Exploring Static Types

Writing safe code that feels like real JavaScript

Josh Duck

Team Lead at ABC

Previously Engineering Manager at Facebook

I work with JavaScript and React

@joshduck

Does JavaScript need to be strongly typed?

We already use types in JavaScript

Guards at the top of functions

Comments for parameter and return types

Hungarian notation (`var $element`)

Object oriented code

Would you refactor this code?

```
function sum(items) {
  let total = null;
  for (let item of items) {
    total = (total === null) ? item : total + item;
  }
  return total;
}

sum([1, 2, 4]); // 7
sum(["a", "b", "c"]); // "abc"
sum("abc"); // "abc"
sum((function* n() { yield 1; yield 2; })()); // 3
```

There’s a difference between code that solves one case, and code that is robust against all cases.

Sarah Mei
@sarahmei

Following

A type system is, first and foremost, a communication mechanism.

2:47 PM - 20 Aug 2017

21 Retweets **95** Likes

1 21 95

Flow and TypeScript

A (very) quick look at what they do

JavaScript-like language from Microsoft and the creator of C#

Static type system for JavaScript from Facebook

Type annotations

```
const pizza: string = "Margherita";
const price: number = 10;
const ingredients: Array<string> = [
  "Basil",
  "Tomato",
  "Mozzarella"
];
```

Type inference

```
const price = 10;
const total = 10 * 5;
```

```
const total: number
```

[Flow]

Great tooling

```
const pizza = "Margherita";
pizza.
```

- charAt (method) String.charAt(pos: number): stri…
- charCodeAt
- concat
- indexOf
- lastIndexOf
- length
- localeCompare
- match
- replace
- search
- slice
- split

```
josh@DESKTOP-PSKEN95:~/Node/TypedExamples$ flow
Error: src/flow/01-basic-types/how-flow-works.js:10
 10:   return price * count;
               ^^^^^ string. The operand of an arithmetic operation must be a number.

Error: src/flow/06-disjoint-unions/ambiguous-union.js:11
 11:     console.log("Payment failed: " + payment.errorCode);
                                          ^^^^^^^^^^^^^^^^^^ undefined. This type cannot
 be added to
 11:     console.log("Payment failed: " + payment.errorCode);
                     ^^^^^^^^^^^^^^^^^^ string


Found 2 errors
```

```
josh@DESKTOP-PSKEN95:~/Node/TypedExamples$ tsc
src/typescript/02-types-before-coding.ts(3,15): error TS2362: The left-hand side of an
arithmetic operation must be of type 'any', 'number' or an enum type.
josh@DESKTOP-PSKEN95:~/Node/TypedExamples$
```

But there are some differences…

What language is it *really*?

TypeScript is a **strict syntactic superset** of JavaScript; it adds new features.

Flow is just JavaScript under the hood

Correctness vs. ease of use

Unsound ⟷ Sound

Allows some type errors

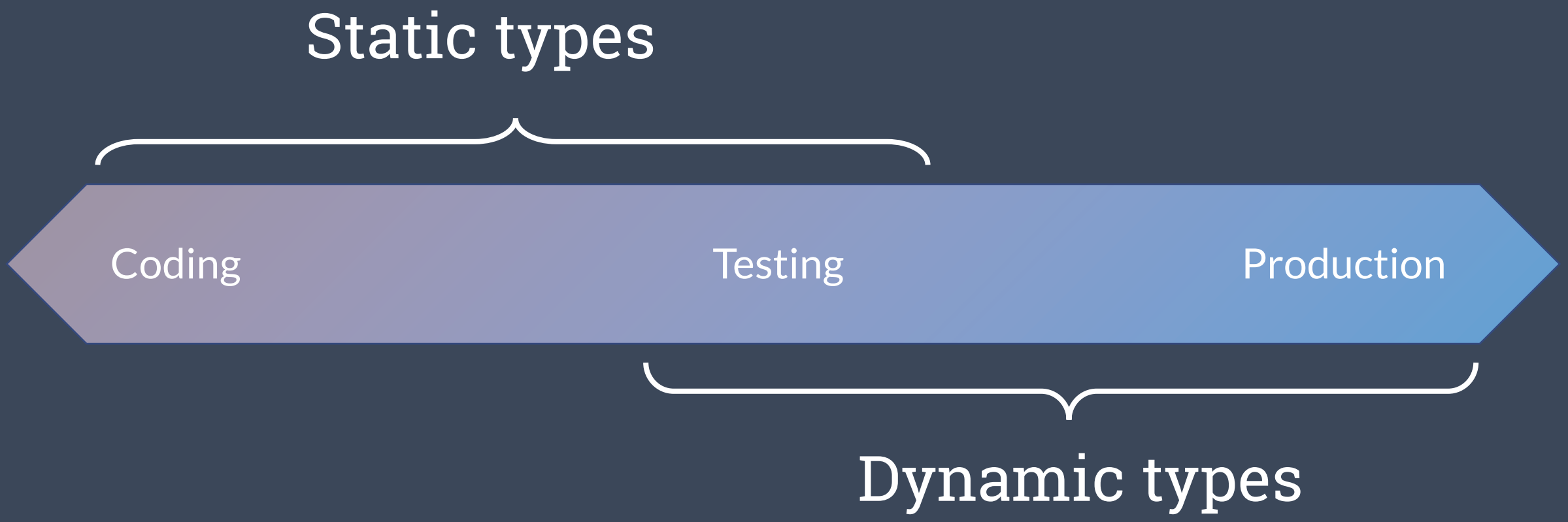
Prevents all type errors

Don’t overthink it:
They are surprisingly similar

How type checking works

No, it’s not magic

When are bugs found?



Discovering a type error

```
const pizza: string = "Chicken";
const price = 10;
const count = 1;

const total = calculateTotal(pizza, count);

function calculateTotal(price, count) {
  return price * count;
}
```

Discovering a type error

```
const pizza: string = "Chicken";
const price = 10;
const count = 1;

const total = calculateTotal(pizza, count);

function calculateTotal(price, count) {
  return price * count;
}
```

string ➡ PizzaType

number ➡ PriceType

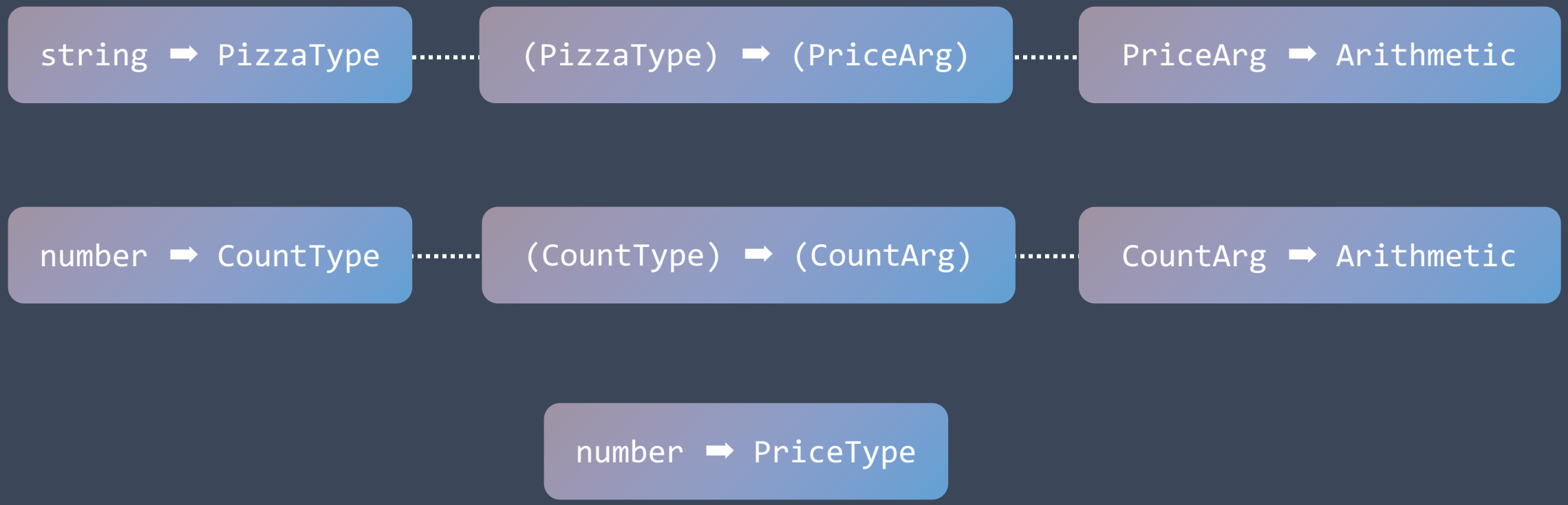
number ➡ CountType

(PizzaType, CountType) ➡ Call(PriceArg, CountArg)

PriceArg ➡ Arithmetic

CountArg ➡ Arithmetic

Collapsing down



Inconsistencies are errors

`string ➡ Arithmetic`

`number ➡ Arithmetic`

`number ➡ PriceType`

```typescript
const pizza: string = "Chicken";
const price = 10;
const count = 1;

const total = calculateTotal(pizza, count);

function calculateTotal(price, count) {
  return price * count;
}
```

The operand of an arithmetic operation must be a number.

Value = Positives – Negatives

Prevents bugs

Replaces JS code

False errors

Extra verbosity

Configuring tooling

Understanding how your type system works allows you to maximize its value

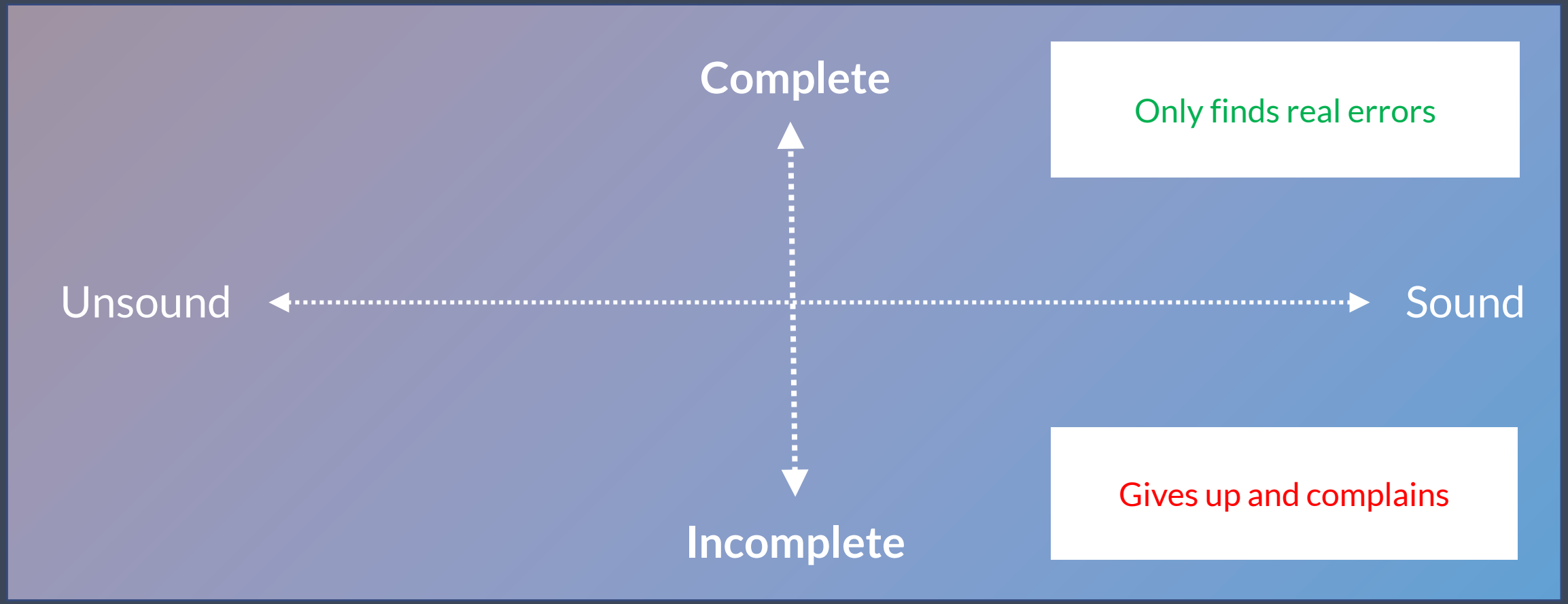
The implications of being sound…

Unsound ⟷ Sound

Allows some type errors

Prevents all type errors

Type systems are limited by their comprehension



Embracing typed code

Don’t fight your type checker

Primitive values

Strings, numbers, and booleans

Literal values

```
function ratePizza(stars: 5) {
  console.log("Tell your friends!");
}

ratePizza(5);
ratePizza(1);
```

```
Expected number literal `5`, got
`1` instead.
```

Literal unions

```
function ratePizza(stars: 1 | 2 | 3 | 4 | 5) { /* ... */ }

ratePizza(5);
ratePizza(1);
ratePizza(100);
ratePizza(-1);

function setSize(size: "small" | "medium" | "large") { /* ... */ }

setSize("small");
setSize("huge");
```

Opaque types

Combining runtime validation with types

Checking complex strings

```
export function sendEmail(address: string) {
  /* ... */
}

const input = 'I am not an email address!'; // Uh oh
sendEmail(input);
```

Opaque types

```
export opaque type Email: string = string;

export default function createEmail(input: string): Email {
  if (EMAIL_REGEX.test(input)) {
    return input;
  } else {
    throw new Error('Invalid email!');
  }
}
```

Opaque types

```
import createEmail from './define-opaque-types';
import type { Email } from './define-opaque-types';

function sendEmail(address: Email) { /* ... */ }

const email: Email = createEmail('hello@example.com');
sendEmail(email);

const unchecked: Email = 'unchecked@example.com';
sendEmail('unchecked@example.com');
```

Structural types

Using object literals as typed records

Methods of typing objects

Nominal typing

"This object is an instance of the Pizza class"

Structural typing

"This object has the properties of the Pizza type"

Structural types treat objects as records

```
type Pizza = {
  name: string,
  size: 'small' | 'me' | 'large'
};

type Order =
  item: Ar a>,
  total: n
  address?: {
    street: string,
    postcode: string
  }
}
```

Nullable types

Fixing a 50-year-old mistake

```
▶Uncaught TypeError: Cannot read property 'addCheese' of null
    at VM118 pen.js:4
▶Uncaught TypeError: order.addPizza is not a function
    at VM160 pen.js:5
```

```
order.pizzas &&
  order.pizzas[0] &&
  order.pizzas[0].addCheese &&
  order.pizzas[0].addCheese();
```

“I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a **billion dollars of pain** and damage in the last forty years.”

– Tony Hoare, inventor of ALGOL W.

```typescript
function setName(name: string) { /* ... */ }

setName('Chicken');
setName(null);
setName(undefined);
```

Nullable types are unions too!

```
type Maybe1 = ?string;

type Maybe2 = string | null | void;
```

Object unions

Simplifying logic for state transitions

```typescript
type PaymentStatus = {
  status: "pending" | "failed" | "complete",
  errorCode?: number,
  successMessage?: string
};

function renderPayment(payment: PaymentStatus) {
  if (payment.status === "failed") {
    console.log("Payment failed: " + payment.errorCode);
  }
}
```

errorCode: number | void

Disjoint Unions
(aka Tagged Unions aka Algebraic data types)

```typescript
type PaymentStatus =
  { status: "pending" } |
  { status: "failed", errorCode: number } |
  { status: "completed", successMessage: string };

function renderPayment(payment: PaymentStatus) {
  if (payment.status === "failed") {
    console.log("Payment failed: " + payment.errorCode);
  }
}
```

Type Driven Design

Designing algorithms around types.

Types are key frames in your code

Key frame

Key frame

```
Shape<Blue, Star>;
```

```
Shape<Pink, Pentagon>;
```

Type Driven Design leads to simpler code

What we've covered...

How JavaScript uses types already

A brief introduction to TypeScript and Flow

Why understanding static types is important

Coding for the type system

Documentation **typescriptlang.org** and **flow.org**

Slides and code **github.com/joshduck/exploring-static-types**

Twitter **@joshduck**

Thanks!