Uvod u programiranje

Skipta za kurs
Uvod u programiranje kroz JavaScript

Una Stanković
una.stankovic@code.edu.rs

10. septembar 2018.

U ovom tekstu predstavljene su teorijske osnove potrebne za savladavanje kursa " Uvod u programiranje kroz JavaScript". Najpre su navedene, ukratko, teorijske osnove računarstva i uvod kroz HTML i CSS, a kasnije se ulazi u rad sa JavaScript-om. Ova skripta je obavezan materijal pri kursu i sa prezentacijama i kodovima formira celinu. Ova skripta sama po sebi nije dovoljna, samostalan rad i istraživanje je neizostavni deo procesa učenja. U slučaju da primetite greške pri čitanju rada ili imate bilo kakve nedoumice, predloge i sugestije javite se mejlom na adresu navedenu na prvoj strani. Materijali su kreirani prvenstveno na osnovu materijala korišćenih u nastavi na Matematičkom fakultetu, iz predmeta Uvod u Veb i Internet tehnologije, Programiranje za Veb, Mrežno računarstvo i Programiranje 1.

Sadržaj

1 Uvod u računarstvo

Računarstvo i informatika predstavljaju jednu od najvažnijih oblasti današnjice koje su u konstantnom razvoju. Danas, ne možemo zamisliti život bez računara, pametnih telefona ili mnogobrojnih uređaja koji se pokreću uz pomoć računara. Razvitak računarstva i tehnologije u poslednjih 70 godina je eksponencijalan, tako da, danas, imamo razvoj prenosnih računara, tableta, pametnih telefona i uređaja, kola, kućnih aparata i ostalih koji se pokreću korišćenjem računarskih sistema. Kako definisati računarski sistem? Postoji više različitih računarskih sistema, od kojih svaki ima svoju posebnu definiciju, ali naš fokus je na digitalnim računarskim sistemima. Oni podrazumevaju mašinu koja može da se programira kako bi izvršavala različite zadatke svođenjem na elementarne operacije nad brojevima. Brojevi se u računaru zapisuju uz pomoć nula i jedinica, odnosno, binarnim zapisom, kao nizovi bitova.

Kada se razmišlja o tome šta sve računarstvo obuhvata, lako se uviđa da računarstvo nije samo računar, već da ono predstavlja mnogo širu oblast koja se bavi izučavanjem teorije i prakse procesa računanja i primene računara u različitim naučnim oblastima, tehnici i svakodnevnom životu. Računar sam po sebi nije cilj, već sredstvo za postizanje različitih ciljeva u zavisnosti od njihove primene. Za današnje računare često ćemo čuti da su "programabilni", ali šta to zapravo govori? Programabilnost računara se ogleda u činjenici da je moguće računaru dati neki skup instrukcija koje će on izvršavati sa ciljem ispunjavanja određenih zadataka, koje mu čovek, odnosno, programer zadaje. Računari kakve danas poznajemo nastali su polovinom XX veka, ali želja za automatizacijom određenih postupaka seže daleko dalje u prošlost. Naime, posmatrajući istorijski, ljudi su vekovima stvarali razne naprave koje su mogle da rešavaju neke numeričke zadatke.

1.1 Istorijat računarstva

Da bismo u potpunosti razumeli računarstvo moramo imati uvid u njegove početke i razvoj. Istorijski gledano, koreni ljudske želje da olakšaju sebi svakodnevni život sežu davno u prošlost. Kao pravi primer takvih težnji možemo uzeti jedne od prvih računaljki abakus. U 18. veku nastale su prve mehaničke sprave koje su mogle da vrše automatsko izvođenje aritmetičkih operacija i pomažu u rešavanju matematičkih zadataka. Blez Paksal[1] je 1642. godine konstruisao mehaničke sprave koje su služile za sabiranje i oduzimanje celih brojeva, zvane Paskaline. Trideset godina nakon njega, Godfrid Lajbnic[2] konstruisao je mašinu, zasnovanu na dekadnom sistemu, koja je mogla da vrši sve četiri osnovne operacije. Lajbnic je bio prvi koji je predlagao koriscenje binarnog sistema.

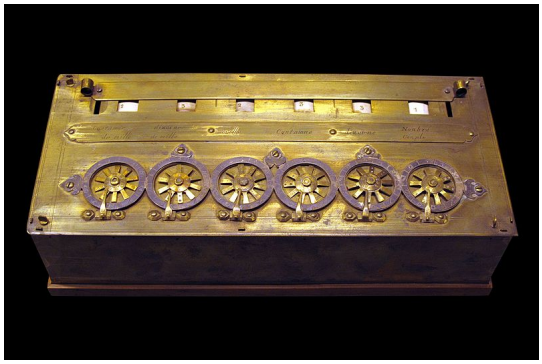
Mehaničke mašine Žozef Mari Žakard[3] je 1801. godine napravio prvu programabilnu mašinu — mehanički tkački razboj. On je pomoću bušenih kartica kreirao kompleksne šare na tkanini. Svaka rupa na kartici određivala je

[1] Blaise Pascal (1623–1662), francuski filozof, matematičar i fizičar. U njegovu čast jedan programski jezik nosi ime PASCAL.

[2] Gottfried Wilhelm Leibniz (1646–1716), nemački filozof i matematičar

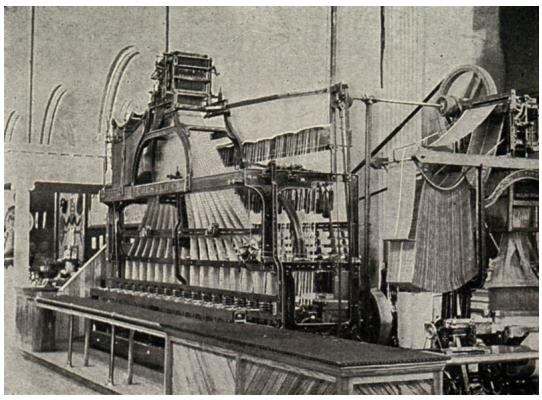
[3] Joseph Marie Jacquard (1752–1834), francuski trgovac.

Slika 1: Abakus



Slika 2: Paskalina

jedan pokret mašine, a svaki red na kartici odgovarao je jednom redu šare. U

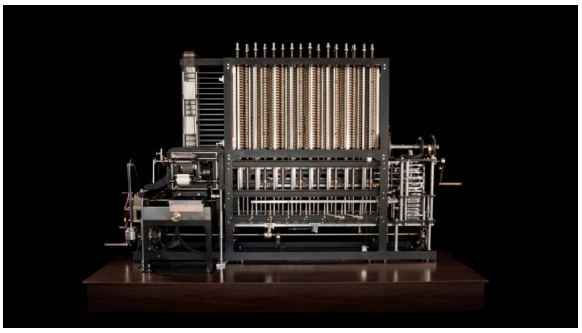


Slika 3: Žakardov razboj

prvoj polovini 19. veka, Čarls Bebidž[4] je dizajnirao prve programabilne računske mašine. Godine 1822. započeo je rad na diferencijskoj mašini, za računanje vrednosti polinomijalnih funkcija. Ime je dobila zbog toga što je koristila tzv.

[4]Charles Babbage (1791–1871), engleski matematičar, filozof i pronalazač.

metod konačnih razlika da bi bila eliminisana potreba za množenjem i deljenjem. Mašina je trebalo da ima oko 25000 delova i da se pokreće ručno, ali nije nikada završena. Ubrzo nakon toga, Bebidž je započeo rad na novoj mašini nazvanoj analitička mašina. Osnovna razlika u odnosu na sve prethodne mašine specifičnih namena, bila je u tome što je analitička mašina zamišljena kao računska mašina opšte namene. "Programiranje" na ovoj mašini vršilo bi se programima zapisanim na bušenim karticama (sličnim Žakardovim), a program bi kontrolisao mehanički računuar koji bi omogućavao sekvencijalno izvršavanje naredbi, grananje i skokove. Osnovni delovi ovog računara trebalo je da budu mlin (engl. mill) i skladište (engl. store), koji po svojoj funkcionalnosti sasvim odgovaraju procesoru i memoriji današnjih računara. Ada Bajron[5] zajedno sa Bebidžem napisala je prve programe za analitičku mašinu i, da je mašina uspešno konstruisana, njeni programi bi mogli da računaju određene složene nizove brojeva (takozvane Bernulijeve brojeve). Upravo je to razlog zašto se ona smatra prvim programerom u istoriji. Ona je bila i prva koja je uvidela da se računarske mašine mogu upotrebiti i u nematematičke namene, čime je naslutila današnjicu.



Slika 4: Bebidžova diferencijska mašina

Elektromehaničke mašine Ove mašine koristile su se od sredine 19. veka do Drugog svetskog rata. Jednu od prvih mašina za čitanje bušenih kartica konstruisana je od strane Hermana Holerita. Njena glavna svrha bila je obrada rezultata popisa stanovništva u Sjedinjenim američkim državama 1890. godine. Koristeći bušene kartice uspešno izvršen je popis za godinu dana, naspram deset godina, koliko je bilo potrebno ranije. Od Holeritove male kompanije nastao je IBM.[6]

Elektronski računari Elektronski računari koriste se od kraja 1930-ih do danas.

1.2 Fon Nojmanova arhitektura

Na osnovu istorijata može se uočiti da svi navedeni računari imaju nedostatak jedne važne karakteristike računara danas, a to je programabilnost. Mašine

[5] Augusta Ada King (rođ. Byron), Countess of Lovelace, (1815–1852), engleska matematičarka. U njenu čast nazvan je programski jezik ADA.

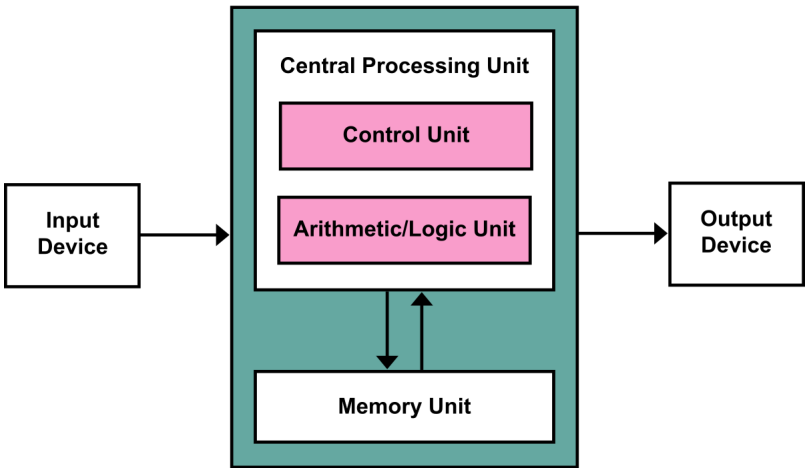
[6] Herman Hollerith (1860–1929), američki pronalazač.

korišćene nekada nisu bile programabilne već su funkcionisale po unapred definisanom programu određenom samom konstrukcijom mašine. Iako ovakav pristup nije sasvim izumro (danas se može videti na primeru digitrona), uglavnom nije poželjan. Prava promena u pristupu, koja je dovela do stvaranja programabilnih računara, nastala je ranih 1940-ih godina sa pojavom računara koji bi programe koje izvršavaju čuvali u memoriji zajedno sa podacima. Takve računare nazivamo računarima sa skladištenim podacima (engl. stored program computers). Jedna od najvažnijih karakteristika ovih računara je da kod njih postoji jasna podela na hardver i softver. Za rodonačelnika ovakve arhitekture smatra se Džon fon Nojman. On je 1945. godine opisao arhitekturu čija je glavna karakteristika da se programi mogu učitavati isto kao i podaci koji se obrađuju. Primeri prvih ovakvih računara su EDVAC, Mark 1 i EDSAC.

Osnovni elementi fon Nojmanove arhitekture su:

1. procesor - koji čine aritmetičko-logička jedinica, kontrolna jedinica i registri, i
2. glavna memorija

koji su međusobno povezani, dok se ostale komponente računara smatraju pomoćnim. Prikaz fon Nojmanove arhitekture je na slici 5. Pod pomoćne komponente ubra-



Slika 5: Šematski prikaz fon Nojmanove arhitekture

jamo ulazno-izlazne jediice, spoljašnje memorije itd., koje da bi funkcionisale moraju biti povezane na centralni deo računara (procesor i glavnu memoriju). Osnovna uloga procesora je obrada podataka, dok je osnovna uloga memorije skladištenje podataka koji se obrađuju, kao i programa. Postoji jedinstven način na koji zapisujemo i podatke i programe, a to je uz pomoć nula i jedinica, odnosno, binarnim zapisom. Tokom rada računara podaci i programi se prenose između procesora i glavne memorije.

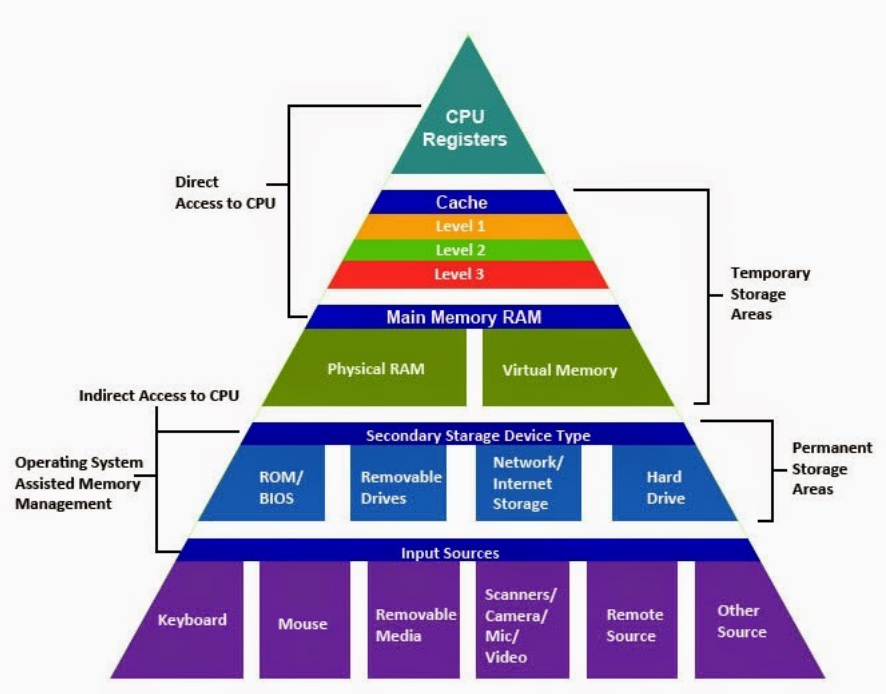
Procesor Prva centralna komponenta fon Nojmanove arhitekture. Procesor, koji je odgovoran za rad računara, sastoji se od *kontrolne jedinice* - koja

upravlja radom procesora i *aritmetičko-logičke jedinice* - koja je zadužena za izvođenje aritmetičkih operacija (sabiranje, oduzimanje, množenje, poređenje, itd.) i logičkih operacija (konjunkcija, negacija, itd.) nad brojevima. Osim dva navedena dela procesor sadrži i određeni broj registara, obično fiksirane širine (8, 16, 32 ili 64 bita), koji privremeno mogu da čuvaju podatke. Danas procesori neretko poseduju više jezgara (engl. core) koja istovremeno izvršavaju instrukcije čime se obezbeđuje paralelno izvršavanje.

Memorija Druga centralna komponenta fon Nojmanove arhitekture je glavna memorija. Memorija predstavlja linearno uređeni niz registara, pri čemu svaki ima svoju adresu. Kao posledica osobine ove memorije da se sadržaju može pristupati u slučajnom redosledu, čest naziv je i memorija sa slobodnim pristupom (engl. RAM - random access memory). Razlikujemo nekoliko parametara koji odlikuju memoriju, to su:

- kapacitet - GB,
- vreme pristupa - vreme potrebno da se memorija pripremi za čitanje ili upis i
- protok - izražava količinu podataka koji se prenose po jedinici merenja (danas obično mereno u GBps).

Na slici 6 može se videti memorijska hijerarhija.



Slika 6: Šematski prikaz memorijske hijerarhije

1.3 Hardver

Bez obzira na činjenicu da osnovu savremenih računarskih sistema i dalje čini fon Nojmanova arhitektura, za rad računara u današnjem smislu reči potreban je i čitav niz hardverskih komponenti koje nam dodatno olakšavaju. Opis komponenti koje čine jedan računar danas ne sastoji se od kućišta, monitora, tastature i miša, već nam je potreban apstraktniji, sveobuhvatniji opis. Upravo, kako bi se jasnije i preciznije opisao računar kaže se da ga čine:

- procesor tj. centralna procesorska jedinica (engl. Central Processing Unit, CPU), koja obrađuje podatke,
- glavna memorija (engl. main memory), u kojoj se istovremeno čuvaju i podaci koji se obrađuju i trenutno pokrenuti programi, i
- različiti periferijski uređaji ili ulazno-izlazne jedinice (engl. peripherals, input-output devices, IO devices), u koje se ubrajaju miševi, tastature, ekrani, štampači, diskovi, a koji služe za interakciju između korisnika i sistema i trajno skladištenje podataka i programa.

Da bi se izvršilo povezivanje svih navedenih komponenti koristimo magistralu. Za funkcionisanje modernih računara neophodni su i hardver i softver. Hardver (tehnički sistem računara) čine opipljive, fizičke komponente računara: procesor, memorija, matična ploča, itd.

1.4 Softver

Softver računara čine programi i prateći podaci koji određuju izračunavanja koja vrši računar. Prvi računari su se odlikovali jezicima specifičnim za konkretni računar - mašinski zavisnim jezicima. Već od 1950-ih, sa pojavom prvih jezika višeg nivoa, programiranje postaje dosta lakše. Danas, programi se najčešće pišu u višim programskim jezicima, a potom se prevode na mašinski jezik, onaj koji je razumljiv računaru. Programom opisujemo računaru koje operacije treba da izvrši sa ciljem ispunjavanja nekog zadatka. U nastavku biće navedeno nekoliko primera koji ilustruju izvršavanje programa napisanih na višim programskim jezicima.

Primer 1.1 *Želimo da izračunamo vrednost izraza $2*x+3$ za neko x. Podatke u računarstvu, kao i u matematici, možemo predstaviti pomoću promenljivih. Međutim, za razliku od matematike, promenljive u računarstvu mogu menjati svoju vrednost. Svakoj promenljivoj je u memoriji računara pridruženo jedno fiksirano mesto i ona može tokom izvršavanja programa da menja vrednost. Recimo da je x ulazni parametar našeg programa, a y izlazna vrednost, tada izračunavanje opisujemo sa*

```
y := 2*x + 3
```

gde $*$ označava množenje, $+$ sabiranje, a $:=$ naredbu dodele, odnosno promenljivoj sa leve strane izraza dodeljujemo vrednost izraza sa desne strane.

Primer 1.2 *Kao naredni primer uzmimo poređenje dva broja, odnosno, kao izlaz treba da dobijemo veći od dva broja. Ovakav izraz možemo zapisati kao:*

```
ako je x >= y onda
    m := x
inače
    m := y
```

Primer 1.3 *Kao još jedan primer uzmimo stepenovanje:*

```
s := 1, i := 0
dok je i < n radi sledeće:
    s := s·x,
    i := i + 1
```

Savremeni softver klasifikujemo u 2 kategorije:

- Sistemski i
- Aplikativni.

Aplikativni softver je onaj koji krajnji korisnici računara direktno koriste u svojim svakodnevnim aktivnostima. Tu spadaju, na primer, pregledači Veba, e-mail klijenti, kancelarijski softver (programi za kucanje teksta, izradu prezentacija,...), video igre, softver za prikaz slika, itd.
Sistemski softver ima ulogu da kontroliše hardver i pruža usluge aplikativnom softveru. Najznačajniji skup sistemskog softvera je operativni sistem (OS), ali u sistemski softver ubrajamo i različite uslužne programe: editore teksta, alate za programiranje (prevodioci, dibageri, profajleri, integrisana okruženja) i slično. Uloga operativnog sistema je da programeru pruži skup funkcija koje programer može da koristi kako bi ispunio određeni cilj, sakrivajući konkretne hardverske detalje - ovaj skup funkcija naziva se programski interfejs za pisanje aplikacija (engl. API - Application Programming Interface).

1.5 Oblasti savremenog računarstva

Savremeno računarstvo sastoji se iz više podoblasti između kojih nema jasnih granica. Prema klasifikaciji američke asocijacije ACM - Association for Computing Machinery, razlikujemo naredne podoblasti [**?**]:

- Algoritmika (procesi izračunavanja i njihova složenost)
- Strukture podataka (reprezentovanje i obrada podataka)
- Programski jezici (dizajn i analiza svojstava formalnih jezika za opisivanje algoritama)
- Programiranje (proces zapisivanja algoritama u nekom programskom jeziku)
- Softversko inženjerstvo (proces dizajniranja, razvoja i testiranja programa)
- Prevođenje programskih jezika (efikasno prevođenje viših programskih jezika, obično na mašinski jezik)
- Operativni sistemi (sistemi za upravljanje računarom i programima)
- Mrežno računarstvo (algoritmi i protokoli za komunikaciju između računara)

- Primene (dizajn i razvoj softvera za svakodnevnu upotrebu)
- Istraživanje podataka (pronalaženje relevantnih informacija u velikim skupovima podataka)
- Veštačka inteligencija (rešavanje problema u kojima se javlja kombinatorna eksplozija)
- Robotika (algoritmi za kontrolu ponašanja robota)
- Računarska grafika (analiza i sinteza slika i animacija)
- Kriptografija (algoritmi za zaštitu privatnosti podataka)
- Teorijsko računarstvo (teorijske osnove izračunavanja, računarska matematika, verifikacija softvera, itd).

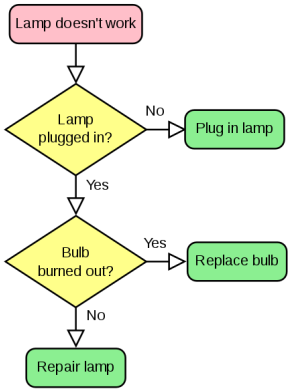
1.6 Osnovni pojmovi i konstrukti

Programiranje predstavlja proces zapisivanja algoritama u nekom programskom jeziku. Algoritam predstavlja precizan opis postupka za rešavanje nekog problema u konačnom broju koraka. Algoritmi se odlikuju svojom složenošću. Ta složenost može biti vremenska ili memorijska. Vremenska složenost se odnosi na vreme potrebno za izvršavanje nekog algoritma. Memorijska složenost označava koliko memorijskih resursa je potrebno za izvršavanje algoritma. Cilj analize algoritama je predviđanje njegovog ponašanja i brzine izvršavanja bez realizacije na nekom konkretnom računaru. Ta procena treba da se odnosi na svaki računar. Nemoguće bi bilo na svakom računaru ispitati izvršavanje nekog algoritma. Zbog toga je analiza algoritama približna tehnika. [?] Postoji uniformna tehnika za grafički prikaz algoritama, međutim, mi nećemo ulaziti u detalje, već će biti dato nekoliko ilustrativnih primera kako bi se dobila glavna ideja.

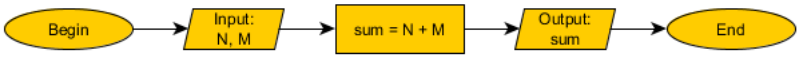
Primer 1.4 *Kao najjednostavniji primer algoritma uzmimo primer sa lampom. U početnom koraku lampa je ugašena, a mi želimo da je upalimo ili, ako ne radi, da je odnesemo na popravku. Najpre, proveravamo da li je lampa uključena u struju. Romb je znak kojim označavamo uslov. Ako lampa nije uključena u struju treba je uključiti. U suprotnom, ako je lampa uključena u struju, ali ne svetli, proveravamo da li je sijalica pregorela. Ako jeste, menjamo sijalicu. Ako nije, lampa je pokvarena i moramo je popraviti. Na slici 7 vidimo kako bismo grafički prikazali opisani postupak.*

Primer 1.5 *Kao primer, više prikladan računarskoj terminologiji, uzmimo sabiranje* 2 *broja, M i N. Na slici 8 vidimo kako bismo grafički prikazali postupak sabiranja brojeva.*

U sekciji 1.2 koja govori o fon Nojmanovoj arhitekturi, videli smo da se podaci (i programi) smeštaju u memoriju računara, najčešće u vidu niza bitova. Međutim, kako se programer ne bi zamarao detaljima i kako bi mogao da radi na dosta apstraktnijem nivou, programski jezici obezbeđuju koncept promenljivih. Promenljive daju mogućnost programeru da podacima dodeli imena i da im na osnovu tih imena i pristupa. Svaka promenljiva u programu ima dodeljen određeni niz bajtova.



Slika 7: Primer najjednostavnijeg algoritma.



Slika 8: Prikaz algoritma za sabiranje dva broja.

Promenenljive se odlikuju svojim tipovima i životnim vekom. Životni vek je koncept koji nam govori u kom delu faze izvršavanja programa je promenljivoj dodeljen memorijski prostor, odnosno, kada je možemo koristiti. Životni vek promenljive omogućava da na različitim mestima u programu koristimo različite promenljive istog imena i pravilo dosega identifikatora (engl. scope) određuje deo programa u kome se uvedeno ime može koristiti. Druga odlika promenljivih su tipovi. Organizovanje podataka u tipove pomaže programeru da ne mora da razmišlja o podacima na nivou njihove binarne reprezentacije, već daleko apstraktnije. Neki od najčešćih tipova su:

- celi brojevi (... -3, -2, -1, 0, 1, 2, 3,...),
- brojevi u pokretnom zarezu (1.0, 3.14, 9.81,...),
- karakteri (a, b, P, „ !, ...),
- niske (ždravo", švima",...)

Osim ovih postoje i složeniji tipovi koji mogu objediniti više istih ili različitih tipova, pa tako imamo nizove, strukture, liste, i dr. Svaki tip podataka se karakteriše vrstom podataka koje opisuje, skupom operacija koje se nad njime vrše i načinom reprezentacije i detaljima implementacije tog tipa.

Osnovni gradivni elementi imperativnih programskih jezika su naredbe. Naredba dodele je osnovna naredba i njom se vrednost neke promenljive postavlja na vrednost nekog izraza definisanog nad konstantama i promenljivim. Šta to, u praksi, znači? To znači da kada kažemo $x = 3 * y$ zapravo promenljivoj x dodeljujemo vrednost $3 * y$. Naredbe se u programu nižu jedna za drugom,

osim u slučaju korišćenja naredbi za kontrolu toka izvršavanja programa. Ove naredbe u zavisnosti od tekućih vrednosti promenljivih neke naredbe mogu da ne izvršavaju, izvršavaju ih više puta(petlje) i slično. Najčešće korišćene kontrolne strukture su granajuće naredbe (if-then-else), petlje (for, while, do-while, repeat-until) i naredbe skoka (goto). Naredba if-then-else ima sledeći opšti oblik:

```
if izraz1
   naredba1
else
   naredba2
```

Treba obratiti pažnju da je else grana neobavezna, odnosno, može, ali ne mora, da postoji.

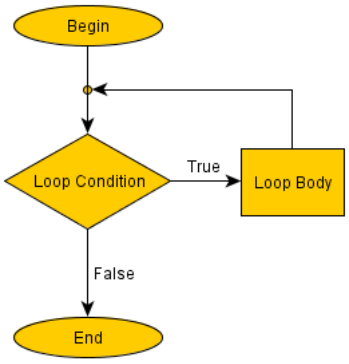
Primer 1.6 *Naredni primer štampa veći od dva broja.*

```
if a > b
   print(a)
else
   print(b)
```

Primer 1.7 *Naredni primer u promenljivu a smešta tekst "Hello world!" ako je vrednost promenljive b veća od 0.*

```
if b > 0
   a = "Hello world!"
```

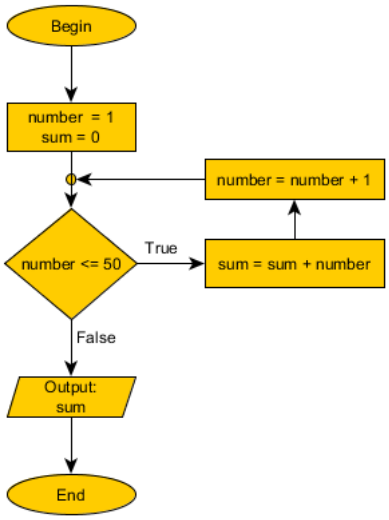
Jedan od najbitnijih koncepata programiranja je pojam *petlje*. Petlje predstavljaju konstrukcije koje nam omogućuju da izvršavamo jednu ili više akcija više puta. Sastoje se iz uslova i tela petlje. Uslov nam definiše u kom slučaju ćemo napustiti petlju (na primer, ako broj iteracija[7] petlje predje 100, ili ako je neka promenljiva n veca od 1000, itd.). Telo petlje sadrži akcije koje želimo da ponavljamo.Na slici 9 može se videti opšti oblik petlji.



Slika 9: Petlja.

[7] Iteracija petlje predstavlja jedno izvršavanje koda koji se nalazi u telu petlje.

Primer 1.8 *Za naredni primer, uzećemo sabiranje prvih 50 brojeva. Naime, naporno bi bilo da pišemo* $1+2+3+...+50$*, a da ne pomišljamo na brojeve poput* 1000, 100000 *ili* 1000000*. To bi bilo gotovo neizvodivo. Zbog toga, želimo da na apstraktniji način opišemo proceduru koja će umesto nas izvršiti sabiranje prvih 50 brojeva. Da bismo to uspeli moramo iskoristiti petlju. Na slici 10 vidimo kako bismo grafički prikazali postupak.*



Slika 10: Prikaz algoritma za sabiranje prvih 50 brojeva.

Petlje su neophodne za uspešno programiranje. Postoji nekoliko različitih vrsta petlji, čija upotreba zavisi od onoga što njom želimo da postignemo, pa tako imamo:

- "*for*" petlju i
- "*while*" petlju

koje predstavljaju osnovne konstrukte u skoro svim programskim jezicima. Osim for i while petlje postoje i druge, ali o njima nece biti reči.

For petlja je najčešće oblika:

```
for(izraz1, izraz2, izraz3)
    naredba
```

*Izraz*1 i *izraz*3 obično predstavljaju naredbe dodele ili inkrementiranje, gde se *izraz*1 obično naziva inicijalizacija, a *izraz*3 je korak. Izraz u sredini, *izraz*2, predstavlja relacijski izraz i služi kao uslov izlaska iz petlje. *Naredba* predstavlja liniju ili blok koda koji želimo da se izvršava. Kao posledica navedenog, *for* petlju možemo posmatrati kao:

```
for(inicijalizacija, uslov izlaska, korak)
    kod koji želimo da se izvrši
```

Primer 1.9 *Naredni primer predstavlja jedan od uobičajenih oblika u kojima se for petlja pojavljuje:*

```
for(i = 0; i < n; i++)
    print(i)
```

Ovaj primer za svaki korak petlje ispisuje broj koraka.

1.7 Klasifikacija programskih jezika

Brojnost programskih jezika raste iz godine u godinu. Stalno se pojavljuju novi i brži jezici, unapređuju se stari i nemoguće je ispratiti sve promene. Da bismo odmah razumeli okvirno kako neki programski jezik funkcioniše moramo znati kojoj paradigmi pripada. Kada kažemo da neki jezik pripada nekoj paradigmi mi zapravo govorimo nešto o karakteristikama tog jezika koje važe za sve jezike koji pripadaju istoj grupi. Programske paradigme su formirane prema načinu programiranja. Neki od najkorišćenijih programskih jezika današnjice spadaju u grupu imperativnih programskih jezika, npr. jezik C. Glavna karakteristika imperativnih programskih jezika je da stanje programa karakterišu promenljive kojima se predstavljaju podaci i naredbe kojima se vrše određene transformacije nad promenljivim (sabiranje, oduzimanje, poređenje, itd.). Osim imperativne, značajne programske paradigme su i objektno-orijentisana (C++, Java (Java NIJE JavaScript!), C# itd.), funkcionalna (Lisp, Haskell, ML, itd.), logička (u nju spada, na primer, Prolog). Sa razvojem savremenih programskih jezika došlo je do brisanja jasnih granica između ovih jezika, pa tako dolazi do mešanja karakteristika različitih paradigmi.

Za većinu programskih jezika danas reći ćemo da su proceduralni. Kada kažemo da je neki jezik proceduralan zapravo želimo da iskažemo činjenicu da je zadatak programera da opiše način (proceduru) kojim će se doći do rešenja problema. Kao idejno potpuno kontrastni, postoje deklarativni programski jezici (poput Prologa) koji od programera zahteva precizan opis problema, a mehanizam programskog jezika se onda bavi pronalaskom rešenja.

Prema tipu konverzije tipova imamo statički i dinamički tipizirane jezike. Kod statički tipiziranih jezika (poput C-a) zahteva se da programer definiše tip svake promenljive i da ga potom više ne menja tokom izvršavanja programa. Kod dinamički tipiziranih jezika ista promenljiva može sadržati podatke različitog tipa tokom različitih faza izvršavanja. Nekada je moguće čak i vršenje operacija nad promenljivima različitog tipa, pri čemu dolazi do implicitne konverzije. Na primer, jezik JavaScript ne zahteva definisanje tipa promenjlivih i dopušta kôd poput $a = 1; b = "2"; a = a + b;$

1.8 Primeri za vežbu

Primer 1.10 *Napisati primer algoritma za kuvanje kafe.*

Primer 1.11 *Napisati primer algoritma za provodjenje jednog dana.*

Primer 1.12 *Napisati primer algoritma za savladavanje nekog kursa.*

Primer 1.13 *Napisati primer algoritma za računanje zbira prvih 10 parnih brojeva.*

Primer 1.14 *Koju vrednost ima promenljiva x nakon izvršavanja narednog koda:*

```
int x = 0;
if (x > 3);
    x++;
```

Primer 1.15 *Napisati pseudo kod za računanje zbira 3 broja.*

Primer 1.16 *Napisati pseudo kod u kome se promenljivoj a dodeljuje vrednost* 3*, promenljivoj b dodeljuje vrednost* 6 *i onda se njihov zbir smešta u promenljivu c. Nakon toga, proveriti da li ostatak pri deljenju promenljive (računa se uz pomoć %) c sa brojem 2 daje 0, odnosno, da li je c paran.*

Primer 1.17 *Napisati pseudo kod*[8] *algoritma za računanje zbira prvih 10 parnih brojeva.*

Primer 1.18

1.9 Domaći zadatak

Domaći zadatak:

- pročitati nešto dodatno o istorijatu računarstva (01_istorijat),
- opisati ukratko svaki od elemenata memorije (01_piramida),
- odraditi sve primere i pitalice ,
- domaci sa osi/tcp slojevima: 01_osi, 01_tcp
- domaci koji je size teksta o browserima: 01_browser
- samostalno pronaći još 10 novih i uraditi ih(01_0,...,01_9).

[8] Pseudo kod predstavlja kod koji nije dat u formalnim terminima, već predstavlja ideju kako bi kod trebao da izgleda.

2 Uvod u web

Danas, ne možemo zamisliti korišćenje računara bez veza ka drugim računarima. Izgradnja računarskih mreža, a posebno sa nastankom i razvojem Interneta i njegovih servisa poput Veba, dovele su do porasta broja korisnika računara i promene uloga računara u odnosu na ranije. Pojava savremenih računarskih mreža smatra se revolucionarnom poput pojave parne mašine u 18. veku. Svake godine uvećava se broj umreženih računara, a sa tim brojem raste i broj usluga koje nam mrežno okruženje nudi. Neke od osnovnih primera upotreba računarskih mreža obuhvataju:

- poslovna: elektronska pošta, razmena datoteka, deljeni štampači, ...
- kućna: filmovi, muzika, igrice, vesti, audio i video komunikacija, razmena poruka, elektronska kupovina,...
- mobilne: pozivi, SMS, igrice, mape, pristup informacijama

2.1 Uloga računarskih mreža

U osnovne uloge računarskih mreža ubrajamo:

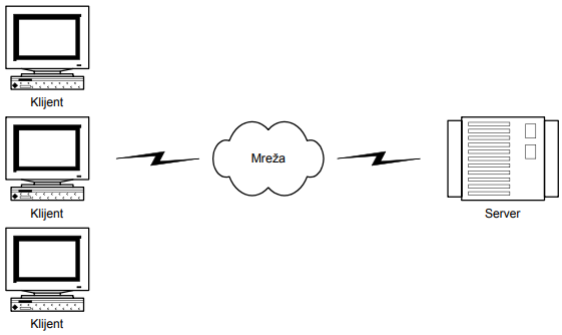
1. komunikaciju - uz pomoć računara ljudi razmenjuju poruke, video pozive, mejlove, ćaskanja (eng. chat), video konferencije, itd.
2. deljenje informacija i podataka - ako postoji mrežno okruženje u kom su računari povezani, tada je moguće pristupiti informacijama na drugim računarima u okviru mreže. Podatke prenosimo na više načina, kao što su preuzimanje datoteka, prenos informacija u okviru lokalnih mreža (obično u okviru jedne kompanije), kao i u okviru globalne svetske mreže. Internet i veb se smatraju glavnim izvorima informacija.
3. deljenje softvera - korisnici povezani u mrežu mogu koristiti mnoge usluge koje im pruža softver koji radi na računarima u okviru mreže. Neke od usluga su kupovina i rezervacija karata preko interneta, ili izvršavanje softvera koji je distribuiran i paralelizovan na više povezanih računara, čime se može ubrzati izvršavanje.
4. deljenje hardverskih resursa - obezbeđuje zajedničko korišćenje hardvera, poput štampača, skenera i ostalih. Često se ovakav pristup koristi u kompanijama.

Računarski resursi u mreži mogu biti raspoređeni na različite načine, tako da obezbeđuju različite načine izvršavanja poslova. Neki od najčešćih su:

- Centralizovana obrada - svi poslovi se izvršavaju na jednom centralnom računaru, dok se ostali uređaji u mreži koriste samo kao terminali za unos podataka i prikaz rezultata. Ovakavim načinom rada odlikovale su se rane računarske mreže.
- Klijent-server okruženje - jedan računar ima ulogu servera na kome se nalaze podaci i aplikativni softver, koji se stavljaju na raspolaganje klijentima. Serveri su obično moćniji računari od klijenata(mada ne mora uvek biti tako) i na njima se obavljaju zadaci koji zahtevaju više resursa.

U današnjem kontekstu, stroga podela na klijentski i serverski računar više nije tako aktuelna. Najčešće govorimo o tome da je jedan računar istovremeno i klijent i server u zavisnosti od zadataka koji su mu zadati. Na primer, isti računar može istovremeno pokretati i Veb server i klijent za elektronsku poštu, čime mu je data i uloga servera i uloga klijenta. na njihov zahtev.

- Mreža ravnopravnih računara (eng. peer-to-peer - $P2P$) - računari direktno komuniciraju deleći podatke i opremu. Sve se češće ovakve mreže koriste za masovnu razmenu velikih količina podataka (npr. torenti - Bittorrent).



Slika 11: Prikaz klijent-server arhitekture.

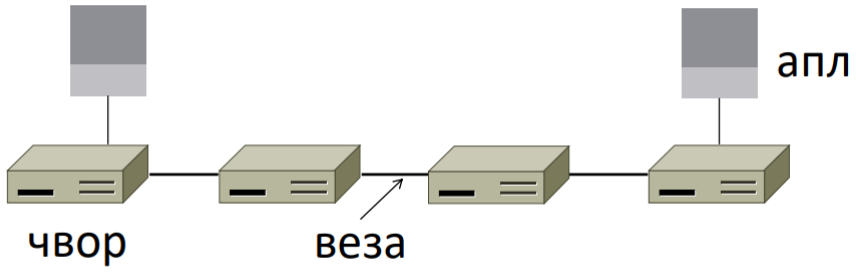
2.2 Komponente računarskih mreža

Pre nego što uđemo u detaljniji opis elemenata koji čine jednu računarsku mrežu, trebalo bi da damo formalnu definiciju šta je računarska mreža. Naime, računarska mreža je sistem koji se sastoji iz skupa hardverskih uređaja koji su međusobno povezani komunikacijskom opremom i koji je snabdeven odgovarajućim kontrolnim softverom kojim se ostvaruje kontrola funkcionisanja sistema tako da je moguć prenos podataka između povezanih uređaja. Neke od osnovnih komponenti računarskih mreža su, dakle:

- mrežni hardver,
- komunikacioni kanali i
- mrežni softver.

2.2.1 Mrežni hardver

Tradicionalno, podrazumeva se povezivanje računara u okviru mreže, ili uz dodatak nekih pomoćnih uređaja poput štampača, skenera itd., kako bi se mogli deljeno koristiti. Međutim, u poslednje vreme, granica između klasičnih računara i digitalnih uređaja specijalizovane namene se briše i sve češće se u oviru mreže mogu povezati i PDA uređaji, mobilni telefoni, foto aparati, kamere



Slika 12: Prikaz mreže.

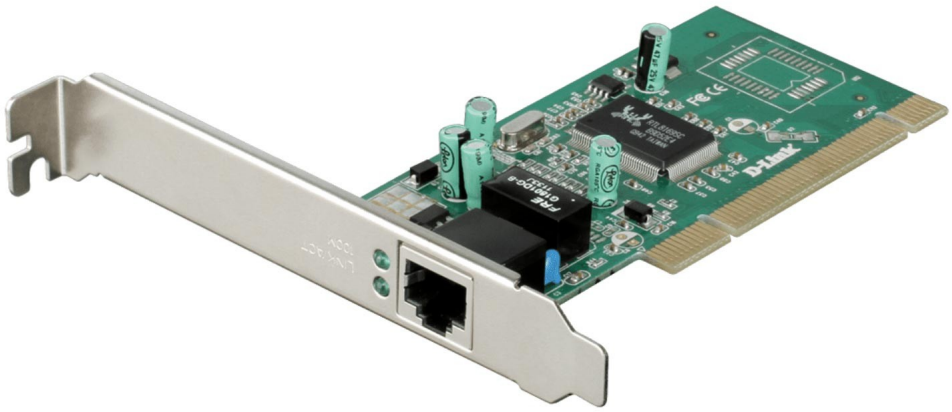
i ostali. Aktivno se radi i na razvoju automobila, frižidera i ostalih mnogobrojnih uređaja kako bi se uključili u mrežu i kako bi se time omogućilo upravljanje istima na daljinu.

Da bismo neki uređaj mogli da ubacimo u mrežu, neophodno je da sadrži određene hardverske komponente koje bi mu to omogućile. Deo hardvera koji je namenjen za umrežavanje i spada u komunikacionu opremu je mrežna kartica ili mrežni adapter (eng. NIC - network interface card) koja omogućava fizički pristup mreži. Svaka mrežna kartica ima svoju jedinstvenu fizičku (MAC) adresu, kojom se uređaj jedinstveno identifikuje prilikom komunikacije. Neke mrežne kartice obezbeđuju pristup žičanim, a neke druge bežičnim komunikacionim kanalima. Osim mrežnih kartica za umrežavanje se koriste i modemi (telefonski, kablovski), kao i još neki uređaji o kojima neće biti reči.

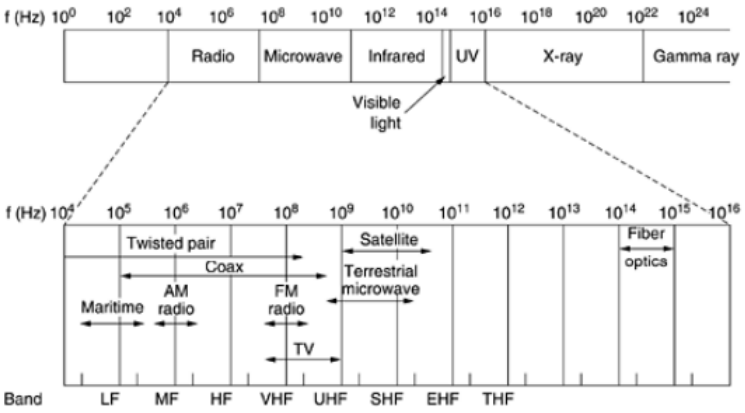
2.2.2 Komunikacioni kanali

Da bi mreža funkcionisala i da bi kroz nju bilo moguće preneti podatke, uređaji koji se nalaze u mreži moraju biti povezani međusobno uz pomoć žičanih ili bežičnih prenosnih sistema, koji predstavljaju komunikacione kanale. Osnovna mera kvaliteta komunikacionog kanala je brzina prenosa koja se meri u bitovima po sekundi (bit/s). Ova mera označava broj bitova koji se mogu preneti kroz komunikacioni kanal u jednoj sekundi. Ako bismo posmatrali aktuelne tehnologije prenosa podataka, najčešće se koriste megabiti (milion bita) u sekundi - Mbps, ili gigabiti (milijarda bita) u sekundi - Gbps. Brzina prenosa predstavlja fizičku karakteristiku komunikacionog kanala i zavisi od frekvencijskog opsega (eng. bandwidth) koji se može propustiti kroz kanala bez gubitka signala. Na slici 14 prikayan je raspon frekvencija za razne prenosne tehnologije.

Komunikacione kanale možemo podeliti u dve grupe prema tipu prenosa



Slika 13: Prikaz mrežne kartice.



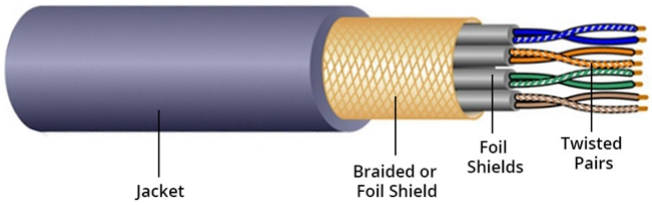
Slika 14: Prikaz frekvencijskih opsega za razne prenosne tehnologije.

informacija:

1. Žičane
2. Bežične

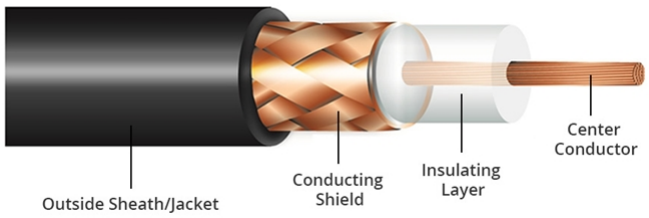
Žičane komunikacije

Parice (eng. twisted-pair wire) - Najkorišćeniji način komunikacije. Uređaji se povezuju korišćenjem uvijenih uparenih izolovanih bakarnih žica. Žice se uparuju i uvijaju kako bi se smanjile smetnje u komunikaciji. Brzina prenosa kroz ovakav medijum obično varira od $2Mbps$ do $100Mbps$.



Slika 15: Prikaz parice.

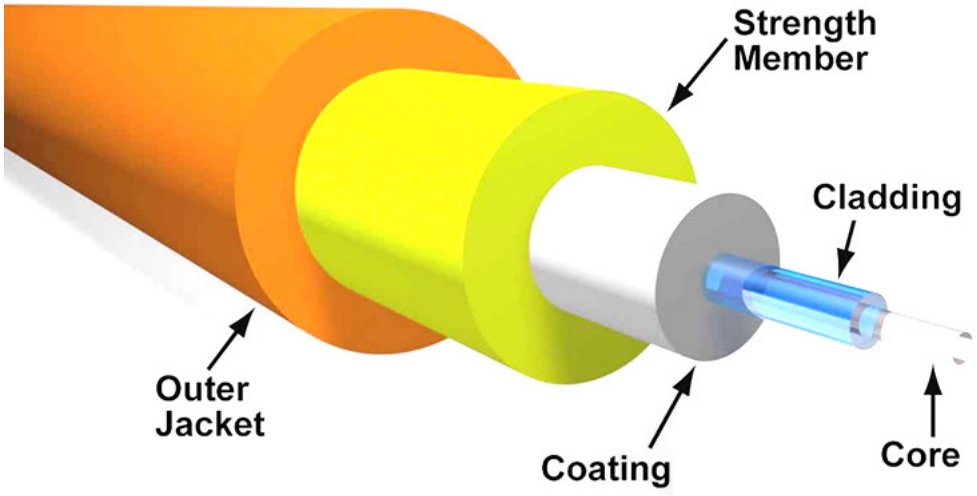
Koaksijalni kablovi svoju upotrebu najčešće nalaze u televizijskim kablovskim sistemima, a koriste se i u lokalnim mrežama u kompanijama. Kablovi se sastoje od centralne bakarne ili aluminijumske žice obmotane savitljivim slojem izolacije, oko kog je obmotan provodni sloj tankih žica, a potom je sve to izolovano. Ovaj tip kablova omogućava brzinu prenosa do $200Mbps$ (nekad čak i do $500Mbps$), uz manju osetljivost na elektromagnetne smetnje.



Slika 16: Prikaz koaksijalnog kabla.

Optički kablovi - prave se od velikog broja (reda veličine nekoliko stotina ili hiljada) veoma tankih staklenih vlakana. Podaci se prenose svetlosnim talasima uz pomoć malog laserskog uređaja. Na ovaj tip kablova elektromagnetne smetnje nemaju uticaja. Najveći nedostatak ovakvih kablova je cena, izuzetno su skupi i komplikovani za komunikaciju, pa se uglavnom koriste za osovinski deo mreže, na koji se potom drugim vrstama kablova povezuju pojedinačni uređaji. Brzina ovih uređaja predstavlja njihovu najveću prednost. Naime, brzina ovih uređaja može ići i do nekoliko triliona bita u sekundi. Najčešće se koriste za mreže sa brzinama do $10Gbps$.

Bežične komunikacije Bežična komunikacija, kao što i samo ime sugeriše, ne koristi kablove za prenos podataka. Ovakav vid komunikacije poseban



Slika 17: Prikaz optičkog kabla.

značaj nalazi kod prenosivih računara, mobilnih telefona ili dosta udaljenih lokacija do kojih bi bilo jako skupo, ako ne i nemoguće, sprovesti kablovsku mrežu. Umesto kablova ove mreže koriste radio talase, mikro talase i infracrvene zrake. Podaci se prenose moduliranjem amplitude, frekvencije ili faze talasa. Neke od danas najkorišćenijih tehnologija su:

- Bluetooth - koristi se za veoma male razdaljine (do 10 ili do 100 metara. Brzina prenosa je do $3Mbps$. Bluetooth tehnologija koristi radio talase i može da prođe i kroz čvrste prepreke. Koristi se najčešće za komunikaciju računara sa periferjskim uređajima, kao i u mobilnoj telefoniji.

- Bežični LAN - Wireless LAN (WLAN, WiFi) je tehnologija koja korsti radio talase za bežičnu komunikaciju više uređaja na ograničenim rastojanjima (nekoliko desetina ili stotina metara). Brzina prenosa ide od $10Mbps$ do $50Mbps$ (u skorije vreme može ići i do $600Mbps$). Najrašireniji standard za ovaj vid komunikacije je IEEE 802.11, o kome će kasnije biti više reči.

- Ćelijski sistemi - Način prenosa je sličan onom koji se koristi u mobilnoj telefoniji. Za komunikaciju se koriste radio talasi i sistemi antena koji pokrivaju određenu geografsku oblast, pri čemu se signal do cilja prenosi preko niza antena.

- Zemaljski mikrotalasi - koriste antensku mrežu na Zemlji, a za komunikaciju koriste mikrotalase niske frekvencije koji zahtevaju da antene budu optički vidljive, pa se iste smeštaju na visoke tačke.

- Komunikacioni sateliti - koriste mikrotalase za komunikaciju tako što se prenos između dve tačke koje nemaju optičku vidljivost ostvaruje poprečnom komunikacijom preko satelita koji se nalaze u orbiti. Na ovaj

način se prenose televizijski i telefonijski signal. Brzina komunikacije je dosta mala $100Mbps$.

2.2.3 Mrežni softver

Mrežna infrastruktura sama po sebi ne služi ničemu bez mrežnog softvera. Uloga mrežnog softvera je da obezbedi korisniku mrežnu komunikaciju. Na primer, programer pregledača Veba, ne treba da misli o tome kako će pregledač primiti informacije, već treba da se fokusira samo na aspekte značajne za njegovu konkretnu aplikaciju, a sve ostale detalje prepusti nižem sloju mrežnog softvera.

Mrežni softver, najgrublje, može da se podeli na dva nivoa:

- niskog nivoa - mrežni softver koji omogućuje korišćenje različitih mrežnih uređaja, poput mrežnih kartica, modema, itd. Ovaj softver se nalazi u jezgru operativnog sistema i u obliku upravljača perifernim uređajima, takoznavnih drajvera (eng. driver). On upravlja računarskim hardverom i komunikacijskom opremom. Korisnik nikad ne koristi ovaj softver direktno, a često nije ni svestan njegovog postojanja
- visokog nivoa.

2.3 Literatura

Čitanje literature predstavlja neizostavan deo gradiva, kako naredni izvori predstavljaju odlican izvor informacija, na vama je da ih procitate:

- Predrag Janičić, Programiranje 1, Matematički fakultet, glava 1
- Filip Marić, Uvod u web i internet tehnologije, Matematički fakultet, glave 1 i 2
- Ajzenhamer Nikola, Bukurov Anja, Stanković Vojislav, Programiranje za Veb skripta, glave 1,2 i 3

3 HTML i CSS

HTML i CSS predstavljanju neizostavan materijal pri učenju web programiranja. Upravo zbog njihovog velikog značaja im posvećujem celo poglavlje. Slajdovi korišćeni prilikom predavanja su apsolutno nedovoljan materijal. Svi materijali korišćeni pri kreiranju časova, kao i ovog materijala su javno dostupni i nalaze se na:

- Aleksandar Veljković, Veb programiranje, Matematički fakultet http://poincare.matf.bg.ac.rs/ aleksandar/files/web/skripta.pdf
- W3Schools: https://www.w3schools.com/
- Filip Marić, Uvod u Veb i Internet tehnologije, Matematički fakultet

3.1 HTML

3.1.1 Zadaci sa časa

Neki od zadataka rađenih na času su navedeni u nastavku.

Primer 3.1 *Prva HTML stranica: Treba kreirati svoju prvu HTML stranicu korišćenjem tagova. Propozicije stranice su sledeće:*

- *Kreirati naslov stranice i iskoristiti barem 3 nivoa heading-a*
- *Napisati neki pasus koji ima smisla, u okviru kog treba iskoristiti break, strong i em tagove*
- *Potom kreirati isto to za ostale nivoe headinga*
- *Kreirati jos barem 3 stranice, od čega: 2 treba da sadrže smisleni tekst u paragrafima, a poslednja može sadržati tekst u vidu lorem ipsum dolor sit...*
- *Druga strana treba da sadrži linkove ka barem 3 spoljašnje strane, npr: ujutru uz kafu volim da čitam, pa linkove ka nekoliko portala*
- *Svaka strana treba da sadrži veze ka svim ostalim stranama*

Primer 3.2 *Španska kuhinja: Kreirati sajt prema sledećim propozicijama:*

1. *Iz kojih jela se sastoje tipični obroci u Španiji (entrada, primer plato, segundo plato,...)?*
2. *Kreirati osnovnu stranicu sa opisom delova obroka, a potom za svaki deo obroka kreirati posebnu stranicu.*
3. *Svaka stranica mora da sadrži*
    - *Naslov, npr.: Primer plato*
    - *Podnaslov, naziv nekog tipičnog jela karakterističnog za taj obrok, npr.: Nachos (za entradas)*
    - *Pasus, nešto o tom jelu, odakle je poteklo, kad je nastalo i slično*
    - *Pasus sa neuređenom listom potrebnih sastojaka*

- *Pasus sa uređenom listom postupka pripreme*

4. *Nekoliko slika jela*
5. *Vezu ka prethodnoj i narednoj stranici*
6. *Poslednja stranica treba da se vraća na prvu i da ima spoljašnju vezu ka opisu nekog grada u Španiji.*

Primer 3.3 *Sportski izveštaj: Kreirati stranicu o sportu. Odabrati sport po izboru, a potom ispuniti specifikaciju:*

1. *Napraviti definicionu listu koja opisuje odabrani sport*
2. *Napraviti listu sa pravilima igre, ako je moguće kreirati listu u listi (ugnježdena lista)*
3. *Kreirati tabelu sa rezultatima nekog takmičenja*
4. *Ubaciti dve ili više slika (iskoristiti width i height kako bi se veličina slike prilagodila)*
5. *Ubaciti tabelu sa rezultatima sa nekog većeg takmičenja iz tog sporta*

Primer 3.4 *Stiven Hoking: Kreirati sajt koji će sadržati informacije o Stivenu Hokingu (eng. Stephen Hawking). Specifikacija sajta je sledeća:*

- *Sajt treba da sadrži naslov, kome će u tooltip-u stajati: engleski teoretski fizičar, kosmolog, autor i direktor istaživanja u Centru za teorijsku kosmologiju na Univerzitetu u Kembrdžu.*
- *Ispod naslova treba da stoji kratki paragraf o Stivenu Hokingu, paragraf treba da sadrži boldovan i italic tekst, kao i nešto što bi bilo highlightovano i precrtano*
- *Kreirati listu sa spiskom njegovih publikacija*
- *Ubaciti sliku, i skalirati je korišćenjem odgovarajućih atributa*
- *Ubaciti neki citat na odgovarajući način*
- *Ubaciti citat nečega sa njegove stranice, i potom referisati stranicu.*
- *Ubaciti vezu ka drugoj stranici na kojoj će biti slike i nazivi nekih od njegovih publikacija*

Primer 3.5 *Forma za registraciju za učešće na nekom kursu. Zadatak je kreirati stranicu koja će sadržati formu, koja bi trebalo da se popuni kako bi se prijavilo za učešće na nekom od kurseva. Sami odaberite nazive kurseva, kao i relevantne informacije polaznika.*

- *Ime*
- *Prezime*
- *Adresa*

- *Grad*
- *Broj telefona*
- *Broj mobilnog*
- *e-mail adresa*
- *kurs za koji se prijavljuje - lista od barem 5 izbora*
- *odabir termina: vikendom, radnim danima ili svejedno - radio buttoni*
- *checklista: radim na svom računaru/ potreban mi je računar*
- *text area u kojoj treba upisati prethodno iskustvo*
- *button za slanje informacija*

Neke od informacija o polaznicima kurseva su obavezne, neke nisu, sami odredite koje jesu. Osim navedenih treba dodati jos barem 2 dodatne informacije po izboru u različitim oblicima.

3.2 CSS

3.2.1 Zadaci sa casa

Primer 3.6 *Kreirati stranicu zdrave hrane.*

3.2.2 Domaći zadaci

Domaći zadaci vezani za ovo poglavlje se odnose na unapređivanje i dodavanje sadržaja i isprobavanje tagova i elemenata nad zadacima rađenim na časovima.

4 JavaScript

JavaScript (skraćeno JS) je skript jezik, nastao 1995. godine kao jezik za pregledač Netscape Navigator kako bi se omogućile programske sposobnosti veb stranicama. Osnovna uloga JavaScripta na Vebu je programiranje korisnickog interfejsa. ECMAScript standard je standard kojim se definiše ponašanje svih pregledača koji podržavaju JavaScrpit. Bitno je naglasiti da programi napisani jezikom Javascript se izvršavaju na klijentskoj mašini (iako postoje i upotrebe na serveru). Uz HTML i CSS, JS predstavlja jezgro tehnologija korišćenih na Vebu.

4.1 Osnovni konstrukti jezika

Pod osnovne konstrukte jezika podrazumevamo minimalan skup neophodnih znanja i pravila potrebnih za rad u određenom programskom jeziku. Osnovni konstrukti su idejno veoma slični, pa i isti, za većinu viših programskih jezika i dobrim poznavanjem osnova barem jednog programskog jezika višeg nivoa, moguće je dosta lako savladavanje i ostalih. U ovom delu biće predstavljeni osnovni konstrukti jezika vezani konkretno za JavaScript.

JavaScript kod možemo ubaciti u postojeću HTML datoteku na naredna dva načina:

- kao novu datoteku u headu: <script src='datoteka.js'></script>
- kroz HTML, na kraju body-ja, između <script> tagova.

4.1.1 Promenljive i tipovi

Nakon što smo na odabrani način obezbedili prostor za unošenje JS koda, želimo da kreiramo neku promenljivu. Za JS se kaže da je slabo tipiziran jezik, to znači da tip promenljive koju kreiramo ne zadajemo eksplicitno, već je on određen tipom vrednosti koju ta promenljiva čuva. Veoma bitno je naglasiti da se sa promenom sadržaja promenljive menja i njen tip u toku izvršavanja programa. Ako bismo na primer u promenljivu *a* stavili broj 13, a kasnije stavili nisku "programer", promenljiva *a* bi najpre imala tip *number*, a potom *string*.

Postoji nekoliko različitih tipova koje možemo smestiti u promenljivu, to su:

1. Brojevi:
   - celi brojevi: ...,-10, -5, 0, 1, 7, 13,...
   - razlomljeni brojevi: 3.14, 9.81, ...
   - beskonačnosti: Infinity, -Infinity
   - NaN - not a number - označava da nešto nije broj, dobija se kao rezultat nedefinisanih operacija (0/0, Infinity/Infinity,...)
2. Niske (stringovi) - predstavljaju nizove karaktera: " Mi volimo programiranje "
3. Bulove vrednosti: True, False

4. Prazne vrednosti: null, undefined

U nastavku će biti ukratko opisano kreiranje, korišćenje i osnovne operacije nad navedenim tipovima. U promenljive je moguće smestiti i složenije tipove kao što su nizovi, objekti i drugi, o kojima će, takođe, biti reči u nastavku.

Kako bismo kreirali promenljivu u programu, navodimo ključnu reč *var* (postoje jos neke ključne reči za kreiranje promenljivih, ali će o njima biti reči kasnije), a potom naziv promenljive, koji ne sme počinjati brojem ili biti ključna reč jezika.

```
var a;
```

Ovako smo samo kreirali promenljivu, ona za sada nema nikakvu vrednost u sebi, a samim time i njen tip nije definisan. Kada želimo da promenljivoj dodelimo neku vrednost to činimo korišćenjem naredbe dodele $*$, pa tako sa

```
a = 5;
```

smeštamo celi broj 5 u promenljivu a, i njen tip postaje *number*. Moguće je i pri kreiranju odmah dodeliti vrednost promenljivoj *a*, ovaj postupak nazivamo inicijalizacijom ili dodeljivanjem početne vrednosti promenljivoj.

```
var a = 5;
```

JavaScript dopušta još jednu stvar, a to je kreiranje promenljive bez navođenja ključne reči var ukoliko bismo joj odmah dodelili vrednost. Napomena: ovo se ne smatra dobrom praksom, a ukoliko bi stajalo samo b bez *var* i bez $*7$ program bi izbacio grešku.

```
b = 7;
```

Kada se dodeljuju nazivi promenljivim treba obratiti pažnju da nazivi nisu ključne reči jezika, kao i da ne počinju brojem.

Brojevi U JavaScript-u možemo predstaviti cele brojeve - int: -1, -32, 0, 13, 173,itd., kao i razlomljene brojeve - float: 3.14, 9.81,itd.. Da bismo broj smestili u neku promenljivu koristimo operator dodele.

```
var a = 5;
```

Osim operatora dodele =, postoje i drugi operatori. Najčešće operatore primenjujemo nad brojevnim (numeric) vrednostima. Nad numeričkim vrednostima definisani su:

- Unarni $-$: -4
- Binarni $+$, $-$, $*$, /, %, =, <, >, <=, >=

Ovi operatori imaju standarne prioritete. Operator % predstavlja ostatak pri deljenju dva broja.

```
var c = a + b; //12
```

Osim ovih binarnih operatora postoje i binarni operatori za proveru jednakosti ==, nejednakosti ! =, za proveru jednakosti sa proverom tipova ===, operatori koji vrše konjunkciju && i disjunkciju ||, kao i složeni operatori ++(unarni), +=, -=, *=,itd.. O njima će biti više reči u nastavku.

Da bismo broj zapisali sa određenim brojem decimala koristimo metod[9] *toFixed*(), kome kao jedini argument šaljemo broj decimala koji želimo da zadržimo.

```
var pi = 3.1415;
pi = pi.toFixed(2); // pi = 3.14
```

Pisanje komentara Pisanje komentara predstavlja dobru praksu i može biti od velike pomoći programeru. Najčešće se radi nad nasleđenim kodovima, koje su pisali programeri pre nas, pa tako komentari mogu dosta olakšati rad i razumevanje koda. Osim toga, vrlo često prilikom vraćanja na ranije pisane kodove, dešava se da nismo sigurni kako smo nešto napisali, iako nam je u datom momentu to delovalo sasvim jednostavno i logično, pa nam komentari mogu pomoći da se prisetimo šta je bila glavna ideja. U početničkom programiranju, komentari pomažu kako bi se neki koncepti utvrdili, dodatno razjasnili i definisali, pa autor predlaže pisanje komentara što češće.

Komentari mogu biti: linijski i blokovski. Linijski komentari, kao što im i samo ime kaže imaju doseg jedne linije (koja ne bi trebalo prema konvenciji da sadrži više od 80 karaktera!) i pišu se nakon znaka //. Blokovski komentari imaju doseg od više linija i pišu se između /**/.

```
//ovo je linijski komentar
/*Ovo
je
blokovski*/
var c = a + b; //12
```

Konzola Konzola omogućava programeru lakše debagovanje koda, lakše pronalaženje grešaka, izbacuje greške, upozorenja i obaveštenja. U konzoli, takođe, možemo ispisati rezultate nekih izvršavanja programa, vrednosti i tipove promenljivih itd.
Da bismo to postigli, moramo najpre otvoriti konzolu, to se radi uz pomoć $F12$. Konzola je, najverovatnije, trenutno prazna. Kako bismo ispisali neki tekst u nju koristimo neke od sledećih funkcija:

- console.log()
- console.warn()
- console.error()
- console.info()

[9] Metodi su, za sad, crne kutije koje nam omogućavaju da vršimo određene promene nad promenljivim. U nastavku će biti detaljnije obrađene.

```
var c = a + b; //12
console.log(c); //ovime ispisujemo 12 u konzolu
```

Operator typeof služi kako bi se ispisao tip promenljive.

```
var c = a + b; //12
console.log(typeof c); //ispis tipa promenljive c u konzolu - numeric
```

Možemo nadovezati i više argumenata pri ispisu u konzolu odvajajući ih zarezom. Korišćenjem zareza kreira se razmak između argumenata.

```
var c = a + b; //12
console.log("Vrednost zbira je", c);
```

Prazne vrednosti Pod prazne vrednosti ubrajamo *null* i *undefined*. One služe da označe odsustvo postojanja vrednosti. U daljem tekstu, susretaćemo se sa ovim vrednostima.

4.1.2 Naredbe za kontrolu toka

Osnovni elementi za opis izračunavanja u programima nazivaju se naredbe (već smo videli naredbu dodele). Naredbe za kontrolu toka omogućavaju različite načine izvršavanja programa, u zavisnosti od vrednosti promenljivih. Naredbe za kontrolu toka mogu biti:

- naredbe grananja i
- petlje.

Osnovni oblik naredbe koji se javlja je takozvana naredba izraza (ova vrsta naredbi obuhvata i naredbu dodele i naredbu poziva funkcije). Naime, svaki izraz završen karakterom ; je naredba. Na primer:

```
3 + 4*5;
n = 3;
c++; //uvecanje promenljive za 1, isto kao c = c + 1
f();
```

Nekada, želimo da više različitih naredbi grupišemo i da ih tretiramo kao jednu jedinstvenu naredbu. Vitičaste zagrade i se koriste da grupišu naredbe u složene naredbe, odnosno blokove, i takvi blokovi se mogu koristiti na svim mestima gde se mogu koristiti i pojedinačne naredbe.

Naredbe grananja (ili naredbe uslova), na osnovu vrednosti nekog izraza, određuju naredbu (ili grupu naredbi) koja će biti izvršena.

```
if (izraz)
    naredba1
else
    naredba2
```

Konkretno, na primeru kroz jezik JavaScript:

```
//primer ispisuje veci od dva broja
if (a > b)
    console.log(a);
else
    console.log(b);
```

Naredbe *naredba*1 i *naredba*2 su ili pojedinačne naredbe (kada se završavaju simbolom ;) ili blokovi naredbi zapisani između vitičastih zagrada (na kraju kojih ne ide ;). Deo naredbe *else* je opcioni, odnosno, ne mora postojati, pa se može napisati samo *if* grana. Izraz *izraz* predstavlja logički uslov:

```
if (5 > 7)
   a = 1;
else
   a = 2;


if (7)
    a = 1;
else
    a = 2;

a = 3;
if (a = 0)
    console.log("a je nula\n");
else
    console.log("a nije nula\n");
```

Često možemo imati višestruke odluke, za šta koristimo *else if* konstrukciju oblika:

```
if (izraz1)
    naredba1
else if (izraz2)
    naredba2
else if (izraz3)
    naredba3
else
    naredba4
```

Primer 4.1 *Primer else if naredbe.*

```
if (a > 20)
   console.log("A je vece od 20\n");
else if (a > 10)
   console.log("A je vece od 10\n");
else if (a < -20)
   console.log("A je manje od -20\n");
else if (a < -10)
   console.log("A je manje od -10\n");
else
   console.log("A pripada intervalu [-10, 10]\n");
```

Ternarni operator je uslovni operator oblika: *uslov* ? *ispunjen* : *inace* i ima isto značenje kao i if else.

Primer 4.2 *Zadatak je smestiti veći od dva broja u promenljivu x. Prvi if else ima isto značenje kao i ternarni operator koji smešta veći broj u x. Ako je a vece od b postavlja se x na a, inače postavlja se na b.*

```
if (a > b)
   x = a;
else
   x = b;
x = (a > b) ? a : b;
```

Osim *if-else* postoji i naredba *switch*, koja se takođe može koristiti za višestruko odlučivanje i ima opšti oblik:

```
switch (izraz) {
  case konstantan_izraz1:
        naredbe1;
        break;
  case konstantan_izraz2:
        naredbe2;
        break;
  ...
  default:
    naredbe_n;
    break;
}
```

Case predstavljaju slučajeve, koji ako su ispunjeni, izvršava se naredba desno od dvotačke. U slučaju da nijedan od case-ova nije ispunjen izvršava se default, ako default nije postavljen i nijedan od case-ova nije ispunjen kroz switch će se samo proći.

Petlje (ciklusi ili repetitivne naredbe) uzrokuju da se određena naredba (ili grupa naredbi) izvršava više puta (sve dok je neki logički uslov ispunjen).

```
while(izraz)
   naredba

while (i < j)
   i++;

while (1)
   i++;

for (izraz1; izraz2; izraz3)
   naredba

izraz1;
while (izraz2) {
   naredba
```

```
    izraz3;
}

for(i = 0; i < n; i++)
...
```

Postoji još jedan vid petlje, to je petlja do-while.

```
do {
naredbe
} while(izraz)
```

Telo (blok naredbi naredbe) naveden izmedu vitičastih zagrada se izvršava i onda se izračunava uslov (izraz *izraz*). Ako je on tačan, telo se izvršava ponovo i to se nastavlja sve dok izraz *izraz* nema vrednost nula (tj. sve dok njegova istinitosna vrednost ne postane netačno). Za razliku od petlje while, naredbe u bloku ove petlje se uvek izvršavaju barem jednom. Navođenje vitičastih zagrada pri korišćenju naredbi za kontrolu toka smatra se dobrom programerskom praksom.

U nekim situacijama pogodno je napustiti petlju ne zbog toga što nije ispunjen uslov petlje, već iz nekog drugog razloga. To je moguće postici naredbom break:

```
for(i = 1; i < n; i++) {
    if(i > 10)
       break;
    ...
}
```

Naredbom continue se prelazi na sledeću iteraciju u petlji. Na primer,

```
for(i = 0; i < n; i++) {
   if (i % 10 == 0)
      continue; /* preskoci brojeve deljive sa 10 */
   ...
}
```

Petlje i uslovne naredbe se mogu kombinovati, korišćenjem jednih u drugima i slično. U slučaju ugnježdenih petlji, naredbe break i continue imaju dejstvo samo na unutrašnju petlju.

Bulove vrednosti Pod Bulovim vrednostima smatramo *true* i *false*, odnosno tačno i netačno, 1 i 0. Ako bismo hteli da kreiramo beskonačnu petlju (iako se to nikako ne preporučuje) bio bi nam potreban uslov koji uvek važi:

```
while(true){
   //radi nesto zauvek
}
if(false)
    console.log("Ovo se nece nikad ispisati");
if(true)
    console.log("Ovo ce se uvek ispisati");
```

```
var a = 10;
var b = -10;
if((a != b) == true){
    console.log("uslov da su a i b razliciti vazi");
}
```

Ako bismo hteli da nadovežemo više uslova, ili da kažemo da želimo da se nešto izvrši ako je bilo koji od uslova ispunjen koristimo binarne operatore && i ||, odnosno "logičko i" i "logičko ili".

```
var a = 10;
if(a > 0 && a%2==0)
   console.log("a je paran broj veci od 0.);
var b = -10;
if(a > 0 || b > 0 )
   console.log("Ako je ili a ili b vece od 0 ovo ce se ispisati);
```

Napomena: uslova može biti više od 2.

Prozori Često se javlja potreba da korisnika stranice obavestimo o nečemu. Ono što nam JS omogućava je da na jednostavan način uz pomoć **alert** funkcije, korisniku jasno prikažemo poruku, koju ne može da ignoriše. Jedini argument ove funkcije je niska koja se prikazuje korisniku.

```
alert("Ovo je iskacuce obavestenje");
```

Nekada se od korisnika zahteva da na neko pitanje odgovori pozitivno ili negativno, za šta se koristi funkcija **confirm**. Argument ove funkcije je isti kao kod *alert*.

```
confirm("Da li ste sigurni?");
```

Ukoliko bi se od korisnika očekivao unos nekog teksta, za to može da se koristi funkcija **prompt**. Argumenti ove funkcije su niska koja se ispisuje u prozoru i podrazumevana vrednost polja za unos.

```
prompt("Unesite broj godina:");
```

Vežba Uraditi zadatke iz uvodne sekcije.

4.1.3 Niske

Niske ili stringovi, predstavljaju nizove karaktera. U promenljivu možemo smestiti pojedinačni karakter, a možemo i čitave reči ili rečenice.

```
var a = 'abc'; //a je niska
a = 'a'; // a je sada karakter
```

Niske se mogu navoditi između " (dvostrukih), ′ (jednostrukih), kao i ‘ iskošenih navodnika (šablon-literali, engl. template literals).

```
var niska1 = "Ovo je prva niska.";
var niska2 = "Ovo je druga niska";
var niska3 = `Ovo je niska
              neka treca`;
```

Primetimo da šablon literali poštuju novi red. Važno je naglasiti da se mora obratiti pažnja na to da navodnici budu pravilno upareni. Naime, program se neće izvršavati na predviđeni način ukoliko, na primer, nakon dvostrukog navodnika pokušamo nisku da zatvorimo jednostrukim navodnikom.

```
var niska1 = "Ovo je prva niska.";
var niska2 = "Ovo je druga niska";
var niska3 = "Ova niska nije pravilna';
```

Ukoliko želimo da unutar niske imamo prave navodnike (na primer kod citata), onda možemo ili upotrebiti drugi vid navodnika ili koristiti escape sekvencu \. Kako bismo nadovezali dve niske koristimo operator +, a ovaj postupak se naziva *konkatenacija*. Treba obratiti pažnju na to da nadovezivanjem niski sa +, ne dobijamo razmak između stringova kao kod korišćenja ,. Da bismo dobili razmak moramo ga dodati kao još jednu nisku, a kako bismo dobili novi red koristimo $\backslash n$.

```
console.log(niska1 + niska2);
niska1 = 'Ovo je neka niska.';
niska1 = 'Kaze Bob: "Alice ne kradi"';
console.log(niska1);
niska2 = 'Kaze Alice: \'Sta cu moram\' \nAlice nikad nije imala \nnameru
da ukradene podatke zloupotrebi.';
console.log(niska2);
niska2 = 'It\'s mine';
console.log(niska2);
niska2 = `Kaze Alice:
          'Necu vise krasti'
          Ukrala sam samo ${2 + 2} jabuke.`;
```

Ako bismo pokušali da na niske nadovežemo numeričke vrednosti, one bi se ponašale kao niske, osim ako bismo ih izdvojili zagradama.

```
console.log("Zbir 1 i 2 je: " + 1 + 2); //12
console.log("Zbir 1 i 2 je: " + (1 + 2)); //3
console.log(`Zbir 1 i 2 je: ${1 + 2}`);  //3
```

Primetimo da pri korišćenju šablon literala ono što želimo da se sračuna navodimo između ${}. Nekada želimo da nisku posmatramo kao broj, da bismo eksplicitno kastovali (izvršili pretvaranje, eksplicitnu konverziju) niske u broj, koristimo funkcije *parseInt*() i *parseFloat*(), koje prave ceo broj ili razlomljen broj redom. U slučaju da niska osim brojeva sadrži i neke druge karaktere, biće pokupljeni samo brojevi koji se nalaze na početku niske. Na primer, za nisku "123.292ajkula" parseInt će uzeti samo broj 123 (zaključno sa tačkom, jer je Int ceo broj), parseFloat će uzeti 123.292, a ostatak će biti ignorisan. Ako bi niska bila "a123.292" ništa ne bi bilo pokupljeno i funkcije bi vratile NaN ("Not a Number").

```
var p = "5";
console.log(typeof p); //string
console.log(typeof parseInt(p)); //number
```

Poređenje niski vrši se leksikografski uz pomoc operatora >,<,>= i <=, kao i uz pomoć operatora ==, != ili ===.

```
if("ana" >= "ane"){
   console.log("ana");
}
else{
   console.log("ane");
}
```

Metodom toString() vršimo eksplicitno konvertovanje u nisku.

```
var br_u_str = 102;
console.log("Tip promenljive je :", typeof br_u_str);
br_u_str = br_u_str.toString();
console.log(br_u_str);
console.log("Metod toString() vraca",typeof br_u_str);
```

Implicitna konverzija JavaScript je programski jezik takav da može da "progura"razne konstrukte, sa ciljem da program izvrši, zbog toga se često javljaju neki više ili manje očekivani rezultati. Implicitna konverzija je postupak pri kom program sam vrši konvertovanje nekog tipa u neki drugi tip kako bi dobio nešto što je njegovim internim mehanizmima smisleno. Prilikom primenjivanja konkatenacije nad brojem i niskom, broj će biti konvertovan u nisku.

```
//zakomentarisati sve linije pa redom oslobadjati
var conv = '5' + 3; //53
conv = 3 + '3'; //33
conv = 5 * null; //0
conv = '7' - 4; //3
conv = !""; //true
console.log(conv);
console.log(typeof conv);
```

Kako bismo proverili da li su dve vrednosti jednake koristimo "==", ali da bismo bili precizniji i proverili da li su vrednosti koje poredimo i istog tipa moramo koristiti "===". Važnost ovog koncepta se jasno vidi u narednom primeru:

```
console.log(10 == '10'); //true
console.log(10 === '10'); //false
```

Vrlo često pri programiranju želećemo da dobijemo dužinu neke niske i potom pristupimo nekom pojedinačnom karakteru. To se lako postiže korišćenjem svojstva *length* u vidu *niska.length*, a potom uz pomoć *charAt*() pristupamo pojedinačnim karakterima niske.

```
//Pristup pojedinacnim karakterima niske
var niska = "Programiranje";
console.log("Svojstvo length daje duzinu niske", niska.length);
//prolaskom kroz petlju obilazimo sve karaktere niske redom
for(i=0; i<niska.length;i++){
    console.log(niska.charAt(i));
}
//Obratiti paznju da brojanje niske krece od 0
console.log("5 karakter niske", niska.charAt(4));
```

Treba primetiti da brojanje karaktera u niski kreće od 0, pa tako

```
P R O G R A M I R A   N   J   E
0 1 2 3 4 5 6 7 8 9 10 11 12
```

Svi karakteri koje je moguće zapisati nalaze se u takozvanoj *ASCII* tabeli i imaju svoje kodne cifre, te cifre nam nekada mogu biti od koristi. Njima se pristupa korišćenjem *charCodeAt*(). Da bismo uradili obrnuto, tj. izvršili konverziju broja u karakter koristimo *fromCharCode*(). Ako bismo želeli da uklonimo beline sa krajeva (i sa početka i sa kraja niske), koristimo *trim*().

```
var voce = "  Ananas  "
console.log(voce.charCodeAt(2)); //A --> 65
console.log(String.fromCharCode(97)); //a
voce = voce.trim();
console.log(voce);
```

Da bismo karaktere neke niske smanjili ili uvećali, koristimo metode *toUpperCase*() ili *toLowerCase*() u zavisnosti od toga šta želimo postići. Primetimo da se mogu primeniti i nad pojedinačnim karakterom.

```
console.log(voce.toLowerCase()); //ananas
console.log(voce.toUpperCase()); //ANANAS
```

Dodatne metode za rad sa niskama Metode nam omogućavaju da izvršimo neke akcije nad objektima. Postoji puno metoda za rad sa niskama, a u ovom delu biće navedene samo neke od njih.

U prethodnom delu već su predstavljene *charAt*() i *charCodeAt*(). Prva nam daje (kaže se i vraća) karakter na poziciji koju pošaljemo kao argument, a druga nam vraća kodnu vrednost karaktera, koji se nalazi na poziciji prosleđenoj kao argument. Osim njih neki od istaknutih metoda su:

- **concat()** - nadovezuje dve niske i vraća novu nisku ne menjajući stare,
- **includes()** - proverava da li niska sadrži određen karakter ili reč, kao drugi argument se šalje od koje pozicije početi sa traženjem,
- **indexOf()** - vraća indeks prvog pojavljivanja niske koju šaljemo kao argument funkcije ili -1 ako ta niska nije pronađena (napomena: lastIndexOf() radi isto samo vraća poslednji),
- **repeat()** - vraća string ponovljen onoliko puta koliko je navedeno u zagradi,
- **replace()** - vraća novi string u kome je reč navedena kao prvi argument, zamenjena rečju navedenom kao drugi argument,
- **search()** - pretražuje string za rečju koja je navedena kao argument i vraća poziciju od koje reč počinje u niski ili vraća -1 ako reč nije pronađena,
- **slice()** - iz niske izdvaja deo niske počevši od indeksa naznačenog kao prvi argument, do indeksa naznačenog kao drugi,
- **substr()** - iz niske izdvaja podnisku počevši od indeksa navedenog kao prvi argument u dužini navedenoj kao drugi argument.

Navedimo sada neke od primera primena ovih metoda.

```
var niska1 = "Danas je.";
var niska2 = "lep dan.";
var niska3 = niska1.concat(niska2);
console.log(niska3);
//hocemo da provermo da li niska3 sadrzi rec dan
console.log("Da li niska sadrzi rec dan", niska3.includes("dan"));
console.log("Da li niska sadrzi rec ana", niska3.includes("ana"));
console.log("Indeks prvog pojavljivanja reci dan", niska3.indexOf("dan"));
console.log("Indeks prvog pojavljivanja reci ana", niska3.indexOf("ana"));
console.log(niska1.repeat(4));
var niska4 = niska2.replace("dan", "covek");
console.log(niska4);
console.log("Na kom indeksu pocinje rec dan", niska3.search("dan"));
console.log("Na kom indeksu pocinje rec ana", niska3.search("ana"));
console.log(niska3.slice(1,4));
console.log(niska3);
console.log(niska3.substr(1,4));
```

Vežba Uraditi zadatke iz sekcije vezane za niske.

4.1.4 Nizovi

Nizovi predstavljaju kolekcije elemenata nekog tipa. U JavaScriptu, za razliku od većine drugih programskih jezika, dozvoljeno je da ti elementi budu različitih tipova. Novi niz kreiramo korišćenjem uglastih zagrada. Ako navedemo samo uglaste zagrade, onda se kreira prazan niz, odnosno, niz koji ne sadrži elemente. Elemente u nizu navodimo pisanjem između uglastih zagrada i odvajamo ih zarezima. Indeksiranje nizova počinje od 0. Indeksiranje nam je veoma važno kako bismo mogli da lako pristupimo vrednosti koju čuva neki element niza. Elementima niza pristupamo navođenjem naziva niza, a potom u uglastim zagradama indeksa kojem želimo da pristupimo. Opisani postupci predstavljeni su kroz primer u nastavku:

```
var niz = [];
var niz2 = [1,2,3,4,5];
var niz3 = ["niska1", "niska2"];
var niz4 = [1, "neki tekst", "!", 3.14];
niz4["kljuc"] = 5;
console.log(niz2[3]);
console.log(niz3[0]);
console.log(niz4[3]);
console.log(niz4);
```

Element niza može biti i neki drugi niz. Kao što se vidi u primeru iznad, osim što nizovi mogu čuvati različite tipove (niske, brojeve, razlomljene brojeve, karaktere), moguće je indeksiranje elemenata niza različitim tipovima. Ukoliko indeksi elemenata nisu eksplicitno navedeni, elementi niza su podrazumevano indeksirani celim brojevima počevši od 0. Kao posledica činjenice da nizovi

mogu čuvati elemente različitih tipova, za element niza možemo postaviti i drugi niz, čime je moguće kreiranje matrica, pa tako:

```
//elementi niza mogu biti drugi nizovi
var niz1 = [niz2,0,1,4];
console.log("Niz 1:",niz1);
//kreiranje matrice
var matrica = [];
matrica[0] = [1,2];
matrica[1] = [3,4];
console.log(matrica[0][1]);
```

Da bismo obišli celu matricu, neophodno je da prođemo kroz sve njene elemente. Odnosno, da krećući se kroz elemente niza redom obiđemo svaki od elemenata. Kako bismo to učinili na najjednostavniji način, neophodno je da koristimo dvostruku *for* petlju. Spoljašnja petlja bi se kretala po vrstama matrice, a unutrašnja po kolonama. Veličinu niza, kao i kod niski, uzimamo korišćenjem svojstva *length*.

```
console.log(matrica.length);
for(i = 0;i<matrica.length;i++){
    for(j = 0; j<matrica.length;j++){
        console.log("element u vrsti",i,"i koloni",j,"je:",matrica[i][j]);
    }
}
```

Treba obratiti pažnju da u primeru gore podrazumevamo da je matrica kvadratna (sa istim brojem vrsta i kolona). Kako bismo obišli matricu koja ima različit broj vrsta i kolona treba da obratimo pažnju na veličinu vrste i veličinu kolone. Čitaocu se ostavlja radi vežbe i boljeg razumevanja funkcionisanja matrica i nizova da isproba obilaske matrice za različit broj vrsta i kolona.

Da bismo nekom elementu niza promenili vrednost dovoljno je da ga nađemo po indeksu, a potom da naredbom dodele navedemo novu vrednost. Važno je istaći da ta nova vrednost ne mora biti istog tipa.

```
var brojevi = [1,2,3,4];
brojevi[3] = 6;
console.log(brojevi); //1,2,3,6
brojevi[3] = "Sest";
console.log(brojevi); //1,2,3,"Sest"
```

Elemente niza možemo postaviti eksplicitno bez obzira na to da li neke indekse izostavljamo, pa je tako moguće postaviti vrednosti *brojevi*[5] i *brojevi*[7], a potom ispisati sve elemente niza. Za one indekse koje smo preskočili *console.log* će ispisati *empty*.

```
brojevi[5] = 15;
brojevi[7] = 99;
console.log(brojevi); //[1,2,3,"Sest",empty,15,empty,99]
```

Nizovi i petlje se dosta dobro kombinuju, u smislu da sa petljama lako obilazimo elemente niza, kao što je i viđeno u prethodnim primerima. Kod *for*

petlje primetili smo da promenljiva i dobija vrednost indeksa svakog od elemenata niza. Osim standardne for petlje, u $JavaScript - u$ moguće je koristiti i poseban vid for petlje, tzv. $for - in$ petlju. Ova petlja funkcioniše na isti način kao standardna for petlja prolazeći kroz elemente redom, ali je njen zapis dosta kraći. Osim toga, $for - in$ petlja će proći samo kroz elemente koji postoje (preskočiće indekse u nizu koji ne čuvaju nikakvu vrednost!). Ovakvo ponašanje se najbolje oslikava kroz primer.

```
for(i=0;i < brojevi.length; i++){
        console.log(brojevi[i]);
}
for(i in brojevi){
        console.log(brojevi[i]);
}
```

Ispisom kroz konzolu, lako se vidi da je prva petlja dala 1, 2, 3, "Sest", *undefined*, 15, *undefined*, 99, a druga for petlja 1, 2, 3, "Sest", 15, 99.

Neke metode za rad sa nizovima Pomenimo, pre svega, još jedan bitan metod za rad sa niskama. Metodom *split*() od postojeće niske kreiramo niz. Argument koji prosleđujemo predstavlja po kom kriterijumu da se izvrši podela niske. Da bismo uradili obratno, odnosno, od niza kreirali nisku, koristimo metod *join*().

```
//Metod za rad sa niskama koji od niske kreira niz: split
var niska = "Danas je lep dan.";
var niz = niska.split(" ");
console.log(typeof niz);
console.log(niz);

//Metod koji od niza formira nisku: join
niz = ["danas", "je", "tmuran", "dan"];
niska = niz.join();
console.log(typeof niska);
console.log(niska);
```

Pomenimo još neke istaknute metode za rad sa nizovima:

- **concat()** - nadovezuje dva niza i vraća novi ne menjajući stare,
- **includes()** - proverava da li niz sadrži određeni element, kao argument se šalje ono što se traži, ako postoji vraća *true*, inače, vraća $false$,
- **push()** - dodaje novi element na kraj niza,
- **pop()** - skida poslednji element sa kraja niza,
- **reverse()** - vraća obrnut niz,
- **slice()** - iz niza izdvaja deo počevši od indeksa naznačenog kao prvi argument, do indeksa naznačenog kao drugi i vraća novi niz,
- **sort()** - sortira niz.

```
var niz_brojeva = [1,2,3,4];
var novi = niz_brojeva.concat(niz);
console.log(novi);

console.log("Da li niz sadrzi broj 13?", niz.includes(13)); //false
niz.push(13);
console.log("Dodat je novi element na kraj niza",niz);
niz.pop()
console.log("Niz kada skinemo poslednji element",niz);

niz.reverse();
console.log("Obrnut niz",niz);

niz = niz.slice(1,3);
console.log(niz);
niz_brojeva.reverse();
console.log(niz_brojeva);
niz_brojeva.sort();
console.log(niz_brojeva);
```

Vežba Uraditi sve zadatke iz sekcije Nizovi.

4.1.5 Svojstva i objekti

Ako bismo u naš niz dodali još jedan element:

```
niz[4] = {svojstvo: 5 };
console.log(niz[4]);
```

Primećujemo da smo u niz dodali nešto novo oblika *naziv_svojstva* : *vrednost*. Takav konstrukt nazivamo objektom. Objekti su strukture podataka koje čine nizovi svojstava i njihovih vrednosti. Većina JavaScript vrednosti imaju neka svojstva, već smo se susreli sa svojstvom *length* za nizove. U primeru navedenom gore imamo samo jedno svojstvo *svojstvo*. Pogledajmo nešto intuitivniji primer. U primeru kreiramo jednog studenta sa svim svojstvima koja su nam za jednog studenta interesantna:

```
var student = { ime : "Una",
prezime : "Stankovic",
datum_rodjenja : "13.10.1994.",
jmbg: 1310994715004,
broj_indeksa: "1095/2016",
nivo_studija: "m",
prosek: 9.75,
ocene: [8,9,10]};
console.log(student);
```

Ispisom objekta student u konzoli, možemo videti svojstva i sve vrednosti svojstava. Primetimo da vrednosti svojstava opet mogu biti različite vrednosti. Zanima nas sad kako možemo pristupiti vrednostima svojstava i to je moguće učiniti na dva načina:

- "tačka notacijom "- *nazivobjekta.svojstvo*, na primer *student.ime*
- "po indeksu" - *nazivobjekta*['*svojstvo*'], na primer *student*['*ime*']

```
console.log(student);
console.log(student.ime);
console.log(student["ime"]);
```

Napomena: u slučaju da je svojstvo broj ili da sadrži razmake u nazivu ne može se koristiti tačka notacija.

Ako pokušamo da pristupimo vrednosti svojstva koje ne postoji dobijamo *undefined*.

```
console.log(student.drzavljanstvo);
```

Svojstvu dodeljujemo vrednost uz pomoć naredbe dodele (operatora =). Ako bismo počeli od praznog objekta kako bismo mu dodali nova svojstva možemo koristiti samo tačka notaciju. JavaScript sam prepoznaje da ta svojstva ne postoje i dodaje ih.

```
var racunar = {};
console.log(racunar);
console.log(typeof racunar);
racunar.licni_racunar = "da";
racunar.os = 'windows';
racunar.marka = 'lenovo';
console.log(racunar);
```

Za uklanjanje svojstva iz objekta koristimo unarni operator *delete*. Ako pokušamo da pristupimo vrednosti svojstva koje smo obrisali dobijamo *undefined*.

```
delete racunar.licni_racunar;
console.log(racunar.licni_racunar);
console.log(racunar);
```

Binarnim operatorom *in* lako se proverava da li neki objekat sadrži neko svojstvo. U zavisnosti od toga da li svojstvo postoji ili ne vraćaju se *true* ili *false*.

```
console.log("licni_racunar" in racunar); //false
console.log("os" in racunar); //true
```

Koncept **refereciranja objekata** je veoma važan. Naime, kod objekata moramo razlikovati da li imamo dve različite reference ka istom objektu ili dva objekta imaju ista svojstva. U prvom slučaju, kada imamo dve različite reference ka istom objektu, dve različite promenljive pokazuju na isti memorijski prostor, a svaka promena izvedena nad objektom preko jedne promenljive propagiraće se i na drugu. U drugom slučaju, kada dva objekta imaju ista svojstva, ali su ti objekti različiti, promene izvedene nad jednom promenljivom neće imati nikakvog uticaja na drugu. Kako bismo ovo bolje razumeli, najbolje da pogledamo naredni primer:

```
var student1 = {ime : "Janko",
        prezime: "Jovanovic",
        prosek: 9.5};
```

```
var student2 = student1;
var student3 = {ime : "Jovana",
        prezime : "Markovic",
        prosek : 8.5}
console.log(student1 === student2); //true
console.log(student2 === student3); //false
console.log(student3 === student2); //false
```

Vidimo da pri poređenju objekata dobijamo da su objekti *student*1 i *student*2 isti. Sada pogledajmo šta se dešava kada pokušamo da izvršimo neku promenu nad *student*1.

```
student1.ime = "Ana";
console.log(student2.ime); //Ana
console.log(student3.ime); //Jovana
```

Obratiti pažnju da je u prvom *console.log*-u *student*2.

4.1.6 Funkcije

Funkcija je jedna od osnovnih konstrukcija jezika koja nam omogućava ponovno korišćenje koda. Omogućavaju nam, čak, i da ih koristimo iako ne razumemo detalje implementacije. Štaviše, već smo koristili neke od ugrađenih funkcija, kao što su prompt, alert, parseInt, itd.. Izdvajanjem koda u funkcije povećava se čitljivost koda i olakšava se njegovo održavanje. Osim toga, olakšava se ponovna upotrebljivost koda (engl. reusability). Naime, vrlo često će nam biti potrebno da ponovo iskoristimo neki kod, neku funkciju koja nam se javlja u više programa. Upravo zbog toga, u *JavaScript*-u , kao i u mnogim drugim programskim jezicima, postoje ugrađene funkcije, kao i dodatne biblioteke sa funkcijama koje je moguće koristiti kako bi se olakšao rad. Moguće je čak i pisanje sopstvenih biblioteka.
Osnovno uputstvo za pisanje funkcija se sastoji iz nekoliko bazičnih koraka:

- Uočiti logičke celine
- Uočiti neophodne parametre i povratnu vrednost
- Dati ime koje odgovara implementaciji
- Implementirati funkciju

Funkcija se odlikuje svojom deklaracijom i definicijom. Deklaracija (ili prototip) funkcije ima opšti oblik:
$function\ ime\_funkcije(niz\_deklaracija\_parametara);$

Definicija funkcije ima oblik:

```
function ime_funkcije(niz_deklaracija_parametara) {
   deklaracije
   naredbe
}
```

```
function sabiranje(a,b){

}
```

Parametri i argumenti funkcije Funkcija može imati parametre koje obrađuje i oni se navode u okviru definicije, iza imena funkcije i izmedu zagrada. Termini parametar funkcije i argument funkcije se ponekad koriste kao sinonimi. Objasnimo kroz primer razliku između ovih pojmova. Naime, n je parametar funkcije $kvadrat(n)$;, a 5 i 9 su njeni argumenti u pozivima $kvadrat(5)$ i $kvadrat(9)$.

Parametri funkcije mogu se u telu funkcije koristiti kao lokalne promenljive[10] te funkcije, a koje imaju početnu vrednost određenu vrednostima argumenata u pozivu funkcije.

```
function sabiranje(a,b){
        var zbir = a + b;
}

var a = 5;
var b = 7;
sabiranje(5,7);
```

Povratna vrednost funkcije **Povratna vrednost** funkcije predstavlja glavni rezultat rada te funkcije.Funkcija rezultat vraća naredbom *return r*; gde je r izraz ili promenljiva. Nakon završetka svih naredbi unutar funkcije ili dolaskom do naredbe *return* izvršavanje se nastavlja od mesta odakle je funkcija pozvana. Naredba *return r*; ne samo da vraća vrednost r kao rezultat rada funkcije, nego i prekida njeno izvršavanje. Funkcija može vraćati i rezultat izračunavanja nekog izraza, kao funkcija *sabiranje*2 u primeru dole. Vrednost koju funkcija vrati možemo smestiti u neku promenljivu kako bismo mogli sa njom na dalje da radimo. Funkcije mogu pozivati druge funkcije, a funkcija može pozivati i samu sebe. Više o slučaju kada funkcija poziva samu sebe biće u sekciji o rekurzivnim funkcijama.

```
function sabiranje(a,b){
        var zbir = a + b;
        return zbir;
}

function sabiranje2(a,b){
        return a + b;
}

var a = 5;
var b = 7;
sabiranje(5,7);
var c = sabiranje2(a,b);
```

Kao što vidimo kada pozivamo funkciju koristimo naziv funkcije, a potom navodimo listu argumenata. Funkcija može primati nula ili više argumenata. U

[10] Lokalne promenljive su vidljive samo unutar tela te funkcije.

slučaju, da funkcija nema argumente pozivamo je kao *ime_funkcije*(). Funkcija bez argumenata će se izvršiti i vratiti povratnu vrednost. Ako funkcija ima jedan ili više argumenata, njih navodimo između zagrada, odvojene zarezima. Bitno je naglasiti da nazivi parametara pri definisanju funkcije i argumenata pri njenom pozivanju ne moraju biti isti, ali promenljive korišćene u telu funkcije i nazivi parametara moraju biti isti! To je posledica toga što promenljive navedene u definiciji dobijaju vrednosti argumenata, koje se kasnije koriste u telu funkcije, one su poput najave da će se na tom mestu očekivati neke vrednosti koje kasnije treba koristiti na tačno određen način (što je definisano u telu funkcije).

```
function sabiranje(a,b){
        var zbir = a + b;
        return zbir;
}

function sabiranje2(a,b){
        return a + b;
}
//funkcije bez argumenata
function pozdrav(){
        console.log("Hello World!");
        return;
}

function dodatak(){
        var broj = Math.random()*100;
        return broj;
}

var a = 5;
var b = 7;
sabiranje(5,7); //nema ispisa nikakvog
var c = sabiranje2(a,b);
console.log(c);
pozdrav(); //"Hello World!"
console.log(dodatak());
```

Primetimo da iako funkcija ne vraća nikakvu konkretnu vrednost na kraju funkcije se nalazi *return*;. Napomena: Funkcija *Math.random*() korišćena u primeru je funkcija iz biblioteke *Math* i vraća nasumični broj između 0 i 1.

Osim navedenog načina moguće je funkciju dodeliti promenljivoj. Obratiti pažnju da nakon poslednje vitičaste zagrade mora stajati tačka-zarez.

```
var stepenovanje = function(osnova, eksponent){
                var rez = 1;
        for(i = 0; i < eksponent; i++){
                rez *= osnova;
        }
```

```
        return rez;
        };
console.log(stepenovanje(2,3));
```

Vežba Uraditi sve zadatke iz uvodne sekcije koristeći funkcije. Uraditi zadatke iz sekcije funkcije.

Anonimne funkcije Anonimne funkcije su funkcije koje nemaju naziv. Anonimne funkcije koristimo kada želimo da se neka funkcija izvrši jednom. Vrednost anonimne funkcije ne možemo koristiti ukoliko je ne smestimo u neku promenljivu ili nije deo neke druge funkcije o čemu će biti više reči kasnije. Definisanjem funkcije dodelom promenljivoj kreiramo anonimnu funkciju sa desne strane naredbe dodele. Pogledajmo još jedan primer:

```
var anonimna = function(){
        console.log("Dosli ste do anonimne funkcije.");
        return;
        };
anonimna();
```

Broj argumenata funkcije Lako se uočava da pri pozivu funkcije možemo funkciji poslati više argumenata nego što ona zahteva. U *JavaScriptu* je to dozvoljeno ponašanje, iako u većini drugih programskih jezika nije. Ako bismo poslali veći broj argumenata nego što funkcija zahteva, ona bi višak argumenata ignorisala. U slučaju da je poslato manje argumenata nego što funkcija zahteva, za oni parametri za koje argumenti nisu poslati postavljaju se na *undefined*. Zaključak, *JavaScript* ne proverava ni broj ni tip poslatih argumenata.

```
var a = 5;
var b = 7;
//fija: sabiranje2(a,b);
console.log(sabiranje2(a,b,4,2,1));
```

Pogledajmo još jedan primer. Funkcija *argumenti*() prima 2 argumenta, posmatrajmo šta se dešava u različitim slučajevima:

```
function argumenti(prvi, drugi){
        console.log(prvi);
        console.log(drugi);
    return;
}
argumenti("Prvi","Drugi"); //prvi drugi
argumenti("Prvi"); //prvi undefined
argumenti(); //undefined undefined
argumenti("Prvi","Drugi","Treci"); //prvi drugi
```

JavaScript funkcije imaju ugrađeni objekat *arguments*. Ovaj objekat možemo koristiti da na jednostavan način proverimo koji je broj argumenata poslat.

```
function prosek(){
        var suma = 0;
```

```
        console.log("Broj argumenataje:",arguments.length);
        for(i=0;i<arguments.length;i++){
    suma += arguments[i];
        console.log(arguments[i]);
    }
        var prosek = suma / arguments.length;
    console.log(suma);
        return prosek;
    }
        var p = prosek(9,10,10,6,8);
    console.log("Prosek ocena je:",p);
```

Na ovaj način možemo pristupiti argumentima koje bismo inače ignorisali. U jeziku *JavaScript* argumenti se prenose po vrednosti. To znači da se njihova originalna vrednost ne menja. Sa druge strane, objekti se prenose po referenci, što znači da se njihova originalna vrednost menja.

Nekada nam nije odmah poznato koliko parametara će funkcija imati. *JavaScript* nam omogućava da funkcije mogu imati i proizvoljan broj parametara. Da bismo napisali ovakvu funkciju koristimo sledeću notaciju:

```
function nova(...parametri){
        for(parametar of parametri){
                //uradi nesto sa parametrima
        }
}
```

Pogledajmo i na primeru:

```
function lista_za_kupovinu(...parametri){
        for(parametar of parametri){
        console.log(parametar);
    }
}
console.log("Lista za kupovinu");
lista_za_kupovinu("mleko","hleb","lazanje","torta");
```

Primetimo da smo koristili $for - of$ petlju za kretanje kroz listu parametara. Ova petlja nam služi za kretanje kroz nizove, karaktere niski, mape, skupove, itd. Napomena: ranije smo videli $for - in$ petlju, ta petlja se najčešće koristi za iteraciju kroz svojstva objekata ili indekse.

```
var ucenik = { ime : "Ana",
        indeks: "1083/2016",
    prosek: 9.41};
for(kljuc in ucenik){
        console.log(`${kljuc} --> ${ucenik[kljuc]}`);
}

var str = "Neki tekst";
for(indeks in str){
        console.log(`Indeks ${str[indeks]}:${indeks}`);
}
```

Periodično izvršavanje funkcija Nekada funkcije želimo da izvršavamo periodično, odnosno, želimo da se kod unutar funkcije izvršava nakon određenog vremenskog intervala. Za ovakvo izvršavanje funkcija koristimo funkcije *setTimeout*() i *setInterval*().

Funkcija *setTimeout*() izvršava naredbe nakon vremenskog intervala zadatog u milisekundama (1000ms = 1s). Prvi argument je funkcija koja se izvršava nakon broja milisekundi navedenih kao drugi argument funkcije. Funkcija koja se prosleđuje kao argument funkcije *setTimeout*() može biti i anonimna funkcija. U tom slučaju nju direktno pišemo u okviru zagrada funkcije.

```
function objava(){
        alert("Obavesteni ste!");
}
setTimeout(objava,4000);
//anonimna fija je prvi argument
setTimeout(function(){
        alert("Obavesteni ste iz anonimne funkcije!");
},5000);
```

Funkciju *setTimeout*() možemo koristiti i za periodično izvršavanje funkcije tako što ćemo je pozvati unutar neke funkcije $f()$ koju želimo da izvršavamo više puta, a kao prvi argument šaljemo naziv te funkcije iz koje se poziva. Napomena: Neophodno je posebno pozvati funkciju $f()$ prvi put.

```
 //Funkciju setTimeout mozemo koristiti i za periodicno izvrsavanje
function objava2(){
        alert("Obavestenje u intervalu!");
    setTimeout(objava2,1000);
        }
objava2();
```

Povratna vrednost funkcije *setTimeout*() je identifikator tajmera koji je pokrenut, za prekidanje tajmera koristimo funkciju *clearTimeout*().

```
var tajmer = setTimeout(function(){
        alert("Ovaj tekst se nece ispisati");
        }, 2000);
clearTimeout(tajmer);
```

Osim funkcije *setTimeout*() za periodično izvršavanje funkcija možemo koristiti funkciju *setInterval*(). Ova funkcija prima iste argumente kao i *setTimeout*(), samo što u njenom slučaju broj milisekundi predstavlja broj nakon kog će se ponovo izvršiti funkcija navedena kao prvi argument.

```
setInterval(function(){
        alert("Ova funkcija ce se izvrsavati svake sekunde.")
        }, 1000);
```

Kao i kod *setTimeout*() povratna vrednost funkcije *setInterval*() je tajmer. Kako bismo prekinuli tajmer moramo pozvati funkciju *clearInterval*().

```
var tajmer = setInterval(function(){
        alert("Ovo ce se izvrsiti samo jednom");
```

```
},1000);
//Nakon 1.5 sekunde ce se pozvati prekidanje setInterval()
setTimeout(function(){
        clearInterval(tajmer);
},1500);
```

4.1.7 Objekat Math

Primetili smo da smo pri pisanju funkcija za rad sa brojevima pisali neke funkcije koje bi trebalo da se dosta često koriste pri matematičkim izračunavanjima. Osim toga, pominjali smo i da se funkcije koriste kako bi nam omogućile da neke postupke koji su nam često potrebni napišemo samo jednom, a potom upotrebljavamo kad god nam je to neophodno. *JavaScript* u okviru objekta **Math** nudi definisane matematičke funkcije i konstante. Neke od njih su:

- **Math.random()** - generiše pseudoslučajne brojeve iz intervala 0,1,
- **Math.round(x)** - zaokružuje broj x na najbliži ceo broj,
- **Math.sqrt(x)** - računa koren broja x,
- **Math.pow(x,y)** - računa y-ti stepen broja x,
- **Math.min(x,y,z,...)** - računa minimum brojeva navedenih kao argumente,
- **Math.max(x,y,z,...)** - računa maksimum brojeva navedenih kao argumente i
- **Math.PI** - konstanta π.

```
var a = 9;
var b = 3;
console.log("Koren ",a,"je", Math.sqrt(a));
console.log(a,"na",b,'je',Math.pow(a,b));
console.log("Minimum brojeva",a,",",b,",
        120 i 2 je",Math.min(a,b,120,2));
console.log("Maksimum brojeva",a,",",b,",
        120 i 2 je",Math.min(a,b,120,2));
console.log("Broj ",a,"puta konstanta pi je",
        a*Math.PI,"\nKada se dobijeni broj podeli sa 14 i
        zaokruzi dobija se:",Math.round((Math.PI*a)/14));
```

Vežba Modifikovati prethodno urađene primere tako da se koriste funkcije objekta *Math*.

4.1.8 Doseg promenljivih

Doseg (engl. scope) važenja promenljive označava u kojim sve delovima koda je neku promenljivu moguće "videti", odnosno, pristupati joj, menjati je i koristiti. Dobro razumevanje i poznavanje dosega važenja promenljive nam je veoma bitno pri programiranju. Nekada se može desiti da pokušavamo da

pristupimo promenljivoj u nekom delu koda u kome ona nije vidljiva ili da, još gore, nesvesno promenimo vrednost neke promenljive, što kasnije može imati posledice po izvršavanje ostatka programa.

Mesto gde uvodimo promenljivu utiče na njen doseg. Promenljive kreirane van svih blokova koda i funkcija imaju **globalni doseg**. Promenljive koje predstavljaju argumente funkcije, kao i one koje kreiramo unutar funkcija imaju doseg te funkcije u kojoj se nalaze. Do sada smo promenljive kreirali koristeći isključivo ključnu reč *var*. Postoje još dve ključne reči kojima kreiramo promenljive to su *let* i *const*. Promenljive kreirane ključnom rečju *let* i *const* imaju **lokalni doseg**. To znači da su one vidljive samo unutar bloka u kom su deklarisane. Promenljive uvedene ključnom rečju *var* za funkciju važe na nivou cele funkcije. Pogledajmo naredni primer:

```
function vidljivost(x){
        console.log(x);
    var x = 5;
        console.log(x);
    if(x == 5){
        //promenljive mozemo kreirati koristeci let
        //let promenljive imaju lokalni doseg
        let n = 7;
        console.log("n iz if-a:",n);
        }
    else{
        console.log("n iz else-a:",n);
    }
    return x;
}
var x = 10;
vidljivost(x);
console.log(x);
```

Iz primera se vidi da je promenljiva x globalna. U funkciji $vidljivost(x)$ najpre ispisujemo vrednost promenljive x, i dobijamo 10, što je ono što smo i prosledili. Potom unutar funkcije menjamo vrednost promenljive x na 5. Nakon toga kreiramo promenljivu n, koja nije vidljiva u *else* grani (probati ovo tako što ćete zakomentarisati liniju $varx = 5;$. Nakon poziva funkcije ponovo ispisujemo vrednost promenljive x i vidimo da ništa nije promenjeno. Posmatrajmo još jedan primer:

```
function zbir(){
        return c+d;
}

var c = 7; //zakomentarisati
var d = 8; //zakomentarisati
console.log(zbir());
var a = 5;
var b = 7;
console.log(zbir(a,b));
```

Primetimo da funkcija uvek radi za promenljive c i d zato što su globalno definisane, pa funkcija može da nađe njihove vrednosti. U slučaju da su c i d nedefinisani izbacuje se greška.

Obratite pažnju i na to šta se dešava sa *let*, kada pokušamo da mu promenimo vrednost. Zameniti *let* sa *var*.

```
function vidljivost_let(){
        let x = 5;
        if(true){
                let x = 7;
                console.log(x); //7
        }
        console.log(x); //5
}
vidljivost_let();
```

Osim promenljivih, funkcije isto mogu biti deklarisane globalno i lokalno. Globalno deklarisane funkcije su vidljive i mogu se pozivati na nivou celog programa, a lokalno deklarisane funkcije samo unutar one funkcije (i njenih drugih podfunkcija) u kojoj je deklarisana. **Izvlačenje deklaracija** (engl. hoisting) je pojam koji se odnosi na to da *JavaScript* izvlači deklaracije funkcija i promenljivih tokom faze kompajliranja što nam omogućava da vršimo pozive funkcija ili promenljivih i pre njihovog definisanja (obratiti pažnju da se izvlače deklaracije, ali ne i definicije!). Navedeni koncepti se oslikavaju kroz sledeće primere. Najpre, vidljivost funkcija:

```
function globalna(){
        console.log("Ova funkcija je vidljiva i moze se
        pozvati na nivou celog programa.");
}
function globalna_2(){
        lokalna(1); //poziv pre definicije
        function lokalna(n){
                console.log("Poziv",n, "iz lokalne:");
                globalna();
        }
        function lokalna2(){
                console.log("Poziv iz druge lokalne:");
                lokalna(2);
        }
    lokalna(3); //poziv nakon definicije
    lokalna2();
}

globalna();
globalna_2();
lokalna(4);
lokalna2();
```

Potom, izvlačenje deklaracija:

```
ispisi(s); //s is not declared
ispisi(parametar); //undefined
var parametar;
function ispisi(p){
        console.log("Vrednost parametra je:",p);
    return;
}
parametar = 10;
ispisi(parametar);
```

4.1.9 Rekurzivne funkcije

Programski jezik *JavaScript* podržava rekurzivne funkcije. Rekurzivne funkcije su one funkcije koje pozivaju same sebe. Uvek mora postojati neki uslov izlaska iz rekurzije. Najčešći primer koji se posmatra pri učenju ovih funkcija je *Fibonačijev niz*.

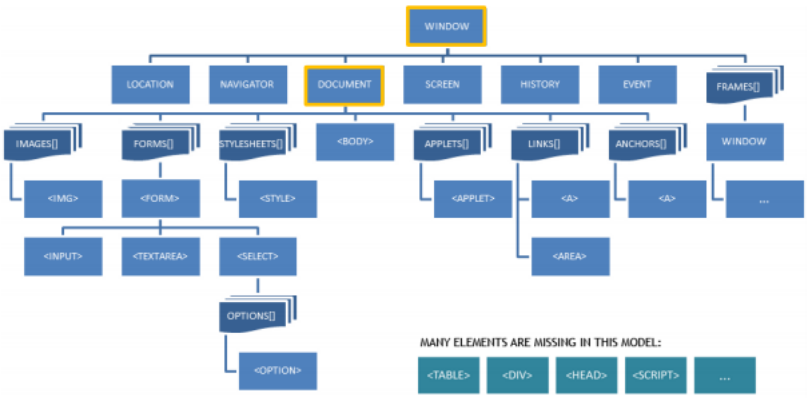
```
function fib(n){
        if(n == 0 || n == 1){
                return 1;
        }
        return fib(n-1) + fib(n-2);
}

var a = 5;
console.log("a-ti Fibonacijev broj", fib(a));
```

Da bismo dobili naredni element potrebna su nam prethodna 2. Zbog toga, pozivamo funkciju za prethodna 2 broja. Uslov zaustavljanja je kada dođemo do 1 ili do 0.

4.2 DOM

Rečeno je da je *JavaScript* služi za davanje interaktivnosti stranici, međutim, do sad smo sve ispisivali u konzoli sa malo interakcije sa stranicom. Da bismo mogli da dodamo interaktivnost sa postojećim elementima stranice (ili da kreiramo nove) moramo najpre naučiti kako da tim elementima pristupimo. Za svaku web stranicu pregledač formira nešto što nazivamo **DOM**[11]. DOM je drvolika struktura čiji su elementi *HTML* elementi (tagovi) i njihovi sadržaji(unutar tagova). *DOM* čuva hijerarhijsku strukturu u smislu da je neki element, koji je u *HTML*-u bio roditeljski element nekog drugog elementa, i u *DOM*-u roditeljski element. Elementi koji su hijerarhijski "ispod" nekog elementa nazivamo decom ili, ako je u pitanju jedan element, detetom (engl. child). Koren *DOM* stabla je objekat *window* koji predstavlja prozor pregledača. Elementi napisani u *HTML*-u odgovaraju čvorovima podstabla čiji je koren objekat *document*, jedan od sinova objekta *window*. Pogledajmo sliku:



Slika 18: Prikaz DOM stabla počevši od window objekta.

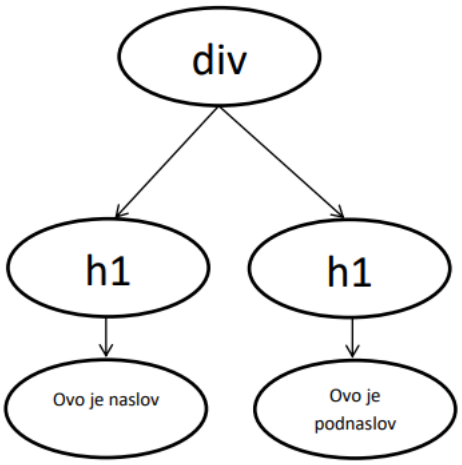
Vežba Izabrati sajt i kreirati strukturu *DOM* stabla. Kao na primeru:

```
<div>
 <h1>Ovo je naslov</h1>
 <h2>Ovo je podnaslov</h2>
</div>
```

4.2.1 Osnove

Kao što je ranije rečeno *JavaScript* je skript jezik čija je osnovna uloga na Veb-u programiranje korisničkog interfejsa. Korišćenje jezika Javascript omogućava interakciju sa DOM stablom, njegovo menjanje i obilaženje. Sve izmene napravljene nad čvorovima DOM stabla biće vidljive na stranici u okviru elemenata čiji su odgovarajući čvorovi DOM-a menjani. Naglasimo još jednom da programi napisani jezikom *JavaScript* se izvršavaju na klijentskoj mašini

[11] engl. Document Object Model



Slika 19: Prikaz primera DOM stabla.

(iako postoje i upotrebe na serveru).

Dohvatanje *DOM* elemenata vrši se funkcijama definisanim u interfejsu *document*. Neke od najvažnijih navedene su u nastavku:

- **document.getElementsByTagName()** - dohvatanje elemenata čiji je naziv etikete (tag-a) prosleđen kao argument funkcije. Funkcija vraća niz objekata(čvorova stabla) koji odgovaraju pronađenim elementima. Niz je, kao i obično, indeksiran počevši od 0.
- **document.getElementById()** - dohvatanje elementa čiji je *id* prosleđen kao argument.
- **document.getElementsByClassName()** - dohvatanje elemenata čiji je naziv klase prosleđen kao argument funkcije. Funkcija vraća niz objekata(čvorova stabla) koji odgovaraju pronađenim elementima. Niz je, kao i obično, indeksiran počevši od 0.
- **document.getElementsByName()** -dohvatanje elemenata čija je vrednost atributa *name* prosleđena kao argument funkcije. Funkcija vraća niz objekata(čvorova stabla) koji odgovaraju pronađenim elementima. Niz je, kao i obično, indeksiran počevši od 0.
- **document.querySelector()** - dohvatanje elemenata CSS selektorom navedenim kao argument funkcije. Ukoliko ih je više, vraća samo prvi.
- **document.querySelectorAll()** - isto kao ponaša se isto kao *querySelector*(), osim što vraća niz svih objekata koji odgovaraju postavljenom kriterijumu. Niz je, kao i obično, indeksiran počevši od 0.

HTML:
```
<p class='pasus'>Neki pasus</p>
```

```
<h1> Naslov nakon prvog pasusa</h1>
<div id='omotac'>
        <p class='pasus' id='pasus1'>Prvi pasus</p>
        <p class='pasus' id='pasus2'>Drugi pasus</p>
        <p class='pasus' id='pasus3'>Treci pasus</p>
</div>

JS:
var omotac_element = document.getElementById("omotac");
var svi_pasusi = document.getElementsByTagName('p');
var pasusi_u_omotacu = document.querySelectorAll('#omotac .pasus'); //
var prvi_pasus = document.querySelector('#omotac .pasus');
var drugi_pasus = document.querySelector('p:nth-child(2)');
var treci_pasus = pasusi_u_omotacu[2];
```

Treba obratiti pažnju da se nazivi tagova navode između navodnika. Osim njih, uslove navodimo između navodnika. Primetimo da se uslovi mogu postavljati korišćenjem istih oznaka kao u *CSS*-u, pa tako uslov *'#omotac .pasus'* tumači se kao: "u elementu sa *id*-jem *omotac* pronađi sve elemente koji pripadaju klasi pasus. " Svojstvima dohvaćenog elementa pristupa se uz pomoć tzv. **tačka notacije**, što znači da nakon naziva promenljive, navodimo tačku, a potom ime svojstva. Sadržaj unutar etiketa *HTML* elementa nalazi se u svojstvu *innerHTML* objekta koji odgovara dohvaćenom elementu. Sadržaj ima tip niske.

```
HTML:
<p id='pasusneki'>Neki<a href=''>link</a> pasus</p>

JS:
var sadrzaj = document.getElementById('pasusneki').innerHTML;
console.log(sadrzaj);
```

Da bismo pristupili tekstualnom sadržaju nekog elementa koristimo svojstvo *textContent*.

```
HTML:
<p id='pasusneki'>Neki <a href=''>link</a> pasus</p>

JS:
var sadrzaj = document.getElementById('pasusneki').textContent;
console.log(sadrzaj);
```

Razlika između ova dva pristupa je što ćemo u prvom slučaju dobiti sav sadržaj unutar $< p >$ etiketa. U drugom slučaju, dobićemo samo tekstualni sadržaj kako je i prikazan na stranici.

Kada koristimo metode koje vraćaju više od jednog elementa, na primer *getElementsByTagName*, kako bismo saznali koliko je takvih elemenata možemo koristiti svojstvo *length*.

```
var broj_elemenata = svi_pasusi.length;
console.log(broj_elemenata);
```

Kako bismo pristupili pojedinačnom elementu koristimo notaciju kao kod nizova (pošto metod zapravo i vraća niz elemenata).

```
var pasus = svi_pasusi[0];
console.log(pasus);
var poslednji = svi_pasusi[broj_elemenata-1].textContent;
console.log(poslednji);
```

CSS svojstvima elementa pristupa se pomoću svojstva *style*. Ovo svojstvo nam dozvoljava da eksplicitno zadajemo, menjamo i očitavamo stilove. Objekat koji odgovara svojstvu *style* sadrži samo eksplicitno zadate CSS vrednosti, ne i one koje nisu navedene CSS-om a koje je pregledač samostalno izračunao. Vrednosti CSS svojstva se pristupa preko njenog naziva, ukoliko naziv sadrži crticu za pristup svojstvu se koristi naziv u kome crtica ne postoji, ali je prvo slovo posle crtice u originalnom zapisu veliko (npr. *background-color* $\rightarrow$ *backgroundColor*). Izračunate CSS vrednosti dobijaju se funkcijom *getComputedStyle* iz interfejsa *windows*, element čija se izračunata svojstva traže se prosleđuje kao argument funkcije. Novu vrednost nekog svojstva postavljamo uz pomoć *element.style.osobina* = *vrednost*. Možemo na ovakav način dodavati i nove klase elementima, koristeći svojstvo *className*.

```
var stil_pasusa = pasus.style;
console.log(stil_pasusa);
var izracunati_stil = window.getComputedStyle(pasus);
console.log(izracunati_stil.color);
stil_pasusa.backgroundColor = 'gray';
document.querySelector('#pasus1').style.fontSize = '12px';
document.querySelector('#pasus2').className = 'doterani_pasus';
```

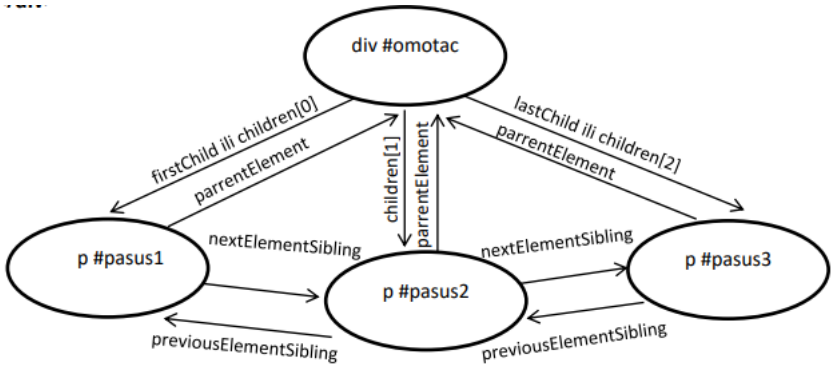
Vezba Uraditi zadatke iz osnova DOM-a.

Kada dohvatimo neki *HTML* element, moguće je kretanjem kroz *DOM* stablo doći i do ostalih elemenata prateći veze sa ostalim elementima. Roditeljskom elementu nekog elementa pristupa se preko svojstva *parentElement*, koja sadrži pokazivač na roditeljski element. Elementu na istom nivou *DOM* stabla *brat/sestra* elementa, koji u *HTML* strukturi sledi nakon dohvaćenog pristupa se svojstvom *nextElementSibling*. Očekivano, njegovom prethodniku pristupa se sa *previousElementSibling*.

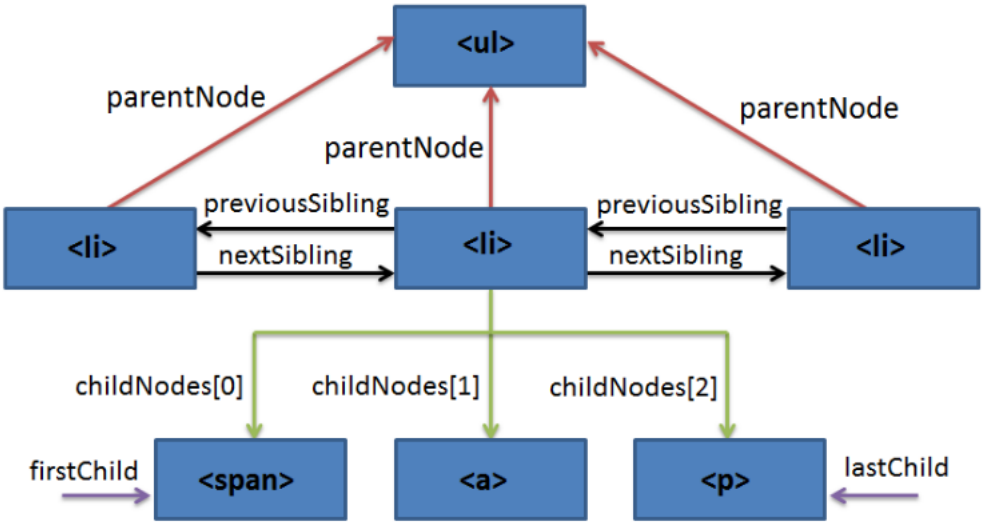
Na veoma sličan način možemo posmatrati i strukturu u odnosu na roditeljski element. Naime, prvom direktnom potomku roditeljskog elementa pristupa se sa *firstElementChild*, poslednjem sa *lastElementChild*. Svim direktnim potomcima se pristupa svojstvom *children*, koje sadrži niz objekata koji odgovaraju direktnim potomcima elementa. Pogledajmo kroz primer:

```
var omotac = document.getElementById('omotac');
var prvi_pasus = omotac.firstElementChild;
var drugi_pasus = prvi_pasus.nextElementSibling;
var treci_pasus = drugi_pasus.parentElement.lastElementChild;
var naslov = omotac.previousElementSibling;
```

Na slici 20 se vidi grafički prikaz. Na slici 21 se na primeru liste jasno vide odnosi između elemenata: Atributima dohvaćenih elemenata pristupa se funkcijom *getAttribute*() kojoj se kao argument prosleđuje naziv atributa čija se vrednost traži. Postavljanje atributa elementima vrši se funkcijom *setAttribute*() čiji je



Slika 20: Prikaz odnosa između elemenata u DOM stablu.



Slika 21: Prikaz odnosa između elemenata u DOM stablu.

prvi argument naziv novog atributa, a drugi argument željena vrednost novog atributa.

```
HTML:
<p id='pasus' title='Neki naslov'>Neki pasus</p>

JS:
var pasus = document.getElementById('pasus');
var title = pasus.getAttribute('title');
console.log(title);
pasus.setAttribute('class', 'klasa1');
```

4.2.2 Kreiranje elemenata

Nekada ne želimo samo da menjamo postojeće elemente, već želimo i da kreiramo nove. Pravljenje novih elemenata vrši se funkcijom *createElement* iz interfejsa *document*. Kao argument ova funkcija prima naziv etikete novog elementa (naziv etikete se navodi pod navodnicima). Element se vezuje u *DOM* stablo kao poslednji sin nekog elementa funkcijom *appendChild* čiji je argument element koji se u stablo dodaje. U slučaju da ne iskoristimo append child, element neće biti prikazan.

```
var omotac = document.getElementById('omotac');
var novi_pasus = document.createElement('p');
novi_pasus.innerHTML = "Treci pasus <a href=''>link</a>";
novi_pasus.setAttribute('id', 'pasus3');
//obavezno vezivanje novog pasusa kao poslednje dete omotaca
omotac.appendChild(novi_pasus);
```

Ukoliko zelimo da postavimo tekstualni sadržaj nad elementom, moramo napraviti novi tekstualni čvor sa *createTextNode*, a potom ga sa *appendChild* dodati odgovarajućem elementu. Primetiti da sa *createTextNode* takođe dodajemo tekst kao sa *innerHTML*. Razlika je u tome što će *innerHTML* tumačiti etikete i prikazivati ih na odgovarajući način dok *createTextNode* nece. Još jedna razlika koju možemo videti je ta da sa *innerHTML* zamenjujemo celokupni sadržaj elementa novim sadržajem, dok sa *createTextNode* dodajemo novi sadržaj na postojeći.

```
var cetvrti = document.createElement('p');
var sadrzaj = document.createTextNode("Cetvrti pasus <a href=''>link</a>");
cetvrti.appendChild(sadrzaj);
omotac.appendChild(cetvrti);

//createTextNode dodaje sadrzaj na postojeci
var pasus1 = document.getElementById('pasus1');
var sadrzaj = document.createTextNode('jos neki tekst unutar pasusa1');
pasus1.appendChild(sadrzaj);
//innerHTML zamenjuje sadrzaj u potpunosti
pasus1.innerHTML = "Tekst unutar pasusa1";
```

Dodavanjem novih elemenata sa *appendChild*, uvek dodajemo element kao poslednjeg naslednika u nizu elementa na koji vršimo "lepljenje". Ako bismo želeli da dodamo novi element pre nekog drugog elementa koristimo *insertBefore*(). Ova funkcija prima dva argumenta. Prvi, novokreirani čvor, koji želimo da ubacimo pre drugog argumenta, koji predstavlja neki postojeći čvor.

```
var pasus_jedan_ipo = document.createElement('p');
var pasus2 = document.getElementById('pasus2');
pasus_jedan_ipo.innerHTML = "pasus i po";
omotac.insertBefore(pasus_jedan_ipo, pasus2);
```

Ako želimo da obrišemo neki element koristimo *removeChild*. Sa *document.body* pristupamo elementima koji se nalaze unutar $<$ *body* $>$ etiketa.

```
HTML:
<p id='obrisi'>Pasus koji zelimo da obrisemo</p>
```

```
JS:
var obrisi = document.getElementById('obrisi');
document.body.removeChild(obrisi);
```

Vezba Uraditi zadatke iz kreiranja elemenata DOM-a.

4.3 Zadaci sa casa

Napomena: Rešenja zadataka nisu jedinstvena. Skoro svi zadaci se mogu uraditi na više načina. Za vežbu pokušati pronaći ili uraditi zadatke na više načina.

4.3.1 Uvodni primeri

Primer 4.3 *Napisati program koji za dva cela broja ispisuje najpre njihove vrednosti, a zatim i njihov zbir, razliku, proizvod, ceo deo pri deljenju prvog broja drugim brojem i ostatak pri deljenju prvog broja drugim brojem.*

Primer 4.4 *Napisati program koji za realne vrednosti dužina stranica pravougaonika ispisuje njegov obim i površinu. Ispisati tražene vrednosti zaokružene na dve decimale.*

Primer 4.5 *Napisati program koji za tri cela broja ispisuje njihovu artimetičku sredinu zaokruženu na dve decimale.*

Primer 4.6 *Napisati program koji za dva cela broja a i b dodeljuje promenljivoj rezultat vrednost 1 ako važi uslov: a) a i b su različiti brojevi b) a i b su parni brojevi c) a i b su pozitivni brojevi, ne veći od 100 U suprotnom, promenljivoj rezultat dodeliti vrednost 0. Ispisati vrednost promenljive rezultat.*

Primer 4.7 *Napisati program koji za dva cela broja ispisuje njihov maksimum.*

Primer 4.8 *Napisati program koji za dva cela broja ispisuje njihov minimum.*

Primer 4.9 *Napisati program koji za tri cela broja ispisuje zbir pozitivnih.*

Primer 4.10 *U prodavnici je organizovana akcija da svaki kupac dobije najjeftiniji od tri artikla za jedan dinar. Napisati program koji za cene tri artikla izračunava ukupnu cenu, kao i koliko dinara se uštedi zahvaljujući popustu.*

Primer 4.11 *Napisati program koji za redni broj dana u nedelji ispisuje ime odgovarajućeg dana. U slučaju pogrešnog unosa ispisati odgovarajuću poruku. Napomena: uraditi zadatak i korišćenjem case*

Primer 4.12 *Napisati program koji za uneti karakter ispituje da li je samoglasnik.*

Primer 4.13 *Napisati program koji 5 puta ispisuje tekst "Mi volimo da programiramo".*

Primer 4.14 *Napisati program koji preko prompta učitava ceo broj n i ispisuje n puta tekst "Mi volimo da programiramo".*

Primer 4.15 *Napisati program koji učitava pozitivan ceo broj n a potom ispisuje sve cele brojeve od 0 do n.*

Primer 4.16 *Napisati program koji učitava dva cela broja n i m ispisuje sve cele brojeve iz intervala [n, m].*
(a) Koristiti while petlju.
(b) Koristiti for petlju.
(c) Koristiti do-while petlju.

Primer 4.17 *Napisati program koji učitava ceo pozitivan broj i izračunava njegov faktorijal.*

Primer 4.18 *Preko prompta unose se realan broj x i ceo pozitivan broj n. Napisati program koji izračunava n-ti stepen broja x.*

Primer 4.19 *Pravi delioci celog broja su svi delioci sem jedinice i samog tog broja. Napisati program za ceo pozitivan broj x ispisuje sve njegove prave delioce.*

Primer 4.20 *Promptom se unosi ceo pozitivan broj n, a potom i n celih brojeva. Izračunati i ispisati zbir onih brojeva koji su neparni i negativni.*

Primer 4.21 *Program učitava realan broj x i ceo pozitivan broj n. Napisati program koji izračunava i ispisuje sumu* $S = x + 2 * x^2 + 3 * x^3 + ...$

Primer 4.22 *Za unetu pozitivnu celobrojnu vrednost n napisati programe koji ispisuju odgovarajuće brojeve. Pretpostaviti da je unos korektan.*
(a) Napisati program koji za unetu pozitivnu celobrojnu vrednost n ispisuje tablicu množenja.
(b) Napisati program koji za uneto n ispisuje sve brojeve od 1 do n^2 *pri čemu se ispisuje po n brojeva u jednoj vrsti.*

Primer 4.23 *Program učitava ceo pozitivan broj n, a potom n realnih brojeva. Odrediti koliko puta je prilikom unosa došlo do promene znaka. Ispisati dobijenu vrednost.*

Primer 4.24 *Kreirati sajt prodavnice u kome svaki kupac određuje sam cenu artikla. U prodavnici se nalazi n artikala čije cene su realni brojevi. Na stranici treba da postoji naziv prodavnice, kao i spisak dostupnih artikala sa slikama, a potom da se uz pomoć prompta učita broj proizvoda koje korisnik želi da kupi. Nakon toga, unose se naziv proizvoda koji korisnik zeli da kupi i cena koju zeli. Za unet broj proizvoda treba u konzoli ispisati naziv proizvoda i ponuđenu cenu, odrediti ukupnu sumu i cenu najjeftinijeg artikla.*

4.3.2 Niske

Primer 4.25 *Za unetu nisku proveriti da li sadrži broj i ispisati u konzoli. Napomena: koristiti ASCII tabelu.*

Primer 4.26 *Zameniti sva pojavljivanja slova a u niski unetoj preko prompta slovom b. Na primer, "Danas je lep dan." sa "Dbnbs je lep dbn.". Napomena: uraditi kreiranjem nove niske.*

Primer 4.27 *Proveriti da li je reč uneta preko prompta palindrom. Palindromi su reči, rečenice koji se mogu čitati unapred i unazad i imaju isto značenje. Primeri za reči: Ana, bob, dovod, kapak, kajak, kuk, melem, neven, oko, pop, potop, ratar, teret.*

Primer 4.28 *Program učitava reč. Napisati program koji proverava da li se od karaktera unete reči može napisati reč Zima.*

Primer 4.29 *Program učitava ceo broj n, a zatim i n karaktera. Napisati program koji proverava da li se od unetih karaktera može napisati reč Zima.*

Primer 4.30 *Napisati program koji učitava karaktere sve do kraja ulaza (do unosa broja 0), a potom ispisuje broj velikih slova, broj malih slova, broj cifara, broj belina i zbir unetih cifara.*

Primer 4.31 *Napisati program koji za unetu nisku proverava da li sadrži reč ima.*

Primer 4.32 *Kreirati sajt zdrave hrane. Sajt treba da sadrži naziv, logo, navigacioni bar (O nama, Proizvodi, Kontakt).*

- *Stranica o nama treba da sadrži: sliku, tekst o radnji, adresu i ostale relevantne informacije.*
- *Kreirati potom stranicu sa proizvodima koja treba da sadrži najpre listu proizvoda (kategorije, pa potom po nekoliko proizvoda), a ispod toga treba da se nalaze neki od popularnih proizvoda sa slikama i cenama na 100 grama.*
- *Stranica za kontakt treba da sadrži kontakt formu (Ime, email, prostor za poruku, itd.).*

Nakon toga koristeci prompt, alert i confirm za sajt sa zdravom hranom napraviti da se za 5 korisnika unese: ime, ocena sajta i predlog novog naziva zdrave hrane. Proveriti ispravnost unetog podatka, ako podatak nije pravilno unet ponuditi korisniku da ponovo unese podatak. Ako jeste, unosi se naredni. Unos ocena dodati na zbir. U konzoli ispisati predlog korisnika u formatu Ime : predlog. Na kraju, ispisati prosečnu ocenu sajta.

4.3.3 Nizovi

Primer 4.33 *Napisati program u kom za niz brojeva program ispisuje niz kvadriranih vrednosti.*

Primer 4.34 *Napisati program u kom se unosi n, a potom i n elemenata niza. Nakon toga, na kraj niza se dodaje suma zadatih brojeva.*

Primer 4.35 *Napisati program koji računa skalarni proizvod vektora a i b. Vektor* $a = (a_1, a_2, ...)$ *i* $b = (b_1, b_2, ...)$*, a skalarni proizvod* $a * b = a_1 * b_1 + a_2 * b_2 + ... + a_n * b_n$*. Najpre se unosi n, a potom po n vrednosti za vektore a i b.*

Primer 4.36 *Napisati program koji za dati niz ispisuje:*
a) elemente na parnim pozicijama u nizu b) parne elemente niza.

Primer 4.37 *Napisati program koji za učitani ceo broj, ispisuje broj pojavljivanja svake od cifara u zapisu tog broja.*

Primer 4.38 *Napisati program koji transformiše uneti niz tako što kvadrira sve negativne elemente niza.*

Primer 4.39 *Sa standardnog ulaza se učitava dimenzija niza, elementi niza i jedan ceo broj k. Napisati program koji štampa indekse elemenata koji su deljivi sa k.*

4.3.4 Objekti

Primer 4.40 *Kreirati objekat "Auto". Objekat treba da sadrži: marku, godište, cenu. Ispisati objekat na izlaz.*

Primer 4.41 *Kreirati objekat "Film", sa imenom filma, godinom izlaska, i ocenom na IMDB-u. Kreirati više takvih objekata i smestiti ih u niz (barem 5). Na izlaz ispisati one filmove koji imaju ocenu preko 8.*

Primer 4.42 *Kreirati objekat "Zaposleni" za neku firmu. Objekat treba da sadrži: ime, prezime, datum rodjenja, identifikacioni broj, poziciju, platu, datum zaposlenja, da li je zaposlen za stalno, pol i spisak iznosa poslednjih 6 plata.*

Primer 4.43 *Za spisak zaposlenih u nekoj firmi (barem 5), uraditi sledeće:*

- *kreirati spisak zaposlenih: ime i prezime, visina plate, da li su stalno zaposleni*
- *izdvojiti one čija su primanja veća od 30000,*
- *izdvojiti stalno zaposlene,*
- *onima koji nisu stalno zaposleni dodati bonuse u iznosu od 10000,*
- *proveriti da li se za zaposlene date firme posebno vodi stavka visina poreza i*
- *za zaposlene koji su stalno zaposleni izračunati prosek poslednje 3 plate.*

Primer 4.44 *Kreirati sajt muzeja. Sajt treba da sadrži navigacioni bar: O nama, Postavke, Karte i Kontakt. Stranica O nama treba da sadrži kratak istorijat muzeja, kao i spisak događaja. Postavke treba da sadrži 3 postavke: Afrička umetnost, Kineska umetnost i Evropska umetnost. Svaka od postavki treba da sadrži bar 3 slike, kratak opis kao i datum početka i završetka postavke. Osim toga treba kreirati inventar za svaku od postavki sa barem 3 stavke. Stranica sa kartama treba da sadrži cene karata prema kategorijama, kao i popuste za specijalne kategorije (mlade do 26 godina, penzionere). Stranica za kontakt treba da sadrži formu: Ime i prezime, E-mail, Poruka.*

U programu kreirati:

- *POSTAVKE: Evropska, Afrička i Kineska. Potom, za svaku od postavki kreirati naziv, datume trajanja (od i do) i sifrarnik inventara (kao niz sifara).*
- *Karte: povlašćene kategorije, standardna cena*
- *Kontakt: ime i prezime, e-mail, poruka.*

4.3.5 Funkcije

Primer 4.45 *Napisati funkciju kvadrat(x) koja računa kvadrat datog broja. Napisati program koji učitava ceo broj i ispuje rezultat poziva funkcije.*

Primer 4.46 *Napisati funkciju apsolutna_vrednost(x) koja izračunava apsolutnu vrednost broja x. Napisati program koji učitava jedan ceo broj i ispisuje rezultat poziva funkcije.*

Primer 4.47 *Napisati funkciju min(x, y, z) koja izračunava minimum tri broja. Napisati program koji učitava tri cela broja i ispisuje rezultat poziva funkcije.*

Primer 4.48 *Napisati funkciju stepen(x, n) koja računa vrednost n-tog stepena realnog broja x. Napisati program koji učitava relan broj x i ceo broj n i ispisuje rezultat rada funkcije.*

Primer 4.49 *Napisati funkciju faktorijel(n) koja računa faktorijel broja n. Napisati i program koji učitava dva cela broja x i y iz intervala [0, 12] i ispisuje vrednost zbira x! + y! (x! znači faktorijel broja x).*

Primer 4.50 *Napisati funkciju sadrzi(x, c) koja ispituje da li se cifra c nalazi u zapisu celog broja x. Funkcija treba da vrati 1 ako se cifra nalazi u broju, a 0 inače. Napisati program koji učitava dva cela broja i ispisuje rezultat poziva funkcije.*

Primer 4.51 *Napisati program za ispitivanje svojstava cifara datog celog broja.*
(a) Napisati funkciju sve_parne_cifre koja ispituje da li se dati ceo broj sastoji isključivo iz parnih cifara. Funkcija treba da vrati 1 ako su sve cifre broja parne i 0 u suprotnom.
(b) Napisati funkciju sve_cifre_jednake koja ispituje da li su sve cifre datog celog broja jednake. Funkcija treba da vrati 1 ako su sve cifre broja jednake i 0 u suprotnom.
Napisati program koji učitava ceo broj i ispisuje da li su sve cifre parne i da li su sve cifre jednake.

Primer 4.52 *Napisati funkciju rastuce(n) koja ispituje da li su cifre datog celog broja u rastućem poretku. Funkcija treba da vrati vrednost 1 ako cifre ispunjavaju uslov, odnosno 0 ako ne ispunjavaju uslov. Napisati i program koji učitava ceo broj i ispisuje poruku da li su cifre unetog broja u rastućem poretku.*

Primer 4.53 *Broj je prost ako je deljiv samo sa 1 i sa samim sobom. Napisati funkciju int prost (int x) koja ispituje da li je dati ceo broj prost. Funkcija treba da vrati 1 ako je broj prost i 0 u suprotnom.*

Primer 4.54 *Napisati funkciju int prebrojavanje(float x) koja prebrojava koliko puta se broj x pojavljuje u nizu brojeva koji se unose sve do pojave broja 0. Napisati program koji učitava vrednost broja x i testira rad napisane funkcije.*

Primer 4.55 *Napisati funkcije za rad sa nizovima celih brojeva.*
(a) Napisati funkciju ucitaj(a, n) koja učitava elemente niza a dimenzije n.
(b) Napisati funkciju stampaj(a, n) koja štampa elemente niza a dimenzije n.
(c) Napisati funkciju suma(a, n) koja računa i vraća sumu elemenata niza a dimenzije n.
(d) Napisati funkciju prosek(a, n) koja računa i vraća prosečnu vrednost (aritmetičku sredinu) elemenata niza a dimenzije n.
(e) Napisati funkciju minimum(a, n) koja izračunava i vraća minimum elemenata niza a dimenzije n.

(f) Napisati funkciju pozicija_maksimuma(a, n) koja izračunava i vraća poziciju maksimalnog elementa u nizu a dimenzije n. U slučaju više pojavljivanja maksimalnog elementa, vratiti najmanju poziciju.
(g)Napisati funkciju koja sve vrednosti niza uvećava za zadatu vrednost m.

Primer 4.56 *Kreirati sajt zdrave hrane. Stranica treba da sadrži odgovarajuće HTML i CSS elemente (slike, naslove, tekst, itd.). Korisnik treba da odabere da li želi da učestvuje u anketi ili ne. ako odabere da želi, koristeći ugrađenu funkciju prompt zahtevati od korisnika predlog za naziv zdrave hrane i ocenu sajta. Postupak uraditi za barem 6 korisnika. Nakon što su svi uneli podatke (ili odlučili da ih ne unesu), ispisati spisak svih predloga, a potom ispisati prosečnu ocenu sajta. Uraditi odgovarajuće provere unosa.*

4.3.6 Periodično izvršavanje funkcija

Primer 4.57 *Kreirati stranicu-prezentaciju pozorista. Sajt treba da sadrži: naslov, paragraf, slike, deo sa kontakt informacijama i ostale odgovarajuće HTML i CSS elemente. Nakon* 10 *sekundi od otvaranja sajta korisnik se obavestava da su tokom januara sve cene karata snižene* 50%.

Primer 4.58 *Kreirati stranicu za brzo anketiranje stanovnika. Po otvaranju stranice nakon 3 sekunde izlazi prozor sa pitanjem "Da li želite da učestvujete u anketi?". Ako je odgovor potvrdan, korisniku se svake 4 sekunde nudi po jedno od 15 pitanja (pitanja idu redom). Pitanja:*

- *Koliko imate godina?*
- *Koji ste nivo obrazovanja stekli?*
- *Da li ste pušač?*
- *Da li jedete meso?*
- *Da li redovno odlazite na zdravstvene preglede?*
- *Da li radite?*
- *Koliko puta nedeljno trenirate?*
- *Posecujete li muzeje?*
- *Posecujete li pozorište?*
- *Imate li partnera?*
- *Koliko stranih jezika govorite?*
- *Imate li kucnog ljubimca?*
- *Koristite li računar svakodnevno?*
- *Koliko sati dnevno gledate TV?*
- *U kom mestu živite?*

Anketiranje traje ukupno minut. Po završetku ankete treba da iskoči prozor sa "Hvala na izdvojenom vremenu!". Na kraju, u konzoli se ispisuje spisak pitanja i korisnikovih odgovora.

4.4 Interakcija sa DOM-om iz jezika JavaScript

4.4.1 Osnove

Primer 4.59 *Kreirati stranicu bioskopa koja sadrži 4 diva. Prvi div sadrzi naslov stranice i logo. Drugi div sadrži sadržaj stranice (content, na primer, najave premijera). Treci div sadrzi informacije poput kontakt telefona i ostalih informacija o cenama karata i datumima nekih od projekcija. Cetvrti div je footer u kome mogu biti smestene adresa preduzeca kao i polje za prijavu za newsletter. Pristupiti pasusu unutar drugog div-a promeniti mu pozadinu sa crvene na sivu. Dohvatiti sve naslove h2 i promeniti im boju na belu. Postaviti boju elementa sa id-jem "neki_odabrani"na plavu. Za liste postaviti veličinu slova na 11px.*

Primer 4.60 *Dohvatiti sve elemente liste unutar elementa sa id-jem cenovnik. Podesiti dohvaćenim elementima slova da budu iskošena, i promeniti boju na belo. Promeniti pozadinsku boju cenovnika na ljubičasto.*

Primer 4.61 *Za kreiranu stranicu koja sadrzi 10 pasusa, postaviti pozadinsku boju svakom drugom na svetlo sivu.*

4.4.2 Kreiranje elemenata

Primer 4.62 *Na stranicu sa pozorištem dodati*

Primer 4.63 *Kreirati stranicu na kojoj se nalazi samo jedan prazan div sa id-jem omotac. U taj div, dodavati elemente korišćenjem funkcije createElement dok se ne dobije stranica koja sadrži.*

Primer 4.64 *Za uneti ceo broj n, izgenerisati tablicu množenja veličine n koristeći table element i dodavanje uz pomoć interakcije sa DOM drvetom. Doterati stranicu koristeći CSS.*

Primer 4.65 *Na osnovu unetog naziva slike, dodati na stranicu primer biografije zadate ličnosti. Na primer, za naziv slike tesla.jpg na stranicu dodati naslov Nikola Tesla, paragraf sa kratkom biografijom, sliku, kao i link ka stranici na wikipediji. Svaka biografija treba da bude prigodno uokvirena. Ponuditi unos za do 3 licnosti, tako sto ce se preko prompta najpre uneti broj licnosti, a potom i nazivi slika. Sajt ukrasiti korišćenjem prigodinih HTML i CSS elemenata.*

Primer 4.66 *Kreirati stranicu koja vrši heširanje tajne poruke koju korisnik unosi preko prompta. Heširanje predstavlja postupak sakrivanja poruke. U ovom zadatku heš treba predstaviti preko tabele, koja u vidu niza predstavlja poruku uz pomoć crvenih, plavih i zelenih kvadratića dimenzija 10x10 piksela. Heš funkcija je sledeća:*

- *ako je ostatak pri deljenju kodnog broja karaktera sa 3 jednak 1 onda zelena boja,*
- *ako je ostatak pri deljenju kodnog broja karaktera sa 3 jednak 2 onda plava boja*
- *inače, crvena.*

Prikaz programa dat je na slici 22.

Slika 22: Prikaz programa.

4.5 Dodatni zadaci

Primer 4.67 *Kreirati stranicu zdrave hrane. Koristeći ugrađenu funkciju prompt zahtevati od korisnika količinu badema koju želi da kupi(u gramima), a potom u alert prozoru ispisati konačnu cenu za unetu količinu. Npr. ako je badem 2000 dinara po kilogramu, a korisnik želi 500 grama, u alertu treba da se ispiše 1000 dinara.*

Primer 4.68 *Kreirati sajt za organizovanje proslava. Sajt treba da sadrži navigacioni bar sa stavkama: o nama, rođendani, proslave jubileja, svadbe i krštenja, kao i kontakt stranu. Rođendanske proslave se sastoje iz više kategorija, kao što su proslave punoletstva, proslave prvih rodjendana i proslave rođendana. Svaka od stavki navigacije vodi ka novim stranicama koje sadrže paragraf sa kratkim opisom usluga koje se nude, tabele sa cenama usluga (Ketering, Premium Ketering, Piće (osnovna i premium ponuda), dekoracija, muzika, transport do lokacije itd.). Svaka od strana sadrži i barem 3 moguće lokacije za proslave, sa sumom cena.*

4.6 Domaći zadaci

Primer 4.69 *Na koje if se odnosi else?*

```
if (izraz1)
    if (izraz2)
        naredba1
else
    naredba2
```

Primer 4.70 *Na koje if se odnosi else?*

```
if (izraz1) {
   if (izraz2)
      naredba1
} else
    naredba2
```

Primer 4.71 *Napisati program koji za uneti pozitivan trocifreni broj na standardni izlaz ispisuje njegove cifre jedinica, desetica i stotina*

Primer 4.72 *Napisati program koji za uneti pozitivan četvorocifreni broj:*
(a) izračunava proizvod cifara
(b) izračunava razliku sume krajnjih i srednjih cifara
(c) izračunava sumu kvadrata cifara
(d) izračunava broj koji se dobija ispisom cifara u obrnutom poretku
(e) izračunava broj koji se dobija zamenom cifre jedinice i cifre stotine

Primer 4.73 *Broj je Armstrongov ako je jednak zbiru kubova svojih cifara. Napisati program koji za dati trocifren broj proverava da li je Armstrongov.*

Primer 4.74 *Napisati program koji ispisuje proizvod parnih cifara unetog četvorocifrenog broja.*

Primer 4.75 *Napisati program koji za ceo broj x ispisuje njegov znak, tj da li je broj jednak nuli, manji od nule ili veći od nule.*

Primer 4.76 *Napisati program koji za uneti prirodan broj ispisuje da li je on deljiv sumom svojih cifara.*

Primer 4.77 *Promptom se unosi ceo pozitivan broj n, a potom i n celih brojeva. Izračunati i ispisati zbir onih brojeva koji su parni.*

Primer 4.78 *Program učitava cele brojeve sve do unosa broja nula 0. Napisati program koji izračunava i ispisuje aritmetičku sredinu unetih brojeva na četiri decimale.*

Primer 4.79 *U prodavnici se nalaze artikli, čije cene su realni pozitivni brojevi. Program unosi cene artikala sve do unosa broja nula 0. Napisati program koji izračunava i ispisuje prosečnu vrednost cena u radnji.*

Primer 4.80 *U prodavnici se nalaze artikala čije cene su realni pozitivni brojevi. Program unosi cene artikala sve do unosa broja nula 0. Napisati program koji izračunava i ispisuje prosečnu vrednost cena u radnji.*

5 Zaključak

A Dodatak