Aufschreibregeln für Funktionen

Einführung in die Programmierung

Johannes Brauer

31. Dezember 2019

Regeln 1 bis 3 zum Aufschreiben von Funktionen

Regel 1

Was tut diese Funktion?

```
(define e
  (lambda [a z l]
    (+ a
       (* a
          (/ z 100 12) l))))
```

Regel 1: Verwende aussagekräftige Namen für Funktionen und Variablen („sprechende Bezeichner")!

```
(define endkapital
  (lambda [anfangskapital jahreszinssatz laufzeit-in-monaten]
    (+ anfangskapital
       (* anfangskapital
          (/ jahreszinssatz 100 12) laufzeit-in-monaten))))
```

Regel 2

Regel 2: Beschreibe den **Zweck der Funktion** durch einen Satz, der das Resultat der Funktion **in Abhängigkeit von ihren Argumenten** beschreibt.

```
;; Aus dem anfangskapital, einem jahreszinssatz  und einer laufzeit-in-monaten wird
;; das Endkapital ermittelt, wobei die Zinsgutschrift einmalig am Ende der Laufzeit
;; erfolgt.
(define endkapital
  (lambda [anfangskapital jahreszinssatz laufzeit-in-monaten]
    (+ anfangskapital
       (* anfangskapital
          (/ jahreszinssatz 100 12) laufzeit-in-monaten))))
```

Regel 3

Regel 3: Schreibe Beispielanwendungen der Funktion auf, die das Resultat der Funktion mit dem Erwartungswert vergleicht. Die Auswertung der Ausdrücke muss dabei immer den Wahrheitswert `#true` liefern:

```
;; Aus dem anfangskapital, einem jahreszinssatz  und einer laufzeit-in-monaten wird
;; das Endkapital ermittelt, wobei die Zinsgutschrift einmalig am Ende der Laufzeit
;; erfolgt.
(define endkapital
  (lambda [anfangskapital jahreszinssatz laufzeit-in-monaten]
    (+ anfangskapital
       (* anfangskapital
          (/ jahreszinssatz 100 12) laufzeit-in-monaten))))
;; Beispielanwendungen:
(= (endkapital 1000 0 24) 1000)    ;; => #true
(= (endkapital 1000 12 12) 1120)   ;; => #true
(= (endkapital 1000 12 24) 1240)   ;; => #true
```

Funktionen, die nicht gemäß diesen Regeln aufgeschrieben sind, sind unzulässig!

(s. Aufgaben 5 und 6)

Ausdrücke in *Racket* (2)

Die in Abschnitt Ausdrücke in *Racket* (1) beschriebenen Operatoren und Operanden werden wie folgt ergänzt:

- Die Standardfunktion = vergleicht zwei Zahlen und liefert einen der beiden Wahrheitswerte `#true` oder `#false` als Resultat.
- Zum Vergleich von Zahlen gibt es außerdem die Standardfunktionen

  `<, >, <=, >=`

Zusammengesetzte Funktionen (Regeln 4 und 5)

Bei der Definition unserer bisherigen Funktionen waren wir darauf angewiesen, bereits existierende Funktionen (in Form der arithmetischen Operatoren) nutzen zu können. Für die Lösung komplexerer Aufgabenstellungen werden wir nicht mit einer einzigen eigenen Funktion auskommen sondern für die Lösung von Teilaufgaben auch eigene Funktionen (auch Hilfsfunktionen genannt) zu nutzen, die dann zur Gesamtlösung zusammengesetzt werden.

Die Nutzung von Hilfsfunktionen erleichtert nicht nur den Entwurf komplexerer Programme sondern erhöht auch ihre Lesbarkeit.

Betrachten wir dazu zwei Varianten einer Funktion, die die Fläche eines Kreisrings berechnet:

```
(define kreisring-flaeche
  (lambda [radius-aussen radius-innen]
    (- (* 3.14 (* radius-aussen radius-aussen))
       (* 3.14 (* radius-innen  radius-innen)))))
```

Unter der Voraussetzung, dass eine Funktion zur Berechnung der Kreisfläche zur Verfügung steht – wie z. B. diese:

```
(define kreis-flaeche
  (lambda [radius]
    (* 3.14 (* radius radius))))
```

–, kann die Funktion `kreisring-flaeche` auch so geschrieben werden:

```
(define kreisring-flaeche
  (lambda [radius-aussen radius-innen]
    (- (kreis-flaeche radius-aussen)
       (kreis-flaeche radius-innen))))
```

In der zweiten Variante der Funktion wird die Hilfsfunktion `kreis-flaeche` benutzt. Damit wird das Problem der Berechnung der Fläche des Kreisrings explizit auf die Subtraktion zweier Kreisflächen zurück geführt, während man dies in der ersten Variante erst aus dem Programmcode herauslesen müsste.

Betrachten wir nun als weiteres Beispiel eine Erweiterung der Problembeschreibung der Profitberechnung für den Eigentümer eines Vorstadtkinos.

Beispiel für die Nutzung von Hilfsfunktionen

> Er kann die Preise für die Eintrittskarten frei festlegen. Er hat einen exakten Zusammenhang zwischen dem Kartenpreis und der durchschnittlichen Besucheranzahl empirisch festgestellt: Bei einem Preis von 500 Währungseinheiten pro Karte kommen im Schnitt 120 Zuschauer. Reduziert er den Preis um 10 Währungseinheiten, erhöht sich die Besucherzahl um 15. Aber mehr Besucher verursachen höhere Kosten. Jede Veranstaltung kostet 18000 Währungseinheiten plus 5 Währungseinheiten für jeden Zuschauer. Der Eigentümer möchte nun wissen, wie groß der Profit bei einem bestimmten Kartenpreis ist.

Lösungsansatz

- Können wir dem Kinoeigentümer helfen?
- Versuchen wir zunächst die Abhängigkeiten zu erkennen, die sich aus der Problembeschreibung ableiten lassen:
  1. Der *Profit* ist die Differenz aus Einnahmen und Kosten.
  2. Die *Einnahmen* sind das Produkt aus Kartenpreis und Besucheranzahl.
  3. Die *Kosten* sind die Summe aus den Fixkosten (18000 Währungseinheiten) und dem Produkt aus Besucheranzahl und den Kosten pro Besucher (5 Währungseinheiten).
  4. Schließlich gibt es noch den Zusammenhang zwischen *Besucheranzahl* und Kartenpreis.

Vorgehensweise

Die weitere Vorgehensweise ist nun dadurch gekennzeichnet, dass für jeden der ermittelten Zusammenhänge je eine Funktion geschrieben wird. Dabei gehen wir in 3 Schritten vor:

1. Für jede Funktion wird zunächst gemäß Regel 2 ihre Zweckbestimmung und ihr Funktionskopf (Name der Funktion und die Parameterliste) aufgeschrieben.

2. Jetzt werden gemäß Regel 3 Beispielanwendungen hinzugefügt.

3. Schließlich wird für jede Funktion die Berechnungsvorschrift, der Funktionrumpf, ermittelt.

Schritt 1 für die Funktion `profit` und ihre Hilfsfunktionen **Die Funktion `profit`:**

```
;; berechnet den Profit aus der Differenz zwischen
;; Einnahmen und Kosten bei gegebenem Kartenpreis
(define profit
  (lambda [kartenpreis]
    nil))
```

Das Symbol `nil` steht hier als Platzhalter für den noch zu bestimmenden Funktionsrumpf.

Die Berechnung der Einnahmen:

```
;; berechnet die Einnahmen aus dem Produkt von
;; Besucherzahl und Kartenpreis
(define einnahmen
  (lambda [kartenpreis]
    nil))
```

Die Berechnung der Kosten:

```
;; berechnet die entstehenden Kosten bei gegebenem
;; Kartenpreis aus Fixkosten und variablen Kosten
(define kosten
  (lambda [kartenpreis]
    nil))
```

Die Berechnung der Besucherzahl:

```
;; berechnet die Besucherzahl bei gegebenem
;; Kartenpreis nach empirisch ermittelter Formel
(define besucherzahl
  (lambda [kartenpreis]
    nil))
```

Schritt 2 für die Funktion `profit` und ihre Hilfsfunktionen In diesem Schritt versuchen wir Beispielanwendungen zu finden; beginnen wir mit der Funktion `besucherzahl`.

Aus der Problembeschreibung lassen sich direkt die beiden folgende Beispiele ableiten:

- (besucherzahl 500) = 120
- (besucherzahl 490) = 135

Für weitere Beispiele müssten Annahmen getroffen werden, die die Problembeschreibung nicht direkt hergibt. So sind die beiden folgenden Beispiele nur dann korrekt, wenn aus den Beispielen der Problembeschreibung eine linearer Zusammenhang zwischen dem Kartenpreis und der Besucherzahl angenommen wird:

- (besucherzahl 510) = 105
- (besucherzahl 400) = 270

Ob diese Annahme sinnvoll ist, müsste letztendlich der Auftraggeber entscheiden. Für die folgenden Betrachtungen legen wir diese Annahme zugrunde.

Ein linearer Zusammenhang ergäbe die folgende Geradengleichung:

$$besucherzahl(kartenpreis) = 120 + \frac{15}{10} \cdot (500 - kartenpreis)$$

Beispielanwendungen für die Funkton `besucherzahl`:

```
;; berechnet die Besucherzahl bei gegebenem
;; Kartenpreis nach empirisch ermittelter Formel
(define besucherzahl
  (lambda [kartenpreis]
    nil))

;; Beispielanwendungen
(= (besucherzahl 500) 120)
(= (besucherzahl 490) 135)
(= (besucherzahl 510) 105)
(= (besucherzahl 400) 270)
```

Beispielanwendungen für die Funkton `kosten`:

```
;; berechnet die entstehenden Kosten bei gegebenem
;; Kartenpreis aus Fixkosten und variablen Kosten
(define kosten
  (lambda [kartenpreis]
    nil))

;; Beispielanwendungen
(= (kosten 500) 18600)
(= (kosten 400) 19350)
```

Beispielanwendungen für die Funkton `einnahmen`:

```
;; berechnet die Einnahmen aus dem Produkt von
;; Besucherzahl und Kartenpreis
(define einnahmen
  (lambda [kartenpreis]
    nil))

;; Beispielanwendungen
(= (einnahmen 500) 60000)
(= (einnahmen 400) 108000)
```

Beispielanwendungen für die Funkton `profit`:

```
;; berechnet den Profit aus der Differenz zwischen
;; Einnahmen und Kosten bei gegebenem Kartenpreis
(define profit
  (lambda [kartenpreis]
    nil))

;; Beispielanwendungen
(= (profit 500) 41400)
(= (profit 400) 88650)
```

Schritt 3: Ermittlung des Funktionsrumpfs für die Funktion `profit` und ihre Hilfsfunktionen Beginnen wir wieder mit der **Funktion `besucherzahl`**. Der für die Ermittlung der Beispielanwendungen zugrunde gelegte lineare Zusammenhang zwischen Besucherzahl und Kartenpreis wird direkt in die Berechnungsvorschrift für die Funktion genutzt:

```
;; berechnet die Besucherzahl bei gegebenem
;; Kartenpreis nach empirisch ermittelter Formel
(define besucherzahl
  (lambda [kartenpreis]
    (+ 120 (* (/ 15 10) (- 500 kartenpreis)))))

;; Beispielanwendungen
(= (besucherzahl 500) 120)
(= (besucherzahl 490) 135)
(= (besucherzahl 510) 105)
(= (besucherzahl 400) 270)
```

Die Berechnungsvorschrift der **Funktion `kosten`** ergibt sich aus dem oben formulierten Zusammenhang (Die Kosten sind die Summe aus den Fixkosten (18000 Währungseinheiten) und dem Produkt aus Besucheranzahl und den Kosten pro Besucher (5 Währungseinheiten).):

```
;; berechnet die entstehenden Kosten bei gegebenem
;; Kartenpreis aus Fixkosten und variablen Kosten
(define kosten
  (lambda [kartenpreis]
    (+ 18000 (* 5 (besucherzahl kartenpreis)))))

;; Beispielanwendungen
(= (kosten 500) 18600)
(= (kosten 400) 19350)
```

Man beachte, dass die Funktion `kosten` die Funktion `besucherzahl` als Hilfsfunktion benutzt.

Die Berechnungsvorschrift für die **Funktion `einnahmen`** als Produkt aus Besucherzahl und Kartenpreis:

```
;; berechnet die Einnahmen aus dem Produkt von
;; Besucherzahl und Kartenpreis
(define einnahmen
```

```
  (lambda [kartenpreis]
    (* (besucherzahl kartenpreis) kartenpreis)))

;; Beispielanwendungen
(= (einnahmen 500) 60000)
(= (einnahmen 400) 108000)
```

Man beachte auch hier die Verwendung von `besucherzahl` als Hilfsfunktion.

Schließlich werden in der **Berechnungsvorschrift für** `profit` die Funktionen `einnahmen` und `kosten` benutzt:

```
;; berechnet den Profit aus der Differenz zwischen
;; Einnahmen und Kosten bei gegebenem Kartenpreis
(define profit
  (lambda [kartenpreis]
    (- (einnahmen kartenpreis)
       (kosten kartenpreis))))

;; Beispielanwendungen
(= (profit 500) 41400)
(= (profit 400) 88650)
```

Zusammenfassung Regel 4 (Hilfsfunktionen):

> **Definiere für jeden Zusammenhang zwischen Größen, die sich aus der Problembeschreibung ergeben, eine Funktion.**

Prozedurale Abstraktion

- Die Funktion `profit`, wie sie oben definiert wurde, zerlegt das Problem der Berechnung des Profit in zwei Teilprobleme, nämlich auf die Berechnung der Einnahmen und der Kosten.
- Die Funktionen `einnahmen` und `kosten` liefern eine Abstraktion von den Details ihrer Berechnungen. Auch wenn sich diese Details verändern, kann die Funktion `profit` unverändert bleiben.

Man könnte die Funktion `profit` auch ohne die Verwendung von Hilfsfunktionen aufschreiben:

```
(define profit
  (lambda [kartenpreis]
    (- (* (+ 120 (* (/ 15 10 )
                    (- 500 kartenpreis)))
          kartenpreis)
       (+ 18000
          (* 5 (+ 120
                  (* (/ 15 10 )
                     (- 500 kartenpreis))))))))
```

Diese Funktion berechnet das Gleiche wie die Variante mit Hilfsfunktionen, ist aber völlig unlesbar. Sollte in dieser Funktion der Zusammenhang zwischen Kartenpreis und Besucherzahl geändert werden müssen, wäre dies eine nur schwer lösbare Aufgabe.

Konstantendefinitionen Die Lesbarkeit der Funktion `profit` (in beiden Varianten) und ihrer Hilfsfunktionen ist auch dadurch eingeschränkt, dass in den Berechnungsvorschriften eine Reihe von Zahlen auftreten, deren Bedeutung nicht unmittelbar einleuchtend sein muss.

Regel 5 Konstantendefinitionen:

> **Ersetze jede Konstante, deren Bedeutung sich nicht aus dem Kontext ergibt, durch einen sprechenden Variablennamen.**

Zum Beispiel für die Funktion `profit` und ihre Hilfsfunktionen:

```
(define fixkosten 18000)
(define kosten-pro-besucher 5)
(define basis-besucherzahl 120)
(define besucher-preis-faktor (/ 15 10))
(define preis-fuer-basis-besucherzahl 500)
```

(Das vollständige Programm `profit` (kino.rkt) steht in moodle zur Verfügung.)

Gegenbeispiel: In der Formel zur Berechnung der Dreiecksfläche

```
(/ (* grundseite hoehe) 2)
```

nicht die Konstante 2 durch eine eine benannte Konstante `zwei` ersetzen. (s. Aufgaben 7 und 8)

Fünf Regeln

Regel 1 Verwende aussagekräftige Namen für Funktionen und Variablen („sprechende Bezeichner“).

Regel 2 Beschreibe den Zweck der Funktion durch einen Satz, der das Resultat der Funktion in Abhängigkeit von ihren Argumenten beschreibt.

Regel 3 Schreibe Beispielanwendungen der Funktion auf, die das Resultat der Funktion mit dem Erwartungswert vergleichen.

Regel 4 Definiere für jeden Zusammenhang zwischen Größen, die sich aus der Problembeschreibung ergeben, eine Funktion.

Regel 5 Ersetze jede Konstante, deren Bedeutung sich nicht aus dem Kontext ergibt, durch einen sprechenden Variablennamen.