UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Miomir N. Stojanović

ELEKTRONSKE LEKCIJE O ECMASCRIPT SPECIFIKACIJI

master rad

Beograd, 2023.

Mentor:

dr Miroslav Marić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Jelena Graovac, docent
Univerzitet u Beogradu, Matematički fakultet

dr Sana Stojanović Đurđević, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Sadržaj

Glava 1

Uvod

Skriptni jezici su programski jezici čiji se kôd prevodi u mašinski pri pokretanju. Skriptni jezici se prvenstveno koriste za pravljenje veb stranica. Kada se pišu skripte, ne pravi se novi program od nule, već se umesto toga povezuju postojeći delovi unutar programa. ECMAScript je specifikacija skriptnog jezika kreirana zbog uspostavljanja kompatibilnosti veb stranica u različitim veb pretraživačima.

ECMAScript specifikacija predstavlja nacrt za kreiranje skriptnih jezika, pruža pravila i smernice koje skriptni jezici moraju da poštuju kako bi bili usaglašeni sa standardom.

Neke od najpoznatijih implementacija ovog standarda su JavaScript i Jscipt. JavaScript je najkorišćeniji programski jezik [4], a njegova popularnost iz dana u dan raste i proširuje se njegova upotreba. Na početku se koristio isključivo na internetu, dok se danas koristi za razvoj mobilnih i računarskih aplikacija. Upravo je zbog toga došlo do potrebe za novim verzijama standarda, zarad lakšeg pisanja koda i napredovanja skriptnih jezika. Do sada je izašlo dvanaest verzija EcmaScript standarda, a nove verzije izlaze svake godine.

Cilj rada je upoznavanje sa verzijama ECMAScript standarda, kao i da se kroz lekcije koje će biti kreirane, JavaScript učini prijemčivijim i lakšim za učenje. Lekcije će biti pogodne onima koji prvi put dolaze u kontakt sa ovim radnim okvirom. Za njihovo razumevanje neophodno je poznavanje osnovnih koncepata programiranja u jeziku JavaScript.

U uvodnom delu rada biće ukratko predstavljena struktura platforme. Glavni deo rada biće podeljen na celine u kojima su opisane: sintaksa, promenljive, petlje, itd. Uvodne lekcije odnosiće se na savladavanje ECMAScript sintakse, opsega promenljivih i kontrole toka. Naredne lekcije govoriće o različitim načinima definisanja

objekata i funkcija u ECMAScript-u, fokusirajući se najviše na promene koje su donele verzije ES5 i ES6. Nakon toga biće predstavljene novije verzije ECMAScript specifikacija, kao i razvoj funkcija JavaScript programskog jezika, koje potiču od tih posebnih specifikacija.

Glava 2

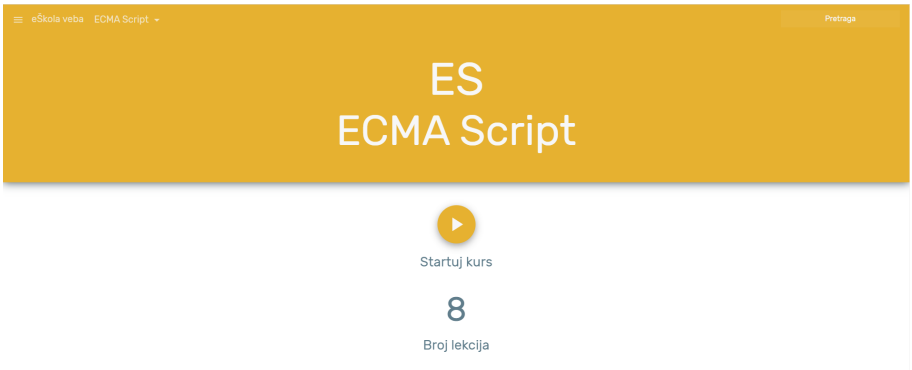
Elektronske lekcije

Elektronske lekcije o ECMAScript specifikaciji i verzijama koje su objavljene prethodnih godina dostupne su na platformi eSkola veba. Lekcije su dostupne na linku http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/ecma [6] i namenjene su svima koji žele da prošire svoja znanja o JavaScript-u. Sve lekcije su besplatne i dostupne na srpskom jeziku. Na slici 2.1 prikazan je izgled platforme.

Slika 2.1: Platforma eSkola veba.

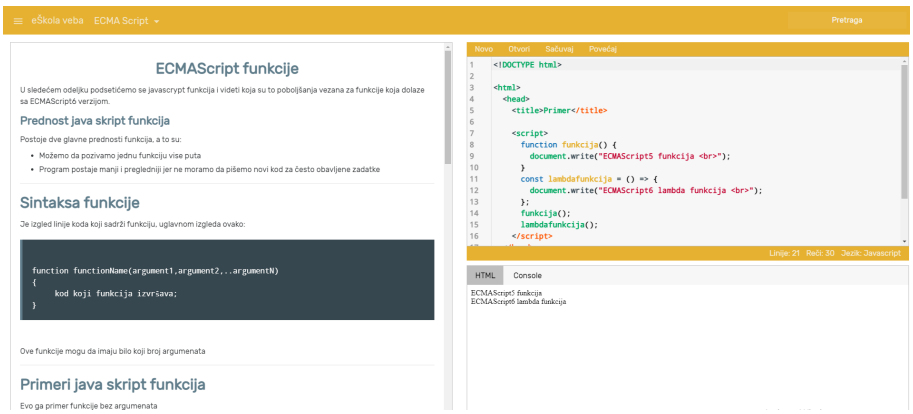
Na platformi se nalaze kursevi o različitim veb tehnologijama. Svaki kurs se

sastoji iz lekcija u kojima je sadržaj praćen primerima. Jedan od kurseva je i kurs ECMAScript koji je kreiran za potrebe ovog rada. Kada se izabere ovaj kurs, otvara se početna strana kao na slici 2.2. Svaka internet stranica u okviru elektronskih



Slika 2.2: Izgled početne strane kursa

lekcija sastavljena je iz tri dela, kao što je prikazano na slici 2.3. Na levoj strani nalaze se tekst elektronske lekcije, dok se na desnoj strani nalaze dva polja. Gornje polje je deo sa primerom koji se može direktno menjati, a donje polje daje prikaz tog primera u veb pregledaču.



Slika 2.3: Prikaz jedne lekcije

Glava 3

EcmaScript

Pre nego što se govori o EcmaScript-u, mora se spomenuti *Ecma Internationa*l, organizacija koja kreira standarde za tehnologije. Oni rade od 1961. godine i za to vreme su stvorili širok spektar globalnih tehnoloških standarda. ECMA-262 je standard koji je objavio Ecma International. On određuje skriptni jezik opšte namene.

3.1 *JavaScript* i *EcmaScript*

JavaScript je dinamičan i interpretiran programski jezik višeg nivoa. ECMAScript je specifikacija na kojoj se JavaScript zasniva. ECMAScript specifikacija (ECMA-262) [2] specifikuje samo sintaksu jezika i implementaciju pojedinih delova kao što su nizovi, funkcije i globalThis, dok važeće implementacije JavaScript-a dodaju sopstvenu funkcionalnost kao što su ulaz-izlaz i rukovanje sistemom datoteka.

ECMAScript specifikacija je važan referentni dokument za programere koji žele da razumeju ponašanje jezika zasnovanih na Ecmascript-u. S druge strane, JavaScript dokumentacija pruža praktične smernice o tome kako koristiti jezik i biblioteke u kontekstu razvoja veb aplikacija.

Od svog nastanka do danas EcmaScript je prošao kroz razne izmene i poboljšanja(ES5, ES6, ES7, itd.). Svaka izmena napravljena tako da prethodna verzija bude kompatibilna sa novom verzijom, tj. kôd koji je pisan u ES5 radiće u ES6. Da bi se postiglo obratno mora se koristiti kompajler Babel[3].

3.2 *EcmaScript* verzije

ECMAScript specifikaciju je za Netscape [5] razvio Brendan Eich [1], američki tehnolog i tvorac programskog jezika JavaScript. ECMAScript je imao nekoliko izdanja, počev od prvog koji je usvojen 1997. godine od strane Ecma General Assembly-a. Uvedene su osnovne karakteristike kao što su promenljive, petlje i funkcije i postavljen je temelj za buduće verzije. Naredne godine je usvojena nova verzija koja je donela nekoliko novih funkcija, uključujući naredbe za rukovanje izuzecima i petlju *do-while*. Nakon toga, 1999. godine objavljena poslednja verzija (ECMAScript 3) pre nego što se ušlo u stanje hibernacije u narednih 10 godina. Tokom ovih 10 godina nisu unapredili svoj proizvod i zato je rođen Firefox i još pregledača kao što su Chrome i Opera.

Tabela 3.1: Početne EcmaScript verzije

Izdanje	Datum objavljivanja	Promene u odnosu na prethodna izdanja	Urednik
1.	Jun 1997	Prvo izdanje	Guy L. Steele Jr.
2.	Jun 1998	Uređivačke promene za održavanje specifikacije u potpunosti usklađene sa međunarodnim standardom ISO/IEC 16262.	Mike Cowlishaw
3.	Decembar 1999	Dodatak običnim izrazima, novim izjavama kontrole, boljem rukovanju niskama, strožim definicijama grešaka, numeričkim izlaznim oblikovanjem, rukovanjem izuzetkom pokušaja/hvatanja i drugih poboljšanja.	Mike Cowlishaw
4.	Napušteno	Četvrto izdanje je napušteno zbog političkih razlika u vezi sa jezičkom složenošću. Nekoliko funkcija predloženih za ovo izdanje je potpuno odbačeno.	
5.	Decembar 2009	Dodavanje strogog režima, koji ima za cilj izbegavanje konstrukcija koje su sklone greškama. Dodavanje nekih novih funkcija, kao što su getteri i setteri, podrška biblioteke za JSON. Ova verzija pojašnjava nekoliko nejasnoća u specifikacijama trećeg izdanja.	Pratap Lakshman, Allen Wirfs-Brock

ECMAScript je objavio svoje peto izdanje 2009.godine (četvrto izdanje je napušteno) sa funkcijama kao što je *strict mode*. Šesti ECMAScript standard [7] je objavljen 2015. godine i uveo je nove funkcije i poboljšanja jezika, uključujući funkcije strelica, klase, module i mnoge nove metode nizova i objekata. Nakon izdavanja ES6, ECMAScript je počeo da prati godišnji raspored izdavanja i svake godine je objavljivao manje verzije. ES7 je objavljen 2016. godine i uveo je dve nove funkcije, dok je ES8 izašao 2017. godine i dodao nekoliko funkcija za rad sa objektima, i rukovanje asinhronim operacijama. ES9 je objavljen 2018. godine i uveo je funkcije kao što su proširena sintaksa za objekte i poboljšanja regularnih izraza. ES10, koji je izašao 2019. godine, dodao je nove metode za rad sa nizovima i uveo opciono *catch* vezivanje. ES11 je objavljen 2020. godine i uključivao je funkcije kao što su *BigInt*,

nove metode stringova i poboljšanja obećanja. ES12, objavljen 2021. godine, dodao je nove metode za rad sa stringovima i uveo novu sintaksu za logičke zadatke.

3.3 *EcmaScript 5* i *EcmaScript 6*

Kada se govori o osnovama JavaScript-a, to obično uključuje osnovne koncepte i funkcije uvedene u početnoj verziji ECMAScript-a, kao i nekoliko novih funkcija uvedenih u narednim verzijama. Bilo je potrebno vreme da nove funkcionalnosti ES5 i ES6 budu široko prihvaćene od strane veb pretraživača i da se programeri upoznaju sa njima. Do tada su veliki broj sajtova i aplikacija već bili napravljeni koristeći starije ECMAScript verzije. Elektronske lekcije predviđaju da je čitaoc već upoznat sa osnovama Javasciript-a i zato se sledeći odeljci: sintaksa, promenljive, petlje, funkcije, dogadjaji, objekti, klase i moduli fokusiraju na promene koje su donele verzije ES5 i ES6. Izmene koje su donele verzije nakon ES6 su znatno manjeg obima i zato se njihovo analiziranje ostavlja za kraj.

Glava 4

Sintaksa *EcmaScript*-a

Sintaksa se odnosi na strukturu i pravila koja regulišu pisanje koda. Sintaksa definiše gramatiku, interpunkciju i simbole koji se koriste za kreiranje izjava i izraza u programskom jeziku. Svaka specifikacija jezika ima svoju sintaksu. Bez sintakse, značenje ili semantiku jezika je nemoguće razumeti. Semantika daje značenje sintaksno ispravnim elementima, odnosno za dati program opisuje akcije definisane tim programom. Program u okviru JavaScript-a sastoji se od: literala, promenljivih, ključnih reči, operatora, komentara i identifikatora.

Literal se može definisati kao oznaka za predstavljanje fiksne vrednosti u izvornom kodu. Generalno, literali se koriste za inicijalizaciju promenljivih. U sledećem primeru može se videti broj 26 i string „zdravo“ koji su oba literali, pošto predstavljaju fiksne vrednosti koje se mogu koristiti direktno u kodu.

Primer 4.1: Inicijalizacija promenljivh

```
1 var x = 26;
2 var str = "zdravo";
```

Promenljiva (engl. „Variable“) je simboličko ime za lokaciju u memoriji u kojoj se čuvaju vrednosti kao što su brojevi ili stringovi. U JavaScript-u, deklarisanje promenljive je moguće i sa i bez ključne reči.

U računarskom programiranju ključna reč (engl. „Keyword“) je reč koja u posebnom kontekstu ima posebno značenje. Ne može se koristiti kao identifikator poput imena promenljive, funkcije i imena funkcije.

Operatori su simboli koji definišu obradu operanda. Neki od uobičajenih primera operatora uključuju aritmetički operatori (+,-, ...), logičke operatore (poput AND,

...) itd.

4.1 Komentari

Komentari pomažu programeru da zabeleži ili naglasi informacije o kodu. JavaScript podržava dva tipa komentara:

- višelinijski (engl. „Multi-line comments“) - Ovi komentari se koriste za komentarisanje u više linija;
- jednolinjski (engl. „ Single-line comments“) - Ovi komentari se koriste za komentarisanje u jednoj liniji.

Jednolinijski komentari počinju sa // i nastavljaju se do kraja reda. Često se koriste za pružanje kratkih opisa ili objašnjenja koda. Višelinijski komentari počinju sa /* i završavaju se sa */. Oni mogu da obuhvataju više redova i često se koriste za duža objašnjenja ili za privremeno onemogućavanje bloka koda.

Primer 4.2: Jednolinijski i višelinijski komentari

```
1 /* Primer
2 viserednog komentara */
3 // Jednolinijski komentar.
```

4.2 Identifikatori

Identifikatori su imena koja se daju elementima unutar programa, kao što su funkcije, promenljive, itd. Postoje pravila koja programeri moraju da poštuju prilikom korišćenja identifikatora.

- Identifikatori ne uključuju specijalne simbole osim znaka ($) ili donja crta(_).
- Ime identifikatora ne sme biti ključna reč.
- Identifikatori uključuju znakove i cifre, ali ime identifikatora ne možete započeti cifrom.
- Ime identifikatora ne sme sadržati blank simbol.

U sledećim primerima mogu se videti ispravni i neispravni identifikatori.

Ispravan primer: userName, user_name, name14, $name.

Neispravan primer: Name@, user name, user-name, 14name.

ECMAScript ignoriše tabulatore (engl. „Tabs“), razmake i oznaku za novi red koji se pojavljuju u programima, njih je moguće lako koristiti, a program slobodno formatirati i uvlačiti na pouzdan način, što povećava čitljivost koda i olakšava razumevanje.

U JavaSciptu, identifikatori razlikuju velika i mala slova. Na primer: username i userName su dve različite promenljive u JavaScript-u.

```
let username, userName;
username = "Anil";
userName = "Sunil";
```

Upotreba ; (tačka zaraza) nije obavezna u JavaScript-u. Ali ako u jednom redu ima više iskaza (niz uputstava), te izjave moraju biti odvojene tačkom i zarezom.

```
<textarea name="code" class="java" rows="3" cols="100">
console.log("Zdravo.")
console.log("Dobrodosli."); console.log("Ucimo EcmaScript.");
```

4.3 Strogi režim

Strogi režim (engl. „Strict mode“) je uveden u ECMAScript-u 5. Ovaj strogi režim pomaže da se napiše pouzdaniji kôd. Kada se omogući strogi režim, dolazi do promena kao što su sledeće.

- Pre nego što se upotrebi promenljiva, neophodno je deklarisati tu promenljivu pomoću neke od ključnih reči. U suprotnom će biti izbačena referentna greška.
- Nemoguće je obrisati promentljive ili funkcije koje su deklarisane pomoću ključne reči.
- Nemoguće je u funkciji definisati više parametara sa istim imenom.

Deklarisanje strogog režima: može se deklarisati dodavanjem „use strict”; na početku funkcije ili skripte.

Deklaracija na početku skripte: kada se deklariše na početku skripte, to će biti globalni domet, tj. sav kôd unutar skripte će se izvršiti u strogom režimu.

Primer 4.3: Korišćenje „use strict”

```
1 "use strict";
2 example();
3 function example() {
4   x= 89;    //Prouzrokovace gresku.
5 }
```

Kada se izvrši ovaj primer, dobiće se greška jer promenljiva x nije deklarisana.

Deklaracija unutra funkcije: Kada se deklariše unutar funkcije, to će biti lokalni domet, tj. kôd unutar funkcije će biti u strogom režimu.

Primer 4.4: Korišćenje „use strict” u funkciji

```
1 y = 89;         // Nece prouzrokovati gresku.
2 example();
3 function example() {
4   "use strict";
5   x = 89;    // Prouzrokovace gresku.
6 }
```

Glava 5

Promenljive

Promenljiva je simboličko ime za lokaciju u memoriji u kojoj se čuvaju vrednosti kao što su brojevi ili stringovi. Promenljive takođe mogu da sadrže složenije tipove podataka, kao što su nizovi ili objekti. Promenljiva se mora deklarisati pre nego što se upotrebi. Pre ES6 koristila se isključivo ključna reč *var* da bi se definisala promenljiva. Sintaksa za definisanje promenljive je sledeća:

```
1 var  promenljiva_ime
```

ES6 uvodi nove ključne reči za deklarisanje promenljivih:

- *let*,
- *const*.

Da bi se razumelo koja je razlika, prvo je neophodno upoznati se sa pojmom opsega promenljivih.

5.1 Opseg promenljivih

Opseg promenljive je deo programa u kome je ona vidljiva. Tradicionalno, JavaScript definiše samo dva opsega - globalni i lokalni.

- Globalni opseg - promenljivoj sa globalnim opsega može se pristupiti iz bilo kojeg dela JavaScript koda.
- Lokalni opseg - promenljivoj sa lokalnim opsegom može se pristupiti iz funkcije u kojoj je deklarisana.

Sledeći primer deklariše dve promenljive po imenu *num* - jedna izvan funkcije, globalni opseg, a druga unutar funkcije (lokalni opseg).

Primer 5.1: Lokalne i globalne promenljive

```
var num = 10
test funkcije()
{
var num = 100
console.log ("vrednost num u testu ()" + broj)
}
console.log ("vrednost num izvan testa ()" + broj)
test()
```

5.2 Ključna reč *Let*

Promenljivoj koja je deklarisana pomoću *let* ili *const* dodeljuje se opseg bloka(block scope). Opseg bloka je oblast unutar *if*, *switch* naredba ili *for* i *while* petlja. Uopšteno govoreći, svaki put kada se jave vitičaste zagrade, to je blok. Dakle promenljiva deklarisana unutar bloka neće biti vidljiva van njega.

Primer 5.2: Opseg bloka

```
<!DOCTYPE html>
<html>
<body>
<script>
function run() {

{
    const ime = "Felix";
     let god = 30;

  document.write(ime +" ima "+god+" godina <br>");
}

document.write(ime +" ima "+god+" godina <br>"); // greska
}
run();
</script>
```

```
18 </body>
19 </html>
```

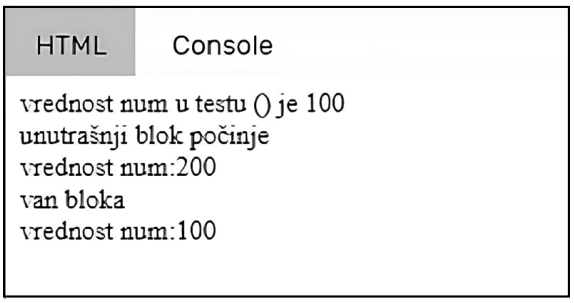
Let omogućava skripti da ograniči pristup promenljivoj do najbližeg ograđenog bloka. Ako je u skripti deklarisana promenljiva *num* u okviru lokalnog opsega funkcije i ponovo je deklarisana unutar bloka koristeći ključnu reč *let*. Vrednost lokalno vidljive promenljive se štampa kada se promenljivoj pristupi izvan unutrašnjeg bloka, dok se u unutrašnjem bloku štampa vrednost promenljive definisane unutar bloka.

Promenljive deklarisane pomoću ključne reči *var* imaju opseg funkcije.

Primer 5.3: Opseg funkcije

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <script>
5  "use strict"
6  function test() {
7  var num = 100;
8  document.write("vrednost num u testu () je " + num + "<br>");
9    {
10     document.write("unutrasnji blok pocinje<br>");
11     let num = 200;
12     document.write("vrednost num:" + num + "<br>");
13   }
14 document.write("van bloka <br>");
15 document.write("vrednost num:" + num + "<br>");
16 }
17 test();
18 </script>
19
20 </body>
21 </html>
```

U primeru 5.3, promenljiva *num* je deklarisana unutar lokalnog opsega funkcije i ima vrednost 100. Ponovo se deklariše unutar bloka koristeći ključnu reč *let* i dobija vrednost 200. Vrednost promenljive lokalnog opsega se štampa kada se promenljivoj pristupa izvan unutrašnjeg bloka, dok se promenljiva s opsegom bloka pristupa unutar unutrašnjeg bloka.



Slika 5.1: Html prikaz nakon izvršenog koda iz primera 5.4

5.3 Konstanta

Rezervisana reč *const* se koristi za deklarisanje promenljive koja se ne može ponovo dodeliti nakon što je postavljena njena početna vrednost. To znači da kada se jednom deklariše promenljiva sa *const*, ne može se promeniti njena vrednost kasnije u kodu. Konstante imaju opseg bloka, slično kao promenljive koje su definisane pomoću izraza *let*.

```
1 const k = 10;
2 k = 12; // rezultirace greskom !!
```

Iako se promenljivoj *const* ne može ponovo dodeliti vrednost, sama vrednost na koju pokazuje može se menjati. Na primer, ako se deklariše promenljiva *const* koja pokazuje na niz ili objekat i dalje je moguće izmeniti sadržaj tog niza ili objekta.

Glava 6

Petlje

Ponekad, određene instrukcije zahtevaju izvršavanje veći broj puta. Petlje su idealan način da se to uradi. Petlja predstavlja skup instrukcija koje se moraju ponoviti. Konceptualno, postoje dve vrste petlji, određene i neodređene, koje se razlikuju po načinu na koji se određuje broj iteracija.
Određena petlja ima definitivan/fiksni broj ponavljanja. U ES6 postoje tri vrste određenih petlji.

- *for(; ;)* petlja - Izvršava blok naredbi određeni broj puta.
- *for..in* petlja - Iterira kroz svojstva objekta.
- *for..of* petlja - Iterira kroz elemente niza.

6.1 *For* petlja

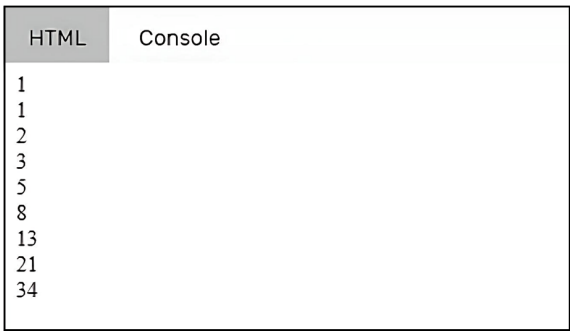
Petlja *for(; ;)* koristi se za ponavljanje dela programa više puta. Ako ima fiksni broj ponavljanja, preporučuje se upotreba petlje *for*.

U *for* petlji se može kombinovati više uslova koristeći operator , (zarez). U sledećem primeru štampaju se Fibonačijevi brojevi koristeći samo jednu for petlju.

Primer 6.1: Fibonačijev niz

```
1  <!DOCTYPE html>
2  <html>
3      <body>
4          <script>
5              "use strict"
6              for(let temp, a = 0, b = 1; b<40; temp = a, a = b, b = a
       + temp)
7              document.write(b+"<br>");
8          </script>
9
10     </body>
11 </html>
```

Nakon izvršavanja prethodnog kôda dobija se sledeći prikaz:



Slika 6.1: Html prikaz nakon izvršenog koda iz primera 6.1

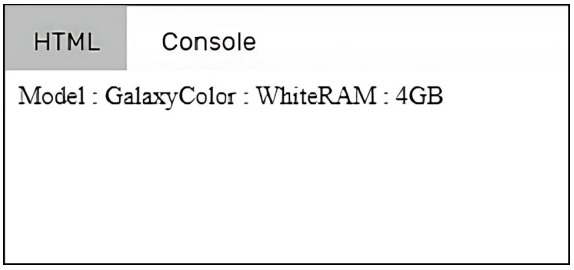
6.2 *For..in* petlja

Petlja *for.. in* se koristi za iteraciju kroz svojstva objekta. U svakoj iteraciji jedno svojstvo objekta dodeljuje se promenljivoj i petlja se nastavlja sve dok se sva svojstva objekta ne pokriju.

Primer 6.2: For..in petlja

```
<!DOCTYPE html>
<html>
<body>
  <script>
    function Mobile(model_no){
      this.Model = model_no;
      this.Color = 'White';
      this.RAM = '4GB';
    }
  var Samsung = new Mobile("Galaxy");
  for(var props in Samsung)
  {
  document.write(props+ " : " +Samsung[props]);
  }
  </script>
</body>
</html>
```

Funkcije u Javascript-u jesu objekti. Baš kao objekti, funkcije imaju svojstva i metode, mogu se skladištiti u promenljivoj ili nizu i proslediti kao argumenti drugim funkcijama.Može se saznati nešto više o tome u narednim lekcijama. Nakon izvršavanja prethodnog kôda dobija se sledeći prikaz:



Slika 6.2: Html prikaz nakon izvršenog koda iz primera 6.2.

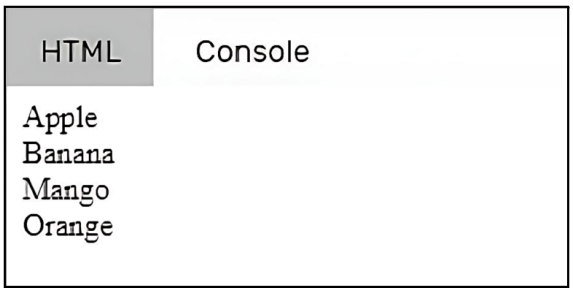
6.3 *For..of* petlja

Za razliku od objektnih literala, ova petlja se koristi za iteracuju kroz nizove. U svakoj iteraciji, jedan član niza dodeljuje se promenljivoj, a petlja se završava kada dođemo do poslednjeg elementa niza.

Primer 6.3: *For..in* petlja

```
<!DOCTYPE html>
<html>
<body>
<script>
let fruits = ['Apple', 'Banana', 'Mango', 'Orange'];
for(let value of fruits)
{
document.write(value+"<br>");
}
</script>
</body>
</html>
```

Nakon izvršavanja prethodnog koda dobija se sledeći prikaz:



Slika 6.3: Html prikaz nakon izvršenog koda iz primera 6.3.

Za iteraciju kroz niz u prethodnom primeru koristila se nova promenljiva *value*, ali moguće je i koristiti i promenljivu koja je već deklarisana.

6.4 *While* i *do-while* petlja

Neodređena petlja ima beskonačno iteracija. Koristi se kada je broj ponavljanja naredbi nepoznat. *While* petlja je kontrolni tok koji omogućava ponavljanje izvršenog koda na osnovu datog uslova. Sastoji se od blok koda i izraza. *While* petlja proverava izraz pre izvršavanja bloka.

Primer 6.4: While petlja

```
let count = 6, fact = 1;
while (count > 0)
{
```

```
4 fact = fact * count--;
5 }
6 console.log(fact);
```

do-while petlja je naredba kontrolnog toka koja barem jednom izvršava blok koda, a onda zavisno od uslova izvršava blok više puta. *do-while* petlja proverava uslov posle izvršenja bloka, zato je ova kontrolna struktura takođe poznata kao post-test petlja.

Primer 6.5: Do..while petlja

```
1 let count = 6, fact = 1;
2 do {
3 fact = fact * count--;
4 } while (count > 0);
```

Ključna razlika između gore dva navedena primera je u tome što *while* petlja izvršava naredbu samo ako je uslov tačan. *do-while* petlja izvršava naredbu barem jedanput, što se dešava jer se uslov proverava na kraju petlje.

Glava 7

Funkcije

U opštim programerskim terminima, funkcija je blok koda koji obavlja određeni zadatak ili skup zadataka. One su važan deo većine programskih jezika, uključujući JavaScript. Postoje mnoge prednosti korišćenja funkcija.

- Funkcije omogućavaju da se napiše kôd koji se može koristiti na više mesta u okviru programa.
- Funkcije olakšavaju podelu složenih programa na manje delove kojima se lakše upravlja.
- Funkcije obezbeđuju način za enkapsulaciju podataka i ponašanja, sprečavajući da im drugi delovi programa pristupaju ili menjaju.

Postoje dve glavne prednosti korišćenja funkcija, a to su:

- može se pozvati jedna funkcija više puta,
- program postaje manji i pregledniji jer se ne mora pisati novi kôd za često ponovljene zadatke.

Sintaksa funkcije je izgled linije koda koji sadrži funkciju i uglavnom izgleda kao što je prikazano na slici 7.1.

Primer 7.1: Sintaksa funkcije

```
1 function functionName(argument1,argument2,..argumentN)
2 {
3       //kod koji funkcija izvrsava;
4 }
```

Ove funkcije mogu da imaju bilo koji broj argumenata. U primeru 7.2. prikazana je funkcija bez argumenata.

Primer 7.2: Funkcija bez argumenta

```
1 function msg()
2 {
3     alert("Ovo je poruka");
4 }
```

Funkcije sa povratnom vrednošću - na slici 7.3. funkcija koja je pozvana će vratiti vrednost na osnovu dobijenih argumenata.

Primer 7.3: Funkcija koja ima povratnu vrednost.

```
1 function add( a, b )
2 {
3     return a+b;
4 }
5 var sum = add(10,20);
6 console.log(sum);
```

Default parametri funkcija - U ES6 funkcije nam omogućavaju inicijalizaciju parametara sa zadatim vrednostima ako parametar nije definisan ili mu se ne prenese vrednost.

Primer 7.4: *Default* parametar

```
1 function show (num1, num2=200)
2 {
3     console.log("num1 = " +num1);
4     console.log("num2 = " +num2);
5 }
6 show(100);
```

7.1 *Rest* parametar

Rest paremetar je uveden u ES6, što je poboljšalo mogućnosti rukovanja parametrima. Parametar *rest* omogućava da se neograničen broj argumenata predstavi kao niz. Korišćenjem parametra rest funkcija se može pozvati sa bilo kojim brojem argumenata. Za deklarisanje parametra rest, ime parametra treba imati prefiks operatora rasprostiranja koji je predstavljen kao tri tačke.

Primer 7.5: Rest paremetar

```
function show(...args) {
  let sum = 0;
  for (let i of args) {
      sum += i;
  }
  console.log("Sum = "+sum);
}
show(10, 20, 30);
```

Nakon izvršavanja ovog kôda u konzoli će se pojaviti Sum = 60.

7.2 Anonimne funkcije

Anonimna funkcija se može definisati kao funkcija bez imena. Anonimna funkcija se ne povezuje sa identifikatorom. Može se dinamički deklarisati tokom izvođenja. Anonimna funkcija nije dostupna nakon njenog prvog kreiranja. Promenljivoj se može dodeliti anonimna funkcija. Sintaksa anonimne funkcije je sledeća:

Primer 7.6: Anonimna funkcija

```
var hello = function() {
   console.log('Zdravo');
   console.log('Ja sam anonimna funkcija');
 }
 hello();
```

Jedna uobičajena upotreba anonimne funkcije je da se koristiti kao argument drugih funkcija.

Primer 7.7: Anonimna funkcija kao argument

```
setTimeout(function()
{
   console.log('Zdravo');
 }, 2000);
```

Nakon izvršavanja ovog koda u konzoli će se ispisati reč „Zdravo”.

7.3 Lambda funkcije

Arrow functions ili lambda funkcije uvedene su u ES6, što je omogućilo tačniji način pisanja funkcija u JavaScript-u. Lambda funkcije čine kôd čitljivijim i strukturiranijim. Lambda funkcije su anonimne funkcije, bez imena i nisu vezane za identifikator. Ne vraćaju nikakvu vrednost i mogu se deklarisati bez ključne reči *function*. Lambda funkcije se ne mogu koristiti kao konstruktori.
Postoje tri dela lambda funkcije:

- parametri - svaka funkcija može, ali ne mora da ima parametre;
- oznaka za strelicu (=>);
- iskaz - predstavlja skup instrukcija funkcije.

Primer 7.8: Funkcija koja ima povratnu vrednost.

```
1  const add = (n1,n2) => n1+n2
2    console.log(add(10,20))
3
4    const isEven = (n1) => {
5       if(n1%2 == 0)
6          return true;
7       else
8          return false;
9    }
10   console.log(isEven(10))
```

Kada se koriste lambda funkcije kôd je kraći i izgleda preglednije. U verzijama pre ES6 neophodno je bilo da se definiše *return* naredba unutar funkcija, ali u ES6 to nije potrebno za jednolinijske funkcije. Zagrade tela funkcije su takođe opcione za jednolinijske funkcije.

Primer 7.9: Funkcija u ES5.

```
1 function show(value){
2 return value/2;
3 }
4 console.log(show(100));
```

Primer 7.10: Funkcija u ES6.

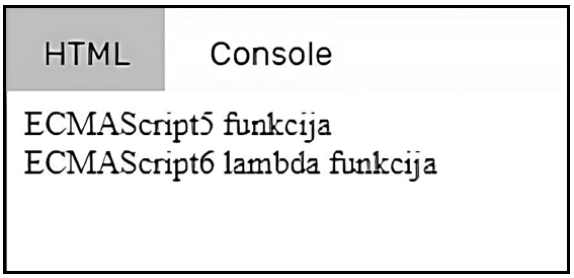
```
1 var show = value => value/2;
2 console.log(show(100));
```

Ako se ne upotrebljavaju vitičaste zagrade na jednom izrazu, onda ne mora da se koristi *return*, ali ako se koriste vitičaste zagrade čak i na jednom izrazu, onda mora da se koristi ključna reč *return*.

Primer 7.11: Razlika ES5 i ES6.

```
<html>
  <head>
    <title>Primer</title>

    <script>
      function funkcija() {
        document.write("ECMAScript5 funkcija <br>");
      }
      const lambdafunkcija = () => {
        document.write("ECMAScript6 lambda funkcija <br>");
      };
      funkcija();
      lambdafunkcija();
    </script>
  </head>

  <body></body>
</html>
```

Nakon izvršavanja prethodnog koda dobija se sledeći prikaz:



Slika 7.1: Html prikaz nakon izvršenog koda iz primera 7.11.

Glava 8

Događaji

JavaScript interaguje sa HTML-om pomoću događaja koji se dešavaju prilikom korišćenja stranice. Događaji su deo Document Object Model (DOM) i svaki HTML element sadrži skup događaja koji mogu pokrenuti Javascript kôd.

Događaj je akcija ili pojava prepoznata od strane softvera. On može biti pokrenut od strane korisnika ili sistema. Neki česti primeri događaja su kada korisnik pritisne dugme, učitavanje veb strane, klikom na hiper link i tako dalje. Svaki HTML element ima svoj skup događaja povezanih sa njim. Može se definisati kako će događaj biti izvršen u Javascript-u koristeći manipulatore događaja.

Tabela 8.1: Najkorišćeniji događaji

Atribut	Vrednost	Opis
offline	script	Pokreće se kada dokument nije dostupan
onabort	script	Pokreće se na prekid događaja
onafterprint	script	Pokreće se nakon štampanja dokumenta
onbeforeonload	script	Pokreće se pre učitavanja dokumenta
onbeforeprint	script	Pokreće se pre štampanja dokumenta
onblur	script	Pokreće se kada prozor izgubi fokus
oncanplay	script	Pokreće se kada mediji mogu da započnu s reprodukcijom, ali možda će morati da prestanu zbog bafera
oncanplaythrough	script	Pokreće se kada se mediji mogu reprodukovati do kraja, bez zaustavljanja zbog baferovanja
onchange	script	Pokreće se kada se element promeni
onclick	script	Pokreće se klikom miša
oncontextmenu	script	Pokreće se kada se aktivira kontekstni meni

Da bi se reagovalo na događaje, može se dodeliti Handler, funkciju koja se pokreće u slučaju da dođe do događaja. Handler događajima može se definisati kao blok koda koji se izvršava prilikom pozivanja događaja.

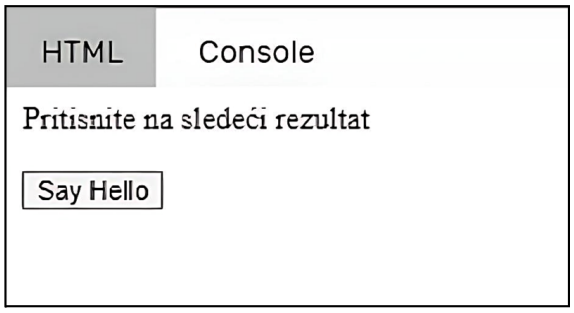
8.1 Klik događaj

Klik događaj (onclick) je jedan od najkorišćenijih događaja koji se izvršava kada se pritisne levi klik na mišu. Ishod može biti potvrda, upozorenje ili slično.

Primer 8.1 Onclick događaj

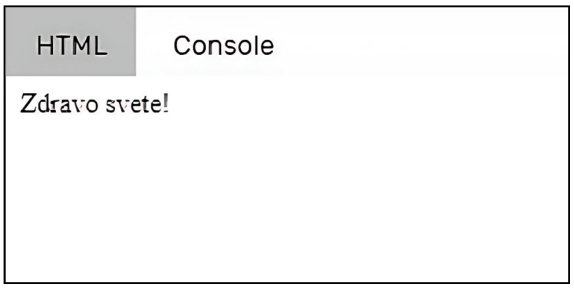
```
<html>
   <head>
      <script type = "text/javascript">
        const sayHello = () => {

    document.write("Zdravo svete!");
         }
      </script>
   </head>

   <body>
      <p> Pritisnite na sledeci rezultat </p>
      <input type = "button" onclick = "sayHello()" value = "Say
   Hello" />
   </body>
</html>
```

Nakon izvršavanja prethodnog kôda dobija se prikaz kao na slici 8.1.



Slika 8.1: Html prikaz nakon izvršenog koda iz primera 8.1.

Nakon što se pritisne dugme dobija se prikaz kao na slici 8.2.



Slika 8.2: Html prikaz nakon što je pritisnuto dugme.

8.2 Događaji pomeraja kursora

Ovi događaji će pomoći da se naprave lepi efekti sa slikama ili tekstom. OnMouseOver događaj započinje kada se stavi miš preko bilo kog elementa koji ima tu komandu, a OnMouseOut započinje kada se pomeri miš od elementa na kome je stavljen kursor.

Primer 8.2 Kursor preko i kursor van događaj

```
1  <html>
2     <head>
3        <script type = "text/javascript">
4           const over = () =>{
5              document.getElementById("div1").style.color
6                                 = "blue";
7           }
8           const out = () =>{
9              document.getElementById("div1").style.color
10                                = "red";
11          }
12       </script>
13    </head>
14
15    <body>
16       <p>Stavite vas mis unutar polja da vidite rezultat:</p>
17       <div id="div1" onmouseover = "over()" onmouseout = "out()">
18          <h2> ovo je unutar polja </h2>
19       </div>
20    </body>
21 </html>
```

Glava 9

Objekti i klase

Objekti su kolekcija ključ-vrednost (engl. „key-value“) parova koji se mogu menjati tokom čitavog životnog ciklusa objekta. Objekti u Javascript-u omogućavaju da se definišu novi tipovi podataka. Za razliku od promenljivih koje mogu da imaju samo jednu vrednost, objekti mogu imati više njih. Vrednosti mogu biti niz drugih objekata ili primitivnih tipova podataka, koje zovemo još i polja.

Objekti igarju dve uloge u Javascript-u:

- izveštaji (records) - objekti kao izveštaji imaju fiksiran broj svojstava, čiji ključevi se znaju još u vreme pisanja programa. Njihova polja mogu biti različitih tipova podataka.
- rečnici (dictionaries) - objekti kao rečnici imaju promenljiv broj svojstava čiji ključevi se ne znaju u vreme pisanja programa. Sva njihova polja su istog tipa.

Literali objekata su jedan od načina stvaranja objekata. Oni su istaknuta karakteristika JavaScript-a: mogu se direktno kreirati objekti, nema potrebe za klasama.

U sledećem primeru je kreiran objekat pomoću literala objekta, koji počinje i završava se sa vitičastim zagradama. Unutar njega su definisana dva svojstva ključ-vrednost:

- prvo polje ima ključ *first* i vrednost 'Jane',
- drugo polje ima ključ *last* i vrednost 'Doe'.

Primer 9.1 Kreiranje objekta

```
const jane = {
  first: 'Jane',
  last: 'Doe', // zadnji zarez nije neophodan
};
```

Kasnije će biti prikazani i drugi načini specificiranja ključeva svojstava, ali sa ovim načinom specifikacije oni moraju slediti pravila naziva JavaScript promenljivih. Na primer, moguće je koristiti ime_ime kao ključ svojstva, ali ne i ime). Međutim, rezervisane reči su dozvoljene.

Primer 9.2 Dozvoljeno je koristiti rezervisane reči

```
const obj = {
if: true,
const: true,
};
```

9.1 Klase

Klase su neizostavni deo objektno orijentisanog programiranja (OOP). Klase se koriste za definisanje nacrta za modeliranje objekata u stvarnom svetu i organizovanje koda u delove koji se mogu ponovo koristiti. Pre ES6, bilo je teško napraviti klasu u JavaScriptu. U ES6 moguće je da se kreiraju klase pomoću ključne reči class.

Primer 9.3 Kreiranje klase

```
class Student{
    constructor(name, age){
    this.name = name;
    this.age = age;
    }

}
```

Definicija klase može da sadrži samo konstruktore i funkcije. Ove komponente se zajedno nazivaju članovi klase, koji se definišu unutar tela klase (deo koda odvojen vitičastim zagradama). Klase sadrže funkcije koje su odgovorne za izvršavanje radnji sa objektima. Kao i u drugim objektno orijentisanim jezicima, pomoću ključne reči *new* može da se instancira objekat klase.

9.2 Nasleđivanje

Nasleđivanje je mogućnost kreiranja novih entiteta od postojećih. Klasa koja je proširena za kreiranje nove klase naziva se nadređena klasa, dok se novostvorena klasa naziva potklasa. Pre ES6, implementacija nasleđivanja zahtevala je nekoliko koraka. Ali ES6 je pojednostavio implementaciju nasleđivanja koristeći *extends* i *super*.

Primer 9.4 Nasleđivanje u ES6

```
class Parent {
   show() {
      document.write("show() metod klase Perent <br>");
   }
}
class Child extends Parent {
   show() {
      super.show();
      document.write("i show() metod klase Child <br>");
   }
}
var obj = new Child();
obj.show();
```

Zadatak: Definisati klasu 'oblik' koja ima atribute x i y (kordinate) i metod koji će omogućiti njegovo pomeranje, a zatim definisati klasu 'krug' koja nasleđuje 'oblik'.

Primer 9.5. Rešenje zadatka

```
<!DOCTYPE html>
<html>
<head>
  <title>Primer</title>
<script>
class Oblik {
    constructor (id, x, y) {
        this.id = id
        this.pomeri(x, y)
    }
    pomeri (x, y) {
        this.x = x
        this.y = y
    }
```

```
15    toString(){
16      document.write(this.x+" "+this.y);
17    }
18  }
19  class Krug extends Oblik {
20      constructor (id, x, y, radius) {
21          super(id, x, y)
22          this.radius = radius
23      }
24  }
25  var k = new Krug(1,1,1);
26  k.pomeri(2,2);
27  k.toString();
28  </script>
29  </head>
30  <body>
31  </body>
32  </html>
```

Glava 10

Moduli

Kako program postaje sve veći, može da sadrži mnogo linija koda. Umesto da se sve stavi u jednu datoteku, moguće je koristiti module za razdvajanje kodova u odvojenim datotekama prema njihovoj funkcionalnosti. Ovo čini kod organizovanim i lakšim za održavanje. Moduli olakšavaju otklanjanje grešaka u kodu i ponovno korišćenje dela koda. Svaki modul je deo koda koji se izvršava čim se učita.

10.1 Izvoženje modula

EcmaScript moduli omogućavaju izvoženje funkcija, objekata, klasa i primitivnih vrednosti pomoću ključne reči *export*.

Postoje dva načina izvoženja modula:

- imenovano izvoženje (engl. „named export“) - izvozi se razlikuju pomoću imena kojim su pozvani. Moguće je izvoziti više promenljivih i funkcija koristeći imenovani izvoz.
- podrazumevano izvoženje (engl. „default export“) - možemo izvoziti samo jednu promenljivu ili funkciju koristeći podrazumevani izvoz.

U sledećem primeru možemo videti kako se definišu izvozi.

Primer 10.1 Definisanje izvoza

```
// utils.js
export function funcA() {
  return " Imenovano izvozenje!";
}

export default function funcB() {
  return "Podrazumevano izvozenje!";
}
```

10.2 Uvoženje modula

Vrednosti koje se izvoze iz modula mogu se uvesti korišćenjem ključne reči *import*. Moguće je da se uvezu izvezene promenljive, funkcije i klase u drugi modul. Da bi se uvezao modul, jednostavno moramo navesti njihovu putanju.

Imenovani izvezeni moduli uvoze se na sledeći način:

```
// consumer.js
import { funcA } from "./util.js";
```

Dok se podrazumevano izvezeni moduli uvoze na sledeći način:

```
// consumer.js
import funcB from "./util.js";
```

Kada se uvozi imenovani izvoz, mora se koristiti isto ime kao i odgovarajući objekat. Kada se uveze podrazumevani izvoz, može se koristiti bilo koje ime odgovarajućeg objekta. Može da se uveze čitav modul koristeći * (zvezdicu).

Primer 10.2 Uvoženje modula

```
import * as myModule from "./util.js";

myModule.funcA();
myModule.default();
```

Glava 11

Ostale verzije

EcmaScript 2015(ES6) je najveće i najvažnije poboljšanje EcmaScript standarda. Svake godine, počev od 2015. izlaze nove verzije, kojima se dodaju nove karakteristike. Ovo poglavlje će biti posvećeno tim karakteristikama.

11.1 ECMAScript 2016 (ES7)

ES7 uvodi novi matematički operator pod nazivom eksponencijalni operator. Ovaj operator je sličan metodu Math.pow(). Eksponencijalni operator je predstavljen dvostrukom zvezdicom **. Operator se može koristiti samo sa numeričkim vrednostima. Sintaksa za korišćenje eksponencijalnog operatora može se videti u sledećem primeru.

Primer 11.1 Eksponencijalni operator

```
1 let base = 2;
2    let exponent = 3;
3    document.write("Eksponencijalni operator: 2**3= " +base**
     exponent);
```

Metod *Array.includes()* prihvata parametar i proverava da li vrednost prosleđena kao parametar postoji u nizu. Ovaj metod daje „true" ako je vrednost pronađena odnosno daje „false" ako vrednost ne postoji.

Primer 11.2 Kako funkcioniše Array.includes()

```
1  let niz = [1,2,3,5,7,11,13,17]
2    //proverimo da li vrednost 7 postoji u nizu.
3    if(niz.includes(50)){
4       document.write("Pronadjen!");
```

```
5     }else{
6         document.write("Nije pronadjen!");
7 }
```

11.2 ECMAScript 2017 (ES8)

ES8 ugrađuje sledeće metode u objekt:

- *Object.entries()*,
- *Object.values()*,
- *Object.getOwnPropertyDescriptors()*.

Object.entries(), metod koji vraća niz koji sadrži sve vrednosti polja objekta:

```
1  const person = { name: 'Fred', age: 87 }
2 Object.values(person) // ['Fred', 87]
```

Object.values(), ovaj metod vraća niz koji sadrži sve vrednosti polja objekta kao par [ključ,vrednost].

```
1 const person = { name: 'Fred', age: 87 }
2 Object.entries(person) // [['name', 'Fred'], ['age', 87]]
```

Prethodna dve funkcije mogu se primeniti i na nizove.

```
1 const people = ['Fred', 'Tony']
2 Object.values(people) // ['Fred', 'Tony']
3 Object.entries(people) // [['0', 'Fred'], ['1', 'Tony']]
```

getOwnPropertyDescriptors(), ovaj metod vraća sve vlastite (nenasleđene) deskriptore svojstava objekta. Deskriptor je skup atributa polja i sastoji se od:

- value: Vrednost polja;
- writable: Ako je netačno vrednost polja se ne može menjati;
- get: Funkcija koja se poziva kada želimo da pročitamo vrednost polja;
- set: Funkcija koja se poziva kada postavljamo neku vrednost u polje;

- configurable: Ako je netačno polje ne može da se izbriše, niti možemo menjati njegove atribute;
- enumerable: Tačno je ako je polje nabrojivo.

11.3 ECMAScript 2018 (ES9)

Kada se destruktuje objekat, *Object Rest Properties* omogućava da se prikupe preostala svojstva objekta i smeste u nov objekat.

Primer 11.3 Prikupljanje preostalih svojstava objekta

```
const input = {x: 1, y: 2, a:3, b:4 };
const { x, y , ...z} =input ;
console.log(x); // 1
console.log(y); // 2
console.log(z); // { a: 3 , b: 4 }
```

Nova konstrukcija *for-await-of* omogućava da koristitimo asinhroni objekat koji se može iterirati pomoću petlje.

```
for await (const line of readLines(filePath)) {
  console.log(line)
}
```

Kako ova konstrukcija koristi *await* [1], moguće ga je koristiti samo unutar funkcija asinhronizacije.

ES2018 je uveo i niz poboljšanja u vezi sa regularnim izrazima, ali se ovaj kurs neće mnogo baviti time.

11.4 ECMAScript 2019 (ES10)

Ova verzija uvodi nov metod instance niza *flat()* koji može kreirati jednodimenzionalni niz od višedimenzionalnog niza.

```
['Pas', ['Ovca', 'Vuk']].flat()
//[ 'Pas', 'Ovca', 'Vuk' ]
```

[1] *await* je ključna reč u JavaScript-u koja se koristi sa asinhronim funkcijama za pauziranje izvršavanja funkcije dok se obećanje ne reši ili odbije.

Podrazumevano *flat()* deluje samo do jednog nivoa, ali može se dodati parametar da bi se postavio broj nivoa na koje će se poravnati niz; podesiti na *infinity* da bi se imalo neograničeno nivoa.

Primer 11.4. Izravnanje niza na neograničeno nivoa

```
['Pas', ['Ovca', 'Vuk']].flat()
//[ 'Pas', 'Ovca', [ 'Vuk' ] ]

['Pas', ['Ovca', 'Vuk']].flat(2)
//[ 'Pas', 'Ovca', 'Vuk' ]

['Pas', ['Ovca', ['Vuk']]].flat(Infinity)
//[ 'Pas', 'Ovca', 'Vuk' ]
```

ES2019 uvodi i opciono vezivanje parametara za *catch* blok, dok se prethodno morao koristiti parametar *catch* čak i kada to nije potrebno.

Primer 11.5. Opcioni parametar

```
try{
  // pre je moralo:
} catch(e) {
 //...
}

try{
// sada moze
} catch {
//...
}
```

Ova verzija uvodi i dve nove metode, *trimEndt()* i *trimStart()*, koje se pozivaju nad string-ovima kako bi se uklonili blank karakteri. trimStart() vraća novi niz karaktera sa uklonjenim blank karakterima sa početka originalnog niza.

```
'Testiranje'.trimStart() //'Testiranje'
' Testiranje'.trimStart() //'Testiranje'
' Testiranje '.trimStart() //'Testiranje '
'Testiranje '.trimStart() //'Testiranje '
```

Analogno trimEnd() vraća novi niz karaktera sa uklonjenim blank karakterima sa kraja originalnog niza.

11.5 ECMAScript 2020 (ES11)

Pre ES11 nije postojala fukcija za pretraživanje svih pojavljivanja string-a u drugom string-u. Zbog toga je u ovoj verziiji dodata funkcija *matchAll()*. U sledećem primeru može se videti upotreba funkcije *matchAll()*.

```
1 let text = "I love cats. Cats are very easy to love. Cats are very
    popular."
2 const iterator = text.matchAll("Cats");
```

Funkcija *matchAll()* će vratiti iterator svih stringova koji odgovaraju prosleđenom stringu. Zato ima više smisla proslediti regularni izraz, kao što se može videti u sledećem primeru.

```
1 let text = "I love cats. Cats are very easy to love. Cats are very
    popular."
2 const iterator = text.matchAll(/Cats/gi);
```

Operator(??) vraća prvi argument ako on nije null ili undefined, u suprotnom vraća drugi argument.

```
1 let name = null;
2 let text = "missing";
3 let result = name ?? text;
```

Operator(?.), slično kao prethodni operator bavi se vrednostima null i undefined, ali sada u objektima. Pre ovog operatora pristup nedefinisanim poljima objekta proizveo bi grešku. Pomoću ovog operatora dobićemo undefined kao povratnu vrednost, što se može videti u sledećem primeru:

```
1 const car = {tip:"Fiat", model:"500", color:"white"};
2 let name = car?.name;
```

11.6 ECMAScript 2021 (ES12)

Numerički separator omogućava da se pomoću donje crte brojevi učine čitljivijiim. Donje crte će se automatski ukloniti prilikom raščlanivanja datoteke. U sledećem primeru može se videti upotreba numeričkog separatora.

Primer 11.6. Primena numeričkog operatora

```
let n1 = 1_000_000_000;
console.log(n1); // Ispisace: 1000000000

let n2 = 1_000_000_000.150_200
console.log(n2); // Ispisece: 1000000000.1502
```

Funkcija *replaceAll()* dozvoljava zamenu svih instanci podniza, bez korišćenja regularnog izraza. Ako se *thereplace()* koristi u stringu, zamenjuje samo prvu instancu te vrednosti. S druge strane, *replaceAll()* pruža funkcionalnost za zamenu svih instanci te vrednosti. U sledećem primeru može se videti upotreba *replaceAll()*.

Primer 11.7. Funkcija *replaceAll()*

```
// Declare a variable and store some value.
const orgStr = 'JavaScript, often abbreviated as JS, is a
    programming language that conforms to the ECMAScript
    specification. JavaScript is high-level, often just-in-time
    compiled and multi-paradigm.';

// To replace single instance, use replace().
let newStr = orgStr.replace('JavaScript', 'TypeScript');
console.log(newStr);
```

Metode i polja klase su podrazumevano javne, ali privatne metode i polja se mogu kreirati korišcenjem prefiksa heš #. Enkapsulacija privatnosti je primenjena od ažuriranja ECMAScript 2021. Ovim privatnim metodama i poljima se može pristupiti samo iz klase. U sledećem primeru može se videti njihova upotreba.

Primer 11.8. Privatne metode i svojstva

```
class Korisnik {
    constructor() {}

    #generisiSifru() {
        return "cf946093107898cb6";
    }

    getPassword() {
        return this.#generisiSifru();
    }
```

```
11 }
12
13 const pera = new Korisnik();
14 const sifra = pera.getPassword();
15 console.log(sifra); // Ispisace: cf946093107898cb6
```

11.7 ECMAScript 2022 (ES13)

Metod *.at()* je podržan indeksiranim vrednostima kao što su nizovi, string ili tipovani nizovi. Uzima pozitivnu ili negativnu celobrojnu vrednost i vraća element sa datim indeksom.

U primeru 11.9. postoji niz sa tri elementa, što znači da je njegova dužina 3. Prvi element je na indeksu 0, a poslednji na indeksu vrednosti dužine niza -1 (u ovom primeru na indeksu 2).

Da bi se pristupilo elementu na određenom indeksu, koristi se operator zagrade (*zivotinje[0]*). Za indekse van opsega, program vraća *„Undefined“*.

Primer 11.9. Pristupanje elementu niza

```
1 let zivotinje = ["Macka", "Papagaj", "Rakun"];
2
3 console.log(zivotinje[0]);
4 console.log(zivotinje[zivotinje.length - 1])
5 console.log(zivotinje[zivotinje.length])
6
7 //Macka
8 //Rakun
9 //undefined
```

Nova funkcija radi isto kao i operator zagrade, ali omogućava negativno indeksiranje elemenata. Poslednjem elementu se lako pristupa zamenom *zivotinje[zivotinje.length–1]* sa *zivotinje.at(-1)*

Primer 11.10. Korišćenje metoda *.at()*

```
1 let zivotinje = ["Macka", "Papagaj", "Rakun"];
2
3 console.log(zivotinje.at(0));
4 console.log(zivotinje.at(-1));
5
6 //Macka
```

```
7 //Rakun
```

Object.hasOwn() vraća tačno, ako navedeni objekat ima naznačeno svojstvo kao sopstveno svojstvo. Ako je svojstvo nasleđeno ili ne postoji, metod vraća netačno. *Object.hasOwn()* je zamišljen kao zamena za *Object.hasOwnProperty()*.

Primer 11.11. Korišćenje metoda .hasOwnProperti()

```
1  let korisnik = {
2  ime: 'Petar Petrovic',
3  uloga: 'Administrator'
4  };
5
6  console.log(korisnik.hasOwnProperty('uloga'));
7  console.log(korisnik.hasOwnProperty('godine'));
8
9  //true
10 //false
```

Nakon ES13 može se koristiti i kôd iz sledećeg primera.

Primer 11.12. Korišćenje metoda *.hasOwn()*

```
1  let korisnik = {
2  ime: 'Petar Petrovic',
3  uloga: 'Administrator'
4  };
5
6  console.log(Object.hasOwn(korisnik, 'uloga'));
7  console.log(Object.hasOwn(korisnik, 'godine'));
8
9  //true
10 //false
```

ES2022 uvodi novi metod *cause* („uzrok“), pomoću *Error.cause* greškama je moguće dodati više bitnih informacija. Trebalo bi da se navedu opcije greške kao drugi parametar, a pomoću ključa „uzrok“ moguće je proslediti grešku koja je povezana.

EcmaScript 2022 je dodao mogućnost da se u klasi definišu statička polja i metode. Statička polja i metode klase se ne koriste na instancama klase. Umesto toga mogu se pozvati nad samom klasom i deklarisati pomoću ključne reči static. Statičke

metode su često pomoćne funkcije, dok su statička svojstva korisna za keš memorije i fiksnu konfiguraciju.

Primer 11.12. Statička polja

```
1 class Korisnik {
2     static prezime = 'Petrovic';
3 };
4
5 console.log(Korisnik.prezime);
6 //Petrovic
```

Glava 12

Zaključak

Nove EcmaScript verzije su definitivno donele mnoga poboljšanja, od jednostavnog sintaksičkog dodatka poput klasa do novih i složenih funkcija kao što su obećanja i moduli. Neke od novih funkcija će učiniti kôd čistijim i čitljivijim (npr. lambda funkcije), druge su korisne da bi olakšale rešavanje logičkih problema (npr. nove metode za nizove).

Ova nova izdanja ECMAScript-a neće samo promeniti način na koji se piše JavaScript, već će promeniti i način na koji razmišlja i organizuje JavaScript kôd. Javascript se pojavljuje svuda: dizajniranje, razvoj korisničkog interfejsa, kodiranje na strani servera (eng. „back-end"), mobilne aplikacije, razvoj igara itd. Veb programer mora da proširuje svoje znanje o novim verzijama ECMAskript-a.

U ovom radu prolazi se kroz različite verzije ECMAScript-a i može se videti kako se funkcije JavaScript-a, koje potiču od tih posebnih specifikacija koriste. Sve funkcije koje su spomenute su podržane u skoro svim veb pregledačima. Iako neke od ovih funkcija možda nisu esencijalne za razvoj veb aplikacije, one pružaju mogućnosti koje su se ranije mogle postići trikovima ili uz mnogo komplikacija.

Elektronske lekcije o ECMAScript specifikaciji napravljene su za potrebe ovog master rada i imaju za cilj da na efikasan način upoznaju korisnika sa ECMAScript-om. Elektronske lekcije namenjene su za sve one koji žele da unaprede svoje znanje JavaScript-a. Svaka elektronska lekcija sadrži odgovarajuće primere koji bi trebalo da čitaocu lekcija pomognu pri učenju. Sve elektronske lekcije su dostupne na linku http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/ecma.

Bibliografija

[1] Brendan Eich: Javascript, Firefox, Mozilla, and Brave | Lex Fridman Podcast #160. https://www.youtube.com/watch?v=krB0enBeSiE , pristupano februara 2023.

[2] Ecma international ECMAScript 2023 Language Specification. https://tc39.es/ecma262/, pristupano februara 2023.

[3] Javascript compajler Babel. https://babeljs.io, pristupano februara 2023.

[4] Najkorišćeniji programski jezici. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages , pristupano februara 2023.

[5] Netscape Communications Corporation, 1994. https://web.archive.org/web/19970220012313/http://home.netscape.com, pristupano februara 2023.

[6] Jurić Nemanja i Marić Miroslav. eSkola veba VII Simpozijum Matematika i primene, 2016. http://edusoft.matf.bg.ac.rs/eskola_veba/.

[7] Dr. Axel Rauschmayer. Javascript for impatient programmers, samoizdavaštvo, 2015.