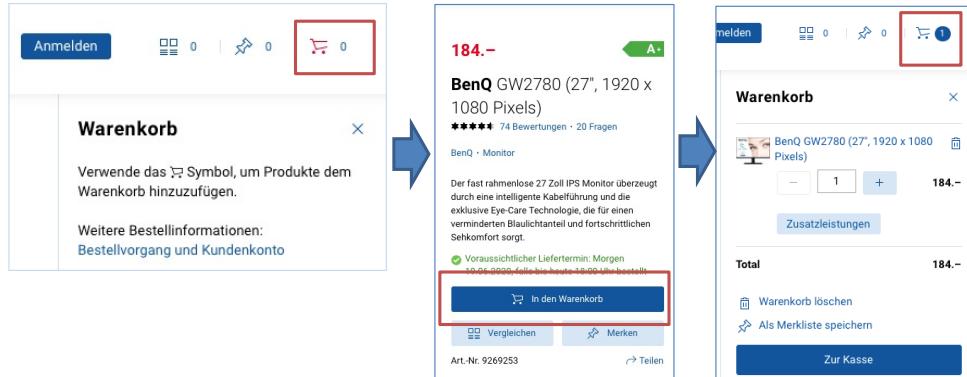
Verwendung von Variable

Initialisierung

- Erstmaliges Speichern von einem Wert in der Variable

Name	Value
shoppingCartSize	0
...	...

Variablen: Produkt im Warenkorb hinzufügen



Variablen: Aktualisierter Warenkorb



```
var shoppingCartSize = 0;
```

```
shoppingCartSize = 1;
```

Zuweisung

Verwendung von Variablen

Zuweisung

- Veränderung des Wertes

Variablen: Bestellübersicht



```
var shoppingCartSize = 1
```

```
var orderOverviewTitle = "Bestellung " +  
    shoppingCartSize + " Artikel"
```

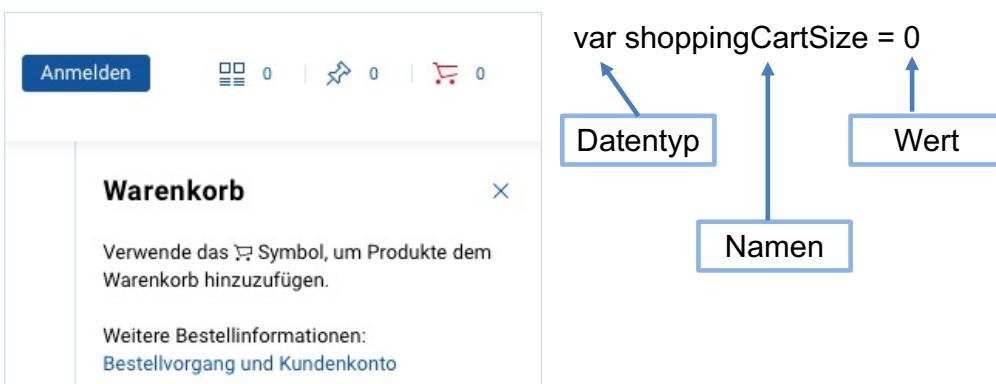
Gebrauch

Verwendung von Variablen

Gebrauch

- Zugriff auf Variable, resp. auslesen des gespeicherten Wertes

Variablen: Eigenschaften



Datentypen: Detailansicht eines Produkts



Datentyp

- beschreibt Menge von Datenobjekten,
- Datenobjekte eines Typs haben alle die gleiche Struktur, und
- auf Datenobjekten des gleichen Typs können die gleichen Operationen durchgeführt werden

Datentypen: Detailansicht eines Produkts



```
var price = 184.00
```

Arten von Datentypen

Numerische Datentypen

- Ganze Zahlen (1, 49)
- Gleitkommazahlen (5.90, 1.999)

Datentypen: Detailansicht eines Produkts



```
var name = "BenQ .."  
var description = "Der fast  
rahmlose ..."  
var image = "../img/2784.jpg"
```

Arten von Datentypen

Zeichen

- Einzelne Ziffern
`var currency = '€'`

Zeichenkette

- Wörter, Sätze
`var productName = "BenQ"`

Datentypen: Detailansicht eines Produkts



```
var ratings = List<Ratings>(74);
```

```
var ratings = ['gut', 'sehr gut', ...]
```

```
var ratings = [rating1, rating2, ...]
```

Arten von Datentypen

Listen

- Kollektion von Datenobjekten

Datentypen: Detailansicht eines Produkts



```
var productInStock = true;
```

Arten von Datentypen

Wahrheitswerte

- Boolean
- Entweder wahr oder falsch

Weiteres Beispiel Datentypen:

The left screenshot shows a product page for a BenQ GW2780 monitor. The price is listed as 184.-. The right screenshot shows a shopping cart with one item: the BenQ GW2780 monitor at 184.-.

```
var product1 = {
  name: "BenQ",
  price: 184.00,
  amount: 1
}
```

```
var products = [product1]
```

Variablen speichern Werte

Datentyp definiert was gespeichert wird, z.B.:

$\downarrow \frac{1}{9}$ Zahl

$\downarrow \frac{n}{z}$ Zeichenketten

$\times \checkmark$ Wahrheitswert

$\frac{1}{2} \equiv \frac{3}{3}$ Liste

...

Referenzen

Boles, D. (1999). Programmieren spielend gelernt mit dem Java-Hamster-Modell (Vol. 2). Teubner.

Referentin

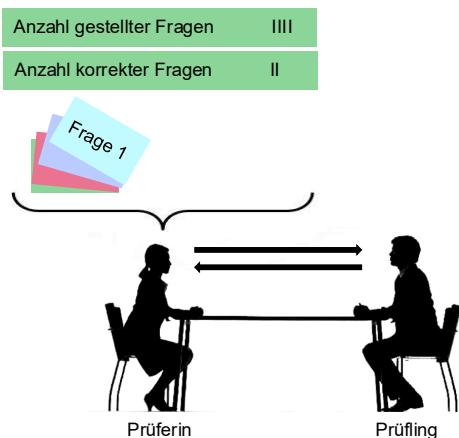
Prof. Thomas Fritz
Institut für Informatik

© Universität Zürich
Digital Society Initiative

Einführung in die Programmierung

- **Szenario: Quiz**
- Code
- Programmierkonstrukte im Detail

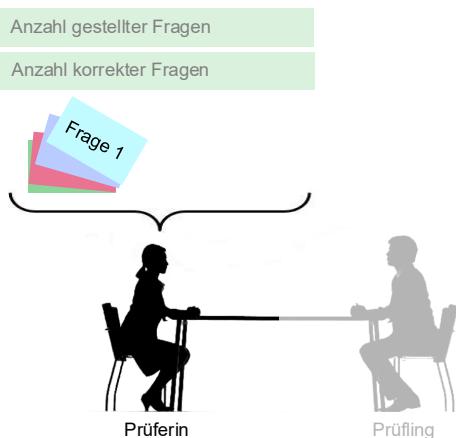
Quiz Beispielprogramm – Szenario



Eine Prüferin fragt einen Prüfling ab.

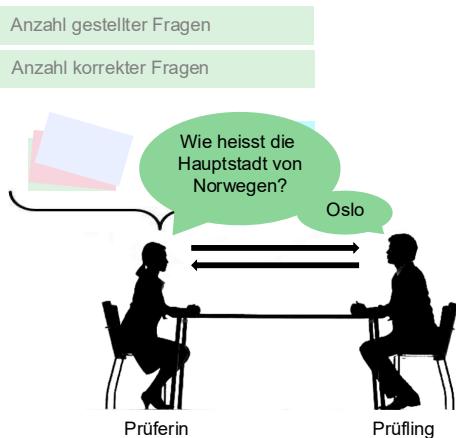
- Gegeben sei ein Stapel mit Lernkarten. Auf der Vorderseite ist jeweils die Frage notiert, auf der Rückseite die Antwort.
- Die Lernkarten werden nun der Reihe nach abgefragt. Der Prüfling antwortet und die Prüferin entscheidet, ob die Antwort korrekt war.
- Wenn alle Lernkarteien abgearbeitet sind, wird der Prüfling über die Anzahl korrekter Antworten informiert.

Quiz Beispielprogramm – Formalisierung der Anforderungen



- ① Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **gestellten Fragen** notiert wird
- ② Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **korrekten Antworten** notiert wird
- ③ Prüferin nimmt den **Stapel** in die Hand

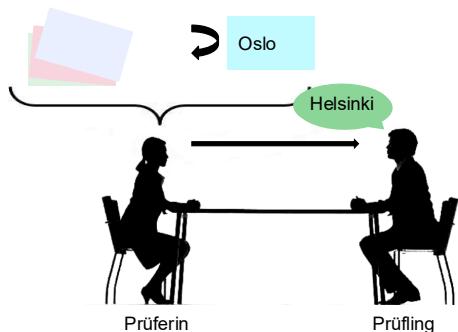
Quiz Beispielprogramm – Formalisierung der Anforderungen



- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**
 - Prüferin nimmt oberste Karte vom Stapel
 - Prüferin stellt dem Prüfling die **Frage**, dieser gibt eine **Antwort**

Quiz Beispielprogramm – Formalisierung der Anforderungen

Anzahl gestellter Fragen	1
Anzahl korrekter Fragen	1

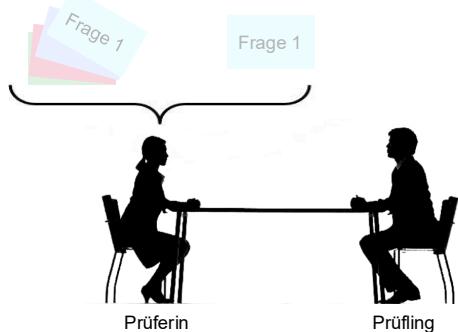


- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- ...
 - Prüferin kontrolliert, ob die **Antwort** korrekt ist
- Wenn korrekt:
- Prüferin gibt dem Prüfling positives Feedback
 - Prüferin erhöht die Anzahl der **korrekten Antworten** um 1
- Wenn falsch:
- Prüferin gibt dem Prüfling negatives Feedback

Quiz Beispielprogramm – Formalisierung der Anforderungen

Anzahl gestellter Fragen	1
Anzahl korrekter Fragen	1



- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- Prüferin nimmt oberste Karte vom Stapel
- Prüferin stellt dem Prüfling die **Frage**, dieser gibt eine **Antwort**
- Prüferin kontrolliert, ob die **Antwort** korrekt ist
 - ...
 - ...
- Prüferin erhöht Anzahl der gestellten Fragen um 1

Quiz Beispielprogramm – Formalisierung der Anforderungen

- ① Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **gestellten Fragen** notiert wird
- ② Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **korrekten Antworten** notiert wird
- ③ Prüferin nimmt den **Stapel** in die Hand
- ④ Prüferin geht durch jede **Lernkarte im Stapel**
 - Prüferin nimmt oberste Karte vom Stapel
- Prüferin stellt dem Prüfling die **Frage**, dieser gibt eine **Antwort**
- Prüferin kontrolliert, ob die **Antwort** korrekt ist
 - ...
 - ...
- Prüferin erhöht Anzahl der gestellten Fragen um 1
- ⑤ Prüferin sagt, wie viele **korrekte Antworten** von allen **gestellten Fragen** der Prüfling beantwortet hat

Quiz Beispielprogramm – Programmierung

```

# Initialize the total number of questions
nr_total_questions_asked = 0

# Initialize the number of correct answers
nr_correct_answers = 0

# 2 dimensional list. Nested list contains a list with 2 entries.
# The first entry holds the question, the second the answer as a string.
questionnaire = [
    ["Hauptstadt von Norwegen?", "Oslo"],
    ["Hauptstadt von Finnland?", "Helsinki"]
]

# Go through every question answer pair
for question_answer_pair in questionnaire:
    # First entry is the question
    # Second entry is the question
    question = question_answer_pair[0]
    answer = question_answer_pair[1]
  
```

https://files_ifi_uzh_ch_ddis_web-python/

Einführung in die Programmierung

- Szenario: Quiz
- **Code**
- Programmierkonstrukte im Detail

Quiz Beispielprogramm – Formalisierung der Anforderungen

- ① Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **gestellten Fragen** notiert wird
 - ② Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **korrekten Antworten** notiert wird
 - ③ Prüferin nimmt den **Stapel** in die Hand
 - ④ Prüferin geht durch jede **Lernkarte** im **Stapel**
 - Prüferin nimmt oberste Karte vom Stapel
- Prüferin stellt dem Prüfling die **Frage**, dieser gibt eine **Antwort**
 - Prüferin kontrolliert, ob die **Antwort** korrekt ist
 - ...
 - ...
 - Prüferin erhöht Anzahl der gestellten Fragen um 1
- ⑤ Prüferin sagt, wie viele **korrekte Antworten** von allen **gestellten Fragen** der Prüfling beantwortet hat

Quiz Beispielprogramm – Von Anforderungen zu Code

Formalisierte Anforderungen

- ① Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **gestellten Fragen** notiert wird
- ② Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **korrekten Antworten** notiert wird
- ③ Prüferin nimmt den **Stapel** in die Hand
- ④ Prüferin geht durch jede **Lernkarte im Stapel**
 - Prüferin nimmt oberste Karte vom Stapel
 - Prüferin stellt dem Prüfling die **Frage**. Dieser gibt eine **Antwort**
 - Prüferin kontrolliert, ob die **Antwort** korrekt ist
 - ...
 - ...
 - Prüferin erhöht Anzahl der gestellten Fragen um 1
- ⑤ Prüferin sagt, wie viele **korrekte Antworten** von allen **gestellten Fragen** der Prüfling beantwortet hat

Code

```

nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
      str(nr_total_questions_asked))

```

Quiz Beispielprogramm – Code

- ① Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **gestellten Fragen** notiert wird
- ② Prüferin bereitet ein Papier vor, auf welchem die Anzahl der **korrekten Antworten** notiert wird

```

nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
      str(nr_total_questions_asked))

```

Quiz Beispielprogramm – Code

- ③ Prüferin nimmt den **Stapel** in die Hand

Stapel

Frage ↘ Antwort

[“Wie heisst die Hauptstadt von Norwegen”, “Oslo”]

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]
for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ③ Prüferin nimmt den **Stapel** in die Hand

Stapel



[
[“Wie heisst die Hauptstadt von Norwegen?”,
“Oslo”],
[“Wie heisst die Hauptstadt von Finnland?”,
“Helsinki”]
]

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]
for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- Prüferin nimmt oberste Karte vom Stapel
- Die **neue** Lernkarte aus questionnaire wird jeweils in der Variablen question_answer_pair zwischengespeichert
- Dieses Konstrukt nennt man eine Schleife
- Alles was zur Schleife gehört, wird eingerückt

```

nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
      str(nr_total_questions_asked))

```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- Prüferin nimmt oberste Karte vom Stapel
- Zuerst wollen wir die Frage in einer separaten Variablen speichern
- Auf die Frage innerhalb der question_answer_pair Variable können wir mit einem Index zugreifen
- Der Index gibt an, auf welchen Eintrag wir zugreifen möchten
- In der Informatik startet die Indexierung bei 0

```

nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
      str(nr_total_questions_asked))

```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- ...
- Prüferin stellt dem Prüfling die **Frage**, dieser gibt eine **Antwort**
 - Dieser Code öffnet ein Fenster mit dem Text der Variablen question und einem Textfeld, in welches der **Prüfling** die Antwort eingeben kann
 - Die Antwort wird in der Variable answer gespeichert

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]
for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- ...
- ...
- Prüferin kontrolliert, ob die **Antwort** korrekt ist

- Auf die Antwort innerhalb der question_answer_pair Variable können wir mit dem Index 1 zugreifen

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]
for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte im Stapel**

- ...
- ...
- ...

Wenn korrekt:

- Prüferin gibt dem Prüfling positives Feedback
- Prüferin erhöht die Anzahl der **korrekten Antworten** um 1

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]

    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1

print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte im Stapel**

- ...
- ...
- ...

Wenn falsch:

- Prüferin gibt dem Prüfling negatives Feedback

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]

    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1

print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ④ Prüferin geht durch jede **Lernkarte** im **Stapel**

- ...
- ...
- ...
- Prüferin erhöht Anzahl der gestellten Fragen um 1
- Danach nimmt die Prüferin die nächste Karte bzw.
question_answer_pair

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Code

- ⑤ Prüferin sagt, wie viele **korrekte Antworten** von allen **gestellten Fragen** der Prüfling beantwortet hat

```
nr_total_questions_asked = 0
nr_correct_answers = 0
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]

for question_answer_pair in questionnaire:
    question = question_answer_pair[0]
    answer = raw_input(question)
    correct_answer = question_answer_pair[1]
    if answer == correct_answer:
        print("Correct!")
        nr_correct_answers = nr_correct_answers + 1
    else:
        print("Wrong!")
    nr_total_questions_asked = nr_total_questions_asked
    + 1
print("Correctly " + str(nr_correct_answers) + " of total " +
str(nr_total_questions_asked))
```

Quiz Beispielprogramm – Demonstration Visualisierung

The screenshot shows the Online Python Tutor interface. On the left, the code for a quiz program is displayed:

```
1 # Initialize the total number of questions
2 nr_total_questions_asked = 0
3
4 # Initialize the number of correct answers
5 nr_correct_answers = 0
6
7 # 2 dimensional list. Nested list contains a list
8 # The first entry holds the question, the second
9 questionnaire = [
10     ["Hauptstadt von Norwegen?", "Oslo"],
11     ["Hauptstadt von Finnland?", "Helsinki"]
12 ]
13
14 # Go through every question answer pair
15 for question_answer_pair in questionnaire:
16     # First entry holds the question
17     question = question_answer_pair[0]
18     # Second entry holds the answer
```

On the right, the state of global variables and objects is visualized:

- Global variables:** nr_total_questions_asked = 0, nr_correct_answers = 0, questionnaire = [list, list]
- Objects:** A list object containing two entries: "Hauptstadt von Norwegen?" and "Oslo". Another list object contains "Hauptstadt von Finnland?" and "Helsinki".

Below the visualization, there are buttons for "Edit code", navigation ("First", "Back", "Step 5 of 6", "Forward", "Last"), and sharing ("Generate URL", "Generate embed code").

<https://files.ifi.uzh.ch/ddis/web-python/>

Einführung in die Programmierung

- Szenario: Quiz
- Code
- **Programmierkonstrukte im Detail**

Konstrukte im Detail – Übersicht

- Kommentare
- Variablen
- Werte
- Operatoren
- Funktionen
- Iteration
- Bedingungen

Konstrukte im Detail – Kommentare

- Kommentare haben keine Auswirkungen auf das Programm
- Sie dienen dazu, den Code zu erklären
- Ein Kommentar beginnt mit einem Hashtag #

```
# Initialize the total number of questions  
nr_total_questions_asked = 0
```

```
# Initialize the number of correct answers  
nr_correct_answers = 0
```

Konstrukte im Detail – Variablen

- Variablen speichern Werte
- Die Zuweisung eines Wertes erfolgt mit dem Gleichheitszeichen =
- Variablen können jederzeit überschrieben werden
- Leerzeichen und Sonderzeichen sind in Variablennamen nicht erlaubt
- Name der Variablen kann beliebig gewählt werden

```
# Initialize the total number of questions  
nr_total_questions_asked = 0
```

```
# Initialize the number of correct answers  
nr_correct_answers = 0
```

...

```
# Increment the question counter  
nr_total_questions_asked =  
    nr_total_questions_asked + 1
```

Konstrukte im Detail – Werte

- Werte sind fundamentale Bausteine für die Konstruktion eines Programmes
- Werte haben einen Datentyp, welcher sich durch unterschiedliche Wertbereiche definiert.
- Bei Zahlen z.B.
 - **Integers (int)**: Ganze Zahlen
 - 3
 - **FLOATS (float)**: Mit Dezimalstelle
Das Dezimalstellentrennzeichen ist ein Punkt!
 - 2.0
 - 3.22123

```
# Initialize the total number of questions
nr_total_questions_asked = 0 ← Integer
```

```
# Initialize the number of correct answers
nr_correct_answers = 0 ← Integer
```

Konstrukte im Detail – Werte

- Zeichenketten sind **Strings (str)**
- Werden mit " oder mit "" deklariert
 - 'Hello World'
 - "Hello World"

```
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]
...
if answer == correct_answer:
    print("Correct!")
```

Konstrukte im Detail – Werte

- **Boolsche Variablen (Booleans, bool)**
kennen nur zwei Werte
 - True
 - False
- Sie werden oft als Entscheidungsvariablen bei Bedingungen oder Schleifen verwendet
- Bedingungen werden ausgewertet und geben eine boolsche Variable zurück
- Vergleiche werden mit == ausgeführt

```
correct_answer = question_answer_pair[1]

if answer == correct_answer:
    print("Correct!")
    nr_correct_answers = nr_correct_answers + 1
```

Konstrukte im Detail – Werte

- **Listen (arrays)** sind eine Reihe von Werten
- Sie werden mit [] erstellt und die darin enthaltenen Werte mit Komma getrennt
 - [1, 2, 4]
 - ['Monty', 'Python']
- Datentypen innerhalb einer Liste können gemischt werden
 - ['monty', 10, 2.2]

```
["Wie heisst die Hauptstadt von Norwegen?", "Oslo"]
```

```
["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
```

Konstrukte im Detail – Werte

- Listen können verschachtelt werden.
D.h., dass eine Liste in einer Liste erzeugt werden kann.
- Die Liste questionnaire besteht aus mehreren Frage/Antwort Paaren, welche wiederum selbst Listen sind
- In der Informatik beginnt die Nummerierung mit 0

```
questionnaire = [
    ["Wie heisst die Hauptstadt von Norwegen?", "Oslo"],
    ["Wie heisst die Hauptstadt von Finnland?", "Helsinki"]
]
```

	0	1
0	Hauptstadt von Norwegen?	Oslo
1	Hauptstadt von Finnland?	Helsinki

Konstrukte im Detail – Werte

- Mit [INDEX] wird auf ein Element in einer Liste zugegriffen
- Der Index ist ein Kennzeichen für die Position eines Elementes

```
question =
    ["Hauptstadt von Norwegen?", "Oslo"]
# "Hauptstadt von Norwegen"
question[0]
# "Oslo"
Question[1]
```

...

```
questionnaire = [
    ["Hauptstadt von Norwegen?", "Oslo"],
    ["Hauptstadt von Finnland?", "Helsinki"]
]
# "Oslo"
Questionnaire[0][1]
```

Konstrukte im Detail – Operatoren

- Werte können mittels Operatoren kombiniert werden
- Ähnlich wie in der Mathematik werden dabei die Operationen nach einer bestimmten Reihenfolge ausgeführt:
 1. ()
 2. ** (potenzieren z.B. $2^{**}4 = 16$)
 3. *, /, %
 4. +, -

```
# Increment the question counter  
nr_total_questions_asked =  
    nr_total_questions_asked + 1
```

Konstrukte im Detail – Operatoren

- Strings (Zeichenketten) können mit dem + Operator zusammengefügt werden
- Es können nur Strings miteinander verbunden werden
 - Im Beispiel verwenden wir zwei Integer (nr_correct_answers, nr_total_questions_asked)
 - Damit diese mit den anderen Strings verbunden werden können, müssen die Zahlen zuerst in Strings konvertiert werden
 - Dies ist möglich mit str()

```
"You have correctly answered " + str(nr_correct_answers) + " of  
total " + str(nr_total_questions_asked) + " questions."
```

Konstrukte im Detail – Funktionen

- In Python gibt es viele vordefinierte Funktionen z.B. print(), str(), raw_input()
- Es können auch eigene Funktionen definiert werden
- In den Klammern können Argumente übergeben werden
- Als Resultat können Funktionen neue Werte zurückgeben

```
print("Correct!")  
  
number_as_string = str(122.20)  
  
answer = raw_input(question)
```

Konstrukte im Detail – Iteration

- Die gleiche Aktion kann beliebig oft wiederholt werden, sie ist also in einer Schleife
- Es gibt verschiedene Ausdrücke um zu iterieren
- Wir besprechen an dieser Stelle die for Schleife, welche erlaubt, durch eine Liste zu iterieren
- Es wird durch jedes Element innerhalb der Liste iteriert und das aktuelle Element wird in der angegeben Variablen zwischengespeichert
- Alles was zur Schleife gehört, **muss einen Tabulator eingerückt sein**

```
for VARIABLE in LIST:  
    BODY  
  
...  
  
# Go through every question answer pair  
for question_answer_pair in questionnaire:  
    # Enter loop with tab, first entry holds the question  
    question = question_answer_pair[0]  
    # Leaving loop  
    Print("not in the loop anymore")
```

Konstrukte im Detail – Bedingungen

- Boolsche Algebra / Logik ist ein Bereich der Mathematik
- Ein boolscher Ausdruck ist entweder wahr oder falsch
 - $5 == 5 \rightarrow \text{True}$
 - $5 == 6 \rightarrow \text{False}$
- Operatoren
 - $x == y$ gleich (Achtung: das einfache Gleichheitszeichen ist eine Wertzuweisung)
 - $x != y$ ungleich
 - $x > y$ grösser als
 - $x < y$ kleiner als
 - $x >= y$ grösser oder gleich
 - $x <= y$ kleiner oder gleich

Konstrukte im Detail – Bedingungen

- Das if Konstrukt erlaubt es, Anweisungen auszuführen, wenn eine Bedingung True ist.
- Ist die Bedingung wahr (True), werden die eingerückten Statements ausgeführt

```
if answer == correct_answer:  
    print("Correct!")  
    # Increment the score  
    nr_correct_answers =  
        nr_correct_answers + 1
```

Konstrukte im Detail – Bedingungen

- Das if Konstrukt kann um eine else Klausel erweitert werden
- Sie wird ausgeführt, wenn die if Bedingung falsch (False) ist
- «Entweder oder» in der deutschen Sprache
- Pro if darf nur ein else vorkommen

```
if answer == correct_answer:  
    print("Correct!")  
    # Increment the score  
    nr_correct_answers =  
        nr_correct_answers + 1  
  
else:  
    print("Wrong!")
```

Fazit

Folgende Konstrukte helfen beim Schreiben eines Programmes:

- Kommentare
- Variablen
- Werte
- Operatoren
- Funktionen
- Iteration
- Bedingungen