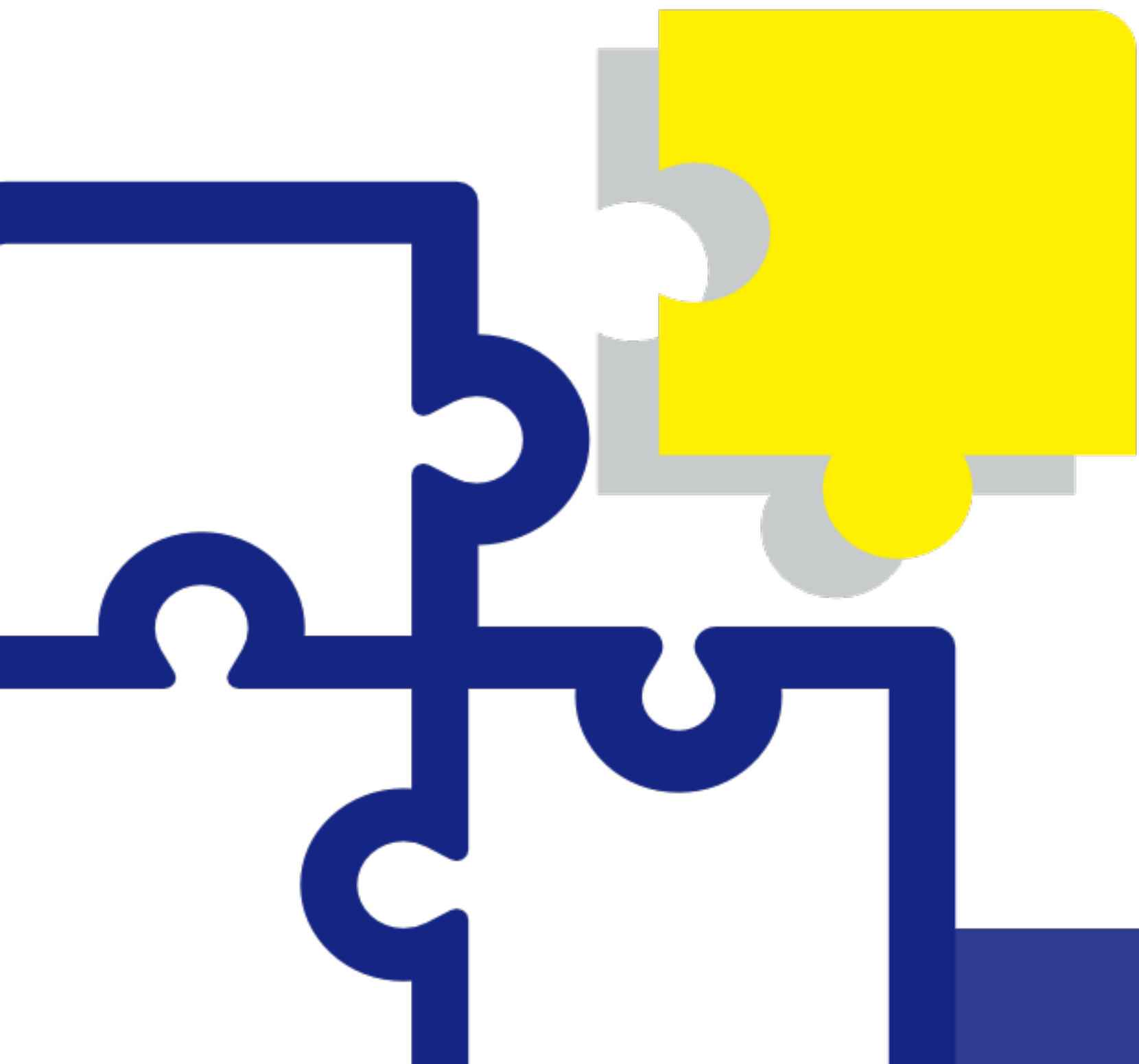


Type Script

勉強会

Type Scriptの
基礎を学ぼう。



はじめに

この勉強会のターゲット

JavaScriptわかるけど、TypeScriptはよくわからん

TypeScript聞いたことある

TypeScript興味ある

初学者向け

アジェンダ

1. TypeScriptってなに？

2. 「型」について理解する

3. 関数で「型」を使う

4. TypeとInterface

深い内容の説明は
省略します。

1.TypeScriptってなに？



TypeScriptとは

TypeScriptとはJavaScriptを静的型付けにした
プログラミング言語であり、
JavaScriptのスーパーセットである。



1.TypeScriptってなに？



スーパーセット

スーパーセットとは

元の言語との互換性を保ちつつ、
元の言語を拡張して作った言語のこと。

JavaScriptの機能はそのままに、
型システムによってさらに強力な言語としてアップデート

スーパーセット

JavaScriptにできることはすべて
TypeScriptにもできるということ



1.TypeScriptってなに？



TypeScriptのどこが良いのか？

①エラーの早期発見(型安全)




②開発体験の向上

メリット①エラーの早期発見(型安全)

コンパイラによる型チェックによって事前に予期しないプログラムやエラーを検知することができる。

```
TS main.ts > ...
1  ✓ function add(a: number, b: number): number {
2    |   return a + b;
3    | }
4
5    // 間違った使用 (実行時エラー)
6    console.log(add(5, 10));
7    ⚡
8
```

型 'string' の引数を型 'number' のパラメーターに割り当てることはできません。 ts(2345)

[Comment Translate]   |  Google

10 => 10

問題の表示 (⌘F8) 利用できるクイックフィックスはありません

コンパイルエラーのダイアログ

1.TypeScriptってなに？



メリット②開発体験の向上

型情報がソースコードに記載されるため、コードを読解する際の手助けとなる。
型情報をもとにIDEが入力を保管してくれる機能もサポートしている。

```
10
11 // ②開発体験の向上
12 ✓ interface User {
13   · name: string;
14   · age?: number; // オプショナルプロパティ
15 }
16
17 ✓ function greetUser(user: User): string {
18   · return `Hello, ${user.name}!`;
19 }
20
21 // 関数の利用
22 console.log(greetUser({ firstName: "Charlie" }));
23 ✓ // コンパイルエラー: 型 '{ firstName: string; }' を型 'User' に割り当てることはできません。
24
```

1.TypeScriptってなに？



JavaScriptとTypeScriptとの違いは
「型」があるかないか

▶ JavaScriptの基本的な知識があれば、
TypeScriptの学習コストは高くない！

2.「型」について理解する

基本的な型システム

基本的な構文

```
let age: number = 25;
```

型注釈

変数宣言： 通常のJavaScriptと同じでlet,constを使用する。
基本的にconst使用でよい。

型注釈： 型を明示的に指定する方法。
複雑なケースやドキュメントとしての役割を持つ。

2. 「型」について理解する

基本的な型システム

プリミティブ型（代表例）

```
let age: number = 25;  
let name: string = "Alice";  
let isStudent: boolean = true;  
let user: string | null = null;  
let score: number | undefined;
```

- ① **string** 文字列を表す型。
- ② **number** 数値(整数、浮動小数点数)を表す型。
- ③ **boolean** 真偽値(true,false)を表す型
- ④ **null/undefined** 値がないことを表す型

2. 「型」について理解する



配列とタプル

```
let numbers: number[] = [1, 2, 3];  
let tuple: [string, number] = ["Alice", 25];
```

配列

同じ型のデータの集まりを扱う。
型名に [] を付けて定義する。

タプル

異なる型のデータを固定の順序で扱う。
要素ごとに型を指定して定義する。

anyとunknownとnever

any

どんな値でも代入可能な型。
しかし型安全性が失われてしまう。
使用例) JavaScriptコードをTypeScriptに移行する

unknown

anyと同じくどんな値でも代入可能な型。
型を明示的にチェックしないと値の操作はできない。
使用例) 柔軟性と型安全性を両立したい場合

never

値を持たない型を意味する。
never型に値を入れることはできない。
never型を値として入れることはできる。

基本的な型システム

オブジェクト型、リテラル型

オブジェクト型

対象の値がオブジェクトであることを表す型。

```
let userInfo: object = {  
  name: "Alice",  
  age: 25,  
  isActive: true  
};
```

リテラル型

通常の型と違ってより具体的な値を受け入れ、その値のみを許容する

```
let greeting: "hello" = "hello";
```

2. 「型」について理解する

基本的な型システム

ユニオン(合併)型、交差(インターセクション)型

Union型

複数の型のいずれかを受け入れることができる型。
"または"を意味する。

string | number

Intersection型

&&

複数の型を組み合わせて、
すべての型の条件を満たすプロパティやメソッド
を持つ型を定義する。
"かつ"を意味する。

2. 「型」について理解する

TypeScriptのポイント

型の定義は具体的であるほど良い。

- ▶ anyのような抽象的な型はできるだけ使用を避けるべきである



3.関数で型を使う

関数の定義

functionキーワードによる関数定義

関数宣言は名前付きで定義され、コードのどこからでも呼び出せます。

```
function 関数名(引数: 型, 引数: 型): 返り値の型 {  
    return;  
}
```

関数式による関数定義

無名関数で、変数に代入して使用します。また、アロー関数はよりシンプルな記法で記述することができます。

```
// 関数式  
const 関数名 = function(引数: 型, 引数: 型): 返り値の型 {  
    return;  
}  
  
// アロー関数  
const 関数名 = (引数: 型, 引数: 型): 返り値の型 {  
    return;  
}
```

2. 「型」について理解する

オプション引数とデフォルト引数

オプション引数

型注釈に?を付けることで
引数の指定を省略可能にする。

```
// オプション引数
✓ function logMessage(message: string, user?: string): void {
  console.log(user ? `${user}: ${message}` : message);
}
```

デフォルト引数

引数を与えられない場合に
初期値を設定してエラーを防ぐ。

```
// デフォルト引数
✓ function calculatePrice(price: number, discount: number = 0.1): number {
  return price * (1 - discount);
}
```

2. 「型」について理解する

4.Type と Interface

Typeについて

型エイリアスという概念

ある型に対して別名をつける機能。

簡単な名前でデータ参照できるようになります。

TypeキーワードはTSでそれを実現するための機能です。

Type

型エイリアスを実現するための記法で特別な型を作成することができます。

```
type 型 = {  
  name: string;  
  age: number;  
};
```

interfaceについて

interface

オブジェクトやクラスの構造を定義するためのものです。オブジェクトのプロパティやメソッドの型を指定するためによく使われます。

```
interface 型 {  
    name: string;  
    age: number;  
    greet(): void;  
}
```

Type と interfaceの違い

1. 記法

type には宣言の後にイコール(=)が必要でinterfaceには必要ない。

2. 定義の形式

型定義をするときにinterfaceはオブジェクト形式のみでしか定義することができない。

3. 拡張の方法

typeは型の上書き不可。新たな型を宣言して"&"で型の継承を行う。
interfaceは型の上書きが可能。新たな型を定義して"extends"で継承することもできる。

Type と interface

細かい部分の使用は若干違うが、
できることはほとんど同じで、
どちらか一方でしか実現できないことはない

使い分けの基準(あくまで一例)

type :

ユニオン型や交差型のような複雑な型を定義したい場合

interface :

拡張性が求められる場面や他の開発者と共同作業する場合

まとめ

TypeScriptにはエラーの検知、
再利用性の向上といった大きな恩恵がある

セキュアでスケーラブルな
TypeScriptの旅に出かけよう！

The TypeScript logo, consisting of a blue rounded square with the white letters 'TS' inside.