TYPESCRIPT

TypeScript je open-source objektno-orjentisani programski jezik koji je razvila i održava kompanija Microsoft, a licensiran je kroz Apache 2 licencu.

PREGLED TYPESCRIPT-A

- **TS** je **tipizirani nadskup** (superset) JavaScript-a koji se kompajlira **(translira)** u čisti JavaScript**.**
- **TS** je razvijen pod vođstvom Anders Hejlsberg-a, koji je takođe vodio razvoj C# programskog jezika. TypeScript je objavljen prvi put Oktobra 2012.
- **Preduslovi**
  - *Bazično poznavanje JavaScript programskog jezika i OOP principa.*

PREGLED TYPESCRIPT-a

- TypeScript proširuje JavaScript dodavanjem tipova podataka, klasa, i ostalih objektno-orjentisanih principa sa provjerom tipa.
- Zvanični websajt: https://www.typescriptlang.org
- TS kod: https://github.com/Microsoft/TypeScript

ZAŠTO TYPESCRIPT?

- **JavaScript** je **dinamički** programski jezik koji nema statički sistem tipova (slično kao Python).
- **JavaScript** uključuje primitivne tipove kao što su string, broj, objekat, itd.; **ali ne provjerava dodijeljene vrijednosti**.
- **JavaScript** promjenljive nijesu vezane za određeni tip, već mogu u toku izvršavanja primati vrijednosti različitih tipova podataka.
- **JavaScript** ne podržava klase i druge objektno orjentisane osobine, pa bez sistema tipova nije jednostavno koristiti je za izradu složenijih aplikacija.

ZAŠTO TYPESCRIPT?

- Sistem tipova povećava kvalitet i čitljivost koda i olakšava njegovo održavanje i refaktorisanje. Još važnije, greške se mogu otkriti u vrijeme kompilacije, a ne tokom izvršavanja programa.
- Zbog toga se TypeScript koristi jer otkriva greške u vrijeme kompilacije, što omogućava njihovo ispravljanje prije pokretanja programa. Podržava osobine objektno orijentisanog programiranja poput tipova podataka, klasa i enumeracija.

ZAŠTO TYPESCRIPT?

- **TypeScript** se kompajlira u **JavaScript**.
- TypeScript kompajler je implementiran u TypeScript-u i može se izvršavati u bilo kom **JavaScript okruženju** kao što je **Node.js**. Za kompalaciju je potreban ECMAScript3 ili noviji, što ispunjavaju svi savremeni pretraživači JS engine-i.
- Neki od popularnih **JavaScript framework-a** kao što su **Angular.js**, **Vue3.js**, **Nest.js** su napisani u TS-u.

KAKO SE KORISTI?

- **TypeScript** kod se piše u fajlu sa **.ts** ekstenzijom. Zatim se kompajlira u JavaScript koristeći se TypeScript kompajlerom.
- **TypeScript** fajl može biti napisan u bilo kom kod editoru. Morate imati instaliran TypeScript kompajler u okruženju.
- Komanda: **tsc <filename>.ts** kompajlira TypeScript kod u čisti JavaScript fajl. JavaScript fajl se onda može uključiti u HTML i pokrenuti na bilo kom pretraživaču.

Kompajliranje TypeScript u JavaScript

TYPESCRIPT OSOBINE

- **Cross-Platform**: TypeScript se pokreće na bilo kojoj platformi na kojoj se može pokrenuti JavaScript. TypeScript kompajler se može instalirati na bilo kom Operativnom sistemu (Windows, macOS, and Linux)
- **Objektno orjentisani jezik**: TypeScript nudi moćne osobine kao što su Klase, Interfejsi, i Moduli. Omogućava pisanje objektno-orjentisanog koda za klijenta (frontend) kao i za server (backend).

TYPESCRIPT OSOBINE

- **Statička provjera tipa**: TypeScript koristi statičko tipiziranje. Ovo se postiže pomoću tipnih anotacija. One omogućavaju provjeru tipova u vrijeme kompajliranja, pa se greške mogu otkriti već tokom pisanja koda, bez potrebe da se skripta pokreće svaki put. Pored toga, zahvaljujući mehanizmu automatskog određivanja tipa, ako je promjenljiva deklarisana bez eksplicitnog tipa, tip će biti automatski određen na osnovu njene vrijednosti.
- **Statičko tipiziranje je opciono**: Može se koristiti i dinamičko tipiziranje, s tim što bi TS kompajler takve varijable posmatrao kao "any" tip podatka.

TYPESCRIPT OSOBINE

- **Upravljanje DOM-om**: Kao JavaScript, TypeScript se može koristiti da upravlja modelom dokumenta.

TYPESCRIPT PREDNOSTI

- **TypeScript** je sličan JavaScriptu i koristi istu sintaksu i semantiku. Omogućava brže usvajanje jezika za front-end programere koji već rade u JavaScriptu.
- **TypeScript** je po svojoj sintaksi takođe bliži jezicima na serverskoj strani, kao što je Java. To backend programerima olakšava brže pisanje front-end koda.

TYPESCRIPT PREDNOSTI

- **TypeScript** kod se može pozivati iz postojećeg JavaScript koda. TypeScript takođe radi sa svim postojećim JavaScript framework-owima i bibliotekama bez problema.
- **TypeScript Definition fajl**, sa **.d.ts** ekstenzijom, pruža podršku za postojeće JavaScript biblioteke poput jQuery, D3.js i drugih. TypeScript kod može koristiti te biblioteke pomoću tipnih definicija, čime se obezbjeđuju prednosti provjere tipova, automatskog dopunjavanja koda i dokumentacije čak i u postojećim dinamički tipiziranim JavaScript bibliotekama.

INSTALACIJA TYPESCRIPT-A

INSTALACIJA

- Postoje **3 načina za instalaciju TypeScript-a**:
  - Instaliranje **TypeScript-a** ka **NPM paketa** na lokalnoj mašini ili unutar projektnog okruženja. - najpraktičniji način za naš predmet.
  - Instalacija putem **TypeScript NuGet Package** u .NET ili .NET Core projektu.
  - Instalacija **TypeScript-a** kao **Plug-in u IDE-u** (Integrated Development Environment)- npr. Visual Studio.

INSTALACIJA TYPESCRIPT-a putem NPM-a

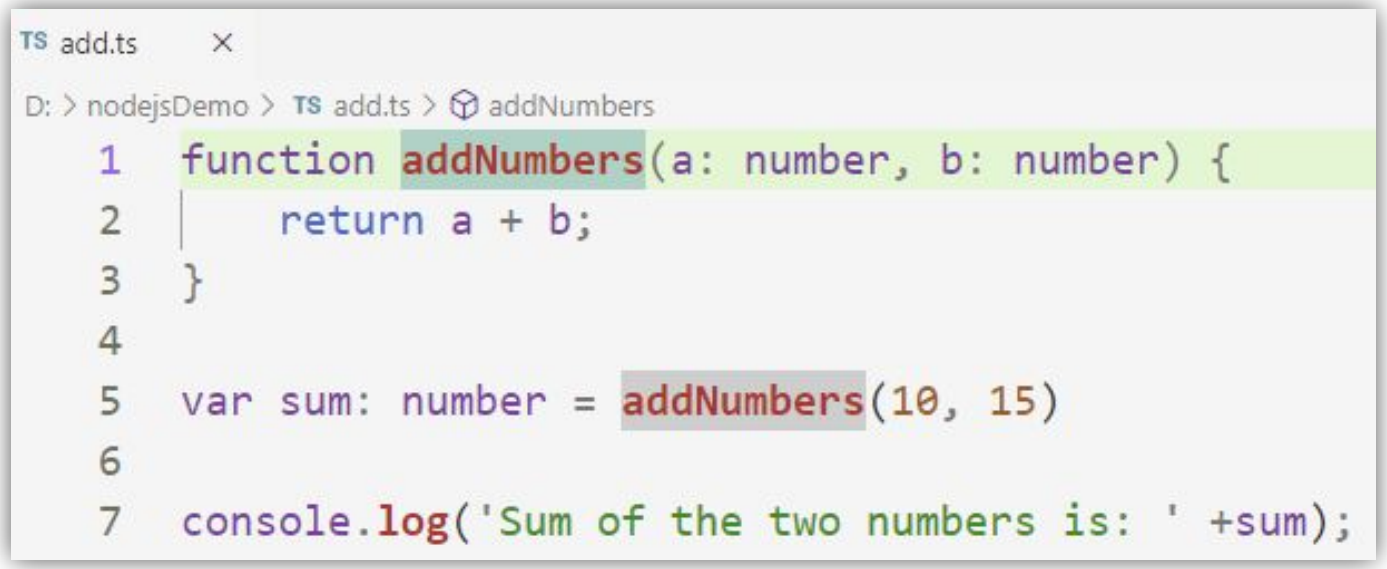
- **Potrebno je imati instaliran Node.js na vašem računaru**. (https://nodejs.org/en/download)
- Da bi se instalirala ili ažurirala najnovija verzija TypeScripta, otvorite komandnu liniju ili terminal i unesite sljedeću komandu:
  - **npm install -g typescript**
- **Napomena:** alati za pokretanje framework projekata (npr. Vite) automatski instaliraju TS u taj projekat

INSTALL TYPESCRIPT USING NPM

- Komanda **npm install -g typescript** instalira TypeScript globalno, što omogućava korišćenje TS u bilo kom projektu. Instaliranu verziju TypeScripta možete provjeriti sljedećom komandom:
  - **tsc -v**
- Sljedeća komandu instalira TypeScript u lokalni projekat kao razvojnu zavisnost (dev dependency):
  - **npm install typescript --save-dev**

PRVI TYPESCRIPT PROGRAM

- Kreirajte novi fajl u vašem kod editor-u i nazovite ga **add.ts** . Prepišite kod dat u nastavku:



PRVI TYPESCRIPT PROGRAM

- Na prethodnom slajdu je definisan TS fajl koji sadrži f-ju **addNumbers()**, poziva je, i log-uje (ispisuje) rezultat u konzoli pretraživača.
- Otvorite command prompt na Windows-u (ili terminal na MacOs-u ili Linux-u), navigirajte do puta gdje ste sačuvali **add.ts**, i kompajlirajte program koristeći sljedeću komandu:
  - **tsc add.ts**
- Ova komanda će "prevesti" TS fajl u JS fajl sa imenom **add.js** na istoj lokaciji. **add.js** fajl je prikazan na sljedećem slajdu.

TS add.ts

D: > nodejsDemo > TS add.ts > addNumbers

```ts
function addNumbers(a: number, b: number) {
    return a + b;
}

var sum: number = addNumbers(10, 15)

console.log('Sum of the two numbers is: ' +sum);
```

TS add.ts JS add.js

D: > nodejsDemo > JS add.js > sum

```js
function addNumbers(a, b) {
    return a + b;
}
var sum = addNumbers(10, 15);
console.log('Sum of the two numbers is: ' + sum);
```

```
D:\nodejsDemo>node add.js
Sum of the two numbers is: 25
```

Ukoliko imate instaliran Node.js na vašem računaru, .js fajl možete pokrenuti naredbom: Node add.js

ili u konzoli pretraživača (html)

```
<script src="add.js"></script>
 <script>
  const rezultat = addNumbers(5, 7);
  console.log(rezultat);
 </script>
```

TYPESCRIPT - TIPOVI

- **TypeScript** je tipiziran jezik, *u kom možemo odrediti* ***tip varijabli, parametara funkcija i atributa objekata***.
- Tip određujemo korišćenjem ":" ***nakon imena varijable, parametra ili atributa.***
- TypeScript uključuje **sve primitivne tipove JavaScript**-a **number**, **string** i **boolean**.

TYPESCRIPT - KORIŠĆENJE TIPOVA

- Ovaj primjer deklarise varijable različitih tipova:

Example: Type Annotation in TypeScript

```
var age: number = 32; // number variable
var name: string = "John";// string variable
var isUpdated: boolean = true;// Boolean variable
```

TYPESCRIPT - OSOBINE TIPOVA

- Vrijednost promjenljive nije moguće mijenjati korišćenjem tipa podataka koji se razlikuje od onoga koji je prethodno deklarisan. Ukoliko se takav pokušaj izvrši, TypeScript kompajler će prijaviti grešku.
- Na primjer, ukoliko se promjenljivoj age (number) dodijeli vrijednost tipa string ili promjenljivoj name (string) vrijednost tipa number, kompajler će prijaviti grešku!

TYPESCRIPT - TIPIZIRANJE PARAMETARA FUNKCIJE

- Ovaj primjer demonstrira korišćenje tipiziranja u parametrima funkcija

Example: Type Annotation of Parameters

```
function display(id:number, name:string)
{
    console.log("Id = " + id + ", Name = " + name);
}
```

TYPESCRIPT - TIPIZIRANJE ATRIBUTA OBJEKATA

Example: Type Annotation in Object

```
var employee : {
    id: number;
    name: string;
};
employee = {
  id: 100,
  name : "John"
}
```

- Ovdje je deklarisan objekat employee sa dva svojstva :id i name čiji su tipovi podataka redom **number** i **string.**

TYPESCRIPT - DEKLARACIJA VARIJABLI

- TypeScript koristi ista pravila za deklaraciju varijabli kao i JS. Varijable se mogu deklarisati korišćenjem ključnih riječi: var, let, and const. Pravila opsega (scope rules) ostaju ista kao i u JS-u.
  -

Example: Variable Declaration using let

```
let employeeName = "John";
// or
let employeeName:string = "John";
```

TYPESCRIPT - SCOPE PRAVILA

- Ključne riječi **let i const** uvedene su kao zamjena za var.
- Za razliku od **var**, koji ima **funkcijski opseg**, **let i const** imaju **blokovski opseg**, što znači da važe samo unutar koda u kojem su definisane (npr. funkcija, if-else, petlja).
- let se koristi za promjenljive vrijednosti, a const za one koje se ne mijenjaju.
- vrijednost varijable deklarisane sa let se može mijenjati, ali se ta ista varijabla ne može re-deklarisati, što je slučaj kod varijabli deklarisanih kao var.

Example: let Variables Scope

```
let num1:number = 1;

function letDeclaration() {
    let num2:number = 2;

    if (num2 > num1) {
        let num3: number = 3;
        num3++;
    }

    while(num1 < num2) {
        let num4: number = 4;
        num1++;
    }

    console.log(num1); //OK
    console.log(num2); //OK
    console.log(num3); //Compiler Error: Cannot find name 'num3'
    console.log(num4); //Compiler Error: Cannot find name 'num4'
}

letDeclaration();
```

- U primjeru su sve promjenljive deklarisane pomoću ključne riječi **let**.
- Promjenljiva **num3** deklarisana je unutar ***if*** bloka, pa je njen opseg ograničen na taj blok i nije joj moguće pristupiti izvan njega
- Na isti način, promjenljiva **num4** deklarisana je unutar ***while*** bloka, pa joj nije moguće pristupiti izvan tog bloka.
- *Zbog toga će pokušaj pristupa promjenljivama **num3** i **num4** izvan njihovih blokova rezultirati greškom pri kompilaciji.*

- Na istom primjeru, varijable su deklarisane sa ključnom riječi ***var,*** **i kod se izvršava bez problema.**

Example: var Variables Scope

```
var num1:number = 1;

function varDeclaration() {
    var num2:number = 2;

    if (num2 > num1) {
        var num3: number = 3;
        num3++;
    }

    while(num1 < num2) {
        var num4: number = 4;
        num1++;
    }

    console.log(num1); //2
    console.log(num2); //2
    console.log(num3); //4
    console.log(num4); //4
}

varDeclaration();
```

PREDNOSTI *let* U ODNOSU na *var*

- 1) **let** varijable se ne mogu čitati, niti im se dodjeljivati vrijednost prije nego što su deklarisane.

```
console.log(num1); // Compiler Error: error TS2448: Block-scoped variable 'num' used before
let num1:number = 10 ;

console.log(num2); // OK, Output: undefined
var num2:number = 10 ;
```

- Neće doći do greške ako se promjenljive koriste prije njihove deklaracije pomoću ključne riječi **var (hoisting)**

PREDNOSTI *let* U ODNOSU na *var*

- **2) Let variables ne mogu biti deklarisane ponovo, ne mogu re-deklarisati ni istoimene var varijable.**
- TypeScript kompajler će prijaviti grešku ako se u istom bloku više puta deklarišu promjenljive sa istim imenom (uzimajući u obzir velika i mala slova) pomoću ključne riječi let.

```
var num:number = 1; // OK
var Num:number = 2;// OK
var NUM:number = 3;// OK
var NuM:number = 4;// OK

let num:number = 5;// Compiler Error: Cannot redeclared block-scoped variable 'num'
let Num:number = 6;// Compiler Error: Cannot redeclared block-scoped variable 'Num'
let NUM:number = 7;// Compiler Error: Cannot redeclared block-scoped variable 'NUM'
let NuM:number = 8;// Compiler Error: Cannot redeclared block-scoped variable 'NuM'
```

≠

```
var num: number = 1; //ok
var Num: number = 2; //ok
var NUM: number = 3; //ok
var NuM: number = 4; //ok


var num: number = 5; //ok
var Num: number = 6; //ok
var NUM: number = 7; //ok
var NuM: number = 8; //ok
```

PREDNOSTI *let* U ODNOSU na *var*

- Varijable sa istim imenom i istim imenom (velika/mala slova) mogu se deklarisati u različitim blokovima.

Example: Same Variable Name in Different Blocks

```
let num:number = 1;

function demo() {
    let num:number = 2;

    if(true) {
        let num:number = 3;
        console.log(num); //Output: 3
    }

    console.log(num);//Output: 2
}
console.log(num); //Output: 1
demo();
```

PREDNOSTI TS u ODNOSU na JS

JS kod koji se izvršava bez problema:

```
function add(a, b) {
  return a + b;
}

console.log(add("2", 3));
// "23" – logička greška, ali bez upozorenja
```

TS kod koji vraća grešku (statičko tipiziranje)

```
function add(a: number, b: number): number {
  return a + b;
}

add(2, 3);      // OK, 5-
add("2", 3);    // Greška pri kompilaciji
```

PREDNOSTI TS u ODNOSU na JS

Provjera strukture objekta

JS kod koji se izvršava

```
function printUser(user) {
  console.log(user.age);
}

printUser({ name: "Ana" }); //user.age je undefined
```

TS kod koji vraća grešku (Interfejski, kontrola strukture objekta, u JS-u ne postoji)

```
interface User {
  name: string;
  age: number;
}

function printUser(user: User) {
  console.log(user.age.toFixed(0));
}

printUser({ name: "Ana" });
//Error: Property 'age' is missing in type '{ name: string }'
// but required in type 'User'.
```

PREDNOSTI TS u ODNOSU na JS

Enumeracija, zastita od pogresnih vrijednosti

JS kod koji se izvršava

```
const Status = { Active: 0, Inactive: 1, Suspended: 2 };

function checkStatus(s) {
  if (s === Status.Active) console.log("Active");
  else if (s === Status.Inactive) console.log("Inactive");
  else if (s === Status.Suspended) console.log("Suspended");
  else console.log("Unknown status!");
}

//ispravne vrijednosti
checkStatus(0);     // Active
checkStatus(1);   // Inactive

// "pogrešna" vrijednost
checkStatus(99);                // Unknown status!
checkStatus("Active");          // Unknown status!
checkStatus(null);              // Unknown status!
checkStatus(undefined);         // Unknown status!
```

PREDNOSTI TS u ODNOSU na JS

Enumeracija, zastita od pogresnih vrijednosti

TS kod koji vraća grešku (enumeracija, zaštita od pogresnih vrijednosti)

```
enum Status {
  Active,
  Inactive,
  Suspended
}

function checkStatus(s: Status): void {
  if (s === Status.Active) {
    console.log("Active");
  } else if (s === Status.Inactive) {
    console.log("Inactive");
  } else if (s === Status.Suspended) {
    console.log("Suspended");
  } else {
    console.log("Unknown status");
  }
}
//ispravni pozivi
+ function checkStatus(s: Status): void
checkStatus(Status.Inactive);   // Inactive
checkStatus(Status.Suspended);  // Suspended

//neispravni – kompajler ih odbacuje
checkStatus(99);       // Error: Argument of type '99' is not
// assignable to parameter of type 'Status'
checkStatus("Active"); // Error: Argument of type 'string' is not
// assignable to parameter of type 'Status'
```

PREDNOSTI TS u ODNOSU na JS

Generički tipni parametri

JS kod koji se uvijek izvršava

za bilo koji tip podatka

```
function wrap(value) {
  return [value];
}

var broj = wrap(10);
var broj = wrap("hello");


//prolazi, nema upozorenja
```

PREDNOSTI TS u ODNOSU na JS

Generički tipni parametri

```
function wrap<T>(value: T): T[] {
  return [value];
}
```

TS kod koji ne prolazi kompajliranje

```
var broj = wrap(10);          // T = number
var broj = wrap(7);
var broj = wrap("Tekst") // T = string
```

<T> predstavlja generički tipni parametar, simbolički označava "bilo koji tip". Type autotatski zaključi tip (inferencija). -

```
//broj: number[]
//broj sada ne moze postati STRING! Subsequent variable
// declarations must have the same type.
//  Variable 'broj' must be of type 'number[]'
```

KONTROLE TIPOVA I OGRANICENJA VRIJEDNOSTI

Unije tipova (kada varijabla može biti različitih tipova, u TS-u to radimo ovako)

```
// Unija tipova: varijabla može biti string ili number
let vrijednost: string | number;
vrijednost = "tekst";  // ok
vrijednost = 42;       // ok
vrijednost = true;  // Greška: boolean nije dozvoljen
```

Literalni tipovi (varijabla može imati samo navedene vrijednosti)

```
// Literalni tip: vrijednost može biti
// samo jedna od navedenih
type Uloga = "admin" | 1 | "guest";
let trenutnaUloga: Uloga = "admin";  // ok
trenutnaUloga = 1
trenutnaUloga = "root";// Greška: "root" nije dozvoljen
```

Objekti definisani sa *type i interface*

- **interface i type u TS-u imaju gotovo identičnu funkcionalnost kada se koriste za opisivanje strukture objekata — oba omogućavaju definisanje svojstava, opcionalnih i readonly polja, kao i tipizaciju metoda unutar objekta.**

```ts
type User = {
  readonly id: number; // samo za citanje
  name: string;
  active?: boolean; // opcioni parametar
};

const user1: User = { id: 1, name: "Ana" };
const user2: User = { id: 2, name: "Marko", active: true };
```

```ts
interface User {
  readonly id: number; //samo za citanje
  name: string;
  active?: boolean; // opcioni parametar
}

const user1: User = { id: 1, name: "Ana" };
const user2: User = { id: 2, name: "Marko", active: true };
```

Objekti definisani sa *type i interface*

- **Oba se mogu proširivati (interface pomoću extends, a type pomoću operatora &), mogu se koristiti kao anotacije tipa za varijable, parametre i povratne vrijednosti funkcija, te obezbjeđuju potpunu statičku provjeru strukture podataka.**

```ts
type User = {
  id: number;
  name: string;
  active?: boolean;
};

type Admin = User & {
  role: string;
};

const admin: Admin = {
  id: 2,
  name: "Marko",
  active: false,
  role: "moderator"
};
```

```ts
interface User {
  id: number;
  name: string;
  active?: boolean;
}

interface Admin extends User {
  role: string;
}

const admin: Admin = {
  id: 1,
  name: "Ana",
  active: true,
  role: "superadmin"
};
```

Objekti definisani sa *type i interface*

- **Međutim, postoje bitne razlike. Interfejs se može re-deklarisati, tj. spojiti sa novim vrijednostima, dok type ne može.**

```
type User = {
  id: number;
  name: string;
};

type User = {
  active: boolean;
}; // Error: Duplicate identifier 'User'
```

```
interface User {
  id: number;
  name: string;
}

interface User {
  active: boolean;
}

const u: User = {
  id: 1,
  name: "Ana",
  active: true
};
```

Objekti definisani sa *type i interface*

- **type nije ograničen na opisivanje objekata, već može predstavljati bilo koji tip podatka u TypeScriptu: primitivne tipove (string, number), funkcije, unije, presjeke, tuple, pa čak i kombinacije svega navedenog. Zato se kaže da je type fleksibilniji, jer omogućava izgradnju složenih i apstraktnih tipova, dok je interface primarno namijenjen za opisivanje struktura objekata i klasa.**

```
//Alias za primitivne tipove
type ID = string | number;

let userId: ID = 42;        //ok
userId = "abc-123";         //ok
userId = true;              // greška - boolean nije dio tipa ID
```

```
//umjesto enumeracije
type Status = "active" | "inactive" | "suspended";

let currentStatus: Status = "active";    // ok
currentStatus = "paused";                // greška - "paused" nije dozvoljen
```

```
//kao tuple
type Point = [number, number];
const uredjeni_par: Point = [0, 0];  // radi normalno

//kao tip funkcije
type Callback = (message: string, code: number) => void;

const log: Callback = (msg, code) => {
  console.log(`${msg}: ${code}`);
};

log("Web Dizajn Nedjelja", 7)
```

Primitivni tipovi u TS-u

```
let ime: string = "Ana";                    // string
let godine: number = 25;                    // number
let aktivan: boolean = true;                // boolean
let nedefinisano: undefined = undefined; // undefined
let nista: null = null;                     // null
let velikiBroj: bigint = 1234567890123456789n; // bigint
let simbol: symbol = Symbol("id");          // symbol
```

- **null i undefined predstavljaju odsustvo vrijednosti, ali u TypeScriptu imaju različu semantiku. undefined označava promjenljivu koja nije inicijalizovana ili argument koji nije prosleđen funkciji; to je podrazumijevana "prazna" vrijednost u JavaScriptu. null se koristi kada programer namjerno postavi vrijednost na "ništa", eksplicitno kaže da podatak postoji, ali trenutno nema smislen sadržaj.**

Složeni tipovi u TS-u

```ts
let biloSta: any = 42;
// any - bez ograničenja
biloSta = "tekst"; // dozvoljeno
biloSta = true;

let nepoznato: unknown = "vrijednost";  // unknown - sličan any, ali sigurniji
// console.log(nepoznato.toUpperCase()); // mora prvo biti provjereno
if (typeof nepoznato === "string") {
  console.log(nepoznato.toUpperCase()); // sigurno
}

function ispisi(): void { // void - nema povratne vrijednosti
  console.log("Pozdrav!");
}

function greska(): never {// never - funkcija nikad ne završava
  throw new Error("Fatalna greška");
}
```

ISPRAVLJANJE JS “NELOGIČNOSTI”

```
null + 1
// JS: 1 (jer null → 0)
// TS: Greška → "Object is possibly 'null'" (strictNullChecks)

console.log([] == ![])
// JS: true (konverzije tipova)
// TS: Greška → "This condition will always return 'false' since the types 'any[]' and 'boolean' have no overlap."

'5' - 3
// JS: 2 (string implicitno pretvoren u broj)
// TS: Greška u strict režimu → "Operator '-' cannot be applied to types 'string' and 'number'"

{} + []
// JS: 0 ({} kao blok, +[] == 0)
// TS: Greška → "Expression expected" (jer TS striktno razlikuje block vs object literal)

NaN === NaN
// JS: false
// TS: dozvoljeno, ali linter upozorava: “Use Number.isNaN() instead of comparing with NaN”
```