



Technologie internetowe

TypeScript

Paweł Rajba

pawel@cs.uni.wroc.pl

<http://pawel.ii.uni.wroc.pl/>

Plan wykładu

- Wprowadzenie to TypeScript
- Podstawowe typy
- Interfejsy
- Klasy
- Generyki
- Moduły
- Przestrzenie nazw

Wprowadzenie do TypeScript

- Strona domowa: <https://www.typescriptlang.org/>
- Nadzbiór JavaScript
- Kompiluje się do JavaScript zgodnego z ES3
- Poprzez wprowadzenie typów umożliwia statyczną weryfikację kodu
- Wsparcie wielu narzędzi deweloperskich
- Łatwo zacząć na dostępnym online playground: <https://www.typescriptlang.org/play/index.html>

Visual Studio



Visual Studio 2015



Visual Studio 2013



Visual Studio Code

And More...



Sublime Text



Atom



Eclipse



Emacs



WebStorm



Vim

Jak zacząć w Microsoft Azure?

- Konsola MS Azure
- Instalacja zgodnie z instrukcją:

```
> npm install -g typescript
```

- ... i kompilacja

```
tsc greeter.ts
```

DEMO

- HelloWorld

- 
- Materiał został przygotowany w oparciu o dokumentację języka:
 - <https://www.typescriptlang.org/docs/tutorial.html>

Podstawowe typy

- Boolean

```
let isDone: boolean = false;
```

- Number

```
let decimal: number = 6;  
let hex: number = 0xf00d;  
let binary: number = 0b1010;  
let octal: number = 0o744;
```

- String

```
let color: string = "blue";  
color = 'red';
```

```
let fullName: string = `Bob Bobbington`;  
let age: number = 37;  
let sentence: string = `Hello, my name is ${ fullName }.  
  
I'll be ${ age + 1 } years old next month.`
```

Podstawowe typy

- Array

```
let list: number[] = [1, 2, 3];
```

```
let list: Array<number> = [1, 2, 3];
```

- Tuple

```
// Declare a tuple type  
let x: [string, number];  
// Initialize it  
x = ["hello", 10]; // OK  
// Initialize it incorrectly  
x = [10, "hello"]; // Error
```


Podstawowe typy

- Enum

```
enum Color {Red, Green, Blue};  
let c: Color = Color.Green;
```

```
enum Color {Red = 1, Green, Blue};  
let c: Color = Color.Green;
```

```
enum Color {Red = 1, Green = 2, Blue = 4};  
let c: Color = Color.Green;
```

```
enum Color {Red = 1, Green, Blue};  
let colorName: string = Color[2];  
  
alert(colorName);
```

Podstawowe typy

- Any

```
let notSure: any = 4;  
notSure = "maybe a string instead";  
notSure = false; // okay, definitely a boolean
```

- Void

```
function warnUser(): void {  
    alert("This is my warning message");  
}
```

- Never

```
function error(message: string): never {  
    throw new Error(message);  
}
```

Interfejsy

- Pierwszy przykład
 - Taki kod

```
function printLabel(labelledObj: { label: string }) {  
    console.log(labelledObj.label);  
}  
  
let myObj = {size: 10, label: "Size 10 Object"};  
printLabel(myObj);
```

- możemy zamienić na następujący

```
interface LabelledValue {  
    label: string;  
}  
  
function printLabel(labelledObj: LabelledValue) {  
    console.log(labelledObj.label);  
}  
  
let myObj = {size: 10, label: "Size 10 Object"};  
printLabel(myObj);
```

Interfejsy

- Właściwości opcjonalne

```
interface SquareConfig {  
    color?: string;  
    width?: number;  
}  
  
function createSquare(config: SquareConfig): {color: string; area: number} {  
    let newSquare = {color: "white", area: 100};  
    if (config.color) {  
        newSquare.color = config.color;  
    }  
    if (config.width) {  
        newSquare.area = config.width * config.width;  
    }  
    return newSquare;  
}  
  
let mySquare = createSquare({color: "black"});
```

Interfejsy

- Funkcje

```
interface SearchFunc {  
    (source: string, subString: string): boolean;  
}
```

```
let mySearch: SearchFunc;  
mySearch = function(source: string, subString: string) {  
    let result = source.search(subString);  
    if (result == -1) {  
        return false;  
    }  
    else {  
        return true;  
    }  
}
```

Interfejsy

- Zagnieżdżanie interfejsów

```
interface Shape {
    color: string;
}

interface PenStroke {
    penWidth: number;
}

interface Square extends Shape, PenStroke {
    sideLength: number;
}

let square = <Square>{};
square.color = "blue";
square.sideLength = 10;
square.penWidth = 5.0;
```

Interfejsy

- Implementacja interfejsu

```
interface ClockInterface {
    currentTime: Date;
    setTime(d: Date);
}

class Clock implements ClockInterface {
    currentTime: Date;
    setTime(d: Date) {
        this.currentTime = d;
    }
    constructor(h: number, m: number) { }
}
```

Klasy

- Podstawowa składnia

```
class Greeter {  
    greeting: string;  
    constructor(message: string) {  
        this.greeting = message;  
    }  
    greet() {  
        return "Hello, " + this.greeting;  
    }  
}  
  
let greeter = new Greeter("world");
```


Klasy

- Dziedziczenie

```
class Animal {
  name: string;
  constructor(theName: string) { this.name = theName; }
  move(distanceInMeters: number = 0) {
    console.log(`${this.name} moved ${distanceInMeters}m.`);
  }
}

class Snake extends Animal {
  constructor(name: string) { super(name); }
  move(distanceInMeters = 5) {
    console.log("Slithering...");
    super.move(distanceInMeters);
  }
}

class Horse extends Animal {
  constructor(name: string) { super(name); }
  move(distanceInMeters = 45) {
    console.log("Galloping...");
    super.move(distanceInMeters);
  }
}

let sam = new Snake("Sammy the Python");
let tom: Animal = new Horse("Tommy the Palomino");

sam.move();
tom.move(34);
```

Klasy

- Dostępność
 - Public (domyślnie)
 - Private
 - Protected
- Zmienne statyczne

```
class Grid {
  static origin = {x: 0, y: 0};
  calculateDistanceFromOrigin(point: {x: number; y: number;}) {
    let xDist = (point.x - Grid.origin.x);
    let yDist = (point.y - Grid.origin.y);
    return Math.sqrt(xDist * xDist + yDist * yDist) / this.scale;
  }
  constructor (public scale: number) { }
}

let grid1 = new Grid(1.0); // 1x scale
let grid2 = new Grid(5.0); // 5x scale

console.log(grid1.calculateDistanceFromOrigin({x: 10, y: 10}));
console.log(grid2.calculateDistanceFromOrigin({x: 10, y: 10}));
```

Klasy

- Klasy abstrakcyjne

```
abstract class Animal {  
    abstract makeSound(): void;  
    move(): void {  
        console.log("roaming the earth...");  
    }  
}
```

Generyki

- Pierwszy przykład
 - Zamiast konstrukcji

```
function identity(arg: number): number {  
    return arg;  
}
```

```
function identity(arg: any): any {  
    return arg;  
}
```

- Można napisać

```
function identity<T>(arg: T): T {  
    return arg;  
}
```

- A później: albo podać typ, albo użyć „wnioskowania”

```
let output = identity<string>("myString"); // type of output will be 'string'
```

```
let output = identity("myString"); // type of output will be 'string'
```

Generyki

- Usprawniając przykład otrzymujemy generyczny interfejs

```
interface GenericIdentityFn<T> {  
    (arg: T): T;  
}  
  
function identity<T>(arg: T): T {  
    return arg;  
}  
  
let myIdentity: GenericIdentityFn<number> = identity;
```

Generyki

- Klasy generyczne

```
class GenericNumber<T> {  
    zeroValue: T;  
    add: (x: T, y: T) => T;  
}  
  
let myGenericNumber = new GenericNumber<number>();  
myGenericNumber.zeroValue = 0;  
myGenericNumber.add = function(x, y) { return x + y; };
```

Generyki

- Ograniczenia

```
interface Lengthwise {  
    length: number;  
}  
  
function loggingIdentity<T extends Lengthwise>(arg: T): T {  
    console.log(arg.length); // Now we know it has a .length property, so no more erro  
r  
    return arg;  
}
```

Moduły

- Pozwala na porządkowanie większego kodu
- Podstawowe mechanizmy
 - Eksport zawartości modułu

```
class ZipCodeValidator implements StringValidator {
    isAcceptable(s: string) {
        return s.length == 5 && numberRegexp.test(s);
    }
}
export { ZipCodeValidator };
export { ZipCodeValidator as mainValidator };
```

- Import w innym pliku

```
import { ZipCodeValidator } from "./ZipCodeValidator";

let myValidator = new ZipCodeValidator();
```

```
import { ZipCodeValidator as ZCV } from "./ZipCodeValidator";
let myValidator = new ZCV();
```


Przestrzenie nazw

- Analogiczny mechanizm do modułów
- Nowsze podejście
- Pozwala zdefiniować jeden namespace w wielu plikach
- Więcej:
 - <https://www.typescriptlang.org/docs/handbook/namespaces.html>
- Rozważania nt. przestrzeni nazw i modułów
 - <https://www.typescriptlang.org/docs/handbook/namespaces-and-modules.html>

DEMO

- Oglądamy przykłady ze strony <https://www.typescriptlang.org/samples/index.html>
 - Hello World
 - Raytracer
 - JQuery Parallax Starfield
 - Warship Combat