Part 1

1. (a) Imperative programing paradigm – Control flow is an explicit sequence of commands. i.e. A sequence of commands states how to get a wanted result.

(b) Procedural programing paradigm - Imperative programming organized around hierarchies of nested procedure calls.

(c) Functional programing - Computation proceeds by (nested) function calls that avoid any global state mutation and through the definition of function composition.

Procedural programming languages are also imperative languages, because they make explicit references to the state of the execution environment. However, in procedural programing, control flow is split into blocks of commands, which allows code organization, and also supports nested procedure calls.

Functional programing avoids and prevents “side effects”. Since a program isn’t dependent on its state, the values of independent sub-expressions may be determined simultaneously, without fear of interference or conflict, and the final result is not affected by evaluation order. This is one example of how functional programing improve over procedural programing.

* 1. <T> (x: T[], y: (element: T) => boolean) => boolean = (x, y) => x.some(y);
  2. (x: number[]) => number = x => x.reduce((acc: number, cur: number) => acc + cur, 0);
  3. <T1, T2>(x: T1, y: T2[]) => T2 = (x,y) => x ? y[0] : y[1];