Part 1:

1. Referential transparency is an expression's property, describing an expression that is both deterministic and has no side effects.
   Meaning, every call to ReferentialTransparentFunction(x) will always result in the same output, and If x = y then ReferentialTransparentFunction(x) = ReferentialTransparentFunction(y).
   Furthermore, calling a referential transparent function does not change the value of any variable that's not discarded at the end of the function (such as a global variable, static local variable and function's arguments).
   Using referential transparent functions, make functions deterministic and therefor simplifies the code algorithm, and makes code debugging significantly easier.
   Example:

```
let transparent : (x:number[]) => number = (x) => {
   return x[0]*x[0]*x[0];
}
let opaque : (x:number[]) => number = (x) => {
   x[0] = x[0]*x[0]*x[0];
   return x[0];
}
```

2.

```
const funcAverageSalaryOver9000: (employees: employee[]) => number = (employees) => {
   const min1 = (x: number) => {
      if (x === 0) {
         return 1;
      }
      return x;
   };
   return reduce((acc, curr) => acc + curr.salary, 0, filter(emp => emp.salary > 9000, employees)) /
min1(reduce((acc, curr) => acc + 1, 0, filter(emp => emp.salary > 9000, employees)));
}
```

3.1 {name: string, degrees: {name: string, years: number} [ ] } [ ]
3.2 (f : (T3) => T4 , g: (T2) => T3, h : (T1) => T2) => T4
3.3 (pred : (T1) => boolean, arr : T1[ ]) => Boolean
3.4 (f : (T1) => number, a : T1[ ]) => number