1.
    a. Imperative programming is characterized by linear control flow. A program is a series of commands that are executed one by one by order of appearance in the code.
    b. Procedural programming is an extension of the imperative programming paradigm – it adds the ability to organize commands in a more hierarchical and nested structure. This helps eliminate duplicate code and make code more readable and modular.
    c. Functional programming is distinct in its avoidance of global state mutations and side effects. It consists of nested function calls and function composition, where each expression has a value of its own. It eliminates the problem with asynchronous programs, and removes unnecessary code (e.g., loops and variable declarations).
2.

```
a. <T> (x : T[], y : (T) => boolean) => Boolean
b. (number[]) => number
c. <T>(boolean, T[]) => T
```

3. Abstraction Barriers – the idea of abstraction barriers is to identify a set of operations as a function or some data object/structure to improve code readability and to ease code reuse.
   Abstraction Barriers can often be used for cyber security reasons.
   For example, multiplying two rational numbers might have several operations to do, and to ease that, we can write a function (named “rat_multiply” or “X” for example) that get 2 rational numbers and returns the solution of multiplying them. Now, whenever we need to multiply in our code two rational numbers, we can use the above function instead of thinking and writing the operations every time we want to multiply two rational numbers.