Simple Types

PROGRAMMING WITH TYPESCRIPT

Objectives

- To understand how TypeScript applies types to variables
- To be aware of additional types used in TypeScript
  - Array
  - Tuple
  - Enum
  - Any
  - Void
  - Never
  - Object
  - Unknown (introduced in TypeScript v3.0)

2

CONTINUED 2

Primitive Types – just like JavaScript

PRIMITIVES

- `boolean`
  - true and false values only
- `number`
  - Floating point values
- `string`
  - 'single' "double" or `backticks`
- `null`
  - To define a value that does not exist
- `undefined`
  - The value of a variable that has never been assigned a value
- `symbol`
  - Values of this type can be used to make anonymous object properties

3

Adding Types to variable declarations

- A colon and the type to assign to the variable name are added after it

```
// Boolean
let isDone:boolean = false;

// String
let fullName: string = `John Smith`;
let greeting: string = `Hello, my name
is ${fullName}`;
```

```
// Number
let decimal: number = 6;
let float: number = 100.232;
let exponential: number = 3.2E1234;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;
```

4

CONTINUED 4

Types – Arrays

- Arrays
  - Statically typed arrays declared in one of two ways
- Familiar to JavaScript developers, the square brackets

```
let list: number[] = [1, 2, 3];
let list2: string[] =[`Pie`, `Gravy`];
```

- Familiar to C# and Java developers, the angle bracket notation

```
let list: Array<number> = [1, 2, 3];
let list2: Attay<string> = [`Pie`,
`Gravy`];
```

5

Types – Tuple

- Express an array of fixed (or variable) length but differing types
  - ? on a type means that it is optional
  - If optional is in middle, all following must have ? too

```
//Declaration
let person:[string, number, number?];

//Initialisation
let person = [`John`,21]
// OK

let person = [21, `John`]
// Error
```

- Accessing Tuples
  - When accessing element with known index, correct type is retrieved too:

```
console.log(person[0].substr(1));
// OK
console.log(person[1].substr(1));
// Error
```

- Adding to Tuples
  - Defined length can still be achieved with correct type – but not more

```
person[2] = `Smith`; // Error
person[2] = 1234;    // OK
person[3] = 1234;    // Error
person[3] = `Smith`; // Error
```

6

In --strictNullChecks mode, ? means undefined is included in the element

QuickLab 3 - Tuples

- Define and experiment with tuples.

7

Types – Enum

- Friendly names for numeric values
- Automatically numbered from 0

```
enum Color {Red, Green, Blue};
let c: Color = Color.Green; //1
```

- Can start from any number

```
enum Color {Red=4, Green, Blue};
let c: Color = Color.Green; //5
```

- Or number them all manually

```
enum Color {Red=4, Green=8, Blue=16};
let c: Color = Color.Green; //8
```

- And go from numeric value to the name

```
enum Color {Red=4, Green=8, Blue=16};
let c: string = Color[8]; //Green
```

8

Types – Any

- All types are subtypes of Any
  - Gives us a route to describe variables that we do not know the type of e.g. from the user or 3rd party libraries

```
let thing: any = `Thing T. Thing`;
thing = false; // OK
```

- This is also handy to start opting in to type checking during compilation
  - And can be helpful if you know parts of a type, but not all

```
let list: any[] = [1, true, `thing`];
```

9

Types – Void

- In some ways, the opposite of **any** this type is the absence of any type at all.
  - Often the type of functions that don't return a value
  - Declaring variables of type void is not useful
    - Can only assign null and undefined to them

```
function absenceOfThing(): void {
       alert(`Thing has gone AWOL`);
}
```

```
Let unusable: void = undefined;
unusable = null;  // OK
Unusable = true;  // Error
```

10

Types – Null and Undefined

- Both primitive types in JavaScript
  - Not extremely useful on their own
- Subtypes of all other types
  - Can assign null and undefined to anything
- Can use strictNullChecks compiler option to ensure that this can only happen if required
  - Use union types

```
let u: undefined = undefined;
let n: null = null;
```

```
let aNumber: number = undefined;
let aString: string = null;
```

```
//tsconfig.json
…
  "strictNullChecks": true
…
```

```
let aNumber: number = null; // Error
let aString: string|null = `Hi`;
aString = null; // OK
```

11

Types – Never

- **never** is a subtype of *every type* but *nothing* is a subtype of **never**
- **never** represents the type of values that never occur

```
//functions that never return
function notEver(): never {
       while (true) {}
}

//functions that always throw
function alwaysThrow(): never {
       throw new Error("throwing");
}
```

12

Object

- Used to represent anything that is not a primitive
  - Allows better representation of APIs, e.g Object.create

```
declare function create(o:
object|null): void;
create({prop: 0});  // OK
create(null);       // OK
create(42);         // Error
create(`string`);   // Error
```

13

Types – Type Assertion

- Sometimes as developers we need to override the compiler
  - Usually when an entity is more specific than its current type
- TypeScript provides two syntaxes

```
//angle-bracket syntax
let thing: any = `Thing T. Thing`;
let nameLength: number =
(<string>thing).length;

//as-syntax
let thing: any = `Thing T. Thing`;
let nameLength: number = (thing as
string).length;
```

14

Unknown top-type

- Type-safe counterpart of **any**
- Anything is assignable to **unknown**
- **unknown** isn't assignable to anything but itself and **any** without a type-assertion or control flow based narrowing
- No operations are permitted on an **unknown** without assertion or narrowing

```
let x: unknown = `Hi`;  // OK
x = 42;
x !== 10;           // OK
x <= 22;            // Error - equality
                    // operators only
(x as number) >= 10 // OK
```

15

QuickLab 3b – Type Assertion and Unknown

- Use the unknown type and type assertion when working with variables.

16

Objectives

- To understand how TypeScript applies types to variables
- To be aware of additional types used in TypeScript
  - Array
  - Tuple
  - Enum
  - Any
  - Void
  - Never
  - Object
  - Unknown (introduced in TypeScript v3.0)

17

CONTINUED 17