https://www.edureka.co/blog/interview-questions/typescript-interview-questions/

Mi a különbség a TypeScript és a JavaScript között?

TypeScript	JavaScript
A TypeScript egy objektum-orientált nyelv	A JavaScript egy szkript nyelv
Statikus típus (static typing) támogatása	Nincs statikus tipizálása
A TypeScript támogatja a modulokat	A JavaScript nem támogatja a modulokat
Támogatja az opcionális parameter a függvényeknél	Nem támogatja az opcionális paraméter a függvényeknél

Mi a TypeScript?

A TypeScript egy típusos JavaScript-készlet, amely egyszerű JavaScript-re forul le. objektum-orientált osztályokkal, interfészekkel és statikus típusokkal rendelkezik. A kód fordításához és létrehozásához a JavaScript fájlban egy fordítóra lesz szükség. Alapvetően a TypeScript a JavaScript ES6 verziója, néhány további funkcióval.

```
var message:string = "Welcome to Edureka!"
console.log(message)
```

A TypeScript kódot egy .ts kiterjesztésű fájlba írják, majd a fordító segítségével JavaScriptbe fordítják. Bármelyik kódszerkesztőbe beírhatja a fájlt, és a fordítót telepíteni kell a platformjára. A telepítés után a tsc <fájlnév> .ts parancs lefordítja a TypeScript kódot egy egyszerű JavaScript fájlba.

Miért van szükségünk TypeScript-re?

Különböző okok vannak arra, hogy egy JavaScript-fejlesztőnek fontolóra kell vennie a TypeScript használatát. Néhány közülük a következőket tartalmazza:

1.) Az ECMAScript új szolgáltatásainak használata: A TypeScript támogatja az új ECMAScript szabványokatTehát használhatja az ES2015 és a továbbiak szolgáltatásait.

2.) Statikus típus (static typing): A JavaScript dinamikus típusú, és csak akkor tudja meg, hogy mi a változó, amíg a futás közben ténylegesen nem példányosítják. A TypeScript típus-támogatást ad a JavaScript-hez.

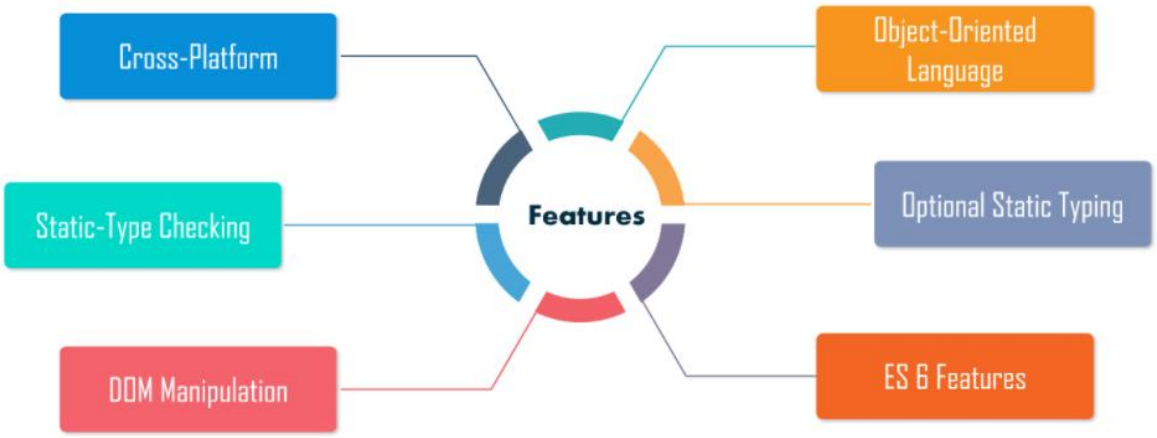
3.) Típuskövetkeztetés (Type Inference): A TypeScript egy kicsit megkönnyíti a típuskövetkeztetés használatát. Még ha nem is írja be kifejezetten a típusokat, akkor is ott vannak, hogy megmentsék Önt egy olyan cselekedettől, amely egyébként futtatási hibát eredményezne.

4.) Jobb IDE támogatás: A TypeScript fejlesztési tapasztalata nagy előrelépés a JavaScript-hez képest. Az IDE-k széles skálája van, amelyek kiválóan támogatják a TypeScript-t, például a Visual Studio és VS kód, az Atom, a Sublime és az IntelliJ / WebStorm.

5.) Szigorú nullellenőrzés (Strict Null Checking): A hibák, mint például a undefined 'x' tulajdonságának beolvasása, gyakran előfordulnak a JavaScript programozásban. Az ilyen típusú hibák többségét elkerülheti, mivel nem lehet olyan változót használni, amelyet a TypeScript fordító nem ismer.

6.) Interoperabilitás (Interoperability): A TypeScript szorosan kapcsolódik a JavaScript-hez, így nagyszerű interoperabilitási képességekkel rendelkezik, de némi extra munkára van szükség ahhoz, hogy a JavaScript-könyvtárak működjenek a TypeScript-ben.

Említse meg a TypeScript néhány jellemzőjét



1.) Cross-platform: A TypeScript fordító bármilyen operációs rendszerre telepíthető, például Windows, MacOS és Linux.

2.) Objektum-orientált nyelv: A TypeScript olyan szolgáltatásokat nyújt, mint az osztályok, az interfészek és a modulok. Így képes objektum-orientált kódot írni kliens- és szerveroldali fejlesztésre.

3.) Statikus típusellenőrzés: A TypeScript statikus típust használ, és segíti a típusellenőrzést fordításkor. Így hibákat találhat a kód írása közben a parancsfájl futtatása nélkül.

4.) Opcionális statikus típus (Optional Static Typing): A TypeScript opcionális statikus típus is engedélyez, ha JavaScript dinamikus típust használja.

5.) DOM-manipuláció: A TypeScript segítségével manipulálhatja a DOM-ot elemek hozzáadásához vagy eltávolításához.

6.) ES 6 jellemzők: A TypeScript tartalmazza a tervezett ECMAScript 2015 (ES 6, 7) legtöbb funkcióját, például osztályt, interfészt, arrow függvényeket stb.

Milyen előnyei vannak a TypeScript használatának?

1.) A TypeScript gyors, egyszerű, könnyen megtanulható, és bármilyen böngészőn vagy JavaScript-motoron fut.

2.) Hasonló a JavaScript-hez, és ugyanazt a szintaxist és szemantikát használja.

3.) Ez segít a backend-fejlesztőknek a front-end kód gyorsabb írásában.

4.) Meghívhatja a TypeScript kódot egy meglévő JavaScript kódból. Emellett a meglévő JavaScript keretrendszerekkel és könyvtárakkal probléma nélkül működik.

5.) A definíciós fájl .d.ts kiterjesztéssel támogatja a meglévő JavaScript könyvtárakat, például a Jquery, a D3.js stb.

6.) Az ES6 és az ES7 olyan szolgáltatásait tartalmazza, amelyek olyan ES5 szintű JavaScript motorokban futtathatók, mint a Node.js.

Melyek a TypeScript hátrányai?

A TypeScript a következő hátrányokkal jár:

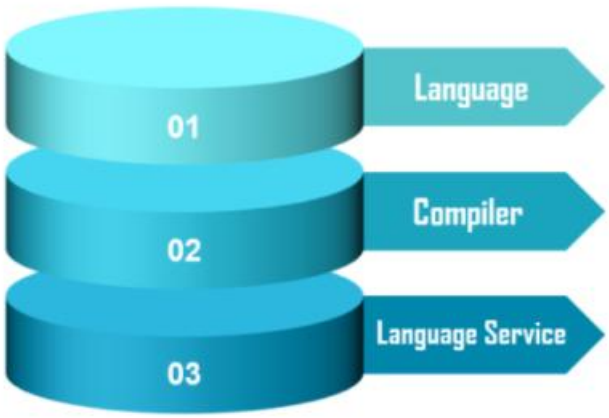
1.) A TypeScript hosszú időbe telik a kód lefordítása.

2.) Nem támogatja az absztrakt osztályokat.

3.) Ha a TypeScript alkalmazást futtatjuk a böngészőben, akkor a fordítási lépésre szükség van a TypeScript JavaScript-vé történő átalakításához.

4.) Bármely harmadik fél könyvtárának használatához a definíciós fájl elengedhetetlen.

Melyek a TypeScript összetevői?



Nyelv - A szintaxist, a kulcsszavakat és a típus annotációkat tartalmazza.

A TypeScript fordító - Ez a fordító (tsc) konvertálja a TypeScript-ben írt utasításokat JavaScript megfelelőjévé.

A TypeScript nyelvszolgáltatás (language service) - A nyelvi szolgáltatás támogatja a tipikus szerkesztőműveletek közös halmazát.

Hogyan kell telepíteni a TypeScript-t?

```
npm install -g typescript
```

Hogyan fordítja le a TypeScript fájlokat?

Bármely TypeScript fájl kiterjesztése .ts. És minden JavaScript-fájl TypeScript-fájl, mivel a JavaScript egy halmaza. Tehát, ha a „.js” kiterjesztését „.ts” -re változtatja, a TypeScript fájl készen áll. Bármely .ts fájl fordítása .js fájlba a következő paranccsal használható:

```
tsc <TypeScript File Name>
```

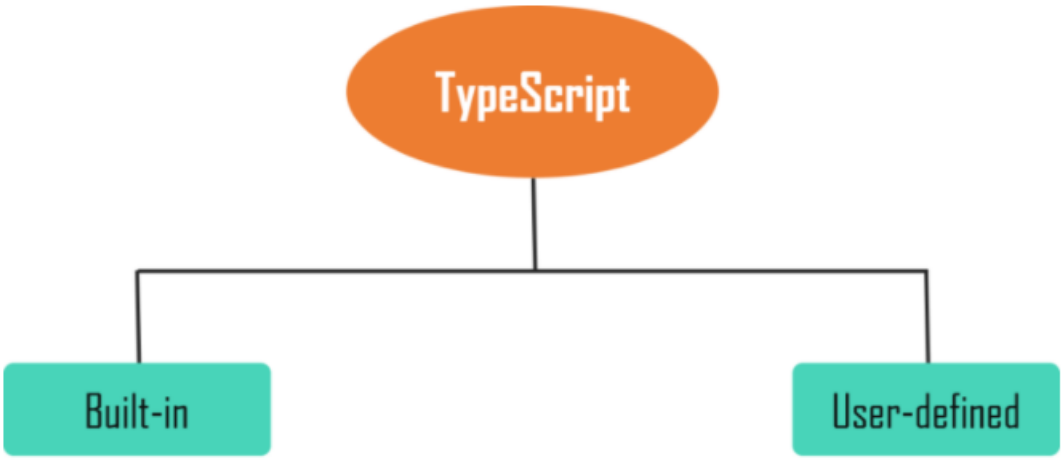
Összeállíthatunk több .ts fájlt egyetlen .js fájlba?

Igen, több fájlt kombinálhatunk. Fordítás közben hozzá kell adnunk az –outFILE [OutputJSFileName] opciót.

```
tsc --outFile comman.js file1.ts file2.ts file3.ts
```

Melyek a TypeScript különböző típusai?

A Type System a nyelv által támogatott különböző típusú értékeket képviseli. Ellenőrzi a megadott értékek érvényességét, mielőtt azokat a program eltárolná vagy kezelné.



Beépített: ide tartozik a szám, a karakterlánc, a logikai érték, az érvénytelen, a null és a undefined.

Felhasználó által definiált: Magában foglalja az enumerations-t (felsorolásokat), osztályokat, interfészeket, tömböket és tuple.

Sorolja fel a beépített adattípusokat a TypeScript-ben.

A TypeScriptben a beépített adattípusokat primitív adattípusoknak is nevezik, és a lista a következőket tartalmazza:

1.) Number: Ez a számtípus értékeit jelöli. A számokat lebegőpontos értékként tároljuk a TypeScript-ben.

2.) String: A karakterlánc Unicode UTF-16 kódként tárolt karakterek sorozatát jelenti.

3.) Boolean: Ez logikai értéket képvisel. A logikai típust használva a kimenetet csak igaz vagy hamis értékben kapjuk meg.

4.) Null: A Null egy olyan változót jelent, amelynek értéke nincs meghatározva. Magára a null típusú értékre nem lehet közvetlenül hivatkozni.

5.) Undefined: A undefined típus az összes inicializálatlan változót jelöli.

6.) Void: A void a függvények visszatérési típusa, amelyek nem adnak vissza semmilyen típusú értéket.

Milyen módon lehet deklarálni a változót?

A változó deklarálásának négy módja van:

```
1  var [identifier] : [type-annotation] = value; //Declaring type and
                      value in a single statement
```

```
var [identifier] : [type-annotation]; //Declaring type without value
var [identifier] = value; //Declaring its value without type
var [identifier]; //Declaring without value and type
```

Lehetséges-e a .ts automatikus fordítása valós idejű változtatásokkal a .ts fájlban?

Igen, automatikusan összeállíthatjuk a .ts fájlt valós idejű változtatásokkal a .ts fájlban. Ezt a –watch compiler opcióval teheti meg:

```
tsc --watch file1.ts
```

A fenti parancs először a file1.ts fájlt fordítja le a file1.js fájlba, és figyeli a fájl változását. Ha bármilyen változást észlelnek, akkor újra lefordítja a fájlt. Itt biztosítanunk kell, hogy a –watch opcióval történő futtatáskor a parancssor ne legyen bezárva.

Milyen objektum-orientált kifejezéseket támogat a TypeScript?

A TypeScript a következő objektum-orientált kifejezéseket támogatja:

- **Modules**
- **Classes**
- **Interfaces**
- **Inheritance**
- **Data Types**
- **Member functions**

Mik az interfészek a TypeScript-ben?

Meghatározza a követendő osztályok szintaxisát. Csak a tagok deklarációját tartalmazza, és a tagok meghatározása az implementáló osztály feladata. A TypeScript fordító interfészt használ a típusellenőrzéshez, és ellenőrzi, hogy az objektumnak van-e meghatározott szerkezete vagy sem.

```
interface interface_name {
// variables' declaration
// methods' declaration
}
```

Mik az osztályok a TypeScript-ben? Soroljon fel néhány funkciót.

A TypeScript olyan osztályokat vezetett be, hogy kihasználhassák az objektumorientált technikák előnyeit, mint a beágyazás és az absztrakció. A TypeScript-osztályt a TypeScript-fordító egyszerű JavaScript-függvényekké állítja össze, hogy platformokon és böngészőkön keresztül működjön.

Egy osztály a következőket tartalmazza:

- **Constructor**
- **Properties**

- **Methods**

```
class Employee {
empID: number;
empName: string;

constructor(ID: number, name: string) {
this.empName = name;
this.empID = ID;
}

getSalary() : number {
return 40000;
}
}
}
```

- **Inheritance**
- **Encapsulation**
- **Polymorphism**
- **Abstraction**

Milyen hozzáférés-módosítókat támogat a TypeScript?

A TypeScript támogatja a public, private és protected hozzáférés-módosítókat, amelyek az alábbiakban meghatározzák egy osztálytag hozzáférhetőségét:

- public - Az osztály minden tagja, gyermekosztályai és minden más osztály hozzáférhet.
- protected- Az osztály és annak gyermekosztályainak minden tagja hozzáférhet hozzájuk
- private - Csak az osztály tagjai férhetnek hozzá.

Ha egy hozzáférés-módosító nincs megadva, az implicit módon public, mivel megfelel a JavaScript kényelmes jellegének.

Hogyan lehet a TypeScript opcionálisan statikusan típus nyelv?

A TypeScriptet opcionálisan statikusan tipizáltnak nevezik, ami azt jelenti, hogy megkérheti a fordítót, hogy hagyja figyelmen kívül a változó típusát.Az any adattípus felhasználásával bármilyen típusú értéket rendelhetünk a változóhoz. A TypeScript nem fog hibát ellenőrizni a fordítás során.

```
var unknownType: any = 4;
unknownType = "Welcome to Edureka"; //string
unknownType = false; // A boolean.
```

Mik a modulok a TypeScript-ben?

A modul a kapcsolódó változók, függvények, osztályok és interfészek stb. csoportjának létrehozásának hatékony módja. Futtatható a saját hatókörén belül, de nem a globális hatókörön belül. Alapvetően nem férhet hozzá közvetlenül a modulon kívül deklarált változókhoz, függvényekhez, osztályokhoz és interfészekhez.

Modul létrehozható az export kulcsszó használatával, és más modulokban is használható az import kulcsszó használatával.

```
module module_name{
class xyz{
export sum(x, y){
return x+y;
}
}
```

Mi a különbség a belső modul és a külső modul között?

Internal Module	External Module
A belső modulok az osztályokat, interfészeket, függvényeket, változókat egyetlen egységbe csoportosítják, és exportálhatók egy másik modulba.	A külső modulok hasznosak a moduldefiníciók belső utasításainak elrejtésében, és csak a deklarált változóhoz tartozó módszereket és paramétereket mutatják be.
A belső modulok a Typescript korábbi verziójának részét képezték.	A külső modulok a legújabb verzióban modulként ismertek.
Ezek más modulok helyi (lokális) vagy exportált tagjai.	Ezek külön betöltött kódtestek, amelyekre külső modulnevek hivatkoznak.
A belső modulokat a ModuleDeclarations segítségével deklarálják, amelyek megadják a nevüket és a testüket.	A külső modul külön forrásfájlként van megírva, amely legalább egy import vagy export utasítást tartalmaz

Mi a névtér a typescript-ben, és hogyan deklarálható?

A névtér logikusan csoportosítja a funkciókat. Összefoglalja azokat a tulajdonságokat és objektumokat, amelyeknek közös tulajdonságaik vannak. A névtér belső modulokként is ismert. A névtér tartalmazhat interfészeket, osztályokat, függvényeket és változókat is a kapcsolódó funkciók egy csoportjának támogatásához.

```
namespace <namespace_name> {
export interface I1 { }
export class c1{ }
}
```

A TypeScript a függvények túlterhelését támogatja?

Igen, a TypeScript támogatja a függvények túlterhelését.

```
class Customer {
name: string;
Id: number;
add(Id: number);
add(name:string);
add(value: any) {
if (value && typeof value == "number") {
//Do something
}
if (value && typeof value == "string") {
//Do Something
}
}
```

Hogyan támogatja a TypeScript az opcionális paramétereket a funkcióban?

A JavaScript-sel ellentétben a TypeScript fordító hibát dob, ha megpróbál egy függvényt meghívni anélkül, hogy megadná a függvény szignatúrában megadott pontos számú és típusú paramétert. A probléma leküzdéséhez választható paramétereket használhat kérdőjel (‘?’) használatával.

```
function Demo(arg1: number, arg2? :number) {
}So, arg1 is always required, and arg2 is an optional parameter.
```

Mi a Scope változó?

A hatókör (scope) objektumok, változók és függvények halmaza, és a JavaScript rendelkezhet globális hatókörű változóval és helyi hatókörű változóval.

Deklarálhat egy változót két különböző hatókörben, például:

- Helyi hatókörű változó (Local Scope Variable) - Ez egy függvényobjektum, amelyet a függvényen belül használnak
- Globális hatókörű változó (Global Scope Variable)- Ezt a window objektumot függvényeken kívül és a függvényeken belül is használhatja

Hogyan debug-olhat egy TypeScript fájl?

Bármely TypeScript fájl hibakereséséhez szüksége van egy .js sourcemap fájlra. Tehát le kell fordítania a .ts fájlt a –sourcemap kapcsolóval.

```
tsc -sourcemap file1.ts
```

Mi az a TypeScript Definition Manager, és miért van rá szükségünk?

A TypeScript Definition Manager (TSD) egy csomagkezelő (package manager), amelyet a TypeScript definíciós fájlok keresésére és telepítésére használnak, közvetlenül a közösség által vezérelt DefinitelyTyped adattárból.

Ha valamilyen jQuery kódot szeretne használni a .ts fájljában:

```
$(document).ready(function() { //Your jQuery code });
```

Itt, amikor megpróbálja lefordítani a tsc használatával, fordítási idejű hibát fog kapni: Nem található a „$” név. Tehát tájékoztatnia kell a TypeScript fordítót arról, hogy a „$” a jQuery-hez tartozik. Ehhez a TSD játszik szerepet. Letöltheti a jQuery Type Definition fájlt, és felveheti a .ts fájlba.

Melyek a típusdefiníciós fájl felvételének lépései?

A típusdefiníciós fájl felvételének lépései a következők:

Először telepítenie kell a TSD-t.

```
npm install tsd -g
```

Ezután a TypeScript könyvtárban hozzon létre egy új TypeScript projektet az alábbi paranccsal:

```
tsd init
```

Ezután telepítse a jQuery definíciós fájlját.

```
tsd query jquery --action install
```

A fenti parancs letölt és létrehoz egy új könyvtárat, amely a jQuery definíciós fájlt a „.d.ts” végződéssel. Most adja hozzá a definíciós fájlt a TypeScript fájl frissítésével, hogy a jQuery definícióra mutasson.

```
/// <reference path="typings/jquery/jquery.d.ts" />
$(document).ready(function() { //To Do
});
```

Végül fordítsd újra. Ezúttal a js fájl hiba nélkül jön létre. Ezért a TSD iránti igény segít abban, hogy megszerezzük a szükséges keretrendszer típusdefiníciós fájlját.

Mi az a TypeScript Declare Keyword?

A JavaScript könyvtárakban vagy keretrendszerekben nincsenek TypeScript deklarációs fájlok. De ha fordítási hiba nélkül szeretné használni őket a TypeScript fájlban, akkor a declare kulcsszót kell használnia. A declare kulcsszót a környezeti deklarációkhoz és metódusokhoz használják, ahol meg akar határozni egy máshol létező változót.

Ha a könyvtárat a TypeScript kódunkban szeretné használni, akkor a következő kódot használhatja:

```
declare var myLibrary;
```

A TypeScript a myLibrary változót any típushoz rendeli.

Mi az alapértelmezett paraméterű függvény (Default Parameters Function) a TypeScript-ben?

A függvény paraméterekhez alapértelmezés szerint hozzá lehet rendelni értékeket. Egy paraméter nem deklarálható opcionálisként és alapértelmezettként egyszerre.

```
let discount = function (price: number, rate: number = 0.40) {
return price * rate;
}
discount(500); // Result - 200
discount(500, 0.45); // Result - 225
```

Mi az a tsconfig.json fájl?

A tsconfig.json fájl JSON formátumú fájl. A tsconfig.json fájlban különböző opciókat adhat meg, hogy a fordító megmondja, hogyan kell lefordítani az aktuális projektet. A tsconfig.json fájl jelenléte a könyvtárban azt jelzi, hogy a könyvtár a TypeScript projekt gyökere.

```
"compilerOptions": {
"declaration": true,
"emitDecoratorMetadata": false,
"experimentalDecorators": false,
"module": "none",
"moduleResolution": "node"
"removeComments": true,
"sourceMap": true
},
"files": [
"main.ts",
"othermodule.ts"
]
}
```

Mi a Generics (generikusok) a TypeScript-ben?

A TypeScript generikus (Generics) egy olyan eszköz, amely lehetőséget nyújt újrafelhasználható komponensek létrehozására. Képes olyan komponenseket létrehozni, amelyek különféle adattípusokkal működhetnek, nem pedig egyetlen adattípussal. Ezenkívül a típusbiztonságot is biztosítja a teljesítmény vagy a termelékenység veszélyeztetése nélkül. A generikusok lehetővé teszik általános osztályok, általános függvények, általános metódusok és általános interfészek létrehozását.

A generikus típusokban egy típusparamétert írnak a nyitott (<) és a záró (>) zárójel közé, ami erősen típusos gyűjteményekké teszi. Speciális típusú változót használ <T>, amely a típusokat jelöli.

```
function identity<T>(arg: T): T {
return arg;
}
let output1 = identity<string>("edureka");
let output2 = identity<number>( 117 );
console.log(output1);
console.log(output2);
```

Mi a különbség az interfész és a type statements között?

Interface	Type

Egy interfész deklaráció elnevezett objektumtípust vezet be	A típus alias deklaráció bármilyen típusú nevet vezet be, beleértve a primitív, az unió stb típusokat is
Megnevezhető extends vagy implements részben	Az objektumtípus literál típusneve nem nevezhető meg az extends vagy implement -ben
Az interfészek új nevet hoznak létre, amelyet mindenhol használnak	Nem hoznak létre új nevet
Több egyesített deklarációja is lehet	Nem lehet több egyesített deklarációja

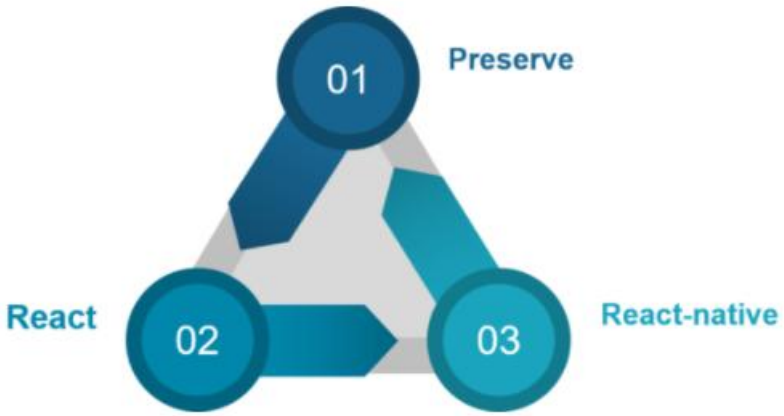
Mi a JSX a TypeScript-ben?

A JSX egy beágyazható XML-szerű szintaxis, amelyet érvényes JavaScript-vé kell átalakítani. A JSX népszerűvé vált a React keretrendszerrel. A TypeScript támogatja a beágyazást, a típusellenőrzést és a JSX fordítását közvetlenül a JavaScript-be.

Ha a JSX-t szeretné használni a fájljában, akkor a fájlját .tsx kiterjesztéssel kell elneveznie, és engedélyeznie kell a jsx opciót.

Mi az összes JSX mód, amelyet a TypeScript támogat?

A TypeScript három JSX módból áll:



Mi az a TypeScript Map fájl?

A TypeScript Map fájl egy forrástérképfájl, amely információkat tárol az eredeti fájljainkról. A .map fájlok olyan forrástérkép fájlok, amelyek segítségével az eszközök leképezhetik a kibocsátott JavaScript kódot és az azt létrehozó TypeScript forrásfájlokat. Ezenkívül a debuger ismerik ezeket a fájlokat, így a TypeScript fájlt hibakereshetjük a JavaScript fájl helyett.

Mi a Type assertion a TypeScriptben?

A type assertion ugyanúgy működik, mint a más nyelveken történő tipizálás, de nem hajt végre más nyelveken, például a C # és a Java-ban, az adatok típusellenőrzését vagy szerkezetátalakítását. A tipizálás futásidejű támogatással jár, míg a típus állításának nincs hatása a futási időre. A típusú állításokat azonban pusztán a fordító használja, és tippeket ad a fordítónak arra vonatkozóan, hogyan akarjuk elemezni a kódunkat.

```
let empCode: any = 007;
let employeeCode = <number> code;
console.log(typeof(employeeCode)); //Output: number
```

Mi a Rest paraméter?

A Rest paraméter arra szolgál, hogy nulla vagy több értéket adjon át egy függvénynek. A paraméter előtti hárompontos karakterek (‘…’) előtagjával deklarálható. Lehetővé teszi, hogy a függvények változó számú argumentummal rendelkezzenek az argumentum objektum használata nélkül. Nagyon hasznos, ha meghatározatlan számú paraméterünk van.

Milyen szabályok vonatkoznak a Rest paraméterek deklarálására? Adj egy példát.

A rest paraméterben követendő szabályok:

- Egy függvény csak egy rest paraméter megengedett.
- Tömbtípusúnak kell lennie.
- A paraméterlista utolsó paraméterének kell lennie

```
function sum(a: number, ...b: number[]): number {
let result = a;
for (var i = 0; i < b.length; i++) {
result += b[i];
}
console.log(result);
}
let result1 = sum(2, 4);
let result2 = sum(2,4,6,8);
```

Mi a „as” szintaxis a TypeScript-ben?

Az „as” a TypeScript-ben a Type állítás további szintaxisa. Az as-szintaxis bevezetésének oka az, hogy az eredeti szintaxis ütközött a JSX-szel.

```
let empCode: any = 007;
let employeeCode = code as number;
```

A TypeScript és a JSX együttes használata során csak as-style állítások engedélyezettek.

Magyarázza el az Enumot a TypeScript-ben.

A Enums or enumerations egy TypeScipt adattípus, amely lehetővé teszi számunkra a elnevezett konstans készletének meghatározását. A kapcsolódó értékek gyűjteménye lehet numerikus vagy string érték.

```
enum Gender {
Male,
Female
Other
}
console.log(Gender.Male); // Output: 0
//We can also access an enum value by it's number value.
console.log(Gender[1]); // Output: Female
```

Magyarázza el a Relatív és Nem relatív modulimportálást.

Nem Relative	Relative

A nem relatív importálás felolható a baseUrl-hez képest vagy útvonal-hozzárendeléssel. Más szavakkal, nem relatív utakat használunk bármely külső függőségünk importálásakor.
 	
Example:
 	
`import * as $ from "jquery";`
`import { Component } from "@angular/core";`
	A relatív import felhasználható saját moduljainkhoz, amelyek garantáltan megtartják relatív helyüket futás közben. A relatív import a /, ./ vagy ../ karakterekkel kezdődik.
	
	Example:
	
	`import Entry from "./components/Entry";`
	`import {DefaultHeaders} from "../constants/http";`

Mi a metódus felüldefiniálás a TypeScript-ben?

Ha az alosztálynak vagy a gyermek osztálynak ugyanaz a metódusa van, mint amelyet a szülő osztályban deklaráltunk, akkor ezt metódus felülírásnak nevezzük. Alapvetően újradefiniálja az alaposztályos módszereket a gyermekosztályban.

A módszer felüldefiniálásának szabályai:

- A metódusnak ugyanazzal a névvel kell rendelkeznie, mint a szülő osztályban
- Ugyanazon paraméterrel kell rendelkeznie, mint a szülő osztályban.
- Legyen IS-A kapcsolat vagy öröklés.