Programmation Web

Tristan Crolard

Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

tristan.crolard@cnam.fr

cedric.cnam.fr/sys/crolard

Programmation Web mobile

- Le **DOM** (*Document Object Model*) impose des contraintes :
  - le langage **JavaScript** pour le développement
  - une programmation **événementielle** (pour l'UI) et **asynchrone** (accès réseau)
  - des **mutations** du DOM pour actualiser l'affichage (sans recharger toute la page)

- Ce style de programmation ne passe pas à l'échelle (trop complexe) :
  - → **TypeScript** : extension **statiquement typée** de JavaScript
  - → des *frameworks* pour faciliter le développement (comme Angular ou React)
  - → des **paradigmes** associés (**fonctionnel** pour React et **réactif** pour Angular)

Les paradigmes de programmation

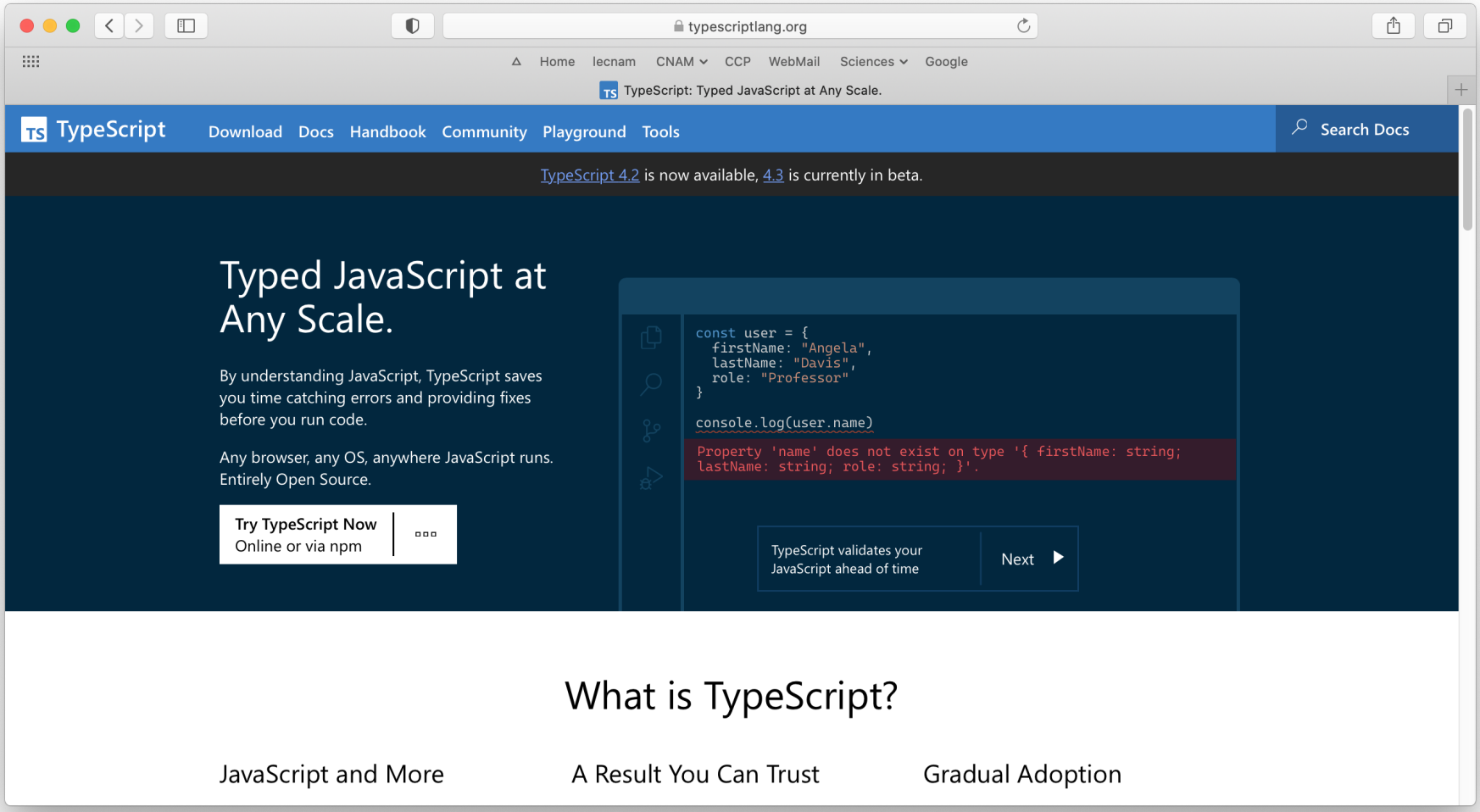
- **Style impératif :** variables (mutables), procédures, instructions (sequence, boucles, ...)
- **Style fonctionnel :** constantes, fonctions et expressions

	Données	
Style	mutables	immutables
impératif	idéal	ok
impératif asynchrone	ko	ok
fonctionnel	/	idéal
fonctionnel asynchrone	/	idéal

$\longrightarrow$ style **impératif :** données **mutables**

$\longrightarrow$ style **fonctionnel :** données **immutables**

TypeScript



https://www.typescriptlang.org

De JavaScript à TypeScript

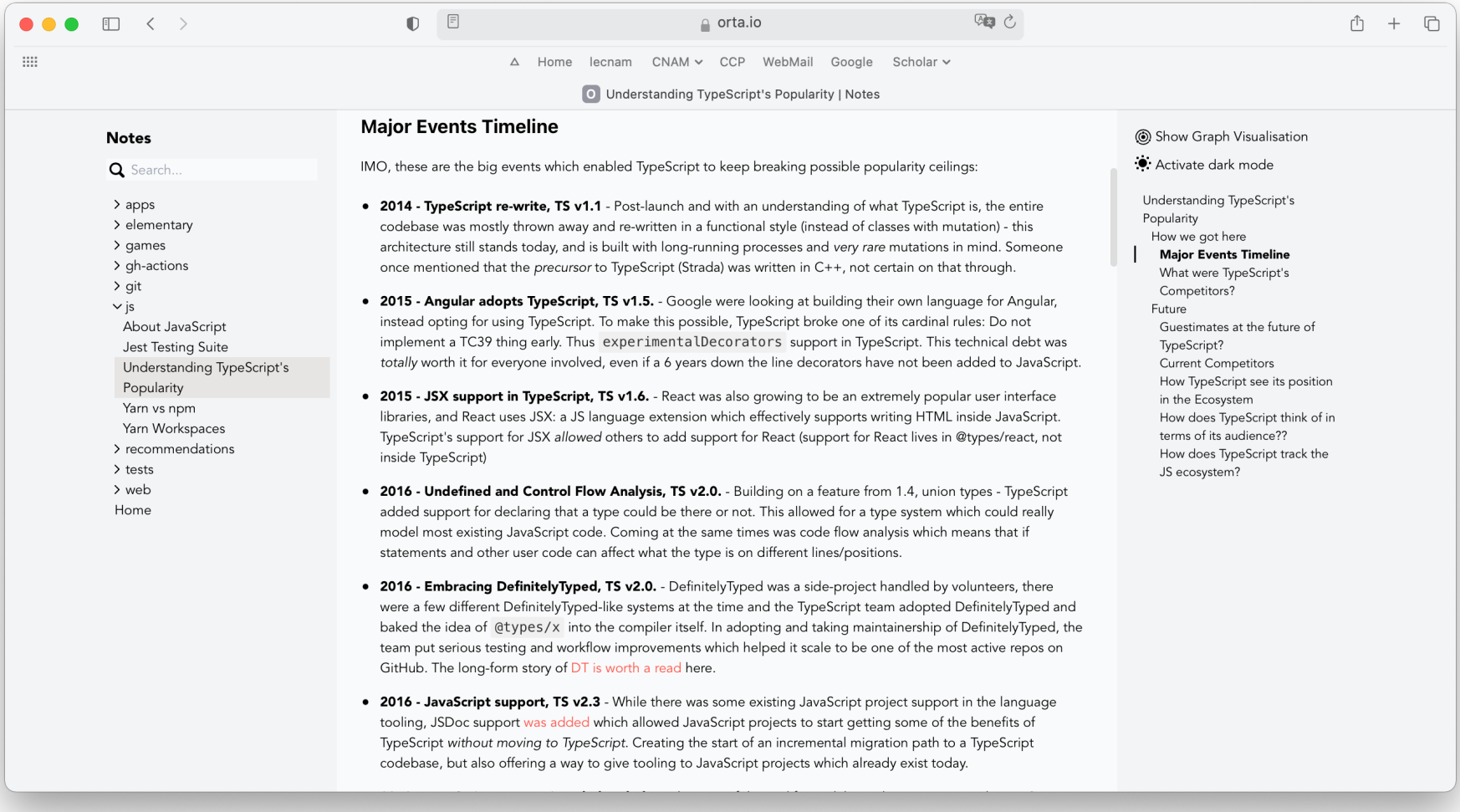
- **JavaScript** est un langage **impératif, objet et fonctionnel** à **typage dynamique.**
- **TypeScript** : extension **statiquement** typée de JavaScript.

  → recommandée, ou imposée, pour les principaux frameworks
- Types **prédéfinis** : number, string, boolean, ...
- Définition de **types de données structurés** : enregistrements et tableaux (*n*-uplets)
- Possibilité de rendre ces **types de données** immutables (attribut **readonly**)

  → notation *spread* de JavaScript pour les mises-à-jour fonctionnelles
- **Annotations** des fonctions (arguments et résultat) avec des types

  → annotations **vérifiées** puis **effacées** par le compilateur TypeScript

De JavaScript à TypeScript



https://orta.io/notes/js/why-typescript

Exemples : enregistrements mutables/immutables

```
type Point = { x: number, y: number }
const p1: Point = { x: 3, y: 7 }
x1 = p1.x; // ok
p1.x = 4; // ok
```

```
type Point = { readonly x: number, readonly y: number }
const p1: Point = { x: 3, y: 7 }
x1 = p1.x; // ok
p1.y = 4; // error
const p2: Point = { x: p1.x, y: 4 } // ok
const p3: Point = { ...p1, y: 4 } // ok, same point
```

Exemples : enregistrements immutables

```
type Address = {
    readonly streetNumber: number,
    readonly streetName: string,
    readonly postalCode: number,
    readonly city: string
}

type Contact = {
    readonly firstName: string,
    readonly lastName: string,
    readonly phone: string,
    readonly address: Address
}
```

Exemples : tableaux mutables/immutables

```
type Points = Point[]

const points: Points = [p1, p2, p3];
const p4: Point = points[2]; // ok
points[2] = p1; // ok
```

```
type Points = readonly Points[]

const points: Points = [p1, p2, p3];
const p4: Point = points[2]; // ok
points[2] = p1; // error
```

Tableaux immutables : insertion

- Cas particuliers : insertion au début (à gauche) ou à la fin (à droite)

```
function addLeft<A>(arr: readonly A[], index: number, value: A):
A[] {
    return [value, ...arr];
}
function addRight<A>(arr: readonly A[], index: number, value: A):
A[] {
    return [...arr, value];
}
```

- Cas général :

```
function insert<A>(arr: readonly A[], index: number, value: A): A[]
{
    return [...arr.slice(0, index), value, ...arr.slice(index)];
}
```

Tableaux immutables : mise-à-jour fonctionnelle

- La mise-à-jour fonctionnelle est similaire à l'insertion (seule la seconde tranche change) :

```
function update<A>(arr: readonly A[], index: number, value: A): A[]
{
    return [...arr.slice(0, index), value, ...arr.slice(index +
1)];
}
```

Remarque. Une nouvelle notation JavaScript pour `slice` est envisagée, cela donnerait :

```
    return [...arr.[:index], value, ...arr.[index + 1:]];
```

- Il est aussi possible de l'écrire avec la méthode `fill` :

```
function update<A>(arr: readonly A[], index: number, value: A): A[]
{
    return [...arr].fill(value, index, index + 1);
}
```