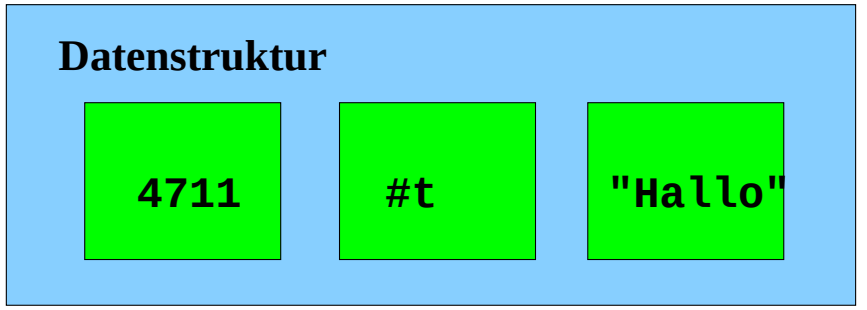
2 Zusammengesetzte und gemischte Daten

Zusammengesetzte Daten



- Ein Wert eines zusammengesetzten Datentyps besteht aus mehreren Werten, die zu unterschiedlichen Datentypen gehören können.
- *Record*, Produkt
- Beispiele
  - Schokokeks
  - Kartesische Koordinate
  - Vier-Gang Menü

Gemischte Daten

- Ein Wert eines gemischten Datentyps besteht aus einem Wert, der eine alternative Auswahl aus mehreren unterschiedlichen Datentypen trifft.
- *Variante*, Summe
- Beispiele
  - Keks = **entweder** Schokokeks **oder** Marmeladenkeks
  - Koordinate = **entweder** kartesische Koordinate **oder** Polarkoordinate
  - Essen = **entweder** Frühstück **oder** Mittagessen **oder** Abendessen
- Kennzeichen eines gemischten Datentyps:
  Gemeinsame Operationen auf allen Alternativen

2.1 Zusammengesetzte Daten

2.1.1 Der Schokokeks

Schokokeks:

Feld	Komponente
Schoko:	12g
Keks:	14g

2.1.2 Schokokekse im Computer

- Vordefinierter zusammengesetzter Datentyp `chocolate-cookie`
- Die **Konstruktion** eines Schokokeks hat den Vertrag

```
; schokokeks erzeugen
(: make-chocolate-cookie (real real -> chocolate-cookie))
```

- `make-chocolate-cookie` ist der **Konstruktor**
- Ergebnis ist ein Wert

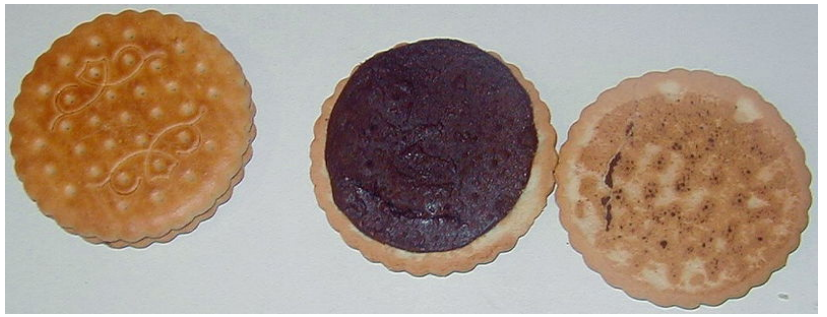
```
   (make-chocolate-cookie 12 14)
=> #<record:chocolate-cookie 12 14>
```

- Schokokekse können an Variable gebunden werden

```
   (define doppelkeks (make-chocolate-cookie 12 28))
```

```
   doppelkeks
=> #<record:chocolate-cookie 12 28>
```

2.1.3 Schokokekse zerlegen



Quelle User:Eldred http://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/Doppelkeks.jpg/800px-Doppelkeks.jpg

- Zugriff auf die Komponenten mit den **Selektoren**

```
; Selektor: Schoko-Anteil ermitteln
(: chocolate-cookie-chocolate (chocolate-cookie -> real))
; Selektor: Keks-Anteil ermitteln
(: chocolate-cookie-cookie (chocolate-cookie -> real))
```

- Beispiel

```
   (chocolate-cookie-chocolate doppelkeks)
=> 12
   (chocolate-cookie-cookie doppelkeks)
=> 28
```

2.1.4 Gleichungen für Konstruktoren und Selektoren

- Das Zerlegen eines gerade konstruierten Schokokekses liefert wieder genau die ursprünglichen Komponenten.
- *Definierende Gleichungen für Selektoren*

  (chocolate-cookie-chocolate (make-chocolate-cookie x y)) $= x$
  (chocolate-cookie-cookie (make-chocolate-cookie x y)) $= y$

2.1.5 Typprädikat für Schokokekse

```
   (: chocolate-cookie? (%value -> boolean))
   (chocolate-cookie? doppelkeks)
=> #t
   (chocolate-cookie? 4711)
=> #f
   (chocolate-cookie? "chocolate-cookie")
=> #f
```

2.1.6 Gewicht eines Schokokekses

Das Gewicht eines Schokokekses ist die Summe der Gewichte der Komponenten.

Mit bisherigen Konstruktionsanleitungen:

```
; Gewicht eines Schokokekses bestimmen
(: chocolate-cookie-weight (chocolate-cookie -> real))
(define chocolate-cookie-weight
  (lambda (c)
    ...))
```

Da `c` vom Typ `chocolate-cookie` ist, *müssen* im Rumpf die Selektoren verwendet werden.

```
(define chocolate-cookie-weight
  (lambda (c)
    (+ (chocolate-cookie-chocolate c)
       (chocolate-cookie-cookie c))))
```

2.2 Definition von Records

- Zusammengesetzte Daten können in Scheme durch den Programmierer als *Records* definiert werden.
- Voraussetzung: Die Komponentendaten sind bekannt.
- Beispiel: Eine kartesische Koordinate in der Ebene besteht aus einer X- und einer Y-Koordinate.
- Was muss definiert werden?
  - Name des Typs
  - Name des Konstruktors
  - Name des Typprädikats (Typtest)
  - Liste der Namen der Selektoren

2.2.1 Definition von kartesischen Koordinaten

- Eine kartesische Koordinate in der Ebene besteht aus einer X- und einer Y-Koordinate: 

- Die Record-Definition

```
(define-record-procedures cartesian
  make-cartesian cartesian?
  (cartesian-x cartesian-y))
```

bindet die folgenden Namen:

`cartesian`	Name des Typs (für Verträge)
`make-cartesian`	Konstruktor
`cartesian?`	Typprädikat
`cartesian-x`	Selektor der ersten Komponente
`cartesian-y`	Selektor der zweiten Komponente

2.2.2 Verwendung der kartesischen Koordinaten

Konstruktor

```
; kartesische Koordinate erzeugen
(: make-cartesian (real real -> cartesian))
```

Beispiele

```
   (make-cartesian 0 0)
=> #<record:cartesian 0 0>
   (make-cartesian 100 50)
=> #<record:cartesian 100 50>
```

Typprädikat

```
; Test auf Vorliegen einer kartesischen Koordinate
(: cartesian? (%value -> boolean))
```

Beispiele

```
   (cartesian? #t)
=> #f
   (cartesian? (make-cartesian 17 4))
=> #t
   (cartesian? (make-chocolate-cookie 22 22))
=> #f
```

Selektoren

```
(: cartesian-x (cartesian -> real))
(: cartesian-y (cartesian -> real))
```

Definierende Gleichungen

(cartesian-x (make-cartesian x y)) $= x$
(cartesian-y (make-cartesian x y)) $= y$

Beispiele

```
   (cartesian-x (make-cartesian 17 4))
=> 17
   (cartesian-y (make-cartesian 17 4))
=> 4
   (cartesian-x (make-cartesian -1 6/7))
=> -1
   (cartesian-y (make-cartesian -1 6/7))
=> 0.857142
```

2.2.3 Allgemeine Form von `define-record-procedures`

`(define-record-procedures` t
  c p
  `(`s_1 `...` s_n`))`

definiert einen zusammengesetzten Datentyp (Record) mit

- t ist der Name des definierten Typs
- c ist der Name des Konstruktors
- p ist der Name des Typprädikats
- s_i sind die Namen der Selektoren

2.2.4 Informelle Datendefinitionen

Muss vor der formellen Definition erstellt werden!

Schokokekse

```
; Ein Schokokeks ist ein Wert
;  (make-chocolate-cookie x y)
; wobei x und y Zahlen sind, die den Schoko- bzw. den Keks-Anteil
; des Schokokekses darstellen.
```

Kartesische Koordinaten

```
; Eine kartesische Koordinate in der Ebene ist ein Wert
;  (make-cartesian x y)
; wobei x und y Zahlen sind, die den X- bzw. den Y-Anteil der
; Koordinate darstellen.
```

2.2.5 Konstruktionsanleitung 3 (Zusammengesetzte Daten)

Wenn bei der Datenanalyse zusammengesetzte Daten vorkommen, so muss zunächst ermittelt werden, zu welchen Sorten die Komponentendaten gehören.

Dann wird eine *informelle Datendefinition* erstellt:

```
; Ein t ist ein Wert
;  (c f1 ... fn)
; wobei ...
```

Dabei ist t der Name des zu definierenden Typs, c der Name des Konstruktors und die f_i die Namen der Komponenten. Die anschließende Beschreibung muss für jede Komponente die Sorte sowie eine kurze Erläuterung der Komponente enthalten.

Daraus ergibt sich die Record-Definition

```
(define-record-procedures t
  c p
  (s1 ... sn))
```

in der noch Namen für das Typprädikat p und die Selektoren s_i gewählt werden müssen.

2.3 Records konsumieren

2.3.1 Abstand vom Ursprung

```
; Abstand vom Ursprung bestimmen
(: distance-to-origin (cartesian -> real))
```

Gerüst dazu

```
(define distance-to-origin
  (lambda (xy)
    ...))
```

Konsumieren eines Records bedeutet, es in seine Komponenten zu zerlegen.
Also müssen im Rumpf die Selektoren vorkommen ⇒ Schablone für den Rumpf:

```
(define distance-to-origin
  (lambda (xy)
    ...(cartesian-x xy) ... (cartesian-y xy) ...))
```

Ausfüllen der Ellipse

- Bekannt: Abstand eines Punktes vom Ursprung

$$d = \sqrt{x^2 + y^2}$$

- Bekannt: Prozedur zum Quadrieren

```
; Eine Zahl quadrieren
(: square (number -> number))
(define square
  (lambda (x)
    (* x x)))
```

- Vordefiniert: Vertrag für die Quadratwurzel

```
; Quadratwurzel ziehen
; (: sqrt (number -> number))
```

Vollständige Prozedur

```
; Abstand vom Ursprung bestimmen
(: distance-to-origin (cartesian -> real))
(define distance-to-origin
  (lambda (xy)
    (sqrt (+ (square (cartesian-x xy))
             (square (cartesian-y xy))))))
```

- aus der Konstruktionsanleitung
- aus der Schablone
- aus Formel unter Verwendung bereits vorhandener Prozeduren

2.3.2 Konstruktionsanleitung 4 (Records als Argumente)

Sei x ein Prozedurargument vom Recordtyp t.

- Ermittle die Komponenten des Recordtyps t, von denen das Ergebnis abhängt.
- Im Rumpf der Prozedur muss für jede dieser Komponenten der Ausdruck $(s\ x)$ auftreten, wobei s der entsprechende Selektor von t ist.
- Vervollständige den Rumpf durch Konstruktion eines Ausdrucks, in dem diese Selektorausdrücke vorkommen.

2.4 Records produzieren

Kartesische Koordinaten können verschoben, gespiegelt, gedreht oder gestreckt werden.

Wir betrachten das Verschieben.

```
; Koordinate verschieben
(: cartesian-move (cartesian real real -> cartesian))
```

Gerüst dazu

```
(define cartesian-move
  (lambda (c dx dy)
    ...))
```

Konstruktionsanleitung "Records als Argumente" verwenden

```
(define cartesian-move
  (lambda (c dx dy)
    ...(cartesian-x c) ... (cartesian-y c) ...))
```

2.4.1 Koordinate verschieben

```
(define cartesian-move
  (lambda (c dx dy)
    (make-cartesian (+ (cartesian-x c) dx)
                    (+ (cartesian-y c) dy))))
```

- Bestimme den Werte beider Komponenten
- Verwende den Konstruktor

2.4.2 Konstruktionsanleitung 5 (Zusammengesetzte Daten als Ausgabe)

Falls eine Prozedur als Ergebnis einen neuen Wert eines zusammengesetzten Datentyps liefert, so muss in ihrem Rumpf der Konstruktor des zugehörigen Recordtyps auftreten.

2.5 Gemischte Daten

2.5.1 Marmelade-Creme-Kekse


```
; Ein Marmelade-Creme-Keks ist ein Wert
;  (make-jelly-cream-cookie x y z)
; wobei x, y und z Zahlen sind, die den Creme-, Marmeladen-
; bzw. Keks-Anteil darstellen.
```

2.5.2 Marmelade-Creme-Kekse als Record

Nach der Konstruktionsanleitung für zusammengesetzte Daten ergibt sich sofort die Recorddefinition:

```
(define-record-procedures jelly-cream-cookie
  make-jelly-cream-cookie jelly-cream-cookie?
  (jelly-cream-cookie-cream
   jelly-cream-cookie-marmalade
   jelly-cream-cookie-cookie))
```

2.5.3 Gewicht eines Marmelade-Creme-Kekses

```
; Gewicht eines Marmelade-Creme-Kekses ermitteln
(: jelly-cream-cookie-weight (jelly-cream-cookie -> real))
(define jelly-cream-cookie-weight
  (lambda (jcc)
    (+ (jelly-cream-cookie-cream jcc)
       (jelly-cream-cookie-marmalade jcc)
       (jelly-cream-cookie-cookie jcc))))
```

2.5.4 Gemischter Datentyp `keks`

- Ein Keks ist entweder ein Schokokeks oder ein Marmelade-Creme-Keks.
- Dies wird durch folgende informelle Datendefinition angemeldet:

```
; Ein Keks ist eine der folgenden Alternativen
; - ein Schokokeks
; - ein Marmelade-Creme-Keks
; Name: cookie
(define-contract cookie
  (mixed chocolate-cookie jelly-cream-cookie))
```

- Dadurch wird eine neue Sorte `cookie` definiert (inkl. Vertrag).
- Neue Form: `define-contract`
- Neuer Vertragsoperator: `(mixed` ⟨*contract*⟩ `...)`
  Der Vertrag `(mixed` ⟨*contract*⟩ `...)` gilt, wenn einer der ⟨*contract*⟩s gilt.

2.5.5 Prozeduren mit gemischtem Datentyp als Argument

- Definiere das Gewicht eines Kekses.

```
; Gewichte eines Kekses ermitteln
(: cookie-weight (cookie -> real))
(define cookie-weight
  (lambda (c)
    ...))
```

- Daten eines gemischten Typs fallen „natürlich“ in Kategorien, die den Fällen in der Definition des Typs entsprechen. Daher findet (eine Spezialisierung) des Musters für Fallunterscheidung Anwendung.
- Jede Alternative in der Definition des gemischten Typs liefert einen alternativen Fall im Rumpf von `cookie-weight`.

2.5.6 Gemischter Datentyp ⇒ Fallunterscheidung

```
; Gewichte eines Kekses ermitteln
(: cookie-weight (cookie -> real))
(define cookie-weight
  (lambda (c)
    (cond
     ((chocolate-cookie? c) ...)
     ((jelly-cream-cookie? c) ...))))
```

- In der Ellipse des ersten Falls ist klar, dass `c` ein `chocolate-cookie` ist.
- In der Ellipse des zweiten Falls ist klar, dass `c` ein `jelly-cream-cookie` ist.
- Also können dort die Prozeduren für die entsprechenden Sorten aufgerufen werden.

2.5.7 Gewicht eines Kekses, Lösung

```
; Gewichte eines Kekses ermitteln
(: cookie-weight (cookie -> real))
(define cookie-weight
  (lambda (c)
    (cond
     ((chocolate-cookie? c)
      (chocolate-cookie-weight c))
     ((jelly-cream-cookie? c)
      (jelly-cream-cookie-weight c)))))
```

2.5.8 Konstruktionsanleitung 6 (gemischte Daten)

Wenn bei der Datenanalyse gemischte Daten auftauchen:

- erstelle informelle Datendefinition der Form

  ; Ein x der Sorte s ist eine der folgenden Alternativen
  ; - ein x_1
  ;   ...
  ; - ein x_n

  Die x_i benennen die unterschiedlichen Sorten, die ein x sein kann.

- Definiere s als Name der neuen Sorte zur Verwendung in Verträgen.

  (define-contract s (mixed x_1 ... x_n))

- Wenn die Typprädikate für die beteiligten Sorten $p_1, \ldots, p_n$ heißen, so tritt in einer Prozedur, die ein Argument a der Sorte s konsumiert, folgende Verzweigung auf:

  (cond ((p_1 a) ...)
        ...
        ((p_n a) ...))

  Rechte Seite des Zweigs p_i entsprechend der Konstruktionsanleitung für Sorte x_i (für a) ausfüllen.

2.5.9 Prozeduren, die gemischte Daten erzeugen

- Bobs Lieblingskeks ist ein Schokokeks
- Philippas Lieblingskeks ist ein Marmelade-Creme-Keks

```
; erzeuge Bobs oder Philippas Lieblingskeks
(: favorite-cookie (one-of "Bob" "Philippa") cookie)
(define favorite-cookie
  (lambda (person)
    ...))
(check-expect (favorite-cookie "Bob")
              (make-chocolate-cookie 10 10))
(check-expect (favorite-cookie "Philippa")
              (make-jelly-cream-cookie 50 50 10))
```

2.5.10 Lieblingskekse

```
; erzeuge Bobs oder Philippas Lieblingskeks
(: favorite-cookie (one-of "Bob" "Philippa") cookie)
(define favorite-cookie
  (lambda (person)
    (cond
     ((string=? person "Bob")
      (make-chocolate-cookie 10 10)
     ((string=? person "Philippa")
      (make-jelly-cream-cookie 50 50 10))))))
(check-expect (favorite-cookie "Bob")
              (make-chocolate-cookie 10 10))
(check-expect (favorite-cookie "Philippa")
              (make-jelly-cream-cookie 50 50 10))
```

2.6 Zusammenfassung

- Zusammengesetzte Daten
- Records
  - definieren,
  - konsumieren und
  - konstruieren
- Gemischte Daten
  - definieren,
  - konsumieren und
  - konstruieren