**Numbers**

main :: IO ()

main = do

putStrLn "Printing Different Data Types:"

-- Int

let intVal = 42 :: Int

putStrLn $ "Int: " ++ show intVal

-- Double

let doubleVal = 3.14 :: Double

putStrLn $ "Double: " ++ show doubleVal

-- Char

let charVal = 'A' :: Char

putStrLn $ "Char: " ++ show charVal

-- String

let stringVal = "Hello, Haskell!" :: String

putStrLn $ "String: " ++ show stringVal

-- Bool

let boolVal = True :: Bool

putStrLn $ "Bool: " ++ show boolVal

-- List

let listVal = [1, 2, 3, 4] :: [Int]

putStrLn $ "List of Int: " ++ show listVal

-- Tuple

let tupleVal = (5, "five") :: (Int, String)

putStrLn $ "Tuple (Int, String): " ++ show tupleVal

1. Print the last digit of a number.

lastd :: Int -> Int

lastd n = n `mod` 10

main :: IO()

main = do

putStrLn "Enter a number: "

num <- getLine

let n = read num::Int

putStrLn ("The last digit of the number is "++ show (lastd n))

**Lists**

1. Create a list and print the last element in the list.

lastelem :: [Int] -> Int

lastelem [x] = x

lastelem (\_:xs) = lastelem xs

main :: IO ()

main = do

putStrLn "Please enter the list elements: "

li <- getLine

let arr = map read ( words li ) :: [Int]

let lastel = lastelem arr

putStrLn "Last element of the list is : "

print lastel

1. Create a list of strings and reverse the list.

rev :: [String] -> [String]

rev (x:xs) = do

if null xs

then x:xs

else

rev xs ++ [x]

main :: IO()

main = do

putStrLn "Enter the list elements : "

a <- getLine

let arr = ( words a ) :: [String]

let b = rev arr

putStrLn ("Reversed list : " ++ show b )

1. Write a function to concatenate two lists

concatt :: [String] -> [String] -> [String]

concatt (x:xs) (y:ys) = (x:xs) ++ (y:ys)

main :: IO()

main = do

putStrLn "Enter the 1st list elements : "

a <- getLine

let arr1 = ( words a ) :: [String]

putStrLn "Enter the 2nd list elements : "

b <- getLine

let arr2 = ( words b ) :: [String]

let c = concatt arr1 arr2

putStrLn ("The concatenated list is : " ++ show c)

1. Write a function to check if an element is present in the list or not.

isElement :: Int -> [Int] -> Bool

isElement a (x:xs) = do

if null xs

then False

else do

if (a == x)

then True

else isElement a xs

main :: IO()

main = do

putStrLn "Please enter the list elements: "

l <- getLine

let arr = map read ( words l ) :: [Int]

putStrLn "Please enter a number to check if ti is an element of the list or not: "

a <- getLine

if isElement (read a) arr

then putStrLn "It is an element of the list."

else

putStrLn "It is not an element of the list."

1. Remove the first n elements in the list.

remove :: Eq a => (Num a, Ord a) => a -> [a] -> [a]

remove n (x:xs) = do

if n < 1

then x:xs

else

if not (null xs)

then remove (n - 1) xs

else []

main :: IO()

main = do

putStrLn "Enter the list elements : "

a <- getLine

let arr = map read ( words a ) :: [Int]

putStrLn "Enter the n value: "

n <- getLine

let num = read n::Int

putStrLn ("Array after deleting the first " ++ n ++ " elements is : ")

print (remove num arr)

**Tuples**

1. Write a Haskell script to swap elements in a tuple.

swap :: (a,b) -> (b,a)

swap ( x , y ) = ( y , x )

main :: IO()

main = do

let tup = ("hello", 22)

let newtuple = swap tup

putStrLn "The swapped tuple is "

print newtuple

1. Explore the following functions wrt tuples : zip, fst, snd

main :: IO ()

main = do

let tup = ("hello", 22)

putStrLn $ "First element of the tuple is " ++ show (fst tup)

putStrLn $ "Second element of the tuple is " ++ show (snd tup)

let a = [1,2,3]::[Int]

let b = ["Jan","feb","march"]::[String]

putStrLn $ "\nArray 1 : " ++ show a

putStrLn $ "Array 2 : " ++ show b

putStrLn $ "Zip of both the lists: " ++ show (zip a b)

**Control Structures**

main = do

let age = 21

if age > 18

then putStrLn "Adult"

else putStrLn "child"

1. Based on the user input, check if it is a positive or a negative number.

main :: IO()

main = do

putStrLn "Enter a number to check its sign: "

n <- getLine

let a = read n::Int

if a>=0

then putStrLn "Positive"

else putStrLn "Negative"

1. Check if a given input is odd or even.

main :: IO()

main = do

putStrLn "Enter a number to check if it is odd or even: "

n <- getLine

let a = read n::Int

if a `mod` 2 == 0

then putStrLn "Even"

else putStrLn "Odd"