

# TypeScript와 Flow: 자바스크립트 개발에 정적 타이핑 도입하기

안희종 ([heejongahn@gmail.com](mailto:heejongahn@gmail.com))  
하이퍼커넥트

# 안희종이라고 합니다



spoqa

HYPERCONNECT

- 스포카를 거쳐 하이퍼커넥트에 다닙니다.
- 웹사이트를 만듭니다.
- <http://ahnheejong.name/>
- github, twitter @heejongahn

# 오늘의 질문

# 오늘의 질문

- 정적 타이핑을 쓰면 뭐가 좋나요?

# 오늘의 질문

- 정직 타이핑을 쓰면 뭐가 좋나요?
- 좋은 건 알겠고, 어떤 선택지가 있나요?

# 오늘의 질문

- 정적 타이핑을 쓰면 뭐가 좋나요?
- 좋은 건 알겠고, 어떤 선택지가 있나요?
- 왜 [ 스포일러 ] 를 써야 하나요?

# 오늘의 질문

- 정적 타이핑을 쓰면 뭐가 좋나요?
- 좋은 건 알겠고, 어떤 선택지가 있나요?
- 왜 [ 스포일러 ] 를 써야 하나요?
- 어떻게 도입하나요?

# 오늘의 질문

- 정적 타이핑을 쓰면 뭐가 좋나요?
- 좋은 건 알겠고, 어떤 선택지가 있나요?
- 왜 [ 스포일러 ] 를 써야 하나요?
- 어떻게 도입하나요?
- 직접 해보면서 느낀 점 없으신가요?

# 오늘의 질문

- 정적 타이핑을 쓰면 뭐가 좋나요?
- 좋은 건 알겠고, 어떤 선택지가 있나요?
- 왜 [ 스포일러 ] 를 써야 하나요?
- 어떻게 도입하나요?
- 직접 해보면서 느낀 점 없으신가요?
- 그래서 오늘 뭐라고 하셨죠?

# 정직 타이핑을 쓰면 뭐가 좋나요?

좋은 건 알겠고, 어떤 선택지가 있나요?

왜 [ 스포일러 ] 를 써야 하나요?

어떻게 도입하나요?

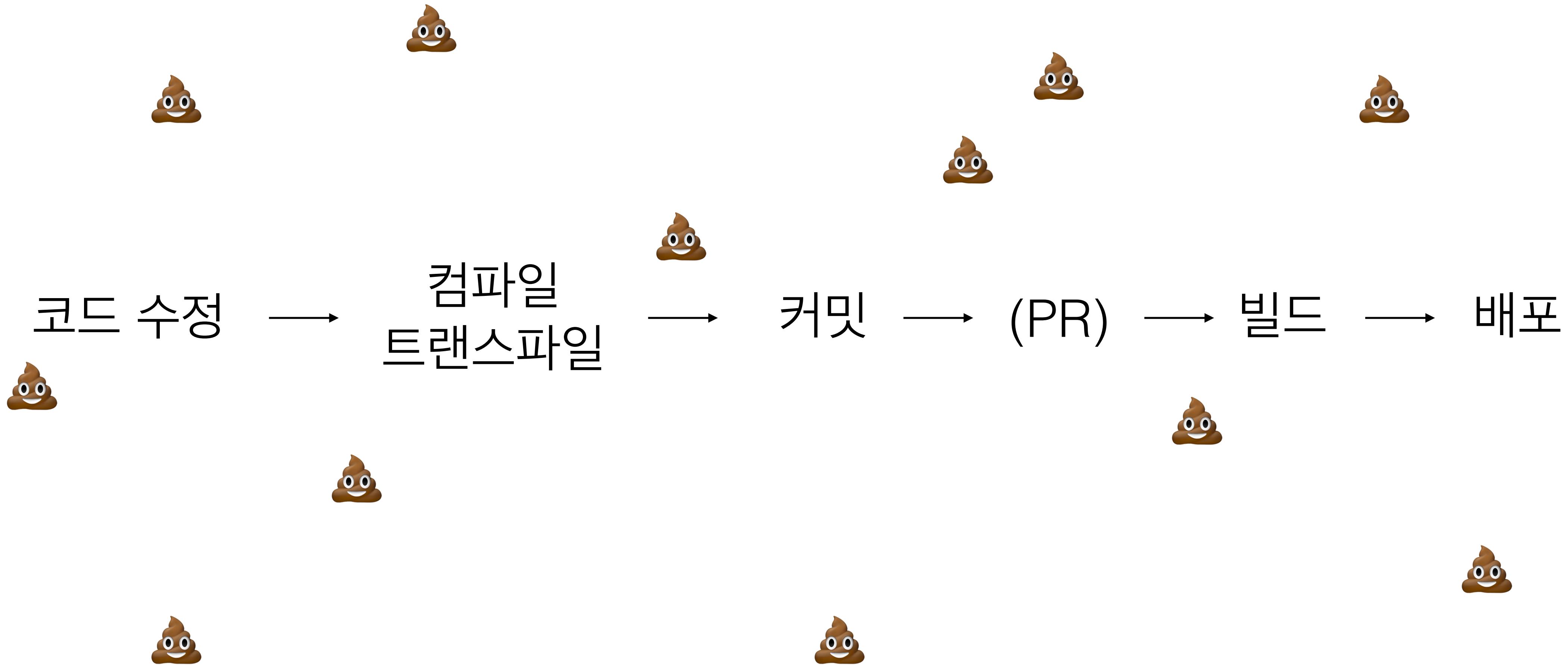
직접 해보면서 느낀 점 없으신가요?

그래서 오늘 뭐라고 하셨죠?

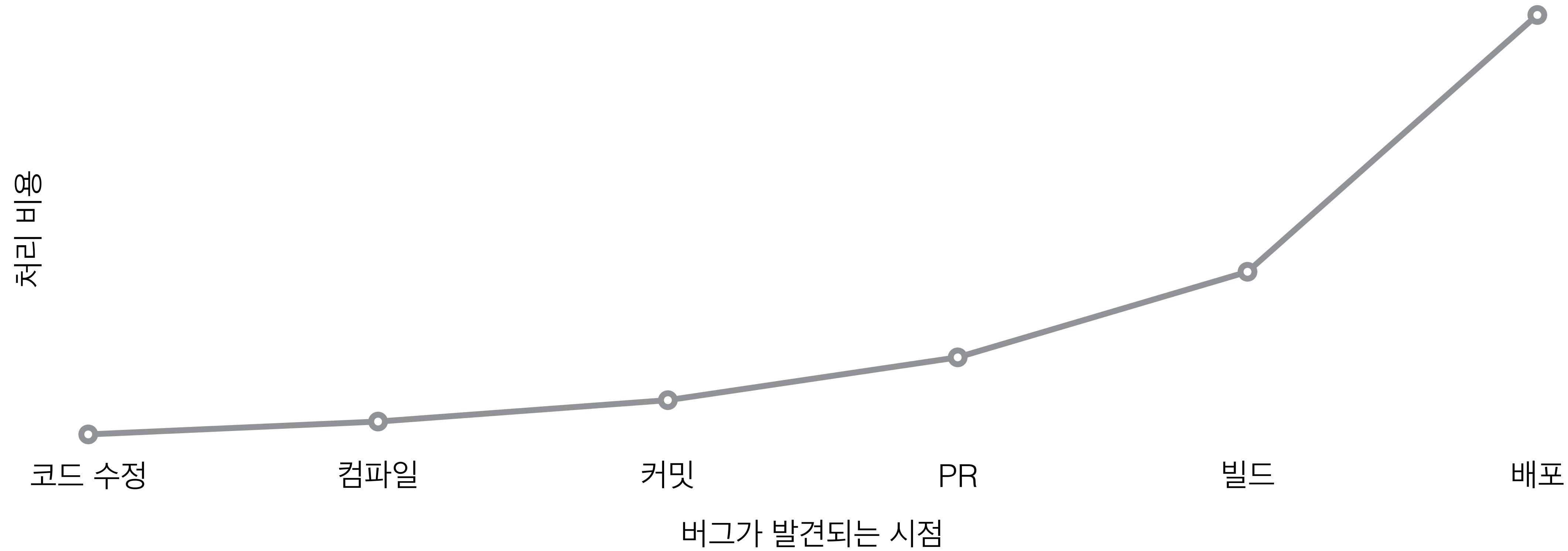
# 흔한 프로그래밍 파이프라인

코드 수정 → 컴파일  
트랜스파일 → 커밋 → (PR) → 빌드 → 배포

# 흔한 프로그래밍 파이프라인

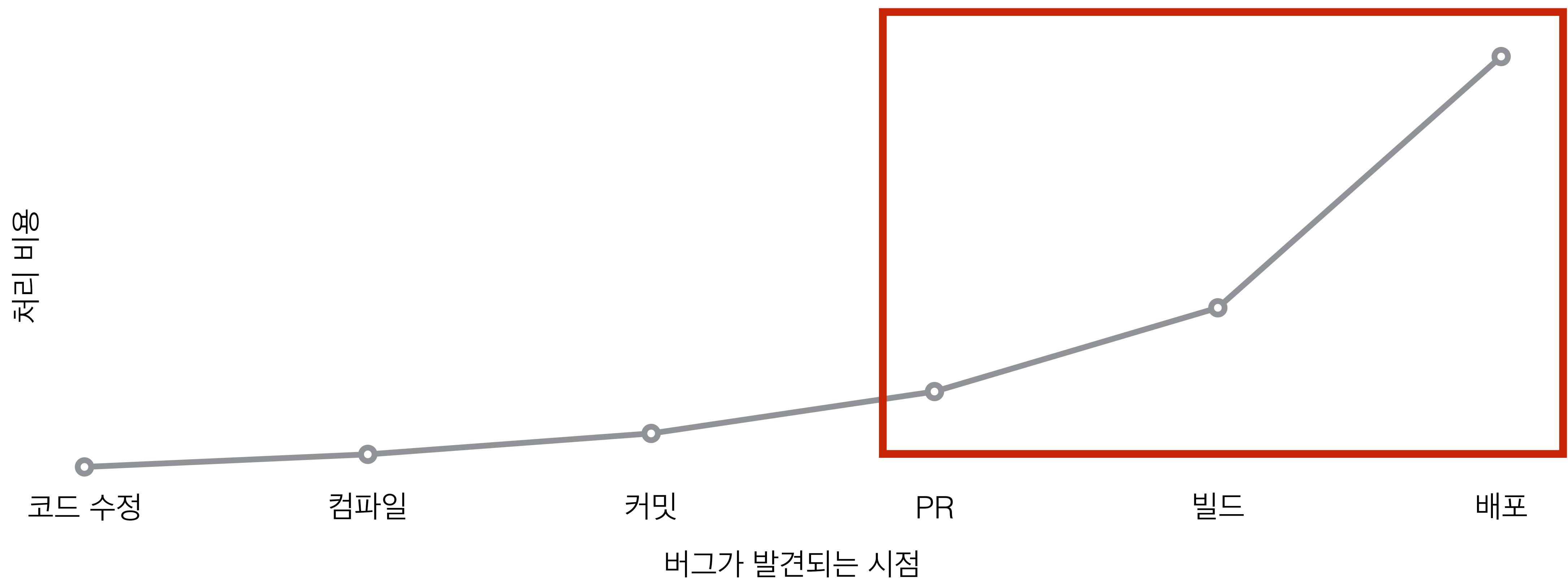


# 버그 발견 시점 - 처리 비용

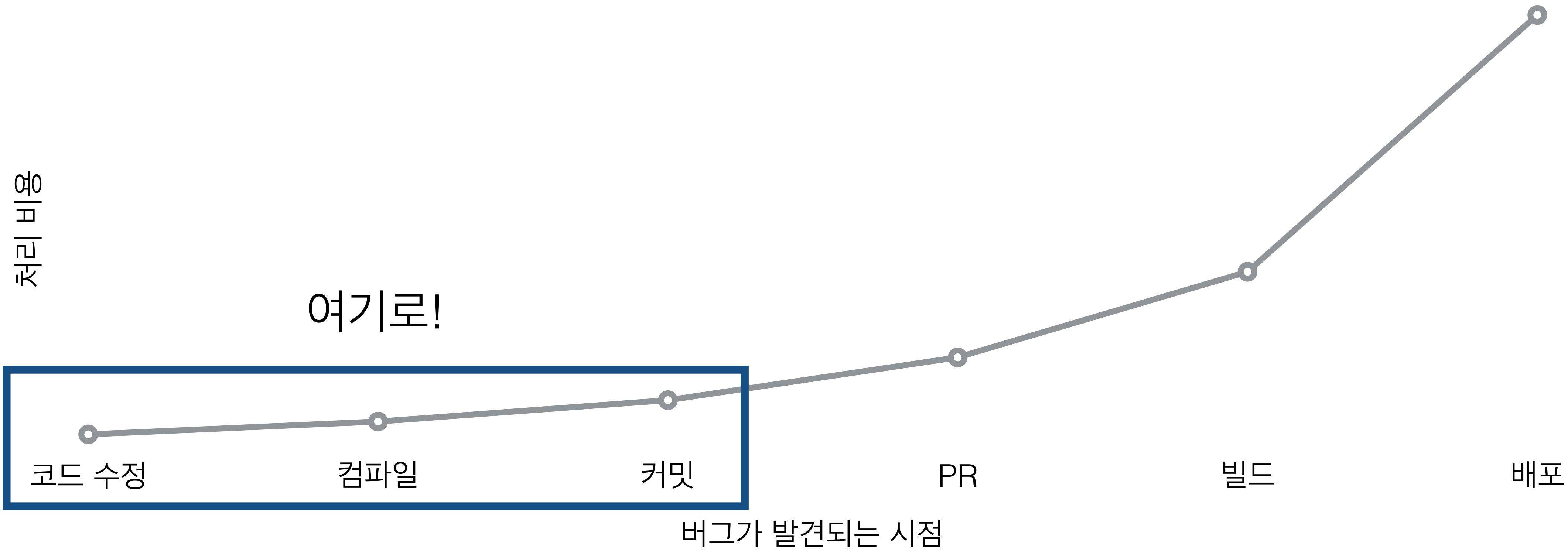


# 버그 발견 시점 - 처리 비용

여기 있는 버그를...



# 버그 발견 시점 - 처리 비용



# 두 정적 타입 시스템 이야기

## To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

Zheng Gao  
University College London  
London, UK  
[z.gao.12@ucl.ac.uk](mailto:z.gao.12@ucl.ac.uk)

Christian Bird  
Microsoft Research  
Redmond, USA  
[cbird@microsoft.com](mailto:cbird@microsoft.com)

Earl T. Barr  
University College London  
London, UK  
[e.barr@ucl.ac.uk](mailto:e.barr@ucl.ac.uk)

<http://earlbarr.com/publications/typestudy.pdf>

# 두 정적 타입 시스템 이야기

- ICSE'17에서 발표.

## To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

Zheng Gao  
University College London  
London, UK  
[z.gao.12@ucl.ac.uk](mailto:z.gao.12@ucl.ac.uk)

Christian Bird  
Microsoft Research  
Redmond, USA  
[cbird@microsoft.com](mailto:cbird@microsoft.com)

Earl T. Barr  
University College London  
London, UK  
[e.barr@ucl.ac.uk](mailto:e.barr@ucl.ac.uk)

<http://earlbarr.com/publications/typestudy.pdf>

# 두 정적 타입 시스템 이야기

## To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

Zheng Gao  
University College London  
London, UK  
z.gao.12@ucl.ac.uk

Christian Bird  
Microsoft Research  
Redmond, USA  
cbird@microsoft.com

Earl T. Barr  
University College London  
London, UK  
e.barr@ucl.ac.uk

- ICSE'17에서 발표.
- 자바스크립트에 정적 타입 시스템 도입시 효과에 대한 연구.

<http://earlbarr.com/publications/typestudy.pdf>

# 두 정적 타입 시스템 이야기

## To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

Zheng Gao  
University College London  
London, UK  
z.gao.12@ucl.ac.uk

Christian Bird  
Microsoft Research  
Redmond, USA  
cbird@microsoft.com

Earl T. Barr  
University College London  
London, UK  
e.barr@ucl.ac.uk

<http://earlbarr.com/publications/typestudy.pdf>

- ICSE'17에서 발표.
- 자바스크립트에 정적 타입 시스템 도입시 효과에 대한 연구.
- “Github에 공개된 open issue 중 타입 시스템으로 잡을 수 있었던 버그의 비율은 얼마나 될까?”

# “탐지 가능한 버그”

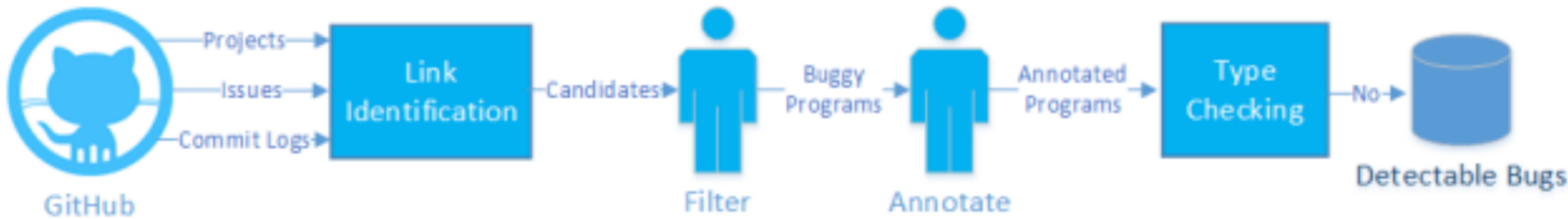
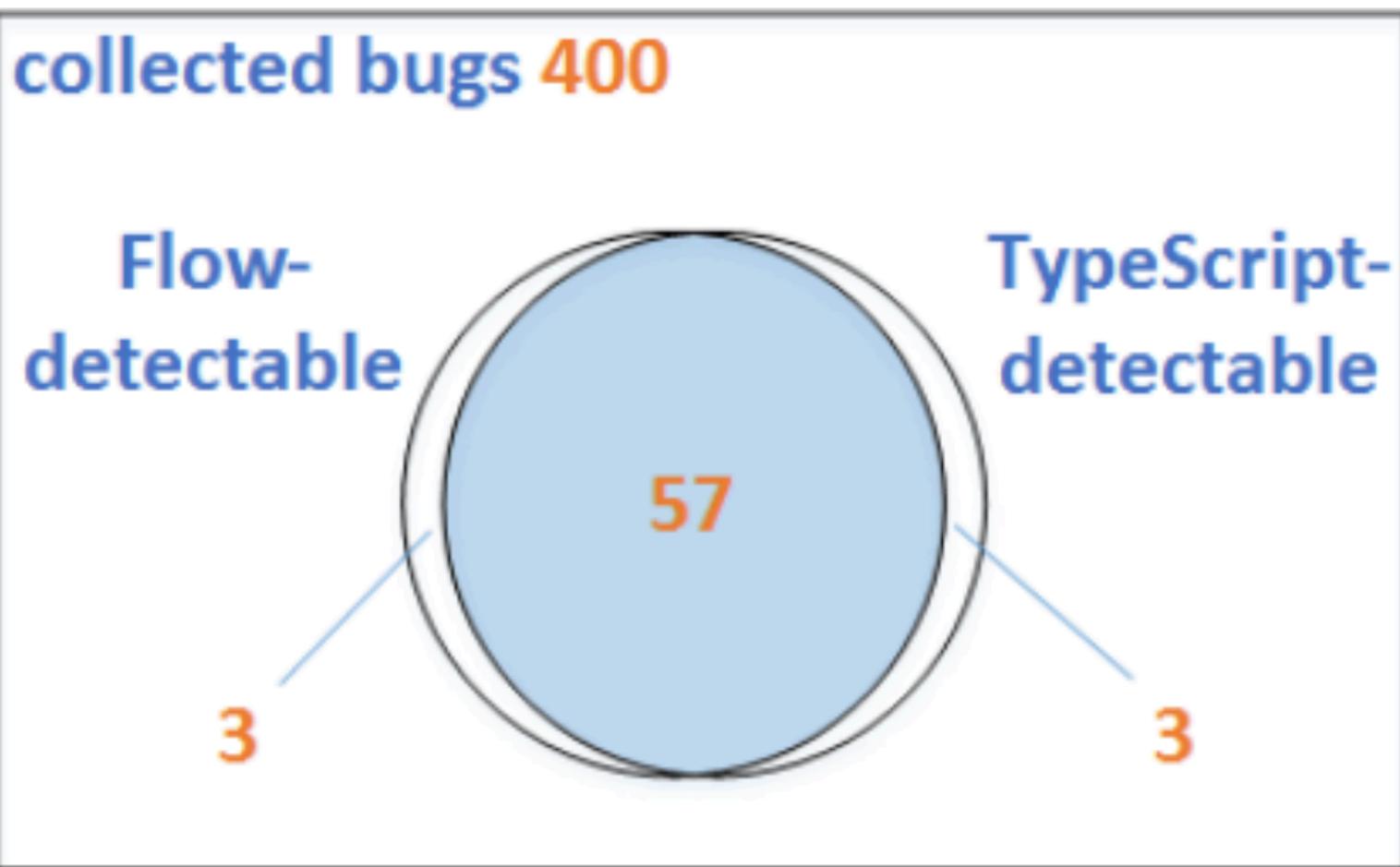


Fig. 4: The workflow of our experiment.

타입 체커가 버그를 고치는 커밋 직전의 코드베이스를 통과시켜주지 않는다면, 해당 버그는 정적 타입 시스템으로 탐지 및 예방할 수 있었을 것.

# 실험 결과



“...using Flow or TypeScript could have prevented **15% of the public bugs** for public projects on GitHub.”

Fig. 5: Venn Diagram of Flow- and TypeScript-detectable bugs.

# Microsoft 엔지니어링 매니저의 코멘트

“충격적이다. 만약 개발하는 방식에 어떤 변화를 주어서 저장소에 들어오는 버그 중 10% 이상을 줄일 수 있다면, 고민할 이유가 전혀 없다. 개발에 쓰이는 시간이 두 배 이상 늘어나거나 하지 않는 한, 우리는 그 변화를 택할 것이다. *That's shocking. If you could make a change to the way we do development that would reduce the number of bugs being checked in by 10% or more overnight, that's a no-brainer. Unless it doubles development time or something, we'd do it.*”

# Microsoft 엔지니어링 매니저의 코멘트

“충격적이다. 만약 개발하는 방식에 어떤 변화를 주어서 **저장소에 들어오는 버그 중 10% 이상을 줄일 수 있다면**, 고민할 이유가 전혀 없다. 개발에 쓰이는 시간이 두 배 이상 늘어나거나 하지 않는 한, 우리는 그 변화를 택할 것이다. *That's shocking. If you could make a change to the way we do development that would reduce the number of bugs being checked in by 10% or more overnight, that's a no-brainer. Unless it doubles development time or something, we'd do it.*”

# Microsoft 엔지니어링 매니저의 코멘트

“충격적이다. 만약 개발하는 방식에 어떤 변화를 주어서 **저장소에 들어오는 버그 중 10% 이상을 줄일 수 있다면, 고민할 이유가 전혀 없다.** 개발에 쓰이는 시간이 두 배 이상 늘어나거나 하지 않는 한, 우리는 그 변화를 택할 것이다. *That's shocking. If you could make a change to the way we do development that would reduce the number of bugs being checked in by 10% or more overnight, that's a no-brainer. Unless it doubles development time or something, we'd do it.*”

(물론 TypeScript 만든 회사가 MS긴 합니다)

# 그뿐만이 아닙니다



# 정적 타입 시스템을 통해…

- 버그를 발견하는 시점을 앞당길 수 있습니다.

# 정적 타입 시스템을 통해…

- 버그를 발견하는 시점을 앞당길 수 있습니다.
- 코드와 더 밀접히 연결된 문서화가 가능해집니다.

# 정적 타입 시스템을 통해…

- 버그를 발견하는 시점을 앞당길 수 있습니다.
- 코드와 더 밀접히 연결된 문서화가 가능해집니다.
- 리팩토링이 용이해집니다.

# 흔한 착각

- “우리는 너무 바빠서 타입 시스템을 도입할 시간이 없어요.”
- 더 나은 문서화 + 더 나은 IDE 사용 + 다양한 버그 사전 예방
- 타입 시스템을 통해 더 안전하게, 그리고 더 빠르게 개발할 수 있습니다.

정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.

# 좋은 건 알겠고, 어떤 선택지가 있나요?

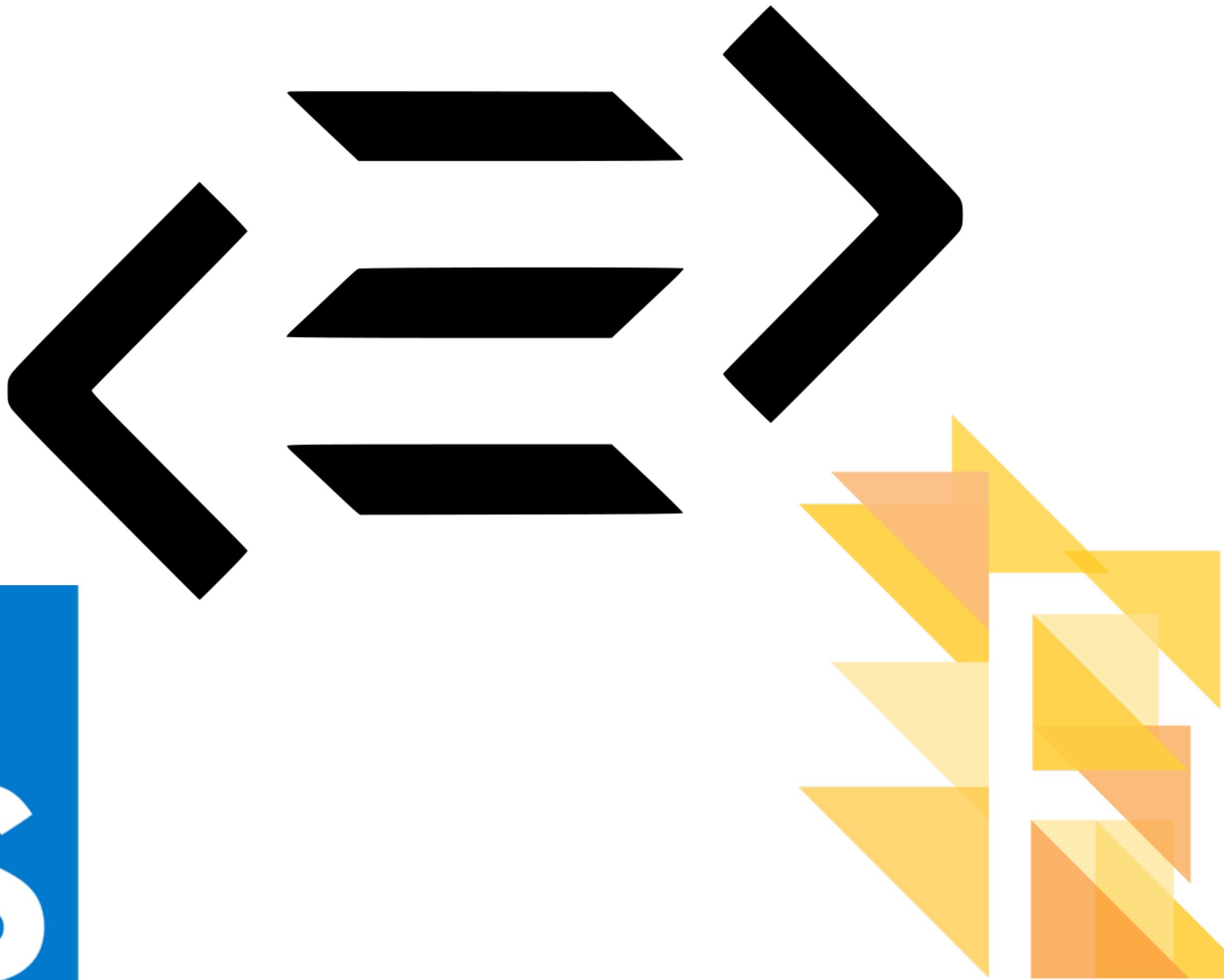
왜 [ 스포일러 ] 를 써야 하나요?

어떻게 도입하나요?

직접 해보면서 느낀 점 없으신가요?

그래서 오늘 뭐라고 하셨죠?

# 수많은 옵션



# 두 가지 옵션



<https://www.typescriptlang.org/>



<https://flow.org/>

# TypeScript vs. Flow

	TypeScript	Flow
만든이	Microsoft	Facebook
첫 릴리즈	2012.10.01	2014.11.19
사용례	Hyperconnect, Reddit, Tumblr, Slack, VS Code, Angular, ... <small>(<a href="http://www.typescriptlang.org/community/friends.html">http://www.typescriptlang.org/community/friends.html</a>)</small>	React, Vue, ...

# 기본 문법

```
// Basics
const num: number = 42
const inferredNum = 1337

// Function
function sum(a: number, b: number): number {
    return a + b
}
sum(1, 2)

// [ts] Argument of type '"string"' is
// not assignable to parameter of type 'number'.
sum('string', 42)

/* Return type can also be inferred to number */
const arrowSum = (a: number, b: number) => a + b
```

```
// Basics
const num: number = 42
const inferredNum = 1337

// Function
function sum(a: number, b: number): number {
    return a + b
}
sum(1, 2)

// 11: sum('string', 42)
//           ^ string. This type is incompatible with the expected param type of
// 8: function sum(a: number, b: number): number {
//           ^ number
sum('string', 42)

/* Return type can also be inferred to number */
const arrowSum = (a: number, b: number) => a + b
```

# 타입 에일리어스와 인터페이스

```
// Type Alias & Interface
type Name = string

interface Animal {
  name: Name;
}

interface Dog extends Animal {
  woof: () => void;
}

const cat: Animal = { name: 'ttypy' }
const dog1: Dog = {
  name: 'flowy',
  woof: () => { console.log('woof!') }
}

let dog2: Dog
dog2 = { name: 'javascripty', woof: () => {} }
```

```
// Type Alias & Interface
type Name = string

interface Animal {
  name: Name;
}

type Dog = Animal & {
  woof: () => void;
}

const cat: Animal = { name: 'ttypy' }
const dog1: Dog = {
  name: 'flowy',
  woof: () => { console.log('woof!') }
}

let dog2: Dog
dog2 = { name: 'javascripty', woof: () => {} }
```

# 제네릭

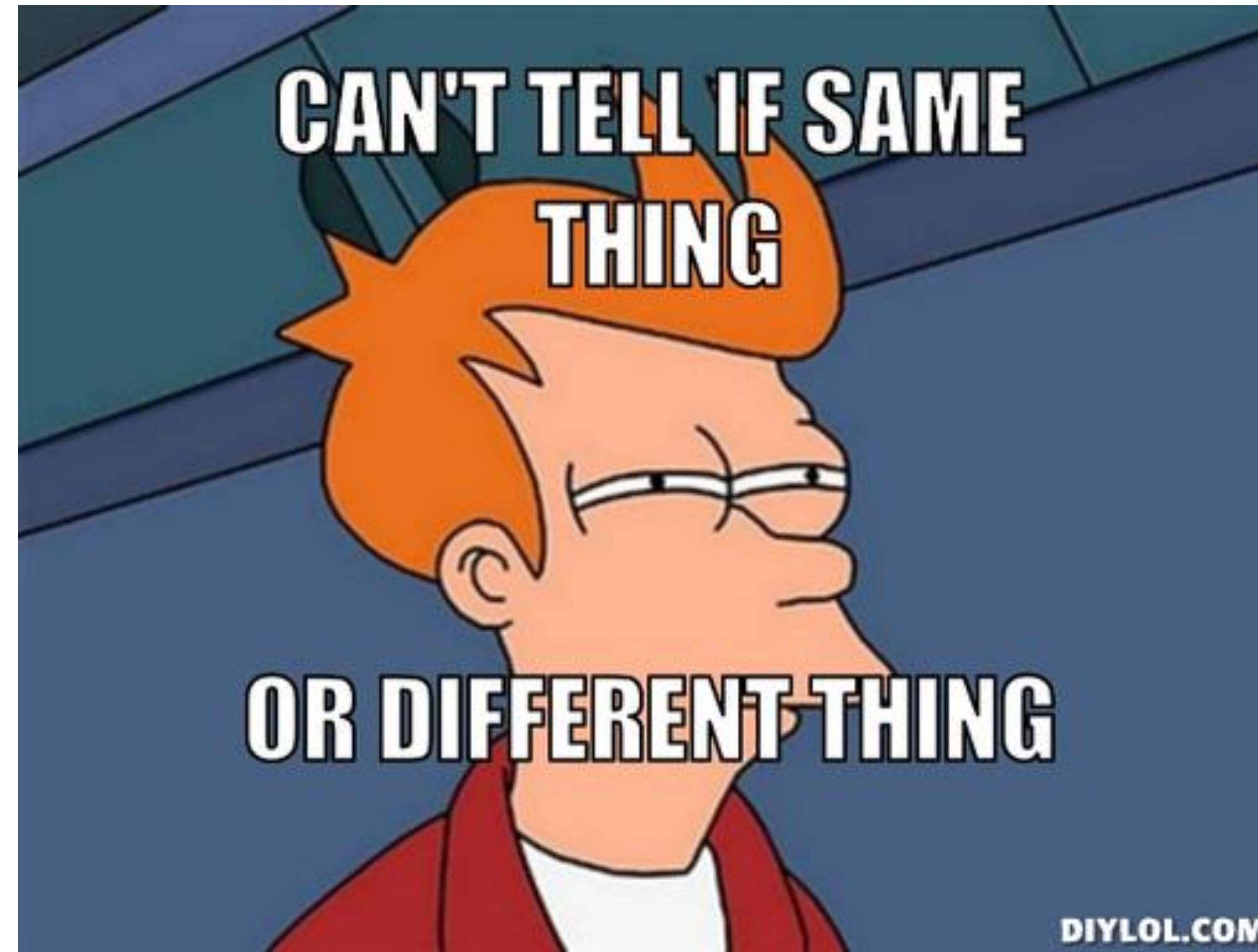
```
// Generic
function findFirstNotNull<T>(arr: T[]): T | null {
    arr.forEach(element => {
        if (element != null) {
            return element
        }
    })
    return null
}

// Inferred to Dog | null
const dog = findFirstNotNull([dog1, dog2])
// Inferred to Animal | null
const animal = findFirstNotNull([dog1, dog2, cat])
```

```
// Generic
function findFirstNotNull<T>(arr: T[]): T | null {
    arr.forEach(element => {
        if (element != null) {
            return element
        }
    })
    return null
}

// Inferred to Dog | null
const dog = findFirstNotNull([dog1, dog2])
// Inferred to Animal | null
const animal = findFirstNotNull([dog1, dog2, cat])
```

# 기본적으로는 많이 비슷합니다



# 뭘 써야 하나요?

- TypeScript와 Flow 둘 다 훌륭한 도구입니다.

# 뭘 써야 하나요?

- TypeScript와 Flow 둘 다 훌륭한 도구입니다.
- 앞서 보셨듯 둘은 상당히 비슷하고, 한 쪽이 더 우월하다 말할 수 없겠죠.

# 뭘 써야 하나요?

- TypeScript와 Flow 둘 다 훌륭한 도구입니다.
- 앞서 보셨듯 둘은 상당히 비슷하고, 한 쪽이 더 우월하다 말할 수 없겠죠.
- 잘 고민 해 보시고 프로젝트의 성격에 맞는 도구를 잘 골라서 쓰세요 😊

# 뭘 써야 하나요?

- TypeScript와 Flow 둘 다 훌륭한 도구입니다.
- 앞서 보셨듯 둘은 상당히 비슷하고, 한 쪽이 더 우월하다 말할 수 없겠죠.
- 잘 고민 해 보시고 프로젝트의 성격에 맞는 도구를 잘 골라서 쓰세요 😇
- 🤞 틀리기도, 실제로 도움이 되기도 힘든 말.

# 뭘 써야 하나요?

- TypeScript와 Flow 둘 다 훌륭한 도구입니다.
- 앞서 보셨듯 둘은 상당히 비슷하고, 한 쪽이 더 우월하다 말할 수 없겠죠.
- 잘 고민 해 보시고 프로젝트의 성격에 맞는 도구를 잘 골라서 쓰세요 😇
- 🤞 틀리기도, 실제로 도움이 되기도 힘든 말.
- 적어도 제가 고민할 때 듣고 싶은 말은 아니었습니다.

# 뭘 써야 하나요?

TypeScript 쓰세요.



정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.  
많은 선택지가 있지만, 그 중 TypeScript를 추천합니다.

# 왜 TypeScript를 써야 하나요?

- 어떻게 도입하나요?
- 직접 해보면서 느낀 점 없으신가요?
- 그래서 오늘 뭐라고 하셨죠?

# 왜 TypeScript를 써야 하나요?

- 안전함과 편리함 사이 설득력 있는 트레이드오프
- 훌륭한 IDE 서포트
- 훨씬 거대한 커뮤니티

# 근본적인 철학 차이

**Non-Goals:** [...] Apply a sound or "provably correct" type system. Instead, strike a balance between correctness and productivity.

(<https://github.com/Microsoft/TypeScript/wiki>TypeScript-Design-Goals>)

... because JavaScript was not designed around a type system, Flow sometimes has to make a tradeoff. When this happens **Flow tends to favor soundness over completeness, ensuring that code doesn't have any bugs.**

(<https://flow.org/en/docs/lang/types-and-expressions/#soundness-and-completeness-a-classtoc-idtoc-soundness-and-completeness-hreftoc-soundness-and-completenessa>)

# 근본적인 철학 차이

**Non-Goals:** [...] Apply a sound or "provably correct" type system. Instead, strike a balance between correctness and productivity.

(<https://github.com/Microsoft/TypeScript/wiki>TypeScript-Design-Goals>)

**“안전하고 증명 가능하게 올바른 타입 시스템”은 목표가 아님.**  
올바름과 생산성 사이에 균형을 잡는 것이 목표다.

# 근본적인 철학 차이

비록 간혹 제대로 된 프로그램을 안 통과시켜 줘서 프로그래머의 불편을 초래하는 한이 있더라도  
문제가 생길 수 있는 프로그램을 최대한 잡아내는게 목표.

... because JavaScript was not designed around a type system, Flow sometimes has to make a tradeoff. When this happens **Flow tends to favor soundness over completeness, ensuring that code doesn't have any bugs.**

(<https://flow.org/en/docs/lang/types-and-expressions/#soundness-and-completeness-a-classtoc-idtoc-soundness-and-completeness-hreftoc-soundness-and-completenessa>)

# 근본적인 철학 차이

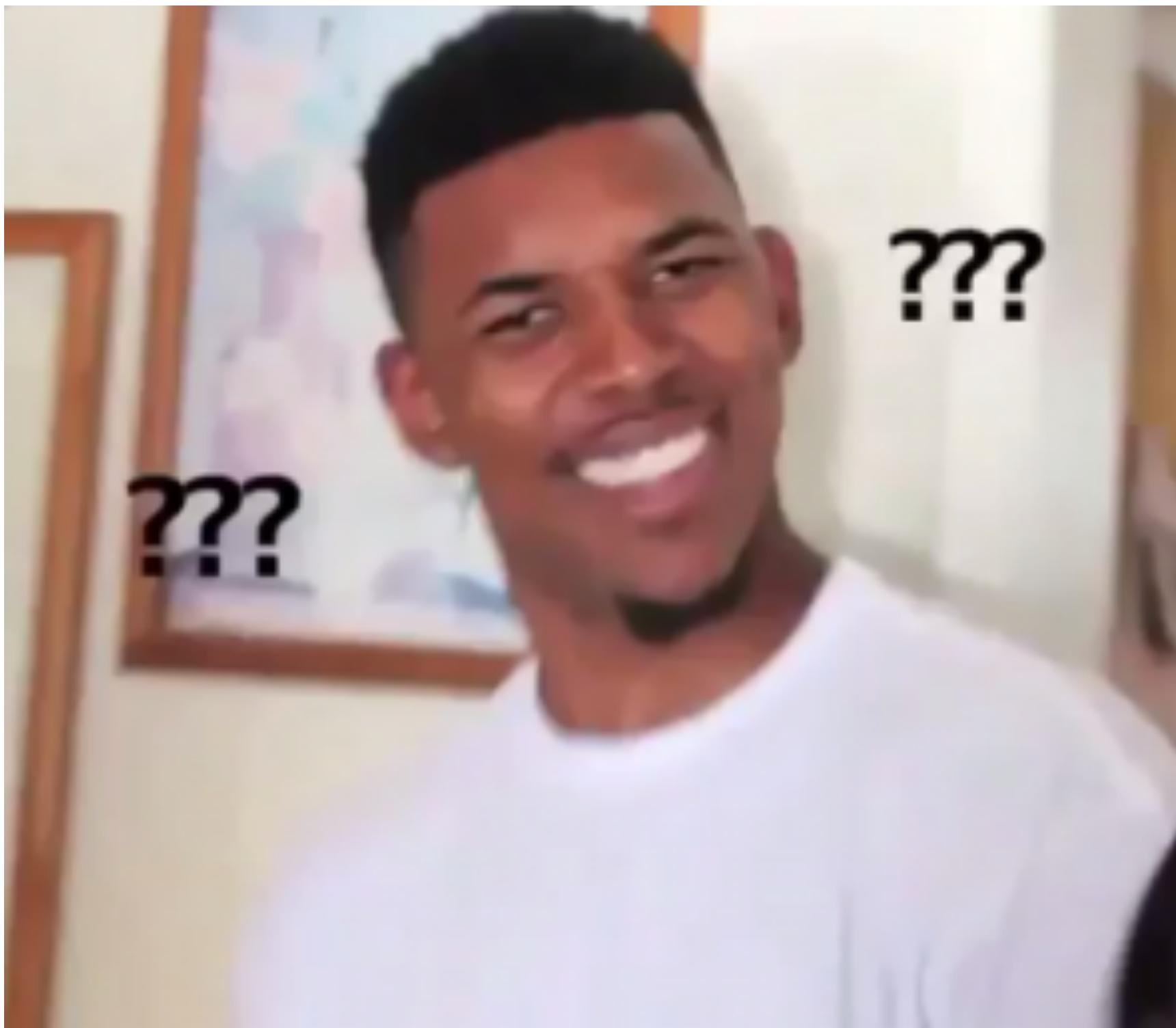
**Non-Goals:** [...] Apply a sound or "provably correct" type system. Instead, strike a balance between correctness and productivity.

(<https://github.com/Microsoft/TypeScript/wiki>TypeScript-Design-Goals>)

... because JavaScript was not designed around a type system, Flow sometimes has to make a tradeoff. When this happens **Flow tends to favor soundness over completeness, ensuring that code doesn't have any bugs.**

(<https://flow.org/en/docs/lang/types-and-expressions/#soundness-and-completeness-a-classtoc-idtoc-soundness-and-completeness-hreftoc-soundness-and-completenessa>)

그러면… flow가 더 좋은 거 아닌가요?



# 무고해 보이는 코드

```
function fn(arg: { x: string | null }) {
    if (arg.x !== null) {
        alert('All is OK!');
        console.log(arg.x.substr(3));
    }
}
```

# 무고해 보이는 코드

```
function fn(arg: { x: string | null }) {
  if (arg.x !== null) {
    alert('All is OK!');
    console.log(arg.x.substr(3));
  }
}
```

- alert는 어떤 함수인가요?

# 무고해 보이는 코드

```
function fn(arg: { x: string | null }) {
    if (arg.x !== null) {
        alert('All is OK!');
        console.log(arg.x.substr(3));
    }
}
```

- alert는 어떤 함수인가요?
- 사용자에게 알럿 창을 띄우는 함수

# 무고해 보이는 코드

```
function fn(arg: { x: string | null }) {
    if (arg.x !== null) {
        alert('All is OK!');
        console.log(arg.x.substr(3));
    }
}
```

- alert는 어떤 함수인가요?
- 사용자에게 알렷 창을 띄우는 함수
- 사실은 알 수 없다!

# 문제적 코드

```
function fn(arg: { x: string | null }) {
  if (arg.x !== null) {
    alert('All is OK!');
    console.log(arg.x.substr(3));
  }
}

let a: { x: string | null } = { x: 'ok' };
function alert(str: string) {
  a.x = null;
}
fn(a);
```

- alert 함수가 실제로는 arg를 변경하는 식으로 재정의되어 있을 수 있음

# 문제적 코드

```
function fn(arg: { x: string | null }) {
  if (arg.x !== null) {
    alert('All is OK!');
    console.log(arg.x.substr(3));
  }
}

let a: { x: string | null } = { x: 'ok' };
function alert(str: string) {
  a.x = null;
}
fn(a);
```

- alert 함수가 실제로는 arg를 변경하는 식으로 재정의 되어 있을 수 있음
- if문 안에서도 다른 함수가 실행 된 후에는 그 조건을 가정할 수 없음

# 문제적 코드

```
function fn(arg: { x: string | null }) {
  if (arg.x !== null) {
    alert('All is OK!');
    console.log(arg.x.substr(3));
  }
}

let a: { x: string | null } = { x: 'ok' };
function alert(str: string) {
  a.x = null;
}
fn(a);
```

- alert 함수가 실제로는 arg를 변경하는 식으로 재정의 되어 있을 수 있음
- if문 안에서도 다른 함수가 실행 된 후에는 그 조건을 가정할 수 없음
- 이런 경우 보다 ‘안전한’ 동작은 많은 경우 그다지 큰 실익 없이 짜증만을 유발함

# 공짜 점심은 없다

- Flow의 안전성(soundness)는 공짜가 아님

# 공짜 점심은 없다

- Flow의 안전성(soundness)는 공짜가 아님
- 언어적 특성 상 자바스크립트 코드에는 항상 나쁜 짓을 할 구석이 숨어있음

# 공짜 점심은 없다

- Flow의 안전성(soundness)는 공짜가 아님
- 언어적 특성 상 자바스크립트 코드에는 항상 나쁜 짓을 할 구석이 숨어있음
  - 일례로 const로 정의한 변수도 재할당이 불가능할 뿐 대부분의 객체가 가변

# 공짜 점심은 없다

- Flow의 안전성(soundness)는 공짜가 아님
- 언어적 특성 상 자바스크립트 코드에는 항상 나쁜 짓을 할 구석이 숨어있음
  - 일례로 const로 정의한 변수도 재할당이 불가능할 뿐 대부분의 객체가 가변
- 이런 ‘안전한 선택’이 보일러플레이트로 돌아오는 경우가 많음

# 공짜 점심은 없다

- Flow의 안전성(soundness)는 공짜가 아님
- 언어적 특성 상 자바스크립트 코드에는 항상 나쁜 짓을 할 구석이 숨어있음
  - 일례로 const로 정의한 변수도 재할당이 불가능할 뿐 대부분의 객체가 가변
- 이런 ‘안전한 선택’이 보일러플레이트로 돌아오는 경우가 많음
- 게다가 오히려 흔한 실수를 잡아주지 못하는 경우도 존재

# switch-case

[flow] number (This type is incompatible with an implicitly-returned undefined.)

```
type TypeSystem =  
  { name: 'TypeScript', definitelyTyped: number }  
 | { name: 'Flow', flowTyped: number }  
  
function getNumberOfDefs(typeSystem: TypeSystem): number {  
  switch (typeSystem.name) {  
    case 'TypeScript':  
      return typeSystem.definitelyTyped  
    case 'Flow':  
      return typeSystem.flowTyped  
    case 'JavaScript':  
      return 0  
  }  
}
```

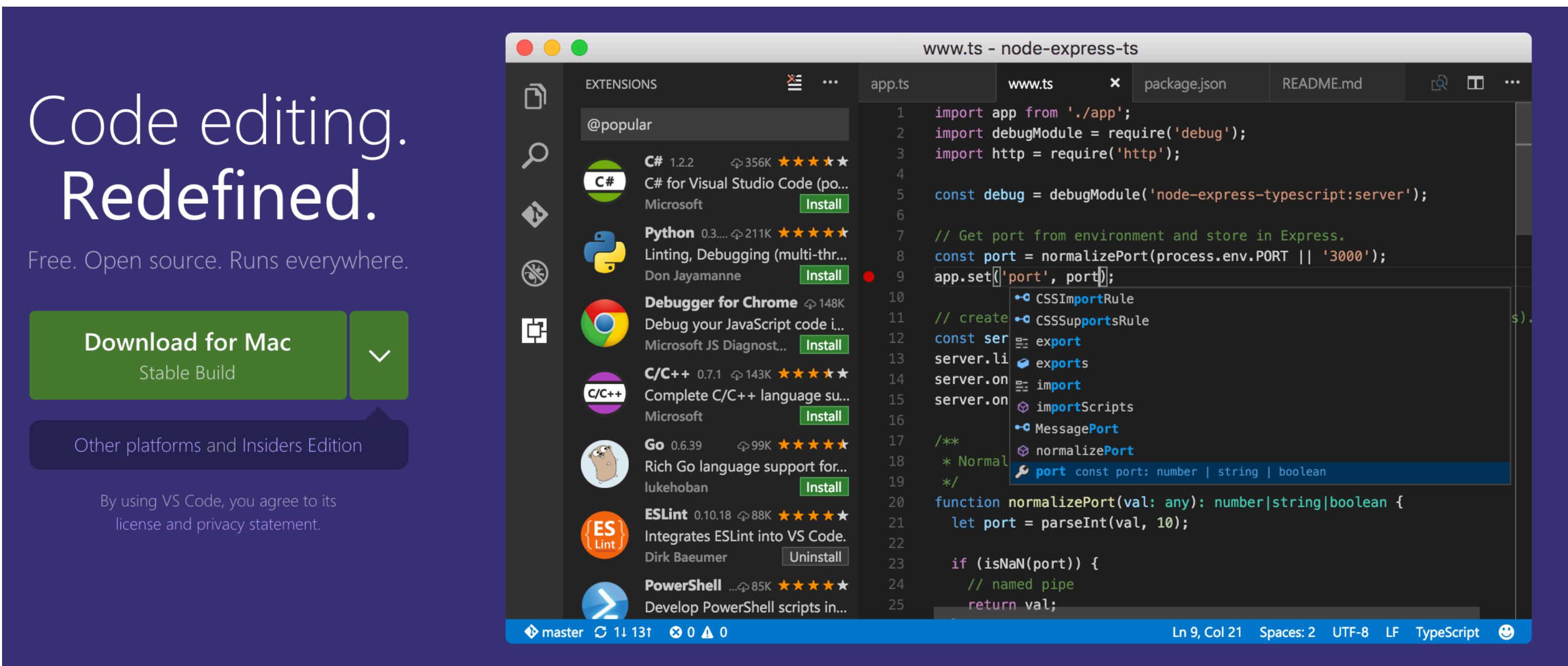
[ts] Type "JavaScript" is not comparable  
to type "TypeScript" | "Flow".

```
type TypeSystem =  
  { name: 'TypeScript', definitelyTyped: number }  
 | { name: 'Flow', flowTyped: number }  
  
function getNumberOfDefs(typeSystem: TypeSystem): number {  
  switch (typeSystem.name) {  
    case 'TypeScript':  
      return typeSystem.definitelyTyped  
    case 'Flow':  
      return typeSystem.flowTyped  
    case 'JavaScript':  
      return 0  
  }  
}
```

# 왜 TypeScript를 써야 하나요?

- 안전함과 편리함 사이 설득력 있는 트레이드오프
- 훌륭한 IDE 서포트
- 훨씬 거대한 커뮤니티

# IDE 서포트 (= VS Code)



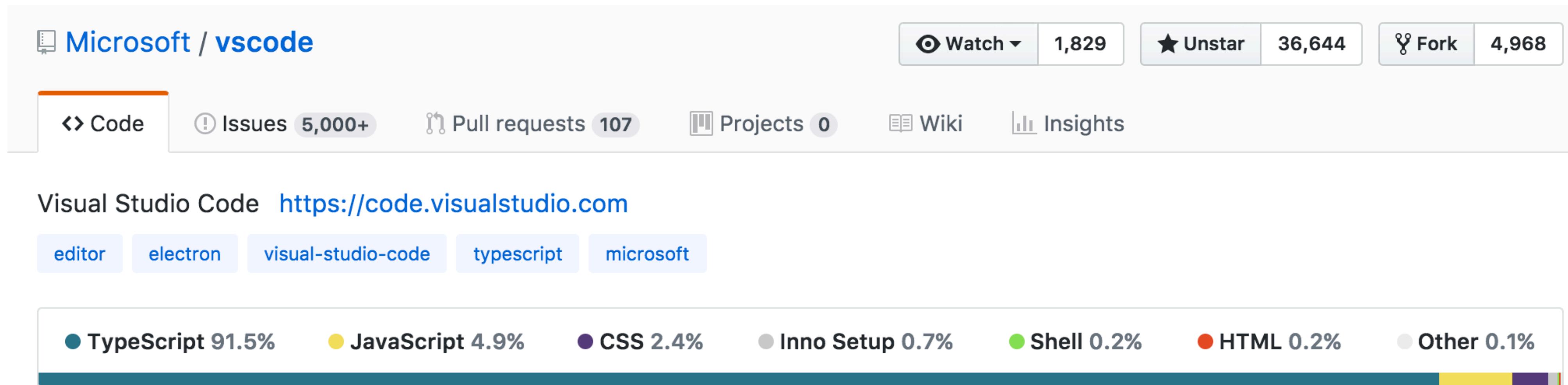
<https://code.visualstudio.com/>

## Projects with the most contributors

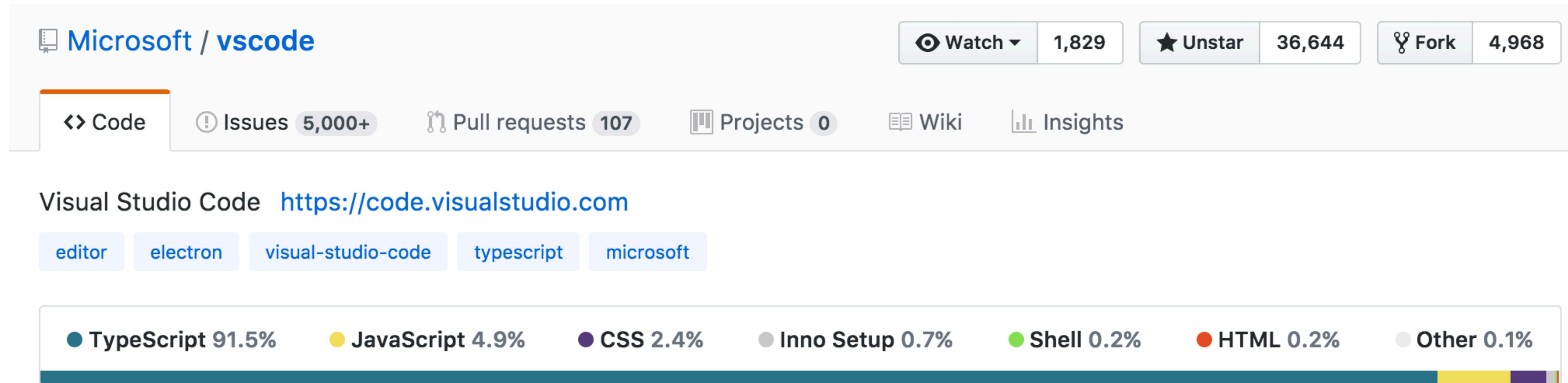


<https://octoverse.github.com/>

# 마이크로소프트의 개밥먹기



# 마이크로소프트의 개밥먹기



- TypeScript로 짜여짐
  - 우리가 TS로 짠 제품이 이 어플리케이션보다 복잡할 가능성은?

# 수많은 기능

- 자동완성, Go To Definition 등을 비롯한 IDE 기본 기능

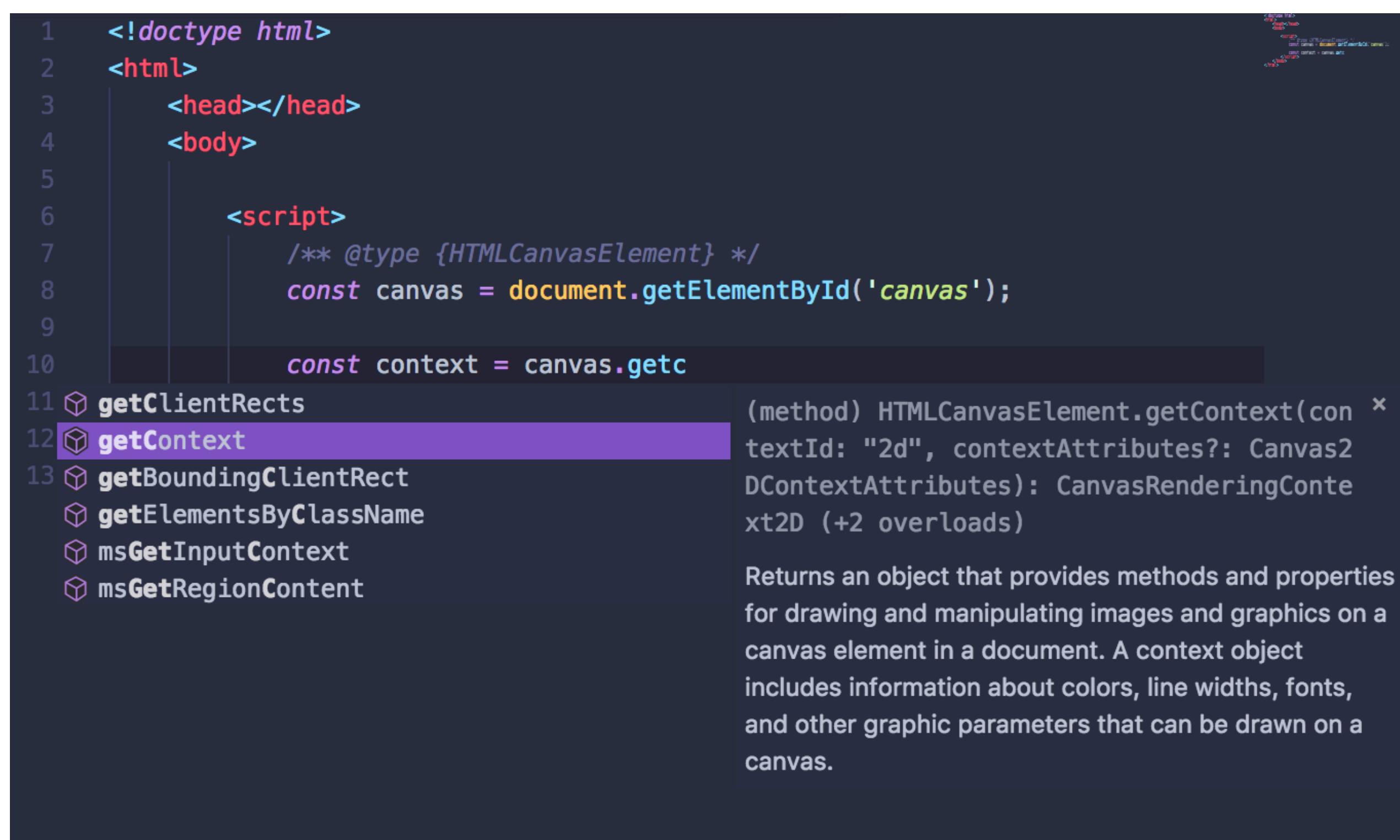
# 수많은 기능

- 자동완성, Go To Definition 등을 비롯한 IDE 기본 기능
- 다양한 리팩토링 기능 (블록, 표현식 함수로 빼기 등) 지원 ([link](#))

# 수많은 기능

- 자동완성, Go To Definition 등을 비롯한 IDE 기본 기능
- 다양한 리팩토링 기능 (블록, 표현식 함수로 빼기 등) 지원 ([link](#))
- JSDoc 정보를 이용한 타입 추론

# VS Code + JSDoc



A screenshot of the Visual Studio Code editor. On the left, there is a code editor window containing the following TypeScript code:

```
1 <!doctype html>
2 <html>
3   <head></head>
4   <body>
5
6     <script>
7       /** @type {HTMLCanvasElement} */
8       const canvas = document.getElementById('canvas');
9
10      const context = canvas.getc
11      ⚡ getClientRects
12      ⚡ getContext
13      ⚡ getBoundingClientRect
14      ⚡ getElementsByClassName
15      ⚡ msGetInputContext
16      ⚡ msGetRegionContent
```

The word "getContext" is highlighted with a purple background, indicating it is being typed or selected. To the right of the code editor, a tooltip provides detailed information about the `getContext` method:

(method) `HTMLCanvasElement.getContext`(`con ×`  
`textId: "2d", contextAttributes?: Canvas2DContextAttributes): CanvasRenderingContext2D (+2 overloads)`

Returns an object that provides methods and properties for drawing and manipulating images and graphics on a canvas element in a document. A context object includes information about colors, line widths, fonts, and other graphic parameters that can be drawn on a canvas.

- TypeScript 2.5~
- JSDoc 주석을 통한 타입 추론
- HTML의 `<script>` 태그 내에서도 이용 가능
- Zero configuration needed
- Flow에선 미지원하는 기능

# 수많은 기능

- 자동완성, Go To Definition 등을 비롯한 IDE 기본 기능
- 다양한 리팩토링 기능 (블록, 표현식 함수로 빼기 등) 지원 ([link](#))
- JSDoc 정보를 이용한 타입 추론
- 써드파티 라이브러리 임포트시 자동으로 타입 정의 설치 ([link](#))

# 수많은 기능

- 자동완성, Go To Definition 등을 비롯한 IDE 기본 기능
- 다양한 리팩토링 기능 (블록, 표현식 함수로 빼기 등) 지원 ([link](#))
- JSDoc 정보를 이용한 타입 추론
- 써드파티 라이브러리 임포트시 자동으로 타입 정의 설치 ([link](#))
- ...

# 왜 TypeScript를 써야 하나요?

- 안전함과 편리함 사이 설득력 있는 트레이드오프
- 훌륭한 IDE 서포트
- 훨씬 거대한 커뮤니티

# 커뮤니티

- <https://eng.lyft.com/typescript-at-lyft-64f0702346ea>

# 커뮤니티

- <https://eng.lyft.com/typescript-at-lyft-64f0702346ea>
- lyft가 타입 시스템 도입을 고민하고 결국 TypeScript를 선택한 경위에 대한 글

# 커뮤니티

- <https://eng.lyft.com/typescript-at-lyft-64f0702346ea>
- lyft가 타입 시스템 도입을 고민하고 결국 TypeScript를 선택한 경위에 대한 글
- 다음 슬라이드 이 글(2017년 9월 28일자)에 언급된 통계 정리
  - 최신은 아니지만 그 동안 급격한 변화가 있진 않았습니다.

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문		
Github 이슈 (Open/Closed)		
Github PR (Open/Closed)		
npm 월간 다운로드		
타입 정의 (외부 리포지토리 + 라이브러리 내장)		

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문	~ 38,000	~ 900
Github 이슈 (Open/Closed)		
Github PR (Open/Closed)		
npm 월간 다운로드		
타입 정의 (외부 리포지토리 + 라이브러리 내장)		

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문	~ 38,000	~ 900
Github 이슈 (Open/Closed)	~2,400 / ~11,200	~1,500 / ~2,200
Github PR (Open/Closed)		
npm 월간 다운로드		
타입 정의 (외부 리포지토리 + 라이브러리 내장)		

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문	~ 38,000	~ 900
Github 이슈 (Open/Closed)	~2,400 / ~11,200	~1,500 / ~2,200
Github PR (Open/Closed)	~100 / ~5,000	~60 / ~1,200
npm 월간 다운로드		
타입 정의 (외부 리포지토리 + 라이브러리 내장)		

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문	~ 38,000	~ 900
Github 이슈 (Open/Closed)	~2,400 / ~11,200	~1,500 / ~2,200
Github PR (Open/Closed)	~100 / ~5,000	~60 / ~1,200
npm 월간 다운로드	<b>~7.2 million</b>	<b>~2.9 million</b>
타입 정의 (외부 리포지토리 + 라이브러리 내장)		

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문	~ 38,000	~ 900
Github 이슈 (Open/Closed)	~2,400 / ~11,200	~1,500 / ~2,200
Github PR (Open/Closed)	~100 / ~5,000	~60 / ~1,200
npm 월간 다운로드	~7.2 million	~2.9 million
타입 정의 (외부 리포지토리 + 라이브러리 내장)	<b>~3,700 + 250k</b>	<b>~340 + 43k</b>

# 커뮤니티

<https://octoverse.github.com/>

# of	TypeScript
StackOverflow 질문	~ 38,000
Github 이슈 (Open/Closed)	~2,400 / ~11,200
Github PR (Open/Closed)	~100 / ~5,000
npm 월간 다운로드	~7.2 million
타입 정의 (외부 리포지토리 + 라이브러리 내장)	~3,700 + 250k

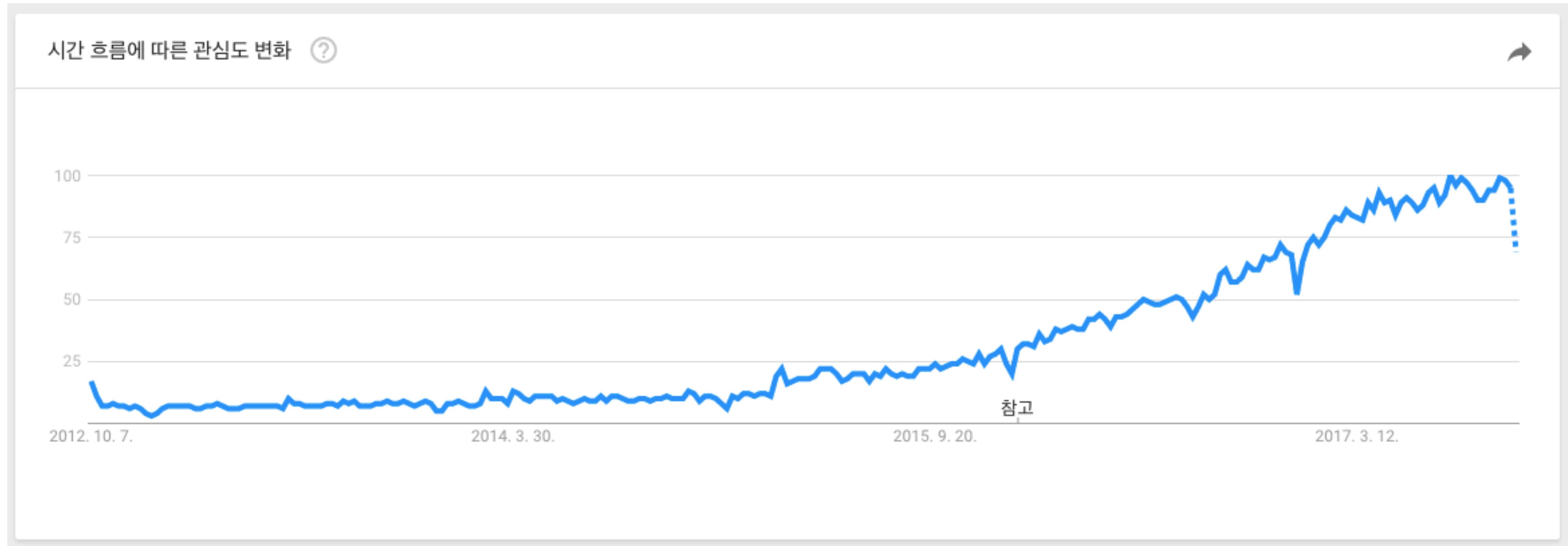
## Projects with the most reviews

-  DEFINITELYTYPED/DEFINITELYTYPED
-  KUBERNETES/KUBERNETES
-  HOMEBREW/HOMEBREW-CORE
-  ANSIBLE/ANSIBLE
-  NODEJS/NODE
-  NIXOS/NIXPKGS
-  APACHE/SPARK
-  RUST-LANG/RUST
-  SYMFONY/SYMFONY
-  TENSORFLOW/TENSORFLOW

# 커뮤니티

# of	TypeScript	Flow
StackOverflow 질문	~ 38,000	~ 900
Github 이슈 (Open/Closed)	~2,400 / ~11,200	~1,500 / ~2,200
Github PR (Open/Closed)	~100 / ~5,000	>> ~60 / ~1,200
npm 월간 다운로드	~7.2 million	~2.9 million
타입 정의 (외부 리포지토리 + 라이브러리 내장)	~3,700 + 250k	~340 + 43k

# TypeScript Google Trend



# 그래도 못 믿으시겠다면…

- 스포카에서...
- 몇 만 LoC Flow 사용 코어 제품 코드베이스를 TypeScript로 갈아엎었습니다.
- 저희 팀에서도 저와 동료 둘이서 비슷한 규모의 일을 했습니다.
- 두 팀 다 결과에 더 할 나위 없이 만족했습니다.

# 간증의 현장



**maru** 2:08 PM

dk

아 진짜

go to definition

차냥해

마이크로소프트님

충성충성충성



2:09 PM

uploaded this image: ⌂ ⌂ ⌂ ⌂ ⌂



2:09 PM

uploaded this image: ⌂ 합니다 ⌂



11:13 AM

ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ

타입스크립트

조와



11:14 AM

타입스크립트 갓갓

MS방향을 향해서 경례



으즈 1

11:16 AM

vscode 에서 잘 쓸 수 있게 되어 있고

조완

# 간증의 현장 (2)

deiangi commented on 29 Jul 2016

@ahejlsberg

I love what you've done with VS Code and TypeScript - that's dream come true for me! Thank you!

I am sure there are other people who contributed to that but for me you Mr. Anders represent all awesome coding tools like Delphi, C# and now TypeScript.

Thank you for existing!

@github How come there is but no tag !?

318 11 83 257 5 366

DanielRosenwasser added Won't Fix Duplicate Canonical Working as Intended Awaiting More Feedback labels on 29 Jul 2016

RyanCavanaugh added Committed Effort: Difficult and removed Duplicate Won't Fix labels on 30 Jul 2016

“Love for TypeScript” (<https://github.com/Microsoft/TypeScript/issues/10011>)

정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.  
많은 선택지가 있지만, 그 중 TypeScript를 추천합니다.  
합리적 트레이드오프, 훌륭한 개발 환경과 거대한 커뮤니티를 갖췄기 때문입니다.

# 어떻게 도입하나요?

직접 해보면서 느낀 점 없으신가요?  
그래서 오늘 뭐라고 하셨죠?

# 프로젝트에 타입스크립트 도입하기

- Plain JavaScript를 사용중인 프로젝트
- Flow를 사용중인 프로젝트

# 타입스크립트 및 타입 의존성 설치

- npm install -g typescript
- 라이브러리 my-cool-lib을 타입 의존성과 함께 설치
  - 따로 설치할 필요 없이 라이브러리 내에 포함된 경우도 많음
- npm install my-cool-lib @types/my-cool-lib

# tsconfig.json

- 타입스크립트 프로젝트 관련 옵션을 모두 담고 있는 파일
- 루트 경로, 모듈 시스템, 타입 시스템의 엄격한 정도, 타입 정의의 위치와 포함/배제 여부, 포함할 라이브러리, ...
- <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>

# tsconfig.json > compilerOptions

- 타입스크립트 컴파일 관련 옵션을 모두 담은 파일
- 모듈 시스템, 타입 시스템의 엄격한 정도, 타입 정의의 위치와 포함/배제 여부, 포함할 라이브러리, ...

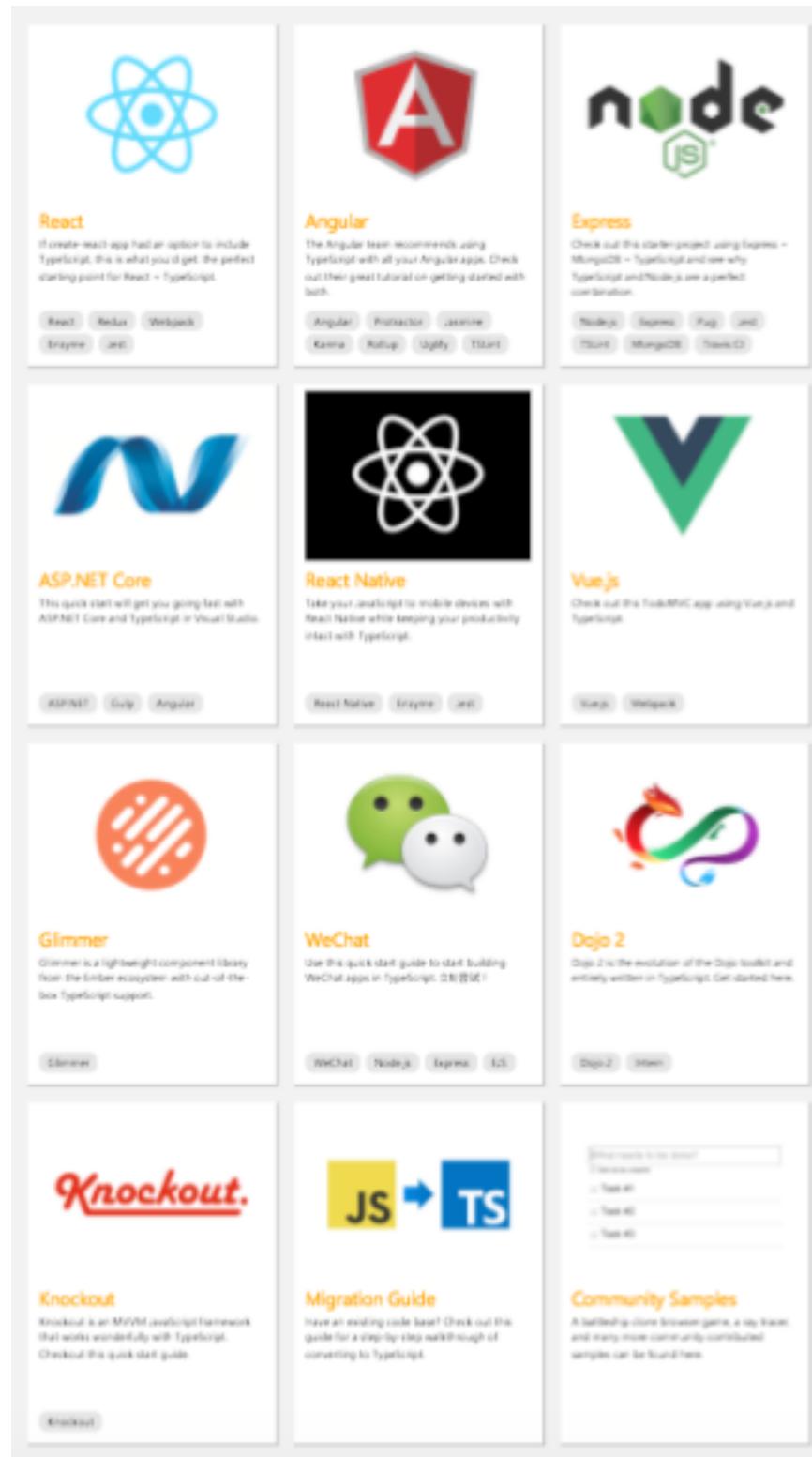
# 중요한 몇 가지 옵션

- —allowSynthesicalDefaultImport
- —strict
  - = —noImplicitAny + —noImplicitThis + —alwaysStrict
  - + —strictNullCheck + —strictFunctionType
- —lib, —target
- 한 번에 다 익히실 필요 없습니다.
  - <https://www.typescriptlang.org/docs/handbook/compiler-options.html>

# Webpack과의 통합

- MS발 공식 로더는 없음
- 두 개의 선택지
  - ts-loader(<https://github.com/TypeStrong/ts-loader>)
  - awesome-typescript-loader(<https://github.com/s-panferov/awesome-typescript-loader>)
- 큰 차이는 없지만 awesome-typescript-loader 추천 (공식 문서 언급)

# 사실 전부 준비되어 있습니다



[https://www.typescriptlang.org/  
samples/index.html](https://www.typescriptlang.org/samples/index.html)

A screenshot of the Microsoft/TypeScriptSamples GitHub repository. The page shows basic repository statistics (209 commits, 4 branches, 0 releases, 30 contributors) and a list of recent commits. The commits are listed in descending order of age, with each entry showing the author, commit message, and date.

[https://github.com/Microsoft/TypeScriptSamples/](https://github.com/Microsoft/TypeScriptSamples)

## Tutorials

- [Using React and Webpack with TypeScript](#)
- [Using Knockout with TypeScript](#)
- [Using ASP.NET Core with TypeScript](#)
- [Using ASP.NET 4 with TypeScript](#)
- [Setting up Gulp with TypeScript, Browserify, Uglify, and Watchify](#)

<https://github.com/Microsoft/TypeScript/wiki>

- [https://  
stackoverflow.com/  
questions/tagged/  
typescript](https://stackoverflow.com/questions/tagged/typescript)

# Declaration File 추가

```
declare module "*.vue" {  
    import Vue from 'vue'  
    export default typeof Vue  
}  
  
declare module "*.json";
```

- <https://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html>
- 컴파일러에게 어떤 모듈의 내용물이 어떻게 생겼는지 알리기 위한 수단
- .js(x), .ts(x) 이외의 모듈을 불러오고 위해선 (e.g. css-in-js) 반드시 필요함

# 점진적 타이핑

- 코드 베이스 전부를 한 번에 옮겨오실 필요 없습니다.

# 점진적 타이핑

- 코드 베이스 전부를 한 번에 옮겨오실 필요 없습니다.
- 파일별 적용이 가능합니다.

# 점진적 타이핑

- 코드 베이스 전부를 한 번에 옮겨오실 필요 없습니다.
- 파일별 적용이 가능합니다.
- 한 파일 내에서도 부분적용이 가능합니다.

# 파일별 적용

- myScript.js -> myScript.ts
- myComponent.jsx -> myScript.tsx

# 파일별 적용

- JS에서 TS 임포트
  - 웹팩을 통해서 (주천) or 컴파일 결과물 임포트
- TS에서 JS 임포트
  - 1.8~: —allowJs 옵션을 통해 가능
  - 2.3~: —checkJs 옵션을 통해 JS 파일도 체크 가능 – JSDoc도 지원  
<https://github.com/Microsoft/TypeScript/wiki>Type-Checking-JavaScript-Files>

# 한 파일 내 부분적용

- 타입 추론

- 명시적인 타입 정보 없이도 추론이 된다

```
// Return type is inferred to any
function a(arg: string) {
  return arg.substr(0, arg.length - 1)
}
```

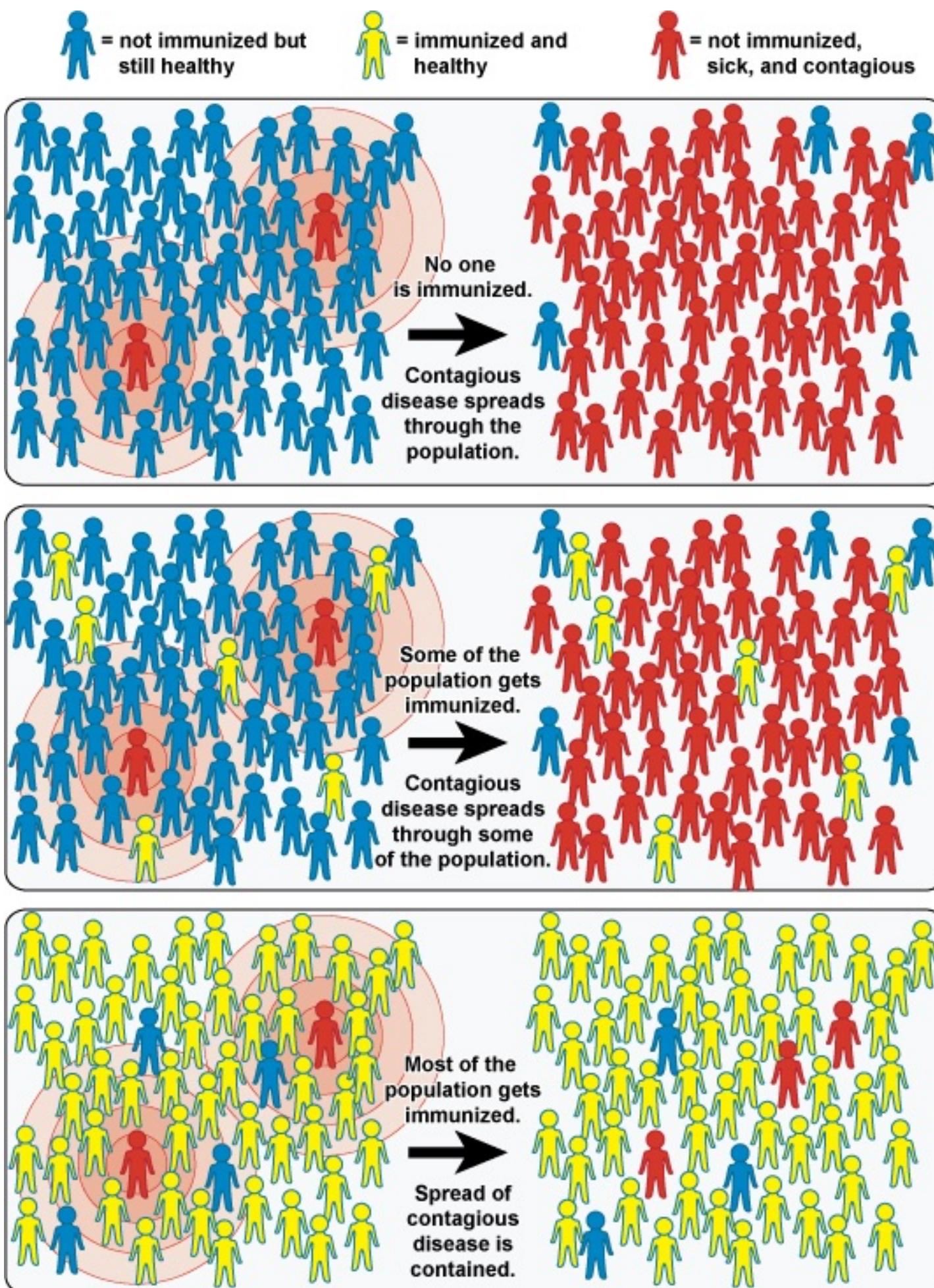
- 타입 강제

- 일단 때우고 넘어갈 수 있다

```
function dangerouslyGetName() {
  // This could be null
  const myValue = window.localStorage.getItem('myKey') as string

  // This could be anything
  const myObj = JSON.parse(myValue) as { name: string }
  return myObj.name
}
```

# 주의할 점 : 집단 면역



- 구성원 대부분이 면역된 경우가 아니면 집단 수준에서 큰 효과를 발휘하기 힘듬
- 타입 정보가 없는 코드 부스러기는 마치 전염병의 바이러스와 같음
- 비록 파일 별/부분 별 도입이 가능할지라도 **대부분의 코드가 타이핑 되어 있을 때 타입 시스템의 진가가 발휘됨**

# Flow → TypeScript

- 앞선 작업들
- + 바이너리/웹팩 설정 수정
- + 문법 수정

# 바이너리/웹팩 설정 수정

- 바이너리
  - flow-bin -> tsc
- 웹팩 설정
  - babel-preset-flow 삭제

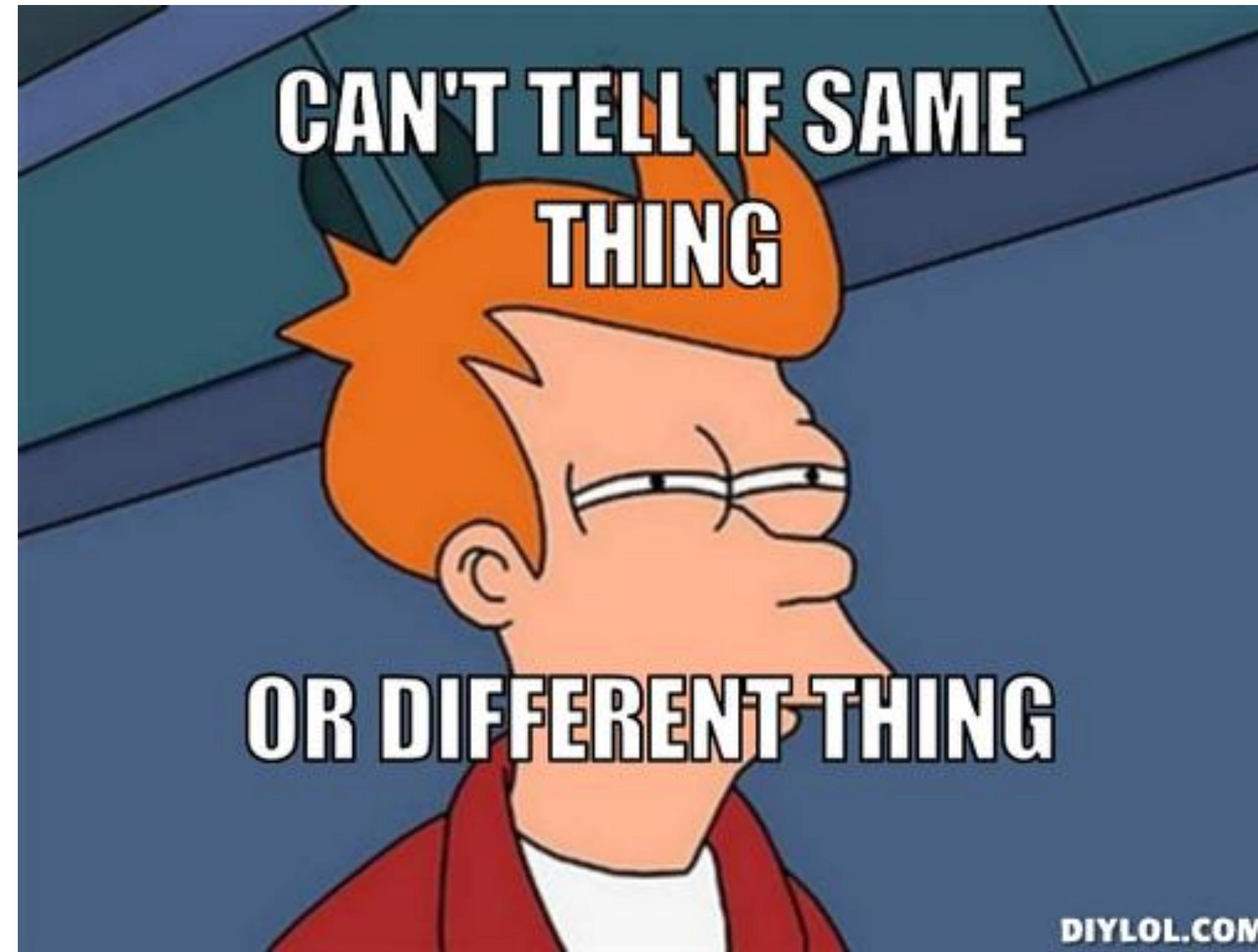
# 주목할 만한 문법 차이

- 타입 임포트
  - Flow: 타입 임포트를 ES6 임포트와 별개로 취급  
(*import type { MyType }* ) 또는 *import { type MyType }* )
  - TS: ES6 임포트와 동일 (*import { MyType }* )

# 주목할 만한 문법 차이

- 인터페이스
  - Flow: *type Animal = { name: string }*
  - TS: *interface Animal { name: string; }*
- 인터페이스 확장
  - Flow: *type Dog = Animal & { bark: () => void }*
  - TS: *interface Dog extends Animal { bark: () => void; }*

# 기본적으로는 많이 비슷합니다



정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.

많은 선택지가 있지만, 그 중 TypeScript를 추천합니다.

합리적 트레이드오프, 훌륭한 개발 환경과 거대한 커뮤니티를 갖췄기 때문입니다.

Webpack을 통해 프로젝트의 작은 부분씩 도입이 가능하며 다양한 리소스가 존재합니다.

# 직접 해보면서 느낀 점 있으신가요?

그래서 오늘 뭐라고 하셨죠?

# 교훈

- 생각만큼 어렵지 않습니다
  - 유용한 전략: 이슈를 해결할 때마다 관련된 파일을 전부 TS로 옮긴다

# 교훈

- 생각만큼 어렵지 않습니다
  - 유용한 전략: 이슈를 해결할 때마다 관련된 파일을 전부 TS로 옮긴다
  - 얼마나 자잘한 버그가 많았는지 깨달음
  - TS의 에러를 보고 “어 그러네... 문제 생길 수 있구나” 했던 적이 여러번

# 교훈

- 생각만큼 어렵지 않습니다
  - 유용한 전략: 이슈를 해결할 때마다 관련된 파일을 전부 TS로 옮긴다
  - 얼마나 자잘한 버그가 많았는지 깨달음
  - TS의 에러를 보고 “어 그러네... 문제 생길 수 있구나” 했던 적이 여러번
- 리팩토링이 너무 쉬워짐
  - 일단 부숴! 그리고 만들어!

# 시행착오

- JS(babel-loader) + TS(awesome-ts-loader)의 공존
- Flow 타입 어노테이션 + —allowJS의 공존
- 타입 가드로서의 `in` 연산자

# **babel-loader + ts-loader**

- TS와 JS 파일이 공존할 때 TS는 babel-loader 태우지 않고 사용

# **babel-loader + ts-loader**

- TS와 JS 파일이 공존할 때 TS는 babel-loader 태우지 않고 사용
- babel-transform-runtime 을 사용중이던 상황

# babel-loader + ts-loader

- TS와 JS 파일이 공존할 때 TS는 babel-loader 태우지 않고 사용
- babel-transform-runtime 을 사용중이던 상황
- TS 파일과 JS 파일에서의 Map, Set 등이 다른 객체를 참조

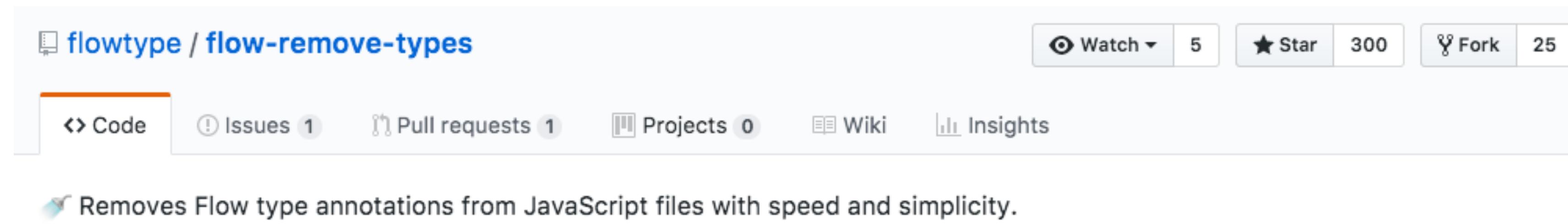
# babel-loader + ts-loader

- TS와 JS 파일이 공존할 때 TS는 babel-loader 태우지 않고 사용
- babel-transform-runtime 을 사용중이던 상황
- TS 파일과 JS 파일에서의 Map, Set 등이 다른 객체를 참조
- 둘이 공존하는 동안은
  - Global State를 더럽히는 폴리필을 쓰거나
  - 타입스크립트 로더의 결과물도 babel을 태웁시다

# Flow 타입 어노테이션 + --allowJs

- —allowJs 옵션을 켜고 Flow의 타입 어노테이션을 사용한 JS를 임포트 시
- error TS8010: 'types' can only be used in a .ts file.

# flow-remove-types



- <https://github.com/flowtype/flow-remove-types>
- flow 타입 어노테이션을 깔끔하게 지워줌
- 해당 캐릭터를 전부 지우거나 공백으로 치환하는 두 옵션이 있음
  - 주석으로 처리하는 옵션이 있으면 편리할텐데... (누가 좀 만들어 주세요)

# 타입 가드로서의 `in` 연산자

```
interface A {  
    x: number;  
}  
interface B {  
    y: string;  
}  
  
let q: A | B = ...;  
if ('x' in q) {  
    // q: A  
} else {  
    // q: B  
}
```

- <https://github.com/Microsoft/TypeScript/issues/10485>
- 왼쪽 코드 같은 동작을 허용하자는 프로포절
- Workaround 1  
-> Disjoint Union 만을 사용한다

# 타입 가드로서의 `in` 연산자

```
export function hasKey<K extends string>(k: K, o: {}): o is { [key: K]: {} } {  
    return typeof o === 'object' && k in o  
}
```

```
type Foo = { x: number } | { y: string }  
  
function f(foo: Foo) {  
    if (hasKey('x', foo)) {  
        console.log(foo.x + 5)  
    } else {  
        console.log(foo.y.length)  
    }  
}
```

- Workaround 2  
-> hasKey (해당 쓰레드 [@pelotom](#))
- optional 필드에 대해선 정상 작동 X

정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.

많은 선택지가 있지만, 그 중 TypeScript를 추천합니다.

합리적 트레이드오프, 훌륭한 개발 환경과 거대한 커뮤니티를 갖췄기 때문입니다.

Webpack을 통해 프로젝트의 작은 부분씩 도입이 가능하며 다양한 리소스가 존재합니다.

실제로 도입하며 몇몇 시행착오도 겪었지만 결과적으로 그 효과를 실감하고 만족했습니다.

# 그래서 오늘 뭐라고 하셨죠?

# 제가 답변 드리고자 했던 질문들

- 정적 타이핑을 쓰면 뭐가 좋나요?
- 좋은 건 알겠고, 어떤 선택지가 있나요?
- 왜 TypeScript를 써야 하나요?
- 어떻게 도입하나요?
- 직접 해보면서 느낀 점 없으신가요?
- 그래서 오늘 뭐라고 하셨죠?

# 제가 드린 답변들

- 정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.
- 많은 선택지가 있지만, 그 중 TypeScript를 추천합니다.
- 합리적 트레이드오프, 훌륭한 개발 환경과 거대한 커뮤니티를 갖췄기 때문입니다.
- Webpack을 통해 프로젝트의 작은 부분씩 도입이 가능하며 다양한 리소스가 존재합니다.
- 실제로 도입하며 몇몇 시행착오도 겪었지만 결과적으로 그 효과를 실감하고 만족했습니다.
- 한국에서도 더 많은 사용자와 경험담, 노하우가 생겼으면 합니다. 지금 바로 시작하세요!

# 제가 드린 답변들

- 정적 타이핑을 통해 코드의 가독성과 안정성을 높일 수 있고, 더 빠른 개발이 가능해집니다.
- 많은 선택지가 있지만, 그 중 TypeScript를 추천합니다.
- 합리적 트레이드오프, 훌륭한 개발 환경과 거대한 커뮤니티를 갖췄기 때문입니다.
- Webpack을 통해 프로젝트의 작은 부분씩 도입이 가능하며 다양한 리소스가 존재합니다.
- 실제로 도입하며 몇몇 시행착오도 겪었지만 결과적으로 그 효과를 실감하고 만족했습니다.
- **한국에서도 더 많은 사용자와 경험담, 노하우가 생겼으면 합니다. 지금 바로 시작하세요!**

# Appendix

# 참고 자료

- To Type or Not to Type: Quantifying Detectable Bugs in JavaScript
- TypeScript Design Goals
- Github Octoverse 2017
- Trade-offs in Control Flow Analysis #9998
- Treat `in` operator as type guard #10485
- Various TypeScript & VS Code release notes

# 참고 자료

- Why We Chose TypeScript
- TypeScript at Lyft
- Flow and TypeScript
- TypeScript vs. Flow
- Flow vs. TypeScript
- Type Systems for JavaScript: Elm, Flow, and TypeScript