Part 1: Theoretical Questions

1. (a) Explain the following programming paradigms:

    i. Imperative: Control flow is an explicit sequence of commands - mainly defined in contrast to “declarative

    ii. Procedural: Imperative programming organized around hierarchies of nested procedure calls.

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

    (b) How does the procedural paradigm improve over the imperative paradigm?

    The procedural paradigm improves over the imperative paradigm by adding layers of abstraction in the form of procedures. Procedures interact through well-defined contracts, and can encapsulate local variables.

    (c) How does the functional paradigm improve over the procedural paradigm?

    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.

2. Convert the following function to adhere to the Functional Programming paradigm, using some or all the functions we saw in class: map, filter, reduce: function

    const sumEven = (numbersAsString: string[]) => numbersAsString.map(num => parseInt(num, 10)).filter(num => num % 2 === 0).reduce((sum, num) => sum + num, 0);

3. Write the most specific types for the following expressions:

    (a) (x, y) => x.some(y):

        <T>(x: T[], y: (z: T) => boolean) => boolean

    (b) x => x.reduce((acc, cur) => acc + cur, 0):

        (x: number[]) => number

    (c) (x, y) => x ? y[0] : y[1]:

        <T>(x: boolean, y: T[]) => T

    (d) (f,g) => x => f(g(x+1)):

        <T1, T2>(f: (b: T1) => T2, g: (a: number) => T1) => (x: number) => T2

4. Explain the concept of “abstraction barriers”'

    Abstraction barriers isolate different “levels” of the system. The implementer of the high-level system need not know about low-level details.