香港中文大學
The Chinese University of Hong Kong

FUNCTIONAL PROGRAMMING AND JAVASCRIPT

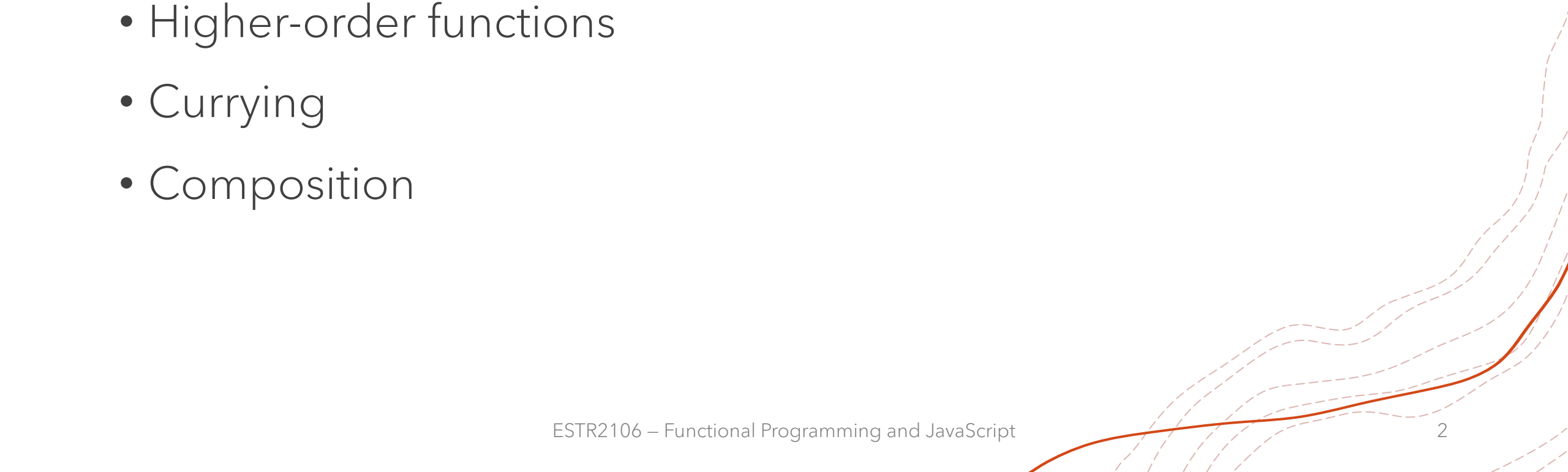
ESTR2106 2022-23 Term 1

Building Web Applications

Dr. Chuck-jee Chau
chuckjee@cse.cuhk.edu.hk

OUTLINE

- Functional Programming
- Pure functions
- Higher-order functions
- Currying
- Composition



FUNCTIONAL PROGRAMMING

- Programming paradigms: classification of languages
  - Imperative programming: what to do
    - Procedural
    - Object-oriented
  - Declarative programming: how to do
    - Functional
- Languages can be a mix of anything above, like JavaScript

FUNCTIONAL PROGRAMMING

- JavaScript is multi-paradigm, but was designed to handle first-class functions
  - ***First-class citizens*** in a language work the same way as variables
    - Assign a function into a variable
    - Pass a function as function arguments
    - Return a function from another function
    - Include function in different data structures

```
let f = function(f2) {
    f2;
    let f3 = f2;
    return f3;
}
```

- *See: https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function*

PURE FUNCTIONS

- In functional programming, functions must be pure
  - Fixed output for fixed inputs
  - No side effects (e.g., screen output)
  - No data mutation
  - Only expressions and declarations

MAP() IN JAVASCRIPT

- A number of array functions can transform array values into a new array when given a transform function, e.g., `map()`

```
let x = [1,2,3,4,5];
let transform = n => n*2;
let y = x.map(transform);
console.log(y); // [2,4,6,8,10]
```

- In the example, transform() is a new function which takes a value as input, and return double of the value
  - The result solely depends on the input
  - No screen output, no variable mutation
- More array functions (not all of the are functional): *https://www.w3schools.com/jsref/jsref_obj_array.asp*

HIGHER-ORDER FUNCTIONS

- `map()`, `filter()`, … are examples of higher-order functions
  - A function accepting another function as arguments
  - Or a function is returned

```
let cout = data => console.log(data);
let aout = data => window.alert(data);
let dout = data => document.write(data);
let decide = num => {
    if (num%2==0) return cout;
    else if (num%2==1) return aout;
    else return dout;
}
decide(5)(10);
```

CURRYING

- It is possible to break a function taking multiple arguments, e.g., add(a,b), into higher-order functions
  - Certain arguments can be reused later

```
let add = (a,b) => a + b;
add(3,4); // 7
let add2 = a => {
  return b => {
    return a + b;
  };
};
add2(3)(4); // 7
```

- Shorter (elegant) syntax
  - e.g., `let add3 = a => b => a+b;`

READ FURTHER…

Currying

https://javascript.info/currying-partials