TypeScript

What’s TypeScript

- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript
- TypeScript is JavaScript plus some additional features
- Designed by Anders Hejlsberg (designer of C#) at Microsoft
- Any regular Javascript is valid TypeScript Code
- TypeScript supports other JS libraries

Why use TypeScript

- Compilation
  - Find bug early
  - Suitable for big project
- Better IDE support
  - Intelligent code completion
  - Easy to refactor
- Other typed-JS alternatives
  - Flow(Facebook), Dart(Google)

```
function createMaterial(texture: string): MeshBasicMaterial {
  const material= new MeshBasicMaterial();
   (property) MeshBasicMaterial.map: Texture
   Type 'string' is not assignable to type 'Texture'. ts(2322)
   Peek Problem    No quick fixes available
  material.map = texture;
  return material;
}
```

```
const material= new MeshBasicMaterial({

})  alphaMap   (property) MeshBasicMate
    alphaTest
    aoMap
    aoMapIntensity
    blendDst
    blendDstAlpha
    blendEquation
```

Who is using TypeScript

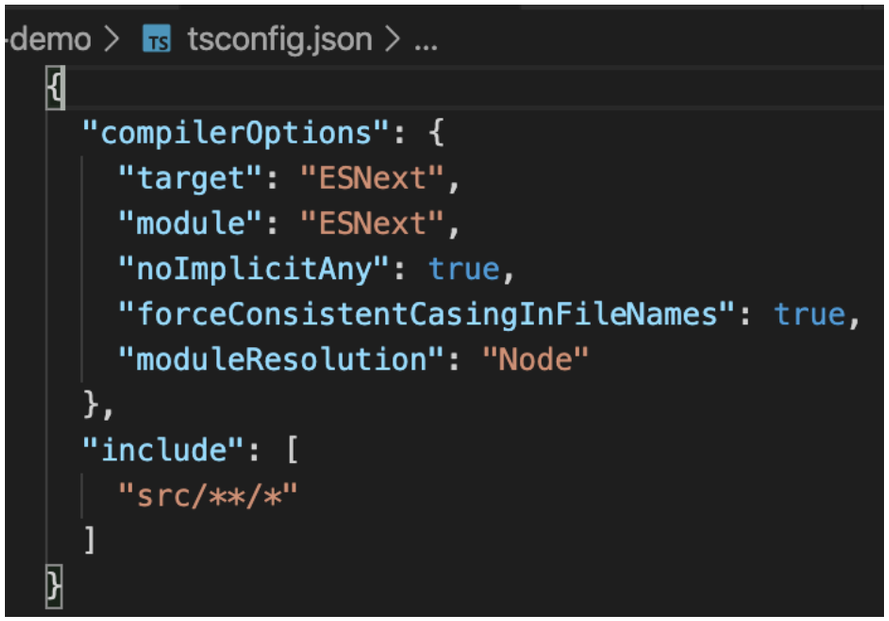
- Native
  - Angular.js
  - Vue.js
  - Babylon.js
  - Deno
  - Nest.js
- Declaration Files: .d.ts
  - React.js
  - Three.js

Resource

- typescriptlang.org
- Handbook
- Playground
- TypeScript Language Specification

Install TypeScript

- npm i -D typescript
  - npx tsc src/index.ts
- npx tsc --init
  - tsconfig.json
  - Compiler Options



- Show all erros
  - "compile": "tsc --noEmit --project './tsconfig.json' || true",
- Visual Studio Code

Rollup

- @rollup/plugin-sucrase
  - A Rollup plugin which compiles TypeScript, Flow, JSX, etc with Sucrase
  - npm i -D rollup @rollup/plugin-sucrase @rollup/plugin-node-resolve
- @rollup/plugin-typescript
  - Slower, but has type checking
  - npm i -D typescript tslib
  - npm i -D rollup @rollup/plugin-typescript
- Webpack

ESLint TypeScript

emo > .eslintrc > {} parserOptions

```
{
  "extends": [
    "airbnb-typescript/base",
    "plugin:@typescript-eslint/recommended"
  ],
  "parserOptions": {
    "project": "./tsconfig.json"
  }
}
```

- eslint-config-airbnb-typescript
  - Airbnb's ESLint config with TypeScript support
  - npm i -D typescript eslint
  - npm i -D eslint-config-airbnb-typescript eslint-plugin-import @typescript-eslint/eslint-plugin
- .eslintrc
  - "extends": ["airbnb-typescript/base", "plugin:@typescript-eslint/recommended"]
- "eslint": "eslint --ext .ts src || true"
  - npm run eslint

```
}
                Unexpected any. Specify a different type. eslint(@typescript-eslint/no-explicit-any)
function        Peek Problem    Quick Fix...
  let a: any;
```

- VS Code ESLint extension

Basic Types

- Boolean
  - let isDone: boolean = false;
- Number
  - let decimal: number = 6;
- String
  - let color: string = "blue";
- Array
  - let list: number[] = [1, 2, 3];
  - let list: Array<number> = [1, 2, 3];
- Tuple
  - let x: [string, number] = ["hello", 10];
- Enum
  - enum Color {Red, Green, Blue}
  - let c: Color = Color.Green;

Basic Types

- Any (Don’t use it, can work with 3rd party library)
  - let list: any[] = [1, true, "free"];
- Void
  - function test(): void
- Never
  - function error(message: string): never { throw new Error(message); }
- Object
  - object is a type that represents the non-primitive type
  - let obj: object = { amount: 10 };
- Type assertions
  - let someValue: any = "this is a string"; let strLength: number = (<string>someValue).length;
  - let someValue: any = "this is a string"; let strLength: number = (someValue as string).length;

Interface

- interface LabeledValue { label: string; }
- function printLabel(labeledObj: LabeledValue) { console.log(labeledObj.label); }
- Optional Properties
    - interface SquareConfig { color?: string; width?: number; }
- Readonly properties
    - interface Point { readonly x: number; readonly y: number; }
    - Variables use const whereas properties use readonly
- Function Types
    - interface SearchFunc { (source: string, subString: string): boolean; }
- Implementing an interface
    - interface ClockInterface { currentTime: Date; setTime(d: Date): void; }
    - class Clock implements ClockInterface { currentTime: Date = new Date(); setTime(d: Date) { this.currentTime = d; } }
- Interfaces Extending Classes
    - inherits the members of the class but not their implementations

Class

- Public, private, and protected modifiers
  - Public by default
- Readonly modifier
  - must be initialized at their declaration or in the constructor
- Parameter properties
  - create and initialize a member in one place
  - constructor(private readonly name: string)
- Accessors
  - accessors with a get and no set are automatically inferred to be readonly
- Static Properties
  - class Grid { static origin = {x: 0, y: 0}; }
- Abstract Classes
  - abstract class Animal { abstract makeSound(): void; }

Function

- Typing the function
  - const log: (text: string) => void = function (text: string) {};
- Optional Parameters
  - function log(text?: string) {}
- this parameters
  - this parameters are fake parameters that come first in the parameter list of a function
  - addClickListener(onclick: (this: void, e: Event) => void): void;
- Overloads
  - function test(a: number): number {}
  - function test(a: string): string {}

Generics

- Generic function
  - function identity<T>(arg: T): T { return arg; }
  - let output = identity<string>("myString");
- Generic interface
  - interface GenericIdentityFn<T> { (arg: T): T; }
  - function identity<T>(arg: T): T { return arg; }
  - let myIdentity: GenericIdentityFn<number> = identity;
- Generic class
  - class GenericNumber<T> { zeroValue: T; add: (x: T, y: T) => T; }
- Generic Constraints
  - function loggingIdentity<T extends Lengthwise>(arg: T): T
- Using Type Parameters in Generic Constraints
  - function getProperty<T, K extends keyof T>(obj: T, key: K) { return obj[key]; }
- Using Class Types in Generics
  - function create<T>(c: {new(): T; }): T { return new c(); }

Enums

- Enums allow us to define a set of named constants
- Numeric enums
  - enum Direction { Up = 1, Down, Left, Right, }
- String enums
  - enum Direction { Up = "UP", Down = "DOWN", Left = "LEFT", Right = "RIGHT", }

Type Inference

- Basics
  - let x = 3;
- Best common type
  - let x = [0, 1, null];
- Contextual Typing
  - window.onmousedown = function(mouseEvent) {};

Type Compatibility

- x is compatible with y if y has at least the same members as x
- The source function should has less same parameters
- The source function's return type be a subtype of the target type's return type
- When comparing the types of function parameters, assignment succeeds if either the source parameter is assignable to the target parameter, or vice versa
- Enum values from different enum types are considered incompatible
- Private and protected members in a class affect their compatibility
- Generic type that has its type arguments specified acts just like a non-generic type

Modules

- export
  - export class
  - export default
  - export * from './test';
  - export { A as B } from './test';
- import
  - import { A } from './test';
  - import A from './test';
  - import * as name from './test';
  - import ‘./test';
- Guidance for structuring modules
  - If you’re only exporting a single class or function, use export default
  - If you’re exporting multiple objects, put them all at top-level
  - Explicitly list imported names

Namespaces

- DO NOT USE IT
- ESLINT no-namespace
- Is typescript Namespace feature deprecated

Module Resolution

- Relative module imports
  - A relative import is one that starts with /, ./ or ../
  - You should use relative imports for your own modules
- Non-relative module imports
  - import { GUI } from ‘dat.gui';
  - Use non-relative paths when importing any of your external dependencies
  - Module Resolution Strategies
    - --moduleResolution Classic(default) or Node
  - Path mapping
    - baseUrl
    - paths

Declaration Files

- Generate
  - tsc -t ESNext -m ESNext -d --emitDeclarationOnly --declarationDir d src/index.ts
- Consumption
  - package with types: npm i three
  - package without types: npm i dat.gui @types/dat.gui

Advanced Features

- Advanced Types
  - Interfaces vs Types
    - Stackoverflow
    - Playground
- Decorators
- Mixins
- Utility Types
- JSX