Yilong Wu

1. **Provide a common definition of functional programming**

   In my opinion, functional programming is to write operations as a series of function call, it doesn't depend on external data, and doesn't change the value of external data, instead return a new value. It also uses functions as variables. It focuses on describing problems rather than solving problems.

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

   This code implements an operation to sum Numbers from 1 to 100. It creats a new reference, and traversal numbers from 1 to 100 and sum them. The store it to the reference. The sum will eventually read from the reference, and print it. We can find that every operation in Haskell will be defined as a function, and it doesn't depend on external data, the code is very neat

3. **"immutability is preferable over mutability". Explain why that is normally considered correct.**

   Immutable objects have many advantages, it is simple for construction and test. Thread-safe and with no synchronization issues, you don't have to worry about the data being modified by other thread. Easy to understanding the logic of the code, you don't need to check the value of variables betweed loops. When it used as an attribute of a class, it doesn't need protective copy.

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.**

```c
void main() {
    int x,y=2,z=3;
    int R1,R2,R3;
    R1 = y;
    R2 = z;
    R3 = R1 + R2;
    x = R3;
    printf("0x%p\n",x);
}
```

   b. **Write the equialent code in Haskell.**

```haskell
main :: IO()
main = do
```

```
sum [] = 0
sum(n:ns) = n + sum ns
sum :: Integral a => [a] -> a
sum[y,z]
```

c. Since this code is mutable, what does it imply for ALL languages?

The assembly language or machine code is the most fundamental or the lowest level language in the computer system, it can directly tell the computer what to do, it doesn't need interpretor to compile. But the same assembly language translate to different languages which will have different performance and type, but essentially they implement the same functionality, the code is mutable, the functionality is immutable.

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?

The first function sqrtx is to compute x squard.
The second function squares each element in the list and add them up.
The third function uses map, map each element in the list to the function sqrtx, then make the sum

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.

About reliability, I think functional programming has a better performance than imperative programming, it doesn't involve changing the values of intermediate variables, it doesn't depend on external variables. And functionalfun's efficiency is also better than

imperativefun, the code is much more concise than imperativefun, and easy to understand. The maintainability and portability of functionalfun is also better, we don't need to check a lot of variables or status, we just need to check the function. And imperative programming focus on the problem-solving process, it reveals the every steps of problem solving process, and functional programming focus on the problem itself, it describe what the problem is more intuitively.

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

```
let sqrtx x = x * x
let imperativefun list =
    let mutable total = 0
    for i in list do
        let x = sqrtx i
        let y = sqrtx x
        total <- total + y
    total
```

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
- Printing to the screen
- Querying the DOM
- Math.random()

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

```
use std::io;    //standard library   input/output
use std::collections::HashMap;
use std::io::Read;

fn main() {
    println!("the result is: {}",result_x());
}

fn caculate(x: i32) -> i32 {
    x*x+1
}
```

```
fn result_x() -> i32 {
    let mut i = 1;
    let mut sum = 0;

    println!("please input x:");
    let mut x = String::new();
     io::stdin().read_line(&mut x) //obtain input
        .expect("Failed to read the line");

    let x: i32 = x.trim().parse() //string to integer
        .expect("Please type a number");

    while i < x {
        sum = sum + caculate(i);
        i = i + 1;
    }
    sum
}
```

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

```
fn volume_sphere(r:f64) -> f64 {
    let pai = 3.14;
    r*r*r*pai*3.0/4.0
}
```

9. What does the following Scheme function do?

We define a list x , if it is empty, then assign it to 0. If the first element is a list, determine if the first element is equal to False, if it is False, then count the rest of it, if it isn't False, count one plus the rest of it, so what this function does is caculate the number of True in the list.

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. Nothing happended, because blue doesn't match any cases.

b. I get an error, it prompts match must be exhaustive, Blue not covered.