Assignment 1:

Question 1:

Imperative :

Imperative programming is a paradigm that change the program's state throw a series of commands that focus on describing how a program operates, each command wil be constructed with primitve functions that will preform one action only, there for if a command should be executed repedetly it has to be written multiple times in the code.

Procidural:

Procidural progeamming is a pardigm that expands Imperative programming by creating hierarchies of nested procedure calls (functions).

Functional:

Functional programming is a paradigm that is also performs computations by (nested) function calls, but avoid any global state mutation and through the definition of function composition.

Question 2:

Original function:

```
function averageGradesOver60(grades: number[])}
        let gradesSum = 0;
        let counter = 0;
        for (let i = 0; i < grades.length; i++)}
                if (grades[i] > 60)}
                        gradesSum += grades[i];
                        counter;++
                {
        {
        return gradesSum / counter;
{
```

Functional implementation:

```
import * as R from 'ramda '

const sum:(acc:number,x:number)=>number = (acc,x) => acc+x

const count:(acc:number)=>number = acc => acc+1

const averageGrades:(lst:number[])=>number = lst =>
R.reduce(sum,0,lst)/R.reduce(count,0,lst)

const isOver60:(x:number)=>boolean = x => x>60;

const averageGradesOver60Functional: (lst: number[])=>number = lst =>
R.compose(averageGrades,R.filter)(isOver60,lst)
```

Question 3:

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

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

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

(d) <T,E>(f(y:T)=>E,g(x:number)=>T) => E = x => f(g(x+1))

Question 4:

The "abstraction barrier" is the idea that every software has a barrier that divides the world into two parts:

The clients and the implementators.

The clients are those who use the software and the implementators are those who write it .

This concept instruct that the client should trust the software to work without thinking about the software implamentation, while on the other hand the implementator have to know how the software works, should always assume that there are bugs in it, and should search for them with the use of testing and other verefication tools.

This concept allow developers on one the implementors side of the barrier to modify, update, maintaine and improve thier code without effecting the client side of the barrier.