{-

Funktionale Programmierung

Übungsblatt (Abgabe: Mo., den 26.10. um 10:10 Uhr)

Author: Boyan Hristov Matrikelnummer: 4980301

Es tut mir leid, dass einige Funktionen oder Kommentare nicht so gut aussehen, aber mein Editors Screen ist viel größer als A4. Code schreibe ich auch immer auf Englisch, deswegen sind die Kommentare ab 3. Aufgabe auch auf Englisch

1. Aufgabe

1. div 1 2 => 0, denn 1/2 = 0.5, aber div wird für ganzzahlige Teilungen verwendet und immer nach unten

gerundet

2. 8^0 => 1, denn x^0 immer 1 ist

3. rem 5 2 => 1 denn rem das Rest nach einer Teilung kalkuliert

4. sqrt 3 => 1.7320508075688772 (Würzel aus drei ist 1.732)

5. "ba"<"aba" => False, weil ‘b’ nach ‘a’ alphabetisch steht

6. 8^200 => 41495155688809929585124639970704894710379428181803401509523685376

(8 hoch 200)

7. 4 /= 5 => True, weil 4 nicht gleich 5 ist

8. mod 5 2 => 1, weil das Rest von 5/2 1 ist

9. sqrt (-1) => NaN (Not a Number), weil kein Würzel von eine negative Zahl gibt

10. True < False => False, weil (ich vermute) True als 1 gelten kann und False als 0. Man kann aber nicht Bool

mit Zahl vergleichen, deswegen... o.O

11. 0.1== 0.3/3 => False, weil 0.1 im Speicher nicht genau vorstellbar ist

12. rem 5 (-2) => 1, weil das positive Rest von 5/(-2) gleich 1 ist

13. exp 1 => e^1 = 2.718281828459045

14. True || False => True, weil bei “ODER” muss nur eine Aussage wahr sein

15. 2\*\*1024 => Infinity – eigentlich ist das 2^1024, aber \*\* ist ^ für nicht ganze Zahle, wobei nicht genug

Speicher gibt (?)

16. abs -7 => Fehler, weil es abs (-7) sein muss. So wäre es 7 sein, weil abs immer eine positive Zahl gibt

17. mod 5 (-2) => -1

18. 'z'<'A' => False, weil ASCII Code von ‘z’ 122 ist und von ‘A‘ 65

19. True && False => False, weil bei “AND“ müssen beide Aussagn wahr sein

2. Aufgabe

1. (-) ((+) ((+) 1 2) 3) (-2) => (-) ((+) 3 3) (-2) => (-) 6 (-2) => 8

2. 2\*\*3 + (2^3) => 8.0 + 8 = 16.0

3. log 1 => 0.0 // e^0.0 = 1

4. div 1 2.0 => Fehler // div kann nur für ganze Zahlen verwendet werden

5. mod 4 (-3) => -2 //das Rest

6. (-4 `mod` 5) == (-4 `rem` 5) => True //-4 == -4

7. (4 `mod` (-5)) == (4 `rem` (-5)) => False // -1 == 4

8. succ 4 \* 8.0 => 5 \* 8.0 = 40.0 //succ gibt das nächste in eine Folge

9. succ (4 \* 8) => succ 32 => 33

10. if (mod 1 2)==0 then "ja" else "nein" => “nein” // weil mod 1 2 => 1

11. True || undefined => True //bei “ODER” muss nur eine Aussage gelten. Wenn

die erste gilt, wird die

12. Zweite gar nicht betrachtet. (lazy evaluation)

13. True && not (True || undefined) => True && not(True) => True && False =>

False

14. True && (undefined || True) => Fehler “Exception: Prelude.undefined”, weil

undefined kein Bool ist, steht erst in “ODER”

15. 3/0 \*\* 2 => 3/0 => Infinity

16. 2 - 0/0 => 2 – NaN => NaN

-}

{-

3. Aufgabe

I am using Double, because money and interest rate are usually not whole numbers and

not Float as Double has bigger capacity => better precision

-}

getTotalCapital :: Double -> Double -> Double -> Double

--Using the given formula, incomeInPercents used for a more readable code

getTotalCapital baseInv anIntRate durInY = baseInv \* incomeInPerc

where incomeInPerc = exp(anIntRateConverted\*durInY)

where anIntRateConverted = if (anIntRate) <= 1

--if the input is more than 1, the user really meant X%

then anIntRate

else anIntRate / 100

{-

4. Aufgabe

In this exercise I am using Doubles and not Floats for more precision

Latitude and Longitude are almost never whole numbers

-}

rad2grad :: Double -> Double

rad2grad rad = rad \* 180 / pi {-one radian is the angle that subtends one radius from given circle

lengthOfCircle = 2\*pi\*radius and radian of 1 radius = 1 => 360 degrees = 2\*pi radians

=> deg/rad = 180/pi => deg = 180\*rad / pi -}

grad2rad :: Double -> Double

grad2rad grad = grad \* pi / 180 -- same as above, from deg/rad = 180/pi => rad = grad\*pi / 180

distanceOnCircle :: Double -> Double -> Double -> Double -> Double

distanceOnCircle x1 y1 x2 y2 = let c = 111.2225685 in c \* (rad2grad longitudeDegree)

-- The given distance constant is per degree, so I convert radians in degrees

where --Haskell works wit radians, so I am converting each given degree in radians

longitudeDegree = acos (sin (grad2rad x1) \* sin (grad2rad x2) +

cos (grad2rad x1) \* cos (grad2rad x2) \* cos (grad2rad (y1 - y2) ))

{-

5. Aufgabe

-}

isBracket :: Char -> Bool

--For the char to be a bracket, it needs to be equal to any of the given bracket symbols

isBracket c = (c == '(') ||

(c == ')') ||

(c == '[') ||

(c == ']') ||

(c == '{') ||

(c == '}')

{-

6. Aufgabe

Colors don't need to be precise or at least not for this example, so I choose Float and not Double

-}

rgb2cmyk :: (Float, Float, Float) -> (Float, Float, Float, Float)

rgb2cmyk (r, g, b) = if (r, g, b) == (0, 0, 0) -- default case

then (0, 0, 0, 1)

else (c, m, y, k) --using where for cleaner code

where

--calculating c, m, y and k using the given formula

w = maximum [r / 255, b / 255, g / 255]

c = (w - (r / 255)) / w

m = (w - (g / 255)) / w

y = (w - (b / 255)) / w

k = 1 - w