Pure function & hooks 👍

Jong

- Jongeun Lee 👄
- Organizer of JavaScript Developer Forum Korea (jsdev.kr) 🤾
- Fullstack Developer 📚
- JavaScript ❤️
- react-native, redux, terraform, AWS... 🔎

✋ Quiz 1. Fill in the blank.

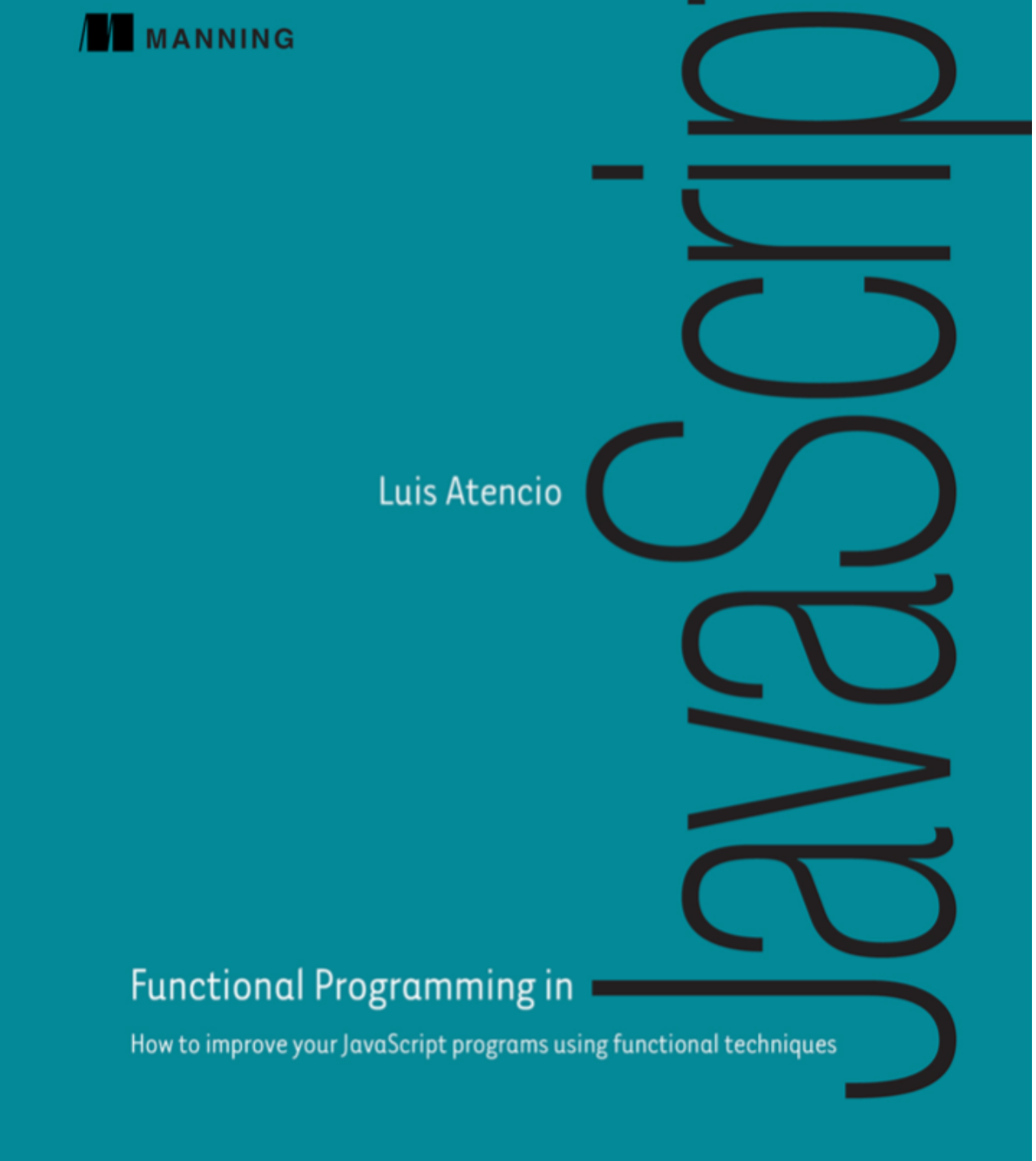
fundamental concepts on `_____` it's based:

- Declarative programming
- Pure functions
- Referential transparency
- Immutability

✋ Quiz 1-1. Fill in the blank.

Fundamental concepts on _____ _____ it's based:

- Declarative programming
- Pure functions
- Referential transparency
- Immutability



Functional programming in Javascript

by Luis Atencio

Battle of concepts

Imperative vs. Declarative

Imperative programming

- tells the computer, in great detail, how to perform a certain task.

Declarative programming

- separates program description from evaluation

Object Oriented vs. Functional

```
// Snippet A
var array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
for (let i = 0; i < array.length; i++) {
  array[i] = Math.pow(array[i], 2);
}
array; //-> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

// Snippet B
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9].map(function(num) {
  return Math.pow(num, 2);
});
//-> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Q. What is Functional programming?

a software development style that places a major emphasis on the use of functions

abstract control flows and operations on data
with functions in order to avoid side effects and reduce mutation of state in your application

> 객체 지향은 데이터와 데이터 관계의 본질에 초점을 두는 반면, 함수형의 관심사는 해야 할 일, 즉 기능입니다.

출처 : "함수형 자바스크립트"

Q. 함수형 프로그래밍의 목표

부수 효과(side effect)를 방지하고 상태 변이(mutation of state)를 감소하기 위해 데이터의 제어 흐름과 연상을 추상하는 것

FP 이면에 깔려 있는 개념

- 선언적 프로그래밍
- 순수 함수
- 참조 투명성
- 불변성

> 다른

Q. 함수형 프로그래밍이란?

Q. 함수형 프로그래밍이란?

A. 순수함수를 쓰는 것

> 시작하기전 기반 다지기

함수

🤔 What is `Function` ?

A. 함수는...

- 입력 => (do something) => 출력
- 입력은 인자, 출력은 리턴 값

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

🤔 Pure Function이란?

A. Pure Function은...

- 💧 순수 함수? 뭐가 순수하다는 걸까?
  - 수학 함수 f(x) = x * 2
  - 변덕이 없다. 한결 같다.
  - 입력이 같으면 결과가 같다.
  - 외부의 무언가를 변경해선 안된다.

어떻게 순수할 수 있나?

순수하지 못한 경우를 보자 1/3

> 호출할 때마다 달라지는 값

```
function notPure(a, b) {
  return a + b + Math.random();
}
notPure(1, 2);
notPure(1, 2);
```

순수하지 못한 경우를 보자 2/3

> 주변 값에 따라 달라질 수 있는 함수

```
var x = 3;
function dirty(a, b) {
  return a + b + x;
}
dirty(1, 2);
x = 4;
dirty(1, 2);
```

순수하지 못한 경우를 보자 3/3

> 출력값은 같으나 외부를 변경하는 함수

```
var count = 0;
function messUp(a, b) {
  count++;
  return a + b;
}
```

어떻게 순수할 수 있나?

- 원칙 1
  - 출력값에 영향을 주는 모든 것를 입력으로 받는다.
- 원칙 2
  - 함수 내부의 코드에서 함수 외부의 변수를 접근하지 않는다.
- 원칙 3
  - 외부의 변화는 외부에서 알아서 정해서 할 수 있도록 길을 열어 둠 (예. callback)

순수하게 바꿔보자 1/4

- 내부에서 random 처럼 가변적인 내용을 제거한다.
- 필요하다면 가변적인 부분을 밖으로 분리한다.

```
function notPure(a, b) {
  return a + b + Math.random();
}

function nowPure(a, b, extra) {
  return a + b + extra;
}

nowPure(a, b, Math.random());
nowPure(a, b, Math.random());
// 입력이 달라진 것, 입력이 같을 때는 출력이 항상 같음
```

순수하게 바꿔보자 2/4

- 내가 하는 일은 모두 내 안에서!
- 내부에서 외부(예. global)의 값을 참조하지 않는다.

```
// 예 2
var x = 3;
function dirty(a, b){
  return a+b+x;
}

function notDirty(a, b, extra){
  return a+b+extra;
}

notDirty(1, 2, x);
x = 4;
notDirty(1, 2, x);
// 입력이 달라진 것, 입력이 같을 때는 출력이 항상 같음
```

순수하게 바꿔보자 3/4

- 외부의 내용을 바꾸지 않아야하며 결과를 받아서 밖에서 처리하도록

```
var count = 0;
function messUp(a, b) {
  count++;
  return a + b;
}

function notMessUp(a, b, count) {
  return {
    value: a + b,
    count: count + 1
  };
}

const result = notMessUp(1, 2, count);
count = result.count;
```

순수하게 바꿔보자 4/4

- 외부의 변화이니 외부가 알아서 하도록 callback에 지정하도록 해준다.

```
function notMessUp2(a, b, callback) {
  callback();
  return a + b;
}

var count = 0;

notMessUp2(1, 2, () => {
  count++;
});
```

어디서 많이 보던 모습?

```
class CountButton extend Component {
  count = 0
  onPress(){
    alert(`${++this.count}번 눌렀어!`);
  }
  render() {
    return <Button
      onPress={this.onPress}
      title="Learn More"
      color="#841584"
    />;
  };
}
```

💧 순수하면 뭐가 좋은가?

- 입력이 있을 때 결과를 예측할 수 있다.
- 예측하면 추가적으로 테스트를 쉽게 할 수 있다.
    - 예측이 충분히 되기에 예측과 다른지를 비교하도록 테스트 코드 작성

🤔 리액트 컴포넌트 === 함수?

A. YES! 리액트 컴포넌트는 함수이다.

- ==입력== : `props`
- ==출력==
  - 의미적 출력 : `render`의 return 값(View)
  - 실제 출력 : 함수의 return 값
    - Stateless Component의 경우 : return 값(View)
    - 일반 Component의 경우 : Component 객체
      - 라이프 사이클
      - render 함수