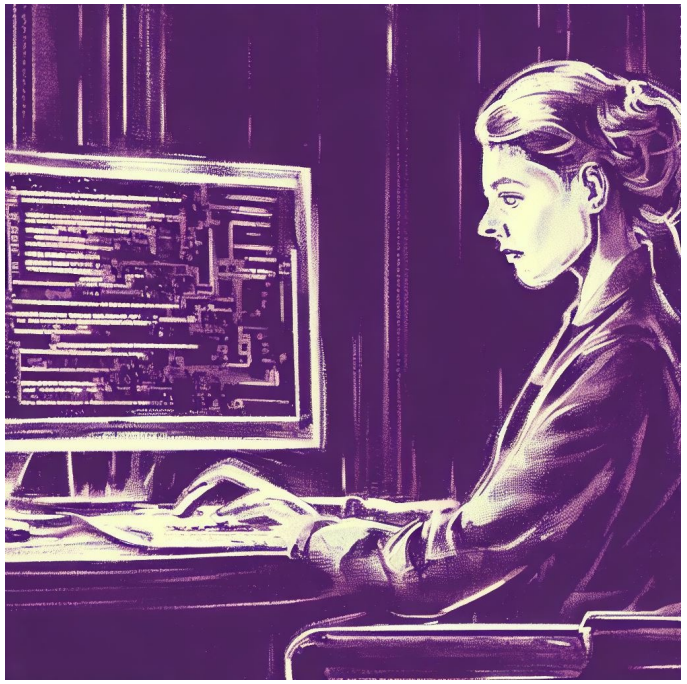

CSE 331

Intro to JavaScript & TypeScript

Kevin Zatloukal

Programming for the Browser

- Today: overview of JavaScript (JS) & TypeScript (TS)

- Both languages can be run in different environments
  - command line (like Java)
  - inside the browser

- Primarily interesting because of the browser
  - neither language would be used much otherwise
  - command line provided so you can use one language for both

- In both environments, print output with `console.log(..)`
  - prints to command line or “Developer Console” in the browser

Programming for the Browser

- JavaScript is the lingua franca of web browsers

- Previously, other languages were tried in the browser
  - Java was used but is no longer supported
  - Flash was used but is largely no longer supported
  - Google's "dart" language is still around (probably)

- Now, other languages used by compiling into JavaScript
  - TypeScript used this way
  - Java can be compiled to JS (but it's not great)

    you can't really get around needing to learn JS

Plan for This Week

- **Today**: overview of JavaScript and TypeScript
- **Tomorrow**: hands-on work in section
- No rush to learn the whole language
  - we will start with a small subset of its features
  - we won't use all the language this quarter
  - have all quarter to get more familiar with JS and TS

JavaScript

History of JavaScript

- Incredibly simple language
  - created in 10 days by Brendan Eich in 1995
  - often difficult to use because it is so simple

- Features added later to fix problem areas
  - imports
  - classes

Relationship to Java

- Initially had no relation to Java
  - picked the name because Java was popular then
  - added Java's Math library to JS also

    e.g., `Math.sqrt` is available in JS, just like Java
  - copied *some* of Java's String functions to JS string

    e.g., `s.charCodeAt(3)` is available in JS, just like Java

- Both are in the "C family" of languages
  - much of the syntax is the same
  - more differences in data types

- We will discuss syntax (code) first and then data...

JavaScript Syntax

- Both are in the “C family” of languages

- Much of the syntax is the same
  - most expressions (`+`, `-`, `*`, `/`, `?:`, function calls, etc.)
  - `if`, `for`, `while`, `break`, `continue`, `return`
  - comments with `//` or `/* .. */`

- Different syntax for a few things
  - declaring variables
  - declaring functions
  - equality (===)

Java vs JavaScript Syntax

- **The following code is legal in <u>both</u> languages:**
  - assume "`s`" and "`j`" are already declared

```
s = 0;
j = 0;
while (j < 10) {
  s += j;
  j++;
}

// Now s == 45
```

OR

```
for (j = 0; j < 10; j++)
```

Differences from Java: Type Declarations

- JavaScript variables have no declared types
  - this is a problem… we will get them back in TypeScript
- Declare variables in one of these ways:

```
const x = 1;
let y = "foo";
```

  - "`const`" cannot be changed; "`let`" can be changed
  - use "`const`" whenever possible!
- Also affects function argument declarations
  - more on this later…

Differences from Java: "===" operator

- JavaScript's "==" is problematic
  - tries to convert objects to the same type

    e.g., `3 == "3"` is true

- We will use "===" (and "!==") instead:
  - no type conversion will be performed

    e.g., `3 === "3"` is false

- Mostly same as Java
  - compares *values* on primitives, *references* on objects
  - but strings are primitive in JS (no `.equals` needed)

    == on strings common source of bugs in Java

Basic Data Types of JavaScript

- **JavaScript includes the following runtime types**

`number`	
`string`	
`boolean`	
`null`	
`undefined`	(another null)
`Object`	
`Array`	(special subtype of Object)

we won't use them
until week 5/6

Numbers

`number` (floating point like Java `double`)

- **JS does not have an "`int`" type!**
  - **floating point can represent integers too, so this is fine**
    will be an issue in TypeScript though…
- **All the usual operators:** `+ - * / ++ -- +=` **…**
- **Math library largely copied from Java**
  - **e.g.,** `Math.sqrt` **returns the square root**
- **Be careful when using division!**
  - **can produce a non-integer!**
  - **use** `Math.floor(x / y)` **to do Java-like integer division**

Strings

- Mostly the same as Java
  - immutable
  - string concatenation with “+”

- A few improvements
  - string comparison with “===” and “<”
  - use either ’ . . ’ or ” . . ” (single or double quotes)
  - new string literals that support variable substitution:

```
const name = “Fred”;
console.log(`Hi, ${name}!`);  // prints “Hi, Fred!”
```

Boolean

- **All the usual operators:** `&& || !`

- **“`if`” can be used with any value**
  - **“falsey” things:** `false`, `0`, `NaN`, `“”`, `null`, `undefined`
  - **“truthy” things: everything else**

- **A common source of bugs…**
  - **best to stick to boolean values**

Record Types

- JavaScript “Object” is something with “fields”
- JavaScript has special syntax for creating them

```
const p = {x: 1, y: 2};
console.log(p.x);  // prints 1
```

- The term “object” is potentially confusing
  - used for many things
  - I prefer it as shorthand for “mathematical object”
- Will refer to things with fields as “records”

Record Types

- Quotes are optional around field names

```
const p = {x: 1, y: 2};
console.log(p.x);  // prints 1

const q = {"x": 1, "y": 2};
console.log(q.x);  // also prints 1
```

- Field names are literal strings, not expressions!

```
const x = "foo";
console.log({x: x});  // prints {"x": "foo"}
```

Checking Types at Run Time

Condition	Code
x is undefined	`x === undefined`
x is null	`x === null`
x is a number	`typeof x === "number"`
x is an integer	**… and** `Math.floor(x) === x`
x is a string	`typeof x === "string"`
x is an object or array	`typeof x === "object"`
x is an array	`Array.isArray(x)`

Hard to check if x is a specific record type at runtime.
Much easier to let the type checker do this!

Functions

- Functions are first class objects
  - “arrow” expressions creates functions
  - store these into a variable to use it later

```
const add2 = (x, y) => x + y;
console.log(add2(1, 2));      // prints 3

const add3 = (x, y, z) => {
  return x + y + z;
};
console.log(add3(1, 2, 3));  // prints 6
```

Functions

- We will declare functions like this

```
const add = (x, y) => {
  return x + y;
};

// add(2, 3) == 5
```

- Functions can be passed around
  - "functional" programming language
  - but we won't do that (much) this quarter

    see CSE 341 for more on that topic

TypeScript

TypeScript Adds Declared Types

- TypeScript includes declared types for variables
- Compiler checks that the types are valid
  - extremely useful!
  - produces JS just by *removing* the types

TypeScript Adds Declared Types

- Type is declared after the variable name:

```
const u: number = 3;
const v: number = 4;

const add = (x: number, y: number): number => {
  return x + y;
};

console.log(add(u, v));  // prints 7
```

  - return type is declared after the argument list (…) and before =>

- "Where types go" is the main syntax difference vs Java
  - other key differences are functions (=>) and equality (===)

Basic Data Types of TypeScript

- JavaScript includes the following types

`number`	
`string`	
`boolean`	
`null`	
`undefined`	
`Object`	(record types)
`Array`	(e.g., `string[]` as in Java)

- TypeScript has the **green** ones and also...

`unknown`	
`any`	(turns off type checking — do not use!)

Ways to Create New Types in TypeScript

- **Union Types** `string | number`
  - can be either one of these

- **Not possible in Java!**
  - TS can describe types of code that Java cannot

- **Unknown type is (essentially) a union**

```
type unknown = number | string | boolean | null | ...
```

Ways to Create New Types in TypeScript

- Can create *compound* types in multiple ways
  - put multiple types together into one larger type

- **Record Types** `{x: number, s: string}`
  - anything with *at least* fields "x" and "s"

```
const p: {x: number, s: string} = {x: 1, s: 'hi'};
console.log(p.x);  // prints 1
```

Ways to Create New Types In TypeScript

- Can create *compound* types in multiple ways
  - put multiple types together into one larger type

- Tuple Types `[number, string]`
  - at runtime, this is an array of length 2
  - create them like this

```
const p: [number, string] = [1, 'hi'];
```

  - give names to the parts to use them

```
const [x, y] = p;
console.log(p.x);  // prints 1
```

Type Aliases

- TypeScript lets you give shorthand names for types

```
type Point = {x: number, y: number};

const p: Point = {x: 1, y: 2};
console.log(p.x);  // prints 1
```

- Usually nicer but not necessary
  - e.g., this does the same thing

```
const p: {x: number, y: number} = {x: 1, y: 2};
console.log(p.x);  // prints 1
```

Structural vs Nominal Typing

- Deeper difference between TypeScript and Java
  - records aren't just a quick way to describe a class

- TypeScript uses "**structural typing**"
  - sometimes called "duck typing"

    "if it walks like a duck and quacks like a duck, it's a duck"

```
type T1 = {a: number, b: number};
type T2 = {a: number, b: number};

const x: T1 = {a: 1, b: 2};
```

  - can pass " `x` " to a function expecting a " `T2` "!

Structural vs Nominal Typing

- Java uses "**nominal typing**"

```
class T1 { int a; int b; }
class T2 { int a; int b; }

T1 x = new T1();
```

  - **cannot** pass "`x`" to a function expecting a "`T2`"

- Libraries do not interoperate unless it was pre-planned
  - create "adapters" to work around this

    example of a design pattern used to work around language limitations

Imports

- JS / TS code can now be split into multiple files
  - JS didn't initially have that feature
- By default, declarations are hidden outside the file
- Add the keyword "export" to make it visible

```
export const MAX_NUMBER = 15;  // in src/foo.ts
```

- Use the "import" statement to bring into another file

```
import { MAX_NUMBER } from './foo';  // in src/bar.ts
```

  - '`./foo`' is relative path from this file to `foo.ts`
  - extension (`.ts`) is not included

Imports

```
export const MAX_NUMBER = 15;          // in src/foo.ts

import { MAX_NUMBER } from './foo';    // in src/bar.ts
```

- For code you write, you will only need this syntax
- JS / TS includes other ways of importing things
  - full explanation is very complicated
  - don't worry about it...
- Starter code will include some that look different, e.g.:

```
import React, { Component } from 'react';

import './foo.png';  // include a file along with the code
```

Next Up

- Software to set up if you're using your own machine
  - see Software Setup Guide on the website
- Section tomorrow
  - start playing with JS / TS yourself
  - starting the server
  - parsing the query parameters in the URL
  - writing HTML into the page
- Friday lecture
  - includes last things you'll need for HW1
  - e.g., how to change the appearance of HTML with styling