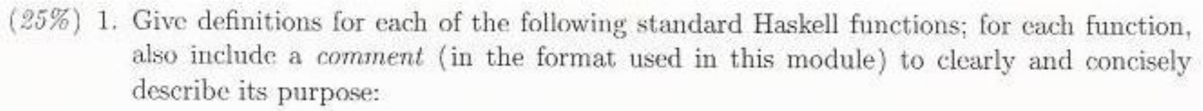
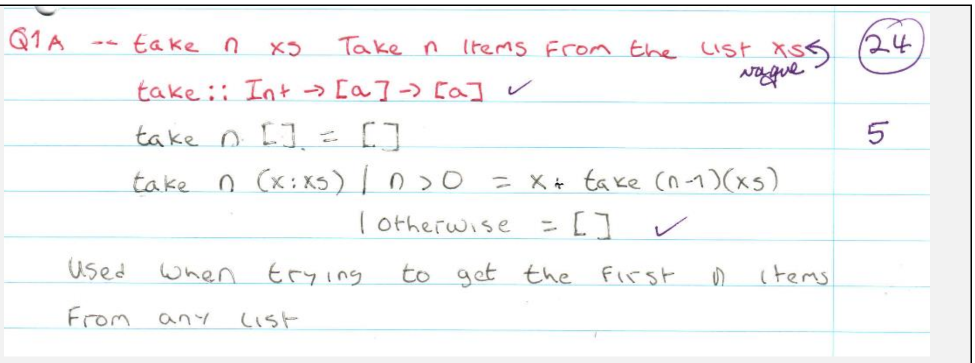
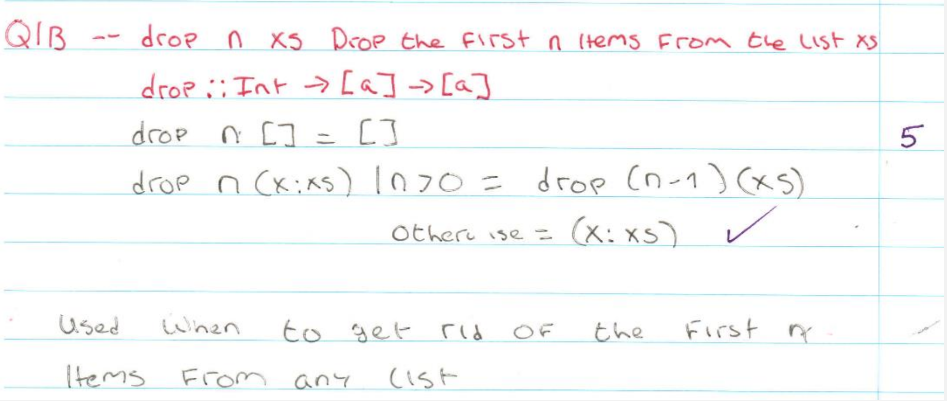
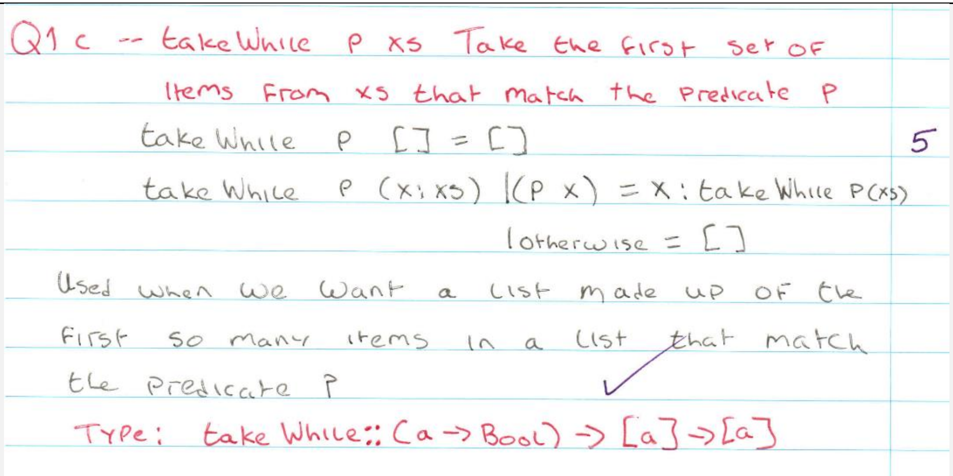
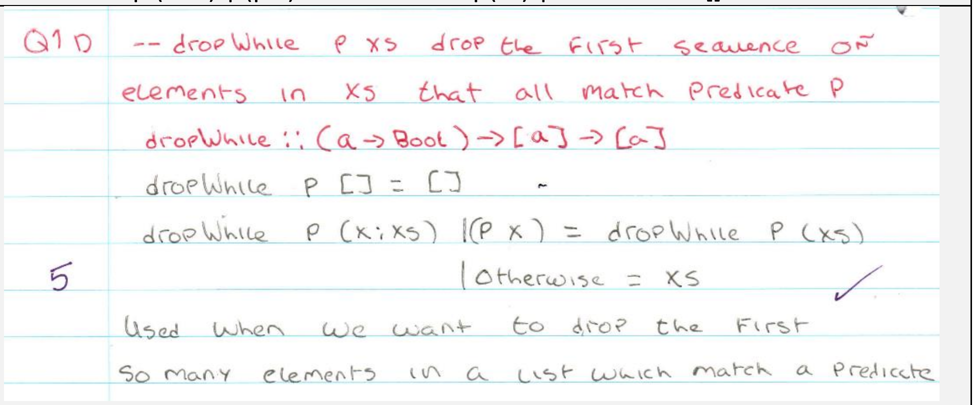
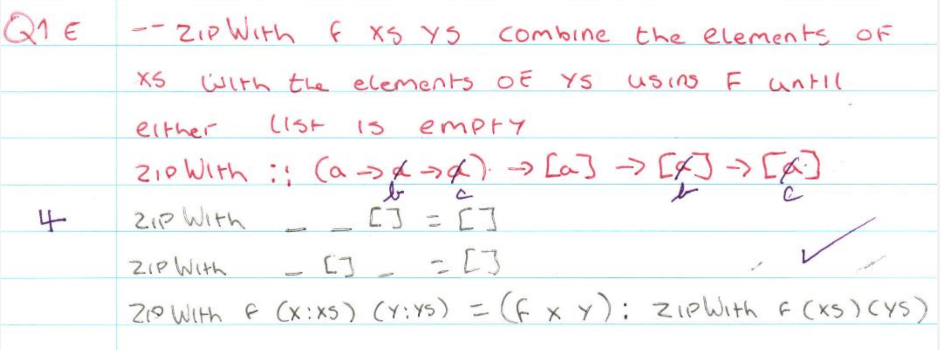
CS4620 – Summer 2014

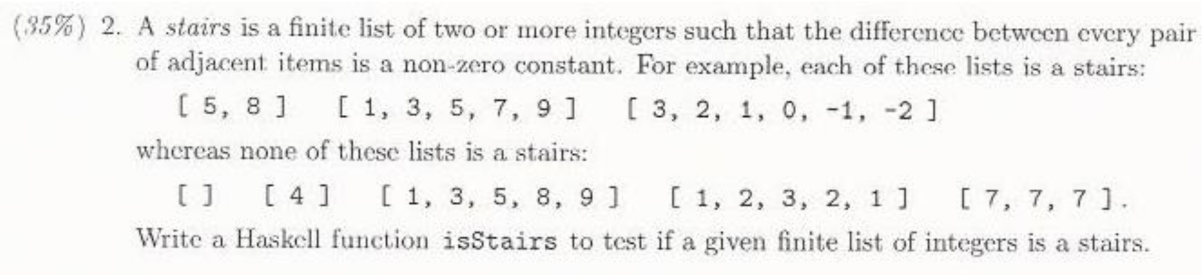


(25%) 1. Give definitions for each of the following standard Haskell functions; for each function, also include a *comment* (in the format used in this module) to clearly and concisely describe its purpose:

a)	take	 Q1A -- take n xs Take n Items From the List xs vague take:: Int -> [a] -> [a] ✓ take n [] = [] take n (x:xs) | n > 0 = x+ take (n-1)(xs) | otherwise = [] ✓ Used when trying to get the First n items From any List 24 5 -- take n xs, take n items from the list xs -- Used when we want to try and get rid of the first n items take n[] = [] take n(x:xs) | n>0 = x take (n-1)(xs) | otherwise = []
b)	drop	 Q1B -- drop n xs Drop the First n Items From the List xs drop :: Int -> [a] -> [a] drop n [] = [] drop n (x:xs) |n >0 = drop (n-1)(xs) otherwise = (x: xs) ✓ Used when to get rid OF the First n Items From any List 5 -- drop n xs drop the first n items form the list xs -- Used when we want to get rid of the first n items from any list drop n[] = [] drop n(x:xs) |n>0 = drop(n-1)(xs) otherwise = (x:xs)

c)	takeWhile	 Q1 c -- takeWhile p xs Take the first set of items from xs that match the Predicate p takeWhile p [] = [] 5 takeWhile p (x:xs) |(p x) = x:takeWhile p(xs) |otherwise = [] Used when we want a list made up of the first so many items in a list that match the predicate p ✓ Type: takeWhile:: (a -> Bool) -> [a] -> [a] -- takeWhile p xs take the first set of items from xs that match the predicate p -- Used when we want a list made up of the first so many items in a list that match the predicate p takeWhile p[]= [] takeWhile p (x:xs) | (p x) = x:takeWhile p(xs) | otherwise = []
d)	dropWhile	 Q1 D -- dropWhile p xs drop the first sequence of elements in xs that all match predicate p dropWhile :: (a -> Bool) -> [a] -> [a] dropWhile p [] = [] dropWhile p (x:xs) |(p x) = dropWhile p (xs) 5 |otherwise = xs ✓ Used when we want to drop the first so many elements in a list which match a predicate -- dropWhile xs drop the first sequence of elements in xs that all match predicate p. -- Used when we want to drop the first so many elememts in a list which match a predicate. dropWhile :: (a -> bool) -> [a] -> [a] dropWhile p [] = [] dropWhile p (x:xs) | (p x) = dropWhile p (xs) |otherwise = xs
e)	zipWith	 Q1 E -- zipWith f xs ys combine the elements of xs with the elements of ys using f until either list is empty zipWith :: (a -> ~~a~~ b -> ~~a~~ c) -> [a] -> [~~a~~ b] -> [~~a~~ c] 4 zipWith _ _ [] = [] zipWith _ [] _ = [] ✓ zipWith f (x:xs) (y:ys) = (f x y) : zipWith f (xs)(ys)

		-- zipWith f x sys combine the elements of xs with the elements of ys using f until either list is empty. zipWith :: (a -> b -> c) -> [a] -> [b] -> [c] zipWith _ _ [] = [] zipWith _ [] _ = [] zipWith f (x:xs) (y:ys_ = (f x y) : zipWith f (xs) (ys)



(35%) 2. A *stairs* is a finite list of two or more integers such that the difference between every pair of adjacent items is a non-zero constant. For example, each of these lists is a stairs:

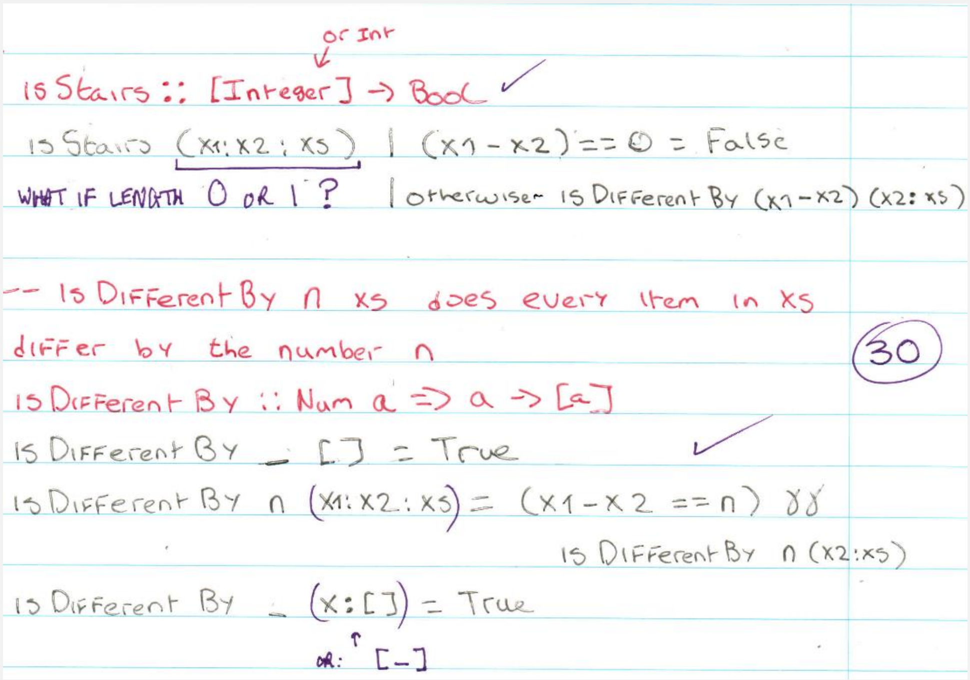
```
[ 5, 8 ]   [ 1, 3, 5, 7, 9 ]   [ 3, 2, 1, 0, -1, -2 ]
```

whereas none of these lists is a stairs:

```
[ ]   [ 4 ]   [ 1, 3, 5, 8, 9 ]   [ 1, 2, 3, 2, 1 ]   [ 7, 7, 7 ].
```

Write a Haskell function `isStairs` to test if a given finite list of integers is a stairs.

Answer ???

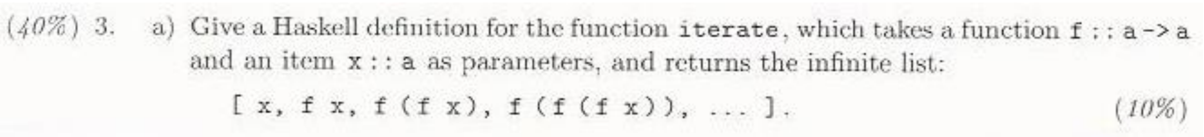


```
                      or Int
                        ↓
isStairs :: [Integer] -> Bool ✓
isStairs (x1:x2:xs) | (x1-x2)==0 = False
WHAT IF LENGTH 0 OR 1 ?  | otherwise isDifferentBy (x1-x2) (x2:xs)

-- isDifferentBy n xs does every item in xs
differ by the number n                                   30
isDifferentBy :: Num a => a -> [a]
isDifferentBy _ [] = True                       ✓
isDifferentBy n (x1:x2:xs) = (x1-x2 ==n) &&
                                  isDifferentBy n (x2:xs)
isDifferentBy _ (x:[]) = True
                    ↑
                OR: [_]
```

isStairs :: [Integer] -> Bool
isStairs (x1:x2:x3) | (x1 – x2) == 0 = False | otherwise isDifferentBy (x1-x2)(x2:x3)

-- isDifferentBy n xs does every item in xs differ by the number n
isDifferentBy _ [] = True
isDifferentBy n(x1:x2:x3) && isDifferentBy n(x2:x3)
isDifferentBy _(x:[_]) = True

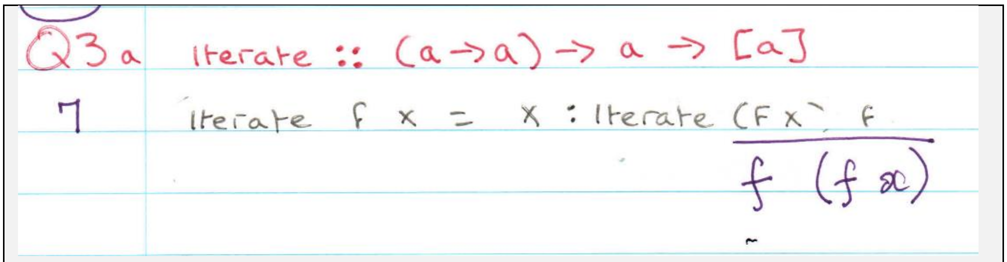


(40%) 3. a) Give a Haskell definition for the function `iterate`, which takes a function `f :: a -> a` and an item `x :: a` as parameters, and returns the infinite list:

```
[ x, f x, f (f x), f (f (f x)), ... ].
```

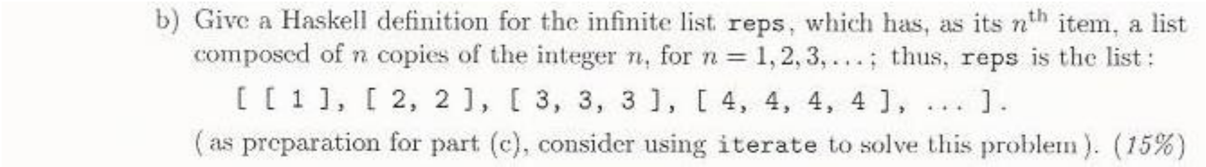
(10%)

Answer



```
Q3 a   iterate :: (a -> a) -> a -> [a]
7      iterate f x = x : iterate (F x` F
                                  f (f x)
```

```
iterate :: (a -> a) -> a -> [a]
iterate f x = x : iterate f (f x)
```

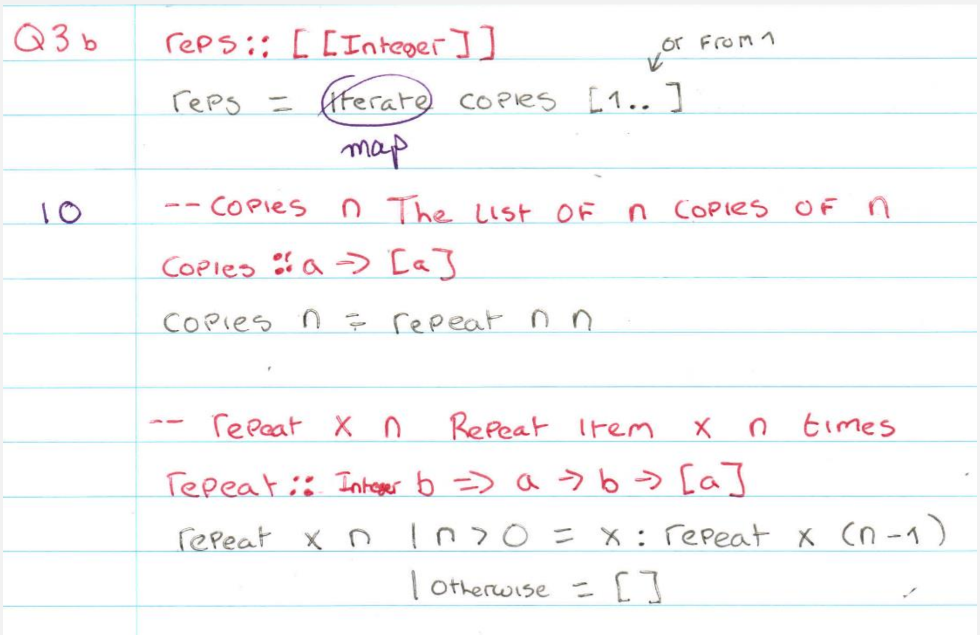


b) Give a Haskell definition for the infinite list `reps`, which has, as its n^{th} item, a list composed of n copies of the integer n, for $n = 1, 2, 3, \ldots$; thus, `reps` is the list:

`[ [ 1 ], [ 2, 2 ], [ 3, 3, 3 ], [ 4, 4, 4, 4 ], ... ].`

(as preparation for part (c), consider using `iterate` to solve this problem). *(15%)*

Answer



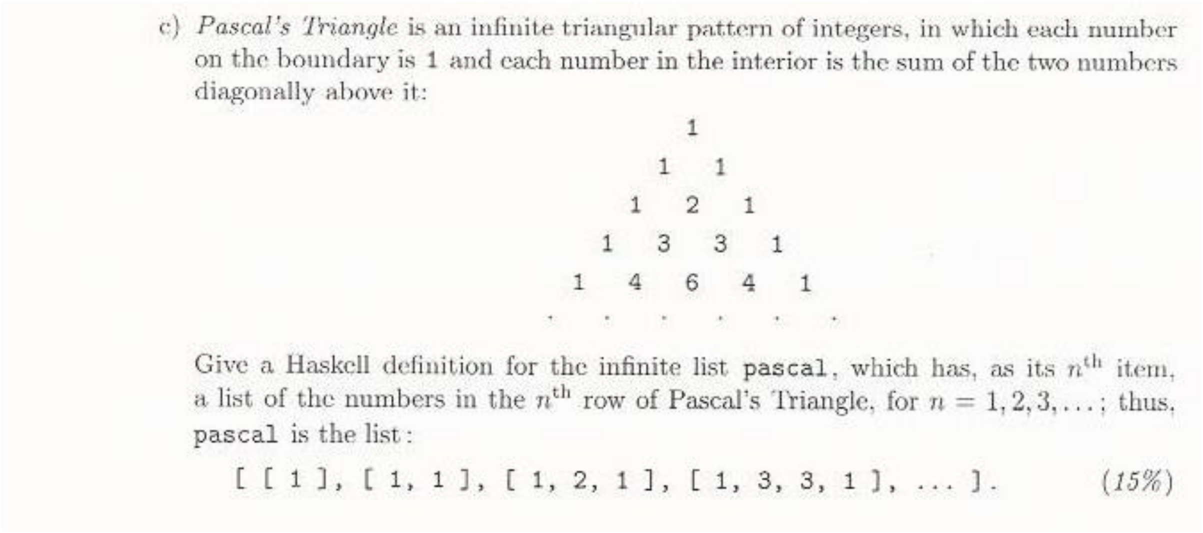
```
Q3 b   reps :: [[Integer]]            or from 1
       reps = ~~iterate~~ copies [1..]
              map
10     -- copies n The list of n copies of n
       copies :: a -> [a]
       copies n = repeat n n

       -- repeat x n  Repeat item x n times
       repeat :: Integer b => a -> b -> [a]
       repeat x n | n > 0 = x : repeat x (n-1)
                  | otherwise = []
```

```
reps :: [[Integer[]
reps = map copies [1..]

-- copies n, the list of n copies of n
copies  :: a -> [a]
copies n = repeat n n

-- repeat x n, repeat item x n times
repeat  :: Integer b => a -> b -> [a]
repeat x n | n>0 = x:repeat x(n-1) | otherwise = []
```



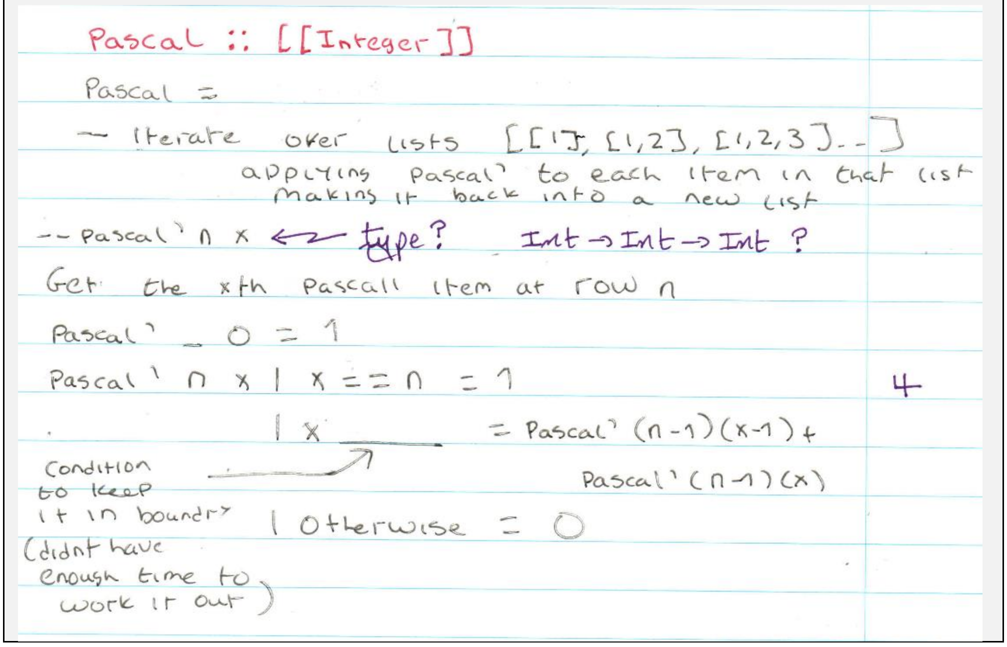
c) *Pascal's Triangle* is an infinite triangular pattern of integers, in which each number on the boundary is 1 and each number in the interior is the sum of the two numbers diagonally above it:

```
            1
         1     1
      1     2     1
   1     3     3     1
1     4     6     4     1
.     .     .     .     .     .
```

Give a Haskell definition for the infinite list `pascal`, which has, as its n^{th} item, a list of the numbers in the n^{th} row of Pascal's Triangle, for $n = 1, 2, 3, \ldots$; thus, `pascal` is the list:

`[ [ 1 ], [ 1, 1 ], [ 1, 2, 1 ], [ 1, 3, 3, 1 ], ... ].` *(15%)*

Answer 4 Marks ???



```
Pascal :: [[Integer]]
Pascal =
-- iterate over lists [[1], [1,2], [1,2,3]..]
        applying pascal' to each item in that list
        making it back into a new list
-- pascal' n x <= type?   Int -> Int -> Int ?
Get the xth pascall item at row n
Pascal' _ 0 = 1
Pascal' n x | x == n = 1
            | x ______ = Pascal' (n-1)(x-1) +
                         Pascal' (n-1)(x)
            | Otherwise = 0
```

Condition to keep it in boundry

(didnt have enough time to work it out)

4