Assignment 1 – Exploring Programming Language Concepts

1. Provide a common definition of functional programming. (2 marks)

2. Haskell is considered by many to be a pure functional programming language. Explain the following piece of Haskell code and discuss its relationship with the definition in Q. 1.

```
import Data.IORef
main :: IO ()
main = do
  putStrLn "I'm going to calculate a sum, hang on a sec"
  totalRef <- newIORef (0 :: Int)
  let loop i
        | i > 100 = pure ()
        | otherwise = do
            oldTotal <- readIORef totalRef
            let newTotal = oldTotal + i
            writeIORef totalRef $! newTotal
            loop $! i + 1
  loop 1
  total <- readIORef totalRef
  putStrLn $ "The total is " ++ show total
```

3. “Immutability is preferable over mutability”. Explain why this is normally considered correct.
4. Consider the following (pseudo-) machine code:

```
mov R1, $y
mov R2, $z
add R3, R1, R2
mov $x, R3
```

   a. Write the equivalent code in C.
   b. Write the equivalent code in Haskell.
   c. Since this code is mutable, what does it imply for ALL languages?

5. Consider the following code in F#:

```
let sqrtx x = x * x
let imperativefun list =
      let mutable total = 0
      for i in list do
        let x = sqrtx i
        total <- total + x
      total
let functionalfun list =
      list
      |> Seq.map sqrtx
      |> Seq.sum
```

   a. What does each function do in the previous code?

b. Consider a subset of ISO 9126
- Reliability
- Efficiency
- Maintainability
- Portability

Argue about the impact, if any, of the two different implementations (imperativefun and functionalfun) on these characteristics.

c. Utilize the sqrtx function in Q5 to write a function which raises its argument to the 4th power.

6. *Pure functions:* A pure function is a function that, given the same input, will always return the same output and does not have any observable side effect. Functional programming likes pure functions; which of the following are pure functions:
   - changing the file system
   - inserting a record into a database
   - making an http call
   - mutations
   - printing to the screen / logging
   - obtaining user input
   - querying the DOM
   - accessing system state
   - Math.random()

7. Based on the definition of *functionalfun* presented in Q5, write a function in Rust that takes a number x and returns $\sum_{i=1}^{x} i^2 + 2$.

8. Write a Rust function that computes the volume of a sphere, given its radius.

9. What does the following Scheme function do?

```
(define (x lis)
 (cond
   ((null? lis) 0)
   ((not (list? (car lis)))
     (cond
        ((eq? (car lis) #f) (x (cdr lis)))
        (else (+ 1 (x (cdr lis))))))
   (else (+ (x (car lis)) (x (cdr lis))))))
```

10. Total functions state that, for every valid input value, there is a valid, terminating output value. In contrast to a total function, a partial function may result in an infinite loop, program crash, or runtime exception for some input.
    a. Explain what happens when you present the following Haskell code to its compiler

```
data Colour = Red | Yellow | Blue

sayColour colour =
  case colour of
    Red -> "red"
    Yellow -> "yellow"

main = putStrLn (sayColour Blue)
```

b. Explain what happens when you present the following Rust code to its compiler

```
enum Colour {
    Red,
    Yellow,
    Blue,
}

fn say_colour(colour: &Colour) -> &'static str {
    match colour {
        Colour::Red => "red",
        Colour::Yellow => "yellow",
    }
}

fn main() {
    println!("{}", say_colour(&Colour::Blue));
}
```