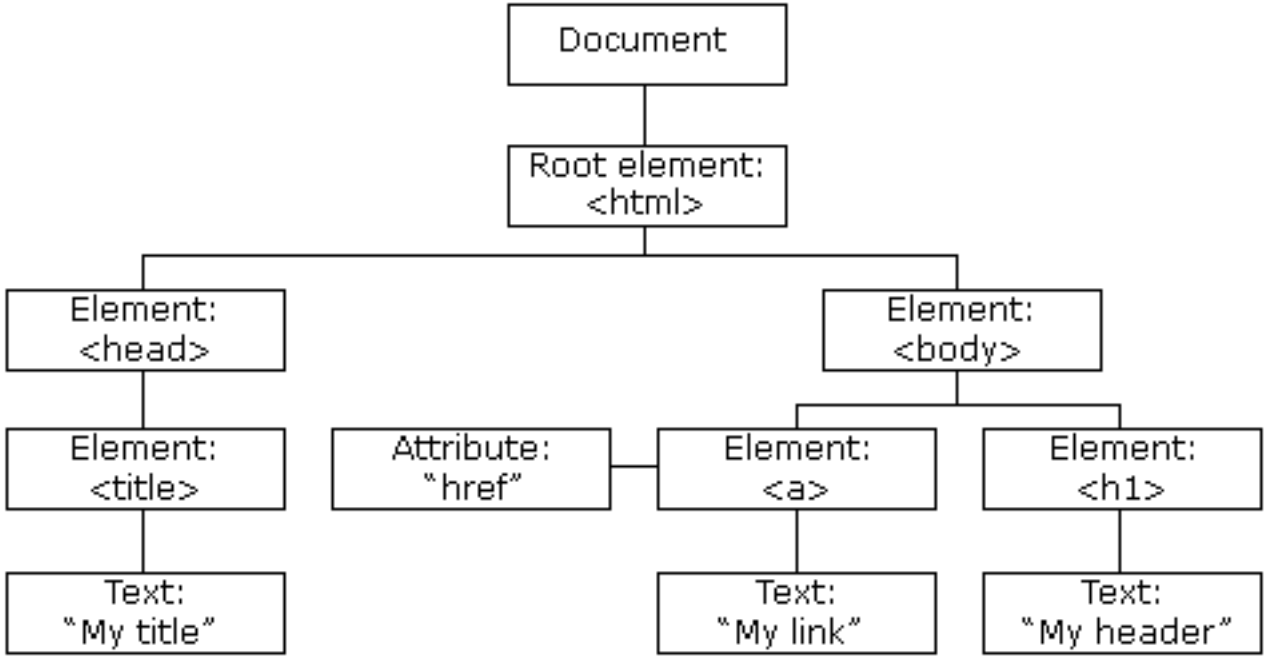
Objektni model dokumenta (DOM)

HTML DOM

- Kada se web stranica učita u browser kreira se objektni model strane i objekat **document**
- HTML DOM (Document Object Model) obezbeđuje programski interfejs za pristup i manipulaciju html dokumentima
- HTML elemente definiše kao objekte
- Definiše metode za pristup HTML elementima
- Definiše događaje za sve HTML elemente

HTML DOM stablo



HTML DOM model se sastoji od stabla objekata

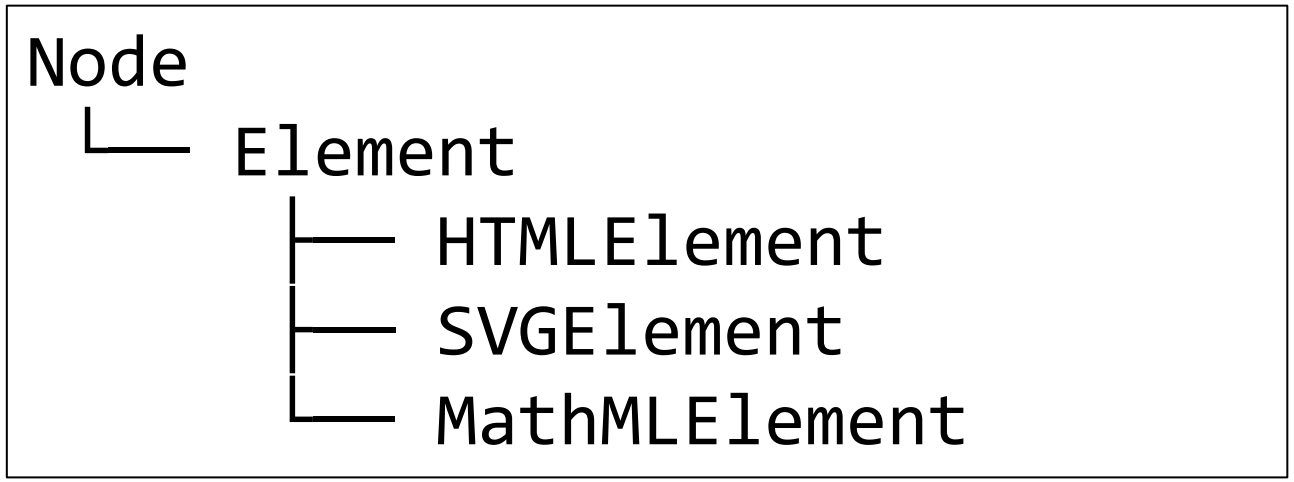
Objekat Node

- **Node** je osnovni objekat u DOM-u koja predstavlja bilo koji deo dokumenta
- Node je apstraktna klasa u JavaScript-u
- To može biti element, tekst, komentar ili čak ceo dokument
- Vrste Node objekata:
  - **Element Node**: predstavlja HTML element
  - **Text Node**: predstavlja tekst unutar elementa
  - **Comment Node**: predstavlja HTML komentar
  - **Document Node**: predstavlja celokupni HTML dokument

Svojstva objekta Node

- nodeType (npr. 1 = element, 3 = tekst, 8 = komentar))
- nodeName (npr. div, p , h1)
- parentNode – vraća roditeljski čvor
- childNodes – vraća kolekciju (listu) podređenih čvorova

Vrste Node objekata



Objekat Element

- **Nod**e je osnovni tip, dok je **Element** njegov podtip
- **Element** je konkretna klasa koja nasleđuje klasu **Node**
- Element se takođe može se smatrati i interfejsom koji definiše zajedničke karakteristike elemenata
- Element je specifičan tip Node objekta koji predstavlja HTML ili XML element
- Svi **Element** objekti su Node objekti, ali nije svaki Node objekat i **Element** objekat

Svojstva objekta Element

- **id** – jedinstveni identifikator elementa
- **className** – naziv klase elementa
- **innerHTML** – HTML sadržaj unutar elementa
- **attributes** – lista atributa elementa
- **tagName** – ime HTML taga (npr. div, p, h1)
- **children** – kolekcija podređenih Element čvorova
- **style** – omogućava pristup CSS svojstvima elementa

Metode objekta Element

- **appendChild**(child) – dodaje novi čvor kao poslednji podređeni čvor
- **removeChild**(child) – uklanja navedeni podređeni čvor
- **setAttribute**(name, value) – postavlja ili menja vrednost atributa
- **getAttribute**(name) – vraća vrednost određenog atributa
- **querySelector**(selector) – vraća prvi element koji odgovara CSS selektoru
- **querySelectorAll**(selector) – vraća kolekciju svih elemenata koji odgovaraju selektoru

Objekat HTMLElement

- **HTMLElement** je osnovni objekat u JavaScript-u koji predstavlja HTML element u DOM strukturi
- Nasleđuje svojstva i metode iz Element i Node objekata
- **HTMLElement** proširuje Element sa svojstvima i metodama specifičnim za HTML, tj. za prikaz i ponašanje elemenata u browseru
- Svi HTML elementi, kao što su , , <p>, <a>, itd., su instance **HTMLElement** objekta

Svojstva objekta HTMLElement

- **element.id** – čita ili postavlja id atribut elementa
- **element.style** – čita ili postavlja stilove elementa
- **element.innerHTML** - čita ili postavlja HTML sadržaj unutar elementa
- **element.innerText** - se koristi za dobijanje ili postavljanje vidljivog (tekstualnog) sadržaja HTML elementa
- **element. className** – dohvata ili postavlja klasu elementa
- **element.attributes** vraća kolekciju atributa elementa
- **element.hidden** – određuje da li je element vidljiv
- **element.title** – prikazuje pomoćni tekst (tooltip) pri prelasku mišem

Metode objekta HTMLElement

- **appendChild**(childNode) - dodaje novi čvor kao poslednji čvor nadređenog elementa
- **removeChild**(childNode) - uklanja podređeni čvor iz roditeljskog elementa
- **setAttribute**(name, value) - postavlja vrednost atributa na elementu
- **getAttribute**(name) - vraća vrednost atributa sa datim imenom
- **focus**() - postavlja fokus na element (ako je to moguće)
- **getElementsByTagName**(name) - vraća kolekciju svih podređenih elemenata sa datim nazivom tag-a
- **getElementsByClassName**(className) - vraća kolekciju svih podređenih elemenata sa datom klasom
- **querySelector(selector)** – vraća prvi element koji odgovara CSS selektoru
- **querySelectorAll(selector)** – vraća kolekciju svih elemenata koji odgovaraju selektoru

Kolekcije **HTMLCollection** i **NodeList**

- Kolekcije omogućavaju **rad sa grupom elemenata**
- Kada HTML DOM metoda vrati više elemenata, rezultat se čuva u kolekciji
- Kolekcija je objekat koji sadrži više DOM elemenata
- Elementima u kolekciji se pristupa pomoću indeksa: kolekcija[0], kolekcija[1], ...
- Kolekcije se mogu prolaziti petljom for ili for...of
- **HTMLCollection**
  - Sadrži samo HTML elemente
  - Ažurira se automatski ako se DOM promeni (živa kolekcija)
  - Vraćaju metode getElementsByTagName() i getElementsByClassName()
- **NodeList**
  - Može sadržati različite tipove čvorova
  - Ne ažurira se automatski – ostaje statična kopija DOM-a
  - vraća metoda querySelectorAll()

Svojstva objekta **document**

- Objekat **document** u JavaScriptu predstavlja celokupan HTML dokument koji je trenutno učitan u browseru
- **document.title -** vraća ili postavlja naslov dokumenta
- **document.body** - vraća referencu na element <body> dokumenta
- **document.head -** vraća referencu na element <head> dokumenta
- **document.URL -** vraća URL trenutnog dokumenta
- **document.cookie** – vraća ili postavlja kolačiće povezane s dokumentom
- **document.forms** – vraća kolekciju svih <form> elemenata u dokumentu

Metode objekta document

- **document.getElementById(id)** – vraća element sa datim ID-om, povratna vrednost je **Element**
- **document.getElementsByClassName(className)** – vraća kolekciju (**HTMLCollection**) svih elemenata sa datom klasom
- **document.getElementsByTagName(tagName)** – vraća kolekciju (**HTMLCollection**) svih elemenata sa datim nazivom taga
- **document.querySelector(selector)** – vraća prvi element koji odgovara datom CSS selektoru (povratna vrednost: Element ili null)
- **document.querySelectorAll(selector)** – vraća kolekciju svih elemenata koji odgovaraju CSS selektoru (povratna vrednost: **NodeList**)

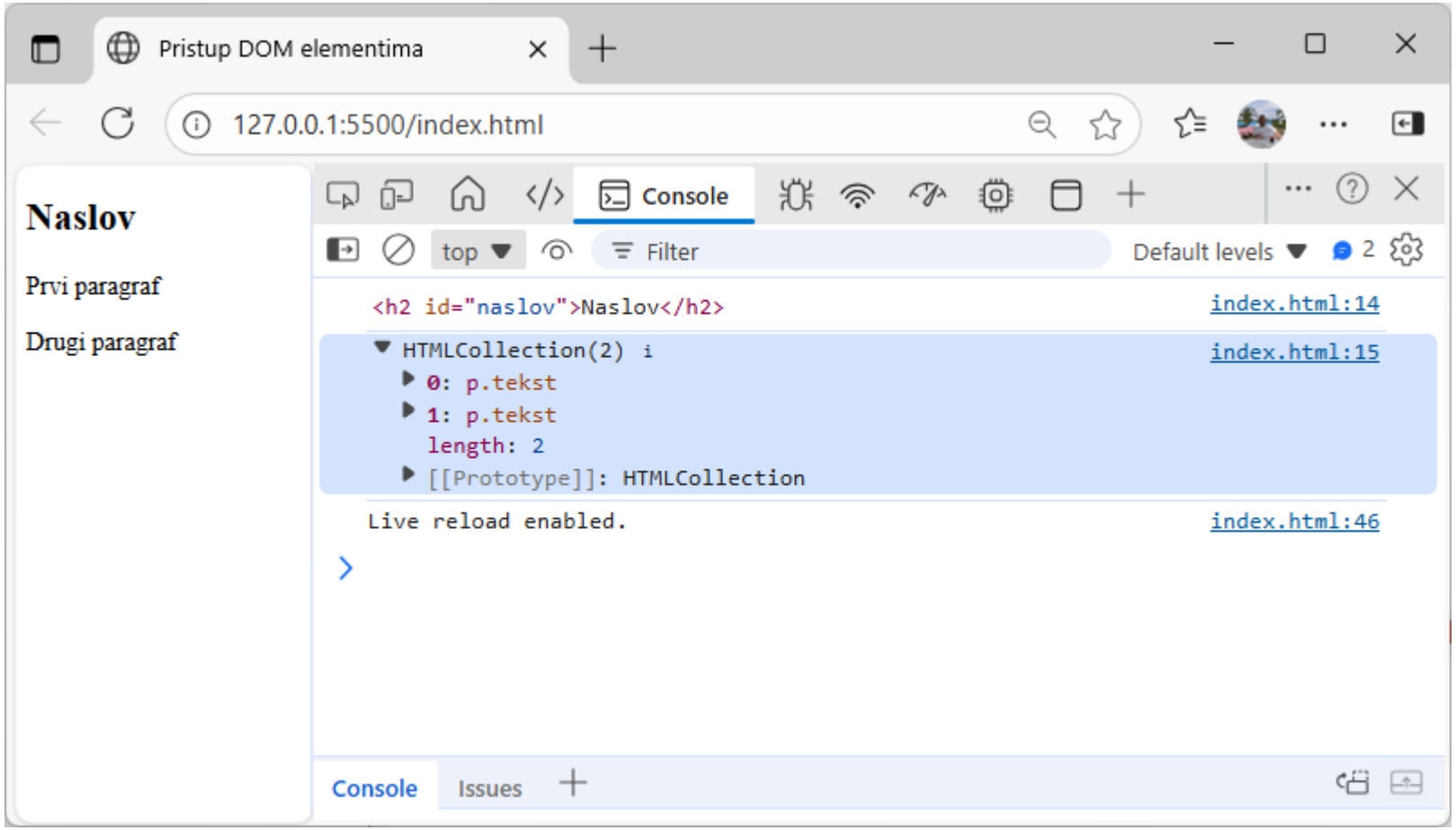
Izvedene klase iz klase HTMLElement

- **HTMLDivElement** - predstavlja element
- **HTMLSpanElement** - predstavlja element
- **HTMLAnchorElement** - predstavlja <a> (link) element
- **HTMLImageElement** - predstavlja <img> element (sliku)
- **HTMLButtonElement** - predstavlja <button> element
- **HTMLInputElement** - predstavlja <input> polje
- **HTMLFormElement** - predstavlja <form> element

Pristup elementima kroz DOM

```
<body>
    <h2 id="naslov">Naslov</h2>
    <p class="tekst">Prvi paragraf</p>
    <p class="tekst">Drugi paragraf</p>


    <script>
        let h2 = document.getElementById("naslov");
        let paragrafi = document.getElementsByClassName("tekst");
        console.log(h2);
        console.log(paragrafi);
    </script>


</body>
```

Pristup elementima kroz DOM



Promena sadržaja HTML elemenata

```
<body>
    <h2 id="naslov">Zdravo svete!</h2>

    <script>
        let naslov = document.getElementById("naslov");
        naslov.innerText = "Tekst je promenjen!";
        naslov.style.color = "blue";
        naslov.style.fontSize = "28px";
    </script>
</body>
```

```html
<body>
  <h2 id="naslov">Naslov stranice</h2>
  <p class="tekst">Prvi paragraf</p>
  <p class="tekst">Drugi paragraf</p>

  <script>
    // 1. Dohvatanje po ID-u
    let h2 = document.getElementById("naslov");
    console.log("getElementById:", h2);

    // 2. Dohvatanje po imenu taga
    let paragrafi = document.getElementsByTagName("p");
    console.log("getElementsByTagName:", paragrafi);

    // 3. Dohvatanje po klasi
    let tekstovi = document.getElementsByClassName("tekst");
    console.log("getElementsByClassName:", tekstovi);

    // 4. Primer rada sa kolekcijom
    tekstovi[0].style.color = "blue";
    tekstovi[1].innerText = "Ovo je izmenjen drugi paragraf.";
  </script>
</body>
```

Događaji u JavaScript-u

- Događaji (events) su akcije ili promene koje se dešavaju u browseru
- JavaScript može reagovati na te događaje izvršavanjem funkcije
- Najčešće potiču od korisnika (klik, unos, pomeranje miša...), ali mogu biti i sistemski (učitavanje stranice, promena veličine prozora itd.)
- Svaki događaj ima rukovaoca (event handler) – funkciju koja se izvršava kada se dogodi događaj

Osnovni događaji u DOM-u

Kategorija	**Događaji**	Opis
🖱️ **Događaji miša**	onclick, ondblclick, onmouseover, onmouseout, onmousedown, onmouseup	Reaguju na klikove i kretanje miša
⌨️ **Događaji tastature**	onkeydown, onkeyup, onkeypress	Aktiviraju se kada korisnik pritisne ili otpusti taster
📄 **Događaji dokumenta / stranice**	onload, onresize, onscroll, onunload	Odnose se na učitavanje i promene prikaza stranice
🔤 **Događaji unosa**	oninput, onchange	Reaguju na promene vrednosti u elementima (npr. poljima za unos)
🧭 **Događaji fokusa**	onfocus, onblur	Aktiviraju se kada element dobije ili izgubi fokus

Definisanje događaja u JavaScript-u

- Kao HTML atributi
  - **onclick="funkcija()"** – klasičan, stariji način
- Putem svojstva elementa
  - **element.onclick = funkcija** – definisano u JavaScript-u
- Pomoću metode addEventListener()
  - **element.addEventListener**("click", funkcija) – savremeni, preporučeni način

Događaji kao HTML atributi

```
<body>

  <h2 onclick="promeniTekst()">Klikni na mene</h2>

  <script>
    function promeniTekst() {
      document.querySelector("h2").innerText = "Klik registrovan!";
    }
  </script>

</body>
```

Događaji putem svojstva elementa

```
<body>


  <h2 id="naslov">Klikni na mene</h2>


  <script>
    let naslov = document.getElementById("naslov");


    naslov.onclick = function() {
      naslov.innerText = "Klik registrovan!";
    };
  </script>


</body>
```

Definisanje događaja pomoću metode addEventListener()

- Savremen i preporučeni način definisanja događaja
- Omogućava dodavanje više funkcija na isti događaj
- Događaj i funkcija se povezuju pomoću metode: **element.addEventListener("dogadjaj", funkcija)**
- Funkcija koja reaguje na događaj odnosno hendler događaja može biti:
  - Anonimna funkcija – jednostavna i koristi se samo na tom mestu
  - Imenovana funkcija – može se ponovo koristiti ili kasnije ukloniti

Prtplata na anonimnu funkciju

```
<body>

  <button id="dugme">Klikni me</button>
  <script>
    let dugme = document.getElementById("dugme");

    dugme.addEventListener("click", function() {
      alert("Događaj pokrenut!");
    });
  </script>

</body>
```

Pretplata na imenovanu funkciju

```
<body>
  <button id="dugme">Klikni me</button>
  <script>
    let dugme = document.getElementById("dugme");

    //Anonimna funkcija
    dugme.addEventListener("click", function() {
      alert("Anonimna funkcija pokrenuta!");
    });

    // Imenovana funkcija
    function prikaziPoruku() {
      alert("Imenovana funkcija pokrenuta!");
    }

    dugme.addEventListener("click", prikaziPoruku);
  </script>
</body>
```

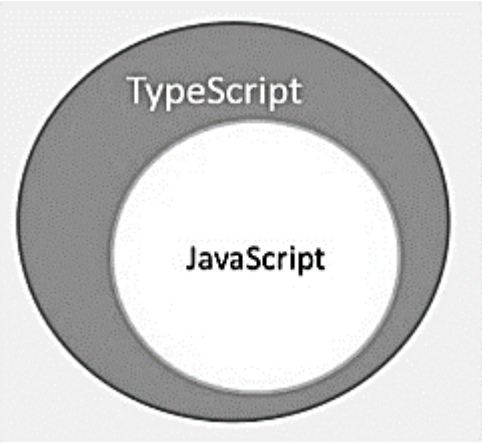
Metoda removeEventListener()

```
<body>

  <button id="dugme">Klikni me</button>

  <script>
    let dugme = document.getElementById("dugme");

    function prikaziPoruku() {
      alert("Klik registrovan!");
      dugme.removeEventListener("click", prikaziPoruku);
    }


    dugme.addEventListener("click", prikaziPoruku);
  </script>

</body>
```

Uvod u TypeScript

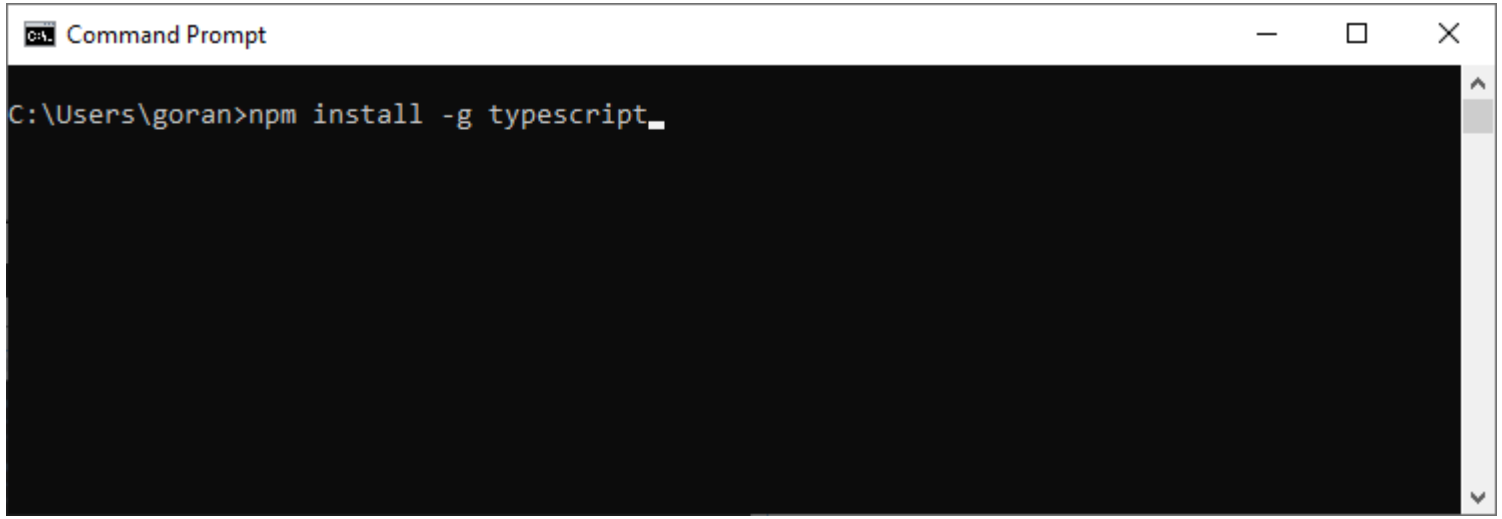
TypeScript

- **TypeScript** je objektno orijentisan programski jezik
- **TypeScript** je tipizirani nadskup JavaScript-a
- Kod napisan u **TypeScript-u** ne može se izvršiti direktno, već se prevodi u JavaScript



Instaliranje TypeScript-a

```
npm install -g typescript
```

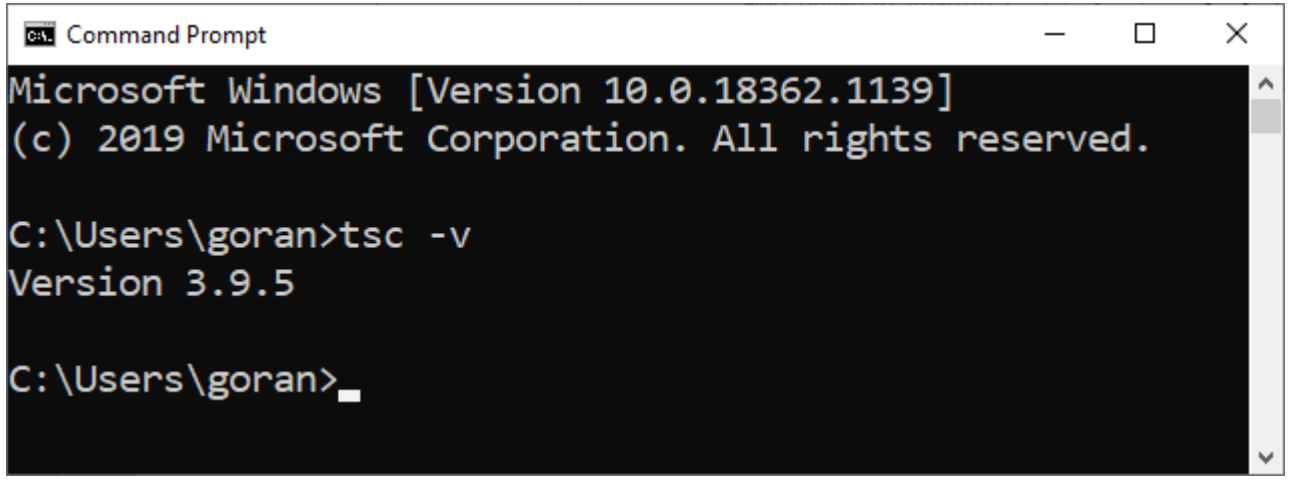


Skraćena verzija komande:

```
npm i -g typescript
```

Provera TypeScript verzije

```
tsc -v
```



Kreiranje konfiguracionih fajlova

```
npm init -y      -  kreira package.json fajl
tsc --init       -  kreira tsconfig.json fajl
```

- Fajl **package.json** sadrži osnovne informacije o projektu: naziv, verziju, autora, skripte i zavisnosti (dependencies)
  - Zahvaljujući fajlu package.json, Node.js i npm znaju koje pakete treba instalirati i kako pokrenuti projekat
- Fajl **tsconfig.json** sadrži podešavanja TypeScript kompajlera uključujući:
  - gde se nalaze izvorišni .ts fajlovi
  - gde se čuvaju prevedeni .js fajlovi
  - koji standard JavaScript-a da koristi (ES6, ESNext itd.)
  - da li se dozvoljavaju stroge provere tipova

Prikaz terminala

```
PS C:\Users\goran\Desktop\it05> npm init -y
Wrote to C:\Users\goran\Desktop\it05\package.json:

{
  "name": "it05",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
```

```
PS C:\Users\goran\Desktop\it05> tsc --init
message TS6071: Successfully created a tsconfig.json file.
PS C:\Users\goran\Desktop\it05>
```

Tipovi podataka u TypeScript-u

- **any** – predstavlja bilo koji tip podataka; isključuje proveru tipova.
- **number** – celi i realni brojevi (TypeScript ne razlikuje int i float)
- **string** – tekstualni podaci, sekvenca Unicode karaktera unutar navodnika ("tekst" ili 'tekst’).
- **boolean** – logička vrednost, može biti true ili false
- **void** – koristi se kao povratni tip funkcija koje ne vraćaju vrednost
- **null** – označava objekat koji namerno nema vrednost
- **undefined** – vrednost dodeljena neinicijalizovanoj promenljivoj

Upotreba okruženja VS Code

- Kreira se fajl sa ekstenzijom .ts npr. *primer01.ts*
- U fajl se upisuje TypeScript kod
- Kompajlira se u terminalu pomoću komande:
  tsc primer01.ts
- Dobijeni JavaScript fajl se izvršava pomoću Node-a:
  node primer01.js

Deklaracija i inicijalizacija promenljivih

```
let ime: string;                    // deklaracija promenljive
let prezime: string = 'Marković'; // deklaracija i inicijalizacija
const grad: string = 'Beograd';    // konstanta (ne može se ponovo dodeliti vrednost)
let osoba = 'Marko';                // tip se automatski određuje (string)
```

TypeScript moduli

- Fajl u TypeScript-u se smatra skriptom ako nema export ili import naredbu, u tom slučaju njegov kod ima globalni opseg
- Fajl se tretira kao **modul** samo ako sadrži **export** ili **import**
- Kod unutar modula ima svoj lokalni opseg – promenljive, funkcije i klase nisu automatski dostupne u drugim fajlovima
- Da bi kod iz jednog fajla bio vidljiv u drugom, mora se izvesti pomoću export i zatim uvesti pomoću import
- Modul se kreira pomoću ključne reči export, a koristi pomoću import
- Moduli omogućavaju organizovan, izolovan i lako održiv kod, bez konflikata imena između fajlova

TypeScript promenljive

```typescript
export {}; // čini fajl modulom, sprečava globalni opseg


let ime: string = 'Marko';
let a: number = 50;
let b: number = 42.5;
let suma: number = a + b;


console.log('Ime:', ime);
console.log('Prvi broj:', a);
console.log('Drugi broj:', b);
console.log('Suma je:', suma);
```

```
PS C:\Users\goran\Desktop\it05> tsc primer02
PS C:\Users\goran\Desktop\it05> node primer02
Ime: Marko
Prvi broj: 50
Drugi broj: 42.5
Suma je: 92.5
PS C:\Users\goran\Desktop\it05>
```

Operatori u TypeScript-u

- TypeScript koristi iste operatore kao JavaScript
- Nema novih operatora niti promena u njihovom ponašanju
- Sve operacije se izvršavaju **na isti način** kao u JavaScript-u
- Jedina razlika je što TypeScript omogućava proveru tipova prilikom korišćenja operatora
- Time sprečava greške koje bi JavaScript dozvolio prilikom izvršavanja
- Ako se operator koristi nad nekompatibilnim tipovima, TypeScript to može prijaviti **tokom kompajliranja**, pre nego što se kod izvrši

Operatori u TypeScript-u

```
let x: number = 5;
let y: string = "10";


let rezultat1:number = x + y;  // Greška: nije dozvoljeno sabirati number i string
let rezultat2 = x + Number(y); //  Ispravno: eksplicitna konverzija tipa
```

TypeScript funkcije

- Funkcije u TypeScript-u rade kao u JavaScript-u,ali omogućavaju definisanje tipova za:
  - parametre funkcije – vrednosti koje funkcija prima
  - povratnu vrednost – vrednost koju funkcija vraća pomoću naredbe **return**
- TypeScript proverava da li funkcija prima i vraća vrednosti odgovarajućih tipova

TypeScript funkcije

```
export {}; // fajl je modul


function saberi1(x: number, y: number): number {
  return x + y;
}


function saberi2(x: number, y: number): void {
  console.log('Rezultat je:', x + y);
}

let rezultat: number = saberi1(5, 6.1);
console.log(rezultat);

saberi2(6.1, 7.8);
```

Funkcija sa opcionim parametrom

- Opcionim parametrima u TypeScript-u se označavaju parametri koji nisu obavezni pri pozivu funkcije
- Označavaju se znakom ? iza imena parametra
- Ako se opcioni parametar ne prosledi, njegova vrednost je undefined
- U TypeScript-u opcioni parametar (?) mora biti definisan posle svih obaveznih parametara

Funkcija sa opcionim parametrom

```
export { };

function prikazi(ime: string, prezime: string, email?: string): void
{
    console.log('Ime:', ime);
    console.log('Prezime:', prezime);
    if (email != undefined) {
        console.log('Email:', email);
    }
    console.log('_________________');
}
prikazi('Marko', 'Markovic');
prikazi('Jovan', 'Jovanovic', 'jovan@gmail.com');
```

```
Ime: Marko
Prezime: Markovic
_________________
Ime: Jovan
Prezime: Jovanovic
Email: jovan@gmail.com
_________________
```

Podrazumevana vrednost parametra funkcije

```
export { };
function racunaj(cena: number, popust: number = 0.2): void {
    let konacnaCena = cena * (1 - popust);
    console.log('Konacna cena:', konacnaCena);
}
racunaj(1000);
racunaj(1000, 0);
racunaj(1000, 0.3);
```

```
goran@DESKTOP-ESB9HU8 MINGW64 ~/Desktop/IT05
$ node primer07
Konacna cena: 800
Konacna cena: 1000
Konacna cena: 700
```

Anonimna funkcija

- Anonimna funkcija je funkcija bez imena
- Definiše se unutar izraza i može se dodeliti promenljivoj, proslediti kao argument ili vratiti iz druge funkcije

```
export{};
let zbir = function (a: number, b: number): void {
    console.log('Zbir je:', a + b);
};
zbir(5, 6);
```

Lambda izrazi za definisanje anonimnih funkcija

- Lambda (ili arrow) funkcije su skraćeni oblik anonimnih funkcija
- Koriste operator **=>** umesto ključne reči function
- Najčešće se koriste kada je funkcija kratka ili se prosleđuje kao argument

Lambda izrazi za definisanje anonimnih funkcija

```
export{};

let prikaziVreme = ()=> new Date().toLocaleTimeString();
let vreme = prikaziVreme();
console.log(vreme);
let uvecaj10 = (x:number)=>x+10;
let r:number = uvecaj10(2);
console.log("rezultat",r);
```

```
goran@DESKTOP-ESB9HU8 MINGW64 ~/Desktop/IT05
$ tsc primer09

goran@DESKTOP-ESB9HU8 MINGW64 ~/Desktop/IT05
$ node primer09
20:34:15
rezultat 12
```

Nizovi

```
export{};

const brojevi:number[] = [1,3,5,7,9];

//for
for (let i = 0; i < brojevi.length; i++) {
    console.log(brojevi[i]);
}
//for...in
console.log('--------------');
for (const i in brojevi) {
    console.log(i,brojevi[i]);
}
console.log('--------------');

//for...of
for (const i of brojevi) {
    console.log(i);
}
//forEach
console.log('--------------');
brojevi.forEach(function (x:number) {
    console.log(x);
});
```

```
const brojevi: Array<number> = [1,3,5,7,9];
```

Filtriranje niza – funkcija filter

```
export{};

const a:number[] = [12,5,8,130,44];

// izdvajanje brojeva većih od 20
const b = a.filter(x=>x>20);

for (const i of b) {
    console.log(i);
}
```

```
$ node primer11
130
44
```

Transformacija niza – funkcija map

```
export{};
const a:number[] = [12,5,8,130,44];
const c = a.map(x=>x*x);

for (const i in c) {
    console.log(i,c[i]);
}
```

```
$ node primer12
0 144
1 25
2 64
3 16900
4 1936
```

Typescript klase

- TypeScript koristi klase kako bi omogućio upotrebu OOP principa kao što su:apstrakcija, enkapsulacija, nasleđivanje i polimorfizam.
- Klasa se sastoji od:
  - polja (properties) — podaci koje objekat čuva
  - metoda (methods) — funkcije koje objekat izvodi
  - konstruktora (constructor) — funkcija koja se poziva pri kreiranju objekta
- Podrazumevano, polja klase su public, osim ako se ne navede drugačiji modifikator (private, protected)
- Klase u TypeScript-u se kompajliraju u JavaScript funkcije — time su potpuno kompatibilne sa starijim verzijama JS-a
- Korišćenjem klasa, TypeScript uvodi tipizaciju i strukturu u JavaScript kod

Klase

```
export{};
class Osoba {
        ime:string;
        prezime: string;
        starost:number;
        constructor(ime:string, prezime: string, starost:number ) {
            this.ime = ime;
            this.prezime = prezime;
            this.starost = starost;
        }

        stampaj1():void
        {
            console.log('Ime',this.ime);
            console.log('Prezime', this.prezime);
            console.log('Starost',this.starost);
        }

        stampaj2():string
        {
            return this.ime + " " + this.prezime;
        }
      }
    const os1 = new Osoba('Marko','Markovic',25);
    os1.stampaj1();
    console.log(os1.stampaj2());
```

```
$ node primer13
Ime Marko
Prezime Markovic
Starost 25
Marko Markovic
```

Automatsko dodeljivanje parametara konstruktora svojstvima klase

```
export { };
class Osoba {
    constructor(public ime: string, public prezime: string, public starost: number) {
    }
    stampaj(): void {
        console.log(this.ime, this.prezime, this.starost);
    }
}
const os1 = new Osoba('Marko', 'Markovic', 25);
os1.stampaj();
```

Klasa sa privatnim poljima

primer15.ts

```
export class Osoba {
    constructor(private ime: string, private prezime: string, private starost: number) {
    }
    stampaj(): void {
        console.log(this.ime, this.prezime, this.starost);
    }
}
```

primer16.ts

```
import { Osoba } from "./primer15";

const os1 = new Osoba('Marko','Markovic',25);
os1.stampaj();
```

Definisanje gettera/settera

```
export { };
class Osoba {
    private _ime: string = '';
    //getter
    public get ime(): string {

        return this._ime;
    }

    //setter
    public set ime(novoIme: string) {
        if (novoIme.length > 10) {
            console.log('Ime ne moze imati vise od 10 karaktera');
            return;
        }
        this._ime = novoIme;
    }
}
const os = new Osoba();
os.ime = 'Marko231334353465754 74';
console.log(os.ime);
```

tsc --target es5 primer17

TypeScript se kompajlira u ES5 ili noviji standard da bi `get` i `set` radili korektno

Interfejsi

- Interfejs u TypeScript-u definiše skup svojstava i metoda koje neki objekat ili klasa mora da poseduje, ali ne sadrži njihovu implementaciju
- Kaže šta objekat mora da ima, ali ne kako to treba da bude implementirano
- Može se koristiti za:
  - definisanje tipa promenljive
  - opisivanje strukture objekata
  - definisanje “ugovora” koji klase treba da slede
- Interfejs sadrži samo potpis svojstava i metoda (bez implementacije)
- Klasa koja implementira interfejs mora da obezbedi implementaciju svih definisanih članova.
- TypeScript kompajler ne prevodi interfejse u JavaScript — oni postoje samo u fazi provere tipova (duck typing).

Interfejs - primer

```ts
export{};
interface IOsoba {
    id: number;
    ime: string;
    prezime: string;
}
const os1: IOsoba = {
    id: 1,
    ime: "Marko",
    prezime: "Markovic"
};
```

.ts fajl

tsc

```js
const os1 = {
    id: 1,
    ime: "Marko",
    prezime: "Markovic"
};
```

.js fajl

Interfejs sa opcionim i read-only svojstvima

```
export {};
interface IOsoba {
  readonly id: number;    // svojstvo koje se ne može menjati
  ime: string;
  prezime: string;
  email?: string;         // opciono svojstvo
}

const os1: IOsoba = {
  id: 1,
  ime: "Marko",
  prezime: "Markovic"
};
// ispis
console.log(os1);
// pokušaj izmene readonly polja
// os1.id = 2; // Greška: Cannot assign to 'id' because it is a read-only property
// opciono svojstvo se može dodati kasnije
os1.email = "marko@gmail.com";
console.log(os1);
```

Interfejs kao tip parametra funkcije

```
export{};

interface IOsoba {
    id: number;
    ime: string;
    prezime: string;
}

function Stampaj(os: IOsoba) {
    console.log(os.id, os.ime, os.prezime);
}

const os1= {
    id: 1,
    ime: "Marko",
    prezime: "Markovic",
    adresa: "Cara Dusana 22"
};
Stampaj(os1);
```

Klasa bazirana na interfejsu

```
interface IOsoba {
  ime: string;
  prezime: string;
}
class Student implements IOsoba {
  // 'ime' i 'prezime' su deo interfejsa, 'smer' je dodatno svojstvo
  constructor(
    public ime: string,
    public prezime: string,
    public smer: string
  ) {}

  stampaj(): void {
    console.log('Ime:', this.ime);
    console.log('Prezime:', this.prezime);
    console.log('Smer:', this.smer);
  }
}
const st = new Student("Marko", "Markovic", "Informacioni sistemi");
st.stampaj();
```

Pitanje 1

Koji je osnovni objekat DOM stabla?

a. Element
b. Node
c. Attribute

Odgovor: b

Pitanje 2

Koja od sledećih klasa nasleđuje klasu Element i predstavlja osnovu za sve HTML elemente?

a. Node
b. HTMLElement
c. Document

Odgovor: b

Pitanje 3

Šta je document objekat u JavaScript-u?

a. Objekat koji predstavlja samo <body> deo stranice
b. Objekat koji predstavlja samo <head> deo stranice
c. Objekat koji predstavlja ceo HTML dokument

Odgovor: c

Pitanje 4

Kojim svojstvom pristupamo HTML sadržaju elementa?

a. innerText
b. innerHTML
c. outerHTML

Odgovor: b

Pitanje 5

Kojom se metodom pristupa elementu na osnovu ID atributa u DOM stablu??

a. document.getElementById("id")
b. document.getElement("id")
c. document.querySelectorAll("id")

Odgovor: a

Pitanje 6

Prevođenje TypeScript koda koji se nalazi u fajlu primer01.ts u JavaScript fajl primer01.js vrši se korišćenjem sledeće komande:

a. compile primer01
b. tsc primer01
c. node primer01

Odgovor: b

Pitanje 7

Polja u TypeScript klasama imaju podrazumevani modifikator pristupa:

a. public
b. private
c. protected

Odgovor: a

Pitanje 8

Konstruktor TypeScript klase je funkcija koja ima:

a. isto ime kao i klasa
b. ime constructor
c. ime get

Odgovor: b

Pitanje 9

Unutar osoba.ts fajla definisan je samo interfejs IOsoba. Šta se dobija izvršavanjem komande: tsc osoba

a. osoba.js fajl koji je prazan
b. osoba.js fajl koji sadrži interfejs
c. osoba.js fajl koji sadrži klasu

Odgovor: a

Pitanje 10

Koji uslov mora da bude ispunjen da bi se TypeScript fajl tretirao kao modul?

a. Da sadrži definiciju klase
b. Da u njemu postoji import ili export naredba
c. Da ima ekstenziju .ts

Odgovor: b