

# TypeScriptを勉強しよう

# 前提知識

- JavaScript を使ったプログラミング経験
- あれば良い
  - 型についての知識・オブジェクト指向

# はじめに

このスライドはまず型についての簡単な知識と、なぜJavaScriptを使わずにTypeScriptを使うのか学びます。

このスライドを読んでいくうえで、大事な気持ち。

- 型を意識しよう
- エラーと友達になろう
- 人間のチェックは信じられない！

みんな嫌いな

# 座学

# **What difference between JS and TS**

JavaScriptとTypeScriptにはどのような違いがあるか。

JavaScriptは動的型付け言語  
TypeScriptは静的型付け言語



?

動的型付け言語である、JavaScriptやPythonはあまり「型」を明示的に書かずコードを書くことができます。逆に静的型付け言語のTypeScriptやJavaは型を意識し、明示的に書く必要性があります。

具体にそれがどういうことなのかという話の前に、まず「型」が何なのかを勉強していきましょう。

# 1. What's Type

型って何

1. What's Type では型がなにかと、型を意識する重要性について考えます。

型はJavaScriptにかかわらず大抵の言語でも必要となる考え方になります。

型は、変数(定数、引数など含む)に適用できるものであり、型を意識するには抽象的に考えるのが大事です。

型には多くの種類がありますが、  
手始めに、`string` 型と `number` 型を例に進めていきましょう。

例えばこのコード

```
const firstName = "Sora";
const lastName  = "Hattori";

const language  = "JavaScript";
```

例えばこのJavaScriptのコードでは

"Sora" や "Hattori" , "JavaScript" など、  
具体的に何が入ってると考えるより  
どの変数も、 string 型が入ってると抽象的に考えます。

`number` 型の場合も見ていきましょう。

```
const age      = 19;  
const year    = 2024;  
const height  = 185.3;
```

この場合も同じです。

`19` や `2024`, `185.3` が入ってると考えるより、`number` 型が入つ  
てると考えます。

**And...?**

それで...?

型を意識する必要がある理由の一つがわかる例があります。  
それは数値同士の計算です。

```
// 三角形の面積を求める
function triangle(bottom, height) {
  return bottom * height / 2;
}

const area = triangle(21, 5);
```

この `triangle` 関数の引数 `bottom` と `height` が求める型は、どちらも `number` 型になります。

```
// 三角形の面積を求める
function triangle(bottom, height) {
  return bottom * height / 2;
}
const input = prompt("底辺を入力してください");
const area = triangle(input, 5);
```

もし、このようなコードを書いてしまった場合。  
引数 `bottom` には文字列<sup>1</sup>が入ってしまい、 `string * number` の  
計算を行おうとしてエラーが発生<sup>2</sup>します。  
このようなエラーを防ぐには、常に型を意識をしなければな  
りません。

\*1 厳密には違いますが今回は便宜上これで説明します

## 2. In TypeScript

TypeScriptでは

2. In TypeScript では、JSではだめな理由とTypeScriptに書き換えたときのコードを比較していきます。

型を意識するだけでエラーを防げるなら意識すればいいじゃん。と思った方はいませんよね？

コードが1000行、ソースファイルも10個以上のような状況になったときあなたとあなたのチームメイトは、**完璧に型を意識してコードを書くことは出来ますでしょうか？**



そう、出来ません。

けれどそれを実現しなければ大量のバグの元になります。

例えばボタンを押したときに先程の `triangle` 関数のような呼び方を気づかず行ってしまうと、ボタンを押すまでエラーが発生しない事に加え、必ずしもそのボタンを押してデバックするとは限らないため、更に発見が遅れる可能性が高いです。

そのような事故を防ぐため、我々が型を明示的に書き実行する前にTypeScriptに型があるか自動的に確認してもらおう！

というのがTypeScriptです。



型と問題点がわかったところで

**座学終わり**

型とJSの問題点がわかったところで今までのコードを  
TypeScriptに書き換えていきましょう。

次のURLからTypeScriptを試せるページに飛びます。

[TypeScript Playground](#)

このTypeScriptを参考に書き換えて見てください(デ キレバ コピペ  
シ行デ カ行ネ)

[TypeScript Playground](#)

# 型注釈

TypeScript特有の型注釈をつけました。

そうすると `const bottom` のところにエラーが出たはずです。

この解説は[ココ](#)に書いてある通りです。

この解決方法は少し踏み込んだ話をしなければならないので一旦このコードとはおさらばして、別のサンプルコードに移ります。

# 型推論

次のコードに先程のコードを参考に型注釈をつけるところ  
全てについてみてください。

型注釈をつけたコードがこちらになります。

これ型注釈をわざわざ明示的に書かなくても推論できるんじ  
やね？

はい、そうです。

このようにに推論できるところは自動で推論してくれます。

# まとめ

- JavaScriptは型を実行時に決まりので気づかない可能性がある
  - これを動的型付け言語という
- TypeScriptは人が型を明示し、明示された型を確認を TypeScriptがするためしょうもないミスが減る
  - これを静的型付け言語という
- 動的型付け言語が型を自動で決定してくれるの楽じゃん、という考えは概ね後からしんどくなる



表にするとこんな感じ

### 型を明示しないとダメか

JavaScript	○ 実行時に自動で決定
TypeScript	△ 一部手動で明示が必要

### 型チェック

JavaScript	✗ 人がやらないといけない
TypeScript	○ 明示した型で自動でチェック

# 参考

- サバイバルTypeScript