TypeScript

Training Agenda

- TypeScript – What & Why
- Types in TypeScript
- OOP in TypeScript
- Generics & Decorators
- Modules

TypeScript is JavaScript with syntax for types.

TypeScript - Introduction

Strongly typed programming language

Built on JavaScript

An optional type system for JavaScript.

Features from future JavaScript

Better tooling

Corporate care-taker

Why TypeScript?

Safety

- Type systems allow many errors to be caught early, without running the code.

Readability

- Explicit types make code easier for humans to understand.

Better Tooling

- Allows tools like IDEs and linters to be more powerful.

Compilation Context

The compilation context is used for grouping of the files that TypeScript will parse and analyze to determine what is valid and what isn't.

Great way to define this logical grouping is using a *tsconfig.json* file.

Types Overview

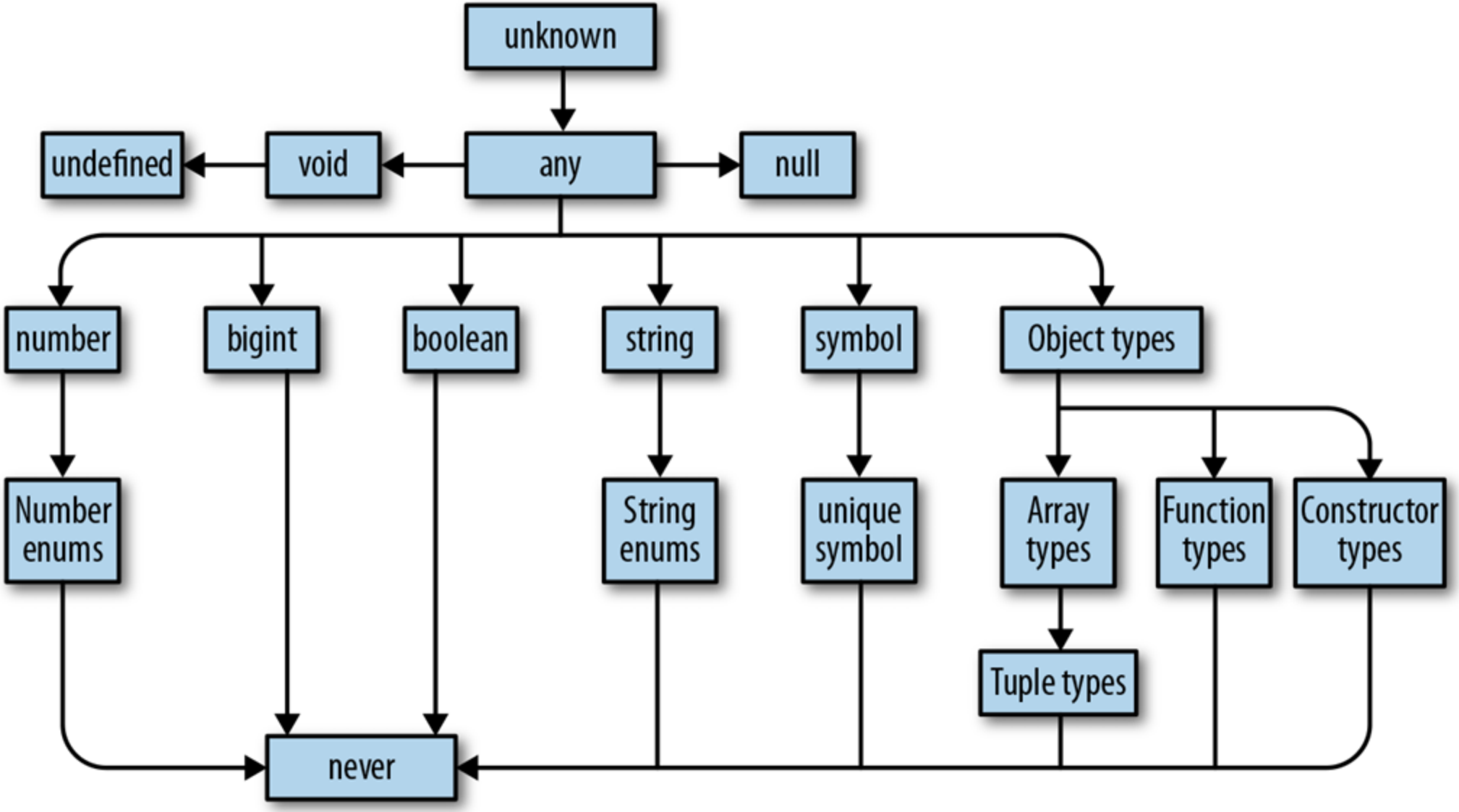
Types can be Implicit

Types can be Explicit

Types are structural

Type errors do not prevent JavaScript emit

Types can be ambient


unknown
any
undefined
void
null
number
bigint
boolean
string
symbol
Object types
Number enums
String enums
unique symbol
Array types
Function types
Constructor types
Tuple types
never

Types that mean an absence of something

null	Absence of a value
undefined	Variable that has not been assigned a value yet
void	Function that doesn’t have a return statement
never	Function that never returns

ES6 Features

- Block Scope
- Template literals
- Promises
- Default Parameters
- Arrow functions
- Destructuring
- Rest & Spread
- Classes

Working with Classes

Few Class Concepts -

- Using ‘this’
- Access Modifiers
- Getters / setters
- Static properties and methods
- Method overriding
- Method overloading
- Abstract classes
- Interfaces

Generics

A placeholder type used to enforce a type-level constraint in multiple places. Also known as polymorphic type parameter.

The generic types declared within the triangle brackets:
<T>

Constraining the generic types is done with the extends keyword:
<T extends Car>

Decorators

Decorators are an experimental TypeScript feature that gives us a clean syntax for metaprogramming with classes, class methods, properties, and method parameters.

Decorators are syntax for calling a function on the thing you're decorating.

Modules

By default, when you start typing code in a new TypeScript file your code is in a global namespace.

- Global namespace is dangerous as it opens your code up for naming conflicts.

File Module / External Module

- If you have an *import* or an *export* at the root level of a TypeScript file then it creates a local scope within that file.
- Using an *import* in file not only allows you to bring in stuff from other files, but also marks the file as a module and therefore, declarations in that file don't pollute the global namespace either.

Reference

BOOKS

TypeScript Deep Dive

- *By Barasat Ali Syed*

Learning TypeScript

- *By Stack Overflow Contributors*

WEB

https://www.typescriptlang.org/

https://basarat.gitbook.io/typescript/

https://www.youtube.com/watch?v=BwuLxPH8IDs