Cześć!

Nazywam się **Konrad Badzio**

Statyczne typowanie w JS za pomocą TypeScript’a

Czym jest typowanie?

Czym jest typowanie?

- Określenie w kodzie rodzaju danych, a co za tym idzie, sposobu ich przechowywania pamięci komputera.
- Od typu zależy rozmiar pamięci przeznaczonej na naszą daną.
- Przykładowe typy danych to: liczba, tekst(ciąg znaków), boolean(prawda lub fałsz), tablica.
- W programowaniu wyróżniamy dwa rodzaje typowania: **statyczne** i **dynamiczne**.

Czym jest typowanie?

Typowanie dynamiczne

W językach typowanych dynamicznie, typ jest określany na podstawie zawartości zmiennej. Może być zmieniany podczas wykonywania programu.

```
//przykładowy kod w języku JavaScript:
let value = 'some text';
value = 123;
value = true;
value = [1, 2, 3, 4, 5];
```

Czym jest typowanie?

Typowanie statyczne

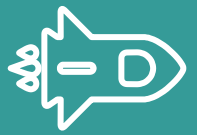
W językach typowanych statyczne typ zmiennej jest określany **jawnie** przy deklaracji zmiennej.

Typ zmiennej nie może być zmieniany podczas wykonywania programu.

```
//przykładowy kod w języku C#:
string value = 'some text';
value = 123; // ERROR!!!
```

Wady i zalety typowania statycznego

Zalety ?

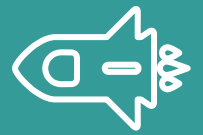


Jakie zalety posiada statyczne typowanie ?

Zalety typowania statycznego

- Statyczne typowanie umożliwia wyłapanie wielu błędów podczas kompilacji.
- Typy są zadeklarowane jawnie, co ułatwia zrozumienie kodu innym programistom, oraz nam, gdy wracamy do kodu po dłuższej przerwie.
- Statyczne typowanie zapewnia nam lepsze podpowiadanie składni.

Wady ?



Czy statyczne typowanie
posiada jakiekolwiek wady ?

(Nie do końca)Wady typowania statycznego

- Zwiększa złożoność programu / projektu.
- Wydłuża proces wytwarzania oprogramowania.
- Przestrzeganie większej ilości restrykcji oraz zasad podczas pisania kodu.

TypeScript !

Czym jest TypeScript?

- TypeScript to nadzbiór języka JavaScript stworzony przez Microsoft w 2012 roku.
- Kod pisany w TypeScript'cie jest transpilowany do Javascript'u.
- W momencie powstania uzupełniał JavaScript nie tylko o statyczne typowanie, ale również o możliwość pisania obiektowego.

TypeScript

Instalacja & first run

W celu instalacji Typescript'a globalnie, należy wykonać poniższe polecenie:

```
npm install -g typescript
```

Aby uruchomić kompilacje należy wykonać polecenie:

```
tsc {nazwa-pliku}
```

lub

```
npx typescript {nazwa-pliku}
```

TypeScript

Deklarowanie typów

Deklaracja typu następuje po nazwie zmiennej, a przed przypisaniem wartości.

```
let message: string = Twoja wiadomość';
```

Nie zawsze musimy deklarować typ. TypeScript posiada mechanizm zwany wnioskowaniem typu (*type inference*), który sam potrafi określić odpowiedni typ, w zależności od wartości przypisanej w inicjalizacji zmiennej.

```
let message = 'Coś bardzo ważnego';
message = 123;
//ERROR: Type '123' is not assignable to type 'string'
```

TypeScript

Typy podstawowe

Typy podstawowe opisują najpowszechniejsze i najprostsze ze struktur danych.

- boolean
- string
- number

```
let isHuman: boolean = true;
let message: string = 'Twoja wiadomość';
let amount: number = 30;
```

TypeScript

Typy podstawowe

Poza najprostszymi typami które omówiliśmy wcześniej, wśród typów podstawowych znajdują się również:

- array
- enum
- any

TypeScript

Typy array i enum

```
let values: number[] = [1, 2, 3, 4, 5];
let values: Array<number> = [10, 20, 30, 40, 50];
//array - obie deklaracje są równoważne

enum Role {
   Admin,
   User = 1,
   Guest
};
let role: Role = Role.Admin;
//enum - typ wyliczeniowy
```

TypeScript

Typ any

Typ **any** jest specjalnych typem, który oznacza, że zmienna może być dowolnego typu. Jest to mechanizm, który ułatwia pracę z już istniejącym kodem, bądź z bibliotekami, które nie posiadają zadeklarowanych typów.

```
let value: any = 'Konrad';
value = 123;
value = [1, 2, 3, 4, 5];
//any: kompilator pozwala na takie przypisania, zmienna może być dowolnego typu
```

Typu **any** powinniśmy używać **tylko** kiedy mamy uzasadnioną potrzebę!!

TypeScript

Rzutowanie typów

Chcąc rzutować typ na inny mamy do dyspozycji dwie metody:

```
let someVariable: any = [1, 2, 3, 4, 5];

let values = someVariable as Array<number>;
let values = someVariable as number[];
//wykorzystując operator "as"

let values = <Array<number>>someVariable;
let values = <number[]>someVariable;
//deklarując typ tuż przed przypisywaną zmienną
```

TypeScript

Konwersja typów

Konwersji typów dokonujemy za pomocą funkcji wbudowanych w język JavaScript:

```
let year: string = "1240";

let value = Number.parseInt(year);
let value: number = Number.parseInt(year);
```

Zadanie 1

Popraw błędy związane z typowaniem

TypeScript

Funkcje

Składnia dla funkcji jest następująca:

```
function getUserRoles(userId: string): Array<Role> {
   //ciało funkcji
}
```

Składnia z wykorzystaniem arrow function:

```
const getUserRoles = (userId: string): Array<Role> => {
   //ciało funkcji
}
```

TypeScript

Funkcje - typ void

Typ **void** opisuje brak posiadania typu. Używamy go, gdy chcemy zaznaczyć, że funkcja nie będzie zwracała żadnej wartości.

```
const refreshPage = (): void => {
   //ciało funkcji
}
```

TypeScript

Funkcje - parametry opcjonalne

Stawiając znak zapytania po nazwie parametru określamy go jako opcjonalny. Kompilator nie będzie wymagał jego przekazania.

```
function sendMessage(title: string, message?: string): boolean {
   //ciało funkcji
}
```

Parametry opcjonalne powinny być deklarowane zawsze na końcu.

Zadanie 2

Uzupełnij funkcje o brakujące typy

Interfejsy w TypeScript’cie

TypeScript

Interfaces

- Obiekty implementują interfejsy.
- Interfejsy są pewnego rodzaju **kontraktem**, który musi zostać spełniony przez obiekt implementujący.
- Właściwościami interfejsów mogą być: typy proste, funkcje, enum itd…
- Interfejsy wykorzystujemy do nadawania “kształtu” obiektom.

```
interface User {
   id: number;
   userName: string;
   role: Role;
}
```

TypeScript

Interfaces

```
//przykładowy kod w JavaScript
let user = {
   id: 1,
   userName: 'sample-user',
   role: 'admin'
}
```

TypeScript

Interfaces - przykład użycia

```
interface User {
   id: number;
   userName: string;
   role: Role;
}
const getUser = (userId: string): User => {
   let user: User = {
       id: 1,
       userName: 'sample-user',
       role: Role.Admin
   }
   return user;
}
```

TypeScript

Słowo kluczowe: extends

Interfejsy można rozszerzać o dodatkowe pola za pomocą słowa kluczowego **extends**. Dzięki temu, możemy łączyć ze sobą dwie lub więcej deklaracji interfejsów.

```
interface User {
   id: number;
   userName: string;
   role: Role;
}
interface UserDetails extends User {
   shoeSize: number;
}
```

Zadanie 3

Wróć do zadania 2 i uzupełnij kod o deklaracje interfejsów

Klasy w TypeScript’cie

TypeScript

Klasy

- Klasy są jednocześnie definicją typu jak i jego implementacją.
- Klasy podobnie jak interfejsy mogą być rozszerzane.
- Klasy mogą implementować interfejsy.

```
class User {
   id: number;
   userName: string;
   role: Role;
}
let user = new User();
```

TypeScript

Klasy

- Klasy w JavaScript'cie nie są w pełni zgodne z doktrynami programowania obiektowego. Typescript'owe klasy natomiast pozwalają na pisanie wg. założeń OOP (Object Oriented Programming).
- Pozwalają na hermetyzację dzięki modyfikatorom dostępu.
- TypeScript pozwala na definiowanie getterów i setterów w klasach.
- Klasy mogą posiadać pola statyczne.

Modyfikatory dostępu

TypeScript

Modyfikatory dostępu - public

Właściwość jest dostępna z zewnątrz klasy. Jeżeli właściwość nie posiada modyfikatora dostępu to domyślnie jest to **public**.

```
class User {
   id: number;
   public userName: string = 'Konrad';
}
var user = new User();
console.log(user.userName);
//Konrad
```

TypeScript

Modyfikatory dostępu - private

Oznacza, że właściwość jest dostępna jedynie z zawierającej ją klasy. Nie mamy do niej dostępu poza klasą.

```
class User {
   id: number;
   private userName: string = 'Konrad';
}
var user = new User();
console.log(user.userName);
//Property 'userName' is private and only accessible within class 'User'
```

TypeScript

Modyfikatory dostępu - protected

Zachowuje się jak private, z wyjątkiem, że właściwość może być dostępna wewnątrz klasy pochodnej.

```
class User {
   protected userName: string = 'Konrad';
}
class SuperUser extends User {
   public setUserName(userName: string = ""){
       this.userName = userName;
   }
}
let user = new SuperUser();
```

TypeScript

Modyfikatory dostępu - readonly

Oznacza, że właściwość jest “tylko do odczytu” oraz jej wartość może być ustawiona podczas inicjalizacji lub w konstruktorze.

```
class User {
    readonly userName: string;
    constructor(userName: string = "") {
        userName = userName;
    }
    public setUserName(userName: string){
        this.userName = userName;
    }
}
//Cannot assign to 'userName' because it is a read-only property.
```

Pola statyczne

TypeScript - pola statyczne

TypeScript pozwala na definicje metod oraz pól statycznych, czyli dostępnych bez konieczności tworzenia obiektu danej klasy.

```
class EmailValidator {
   public static validate(email: string): boolean {
       //validation
   }
}
EmailValidator.validate('example@domain.com');
```

Gettery i Settery

TypeScript - gettery i settery

```
Class User {
    firstName: string = 'Konrad';
    lastName: string = 'Badzio';
    private _password: string = 'secret';
    get fullName(): string {
        return this.firstName + ' ' + this.lastName;
    }
    set password(password: string) {
        if (password.length < 10) throw 'Password must have at least 10 characters.';
        this._password = password;
    }
}
let user = new User();
console.log(user.fullName); //Konrad Badzio
user.password = 'super-secret';
```

Zadanie 4

Stwórz service, który będzie potrafił zarządzać użytkownikami

TypeScript w React

TypeScript w React

Aby utworzyć project create-react-app z TypeScript'em należy wykonać polecenie:

```
npx create-react-app my-app --typescript
```

Aby dodać do już istniejącego projektu należy wykonać polecenie:

```
npm install --save typescript @types/node @types/react @types/react-dom @types/jest
```

Dzieki!