Développement front avancé

MMI 3 – TP#6 S6

IUT CERGY-PONTOISE

Danielo **JEAN-LOUIS**

Typescript

- Langage open source et gratuit développé par Microsoft
- Syntaxe inspiré par le C#
- Extension de fichier : .ts
  - .tsx si on utilise le JSX
- Appelé aussi Typescript

Source(s) :

- https://www.typescriptlang.org/ - anglais

Typescript

- Superset du javascript
  - Tout ce qui est valide en js l’est en Typescript
  - **Apporte le typage des variables**
- Fonctionne aussi bien en mode serveur que client

Source(s) :

- https://www.typescriptlang.org/ - anglais

Typescript

- Transpile le ts en javascript
  - Les navigateurs ne savent pas lire les fichiers typescript
- Transpilation :
  - Soit via la commande “tsc” (installée avec Typescript)
  - Soit via vite (géré nativement)

Source(s) :

- https://www.typescriptlang.org/ - anglais

Typescript

- Fonctionne via un système d’annotation qui ne sera pas présent dans les fichiers javascript

Source(s) :

- https://www.typescriptlang.org/ - anglais

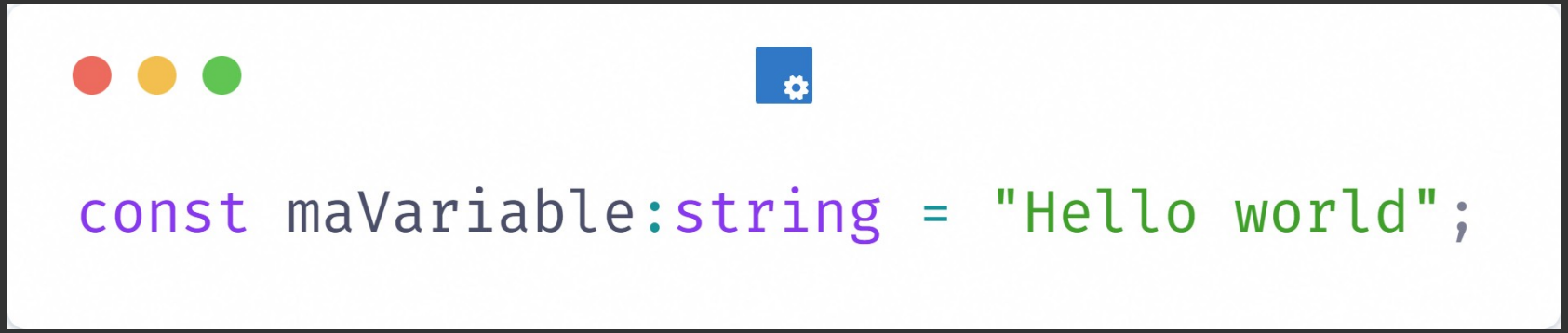
Installer Typescript

Installation de typescript dans le projet et donne accès à la commande “tsc”

Source(s) :

- https://www.npmjs.com/package/typescript - anglais

Typescript



Définition d’une constante de type “string”

Source(s) :

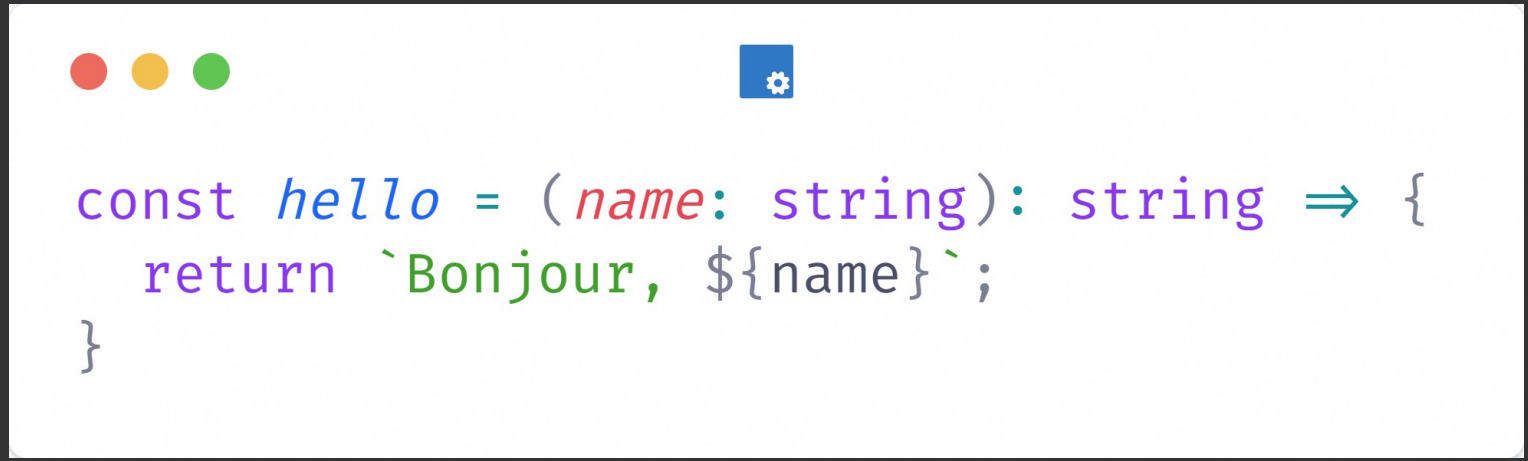
- https://www.typescriptlang.org/ - anglais

Intérêt du typage

- Réduction des erreurs de lors de l’exécution du code
  - Ex : multiplication entre un nombre et une chaîne de caractères
- Amélioration de l’autocomplétion de l’éditeur de code (IDE)

Intérêt du typage

- Rend la documentation du code naturelle



```
const hello = (name: string): string ⇒ {
  return `Bonjour, ${name}`;
}
```

Ici, on sait que notre fonction attend et retourne une “string”

Nouveaux types introduits (liste non exhaustive)

- Primitives : string, number et boolean
- any : “n’importe quoi”. Typescript ne vérifie pas le contenu
- unknown : comme “any”, mais typescript vérifie le type avant de l’utiliser
- never / void : Ne retourne rien (pour les fonctions)

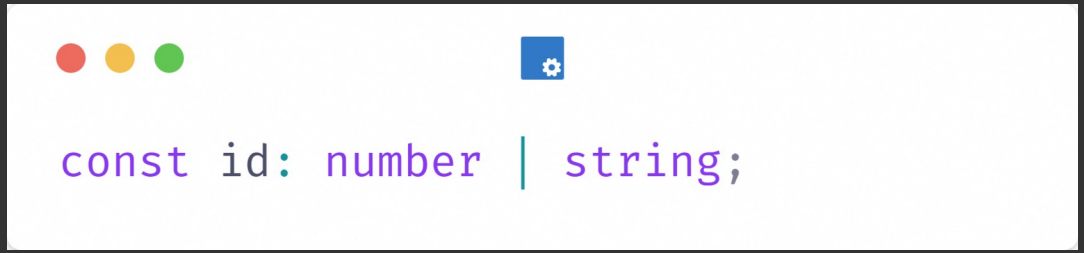
Source(s) :

- https://www.typescriptlang.org/fr/docs/handbook/2/everyday-types.html - anglais

Types combinés (liste non exhaustive)

- Tableau : (type)[]

- Union : typeA | typeB… (typeA ou typeB)



Source(s) :

- https://www.typescriptlang.org/fr/docs/handbook/2/everyday-types.html - anglais

Types combinés (liste non exhaustive)

- Objet: { [clé1] : type, [clé2] : type }

```
const myObject = {name: string; age: number};
```

- Tuple: [type1, type2]

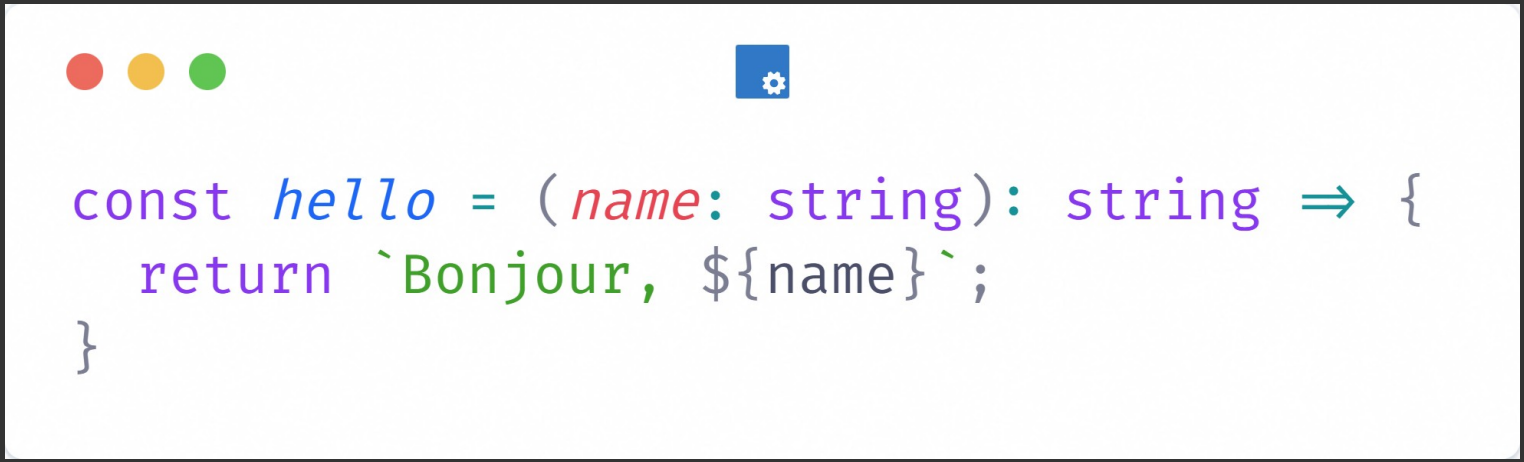
```
let tuple:[string, number];
```

Source(s) :

- https://www.typescriptlang.org/docs/handbook/basic-types.html#tuple - anglais

Fonctions

Le typage, comme vu précédemment, s’applique également aux fonctions



```
const hello = (name: string): string ⇒ {
  return `Bonjour, ${name}`;
}
```

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#functions - anglais

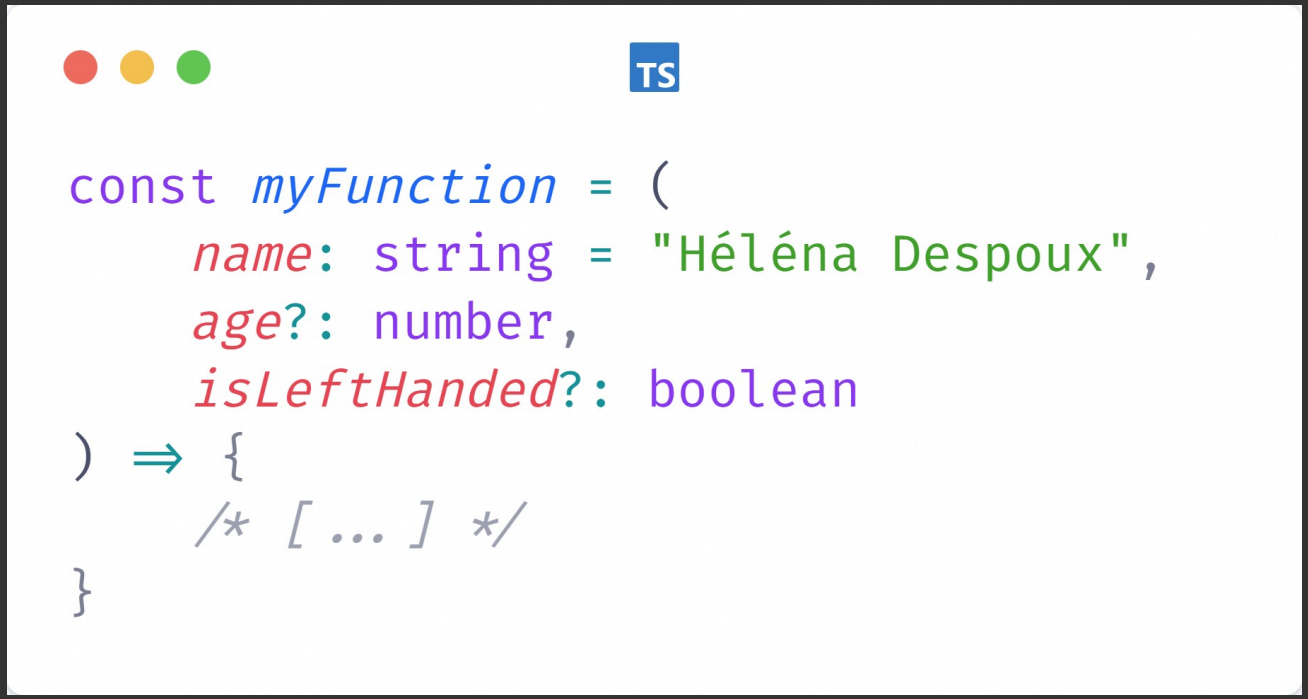
Valeur optionnelle

- Utilisation d’un point d’interrogation
- Indique qu’un paramètre ou une clé peut ne pas être présent
  - Dans le cadre d’une fonction, il doit toujours être en dernier
- Incompatible avec les valeurs par défaut d’une fonction

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/objects.html#optional-properties - anglais

Valeur optionnelle



```ts
const myFunction = (
    name: string = "Héléna Despoux",
    age?: number,
    isLeftHanded?: boolean
) => {
    /* [...] */
}
```

Les paramètres “age” et “isLeftHanded” sont optionnels.
Dans la structure de la fonction, on doit s’assurer que les paramètres existent

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/objects.html#optional-properties - anglais

Pratiquons ! - Typescript (Partie 1)

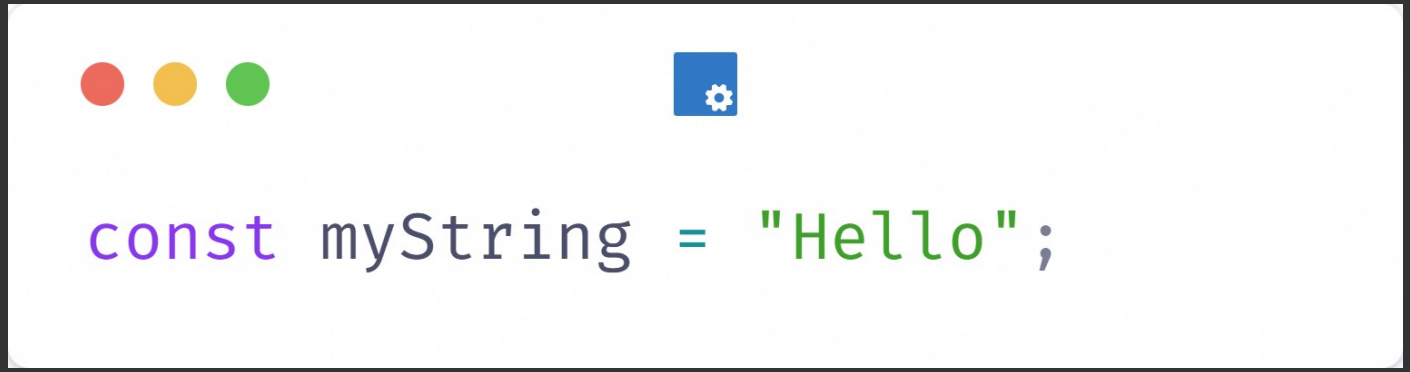
Pré-requis :

- Avoir la ressource ressources/typescript

A télécharger ici : https://github.com/DanYellow/cours/raw/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-6/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-6.ressources.zip

Inférence de type

- Typescript peut deviner le type des variables à partir de leur déclaration



Sans même définir le type, Typescript sait qu'on manipule une string (ou un literal pour être plus précis)

Source(s) :

- https://www.typescriptlang.org/docs/handbook/type-inference.html - anglais

Inférence de type

- **Ne fonctionne qu’avec les types simples**

Typescript sait qu’on manipule un élément HTML, mais il ne sait pas de quel type, il faut explicitement définir le type

Source(s) :

- https://www.typescriptlang.org/docs/handbook/type-inference.html - anglais

Assertion de type

- Utilisation du mot-clé “as”
- Aide Typescript à savoir le type de la variable et améliorer l’autocomplétion

```
const input = document.getElementById("name") as HTMLInputElement;
```

On indique que notre variable contient un élément HTML de type input

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-assertions - anglais

Assertion de type – Syntaxe alternative

```typescript
const input = <HTMLInputElement>document.getElementById("name");
const myString = <string>"hello";
```

Syntaxe alternative de l'assertion de type
(ne fonctionne pas dans le fichier .tsx)

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-assertions - anglais

Assertion de type – mot-clé “unknown”

- Permet d’outre-passer la vérification d’assertion
  - Ts lève une erreur si on essaye de transformer un number en string (et vice-versa)
- Utilisation du mot-clé “unknown” pour réinitialiser le type avant assertion

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-assertions - anglais

Assertion de type – mot-clé “unknown”

```
const myString = "Hello" as unknown as number;
```

On demande à typescript de nous laisser faire

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-assertions - anglais

Assertion de type – mot-clé “any”

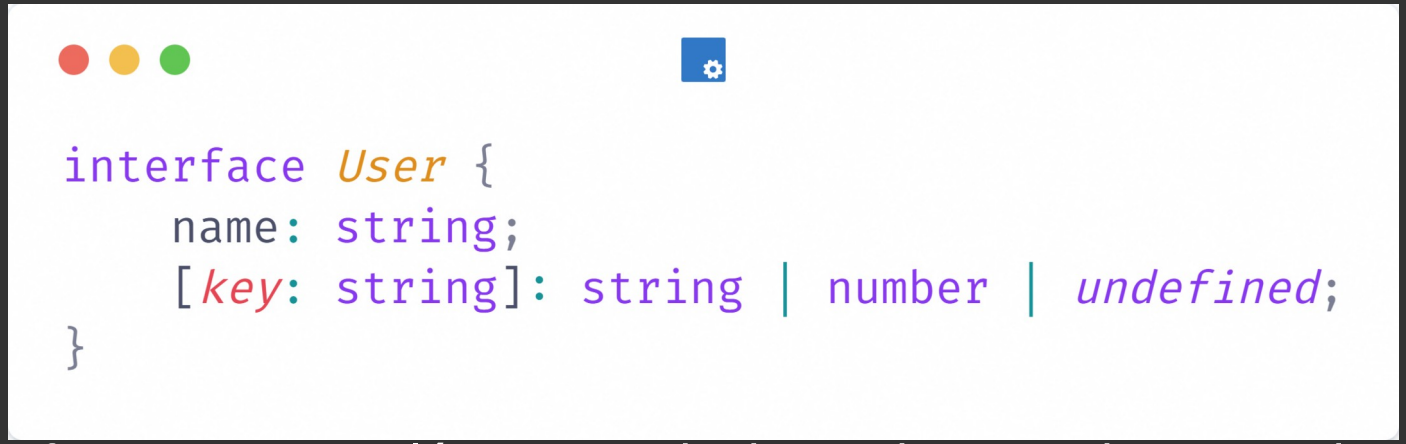
- Version primitive de “unknown”
- Indique à typescript de ne pas vérifier l’utilisation de la variable
- **Très mauvaise pratique**

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-assertions - anglais

Clés inconnues / Index signature

- Permet de définir le type d'une clé d'un objet sans en connaître le nom à l'avance



```
interface User {
    name: string;
    [key: string]: string | number | undefined;
}
```

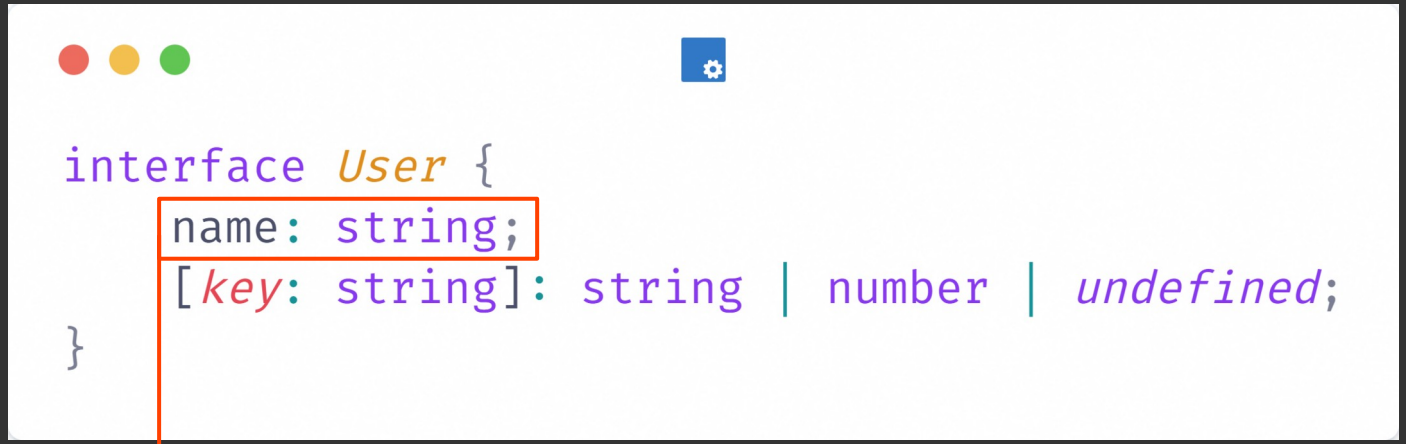
Notre interface aura une clé "name" ainsi que d'autres du type string, number ou undefined

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/objects.html#index-signatures - anglais

Clés inconnues / Index signature

- Ceci s’applique à l’ensemble de la structure



```
interface User {
    name: string;
    [key: string]: string | number | undefined;
}
```

“name: string” est aussi affecté par notre index signature, il faut donc que le type de “name” soit listé parmi l’index signature

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/objects.html#index-signatures - anglais

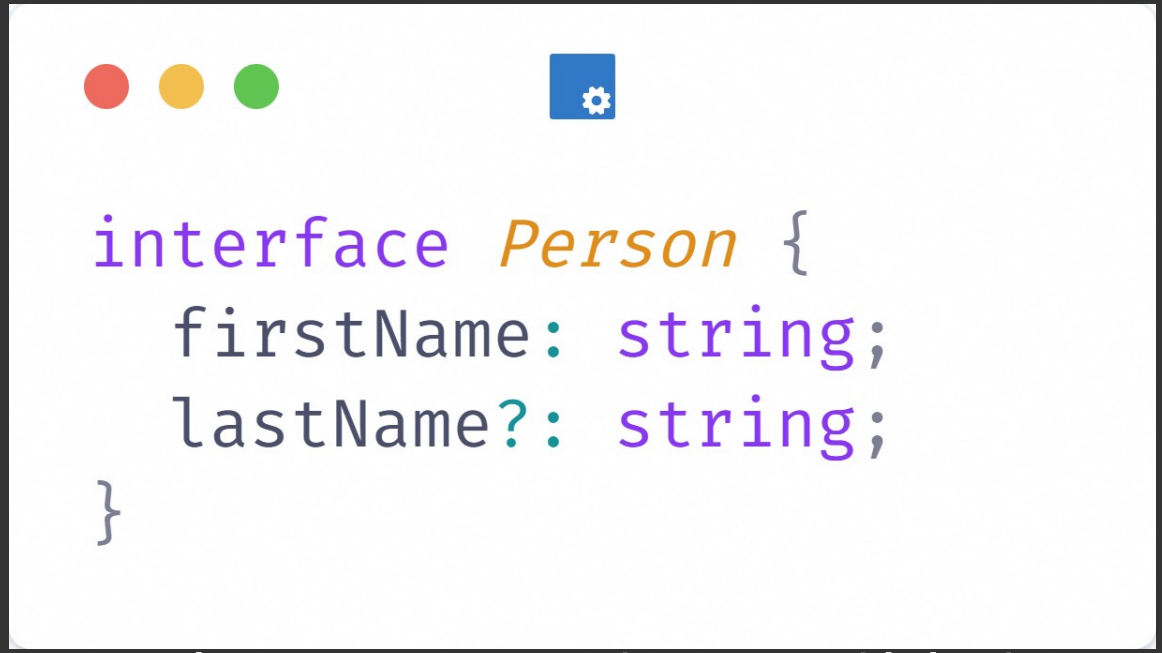
Types personnalisés

- Permettent d’ajouter de nouveaux types
- Utiles pour éviter la duplication
- Deux possibilités : type ou interface
  - Font la même chose à quelques exceptions près
- Peuvent d’hériter d’un autre type / interface
  - Syntaxes différentes

Source(s) :

- https://grafikart.fr/tutoriels/typescript-type-vs-interface-1955 - anglais

Types personnalisés



```
interface Person {
  firstName: string;
  lastName?: string;
}
```

Une interface nommée “Person” avec deux propriétés dont une optionnelle

Types personnalisés – Étendre l’existant

- Possibilité d’ajouter de nouvelles propriétés / méthodes à des objets natifs
- Utilisation d’un fichier “.d.ts”
  - Possibilité d’en avoir plusieurs par projet

Types personnalisés – Étendre l’existant

```
declare global {
    interface Window {
        createNotification: (message: string) ⇒ void;
    }
}
```

On ajoute une méthode “createNotification” à l’objet javascript “Window”

Types personnalisés – Étendre l’existant

```
declare module "my-module" {
    const myMethod: (text: string, options: {
        lang?: "fr" | "en",
        spellcheck?: boolean
    }) => void

    export default myMethod
}
```

On ajoute une méthode “myMethod” au module “my-module”

Importation de types

- Fonctionne comme l'importation de module
- Demande de préciser qu'on importe un type via le mot-clé "type"

```
import type { Person } from "./module.ts"
```

Source(s) :

- https://www.typescriptlang.org/docs/handbook/release-notes/typescript-3-8.html - anglais

Importation de types

- Syntaxe s’appliquant aussi bien aux types qu’aux interfaces
- Possibilité de combiner type et module

```
import { type Person, getUser } from "./module.ts"
```

Person est importé en tant que type, getUser en tant que fonction (ou variable)

Source(s) :

- https://www.typescriptlang.org/docs/handbook/release-notes/typescript-3-8.html - anglais

Pratiquons ! - Typescript (Partie 2)

Pré-requis :

- Avoir la ressource ressources/typescript

A télécharger ici : https://github.com/DanYellow/cours/raw/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-6/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-6.ressources.zip

tsconfig.json

- Fichier de configuration de typescript
- Indique quelle version de javascript on cible
  - Par défaut, ts compile dans la version la moins moderne de javascript
- Se place à la racine du projet (par défaut)
  - Le fichier est donc lu automatiquement

Source(s) :

- https://www.typescriptlang.org/docs/handbook/tsconfig-json.html

tsconfig.json

```jsonc
{
  "compilerOptions": {
    "target": "ES2022",
    "useDefineForClassFields": true,
    "module": "ESNext",
    "lib": ["ES2022", "DOM", "DOM.Iterable", "es2024"],
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "verbatimModuleSyntax": true,
    "moduleDetection": "force",
    "noEmit": true,

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "erasableSyntaxOnly": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedSideEffectImports": true
  },
  "include": ["src"],
}
```

Exemple de fichier tsconfig.json

Source(s) :

- https://www.typescriptlang.org/docs/handbook/tsconfig-json.html

tsconfig.json

- Existence de générateurs en ligne :
  - https://generator.tsconfigdemystified.com/
  - https://tsconfig.guide/
- Liste des clés de configuration :
  - https://www.typescriptlang.org/tsconfig/

Source(s) :

- https://www.typescriptlang.org/docs/handbook/tsconfig-json.html

Pratiquons ! - Typescript (Partie 3)

Pré-requis :

- Avoir la ressource ressources/typescript

A télécharger ici : https://github.com/DanYellow/cours/raw/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-6/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-6.ressources.zip

DOM

- Représentation sous forme d’objet de la structure d’une page HTML
- Compatible avec typescript
  - Ajout d’un ensemble de types dédiés

Source(s) :

- https://symfony.com/doc/current/routing.html

DOM

- Tous les types des éléments sont une union du type HTMLElement | null
  - Chaque élément peut être potentiellement être nul

Source(s) :

- https://www.typescriptlang.org/docs/handbook/dom-manipulation.html

DOM – Gestion de la nullité

- Trois options :
  - Vérifier avant utilisation l’existence de l’élément (structure if) ou chaining operator
  - Asservir la valeur (mot-clé “as”)
  - Utilisation du *non-null assertion operator*

Source(s) :

- https://www.typescriptlang.org/docs/handbook/dom-manipulation.html

Non-null assertion operator

- Représenté par un point d'exclamation (!)
- Permet d'indiquer à Typescript que l'élément existe et qu'il doit vous croire

```ts
const button = document.querySelector(".my-button");
button!.addEventListener('click', () => { /* [...] */})
```

On assure Typescript que notre bouton existe dans la page

Source(s) :

- https://symfony.com/doc/current/routing.html

Type narrowing – Rétrécissement des types

- Aide Typescript à réduire les possibilités de type
- Limite les erreurs
  - Exemple : Math.pow() sur une string
- Plusieurs opérateurs possibles : in, typeof, if...

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/narrowing.html

Type narrowing – Rétrécissement des types

```ts
type Student = { learnt: string }
type Professor = { teached: string }

const userKnowledge = (user: Student | Professor) => {
    if ("learnt" in user) {
        return user.learnt;
    }

    return user.teached;
}
```

Grâce à l'opérateur "in", on vérifie si le paramètre possède la clé "learnt", si c'est le cas, le paramètre est forcément du type "Student"

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/narrowing.html

Pratiquons ! - Typescript (Partie 4)

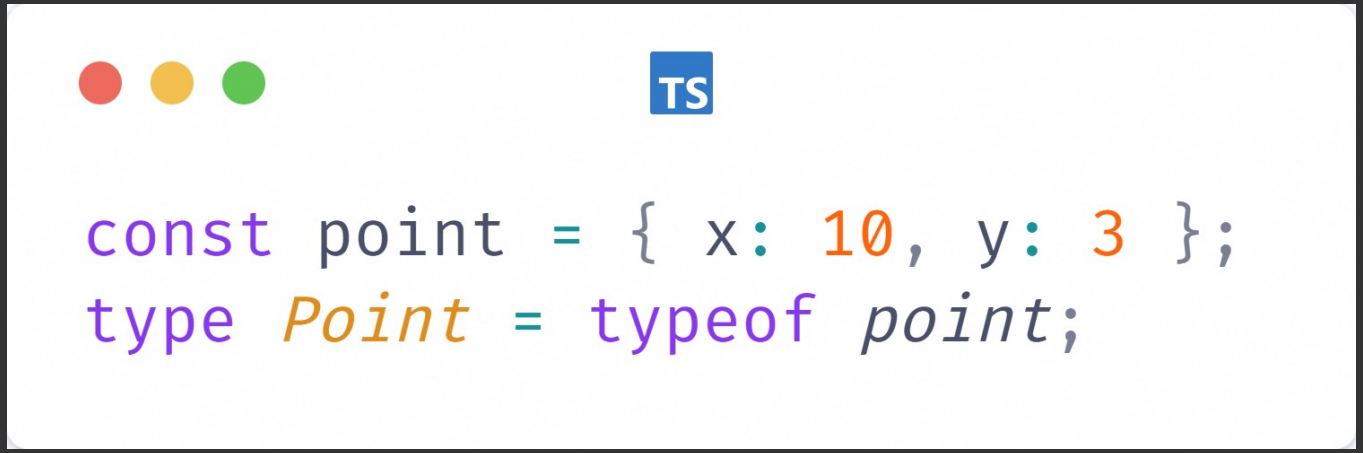
Pré-requis :

- Avoir la ressource ressources/typescript

A télécharger ici : https://github.com/DanYellow/cours/raw/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-6/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-6.ressources.zip

Opérateur typeof

- Permet de définir un type à partir d'un objet



Le type Point possède la même structure que la variable "point"

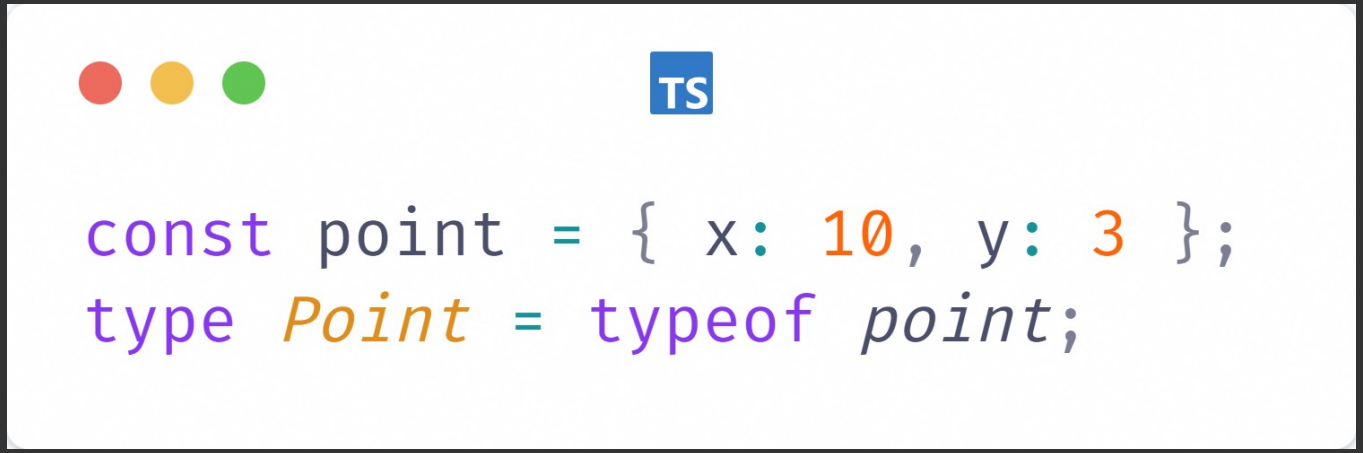
Note : Ne fonctionne qu'avec le mot-clé "type"

Source(s) :

- https://www.typescriptlang.org/docs/handbook/type-inference.html - anglais

Opérateur keyof typeof

- Permet de définir une union à partir d'un objet / interface / type



```ts
const point = { x: 10, y: 3 };
type Point = typeof point;
```

Le type n'accepte comme valeur que "x" et "y"

Note : Ne fonctionne qu'avec le mot-clé "type"

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/keyof-types.html#handbook-content - anglais

Pourquoi utiliser typeof et keyof ?

- Typage dynamique
  - Si on modifie la structure de base, ceci se répercute sur le reste
- Limitation de la duplication de code

Generics

- Type “adaptable”
- Utilisés dans les fonctions de récupération du DOM. Ex : querySelector()

```ts
const element = document.querySelector<HTMLAnchorElement>("a");
```

La variable n’est plus du type Element | null mais HTMLAnchorElement | null

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/generics.html

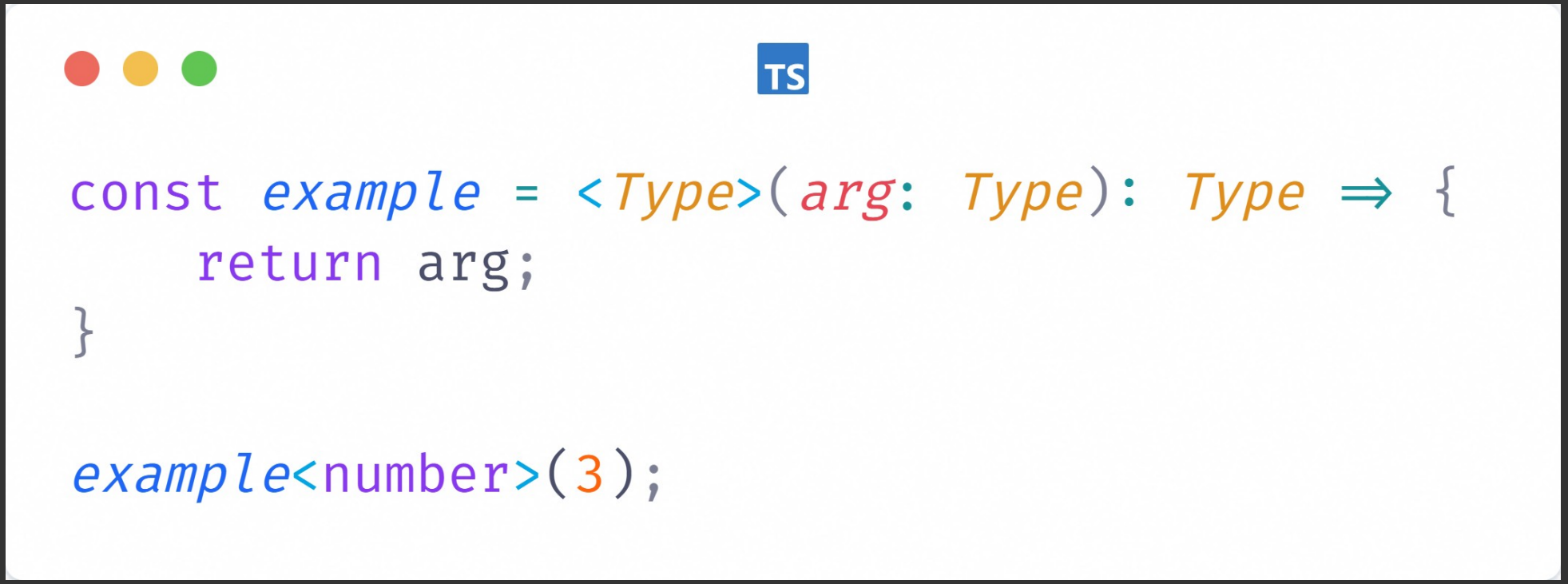
Generics

- Permettent de garder le type en entrée
  - Comparé au mot-clé “any”. Préférer les generics

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/generics.html

Generics



```ts
const example = <Type>(arg: Type): Type => {
    return arg;
}

example<number>(3);
```

Notre fonction “example” peut accepter n’importe quel type et Typescript avisera (Note: l’inférence de type s’applique aussi avec les generics)

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/generics.html

Generics

```ts
const example = <Type extends {age: number}>(arg: Type): number => {
    return arg.age;
}

type User = { age: number }
example<User>({ age: 25 });
```

Cette fois-ci notre fonction accepte n’importe quelle valeur à condition qu’elle ait une clé “age” de type number

Source(s) :

- https://www.typescriptlang.org/docs/handbook/2/generics.html

@ts-expect-error et @ts-nocheck

- @ts-expect-error : Ignore l’erreur jusqu’à ce que je la corrige (une ligne)
- @ts-nocheck : Ne vérifie pas le fichier
  - Désactivable avec @ts-check

Source(s) :

- https://www.stefanjudis.com/today-i-learned/the-difference-ts-ignore-and-ts-expect-error/
- https://www.typescriptlang.org/docs/handbook/intro-to-js-ts.html#ts-check

Typescript et l’IA

- Peut être très utile pour débugger certaines erreurs de typage de variable
  - Pensez à l’utiliser

Questions ?