**a. In your own words, explain what is meant by the functional paradigm in computer programming? What are the perceived advantages of this programming paradigm?**

1. I think that Functional paradigm means computation by pure function calls that avoid any global mutable state or side-effects-based on Category Theory in mathematics.

2.

1) The good idea is the separation of the functions that operate on data from the data on which they operate.

2) The benefit is the resultant composition can implement potentially complex transformation in a very readable and maintainable way.

**b. List five features of the JavaScript language which allows it to support functional programming?**

1. Higher-Order Function,

2. Function Closure,

3. Curried Function

4. Lazy Evaluation (through generators)

5. Immutability (through the const and recursion)

**c. In functional programming, what is meant by the term functor? In your show, using JavaScript code, an example of a functor.**

1. A functor is a data object that can hold elements of any data type and which implements the map operation. The functor’s map( ) function takes, as an argument, another function and calls that function for each element of the functor resulting in a new functor.

2. Eg:

// JavaScript arrays are functions

[ ‘ 123 ’ , ‘ 456’ , ‘789’ ]

.map( s => parseInt (s) )

.map( n=> n/10) //=>[12.3, 45.6, 78.9]

**d. What is the visitor pattern? Describe how the visitor pattern is implemented on JavaScript arrays.**

1. The visitor pattern is an operation that operates elements of an object structure.

2. The visitor pattern is defined on the Array’s prototype property. Visitor pattern, as a single callback argument, we can loop over the keys of the array object and call the callback function passing the index and the element at the index. When all elements have been processed, visitor pattern terminates.

**e. Explain the difference between the map operation and the reduce operation, illustrating your answer with JavaScript code.**

1. Map: return a new array of same size

reduce: return a accumulator value( EG: (1) and (1’) )

2. Map: creat a new array

reduce: creat a initial value(EG: (2) and (2’) )

3. Map: Map takes, as a single callback argument

reduce: reduce takes two arguments( EG: (3) and (3’) )

Array.prototype.**map** =( callback ) =>( ){

let o=object(this) ;

let arr =[ ]; //create a new array and return (2)

o.forEach( x => {

arr.push( callback (x) ); //a single callback argument(3)

})

return arr; //return a new array (1)

}

Array.prototype.**reduce** =( callback ) =>( ){

let o=object(this) ;

let acc =initial ; // create acc (2‘)

o.forEach( x => {

arr.push( callback (x, acc) ); //takes two arguments(3’)

})

return acc; //return the accumulator value (1‘)

}

Eg:

let arr =[1, 2, 3];

arr.map ( x => x\*x ) ; //=>[1, 4, 9] //new array

arr.reduce( (x, y) => x+y , 0); //=>6 //the accumulator value