PPL-Assignment 1

1. Imperative-

focuses on describing how a program operates step by step. Programs are written as a series of statements that modifying the program state.

Characteristics-

* program maintain mutable data structures, - variables can change during execution
* can have side effects
* the programmer specifies the order of the instruction.

Examples- java, python, c++, assembly.

1. Procedural-

The language allows to define a series of statements as a procedure so you can call it from different location in the code.

Characteristics-

* Top-down design- starts with the main procedure and call other procedures.
* Modularity- programs are structured around procedures.

Examples- java, python, c++.

1. Functional-

The program is an expression or a series of expressions and not a sequence of orders, executing the program is to calculate the expressions.

Characteristics-

* Functions are expressions with a value
* You can use a function as an argument in other functions and to return a function
* No assignments and therefore no side effects

Examples- java, python, js, scheme and L1-L5.

(b) procedural paradigm benefits over the imperative paradigm-

* allows us to write the code more clearly by calling actions (procedures) in informative name which makes the code easier to understand.
* It prevents mistakes by avoiding writing the same code twice.
* Because each function preforms small number of actions it's Easier to debug and to find mistakes in the code.
* In case a change in the code is required- relatively small number of functions needs to be changed.

(c) functional paradigm benefits over the procedural paradigm-

* By avoiding using mutable data structures its easier to safely execute functions in parallel.
* Predictable behavior- data doesn’t change unexpectedly during program execution.
* It is possible to get a function as an argument which allows us to create higher order functions to help the code to abstract common patterns.

2.

Const isDiscounted =( p : product): boolean => { p.discounted == true};

Const getPrice =( p : product): number => { p.price};

Const getDiscountedProductAveragePrice= (inventory: Product[]) : number =>{

let discountItems : product[] = filter(isDiscounted, inventory);

let discountedPrices: number[] = map(getPrice, discountItems);

let avr: number = reduce (+, 0, discountedPrices) / discountItems.length ;

Return avr;

}

3.

a. <T1,T2>[T1 \* T2] =>Boolean

b. (reducer:(acc:number ,cur:number)=>number,init:number,arr: number[])=>number

c. <T>[Boolean \* T[] ]=>T

d. <T1,T2> (f: (y:T1)=>T2 , g:(x:number)=>T1)=>(x:number)=>T2