Tartalom

Változók létrehozása

A változók létrehozása a következőképpen történik

hatáskör valtozoNev : tipus = érték;

Hatáskör: Let, var, vagy const

A változóknak JavaScriptben HÁROM típusát megkülönböztetjük, de ez nem a bennük tárolandó értékek típusát határozza meg hanem a változók funkcióit.

let:
Hasonlóan működik, mint korábban JavaScript eseténben.

const:
Szintén hasonlóan működik, mint a korábban bemutatott JavaScript esetén.

var:
A TypeScript változói a var kulcsszóval is deklarálhatók, ugyanúgy, mint a JavaScriptben. A hatókör szabályai ugyanazok maradnak, mint a JavaScriptben. Újra deklarálás esetén, a rendszer külön figyelmeztet, ha más típusú változót szeretnénk létrehozni ugyanazon a néven!

Változó típusok

Egy **erősen típusos** nyelv (pl. TypeScript, Java, C#) azt jelenti, hogy:

A nyelv ragaszkodik ahhoz, hogy a változók és függvények mindig a meghatározott típusnak megfelelő értékeket kapjanak.

Miért jó az erősen típusosság?

1. **Kezedben a kontroll:** Nem történhet „véletlenül” típuscsere, amit csak futás közben vennél észre.
2. **Fordítás előtti hibakeresés:** Már kódolás közben hibát jelez, így kevesebb runtime-hibád lesz.
3. **Jobb IDE-támogatás:** Az automatikus kiegészítés (IntelliSense) csak akkor működik igazán jól, ha tudja, milyen típusú a változód.
4. **Nagyobb projektekben elengedhetetlen:** A típusosság segít olvasni és karbantartani a kódot akkor is, ha más írta.

valtozoNev: Névadási szabályok

A JavaScript nyelvben az alábbi dolgokra kell ügyelnünk:

- A **változó nevében lehet betű akár kicsi akár nagy**, ugyanakkor bár lehet **ékezetes** betűt használni azokat mégis próbáljuk meg kerülni.
- A változó nevében **NEM lehet space** karakter.
- A változó **NEM kezdődhet számmal**!
- A változó **kezdődhet „_”***(alulvonás)* és **„$”***(dollár jel)* karakterekkel bár ez utóbbi ritkább.
- A betűk mellett a „_” és „$” karakterek is bárhol lehetnek a változó nevében.
- Törekedjünk arra, hogy a **név utaljon a** benne eltárolt **változó feladatára**.
- A nyelv **case-sensivitve**, ami azt jelenti, hogy a kis és nagybetűk között különbséget tesz! tehát „alma” nem egyenlő az „Alma” változó névvel
- A legtöbbször a **camelCase** változó neveket használjuk, a változót több szóra bontjuk, az első szó kisbetűvel a többi szó nagybetűvel kezdődik és az összes szót összefűzzük.
  pl.: camelCaseValtozonev, ezIsEgyValtozo, ittEgyFontosValtozo stb...
  Esetleg használhatjuk az alulvonás karaktert is mint a szavakat elválaszó elemet, de ez is ritkább:
  pl.: camel_case_valtozonev, ez_is_egy_valtozo, itt_egy_fontos_valtozo stb...

Érdekesség: Egyes programozók a const-t típusú változók neveit csupa nagybetűvel írják, a könnyebb olvashatóság érdekében, de ez egyéni megszokás kérdése.

Típus meghatározás

Implicit típusmeghatározás

A JavaScript-ben nem adunk meg típusokat, a nyelv futásidőben dönti el a változó tartalma alapján, hogy az milyen típusú. Ez rugalmas, de sok hibát rejthet, mert semmi sem figyelmeztet, ha rossz típust használunk.

```
var programozasiNyelv1 = "JavaScript";
```

Explicit típusmeghatározás

A TypeScript-ben megadhatjuk a típusokat. Itt például egyértelműen jelezzük, hogy csak szöveg (string) lehet az érték. Ha mást próbálnánk adni neki, a fordító hibaüzenetet ad!

```
var programozasiNyelv2: string = "TypeScript";
```

Változók típusai

:string – szöveges adatok tárolása

szöveg (karakterlánc típus, karakterTömb), idézőjelek között adhatunk meg neki értéket

```
var szoveg: string = "Szeretem a programozást!";
```

Mire való?

Felhasználónevek, üzenetek, címek – minden, ami szövegként értelmezhető.

Mi történik, ha hibázunk?

```
var felhasznaloNev = 123; //HIBA! Nem lehet számot szövegbe rakni.
```

Miért baj? Egy funkció, amely mondjuk .toUpperCase()-t vár, hibára futna, mert a számokon ilyen nincs.

:number – egész vagy valós számok

szám típus, egész és valós számok rendelhetők hozzá

```
let szam: number = 21;
```

Mire való?

Számításokhoz szükséges típus. Kor, ár, pontszám, statisztikai adatok.

Mi történik, ha hibázunk?

```
var kor = "huszonöt"; //HIBA! Szöveget nem lehet számként kezelni.
```

Miért baj? Egy számításnál (kor + 5) furcsa eredményhez vagy hibához vezetne. Az eredeti értékhez hozzáfűzni az 5-öt, mivel string esetén a + művel a hozzáfűzés művelete.

:boolean – logikai típus

Eldöntendő logikai állapotokat jelez.

```
var logikai: boolean = true;
```

Mire való?

Kapcsolók, feltételek, például: „be van-e jelentkezve?”, „látható-e?”, „elküldve?”.

Mi történik, ha hibázunk?

```
var bejelentkezve = "igen"; //HIBA! A TS csak `true` vagy `false` értéket enged meg.
```

Miért baj? Egy if-ágban félreértéshez vezethet: if (bejelentkezve) sosem úgy működik, ahogy vártuk.

Speciális változó típusok

:any (bármi) – szöveges, szám, logikai, tömb típus stb.

any-hez rendelhetünk bármilyen típust, mint ahogy JavaScriptnél már megszoktuk, de ugyanazokkal a veszélyekkel jár, mint anno, ha rossz változótípust adunk meg egy művelethez az eredmény hibára futnak. Lényegében lekapcsolja a TypeScript típusellenőrzését.

```
var barmi: any = true;
barmi = "Pisztácia";
barmi = 42;
```

Előnye

Hasznos lehet *nagyon dinamikus adatoknál* (pl. JSON válasz, külső API, user input), amikor még nem tudjuk pontosan a típust.

Veszélye

Eltűnik a típusellenőrzés, tehát:

- Nem jelez hibát a fordító.
- Könnyen belecsúszhatsz futásidejű hibába, pl. egy nem létező metódust próbálsz hívni.

:unknow(ismeretlen) – szöveges, szám, logikai, tömb típus stb.

Az unknown típus **nagyon hasonlít az any-hez**, mivel bármilyen értéket tárolhatunk benne.

```
var ismeretlen: unknown = false;
ismeretlen = "Csokoládé";
ismeretlen = 21;
```

Mégis miért jobb? Mint az any?

Míg az **any** mindent elfogad és bármit megenged**, az unknown** kényszerít minket a típusellenőrzésre használat előtt**.**

Ez azt jelenti, hogy **nem végezhetünk rajta semmilyen konkrét műveletet (pl. metódushívás)**, amíg meg nem győződünk róla, hogy az adott érték ténylegesen *olyan típusú*, amit használni akarunk.

```
let valami: unknown = "Helló világ";
if (typeof valami === "string") {
  console.log(valami.toUpperCase()); //Most már biztos, hogy string, ezért
működik
}
```

- Használj unknown típust, ha **még nem tudod biztosan**, milyen adat érkezik (pl. JSON-ből), de **nem akarod eldobni a típusvédelmet**.
- Típusellenőrzés nélkül nem tudod használni – és **ez így van jól!**

Mire figyelj?

- Ne próbálj rögtön „használni” egy unknown típusú változót.
- Előbb typeof ellenőrzéssel, vagy típusként konvertálva (as string, as number, stb.) határozd meg, mit tartalmaz.

:null/undefined

Jelölheted vele, hogy „jelenleg nincs érték” – pl. beviteli mező még üres, vagy adat nincs betöltve.

```
let email: string | null = null;
```

Egyszerű tömbök

Olyan adatokat tárolunk benne, amik **ugyanolyan típusúak**, és **sorrend szerint hozzáférhetők** – például felhasználónevek, termékárak, pontszámok.

Tömbök esetében is, mint az előbb bemutatott primitív változók esetében, is megadhatók annak típusai az alábbi módokon.

```
var szovegesTomb: string[] = ["alma", "körte", "barack", "szilva"];
var szamTomb: number[] = [21, 13, 42, 3, 69, 33, 81, 66];
```

vagy az alábbi módon:

```
var szovegesTomb2: Array<string> = ["rpg", "fps", "tps", "szimulátor",
"rts", "4x", "mmo"];
var szamTomb: Array<number>=[3,7,13,21,42];
```

A feltöltés menete a JS-ben megszokott módon történik.

```
szamTomb.push(7);
```

Aki a JavaScript-ben megszokott tömböt szeretne, használhatja az :any[] típust...

```
var szemetesKuka: any[] = ["almacsutka", 13, false];
szemetesKuka.push("Pelenka");//Any típusú tömbbe bármit bele lehet
helyezni... Mint JS esetén
```

Több típus engedélyezése (union types – unió típus)

A TypeScript-ben lehetőség van arra, hogy egy változó többféle típust is elfogadjon, de szigorú szabályozás mellett.

Ehhez az úgynevezett unió típus szintaxist használjuk: **type1 | type2 | type3** stb.

Union példa

```
let valasz: string | number;

valasz = "OK";       //Megfelel
valasz = 200;        //Szintén megfelel
valasz = true;       //Fordítási hiba! Nem szerepel a lehetséges típusok
között
```

Mire figyelj?

- A string | number típus *nem azt jelenti*, hogy **bármilyen** típus lehet – **csak azokat fogadja el, amelyeket megadtál**.
- Ha konkrét típusspecifikus műveletet akarsz végezni (pl. toUpperCase() vagy toFixed()), akkor **típusellenőrzésre** van szükség:

```
function kezeles(ertek: string | number) {
  if (typeof ertek === "string") {
    console.log(ertek.toUpperCase());
  } else {
    console.log(ertek.toFixed(2));
  }
}
```

FONTOS!!!

Az union nem ugynaz mint az any, itt csak adott típusú változókat fogadunk el, a típusellenőrzés megmarad, míg any esetében nincs!!!

```
// FONTOS különbség!
let x: any = "alma";      // bármit elfogad
let y: string | number;   // csak stringet vagy számot fogad el

y = true;   //Hiba! Ez nem megengedett, míg any esetén igen
```