style of coding

1. imperative style of coding

```
// E.g

let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

// imperative style ( intention + implemention tighly coupled )

function getEvens(inp) {
    let out = []
    for (let i = 0; i < inp.length; i++) {
        let n = inp[i];
        if (n % 2 === 0)
            out.push(n)
    }
    return out;
}

function getEvensAboveFive(inp) {
    let out = []
    for (let i = 0; i < inp.length; i++) {
        let n = inp[i];
        if (n > 5 && n % 2 === 0)
            out.push(n)
    }
    return out;
}

function getOdds(inp) {
    let out = []
    for (let i = 0; i < inp.length; i++) {
        let n = inp[i];
        if (n % 2 !== 0)
            out.push(n)
    }
    return out;
}
```

2. declarative/functional style

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
//-------------------------------------------------------------
// declarative/functional style ( intention + implemention loosly coupled )
//-------------------------------------------------------------
function filter(inp, predicate) {
    let out = []
    for (let i = 0; i < inp.length; i++) {
        let n = inp[i];
        if (predicate(n))
            out.push(n)
    }
    return out;
}
//-----------------------------------------------------

function getEvens(inp) {
    // return filter(inp, function (n) { return n % 2 === 0 }) //
    // return inp.filter(function (n) { return n % 2 === 0 })
    return inp.filter( n => n % 2 === 0)
}

function getEvensAboveFive(inp) {
    // return filter(inp, function (n) { return n > 5 && n % 2 === 0 }) //
    // return inp.filter(function (n) { return n > 5 && n % 2 === 0 })
       return inp.filter( n => n > 5 && n % 2 === 0)
}

function getOdds(inp) {
    // return filter(inp, function (n) { return n % 2 !== 0 }) //
    // return inp.filter(function (n) { return n % 2 !== 0 })
       return inp.filter( n => n % 2 !== 0)
}

//-----------------------------------------------------
```