Javascript

Program

- Program je niz instrukcija koje govore računaru kako da uradi neki posao
- Možemo posmatrati program kao recept za pripremu nekog jela, imamo potrebne sastojke, njihove kolicine i korake koje je potrebno da obavimo da bismo od sastojaka pripremili jelo

- Ljudi uglavno obavljaju poslove bez da previše razmiljaju o njima (npr. možemo voziti auto, spremati doručak, placćati račune, bez da pratimo neke instrukcije)
- Medjutim da bi računar obavio neki posao, svaki put mora da prati korake, zbog čega je potrebno da ima detaljne instrukcije

Program

- Pisanje programa možemo podeliti u tri osnovna koraka:

1. Postavljanje cilja
   - odgovor na pitanje *Šta to želimo da uradimo/postignemo?*

2. Podela na sitne zadatke
   - odgovor na pitanje *Koji su to zadaci koje treba da obavimo da bismo završili ciljni zadatak?*

3. Zapisivanje redom svih koraka koji vode do cilja
   - programiranje je zapisivanje ovih koraka u nekom programskom jeziku koji računar može da razume (u našem slučaju to je Javascript)

Primer

- Na primer zadatak nam je da sredimo sobu
- Da bismo uradili taj posao, potrebno je da pratimo ovaj niz koraka:

  - skinemo prljavu posteljinu
  - stavimo čistu posteljinu
  - namestimo krevet
  - obrisemo prašinu
  - usisamo pod
  - obrišemo pod

Programski jezik

- Računari razumeju ono što je napisano na programskom jeziku
- Programski jezik, kao i drugi govorni jezici, imaju:
  - Vokabular – reči koje računar razume
  - Sintaksa – pravila za sklapanje rečenica od reči iz vokabulara
- Pored vokabulara i sintakse programskog jezika, potrebno je da izučimo pristup rešavanju problema odnosno na koji način računar obavlja neki zadatak
- U žargonu se to zove *programmatic approach*

Primer

- Posmatramo red u banci, treba da objasnimo računaru kako da odredi ko je najviši u redu
- Računaru treba niz koraka:
  - Izračunaj visinu prvog čoveka u redu
  - Pretpostavi da je on najviši
  - Pogledaj ostale ljude u redu, jednog po jednog, izračunaj visinu i uporedi je sa najvišom čovekom do sad
  - U svakom koraku, ukoliko pronađeš nekog ko je viši od trenutno najvišeg čoveka, on postaje nov najviši čovek
  - Kada uporediš sa svim ljudima u redu, rezultat je onaj koji je ostao najviši
- To znači da računar treba da pogleda svakog čoveka i ispita da li je on viši od trenutno najvišeg, tj. mora da prođe kroz ceo red da bi odredio najvišeg

Objekti

- Kako računar modeluje svet?
  - Npr. kako da objasnimo računaru šta je hotel, automobil, vožnja, broj soba?

- U programiranju, fizičke stvari predstavljamo objektima

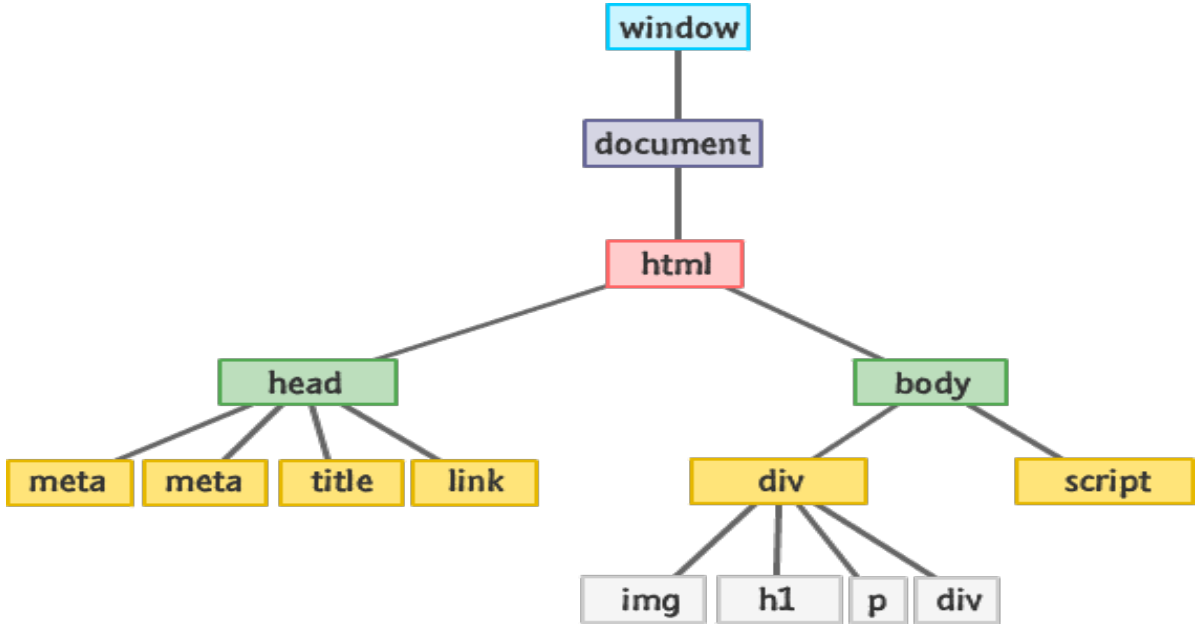
- Objekti mogu biti različitog tipa
  - automobil i hotel jesu različiti tipovi objekta
  - Golf je jedna instanca objekta tipa automobil
  - Hilton je jedna instanca objekta tipa hotel

Objekti

- Objekti imaju:
  - svojstva
  - dogadjaje na koje reaguju
  - metode koje mogu da izvršavaju
- Npr. automobil ima:
  - boju - svojstvo
  - reaguje na klik dugmeta za otkljucavanje - dogadjaj
  - greje - metoda
- Instance objekata imaju određene vrednosti svojstava
- Npr. Lukin Golf je zelene boje
- Svojstva objekata takođe mogu biti objekti
- Npr. boja automobila može da se predstavi kao objekat Boja koji ima svojstva *jacina_crvene, jacina_zelene, jacina_plave*

Document object

- Kada ukucamo adrese neke stranice, browser učita HTML kod stranice i od njega napravi objekat tipa dokument i sačuva ga u memoriji
- Nakon toga alat za prikazivanje stranica ovaj objekat prikaže ga na ekranu na određen način
- Kao i drugi objekti, dokument ima svojstva, metode i dogadjaje na koje reaguje



Javascript

- Jezik koji nam služi da dodamo interaktivnost web stranicama
- Pristupamo i menjamo obeležavanja i sadržaj stranica dok ih korisnik gleda u browseru
- Reagujemo na dodađaje i sprovodimo određene akcije u zavisnosti od dogadjaja
- Potrebno je da na neki način kažemo browseru šta treba da uradi u određenim trenucima
- Ideja je da browseru damo niz koraka koje treba da prati kada se desi neki dogadjaj

Javascript

- Npr. želimo da kada korisnik klikne na dugme ‘*Next picture*’ prikažemo sledeću sliku

- Kada se desi klik na dugme ‘*Next picture*’ browser treba da:
  - pronađe gde se na stranici nalazi trenutka slika
  - skloni trenutnu sliku
  - pronađe sledeću koju treba da prikaže
  - prikaže sledeću sliku

Javascript – prvi program

- Želimo da napišemo program koji ispisuje *Hello world!*
- U `body` tag naše stranice dodamo:

```
<script type="text/javascript">

console.log("Hello world!");

</script>
```

- Da bismo videli ispis potrebno je da otvorimo **konzolu** (eng. Console)
- Otvorimo stranicu u browseru, kliknemo desni klik, izaberemo *Inspect* , otvoriće nam se prozor, izaberemo karticu gde piše *Console,* osvežimo stranicu (refresh, klikom na dugme u browseru ili F5) i ispisaće nam se *Hello world!*

Javascript – prvi program

- Šta se tačno desilo u prethodnom primeru?
- Ukucali smo adresu veb stranice u browseru i pritisnuli dugme za prikazivanje
- Browser je sa te adrese počeo da čita HTML kod veb stranice i pravi objekat dokumenta na osnovu strukture stranice
- Kada naleti na `script` tag, zastane i pročita Javascript kod (u našem slučaju to je ispisivanje Hello world!)
- Alat koji se zove **interpreter** naš program sa jezika Javascript prevodi na jezik koji računar razume
- Nakon toga računar izvršava naš program, browser nastavlja sa čitanjem stranice i pravljenjem objekta
- Zbog toga je važno da `script` tag stavimo na kraj `body` taga, kako bi browser konstruisao bitne delove dokument objekta

Javascript - uključivanje

- Javascript kod možemo uključiti u HTML stranicu na više načina
- Glavna tri načina su:
  - pišemo kod unutar `script` taga

```
<script type="text/javascript">
console.log('Kod koji kucamo unutar script taga');
</script>
```

  - pišemo kod u posebnom fajlu (sa ekstenzijom .js) i `script` tagu postavimo link ka tom fajlu pomoću atributa `href`

```
<script type="text/javascript" href="nas_fajl.js"></script>
```

  - pišemo kod unutar nekih atributa (npr. `onclick, onmouseover` ...)

```
<p onclick="console.log('Onclick se desio')"></p>
```

- Kako bismo lakše testirali možemo otvoriti konzolu i kucati kod direktno u browseru, ali taj kod ne ostaje zapamćen na našoj stranici
- Isto tako mozemo kucati kod u polju gde kucamo adresu web stranice, ali on takodje ne ostaje zapamćen

Javascript - naredbe

- Već smo pomenuli da program sadrži niz koraka odnosno instrukcija koje računar izvršava

- Svaki instrukcija se naziva **naredba** (eng. statement) i završava se karakterom **;**

```
console.log('Ovo je prva naredba');
console.log('Ovo je druga naredba');
console.log('Ovo je treća naredba');
```

- Dobra praksa je da pišemo jednu naredbu u jednom redu
- Naredbe mogu da se grupišu u blokove
- Blokovi su okruženi zagradama **{** i **}** nakon kojih ne stoji karakter **;**

Javascript – komentari

- Komentare pisemo nakon karaktera // ukoliko su u jednom redu, odnosno izmedju karaktera /* i */ ukoliko su u više redova

```
<script type=“text/javascript”>
// Ovo je jednolinijski komentar

/*
Ovo je komentar
koji zauzima
vise linija
*/
console.log(‘Hello world’);
</script>
```

- Komentare pišemo kako bismo bolje objasnili šta neki kod radi i jako su korisni kada više ljudi radi na istom projektu

Javascript - promenljive

- U prethodnim primerima, ispisivali smo tekst naredbom:

```
console.log('Hello world');
```

- Svaki put kad otvorimo stranicu ispisaće se isti tekst “Hello world” koji smo unapred zadali
- Da bismo promenili tekst koji se ispisuje, potrebno je da promenimo naredbu programa:

```
console.log('Zdravo svete');
```

- Međutim ukoliko želimo da ispišemo tekst koji korisnik unese u tekstualno polje, taj tekst ne znamo unapred?

Javascript - promenljive

- Često nam je potrebno da zapamtimo neku informaciju da bismo je posle iskoristili u programu
  - recimo da zapamtimo koji tekst je korisnik uneo u tekstualno polje, da bismo mogli kasnije da ga ispišemo
- Tome služe **promenljive** (eng. variable)
  - možemo ih posmatrati kao kutije, koje imaju ime, i u kojima možemo čuvati podatak
- Podatak koji promenljiva čuva se takođe naziva **vrednost** promenljive (eng. value)
- Npr. možemo napraviti promenljivu, dati joj neko ime i u njoj sačuvati tekst koji korisnik unese
- Kasnije, kada želimo da ispišemo taj tekst, zadaćemo instrukciju računaru da ispiše podatak koji se nalazi u promenljivoj sa datim imenom
- Podatak koji se čuva u promenljivoj se može menjati (ukoliko korisnik obrise ili promeni tekst koji je uneo)

Javascript - promenljive

- Da bismo koristili promenljive za čuvanje podataka, potrebno je prvo da ih napravimo tj. **deklarišemo** (eng. declaration)

```
var tekst;
```

- `var` – **ključna reč** (eng. keyword)
- `tekst` – ime tj. **identifikator** promenljive (eng. identifier)

- Kada naleti na ključnu reč `var` interpreter zna da je treba da napravi mesto u memoriji za vrednost promenljive koja se zove `tekst`

Javascript - promenljive

- Ukoliko želimo da sačuvamo neki podatak u promenljivoj koju smo napravili, to radimo na sledeći način:

```
tekst = ‘Hello world’;
```

- = je operator **dodeljivanja** (eng. assignment)

- Ovom naredbom smo rekli računaru: u prostor koji je rezervisan za promenljivu sa imenom `tekst` sačuvaj podatak `‘Hello world’`, odnosno promenljivoj sa imenom `tekst` dodeli vrednost `‘Hello world’`
- Ako promenljivoj ne dodelimo nikakvu vrednost, imaće posebnu vrednost `undefined` (indikator da joj nismo ništa dodelili)

Javascript - promenljive

- Nakon što smo napravili promenljivu i dodelili joj vrednost, možemo je koristiti dalje u programu
- Npr.

```
console.log(tekst);
```

- Ova naredba će ispisati onaj tekst koji se nalazi u promenljivoj `tekst`
- Deo programa u kome možemo koristiti neku promenljivu se zove **doseg** (eng. scope)

Javascript – tipovi podataka

- Podaci koje promenljive čuvaju mogu biti različitog tipa
- Svaka promenljiva može da čuva bilo koji tip podatka

- Neki tipovi su:
  - numerički podaci – **brojevi**
    - `100`
    - `0.5`
    - `-4`

```
var temperatura;
temperatura = 28.3;
```

ili skraćeno

```
var temperatura = 28.3;
```

Javascript – tipovi podataka

- tekstualni podaci – **string**
  - ‘ovo je neki tekst’
  - možemo ih pisati između jednostrukih ‘ovo je neki tekst’ i dvostrukih navodnika “ovo je neki tekst”
  - bitno je da isti tip navodnika pisemo sa obe strane (ovo je pogresno ‘tekst”)
  - navodnici “, ‘ i kosa crta \ su posebni karakteri, ako hoćemo njih da ispišemo to radimo ovako:
    - “ Ovo: \” je dvostruki navodnik ”
    - ‘ Ovo: \’ je jednostruki navodnik ’
    - “ Ovo: \\ je kosa crta”
  - da bismo ispisali novi red ili tab (veliki razmak) to radimo ovako:
    - “ Ovo: \n je novi red”
    - “ Ovo: \t je tab (veliki razmak)”

```
var poruka = “Uspesno ste rezervisali proizvod.”;
```

Javascript – tipovi podataka

- **boolean** podaci
  - mogu imati vrednosti `true` i `false`
  - ovaj tip podataka može biti vrlo korisan, videćemo na nekim primerima

- **nizovi** podataka
  - poseban tip podataka koji može da čuva listu (niz) vrednosti
  - niz ne mora da čuva vrednosti istog tipa, ali dobra praksa je da ipak sadrži vrednosti istog tipa
  - niz bi trebalo da čuva podatke koji imaju nešto zajedničko (recimo niz sastojaka za pravljenje torte)
  - niz konstruišemo pomoću uglastih zagrada `[` i `]` a vrednosti razdvajamo zarezima `,` (ovakav način pravljenja niza se naziva *array literal*)

```
var sastojci = [“jaja”, “brasno”, “secer”, “puter”, “slag”, “jagode”];
```

Javascript – tipovi podataka

- niz može da čuva koliko god želimo podataka i ne moramo unapred znati koliko će podataka čuvati
- niz može da bude prazan (ne čuva ni jedan podatak)

```
var sastojci = [];
```

- drugi način konstruisanja niza je pomoću ključne reči `new` nakon čega sledi `Array()` (ovaj način se naziva *array constructor*)

```
var sastojci = new Array("jaja", "brasno", "secer", "puter", "slag",
"jagode");
```

- elementi niza su poređani u onom redosledu u kom smo ih konstruisali
- da bismo pristupili nekom elementu iz niza moramo znati koji je po redu odnosno njegov **indeks**
- redni brojevi elemenata u nizu kreću od 0, tako da je u našem primeru element `"secer"` na poziciji tj. indeksu 2

Javascript – tipovi podataka

- elementima u nizu pristupamo na sledeći način:

  ime_promenljive[*indeks_elementa*]

  “jaja” `sastojci[0]`
  “brasno” `sastojci[1]`
  “secer” `sastojci[2]`
  …

- ukoliko želimo da postavimo element u niz na određenu pozciju to radimo ovako

```
sastojci[2] = “cokolada”
```

- svaki niz ima svojstvo koje se zove `length` i koje sadrži broj elemenata u nizu

  `sastojci.length` je broj 6

- Postoji još različitih tipova podataka u Javascript jeziku **objekti** (eng. object), `undefined`, `null`, ali njih ćemo proučavati kasnije

Javascript – imena promenljivih

- Pravila
  - mogu pocinjati slovima, $ ili _
  - mogu sadržati slova, brojeve, $ i _
  - ne smeju biti neke od ključnih reči (npr. `var`)

- Dobra praksa je nazivati promeljive smisleno tako da se može zaključiti za šta su namenjene (npr. `message`)

- Ukoliko se ime promenljive sastoji od dve reči (npr. `first name`) praksa je da ime konstruišemo na jedan od sledećih načina
  - `firstName` (ovaj način se naziva *camel case*)
  - `first_name` (ovaj način se naziva *snake tail*)

Javascript – operatori i izrazi

- **Operator** je operacija tj. funkcija koja se primenjuje na argumente i daje rezultat
- Npr. `+` je operator sabiranja, primenjuje se na dva sabirka i kao rezultat izračunava zbir

```
x = 3 + 5;
```

- **Izraz** je konstrukcija koja se izračunava
- Npr. u gornjem primeru `3 + 5` je izraz
- U Javascript-u postoji više različitih tipova operatora, neki od njih su:
  - operatori dodeljivanja
  - aritmetički operatori
  - string operatori
  - operatori poređenja
  - logički operatori

Javascript – aritmetički operatori

`+` sabiranje `2 + 4` daje rezultat `6`
`-` oduzimanje `9 - 5` daje rezultat `4`
`*` množenje `5 * 5` daje rezultat `25`
`/` deljenje `15 / 5` daje rezultat `3`
`%` ostatak pri deljenju `9 % 4` daje rezultat `1`
`**` stepenovanje `2 ** 3` daje rezultat `8`
`++` povećavanje za jedan `5++` daje rezultat `6`
`--` umanjenje za jedan `12--` daje rezultat `11`

- Izraz može da sadrži više operatora (npr. `2 + 3 + 9`), onda se izračunavanje vrši sa leva na desnu stranu
- Ukoliko imamo različite operatore u izrazu onda se izračunavaju onr operacije koje imaju veći prioritet

Javascript – aritmetički operatori

`+` sabiranje `2 + 4` daje rezultat `6`
`-` oduzimanje `9 - 5` daje rezultat `4`
`*` množenje `5 * 5` daje rezultat `25`
`/` deljenje `15 / 5` daje rezultat `3`
`%` ostatak pri deljenju `9 % 4` daje rezultat `1`
`**` stepenovanje `2 ** 3` daje rezultat `8`
`++` povećavanje za jedan `5++` daje rezultat `6`
`--` umanjenje za jedan `12--` daje rezultat `11`

- Izraz može da sadrži više operatora (npr. `2 + 3 + 9`), onda se izračunavanje vrši sa leva na desnu stranu
- Ukoliko imamo različite operatore u izrazu onda se izračunavaju one operacije koje imaju veći prioritet

Javascript – string operator

- Ovaj operator predstavlja operaciju nadovezivanja stringova
- Piše se + spaja dva stringa i kao rezultat daje jedan string

```
var ukus = “cokoladna”
var jelo = “torta”
var ukusnoJelo = ukus + “ ” + jelo
```

- Promenljiva ukusnoJelo je string “cokoladna torta”

Javascript - funkcije

- Funkcije su grupisane naredbe koje završavaju određen posao
- Pravimo ih kada znamo da ćemo neki zadatak, posao, izračunavanje koristiti više puta u programu
- Služe da bolje organizujemo programski kod (recimo da ne ponavljamo iste naredbe više puta u kodu)
- U većini slučajeva funkcijama dajemo **ime** (eng. function name) kako bismo mogli da ih koristimo kada su nam potrebne
- Zadatak koji funkcija obavlja se sastoji od naredbi koje su grupisane u blok koji se naziva **telo** funkcije (eng. function body)
- Funkciji mogu da zatrebaju neki ulazni podaci, oni se nazivaju **argumenti** funkcije (eng. function arguments)
- Funkcija može da vrati neki rezultat, on se naziva **povratna vrednost** (eng. return value)

Javascript - funkcije

- Bitno je da razlikujemo definisanje funkcije od pozivanja funkcije
- Definisanje funkcije je davanje imena funkciji i pisanje bloka naredbi koje funkcija izvršava
- Funkciju pozivamo u programu onog trenutka kada želimo da ona završi posao i vrati nam rezultat
- Npr. Imamo funkciju koja računa cenu nekog materijala na osnovu količine
- Pozivamo je tek kada korisnik unese količinu materijala ili bilo gde u programu gde nam treba izračunata cena materijala
- Pre toga smo je definisali, dali joj ime i napisali blok naredbi koji, na osnovu količine koju smo prosledili kao argument, pravilno računa cenu i vraća rezultat
- Funkcija ne mora da ima ime – **anonimna funkcija** (eng. anonymous function), argumente ni povratnu vrednost

Javascript - funkcije

- Definisanje:

```
function izracunajCenu(kolicina) {
    var jedinicnaCena = 128.5;
    var rezultujucaCena = jedinicnaCena * kolicina;
    return rezultujucaCena;
}
```

- Pozivanje:

```
var cena = izracunajCenu(17);
```

Javascript - funkcije

- Funkciju deklarišemo pomoću ključne reči `function` nakon koje sledi ime funkcije - `izracunajCenu`
- Unutar zagrada `(` i `)` se nalaze imena argumenata (razdvojeni zarezom ako ih ima više
- Zagrade `{` i `}` okružuju blok naredbi koje funkcija izvršava
- Pomoću ključne reči `return` vraćamo rezultat
- Funkciju pozivamo tako što navedemo njeno ime i u zagradama prosledimo vrednosti argumenata (razdvojene zarezom ukoliko ih ima više)

```
    // programski kod koji se izvrsa pre poziva funkcije
izracunajCenu(17)
// programski kod koji se izvrsava kada funkcija zavrsi i vrati rezultat
```

- Kada smo pozvali funkciju, izvršavaju se naredbe koje se nalaze u telu funkcije i kada funkcija vrati rezultat, naš program nastavlja dalje sa radom

Javascript - funkcije

- Funkcije možemo definisati i ovako:

```
var izracunajCenu = function(kolicina) {
var jedinicnaCena = 128.5;
var rezultujucaCena = jedinicnaCena * kolicina;
return rezultujucaCena;
}
```

- Na ovaj način smo funkciju sačuvali u promenljivoj sa imenom `izracunajCenu`
- Primetimo da na ovaj način nismo dali ime funkciji (eng. anonymous function) već smo je sačuvali u promenljivu koja ima ime
- Funkciju pozivamo na isti način:

```
izracunajCenu(17)
```

Javascript – funkcije

- Promenljive koje smo deklarisali unutar funkcije možemo koristiti samo unutar te funkcije
- Ovakve promenljive se nazivaju **lokalne** (eng. local variables) jer imaju **lokalni opseg** (eng. local scope) odnosno ne možemo ih koristiti van tela funkcije
- One se konstruišu u trenutku kada se funkcija pozove i postoje sve dok funkcija ne završi sa radom, nakon toga se brišu
- Sledeći put kada pozovemo funkciju, one se opet konstruišu, postoje dok se funkcija ne završi, i tako svaki put
- Promenljive koje smo deklarisali van funkcije nazivamo **globalne** (eng. global variables) i one imaju **globalni opseg** (eng. global scope)
- Globalne promenljive možemo koristiti bilo gde u programu

```
// globalna promenljiva
var popust = 20;

function izracunajCenu(kolicina) {
        // lokalna promenljiva
     var jedinicnaCena = 128.5;
        // mozemo koristiti promenljivu popust jer je globalna i promenljivu jedinicnaCena
        var rezultujucaCena = (jedinicnaCena – popust) * kolicina;
     return rezultujucaCena;
}
// ovde mozemo koristiti promenljivu popust ali ne mozemo promenljivu rezultujucaCena
```

Javascript - objekti

- Objekat je poseban tip podataka koji ima svojstva i metode (odnosno promenljive i funkcije)
- Koristimo ih kada želimo da modelujemo kompleksnije podatke (npr. automobil, restoran, korisnika, itd)
- Svojstva nam govore nešto o objektu (npr. ime restorana, broj stolova) i svaka instanca objekta ima određene vrednosti svojstava
- Metode predstavljaju akcije, zadatke koji su vezani za objekat (npr. provera da li ima slobodnih stolova u restoranu)
- Objekti pravimo od parova **ključ** (eng. key) i **vrednost** (eng. value)
- Ukoliko je vrednost za određeni ključ funkcija, onda taj ključ predstavlja metodu, inače predstavlja svojstvo
- Ključevi moraju biti jedinstveni, ne smeju postojati dva ista ključa u jednom objektu

Javascript - objekti

- Ime (ključ) – vrednost parovi se često koriste
- Promenljive imaju svoje ime (ključ) i možemo im dodeliti vrednost (broj, string, boolean)
- Nizovi imaju svoje ime i listu vrednosti
- Funkcije imaju svoje ime i kao vrednost blok naredbi koje izvršavaju
- Objekti imaju skup parova ime (ključ) – vrednost

- Podsetimo se i u CSS-u smo imali pravila koja se sastoje od imena i vrednosti

Javascript - objekti

```
  var restoran = {
  // svojstva
ime: ‘Mia’,
brojStolova: 20 ,
zauzeto: 7,
cigarete: true,
jela: [‘pizza’, ’lazanje’, ’pasta’],
  // metodi
proveriSlobodne: function() {
return this.brojStolova – this.zauzeto;
}
}
```

- Primetimo ključnu reč `this`, u ovom slučaju `this` predstavlja naš objekat restorana
- Tako da je `this.brojStolova` broj 20 a `this.proveriSlobodne` funkcija koja vraća broj slobodnih stolova u tom restoranu

Javascript - objekti

- Pomoću *ime_objekta.kljuc* ili *ime_objekta[‘kljuc’]* pristupamo svojstvima i metodima objekta

`restoran.zauzeto` ili `restoran[‘zauzeto’]` je broj `7`

- Možemo promeniti vrednosti:

```
restoran.zauzeto = 8;
```

- Možemo dodati nove vrednosti koje nisu postojale:

```
restoran.pice = [‘voda’, ‘sok’, ‘vino’];
```

- Možemo obrisati vrednosti pomoću ključne reči `delete`:

```
delete restoran.cigarete;
```

- Ako pokušamo da pristupimo svojstvu ili metodi u objektu koju smo obrisali, dobićemo vrednost `undefined`

Javascript - objekti

- U prethodnom primeru smo napravili objekat restorana sa konkretnim vrednostima
- Ali često nam je potrebno da imamo više objekata istog tipa (npr. više restorana) koji imaju različite vrednosti (svaki restoran ima svoje ime, broj stolova, itd.)
- U tom slučaju bi bilo korisno da napravimo funkciju koja će praviti objekat restorana od vrednosti koje joj prosledimo
- Ovakva funkcija se zove **konstruktor** (eng. constructor)
- Primetimo da se za sve restorane na isti način računa broj slobodnih stolova, tako da je funkcija `proveriSlobodne` ista za sve restorane

```
  function Restoran(ime, brStolova, zauzeto, cigarete, jela) {
  this.ime = ime;
this.brojStolova = brStolova;
this.zauzeto = zauzeto;
this.cigarete = cigarete;
this.jela = jela;
   this.proveriSlobodne = function() {
return this.brojStolova – this.zauzeto;
};
  }
```

Javascript - objekti

- Pomoću konstruktor funkcije nov objekat pravimo koristeći ključnu reč new
- Praksa je da konstruktor funkcije imaju prvo slovo veliko

```
var italijanski = new Restoran('Mia', 30, 0, true, ['pizza', 'pasta',
'lazanje']);

var meksicki = new Restoran('Huros', 20, 0, false, ['nacos', 'burito',
'paelja']);
```

- Ako želimo da napravimo prazan objekat, pa naknadno da mu dodajemo svojstva i metode to možemo učiniti:

```
var prazan1 = {};
var prazan2 = new Object();
```

Javascript – `this` objekat

- `this` je ključna reč koji se često koristi unutar funkcija
- Predstavlja objekat unutar kog je funkcija napravljena tako da možemo pristupati svojstvima i metodima tog objekta unutar funkcije
- Ukoliko je funkcija napravljena globalno, a ne unutar nekog objekta, onda `this` predstavlja globalni objekat `window` koji sadrži sve globalne promenljive i funkcije koje smo napravili

```
var cena = 125;
function izracunajCenu(kolicina) {
    return kolicina * this.cena;
}
```

Javascript - objekti

```
var cement = {
    cena: 125,
    izracunajCenu: function(kolicina) {
        return kolicina * this.cena;
    }
}
```

Javascript - objekti

- Možemo napraviti niz objekata ili u objekat sačuvati niz

```
var slatkisi = [
{ime: ‘Bananica’, cena: 15},
{ime: ‘Keks’, cena: 50},
{ime: ‘Cokolada’, cena: 100}
];

var proizvod = {
ime: ‘Parfem’,
   tipovi: [‘cvetni’, ‘vocni’, ‘orjentalni’]
};
```

- Takođe, objekat može sadržati drugi objekat

```
var firma = {
ime: ‘Best’,
   vlasnik: {
ime: ‘Pera’,
prezime: ‘Peric’
}
};
```

Javascript – built-in objekti

- Objekti koje sami pravimo treba da modeluju podatke koji su specificni za potrebe našeg programa
- Međutim postoje neki podaci koji su nam cesto potrebni u različitim programima (npr. sirina prozora,)
- Browseri poseduju već gotove objekte koje možemo koristiti u našem programu i oni se zovu **built-in** objekti
- Možemo ih podeliti u tri grupe:
  - Browser object model
  - Document object model
  - Global Javascript objects

Javascript – Browser object model

- Na raspolaganju nam je objekat `window`
- Neka svojstva objekta `window` su:
  - `window.location` – URL trenutno otvorene web stranice
  - `window.history` – objekat koji čuva detalje o stranicama koje je korisnik posećivao
    - `window.history.length` – broj posećenih stranice
  - `window.screen` – objekat koji modeluje ekran
    - `window.screen.width` – širina ekrana u pikselima (px)
    - `window.screen.height` – visina ekrana u pikselima (px)

- Neki metodi objekta su:
  - `window.alert(poruka)` – ispisuje poruku u prozorčiću
  - `window.open(url)` – otvara stranicu sa zadatim URLom

- Spisak svih svojstava i metoda se može pronaći u literaturi

Javascript – Document Object Model

- Modeluje web stranicu koju smo otvorili u browser
- Svi HTML tagovi su predstavljeni pojedinačnim objektima i nalaze se u objektu `document`
- Neka svojstva objekta `document` su:
  - `document.title` – naslov dokumenta
  - `document.lastModified`– datum poslednje izmene
- Neki metodi objekta su:
  - `document.getElementById(id)` – dohvata objekat koji ima prosleđen id
  - `document.createElement(element)` – pravi novi objekat
  - `document.querySelectorAll(selektor)` – vraća niz objekata koji zadovoljavaju prosleđen selektor

- Ovaj objekat ćemo detaljnije učiti kasnije
- Spisak svih svojstava i metoda se može pronaći u literaturi

Javascript – Global objects

- Neki globalni objekati koji su nam dostupni su:
  - String
  - Number
  - Math
  - Date

- Poseduju korisna svojstva i metode
- Detaljnije ćemo ih učiti kasnije

Javascript – naredba grananja IF

- Često se susrećemo sa situacijama gde želimo da izvršimo jedan zadatak ukoliko važi neki uslov, a drugi zadatak ukoliko taj uslov ne važi
- Npr. kupujemo namirnice u prodavnici, ukoliko imamo novca, moćićemo da platimo, inaće nećemo moći
- Ovo zovemo **grananje** i implementiramo ga pomoću naredbe **if** na sledeći način:

```
if(uslov) {
   // ukoliko je uslov tacan izvsavace se naredbe iz ovog bloka
}
else {
   // inace ce se izvrsavati naredbe iz ovog bloka
}
```

Javascript - naredba grananja IF

- Možemo imati više različitih grananja:

```
if(uslov1) {
    // ukoliko je uslov1 tacan izvsavace se naredbe iz ovog bloka
}
else if(uslov2) {
    // inace, ukoliko je uslov2 tacan izvsavace se naredbe iz ovog bloka
}
else if(uslov3) {
    // inace, ukoliko je uslov3 tacan izvsavace se naredbe iz ovog bloka
}
...
else {
    // inace ce se izvrsavati naredbe iz ovog bloka
}
```

- Primetimo da će se izvršavati tačno jedan od ovi blokova naredbi, prvi blok čiji uslov je zadovoljen
- Npr. ukoliko su zadovoljeni i *uslov2* i *uslov3* izvršavaće se **samo** one naredbe iz drugog bloka jer se prvo ispitalo da li je *uslov2* tačan, i pošto jeste, ušlo se u tu granu programa

Javascript - naredba grananja IF

- *`uslov`* je izraz koji kada se izračuna daje rezultat `true` ili `false`
- Ovakvi izrazi se zovu **boolean expressions** zato što je rezultat njihovih izračunavanja vrednost boolean tipa
- Uglavnom taj izraz bude neko poređenje
- Npr. ukoliko je broj dana u godini jednak 365 onda godina nije prestupna, inace jeste prestupna
- Za poređenje koristimo **operatore poređenja**

Javascript – operatori poređenja

`==` jednakost `'hello' == 'ok'` daje rezultat `false`
`!=` nejednakost `'hello' != 'ok'` daje rezultat `true`
`>` veće `9 > 4` daje rezultat `true`
`<` manje `5 < 3` daje rezultat `false`
`>=` veće ili jednako `5 >= 5` daje rezultat `true`
`<=` manje ili jednako `6 <= 1` daje rezultat `false`

- poželjno je koristiti striktne operatore jednakosti i nejednakosti, koji porede ne samo vrednost, već i tip

`===` striktna jednakost `9 === 9` daje rezultat `true`
`9 === '9'` daje rezultat `false` jer je `9` broj a `'9'` string, dok
`9 == '9'` daje rezultat `true`

`!==` striktna nejednakost `9 !== '9'` daje rezultat `true`

Javascript – operatori poređenja

- Izrazi koje poredimo operatorima poređenja ne moraju nužno da budu jedna vrednost, npr:

```
(cenaKeks + cenaMleko + cenaJaja) <= (novacMuz + novacZena)
```

- Rezultat poređenja možemo sačuvati u promenljivu

```
var nemaPobednika = rezultatJovan == rezultatLazar
```

Javascript – logički operatori

- Ponekad želimo da proverimo da li važi više uslova u istom trenutku
- Npr. imamo x i y koordinatu neke tačke, želimo da proverimo da li je tačka u centru koordinatnog sistema ili nije, tj. želimo da proverimo da li je x koordinata jednaka 0 i y koordinata jednaka 0

`&&` operator *i* `A && B` daje rezultat `true` ukoliko su oba `A` i `B` tačni,
inače je `false`

`||` operator *ili* `A || B` daje rezultat `false` ukoliko su oba `A` i `B` netačni,
inače je `true`

`!` operator *ne* `!A` daje rezultat `true` ako je `A` netačno,
inače je `false`

Javascript – naredba SWITCH

```
switch(vrednost) {
// ukoliko je vrednost jednaka vrednost1 izvsavace se naredbe iz ovog bloka
case vrednost1: {

        break;
    }
// ukoliko je vrednost jednaka vrednost2 izvsavace se naredbe iz ovog bloka
case vrednost2: {

        break;
    }
…
// ukoliko vrednost nije jednaka nijednoj prethodnoj izvsavace se naredbe iz ovog
bloka
default: {

        break;
}}
```

Javascript – konverzija tipova

- Ukoliko pokušamo da iskoristimo vrednost nekog tipa koji nije očekivan (npr. `'1' > 0`) Javascript će pokušati da konvertuje vrednost neočekivanog tipa u očekivan tip (tako da će se `'1'` konvertovati u `1` i rezultat će biti `true`)
- Programski jezici koji imaju ovakva svojstva se zovu **slabo tipizirani** (eng. weakly (loosely) typed)
- Ovakvo ponašanje nije uvek dobro i može dovesti do grešaka u programu, tako da je dobra praksa koristiti striktne operatore `===` i `!==`
- Pomenimo jos par tipova:
  - `undefined` – promenljiva koja je deklarisana ali joj nije dodeljena vrednost
  - `null` – prazna vrednost
  - `NaN` – broj koji označava da je došlo do nevalidne operacije (npr. rezultat `'deset'/2`)
  - `Infinity -` broj koji označava beskonačno (npr. rezultat 2/0)

Javascript – truthy i falsy vrednosti

- Prilikom konverzije u Boolean tip, vrednosti delimo na one koje se konvertuju u `true` (eng. truthy) i one koje se konvertuju u `false` (eng. falsy)
- Truthy:
  - `true`
  - brojevi različiti od `0`
  - stringovi koji nisu prazni

    `'abc', 'false', 'true'`
- Falsy:
  - `false`
  - broj `0`
  - prazan string `''`
  - `NaN` od `12/'a'`
  - `undefined`

    `var x`

Javascript – petlje (eng. loops)

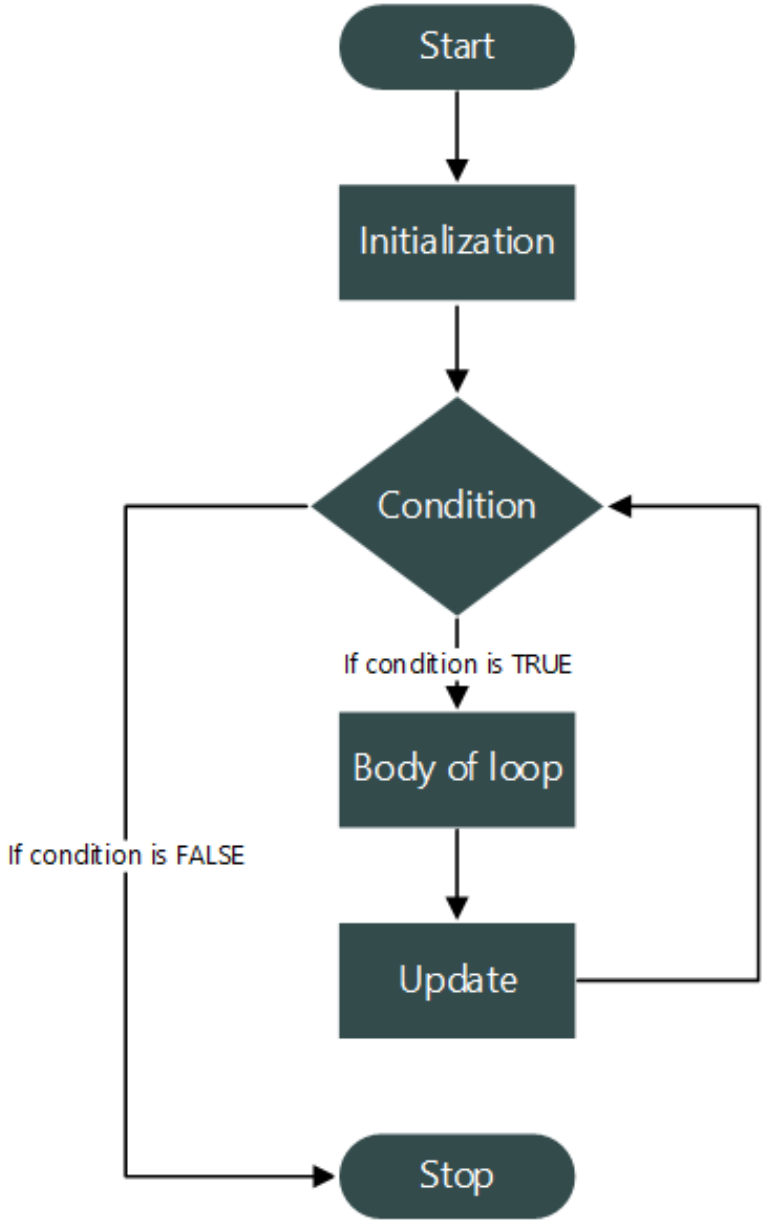
- Često nam je potrebno da neki zadatak ponovimo više puta
- Npr. slanje email svim zaposlenima, brisanje svih komentara
- Odnosno potrebno je izvršavamo isti zadatak sve dok je neki uslov ispunjen
- Npr. šaljemo email svakom elementu iz niza u kome čuvamo podatke o zaposlenima sve dok nismo poslali email svima, tj. prošli kroz ceo niz
- Ovakvi zadaci se implementiraju korišćenjem posebnih naredbi koje se nazivaju **petlje**
- Postoje tri naredbe petlji `for`, `while` i `do while`
- Razlika u ovim naredbama je samo redosled izvršavanja, tj. svaki zadatak koji možemo da rešimo pomoću jedne, možemo i pomoću ostalih petlji

Javascript – F0R petlja

```
for( 1. inicijalizacije ; 2. uslov ; 4. ažuriranje) {
    // 3.a telo petlje (eng. body)
}
// 3.b izlazak iz petlje
```

- `inicijalizacije` – naredbe koje se izvršavaju tačno jednom pre nego što se uđe u petlju
- `uslov` – izraz kao kod `if` naredbe, ispituje se da li je njegova vrednost `true` i ukoliko jeste, ući će se u telo petlje, ukoliko je vrednost `false` izaći će se iz petlje
- `telo petlje` – naredbe koje se izvršavaju kada se uđe u petlju, nakon što se izvrše sve naredbe iz tela petlje, izvršiće se `ažuriranje`
- `ažuriranje` – naredbe koje se izvršavaju svaki put kad se sve naredbe iz tela petlje završe
- Redosled je sledeći:
  - na početku se izvrši `1.`
  - nakon toga se ispituje vrednost `2.`
    - ako je `true` izvršava se `3.a`
      - nakon toga se izvršava `4.` i ponovo ispituje vrednost `2.` i ako je tačno ponovo se izvršava `3.a` pa `4.` i ponovo ispituje vrednost `2.`, i tako sve dok vrednost `2.` ne bude `false`
    - ako je `false` izvršava se `3.b` i izlazi iz petlje i nastavlja dalje sa programom

Javascript - F0



Javascript - F0R petlja

- Primer: želimo da ispišemo brojeve od 0-5 korišćenjem `for` petlje

```
for(var i = 0; i < 5; i++) {
console.log(i);
}
```

- Redosled izvršavanja:

1. `var i = 0` – napravili smo brojač `i` kojim ćemo brojati koliko puta smo ispisali vrednost i postavili ga na `0`
2. `i < 5` – ispitujemo da li je brojač manji od `5,` ako jeste idemo na korak 3., ako nije idemo na korak 5.
3. `console.log(i)` – ispisujemo vrednost brojača
4. `i++` – povećavamo brojač za jedan i vraćamo se na korak 2.
5. Izlazimo iz petlje

Javascript - F0R petlja

`i = 0`

`i < 5` – da li je `0 < 5`? Jeste.

`console.log(i)` – Ispisuje `0`

`i++` – `0++` je `1, i = 1`

`i < 5` – da li je `1 < 5`? Jeste.

`console.log(i)` – Ispisuje `1`

`i++` – `1++` je `2, i = 2`

`i < 5` – da li je `2 < 5`? Jeste.

`console.log(i)` – Ispisuje `2`

`i++` – `2++` je `3, i = 3`

`i < 5` – da li je `3 < 5`? Jeste.

`console.log(i)` – Ispisuje `3`

`i++` – `3++` je `4, i = 4`

`i < 5` – da li je `4 < 5`? Jeste.

`console.log(i)` – Ispisuje `4`

`i++` – `4++` je `5, i = 5`

`i < 5` – da li je `5 < 5`? Nije, završavamo petlju

Javascript - WHILE petlja

```
while(1. uslov) {
    // 2.a telo petlje (eng. body)
}
// 2.b izlazak iz petlje
```

- Redosled:
  - ispituje se `1. uslov`
  - ukoliko je `true` izvrsava se `2.a` i ponovo se ispituje `1. uslov`
  - ukoliko je `false` izlazi se iz petlje `2.b`

Javascript - WHILE petlja

- Primer: želimo da ispišemo brojeve od 0-5 korišćenjem `while` petlje

```
var i = 0;
while(i < 5) {
      console.log(i);
      i += 1;
  }
```

- Redosled izvršavanja:

`var i = 0` – napravili smo brojač `i,` pre nego što smo ušli u petlju, kojim ćemo brojati koliko puta smo ispisali vrednost i postavili ga na `0`

`i < 5` – ispitujemo da li je brojač manji od `5`

ako jeste idemo izvršavamo telo petlje

`console.log(i)` – ispisujemo vrednost brojača

`i += 1` – povećavamo brojač za jedan i vraćamo se na korak 2.

ako nije izlazimo iz petlje

Javascript - DO WHILE petlja

```
do {
// 1. telo petlje (eng. body)
} while(2. uslov);
// 3 izlazak iz petlje
```

- Redosled:
  - izvrsava se `1. telo petlje`
  - ispituje se `2. uslov`
    - ukoliko je `true` izvrsava se `1. telo petlje` ponovo i opet se ispituje `2. uslov`
    - ukoliko je `false` izlazi se iz petlje
- do while petlja se razlikuje od po tome što se telo petlje izvrši sigurno jedanput

Javascript - Nizovi

- Nizovi su objekti koji se često koriste u kombinaciji sa petljama
- Npr. Ispisivanje svih elemenata nekog niza

```
var niz = [1,2,3,4,5];
for(var i = 0; i < niz.length; i++) {
        console.log(niz[i]);
    }
```

- ili

```
var niz = [1,2,3,4,5];
var i = 0, duzina = niz.length;
while(i < duzina) {
        console.log(niz[i]);
        i += 1;
    }
```

Javascript - Nizovi

- Neke korisne metode za rad sa nizovima:
  - `niz.push(vrednost)` – dodaje prosleđenu vrednost na kraj niza
  - `niz.pop()` – uklanja element sa kraja niza i vraća ga kao povratnu vrednost
  - `niz.indexOf(vrednost)` – vraća indeks prvog pojavljivanja prosleđene vrednosti, -1 ukoliko vrednost ne postoji u nizu
  - `niz.join(separator)` – spaja sve elemente niza u string, koristeći prosleđen separator (string), ukoliko separator nije prosleđen, spaja elemente sa zarezom ,
  - `niz1.concat(niz2)` – spaja dva niza, `niz1` sa `niz2`
  - `niz.slice(poc_indeks, kraj_indeks)` – vraća deo niza od `poc_indeks` do `kraj_indeks`
  - `niz.sort()` – sortira elemente u nizu leksikografski
  - `niz.sort(poredi_func)` – možemo proslediti našu funkciju poređenja na osnovu koje će se sortirati elementi

Javascript - Nizovi

- `poredi_func` treba da izgleda ovako:

```
function poredi_func(elem1, elem2) { }
```

- funkcija treba da uporedi dva elementa iz niza i da vrati rezultat koji će da bude 1, 0 ili -1
- ako `elem1` treba da se nalazi pre `elem2` u sortiranom nizu, onda funckija treba da vrati 1
- ako `elem2` treba da se nalazi pre `elem1` u sortiranom nizu, onda funckija treba da vrati -1
- ako su elementi jednaki, tj. ako nije važno koji će element biti prvi u sortiranom nizu, onda funkcija treba da vrati 0
- `sort` će porediti elemente niza koristeći našu funkciju i sortirati ih u tom poretku

Javascript – Nizovi

- `niz.find(uslov_func)` – pronalazi prvi element koji zadovoljava zadati uslov, vraća `undefined` ako ne postoji
- Uslov je naša funkcija koja treba da izgleda ovako:

```
function uslov_func(element) { }
```

- Ukoliko `element` zadovoljava naše kriterijume, funkcija treba da vrati `true` inače `false`

Javascript - Nizovi

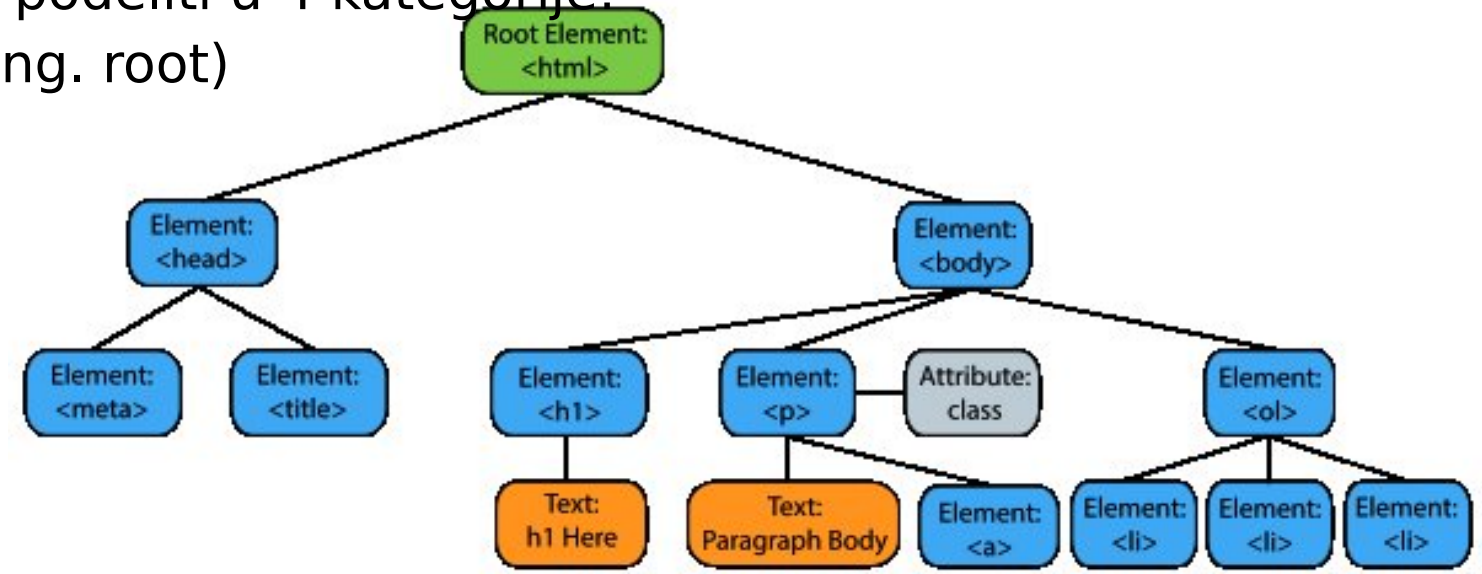
- `niz.filter(uslov_func)` – izdvaja sve elemente koji zadovoljavaju zadati uslov, vraća prazan niz (`[]`) ako ne postoji nijedan element
- Uslov je naša funkcija koja treba da izgleda ovako:

```
function uslov_func(element) { }
```

- Ukoliko `element` zadovoljava naše kriterijume, funkcija treba da vrati `true` inače `false`

Javascript – Document Object Model

- Kada browser učita web stranicu, u pozadini se konstruise objekat DOM drveta
- To je objekat koji predstavlja HTML strukturu stranice, svaki HTML tag je predstavljen objektom
- Definisana su svojstva i metode pomoću kojih možemo pristupati, menjati, brisati, dodavati objekte u drvo
- Objekte DOM drveta možemo podeliti u 4 kategorije:
  - `document` objekat, koren (eng. root)
  - elementi (tagovi)
  - atributi
  - tekst



Javascript - Document Object Model

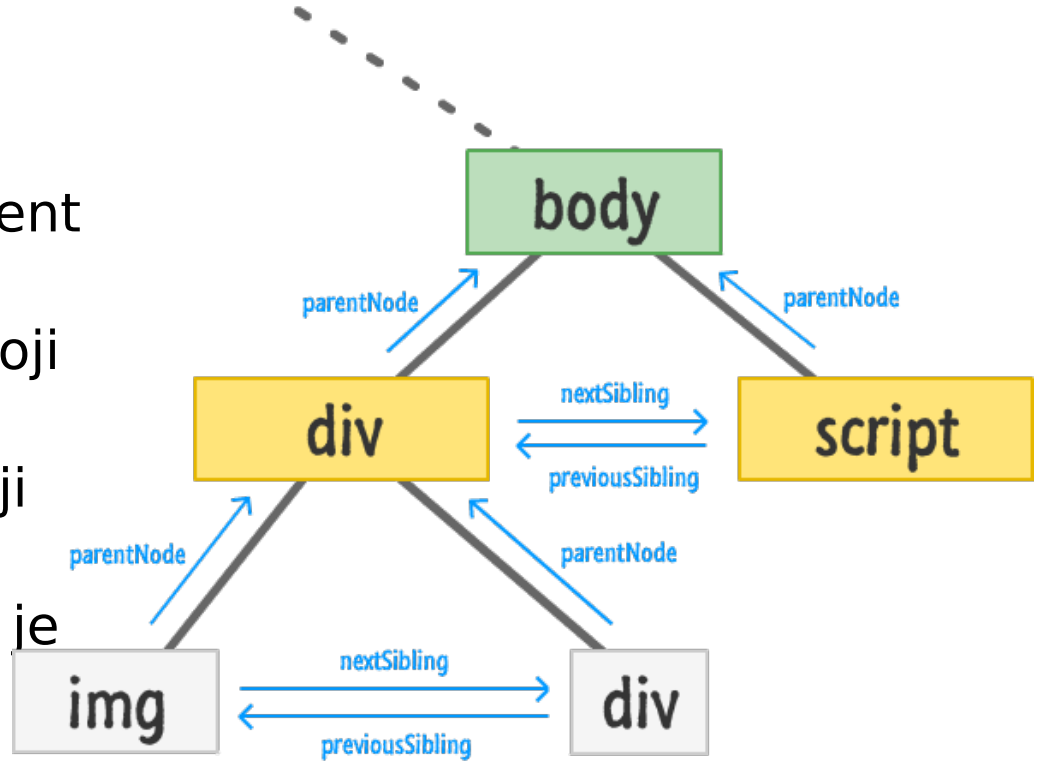
- `document` objekat je koren (početak) drveta i predstavlja celu HTML stranicu (sadržaj koji se nalazi između html tagova
- Elementi (tagovi) su objekti koji se konstruišu za svaki HTML tag i nalaze se u DOM drvetu
- Atributi su objekti koji se konstruišu za svaki atribut koji element sadrži
- Tekstualni objekti predstavljaju tekstualni sadržaj elementa, ukoliko on postoji
- Svakom od objekata DOM drveta možemo pristupiti, ažurirati ga, obrisati, dodati nov objekat i slično

Javascript - Document Object Model

- DOM objektima možemo pristupiti koristeći metode koje se nalaze u `document` objektu
- Metode koje dohvataju jedan objekat:
  - `document.getElementById(id)` – dohvata element sa prosledjenom vrednošću atributa id
  - `document.querySelector(selektor)` - dohvata prvi element koji zadovoljava zadat CSS selektor
- Metode koje dohvataju niz objekata:
  - `document.getElementsByClassName(ime_klase)` - dohvata sve elemente koji imaju prosleđenu vrednost atributa `class`
  - `document.getElementsByTagName(ime_taga)` - dohvata sve elemente sa datim tagom
  - `document.querySelectorAll(selektor)` - dohvata sve elemente koji zadovoljavaju zadat CSS selektor
- Kada dohvatimo neki element ili niz elemenata, dobra praksa je da ih sačuvamo u promenljivu kako bismo mogli da ih koristimo kasnije kada nam zatrebaju
- Dohvatanje elemenata je skupa operacija, može oduzimati dosta vremena i usporavati program, zato se trudimo da je koristimo što manje možemo

Javascript - Document Object Model

- Možemo se kretati kroz DOM drvo (prolaziti njegove elemente)
  - `element.parentNode` – dohvata roditeljski element
  - `element.previousSibling` – dohvata element koji je prethodni rođak datom elementu
  - `element.nextSibling` – dohvata element koji je sledeći rođak datom elementu
  - `element.firstChild` – dohvata element koji je prvo dete datog elementa
  - `element.lastChild` – dohvata element koji je poslednje dete datog elementa



Javascript - Document Object Model

- Elemente možemo ažurirati
- Svojstva:
  - `element.innerHTML` – HTML sadržaj elementa
  - `element.textContent` – tekstualni sadržaj elementa
  - `element.className` – vrednost atributa `class`
  - `element.id` – vrednost atributa `id`
- Metodi:
  - `element.createElement(ime_taga)` – konstruiše objekat elementa
  - `element.createTextNode(tekst)` – konstruiše tekstualni objekat
  - `element.appendChild(element)` – dodaje dete element u DOM drvo
  - `element.removeChild(element)` – uklanja dete elementa iz DOM drveta
  - `element.hasAttribute(ime_atributa)` – ispituje da li postoji atribut
  - `element.getAttribute(ime_atributa)` – dohvata atribut
  - `element.setAttribute(ime_atributa, vrednost_atributa)` – postavlja vrednost atributa
  - `element.removeAttribute(ime_atributa)` – uklanja atribut

- Primetimo da ažuriranje elemenata DOM drveta ne menjamo sadržaj HTML stranice koja se nalazi na serveru, ona ostaje na serveru kakva je i bila ranije, ažuriranjem DOMa menjamo prikazivanje stranice u browseru

Javascript – String global object

- Nad svakim objektom koji je tipa String možemo koristiti neka svojstva i metode globalnog String objekta
- Najčešće korišćeno svojstvo je `length`
- Neki korisni metodi su:
  - `toUpperCase()` – pretvara sva slova u velika
  - `toLowerCase()` – pretvara sva slova u mala
  - `charAt(`*`indeks`*`)` – vraća slovo (karakter) koji se nalazi na prosleđenom indeksu
  - `indexOf(`*`podstring`*`)` – vraća indeks prvog pojavljivanja prosleđenog podstringa u našem stringu, -1 ako podstring ne postoji u našem
  - `lastIndexOf(`*`podstring`*`)` – vraća indeks poslednjeg pojavljivanja prosleđenog podstringa u našem stringu, -1 ako podstring ne postoji u našem
  - `substring(`*`poc_indeks, kraj_indeks`*`)` – vraća podstring karaktera od *`poc_indeks`* do *`kraj_indeks`*
  - `split(`*`string`*`)` – razlaže string na niz podstringova koji su bili razdvojeni prosleđenim stringom
  - `trim()` – briše beline sa početka i kraja stringa
  - `replace(`*`podstring1, podstring2`*`)` – zamenjuje *`podstring1`* podstringom *`podstring2`*

Javascript – Number global object

- Vrednosti tipa Number mogu biti **celi** (eng. integer) i **decimalni** brojevi (eng. float)
- Neke korisne metode Number globalnog objekta su:
  - `Number.isNaN(broj)` – ispituje da li je broj NaN
  - `Number.isInteger(broj)` – ispituje da li je broj ceo tj. integer
  - `Number.parseInt(string)` – od stringa pravi integer broj ukoliko je to moguće
  - `Number.parseFloat(string)` – od stringa pravi float broj ukoliko je to moguće

- Nad objektima koji su tipa Number možemo pozvati metod `toString()` koji vraća string napravljen od našeg objekta

Javascript – Math globalni objekat

- Ima neka korisna svojstva i metode za matematičke funkcije
  - `Math.PI` – vrednost broja PI (3.14159...)
  - `Math.round(`*`broj`*`)` – zaokružuje decimalni broj na najbliži ceo broj
  - `Math.sqrt(`*`broj`*`)` – vraća kvadratni koren broja
  - `Math.random()` – vraća slučajan broj između 0 i 1
  - `Math.abs(`*`broj`*`)` – vraća apsolutnu vrednost broja
  - `Math.log(`*`broj`*`)` – vraća logaritam broja
  - `Math.min(`*`niz_brojeva`*`)` – vraća najmanji broj iz niza brojeva
  - `Math.max(`*`niz_brojeva`*`)` – vraća najveći broj iz niza brojeva

Javascript – Date globalni objekat

- Da bismo napravili objekat datuma koristimo `Date()` konstruktor funkciju
- `var datum = new Date();`
- Ovaj objekat sadrži i trenutni datum i vreme, možemo i zadati vrednosti za datum i vreme `Date(godina,mesec,dan,sati,minuti,sekunde)`
- Korisne metode Date objekta su:
  - `getHours()` – vraća broj sati
  - `setHours(sati)` – postavlja broj sati (0-23)
  - `getMinutes()` – vraća broj minuta
  - `setMinutes(minuti)` – postavlja broj minuta (0-59)
  - `getSeconds()` – vraća broj sekundi
  - `setSeconds(sekunde)` – postavlja broj sekundi (0-59)
  - `getDate()` – vraća dan u mesecu
  - `setDate(dan)` – postavlja dan u mesecu (1-31)
  - `getMonth()` – vraća broj meseca
  - `setMonth(mesec)` – postavlja broj meseca
  - `getFullYear()` – vraća broj godine
  - `setFullYear(godina)` – postavlja broj godine
  - `toString()` – vraća string reprezentaciju datuma

Javascript – Događaji (eng. events)

- Često želimo da reagujemo na neke događaje (eng. events) kako bismo napravili našu stranicu interaktivnom
- Npr. kada korisnik klikne na dugme *'Next'* želimo da mu otvorimo sledeću stranicu
- Klik na dugme je u ovom slučaju događaj i želimo da na neki način kažemo browseru da kada se ovaj događaj, treba da se otvori sledeća stranica
- Odnosno, možemo napraviti funkciju koja će da se izvrši svaki put kada se desi neki događaj
- Postoji već predefinisana lista događaja na koje možemo reagovati, neki od njih su klik miša, klik dugmeta na tastaturi, skrolovanje stranice (eng. scroll) gore-dole, levo-desno,...
- Terminologija:
  - kada se događaj desi kazemo da je **opaljen** (eng. event is **fired**, event is **raised**)
  - funkcija koja se izvršava nakon što je događaj opaljen se naziva eng. **event handler** ili **event listener**
  - izvršavanje ove funkcije nazivamo **okidanje** (eng. function is **triggered** by event)

Javascript – Događaji (eng. events)

- Postoje događaji koji se odnose na jedan ili više HTML elementa (klik miša na neki konkretan link) i događaji koji se odnose na celu stranicu (učitavanje stranice)
- Potrebno je da nekako zadamo funkciju (event handler) koja će se izvršiti kada se događaj opali (event fires)
- Ovo se zove **vezivanje** funkcije (eng. **binding** event handler to event)
- Ovo možemo uraditi na više načina, mi ćemo naučiti jedan način koji je najbolja praksa, a ostale ćemo samo pokazati
- Primer: želimo da promenimo boju teksta kada se klikne na dugme *‘Promeni’*
- Potrebno je da postavimo funkciju koja će se izvršavati kada se desi klik na dugme *‘Promeni’* i koja će menjati boju teksta nekom paragrafu

Javascript - Događaji (eng. events)

- Ovako vezujemo event handler:

```
dom_element.addEventListener(ime_događaja, ime_funkcije)
```

- *dom_element* – DOM objekat nad kojim se događaj okida
- *ime_događaja* – ime događaja na koji reagujemo
- *ime_funkcije* – funkcija koja se izvršava kada se događaj okine

- U našem slučaju
  - *dom_element* će biti dugme *'Promeni'*
  - *ime_događaja* će biti dugme *'click'*
  - *ime_funkcije* će biti recimo *promeniBoju*

- Ukoliko event handler funkciju nećemo nigde više koristiti u programu, možemo napraviti anonimnu funkciju

Javascript - Događaji (eng. events)

```
// dohvatamo DOM objekat dugmeta
var dugmePromeni = document.getElementById('promeni');

// pravimo funkciju za promenu boje
function promeniBoju() {

// dohvatamo paragraf
var paragraf = document.getElementById('parag1');

// u niz CSS klasa dodajemo klasu koja menja boju teksta u crveno
paragraf.classList.add('crveni-tekst');
}

// vezujemo funkciju promeniBoju na događaj click nad dugmetom promeni
dugmePromeni.addEventListener('click', promeniBoju);
```

Javascript - Događaji (eng. events)

- Ukoliko koristimo anonimnu funkciju

```
// dohvatamo DOM objekat dugmeta
var dugmePromeni = document.getElementById('promeni');

// vezujemo anonimnu funkciju na događaj click nad dugmetom promeni
dugmePromeni.addEventListener('click', function() {

// dohvatamo paragraf
var paragraf = document.getElementById('parag1');

// u niz CSS klasa dodajemo klasu koja menja boju teksta u crveno
paragraf.classList.add('crveni-tekst');

});
```

Javascript - Događaji (eng. events)

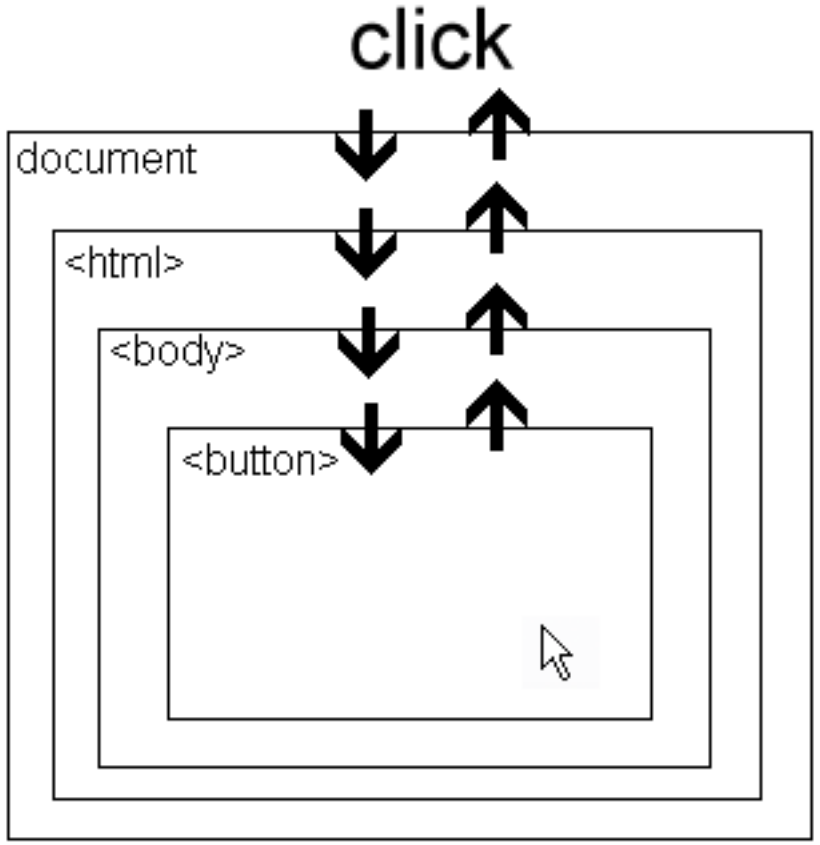
- Event handler funkciji možemo prosleđivati argumente ukoliko su nam potrebni
- Prvi argument po pravilu je event objekat a ostali argumenti mogu biti naši
- Često se event objektu daje ime `e` ili `ev`

```
element.addEventListener(event, function(e, arg1, arg2, ...) { })
```

- Event objekat čuva podatke o tome koji događaj se desio, nad kojim elementom, i sl.
- Neka svojstva:
  - `e.target` – element nad kojim se desio događaj
  - `e.type` – tip događaja
- Neki metodi:
  - `e.preventDefault()` – sprečava da se izvrši default ponašanje
  - `e.stopPropagation()` – sprečava propagiranje događaja na roditelje i decu elementa nad kojim se događaj desio

Javascript - Događaji (eng. events)

- HTML elementi su ugnježdeni jedni u druge zbog čega se dešava da kada, recimo, kliknemo na neko dugme, kliknuli smo na sve njegove roditelje
- Ukoliko postoje funkcije event handler za isti događaj nad svim elementima u nizu, redosled kojim će se izvršavati može biti u dva smera
- Prvi smer je da se prvo izvrši funkcija event handler nad poslednjim detetom (na slici pored je to dugme) pa se onda izvrši nad njegovim roditeljem (body) i tako dalje (eng. **event bubbling**)
- Drugi smer je da se prvo izvrši funkcija nad prvim roditeljem pa onda nad decom redom (eng. **event capturing**)
- Smer u kome će se propagirati događaji se postavlja kao treći (opcioni) argument funkcije `addEventListener(event, func, smer)`
  - `true` – event capturing
  - `false` – event bubbling



Javascript - Događaji (eng. events)

- Neki događaji koji se često koriste:
  - ‘click’ – korisnik je kliknuo na taster na mišu
  - ‘mousedown’ – korisnik je pritisnuo taster na mišu
  - ‘mouseup’ – korisnik je pustio pritisnut taster na mišu
    - ova tri događaja se dešavaju kada korisnik klikne na taster miša u sledećem redosledu ‘mousedown’, ‘click’, ‘mouseup’
  - ‘mousemove’ – korisnik je pritisnuo taster miša, nije ga pustio i pomera miš
  - ‘keydown’ – korisnik je pritisnuo dugme na tastaturi
  - ‘keyup’ – korisnik je pustio pritisnuto dugme na tastaturi
  - ‘keypress’ – korisnik je kliknuo dugme na tastaturi
    - ova tri događaja se dešavaju kada korisnik klikne na dugme na tastaturi u sledećem redosledu ‘keydown’, ‘keypressed’, ‘keyup’
  - ‘focus’ – korisnik je fokusirao element (kliknuo na njega, strelicama na tastaturi došao do njega, i sl)
  - ‘blur’ – element je izgubio fokus
  - ‘load’ – stranica se učitala

Javascript - Timers

- Postoje funkcije koje mogu da zakažu kada će da se izvrši neka funkcija
  - Recimo hoćemo da prikažemo veliku reklamu 1 minut nakon što je korisnik došao na našu stranicu

`setTimeout(ime_funkcije, br_milisek)`
  - zakazuje izvršavanje funkcije `ime_funkcije` nakon `br_milisek` milisekundi

`clearTimeout(ime_funkcije)`
  - otkazuje izvršavanje funkcije `ime_funkcije`

`setInterval(ime_funkcije, br_milisek)`
  - zakazuje izvršavanje funkcije `ime_funkcije` nakon svakih `br_milisek` milisekundi

`clearInterval(ime_funkcije)`
  - otkazuje izvršavanje funkcije `ime_funkcije`