TypeScript

Quick Start

- typescript 설치

```
$ npm i -g typescript  # install globally, use $ tsc <typescript file>
# or
$ npm i -D typescript  # install devDependency, use $ npx tsc
<typescript file>

$ npm ls -g --depth=0  # 확인
$ tsc -v  # global
$ npx tsc -v  # local
```

- prettier

```
$ npm i tslint-config-prettier
```

- set tslint.json

```
{
  "defaultSeverity": "error",
  "extends": ["tslint:latest", "tslint-config-prettier"],
  "jsRules": {},
  "rules": {
    "semicolon": true
  },
  "rulesDirectory": []
}
```

- generate tsconfig.json

```
$ tsc --init
```

- edit package.json for prettier

```
  "prettier": {
    "printWidth": 80,
    "useTabs": false,
    "tabWidth": 2,
    "bracketSpacing": true,
    "semi": true,
    "singleQuote": false
  }
}
```

- install ts-node

```
$ npm install -g ts-node  # global
$ npm install --save-dev ts-node  # local
```

- 프로젝트 시작하기

```
$ npm init -y  # create package.json in node.js project
$ npm install typescript --save-dev  # add typescript
$ npm install @types/node --save-dev  # add node.d.ts
$ npx tsc --init --rootDir src --outDir lib --esModuleInterop --
resolveJsonModule --lib es6,dom --module commonjs  # initialize
typescript with tsconfig.json
$ npm install ts-node --save-dev  # to support realtime compile using
ts-node
$ npm install nodemon --save-dev  # monitoring file change using
nodemon
```

- package.json

```
"scripts": {
  "start": "npm run build:live",
  "build": "tsc -p .",
  "build:live": "nodemon --watch 'src/**/*.ts' --exec \"ts-node\"
src/index.ts"
},
```

- 실행

```
$ npm start
```

- Node.js + TypeScript 프로젝트 만들기
- [TypeScript] nodemon, ts-node 모듈 설치하기
- Node.js 시작하기

Setup

```
npm i -D typescript  // devDependency로 설치
npm i -g typescript  // global로 설치
```

- PowerShell 관리자모드 실행

```
Set-ExecutionPolicy Unrestricted
```

```
npx tsc myFirstTypescript.ts  // devDependency로 설치 시
tsc myFirstTypescript.ts  // global 설치 시
```

```
tsc --init
```

```
  "outDir": "./dist",
  "rootDir": "./src",
```

TSLint 설치

```
npm i -g tslint  // tslint package module 설치
```

```
tslint --init
```

```
{
  ...
  "rules": {
    "semicolon": true
  },
  ...
}
```

```
  "editor.formatOnSave": false,
  "tslint.autoFixOnSave": true,
```

Prettier 설치

```
npm i tslint-config-prettier  // 패키지 설치 후 아래 진행
```

```
{
  ...
  "extends": ["tslint:recommended", "tslint-config-prettier"],
  ...
}
```

```
  editor.formatOnSave: true
  ...
  "prettier.tslintIntegration": true
```

Fundamentals

- 트랜스파일러(transpiler)
- 비구조화 할당(destructuring assignment
- 화살표 함수(arrow function)
- 캡슐화(encapsulation), 상속(inheritance), 다형성(polymorphism)
- 반복기(iterator)
- 생성기(generator)
- Promise, await/async
- 비동기 콜백 함수 asynchronous callback function
- 타입 주석(type annotation)
- 타입 추론(type inference)
- 튜플(tuple)
- 제네릭 타입(generic type)
- 추상 데이터 타입(abstract data type)
- 합집합 타일(union 또는 sum type), 교집합 타입(intersection 또는 product type)
- package.json
- tsconfig.json
    - module 키
    - moduleResolution 키
    - target 키
    - baseUrl과 outDir 키
    - paths 키
    - esModuleInterop 키
    - sourceMap 키
    - downlevelIteration 키
    - noImplicitAny 키
- number, boolean, string, object, any, undefined
- let, const
- 템플릿 문자열; `` 역따옴표(backtick) ${}
- object, interface, class, abstract class
- 선택속성(optional property) ?
- 익명 인터페이스(anonymous interface)
- 클래스; 접근제한자(access modifier), 생성자(constructor)
- 구현 implements
- 상속 extends, super 키워드
- static 속성 in class
- 구조화 structuring ↔ 비구조화 destructuring
- 잔여 연산자 rest operator, 전개 연산자 spread operator
- 타입 변환 type conversion, cf.) type casting, type coercion
- 타입 단언 type assertion
- 매개변수 parameter, 인수 혹은 인자 argument
- 함수 시그니처 function signature
- 타입 별칭 type alias
- undefined, null
- 선택적 매개변수 optional parameter
- 함수 표현식 function expression

- 일등 함수 first-class function
- 표현식, 함수 표현식, 계산법, 함수 호출 연산자, 익명 함수
- 실행문 지향 언어 execution-oriented language, 표현식 지향 언어 expression-oriented language, 다중 패러다임 언어 multi-paradigm language
- 표현식 문 expression statement
- 복합 실행문 compound statement
- 고차 함수 high-order function, 클로저 closure, 부분 애플리케이션 partial application, 부분 적용 함수 partially applied function
- 디폴트 매개변수 default parameter
- 클래스 메서드, this, 메서드 체인 method chain

- 배열, 튜플, 인덱스 연산자; []
- for...in 인덱스값, for...of 아이템값
- 제네릭 타입 generics
- 선언형 프로그래밍 declarative programming, 명령형 프로그래밍 imperative programming,
- 배열의 map, reduce, filter 메서드
- 순수 함수 pure function, 불순 함수 impure function
- 타입 수정자 readonly
- 불변 immutable, 가변 mutable
- 깊은 복사 deep-copy, 얕은 복사 shallow-copy
- 가변 인수 variadic arguments
- 반복기 iterator, Iterable<T>와 Iterator<T> 인터페이스
- 생성기 generator, yield, function*, 세미코루틴 semi-coroutine 반협동 루틴, 멀티스레드 multi-thread, 싱글스레드 single-thread
- 세미코루틴, 코루틴

- 동기 synchronous, 비동기 asynchronous
- Promise → resolve와 reject 함수, then-체인, .resolve 메서드, .reject 메서드, .race 메서드, .all 메서드
- async/await;

- 함수형 프로그래밍 = 순수 함수와 선언형 프로그래밍의 토대 위에 함수 조합(function composition)과 모나드 조합(monadic composition)으로 코드를 설계하고 구현하는 기법
  - 람다수학(ramda calculus): 조합 논리와 카테고리 이론의 토대가 되는 논리 수학
  - 조합 논리(combinatory logic): 함수 조합의 이론적 배경
  - 카테고리 이론(category theory): 모나드 조합과 고차 타입의 이론적 배경
  - 정적 타입(static type), 자동 메모리 관리(automatic memory management), 계산법(evaluation), 타입 추론(type inference), 일등 함수(first-class function)에 기반을 두고, 대수 데이터 타입(algebraic data type), 패턴 매칭(pattern matching), 모나드(monad), 고차 타입(high order type) 등의 고급 기능 제공
  - cf.) LISP, meta language ML, Haskell, scala
- 제네릭 함수, 제네릭 타입 generic type
- 일대일 관계 one-to-one relationship, mapping, map
- 고차함수, 커리, 애리티(arity) 매개변수의 개수
- 고차 함수 high-order function, 1차 함수 first-order function, 2차 고차 함수 second-order function, 3차 고차 함수 third-order function
- 커리 curry
- 부분 적용 함수 partially applied function, 부분 함수 partial function
- 클로저 closure, 지속되는 유효 범위 persistence scope
- 바깥쪽 유효 범위 outer scope, 내부 범위 inner scope, 자유 변수 free variable

- 함수 조합 function composition
- compose 함수, pipe 함수
- 포인트가 없는 함수 pointless function

- 람다 라이브러리
  - 타입스크립트 언어와 100% 호환
  - compose와 pipe 함수 제공
  - 자동 커리(auto curry) 기능 제공
  - 포인트가 없는 고차 도움 함수 제공
  - 조합 논리(combinatory logic) 함수 일부 제공
  - 하스켈 렌즈(lens) 라이브러리 기능 일부 제공
  - 자바스크립트 표준 모나드 규격(fantasyland-spec)과 호환
  - https://ramdajs.com/docs : 함수를 알파벳 순서로 분류
  - https://devdocs.io/ramda/ : 함수를 기능 위주로 분류
- 기본 사용, 배열에 담긴 수 다루기, 서술자와 조건 연산, 문자열 다루기, 렌즈를 활용한 객체의 속성 다루기, 객체 다루기, 배열 다루기, 조합 논리

ramda 패키지가 제공하는 함수 구분	
구분	**내용**
함수(function)	R.compose, R.pipe, R.curry 등 52개 함수
리스트(list)	배열을 대상으로 하는 R.map, R.filter, R.reduce 등
로직(logic)	R.not, R.or, R.cond 등 불리언 로직 관련
수학(math)	R.add, R.subtract, R.multiply, R.divide 등 수 관련
객체(object)	R.prop, R.lens 등 객체와 렌즈 관련
관계(relation)	R.lt, R.lte, R.gt, R.gte 등 두 값의 관계를 파악
문자열(string)	R.match, R.replace, R.split 등 문자열을 대상으로 정규식(regular expression) 등
타입(type)	R.is, R.isNil, R.type 등 대상의 타입을 파악

- 자동 커리 auto curry
- 가변 인수 variadic arguments
- 순수 함수 PURE FUNCTION
- 선언형 프로그래밍 declarative programming
- 2차 방정식 quadratic equation
- 서술자 predicate
- 낙타등 표기법 camel case convention
- 렌즈
  1. R.lens 함수로 객체의 특정 속성에 대한 렌즈를 만든다.
  2. 렌즈를 R.view 함수에 적용해 속성값을 얻는다.
  3. 렌즈를 R.set 함수에 적용해 속성값이 바뀐 새로운 객체를 얻는다.
  4. 렌즈와 속성값을 바꾸는 함수를 R.over 함수에 적용해 값이 바뀐 새로운 객체를 얻는다.

*조합자 combinator

람다가 제공하는 조합자		
조합자 이름	**의미**	**람다 함수 이름**
I	identity	R.identity
K	constant	R.always
T	thrush	R.applyTo
W	duplication	R.unnest

람다가 제공하는 조합자		
조합자 이름	의미	람다 함수 이름
C	flip	R.flip
S	substitution	R.ap

- 제네릭 프로그래밍
- 제네릭 타입의 이해
- 제네릭 사용
- 제네릭 타입 제약(generic type constraint)
- new 타입 제약
    - 팩토리 함수(factory function)
- 인덱스 타입 제약(index type constraint)
- 대수 데이터 타입(algebraic data type)
    - ADT 추상 데이터 타입(abstract data type)
    - 합집합 타입(union type) or |, 교집합 타입(intersection type) and &
- 식별 합집합(discriminated unions)
- 타입 가드(type guard)
    - instanceof 연산자
    - is 연산자
- F-바운드 다형성(F-bound polymorphism)
- nullable 타입
- 옵션 체이닝(option chaining)
- 널 병합 연산자(nullish coalescing operator)
- 세이프 내비게이션 연산자(safe navigation operator)
- 펑터(functor)
- NaN(Not a Number)

- 모나드(Monad)
- 카테고리 이론(category theory)
- 코드 설계 패턴(design pattern)
- 타입 클래스(type class)
- 고차 타입(higher-kinded type)
- 판타지랜드 규격(fantasy-land)
- 모나드 조건
    - 펑터(Functor): map이라는 인스턴스 메서드를 가지는 클래스
    - 어플라이(Apply): 펑터이면서 ap라는 인스턴스 메서드를 가지는 클래스
    - 애플리커티브(Applicative): 어플라이이면서 of라는 클래스 메서드를 가지는 클래스
    - 체인(Chain): 애플리커티브이면서 chain이라는 메서드를 가지는 클래스
- 모나드 룰, 왼쪽 법칙, 오른쪽 법칙
- 엔도(endo)-, 엔도펑터(endofunctor)
- Maybe 모나드; Just 모나드, Nothing 모나드
- Validation 모나드; Success 모나드, Failure 모나드, → 비밀번호 검증, 이메일 주소 검증,
- IO 모나드;

Examples

```
  this.listJoinPath[Object.keys(this.listJoinPath).find(key =>
this.listJoinPath[key].dtlCd === thisjoinPathCd)].joinPathCont
```

- [JS] value값으로 key값 찾기!

References

- TypeScript Documentation
- TypeScript-Handbook 한글 문서 The TypeScript Handbook
- 타입스크립트 핸드북
- VScode 에서 Typescript 환경 설정하기