

# CIS 658 Web Architectures

## TypeScript IV



GRAND VALLEY  
STATE UNIVERSITY®

Lecturer: **Dr. Yong Zhuang**

# Recall

- TypeScript Functions (& Lambdas):
  - Named, Anonymous, Lambda(Fat Arrow) Function
  - Single-line return contraction
- High-Order Function and callback function (Functions as Arguments)
  - `Array.sort()`
  - `Array.reduce()`

**Practice**

**Answer**

# Reduce: Array of objects

```
type River = {  
  name: string;  
  countries: Array<string>; // the river passes thru these countries  
  lenInMiles: number; // river length in miles  
};
```

```
const waters: Array<River> = [  
  {  
    name: "Amazon",  
    countries: ["Brazil", "Columbia", "Peru"],  
    lenInMiles: 4132,  
  },  
  { name: "Nile", countries: ["Egypt"], lenInMiles: 4388 },  
  { name: "Mississippi", countries: ["US"], lenInMiles: 2340 },  
  {  
    name: "Mekong",  
    countries: ["China", "Myanmar", "Laos", "Thailand", "Vietnam"],  
    lenInMiles: 2703,  
  },  
  { name: "Ganges", countries: ["India", "Bangladesh"], lenInMiles: 1560 },  
];
```

# The name of the longest river?

```
type River = {  
  name: string;  
  countries: Array<string>;  
  lenInMiles: number;  
};
```

**Practice**

# The name of the longest river?

```
type River = {  
  name: string;  
  countries: Array<string>;  
  lenInMiles: number;  
};
```

```
function lengthCompare(prev: River, curr: River): River {  
  if (prev.lenInMiles > curr.lenInMiles) return prev;  
  else return curr;  
}  
  
let winner: River;  
winner = waters.reduce(lengthCompare);  
console.log(winner.name);
```

```
let winner: River;  
winner = waters.reduce((prev: River, curr: River): River => {  
  if (prev.lenInMiles > curr.lenInMiles) return prev;  
  else return curr;  
});  
console.log(winner.name);
```

**Practice**

# What is the length of the longest river (in miles)?

```
type River = {  
  name: string;  
  countries: Array<string>;  
  lenInMiles: number;  
};
```

**Practice**

# What is the length of the longest river (in miles)?

```
type River = {  
  name: string;  
  countries: Array<string>;  
  lenInMiles: number;  
};
```

```
function compLength(prev: River, curr: River): River {  
  if (prev.lenInMiles > curr.lenInMiles) return prev;  
  else return curr;  
}  
let winner: River;  
winner = waters.reduce(compLength);  
console.log("Longest mile is", winner.lenInMiles);
```

```
function compRivLen(prev: number, curr: River): number {  
  if (prev > curr.lenInMiles) return prev;  
  else return curr.lenInMiles;  
}  
let winner: number;  
winner = waters.reduce(compRivLen, Number.MIN_VALUE);  
console.log("Longest mile is", winner);
```

**Practice**

# Which river flows through the most countries?

```
type River = {  
  name: string;  
  countries: Array<string>;  
  lenInMiles: number;  
};
```

**Practice**

# Which river flows through the most countries?

```
type River = {  
  name: string;  
  countries: Array<string>;  
  lenInMiles: number;  
};
```

```
function countryCompare(prev: River, curr: River): River {  
  if (prev.countries.length > curr.countries.length) return prev;  
  else return curr;  
}  
let winner: River;  
winner = waters.reduce(countryCompare);  
console.log(winner.name);
```

```
let winner: River;  
winner = waters.reduce((prev: River, curr: River): River => {  
  if (prev.countries.length > curr.countries.length) return prev;  
  else return curr;  
});  
console.log(winner.name);
```

Practice

# .reduce() vs .reduceRight()

## Build a string of river names with reduce

```
const riverNames1 = waters.reduce((acc, river) => {  
  return acc + " -> " + river.name;  
}, "");  
  
console.log(riverNames1);
```

**.reduce()**

Run the sample code **here** to Compare the outputs of .reduce() and .reduceRight()



What is the difference between reduce and reduceRight?

```
const riverNames2 = waters.reduceRight((acc, river) => {  
  return acc + " -> " + river.name;  
}, "");  
  
console.log(riverNames2);
```

**.reduceRight()**

# More Array high-order functions

- `Array.every()`, `Array.some()`
- `Array.find()`, `Array.findIndex()`
- `Array.filter()`, `Array.map()`, `Array.flatMap()`
- `Array.forEach()`
- ... and many others
- `flatMap()` is available in ES2019

```
// tsconfig.json {  
  "compilerOptions": {  
    "target": "ES2019",  
    // other options go here  
  }  
  ...  
}
```

## Array Operations

# Array high-order functions

```
type Shape = {  
  color: string;  
  numSides: number;  
  sideDims: Array<number>; // the length of each side  
};
```

```
let shapes: Array<Shape> = [____]
```



# Array.some(): do we have any green shape?



shapes.some (?????) → Yes

```
function isGreen(s: Shape): boolean {  
  return s.color === "green"  
}
```

```
const someGreen = shapes.some(isGreen);  
console.log(someGreen); // true
```

- Purpose: Test if at least one element in the array passes the test implemented by the provided function.
- Return value: **A Boolean** (true if at least one passes the test, otherwise false).

```
const someGreen = shapes.some(function (s: Shape): boolean {  
  return s.color === "green";  
});  
// Anonymous function
```

```
const someGreen = shapes.some((s: Shape): boolean => {  
  return s.color === "green";  
});  
// Anonymous fat arrow
```

```
const someGreen = shapes.some((s: Shape): boolean => s.color === "green");  
// 1 line return elimination
```

```
const someGreen = shapes.some((s) => s.color === "green");  
// No explicit type
```

# Array.some(): do we have any green shape?



shapes.some (?????) → Yes

**Incorrect!!!**



```
function isGreen(col: string): boolean {  
  return col === "green";  
}
```

```
const someGreen = shapes.some(isGreen);  
console.log(someGreen); // true
```

***// isGreen must take a Shape as its input parameter  
// NOT a string!!!***

# Array.every(): are all shapes triangle?



```
function isTriangle(s: Shape): boolean {  
  return s.numSides === 3;  
}
```

```
const allTriangle = shapes.every(isTriangle);  
console.log(allTriangle); // false
```

- Purpose: Tests whether all elements in the array pass the test implemented by the provided function.
- Return value: **A Boolean** (true if every element passes the test, otherwise false).

shapes.every (?????) → No

```
const allTriangle = shapes.every(function (s: Shape): boolean {  
  return s.numSides === 3; // Anonymous function  
});
```

```
const allTriangle = shapes.every((s: Shape): boolean => {  
  return s.numSides === 3; // Anonymous fat arrow  
});
```

```
const allTriangle = shapes.every((s: Shape): boolean => s.numSides === 3);  
// 1 line return elimination
```

```
const allTriangle = shapes.every((s) => s.numSides === 3);  
// No explicit type
```

# Array.forEach(): inspect all shapes



```
function printShape(s: Shape): void {  
  console.log("# of sides", s.numSides);  
}
```

```
shapes.forEach(printShape);
```

shapes.forEach (?????)

- Purpose: Executes a provided function once for each array element.
- Return value: **undefined**.

```
shapes.forEach(function (s: Shape): void {  
  console.log("# of sides", s.numSides);  
}); // Anonymous function
```

```
shapes.forEach((s: Shape): void => {  
  console.log("# of sides", s.numSides);  
}); // Anonymous fat arrow
```

```
shapes.forEach((s) => {  
  console.log("# of sides", s.numSides);  
}); // No explicit type
```

# Array.findIndex(): where is ...?



```
function isTriangle(s: Shape): boolean {  
  return s.numSides === 3;  
}
```

```
const idxTriangle = shapes.findIndex(isTriangle);  
console.log(idxTriangle); // 1
```

- Purpose: To find the index of the **first element** in the array that satisfies a provided testing function.
- Return value: the index of the **first element** in the array that passes the test. If **no elements** pass the test, it returns **-1**.

shapes.findIndex (a test)

```
const idxTriangle = shapes.findIndex(function (s: Shape): boolean {  
  return s.numSides === 3;  
});  
  
const idxTriangle = shapes.findIndex((s: Shape): boolean => {  
  return s.numSides === 3;  
});  
  
const idxTriangle = shapes.findIndex((s: Shape): boolean => s.numSides === 3);  
  
const idxTriangle = shapes.findIndex((s) => s.numSides === 3);
```

# Array `.find()` Family of Functions

- If you need the actual element that satisfies a condition in the array, use `find()`.
- If you need the index of the found element in the array that satisfies a condition, use `findIndex()`.
- If you need to find the index of a specific value in the array, use `indexOf()`. (It's similar to `findIndex()`, but checks each element for equality with the value instead of using a testing function.)
- If you need to determine whether an array includes a specific value, use `includes()`. Again, it checks each element for equality with the value instead of using a testing function.
- If you need to find if any element satisfies the provided testing function, use `some()`.

# Array.filter(): give me only green shapes



```
function isGreen(x: Shape): boolean {  
  return x.color === "green";  
}  
const greenOnly: Shape[] = shapes.filter(isGreen);
```

shapes.filter(...)



- *Purpose: creates a new array with all elements that pass the test implemented by the provided function.*
- *Return value: a new array with the elements that pass the test. If no elements pass the test, an empty array will be returned.*

```
const greenOnly = shapes.filter((shp: Shape): boolean => {  
  return shp.color === "green";  
});  
  
const greenOnly = shapes.filter((x) => x.color === "green");
```

# Array.map(): extract all colors/num sides



`shapes.map(--color--)`

`shapes.map(--numSides--)`

6-item array

array of strings

array of numbers

```
["yellow", "green", "purple", "red", "green", "orange"]
```

```
[4, 3, 4, 5, 4, 3]
```

```
let colors: string[];
```

```
colors = shapes.map((x: Shape) => {  
  return x.color;  
});
```

```
//or  
colors = shapes.map((x) => x.color);
```

```
let sides: number[];
```

```
sides = shapes.map((x: Shape) => {  
  return x.numSides;  
});
```

```
//or  
sides = shapes.map((x) => x.numSides);
```

# Array.filter() & Array.map()

## .filter()

```
const numbers = [2, -30, 0, 17, 9, -11];  
function isPos(x: number): boolean {  
  return x > 0;  
}  
const out = numbers.filter(isPos);  
console.log(out); // [2, 17, 19]
```

```
const numbers = [2, -30, 0, 17, 9, -11];  
const out = numbers.filter((x: number) => {  
  return x > 0;  
});  
console.log(out); // [2, 17, 19]
```

*To create a new array with elements that pass a condition (or test) from the original array.*

## .map()

```
const numbers = [2, -30, 0, 17, 9, -11];  
function plus10(x: number): number {  
  return x + 10;  
}  
const out = numbers.map(plus10);  
console.log(out); // [12, -20, 10, 27, 19, -1]
```

```
const numbers = [2, -30, 0, 17, 9, -11];  
const out = numbers.map((x: number) => {  
  return x + 10;  
});  
console.log(out); // [12, -20, 10, 27, 19, -1]
```

*To create a new array by transforming each element in the original array.*

# Chaining multiple Array functions



`map(--color--)`

```
["yellow", "green", "purple", "red", "green", "orange"]
```

`filter(c === "green")`

```
["green", "green"]
```

```
const someBlue = shapes.map((x) => x.color).filter((c) => c === "green");
```

# Array.flatMap(): from one to many



`shapes.flatMap(--sideDims--)`

*length = 13*

```
[7, 7, 7, 7, 8, 11, 8, 11, 6.1, 6.1, 6.1, 6.1, 6.1]
```

`shapes.map(--sideDims--)`

*length = 3*

```
[  
  [7, 7, 7, 7],  
  [8, 11, 8, 11],  
  [6.1, 6.1, 6.1, 6.1, 6.1],  
]
```

```
let stats: number[];  
stats = shapes.flatMap((s: Shape): number[] => {  
  return s.sideDims;  
});  
stats = shapes.flatMap((s: Shape): number[] => s.sideDims);
```

```
let stats: number[][];  
stats = shapes.map((s: Shape): number[] => {  
  return s.sideDims;  
});  
stats = shapes.map((s: Shape): number[] => s.sideDims);
```

# Practical Use Case of flatmap()

```
type Course = {
  name: string;
  credits: number;
  classList: Array<string>;
};
let allCourses: Array<Course> = [
  {
    name: "MTH101 Calculus",
    credits: 4,
    classList: [
      /* 25 student names */
    ],
  },
  {
    name: "HTM 203 Beer Brewing",
    credits: 2,
    classList: [
      /* 70 student names */
    ],
  },
];
```

*Find all students whose name begins with "Eli"*



```
const studentList = allCourses
  .flatMap((c: Course) => {
    return c.classList;
  })
  // you'll get 95 names from flatMap
  .filter((who: string) => {
    return who.startsWith("Eli");
  });
```

```
const studentList = allCourses
  .flatMap((c: Course) => c.classList)
  // you'll get 95 names from flatMap
  .filter((who: string) => who.startsWith("Eli"));
```

# Some examples

## How many green shapes?

```
shapes.filter((s) => s.color === "green").length;
```

## How many equilateral triangles?

```
shapes.filter(  
  (s) =>  
    s.numSides === 3 &&  
    s.sideDims[0] === s.sideDims[1] &&  
    s.sideDims[1] === s.sideDims[2]  
).length;
```

```
shapes  
  .filter((s) => s.numSides === 3)  
  .filter(  
    (s) => s.sideDims[0] === s.sideDims[1] && s.sideDims[1] === s.sideDims[2]  
  ).length;
```

## Largest perimeter?

```
shapes  
  .map((shp) => {  
    let perimeter = 0;  
    // Compute perimeter  
    return perimeter;  
  })  
  .reduce((acc: number, curr: number) => {  
    if (acc > curr) return acc;  
    else return curr;  
  });
```

# Practice