Functional Programming with JavaScript

And introduction to dotless

Housekeeping

READ ME

- The session is planned for 35 mins.
- Post your questions in chat window during the session, I will try to address them as much as possible.
- Once the session is over I will share link to Google Meet to address any specific questions.
- This is first time I am doing a session in this format and channel, so please bare with me.
- If this experiment turns to be successful (which will be judged based on your feedback), then I will plan for similar sessions in future.
- Link to presentation will be posted on Facebook event page.
- After session kindly share any feedback via comments on Facebook event page.

STATUTORY WARNING

There are thousand ways to do a thing.
FP (Functional programming) is one of them.
What I am showing is one of the thousand way of doing FP.
It may not be suitable for your problem domain.
There are 999 other better ways to do it.

And don’t ask me about “Monads”

What is functional programming?

A programming paradigm where programs are constructed by applying and composing functions (wikipedia)

No side effects

Pure Functions

Higher Order Functions

Railway Oriented Programming

Composition

Currying

Partial Application

Pattern Matching

My Approach

- Small focused functions
- Expressive and intent revealing functions
- Readable code
- Composability to improve reusability
- Adopt FP concept incrementally
- Pipeline mindset to transform data

Getting familiar with latest JavaScript Syntax

```
function f1(a, b) {
    return a + b;
}
const f2 = (a, b) => a + b;

const r1 = f1(1, 2);
const r2 = f2(1, 2);
```

This is “Arrow function expressions”

Show Me Code

Higher Order Function

A function that takes a function as an argument, or returns a function

```
function where (predicate) {
    return (numbers) => {
        const result = [];
        for(const number of numbers) {
            if (predicate(number)) {
                result.push(number);
            }
        }
        return result;
    }
}

const takeOnlyEven = where(x => x % 2 === 0);
const isOdd = x => x % 2 === 1;
const takeOnlyOdd = where(isOdd);

console.log(takeOnlyEven([1, 2, 3, 4]));
console.log(takeOnlyOdd([1, 2, 3, 4]));
```

Code Please

Currying

technique of converting a function that takes multiple arguments into a sequence of functions that each take a single argument.

```
function sum(a, b) {
    return a + b;
}

function addToA(a) {
    return function (b) {
        return sum(a, b);
    };
}

const addTo2 = addToA(2);
const addTo4 = addToA(6);

console.log(sum(3, 4));
console.log(addTo2(4));
console.log(addTo4(6));
```

JavaScript do not have direct support for currying

Function Composition

An operation that takes two functions f and g and produces a function h such that h(x) = g(f(x))

- j(x) = i(h(g(f(x))))
- j = COMPOSE(f, g, h, i)
- Here COMPOSE is a utility function, which combines all these function together.
- Function ***f*** returns a value which is passed to ***g*** and so on.

Code Please

Railway Oriented Programming

- Honest function signatures
- Handling errors and failures in functional way
- Based on Either/ Maybe / Result

Code Please

Links

- Dotless Github Repo
- Dotless package on NPM
- Blog post introducing dotless
- “Railway Oriented Programming” by Scott Wlaschin
- "Functional C#: Handling failures, input errors" by Vladimir Khorikov