Part1

1. Question 1
    a. **Imperative Programming paradigm**- is a programming paradigm that goes by a sequence of commands. A good example of that is the Basic programing language.
    b. **Procedural Programming paradigm -** is a programming paradigm that is imperative programing that organized around hierarchies of nested procedure calls (And this why it calls like that). A good example of that is the C programing language.

       The procedural paradigm is improved over the imperative because in the procedural paradigm we can use procedures calls. That makes our code and more readable. Moreover, in the imperative paradigm, we can get to “Spagahti Coding” because of the GOTO/Jump function, which can’t happen in the Procedural paradigm because is working in hierarchies.

    c. **Functional Programming paradigm -** is a programming paradigm that in the language we have functions (such as math function, i.e. sqrt), this paradigm avoid from mutation and use nested function calls.
       The Functional paradigm is improved over the procedural because in this paradigm we only depended on the arguments we get in the function. This makes the programming easier and returning all the time the expected values as in the Math function.

2. Question 2
    a. (x, y) => x.some(y)
        i. x is array
        ii. y is pred
    b. x => x.reduce((acc, cur) => acc + cur, 0)
        i. x is an array
        ii. acc is the accumulator
        iii. cur is an item in a
        iv. and 0 is init of the function
    c. (x, y) => x ? y[0] : y[1]
        i. x is a condintion
        ii. y is an array
        iii. if x= true than we return y[0] else we return y[1]

3. “abstraction barriers” is that when we use data type or function we don’t know how it is implanted. We have a “barrier” that we can’t see any lower-level procedures. For example, we learned in class that if we use a function (let call it f) on data type (such as an array) and that function is running a function (let call it f’) on each member of the data type there is a barrier between the “user” that user that use “f” doesn’t know that he use also the function f’.