Part 1: Theoretical Questions

1. The answer:
    a. Imperative programming is a way of programming in which we control the flow of the program by explicit code commands. This means that we won't give much flexibility to the program in the hopes that the program will return the value we expect to get, but instead, the value returned by the program is computed by following the control flow of an explicit sequence of commands.
    b. Procedural programming follows the imperative paradigm in the sense that its code is also an explicit list of commands performed one after the other, with an emphasis on separation into nested procedures. The code is divided into smaller tasks (procedures) and each task performs a different logical part of the code, these tasks can themselves use other tasks to facilitate their computations. These tasks are each performed separately and have their own local variables that don't affect the state of the program outside the scope of the tasks.
    c. Functional programming is a paradigm in which the computation is handled only using given values that won't change during computation. In functional programming we use only pure functions that don't change shared variables between threads and use immutable objects as opposed to mutable objects used in traditional object-oriented programming. In other words, the state of the program is never changed. The idea of functional programming is that we start with certain arguments and perform consecutive computations on said arguments until we get our output, the goal of the program is the computations and therefore there are no side effects (for example: printing to the screen or changing the value in an array)
2. The answer:

```
const functionalAverageOver60 = (grades: number[]):number => {
   const above60:number[] = grades.filter((grade:number) => grade>60);
   const divser:number =R.length(above60);
   return above60.reduce((acc:number,cur:number) => acc + cur,0)/divser;
}
```

3. The answer:
    a) The expression is of the function that receives an array x of generic type T, and a function y that receives a generic type T and returns a boolean. The type of this function returns a boolean.
    b) The expression is of the type function that receives a number array and returns a number.
    c) The expression is of the type function that receives a boolean and an array of generic type T and returns a value of type T.
    d) The expression is of the type function that receives two functions: function g of the type that receives a number and its return type is of generic type T, and function f of the type that receives a value of type T and returns a generic type S. The function returns a function that receives a number and returns a value of the generic type S that the f function returns.

4. The answer:

   Abstraction barriers are what separate procedures of high-level and low-level. They also separate between the implementation of data types and the usage of these data types by high-level functions. High-level functions will only call low-level functions. The implementation of low-level functions is isolated from the user.