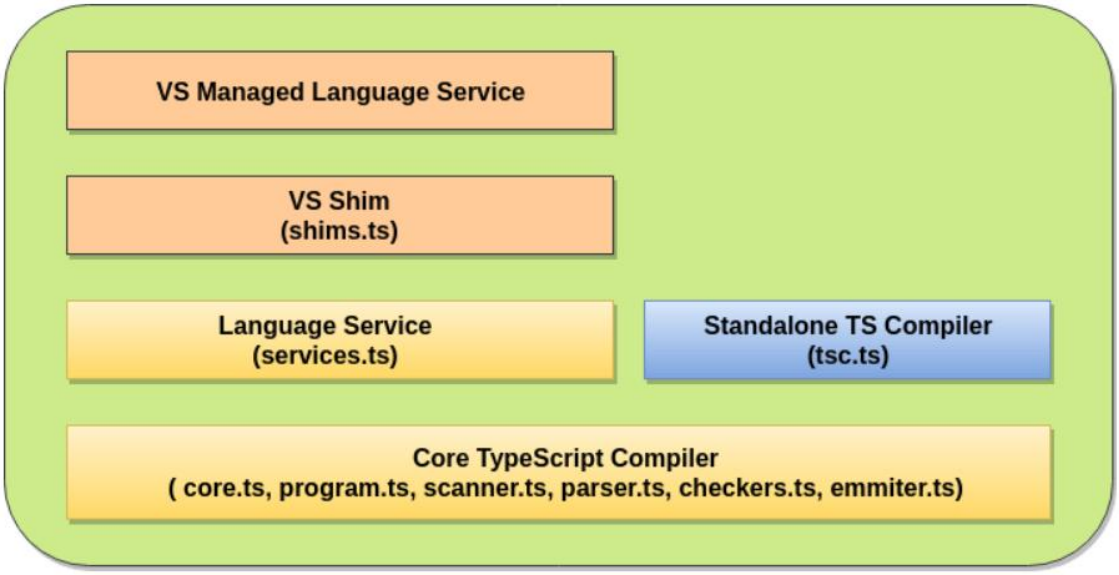
https://www.javatpoint.com/typescript-interview-questions

Miben különbözik a TypeScript a JavaScript-től?

SN	JavaScript	TypeScript
1	A Netscape fejlesztette ki 1995-ben.	Anders Hejlsberg fejlesztette ki 2012-ben.
2	A JavaScript forrásfájl ".js" kiterjesztésű.	A TypeScript forrásfájl ".ts" kiterjesztésű.
3	A JavaScript nem támogatja az ES6-ot.	A TypeScript támogatja az ES6-ot.
4	Nem támogatja az erősen típusos vagy statikustípust.	Támogatja az erősen típusos vagy statikus típust.
5	Ez csak egy szkriptnyelv.	Támogatja az objektum-orientált programozási koncepciókat, mint például osztályok, interfészek, öröklés, generikusok stb.
6	A JavaScript-nek nincs opcionális paraméterfunkciója.	A TypeScript opcionális paraméter funkcióval rendelkezik.
7	Interpretált nyelv, ezért futás közben mutatja csak a hibákat.	A fordítási idő alatt jelzi a hibákat
8	A JavaScript nem támogatja a modulokat.	A TypeScript támogatja a modulokat.
9	Ebben a number, string az objektum.	Ebben a number, string az interfész
10	A JavaScript nem támogatja generikusokat.	A TypeScript támogatja a generikusokat.

Melyek a TypeScript különböző elemei?



Components of TypeScript

Nyelv (Language): A nyelv olyan elemeket tartalmaz, mint new syntax, keywords, type annotations, és lehetővé teszi számunkra a TypeScript írását.

Fordítóprogram (Compiler): A TypeScript fordító nyílt forráskódú, platformokon átívelő, és TypeScript-be van írva. A TypeScript-ben írt kódot ekvivalensvé alakítja a JavaScript-kóddal. Elvégzi a TypeScript kódunk és a JavaScript kód elemzését, típusellenőrzését. Ez segíthet a különböző fájlok összefűzésében az egyetlen kimeneti fájlban és a source map (a source map a debuggoláshoz kell) létrehozásában is.

Language Service: A nyelvi szolgáltatás információkat nyújt, amelyek segítenek a szerkesztőknek és más eszközöknek abban, hogy jobb segítséget nyújtsanak, mint például az automatikus refaktorálás és az IntelliSense.

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

SN	Internal Module	External Module
1	A belső modulokat használták az osztályok, interfészek, függvények, változók logikai csoportosításához egyetlen egységbe, és exportálhatók egy másik modulba.	A külső modulok hasznosak a moduldefiníciók belső utasításainak elrejtésében, és csak a deklarált változóhoz tartozó metódusokat és paramétereket mutatják be.
2	A belső modulok a Typescript korábbi verziójában voltak. De még mindig támogatják őket a névtér használatával a TypeScript legújabb verziójában.	A külső modulokat egyszerűen a TypeScript legújabb verziójában modulként ismerik.

3	A belső modulok más modulok helyi vagy exportált tagjai (beleértve a globális modult és a külső modulokat is).	A külső modulok külön betöltött kódtestek, amelyekre külső modulok nevei hivatkoznak.
4	A belső modulokat a ModuleDeclarations segítségével deklarálják, amelyek megadják a nevüket és a törzsrészüket (body).	A külső modul külön forrásfájlként van megírva, amely legalább egy import-ot vagy export-ot tartalmaz.

5	Example:
	`module Sum {`
	`  export function add(a, b)`
	`{`
	`  console.log("Sum: "`
	`+(a+b));`
	`  }`
	`}`
		Example:
		`export class Addition{`
		`  constructor(private x?: number,`
		`private y?: number){`
		`  }`
		`  Sum(){`
		`  console.log("SUM: "`
		`+(this.x + this.y));`
		`  }`
		`}`

A TypeScript támogatja a függvények túlterhelését, mivel a JavaScript nem támogatja a függvények túlterhelését?

Igen, a TypeScript támogatásja. De a megvalósítás furcsa. Amikor a TypeScript-ben függvény túlterhelést hajtunk végre, akkor csak egy, több szignatúrával rendelkező függvényt valósíthatunk meg.

```
//Function with string type parameter
function add(a:string, b:string): string;

//Function with number type parameter
function add(a:number, b:number): number;

//Function Definition
function add(a: any, b:any): any {
   return a + b;
}
```

A fenti példában az első két sor a függvény túlterhelés deklarációja. Két túlterhelése van. Az első szignatúrának karakterlánc típusú paramétere van, míg a másodiknak number van. A harmadik függvény tartalmazza a tényleges megvalósítást, és bármilyen (any) típusú paraméterrel rendelkezik. Bármely adattípus bármilyen adatot felvehet.

Mi a különbség az "interface vs type" utasítások között?

```
interface X {
    a: number
    b: string
}
type X = {
    a: number
    b: string
};
```

SN	interface	type
1	Az interfész deklaráció mindig megnevezett objektumtípust vezet be.	A típusalias deklaráció bármilyen típusú nevet bevezethet, beleértve a primitív, az union stb típusokat is.
2	Az interfész extends vagy implements-ben használható	Az objektumtípus literál típusneve nem használható az extends vagy implement-ben.
3	Az interfészek új nevet hoznak létre, amelyet mindenhol használnak.	A típusaliasok nem hoznak létre új nevet.
4	Egy interfésznek több egyesített deklarációja lehet.	Az objektumtípus literál típusneve nem tartalmazhat több egyesített deklarációt.

Mi a Type assertion a TypeScriptben?

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

```
let empCode: any = 111;
let employeeCode = <number> code;
console.log(typeof(employeeCode)); //Output: number
```

Mi a "as" szintaxis a TypeScriptben?

Az as-szintaxis bevezetésének oka az, hogy az eredeti szintaxis (<type>) ütközött a JSX-szel.

```
et empCode: any = 111;
let employeeCode = code as number;
```

Ha a TypeScript-t JSX-szel használja, akkor csak as-stílusú állítások engedélyezettek.