INFORMATIK I

Tutorium 5 — 15. November 2024

Aufgepasst!

Name Last
Universität Münster

LaTeX-Vorlage von
Florian Sihler

Übungsblatt 4

1 ○

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: a) `3 * 4 - 2 ^ 2 ^ 4`

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: a) `3 * 4 - 2 ^ 2 ^ 4`

- **Potenz**: Bindungsstärke 8, rechts-assoziativ `3 * 4 - (2 ^ (2 ^ 4))`

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: a) `3 * 4 - 2 ^ 2 ^ 4`

- **Potenz**: Bindungsstärke 8, rechts-assoziativ
- **Multiplikation**: Bindungsstärke 7

```
3 * 4 - (2 ^ (2 ^ 4))
(3 * 4) - (2 ^ (2 ^ 4))
```

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: a) `3 * 4 - 2 ^ 2 ^ 4`

- **Potenz**: Bindungsstärke 8, rechts-assoziativ
- **Multiplikation**: Bindungsstärke 7
- **Subtraktion**: Bindungsstärke 6

```
3 * 4 - (2 ^ (2 ^ 4))
(3 * 4) - (2 ^ (2 ^ 4))
(3 * 4) - (2 ^ (2 ^ 4))
```

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: a) `3 * 4 - 2 ^ 2 ^ 4`

- **Potenz**: Bindungsstärke 8, rechts-assoziativ
- **Multiplikation**: Bindungsstärke 7
- **Subtraktion**: Bindungsstärke 6

```
3 * 4 - (2 ^ (2 ^ 4))
(3 * 4) - (2 ^ (2 ^ 4))
(3 * 4) - (2 ^ (2 ^ 4))
```

$= -65524$

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: b) `add 5 3 + add 7 4 * 2 - 3`

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: b) `add 5 3 + add 7 4 * 2 - 3`

- **Funktion**: Bindungsstärke 10

```
(add 5 3) + (add 7 4) * 2 - 3
```

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: b) `add 5 3 + add 7 4 * 2 - 3`

- **Funktion**: Bindungsstärke 10 `(add 5 3) + (add 7 4) * 2 - 3`
- **Multiplikation**: Bindungsstärke 7 `(add 5 3) + ((add 7 4) * 2) - 3`

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: b) `add 5 3 + add 7 4 * 2 - 3`

- **Funktion**: Bindungsstärke 10 `(add 5 3) + (add 7 4) * 2 - 3`
- **Multiplikation**: Bindungsstärke 7 `(add 5 3) + ((add 7 4) * 2) - 3`
- **Addition/Subtraktion**: Bindungsstärke 6, daher Assoziativität (hier: links)
  `((add 5 3) + ((add 7 4) * 2)) - 3`

Aufgabe 1

Setzen Sie Klammern, um zu zeigen, in welcher Reihenfolge die Teilausdrücke ausgewertet werden, und geben Sie das Ergebnis der Auswertung an.

Lösung 1: b) `add 5 3 + add 7 4 * 2 - 3`

- **Funktion**: Bindungsstärke 10 `(add 5 3) + (add 7 4) * 2 - 3`
- **Multiplikation**: Bindungsstärke 7 `(add 5 3) + ((add 7 4) * 2) - 3`
- **Addition/Subtraktion**: Bindungsstärke 6, daher Assoziativität (hier: links)

`((add 5 3) + ((add 7 4) * 2)) - 3`

$= 27$

Aufgabe 2: a) + b)

Programmieren Sie in Haskell einen zweistelligen, links-assoziativen Infix-Operator ~~ mit der Bindungsstärke 5, der den Mittelwert zweier Float-Zahlen berechnet.

Lösung 2: a) + b)

Aufgabe 2: a) + b)

Programmieren Sie in Haskell einen zweistelligen, links-assoziativen Infix-Operator ~~ mit der Bindungsstärke 5, der den Mittelwert zweier Float-Zahlen berechnet.

Lösung 2: a) + b)

```haskell
-- Bindungsstärke 5
infixl 5 ~~
```

Aufgabe 2: a) + b)

Programmieren Sie in Haskell einen zweistelligen, links-assoziativen Infix-Operator ~~ mit der Bindungsstärke 5, der den Mittelwert zweier Float-Zahlen berechnet.

Lösung 2: a) + b)

```haskell
-- Bindungsstärke 5
infixl 5 ~~

-- Signatur, Operator in Klammern
(~~) :: Float -> Float -> Float
```

Aufgabe 2: a) + b)

Programmieren Sie in Haskell einen zweistelligen, links-assoziativen Infix-Operator ~~ mit der Bindungsstärke 5, der den Mittelwert zweier Float-Zahlen berechnet.

Lösung 2: a) + b)

```haskell
-- Bindungsstärke 5
infixl 5 ~~

-- Signatur, Operator in Klammern
(~~) :: Float -> Float -> Float

-- Mittelwert: Summe, geteilt durch 2
x ~~ y = (x + y)/2
```

Aufgabe 2: c)

Geben Sie an, wie der Ausdruck `3 + 4 ~~ 5 * 6` ausgewertet wird. Welche Änderungen würden sich bei den Bindungsstärken 6 oder 7 ergeben?

Lösung 2: c)

Aufgabe 2: c)

Geben Sie an, wie der Ausdruck `3 + 4 ~~ 5 * 6` ausgewertet wird. Welche Änderungen würden sich bei den Bindungsstärken 6 oder 7 ergeben?

Lösung 2: c)

- Bindungsstärke 5: `* > + > ~~`: `(3 + 4) ~~ (5 * 6)` = 18.5

Aufgabe 2: c)

Geben Sie an, wie der Ausdruck `3 + 4 ~~ 5 * 6` ausgewertet wird. Welche Änderungen würden sich bei den Bindungsstärken 6 oder 7 ergeben?

Lösung 2: c)

- Bindungsstärke 5: `* > + > ~~`: `(3 + 4) ~~ (5 * 6)` = 18.5
- Bindungsstärke 6: `* > +|~~`, also links-assoziativ: `(3 + 4) ~~ (5 * 6)` = 18.5

Aufgabe 2: c)

Geben Sie an, wie der Ausdruck `3 + 4 ~~ 5 * 6` ausgewertet wird. Welche Änderungen würden sich bei den Bindungsstärken 6 oder 7 ergeben?

Lösung 2: c)

- Bindungsstärke 5: $* > + > \sim\sim$: `(3 + 4) ~~ (5 * 6)` $= 18.5$
- Bindungsstärke 6: $* > +|\sim\sim$, also links-assoziativ:
  `(3 + 4) ~~ (5 * 6)` $= 18.5$
- Bindungsstärke 7: $*|\sim\sim > +$, also links-assoziativ:
  `3 + ((4 ~~ 5) * 6)` $= 30$

Aufgabe 3

Betrachten Sie eine zweistellige Funktion `power`, die für natürliche Zahlen b und e den Wert b^e berechnet.

Lösung 3: a) Geben Sie `power` als mathematische Funktion an

Aufgabe 3

Betrachten Sie eine zweistellige Funktion `power`, die für natürliche Zahlen b und e den Wert b^e berechnet.

Lösung 3: a) Geben Sie `power` als mathematische Funktion an

$$power : \mathbb{N} \times \mathbb{N} \to \mathbb{N},\ (b, e) \mapsto b^e$$

Aufgabe 3

Betrachten Sie eine zweistellige Funktion `power`, die für natürliche Zahlen b und e den Wert b^e berechnet.

Lösung 3: b) Programmieren Sie `power` in haskell

Aufgabe 3

Betrachten Sie eine zweistellige Funktion `power`, die für natürliche Zahlen b und e den Wert b^e berechnet.

Lösung 3: b) Programmieren Sie `power` in haskell

```
power :: Integer -> Integer -> Integer
```

Aufgabe 3

Betrachten Sie eine zweistellige Funktion `power`, die für natürliche Zahlen b und e den Wert b^e berechnet.

Lösung 3: b) Programmieren Sie `power` in haskell

```haskell
power :: Integer -> Integer -> Integer
power b e =
    if e >= 0 && b >= 0
    then b ^ e
    else -1
```

Aufgabe 3

Betrachten Sie eine zweistellige Funktion `power`, die für natürliche Zahlen b und e den Wert b^e berechnet.

Lösung 3: b) Programmieren Sie `power` in haskell

```haskell
power :: Integer -> Integer -> Integer
power b e =
    if e >= 0 && b >= 0
    then b ^ e
    else -1
```

- Wegen *Currying* gibt es nur einstellige Funktionen: $\mathbb{D} =$ `Integer`.
- Zielmenge: Funktion $\mathbb{W} = \{$`Integer -> Integer`$\}$

Aufgabe 3

Programmieren Sie die einstellige Funktion `powerOfTwo`, die für eine natürliche Zahl e den Wert 2^e berechnet.

Lösung 3: c)

Aufgabe 3

Programmieren Sie die einstellige Funktion `powerOfTwo`, die für eine natürliche Zahl e den Wert 2^e berechnet.

Lösung 3: c)

```
-- mit Parameter
powerOfTwoA :: Integer -> Integer
powerOfTwoA e = power 2 e
```

Aufgabe 3

Programmieren Sie die einstellige Funktion `powerOfTwo`, die für eine natürliche Zahl e den Wert 2^e berechnet.

Lösung 3: c)

```haskell
-- mit Parameter
powerOfTwoA :: Integer -> Integer
powerOfTwoA e = power 2 e

-- wir nutzen Currying
powerOfTwoB :: Integer -> Integer
powerOfTwoB = power 2
```

Aufgabe 4: Betrachten Sie das Cavete-Szenario.

- Weitergehende Studien haben ergeben, dass die Änderungen an den Anzahlen verkaufter Gerichte besser durch ein kubisches Polynom beschrieben werden können als durch die Konstante `dec_portions`, und zwar durch $f(x) = x^3 + ax$ für eine Preiserhöhung um x Euro. (Positive x-Werte stellen Preiserhöhungen dar; der Funktionswert gibt dann an, wie viele Portionen weniger verkauft werden. Negative x-Werte stellen reduzierte Preise dar, bis hin zu „Bezahlungen" für Bestellungen, und führen zu weiteren Verkäufen.)
- Es kommt ein Einkaufsrabatt von 5% auf den `price_per_portion` zustande, wenn mindestens `150` Gerichte verkauft werden.
- Aufgrund eines beschränkten Lagerraumes für Zutaten kann maximal die Obergrenze von 300 Gerichten verkauft werden.

a) erechnen Sie den Wert der Konstante a in obigem Polynom, so dass $f(1) = 10$ (was den Wert `10` für `dec_portions` verallgemeinert).

b) Ergänzen Sie das Programm der Vorlesung um geeignete Konstanten

Aufgabe 4: Betrachten Sie das Cavete-Szenario.

- Weitergehende Studien haben ergeben, dass die Änderungen an den Anzahlen verkaufter Gerichte besser durch ein kubisches Polynom beschrieben werden können als durch die Konstante `dec_portions`, und zwar durch $f(x) = x^3 + ax$ für eine Preiserhöhung um x Euro. (Positive x-Werte stellen Preiserhöhungen dar; der Funktionswert gibt dann an, wie viele Portionen weniger verkauft werden. Negative x-Werte stellen reduzierte Preise dar, bis hin zu „Bezahlungen" für Bestellungen, und führen zu weiteren Verkäufen.)
- Es kommt ein Einkaufsrabatt von 5% auf den `price_per_portion` zustande, wenn mindestens 150 Gerichte verkauft werden.
- Aufgrund eines beschränkten Lagerraumes für Zutaten kann maximal die Obergrenze von 300 Gerichten verkauft werden.

a) erechnen Sie den Wert der Konstante a in obigem Polynom, so dass $f(1) = 10$ (was den Wert 10 für `dec_portions` verallgemeinert).

b) Ergänzen Sie das Programm der Vorlesung um geeignete Konstanten für obige Angaben.

c) Ändern Sie das Programm der Vorlesung, um obige Sachverhalte abzubilden. Folgen Sie dem Vorgehen der Vorlesung, um notwendige weitere Funktionen einzuführen (von Funktionsköpfen über Beispiele, die typische Fälle zeigen und sich leicht berechnen lassen, zu Funktionsrümpfen und exemplarischen Aufrufen). Beachten Sie zudem nachfolgende Vorgaben.
   - Lassen Sie die Signaturen der Funktionen `costs` und `portions` unverändert. Ändern Sie nur deren Definitionen.
   - Nutzen Sie `if-then-else` in einer der neuen Funktionen, um den Preis pro Portion in

Aufgabe 4: Betrachten Sie das Cavete-Szenario.

- Weitergehende Studien haben ergeben, dass die Änderungen an den Anzahlen verkaufter Gerichte besser durch ein kubisches Polynom beschrieben werden können als durch die Konstante `dec_portions`, und zwar durch $f(x) = x^3 + ax$ für eine Preiserhöhung um x Euro. (Positive x-Werte stellen Preiserhöhungen dar; der Funktionswert gibt dann an, wie viele Portionen weniger verkauft werden. Negative x-Werte stellen reduzierte Preise dar, bis hin zu „Bezahlungen" für Bestellungen, und führen zu weiteren Verkäufen.)
- Es kommt ein Einkaufsrabatt von 5% auf den `price_per_portion` zustande, wenn mindestens 150 Gerichte verkauft werden.
- Aufgrund eines beschränkten Lagerraumes für Zutaten kann maximal die Obergrenze von 300 Gerichten verkauft werden.

a) erechnen Sie den Wert der Konstante a in obigem Polynom, so dass $f(1) = 10$ (was den Wert 10 für `dec_portions` verallgemeinert).

b) Ergänzen Sie das Programm der Vorlesung um geeignete Konstanten für obige Angaben.

c) Ändern Sie das Programm der Vorlesung, um obige Sachverhalte abzubilden. Folgen Sie dem Vorgehen der Vorlesung, um notwendige weitere Funktionen einzuführen (von Funktionsköpfen über Beispiele, die typische Fälle zeigen und sich leicht berechnen lassen, zu Funktionsrümpfen und exemplarischen Aufrufen). Beachten Sie zudem nachfolgende Vorgaben.

- Lassen Sie die Signaturen der Funktionen `costs` und `portions` unverändert. Ändern Sie nur deren Definitionen.
- Nutzen Sie `if-then-else` in einer der neuen Funktionen, um den Preis pro Portion in Abhängigkeit von der Menge (und damit dem möglichen Rabatt) zu berechnen.
- Stellen Sie durch die Verwendung von *Wächtern* in einer der neuen Funktionen sicher, dass weder negative Anzahlen von Gerichten berechnet werden noch mehr als die oben genannte Obergrenze.
- Nutzen Sie lediglich in der Vorlesung vorgestellte Haskell-Konstrukte.

Lösung 4

```
----------------------------------------neu----------------------------------------
-- b)

-- Rabatt im Einkauf, wenn "viele" Gerichte gekauft werden.
discount :: Double
discount = 0.05

-- Schwellwert, ab dem ein Rabatt fuer "viele" Gerichte gewaehrt wird.
discount_threshold :: Int
discount_threshold = 150

-- Anzahl Portionen, die maximal verkauft werden koennen.
max_portions :: Int
max_portions = 300
```

Aufgabe 4: Betrachten Sie das Cavete-Szenario.

- Weitergehende Studien haben ergeben, dass die Änderungen an den Anzahlen verkaufter Gerichte besser durch ein kubisches Polynom beschrieben werden können als durch die Konstante `dec_portions`, und zwar durch $f(x) = x^3 + ax$ für eine Preiserhöhung um x Euro. (Positive x-Werte stellen Preiserhöhungen dar; der Funktionswert gibt dann an, wie viele Portionen weniger verkauft werden. Negative x-Werte stellen reduzierte Preise dar, bis hin zu „Bezahlungen" für Bestellungen, und führen zu weiteren Verkäufen.)
- Es kommt ein Einkaufsrabatt von 5% auf den `price_per_portion` zustande, wenn mindestens 150 Gerichte verkauft werden.
- Aufgrund eines beschränkten Lagerraumes für Zutaten kann maximal die Obergrenze von 300 Gerichten verkauft werden.

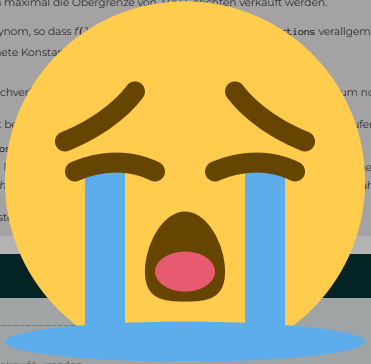
a) erechnen Sie den Wert der Konstante a in obigem Polynom, so dass $f($ `tions` verallgemeinert).

b) Ergänzen Sie das Programm der Vorlesung um geeignete Konsta

c) Ändern Sie das Programm der Vorlesung, um obige Sachver um notwendige weitere Funktionen einzuführen (von Funktionsköpfen

über Beispiele, die typische Fälle zeigen und sich leicht b ufen). Beachten Sie zudem nachfolgende Vorgaben.

- Lassen Sie die Signaturen der Funktionen co
- Nutzen Sie `if-then-else` in einer der neuen F e (und damit dem möglichen Rabatt) zu berechnen.
- Stellen Sie durch die Verwendung von *Wäch* ahlen von Gerichten berechnet werden noch mehr als die oben genannte Obergrenze.
- Nutzen Sie lediglich in der Vorlesung vorgest



Lösung 4

```
----------------------------------------neu----------------
-- b)

-- Rabatt im Einkauf, wenn "viele" Gerichte gekauft werden.
discount :: Double
discount = 0.05

-- Schwellwert, ab dem ein Rabatt fuer "viele" Gerichte gewaehrt wird.
discount_threshold :: Int
discount_threshold = 150

-- Anzahl Portionen, die maximal verkauft werden koennen.
max_portions :: Int
max_portions = 300
```

Aufgabe 4: a)

Die Veränderung der verkauften Portionen bei einer Preiserhöhung um x wird besser durch die Funktion $f(x) = x^3 + ax$ beschrieben.

Lösung 4: a) Berechnen Sie a, so dass $f(1) = 10$.

Aufgabe 4: a)

Die Veränderung der verkauften Portionen bei einer Preiserhöhung um x wird besser durch die Funktion $f(x) = x^3 + ax$ beschrieben.

Lösung 4: a) Berechnen Sie a, so dass $f(1) = 10$.

$$f(x) = x^3 + ax$$

$$f(1) = 1^3 + a \cdot 1$$

$$1 + a = 10 \implies a = 9$$

Aufgabe 4: b)

Es können maximal 300 Gerichte verkauft werden, mit einem Mengenrabatt von 5% ab 150 Gerichten.

Lösung 4: b) Ergänzen Sie das Programm um geeignete Konstanten

Aufgabe 4: b)

Es können maximal 300 Gerichte verkauft werden, mit einem Mengenrabatt von 5% ab 150 Gerichten.

Lösung 4: b) Ergänzen Sie das Programm um geeignete Konstanten

```haskell
-- Mengenrabatt
discount :: Double
discount = 0.05
```

Aufgabe 4: b)

Es können maximal 300 Gerichte verkauft werden, mit einem Mengenrabatt von 5% ab 150 Gerichten.

Lösung 4: b) Ergänzen Sie das Programm um geeignete Konstanten

```haskell
-- Mengenrabatt
discount :: Double
discount = 0.05

-- Schwellwert
discount_threshold :: Int
discount_threshold = 150
```

Aufgabe 4: b)

Es können maximal 300 Gerichte verkauft werden, mit einem Mengenrabatt von 5% ab 150 Gerichten.

Lösung 4: b) Ergänzen Sie das Programm um geeignete Konstanten

```haskell
-- Mengenrabatt
discount :: Double
discount = 0.05

-- Schwellwert
discount_threshold :: Int
discount_threshold = 150

-- Maximalmenge
max_portions :: Int
max_portions = 300
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Wir haben bereits...

```haskell
-- Mengenrabatt
discount :: Double
discount = 0.05

-- Schwellwert
discount_threshold :: Int
discount_threshold = 150

-- Maximalmenge
max_portions :: Int
max_portions = 300
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Einfach übernehmen:

```haskell
-- Einkaufspreis
price_per_portion :: Double
price_per_portion = 2.0

-- Grundkosten
base_costs :: Double
base_costs = 200.0

-- aktueller Verkaufspreis
base_price :: Double
base_price = 9.90

-- Grundanzahl Portionen
base_portions :: Int
base_portions = 100
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Die neue Formel:

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Die neue Formel:

```haskell
-- Verkaufsrückgang pro Euro Preisanstieg
demand_decrease :: Double -> Int
demand_decrease delta = round (delta^3 + 9*delta)
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Die neue Formel:

```haskell
-- Verkaufsrückgang pro Euro Preisanstieg
demand_decrease :: Double -> Int
demand_decrease delta = round (delta^3 + 9*delta)

{-
-- wir entfernen dafür dec_portions
dec_portions :: Int
dec_portions = 10
-}
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Obergrenze für den Verkauf:

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Obergrenze für den Verkauf:

```
-- Beschränkt die Anzahl der Portionen
capped_portions :: Int -> Int
capped_portions portions
    | portions < 0             = 0
    | portions > max_portions = max_portions
    | otherwise                = portions
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Rabatt einberechnen:

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Rabatt einberechnen:

```haskell
-- Berechnet Preis inklusive Rabatt
portion_price :: Int -> Double
portion_price meals =
    if meals >= discount_threshold
    then price_per_portion * (1 - discount)
    else price_per_portion
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) `portions` und `costs`:

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) `portions` und `costs`:

```
-- Verkaufte Portionen für einen Preis
portions :: Double -> Int
portions price = capped_portions (base_portions - ↩
   demand_decrease (price - base_price))
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) `portions` und `costs`:

```
-- Verkaufte Portionen für einen Preis
portions :: Double -> Int
portions price = capped_portions (base_portions - ↩
   demand_decrease (price - base_price))

-- Ausgaben für einen Preis
costs :: Double -> Double
costs price = base_costs + fromIntegral (portions price) * ↩
   portion_price (portions price)
```

Aufgabe 4: c)

Passen Sie das Programm an und führen Sie neue Funktionen mit passenden Beispielen ein.

Lösung 4: c) Der Rest:

```haskell
-- Einnahmen für einen Preis
revenue :: Double -> Double
revenue price = price * fromIntegral (portions price)

-- Gewinn für einen Preis
profit :: Double -> Double
profit price = (revenue price) - (costs price)
```

Anhang

ANHANG

- Es gibt eine LaTeX-Vorlage für Übungsblätter von der AG Vahrenhold:

(hier die `minted`-Version)

Interesse an einer LaTeX-Einführung? :D

Name Last
Münster, 5. Dezember 2024
name.last@uni-muenster.de