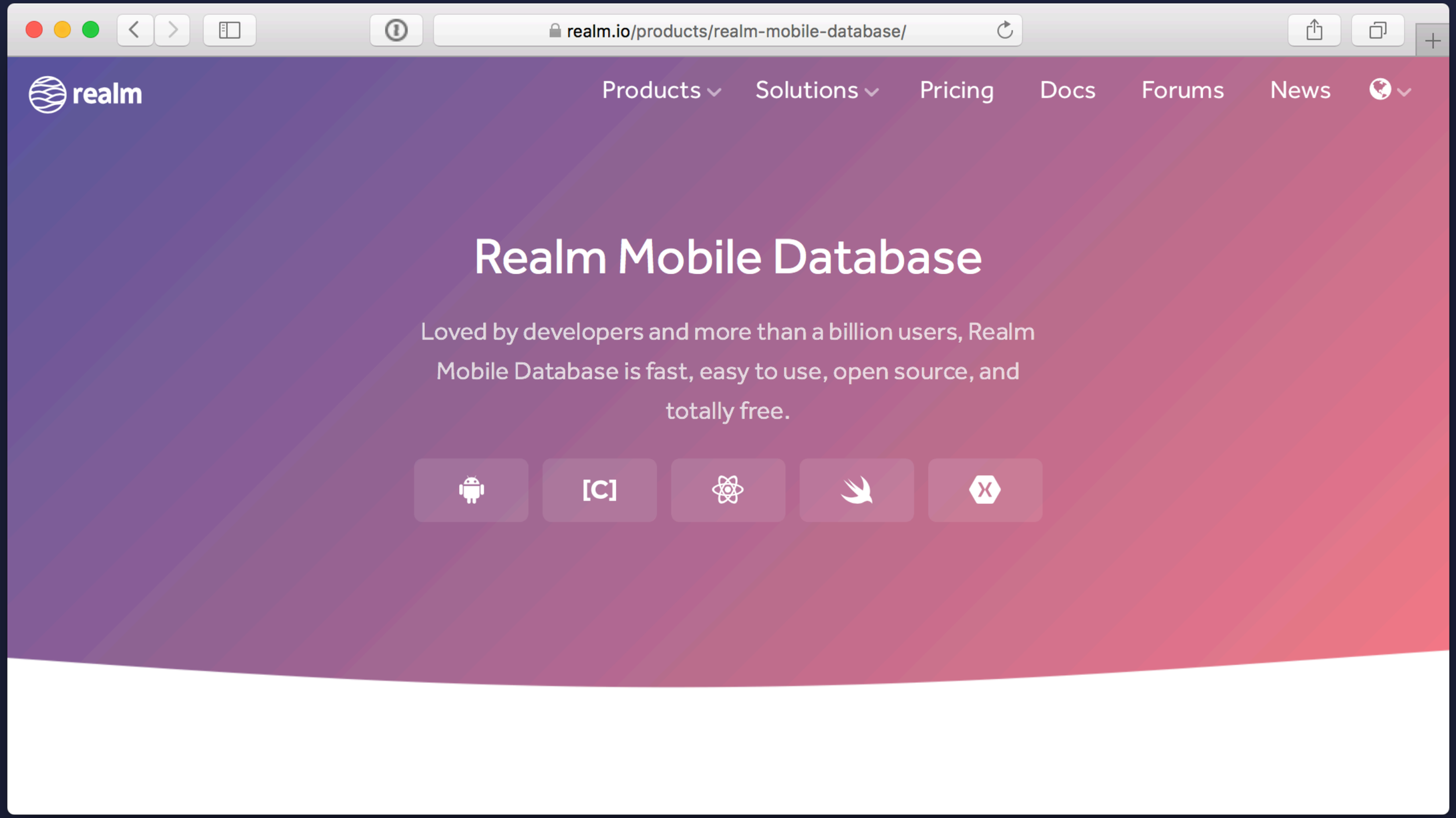
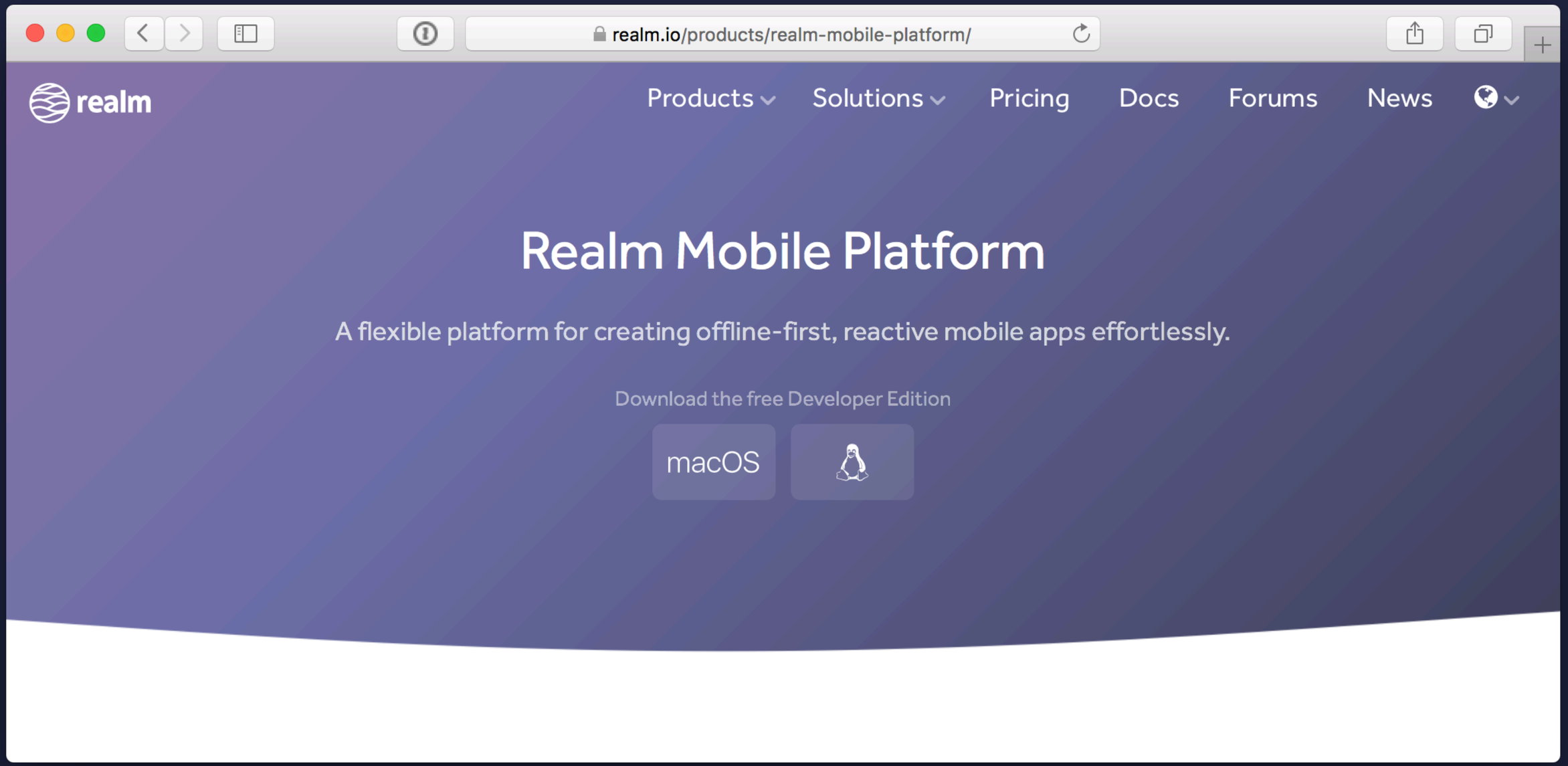
# JAVASCRIPT

## for Swift Developers


realm.io/products/realm-mobile-database/
realm
Products
Solutions
Pricing
Docs
Forums
News
Realm Mobile Database
Loved by developers and more than a billion users, Realm Mobile Database is fast, easy to use, open source, and totally free.
[C]


realm.io/products/realm-mobile-platform/
realm
Products
Solutions
Pricing
Docs
Forums
News
Realm Mobile Platform
A flexible platform for creating offline-first, reactive mobile apps effortlessly.
Download the free Developer Edition
macOS

# SWIFT

```
let dog = Dog(age: 3, furColor: "brown")
print(dog) // => "Dog(age: 3)"
```

# JAVASCRIPT

```
let dog = new Dog(3, "brown");
console.log(dog); // => "Dog(age: 3)"
```


WOW
SOO EASY
PRETTY SWIFT
CALL ME
JS DEVELOPER
imgflip.com

# SUPERFICIAL

- Both have variable declarations with `let`.
  - Swift has parameter names.
- Different calls to make console prints.
  - JavaScript has a new operator.
  - JavaScript has semicolons.

# VARIABLE DECLARATIONS

- `var`
- `let`
- `const`

# `var`

- In Swift: declares a mutable variable
- In JavaScript: declares a variable which is **hoisted** within the function or global scope

# `let`

- In Swift: declares an immutable variable, enforced beyond re-assignments for value types
- In JavaScript: declares a variable which is block-scoped

# const

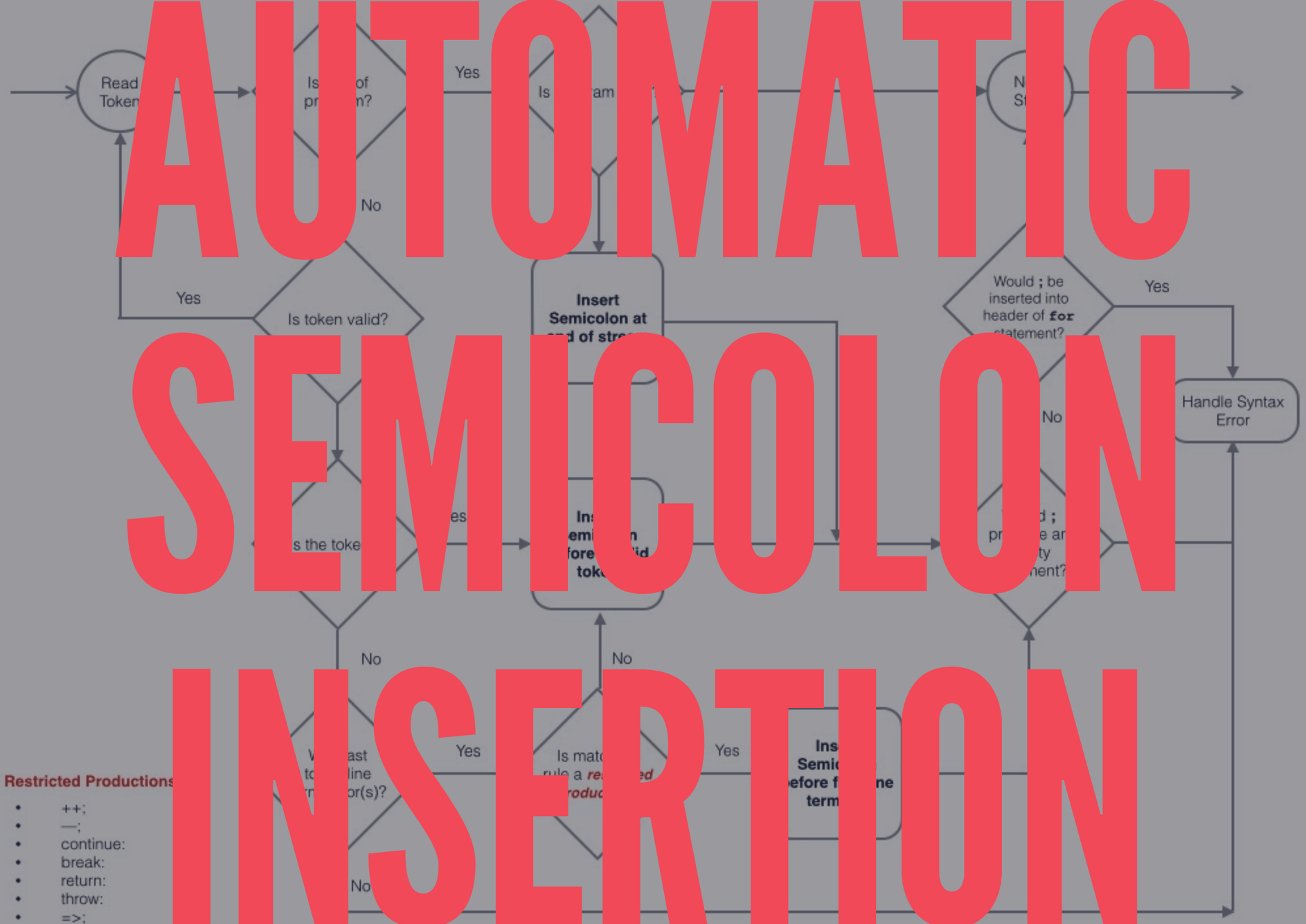
- Only in JavaScript: declares a variable which is block-scoped and not re-assignable

# WRITE MULTIPLE STATEMENTS IN A SINGLE LINE

# IN SWIFT IT'S OPTIONAL.

IN JAVASCRIPT IT'S *SOMETIMES* OPTIONAL.

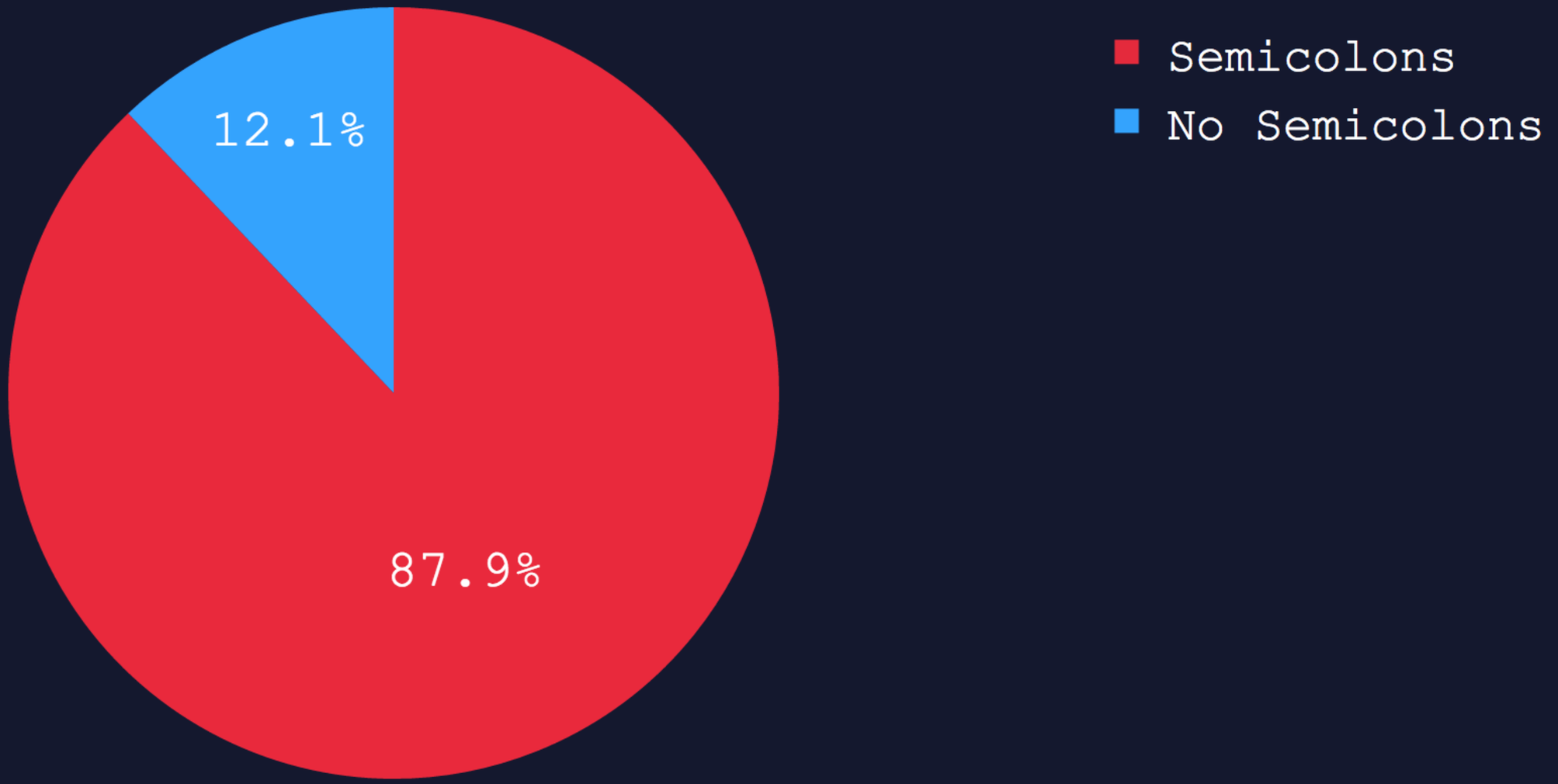
# AUTOMATIC SEMICOLON INSERTION



```
return
{
    name: "This is fine."
};
```

```
return;
{
    name: "This is fine."
};
```


THIS IS FINE.


12.1%
87.9%
Semicolons
No Semicolons

ESLint

```
semi: ["error", "always"] // or: ["error", "never"]
```

# BUT THERE IS MORE THAN THE SUPERFICIAL...

# STEP BACK

## *What do we compare?*

# SWIFT

- 2014: 1.0
- ...
- 2017-03-27: 3.1

# JAVASCRIPT

- 1996: 1.0
- 2000: 1.5 - ECMA 3rd edition
- 2010: ECMA 5th edition
- 2015: ECMA 6th edition - ES6 / ES2015

# ENGINES

- JavaScriptCore
- V8
- SpiderMonkey, Chakra, Carakan, ...

# BABEL

SWIFT & JAVASCRIPT *SUPPORT* DIFFERENT PROGRAMMING PARADIGMS.

- Imperative Programming
- Object-oriented Programming
- Declarative Programming
- Functional Programming
- Many things in between ...

# Let's talk about

# TYPES

SWIFT HAS A STRONG TYPE SYSTEM.

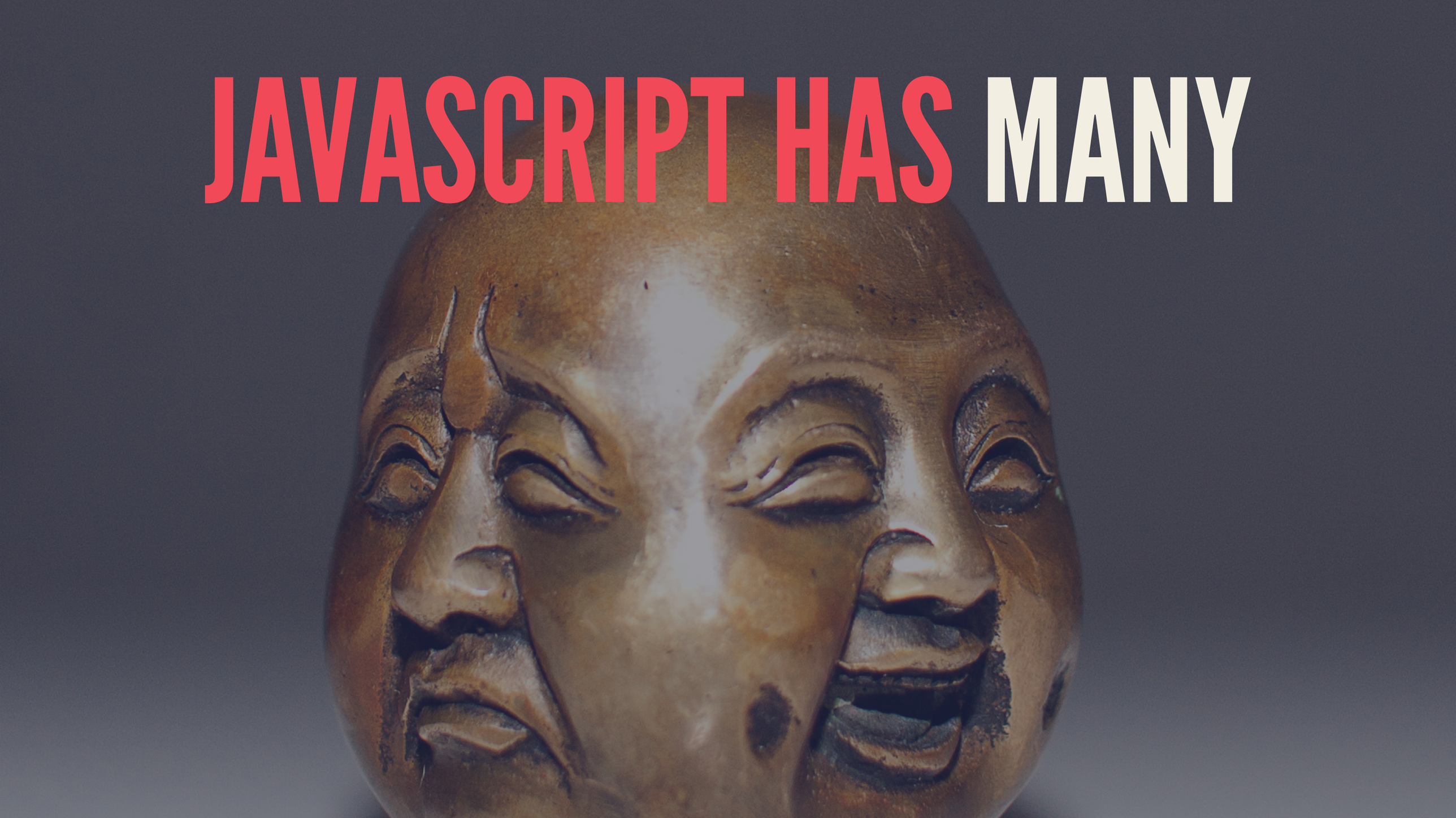
JAVASCRIPT HAS A DYNAMIC TYPE SYSTEM.

# INHERITANCE

# CLASSES IN SWIFT

```
class Animal : CustomStringConvertible {
  var age: Int

  init(age: Int) {
    self.age = age
  }

  toString() {
    let className = String(describing: type(of: self))
    return "\(className)(age: \(age))"
  }
}

class Dog : Animal {
  var furColor: String

  init(age: Int, furColor: String) {
    super(age: age)
    self.furColor = furColor;
  }
}
```

# JAVASCRIPT HAS MANY



# JAVASCRIPT'S PROTOTYPAL INHERITANCE

```
function Animal(age) {
  this._age = age;
}

Object.defineProperty(Animal.prototype, "age", {
  get: function() {
    return this._age;
  },
});

Animal.prototype.toString = function() {
  return this.constructor.name + "(age: " + this.age + ")";
}

function Dog(age, furColor) {
  Object.getPrototypeOf(Dog).call(this, age);
  this.furColor = furColor;
}

Dog.prototype = Object.create(Animal.prototype, {
  constructor: { value: Dog }
});
```

# ES6 SYNTAX

```
class Animal {
  constructor(age) {
    this._age = age;
  }

  get age() {
    return this._age;
  }

  toString() {
    return `${this.constructor.name}(age: ${this.age})`;
  }
}

class Dog extends Animal {
  constructor(age, furColor) {
    super(age);
    this.furColor = furColor;
  }
}
```


TypeScript

# TYPESCRIPT

```
class Animal {
  age: number;

  constructor(age: number) {
    this.age = age;
  }

  toString() {
    return `${this.constructor.name}(age: ${this.age})`;
  }
}

class Dog extends Animal {
  constructor(age: number, furColor: string) {
    super(age);
    this.furColor = furColor;
  }
}
```

# WHY THE UGLY?

## UNDER THE HOOD IT'S STILL THE SAME!*

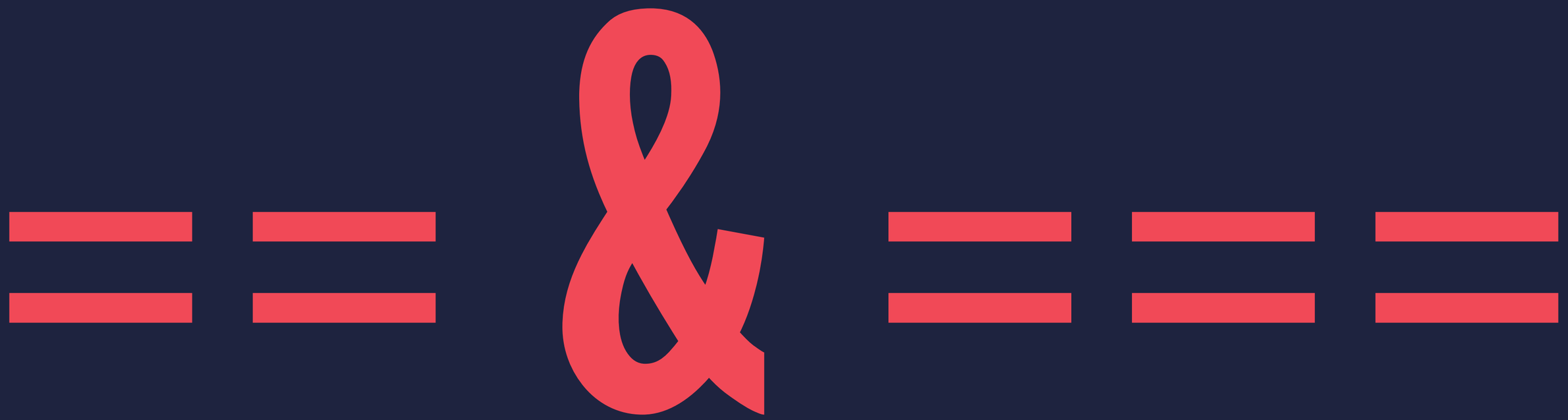
# EQUALITY?

# SWIFT HAS ==.

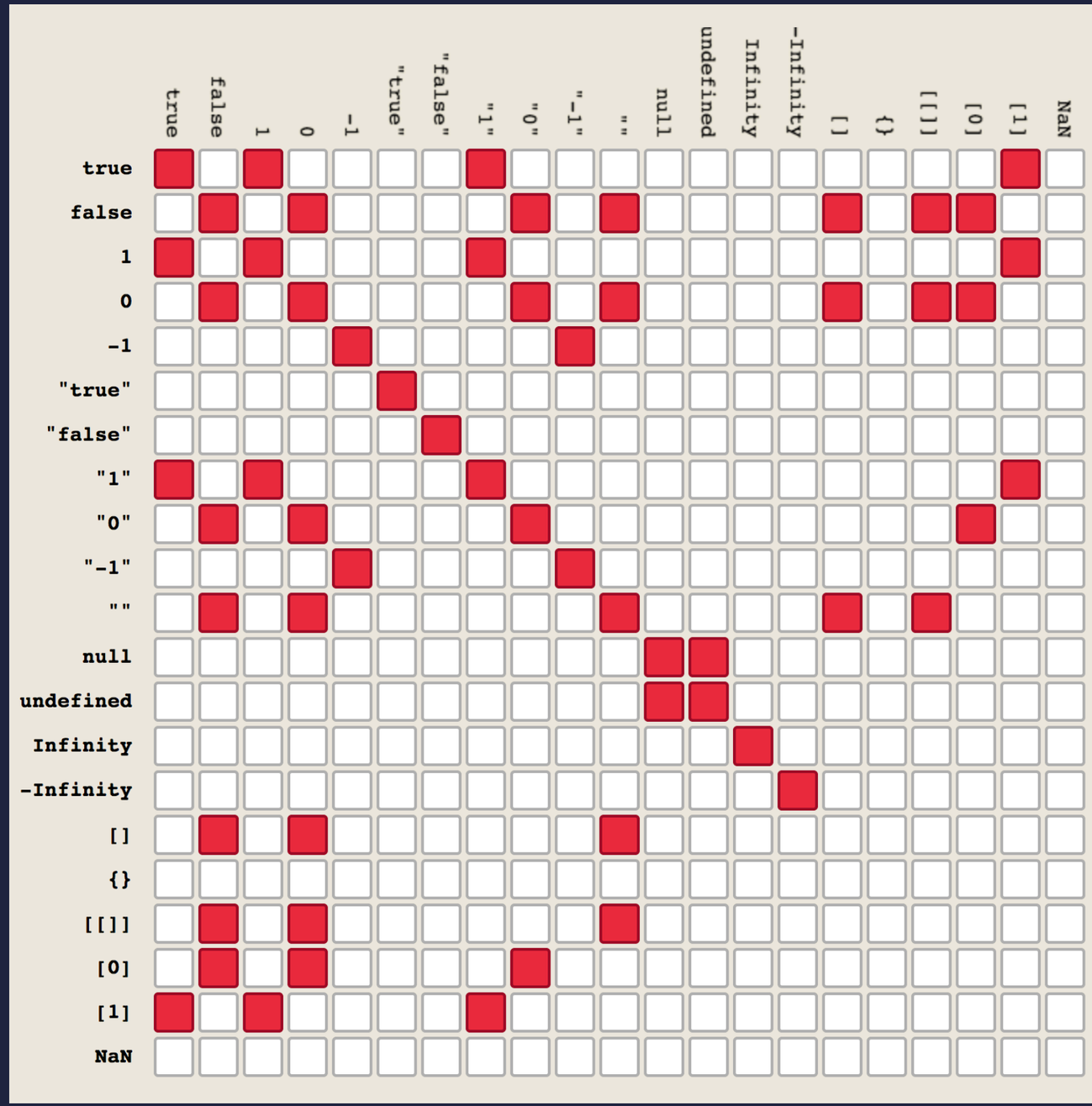
SEMANTICS ARE
ENCODED IN THE STANDARD LIBRARY
AND *EXTENSIBLE*

```
extension Animal : Equatable {}

public function ==(lhs: Animal, rhs: Animal) {
  return lhs.age == rhs.age;
}
```

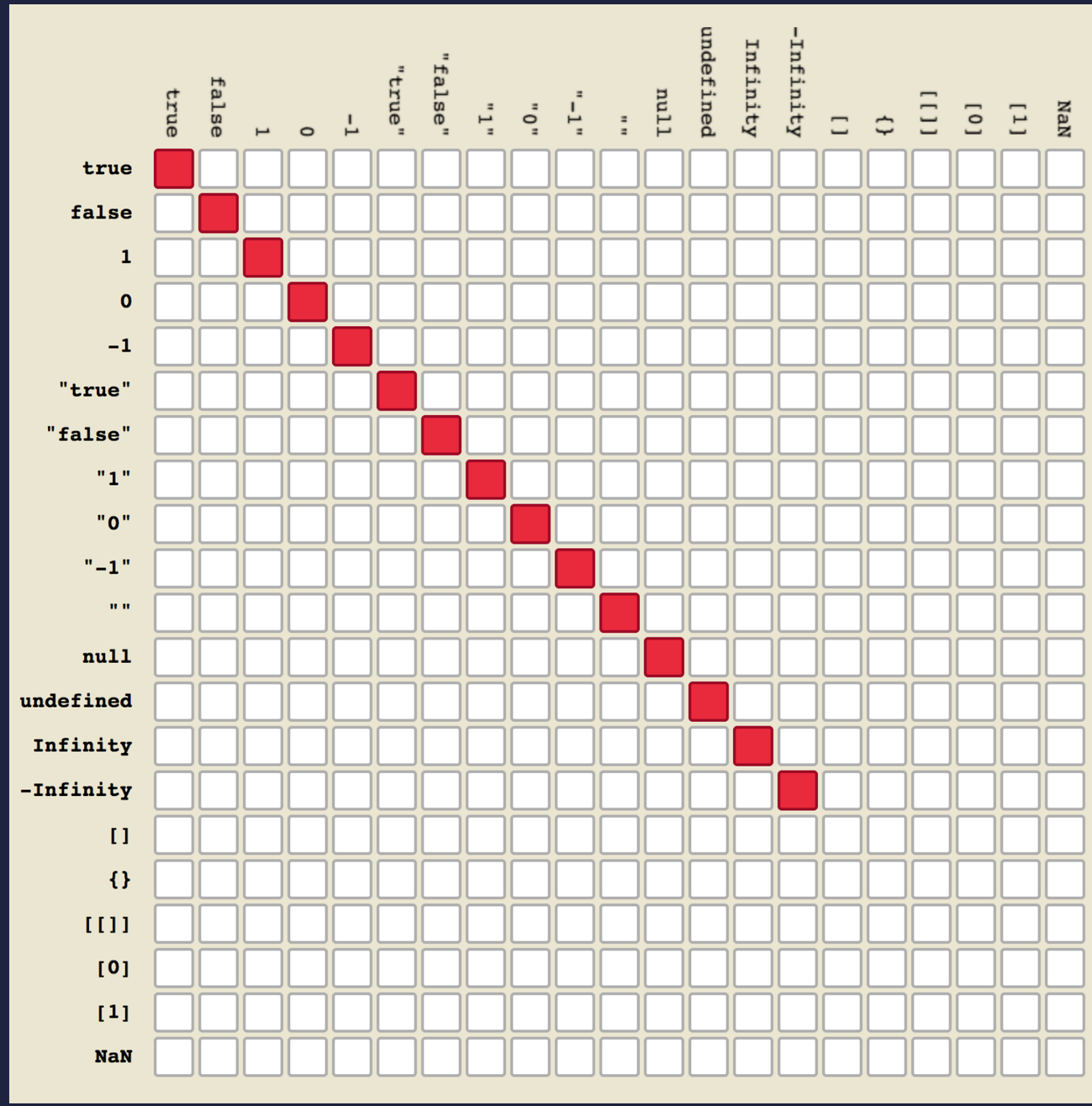
# JAVASCRIPT HAS MORE:



==


true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
NaN

# ===


true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
NaN

# ROLL YOUR OWN

```
class Animal {
  …

  isEqual(other) {
    return this.age === other.age;
  }
}
```

Let's talk about

# `null`

Swift has an explicit concept of nullability encoded in the type system.

JavaScript hasn't.

IN ADDITION TO `null`,

EVERYTHING CAN BE `undefined`.

# null

- `typeof null => "object"`
  - Literal

# `undefined`

- `typeof undefined => "undefined"`
  - Property of the global object
  - Can be overwritten 😭


# TYPESCRIPT
# TO THE RESCUE!

```
let u: undefined = undefined;
let n: null = null;
```

```
// In Swift
func greet(name: string) { … }
func greet(name: string?) { … }
```

```
// In TypeScript
// when compiled with --strictNullChecks
function greet(name: string) { … }
function greet(name: string | undefined = undefined) { … }
```


ECO SYSTEM

# SWIFT RUNS ON ...

- Mac and iOS devices
  - Linux
  - (Android)
  - (Windows)

# JAVASCRIPT RUNS ...

*everywhere*

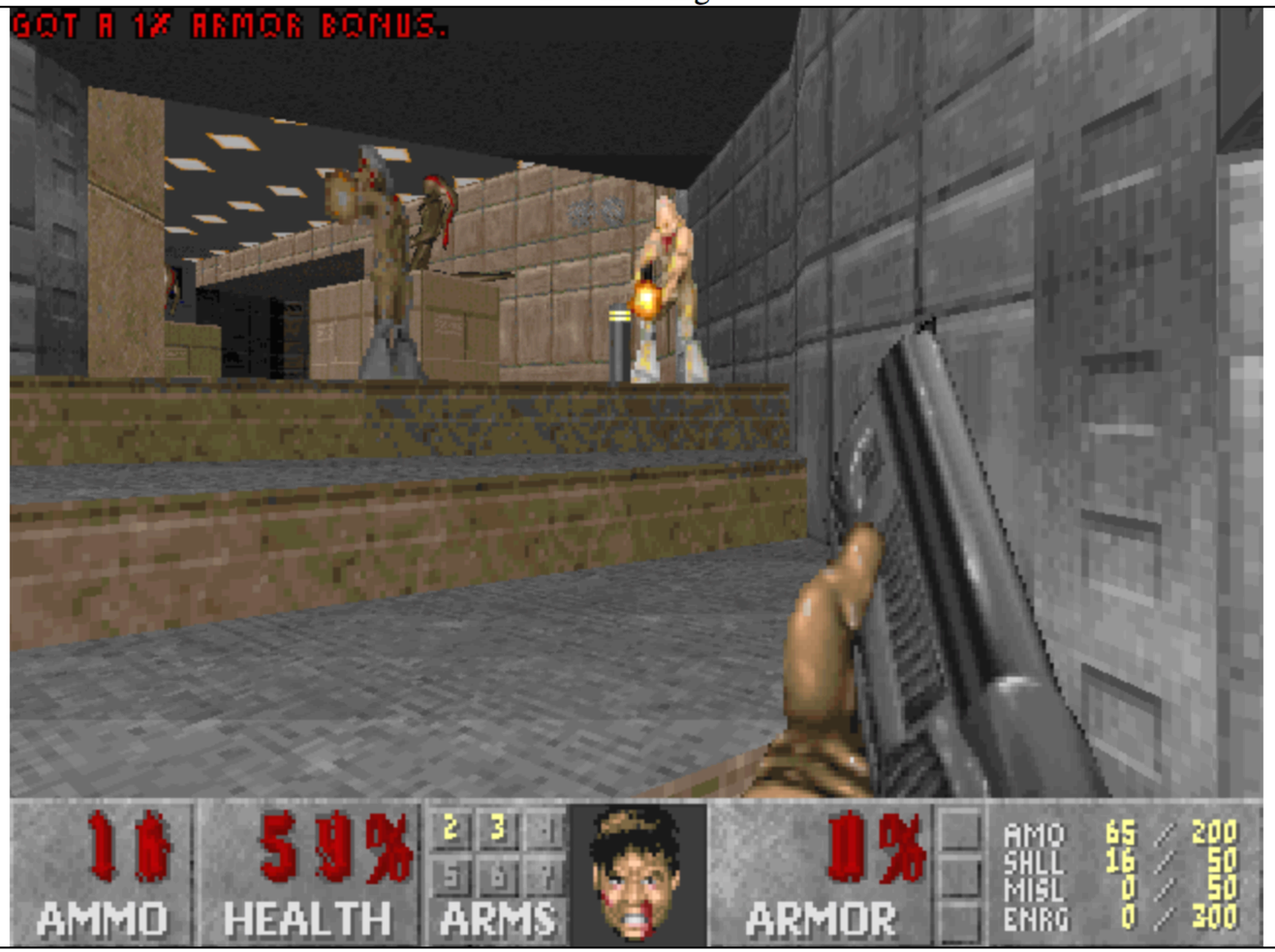
JavaScript is an assembly language.

– Erik Meijer

# ASM.JS

kripken.github.io/boon/boon.html

Downloading...

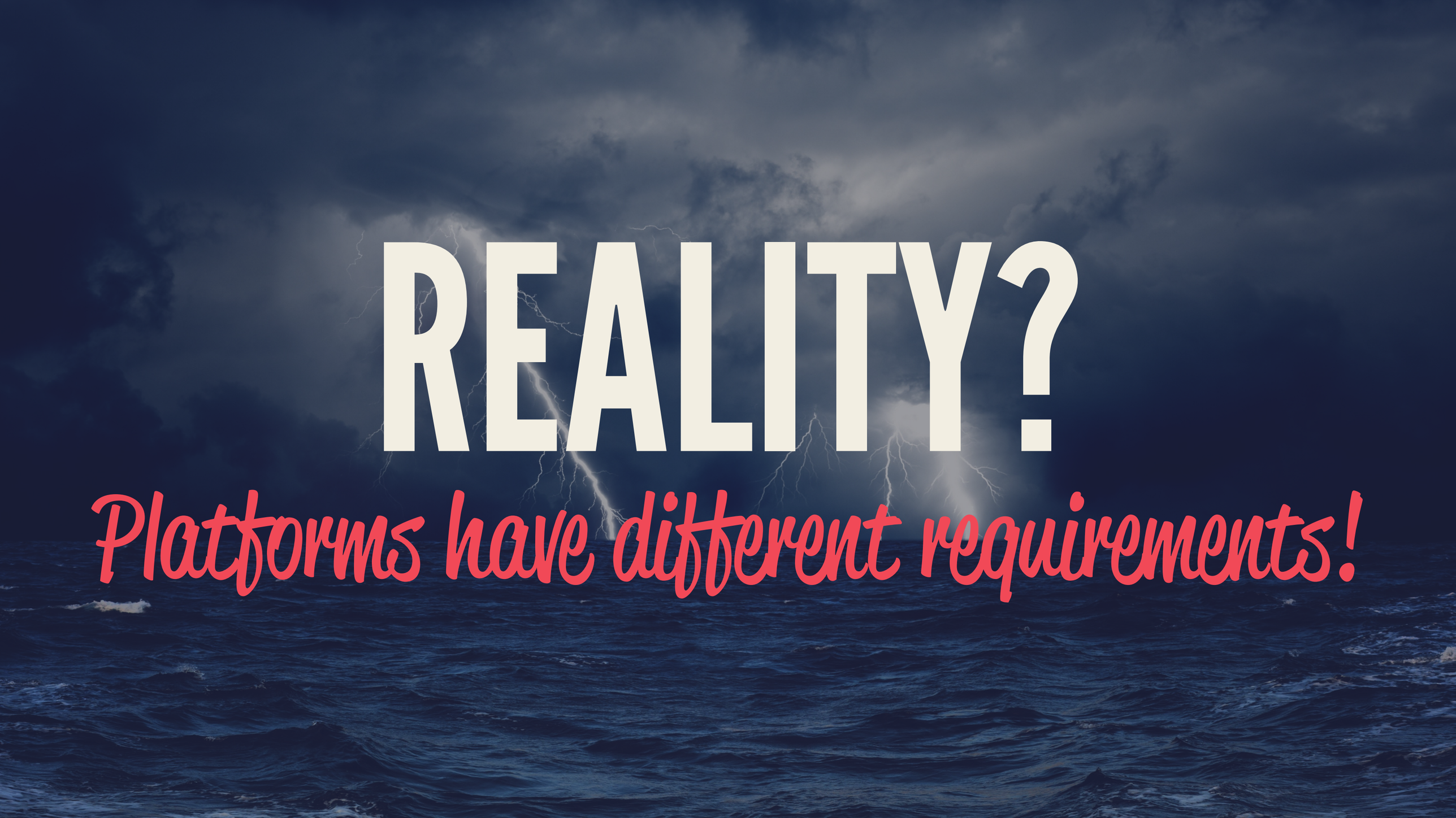



Resize canvas Lock/hide mouse pointer Fullscreen

# Let's talk about

# React Native


The Dream
IMPLEMENT THE APP ONCE &
DEPLOY IT ON ALL PLATFORMS


REALITY?
Platforms have different requirements!

# SHARE THE BUSINESS LOGIC

*But not UI*

# DEPENDENCY MANAGEMENT

*Reminder*

# SWIFT IS INTEROPERABLE WITH OBJECTIVE-C

# OBJECTIVE-C DEVELOPERS HISTORICALLY USED ...

- No dependencies
- Git Submodules
- CocoaPods

# WITH SWIFT, MOST USE:

- CocoaPods
- Carthage
- Swift Package Manager

NO CENTRAL CODE REGISTRY,
YOU RELY ON PRIVATE HOSTED
REPOSITORIES.

# JAVASCRIPT HAS ...

- NPM for Node.js
- Different approaches for frontend code: Bower or Browserify / Webpack etc.

# NPM IS A PACKAGE MANAGER AND A PLATFORM.

YOU SUBMIT ACTUAL CODE.

# NO FULL DEPENDENCY RESOLUTION BY DEFAULT

# PITFALLS OF RECURSIVE RESOLUTION

```
MyApp@1.4.2
├── BananaKit@1.3.2
|    └── monkey@1.1.0
└── monkey@1.0.7
```

```
import BananaKit from 'bananakit';
import Monkey from 'monkey';

const monkey = new Monkey();
const tree = new BananaKit.Tree();
monkey.visit(tree);
// => TypeError: m.climb is not a function
//     at tree.accept (bananakit.js)
//     at monkey.visit (monkey.js)
```

# DIFFERENT APPROACHES FOR LOCKING YOUR DEPENDENCIES 📌

- Commit `node_modules`
- npm shrinkwrap
- Yarn


# THE STATE OF AFFAIRS?

```
questions.forEach((question) => {
    question.ask();
});
```

# THANKS FOR YOUR ATTENTION!

@MRACKWITZ

MR@REALM.IO