**20CYS312 - PPL - Lab Exercise 3**

**Name:Prasanna N R**  
**Reg.No:CH.EN.U4CYS22039**

**Objective:**  
The goal is to write functions to perform arithmetic operations, list manipulations, basic function definitions, and function composition.

**1. Basic Data Types**

a. Sum of Two Integers

**Task**: Define a function sumIntegers that takes two Int values and returns their sum.

**Solution**:

sumIntegers :: Int -> Int -> Int

sumIntegers x y = x + y

main :: IO ()

main = do

print (sumIntegers 3 5)

Explanation: The function sumIntegers takes two integers as input (x and y) and returns their sum.

Output:  


b. Check if a Number is Even or Odd

**Task**: Write a function isEven that takes an Int and returns a Boolean value indicating whether the number is even.

**Solution**:

isEven :: Int -> Bool

isEven x = x `mod` 2 == 0

main :: IO ()

main = do

print (isEven 4)

print (isEven 7)

* **Explanation**: The function isEven checks if the remainder when dividing x by 2 is zero. If it is, the number is even and the function returns True; otherwise, it returns False.

Output:  


c. Absolute Value

**Task**: Define a function absolute that takes a Float and returns its absolute value.

**Solution**:

absolute :: Float -> Float

absolute x

| x < 0 = -x

| otherwise = x

main :: IO ()

main = do

print (absolute (-5.6))

print (absolute 3.2)

* **Explanation**: The function absolute checks if the given number is negative. If so, it returns its negation (making it positive), otherwise, it returns the number itself.

Output:  


### **2. List Operations**

**a. Sum of All Elements in a List**

**Task**: Define a function sumList that takes a list of integers and returns the sum of all the elements in the list.

**Solution**:

sumList :: [Int] -> Int

sumList = foldr (+) 0

main :: IO ()

main = do

print (sumList [1, 2, 3])

* **Explanation**: The function sumList uses foldr to recursively apply the addition operation to all elements in the list, starting with an initial value of 0.

Output:  


**b. Filter Even Numbers**

**Task**: Write a function filterEven that takes a list of integers and returns a list containing only the even numbers.

**Solution**:

filterEven :: [Int] -> [Int]

filterEven xs = filter even xs

main :: IO ()

main = do

print (filterEven [1, 2, 3, 4])

* **Explanation**: The function filterEven uses the built-in filter function, which applies the even function to each element of the list and keeps only those that return True.

Output:



**c. Reverse a List**

**Task**: Define a function reverseList that takes a list and returns a new list with the elements in reverse order.

**Solution**:

reverseList :: [a] -> [a]

reverseList = reverse

main :: IO ()

main = do

print (reverseList [1, 2, 3])

* **Explanation**: The function reverseList uses the built-in reverse function, which reverses the order of elements in the list.

Output:



**3. Basic Functions**

**a. Increment Each Element**

**Task**: Define a function incrementEach that takes a list of integers and returns a new list where each element is incremented by 1.

**Solution**:

incrementEach :: [Int] -> [Int]

incrementEach = map (+1)

main :: IO ()

main = do

print (incrementEach [1, 2, 3])

* **Explanation**: The function incrementEach uses the map function, which applies the given function ((+1)) to each element in the list, producing a new list.

Output:



**b. Square a Number**

**Task**: Write a function square that takes an integer and returns its square.

**Solution**:

square :: Int -> Int

square x = x \* x

main :: IO ()

main = do

print (square 4)

* **Explanation**: The function square multiplies the integer x by itself to return the square.

Output:



**4. Function Composition**

**a. Compose Functions to Add and Multiply**

**Task**: Write a function addThenMultiply that first adds two integers and then multiplies the result by another integer. Use function composition to define this.

**Solution**:

addThenMultiply :: Int -> Int -> Int -> Int

addThenMultiply x y z = (z \*) . (+ x) $ y

main :: IO ()

main = do

print (addThenMultiply 2 3 4)

* **Explanation**: The function addThenMultiply uses function composition (.). It first adds x to y, and then multiplies the result by z.

Output:



**b. Apply Multiple Transformations to a List**

**Task**: Define a function transformList that takes a list of integers and first squares each element, then adds 10 to each squared element. Use function composition to implement this.

**Solution**:

square :: Int -> Int

square x = x \* x

addTen :: Int -> Int

addTen x = x + 10

transformList :: [Int] -> [Int]

transformList = map (addTen . square)

main :: IO ()

main = do

let numbers = [1, 2, 3, 4]

print (transformList numbers)

* **Explanation**: The function transformList first squares each element using the square function and then adds 10 to each squared element. This is done using function composition.

Output:  


**Conclusion:**

This lab exercise helped understand fundamental concepts of Haskell, including basic data types, list operations, simple functions, and function composition.