MOISE GABRIEL

FUNCTIONAL PROGRAMMING 2016

QUESTION 4

```
> type Matrix a = [[a]]
```

(a) First, we create a function that checks if two matrices have the same sizes (assuming that the input arguments are valid matrices)

```
> compareSize :: Matrix a -> Matrix a -> Bool
> compareSize [] [] = True
> compareSize [] _ = False
> compareSize _ [] = False
> compareSize (xs:xss) (ys:yss) = (length xs == length ys) && (n == m)
>                                 where n = length xss
>                                       m = length yss
```

(i)

```
> addMat :: Matrix Int -> Matrix Int -> Matrix Int
> addMat xss yss = if compareSize xss yss then [[x+y | (x,y) <- zip xs ys] | (xs,ys) <- zip xss yss]
>                  else error "Sizes not compatible
```

(ii)

```
> addMat' :: Matrix Int -> Matrix Int -> Matrix Int
> addMat' xss yss = if compareSize xss yss then zipWith (zipWith (+)) xss yss
>                   else error "Sizes not compatible
```

(b) (i)

```
> transpose :: Matrix a -> Matrix a
> transpose [] = []
> transpose xss = if (length (head xss) > 0) then [head xs | xs <- xss]: transpose xss'
>                 where xss' = [tail xs | xs <- xss]
```

(ii)

```
> transpose' :: Matrix a -> Matrix a
> transpose' xss = iter xss yss
>                  where yss = replicate (length yss) []

> iter :: Matrix a -> Matrix a -> Matrix a
> iter [] yss = yss
> iter (xs:xss) yss = iter xss (zipWith (\x ys -> ys ++ [x]) xs yss)
```

(c) First, to be able to multiply the matrices, the number of columns of xss must equal the number of rows of yss.

```
> multMat :: Matrix int -> Matrix int -> Matrix int
> multMat [] _ = []
> multMat _ [] = []
> multMat xss yss = if length (head xss) == length yss then [[multLine (xss!! p) (yss'!! q
| q <- [0.. length yss' - 1]] | p <- [0.. length xss - 1]]
>                                                    else error "Sizes not compatible"
>                  where yss' = transpose yss
> multLine :: [int] -> [int] -> Int
> multLine xs ys = sum (zipWith (*) xs ys)
```

(d)

```
> powersMat :: Matrix int -> [Matrix int]
> powersMat m = iterate (multMat m) m
```

This gives as result the list $[m, m^2, m^3, \ldots]$

```
> seriesMat :: Matrix int -> [Matrix int]
> seriesMat m = scanl1 addMat (powersMat m)
```