EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



Praktische Informatik 1
Wintersemester 2026/27
Prof. Torsten Grust, Björn Bamberg, Nico Faden, Ludwig Kolesch
WSI — Database Systems Research Group

Übungsblatt 2

Abgabe bis spätestens **Mittwoch den 4. November 2026, 23:59 Uhr** auf Hand-In 🌐.

🪧 **Wähle die richtige Sprachebene!**
Bearbeitet die Aufgaben auf diesem Blatt mit der Sprachebene **Schreibe Dein Programm! — Anfänger**.

🤔 **Stehst du auf dem Schlauch?**
Steckst du irgendwo fest? Sind einige der Aufgaben unklar? Brauchst du den einen kleinen Schubs, um ins Rollen zu kommen? Schau in unser `Forum` 🌐—vielleicht findest du genau die Hilfe, die du brauchst.

Ⓚ **Klausur-ähnliche Aufgaben**
Die mit Ⓚ markierten Aufgaben sind ähnlich im Stil zu den Aufgaben, die ihr in der Klausur finden werdet. Nutzt diese Aufgaben also dazu, deine Erwartungen an die Klausur zu überprüfen und deine Fähigkeiten besser einzuschätzen.

ℹ️ **Wie sieht eine Funktionsdefinition aus?**
In dieser Vorlesung definieren wir Funktionen immer nach einem festen Schema. Dieses Schema besteht aus den fünf folgenden Teilen:

(1) einer Kurzbeschreibung (ein- bis zweizeiliger Kommentar) mit Bezug auf die Parameter(namen) und den Ergebniswert der Funktion,
(2) einer Signatur der Funktion,
(3) einigen Testfälle für die Funktion und
(4) einer Definition der Funktion mit ihrem
(5) Funktionsrumpf.

```
1  ; ...                                  ; (1)
2  (: f (... -> ...))                     ; (2)
3  (check-expect (f ...) ...)             ; (3)
4    ⋮
5  (check-within (f ...) ...)
6  (define f                              ; (4)
7    (lambda (p₁ p₂ ...)
8      ...))                              ; (5)
```

Aufgabe 1: Abstraktion von Teilproblemen Ⓚ

FUNKTION JE TEILPROBLEM

In den USA und in Europa gibt es unterschiedliche Maße für die Energieeffizienz von Kraftfahrzeugen:

- In Europa ist das gängige Maß der ***Verbrauch*** in Litern pro 100 Kilometer.
- In den USA ist das gängige Maß die ***Reichweite*** in Meilen pro Gallone.

💡 **Reduce, Reuse, Recycle!**
In den folgenden Teilaufgaben solltet ihr — sofern möglich — immer vorher definierte Funktionen und Konstanten verwenden, um die Aufgaben zu lösen. Das bedeutet, dass ihr die Funktionen/Konstanten aus den vorherigen Teilaufgaben in den nachfolgenden Teilaufgaben verwenden solltet.

A Schreibt eine Funktion `(: liters-per-100km (real real -> real))`, welche eine Menge Benzin in Litern und die Reichtweite dieser Menge Benzin in Kilometer akzeptiert und daraus den Verbrauch in Litern pro 100 Kilometer berechnet.

B Schreibt eine Funktion `(: miles-per-gallon (real real -> real))`, welche eine Entfernung in Meilen und den Benzinverbrauch auf diese Entfernung in Gallonen akzeptiert und daraus die Reichweite in Meilen pro Gallone berechnet.

C Definiert eine Konstante `kilometers-per-mile` (eine US-Meile entspricht 1,61 Kilometern). Schreibt anschließend unter Verwendung dieser Konstante zwei Funktionen `(: kilometers->miles (real -> real))` und `(: miles->kilometers (real -> real))`, welche jeweils eine Entfernung von einer Maßeinheit in die andere Maßeinheit umrechnen.

D Definiert eine Konstante `liters-per-gallon` (eine US-Gallone entspricht 3,79 Litern). Schreibt anschließend unter Verwendung dieser Konstante zwei Funktionen `(: liters->gallons (real -> real))` und `(: gallons->liters (real -> real))`, welche jeweils eine Menge Benzin von einer Maßeinheit in die andere Maßeinheit umrechnen.

E Schreibt eine Funktion `(: liters-per-100km->miles-per-gallon (real -> real))`, welche den Verbrauch in Litern pro 100 Kilometer akzeptiert und die Reichweite in Meilen pro Gallone berechnet.

> 💡 **Du bist dir unsicher, ob du auf dem richtigen Weg bist?**
> Dann kannst du deine Funktion mit folgendem Testfall überprüfen:

```
(check-within (liters-per-100km->miles-per-gallon 20) 11.76 0.1)
```

F Schreibt eine Funktion `(: miles-per-gallon->liters-per-100km (real -> real))`, welche die Reichweite in Meilen pro Gallone akzeptiert und den Verbrauch in Litern pro 100 Kilometer berechnet.

> 💡 **Du bist dir unsicher, ob du auf dem richtigen Weg bist?**
> Dann kannst du deine Funktion mit folgendem Testfall überprüfen:

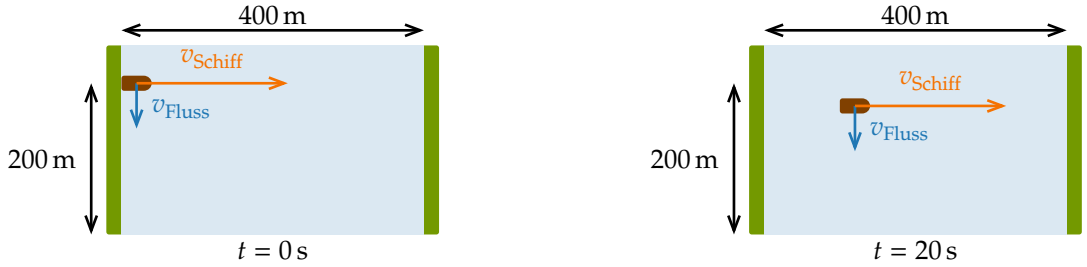
```
(check-within (miles-per-gallon->liters-per-100km 47.04) 5 0.01)
```

Aufgabe 2: Bilder und Simulation

WUNSCHDENKEN

Ein Schiff möchte einen $400\,\mathrm{m}$ breiten Fluss überqueren. Leider ist es dabei in seiner Manövrierfähigkeit dahingehend eingeschränkt, dass es nur genau senkrecht zur Flussrichtung steuern und mit einer konstanten Geschwindigkeit von $v_{\text{Schiff}} = 5\,\frac{\mathrm{m}}{\mathrm{s}}$ fahren kann. Wegen der Strömung des Flusses wird es dadurch mit $v_{\text{Fluss}} = 1.5\,\frac{\mathrm{m}}{\mathrm{s}}$ entlang des Flusses abgetrieben.

Der Kapitän des Schiffs hat gehört, dass ihr ihm bei solchen Problemen helfen könnt und fragt euch um einen Rat. Er möchte wissen, ob es ihm gelingen kann, noch vor einem Wasserfall, der $200\,\mathrm{m}$ flussabwärts des Ablegestegs liegt, das andere Ufer zu erreichen — und das am besten bevor er ablegt, versteht sich.



Basierend auf den in der Vorlesung erlernten Techniken stehen euch dazu zwei verschiedene Wege offen: *Berechnung* und *Simulation*.

Teil 1: Berechnung

Bei der Berechnung sind wir auf der Suche nach einer analytischen Lösung, welche uns erlaubt, die Frage direkt zu beantworten. Dazu können wir die Bewegung des Schiffes entlang des Flusses berechnen und damit prüfen, ob das Schiff den Wasserfall hinunterfallen wird oder nicht.

A Ⓚ Schreib eine Funktion `(: plunges? (real real -> boolean))`, welche gegeben eine Geschwindigkeit v_{Schiff} des Schiffes und eine Fließgeschwindigkeit v_{Fluss} (jeweils in $\frac{\mathrm{m}}{\mathrm{s}}$), bestimmt, ob das Schiff den Wasserfall hinunterfallen wird (`#t`) oder zuvor das andere Ufer erreicht (`#f`).

> 💡 **Programmieren mit Wunschdenken!**
> Geht bei der Definition von `plunges?` davon aus, dass die Funktion `drift` aus der nächsten Teilaufgabe schon existiert. Sie berechnet die für eine Geschwindigkeit v_{Schiff}, eine Fließgeschwindigkeit v_{Fluss} so wie eine gefahrene Strecke s (in Metern, senkrecht zur Flussrichtung), wie viele Meter d das Schiff in Flussrichtung „abdriftet".

B Ⓚ Implementiert anschließend die Funktion `(: drift (real real real -> real))` mithilfe der Formel:

$$d = \frac{s}{v_{\text{Schiff}}} \cdot v_{\text{Fluss}} \,.$$

Teil 2: Simulation

Die Berechnung ist zwar schön und gut, aber manche Probleme lassen sich nur schwer analytisch lösen. In solchen Fällen kann es hilfreich sein, eine Simulation zu verwenden, um das Problem zu visualisieren und so eine Lösung zu finden. Das Problem des Kapitäns hat sich zwar als analytisch lösbar herausgestellt, aber wir können die Simulation trotzdem nutzen, um das Verhalten des Schiffes zu beobachten.

> ⚠️ Achtung! Teachpack nicht vergessen.
> Die Simulation werden wir mittels des `image.rkt` Teachpacks aus der Vorlesung realisieren. Um das Teachpack zu verwenden, musst du es zuerst einbinden. Das entsprechende Menü dazu findest du unter *Sprache → Teachpack hinzufügen*, darin musst du dann das Teachpack auswählen und *OK* drücken.

C Als Erstes legen wir uns das Bildmaterial zurecht, das wir zum Zeichnen der Simulation brauchen werden. Lege drei Konstanten der Signatur `image` an:

- `river` für die Darstellung des Flusses,
- `banks` für die Darstellung der Ufer und
- `boat` für die Darstellung des Schiffes.

> 🎨 Deine Art-Skills sind gefragt!
> Hier könnt ihr euch kreativ austoben und bis zu 2 Bonuspunkte dafür bekommen, dass ihr die Simulation besonders anschaulich und individuell gestaltet. Ob ihr nun alles mittels des Teachpacks zusammenbaut oder die Bilder in einem Grafikprogramm erstellt, bleibt euch überlassen. Die kreativsten Simulationen werden dann in der Vorlesung vorgestellt.[1]

D Implementiert anschließend die Funktion `(: boat-at (natural -> image))`, die das Schiff — also `boat` aus Teilaufgabe **C** — an diejenige Position verschiebt, an der es sich zu einem gegebenen Zeitpunkt t (in Sekunden) nach der Abfahrt befindet. Nutzt dabei die Funktion `drift` aus Teilaufgabe **B** um die aktuelle (x, y)-Position des Schiffes zu ermitteln.

> 💡 Divide and Conquer!
> Denkt daran, größere Funktionen in Teilprobleme zu zerlegen und diese dann einzeln zu implementieren. So behaltet ihr den Überblick und könnt einfacher Fehler finden.

> 🖼️ Bilder malen leicht gemacht!
> Um Bilder an einer bestimmten Position zu zeichnen, könnt ihr — wie in der Vorlesung — die Funktion `put-pinhole` 🌐 aus dem `image.rkt` Teachpack verwenden.

E Zu guter Letzt kommt jetzt die eigentliche Simulation. Implementiere dazu die Funktion `(: crossing (natural -> image))`, die das Schiff auf seiner Reise über den Fluss zum Zeitpunkt t darstellt. Kombiniert dazu entsprechend den Fluss und das Ufer aus Teilaufgabe **C** und das Boot mittels `boat-at` aus Teilaufgabe **D** mittels `overlay/pinhole` 🌐 und `clear-pinhole` 🌐 — entsprechend dem Vorgehen in der Vorlesung.

> 🎬 Lights, Camera, Action!
> Statische Bilder sind zwar schön, aber bewegte Bilder sind noch schöner! Um die Simulation zu animieren, könnt ihr die Funktion `animate` 🌐 aus dem `universe.rkt` Teachpack verwenden — ruft dazu einfach `(animate crossing)` auf.
>
> Dabei wird die Funktion `crossing` 28× die Sekunde mit aufsteigenden Werten für t aufgerufen und das Ergebnis als Animation dargestellt. Diese Werte für t entsprechend dabei aber natürlich keinen ganzen Sekunden, sondern 28-steln davon, weshalb die Simulation ein wenig schneller läuft. Ihr könnt einen entsprechenden Korrekturfaktor innerhalb eurer Funktionen mittels Division/Multiplikation einbauen, um die Geschwindigkeit der Animation nach eurem Geschmack anzupassen.

[1] Solltet ihr nicht wollen, dass eure Simulation in der Vorlesung gezeigt wird, dann hinterlasst uns einfach einen entsprechenden Kommentar in eurem Code.