TypeScript

Eko Kurniawan Khannedy

License

- Dokumen ini boleh Anda gunakan atau ubah untuk keperluan non komersial
- Tapi Anda wajib mencantumkan sumber dan pemilik dokumen ini
- Untuk keperluan komersial, silahkan hubungi pemilik dokumen ini

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- youtube.com/c/ProgrammerZamanNow

Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/khannedy
- Twitter : twitter.com/khannedy
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com

Pengenalan TypeScript

Sebelum Belajar TypeScript

- Ada baiknya kita mengerti JavaScript
- Mengerti dasar NodeJS dan NPM (Node Package Manager)

Apa itu TypeScript

- TypeScript adalah bahasa pemrograman berorientasi objek yang dibuat oleh Microsoft
- TypeScript adalah bahasa pemrograman yang di compile menjadi kode JavaScript
- Menggunakan TypeScript akan membuat kode kita lebih mudah di baca dan di debug dibandingkan menggunakan JavaScript
- TypeScript adalah bahasa pemrograman yang Strongly Type (Seperti Java, C#, dan C/C++)

Proses Development TypeScript

Website TypeScript

- https://www.typescriptlang.org/
- https://github.com/microsoft/TypeScript

Menginstall TypeScript

```
npm install -g typescript
```

Program Hello World

Mengkompilasi Kode TypeScript

```
tsc namafile.ts

// akan menghasilkan namafile.js
// gunakan nodejs untuk menjalankan

node namafile.js
```

Hello World di TypeScript

```
// hello.ts

function sayHello(name: string){
    return `Hello ${name}`;
}

let hello: string = sayHello(“Eko”);

console.log(hello)
```

Tipe Data Primitif

Sebelum Lanjut Materi

```
npm install -g ts-node

ts-node namafile.ts
```

Tipe Data Boolean

```
let isDone: boolean = false;

let isMarried: boolean = true;
```

Tipe Data Number

```
let decimal: number = 6;

let hex: number = 0xf00d;

let binary: number = 0b1010;

let octal: number = 0o744;
```

Tipe Data String

```
let nama: string = “Eko Kurniawan”;

nama = nama + “ Khannedy”;


let hello: string = `Hello ${nama}`;
```

Tipe Data Array

Tipe Data Array

```
let names: string[] = [“Eko”, “Kurniawan”, “Khannedy”]

let fruits: Array<number> = [“Apple”, “Orange”]
```

Tipe Data Tuple

```
let student: [string, string, number] = [“1001”, “Eko”, 100]
```

Mengakses Array & Tuple

```
let names: string[] = [“Eko”, “Kurniawan”, “Khannedy”]
let student: [string, string, number] = [“1001”,
“Eko”, 100]

console.log(names[0]);
console.log(names[1]);
console.log(student[0]);
```

Mengubah Data Array & Tuple

```
let names: string[] = [“Eko”, “Kurniawan”, “Khannedy”]
let student: [string, string, number] = [“1001”,
“Eko”, 100]

names[0] = “TypeScript”;
names[1] = “Programmer”;
student[0] = “Now”
```

Menghapus Data Array & Tuple

```
let names: string[] = [“Eko”, “Kurniawan”, “Khannedy”]
let student: [string, string, number] = [“1001”,
“Eko”, 100]

delete names[0];
delete names[1];
delete student[0];
```

Tipe Data Enum

Tipe Data Enum

```
enum Gender { Male, Female, Unknown }


let jenisKelamin: Gender = Gender.Male;
```

Merubah Nilai Enum

```
enum Gender { Male = 1, Female = 2, Unknown = -1 }


let jenisKelamin: Gender = Gender.Male;
```

Enum String

```
enum Gender {
    Male = “MALE”,
    Female = “FEMALE”,
    Unknown = “UNKNOWN”
}

let jenisKelamin: Gender = Gender.Male;
```

Tipe Data Lainnya (Any, Void, Null, Undefined dan Never)

Tipe Data Any

```
let data: any = callLibraryFunction(...);

data = 100;

data = “Eko Kurniawan”;
```

Tipe Data Void

```
function printHello(name: string): void {

    console.log(`Hello ${name}`);

}

printHello(“Eko”);
```

Tipe Data Null dan Undefined

```
let fullName: string = null;

let nilai: number = undefined;
```

Tipe Data Never

```
function notImplemented(): never {
    throw new Error(“Not Yet Implemented”);
}

function infiniteLoop(): never {
    while(true){}
}
```

Tipe Data Object

Tipe Data Object

- Tipe data object adalah tipe data yang bukan primitive : string, number, boolean
- Enum juga bukan tipe data object
- Array dan Tuple adalah tipe data Object

Membuat Data Object

```
let student: object = {
    “nim” : “10001”,
    “name” : “Eko Kurniawan”,
    “value” : 100,
    “address” : {
        “country” : “Indonesia”
    }
}
```

Kenapa Error?

```
let nim: string = student.nim;

// error TS2339: Property 'nim' does not exist on type
'object'.
```

Kenapa Error?

- TypeScript adalah strongly type language
- Untuk memastikan tidak ada kesalahan pada saat runtime
- Kita bisa membuatnya menjadi tipe data any jika ingin, namun konsekuensinya bisa memungkinnan terjadi error pada saat runtime, misal mengakses atribut yang tidak ada.

Type Assertion

Apa itu Type Assertion?

- Type Assertion biasa di bahasa pemrograman lain biasa disebut Type Cast.
- Type Assertion adalah kemampuan memberitahu compiler untuk merubah tipe data sesuai dengan kemauan kita.
- Biasanya Type Assertion digunakan ketika mengakses kode JavaScript sehingga compiler tidak tahu tipe data balikan kode JavaScript tersebut.

Cara Menggunakan Type Assertion

```
// misal ada fungsi js sayHello(name):any

let response1: string = sayHello(“Eko”) as string;

let response2: string = <string>sayHello(“Eko”);

let number1: number = (<string>sayHello(“Eko”)).length
```

Variable Var

Deklarasi Variable

TypeScript mendukung 3 keyword untuk mendeklarasikan variable

- var
- let
- const

Menggunakan Var

```
var fullName: string = “Eko Kurniawan Khannedy”;

var value: number;

value = 1000;
```

Scope

```
var fullName: string = "Eko";

function sayHello() {
    console.log(`Hello ${fullName}`)
}

sayHello();
```

Masalah di Var

- Mayoritas dalam bahasa pemrograman lain, biasanya variable hanya bisa diakses di function/scope dibawahnya, dan tidak bisa diakses dari function/scope diatas nya.
- Sayangnya, dalam JavaScript, Var bisa diakses dari mana saja. Tidak hanya di function/scope diatasnya, bahkan dalam satu file, satu package.
- Oleh karena itu penggunaan Var sudah tidak direkomendasikan lagi.
- Selalu gunakan Let sebagai pengganti Var.

Contoh Masalah di Var

```
var response: string = "Hello";

function sayHello(isBoss: boolean) {
    if (isBoss) {
        var response: string = "Hello Bos";
    }
    console.log(response);
}

sayHello(true);
```

Variable Let

Menggunakan Let

```
let fullName: string = “Eko Kurniawan Khannedy”;

let value: number;

let = 1000;
```

Solusi Masalah di Var dengan Let

```
let response: string = "Hello";

function sayHello(isBoss: boolean) {
    if (isBoss) {
        let response: string = "Hello Bos";
    }
    console.log(response);
}

sayHello(true);
```

Variable Const

Variable Const

- Penggunaan variable Const sama seperti Var dan Let
- Yang membedakan Const dengan yang lainnya adalah, value yang sudah di set ke dalam variable, tidak bisa diubah lagi.
- Scope pada Const sama seperti scope pada Let

Menggunakan Const

```
const fullName: string = “Eko”;

fullName: string = “Eko Kurniawan”; // ERROR
```

If Else

If Else

- Hampir semua bahasa pemrograman memiliki fitur percabangan berdasarkan kondisi tertentu
- Pada TypeScript fitur tersebut sama dengan di JavaScript ataupun bahasa seperti Java, C# dan C/C++

If

```
if(kondisi boolean terpenuhi){
    // eksekusi kode program
}
```

If Else

```
if(kondisi boolean terpenuhi){
    // eksekusi kode program
} else {
    // eksekusi jika false
}
```

If Else If

```
if(kondisi1 boolean terpenuhi){
    // eksekusi kode program
} else if (kondisi2 boolean terpenuhi) {
    // eksekusi kode program
} else {
    // eksekusi jika tak ada kondisi terpenuhi
}
```

Ternary Operator

```
kondisi boolean ? statement true : statement false
```

Switch

Switch

- Switch merupakan versi sederhana dari If Else
- Switch digunakan untuk mengecek sebuah nilai dengan beberapa kemungkinan

Switch

```
let ipk : number = 4;
switch(ipk){
    case 4 :
        console.log(“ISTIMEWA”);
        break;
    case 3 :
        console.log(“BAIK);
        break;
    case 2 :
        console.log(CUKUP);
        break;
    case 1 :
        console.log(BURUK”);
        break;
    default :
        console.log(“BURUK SEKALI”);
        break;
}
```

For Loop

For Loop

- For Loop adalah salah satu fitur untuk melakukan perulangan di TypeScript
- Terdapat 3 jenis For Loop di TypeScript
  - For Loop
  - For of Loop
  - For in Loop

For Loop

```
for( first; second; third){
    // kode perulangan
}

// first akan dieksekusi sebelum perulangan dilakukan
// second adalah kondisi boolean apakah perulangan harus
terus berjalan atau berhenti
// third akan dieksekusi di setiap akhir kode perulangan
```

For of Loop

```
for( let/var namaVariable of namaArray){
    console.log(namaVariable);
}

// for of loop digunakan untuk melakukan perulangan dengan
data yang ada di setiap array
```

For in Loop

```
for( let/var index in namaArray){
    console.log(namaArray[index]);
}

// for in loop digunakan untuk melakukan perulangan dengan
index yang ada di setiap array
```

While Loop

While Loop

- While Loop merupakan fitur perulangan selain For loop
- While Loop lebih sederhana dan flexible dibanding For Loop, karena aturan pembuatan While Loop lebih sederhana
- Ada 2 jenis While Loop di TypeScript :
    - While Loop
    - Do While Loop

While Loop

```
while(kondisi boolean){
    // kode perulangan
}
```

Do While Loop

```
do {
    // kode perulangan
} while (kondisi boolean);
```

Interface

Apa itu Interface

- Interface di TypeScript mungkin sedikit berbeda dengan Interface di bahasa pemrograman lain
- Pada TypeScript, interface biasanya digunakan sebagai kontrak dalam kode program yang kita buat

Membuat Interface

```
interface Mahasiswa {
    nim: string;
    name: string;
}
```

Menggunakan Interface

```
interface Mahasiswa {
    nim: string;
    name: string;
}

let mahasiswa1: Mahasiswa = {
    nim: "1001",
    name: "Eko"
};
```

Optional Property di Interface

Properties di Interface Wajib Dibuat

```
interface Mahasiswa {
    nim: string;
    name: string;
}

let mahasiswa1: Mahasiswa = {
    nim: "1001"
};

// error karena property name tidak dibuat
```

Optional Properties

```
interface Mahasiswa {
    nim: string;
    name?: string;
}

let mahasiswa1: Mahasiswa = {
    nim: "1001"
};
```

Readonly Property di Interface

Properties di Interface Wajib Dibuat

```
interface Mahasiswa {
    nim: string;
    name: string;
}

let mahasiswa1: Mahasiswa = {
    nim: "1001"
};

mahasiswa.nim = “1002”; // sukses
```

Readonly Properties

```
interface Mahasiswa {
    readonly nim: string;
    name: string;
}

let mahasiswa1: Mahasiswa = {
    nim: "1001"
};

mahasiswa.nim = “1002”; // error
```

Function Interface

Function Variable di JavaScript

```
var sayHello = function(name){
    return “Hello ” + name;
}

sayHello(“Eko”); // sukses
sayHello(); // sukses
```

Function Interface

```
interface SayHello {
    (name: String): string;
}

let hello: SayHello = function (name: string) {
    return `Hello ${name}`;
};

hello("Eko"); // sukses
hello(); // error
```

Indexable Interface

Indexable Number Interface

```
interface StringArray {
    [index: number]: string;
}

let array: StringArray = ["Eko", "Kurniawan", "Khannedy"];
```

Indexable String Interface

```
interface NumberArray {
    [property: string]: number
}

let numbers: NumberArray = {
    "satu": 1,
    "dua": 2,
    "tiga": 3
};
```

Indexable Dictionary Interface

```
interface MahasiswaDictionary {
    [property: string]: number | string;

    nim: string;
    name: string;
    value: number;
}

let mahasiswa: MahasiswaDictionary = {
    nim: "1001",
    name: "Eko",
    value: 100
};
```

Extending Interface

Interface Bisa Extends Interface Lain

```
interface HasName {
    name: string
}

interface HasAddress extends HasName {
    address: string
}
```

Dan Bisa Extend Banyak Interface

```
interface HasName {
    name: string
}

interface HashAge {
    age: number
}

interface HasAddress extends HasName, HashAge {
    address: string
}
```

Yang Di-Extend Wajib Implementasikan

```
let person: HasAddress = {
    name: "Eko",
    age: 30,
    address: "Indonesia"
};
```

Function di Interface

Function di Interface

```
interface Human {
    name: string;
    sayHello(name: string): string;
}

let human: Human = {
    name: "Eko",
    sayHello(name: string): string {
        return `Hello ${name}, my name is ${this.name}`;
    }
};
```

Hybrid Interface

Apa itu Hybrid Interface?

- Hybrid Interface artinya Interface yang bisa merepresentasikan Object dan juga Function
- Karena JavaScript bisa sangat dynamic dan flexible, maka tak jarang kita suka menemui kombinasi Objek dan Function di JavaScript
- Hybrid Interface ini dapat digunakan agar hal ini bisa lebih aman

Deklarasi Hybrid Interface

```
interface Executor {
    (name: string): string;

    success: boolean
}
```

Membuat Hybrid Interface

```
let executor: Executor = (function (name: string): string {
    return "Executed";
}) as Executor;

executor.success = true;
```

Function

Apa itu Function?

- Function adalah sekumpulan unit dari kode program
- Function biasanya memiliki nama, input dan juga output
- Cara pembuatan function di TypeScript hampir sama dengan di JavaScript

Contoh Function Sederhana

```
function sum(x, y){

    let total = x + y;

    return total;

}

console.log(sum(10, 10));
```

Function Types

Function Types

- Biasanya dalam JavaScript, programmer jarang sekali menyebutkan tipe data dari input dan output sebuah Function
- Di TypeScript walaupun tidak wajib, tapi sangat direkomendasikan untuk menyebutkan tipe data dari input dan output sebuah Function

Contoh Function Types

```
function sum(x: number, y: number): number {

    let total: number = x + y;

    return total;

}

console.log(sum(10, 10));
```

Optional dan Default Parameter

Optional dan Default Parameter

- Kadang dalam input Function, ada beberapa parameter yang optional atau tidak wajib untuk diberi nilai.
- Dan kadang pula dalam input Function, ada beberapa parameter yang memiliki nilai default jika tidak dimasukkan.
- TypeScript mendukung Optional dan Default Parameter dalam Function

Optional Parameter

```
function fullName(firstName: string, lastName?: string) : string {
    if(lastName){
        return firstName + “ ” + lastName;
    } else {
        return firstName;
    }
}

fullName(“Eko”);
fullName(“Eko”, “Kurniawan”);
```

Default Parameter

```
function hello(name: string = "World") {
    return `Hello ${name}`;
}

hello();
hello("Eko");
```

Default Parameter (2)

```
function hello(title: string = “Mr.” name: string) {
    return `Hello ${title} ${name}`;
}

hello(“Miss.”, “Sarah”);
hello(undefined, "Eko");
```

Rest Parameter

Rest Parameter

- Secara default, parameter di Function hanya bisa digunakan untuk satu input.
- Jika ingin menerima input lebih dari satu, maka kita harus membuat sejumlah parameter nya.
- Rest Parameter adalah kemampuan menerima lebih dari satu input dalam satu parameter.
- Rest Parameter hanya dapat dibuat di posisi paling akhir sebuah parameter

Contoh Rest Parameter

```
function sum(initial: number, ...numbers: number[]): number {
    let total: number = initial;
    for (let number of numbers) {
        total = total + number;
    }
    return total;
}

sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
sum(...[1, 2, 3]);
```

this

Penggunaan this

- Kata kunci this biasanya digunakan untuk mengakses pihak dimana function dipanggil

Type	this
function(...)	window / global
path.to.obj.function(...)	path.to.obj
obj.function(...)	obj

Contoh this di Function

```
function callThis() {

    console.log(this);

}

callThis(); // print window atau global
```

Contoh this di Function Object

```
let student: object = {
    "nim": "1001",
    "hello": function () {
        console.log(this);
    }
};

student["hello"](); // print student
```

Contoh this di Function Nested Object

```
let student: object = {
    "nim": "1001",
    "parent": {
        "name": "Eko",
        "hello": function () {
            console.log(this);
        }
    }
};

student["parent"]["hello"](); // print student.parent
```

Function Overloading

Function Overloading

- Function Overloading adalah kemampuan untuk membuat Function dengan nama yang sama, namun dengan parameter input yang berbeda
- Di JavaScript, membuat satu Function dengan input data yang berbeda dan output yang bisa menghasil data yang berbeda sudah biasa dilakukan
- Namun hal ini kadang bisa membuat sebuah Function tidak akan, karena bisa menghasilkan berbeda-beda tipe data.
- TypeScript memberikan kemampuan Function Overloading, agar pembuatan Function seperti ini lebih aman dilakukan

Contoh Function Overloading (Tidak Aman)

```
function sample(a) {
    if (typeof a == 'number') {
        return a * 10;
    } else if (typeof a == 'string') {
        return `Hello ${a}`;
    }
}

sample(10);
sample("Eko");
```

Contoh Function Overloading (Aman)

```
function sample(value: number) : number;
function sample(name: string) : string;
function sample(a) {
    if (typeof a == 'number') {
        return a * 10;
    } else if (typeof a == 'string') {
        return `Hello ${a}`;
    }
}
```

Function Sebagai Parameter

Function Sebagai Parameter (Tidak Aman)

```
function sample(sayHello) {
    console.info(sayHello("World"));
}

function sayHello(name: string) : string {
     return `Hello ${name}`;
}

sample(sayHello);
sample("Eko"); // compile sukses, runtime error
```

Function Sebagai Parameter (Aman)

```
function sample(sayHello: (name: string) => string) {
    console.info(sayHello("World"));
}

function sayHello(name: string) : string {
     return `Hello ${name}`;
}

sample(sayHello);
sample("Eko"); // compile error
```

Anonymous Function

Function sebagai Parameter

```
function sample(sayHello: (name: string) => string) {
    console.info(sayHello("World"));
}

function sayHello(name: string) : string {
     return `Hello ${name}`;
}

sample(sayHello);
```

Anonymous Function

```
function sample(sayHello: (name: string) => string) {
    console.info(sayHello("World"));
}

sample(function(name: string): string {
     return `Hello ${name}`;
});
```

Anonymous Function (2)

```
function sample(sayHello: (name: string) => string) {
    console.info(sayHello("World"));
}

let sayHello = function(name: string): string {
     return `Hello ${name}`;
};

sample(sayHello);
```

Arrow Function

Anonymous Function dengan Arrow

```
function sample(sayHello: (name: string) => string) {
    console.info(sayHello("World"));
}

sample((name: string): string => {
    return `Hello ${name}`;
});
sample((name: string): string => `Hello ${name}`);
sample((name) => `Hello ${name}`);
```

Anonymous Function dengan Arrow (2)

```
function sample(sayHello: (name: string) => string) {
    console.info(sayHello("World"));
}

let sayHello = (name) => `Hello ${name}`;
sample(sayHello);
```

Masalah Dengan this

```
let masalah: object = {
    name: "Eko",
    createSayHello: function () {
        return function (name: string): string {
            return `Hai ${name}, Nama Saya ${this.name}`;
        }
    }
};

let sayHello = masalah["createSayHello"]();
console.log(sayHello("Joko"));
```

Memperbaiki this Dengan Arrow Function

```
let masalah: object = {
    name: "Eko",
    createSayHello: function () {
        return (name: string): string => {
            return `Hai ${name}, Nama Saya ${this.name}`;
        }
    }
};

let sayHello = masalah["createSayHello"]();
console.log(sayHello("Joko"));
```

Class

Pemrograman Berorientasi Objek

- Tradisional JavaScript menggunakan Function dan Prototype untuk membuat component yang bisa digunakan.
- Agak sedikit aneh jika programmer terbiasa dengan pemrograman berorientasi objek.
- Pada pemrograman berorientasi objek, biasa menggunakan Class sebagai prototype dan Object sebagai hasil pembuatan dari Class

Apa itu Class?

- Class merupakan kerangka atau prototype Object yang akan dibuat.
- Class biasanya berisikan attribute/property dan juga function
- Object adalah hasil pembuatan/instansiasi dari Class
- Object dapat dibuat berkali-kali dari Class yang sama, dan setiap Object yang dihasilkan adalah independent

Membuat Class dan Object

```
class Student {
}

let student1: Student = new Student();
let student2: Student = new Student();
```

Property pada Class

Property pada Class

- Class biasanya memiliki tempat untuk menyimpan data, biasa disebut dengan property atau attribute.
- Property pada class bisa beruba tipe data primitive, array bahkan object lain

Property pada Class

```
class Address {
    street: string;
    country: string;
}

class Student {
    nim: string;
    name: string;
    address: Address;
}
```

Mengubah Nilai Property pada Class

```
let student: Student = new Student();
student.nim = "1001";
student.name = "Eko";

let address: Address = new Address();
address.street = "Jalan xxx";
address.country = "Indonesia";

student.address = address;
```

Function pada Class

Function pada Class

- Selain Property, yang biasa disimpan dalam class adalah Function
- Function yang berhubungan dengan Class tersebut, biasanya dibuat dalam Class itu sendiri

Function pada Class

```
class Person {
    name: string;
    sayHello(): string {
        return `Hello, My Name is ${this.name}`;
    }
}

let person: Person = new Person();
person.name = "Eko";
console.log(person.sayHello());
```

Constructor pada Class

Apa itu Constructor?

- Constructor adalah Function yang bisa kita paksa untuk dipanggil ketika sebuah Object dibuat dari Class
- Biasanya Constructor digunakan untuk memaksa programmer untuk memasukkan data yang wajib diisi

Constructor pada Class

```
class Person {
     name: string;

     constructor(name: string){
          this.name = name;
     }
}

let person: Person = new Person(“Eko”);
```

Readonly Property pada Class

Readonly Property pada Class

- Class juga mendukung Readonly Property
- Readonly Property memastikan bahwa property tidak bisa diubah lagi
- Readonly Property harus diisi nilainya pada saat deklarasi Property atau di Constructor

Readonly Property pada Class

```
class Student {
     readonly nim: string
     name: string;
     constructor(nim: string, name: string){
          this.nim = nim;
          this.name = name;
     }
}

let person: Person = new Person(“10001”, “Eko”);
person.nim = “xxx”; // error
```

Inheritance pada Class

Inheritance

- Inheritance atau pewarisan adalah salah satu fitur yang sudah biasa ada di pemrograman berorientasi objek
- Inheritance adalah kemampuan sebuah Class meng-extends Class lain.
- Dengan meng-extends Class lain, secara otomatis kemampuan yang ada di Class lain bisa didapat juga di Class yang meng-extends nya.

Inheritance pada Class

```
class Flyable {
    fly(): void {
        console.log("I can fly");
    }
}

class Bird extends Flyable {
}

let bird: Bird = new Bird();
bird.fly();
```

Overriding pada Class

Overriding pada Class

- Overriding merupakan kemampuan mendeklarasikan ulang Property atau Function yang telah ada di parent Class
- Dengan mendeklarasikan ulang Property atau Function, secara otomatis akses ke parent Class harus menggunakan kata kunci super.

Overriding pada Class

```
class Flyable {
    name: String = "Flyable";
    fly(): void {
        console.info("I can fly");
    }
}

class Bird extends Flyable {
    fly(): void {
        super.fly();
        console.log(`${this.name} can fly`);
    }
}
```

Modifier pada Class

Abstract Class

Class Implement Interface

Class Implement Interface

https://www.typescriptlang.org/docs/handbook/interfaces.html#class-types

Interface Extend Class

Interface Extend Class

https://www.typescriptlang.org/docs/handbook/interfaces.html#interfaces-extending-classes

Materi Selanjutnya

Materi Selanjutnya

- TypeScript Object Oriented Programming

Kontak

- Eko Kurniawan Khannedy
- Telegram : @khannedy
- Facebook : fb.com/khannedy
- Instagram : instagram.com/programmerzamannow
- Email : echo.khannedy@gmail.com