

타입스크립트 시작하기

TypeScript

박용준(RedPlus)

Microsoft MVP 2020-2021

<https://www.dotnetkorea.com>



타입스크립트 강의 준비 사항

TypeScript

박용준(RedPlus)

Microsoft MVP 2020-2021

<https://www.dotnetkorea.com>



타입스크립트 시작하기

타입스크립트와
자바스크립트
함께 배우기

현재 강의는 특정 자바스크립트
버전을 강요하지 않습니다.
오로지 타입스크립트만을 다룹니
다.

다만, 자바스크립트와 타입스
크립트의 공통부분을 강조해
서 강의가 진행됩니다.

참고로, 타입스크립트 기본 문
법을 다루는 강좌이지 활용을
다루는 강좌는 아닙니다.

자바스크립트를 아주 잘하면,
타입스크립트가 필요없을 수 있습니다.

짧은 코드는 자바스크립트가 훨씬 좋습니다.

"TypeScript is a typed
superset of JavaScript that
compiles to plain JavaScript."

typescriptlang.org

타입스크립트 3가지 큰 특징

- 정적 형식 제공
 - 강력한 형식
 - 형식 추론
- 묶어서 관리
 - 클래스, 네임스페이스, 열거형
 - 공용 구조체 형식
- 훌륭한 도구들
 - Visual Studio Code
 - Visual Studio

타입스크립트를 사용해야하는 이유

- 미래 버전의 자바스크립트를 지금 당장 사용 가능
- 강력한 형식
 - 형식 안정성
- Angular, React, Vue와 타입스크립트를 사용
 - 많은 SPA 프레임워크에서 TypeScript를 기본으로 도입

이 강좌가 다룰 내용들

- 자바스크립트 기초를 타입스크립트로 다루기
- 웹 브라우저 또는 Node로 자바스크립트 코드 실행
- NPM: 노드 패키지 관리자
 - 기본적인 사용법을 알고 있다고 가정
 - 강의에서는 모두 Step by Step 형태로 진행

이 강좌가 다룰 내용들

```
tsc abc.ts  
  --target ES2015 --outDir js
```

이 강좌가 다룰 내용들

타입스크립트 설정 파일

tsconfig.json

tsc --init

tsc -w

tsc

이 강좌가 다룰 내용들

- 웹팩(Webpack)
 - 웹팩 번들러에 대한 내용은 미니 프로젝트에서 간단히 소개

이 강좌가 다룰 내용들

Webpack.config.js

```
npm install ts-loader --save-dev
```


강의 목적

- 자바스크립트 대신에 타입스크립트로 시작
- Angular, Vue, React 등에서 바로 타입스크립트 사용

강의 관련 소스 모음

- 비주얼아카데미
 - <https://github.com/VisualAcademy>
 - TypeScript
 - SubscriberCounterApp

강의 관련 Q&A

- 닷넷코리아
 - <https://www.dotnetkorea.com/>
 - Q&A 게시판

타입스크립트 강의
시작합니다.



쉽게 배우는 TypeScript

박용준 강사



Microsoft MVP(Most Valuable Professional)
데브렉(<http://www.devlec.com/>) 온라인 강의
닷넷코리아(<https://www.dotnetkorea.com/>) 운영자

TypeScript 소개

- 2012년 시작
 - Anders Hejlsberg
 - <https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript>
- 오픈 소스
- JavaScript의 Superset
- 순수 JavaScript로 변환됨
- 강력한 형식
 - *.d.ts 지원
- 클래스 기반 개체 지향



타입스크립트의 특징

- 모던 자바스크립트 기능 제공
 - 화살표 함수
 - let 키워드
 - const 키워드
- C#과 같은 확장 기능들
 - 제네릭
 - 인터페이스
 - 튜플



준비사항

- 선수 학습
 - JavaScript
 - 변수(Variable)
 - 함수(Function)
 - OOP 프로그래밍 언어
 - C++/C#/Java/VB.NET
 - 클래스(Class)
 - 인터페이스(Interface)
 - 상속(Inheritance)
- 개발환경
 - Visual Studio Community 2017 설치된 환경
 - Visual Studio 2017 + Web Essentials 2017 == 최고의 환경
 - 일반적인 개발도구가 설치된 환경
 - Mac / Linux: Visual Studio Code 추천
 - <https://code.visualstudio.com/>
 - WebStorm, Atom, SublimeText, Eclipse 등등
 - Node.js 기반 설치
 - NPM 설치
 - TypeScript(TSC) 설치
 - TypeScript Definition Manager(TSD) 설치



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\wredol>tsc -v
Version 1.8.9

C:\Users\wredol>
```

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\wredol>tsc -v
Version 1.0.3.0

C:\Users\wredol>정확한 버전을 보이려면?
```




```
npm install -g typescript  
tsc --version  
tsc --help  
tsc --init  
tsc app.ts
```



```
C:\Users\VisualAcademy>tsc
```

'tsc'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는 배치 파일이 아닙니다.

```
C:\Users\VisualAcademy>node -v  
v12.18.0
```

```
C:\Users\VisualAcademy>npm -v  
6.14.4
```

```
C:\Users\VisualAcademy>npm i -g typescript
```

```
C:\Users\VisualAcademy>node -v  
v12.18.0
```

```
C:\Users\VisualAcademy>npm -v  
6.14.4
```

```
C:\Users\VisualAcademy>tsc --version  
Version 3.9.7
```



큰 규모의 JavaScript 개발은 어렵다.

"JavaScript is the assembly language of the Web"

- *Erik Meijer*

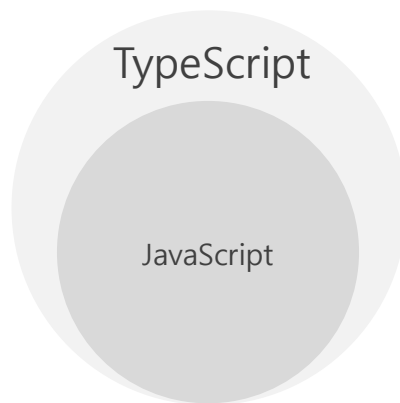
"You can write large programs in JavaScript. You just can't maintain them"

- *Anders Hejlsberg*

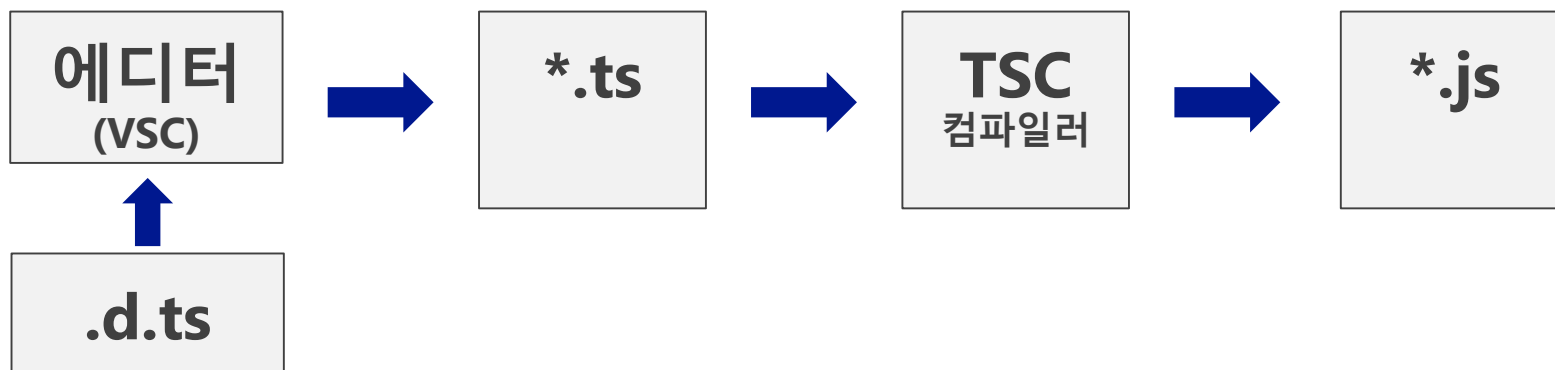
TypeScript는
생산적이고 재미있게 개발을 위한
큰 규모의 JavaScript 개발을 위한 언어

TypeScript는 컴파일시 JavaScript 자체로 변환된다.

TypeScript === JavaScript의 확대집합(SuperSet)



모든 브라우저
모든 호스트
모든 OS



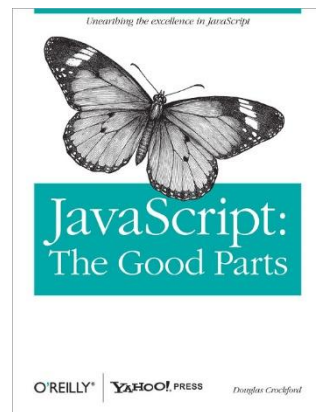
오픈 소스



강력한 형식으로
도구의 도움 지원



미래에서 온 기능들
사용 가능

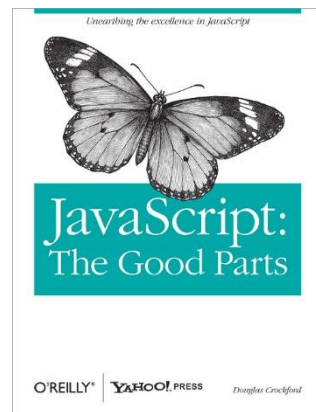


+



+





+



+



+

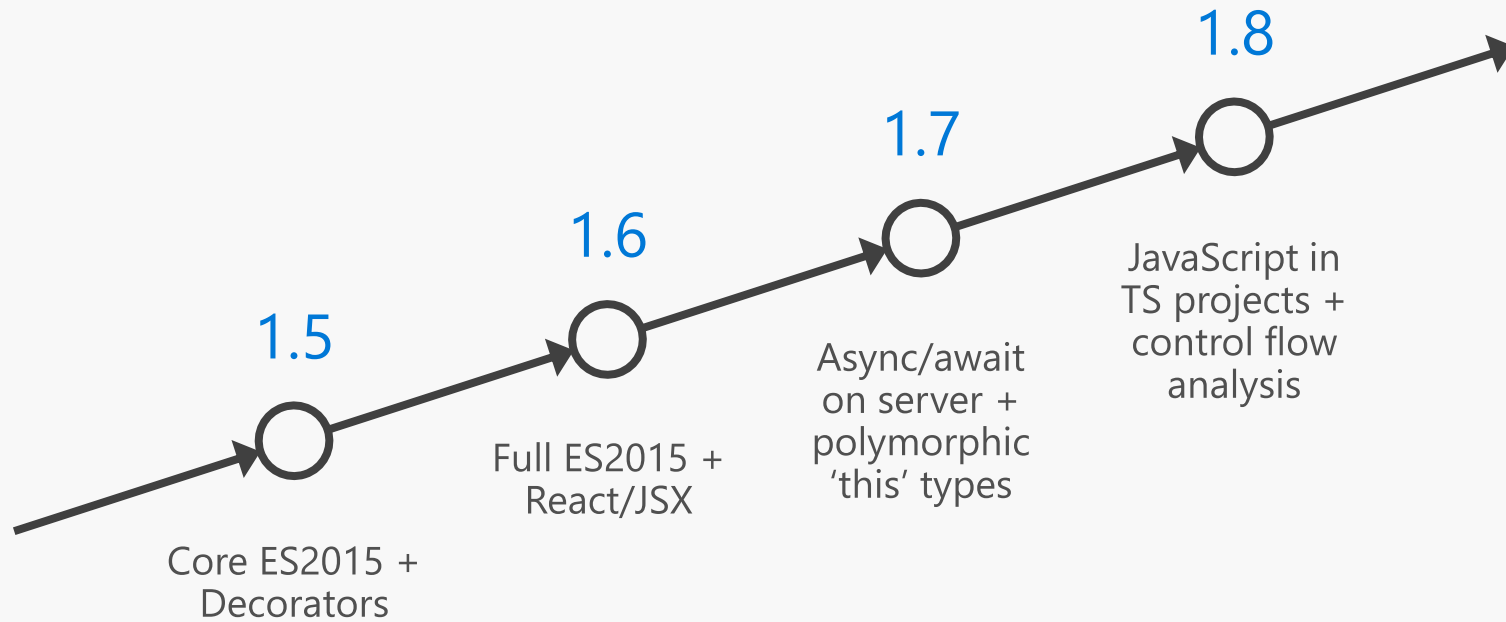


그래서 만들어진 작품이 TypeScript

현재 TypeScript 4.X 버전 사용 가능



빠른 릴리스: 1년에 4가지 릴리스 버전 출시



타입스크립트의 새로운 특징들...

ECMAScript 2015 support

tsconfig.json files

TS Server language service

Decorators

Local types

Generic type aliases

User defined type guard functions

JSX support

Intersection types

Abstract classes and methods

ES2016 exponentiation operator

Polymorphic this type

Async/await

Reachability analysis

Checking of destructuring with literal initializers

JavaScript in TypeScript compilations

String literal types

Stateless Functional Components in JSX

Support for F-bounded polymorphism

JSDoc support in JavaScript files

Support for default import interop with SystemJS

Recognize constructor functions in JavaScript files

Module augmentations

this-based type guards

Support for custom JSX factories

Improved checking of for-in statements

제가 생각하는 타입스크립트의 장점

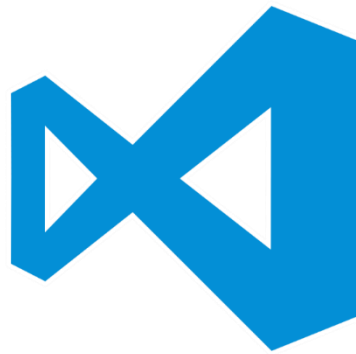
강력한 형식(Type)

+

묶어서 관리(Class)

+

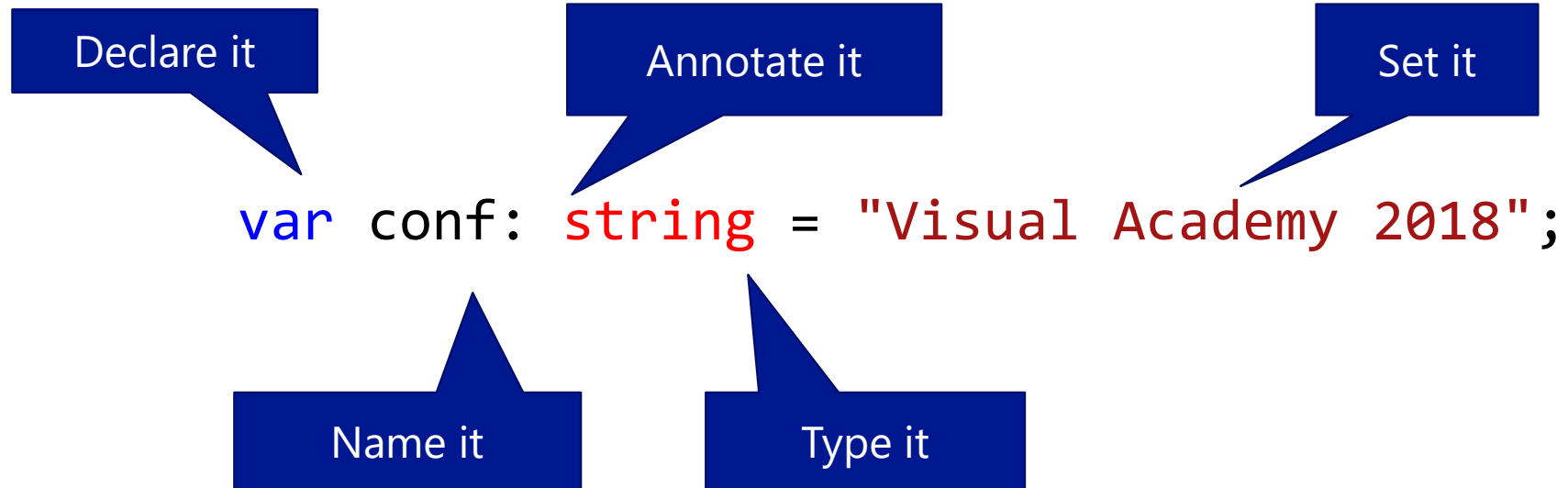
도구(Tool)



ANGULARJS
by Google

Why?

Type Annotations

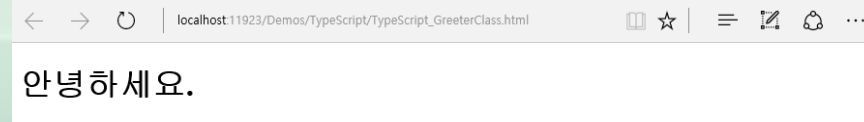


TypeScript_GreeterClass.ts

```
// 클래스 선언
class Greeter {
    // 생성자 선언할 때 바로 속성까지 추가
    constructor(public greeting: string) { }
    // 메서드 선언
    getGreeter() {
        return "<h1>" + this.greeting + "</h1>";
    }
}

// 클래스의 인스턴스 생성
var greeter = new Greeter("안녕하세요.");

// 개체의 멤버(메서드) 출력
document.body.innerHTML = greeter.getGreeter();
```



타입스크립트(TypeScript)

- JavaScript의 확대집합(Superset)
- 순수 JavaScript로 컴파일됨
- 강력한 형식(Strongly Typed)
- 클래스 기반 개체지향프로그래밍
- Function
- Interface
- Class
- Module
- Namespace
- Generic



타입스크립트 정의

Strongly
Typed

Classes

Interfaces

Generics

Modules

Type
Definitions

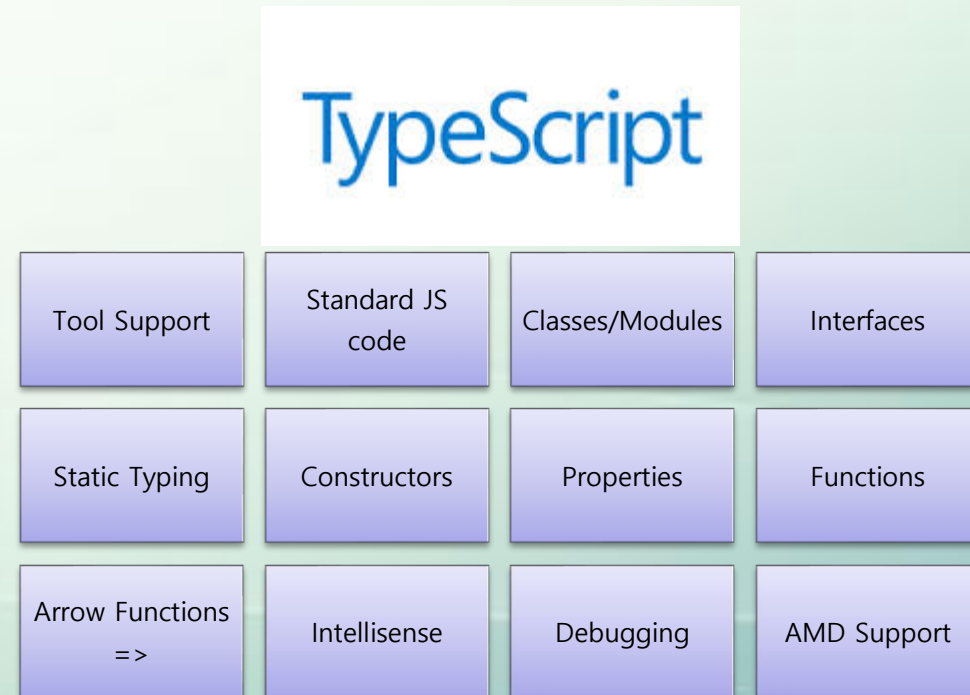
Compiles to
JavaScript

EcmaScript 6
Features



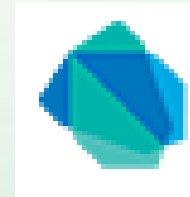
TypeScript 특징

- 정의
 - "TypeScript is a typed superset of JavaScript that compiles to plain JavaScript."
 - typescriptlang.org
- 공식 사이트
 - <http://typescriptlang.org>
- 언어 특징
 - 클래스(Classess)
 - 모듈(Moduels)
 - 데코레이터(Decorators)
 - 로직과 설정 분리



TypeScript 비교 대상

- 하드코딩 JavaScript 개발
- JavaScript 전처리기(PreProcessors)
 - CoffeeScript
 - <http://coffeescript.org>
 - Dart
 - <http://dartlang.org>
 - Clojurescript
 - <https://github.com/clojure/clojurescript>
 - Script#
 - <http://scriptsharp.com/>



왜?

- JavaScript를 잘 사용하는 개발자라면 TypeScript가 필요가 없을 수 있다.
- 하지만,
- C#/Java 기반 개발자라면 TypeScript의 장점을 몸으로 느낄 수 있다.

OOP 스타일의 JavaScript

유지보수가 쉬운 코드 작성

정적 타입 체크

코드 리팩터링

...



JavaScript와 TypeScript

- JavaScript와 TypeScript
 - 모든 JavaScript 코드는 TypeScript 코드임
 - 단순히 복사 및 붙여넣기로 가져올 수 있음
 - 모든 JavaScript 라이브러리는 TypeScript와 함께 사용 가능
- 정적(Static) 타입, 클래스, 모듈, 인터페이스
 - Visual Studio와 같은 도구의 힘을 빌려 디버깅/확장성 높은 응용프로그램 개발 가능
 - 실제 JavaScript로 만들어진 결과물은 static 형식이 제거됨
- JavaScript 시대
 - 컴파일된 TypeScript는 모두 100% JavaScript로 변환됨
 - Visual Studio + Web Essentials => 최상의 TypeScript IDE
 - 모든 브라우저, 호스트, OS에서 실행됨



TypeScript 놀이터(Playground)

The screenshot shows the TypeScript Playground interface. The browser address bar displays 'typescriptlang.org/Playground'. The page features the TypeScript logo and navigation links: 'learn', 'play', 'download', 'interact', 'tutorial', 'handbook', 'samples', and 'language spec'. Below the navigation, there are tabs for 'TypeScript' and 'JavaScript'. The 'TypeScript' tab is active, showing a code editor with the following code:

```
1 class Greeter {
2   greeting: string;
3   constructor(message: string) {
4     this.greeting = message;
5   }
6   greet() {
7     return "Hello, " + this.greeting;
8   }
9 }
10
11 var greeter = new Greeter("world");
12
13 var button = document.createElement('button');
14 button.textContent = "Say Hello";
15 button.onclick = function() {
16   alert(greeter.greet());
17 }
18
19 document.body.appendChild(button);
20
```

The 'JavaScript' tab is also visible, showing the equivalent JavaScript code:

```
1 var Greeter = (function () {
2   function Greeter(message) {
3     this.greeting = message;
4   }
5   Greeter.prototype.greet = function () {
6     return "Hello, " + this.greeting;
7   };
8   return Greeter;
9 })();
10 var greeter = new Greeter("world");
11 var button = document.createElement('button');
12 button.textContent = "Say Hello";
13 button.onclick = function () {
14   alert(greeter.greet());
15 };
16 document.body.appendChild(button);
17
```

Buttons for 'Share', 'Run', and 'JavaScript' are visible. The 'Walkthrough: Classes' dropdown is also present.



TS에서 JavaScript 그대로 사용

- 공백 제거 정도의 차이만 있음

TypeScript

Walkthrough: JavaScript

Share

```
1 function Greeter(greeting) {
2   this.greeting = greeting;
3 }
4
5 Greeter.prototype.greet = function() {
6   return "Hello, " + this.greeting;
7 }
8
9 // Oops, we're passing an object when we want a string.
10 // "Hello, [object Object]" instead of "Hello, world" w
11 var greeter = new Greeter({message: "world"});
12
13 var button = document.createElement('button');
14 button.textContent = "Say Hello";
15 button.onclick = function() {
16   alert(greeter.greet());
17 };
18
19 document.body.appendChild(button);
20
```

Run

JavaScript

```
1 function Greeter(greeting) {
2   this.greeting = greeting;
3 }
4 Greeter.prototype.greet = function () {
5   return "Hello, " + this.greeting;
6 };
7 // Oops, we're passing an object when we want a string.
8 // "Hello, [object Object]" instead of "Hello, world" w
9 var greeter = new Greeter({ message: "world" });
10 var button = document.createElement('button');
11 button.textContent = "Say Hello";
12 button.onclick = function () {
13   alert(greeter.greet());
14 };
15 document.body.appendChild(button);
16
```



TypeScript 설치(크로스 플랫폼)

- Node.js 설치(npm 설치)
 - npm: Node.js 패키지 관리자
 - <https://www.npmjs.com/>
 - <https://nodejs.org/download/>
- TypeScript 설치(tsc 설치)
 - npm install -g typescript
 - TypeScript 컴파일러는 node.js 패키지
 - npm i -g typescript
 - cmd: typescript -v
- TypeScript Definition Manager 설치(tsd 설치)
 - npm install -g tsd



tsc

- tsc -version
- tsc -help
- tsc -init
 - 현재 폴더에 tsconfig.json 파일 생성
- tsc -w
 - Watch Mode
 - 변경된 내용을 자동으로 컴파일



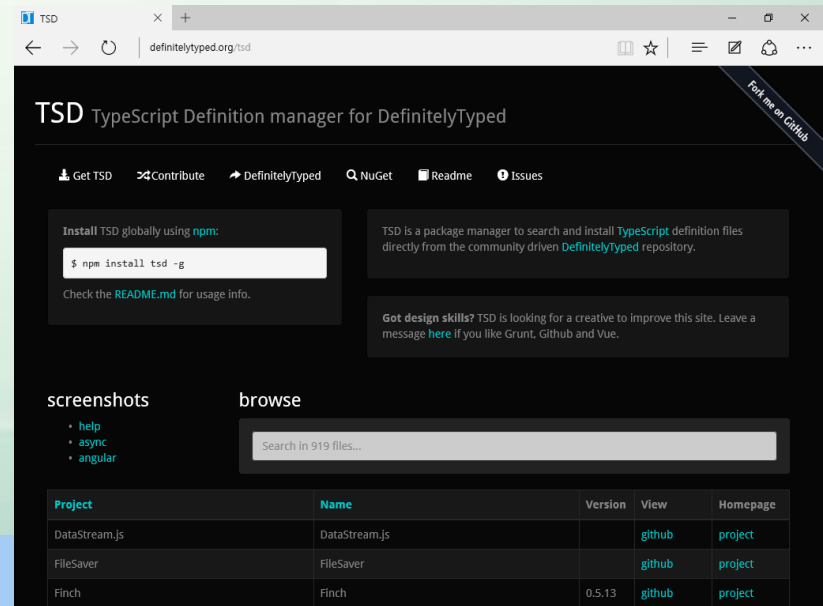
Type Definition File: 데코레이션, 헤더(h)

- TypeScript 내에서 다른 JavaScript 라이브러리에 대한 인텔리센스를 제공받고자한다면
 - TypeScript Definition File(*.d.ts) 파일을 인클루드
 - .d.ts
 - ~/typings/ 폴더에 주로 저장됨
 - 컴파일 타임에서만 사용
 - lib.d.ts를 통한 DOM 요소에 대한 기본 타입 제공
 - HTMLInputElement
 - VS-[정의로 이동]-[lib.d.ts] 확인
 - HTMLTableElement
 - DefinitelyTyped에서 주요 JavaScript용 ts 파일 다운로드
 - <http://definitelytyped.org/>
 - angular.d.ts
- 사용 예(jQuery 라이브러리 사용)
 - `/// <reference path="jquery.d.ts" />`
 - `declare var $;`
 - `var data = "안녕하세요.";`
 - `$("#lblMessage").text(data);`



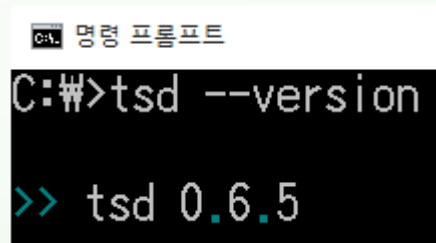
TypeScript Definition Manager(tsd)

- d.ts 파일 설치를 위한 새로운 형태의 패키지 관리자
 - DefinitelyTyped 리파지토리에서 다운로드
 - <http://definitelytyped.org/tsd/>
- tsd 설치
 - npm install -g tsd
 - tsd install angular --resolve --save
 - --resolve
 - 의존 라이브러리 추가
 - » jQuery
 - --save
 - tsd.d.ts
 - tsd.json



데모: tsd 설치

- npm install -g tsd
 - 설치 확인: npm list -g --depth 0

A screenshot of a Windows command prompt window. The title bar reads '명령 프롬프트' (Command Prompt). The command prompt shows the command 'C:\>tsd --version' being entered, followed by the output '>> tsd 0.6.5' on the next line.

```
C:\>tsd --version
>> tsd 0.6.5
```



타입스크립트 Hello World



TypeScript with Visual Studio Code

- 폴더 생성
- Visual Studio Code로 폴더 열기
- 설정 파일 생성(tsconfig.json)
- 태스크 러너 구성
- 빌드(Ctrl+Shift+B)
- 샘플 HTML 페이지 생성
- 심플 웹 서버 구성(http-server 또는 ASP.NET 5)
- 실행



TypeScript tools



Visual Studio



Sublime Text



Atom



WebStorm



Visual Studio Code



Eclipse



Emacs



Vim



TypeScript 에디터

- Visual Studio
- Visual Studio Code
- Sublime Text
- WebStorm
- Atom
- Vim
- EditPlus
- 메모장



tsc -w : Watch 모드

- tsconfig.json 파일에 정의된 ts 파일 자동 빌드



TypeScript 주요 특징 미리보기



TypeScript 주요 특징

- 인터페이스(Interfaces)
 - 인터페이스를 통해서 엔티티(Entity) 형태의 새로운 데이터 타입 정의
 - 기본 규칙 정의
- 클래스(Classess)
 - 일반적인 OOP 특징을 가짐
 - JavaScript의 차후 버전과 호환됨
- 제네릭(Generics)
 - 가볍고 재사용 가능한 코드 생성
- 모듈(Modules)
 - 관련된 인터페이스, 클래스, 함수 등을 묶어서 관리
 - 전역(Global) 스코프에서 작성된 코드 분리



TypeScript 키워드

Keyword	Description
class	클래스: 멤버 포함(필드, 생성자, 속성, 메서드(함수))
constructor	생성자(클래스 초기화)
export	모듈내에 모듈 밖에서도 호출(public)되는 멤버 생성
extends	상속시 사용(Java에서의 extends와 동일)
implements	인터페이스 구현 코드(C#에서는 그냥 콜론(:))
import	모듈에 포함(네임스페이스): 네임스페이스 축약형 생성: <code>import abc = Namespace.Sub;</code>
interface	인터페이스: 코드 규약(계약, 표준) 정의
module	컨테이너(클래스 또는 모든 코드)
public/private	액세스 접근 한정자(Member visibility)
=>	this 키워드 범위 내에서의 함수 구문 표현: 람다식(ES6 특징)
:	형식 지정 구분자(VB의 As 키워드와 동일)
...	트리플 닷(나머지 매개변수 문법) (C# "params" 키워드와 동일)
<type name>	형식 변환(Cast/Convert)



변수 선언

```
// str 변수를 string으로 처리  
var str: string = 'hello';
```

```
// 매개변수와 함수 반환값을 모두 string으로 처리  
function foo(name: string) : string {  
    return 'hello' + name;  
}
```



TypeScript 코드를 JavaScript 코드로 변환



```
class Greeter {  
  greeting: string;  
  constructor(message: string) {  
    this.greeting = message;  
  }  
  greet() {  
    return "안녕," + this.greeting;  
  }  
}
```

```
var Greeter = (function () {  
  function Greeter(message) {  
    this.greeting = message;  
  }  
  Greeter.prototype.greet = function () {  
    return "안녕," + this.greeting;  
  };  
  return Greeter;  
})();
```



타입 시스템



let : 끝을 허용하지 않음



JavaScript/TypeScript의 타입 시스템

- Number => number
- Boolean => boolean
- String => string
- Function => void
 - functionName : (s: string, t: string) => void = function(s, t) {};
- Object => any
- Array => any[]
- 등등

TS VariableExample.ts ✕

```
1  // 데이터 형식을 갖는 변수
2  let n: number = 1234;
3  let b: boolean = true;
4  let s: string = '안녕하세요.';
5  let a: any;
6      a = 1234;
7      a = true;
8      a = '반갑습니다.';
9
10 // 데이터 형식을 갖는 상수
11 const PI: number = 3.14;
12
13 // 데이터 형식을 갖는 배열
14 let arr: number[] = [1, 2, 3];
15 let all: any[] = [ 1234, true, '또 만나요.'];
```



TypeScript 타입

Keyword	Description
any	모든 JS 값(C# 예, System.Object)
number	정수형(integer) 또는 실수형(floating point number) Double-precision 64-bit
boolean	참(true) 또는 거짓(false)
string	텍스트(문자열)
null	JS null 리터럴
undefined	JS undefined 리터럴
(object types)	모든 클래스, 모듈, 인터페이스 등등
void	반환값이 없는 함수
(Array)	배열형, 컬렉션 타입
enum	열거형



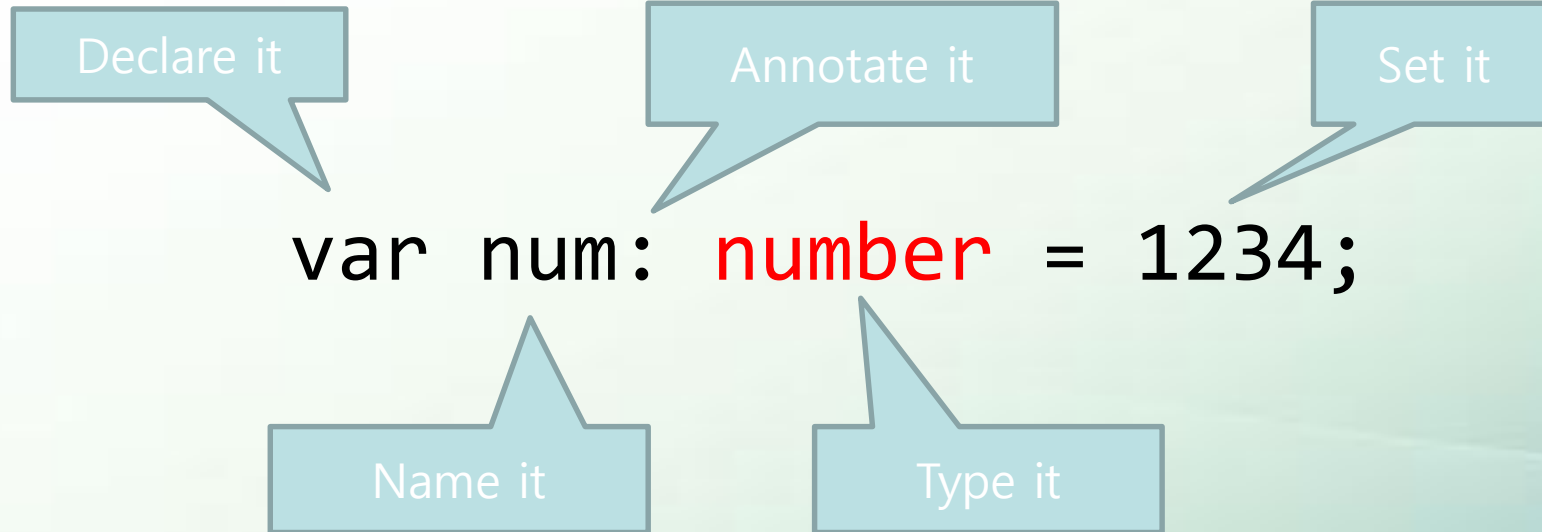
TypeScript 타입 시스템

- 강력하게 형식화된(Strongly Typed)
 - C#/Java와 같음
 - TypeScript(Static Typing)
 - JavaScript(Dynamic Typing)
- 개발시에만 존재하는 타입시스템
 - 컴파일된 후에는 일반적인 JavaScript 타입 시스템
- declare
 - TypeScript
 - declare var document; document.title = "제목";
 - JavaScript
 - document.title = "제목";



문법: 타입 추론(Type Inference)

Type Annotations



문법: 타입 추론(Type Inference)

num === string

Declare it

Set it

let num = "1234";

Name it



Any 타입

모든값(any)으로 설정

```
var anyData;
```



기본 타입들

```
var currentName: string;  
var hasPassed: boolean;  
var averageMark: number;  
var currentCourses: string[]; // Array<string>  
var additionalInfo: any;  
var currentState: State;  
  
enum State { Onsight, Online, NotEnrolled }  
  
function setStudent(  
    name: string, passed: boolean,  
    mark: number, courses: string[], info: any,  
    state: State) : void {  
    ...  
}
```



기본 타입들

TypeScript 소스

```
var typeScriptBoolean: boolean = true;  
var typeScriptNumber: number = 1234;  
var typeScriptString: string = "Hello World";  
var anyOldType: any = "모든 타입";
```

JavaScript 소스: 컴파일된

```
var typeScriptBoolean = true;  
var typeScriptNumber = 1234;  
var typeScriptString = "Hello World";  
var anyOldType = "모든 타입";
```



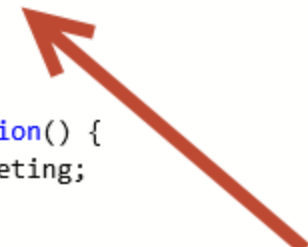
변수 및 매개변수 선언시 강력한 형식 사용

TypeScript

Walkthrough: Types

Share

```
1 function Greeter(greeting: string) {  
2     this.greeting = greeting;  
3 }  
4  
5 Greeter.prototype.greet = function() {  
6     return "Hello, " + this.greeting;  
7 }  
8  
9 var greeter = new Greeter("world");  
10  
11 var button = document.createElement('button');  
12 button.textContent = "Say Hello";  
13 button.onclick = function() {  
14     alert(greeter.greet());  
15 };  
16  
17 document.body.appendChild(button);  
18
```



Run

JavaScript



데모: 변수 선언 및 초기화

- C:\TypeScript\Variable.ts

```
Variable.ts  X
<전역>
1 // 기존 JavaScript의 변수 선언
2 var i = 10;
3
4 // any 타입: 모든 값
5 var j: any;
6 j = "안녕";
7
8 // number: 숫자
9 var k: number;
10 //k = "안녕"; // 에러
11 k = 1234;
12
13 // 선언과 동시에 초기화
14 var b: boolean = true;
15 b = false;
16
17 // 문자열
18 var str: string = "안녕하세요.";
19
20 // 배열형
21 var arr: string[] = ["안녕", "잘가"];
22
23 // 모든 형식 가능
24 |

arr (var)
1 // 기존 JavaScript의 변수 선언
2 var i = 10;
3
4 // any 타입: 모든 값
5 var j;
6 j = "안녕";
7
8 // number: 숫자
9 var k;
10
11 //k = "안녕"; // 에러
12 k = 1234;
13
14 // 선언과 동시에 초기화
15 var b = true;
16 b = false;
17
18 // 문자열
19 var str = "안녕하세요.";
20
21 // 배열형
22 var arr = ["안녕", "잘가"];
23 // 모든 형식 가능
24 //# sourceMappingURL=Variable.js.map
```



문자열 보간법(String Interpolation)

문자열 보간법 또는 보간된 문자열로 불리는 기능은 문자열을 묶을 때 편리하게 사용될 수 있는 기능입니다. 문자열 보간법은 또 다른 표현법으로 문자열 템플릿(String Template) 또는 템플릿 문자열(Template String)이라고도 합니다. 프로그래밍을 하다보면 문자열을 묶어서 결과를 출력할 일이 아주 많이 있습니다. 타입스크립트는 템플릿 문자열로 불리는 문자열 보간법을 제공해서 `\${}` 형태로 문자열을 묶어서 출력하는 방법을 제공합니다. 이 방식은 처음에는 복잡해 보일 수 있으나 사용하면 사용할수록 편리함을 느낄 것입니다.

```
> var message = "Hello";  
> `${message}`  
'Hello'
```

문자열 보간법 또는 보간된 문자열은 타입스크립트에서 효과적으로 문자열을 묶어서 표현할 수 있는 기능을 제공합니다. `console.log()` 메서드에서 더하기(+) 연산자로 문자열을 묶지 않고 직접 특정 변수의 값을 중괄호 기호를 사용하여 표현할 수 있습니다. 이때 중괄호 앞에는 \$ 기호가 와야 합니다.

<코드> StringInterpolationDescription.ts

```
// 템플릿 문자열(문자열 보간법, 보간된 문자열)
```

```
let num = 3;
```

```
let result = "홀수";
```

```
console.log(`${num}은(는) ${result}입니다.`);
```

</코드>

<실행>

3은(는) 홀수입니다.

</실행>

문자열 보간법에 사용되는 변수의 값은 모두 문자열로 처리됩니다.

배열: any[]

- var a: any
 - 모든 자바스크립트 값 저장 가능
- var arr: any[]
 - 배열형
- arr.
 - 모든 배열 관련 속성/메서드 제공
 - arr[0] 식으로 인덱서로 접근 가능



배열인지 확인: `arg is Array<any>`

- `isArray(arg: any): arg is Array<any>;`
 - 매개변수가 배열이면 참을 반환하는 필드 생성 가능



배열(Array) 사용 예

```
var list: number[] = [1, 2, 3];
```

```
var list: Array<number> = [1, 2, 3];
```

```
var list = [1, 2, 3];
```



함수



함수(Functions)

- 매개변수 타입 설정 가능
 - 옵션(Optional) 또는 기본(Default) 매개변수 설정 가능
 - 컬렉션 파라미터 설정 가능: ...
 - params(C#)
- 리턴값 설정 가능
 - void: 반환값이 없음을 의미
- 람다식 사용 가능
 - =>
 - (goes to)



Optional Parameter

- `rect { h: number; w?: number; }`
 - `w` 변수는 Optional



Arrow Function

```
var myFunc = function(a: number) {  
    return a * a  
} → var myFunc = function(a) {  
    return a * a;  
}
```

- 또는 -

```
var myFunc = (a: number) => a * a;
```



함수(Functions)

```
function calculateSum(x: number, y: number, z?: number,
    ...restNumbers: number[]): number {
    var sum = x + y;
    for (var i = 0; i < restNumbers.length; i++) {
        sum += restNumbers[i];
    }
    return sum;
}

var calc: (x: number, y: number) => number = calculateSum;

var calcSum = (x, y) => x + y;
```



데모: 타입스크립트 함수 사용

- C:\TypeScript\FunctionDemo.ts
 - 매개변수 형식
 - 널가능 형식
 - 기본 매개변수 값
 - 가변 길이 매개변수

```
// Definition
function printName(
  name: string, age: number = 20, ...email: string[]): string {
  //return "name: " + name + ", age: " + age + ", emails: " + email.join(", ");
  return `name: ${name}, age: ${age}, emails: ${email.join(', ')} `;
}

// Call
var r = printName("Red", 21, "a@a.com", "b@b.com", "c@c.com");
```



함수(Function) 타입

```
interface SearchFunc {  
    (source: string, subString:string): boolean;  
}  
  
var mySearch: SearchFunc;  
mySearch = function(src: string, sub: string) {  
    //some implementation  
    return true;  
}  
  
var mySearch;  
mySearch = function (src, sub) {  
    //some implementation  
    return true;  
};
```



열거형(Enum)

TypeScript 소스

```
enum Color { Red, Green, Blue };  
var c: Color = Color.Green;
```

JavaScript 소스: 컴파일된

```
var Color;  
(function (Color) {  
    Color[Color["Red"] = 0] = "Red";  
    Color[Color["Green"] = 1] = "Green";  
    Color[Color["Blue"] = 2] = "Blue";  
})(Color || (Color = {}));  
;  
var c = 1 /* Green */;
```



열거형(Enum)

- `enum Color { Red, Green, Blue }; // 0, 1, 2`
- `enum Color { Red = 1, Green, Blue }; // 1, 2, 3`
- `enum Color { Red = 3, Green = 5, Blue = 7 }; // 3, 5, 7`
- `let myColor: Color = Color.Red;`
- `console.log(myColor); // 3`
- `Color[3] === "Red";`

클래스(Class)



클래스(Class)

- 클래스는 컨테이너(Container) 역할
 - 속성들 구현
 - 생성자
 - 메서드
- 인스턴스 가능
- 다른 클래스로 상속 가능
 - `super()`로 base 클래스 접속
- 액세스 접근 한정자: `private`, `public`
- `static` 멤버 생성 가능
- `getter`와 `setter` 구현 가능



클래스 멤버

필드(Fields)

생성자
(Constructors)

속성
(Properties)

함수
(Functions)



클래스(Class)

```
interface ClockInterface {  
    currentTime: Date;  
}  
class Clock implements ClockInterface {  
    currentTime: Date;  
    constructor(h: number, m: number) { }  
}
```

```
var Clock = (function () {  
    function Clock(h, m) {  
    }  
    return Clock;  
})();
```

클래스와 JavaScript

TypeScript

Walkthrough: Classes

Share

```
1 class Greeter {
2   greeting: string;
3   constructor(message: string) {
4     this.greeting = message;
5   }
6   greet() {
7     return "Hello, " + this.greeting;
8   }
9 }
10
11 var greeter = new Greeter("world");
12
13 var button = document.createElement('button');
14 button.textContent = "Say Hello";
15 button.onclick = function() {
16   alert(greeter.greet());
17 }
18
19 document.body.appendChild(button);
20
```

Run

JavaScript

```
1 var Greeter = (function () {
2   function Greeter(message) {
3     this.greeting = message;
4   }
5   Greeter.prototype.greet = function () {
6     return "Hello, " + this.greeting;
7   };
8   return Greeter;
9 })();
10 var greeter = new Greeter("world");
11 var button = document.createElement('button');
12 button.textContent = "Say Hello";
13 button.onclick = function () {
14   alert(greeter.greet());
15 };
16 document.body.appendChild(button);
17
```



클래스 상속: extends

TypeScript

Walkthrough: Inheritance

Share

```
1 class Animal {
2     constructor(public name: string) { }
3     move(meters: number) {
4         alert(this.name + " moved " + meters + "m.");
5     }
6 }
7 class Snake extends Animal {
8     constructor(name: string) { super(name); }
9     move() {
10         alert("Slithering...");
11         super.move(5);
12     }
13 }
14 class Horse extends Animal {
15     constructor(name: string) { super(name); }
16     move() {
17         alert("Galloping...");
18         super.move(45);
19     }
20 }
21 var sam = new Snake("Sammy the Python");
22 var tom: Animal = new Horse("Tommy the Palomino");
23 sam.move();
24 tom.move(34);
```

Run

JavaScript

```
1 var __extends = (this && this.__extends) || function (d
2     for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[
3     function __() { this.constructor = d; }
4     __.prototype = b.prototype;
5     d.prototype = new __();
6 };
7 var Animal = (function () {
8     function Animal(name) {
9         this.name = name;
10    }
11    Animal.prototype.move = function (meters) {
12        alert(this.name + " moved " + meters + "m.");
13    };
14    return Animal;
15 })();
16 var Snake = (function (_super) {
17     __extends(Snake, _super);
18     function Snake(name) {
19         _super.call(this, name);
20    }
21    Snake.prototype.move = function () {
22        alert("Slithering...");
23        _super.prototype.move.call(this, 5);
24    };
25    return Snake;
```



클래스(Classess)

```
class CarDriver extends BasePerson implements Driver {  
    private static LicenseNumber: string = '1234-5678';  
    private _health: number;  
    vehicles: Vehicle[];  
  
    constructor(name: string, public experience: number) {  
        super(name);  
    }  
  
    get health() {...}  
    set health(newHealth: number) {...}  
  
    static CurrentLicenseNumber(): string {...}  
    addVehicle(vehicle: Vehicle) {...}  
    greet() : string {  
        return super.greet() + ...;  
    }  
}
```



클래스와 인터페이스

```
interface IGreeter {  
    greet(): void;  
}  
  
class Greeter implements IGreeter {  
    greeting: string;  
    greet() {  
        console.log(this.greeting);  
    }  
}
```



```
var Greeter = (function () {  
    function Greeter() {  
    }  
    Greeter.prototype.greet = function () {  
        console.log(this.greeting);  
    };  
    return Greeter;  
})();
```



인터페이스를 implements한 클래스

- 인터페이스 상속
 - implements 키워드 사용

```
1 interface ICalculator
2 {
3     Add(): void;
4     Subtract(): void;
5     Multiply(): void;
6     Divide(): void;
7 }
8
9 class Calculator implements ICalculator
10 {
11     Add(): void {
12
13     }
14     Subtract(): void {
15
16     }
17     Multiply(): void {
18
19     }
20     Divide(): void {
21
22     }
23 }
24
25 var calc = new Calculator();
26 calc.
```

Add

Divide

Multiply

Subtract

(method) Calculator.Add(): void



생성자 매개변수 + public

- public을 붙인 생성자 매개변수
 - 자동으로 필드(private)를 생성함



부모 클래스와 자식 클래스

```
InheritanceDemo.ts
TypeScript
<global>
InheritanceDemo

1  // [?] 상속(Inheritance): 부모 클래스의 기능을 자식 클래스에서 물려받아 사용
2  module InheritanceDemo {
3      // 부모 클래스 선언
4      class Parent {
5          foo = (): void => console.log('부모 클래스의 멤버 호출');
6      }
7
8      // 자식 클래스 선언
9      class Child extends Parent {
10         bar = (): void => console.log('자식 클래스의 멤버 호출');
11     }
12
13     // 자식 클래스의 인스턴스 생성
14     let child = new Child();
15     child.foo(); // 부모 클래스의 멤버 호출
16     child.bar(); // 자식 클래스의 멤버 호출
17 }
```

```
C:\Windows\System32\cmd.exe
C:\dev\VisualAcademy\TypeScript\InheritanceDemo>node InheritanceDemo.js
부모 클래스의 멤버 호출
자식 클래스의 멤버 호출
C:\dev\VisualAcademy\TypeScript\InheritanceDemo>
```



인터페이스(Interface)



인터페이스(Interface)

- 속성과 메서드에 대한 스펙 정의 용도
 - 속성들(옵셔널 포함) 정의 가능
 - 메서드(함수) 정의
 - 인덱서 정의
- 데이터 형식처럼 사용
- 다른 인터페이스로 확장
- 기존 JavaScript의 클래스를 TypeScript로 옮길 때는 interface 키워드 붙이면 됨
- 개발시에 ts 파일에만 존재(Development Time)
 - 컴파일된 js 파일에는 interface 키워드가 없음



인터페이스(Interface)

```
export interface ILabelledValue {  
    label: string;  
}  
function printLabel(  
    labelledObj: ILabelledValue) {  
    console.log(labelledObj.label);  
}  
var myObj = {  
    size: 10, label: "Size 10" };  
printLabel(myObj);
```

인터페이스(Interface) – 컴파일된 소스

```
function printLabel(labelledObj) {  
    console.log(labelledObj.label);  
}  
  
var myObj = {  
    size: 10, label: "Size 10" };  
printLabel(myObj);
```



인터페이스

```
interface Person {  
    firstName: string;  
    lastName: string;  
    age?: number;  
}
```

```
interface Driver extends Person {  
    yearsExperience: number;  
    vehicles: Vehicle[];  
    addVehicle(vehicle: Vehicle): void;  
    removeVehicle(vehicle: Vehicle): Vehicle;  
}
```



```
InterfaceNote.ts  -  X
1  // [?] 인터페이스: 특정 멤버가 반드시 구현되어야 함을 보증
2  module InterfaceNote {
3      // ICar 인터페이스 선언
4      interface ICar {
5          go(): void; // 함수 시그니처만 제공
6      }
7
8      // ICar 인터페이스를 상속하는 Car 클래스 선언
9      class Car implements ICar {
10         go(): void {
11             console.log('상속한 인터페이스에 정의된 모든 멤버를 반드시 구현해야 합니다.');

```
C:\Windows\System32\cmd.exe
C:\dev\VisualAcademy\TypeScript\InterfaceNote>tsc InterfaceNote.ts
C:\dev\VisualAcademy\TypeScript\InterfaceNote>node InterfaceNote.js
상속한 인터페이스에 정의된 모든 멤버를 반드시 구현해야 합니다.
C:\dev\VisualAcademy\TypeScript\InterfaceNote>
```


```



제네릭(Generics)

- 재 사용성
- 제네릭 클래스
- 제네릭 함수
- 타입 체킹 및 제약조건 부여



제네릭(Generic) 사용 전

// 제네릭 사용 전: 각각의 스타일대로 구현

```
class AccountString
```

```
{
```

```
    tag: string;
```

```
}
```

```
var as = new AccountString();
```

```
as.tag = "문자열만"; // 1234;
```

```
class AccountNumber
```

```
{
```

```
    tag: number;
```

```
}
```

```
var an = new AccountNumber();
```

```
an.tag = 1234; // "안녕";
```



제네릭(Generic) 사용 후

```
// 제네릭 클래스: Cup of T => Cup<T>
class Account<T>
{
    tag: T;
}
var a1 = new Account<string>();
a1.tag = "안녕하세요.";
var a2 = new Account<number>();
a2.tag = 1234;
var a3 = new Account<boolean>();
a3.tag = true;
```



제네릭(Generics)

```
class List<T> {  
    private _collection: T[];  
  
    add(item: T) {  
        this._collection.push(item);  
    }  
  
    remove(item: T) {  
        ...  
    }  
  
    get count() {  
        return this._collection.length;  
    }  
}
```



제네릭 클래스

TypeScript

Walkthrough: Generics

Share

```
1 class Greeter<T> {
2     greeting: T;
3     constructor(message: T) {
4         this.greeting = message;
5     }
6     greet() {
7         return this.greeting;
8     }
9 }
10
11 var greeter = new Greeter<string>("Hello, world");
12
13 var button = document.createElement('button');
14 button.textContent = "Say Hello";
15 button.onclick = function () {
16     alert(greeter.greet());
17 }
18
19 document.body.appendChild(button);
20
```

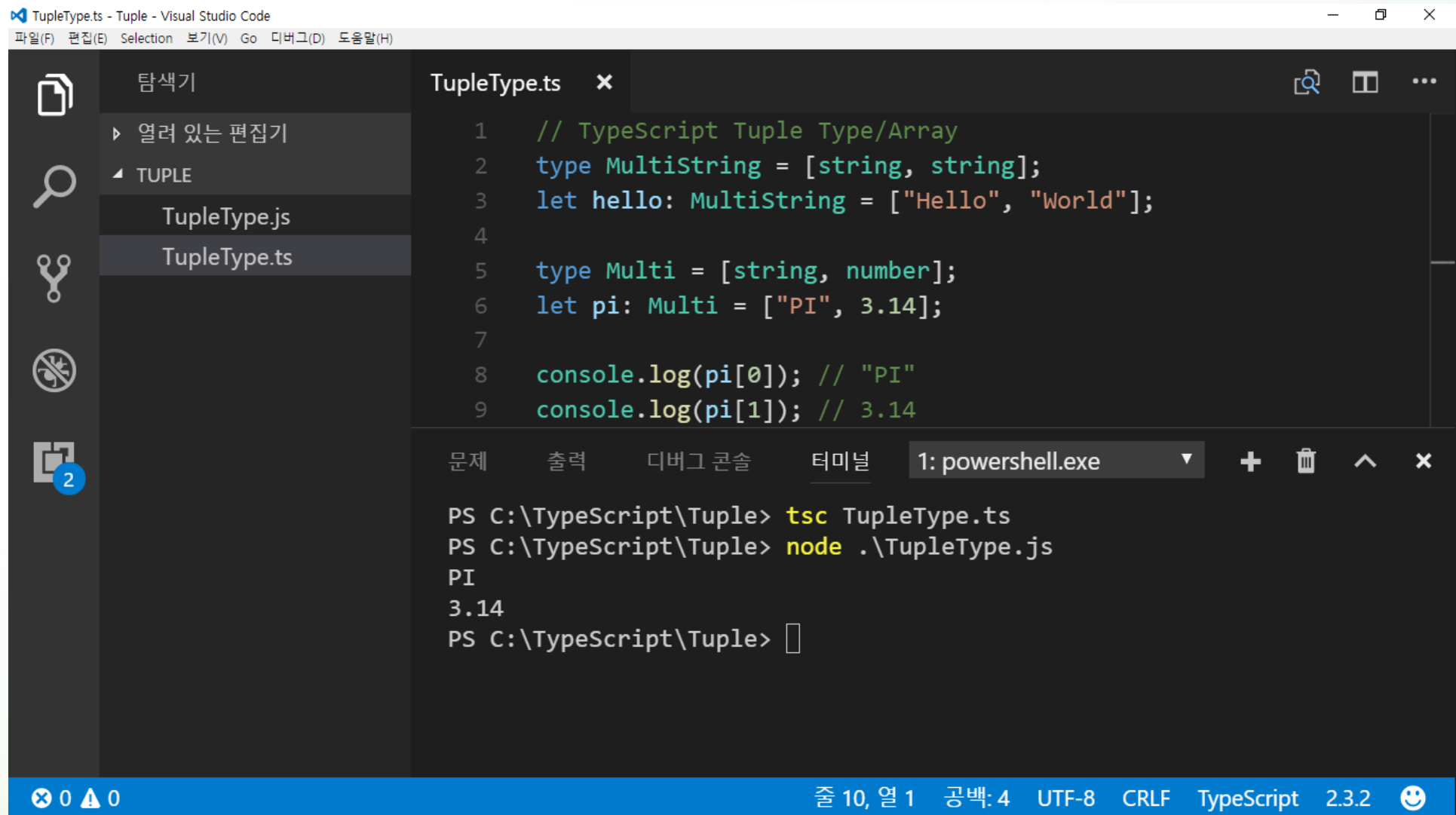
Run

JavaScript

```
1 var Greeter = (function () {
2     function Greeter(message) {
3         this.greeting = message;
4     }
5     Greeter.prototype.greet = function () {
6         return this.greeting;
7     };
8     return Greeter;
9 })();
10 var greeter = new Greeter("Hello, world");
11 var button = document.createElement('button');
12 button.textContent = "Say Hello";
13 button.onclick = function () {
14     alert(greeter.greet());
15 };
16 document.body.appendChild(button);
17
```



튜플



The screenshot shows the Visual Studio Code interface with a file named `TupleType.ts` open. The left sidebar shows the Explorer view with the file `TupleType.ts` selected. The main editor displays the following TypeScript code:

```
1 // TypeScript Tuple Type/Array
2 type MultiString = [string, string];
3 let hello: MultiString = ["Hello", "World"];
4
5 type Multi = [string, number];
6 let pi: Multi = ["PI", 3.14];
7
8 console.log(pi[0]); // "PI"
9 console.log(pi[1]); // 3.14
```

Below the editor, the Output view shows the terminal output for the command `node .\TupleType.js`:

```
PS C:\TypeScript\Tuple> tsc TupleType.ts
PS C:\TypeScript\Tuple> node .\TupleType.js
PI
3.14
PS C:\TypeScript\Tuple>
```

The status bar at the bottom indicates 0 errors, 0 warnings, and 0 info messages. The encoding is UTF-8, and the language mode is TypeScript 2.3.2.



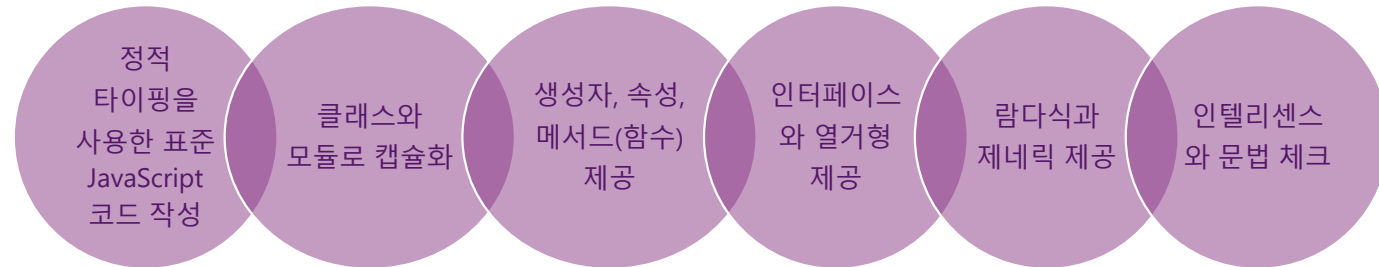
TypeScript 새로운 특징

특징

Classes

Modules

Decorators



TypeScript 새로운 특징

특징

Classes

Modules

Decorators



Classes

```
class C {  
    constructor () { ... }  
    property: string;  
    method () { ... }  
}
```

TypeScript 새로운 특징

특징

Classes

Modules

Decorators



Modules

```
import { add, divide } from  
"math";
```

```
import * as Math from "math";
```

```
export function bar() { ... }
```

TypeScript 새로운 특징

특징

Classes

Modules

Decorators



Decorators

```
@View({template: "<b>hi</b>"})  
class C {  
    ...  
}
```


모듈(Module)



모듈

전역 스코프 분리

캡슐화

재 사용성 높이기

의존성 관리

AMD, CommonJS 제공



모듈(Modules)

- 변수, 인터페이스, 클래스 등을 묶어서 캡슐화
 - C#/Java의 네임스페이스/패키지와 같은 개념
 - System.Data, System.IO 등
 - 서브시스템으로 코드를 조직화
 - module 키워드로 생성
- Internal 모듈과 External 모듈로 구분됨
 - export 키워드로 모듈 내에 선언된 멤버를 모듈 밖에서도 public하게 사용 가능하도록 설정
- 하나의 모듈을 서로 다른 파일로 분리 가능
- 컴파일 후 하나로 통합 가능
- 외부 라이브러리 사용 가능
- 하나의 스코프로 묶음 가능



모듈(Module)

```
module Time {  
    export interface ClockInterface {  
        currentTime: Date;  
    }  
    export class Clock implements ClockInterface {  
        currentTime: Date;  
        constructor(h: number, m: number) { }  
    }  
}  
  
var Time;  
(function (Time) {  
    var Clock = (function () {  
        function Clock(h, m) {  
        }  
        return Clock;  
    })();  
    Time.Clock = Clock;  
})(Time || (Time = {}));
```



모듈(Modules)

```
module Drivers {  
    export class BasePerson implements Person {  
        ...  
    }  
  
    export class CarDriver extends BasePerson {  
        ...  
    }  
}
```

```
var someDriver = new Drivers.CarDriver(...);
```



모듈

- 클래스, 인터페이스를 묶는 역할로 모듈 정의
 - import, export 키워드 사용

```
module app {  
  export interface IGreeter {  
    greet(): void;  
  }  
  export class Greeter implements IGreeter {  
    greeting: string;  
    greet() {  
      console.log(this.greeting);  
    }  
  }  
}
```



```
var app;  
(function (app) {  
  var Greeter = (function () {  
    function Greeter() {  
    }  
    Greeter.prototype.greet = function () {  
      console.log(this.greeting);  
    };  
    return Greeter;  
  })();  
  app.Greeter = Greeter;  
})(app || (app = {}));
```



모듈로 묶어서 관리

TypeScript

Walkthrough: Modules

Share

```
1 module Sayings {
2   export class Greeter {
3     greeting: string;
4     constructor(message: string) {
5       this.greeting = message;
6     }
7     greet() {
8       return "Hello, " + this.greeting;
9     }
10  }
11 }
12 var greeter = new Sayings.Greeter("world");
13
14 var button = document.createElement('button');
15 button.textContent = "Say Hello";
16 button.onclick = function() {
17   alert(greeter.greet());
18 };
19
20 document.body.appendChild(button);
```

Run

JavaScript

```
1 var Sayings;
2 (function (Sayings) {
3   var Greeter = (function () {
4     function Greeter(message) {
5       this.greeting = message;
6     }
7     Greeter.prototype.greet = function () {
8       return "Hello, " + this.greeting;
9     };
10    return Greeter;
11  })();
12  Sayings.Greeter = Greeter;
13 })(Sayings || (Sayings = {}));
14 var greeter = new Sayings.Greeter("world");
15 var button = document.createElement('button');
16 button.textContent = "Say Hello";
17 button.onclick = function () {
18   alert(greeter.greet());
19 };
20 document.body.appendChild(button);
```



모듈과 모듈 로더

- AMD
- CommonJS
- System
- UMD
- ES2015
- RequireJS
- SystemJS
- Webpack
- Browserify



캐스팅(Casting;Convert): Type Assertion

- `var table: HTMLTableElement = get('table');`
 - -에서-
- `var table: HTMLTableElement`
`= < HTMLTableElement>get('table');`
 - -으로-
- 애 니 좋아하세요?
 - 그러면 모든 형태에 `<any>`를 붙이면 됩니다.



let 키워드: var 키워드와 거의 비슷

- 함수 레벨 스코프 핸들링(블록 내에서만 사용)

```
function f() {  
    let index = 0; // number  
    for (var i = 0; i < 10; i++) {  
        let index = "abc"; // string  
        index += "def";  
    }  
}
```



```
function f() {  
    var index = 0; // number  
    for (var i = 0; i < 10; i++) {  
        var index_1 = "abc"; // string  
        index_1 += "def";  
    }  
}
```



유니온 타입(Union Type)

TypeScript 소스

```
var x : number | string;  
type NameOrNameArray = string | string[];  
function createName(name: NameOrNameArray){}
```

비교 사용

```
if (typeof name === "string")
```

유니온 타입(Union Type)

- `var x : number | null;`
 - `x = 1;`
 - `x = null;`



모듈: import {} from “외부모듈”

moduletest.ts	util.ts	tsconfig.json
<pre>7 import {add, divide} from "util"; 8 9 add(1, 2); 10 divide(2, 4); 11</pre>	<pre>1 export function add(x: number, y: number) { 2 return x + y; 3 } 4 5 export function divide(x: number, y: number) { 6 return x / y; 7 } 8</pre>	<pre>1 { 2 "compilerOptions": { 3 "module": "amd", 4 "target": "ES5" 5 }, 6 "files": [7 "./moduletest.ts", 8 "./util.ts" 9] 10 }</pre>



모듈: ES6 모듈과 Node 함께 사용하기

- tsd install node
- code . tsconfig.json server.ts client.ts

tsconfig.json

```
1 {
2   "compilerOptions": {
3     "target": "ES5",
4     "module": "commonjs"
5   },
6   "files": [
7     "server.ts", "client.ts"
8   ]
9 }
10
```

server.ts

```
1 /// <reference path="./typings/node/node.d.ts" />
2 import { createServer } from "http";
3
4 export function simpleServer(port:number, message: string) {
5   createServer((req, res) => {
6     res.writeHead(200, { "Content-Type": "text/html" });
7     res.write("<h1>" + message + "</h1>");
8     res.end();
9   }).listen(port);
10 }
11
```

client.ts

```
1 import { simpleServer } from "./server";
2 simpleServer(1337, "Hello TypeScript on DevLec...");
3 console.log("Listening on 1337...");
4
```



TSD로 관련 모듈 검색 및 설치

- `tsd query node express`
 - 모듈 검색
- `tsd query node express --action install`
 - 검색과 동시에 설치



기타: 리팩터링 by Visual Studio

참조 확인

(Peek Definition)

(Alt+F12)

정의로 이동

(Go to Definition)

(F12)

모든 참조 찾기

(Find All References)

(Shift+F12)

이름 변경

(Rename Symbol)

(F2)



HTML 폼과 연동

- `document.querySelector("#id");`
- `<HTMLButtonElement>`
- `<HTMLInputElement>`
- `<HTMLUListElement>`



데모: TypeScript의 모든 기능 사용한 예

- HTML5 기반 인공지능(AI) 장기 게임

- 5개 이상 모듈 사용
- 5개 이상 인터페이스 사용
- 7개 이상 클래스 사용
- 상속 사용
- 30개 이상 메서드 구현
- Static 멤버 구현
- 제네릭 사용
- 모두 강력한 형식으로 구현



Type System

강력한 형식의
타입 시스템을
제공

- Structural typing and type inference
 - In practice very few type annotations are necessary
- Interfaces, generics, union types, tuple types
 - Increases accuracy and expressiveness of type system
- Works with existing JavaScript libraries
 - Declaration files can be written and maintained separately
- Types enable tooling
 - Provide verification and assistance, but not hard guarantees

Classes, Interfaces, Modules

명확한 타입을
지정하여
JavaScript를 구현하고,
이를 순수 JavaScript의
동적인 타입시스템으로
변경

- Scalable application structuring
 - Classes, interfaces, and modules enable clear contracts in code
- Aligned with emerging standards
 - Class and lambda syntax aligns with ECMAScript 6 proposals
- Supports popular module systems
 - CommonJS and AMD modules in any ECMAScript 3 environment

최신 JavaScript 특징 포함

CommonJS 또는
AMD 모듈을 사용하여
ECMAScript 5 코드를
컴파일하여 EC3 또는
EC5 코드로 변환

- Modules
- Classes
- Arrow functions
- Default parameters
- Destructuring
- Spread and rest
- Let and const
- for...of
- Object literal methods
- Shorthand properties
- Computed properties
- Octal / binary literals
- Symbols
- Template strings
- Generators
- Async/await
 - ES7

TypeScript Ecosystem

큰 규모의
JavaScript 앱
개발을 위한
오픈소스 언어

- Community
 - <https://github.com/Microsoft/TypeScript>
 - Over 3500 StackOverflow questions
- Frameworks
 - Over 800 .d.ts library definitions on Definitely Typed repository covering practically all popular JavaScript frameworks
- Tool support
 - IDEs: Visual Studio, Sublime Text, Atom, Eclipse, WebStorm, ...
 - Build and test: ASP.NET, grunt, gulp, node.js, tsUnit

TypeScript 로드맵

큰 규모의
JavaScript 앱
개발을 위한
오픈소스 언어

- TypeScript 1.4
 - Union types, type guards, const enums, let and const, template strings
- TypeScript 1.5
 - Most of ES6, tsconfig.json, TSServer, Sublime Text plug-in
 - Ratification targeted for December 2014
 - Classes, Arrows, Modules, Destructuring, let + const, for...of, Generators, Comprehensions, ...
- TypeScript 1.6
 - Async/await, generators, module bundling
- TypeScript 2.0
 - Asynchronous Functions (async/await)
 - Class expressions, local types
 - Investigate mixins, abstract classes, etc.

TypeScript roadmap

TypeScript 2.0

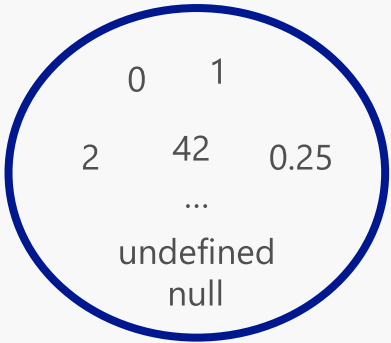
- Non-nullable types
- Control flow based type analysis
- Async/await downlevel support
- Readonly properties
- Declared 'this' type in functions
- Improved typings acquisition

TypeScript 2.1 and beyond

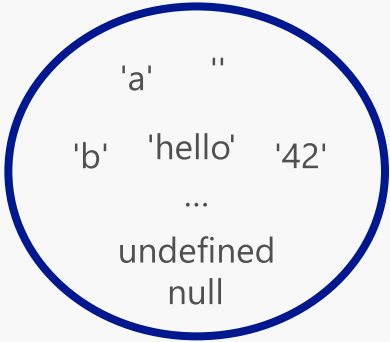
- New JS language service in Visual Studio
- More refactoring support
- Improved support for 'this' in functions
- And much more...

Nullable types

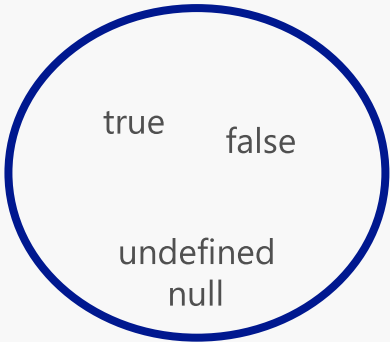
number



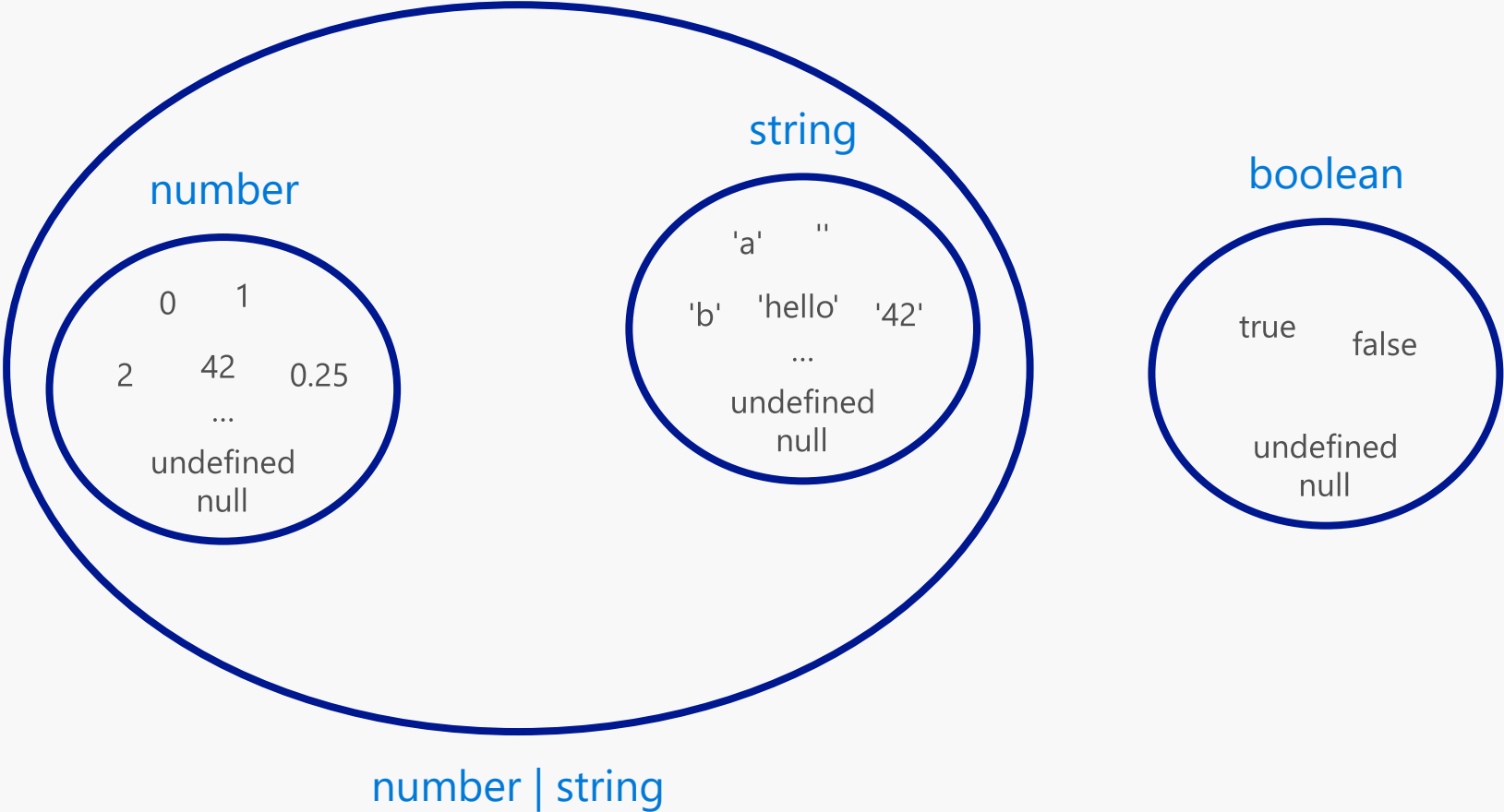
string



boolean

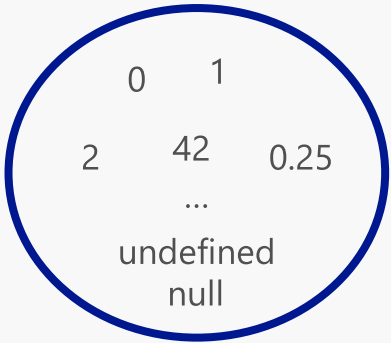


Nullable types

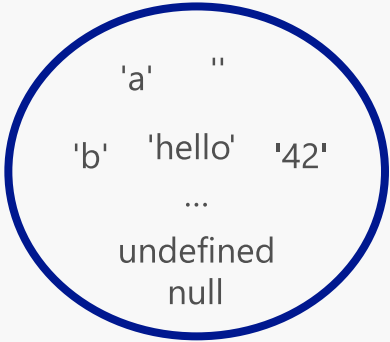


Nullable types

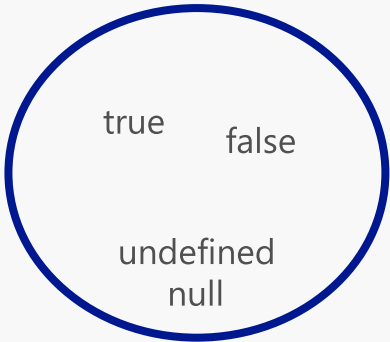
number



string

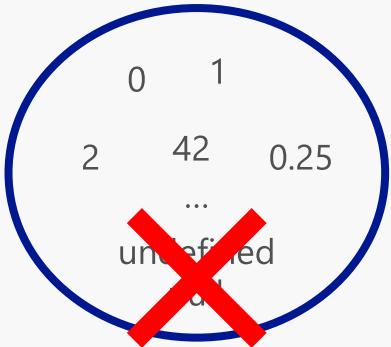


boolean

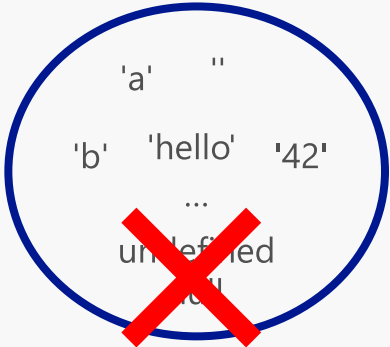


Non-nullable types

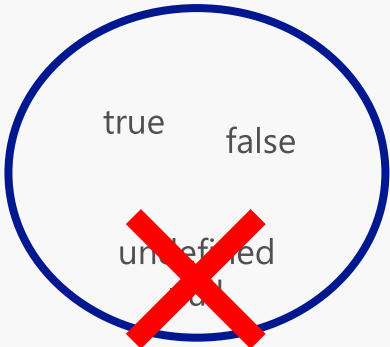
number



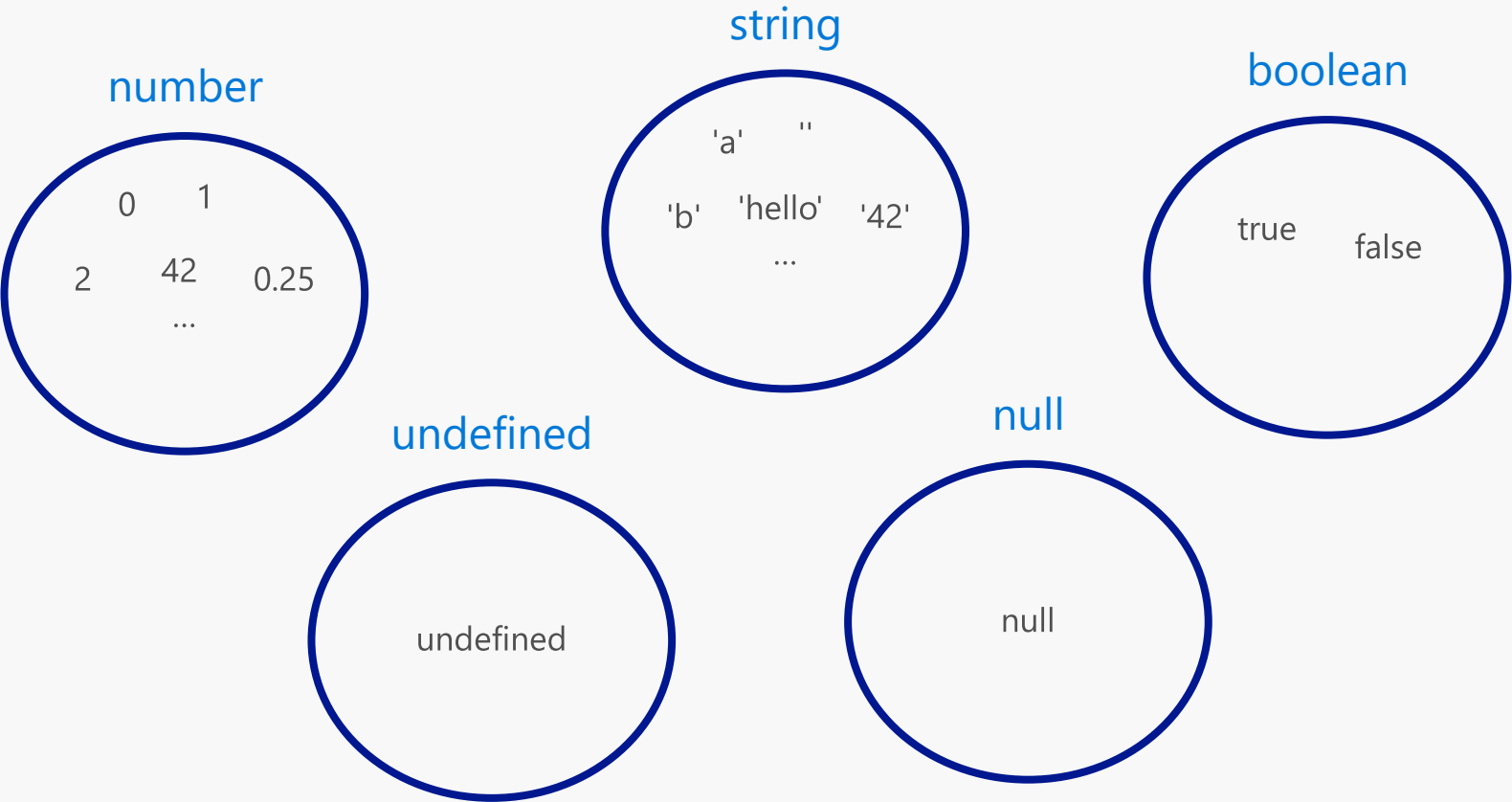
string



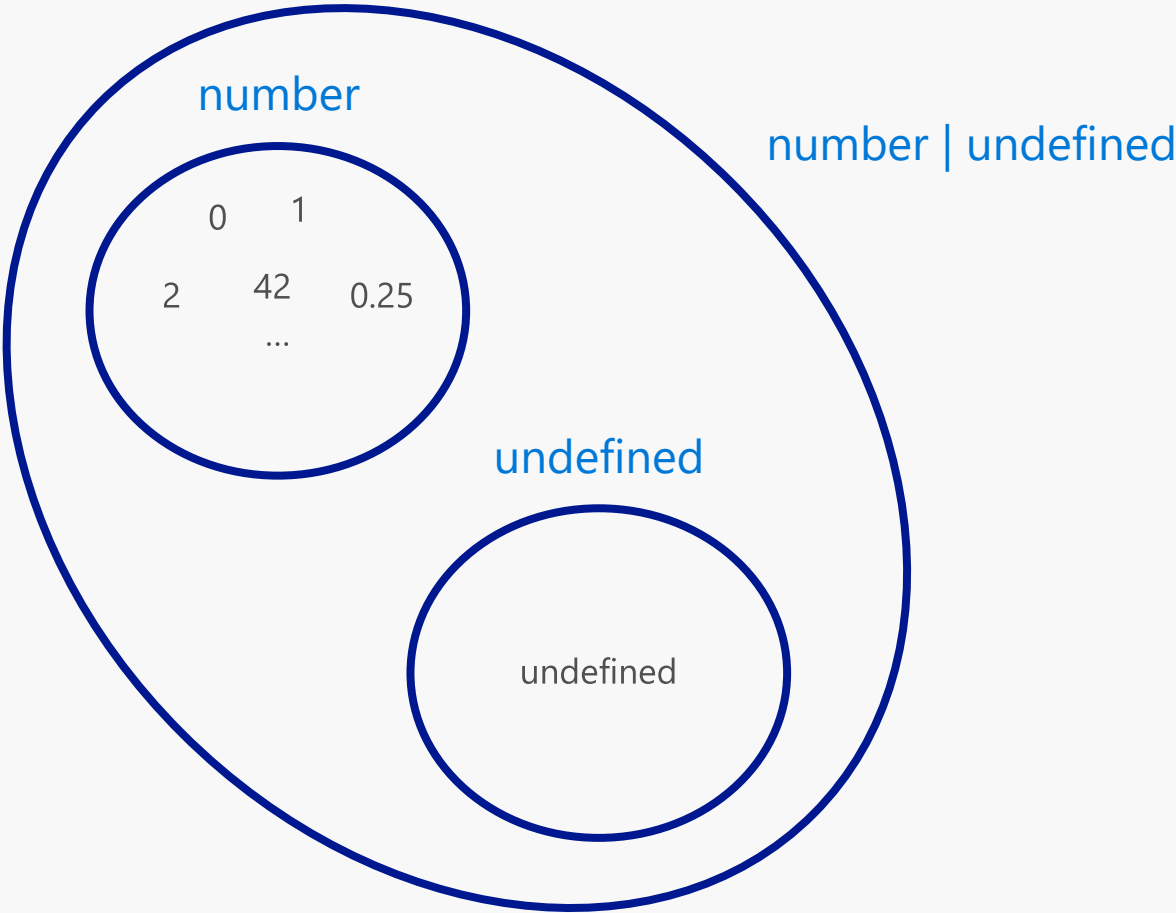
boolean



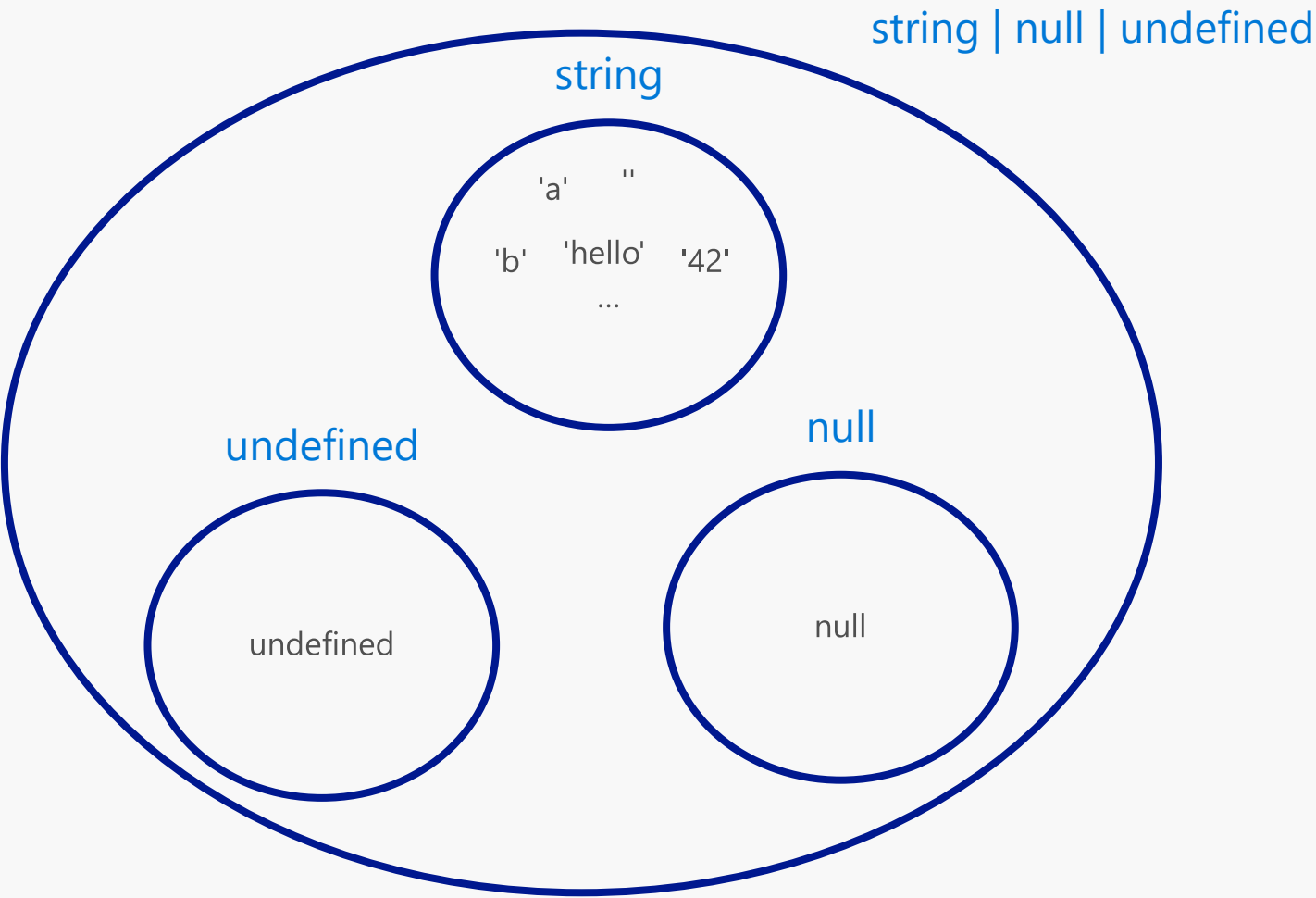
Non-nullable types



Non-nullable types



Non-nullable types



널 가능 형식(Nullable 타입)

```
interface IDog {  
    name : string;           // 필수  
    age? : number;          // 옵션  
}
```



SubscriberCounter 추상 클래스

- 구독자 수 관리 주요 기능 제공 부모(추상) 클래스
- API
 - count 속성
 - increment()
 - decrement()
 - update()
- 추상 멤버
 - counterType
 - getCounterInfo()

CounterList 클래스

- 여러 항목의 카운터를 관리
 - 유튜브 카운터
 - 블로그 카운터
 - 깃허브 카운터
 - 등등
- API
 - add()
 - getAll()

출처 및 참고자료



- TypeScript
 - <http://www.typescriptlang.org>
- TypeScript Source Code
 - <https://github.com/Microsoft/TypeScript>
- Definitely Typed
 - <https://github.com/borisyankov/DefinitelyTyped>
- Channel9
 - <https://channel9.msdn.com/>
 - Anders Hejlsberg: Introducing TypeScript
 - <https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript>
 - Build 2015: 강력 추천
 - Microsoft Ignite
- Stack Overflow
 - <http://stackoverflow.com/questions/tagged/typescript/>



참고 동영상

- Anders Hejlsberg on TypeScript 2
 - <https://channel9.msdn.com/Blogs/Seth-Juarez/Anders-Hejlsberg-on-TypeScript-2>



결론

- 큰 규모의 JavaScript 개발은 어렵다.
- 하지만,
- TypeScript를 사용하면 쉽게 개발할 수 있다.



<http://typescriptlang.org>



<http://typescriptlang.org>



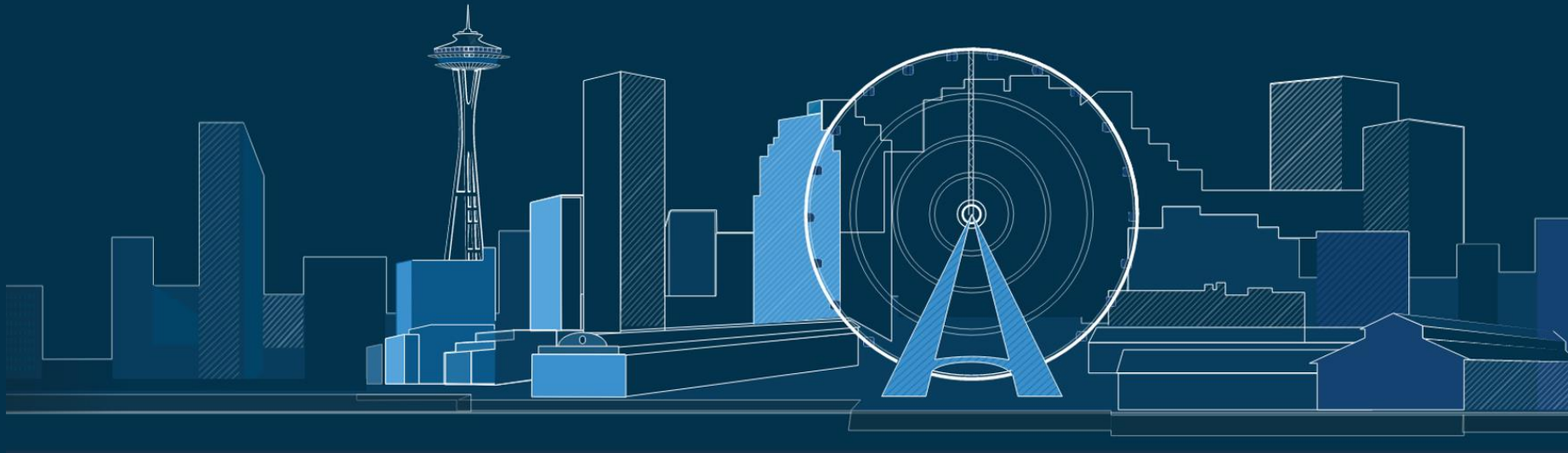
Starts and ends with JavaScript



Strong tools for large apps



State of the art JavaScript



마무리

- 기본 JS 기반 앱을 TS로 마이그레이션
- *.js가 아닌 *.ts로부터 시작



감사합니다

데브렉(<http://www.devlec.com/>) : 쉽게 배우는 TypeScript

감사합니다.
박용준

