20CYS312 - PPL - Lab Exercise 5

Roll Number: CH.EN.U4CYS22034

Name: Naren S

Github_Link: https://github.com/narenss/22034_20CYS312

Program 1: Simple Pattern Matching with Integers Function Code:

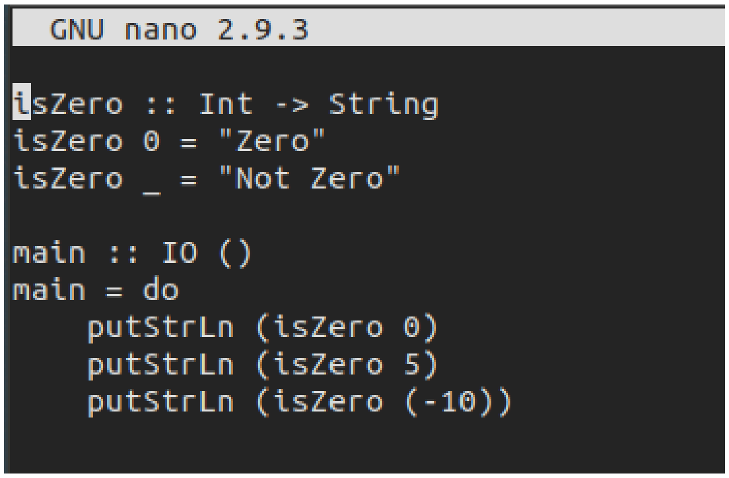
```
isZero :: Int -> String
isZero 0 = "Zero"
isZero _ = "Not Zero"
```

Explanation :

The isZero function checks if the input integer is zero. If it is, the function returns "Zero"; otherwise, it returns "Not Zero".

Input/Output Example:

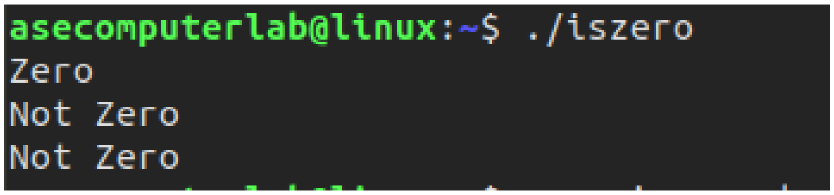
- Input: isZero 0
- Output: "Zero"
- Input: isZero 5
- Output: "Not Zero"



```
  GNU nano 2.9.3

isZero :: Int -> String
isZero 0 = "Zero"
isZero _ = "Not Zero"

main :: IO ()
main = do
    putStrLn (isZero 0)
    putStrLn (isZero 5)
    putStrLn (isZero (-10))
```



```
asecomputerlab@linux:~$ ./iszero
Zero
Not Zero
Not Zero
```

Program 2: Pattern Matching on Lists Function Code:

countElements :: [a] -> Int
countElements [] = 0

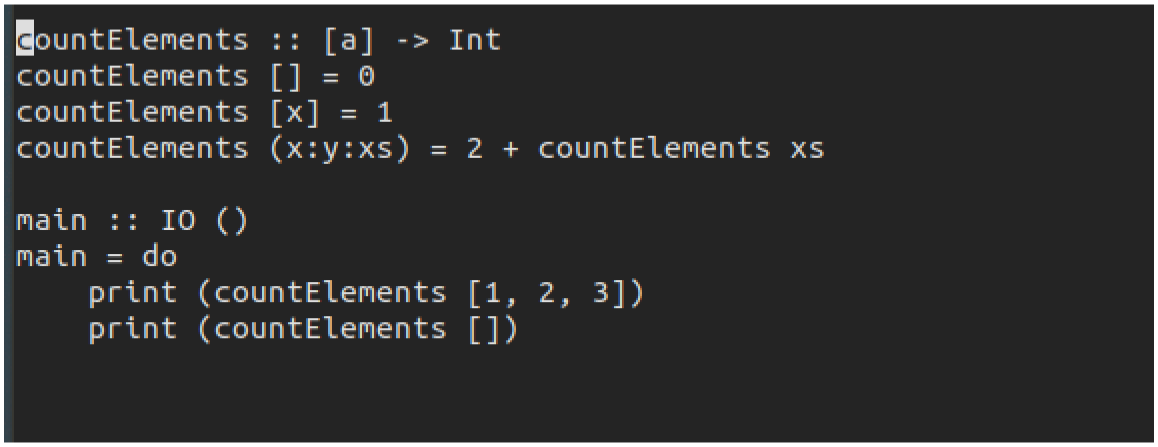
countElements [x] = 1
countElements (x:y:xs) = 2 + countElements xs

Explanation:

The countElements function recursively counts the number of elements in a list using pattern matching. An empty list returns 0, and a non-empty list increments the count by 1.

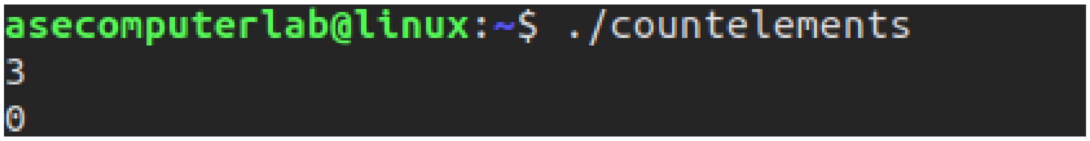
Input/Output Example:

- Input: countElements [1, 2, 3]
- Output: 3
- Input: countElements []
- Output: 0



```haskell
countElements :: [a] -> Int
countElements [] = 0
countElements [x] = 1
countElements (x:y:xs) = 2 + countElements xs

main :: IO ()
main = do
    print (countElements [1, 2, 3])
    print (countElements [])
```



```
asecomputerlab@linux:~$ ./countelements
3
0
```

Program 3: Pattern Matching with Tuples Function Code:

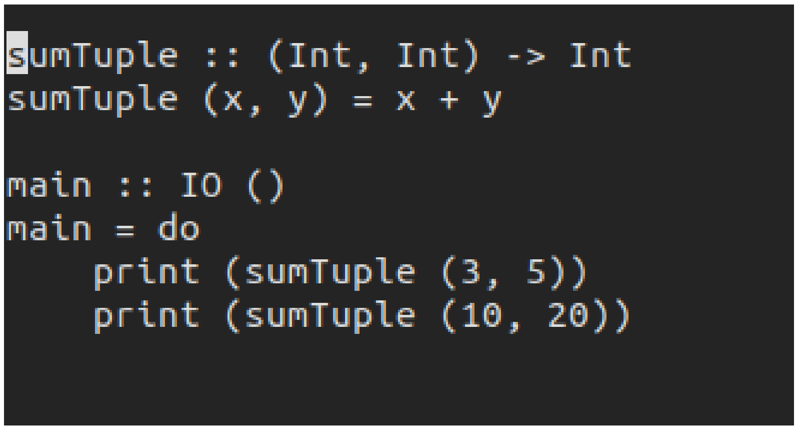
sumTuple :: (Int, Int) -> Int
sumTuple (x, y) = x + y

Explanation

The sumTuple function takes a tuple of two integers and returns their sum by pattern matching on the tuple elements.

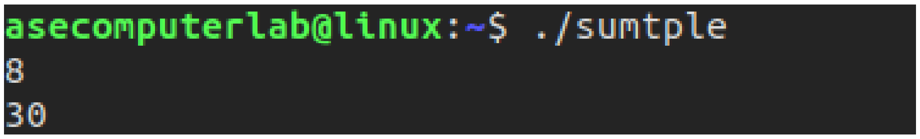
Input/Output Example:

- Input: sumTuple (3, 5)
- Output: 8
- Input: sumTuple (10, 20)
- Output: 30



```haskell
sumTuple :: (Int, Int) -> Int
sumTuple (x, y) = x + y

main :: IO ()
main = do
    print (sumTuple (3, 5))
    print (sumTuple (10, 20))
```



```
asecomputerlab@linux:~$ ./sumtple
8
30
```

<u>Program 4: Pattern Matching on a Custom Data Type Function Code:</u>

data Color = Red | Green | Blue

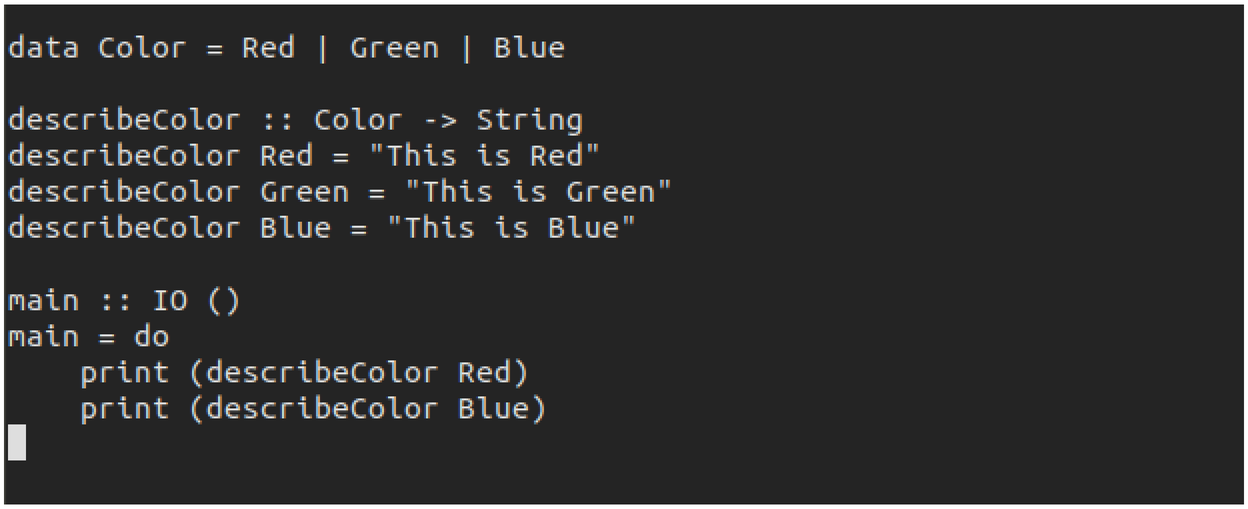
describeColor :: Color -> String
describeColor Red = "This is Red"
describeColor Green = "This is Green"
describeColor Blue = "This is Blue"

Explanation:

The describeColor function matches against the Color data type and returns a string describing the color.

Input/Output Example:

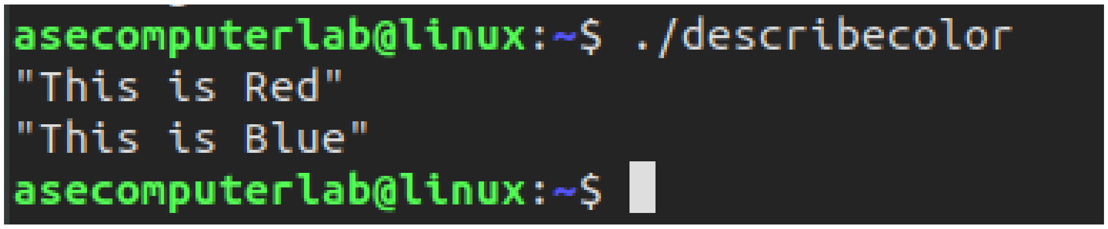
- Input: describeColor Red
- Output: "This is Red"
- Input: describeColor Blue
- Output: "This is Blue"



```
data Color = Red | Green | Blue

describeColor :: Color -> String
describeColor Red = "This is Red"
describeColor Green = "This is Green"
describeColor Blue = "This is Blue"

main :: IO ()
main = do
    print (describeColor Red)
    print (describeColor Blue)
```



```
asecomputerlab@linux:~$ ./describecolor
"This is Red"
"This is Blue"
asecomputerlab@linux:~$
```

Program 5: Pattern Matching with Lists (Head and Tail) Function Code:

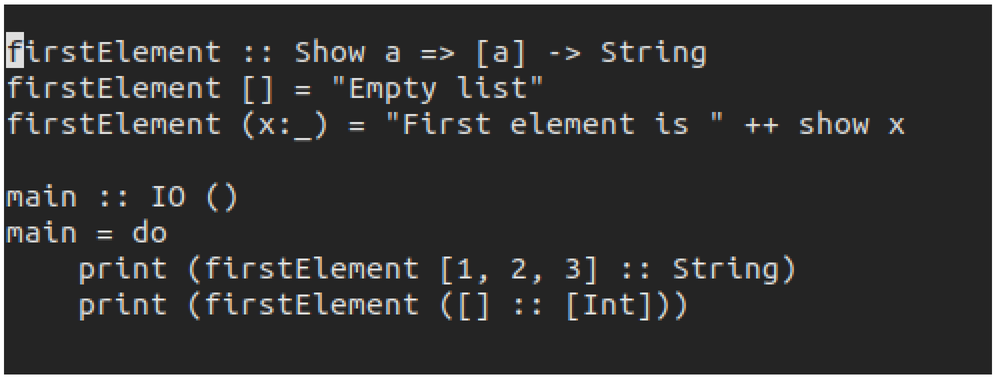
```
firstElement :: [a] -> String
firstElement [] = "Empty list"
firstElement (x:_) = "First element is " ++ show x
```

Explanation:

The firstElement function checks if the list is empty. If not, it returns the first element.

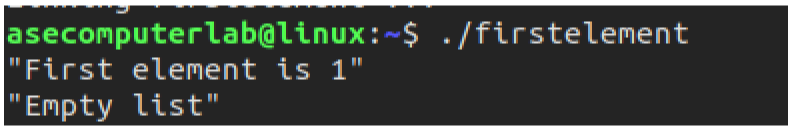
Input/Output Example:

- Input: firstElement [1, 2, 3]
- Output: "First element is 1"
- Input: firstElement []
- Output: "Empty list"



```
firstElement :: Show a => [a] -> String
firstElement [] = "Empty list"
firstElement (x:_) = "First element is " ++ show x

main :: IO ()
main = do
    print (firstElement [1, 2, 3] :: String)
    print (firstElement ([] :: [Int]))
```



```
asecomputerlab@linux:~$ ./firstelement
"First element is 1"
"Empty list"
```

Program 6: Pattern Matching with Simple List Processing Function Code:

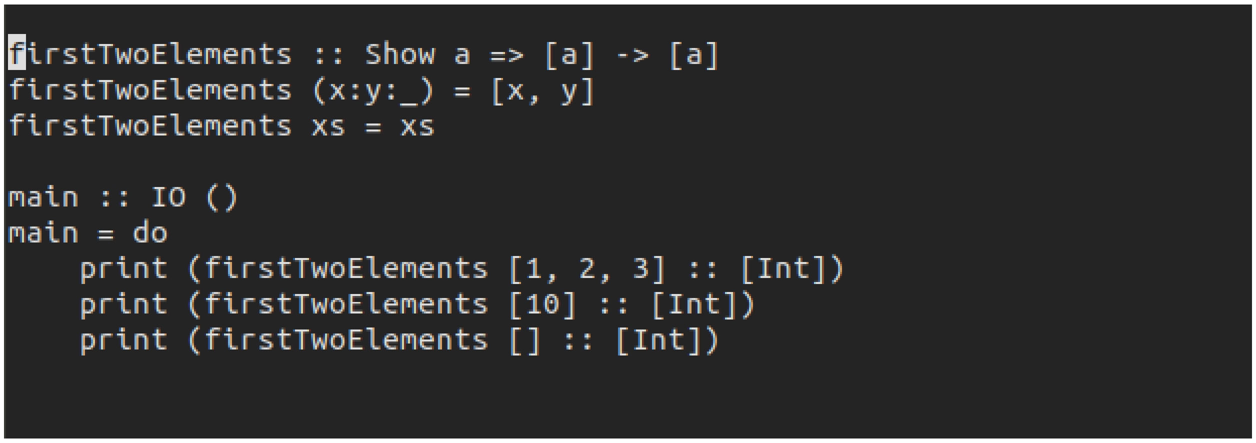
```
firstTwoElements :: [a] -> [a]
firstTwoElements [] = []
firstTwoElements [x] = [x]
firstTwoElements (x:y:_) = [x, y]
```

Explanation:

The firstTwoElements function returns the first two elements of the list or the entire list if it has fewer than two elements.

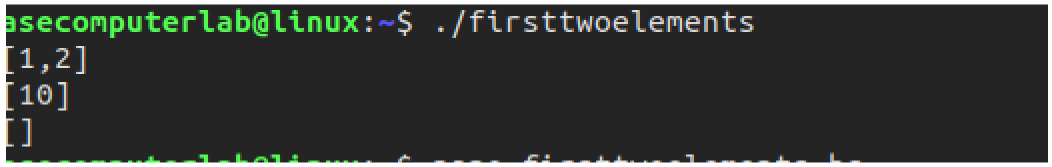
Input/Output Example:

- Input: firstTwoElements [1, 2, 3]
- Output: [1, 2]
- Input: firstTwoElements [10]
- Output: [10]



```
firstTwoElements :: Show a => [a] -> [a]
firstTwoElements (x:y:_) = [x, y]
firstTwoElements xs = xs

main :: IO ()
main = do
    print (firstTwoElements [1, 2, 3] :: [Int])
    print (firstTwoElements [10] :: [Int])
    print (firstTwoElements [] :: [Int])
```



```
asecomputerlab@linux:~$ ./firsttwoelements
[1,2]
[10]
[]
```

Program 7: Pattern Matching with Multiple Cases Function Code:

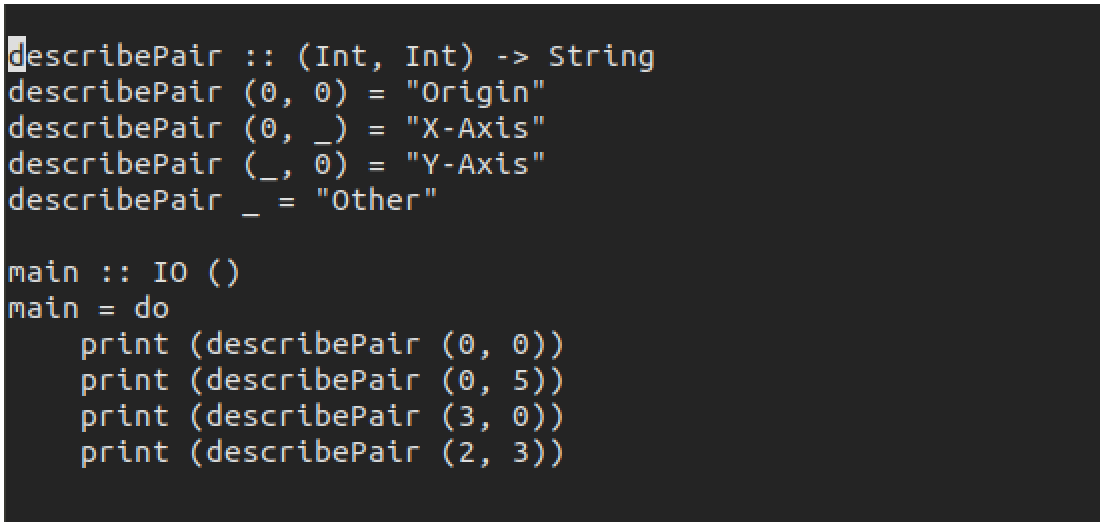
```
describePair :: (Int, Int) -> String
describePair (0, 0) = "Origin"
describePair (0, _) = "X-Axis"
describePair (_, 0) = "Y-Axis"
describePair _ = "Other"
```

Explanation :

The describePair function matches against a tuple of integers to classify its location.

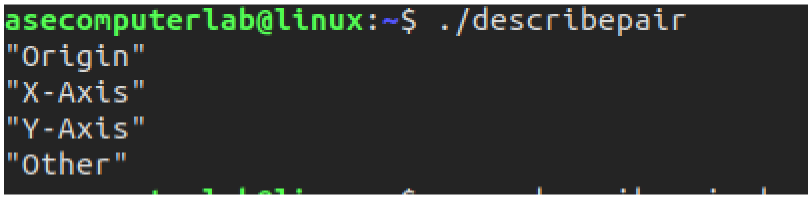
Input/Output Example:

- Input: describePair (0, 0)
- Output: "Origin"
- Input: describePair (0, 5)
- Output: "X-Axis"



```
describePair :: (Int, Int) -> String
describePair (0, 0) = "Origin"
describePair (0, _) = "X-Axis"
describePair (_, 0) = "Y-Axis"
describePair _ = "Other"

main :: IO ()
main = do
    print (describePair (0, 0))
    print (describePair (0, 5))
    print (describePair (3, 0))
    print (describePair (2, 3))
```



```
asecomputerlab@linux:~$ ./describepair
"Origin"
"X-Axis"
"Y-Axis"
"Other"
```

<u>Program 8: Pattern Matching for List Recursion Function Code:</u>

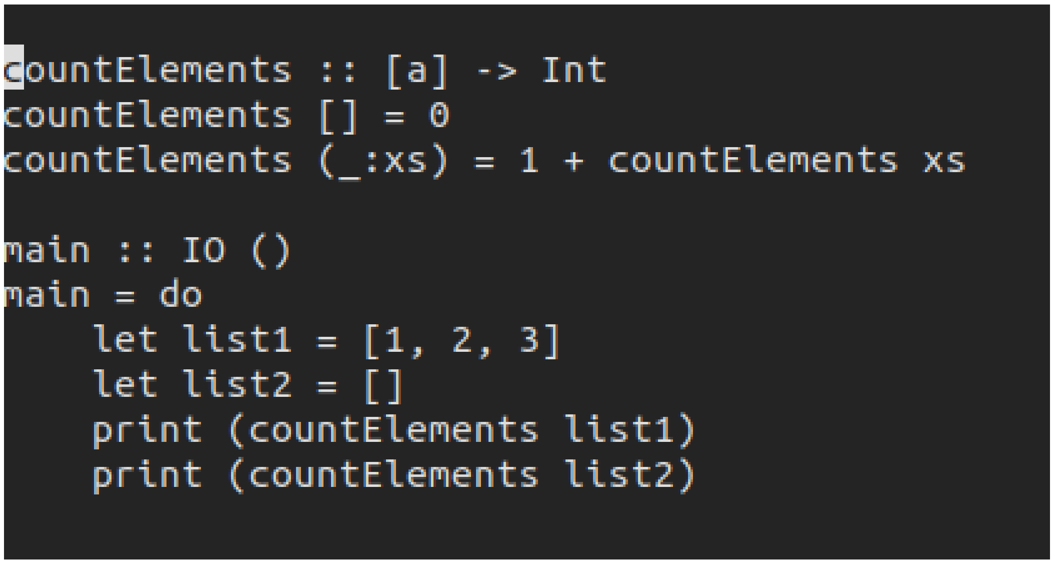
```
listLength :: [a] -> Int
listLength [] = 0
listLength (_:xs) = 1 + listLength xs
```

Explanation:

The listLength function calculates the length of a list recursively.

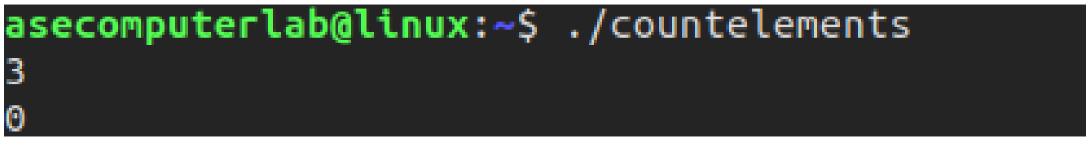
Input/Output Example:

- Input: listLength [1, 2, 3]
- Output: 3
- Input: listLength []



```haskell
countElements :: [a] -> Int
countElements [] = 0
countElements (_:xs) = 1 + countElements xs

main :: IO ()
main = do
    let list1 = [1, 2, 3]
    let list2 = []
    print (countElements list1)
    print (countElements list2)
```



```
asecomputerlab@linux:~$ ./countelements
3
0
```