Part 1.1:

Question 1:

Imperative — The control flow is an explicit sequence of commands

Procedural — Imperative programming organized around hierarchies of nested procedure calls.

Functional —Computation proceeds by (nested) function calls that avoid any global state mutation and through the definition of function composition.

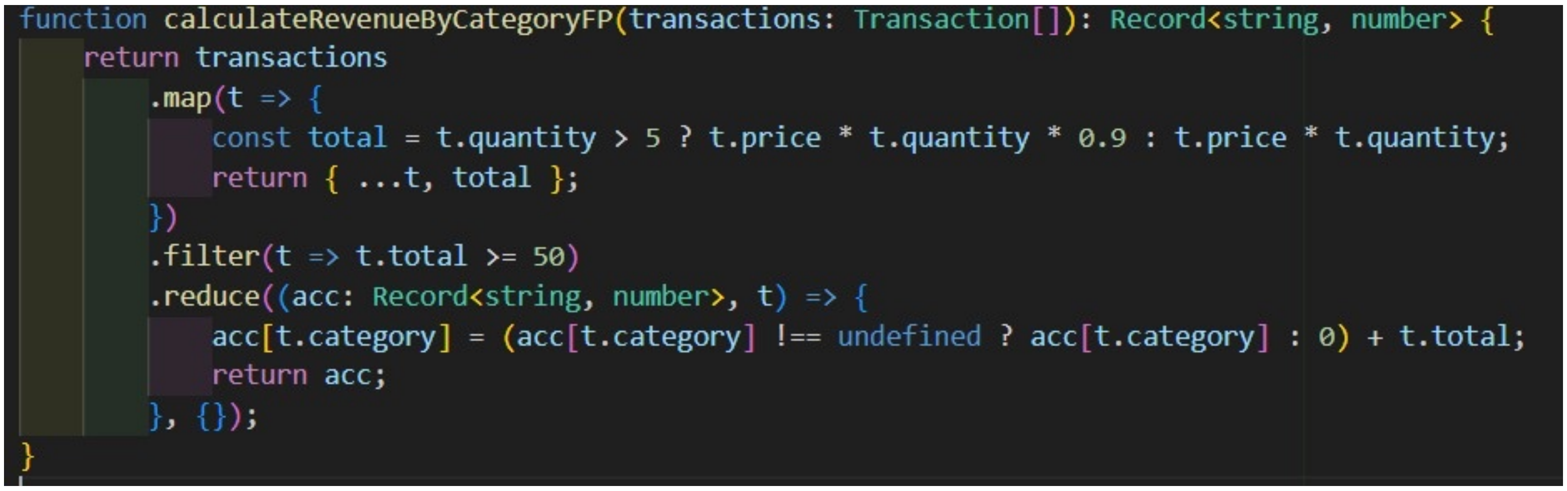
Question 2:

The procedural paradigm improves over the imperative paradigm by adding layers of abstraction in the form of procedures. Procedures interact through well-defined contracts and can encapsulate local variables.

Question 3:

The functional paradigm improves over the procedural paradigm by discouraging the use of shared state and mutation, which makes testing, formal verification, and concurrency easier.

Part 1.2:



```
function calculateRevenueByCategoryFP(transactions: Transaction[]): Record<string, number> {
    return transactions
        .map(t => {
            const total = t.quantity > 5 ? t.price * t.quantity * 0.9 : t.price * t.quantity;
            return { ...t, total };
        })
        .filter(t => t.total >= 50)
        .reduce((acc: Record<string, number>, t) => {
            acc[t.category] = (acc[t.category] !== undefined ? acc[t.category] : 0) + t.total;
            return acc;
        }, {});
}
```

Part 1.3:

1) <T>(x: T[ ], y: (z: T) => boolean) => boolean

2)(x:number[ ])=> number [ ]

3)(x: T[ ] , y: (z : T) => boolean) => T [ ]

4) (x: number[ ]) => number

5) <T>(x: boolean, y: T[ ]) => T

6) <T1, T2>(f: (b: T1) => T2, g: (a: number) => T1) => (x: number) => T2