http://bit.ly/fem-ts

TypeScript Fundamentals v2

March 5, 2019
Mike North

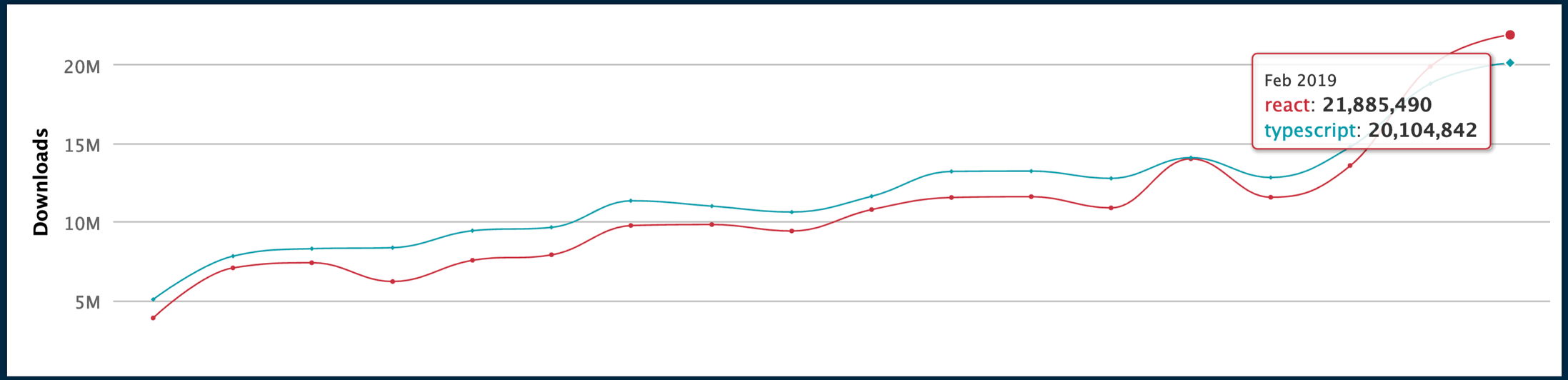
MIKE WORKS

What's TypeScript?

- An open-source **typed, syntactic superset** of JavaScript, developed by **Microsoft**
- Compiles to readable JavaScript
- Comes in three parts: **Language**, **Language Server** and **Compiler**
- Works seamlessly with **Babel 7**

What's TypeScript?

- Since 2017...
  - +300% increase in downloads, now tied w/ React
  - 2018 NPM survey: 46% of respondents use TypeScript



...but why add types?

- Encode constraints and assumptions, as part of developer intent
- Catch common mistakes (i.e. incomplete refactors)
- Move some runtime errors to compile time
- Provide your consumers (including you) with a great DX

In this class, we'll learn about ...

- Adding type information to variables, functions and classes
- Configuring the compiler
- A practical strategy for incrementally converting JS to TS
- Parameterizing interfaces and type aliases with generics
- Conditional, mapped and branded types
- TS Compiler API basics

http://bit.ly/fem-ts

Compiling

```
./examples/hello-ts/
```

tsconfig.json

Transform JSX

Enable "strict" features

Forbid implicit any

Check + compile JS

Target environment

```
{
  "compilerOptions": {
    "jsx": "react",
    "strict": true,
    "sourceMap": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "allowJs": true,
    "types": [],
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "moduleResolution": "node",
    "target": "es2015"
  }
}
```

`hello: "TypeScript"`

- ☑ Simple Variables
- ☑ Arrays & Tuples
- ☑ Objects
- ☑ Union & Intersection Types

```
./notes/1-basics.ts
```

Type Systems & Type Equivalence

```
function validateInputField(input: HTMLInputElement) {
  /*  ...  */
}

validateInputField(x);
```

Can we regard x as an HTMLInputElement?

- **Nominal Type Systems** answer this question based on whether x is an instance of a class/type **named** HTMLInputElement
- **Structural Type Systems** only care about the **shape** of an object.
  This is how typescript works!

Object Shapes

- When we talk about the **shape** of an object, we're referring to the names of properties and types of their values

Car

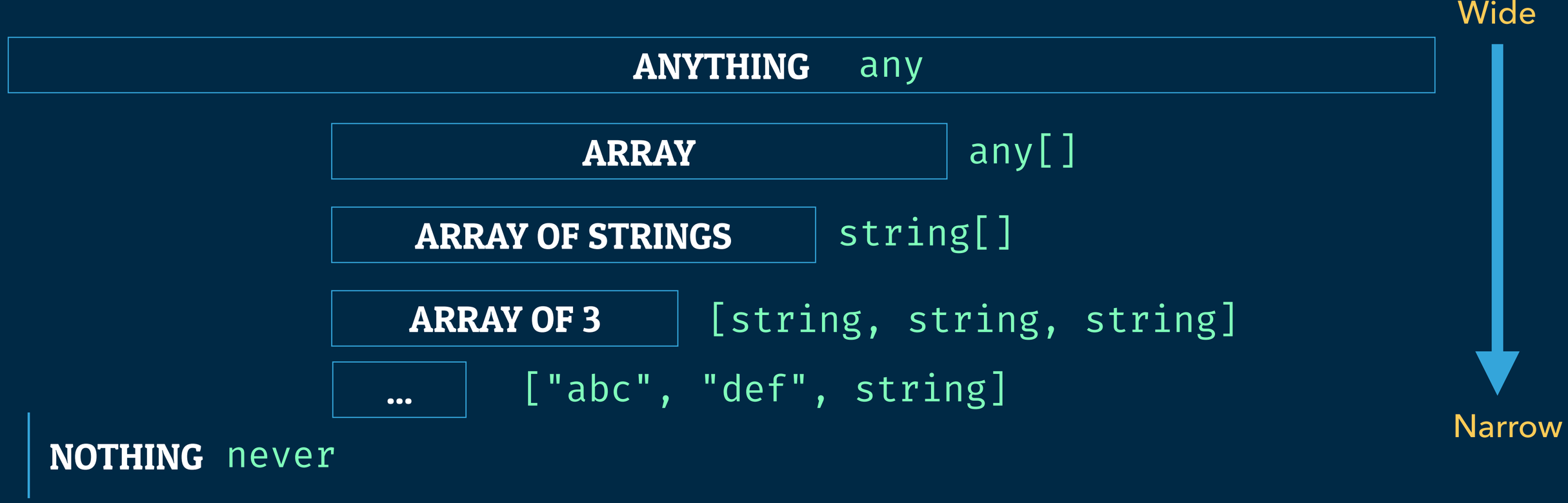
Make String

Model String

Year Number

Wider vs. Narrower

- Describes: relative differences in range of a type's allowable values



`function hello();`

- ☑ Overloaded signatures
- ☑ This types
- ☑ Function declarations & expressions

`./notes/2-function-basics.ts`

`interface hello {}`
`type hello`

- ☑ **Call, Construct and Index signatures**
- ☑ **"Open interfaces"**
- ☑ **Access modifier keywords**
- ☑ **Heritage clauses ("extends", "implements")**

```
./notes/3-interface-type-basics.ts
```

JSON Types

1

- We'll want to implement types that describe arbitrary JSON value

```
2.1.  Values

   A JSON value MUST be an object, array, number, or
string, or one of the following three literal names:

      false null true
```

- Export types JSONValue (any value), JSONArray (array value), JSONObject (object value)

./challenges/json-types/index.ts

`class hi {}`

- ☑ Implementing interfaces
- ☑ Fields
- ☑ Access modifier keywords (public, private, protected)
- ☑ Param Properties

`./notes/4-class-basics.ts`

Converting JS ▸ TS

3 STEPS FOR SUCCESS

What not to do

- Functional changes at the same time
- Attempt this with low test coverage
- Let the perfect be the enemy of the good
- Forget to add tests for your types
- Publish types for consumer use while they're in a "weak" state

1. Compliling in "loose mode"

- Start with tests passing
- Rename all .js to .ts, allowing implicit any
- Fix only things that are not type-checking, or causing compile errors
- Be careful to avoid changing behavior
- Get tests passing again

2. Explicit Any

- Start with tests passing
- Ban implicit any
```
"noImplicitAny": true,
```
- Where possible, provide a specific and appropriate type
  - Import types for dependencies from DefinitelyTyped
  - otherwise explicit any
- Get tests passing again

3. Squash explicit anys, enable strict mode

- Incrementally, in small chunks...
- Enable strict mode

```
"strictNullChecks": true,
"strict": true,
"strictFunctionTypes": true,
"strictBindCallApply": true
```

- Replace explicit anys w/ more appropriate types
- Try really hard to avoid unsafe casts

Address Book

2

- We have a address book program that we want to convert from JS to TS using our 3-step approach

`./challenges/address-book/index.ts`

hello<Generics>

- ☑ When to use them
- ☑ Type parameters
- ☑ Constraints

`./notes/5-generics-basics.ts`

Dict<T> and Friends

3

- A Dictionary (a.k.a. Associative Array) is a collection of key-value pairs, that ensures key uniqueness.
- Build a Dict that's generic over its value type
- Higher-order functions on JS Arrays are awesome! Let's also make a `mapDict` and `reduceDict`.

```
const fileExtensions = {
  typescript: ['ts'],
  javascript: ['js'],
  jpeg: ['jpg', 'jpeg'],
  html: ['html', 'htm']
}
```

```
mapDict(fileExtensions, exts ⇒
  exts.map(e ⇒ `*.${e}`).join(", ")
);
```

```
{ typescript: "*.ts",
  javascript: "*.js",
  jpeg: "*.jpg, *.jpeg",
  html: "*.html, *.htm" };
```

./challenges/dict/

Top and bottom types

- ☑ Passing private values through typed code
- ☑ Exhaustive Conditionals
- ☑ Type Guards
- ☑ Branded Types

```
./notes/6-guards-and-extreme-types.ts
```

Advanced Types

- ☑ Mapped Types
- ☑ Type Queries
- ☑ Conditional Types, and `infer`
- ☑ TypeScript's Built-in Utility Types

```
./notes/7-advanced-types.ts
```

Advanced Type Challenges 4

- In the `./challenges/advanced-types` folder, there are several JSDoc comments describing some types, but the types have been deleted!
- Do your best to implement the types again, and ensure all the tests pass

`./challenges/advanced-types/`

Declaration Merging

- [x] How typescript understands your code
- [x] Stacking values, types and namespaces

```
./notes/8-declaration-merging.ts
```

Compiler API

BUILDING ON TOP OF TYPESCRIPT

```
./notes/9-compiler-api.ts
```