Houzair Koussa

Software Engineer

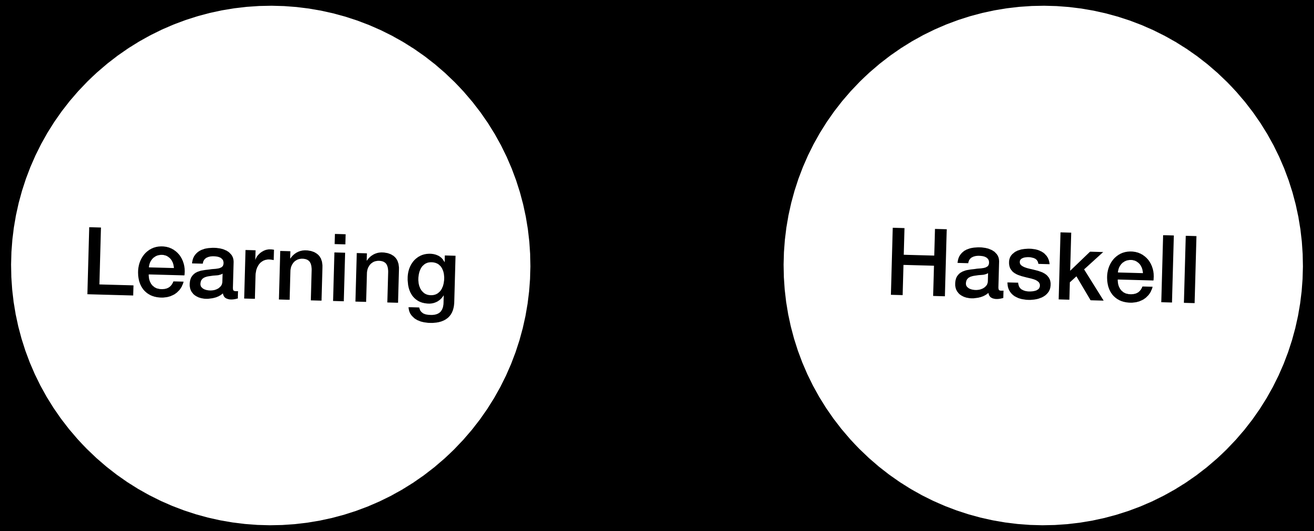
linkedin.com/in/houzairk (Houzair Koussa)

github.com/houzyk

medium.com/@houzairmk

An Overview Of Functional Programming (FP)

Disclaimers



Agenda

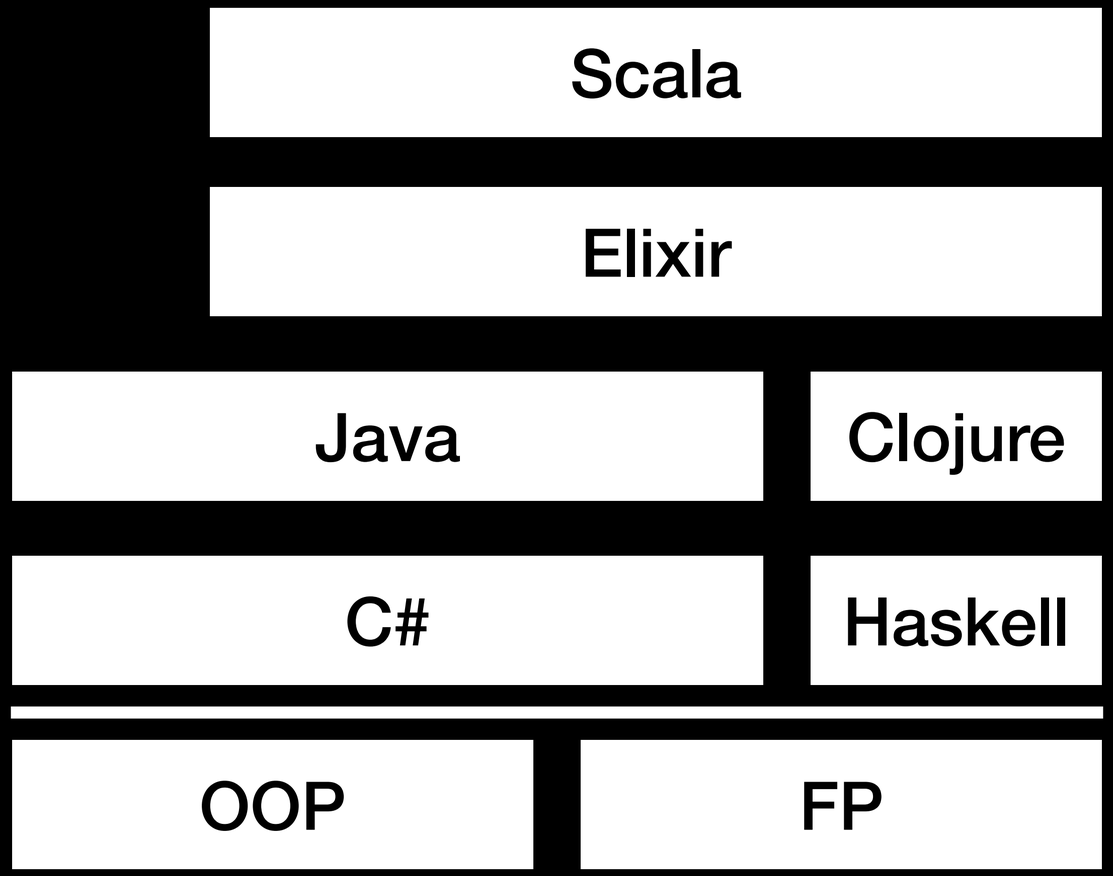
1. Context

2. Core features

3. References

4. Q & A

For Context



1. Declarative Code

We tell the computer *what* to do instead of *how*

```
const getVowelsCount = (word) ⇒ {
    const vowels = ['a', 'e', 'i', 'o', 'u'];
    const vowelsInWord = [];
    for (letter of word) {
      if (vowels.includes(letter)) vowelsInWord.push(letter);
    }
    return vowelsInWord.length;
}
```

```
getVowelsCount :: [Char] → Int
getVowelsCount word =
    length [letter | letter ← word, letter `elem` ['a', 'e', 'i', 'o', 'u']]
```

1. Declarative Code

Weird consequence - No loops

Instead, we use recursion or built-in functions (map, filters, …)

```
const factorial = (num) ⇒ {
  if (num ⩽ 1) return 1;
  for (let i = num - 1; i ⩾ 1; i--) {
    num *= i;
  }
  return num;
}
```

```
factorial :: Integer → Integer
factorial 0 = 1
factorial num = num * factorial(num - 1)
```

1. Declarative Code

Feels Restrictive But Ok

We’re already writing declarative code - JSX, CSS, GraphQL, …

…and we’re already avoiding loops with built-in functions

```
Array.prototype.map(() ⇒ <Component key={}/>)
```

2. Pure Functions

Bit more like maths functions

Core idea - Black-box that *takes an input and returns an output*

Characteristics:

a. No side-effects & mutations

b. Strict returns

c. Transparent

Anything else is “impure” (and probably won’t compile)

2. Pure Functions

No Side-Effects & Mutations

We cannot interact with outside things

So, no API calls, no mutation of external values, … (These are “impure” actions)

Also, we cannot mutate the input

```
const someOutsideValue = 24062023;

const containsSideEffectsAndMutations = async (input) ⇒ {
    const { responseNum } = await someAPICall(); // Side-Effect
    someOutsideValue += responseNum; // Side-Effect
    input *= someOutsideValue; // Mutation
    return input;
}
```

2. Pure Functions

Strict Returns

A pure function always explicitly returns

…and it always returns the *same* type

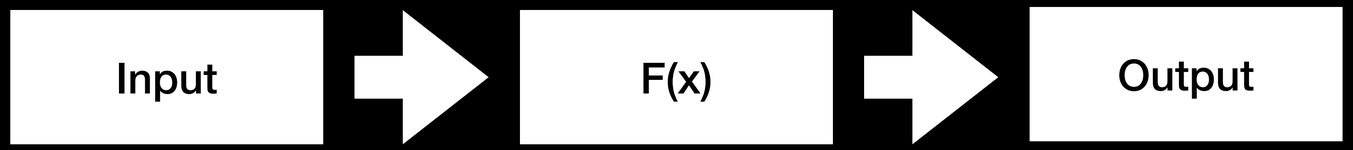
```
const notExplicitlyReturning = () ⇒ {
    window.addEventListener("scroll", onScrollCallback);
}
```

```
const returningDifferentTypes = (isInLove) ⇒ {
  if (isInLove) return ["Butterflies", "Rainbows"];
  return "Sorry, Not My Type ... ";
}
```

2. Pure Functions

Transparent

A pure function always produces the same output given a particular input




```
const notTransparent = (num) => {
  return Math.random() + num;
}
```

2. Pure Functions

Feels Restrictive But Ok

Perks:

We can still do the “impure” but highly crucial actions - Monads!

```haskell
main :: IO()
main = do
    -- Impure Actions Go Here
    -- ....
    -- ....
```

3. Functions As Data

Functions are first-class citizens

Whatever we do with data, we can do with functions (kinda)

Conceptually;

a. We can instantiate functions - Anonymous and self-invoked functions

b. We can manipulate them just like data

3. Functions As Data

Higher-Order Functions (HOFs)

A function that takes a function as argument

…and/or returns a function as output

```
const onResize = () => {
  // ...
}

useEffect(() => {
  window.addEventListener("resize", onResize);

  return () => {
    window.removeEventListener("resize", onResize);
  }
}, []);
```

3. Functions As Data

Currying

Cascade functions into one another

```
const currying = (x) ⇒ (y) ⇒ (z) ⇒ {
  // ....
}
currying('x')('y')('z');
```

3. Functions As Data

Composition

HOF + currying = composition

Core idea - To “add” functions just like we “add” data

1 + 1 = 2

“apple” + “pen” = “applepen”

[“pineapple”] ++ [“pen”] = [“pineapple”, “pen”]

4. Data Is Immutable

Once we initialise data, we can *never* change it. To modify it, we gotta create new data

Not to be confused with JavaScript's "const"

```
const imNotImmutable = [06, 2023];
imNotImmutable.unshift(24);

// imNotImmutable ⇒ [24, 06, 2023]
```

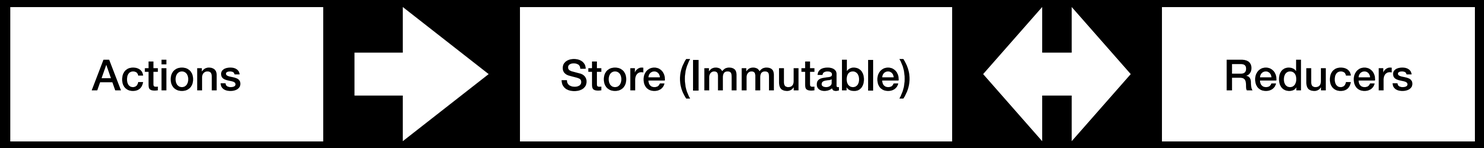
```
iAmImmutable = [6, 2023]

24:iAmImmutable -- Unshifting
-- iAmImmutable ⇒ [6, 2023]

iAmImmutableToo = 24:iAmImmutable
-- iAmImmutableToo ⇒ [24, 06, 2023]
```

4. Data Is Immutable

Feels Restrictive But Ok

Perks: We’re not unexpectedly breaking things

It’s kinda familiar - Redux



It does not hog memory nor affect performance - Persistent Data Structures!

References

1. Monads - *The Absolute Best Intro to Monads For Software Engineers* By *Studying With Alex* (YouTube)
2. Persistent Data Structures - *Anjana Vakil: Immutable data structures for functional JS* By *JSConf EU* (YouTube)
3. Haskell - learnyouahaskell.com
4. Integrated Haskell Platform - ihp.digitallyinduced.com
5. Clojure - braveclojure.com
6. Pedestal - pedestal.io
7. Functional libraries in JS - github.com/stoeffel/awesome-fp-js

Thank You!

github.com/houzyk/FECM-june-23-FP