

# ReScript 같이 해요

# 발표자 소개



 [MiryangJung](#)

 [정미량 \(Miryang Jung\)](#)

 [@MiryangJung](#)

그린랩스 프론트엔드 엔지니어

여행을 좋아합니다.

지방 거주 재택러.

함수형 프로그래밍 입문자.

개발 블로그. [miryang.dev](#) (게스트북 남겨주세요 🐼)

# 목차

가볍게 알아보기

작게 좋았던 점

크게 좋았던 점

아직도 망설인다면

아쉬웠던 점

# 예상 청자

## 내가 만든 발표, 같이하기 위해 구웠지

프론트엔드 비기너이신 분

새로운 언어를 가볍게 알아보고 싶으신 분

함수형 프로그래밍 언어에 관심 있으신 분

강력한 타입 언어를 사용해 보고 싶으신 분

ReScript 도입을 망설이고 있으셨던 분



**가볍게 알아보기**

# ReScript란?




읽을 수 있는 JavaScript로 컴파일되는 강력한 타입 언어





JavaScript 개발자에게 친숙한 구문을 제공

모든 JavaScript 라이브러리를 ReScript와 함께 사용 가능

# Hello World!

 **rescript**

[Docs](#) [API](#) [Playground](#) [Blog](#) [Community](#)

Format

Copy Share Link

JavaScript

Problems

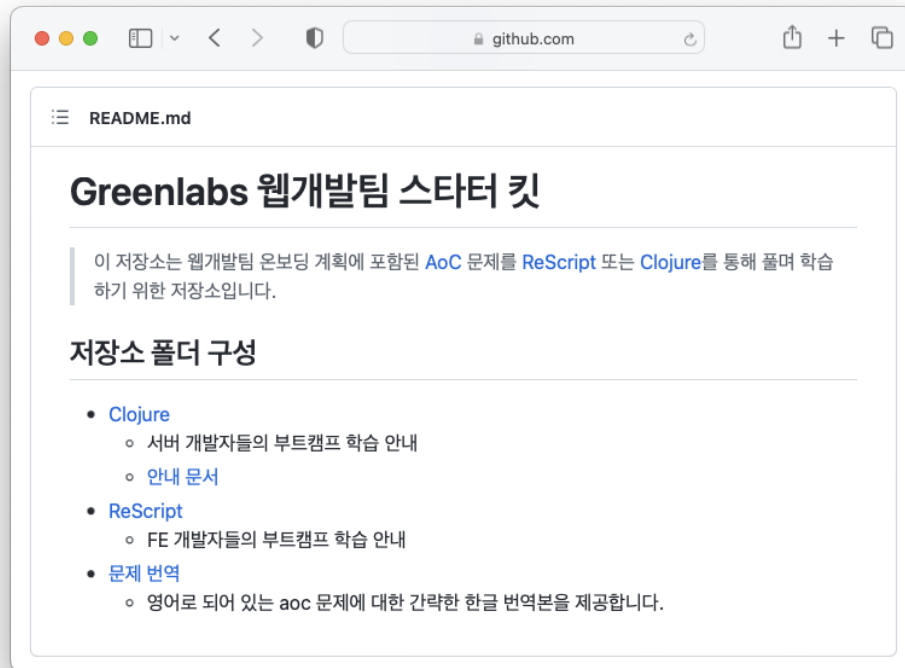
Settings

1

```
// Generated by ReScript, PLEASE  
EDIT WITH CARE  
/* This output is empty. Its  
source's type definitions,  
externals and/or unused code got  
optimized away. */
```

# ReScript 첫인상

프로그래밍 언어 문법은 금방 익힌다는 자신감 Down





# 달랐던 것들

달랐던 점	설명
😓 let만 있다.	const와 비슷한 불변 변수 선언
😂 화살표를 사용한다.	-> 파이프 연산자, a(b)를 b->a
😓 return이 없다.	마지막 라인은 암묵적 반환
😓 import & export가 없다.	모든 모듈은 내보내진다
🤖 타입 어노테이션 없이도 타입이 있다.	타입 추론 시스템

**작게 좋았던 점**

# 빌트인 포맷터

작은 따옴표 vs 큰 따옴표 / 탭 사이즈 2 vs 4 / 세미콜론 붙이기 vs 말기

논쟁할 필요 없다

# 좌에서 우로, 위에서 아래로 읽기

```
1  validateAge(getAge(parseData(person)))
2
3  person -> parseData -> getAge -> validateAge
4
5  person
6    -> parseData
7      -> getAge
8        -> validateAge
```

파이프 연산자 `->` 책 읽듯이 한 방향으로 읽기

# export & import 신경 안쓰기

```
1  export { default as Button } from "./Button";
2
3  export { default as Card } from "./Card";
4
5  export { default as Modal } from "./Modal";
6
7  export { default as Toast } from "./Toast";
8
9  export { default as Layout } from "./Layout";
10
11 export { default as Footer } from "./Footer";
12
13 export { default as Header } from "./Header";
14
15 export { default as Container } from "./Container";
16
17 export { default as Toc } from "./Toc";
18
19 export { default as Title } from "./Title";
```

# export & import 신경 안쓰기

## 모든 .res 파일은 모듈

모든 모듈은 내보내진다.

인터페이스 파일을 사용하면 내보내고 싶은 모듈만 지정할 수도 있다.

```
1 // TestComponent.res
2 module Button = {
3   @react.component
4   let make = () =>
5     <button> {`click`->React.string} </button>
6 }
```

```
1 // Generated by ReScript, TestComponent.js
2 function Playground$Button(Props) {
3   return React.createElement("button", undefined, "click");
4 }
5 var Button = { make: Playground$Button };
6
7 exports.Button = Button;
```

# export & import 신경 안쓰기

```
1  import Container from '../components/Container';  
2  
3  import Button from '../components/Button';  
4  
5  import Button form '../어디더라?';
```

# export & import 신경 안쓰기

자동으로 프로젝트 안에서 모듈을 찾는다.

import할 파일의 위치를 몰라도, import 한 파일의 위치가 바뀌어도

코드를 변경하지 않아도 된다.

```
1 // TestComponent.res
2
3 module Button = {
4   @react.component
5   let make = () =>
6     <button> {'click'->React.string} </button>
7 }
```

```
1 // TestPage.res
2
3 <TestComponent.Button />
```



**크게 좋았던 점**

# 타입 추론

타입 어노테이션 없이 모든 표현식의 타입을 힌들리-밀너 타입 추론으로 확인


```
1 let add1 = (a, b) => a + b
2
3 let add2 = (a, b) => a ++ b
4
5 let add3 = (a, b) => a +. b
```

```
1 (int, int) => int
2
3 (string, string) => string
4
5 (float, float) => float
```


# 타입 추론

값의 형태가 맞는 레코드 타입 선언을 찾는다. data는 person 타입으로 추론된다.

타입 검사를 통과한다면 런타임에 잘못 처리되는 값이 없음이 보장된다.

 rescript

[Docs](#) [API](#) [Playground](#) [Blog](#) [Community](#)



Format

Copy Share Link

JavaScript

Problems

Settings

```
1 type person = {
2   age: int,
3   name: string,
4 }
5
6 ! let data = {
7   age: 27,
8 }
```

Type Errors

[E] Line 6, column 11:  
Some record fields are undefined: name

Type Errors

# Variant

## 합타입

Red 또는 Blue 또는 Yellow 표현한다.

```
1 type color = Red | Blue | Yellow
2
3 let myColor = Red
```

배리언트 생성자는 추가 값을 가질 수 있다.

```
1 type result = Pending | Success | Fail
2
3 type result = Pending | Success({data: string}) | Fail
```

# 패턴 매칭

## 데이터 형태에 따른 switch 구문

구조 분해를 하고, 각각 분해된 결과의 오른 편에 작성된 코드가 실행된다.

```
1  type sns = Facebook(string) | Twitter(string) | None
2
3  let name = switch data {
4    | Facebook(name) => name
5    | Twitter(name)  => name
6    | None           => ""
7  }
```

# 패턴 매칭

누락 된 패턴이 있는지 컴파일 시점에 검사한다.

 **rescript**

[Docs](#) [API](#) [Playground](#) [Blog](#) [Community](#)

Format

Copy Share Link

JavaScript

Problems

Settings

```
1 let data = (true, false)
2
3 let component = switch data {
4 | (true, true) => "tt"
5 | (true, false) => "tf"
6 | (false, true) => "ft"
7 }
8
```

Compiled with 1 Warning(s)

[W] Line 3, column 16:  
You forgot to handle a possible case here, for example:  
(false, false)

# 패턴 매칭 & 배리언트

```
1  <button onClick={() => router.push('/')}>
2    Home으로
3  </button>
4
5  <button onClick={() => router.push('/post')}>
6    Post로
7  </button>
8
9  <button onClick={() => router.push('/post?id=123')}>
10   id와 함께 Post로
11 </button>
```

# 패턴 매칭 & 배리언트

```
1  type page = Home | Post
2
3  let toString = page => {
4    switch page {
5      | Home => "/"
6      | Post => "/post"
7    }
8  }
9
10 <button onClick={_=> router->Next.Router.push(Route.toString(Home))}>
11   {'Home으로' -> React.string}
12 </button>
13
14 <button onClick={_=> router->Next.Router.push(Route.toString(Post))}>
15   {'Post로' -> React.string}
16 </button>
```



# 패턴 매칭 & 배리언트

```
1  type page = Home | Post({id: string})
2
3  let toString = page => {
4    switch page {
5      | Home => "/"
6      | Post({id}) => {...}
7    }
8  }
9  ...
10 <button onClick={_=>
11   router->Next.Router.push(Route.toString(Post{id: 123}))}>
12   {'id와 함께 Post로' -> React.string}
13 </button>
14
15 // 컴파일 에러 발생
16 <button onClick={_=> router->Next.Router.push(Route.toString(Post))}>
17   {'Post로' -> React.string}
18 </button>
```

# Null

```
1 Uncaught TypeError: Cannot read properties of null (reading 'value')
```

*I call it my billion-dollar mistake.  
It was the invention of the null reference in 1965  
- Tony Hoare -*

# option

타입으로 존재하지 않는 값을 표현한다.

ReScript에 null 또는 undefined에 대한 개념이 없습니다.

option 타입은 배리언트입니다.

```
1  type option<'a> = None | Some('a)
2
3  let 계세요? = 없음 | 사람(정미량)
```

# 예시

## URL 생성하는 API 요청 후 받은 응답을 보여주기

[URL 생성] 요청

요청 중...

[URL 생성] 완료

miryang.dev

[URL 생성] 실패

요청 실패

# 예시

```
1  type result_t = Pending | Success(string) | Fail
2
3  @react.component
4  let default = () => {
5      let (result, setResult) = React.useState(_ => Pending)
6
7      let handleClick = e => {
8          e->ReactEvent.Synthetic.preventDefault
9          let response {
10              | Some(res) => setResult(_ => Success(res))
11              | None => setResult(_ => Fail)
12          }
13      }
14
15      <>
16      <button onClick={handleClick}> {'URL 생성' -> React.string} </button>
17      {
18          switch result {
19              | Pending => {'요청 중...' -> React.string}
20              | Fail => {'요청 실패' -> React.string}
21              | Success(url) => {url -> React.string}
22          }
23      }
```

# 예시

```
1  type result_t = Pending | Success(string) | Fail
2
3  @react.component
4  let default = () => {
5      let (result, setResult) = React.useState(_ => Pending)
6
7      let handleClick = e => {
8          e->ReactEvent.Synthetic.preventDefault
9          let response {
10              | Some(res) => setResult(_ => Success(res))
11              | None => setResult(_ => Fail)
12          }
13      }
14
15      <>
16      <button onClick={handleClick}> {`URL 생성` -> React.string} </button>
17      {
18          switch result {
19              | Pending => {`요청 중...` -> React.string}
20              | Fail => {`요청 실패` -> React.string}
21              | Success(url) => {url -> React.string}
22          }
23      }
```

```
1  type result_t = Pending | Success(string) | Fail
2
3  @react.component
4  let default = () => {
5      let (result, setResult) = React.useState(_ => Pending)
6
7      let handleClick = e => {
8          e->ReactEvent.Synthetic.preventDefault
9          let response {
10              | Some(res) => setResult(_ => Success(res))
11              | None => setResult(_ => Fail)
12          }
13      }
14
15      <>
16      <button onClick={handleClick}> {`URL 생성` -> React.string} </button>
17      {
18          switch result {
19              | Pending => {`요청 중...` -> React.string}
20              | Fail => {`요청 실패` -> React.string}
21              | Success(url) => {url -> React.string}
22          }
23      }
24
25  </>
26  }
```

아직도 망설인다면



# 점진적 채택

*ReScript를 도입하고 싶은데  
프로젝트를 폭파하 X 새로 시작하기에  
리스크가 너무 큼니다.*

**ReScript를 부분적으로 적용하면서  
원활한 통합이 가능**

# 점진적 채택

raw JavaScript 코드를 작성할 수 있다.

```
1  let add = %raw(`  
2    function(a, b) {  
3      console.log("hello from raw JavaScript!");  
4      return a + b  
5    }  
6  `)  
7  
8  Js.log(add(1, 2))
```

타입 어노테이션이 없으면 추론이 된다.

타입 어노테이션을 줘서 타입 안전하게 할 수도 있다.

# genType

## ts-belt

```
1 // ts-belt/src/Option/Option.res
2 %comment("Returns `Some(value)` if the provided value is not falsy, otherwise, returns
3 @gentype
4 let fromFalsy = value => value ? Some(value) : None
```



```
1 // ts-belt/src/Option/Option.ts
2 export declare type ExtractValue<T> = Exclude<T, null | undefined>;
3 export declare type Option<A> = A | null | undefined;
4 export declare const Some: <A>(value: NonNullable<A>) => Option<A>;
5 export declare const None: Option<never>;
6
7 export declare function fromFalsy<A>(value: A): Option<ExtractValue<A>>;
```

# 점진적 채택

**TypeScript로 작성된 블로그 코드를 ReScript로 변경하고 있습니다.**

TypeScript & ReScript & JavaScript 3가지 언어를 동시에 사용할 수 있다.

## Languages



아쉬웠던 점

# 바인딩

자바스크립트 함수를 사용하기 위해 바인딩을 해야 한다.

Next.js의 Head 컴포넌트 바인딩 코드

```
1 // https://github.com/MoOx/rescript-next
2 module Head = {
3   @module("next/head") @react.component
4   external make: (~children: React.element) => React.element = "default"
5 }
```

Web api의 addEventListener, removeEventListener 바인딩 코드

```
1 @val @scope("document")
2 external addEventListener: (string, t => unit) => unit = "addEventListener"
3 @val @scope("document")
4 external removeEventListener: (string, t => unit) => unit = "removeEventListener"
```

# rescript-bindings

## rescript-bindings

---

| rescript bindings monorepo

### List

---

- [@greenlabs/rescript-next](#)
- [@greenlabs/rescript-nock](#)
- [@greenlabs/rescript-jest](#)
- [@greenlabs/rescript-react-hook-form](#)
- [@greenlabs/rescript-testing-library](#)
- [@greenlabs/rescript-daum-postcode](#)
- [@greenlabs/rescript-react-linkify](#)

# 커뮤니티

stackoverflow 또는 검색으로는 도움 받기 힘들다

**ReScript Forum** 을 이용합니다.

stackoverflow Products [rescript]

Home

PUBLIC

**Questions**

Tags

Users

Companies

## Questions tagged [rescript]

Ask Question

The `rescript` tag has no usage guidance, but it has a [tag wiki](#), can you [help us summarize it](#)?

[Unwatch tag](#) [Ignore tag](#) [Learn more...](#) [Top users](#) [Synonyms](#)

**69 questions** [Newest](#) [Active](#) [Bountied](#) [Unanswered](#) [More ▾](#) [Filter](#)



마무리

# Relay

[relay.dev](https://relay.dev)

그래프큐엘 클라이언트 라이브러리

```
1  module Query = %relay(`
2    query IndexQuery {
3      user {
4        name
5        age
6      }
7    }
8  `)
9
10 type props = { data: IndexQuery_graphql.Types.response}
```

# 지금 설치하세요

```
1  npm install rescript --save-dev
```

**감사합니다!**

# 레퍼런스

<https://unsplash.com/photos/z8kriatLFdA>

<https://www.youtube.com/watch?v=WN5UfUjVTNE>

<https://www.youtube.com/watch?v=tn0zt7U7roY>

<https://www.youtube.com/watch?v=dr1PskGSdU4>

<https://www.youtube.com/watch?v=ZY4n7B0pZFW>

<https://www.youtube.com/watch?v=XWgL51JSbGI>

<https://www.youtube.com/watch?v=kAWP0laFz6M>

<https://green-labs.github.io/why-rescript>

<https://rescript-lang.org>