Vorkurs für Informatikstudierende

Dr. Julien Klaus

Skript zur Vorlesung

Inhaltsverzeichnis

1 Einführung Informatik

> Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mithilfe von Computern. - Gesellschaft für Informatik
>
> Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mithilfe von Digitalrechnern - Wikipedia

Angewandte Informatik beschäftigt sich mit konkreten Anwendungsgebieten der Informatik und ihren spezifischen Anforderungen (Beispiele: Bioinformatik, Wirtschaftsinformatik, medizinische Informatik).

Praktische Informatik behandelt Konzepte der Programmierung und der Entwicklung von Programmen und komplexen, informationsverarbeitenden Systemen (Beispiele: Softwaretechnik, Datenbanksysteme).

Technische Informatik beschäftigt sich mit Computer-Hardware, deren Entwurf und maschinennahen Aspekten (Beispiele: Rechnerarchitektur, Signalanalyse).

Theoretische Informatik biete theoretische Grundlagen für alle Teilbereiche der Informatik (Beispiele: Komplexitätstheorie, Algorithmik).

Abbildung 1.1: Grundlegender Ablauf in der Informatik

Die Informatik ist im Grunde die ganze Zeit damit beschäftigt Probleme zu lösen. Den grundlegenden Ablauf vom Problem bis zu Lösung oder Ausführung ist in Abbildung 1.1 dargestellt.

2 Algorithmik

Wir kommen nun von der Formalisierung unseres Problems zur Entwicklung und Analyse von Algorithmen.

2.1 Motivation

Algorithmen werden in vielen Bereichen des täglichen Lebens verwendet. Ob es ein Kochrezept ist, die IKEA Anleitung oder einfach die Reihenfolge in der wir uns am Morgen anziehen oder unsere Kleidung auswählen. Algorithmen sind allgegenwärtig. Selbst in diesem Vorkurs habt ihr bereits einige Algorithmen verwendet. Beispielsweise zur Umrechnung von dezimalen Zahlen in binär Zahlen.

Versuchen wir einmal einen eigenen Algorithmus für ein Problem zu entwickeln.

Wer ist der jüngste Studierenden hier im Raum? Entwickeln wir einen Algorithmus:

Problem Wir wollen den jüngsten Studierenden in der Vorlesung finden.

Formalisierung Wir könnten das Alter oder das Geburtsdatum benutzen. Weiter müssen wir uns überlegen ob wir nur die Alter der gerade anwesenden oder die Liste mit Anmeldungen benutzen wollen. Ist die Lösung eindeutig?

Algorithmenentwurf Wir durchlaufen die Liste und merken uns den kleinsten Wert. Weiter können wir auch eine Divide and Conquer Variante nehmen oder eine geeignete Datenstruktur nutzen.

2.2 Eigenschaften von Algorithmen

Jede Abfolge von Aktionen ist nicht gleich ein Algorithmus. Für Algorithmen gelten bestimmte Eigenschaften. So müssen Algorithmen Eindeutig, Ausführbar, Endlich, Determiniert, Deterministisch und Terminiert sein.

Definition 1 (Eindeutig)**.** *An jeder Stelle des Algorithmus muss* ***eindeutig*** *festgelegt sein, was als nächstes zu tun ist. Die Anweisung muss also unmissverständlich formuliert sein.*

Definition 2 (Endlichkeit)**.** *Ein Algorithmus darf nur eine* ***endliche*** *Länge besitzen. Zudem dürfen seine Daten nur endlich viel Platz belegen.*

Definition 3 (Deterministisch)**.** *Für jede Anweisung gibt es genau eine folge Anweisung. Wenn man sich aussuchen kann wie man weiter macht spricht man von* ***Nichtdeterministisch****.*

Definition 4 (Determiniertheit)**.** *Für die gleichen Eingabedaten muss der Algorithmus das gleiche Resultat liefern.*

Definition 5 (Terminierung)**.** *Ein Algorithmus muss nach einer* ***endlichen*** *Anzahl von Anweisungen beendet sein oder anhalten.*

Beispiel 1 (Sortieralgorithmus)**.** *Wir können uns die Eigenschaften von Algorithmen an einem einfachen Sortieralgorithmus verdeutlichen.*

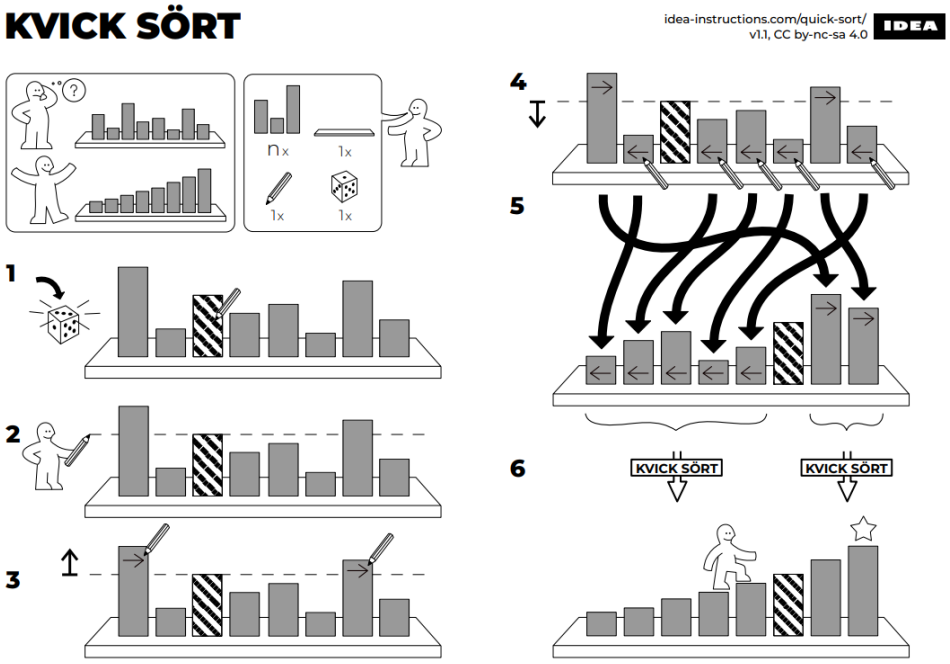


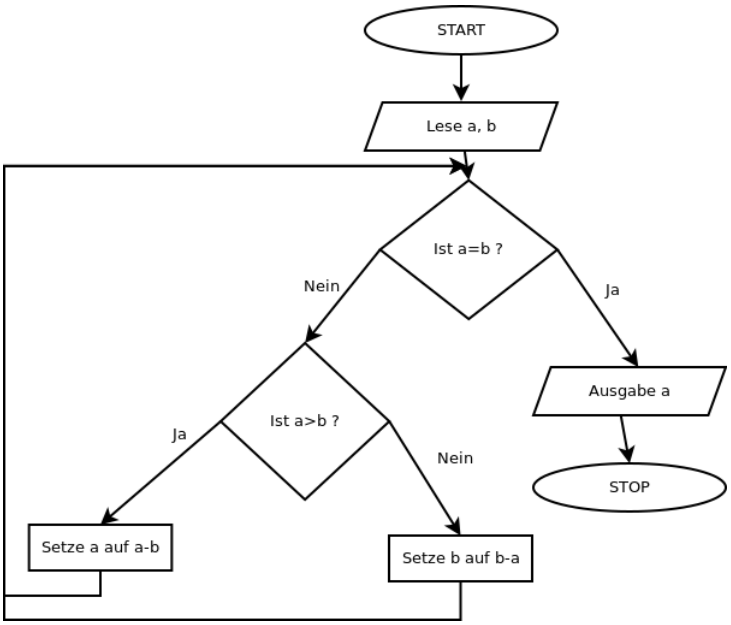
Abbildung 2.1: Erklärung des Quicksort Algorithmus mithilfe einer IKEA Anleitung.

Der dargestellte Sortieralgorithmus arbeitet, indem in jedem Schritt ein Element zur Teilung (in der Abbildung 2.1 schraffiert) ausgewürfelt wird, die Elemente bezüglich dieses Elementes sortiert und anschließend die Elemente kleiner und größer jeweis wieder mit dem Quicksort Algorithmus sortiert werden.

Die Schritte für diesen Algorithmus sind an jeder Stelle ***eindeutig*** *und durch die Nummerierung auch* ***deterministisch****. Weiter werden die Elemente in jedem Schritt geteilt und die Listen links und rechts werden kleiner (das Element zur Teilung wird nie wieder betrachtet). Dies führt dazu, das der Algorithmus nach einer* ***endlichen*** *Dauer endet (****Terminierung****) und jede Liste von Elementen am Ende sortiert ausgibt (****Determiniertheit****).*

2.3 Flussdiagramme und Pseudocode

Wenn man Algorithmen entwickelt muss man sie irgendwie darstellen. Viele tun dies mit Flussdiagrammen, noch mehr mit Pseudocode und sehr wenige mit Struktogrammen[1]




```
PROCEDURE fun (a,b)
  BEGIN
    INPUT a, b
    WHILE a is not b
      IF a > b
        THEN a <- a - b
        ELSE b <- b - a
      END IF
    END WHILE
    PRINT a
  END
END PROCEDURE
```

Pseudocode verwendet dafür bereits definierte Schlüsselworte. Vermutlich werden Sie recht schnell eine eigene Art entwickeln das Verhalten von Programmen zu beschreiben.

2.3.1 Flussdiagramme

Flussdiagramme sind eine grafische Darstellung von Algorithmen. Sie bestehen im Grunde aus Pfeilen und Symbolen. Wie diese Pfeile und Symbole auszusehen haben und welche Funktion sie haben sind in der DIN 66001 genormt.

2.3.2 Start und Stop

Um den Start und das Ende von Ablaufplänen zu zeigen gibt es spezielle Symbole, welche START und STOP beinhalten.

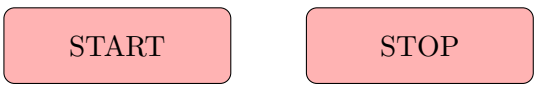


Abbildung 2.3: Beginn und Ende des Flussdiagrammes.

[1] Vergessen Sie gleich wieder.

2.3.3 Anweisungen

Für einfache Anweisungen wie $a = b + c$ verwendet man Rechtecke.

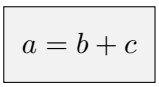


Abbildung 2.4: Eine einfache Anweisung.

2.3.4 Verzweigungen

Verzweigungen werden genutzt um zwischen Alternativen zu unterscheiden. Beispielsweise kann man das Maximum zweier Zahlen a, b bestimmen indem man fragt ob $a < b$ ist. Die Ausgehenden Pfeile werden meist links und rechts angebracht und mit JA und NEIN oder True und False oder $+$ und $-$ bezeichnet.

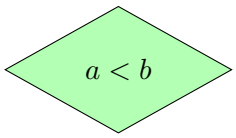


Abbildung 2.5: Eine Verzweigung.

2.3.5 Ein- und Ausgabe

Natürlich möchte man in seinem Algorithmus auch Daten eingeben oder Dinge ausgeben. Hierzu werden Ein- und Ausgabeblöcke verwendet.

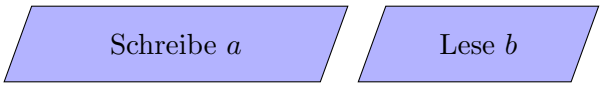


Abbildung 2.6: Eine Ausgabe und eine Eingabe.

2.3.6 Programmaufrufe

Oftmals möchte man wiederkehrende Funktionen nicht mehrfach aufschreiben und verwendet deshalb Programmaufrufe oder Unterprogramme.

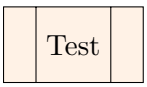
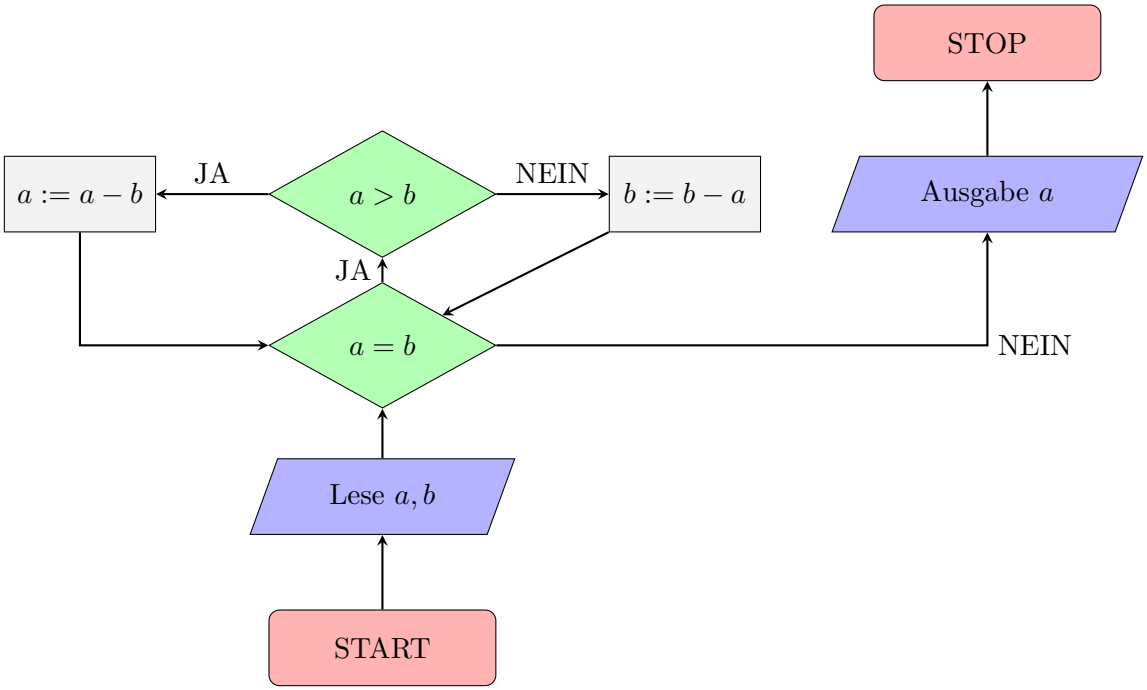


Abbildung 2.7: Ein Aufruf eines Unterprogrammes.

2.3.7 Anwendung von Flussdiagrammen

Hier nun nochmal das Beispiel von oben mit schönen bunten Bildern.



2.3.8 Andere Darstellungen

Pseudocode

Hier soll das oben dargestellte Beispiel noch einmal kurz im Pseudocode angegeben werden.

Algorithm 1 Beispielpseudocode

```
1: Lese a, b
2: while a ≠ b do
3:     if a > b then
4:         a := a − b
5:     else
6:         b := b − a
7:     end if
8: end while
9: Ausgabe a
```

Wie bereits erwähnt benutzt Pseudocode Schlüsselworte um die verschiedenen Kontrollfunktionen aus Flussdiagrammen zu ersetzen. Verzweigungen werden oftmals mit **if,**

else und Schleifen mit **while, for** eingeleitet. Weiter gibt es diverse Möglichkeiten die Blöcke zu begrenzen (in unserem Beispiel **do, end if, end while**).

Struktogramme

Der Vollständigkeit halber möchte ich zusätzlich ein Struktogramm zeigen. Dieses ist in Abbildung 2.8 abgebildet.

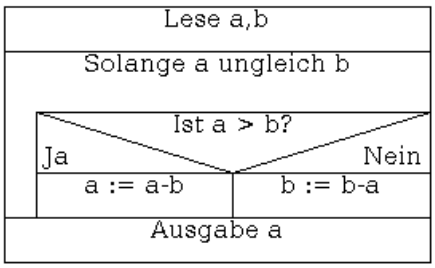


Abbildung 2.8: Ein Struktogramm unseres Beispieles.

3 Python

3.1 Geschichte

Python wurde 1991 von Guido van Rossum als Weiterentwicklung von ABC entwickelt. Das Ziel war eine leicht zu erlernende Programmiersprache zu entwickeln, die der Englischen Sprache angelehnt ist. Die erste Vollversion wurde 1994 veröffentlicht. Danach wurde 2000 Python 2.0, 2008 Python 3.0, 2015 Python 3.5 und 2016 Python 3.6 veröffentlicht. Der Name geht nicht wie viele Denken auf die Schlange zurück, sondern auf die Komikertruppe MontyPython. Trotzdem gibt es viele Anspielungen auf Schlangen in weiteren Paketen wie Boa oder in dem Tool Anacondo.

3.2 Installation und Start

Python kann sehr einfach über die Website `www.python.org` installiert werden. Wir verwenden in der Vorlesung Python 3. Die interaktive Konsole kann unter Linux mithilfe des Befehles `$python3` gestartet werden. Danach können hinter `>>>` die Python Befehle geschrieben werden.

3.3 Syntax

> Unter Syntax versteht man allgemein ein Regelsystem zur Kombination elementarer Zeichen zu zusammengesetzten Zeichen in natürlichen oder künstlichen Zeichensystemen. Die Zusammenfügungsregeln der Syntax stehen hierbei den Interpretationsregeln der Semantik gegenüber. - Wikipedia

Im folgenden besprechen wir einige Syntaxregeln der Sprache Python 3.0. Um mit Python zu programmieren können wir die interaktive Umgebung nutzen. Wir starten sie über die Konsole Mithilfe des Befehls `python3`.

```
$ python3
>>>
```

Listing 3.1: Start der interaktiven Konsole von Python3

3.3.1 Anweisungen

Anweisungen werden einfach eingegeben und mit ENTER bestätigt. Zuweisungen können mit `=` vorgenommen werden. In dem Beispiel wird a der Wert 3 zugewiesen und in der nächsten Zeile $2 + a$ berechnet. Das Ergebnis 5 wird sofort ausgegeben.

```
>>> i = 0
>>> n = 10
>>> while i < n:
        print(i)
        i = i+1
0
1
..
9
>>> for i in range(10):
        # range produziert d
        print(i)
0
1
..
9
```

Listing 3.4: Beispiele zu Schleifen

```
>>> a = 3
>>> 2 + a
5
```

Listing 3.2: Einfache Anweisungen

3.3.2 Verzweigungen

Verzweigungen werden Mithilfe des Schlüsselwortes `if` eingeleitet. Nach der Bedingung folgt ein :. In der nächsten Zeile werden nun mit einem Einschub die Anweisungen der `wahr` Fälle eingegeben. Folgend kann optional `elif` oder `else` angegeben werden. `elif` erwartet wieder eine Bedingung. Beide werden mit einem : beendet und in der nächsten Zeile mit Einschub die Anweisungen angegeben.

```
>>> if a > 2:
      print(a)
    elif a == 2:
      print("a ist 2")
    else:
      print("Fehler")
a ist 2
```

Listing 3.3: Verzweigung über a

3.3.3 Schleifen

Schleifen werden entweder mit `while` oder `for` eingeleitet. Der Unterschied besteht in der Verwendung. Die `while`-Schleife erwartet eine Bedingung und wird so lange ausgeführt, wie die Bedingung zu wahr ausgewertet wird. Die `for`-Schleife im Gegensatz kann dazu verwendet werden Elemente einer Liste oder eines Bereiches (`range`) zu durchlaufen.

```
>>> a = list()
>>> a.append(1)
>>> a.append(4)
>>> a.append(3)
>>> a.sort()
>>> print(len(a))
3
>>> for i in range(len(a)):
      print(i)
0
1
2
>>> for i in a:
      print(i)
1
3
4
>>> a.pop(1)
3
>>> print(a)
[1,4]
```

Listing 3.6: Verzweigung über a

3.3.4 Ein- und Ausgabe

Ausgaben werden einfach mit dem `print` Befehl realisiert. Mithilfe des Befehls `input` können Inhalte eingelesen werden. Diese werden als Zeichenkette eingelesen und können mittels `int` oder `float` umgewandelt werden.

```
>>> a = int(input("Zahl:-"))
Zahl: 3
>>> print(a)
3
```

Listing 3.5: Ein- und Ausgabe

3.3.5 Listen

Listen sind ein elementarer Datentyp von Python. Eine Liste kann einfach als `[]` oder `list()` angelegt werden. Mithilfe von `append` können Elemente hinzugefügt werden und `pop` entfernt ein Element. Die Länge der Liste liefert der Befehl `len`. Mithilfe von `sort` können Listen sortiert werden.

3.3.6 Funktionen

Wiederkehrende Aufgaben möchte man häufig nicht jedes mal neu schreiben sondern wiederbenutzen. Hierzu eigenet sich die Definition einer Funktion. Eine Beispielfunktion ist die Fakultät. Wir wollen nicht jedes mal neu beschreiben, wie man die Fakultät aussrechnen sondern lieber einfach die Zahl n angeben und das Ergebnis erhalten. Dies könnte die Funktion fak sein. n nennt man hierbei den Parameter.

```
def fak(n):
  m = 1
  for i in range(1,n+1):
    m *= i
  return m
```

Abbildung 3.1: Beispiel einer Fakultätsfunktion mit Parameter n.

Beispiel 2 (Binomialkoeffizient)**.** *Der Binomialkoeffizient ist eine Funktion die Anzahl der Möglichkeiten aus einer Menge mit n Objekten k Objekte auszuwählen zu berechnen. Angenommen wir schauen uns sämtliche Binomialkoeffizienten von* $n <= 100$ *und* $k < n$ *an. Wie viele dieser sind größer als* 10^6*?*

Um dieses Problem zu lösen, brauchen wir zuerst zwei Funktionen die uns die Fakultät und den Binomialkoeffizienten aussrechnet.

```
def fak(n):
  m = 1
  for i in range(1,n+1):
  m *= i
  return m

def binom(n,k):
  return fak(n)/(fak(k)*fak(n-k))
```

Abbildung 3.2: Die Funktionen Fakultät und Binomialkoeffizient

Nun können wir alle Möglichkeiten durchlaufen und schauen ob der Binomialkoeffizient größer als $10**6$ *ist. Wenn dies der Fall ist zählen wir eine Variable l eins hoch.*

```
l = 0
for n in range(1,101):
  for k in range(1,n+1):
    if binom(n,k) > 10**6:
      l += 1

print(l)
```

Abbildung 3.3: Durchtesten aller Möglichkeiten

Das `print` *an der letzten Stelle liefert uns die Antwort* 4075*.*

3.3.7 Rekursion

Wenn eine Funktionen einen Funktionsaufruf von sich selbst enthält reden wir von einer Rekursion. So können wir die Fakultätsberechnung aus dem vorherigen Beispiel auch rekursiv ausrechnen.

Der Aufruf `fak(3)` würde in diesem Fall als `fak(3) = 3*fak(2) = 3*2*fak(1) = 3*2*1 = 6` ausgewertet werden. Weitere prominente Beispiele sind die Berechnung der

Fibonacci Zahlen $f_n = f_{n-1} + f_{n_2}, f_1 = f_2 = 1, n \geq 3$ oder die Ackermann Funktion

$$\begin{aligned} a(0,m) &= m+1, \\ a(n+1,0) &= a(n,1), \\ a(n+1,m+1) &= a(n,a(n+1,m)). \end{aligned}$$

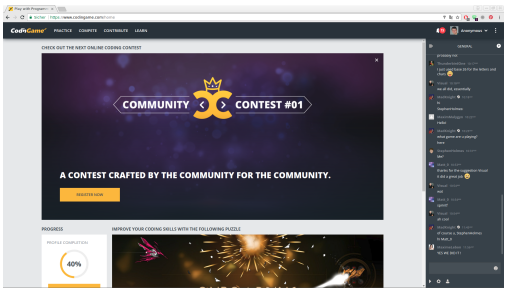
Sie können diese zur Übung selbstständig in Python Code übersetzen.

```
def fak(n):
  if n <= 1:
    return 1
  else:
    return n*fak(n-1)
```

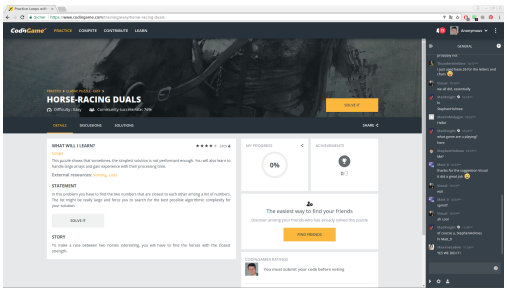
Abbildung 3.4: Rekursive Fakultätsberechnung

3.4 CodinGame

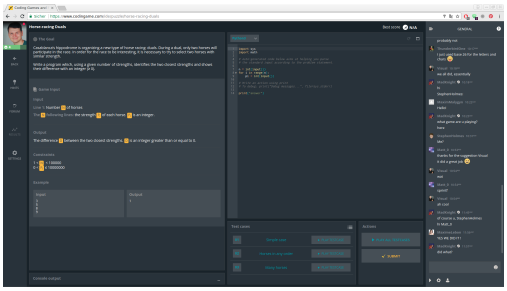
Für die Übung verwenden wir die Website CodinGame `www.codingame.com`. Die folgenden Bilder enthalten eine Übersicht über das Lösen eines Spieles. Wir verwenden als Beispiel das Spiel **Horse-racing Duals**.



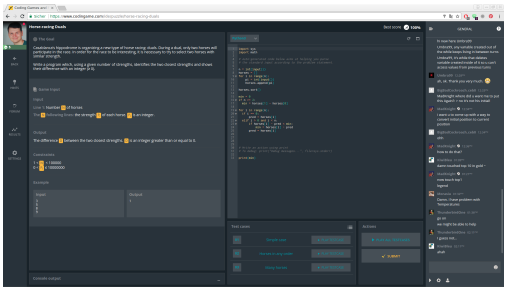
(a) Startseite von CodinGame



(b) Ein Beispielspiel



(c) Die Umgebung



(d) Die Umgebung mit Lösung

4 Kommandozeile

Das folgende Kapitel soll den Umgang mit der Konsole an einigen Beispielen demonstrieren. Hierzu werden Befehle angegeben. Die Befehle haben die Form `Befehl` *Parameter*.

4.1 Bewegen in Verzeichnissen

Am meisten muss man in der Konsole von einem Verzeichnis in ein anderes Verzeichnis mit den benötigten Dateien wechseln. Hierzu helfen die folgenden Befehle.

`cd` **Verzeichnis** wechselt in das Verzeichnis. Mit `cd ..` wird ein Verzeichnis nach oben gegangen.

`pwd` gibt den aktuellen Verzeichnispfad aus.

`ls` listet alle Ordner und Dateien des Verzeichnisses auf.

4.2 Hilfe

Sollte man einmal nicht weiter wissen kann man mit `man` *Befehl* das Manual zu dem Befehl öffnen oder mit `info` *Thema*, Befehle zu einem Thema finden.

4.3 Dateioperationen

Wenn man nun einmal in dem Verzeichnis gelandet ist, welches man benötigt kann man dort Dateien erzeugen, betrachten und löschen. Hierzu helfen folgende Befehle.

`cp` **Quelle Ziel** kopiert die Datei von der Quelle zum Ziel.

`mv` **Quelle Ziel** verschiebt die Datei von der Quelle zum Ziel.

`mkdir` **Verzeichnis** legt das Verzeichnis an. `rmdir` kann zum Löschen verwendet werden.

`touch` **Datei** erzeugt die Datei. `rm` kann zum Löschen verwendet werden.

`cat` **Datei** zeigt den Inhalt der Datei in der Konsole an. Ein alternativer Befehl ist `less`. Wenn man nur den Anfang oder das Ende der Datei anzeigen möchte kann man `head` oder `tail` benutzen.

`nano` **Datei** öffnet die Datei in einen Texteditor in der Konsole. Wenn die Datei noch nicht existiert, wird sie erzeugt.

`gedit` **Datei** öffnet die Datei in einem externen Texteditor.

4.4 Programme

Oft möchte man Programme in der Konsole starten. Diese werden einfach als `Befehl` gestartet. In unserem Kurs haben wir beispielsweise Java als `java` *-jar petri.jar* gestartet oder aber die interaktive Umgebung von Python3 als `python3`.

4.5 Weitere Befehle

Es stehen im Terminal noch weitere wichtige Befehle zur Verfügung, die uns die Arbeit im Linux Betriebssystem vereinfacht.

`clear` fügt genügend Leerzeilen ein, damit die aktuelle Befehlszeile wieder am oberen Anfang des Terminals steht.

`whatis` **Programm** liefert eine kurze Beschreibung eines übergebenen Programmes.

`which` **Programm** gibt den Installationspfad zum Programm an.

`df` zeigt die Auslastung des Dateisystems (disk free).

`ps -e` gibt eine Übersicht über die aktuell laufenden Prozesse aus. Jedem Prozess wird dabei eine PID zugewiesen.

`sudo killall -9` **PID** beendet den Prozess mit der übergebenen PID.

`shutdown -h` **delay** fährt den Rechner nach delay Minuten herunter. Falls kein delay angegeben wird fährt der Rechner sofort herunter.