Part 1:

Question 1:

(a). I. **Imperative Programming Paradigm** is a style of programming where the program is a sequence of explicit commands and instructions that executed in a specific order to reach a specific goal. The focus is on how to perform tasks.

II. **Procedural Programming Paradigm** is a subset of the imperative paradigm that structures the program around procedures/functions. The language syntax can define a piece of code as a procedure, and can be called from different places in the code.

III. In the **functional programming paradigm**, the program is a series of expressions with a value. By running the program, we calculate the expressions and finding their value.

Functional programming includes immutability and the use of higher order functions. Also there are no side effects, the output of the functions depends only on their input. Functions are also expressions (meaning that the invocation of a function is an expression, with a value).

(b). The procedural paradigm allows different pieces of code to be defined as procedures, this has several advantages:

Reusability: Procedures can be called from multiple parts of the program, reducing code duplication. Reducing the need to change code when running a procedure on different parameters and therefore reduce potential mistakes.

Readability: the syntax of the language contributes to understanding the code more easily and Improves code organization and maintenance.

(c). functions without side effects: the immutability reduces bugs since functions behavior is more predictable and consistent. In addition, functional programming approach improve maintenance and enables high order functions which results more expressive.

Concurrency: immutability improved concurrent execution safety, as there are no mutable shared states.

Testing: Easier to write unit tests for functions as their behavior is deterministic.

Question 2:

```
const getDiscountedProductAveragePrice = (inventory: Product[]): number => {
  const discountedProductsPrices = (inventory.filter(product=> product.discounted))
      .map(product=> product.price);
  const discountedPricesSum = discountedProductsPrices
      .reduce((sum, price) => sum+price, 0);
  return discountedProductsPrices.length === 0 ?
      0 : discountedPricesSum / discountedProductsPrices.length;
}
```

Question 3:

(a) $< T > (x{:}\ T[\,]\ ,\ y{:}(pred{:}T) \Rightarrow boolean) \Rightarrow boolean$
(b) $(x{:}number[\,]) \Rightarrow number$
(c) $< T > (x{:}boolean,\ y{:}\ T[\,]) \Rightarrow T$
(d) $< T_1, T_2 > (f{:}(y{:}\ T_1) \Rightarrow T_2\ ,\ g{:}(z{:}number) \Rightarrow T_1)\ \Rightarrow ((x{:}number)\ \Rightarrow\ T_2\ )$