Intro to Functional Programming with JavaScript

1.

The Lanscape

A Few Programming Languages

Programing Paradigms

Programming languages can be classified by their paradigm. The main ones are the following:

- Procedural
- Object-Oriented
- Functional

Domains of application

Programming languages can be more suited to specific domains of applications or platforms.

- ✖ Data Science
- ✖ Machine Learning
- ✖ IOT
- ✖ Mobile

Programming Paradigms

Procedural

Code that instructs a computer how to complete a task divided into detailed steps (imperative programming).

C Go

Lower Level Languages

Historically, Fortran, Cobol, Basic, Pascal, Ada

Programming Paradigms

Object Oriented

Code is express in terms of objects containing attributes (data) and behaviors (methods).

- ✖ **C++**
- ✖ **Java**
- ✖ **Ruby**
- ✖ **PHP (OOP added)**
- ✖ **Kotlin**

Dominant paradigm for more than 20 years.

Programming Paradigms

Functional

Process of building software by composing pure functions, avoiding shared state, mutable data, and side-effects. Functional programming is declarative rather than imperative

- ✖ **Clojure**
- ✖ **Elixir**
- ✖ **Elm**
- ✖ **F#**
- ✖ **Haskell**
- ✖ **Scala**
- ✖ **Lisp**

An older paradigm that became hot a few years ago.

Programming Paradigms

Mix Paradigms

Some languages are mixed and be both object-oriented and functional.

- **Python**
- **Swift**
- **JavaScript**

Data Science and Machine Learning

- Python
- R
- MathLab
- Scala
- Julia
- SQL
- Java
- JavaScript
- C / C ++

IOT

- ✖ Assembly
- ✖ C/C++
- ✖ Go
- ✖ JavaScript
- ✖ Java
- ✖ Python
- ✖ PHP
- ✖ B#
- ✖ Swift

Mobile

Native

- Java (Android)
- Swift (IOS)
- Legacy code in Objective-C (IOS)
- C# (Windows)

Hybrid

- React Native (React)
- Ionic (Angular)
- PhoneGap (Apache Cordova)
- Framework 7 (JavaScript)
- Xamarin (Microsoft)
- Onsen UI
- Mobile Angular UI

What companies are using?

- https://discovery.hgdata.com/

A Few Surveys

- Stack Overflow Developer Survey Results 2018
- Code Mentor - Job Opportunities
- Tiobe Index
- The State of JavaScript 2017
- The Incredible Growth of Python

JavaScript

Why JavaScript

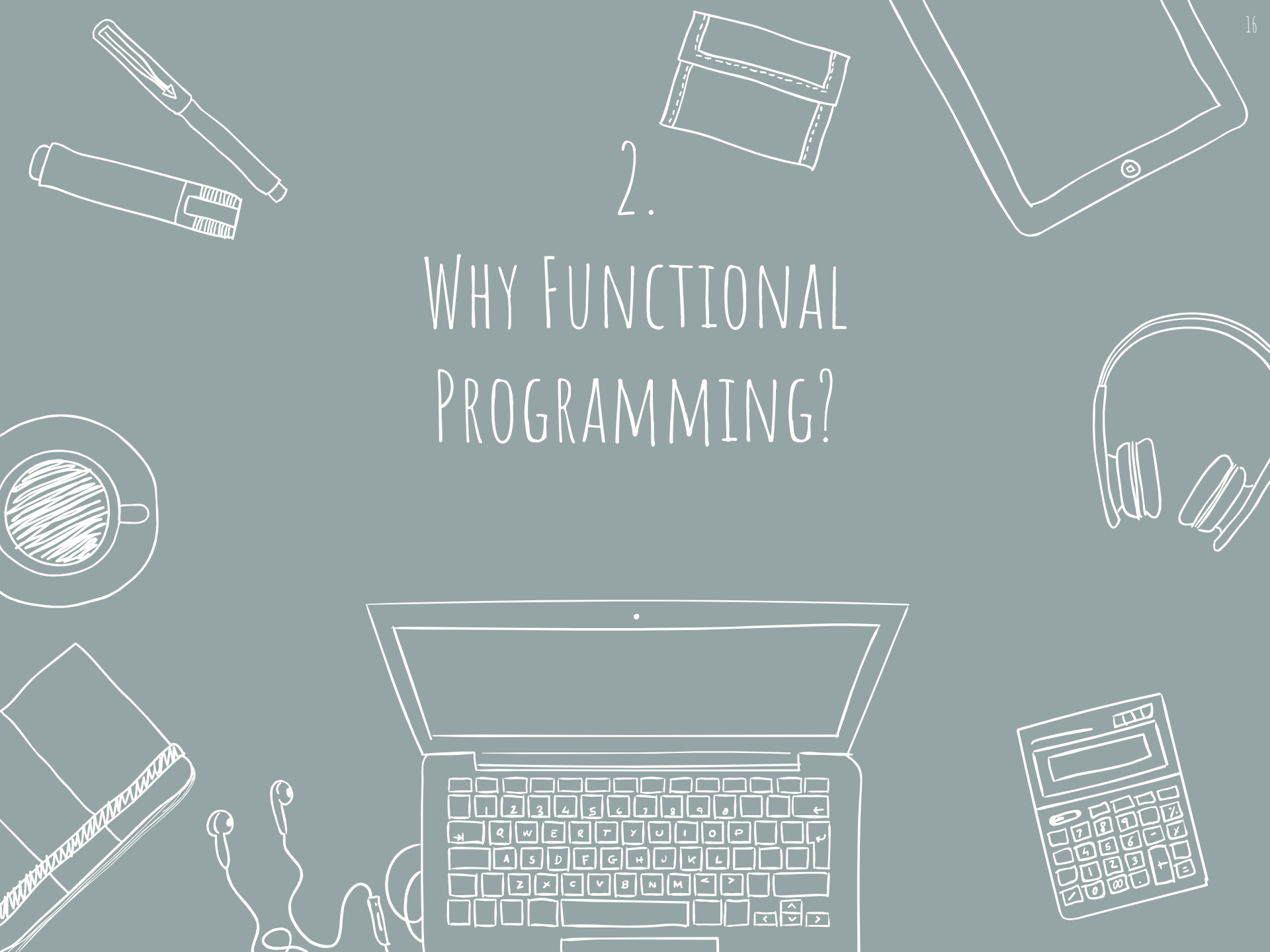
JavaScript is a flexible language that can be both object-oriented and functional.

- JavaScript is everywhere!

  Mobile applications, websites, web servers, desktop and embedded applications, and even databases.

- JavaScript was inspired by other functional languages such as *LISP* and *Scheme*

- JavaScript supports **higher-order functions** and closures (it's pretty powerful!)



2.

Why Functional Programming?

Because of App Complexity

Today Web applications are way more complex than back in the days:

- ✖ Async processes
- ✖ Events firing
- ✖ Complex user interface
- ✖ Behave like native / mobile apps
- ✖ As applications get bigger, so does their complexity

Because we are communicators!

"I believe very deeply that the vastly more important role of code is as a means of communication with other human beings...It's widely estimated that developers spend 70% of code maintenance time on reading to understand it."

-- *Kyle Simpson*

We need to focus on readability.

More Readable Code

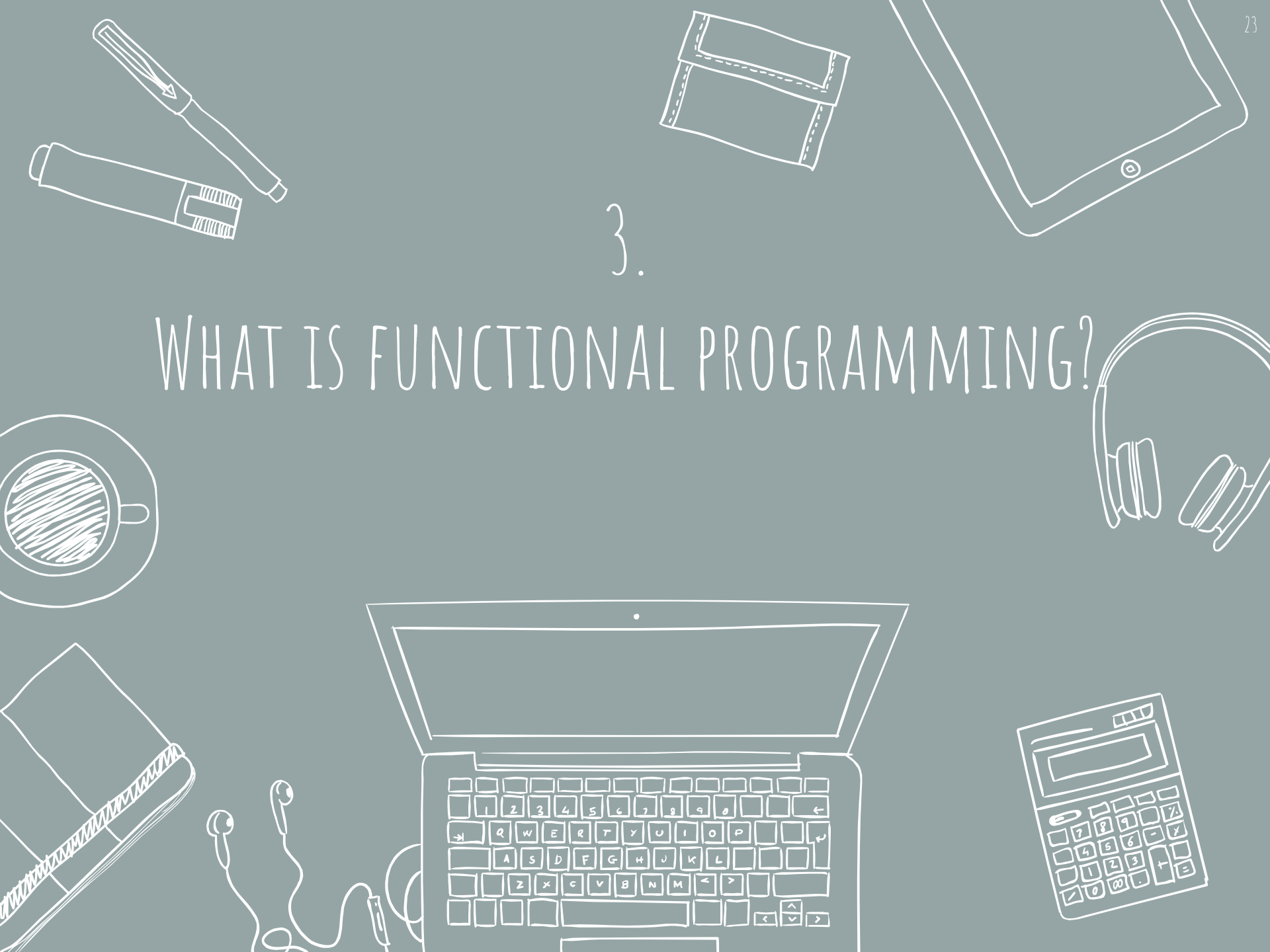
Thinking in terms of functional programming help us to:

- ✖ Write cleaner code
- ✖ Write less code
- ✖ Write more modular code

Less bugs

Thinking in terms of functional programming help us to:

- ✖ Write code with less bugs
- ✖ Write code that is easier to test and debug
- ✖ No ***this*** keyword!



3.

What is functional programming?

Functions

- In functional programming we want to express everything in our program with functions
- Functions does not modify the state of our app and avoid producing side effects.

What is a function?

- Any expression that can be called with ()
- Functions can return either a computed value or undefined.
- Functions can have input parameters.

What is a function?

Input parameter

```
const average = numbers => {
  let sum = 0;
  for (let i = 0; i < numbers.length; i++) {
    sum += numbers[i];
  }
  return Math.round((sum / numbers.length) *
100) / 100;
};
```

Output Value

FP STYLE FUNCTIONS

- functions should have a single purpose (singularity principle).
- function should always have ***input parameters*** and ***return a value.***
- *The use of **this** keyword is avoided.*
- *In FP, functions are First-Class and Higher-Order.*

Decomposition

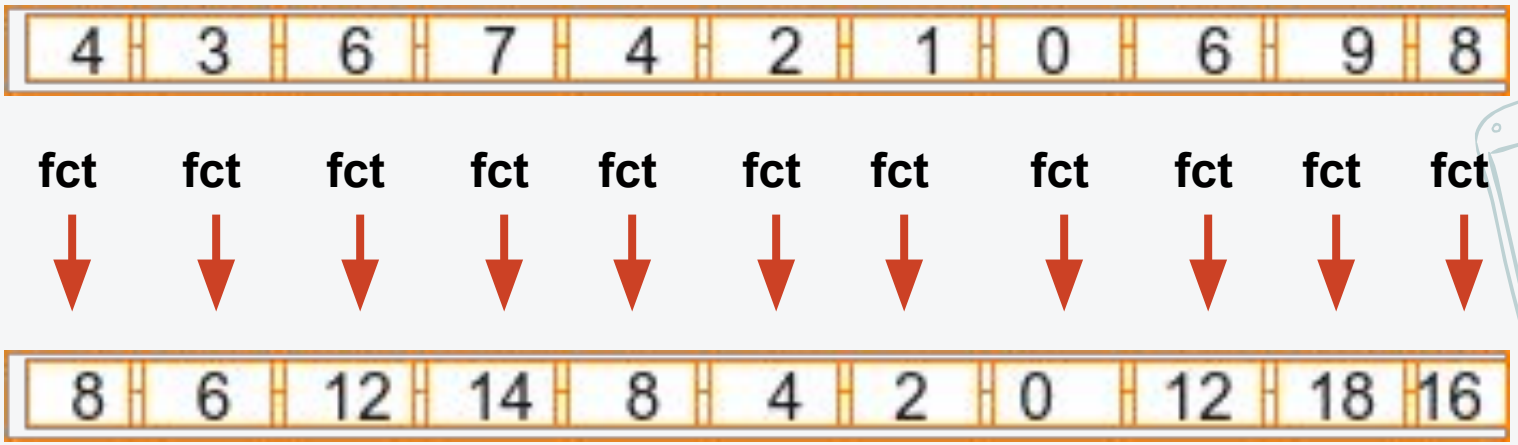
```
// Do the functions have a single purpose?

const sum = numbers => {
 let sum = 0;
 for (let i = 0; i < numbers.length; i++) {
   sum += numbers[i];
 }
 return sum;
};

const average = numbers => {
 return round(sum / numbers.length);
};

const round = number => Math.round(number * 100) / 100;
```

Built-in FP Style Functions in JS

- ✖ map
- ✖ filter
- ✖ reduce
- ✖ find
- ✖ some
- ✖ every

Transforming data with map

Input Array



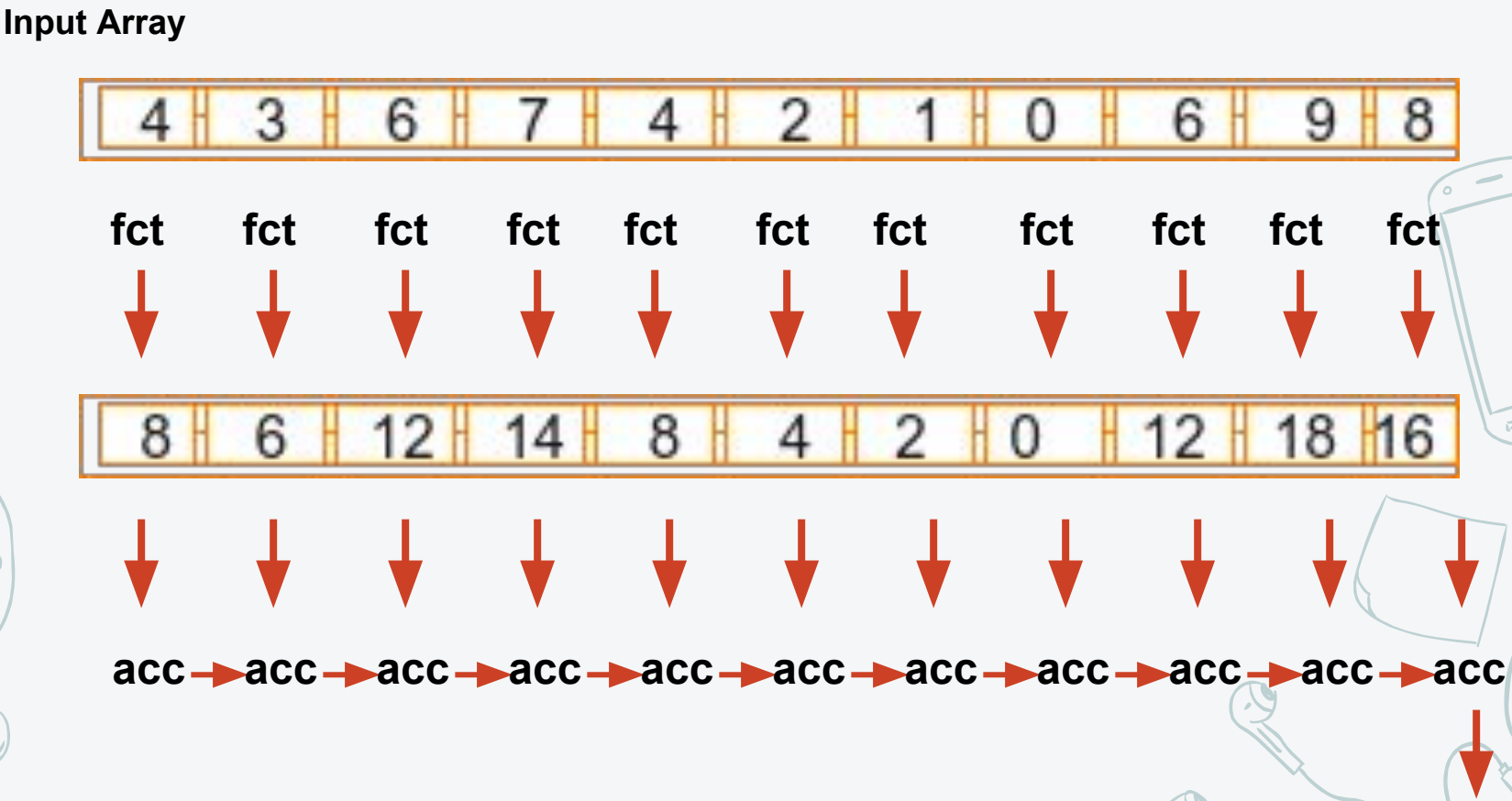
Output Array

Filtering data with filter

Input Array

Output Array

Gathering data with reduce



Gathering data with reduce

Reduce is very powerful. You can do a lot with it.

- ✖ Sum the values or an array
- ✖ Find the min or max value of an array
- ✖ Calculate the average
- ✖ Even create an object!

Chain map, filter, reduce

Example

```
const products = [
 {
  id: 1,
  name: "Art piece",
  price: 5000,
  quantity: 1,
  category: "Art"
 },
 {
  id: 2,
  name: "Watch",
  price: 500,
  quantity: 10,
  category: "Jewelry"
 },...
]
```

Filter by electronics

```
products.filter
    (product => product.category === "Electronics")
```

Calculate Each Product Total Value with Map

```
products.map
    (product => product.quantity * product.price)
```

Calculate Total value of the inventory with reduce

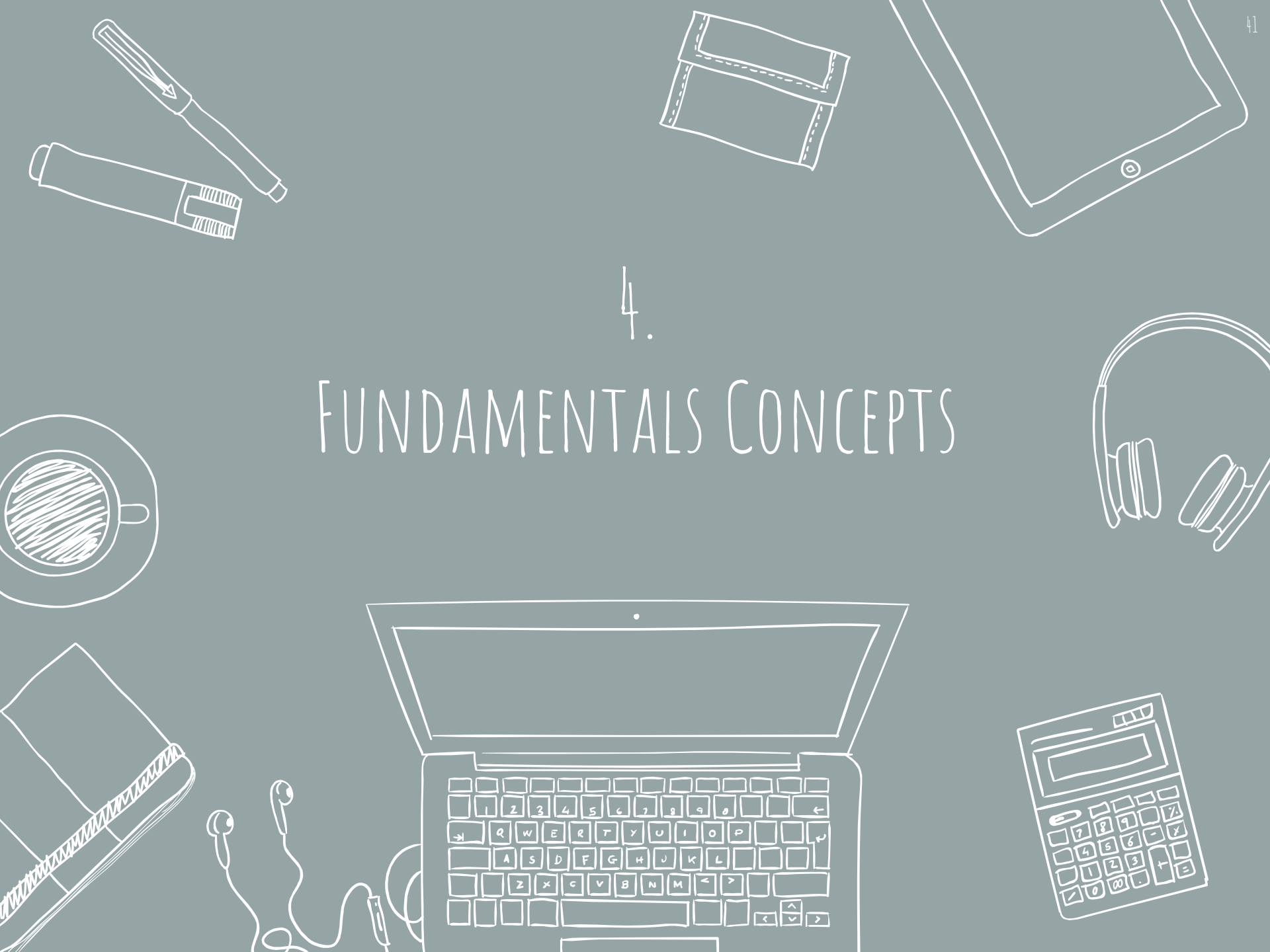
```
products.reduce
    ((total, next) => (total += next.price * next.quantity), 0)
```

Chaining them

```
const inventory = products
 .filter(product => product.category === "Electronics")
 .map(product => product.quantity * product.price)
 .reduce((total, next) => (total += next), 0);
```

The same result with imperative coding

```
const inventory = products => {
 let total = 0;
 for (let i = 0; i < products.length; i++) {
   if (products[i].category === "Electronics") {
     total += products[i].price * products[i].quantity;
   }
 }
 return total;
};
```



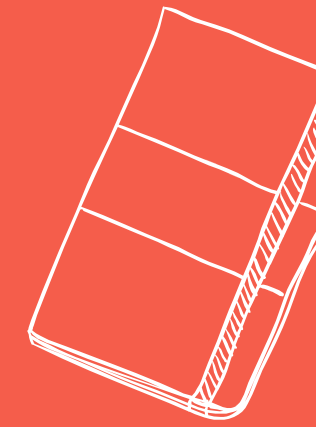
4.

Fundamentals Concepts

FP Concepts

- ✖ Declarative programming
- ✖ Pure functions
- ✖ First-Class functions
- ✖ Closures
- ✖ Immutability
- ✖ Functions Composition
- ✖ Currying

Declarative Programming



Imperative Programming

- A set of details step by step instructions that tells the computer how to perform a task we
- It’s how we usually code

Imperative Programming

```
var numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
for (let i = 0; i < numbers.length; i++) {
  numbers[i] = numbers[i] * 2;
}
```

Declarative Programming

- The goal is to write expressions, as opposed to step by step instructions.
- Focus on the expected result.

Ex.: Driving by following directions to a destination vs taking Uber, SQL

Declarative Programming

```
var numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
numbers.map(number => number * 2)
```

Pure Functions

Pure Functions

- ✖ It depends only on the input provided and not on any hidden or external state that may change during its evaluation or between calls.
- ✖ It doesn't produces changes beyond their scope, such as modifying a global object or a parameter passed by reference.

Impure Functions

```
var numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
const double = () => {
    for (let i = 0; i < numbers.length; i++) {
      numbers[i] = numbers[i] * 2;
    }
}
```

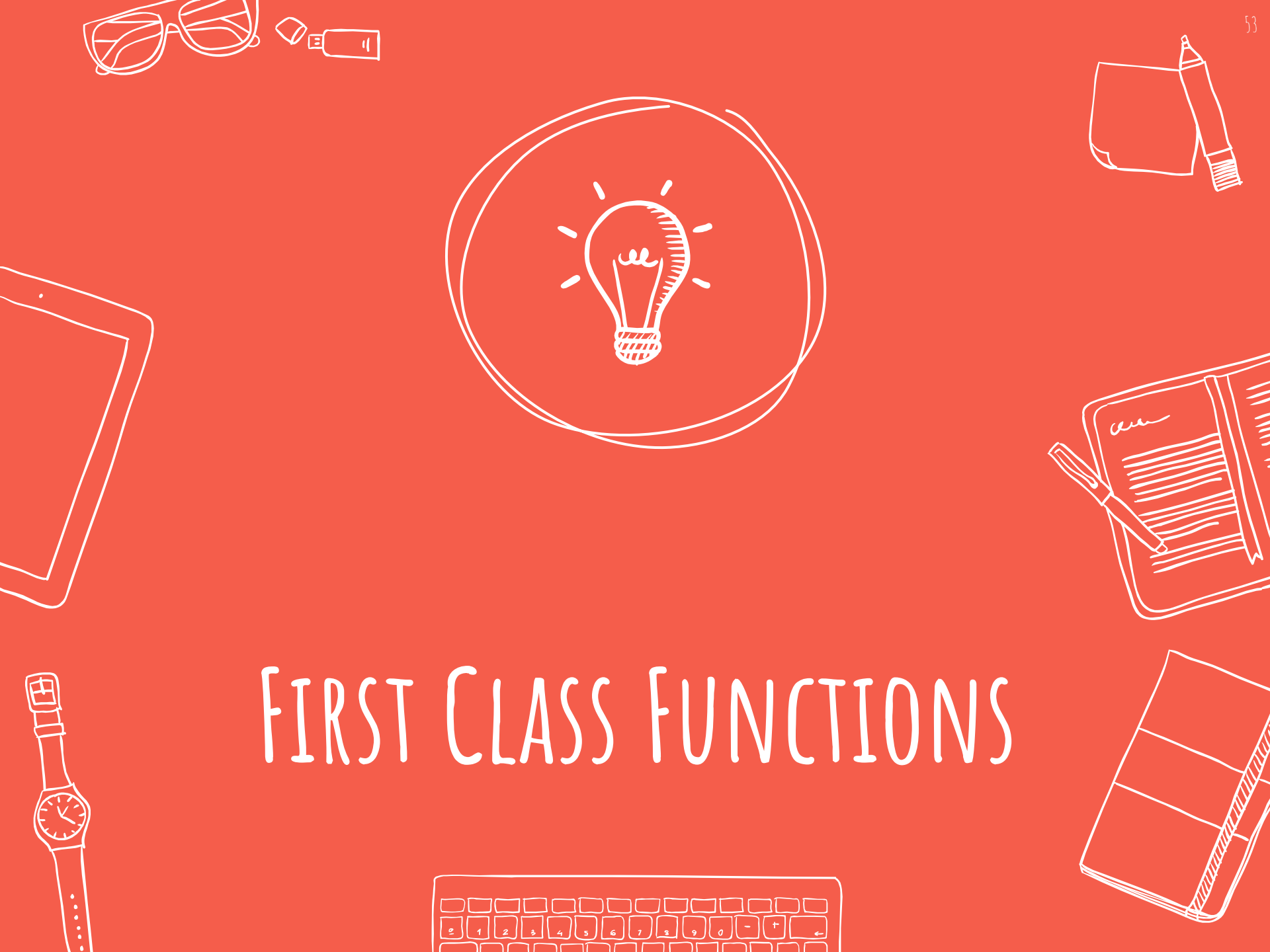
- ✖ The function changes numbers that is in the outerscope
- ✖ Executing double will have a different result each time

Pure Functions

- A program where functions modify global data will cause problems.
- It makes your program much harder to reason about because it produces side effects.
- You have to keep track of the state of your app. As the program gets larger, this process becomes nearly impossible.

Pure Functions

- ✖ **Avoid** manipulating the global scope at all cost within your functions.



First Class Functions

First-Class Functions

- First class functions treats functions as first class citizen (What??)
- A function can be assign to a variable like any other types of data
- As with any other data types, it can be passed around as arguments

First-Class Functions

```
const  aVariable = true // value is a boolean
const  aVariable = “hello” // value is string
const  aVariable = 10 // value is integer

//  the value is a function
const  aVariable = word => return word + “ World!”
```

Higher Order Functions

- A function that takes other functions as parameters and/or return another function

Higher Order Functions

Takes a function as a parameter

```
const listMessages = (messages, fct) => {
 messages.forEach(fct);
};

listMessages(msgArr, message =>
console.log(message));
```

Closures

CLOSURES

- The function that is being returned closes over its outerscope
- I.e. it will keep values in its outerscope

Closures

x = 5

```
const createAddFct = x => {
 return y => x + y;
};
const add5 = createAddFct(5);
add5(10) // 15
```

- add5 = y => x + y
- When called, add5 will remember that x is 5
- Add5 = y => 5 + y

Function composition

Function Composition

- Process of passing the return value of one function as an argument to another function.

Example:

```
const double = n => n * 2;
const square = n => Math.sqrt(n);
double(square(49); // 14
```

Function Composition

- Difficult to map our head around if we have a lot of functions to compose.
- We're going to pipe them! (with Reduce or pipe of Ramda.js

```
const doubleSquare = pipe(
 square,
 double
);

doubleSquare(49)
```

Currying

Currying (or Partial Application)

- A function with multiple arguments is transformed into sequence of function with single argument.

- Calling a function f(a, b, c) would be translated into f(a)(b)(c)

4.

FP JS Libraries

FP JS Libraries

- ✖ Underscore.js
- ✖ Lodash
- ✖ Ramdajs
- ✖ Immutable.js

Functional Programming

- ✖ Improves the readability of your code
- ✖ Makes your code more bullet proof
- ✖ Produce more modular and reusable code
- ✖ Testing and debugging easier

Thanks!

Any questions?

You can find me at:

- twitter.com/DCTremblay
- in/dominictremblay/