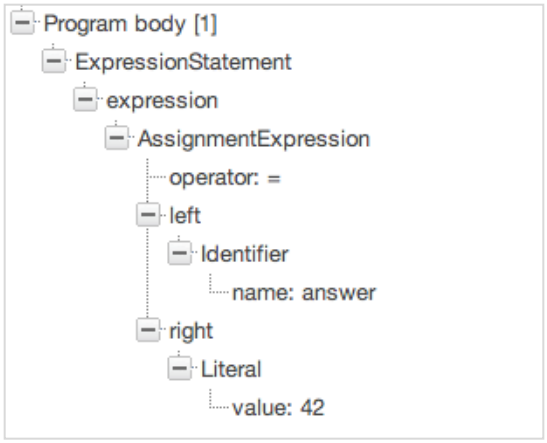
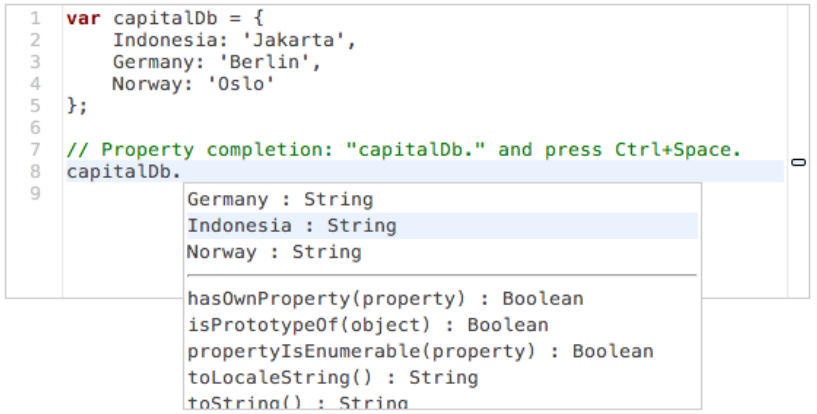
# JSDoc*, JSIDL*, Code Editors

# Static Analysis of JavaScript

1. Tokenize and Parse
2. ???
3. Profit  (Program Understanding)




???



# Develop-time uses of Type Inferencing

- Content Assist / Code Completion
- Code Validation
- Jump to Declaration / References
- Refactoring
- Source Templates

# Type Inferencing is a Hard Problem

What we get for free…

- Grammar and Keywords
- Scope Analysis
- Flow Analysis

- Dynamic Code Analysis, Code Heuristics
  - Interesting but increasingly imprecise

# (JSDoc*) - We Need Help from the User

```
/**
 * @typedef Person {firstname: string, lastname: string}
 *
*/

/**
 * @param {Person} person
 */
function greeter(person) {
    return "Hello, " + person.firstname + " " + person.lastname;
}

/** @type Person */
var user = {firstname: "Jane", lastname: "User"};

document.body.innerHTML = greeter(user);
```

- Comment-based
- Descriptive tag (@tagName)
- Text and Location Semantics

# TypeScript to JS+JSDoc

```ts
interface Person {
    firstname: string;
    lastname: string;
}

function greeter(person : Person) {
    return "Hello, " + person.firstname + " " + person.lastname;
}

var user = {firstname: "Jane", lastname: "User"};

document.body.innerHTML = greeter(user);
```



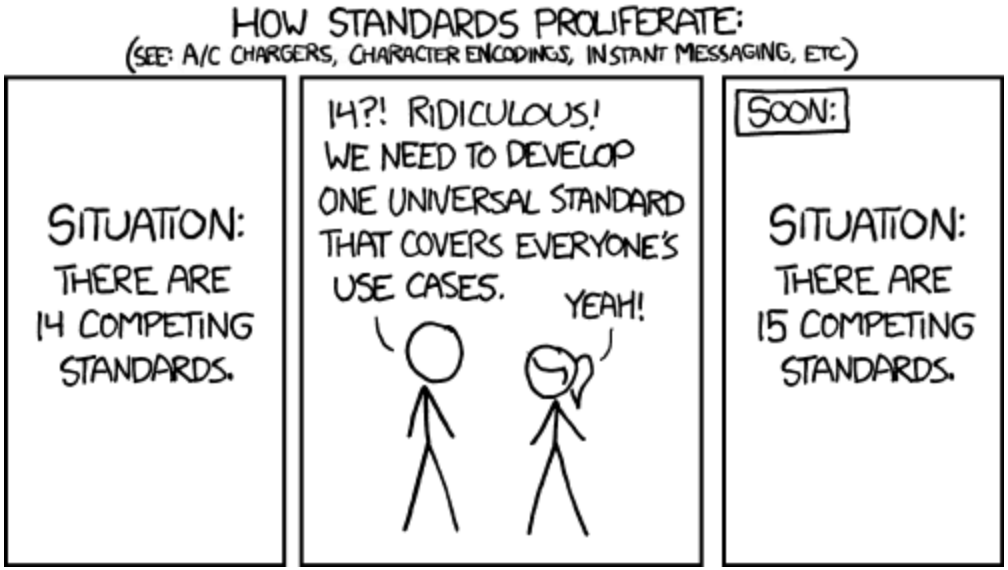
```ts
/**
 * @typedef Person {firstname: string, lastname: string}
 *
*/

/**
 * @param {Person} person
 */
function greeter(person) {
    return "Hello, " + person.firstname + " " + person.lastname;
}

/** @type Person */
var user = {firstname: "Jane", lastname: "User"};

document.body.innerHTML = greeter(user);
```

# Standardize JSDoc?


JSDoc 3

JSDuck

dojo toolkit



# JSDoc Proposal

- All JSDoc groups do not currently support ES6 or even ES5 particularly well

- Socialize idea of common work
- Work with an existing JSDoc definition and set of adopters
- Standardize meaning of  tags, tag content and location semantics

# JS IDL?

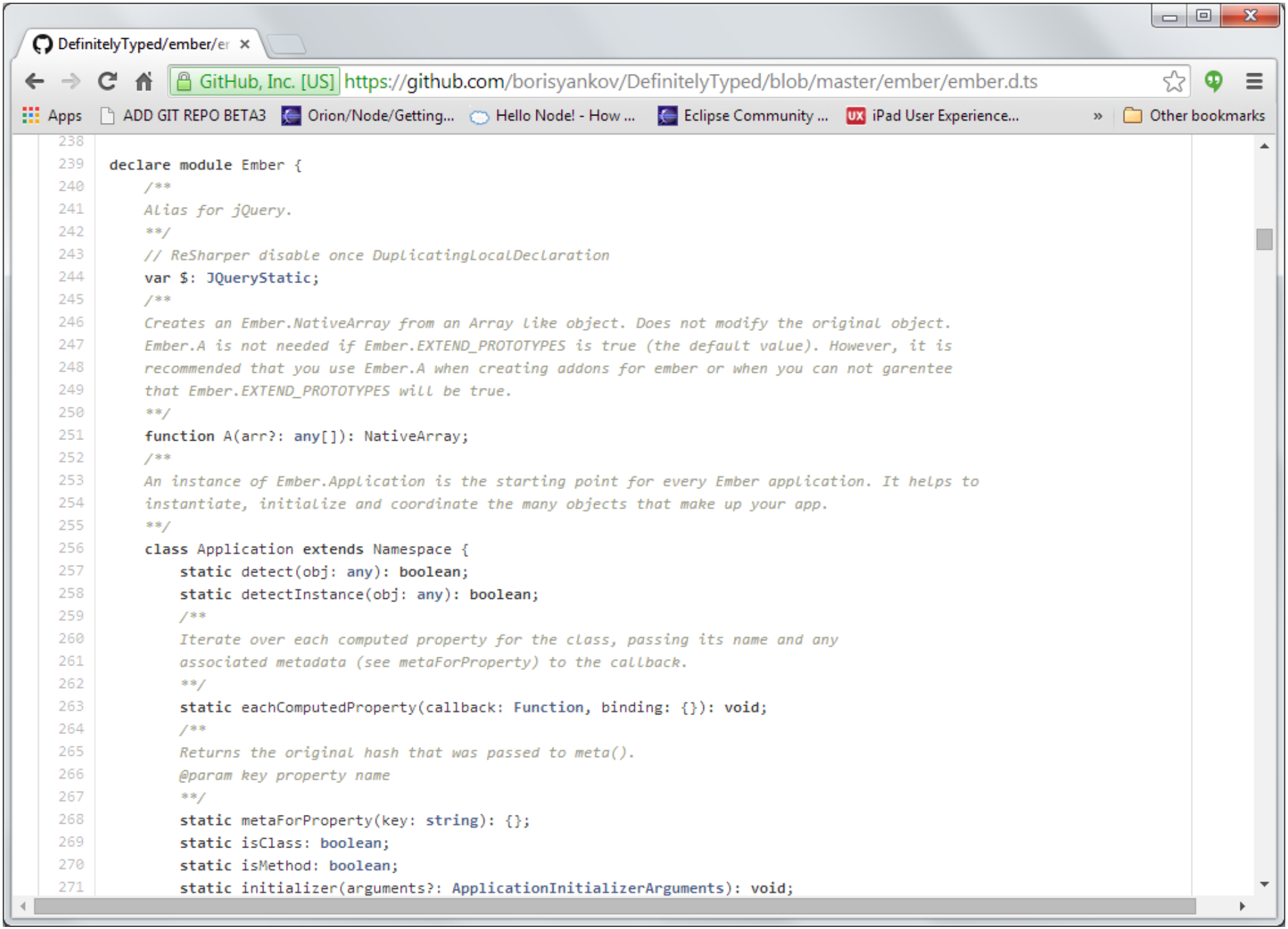
- Representation of the public interface of a software component / artifact
- viewed in the opposite direction is this the product of type inferencing? (e.g. a type definition)

# Type definition…

- TypeScript Definition
  - (Definitely Typed)
- Tern style JSON definition
  - (Supports Type Expressions)(Tern, Orion)
- JSDoc mixed with simple JS definition
  - (common a few years ago)

---

- Green field approaches
  - WebIDL --- > JS IDL
  - JS --- > JS IDL




```
238
239 declare module Ember {
240     /**
241     Alias for jQuery.
242     **/
243     // ReSharper disable once DuplicatingLocalDeclaration
244     var $: JQueryStatic;
245     /**
246     Creates an Ember.NativeArray from an Array like object. Does not modify the original object.
247     Ember.A is not needed if Ember.EXTEND_PROTOTYPES is true (the default value). However, it is
248     recommended that you use Ember.A when creating addons for ember or when you can not garentee
249     that Ember.EXTEND_PROTOTYPES will be true.
250     **/
251     function A(arr?: any[]): NativeArray;
252     /**
253     An instance of Ember.Application is the starting point for every Ember application. It helps to
254     instantiate, initialize and coordinate the many objects that make up your app.
255     **/
256     class Application extends Namespace {
257         static detect(obj: any): boolean;
258         static detectInstance(obj: any): boolean;
259         /**
260         Iterate over each computed property for the class, passing its name and any
261         associated metadata (see metaForProperty) to the callback.
262         **/
263         static eachComputedProperty(callback: Function, binding: {}): void;
264         /**
265         Returns the original hash that was passed to meta().
266         @param key property name
267         **/
268         static metaForProperty(key: string): {};
269         static isClass: boolean;
270         static isMethod: boolean;
271         static initializer(arguments?: ApplicationInitializerArguments): void;
```

```
{
  "!name": "mylibrary",
  "!define": {
    "point": {
      "x": "number",
      "y": "number"
    }
  },
  "MyConstructor": {
    "!type": "fn(arg: string)",
    "staticFunction": "fn() -> bool",
    "prototype": {
      "property": "[number]",
      "clone": "fn() -> +MyConstructor",
      "getPoint": "fn(i: number) -> point"
    }
  },
  "someOtherGlobal": "string"
}
```