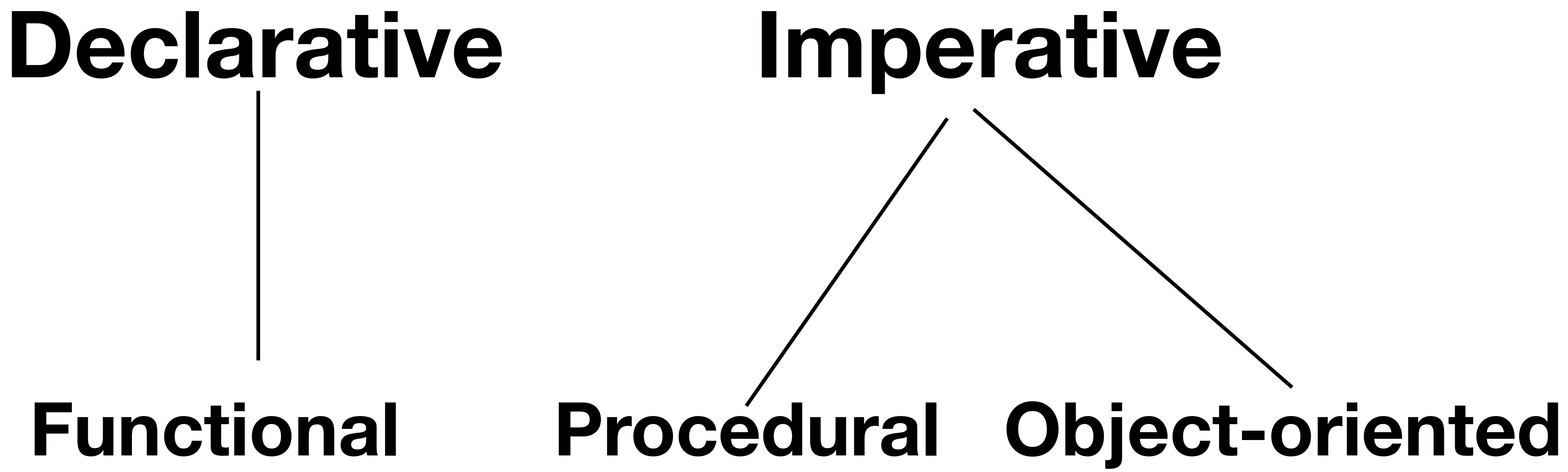
Functional Programming with JavaScript

Slides: sunneversets.studio/fp

Sample Code

- Javascript: https://github.com/Sunneversets-Studio/workshop/tree/master/3-Mar-2/Functional%20Programming%20with%20JavaScript/code/js
- Typescript: https://github.com/Sunneversets-Studio/workshop/tree/master/3-Mar-2/Functional%20Programming%20with%20JavaScript/code/ts
- You can run code in node.js or just browser
  - Open Devtools: F12 (Windows) / ⌥⌘i (macOS)
  - Click “Console”

Programming Paradigms



Construction
What to do

Instruction
How to do

Functional vs Procedural

- Construct a fridge with a elephant
- Length of linked list
    - L(p) = 1 + L(p.next)

- Open fridge, Put elephant in, Close fridge
- Length of linked list
    - Count until tail

code: 1-qsort

Features of Pure Function

- No Side Effect
- No Mutable State

- Everything is Expression

- (First Class Function)
- (Closure)

Referential Transparency (Pure Function)

- A function call can be replace with its return value
  - factorial(10) <=> 3628800
- Requires
  - no side effect (explicit output)
  - explicit input

code: 2-no-side-effect

No Side Effect
Explicit Output

- Explicit is better than implicit. (*The Zen of Python*, L2)

- Side Effect: Affect the environment
    - i.e. Any output other than return value
- Examples:
    - Modify global/static variable
    - Print to console
    - Write files
    - Modify Parameter

Explicit Input

- Can only get input from arguments
- Examples of implicit input:
  - Reading non constant global/static state
  - User input
  - Network

No Mutable State

- No variables, only bindings = constant
    - C++, ES6+: const
    - Java: final
    - Rust, Swift: let
    - Kotlin: val
- Important in closure

Everything is Expression

- Implication of No Side Effect
- Statements without a return value can be removed
- Same as no-operation in pure function:

```
function no_op(...): void
```

First Class Function

- Function as Argument
- Function as Return Value
- Function Literals

code: 3-first-class-and-closure

Closure

- Internal Functions that captures outside scope
- Modifying outside variables is side effect

Examples in JavaScript

- Transform data without modifying it
- Useful in list processing
  - map, filter, reduce, every, some
- Prefer const
- Prefer pure function

code: 4-example

Prefer Const

- ESLint Rule: prefer-const
- Prevent incidentally assignment

```
let todos = {};
window.setTimeout(() => {
    let todo = {};
    todos = { /* result from remote API */};
}, 0)
```

Prefer pure function

- A function should be either
  - pure with meaningful return value, OR
  - no return value with side effects only
- Practices common in C++ STL
  - void vector::pop_back();
  - T& vector::back();
  - cout << v.pop() << v.pop()
- Compilation time evaluation

Applications of FP Thoughts

- Java 8 Stream
- Spark
- Redux

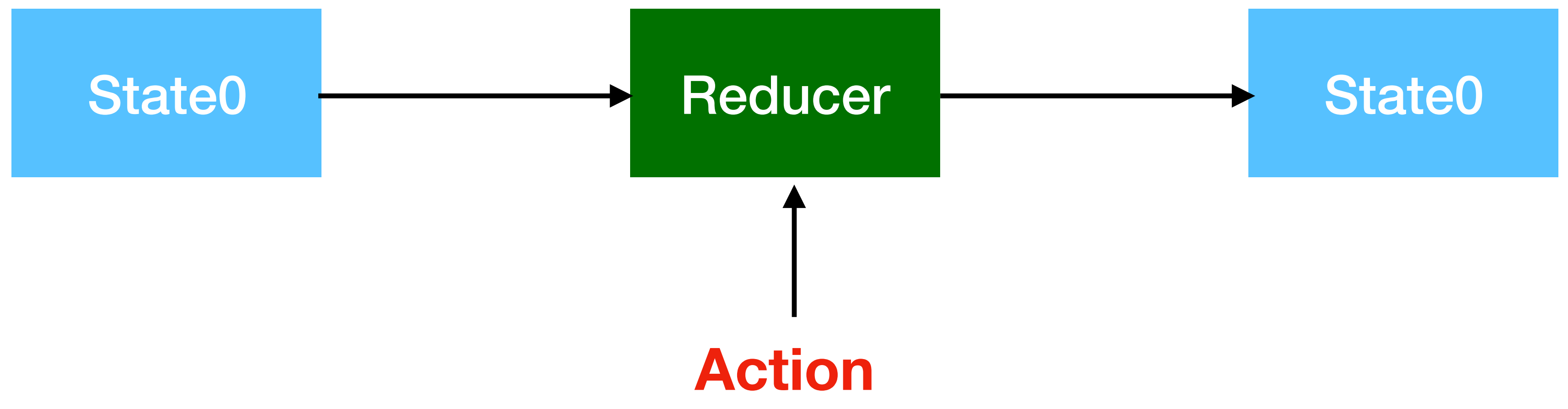
Java 8 Stream

```
int[] arr = {3, 4, 5, 1, 3, 7, 6};
IntStream stream = Arrays.stream(arr);
OptionalDouble result = stream
        .filter(x -> x > 2)
        .map(x -> x + 10)
        .average();
System.out.println(result);
```

Spark

```
text_file = sc.textFile("input")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile(“output")
```

Redux



```
reducer: (State, Action) => State
```

- Reducers must be pure functions

code: 5-redux

Further Topics

- Partial Application
- Currying
- Lazy Evaluation

- FP Languages
  - Lisp Family: Common Lisp, Scheme, Clojure
  - ML Family: Standard ML, OCaml, F#
  - Haskell
  - Scala (Twitter)
  - Erlang

References

- https://en.wikipedia.org/wiki/Functional_programming
- http://www.ruanyifeng.com/blog/2012/04/functional_programming.html