Tartalom

A függvényekről TypeScript-ben

A függvények hasonlóan működnek, mint JavaScriptben, lévén arra fordul le a végén, a tsc parancs kiadása után, de ugyanakkor, a gyengén típusosság, erősen típusosságba vált át, ami azt jeleni, minden egyes változót, amit használunk, és annak típusát, meg kell adnunk!

Visszatérési érték, és paraméter nélküli eljárások

Ebben az esetben a függvény nem kap bemeneti paramétert, és nem is ad vissza semmilyen értéket. Ilyenkor a void típus jelzi, hogy **nem várható visszatérési érték**.

```
function IloveTypeScipt(): void {
    console.log("Szeretem a TypeScriptet");
}
```

Mire való?

- Ilyen eljárásokat (függvényeket) akkor használunk, ha egy **műveletet vagy oldalszintű hatást (pl. kiírás, DOM módosítás)** szeretnénk végrehajtani, de **nem szeretnénk vele adatot visszakapni**.
- Tipikus példa: figyelmeztetések kiírása, naplózás (console.log), DOM-műveletek (document.write, appendChild, stb.)

Mire figyelj?

- **Technikailag működni fog akkor is, ha nem adod meg a** void **típust** — a TypeScript képes kikövetkeztetni — de ha **kifejezetten pontos kódot** szeretnél írni, akkor **érdemes jelezni**, hogy a függvény nem ad vissza értéket.
- Ha tévedésből visszatérési értéket adnál, a void figyelmeztethet erre.

Tipp:

- Érdemes ilyen függvényeket jól **beszédes névvel ellátni**, pl. logSikeresMentés() vagy showHelpMessage(), hogy látszódjon: nem adnak vissza adatot, csak "tesznek valamit".
- Magyarázó, oktató célú, vagy vizuális visszacsatolást adó műveleteket érdemes ilyen formában megvalósítani.

Visszatérési érték nélküli, paraméteres eljárások

Ebben az esetben a függvény ugyan **nem ad vissza értéket**, de **kap bemenetet**. Ez lehetővé teszi, hogy dinamikusabb műveleteket hajtsunk végre a paraméter(ek) alapján.

```
function NegyzetKeruletTeruletEljaras(a: number): void {
    let kerulet = 4 * a;
    let terulet = a * a;
 console.log(`A ${a} oldalú négyzet kerülete:${kerulet} területe${terulet}`);
}
```

további példa...

```
function ellenorizNev(nev: string): void {
    if (nev.length < 5) {
        alert("A felhasználónévnek legalább 5 karakter hosszúnak kell lennie!");
    } else {
        alert(`A megadott felhasználónév érvényes: ${nev}`);
    }
}
```

Mire való?

- Akkor használjuk, ha egy **bemenő érték alapján szeretnénk valamilyen műveletet végrehajtani**, de nem kell visszaadni adatot a hívó félnek.
- Kiváló **állapotmegjelenítéshez**, **DOM-manipulációhoz**, **egyszerű**

Mire figyelj?

- A paraméter(ek) típusát **kötelező megadni**, különben a TypeScript nem tudja biztosan ellenőrizni, hogy milyen érték kerül átadásra.
- Ha **nem megfelelő típust** adunk át (pl. string helyett number), **fordítási vagy futási hiba** keletkezhet.
- A dokumentumban történő kiírás (document.write) helyett komolyabb alkalmazásban inkább console.log vagy DOM-manipuláció (innerText, textContent) a javasolt.

Tipp:

- Ezek az eljárások **jól kombinálhatók vezérlési szerkezetekkel** (pl. if, switch), így akár többféle bemenetet is lekezelhetsz.

Visszatérési értékes, paraméter nélküli függvények

Egy olyan függvény, amely *nem vár bemenetet (paramétert)*, de **visszatérési értékkel rendelkezik**. A függvény neve után a zárójelek következnek, majd kettősponttal meg kell adni a visszatérési érték típusát, pl.: number, string, boolean, stb.

```
function RandomGeneralo100Fuggveny(): number {
    return Math.round(Math.random() * 100);
}
```

Mire való?

- Leggyakrabban **alap értékek generálására** használjuk (pl. random szám, dátum, aktuális állapot stb.)
- Szerepe lehet **függvények közötti adatátadásban**, vagy **érték visszaadásában egy logikából**

Mire figyelj?

- A visszaadott érték **típusát előre meg kell határozni**, különben a TypeScript figyelmeztet vagy hibát ad, ha nem egyezik a return érték típusa.
- Csak **azonos típusú változóban tárolható** az eredmény:

Tipp:

- Az ilyen függvényekkel **egységteszteléshez is lehet alapértékeket generálni**

Visszatérési értékes, paraméteres függvények

Ez a függvénytípus **paramétereket fogad és visszatérési értékkel is rendelkezik**. A zárójelben felsoroljuk a paramétereket, mindegyikhez megadjuk a típusát, majd a zárójelek után kettősponttal megadjuk a visszatérési érték típusát is.

```
function NegyzetKeruletFuggveny(a: number): number {
    return 4 * a;
}
```

Mire való?

- Akkor használjuk, **ha egy konkrét bemeneti érték (vagy értékek) alapján szeretnénk egy eredményt visszaadni**, például egy számítás vagy lekérdezés eredményét.
- Kiváló szerkezet például matematikai műveletek, feldolgozási logikák vagy validálások esetén.

Mire figyelj?

- Minden **paraméterhez szükséges típusdefiníció**, különben a TypeScript figyelmeztet vagy hibát dob.
- A visszatérési érték **típusa nem lehet ellentétes** a tényleges return értékkel.
- Ha például a függvény number típust ígér, de string típust ad vissza, az fordítási hibát eredményez

Tipp:

- Több bemeneti adatot is kezelhetsz, pl. function osszeg(a: number, b: number): number.
- Érdemes először a **paraméterekre koncentrálni**, és csak ezután felépíteni a return logikát, így átláthatóbb lesz a függvény.

Visszatérési értékes, paraméteres függvények, TUPLE adatszerkezettel

Ha egy függvénynek **egynél több értéket kell visszaadnia**, akkor célszerű a **Tuple** típust használni. Ez olyan, mint egy tömb, **de előre meghatározott számú és típusú elemeket tartalmaz**.

```
function NegyzetKeruletTupleFuggveny(a: number): [number, number] {
    let keruletEredmeny = 4 * a;
    let teruletEredmeny = a * a;
    return [keruletEredmeny, teruletEredmeny];
}
```

Mire való?

- Többféle értéket adhatunk vissza **egy függvényből**, külön típusokkal.
- Elkerüljük az objektumhasználatot, ha csak pár értékről van szó.
- Tipikus példa: számítási eredmények visszaadása (pl. kerület + terület), validálás (pl. sikeresség + hibaüzenet), stb.

Mire figyelj?

- A Tuple típust **szigorúan ellenőrzi a TypeScript**, tehát a visszatérési értéknek **pontosan ugyanannyi és olyan típusú eleme kell legyen, mint amit megadtál**.
- Index alapján hivatkozhatsz az elemekre, de ez néha **kevésbé olvasható**

Tuple függvény eredményének meghívása

```
    console.log("Tuple értékkel való függvény eredménye:")
    var tupleFuggvenyEredmeny = NegyzetKeruletTupleFuggveny(5);
    console.log("A négyzet kerülete:" + tupleFuggvenyEredmeny[0]);
    console.log("A négyzet kerülete:" + tupleFuggvenyEredmeny[1]);
```

Tipp:

- Tuple típust akkor érdemes használni, **ha pontosan tudod, hány értéked lesz**, és azok sorrendje/típusa állandó.
- Ha rugalmasabb szerkezet kell (pl. kulcs–érték párosok), akkor inkább object típus ajánlott.

Speciális funkciók függvények esetén

Opcionális paraméterek (param?: típus)

```
function udvozlet(nev?: string): void {
  if (nev) {
    console.log("Szia, " + nev + "!");
  } else {
    console.log("Szia, ismeretlen!");
  }
}
```

Mire való?

- Nem kötelező megadni minden paramétert. Ha nincs érték, `undefined` lesz, de nem hibázik a kód.

Mire figyelj?

- Ha kötelezőként kezeled, és nincs megadva érték, hiba léphet fel.

Tipp:

- Opcionális paramétereket mindig a paraméterlista végére tegyél!
- Használható több opcionális paraméter is, ha azok a végén vannak.

Alapértelmezett értékű paraméterek

Ilyenkor, ha nem adunk meg értéket meghíváskor, akkor az alapértelmezett értéket kapja a függvény paramétere.

```
function udvozlet2(nev: string = "vendég"): void {
  console.log("Üdvözöllek, " + nev + "!");
}
```

Mire való?

- Nem kötelező megadni a paramétert, ha van előre beállított érték.

Mire figyelj?

- Nem jelent hibát, de figyelj rá, hogy az érték ne legyen félrevezető.

Tipp:

- Alapértelmezés és opcionális paraméter kombinálható.

Arrow function típusozása

```
const osszead = (a: number, b: number): number => {
  return a + b;
};
```

Mire való?

- Rövidebb szintaxis, különösen callbackeknél előnyös.

Mire figyelj?

- Ha elmarad a típusozás, nehezebb hibát találni.

Tipp:

- Mindig add meg a bemenet és kimenet típusát is!

:never típus – ha sosem tér vissza

```
function hibaUzenet(msg: string): never {
  throw new Error(msg);
}
```

Mire való?

- Az olyan függvényeknél használatos, amelyek mindig hibát dobnak vagy végtelen ciklusban futnak.

Mire figyelj?

- Nehéz olvasni, ha nem ismerjük ezt a típust – de nem hiba, csak speciális jelzés.

Tipp:

- A TypeScript gyakran automatikusan felismeri ezt, nem kell gyakran kézzel megadni.