**JavaScrip Full Stack Web Development**

*Mihajlo Milojević*

Sadržaj

[Sadržaj: 3](#_Toc116911688)

[1. JavaScript (napredno) 7](#_Toc116911689)

[1.1 Konzola 7](#_Toc116911690)

[1.1.1 Console.clear 8](#_Toc116911691)

[1.1.2 Console.log 8](#_Toc116911692)

[1.1.3 Console.warn 9](#_Toc116911693)

[1.1.4 Console.error 9](#_Toc116911694)

[1.1.5 Console.trace 10](#_Toc116911695)

[1.1.6 Console.dir 11](#_Toc116911696)

[1.1.7 Console.table 12](#_Toc116911697)

[1.1.8 Console.count 13](#_Toc116911698)

[1.1.9 Console.countReset 14](#_Toc116911699)

[1.1.10 Console.group 14](#_Toc116911700)

[1.1.11 Console.groupCallapsed 15](#_Toc116911701)

[1.1.12 Console.groupEnd 15](#_Toc116911702)

[1.1.13 Console.time 15](#_Toc116911703)

[1.1.14 Console.timeLog 16](#_Toc116911704)

[1.1.15 Console.timeEnd 16](#_Toc116911705)

[1.2 Hoisting 18](#_Toc116911706)

[1.3 Let i const 19](#_Toc116911707)

[1.4 Arrow funkcije 22](#_Toc116911708)

[1.5 For in i for of 24](#_Toc116911709)

[1.6 Falsy vrednosti sa operatorima 26](#_Toc116911710)

# Sadržaj:

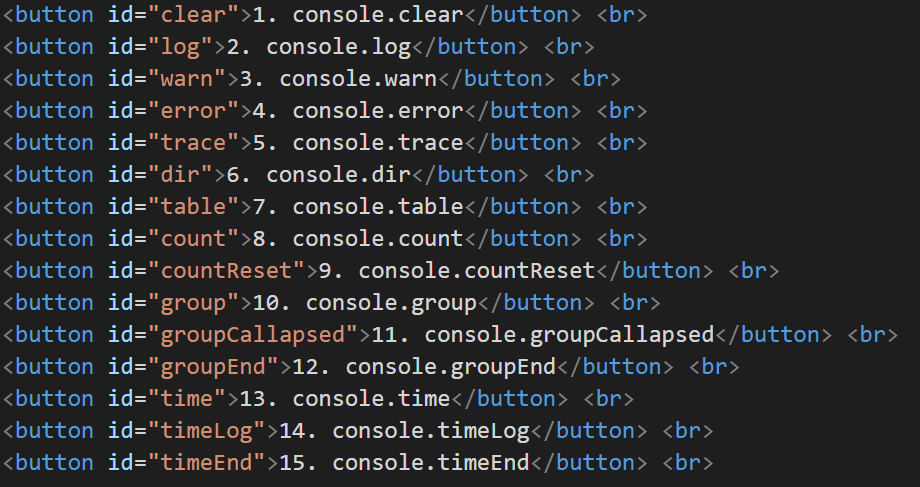
1. JavaScript (napredno)
   1. Konzola
   2. Hoisting
   3. Var, let i const
   4. Arrow funkcije
   5. *For in* i *for of*
   6. *Falsy* vrednosti, „prespajanje“ logičkih operatora, opcionalno povezivanje
   7. Metode nad stringovima
   8. Metode nad nizovima
   9. Metode nad objektima
   10. Desktruktuiranje objekata i nizova
   11. Klase
       1. Definicija
       2. Kreiranje
       3. Svojstva
       4. Metode
       5. Privatna svojstva
       6. Staticka svojstva i metode
       7. Nasledjivanje
   12. Asinhrono programiranje
       1. Funkcije povratnog poziva
       2. Obećanja (Promisi)
       3. Async/await
   13. Moduli
   14. JSON
   15. Fetch API
2. Nodejs
   1. Uvod
   2. Instalacija
   3. Globalne promenljive
   4. Moduli
   5. Ugrađeni moduli
      1. OS
      2. Path
      3. FS
      4. HTTP
   6. Veb serveri
   7. Fajl package.json
   8. Npm
   9. Folder node\_modules
   10. Instalacija paketa
   11. API
       1. Šta je api
       2. Zahtev i odgovor
       3. Metode
       4. Statusni kodovi
   12. Radni okvir Express
       1. Zašto?
       2. Početak
       3. Params i queries
       4. Middleware
          1. Šta?
          2. Kako?
          3. Cors
          4. Isporuka statičnih fajlva
       5. Rute
       6. Kontroleri
       7. Baza
       8. Autentifikacija
          1. JWT
          2. Kolačići
          3. Sesije
   13. Projekat
       1. Prikaz
       2. Projektovanje
       3. Baza
       4. Server
       5. API
       6. Autentifikacija i autorizacija
3. Git i GitHub
   1. Git
      1. Šta je git?
      2. Instalacija
      3. Konfiguracija
      4. Folder .git
      5. Repozitorijumi
      6. Izmene repozitorijuma
      7. Grane
   2. GitHub
      1. Šta je Github
      2. Početak
      3. Repozitorijumi
      4. Izmene repozitorijuma
      5. Preizimanje repozitorijuma
      6. Dodavanje saradnika
   3. Git u Visual Studio Code-u
   4. Veb hostovanje
   5. Projekat
      1. Prebacivanje koda na Githuba
      2. Hostovanje aplikacije
4. React
   1. Šta je react?
   2. Kreiranje i pokretanje projekta
   3. Struktura foldera i fajlova
   4. Renderovanje aplikacije na stranici
   5. JSX
   6. React komponente
      1. Klasne komponente
      2. Funkcionalne komponente
      3. Životni ciklus komponenata
   7. Props
   8. State
   9. Događaji
   10. Uslovno Renderovanje
   11. Liste
   12. Forme
   13. Rute
   14. Kuke
       1. useState
       2. useEffect
       3. useRef
       4. useContext
       5. Ostale
       6. Kreiranje sopstvenih kuka
   15. Projekat
5. Nextjs
   1. Uvod
   2. Kreiranje i pokretanje projekta
   3. Rute
   4. Dinamičke rute
   5. Kuka useRouter
   6. API rute
   7. SSR
   8. getStaticProps
   9. getStaticPaths
   10. getServerSideProps

# 1. JavaScript (napredno)

## 1.1 Konzola

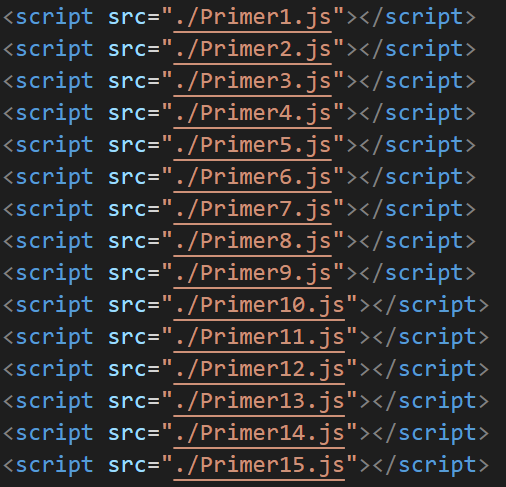
Objekat *console* je globalni objekatak koji sadrži mnoštvo metoda za debag-ovanje koga.

Da bi isprobali nekoliko najkorisnijih metoda napravićemo html stranicu sa sledećom strukturom:



*index.html*

Kod za svaku od ovih metoda smestićemo u posebne fajlove i povezati sa našim html-om pomoću sledećih scripti:

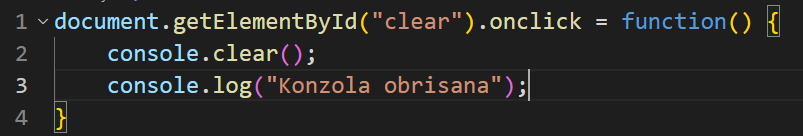


Sada možemo probati svaku od ovih metoda dodavanjem odgovarajućih *onClick* dogadjaja.

### 1.1.1 Console.clear

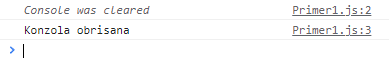
Metoda *clear* je jednostavna metoda koja briše sadržaj konzole.

Kod za prikaz ove metode je unutar fajla *Primer1.js*:



*Primer1.js*

Nakon klika na dugme „1. console.clear“ dobijamo sledeći izgled konzole:

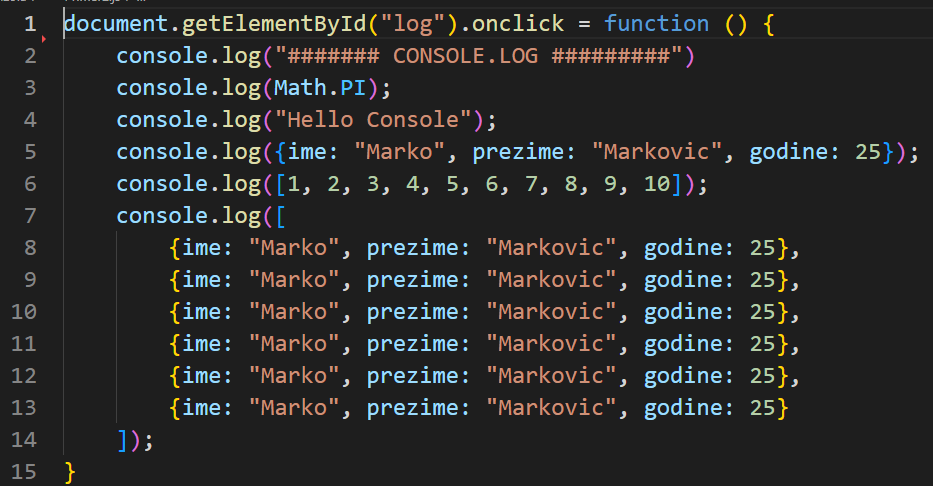


*Izgled konzole nakon pritiska na dugme „1. console.clear“*

### 1.1.2 Console.log

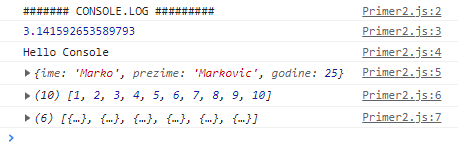
Metoda *log* je najkorišćenija i najjednostavnija. Ona prosto ispisuje ono što joj se prosledi kao parametar u konzolu.

Kod za prikaz ove metode je unutar fajla *Primer2.js*:



*Primer2.js*

A izgled konzole nakon klika na dugme „2. console.log“ je sledeći:

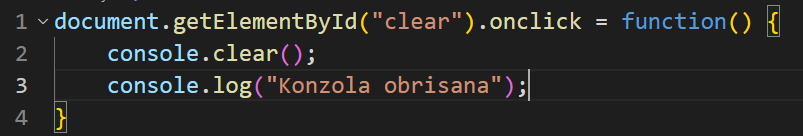


*Izgled konzole nakon pritiska na dugme „2. console.log“*

### 1.1.3 Console.warn

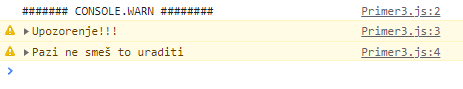
Metoda *warn* je veoma slična metodi *log*. Jedina razlika je što ona ispisuje upozorenje koje se drugačije prikazuje unutar konzole.

Kod za prikaz ove metode je unutar fajla *Primer3.js*:



*Primer3.js*

A izgled konzole nakon klika na dugme „3. console.warn“ je sledeći:

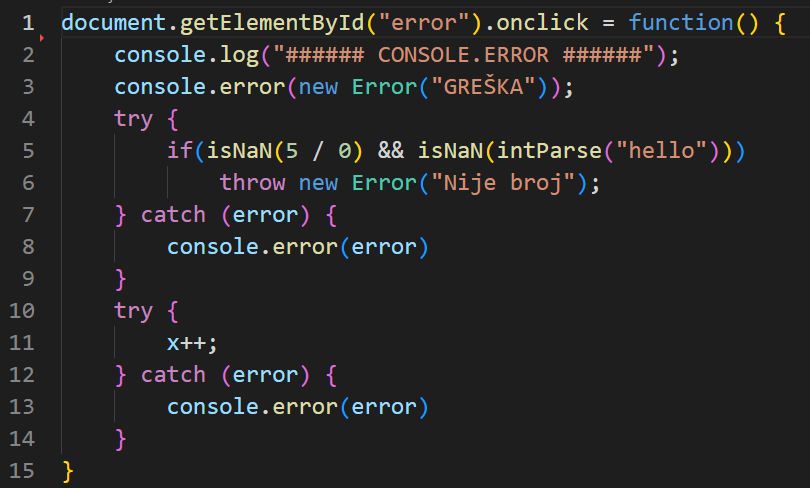


*Izgled konzole nakon pritiska na dugme „3. console.warn“*

### 1.1.4 Console.error

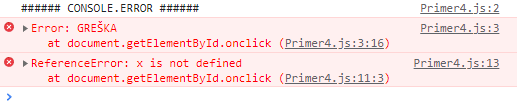
Metoda *error* je slična metodi *warn*, samo što umesto upozorenja ispisuje grešku:

Kod za prikaz ove metode je unutar fajla *Primer4.js*:



*Primer4.js*

A izgled konzole nakon klika na dugme „4. console.error“ je sledeći:

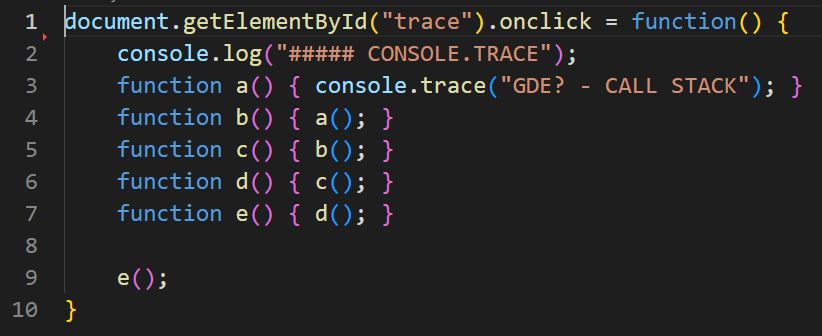


*Izgled konzole nakon pritiska na dugme „4. console.error“*

### 1.1.5 Console.trace

Metoda *trace* u konzolu ispusuje trenutni *call stack*. Ova metoda može biti veoma korisna za otkrivanje redosleda poziva ugnježdenih funkcija i otkrivana grešaka u istim.

Kod za prikaz ove metode je unutar fajla *Primer5.js*:



*Primer5.js*

A izgled konzole nakon klika na dugme „5. console.trace“ je sledeći:

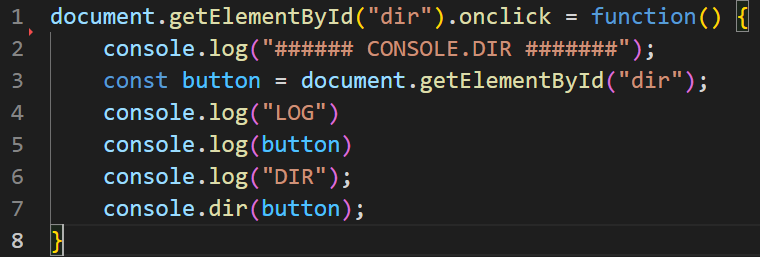


*Izgled konzole nakon pritiska na dugme „5. console.trace“*

### 1.1.6 Console.dir

Metoda *dir* u konzolu ispusuje svoje parametre kao objekte. Ovo je pogodno zato što nekad želimo da ispitamo svojstva html elemenata, a običnim logovanjem dobijamo samo html tog elementa.

Kod za prikaz ove metode je unutar fajla *Primer6.js*:



*Primer6.js*

A izgled konzole nakon klika na dugme „6. console.dir“ je sledeći:

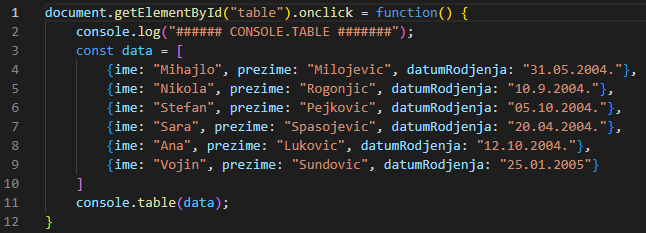


*Izgled konzole nakon pritiska na dugme „6. console.dir“*

### 1.1.7 Console.table

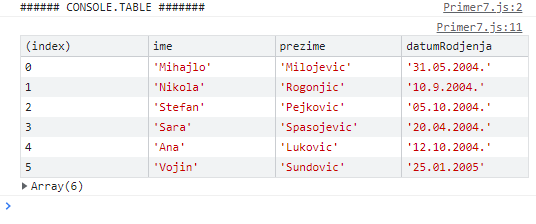
Metoda *table* tabelarno ispisuje podatke koji su prosledjeni kao parametri. Ova metoda može da bude veoma korisna za lakši prikaz nekih podataka (npr. podataka iz baze).

Kod za prikaz ove metode je unutar fajla *Primer7.js*:



*Primer7.js*

A izgled konzole nakon klika na dugme „7. console.table“ je sledeći:

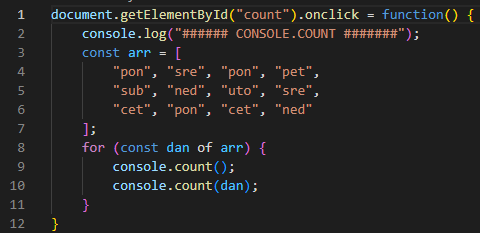


*Izgled konzole nakon pritiska na dugme „7. console.table“*

### 1.1.8 Console.count

Metoda *count* prosto broji svoja pozivanja. Kao parametar uzima naziv brojača i svaki put kada se pozove *console.count()* sa istim nazivom brojač se povećava. Ako brojač pod tim nazivom ne postoji on se kreira i povećava za 1. To znači da nije potrebno kreirati brojače pre upotrebe. Ukoliko nije prosledjen naziv metodi *count* koristi se naziv *„default“*. Ovom metodom se lako može proveriti koliko puta se izvršava neka petlja/funkcija...

Kod za prikaz ove metode je unutar fajla *Primer8.js*:



*Primer7.js*

A izgled konzole nakon klika na dugme „8. console.count“ je sledeći:



*Izgled konzole nakon pritiska na dugme „8. console.count“*

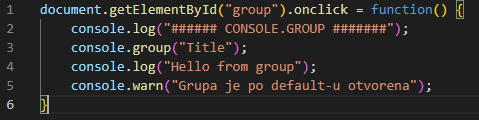
### 1.1.9 Console.countReset

Metoda *countReset*, kao što joj i naziv kaže, resetuje brojač pod nazivom koji je prosledjen, tj. *„default“* ako nije ništa prosledjeno. Ovom metodom možemo resetovati brojače nakon izvršenja odredjenog bloka koda i spremiti brojače za sledeći.

### 1.1.10 Console.group

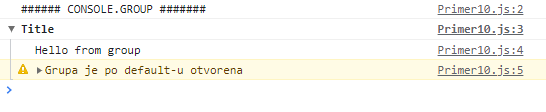
Metoda *group* otvara novu grupu pod nazivom koji je prosledjen kao parametar. Na ovaj način možemo objediniti sve naredne log-ove, warn-ove..., pa čak i druge grupe pod jednim nazivom koji možemo lako sklupljati i proširivati. Metoda *group* automatski otvara grupu. Ako želimo da grupa bude zatvorena pri kreiranju potrebno je koristiti metodu *groupCallapsed*.

Kod za prikaz ove metode je unutar fajla *Primer10.js*:



*Primer10.js*

A izgled konzole nakon klika na dugme „10. console.group“ je sledeći:



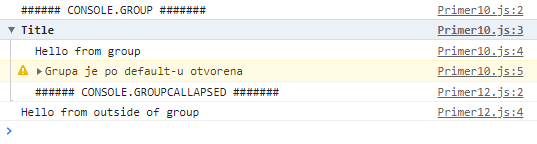
*Izgled konzole nakon pritiska na dugme „10. console.group“*

### 1.1.11 Console.groupCallapsed

Metoda *groupCallapsed* potpuno je identična metodi *group* sa jedinom razlikom da je grupa po default-u zatvorena/skupljena.

### 1.1.12 Console.groupEnd

Metoda *groupEnd* ima potpuno suprotnu upotrebu od metoda *group* i *groupCallapsed*. Ona zatvara trenutno otvorenu grupu i svi naredni ispisi u konzoli biće prikazani van grupe. GroupEnd kao parametar prima naziv grupe koju treba da zatvori.



*Prikaz konzole nakon metode groupEnd*

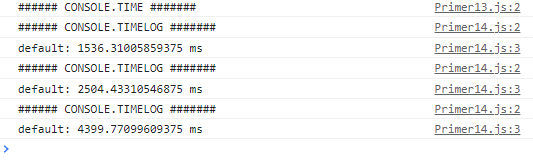
### 1.1.13 Console.time

Metoda *time* je slična metodi *count* s tim što umesto brojača kreiramo tajmer(štopericu). Štoperica je odmah pokrenuta i broji u milisekndama.

### 1.1.14 Console.timeLog

Metoda *timeLog* u konzoli ispisuje trenutno stanje nekog tajmera, bez njegovog zaustavljanja.

Pogledajte šta se ispisuje u konzoli nakon par timeLog poziva. Ne zaboravite da je prethodno pokrenut tajmer metodom *time*.

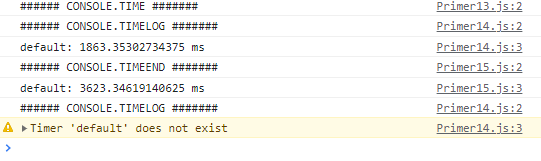


*Prikaz konzole nakon metode timeLog nekoliko puta*

### 1.1.15 Console.timeEnd

Metoda *timeEnd* ima isti ispis u konzoli kao *timeLog* samo što *timeEnd* zaustavlja odgovarajući tajmer.

Kombinacija metoda time, timeLog i timeEnd je veoma pogodna za brzo testiranje perormansi našeg koda. Pomoću njih možemo da proverimo koliko dugo se odrećeni blok izvršava.

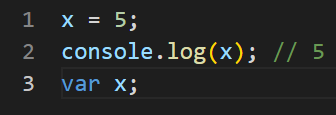


*Prikaz konzole nakon metode timeEnd*

## 1.2 Hoisting

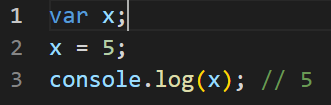
Hoisting u JavaScript-u je podizanje deklaracije na vrh.

Primer:



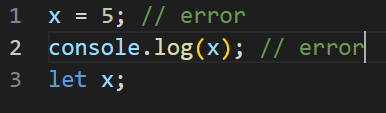
*Primer 1 - 1*

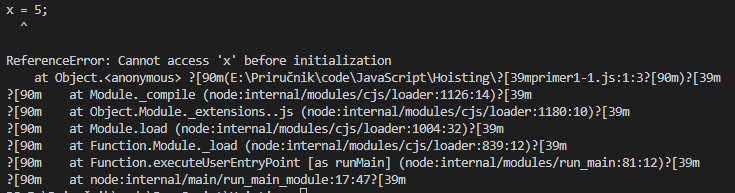
Ovde smo uspeli da dodelimo vrednost promenljive *x* i ispišemo je u konzolu pre deklarisanja. To je zato što je JavaScript podigao deklaraciju na vrh oblasti važenja promenljive *x*. Tako je da je prethodni kod identičan sledećem:



*Primer 1 - 2*

Ovo važi samo za promenljive deklarisane pomoću ključne reči *var*.

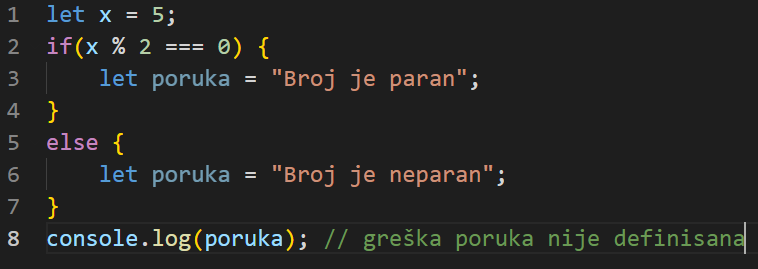


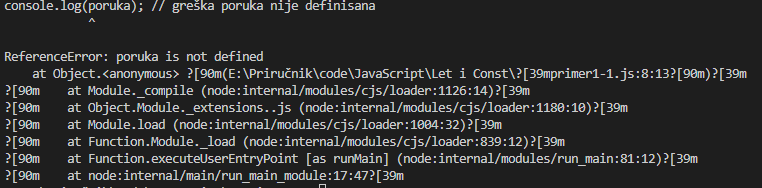


*Primer 1 – 3*

## 1.3 Let i const

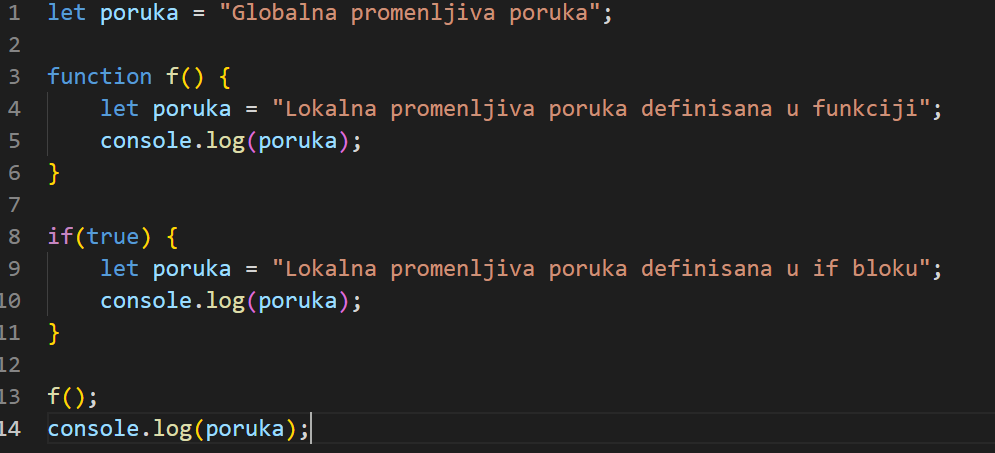
Ključna reč *let* nam omogućava da definišemo promenljive sa blokovskom oblašću važenja (za razliku od ključne reči *var* koja ima ili globalnu ili funkcijsku oblast važenja).





*Primer 1 - 1*

Ovde je promenljiva poruka definisana samo izmedju „{„ i „}“ u *if* (*else*) bloku tako da bilo koja upotreba van tog bloga nije važeća.



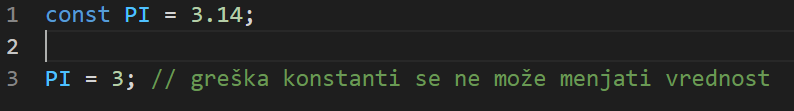


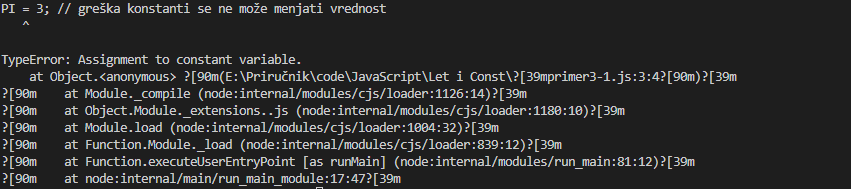
*Primer 2 - 1*

Promenljive definisane pomoću ključne reči *let*:

* ne mogu biti redefinisane unutar istog bloka važenja
* moraju biti definisane pre upotrebe
* imaju blokovsku oblast važenja

Ključna reč *const* nam omogućava da definišem promenljive istih osobina kao i one definisane pomoću *let* samo što njihova vrednost ne može biti promenjena nakon inicijalizacije.

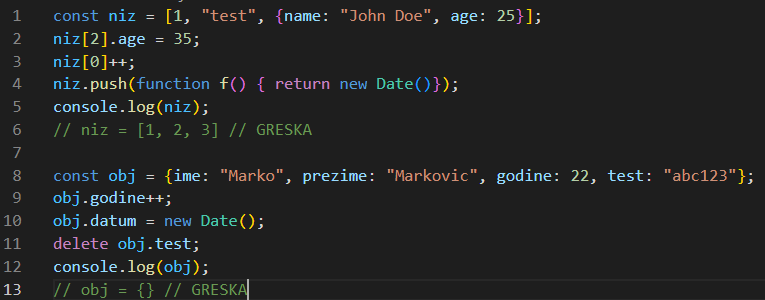


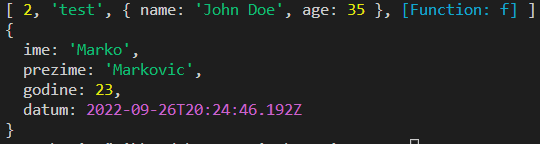


*Primer 3 – 1*

Nažalost const ne može da definiše prave konstante.

Objekti i nizovi su u JavaScriptu predstavljeni kao reference, tako da kada definisemo niz ili objekat pomoću ključne reči *const* mi kreiramo konstantu referencu, dok se u niz i dalje mogu dodavati, brisati i menjati elementi, a objektima se mogu dodavati, brisati i menjati polja.

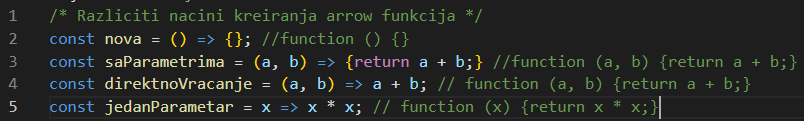




*Primer 4 - 1*

## 1.4 Arrow funkcije

Arrow funkcije su skraćeni oblik definisanja funkcija. Postoji više načina za njihovo definisanje.

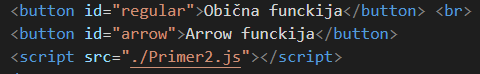


*Primer 1*

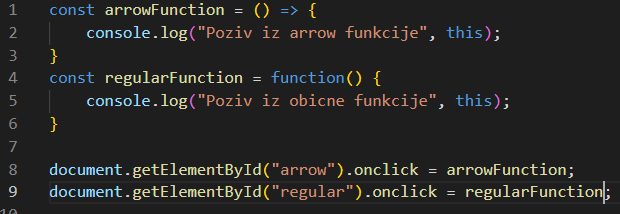
Arrow funkcije se definisu sintaksom: *(parametri) => {/\*telo\*/}*, ali postoje i skraćene varijante.

* Ako imamo samo jedan parametar možemo izostaviti zagrade: *event => {console.log(event);}*.
* Ako funkcija odmah vraća vrednost možemo izostaviti vitičaste zagrade i naredbu return: *(value) => parseInt(value)*

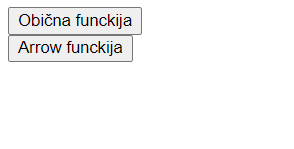
Kod arrow funkcija je veoma bitno da se kljucna reč this ponaša drugačije u odnosu na običnu funkciju:



*Primer 2 - HTML*



*Primer 2 - JavaScript*



*Primer 2 - Izgled u pregledaču*



*Primer 2 - Ispis u konzoli nakon klika na oba dugmeta*

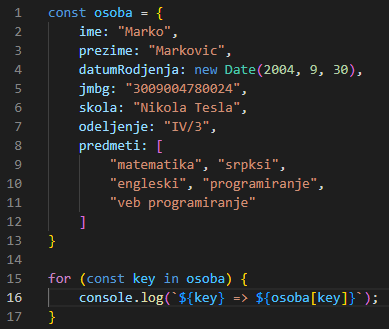
Kao što možete da primetite *this* u arrow funkciji i običnoj funkciji nije isto. U običnoj funkciji *this* se odnosi na objekat koji je pozvao funkciju (u ovom slučaju button), dok se u arrow funkcijama *this* odnosi na objekat u kom je funkcija definisana (u ovom slučaju *window* jer je *arrowFunction* globalna promenljiva).

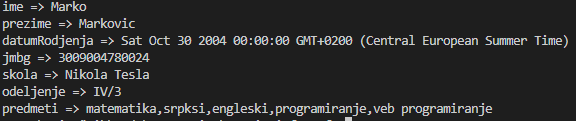
Ovo ponašanje može biti problematično ako se radi metodama objekata. Kada pozovemo metodu nad objektom očekujemo da ona sadrži referencu na sam objekat, a samim tim i na sva njegova svojstva i metode.

## 1.5 For in i for of

*For in* i *for of* se koriste za lakse prolaženje kroz objekte i nizove.

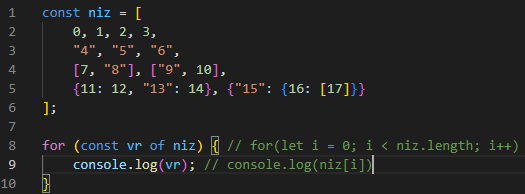
For in se koristi za prolazak kroz sve ključeve (svojstva obejka).

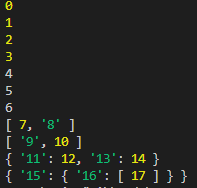




*Primer 1*

For of se koristi za prolazak kroz sve vrednosti niza:





*Primer 2*

## 1.6 Falsy vrednosti sa operatorima

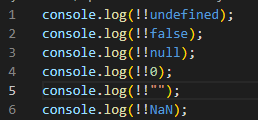
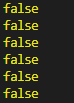
U JavaScript-u postoji nekoliko osnovnih tipova podataka: string, number, object, array, undefinded, null, function, boolean. Njihov tip se može proveriti operatorom *typeof* (uz oprez da neki od ovih tipova vraća *„object“*).

Da bi neku vrednost konvertovali u boolean tip možemo koristiti konstruktor *Boolean(value)*, ali postoji i lakši način. Kao što verovatno već znate unarni operator *!* negira logičke vrednosti, ali osim što true pretvara u false i obrnuto, on i konvertuje vrednsti u true ili false pre negiranja. Upotrebom dvosturuke negacije (*!!*) možemo vrednost konvertovati u logičku vrednost.

Pri konverziji bilo koje vrednosti u boolean tip dobija se vrednost *true* osim narednih 7 vrednosti i to:

* false
* null
* undefined
* 0
* ""
* NaN
* document.all

Ove vrednosti se nazivaju *falsy* vrednosti.

*Primer1 – konverzija falsy vrednosti u boolean*

Pri evaluaciji logičkih izraza JavaScript „pametno“ (ne)izvršava naredbe.

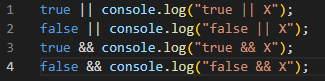
Ukoliko se radi sa logičkim ili operatorom (*||*) i prvi parametar je tačan, JS engine neće ni izvšiti kod s druge strane operatora jer bez obzira na vrednost drugog operanta ceo izraz će imati vrednost logičke istine. A ako je prvi parametar je netačan izvšiće se kod drugog operatora da bi se utvrdila vrednost celog izraza:

*true || X = true  
false || X = X*

Logički opertor i (&&) radi na istom principu:

*true && X = X  
false && X = false*

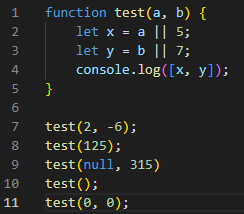
Ovakvo ponašanje omogućava da izvršavamo odrećene naredbe uslovno, bez upotrebe naredbe *if* ili upotrebe teranrnog operatora. Ovaj proces se naziva prespajanje logičkih operatora.





*Primer2 – Prespajanje logičkih operatora*

Operator ili može biti veoma koristan za zadavanje podrazumevanih vrednosti ukoliko vrednost koju pokušavamo da dodelimo *falsy*:



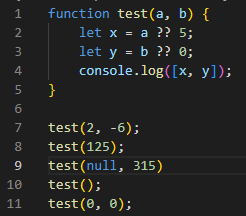


*Primer3-1 – Korišćenje ili operatora za podešavanje podrazumevanih vrednosti*

Ovde se javlja jedan problem. Pošto || operator radi sa falsy vrednostima može se desiti da nam ne prepozna vrednost koju očekujemo.

Razmotrimo prethodni primer. Ovde želimo da postavimo vrednost promenljive x na 5 ukoliko nije prosledjena vrednost za parametar *a*. Ali šta ako je parametar *a* prosledjen sa vrednošću 0? Ne želimo da pregazimo tu vrednost, ali 0 je jedna od sedam falsy vrednosti. Šta bi bilo da umesto 5 želimo da podesimo vrednost na 0?

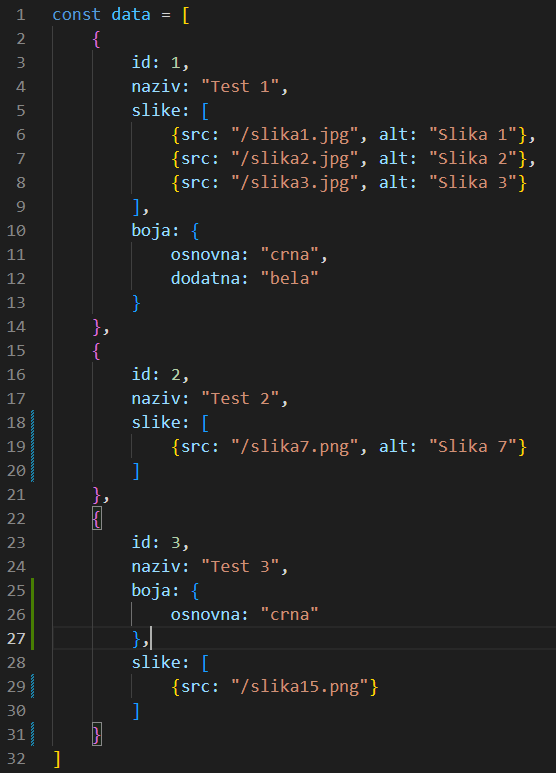
Zbog ovog problema u JavaScriptu postoju poseban operator koji radi isto kao i ili operator osim što umesto falsy vrednosti proverava samo za *null* i *undefinded.* Ovaj operator se obeležava sa dva znaka pitanja (??) i naziva se *nullish coalescing operator.*



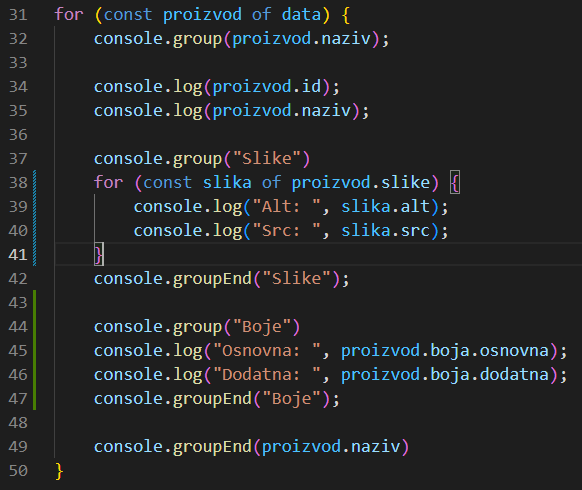


*Primer 3-2 - nullish coalescing operator*

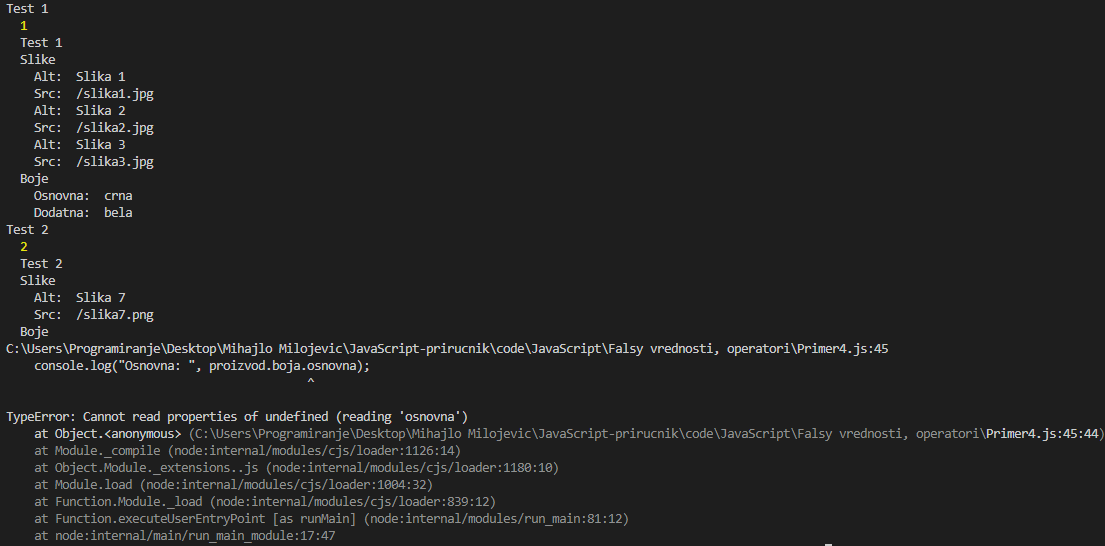
Pogledajte sledeći objekat:



Ovo bi mogli da budu podaci o proizvodima koje uzimamo iz nekog izvora. Pokušajmo da ispišemo podatke o ovim proizvodima:



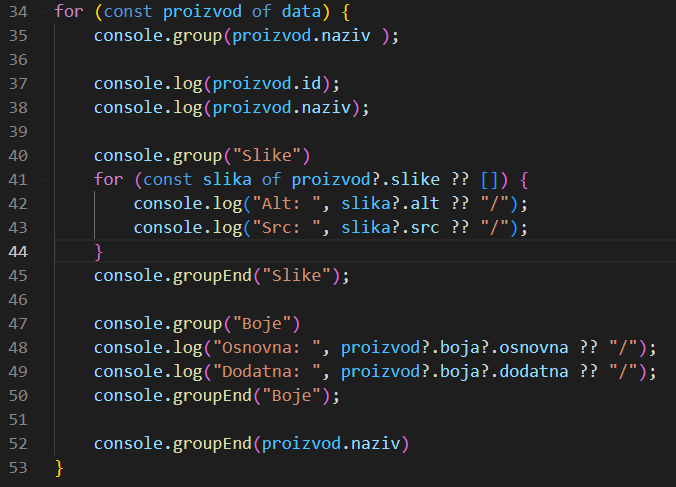
Ali ovde se javlja greška:



Ovde pokusavamo da ispisemo polje osnovna iz objekta boje, ali proizvid ne sadrži polje boje tako da mi u suštini pokušavamo da isčitamo osnovna iz undefined (*undefined.osnovna*), što naravno ne postoji.

U JavaScriptu postoji nešto što se naziva opcional chaining. Sintaksa je prosta, samo dodamo znak pitanja pre tačke (boje?.osnovna). Ovom sintaksom lako možemo proveriti da li neko polje postoji (pa čak i funkciju pozvati ukoliko jeste funkcija pomoću sintakse *„?.()“*), a u kombinaciji sa ?? operatorom možemo se lako rešiti gotovo svih problema opcionih polja.

Prepravljeni kod izgleda ovako:



A ispis je onakav kakav ga i očekujemo:

