

# PI2 WE-Übungsblatt 1

## Aufgabe 1

```

maximum :: [Double] -> Double
maximum [] = error "no maximum of empty list"
maximum [x] = x
maximum (x:y:xs) | x>y      = maximum (x:xs)
                  | otherwise = maximum (y:xs)

rgb2cmyk :: (Int,Int,Int) -> (Double,Double,Double,Double)
rgb2cmyk (0,0,0) = (0.0,0.0,0.0,1.0)
rgb2cmyk (r,g,b) = (c,m,y,k)
                  where
    w = maximum [rd/255.0, gd/255.0, bd/255.0]
    c = (w-(rd/255.0))/w
    m = (w-(gd/255.0))/w
    y = (w-(bd/255.0))/w
    k = 1.0-w
    rd = fromIntegral r
    gd = fromIntegral g
    bd = fromIntegral b

```

## Aufgabe 2

```

minNeighborsDistance :: [Int] -> Int
minNeighborsDistance (x:y:xs) = fst (mnd (abs (x-y) ,xs))
                                where
    mnd :: (Int,[Int]) -> (Int,[Int])
    mnd (m,[])      = (m,[])
    mnd (m,[x])      = (m,[x])
    mnd (m,(x:y:xs)) | dist<m = mnd (dist,(y:xs))
                      | otherwise = mnd (m,(y:xs))
                      where dist = abs (x-y)

minNeighborsDistance _ = error "list needs at least 2 elements"

```

## Aufgabe 3

```

smap :: Int -> Int -> (Int -> Double) -> Double
smap s n func = sum (map func [s..n])

bla :: Int -> Double
bla k = 4.0*(-1.0)^k/(2.0*(fromIntegral k)+1.0)

```

```
piappr :: Int -> Double
piappr n = smap 0 n bla
```

## Aufgabe 4

```
echtTeiler :: Int -> [Int]
echtTeiler n = [ a | a <- [1..n-1], n`mod`a==0 ]
```

## Aufgabe 5

```
mysum :: [Int] -> Int
mysum []      = 0
mysum (x:xs) = x+(mysum xs)

amicable :: Int -> Int -> Bool
amicable n m | n == m      = False
              | otherwise = n == mysum (echtTeiler m) && m == mysum (echtTeiler n)
```

## Aufgabe 6

```
nextCollatz :: Int -> Int
nextCollatz n | n`mod`2 == 0 = n`div`2
              | otherwise   = n*3+1

collatzSeq :: Int -> [Int]
collatzSeq 1 = [1]
collatzSeq n = n:collatzSeq (nextCollatz n)

collatzSeqs :: Int -> [[Int]]
collatzSeqs n = [ collatzSeq a | a <- [1..n] ]
```

## Aufgabe 7

```
square :: (Int, Int, Int) -> Char
square (x,y,s) = if y>x
                  then if x>(-y)+s
                        then 'B'
                        else '.'
                  else if (y<n || y>2*n && y<m) && (x>m) || (x>m+2*n || x>m &&
x<m+n ) && y<m
                        then '*'
                        else ' '
                  where
                    m = s`div`2
                    n = s`div`6
```

```

chessboard :: (Int, Int, Int) -> Char
chessboard (x,y,s) = if (y-1)`mod`8<4 && (x-1)`mod`8<4 || (y-1)`mod`8>=4 &&
(x-1)`mod`8>=4
    then '■-'
    else ' '

circ :: Int -> Int -> Int -> Double
circ x y r = ( ((xd-rd)/rd)^2 + ((yd-rd)/rd)^2 )
    where
        xd = fromIntegral x
        yd = fromIntegral y
        rd = fromIntegral r

easteregg :: (Int, Int, Int) -> Char
easteregg (x,y,s) = if 1.0 > (circ x y m)
    then if x<m
        then if y`mod`6==0 && x`mod`3==0
            then '@'
            else '-'
        else if y`mod`8<4
            then '■-'
            else '='
    else if x>m+m`div`2
        then '||'
        else ' '
    where
        m = s`div`2
        r = (fromIntegral s)/2.0
        xd = fromIntegral x
        yd = fromIntegral y

smiley :: (Int, Int, Int) -> Char
smiley (x,y,s) = if 1.0 > (circ x y m) && 0.8 < (circ x y m)
    then '||'
    else if 1.0 > (circ (x-a) (y-b) r3)
        || 1.0 > (circ (x-a) (y-3*b) r3)
    then '@'
    else if 1.0 > (circ (x-c) (y-d) r4)
        && x>m+m`div`3 && 0.7 < (circ (x-c) (y-d) r4)
    then '='
    else ' '
    where
        m = s`div`2
        a = s`div`4
        b = s`div`5
        c = s`div`8
        d = s`div`7
        r3 = s`div`10
        r4 = m`div`10*9

```