Aufgabe 1

a

$p : N \to N$
$p(x) : \left\{ \begin{matrix} \nexists y \in N : y|x, 1<y<x \mapsto 1 \\ sonst \mapsto 0 \end{matrix} \right\}$

Die Zahl 0,1 sind nach Def. keine Primzahlenm Da die Funktion unten nur für $x > 1$ definiert ist, hier die beiden Werte explizit, was mit der Definition der μ-Rekursion erlaubt ist:

$p(0) = 0$
$p(1) = 0$

Wenn der kleinste Teiler von x $\neq 1$ gleich x ist, dann ist x eine Primzahl.

$p(x+1) = f(x, p(x))$
$f(x, y) = if(md(kt(S(x)), x), K_0^0, K_1^0)$

Die μ-Rekursion wird implizit durch die Verwendung des kt benutzt. Das if primitiv rekursiv ist, haben wir bereits im Tutorium gesehen.

b

$\mu s : N \to N$
$\mu s(x, x_1) \mapsto min\{x | s(x, x_1) = 0\}$
$s(1, x_1) = md(x_1, e(K_1 0, 1))$
$s(x+1, x_1) = f(x, s(x, x_1), x_1)$
Wenn eine Zahl $z \dot{-} 10^x = 0$ und ihr Vorgänger $z \dot{-} 10^{x-1} \neq 0$, so hat z x Dezimalstellen:
$f(x, y, z) = md(z, e(K_{10}^0, S(x))$

Ausgabe 2

qu liefert eine Liste aller Quadratzahlen die kleiner sind als n
List comprehension ist toll.... man mus nicht wissen was man tut, man muss es nur hinschreiben ;-)
Bei negativen Zahlen ist kein quadrat keiner... qu gibt die leere Liste zurück

```
> qu :: Int -> [Int]
> qu n
>       | n < 1     = []
>       | otherwise = [ m*m | m <- [ 1 .. n], n > m*m ]
```

Testläufe:

```
Main> qu (-5)
[]
Main> qu (0)
[]
Main> qu (1)
[]
Main> qu (2)
[1]
Main> qu (23)
[1,4,9,16]
Main> qu (1000)
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,
529,576,625,676,729,784,841,900,961]
Main> qu (10000)
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,
529 ,576,625,676,729,784,841,900,961,1024,1089,1156,1225,1296,1369,1444,1521,
1600,1681,1764,1849,1936,2025,2116,2209,2304,2401,2500,2601,2704,2809,2916,
3025,3136,3249,3364,3481,3600,3721,3844,3969,4096,4225,4356,4489,4624,4761,
4900,5041,5184,5329,5476,5625,5776,5929,6084,6241,6400,6561,6724,6889,7056,
7225,7396,7569,7744,7921,8100,8281,8464,8649,8836,9025,9216,9409,9604,9801]
Main>
```

teiler liefert eine Liste aller Teiler von n falls n negativ ist hat sie totzdem positive Teiler, denn für die Teilbarkeit ist nur der Betrag entscheident. Null ist n.D. nur Teiler von sich selbst

```
> teiler :: Int -> [Int]
> teiler n
>          | (n == 0) = [0]
>          | (abs(n) == 1) = [1]
>          | n >  1  = [ m | m <- [ 1 .. n], 0 == n `mod` m ]
```

```
>          | n < -1  = [ m | m <- [ 1 .. (abs n)], 0 == n `mod` m ]

Testläufe:

Main> teiler (-28)
[1,2,4,7,14,28]
Main> teiler (0)
[0]
Main> teiler (25)
[1,5,25]
Main> teiler (23)
[1,23]
Main> teiler (3128)
[1,2,4,8,17,23,34,46,68,92,136,184,391,782,1564,3128]
Main>
```

anf liefert aus einer Liste von Listen eine Liste der Anfangselemente *!!! Achtung !!! alle Listen müssen vom gleichenTyp sein, ansonsten ist diese Funktion in Haskell nicht implementierbar!!!* list comprehension ist wirklich toll.... man mus nicht wissen was man tut, man muss es nur hinschreiben ;-)
Die Liste von Listen nehme ich als List Generator, so das ich jedes Element (sprich jede einzelne Liste am Schopfe herbeiziehen kann, mir das gewünschte (Rosinen;-)Element aus der unterliste mittels head herauspicken kann, die darausentstandene Liste besteht dann genau eus den Anfangselementen. die Prüfung ob die Listenlänge ¿ 0 ist ist nötig um keine Fehler zu erhalte, falls ll eine oder mehrere leere Listen enthält. Das war ein Kommentar der um 937 Zeichen (1398%) Größer war als der Haskell Code den er Kommentiert...

```
> anf :: [[l]] -> [l]
> anf ll = [ head t | t <- ll, length t > 0]

Testläufe:

Main> anf [[1,2..23],[13,15..37],[25,30..100]]
[1,13,25]
Main> anf ["Luft","ist","mehr als","es","schein"]
"Limes"
Main>
```

...war zwar nicht gefragt aber ich konne nicht wiederstehen...
end liefert aus einer Liste von Listen eine Liste der Letzten Elemente *!!! Achtung !!! alle Listen müssen vom gleichenTyp sein, ansonsten ist diese Funktion in Haskell nicht implementierbar!!!*
Kommentar Analog zu dem darüber

```
> end :: [[l]] -> [l]
> end ll = [ last t | t <- ll, length t > 0]
```

Ausgabe 3

zunächst einige nützliche Hilfsfunktionen/Typen:
SFFormat beschreibt wie der Text Formatiert werden soll:

```
('l', 15, '~')
  |   |    |
  |   |    +- Zeichen, mit dem bis Länge Aufgefüllt wird
  |   +------ Gewünschte Länge des Textes
  +---------- Ausrichtung (l-> Links, r-> Rechts, c-> Mitte)
```

```
> type SFFormat = (Char, Int, Char)
```

stringFormat formatiert einen String wie durch ein SFFormat beschrieben, zu lange Strings werden auf die angegebene Länge bschnitten.
der Algorithmus ist nicht kompliziert aber wie alles was Text Layoutet extrem häßlich... der einzige Trick ist die lokale Definition t, die sicherstellt das der eingabetext nie länger als der Ausgabetext wird, und replicate nie negative parameter bekommt. Für die Ästetiche Wirkung bitte keine Kritik ;-)

```
> stringFormat :: SFFormat -> String -> String
```

```
> stringFormat ('l',size,c) ft = t ++ (replicate (s-(length t)) c)
>           where t = if ((s - length(ft)) >= 0) then ft else (take s ft)
>                 s = if (size >= 0) then size else (error "cannot format to neg
> stringFormat ('r',size,c) ft = (replicate (s-(length t)) c) ++ t
>           where t = if ((s - length(ft)) >= 0) then ft else (take s ft)
>                 s = if (size >= 0) then size else (error "cannot format to neg
> stringFormat ('c',size,c) ft = (replicate ((s-(length t)) `div` 2) c)
>                                ++ (replicate ((s-(length t)) `mod` 2) c)
>                                ++ t
>                                ++ (replicate ((s-(length t)) `div` 2) c)
>           where t = if ((s - length(ft)) >= 0) then ft else (take s ft)
>                 s = if (size >= 0) then size else (error "cannot format to neg
> stringFormat _ _ = error "braindead format given - giving up formating"
```

fp2fs kovertiert einen Float f als String mit einer feste Kommastellenzahl s Wenn der Float zu wenig Nachkommastellen hat, wird er mit Nullen aufgefüllt, unnötige Nachkommastellen werden abgeschnitten.

```
> fp2fs :: Int -> Float -> String
> fp2fs k f
>           | k < 0     = error "tried to truncate float where it was sane"
>           | pcd < k   = ((show f) ++ (replicate (k - pcd) '0'))
>           | pcd == k  = (show f)
>           | otherwise = take ((length (show f)) - pcd +k) (show f)
>       where

             pcd ist die ANzahl der Nachkommastellen

>            pcd = (length (show f) - fcp (show f))

             fcp ist die Position des Dezimalpunktes im String s:se

>            fcp :: String -> Int
>            fcp [] = 1
>            fcp (s:se)
>                   | s == '.'  = 1
>                   | otherwise = ((fcp se) +1)
```

Wir brauchen diese nützliche Funktion aber hier nur für 2 Stellen - eine prise curry rundet sie dafür ab....

```
> fp2s = fp2fs 2
```

BillItem: Die Rechnungsposten besteht aus Tripeln (Anzahl, Beschreibung, Preis)
Bill: Eine Rechnung aus einer Liste von Rechnungsposten (btw: keine Anspielung Hr. Gates noch die allseits beliebten Kondome...)

```
> type BillItem = (Int, String, Float)
> type Bill = [BillItem]
```

formatBillItem formatiert ein BillItem und gibt es als String zurück

```
> formatBillItem :: BillItem -> String
> formatBillItem (s, d, p) = (stringFormat ('r',5,' ') (show s)) ++ "x "
>                          ++ (stringFormat ('l',30,'.') d)
>                          ++ (stringFormat ('r',9,'.') ("("++(fp2s p)++")"))
>                          ++ (stringFormat ('r',9,' ') (fp2s (p*(fromInt s))))
```

sumBill berechnet die Summe über die Rechnungsposten

```
> sumBill :: Bill -> Float
```

keine Rechnung $\Rightarrow$ Summe = 0

```
> sumBill [] = 0
```

ansonsten mit ListComprehension über die Liste iterieren und die Summe von allem bilden...

```
> sumBill b = sum [ isum i | i <- b]
>         where
>               isum :: BillItem -> Float
>               isum (s, d, p) = p*(fromInt s)
```

printBill gibt die Rechnung bauf dem Bildschrim aus... Als IO-Funktion darf sie quasi imparativ implementiert sein

```
> printBill :: Bill -> IO ()
> printBill []= error "Money for Nothing is on the DireStraits Album - not here"
> printBill b = do putStr "\nRECHNUNG:\n"
>                  putStr ((replicate 55 '-') ++ "\n") -- linie mit -
```

wir nutzen ListComprehension um durch die BillItems zu iterieren, an den String hängen wir ein
n an und fügen die Liste von Strings mit concat zu einem zusammen

```
>                  putStr (concat [ ((formatBillItem i) ++ "\n") | i <- b])
>                  putStr ((replicate 55 '-') ++ "\n") -- linie mit -
```

Nun Summe, Mehrwertsteuer, Total, wir nehmen die unten berechnete Summe und multiplizieren sie mit den entspr. Faktoren...

```
>                  putStr ((replicate 40 ' ') ++ "SUMME:"
>                         ++ (stringFormat ('r',9,'.') (fp2s bs))++"\n")
>                  putStr ((replicate 40 ' ') ++ "MWST:"
>                         ++ (stringFormat ('r',10,'.') (fp2s (bs*0.16)))++"\n")
>                  putStr ((replicate 40 ' ') ++ "TOTAL:"
>                         ++ (stringFormat ('r',9,'.') (fp2s (bs*1.16)))++"\n")
>             where bs = sumBill b
```

Testmuster Tippen ist zu anstrengend:

```
> testBill :: Bill
> testBill =( ( 23, "Bloody Marry with Coconut Fur and Carpet Split", 13.23)
>           : (  5, "Oliven mit Schlangsahne", 4.07)
>           : (  3, "Massage mit Schlammpackung", 35.73)
>           : (  1, "Pizza Ugly and Oily", 6.10)
>           : (  2, "Schuhe putzen", 2.00)
>           : (  3, "durch den Kakao zehen lassen", 2.00)
>           : ( 13, "Tonicwater", 4.07) : [])
```

dieRechnungBitte - we proudly present die Till'sche Funktion:

```
> dieRechnungBitte :: IO()
> dieRechnungBitte = printBill testBill

Testlauf:
        Hugs session for:
        /usr/share/hugs98/lib/Prelude.hs
        alp1-uez-2002-11-28.lhs
        Main> dieRechnungBitte

        RECHNUNG:
        ------------------------------------------------------
           23x Bloody Marry with Coconut Fur ..(13.23)   304.29
            5x Oliven mit Schlangsahne.........(4.07)    20.35
            3x Massage mit Schlammpackung......(35.73)   107.19
            1x Pizza Ugly and Oily.............(6.10)     6.10
            2x Schuhe putzen...................(2.00)     4.00
            3x durch den Kakao zehen lassen.....(2.00)     6.00
           13x Tonicwater......................(4.07)    52.91
        ------------------------------------------------------
                                                SUMME:...500.84
                                                MWST:.....80.13
                                                TOTAL:...580.97

        Main>
```