CIS 371 Web Application Programming

TypeScript II

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Recall

- JavaScript and TypeScript,
- Initial Setup For Node JS,
- Function vs. Method,
- Data Types,
- multiple types,
- == vs ===,
- Arrays,
  - for-in vs. for-of,
  - push() and .pop(),
  - creates a copy using slice(),
  - delete/replaces elements splice(),

Objects

Java Classes and Objects

```
// Java objects must be instantiated from a class
// In Sub.java
class Sub() {
  public String name;
  public int calorie;
}

// In AnotherFile.java
Sub my_order = new Sub();
my_order.name = "Spicy Turkey";
my_order.calorie = 182;

my_order.price = 3.17;  // ERROR!
```

vs.

TS Objects

```
// TypeScript (no class needed)
const my_order = {
  name: "Spicy Turkey",
  calorie: 182
}
```

Objects can be created without a class definition

Objects in TypeScript

```
// Typeless objects
const in_a_month = {
  name: "September",
  days: 30
}

const employee_vacation = {
  name: "Bob",  days: 11
}
```

```
// Typed objects
type Monthly = {
  name: string,
  days: number
}
const in_a_month:Monthly {
  name: "September",
  days: 30
}
```

```
type VacationDays = {
  name: string,
  days: number
}
const employee_vacation:VacationDays = {
  name: "Bob",
  days: 11
}
```

Objects with Sub-Objects & Array property

```
type City = {
  name: string,
  population: number,
  geopos: {
    lat: number,
    lon: number
  },
  univs: Array<string>
}
```

```
const ours:City = {
  name: "Grand Rapids",
  population: 198400,
  geopos: {
    lat: 42.9633599,
    lon: -85.6680863
  },
  univs:  [
    "Calvin", "Cornerstone",
    "GVSU"
  ]
}
```

```
const theirs:City = {
  name: "East Lansing",
  population: 48729,
  geopos: {
    lat: 42.737652,
    lon: -84.483788
  },
  univs:  [
    "MSU",
  ]
}
```

```
console.log(ours.name);
for (let u of ours.univs) console.log(u);

console.log(theirs.geopos.lat);
```

```
Grand Rapids
Calvin
Cornerstone
GVSU
42.737652
```

for-in to enumerate object properties

```
const theirs:City = {
  name: "East Lansing",
  population: 48729,
  geopos: {
    lat: 42.737652,
    lon: -84.483788
  },
  univs:  [
    "MSU",
  ]
}
```

```
for (let z in theirs) {
  console.debug(z)
}
```

```
name
population
geopos
univs
```

```
for (let z in theirs) {
  console.debug(z, theirs[z]);
 }                    ^-------^ ERROR
```

```
const eLan = theirs as any;
for (let z in theirs) {
  console.debug(z, "==>", eLan[z])
}
```

```
name ==> East Lansing
population ==> 48729
geopos ==> {lat: 42..., lon: -84..}
univs == > ["MSU"]
```

Array of Objects

```
// In Atom.java
class Atom {
  public String name;
  public weight double;
}

// In AnotherFile.java
ArrayList<Atom> atoms = new ArrayList<>();
Atom a = new Atom("Carbon", 12);
atoms.add(a);
Atom b = new Atom("Oxygen", 16);
atoms.add(b);
atoms.add(new Atom("Natrium", 23);
```

TS: option 1

```
// TypeScript (no class required)
 const atoms = [];
 atoms.push({ name: "Carbon", weight: 12});
 atoms.push({ name: "Oxygen", weight: 16});
 atoms.push({ name: "Natrium", weight: 23});
```

TS: option 2

```
// Or initialize the array
const atoms = [
  { name: "Carbon", weight: 12},
  { name: "Oxygen", weight: 16},
  { name: "Natrium", weight: 23}
];
```

Array of Typed Objects

Typeless array

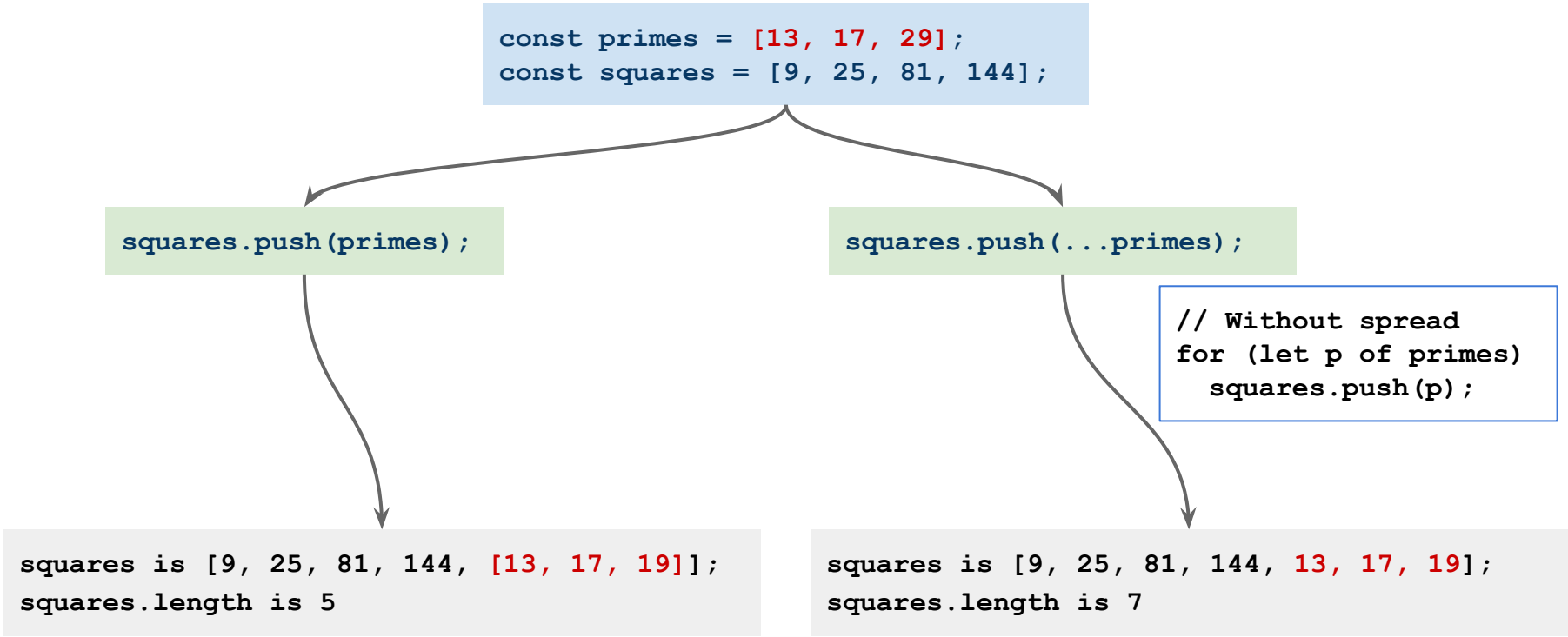
```
const atoms = [];
atoms.push({ name: "Carbon", weight: 12});
atoms.push({ namme: "Fluor", weight: 12}); // OK
atoms.push({ name: "Oxygen"}); // OK
atoms.push({ name: "Natrium", weight: 23, isMetal: false}); // OK
```

```
// Declare a type
type Atom = {
  name: string,
  weight: number
}
```

Typed array

```
const atoms:Array<Atom> = [];
 atoms.push({ name: "Carbon", weight: 12});
 atoms.push({ namme: "Fluor", weight: 12});        // ERROR: "namme" does not exist
 atoms.push({ name: "Oxygen"});                    // ERROR: property "weight" is missing
 atoms.push({
   name: "Natrium",
   weight: 23,
   isMetal: false});                               // ERROR: "isMetal" does not exist
```

Spreading an Array



Spreading an Object

```
const name = { first: "Bob", last: "Dylan"};
const job = { position: "Web Developer", salary: 75000};
```

```
const one = {name, job};
```

```
{
  name: {
    first: "Bob",
    last: "Dylan"
  },
  job: {
    position: "Web Developer",
    salary: 75000
  }
}
```

```
const two = {name, ... job}
```

```
{
  name: {
    first: "Bob",
    last: "Dylan"
  },
  position: "Web Developer",
  salary: 75000
}
```

```
const three = {
   ... name,
    ... job
}
```

```
{
  first: "Bob",
  last: "Dylan",
  position: "Web Developer",
  salary: 75000
}
```

Spread on Objects (with duplicate props)

If objects have duplicate properties…

Spread on Objects (with duplicate props)

```
const prop1 = {name: "Carbon", abbrev: "Cb"}
const prop2 = {weight: 12, abbrev: "C"}
// without spread on prop1
const element = {prop1, ... prop2};
```

```
{
  prop1: {
    name: "Carbon", abbrev: "Cb"
  },
  weight: 12, abbrev: "C"
}
```

```
const prop1 = {name: "Carbon", abbrev: "Ca"}
const prop2 = {weight: 12, abbrev: "C", name: "Clue"}
// with spread
const element = {...prop1, ...prop2, isMetal: false};
const el2     = {...prop2, ...prop1, isMetal: false};
```

With spread

```
{
  isMetal: false,
  name: "Clue",
  abbrev: "C",
  weight: 12,
}
```

```
{
  isMetal: false,
  name: "Carbon",
  abbrev: "Ca",
  weight: 12,
}
```

Later values overwrite previous values of the same key

Object spread: copy and modify

```
const bob = {
  first: "Bob",
  last: "Dylan",
  position: "Web Developer",
  salary: 75000
}
```

```
const bob_now = {
  ...bob,
  workFromHome: true,
  position: "Cloud Data Egr.",
  salary: 78000
}
```

bob_now

```
{
  first: "Bob",
  last: "Dylan",
  workFromHome: true,
  position: "Cloud Data Egr.",
  salary: 78000
}
```

This won't work (no copy created).

```
const bob_now = bob;
bob_now.position = "Cloud Data Egr.";
bob_now.salary = 78000;
```

Array Destructuring

```
const nums:number[] = [1,2,3,4,5];
const [first,rest] = nums;
```

Without spread

```
// first is 1   (number)
// rest is 2    (number)
```

```
const nums:number[] = [1,2,3,4,5];
const [first, ...rest] = nums;
```

With spread

```
// first is 1 (number)
// rest is [2,3,4,5] (number[])
```

```
function splitIt([f, ...r]: number[]): void {
  console.log(f);
  console.log(r);
}

splitIt([5, 20, 31, 19]);
```

With spread on func args

```
// 5    a number
// [20, 31, 19]  an ARRAY of numbers
```

Optional Chaining (?) operator

```
type City = {
  name: string,
  population: number,
  geopos: {
    lat: number,
    lon: number
  } | null,
  univs: Array<string>
}
```

```
let newCity: City = {
  name: "East Lansing",
  population: 48729,
  geopos: null,
  univs:  [
    "MSU",
  ]
}
```

```
if (newCity.geopos){
    const lat = newCity.geopos.lat
    console.log(lat)
}
else{
    console.log("No Geo Info")
}
```

```
const lat = newCity.geopos.lat
```

null?

```
const lat = newCity.geopos? newCity.geopos.lat: "No geo info"
console.log(lat)
```

ternary operator

Optional Chaining (?) operator

```
type City = {
  name: string,
  population: number,
  geopos: {
    lat: number,
    lon: number
  } | null,
  univs: Array<string>
}
```

```
let newCity: City = {
  name: "East Lansing",
  population: 48729,
  geopos: null,
  univs:  [
    "MSU",
  ]
}
```

```
const lat = newCity.geopos.lat
```

null?

```
const lat = newCity.geopos?.lat
```

Chaining

undefined

Coalesce operator (??) & non-null assertion operator (!)

```
let aName: string | null;
```

```
const theName:string = aName? aName : "No name"
console.log(theName)
```

ternary operator

```
if (aName){
    const theName:string = aName
    console.log(theName)
}
else{
    console.log("No name")
}
```

if **aName** is **null**, **theName** will be **null** even it should not have that type.

```
const theName:string = aName
```

null?

```
const theName:string = aName ?? "no name"
```

Coalesce

```
const theName:string = aName!
```

non-null assertion

Logical OR (| |) operator

```
const aString = '';
console.log(aString ?? 'Empty Value');

const aNumber = 0;
console.log(aNumber ?? 'Zero Value');

const aBool = false;
console.log(aBool ?? 'False Value');
```

```
0
false
```

```
Empty Value
Zero Value
False Value
```

```
const aString = '';
console.log(aString || 'Empty Value');

const aNumber = 0;
console.log(aNumber || 'Zero Value');

const aBool = false;
console.log(aBool || 'False Value');
```

Enum vs. Literal Types

```
enum CollegeYear {
  Freshman,
  Sophomore,
  Junior,
  Senior
}
```

Sort order (enum order): Freshman < Sophomore < Junior < Senior

```
let yr: CollegeYear;
yr = CollegeYear.Junior;
console.debug(yr);
console.debug(CollegeYear[yr]);
```

```
2
Junior
```

```
type CollegeLiteral =
   "Freshman" |
   "Sophomore" |
   "Junior" |
   "Senior";
```

Sort order (alphabetical): "Freshman" < "Junior" < "Senior" < "Sophomore"

```
let yr: CollegeLiteral;
yr = "junior";                  // Compile error
yr = "Junior"
console.debug(yr);              // Output "Junior"
```

Literal Types: Narrowing

```
// TypeScript
let dayOfWeek: string;
dayOfWeek = "Monday";   // No error

let strictDOW: "Mon" | "Tue" | "Wed" | "Thu";
strictDOW = undefined; // Error
strictDOW = "Fri"; // Error

let dieValue: 1 | 2 | 3 | 4 | 5 | 6;
dieValue = undefined; // Error
dieValue = 0; // Error
```

- Use this for data with one a small number of valid values.
- Invalid values are detected at compile time (not at runtime)

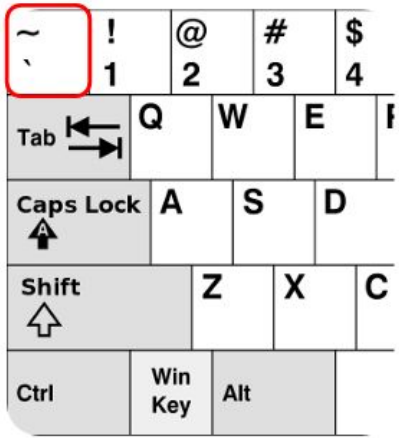
String Interpolation (backquotes)

```
`Some text here ${var} and here`
`More text ${expression} also here`
```

```
const x = "Eleven";
const arr = [3, 5, 13];

// Java-like string concatenation
let oldStore = (4 + arr[0]) + "-" + x;     // 7-Eleven

// Use backquotes string interpolation
let store = `${4 + arr[0]}-${x}`;          // 7-Eleven
```



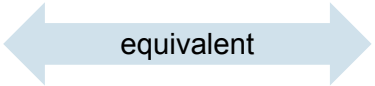
ES6 key/value Shortcut

```
let cityName = "Allendale";
let zipCode = "49401";

let location = {
  city: cityName,
  zip: zipCode;
};
```



```
let city = "Allendale";
let zip = "49401";
let location = {
  city: city,
  zip: zip;
};
```



When both key and value refer to the same name, you don't have to write them both. Only one is required

```
let city = "Allendale";
let zip = "49401";
let location = {
  city,
  zip;
};
```