CIS 371 Web Application Programming TypeScript IV



Lecturer: Dr. Yong Zhuang

Recall

- Type Alias vs. Interface vs. Class, Inheritance
- TypeScript Functions (& Lambdas):
 - Named, Anonymous, Lambda(Fat Arrow) Function
 - Single-line return contraction
- Functions as Arguments
 - Array.sort()

Array high-order functions

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

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

Array Operations



Array high-order functions

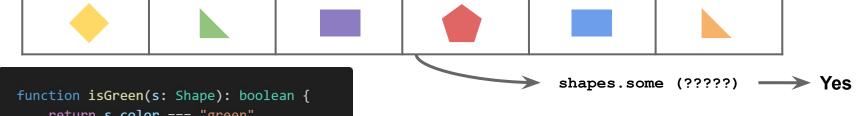
```
type Shape = {
  color: string;
  numSides: number;
  sideDims: Array<number>; // the length of each side
};
```

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





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



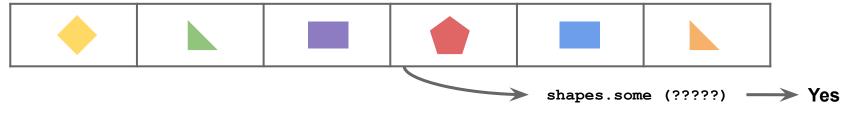
```
return s.color === "green"
}

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).



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



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).

Array.forEach(): inspect all shapes

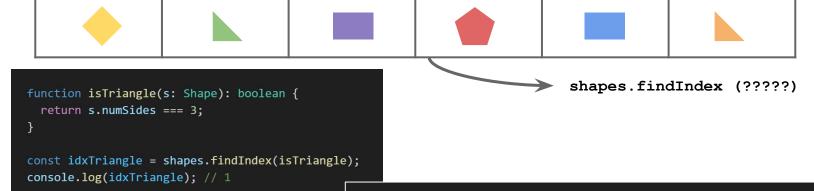


```
function printShape(s: Shape): void {
  console.log("# of sides", s.numSides);
}
shapes.forEach(printShape);
```

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



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



- 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**.

```
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);
```

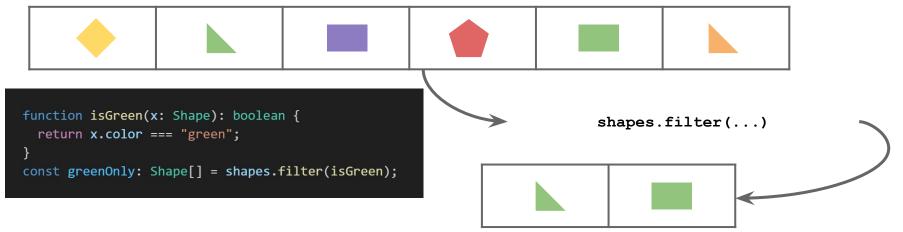


Array find() functions

- If you need the actual element that satisfies a condition in the array, use <u>find()</u>.
- If you need the index of the found element in the array that satisfies a condition, use <u>findIndex()</u>.
- 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 <u>includes()</u>. 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 <u>some()</u>.



Array.filter(): give me only green shapes

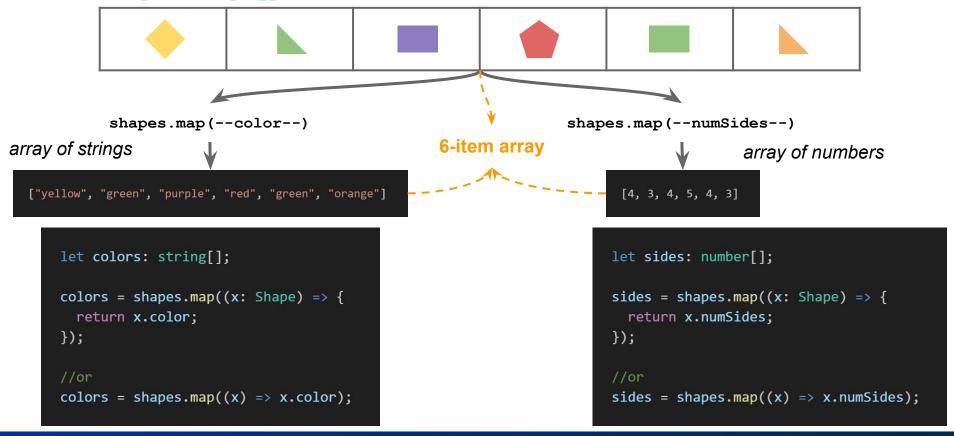


- Purpose: creates a new array with all elements that pass the test implemented by the provided function.
- Return value: a new array with the elements that pass the test. If no elements pass the test, an empty array will be returned.

```
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





Array.filter() & Array.map()

.filter() .map()

```
// Named
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]
```

```
Function
  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.filter((x: number) => {
    return x > 0;
});
console.log(out); // [2, 17, 19]
```

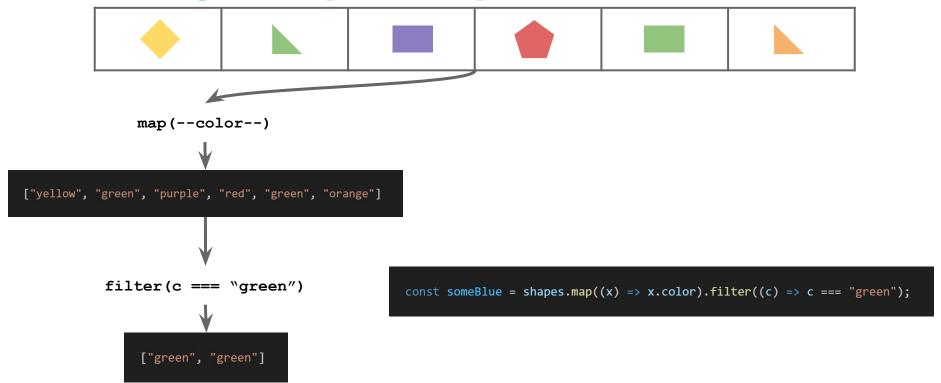
```
To create a new array with elements that pass a condition (or test) from the original array.
```

```
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.

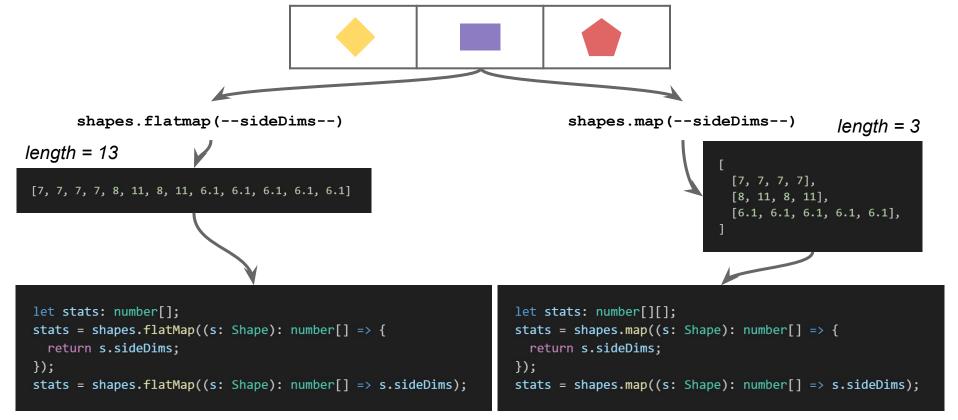


Chaining multiple Array functions





Array.flatmap(): from one to many





Practical Use Case of flatmap()

```
type Course = {
 name: string;
  credits: number;
  classList: Array<string>;
};
let allCourses: Array<Course> = [
    name: "MTH101 Calculus",
    credits: 4,
    classList: [
   ],
  },
    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");
});
```

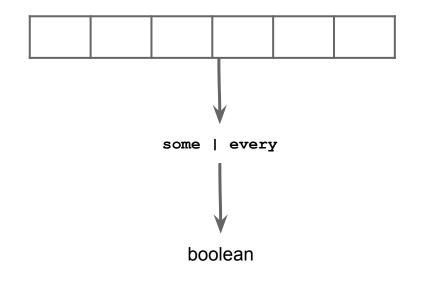


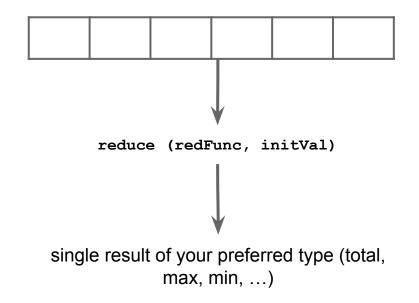
```
const studentList = allCourses
  .flatMap((c: Course) => c.classList)
  // you'll get 95 names from flatMap
  .filter((who: string) => who.startsWith("Eli"));
```



Practice

Introducing: Array.reduce()





Not limited to only boolean output!!!



Array.reduce(): sum of values

```
const scores = [23, -31, 17, 31, 19];
const computeSum = (accumulator: number, currentValue: number): number => {
  return accumulator + currentValue;
};

const totalScore = scores.reduce(computeSum);
console.log("Total ", totalScore); // Total 59
```

pos	accumulator	currentValue	return
1	23	-31	-8
2	-8	17	9
3	9	31	40
4	40	19	59

- Acc is initialized from the first array item
- Work begins at position 1



Array.reduce(): sum of values (with initial value)

```
const scores = [23, -31, 17, 31, 19];
const computeSum = (accumulator: number, currentValue: number): number => {
   return accumulator + currentValue;
};

const totalScore = scores.reduce(computeSum, 2000);
console.log("Total ", totalScore); // Total 2059
```

pos	accumulator	currentValue	return
0	2000	23	2023
1	2023	-31	1992
2	1992	17	2009
3	2009	31	2040
4	2040	19	2059

- Acc is initialized from the initial value
- Work begins at position 0



Array.reduce(): shortest river name(with initial value)

```
const rivers = ["Amazon", "Mississippi", "Nile", "YangTze", "Yenisei"];
const shorterOf = (accumulator: string, currentValue: string): string => {
  if (currentValue.length < accumulator.length) return currentValue;
  else return accumulator;
};

const riverName = rivers.reduce(shorterOf, "Yellow");
console.log("Shortest ", riverName); // Nile</pre>
```

pos	accumulator	currentValue	return
0	Yellow	Amazon	Yellow
1	Yellow	Mississippi	Yellow
2	Yellow	Nile	Nile
3	Nile	YangTze	Nile
4	Nile	Yenisei	Nile

- Acc is initialized from the provided value
- Work begins at position 0



Array.reduce(): shortest river name(with initial value)

```
const rivers = ["Amazon", "Mississippi", "Nile", "YangTze", "Yenisei"];
const shorterOf = (accumulator: string, currentValue: string): string => {
  if (currentValue.length < accumulator.length) return currentValue;
  else return accumulator;
};

const riverName = rivers.reduce(shorterOf, "Roe");
console.log("Shortest ", riverName); // ?</pre>
```

pos	accumulator	currentValue	return
0	Roe	Amazon	Roe
1	Roe	Mississippi	Roe
2	Roe	Nile	Roe
3	Roe	YangTze	Roe
4	Roe	Yenisei	Roe





Array.reduce() with initial value

```
const rivers = ["Amazon", "Mississippi", "Nile", "YangTze", "Yenisei"];
const shorterLen = (accumulator: number, currentValue: string): number => {
  if (currentValue.length < accumulator) return currentValue.length;
  else return accumulator;
};
// Use 37 to initialize riverLen
const riverLen = rivers.reduce(shorterLen, 37);
console.log("Shortest ", riverLen); // 4</pre>
```

pos	accumulator(num)	currentValue(str)	return(num)
0	37	Amazon	6
1	6	Mississippi	6
2	6	Nile	4
3	4	YangTze	4
4	4	Yenisei	4

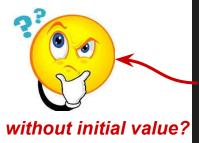
- Type of acc and curr may be different
- Type of acc and type of initial value must match
- Type of acc determines the type of return



Array.reduce()

let myArray: Array<XYZ>;

```
function myFunction(prev: XYZ, curr: XYZ): XYZ {
    // More code here
    return ____;
}
const result: XYZ = myArray.reduce(myFunction);
```



```
function myFunction(prev: resultType, curr: XYZ): resultType {
   // More code here
   return ____;
}
const initValue: resultType = ____;
const result: resultType = myArray.reduce(myFunction, initValue);
```



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;
};
```

```
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);
```

```
Practice
```

```
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);
```



The longest mile?

```
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



Goes through most countries?

let winner: River;

```
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);
```

```
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



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;
});
```