Part 1:

Q1:

A:

i. Imperative: The program is a sequence of commands, running a program is the execution of the commands, one after the other.
Examples: Java, C++, Python, assembly.

ii. Procedural: The syntax of the language allows code to be defined as a procedure, so that it can be read from different places in the code.
Examples: Java, C++, Python.

iii. Functional: The program is an expression, or a series of expressions, not a sequence of commands. Running the program is a calculation of the expression(s), i.e. finding its value, rather than executing the commands.

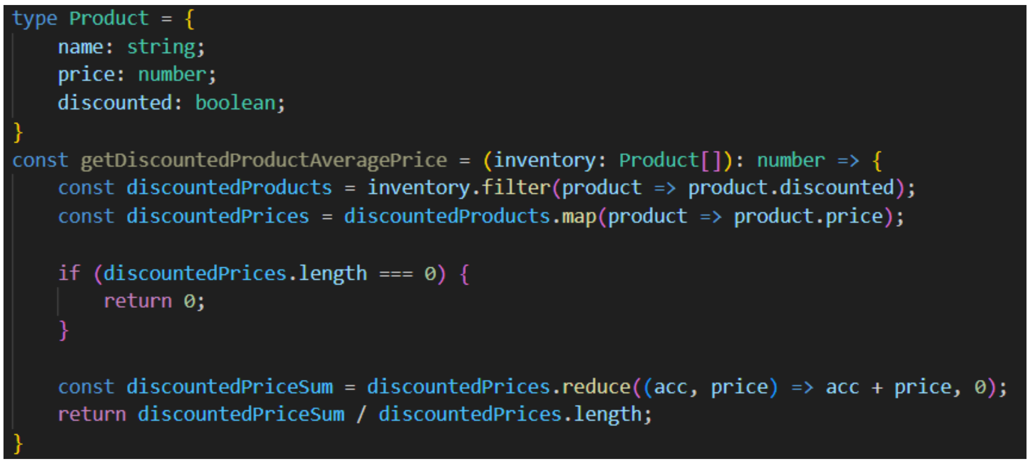
B:

In the beginning, languages evolved from a language with only a sequence of commands, with the need to write the same commands to make a specific addition or simple change, to a programming language with variables, data structures, and loops, and then programming languages improve over by adding layers of abstraction in the form of procedures, well-defined contracts and can encapsulate local variables.

C:

The functional paradigm improves over the procedural paradigm by discouraging the use of shared state and mutation, which makes testing, formal verification, and concurrency easier.

Q2:



```
type Product = {
    name: string;
    price: number;
    discounted: boolean;
}
const getDiscountedProductAveragePrice = (inventory: Product[]): number => {
    const discountedProducts = inventory.filter(product => product.discounted);
    const discountedPrices = discountedProducts.map(product => product.price);

    if (discountedPrices.length === 0) {
        return 0;
    }

    const discountedPriceSum = discountedPrices.reduce((acc, price) => acc + price, 0);
    return discountedPriceSum / discountedPrices.length;
}
```

Q3:

(a) <T> (x: T [], y: (value: T) => boolean) => boolean
(b) (x: number []) => number
(c) <T> (x: boolean, y: T []) => T
(d) <T, U> (f: (b: T) => U, g: (a: number) => T) => (x: number) => U