TypeScript

Jonathan Kula
12/3/2018
Introducing TypeScript

- A superset of JavaScript. This means that all JavaScript code is valid TypeScript code!
- TypeScript just adds some new features that will make your life easier.
- TypeScript adds a step between “code” and “browser” that checks your code for consistency. (This is called the compiler.)
- TypeScript also adds additional syntax so you can tell the compiler what you’re trying to do; then, it’ll try to help you do that.
- It’s all about making your code more consistent.
Uncaught TypeError: Cannot read property 'add' of undefined
  at makeCircle (FunctionWhoops.js:3)
  at FunctionWhoops.js:8
function makeCircle(radius, x, y, gw) {
    let oval = GOval(x, y, radius * 2, radius * 2);
    gw.add(oval);
}

let gw = GWindow(400, 400);

makeCircle(50, 200, 200);
function makeCircle(radius, x, y, gw) {
    let oval = GOval(x, y, radius * 2, radius * 2);
    gw.add(oval);
}

let gw = GWindow(400, 400);

makeCircle(50, 200, 200);
Creating square with color green!
The new square's color is: undefined
function createSquare(config) {
  return {
    width: config.width,
    height: config.height,
    color: config.color,
    area: config.width * config.height,
    perimeter: config.width * 2 + config.height * 2
  }
}

console.log("Creating square with color green!");

let square = createSquare({width: 2, height: 2, colour: "Green"});

console.log("The new square's color is: " + square.color);
function createSquare(config) {
  return {
    width: config.width,
    height: config.height,
    color: config.color,
    area: config.width * config.height,
    perimeter: config.width * 2 + config.height * 2
  }
}

console.log("Creating square with color green!");

let square = createSquare({width: 2, height: 2, colour: "Green"});

console.log("The new square's color is: " + square.color);
Enter month number: 4
Not a a month!
let variableName: TypeName;

const CONSTANT_NAME: TypeName;

function functionName(param1: Type1, param2: Type2): ReturnType {
let age: number = 20;

const ALPHABET: string = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

function divides(divisor: number, dividend: number): boolean {

Type Inference

```
let foo = "Hello";
let foo: string = "Hello";
```

These statements are equivalent because of type inference.
Type Inference

let foo = 1;

let foo: string = 1;

These statements are not equivalent because of type inference.  
(The second statement throws an error; 1 is not a string!)
What types are there?

- Here are types you’ve worked with!
  - number
  - string
  - boolean
  - null
  - undefined
  - object
  - function
  - any
What types are there?

- Here are types you’ve worked with!
  - number
  - string
  - boolean
  - null
  - undefined
  - object
  - function
  - any

**Primitive Types:**
“simple” types – You build all other types out of primitive types.
What types are there?

- Here are types you’ve worked with!
  - number
  - string
  - boolean
  - null
  - undefined
  - object
  - function
  - any

**Primitive Types:**
“simple” types – You build all other types out of primitive types.

**Non-Primitive Types:**
Everything else.
What types are there?

- Here are types you’ve worked with!
  - number
  - string
  - boolean
  - null
  - undefined
  - object
  - function
  - any

**Primitive Types:**
“simple” types – You build all other types out of primitive types.

**Non-Primitive Types:**
Everything else.

**any:**
A special type that can represent anything!
What types are there?

- Here are types you’ve worked with!
  - number
  - string
  - boolean
  - null
  - undefined
  - object
  - function
  - any

- And some types you haven’t.
  - symbol
  - never ← ask me about these after class!

**Primitive Types:**
“simple” types – You build all other types out of primitive types.

**Non-Primitive Types:**
Everything else.

**any:**
A special type that can represent anything!
What types are there?

Here are types you’ve worked with!

- number
- string
- boolean
- null
- undefined
- object
- function
- any

Primitive Types:
“simple” types –
You build all other types out of primitive types.

Non-Primitive Types:
Everything else.

any:
A special type that can represent anything!
What types are there?

- Here are types you’ve worked with!
  - number
  - string
  - boolean
  - null
  - undefined
  - **object** ← Seems like this describes an awful lot...
  - function
  - any
let dog = {
  type: 'mammal',
  name: 'dog',
  sounds: ['woof', 'bark', 'yip', 'ruff']
};

let cat = {
  type: 'mammal',
  name: 'cat',
  sounds: ['meow', 'purr', 'hiss']
};

let enigma = {
  rotors: [],
  lamps: [],
  keys: []
};

let key = {};
key.letter = "A";
key.mouseDownAction = function () {

};

let jonathan = {
  favoriteColor: "Green",
  name: "Jonathan Kula",
  status: "Active",
  classes: [
    {
      name: "CS106AJ",
      role: "SL",
      grade: -1
    },
    {
      name: "CS103",
      role: "Student",
      grade: 87.5
    }
  ]
};

let profile = {
  name: "Jonathan Kula",
  imageUrl: "http://image.url/img.png",
  language: "English"
};
Interfaces

- Interfaces describe the structure of objects.
Interfaces

- Interfaces describe the structure of objects.
- Interfaces are not objects.
- Interfaces have no functionality – they only describe other objects.

```java
interface InterfaceName {
    property1: Type1
    property2: Type2
}
```
Interfaces

- Interfaces describe the structure of objects.
- Interfaces are not objects.
- Interfaces have no functionality – they only describe other objects.

```typescript
interface Point {
    x: number
    y: number
}
```
What if we wanted to make an interface for an Enigma key?
Function Annotations

- What if we wanted to make an interface for an Enigma key?

```typescript
interface WithFunction {
    func: (param1: Type1, param2: Type2) => Return_type
}
```
Function Annotations

- What if we wanted to make an interface for an Enigma key?

```java
interface Key {
    letter: string
    onMouseDown: () => void
}
```

- `void` is a special type meaning “doesn’t return anything”
Map Annotations

- What about using objects as maps?
Map Annotations

- What about using objects as maps?

```java
interface Phonebook {
    [name: KeyType]: ValueType
}
```

- The `KeyType` can be either `string` or `number`. 
Map Annotations

- What about using objects as maps?

```typescript
interface Phonebook {
    [name: string]: string
}
```

- The KeyType can be either string or number.
Classes in TypeScript

- Think of them like “Interfaces with functionality”
- You use “class-like factory functions” in Teaching Machine, Adventure, and when coding using object-oriented ideas.
- Classes are types too, much like interfaces!
Classes in TypeScript

- Make an object of a class by using the `new` keyword.
- Refer to *properties of the class* using the `this` keyword.
- `this` inside a class refers to “the current object.”

```typescript
let jonathan = new Profile("Jonathan Kula", "http://image.url/", "English");
let ryan = new Profile("Ryan Eberhardt", "http://image.url/", "English");

jonathan.getName(); // “this” now refers to jonathan – returns “Jonathan Kula”
ryan.getName(); // “this” now refers to ryan – returns “Ryan Eberhardt”
```
Acquiring Typescript

- Download nodejs LTS from [https://nodejs.org/en/](https://nodejs.org/en/)
- Open a **Powershell** (Windows) or **Terminal** (macOS or Linux)
- Type `npm install -g typescript`
Setting Up Typescript

- Download TypeScript configuration file from the course website.
  - I can break it down after class if you’re interested!
- Put that file in your project folder.
Using TypeScript

Manually:

- Open Powershell/Terminal, go to your project directory using `cd`, then type `tsc` to build all `.ts` files into `.js` files!

Better:

- Get an IDE that supports TypeScript!
- I use both WebStorm and Visual Studio Code.
  - I prefer WebStorm, but it’s only free while you’re a student. Visual Studio Code is also quite good, and free. I have a slide deck about how to acquire WebStorm [here](#).