ECE 421 Assignment 1
Jiannan Lu 1577618

1. Functional programming is declarative and mathematical, the meaning of a program is focusing on evaluation expressions than executing commands; functions are first-class, their value can be used in the same way as any other kinds of values. The same inputs will always give the identical output.
2. Purpose: calculate the sum of 0-100 numbers
   The program is applying the IORef library at the beginning, which has a mutable 'side effect' introduced, conflicts with the principle of pure. A mutable 'i' is introduced for a loop as well. An imperative style is revealed for this code.
3. Immutability helps prevent side effects, immutable data is fixed after the creation, which helps avoid data being changed elsewhere in the program (changes may lead to errors). Output is predictable with immutability, hence make debugging and testing easier as well.
4. (a) int R3 = R1 + R2
   (b) let R3 = R1 + R2
   (c) It imply for other languages that mutable operations produce side effects and is challenging, but sometimes necessary as well, languages shall provide rules for usage of mutable operations to program with reliability.
5. (a) imperativefun: iterate through a list of number, and sum up the squared values of them.
   functionalfun: map each of item in the list through 'squared', and sum up them in a declarative way instead of the imperative one.
   (b) reliability: no difference
   efficiency: not sequential for functionalfun -> may utilize CPU in a better way, cannot tell
   maintainability: functionalfun is more maintainable due to a more understandable and declarative coding style
   portability: cannot tell
   (c) let fourthpower list = list |> sqrtx |> sqrtx
6. changing the file system: ×, side effect
   inserting a record into a database: ×, mutate database
   making an http call: ×, not sure if return values are the same for the same input
   mutations: ×
   printing to the screen / logging: ×, including IO, side effect
   obtaining user input: ×, side effect
   querying the DOM: √
   accessing system state: ×, not sure if return values are the same for the same input
   Math.random(): ×, return values are not the same for each call
7.
```
fn functionalfuninrust(x: Vec<i32>) -> i32 {
    let sum: i32 = x.iter().map(|&i| i*i+2).sum();
    sum
}
```
8.
```
fn volume(r: f64) -> f64 {
    let vol: f64 = 4.0 / 3.0 * std::f64::consts::PI * r * r * r;
    vol
}
```

9. There exists no specific case for the sayColour Blue, which results in a non-exhaustive pattern warning, because the sayColour function shall cover all possible cases for Colour’s cases, but no error produced.