CIS 371 Web Application Programming

TypeScript III

GRAND VALLEY STATE UNIVERSITY®

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Recall

- Objects: Typeless, Typed, Sub-Objects, For-in loop to enumerate object
- Array of Objects: Typeless, Typed
- Spreading:
  - Array, Array Destructuring,
  - Object, with duplicate props, copy and modify object
- Optional Chaining (?) operator & Function Optional Parameters
- Coalesce operator (??) & non-null assertion operator (!)
- Logical OR (||) operator
- Enum vs. Literal Types
- String Interpolation
- ES6 key/value Shortcut

TypeScript Functions (& Lambdas)

Three variations of Function Declarations

```
function plus2 (a:number, b:number): number {
    return a + b;
}
```

named

```
const plus2 = function (a:number, b:number): number {
    return a + b;
}
```

anonymous func

```
const plus2 = (a:number, b:number) : number => {
    return a + b;
}
```

lambda function

Any of these function declarations can be invoked using ONE syntax:

```
let out:number;
out = plus2(5.0, 2.9);
```

Vars of "function" type

typeless AND 1-line return contraction

```
const plus2 = (a, b) => a + b
```

Fat Arrow fns: single-line return contraction

```
const plusTwo = (a:number, b:number) : number => {
    const sum = a + b;
    return sum;
}
```

no ‘function’ keyword.

```
const plusTwo = (a:number, b:number) : number => {
    return a + b;
}
```

If ‘return’ can be the only statement

omit both the curly braces { } and the ‘return’ keyword.

```
const plusTwo = (a:number, b:number) : number => a + b;
const plusTwo = (a,b) => a + b;    // typeless
```

implicit return

Variables of func type

plus20 and plus22 are variables that hold your DATA

```
const plus20 = "+20";
const plus22 = { positive: true, value: 22 }
```

```
const plus2 = function (a:number, b:number): number {
    return a + b;
}
const plusTwo = (a:number, b:number) : number => {
    return a + b;
}
```

plus2 and plusTwo are variables that hold your ***CODE***

```
console.log(typeof plus20); // string
console.log(typeof plus22); // object
console.log(typeof plus2);   // function
console.log(typeof plusTwo); // function
```

Important Takeaway Concept

- *Assigned to a variable*
- *Passed as an argument to another function*
- *Returned as a value from other functions*

$\xrightarrow{}$ **High-Order Functions**

JS & TS allow variables of type Function

JS & TS variables can hold either data or code

- *JS & TS variables can be assigned typical data values like numbers, strings, and objects,*
- *or they can be assigned functions*

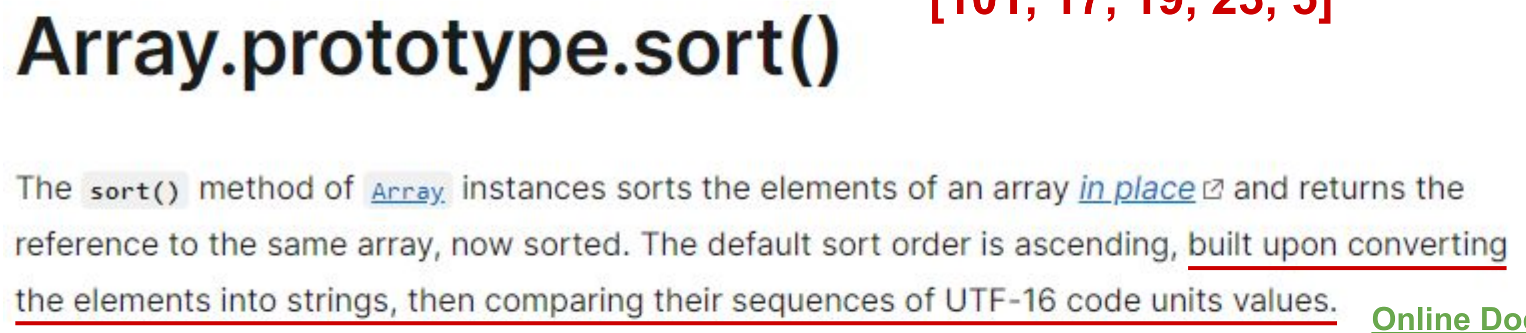
High-Order Function and callback function (Functions as Arguments to another Function)

Array.sort()

```
const atoms = ["Neon", "Iron", "Calcium", "Hydrogen"]
console.log(atoms.sort())
// ["Calcium", "Hydrogen", "Iron", "Neon"]
```

```
const primes = [23, 17, 5, 101, 19]
const sorted_nums = primes.sort()
console.log(sorted_nums)
```

[101, 17, 19, 23, 5]

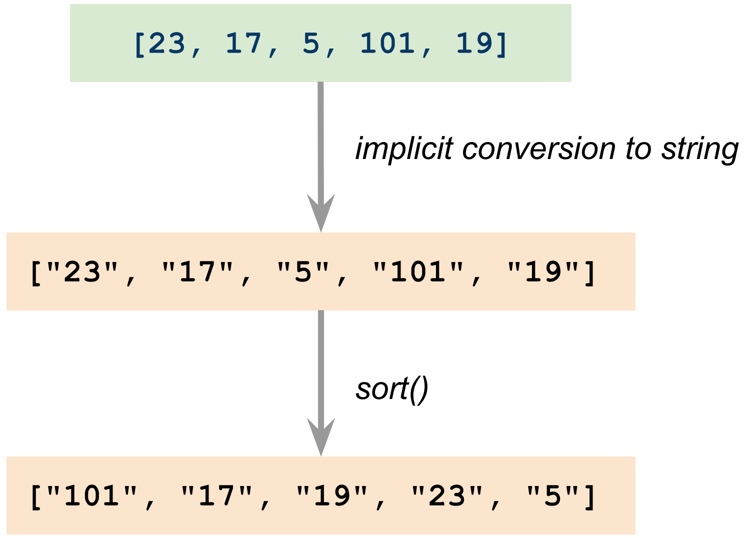


Array.prototype.sort()

The `sort()` method of `Array` instances sorts the elements of an array *in place* and returns the reference to the same array, now sorted. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

Online Doc

Array.sort() built in behavior



```
[23, 17, 5, 101, 19]
```

```
["23", "17", "5", "101", "19"]
```

```
["101", "17", "19", "23", "5"]
```

To fix this ***"bug"***, we have to tell sort() the **collating order between two data items**

Array.sort() with collating order

```
function numericOrder(a:number, b:number): number {
    if (a < b) return -1;               // any negative number
    else if (a > b) return +1;          // any positive number
    else return 0;
}
```

```
const primes = [23, 17, 5, 101, 19]
const sorted_nums = primes.sort(numericOrder)
console.log(sorted_nums) // [5, 17, 19, 23, 101] Ok
```

The collating function must return a ***number***

- *Negative when the “first” item should be placed BEFORE the “second” item*
- *Positive when the “first” item should be placed AFTER the “second” item*
- *Zero when the order of the two items is irrelevant*

Array.sort() on objects

```
type Language = {
    name: string;  yearCreated: number
}
const langs: Language[] = [
    { name: "C", yearCreated: 1970},
    { name: "JavaScript", yearCreated: 1995},
    { name: "Fortran", yearCreated: 1954}
]
function orderByName(a:Language, b:Language): number {
    return a.name.localeCompare(b.name)
}
function orderByYear(a:Language, b:Language): number {
    return a.yearCreated - b.yearCreated
}
langs.sort(orderByYear)
```

ascending or descending?

- *Negative when the referenceStr occurs before compareString*
- *Positive when the referenceStr occurs after compareString*
- *Returns 0 if they are equivalent*

*The collating function takes two parameters of **type Language** but must **return a number***

Array.sort() on objects

```
type Language = {
    name: string;  yearCreated: number
}
const langs: Language[] = [
    { name: "C", yearCreated: 1970},
    { name: "JavaScript", yearCreated: 1995},
    { name: "Fortran", yearCreated: 1954}
]
```

```
function orderByName(a:Language, b:Language): number {
    return a.name.localeCompare(b.name)
}
langs.sort(orderByName)
```

Option 1: named function

```
langs.sort(
    function (a:Language, b:Language): number {
        return a.name.localeCompare(b.name)
    }
)
```

Option 2: unnamed function

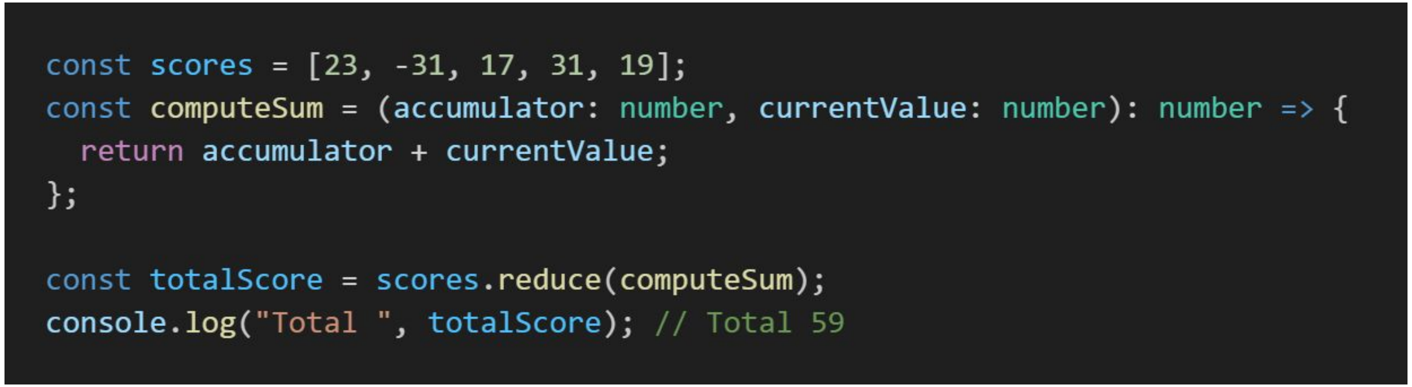
```
langs.sort(
    (a:Language, b:Language): number => {
        return a.name.localeCompare(b.name)
    }
)
```

Option 3: lambda function

```
langs.sort(
    (a, b) => a.name.localeCompare(b.name)
)
```

Opt 4: typeless lambda & 1-line return contraction

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)

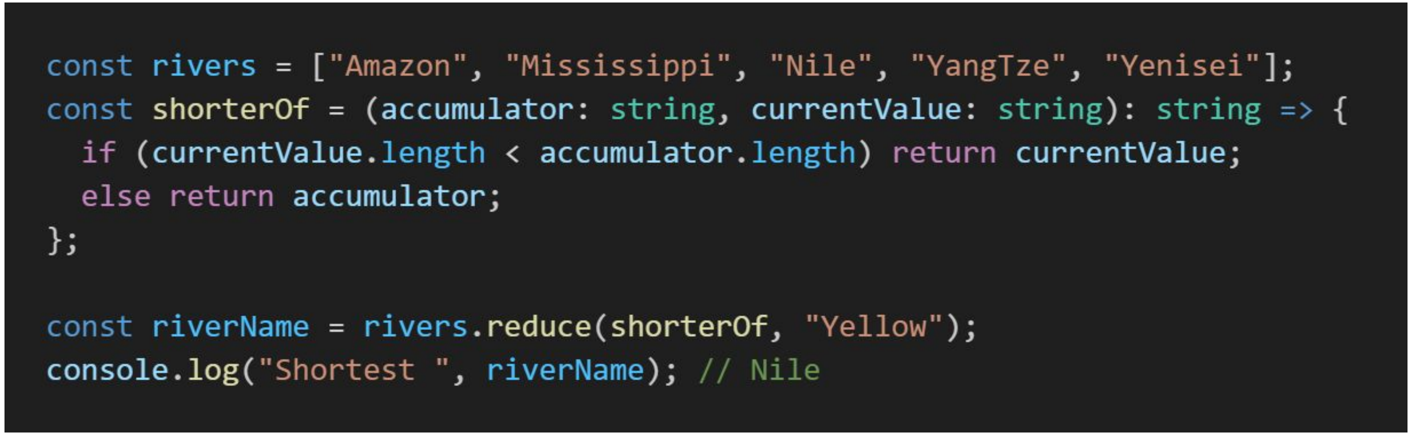
```
const scores = [23, -31, 17, 31, 19];
const computeSum = (accumulator: number, currentValue: number): number => {
  return accumulator + currentValue;
};

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)



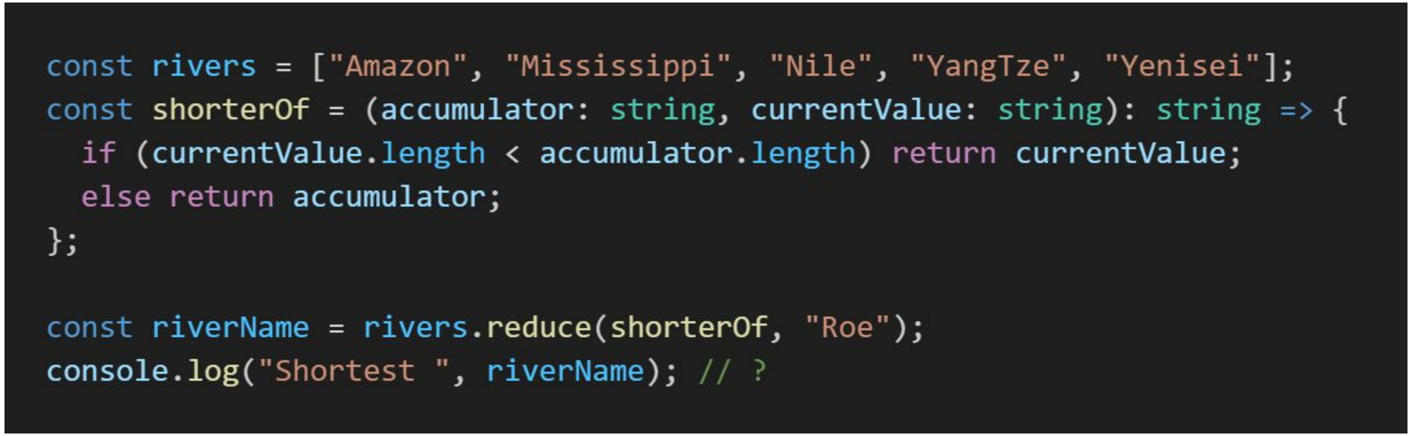
```
const rivers = ["Amazon", "Mississippi", "Nile", "YangTze", "Yenisei"];
const shorterOf = (accumulator: string, currentValue: string): string => {
  if (currentValue.length < accumulator.length) return currentValue;
  else return accumulator;
};

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

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)

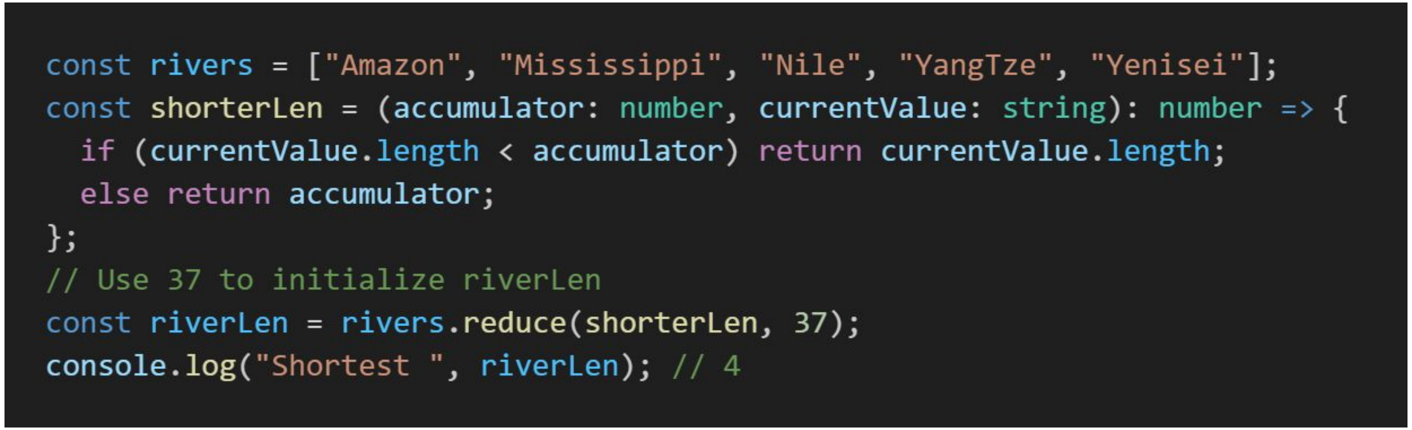


```
const rivers = ["Amazon", "Mississippi", "Nile", "YangTze", "Yenisei"];
const shorterOf = (accumulator: string, currentValue: string): string => {
  if (currentValue.length < accumulator.length) return currentValue;
  else return accumulator;
};

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

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
```

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);
```

without initial value?

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

Practice