NAČRTOVANJE IN RAZVOJ SPLETNIH APLIKACIJ

[Pritegnite pozornost bralca z zanimivim povzetkom. Običajno je to kratek povzetek dokumenta. Ko boste želeli dodati vsebino, kliknite tukaj in začnite tipkati.]

LUR

Vsebina

[Uvod 10](#_Toc50575034)

[ZGODOVINA INTERNETA 10](#_Toc50575035)

[INTERNET DANES 11](#_Toc50575036)

[ORGANIZACIJE INTRANETA 12](#_Toc50575037)

[INTRANET 12](#_Toc50575038)

[EKSTRANET 13](#_Toc50575039)

[INTERNETOVA IMENA IN NASLOVI IP 14](#_Toc50575040)

[DVOČRKOVNE DOMENE 15](#_Toc50575041)

[DOMENE IN PRESLIKAVE NASLOVOV IP 15](#_Toc50575042)

[SVETOVNI SPLET 16](#_Toc50575043)

[SVETOVNI SPLET 17](#_Toc50575044)

[PRINCIP KOMUNIKACIJE ODJEMALEC (BRSKALNIK) IN SPLETNI STREŽNIK 17](#_Toc50575045)

[PRISTOP AJAX 17](#_Toc50575046)

[PROTOKOL HTTP 18](#_Toc50575047)

[vrstice glave HTTP 18](#_Toc50575048)

[METODE HTTP 18](#_Toc50575049)

[STATUS HTTP 19](#_Toc50575050)

[EVOLUCIJA SPLETNIH APLIKACIJ 19](#_Toc50575051)

[STATIČNE SPLETNE APLIKACIJE 19](#_Toc50575052)

[STREŽNIŠKI PROGRAMI CGI 20](#_Toc50575053)

[STREŽNIŠKI MODUL 20](#_Toc50575054)

[INTERPRETIRANI STREŽNIŠKI PROGRAM 21](#_Toc50575055)

[Označevalni jezik HTML5 23](#_Toc50575056)

[ELEMENTI HTML 5 24](#_Toc50575057)

[Head 24](#_Toc50575058)

[<!DOCTYPE HTML> 24](#_Toc50575059)

[Razlike med HTML 4.01, XHTML1 in HTML5. 24](#_Toc50575060)

[ELEMENT HTML 24](#_Toc50575061)

[ELEMENT HEAD 25](#_Toc50575062)

[ELEMENT TITLE 26](#_Toc50575063)

[ELEMENT BASE 27](#_Toc50575064)

[ELEMENT LINK 27](#_Toc50575065)

[ELEMENT META 28](#_Toc50575066)

[ELEMENT STYLE 29](#_Toc50575067)

[ELEMENT SCRIPT 30](#_Toc50575068)

[NOSCRIPT 31](#_Toc50575069)

[Sections 32](#_Toc50575070)

[ body 32](#_Toc50575071)

[ article 32](#_Toc50575072)

[ nav 32](#_Toc50575073)

[ aside 32](#_Toc50575074)

[ section 32](#_Toc50575075)

[ header 32](#_Toc50575076)

[ footer 32](#_Toc50575077)

[ h1-h6 32](#_Toc50575078)

[ hgroup 33](#_Toc50575079)

[ address 33](#_Toc50575080)

[ELEMENT BODY 33](#_Toc50575081)

[ELEMENT ARTICLE 33](#_Toc50575082)

[ELEMENT NAV 34](#_Toc50575083)

[ELEMENT ASIDE 35](#_Toc50575084)

[ELEMENT SECTION 35](#_Toc50575085)

[ELEMENT HEADER 36](#_Toc50575086)

[ELEMENT FOOTER 37](#_Toc50575087)

[ELEMENT H# 37](#_Toc50575088)

[ELEMENT HGROUP 38](#_Toc50575089)

[ELEMENT ADDRESS 39](#_Toc50575090)

[ZNAČKE HTML 5 39](#_Toc50575091)

[GLOBALNI ATRIBUTI HTML 5 41](#_Toc50575092)

[DOGODKI DOKUMENTOV 42](#_Toc50575093)

[DOGODKI S TIPKOVNICO 42](#_Toc50575094)

[DOGODKI OBRAZCEV 42](#_Toc50575095)

[MIŠKINI DOGODKI 43](#_Toc50575096)

[MEDIJSKI DOGODKI 43](#_Toc50575097)

[KAJ JE HTML? 45](#_Toc50575098)

[KAJ JE XML? 45](#_Toc50575099)

[KAJ JE XHTML? 45](#_Toc50575100)

[ZGODOVINA HTML 46](#_Toc50575101)

[HTML 5 46](#_Toc50575102)

[Novi elementi 47](#_Toc50575103)

[HTML 5 47](#_Toc50575104)

[Novi pomensko usmerjeni medvrstični elementi v HTML 5.0 47](#_Toc50575105)

[Interaktivnost 47](#_Toc50575106)

[HTML 5 47](#_Toc50575107)

[Dodatne lastnosti pri znački <input> v obrazcih 47](#_Toc50575108)

[Vključevanje MathML in SVG 48](#_Toc50575109)

[Element 48](#_Toc50575110)

[Male črke 48](#_Toc50575111)

[**Gnezdenje in odprte značke** 49](#_Toc50575112)

[Gnezdenje 49](#_Toc50575113)

[Odprte značke 49](#_Toc50575114)

[Vrste elementov 50](#_Toc50575115)

[Blokovni elementi 50](#_Toc50575116)

[Medvrstični elementi 50](#_Toc50575117)

[Drugi elementi 50](#_Toc50575118)

[Atributi 50](#_Toc50575119)

[Dostopanje in spreminjanje z JavaScriptom 51](#_Toc50575120)

[Pravilno pisanje atributov 52](#_Toc50575121)

[Velikost črk 52](#_Toc50575122)

[Okrajšave 52](#_Toc50575123)

[HTML - STRUKTURA 52](#_Toc50575124)

[Deklariranje dokumenta 52](#_Toc50575125)

[Pregledovanje kode HTML z DTD 53](#_Toc50575126)

[Tipi dokumentov XHTML 53](#_Toc50575127)

[XHTML 1.0 Strict 53](#_Toc50575128)

[HTML5/XHTML5 53](#_Toc50575129)

[**DOM** 54](#_Toc50575130)

[SORODSTVO 54](#_Toc50575131)

[POTOMCI 55](#_Toc50575132)

[PREDNIKI 55](#_Toc50575133)

[STARŠI 55](#_Toc50575134)

[OTROCI 55](#_Toc50575135)

[SOROJENCI 55](#_Toc50575136)

[ZNAKI 55](#_Toc50575137)

[KODNE TABELE 55](#_Toc50575138)

[KODIRANJE ZNAKOV 56](#_Toc50575139)

[Določanje znakov za kodiranje 56](#_Toc50575140)

[KODIRANJE ZNAKOV V URI 57](#_Toc50575141)

[KODIRANJE ZNAKOV V URI 57](#_Toc50575142)

[RELATIVNI URI 57](#_Toc50575143)

[Izračun relativnega URI 57](#_Toc50575144)

[CSS 58](#_Toc50575145)

[ZAKAJ UPORABLJAMO PREKRIVNE SLOGE 58](#_Toc50575146)

[KAKO BO VIDETI STRAN 58](#_Toc50575147)

[Različice 59](#_Toc50575148)

[KAJ JE CSS 59](#_Toc50575149)

[Prekrivni Slogi lahko prihranijo veliko časa 59](#_Toc50575150)

[SINTAKSA 59](#_Toc50575151)

[KOMENTARJI V CSS 60](#_Toc50575152)

[VKLJUČITEV V HTML 61](#_Toc50575153)

[PREKRIVANJE SLOGOV 61](#_Toc50575154)

[PREKRIVANJE 61](#_Toc50575155)

[Kateri slog ima prednost 62](#_Toc50575156)

[1. PRIMER 62](#_Toc50575157)

[2. Primer 62](#_Toc50575158)

[VEČKRAT DOLOČENO 62](#_Toc50575159)

[Primer 62](#_Toc50575160)

[Aktivnost 63](#_Toc50575161)

[PREKRIVANJE IN HTML 63](#_Toc50575162)

[Primer 63](#_Toc50575163)

[Aktivnost 63](#_Toc50575164)

[GLAVA DOKUMENTA HTML 63](#_Toc50575165)

[ZAKAJ POTREUJEMO GLAVO 63](#_Toc50575166)

[DELI GLAVE 64](#_Toc50575167)

[ELEMENT META 64](#_Toc50575168)

[SEZNAMI 64](#_Toc50575169)

[NEUREJEN SEZNAM 64](#_Toc50575170)

[OBLIKOVANJE OZNAK SEZNAMA 65](#_Toc50575171)

[UREJEN SEZNAM 65](#_Toc50575172)

[DODATNE MOŽNOSTI 65](#_Toc50575173)

[reversed 65](#_Toc50575174)

[start 66](#_Toc50575175)

[OBLIKOVANJE OZNAK SEZNAMA 66](#_Toc50575176)

[LEGA OZNAKE 66](#_Toc50575177)

[Skriptni jeziki 67](#_Toc50575178)

[Uporaba skriptnih jezikov 67](#_Toc50575179)

[Različice 68](#_Toc50575180)

[Jezik CSS 68](#_Toc50575181)

[Kaj je CSS 71](#_Toc50575182)

[Prekrivni Slogi lahko prihranijo veliko časa 71](#_Toc50575183)

[Sintaksa 71](#_Toc50575184)

[Komentarji v CSS 72](#_Toc50575185)

[**Vključitev v HTML** 72](#_Toc50575186)

[Prekrivanje 73](#_Toc50575187)

[Kateri slog ima prednost 73](#_Toc50575188)

[Večkrat določeno 73](#_Toc50575189)

[Primer 74](#_Toc50575190)

[Aktivnost 74](#_Toc50575191)

[Prekrivanje in HTML 74](#_Toc50575192)

[Primer 74](#_Toc50575193)

[Pravilo !important 74](#_Toc50575194)

[Primer 74](#_Toc50575195)

[Zakaj potrebujemo glavo 75](#_Toc50575196)

[Deli Glave 75](#_Toc50575197)

[Element Meta 75](#_Toc50575198)

[Neurejen seznam 76](#_Toc50575199)

[**Oblikovanje oznak seznama** 76](#_Toc50575200)

[Urejen seznam 76](#_Toc50575201)

[Dodatne možnosti 76](#_Toc50575202)

[reversed 77](#_Toc50575203)

[start 77](#_Toc50575204)

[**Oblikovanje oznak seznama** 77](#_Toc50575205)

[Lega oznake 77](#_Toc50575206)

[Dodajanje slik 77](#_Toc50575207)

[Atribut alt 78](#_Toc50575208)

[Velikost slike 78](#_Toc50575209)

[Optimizacija slike 78](#_Toc50575210)

[Odzivna velikost 79](#_Toc50575211)

[Slika kot povezava 79](#_Toc50575212)

[Uporaba barv 79](#_Toc50575213)

[Faktor Gama 80](#_Toc50575214)

[Kaj se oblikuje 80](#_Toc50575215)

[Poimenovane barve 80](#_Toc50575216)

[Barve v SGML 81](#_Toc50575217)

[šestnajstiški zapis 81](#_Toc50575218)

[Primer: 81](#_Toc50575219)

[Ena števka za barvo 81](#_Toc50575220)

[Primer: 81](#_Toc50575221)

[Pretvorba 82](#_Toc50575222)

[Zapis rgb( ) 82](#_Toc50575223)

[Primer 82](#_Toc50575224)

[Aktivnost 82](#_Toc50575225)

[Zapis rgbA( ) 82](#_Toc50575226)

[Primer 83](#_Toc50575227)

[Barve HSL 83](#_Toc50575228)

[Primer 83](#_Toc50575229)

[Barve HSLA 83](#_Toc50575230)

[Primer 83](#_Toc50575231)

[Ustvarjanje naslovov 83](#_Toc50575232)

[Velikost pisave 84](#_Toc50575233)

[Velike črke 84](#_Toc50575234)

[**Barva naslovov** 84](#_Toc50575235)

[Krepkost naslovov 84](#_Toc50575236)

[Druge možnosti 85](#_Toc50575237)

[Povezave 85](#_Toc50575238)

[Notranje povezave 85](#_Toc50575239)

[Povezave 85](#_Toc50575240)

[AbsolutnE povezavE 85](#_Toc50575241)

[Relativni URI 86](#_Toc50575242)

[Izračun relativnega URI 86](#_Toc50575243)

[Povezava za e-pošto 86](#_Toc50575244)

[**Oblikovanje povezav** 86](#_Toc50575245)

[**Digitalni zvok** 87](#_Toc50575246)

[**Osnovne lastnosti digitalnega zvoka** 87](#_Toc50575247)

[Formati zvočnih datotek 88](#_Toc50575248)

[Vstaljanje v HTML 89](#_Toc50575249)

[Digitalni video 89](#_Toc50575250)

[Osnovne lastnosti 89](#_Toc50575251)

[Gostota okvirjev 89](#_Toc50575252)

[Vzorčenje 89](#_Toc50575253)

[Osnovne lastnosti 89](#_Toc50575254)

[Gostota podatkov 89](#_Toc50575255)

[Stiskanje 90](#_Toc50575256)

[FORMATI datotek 90](#_Toc50575257)

[format AVI 90](#_Toc50575258)

[format Windows Media format 90](#_Toc50575259)

[format Adobe (Flash) 91](#_Toc50575260)

[format MPEG 91](#_Toc50575261)

[Format QuickTime 91](#_Toc50575262)

[format RealVideo 91](#_Toc50575263)

[format Matroska 91](#_Toc50575264)

[Zaključek 92](#_Toc50575265)

[Obroba 92](#_Toc50575266)

[**Slog obrobe** 92](#_Toc50575267)

[**Širina obrobe** 92](#_Toc50575268)

[**barva obrobe** 93](#_Toc50575269)

[Skrajšani zapis 93](#_Toc50575270)

[Osnovne animacije 93](#_Toc50575271)

[Animacija 93](#_Toc50575272)

[Kaj so animacije? 93](#_Toc50575273)

[**Lastnosti animacij** 94](#_Toc50575274)

[Prehodi 94](#_Toc50575275)

[**Osnovno** 94](#_Toc50575276)

[Delovanje 94](#_Toc50575277)

[Dodatne spremembe 94](#_Toc50575278)

[**Transition lastnosti** 95](#_Toc50575279)

[Border-radius - uvod 95](#_Toc50575280)

[**Kako deluje** 95](#_Toc50575281)

[Box-shadow 95](#_Toc50575282)

[**Osnove lastnosti clip** 96](#_Toc50575283)

[**Sintaksa** 96](#_Toc50575284)

[Rect() funkcija 96](#_Toc50575285)

[Skriptni jezik JavaScript 97](#_Toc50575286)

[Zgradba JavaScript 98](#_Toc50575287)

[Prevajanje dokumentov na strežniku 99](#_Toc50575288)

[Jedro (Core) 99](#_Toc50575289)

[Gostitelj (Host) 99](#_Toc50575290)

[Uporabnik (User) 99](#_Toc50575291)

[Jedro 99](#_Toc50575292)

[Sintaksa 100](#_Toc50575293)

[Stavki 100](#_Toc50575294)

[Aplikacije JavaScript 100](#_Toc50575295)

[Ajax tehnologija 102](#_Toc50575296)

[Knjižnica JQuery za skriptni jezik JavaScript 103](#_Toc50575297)

[Odzivni spletni dizajn 104](#_Toc50575298)

[Bootstrap 106](#_Toc50575299)

[Programski jezik PHP 107](#_Toc50575300)

[MySQL 108](#_Toc50575301)

[Ogrodje CodeIgniter 109](#_Toc50575302)

[Prednosti in namen uporabe ogrodij 109](#_Toc50575303)

[Arhitektura MVC 111](#_Toc50575304)

[Krmilniki 112](#_Toc50575305)

[Pogledi 113](#_Toc50575306)

[Modeli 115](#_Toc50575307)

[Podatkovna baza 116](#_Toc50575308)

[Razredi in funkcije 118](#_Toc50575309)

[Dodatne funkcionalnosti 119](#_Toc50575310)

[Pregled in primerjava ostalih php ogrodij 120](#_Toc50575311)

[Zend Framework 120](#_Toc50575312)

[CakePhp 120](#_Toc50575313)

[Symfony 121](#_Toc50575314)

[Yii framework 121](#_Toc50575315)

[Kako narediti spletno stran, del 1 122](#_Toc50575316)

[Korak 1: Pomislite na Privlačen ime 122](#_Toc50575317)

[Korak 2: Najti gostitelja 123](#_Toc50575318)

[Vprašanja za 2. izpitno enoto poklicne mature Strokovni predmet NPA 124](#_Toc50575319)

# Uvod

Spletne aplikacije so v zadnjih dvajsetih letih doživele velik razcvet, kar je seveda posledica razvoja zmogljivejše strojne opreme (zmogljivejših strežnikov) in stalnega večanja hitrosti ter razširjenosti internetne povezave. Na tem področju se vsakodnevno pojavljajo nove tehnologije, zato je tudi področje, obravnavano v diplomski nalogi, precej široko, kajti vsebuje velik nabor tehnologij.

Spletna aplikacija je aplikacija, do katere uporabniki dostopajo preko omrežja, ki je lahko javno (internet) ali pa zasebno (intranet). Ravno to je tista lastnost, ki daje spletnim aplikacijam veliko prednost pred namiznimi aplikacijami. Če uporabnik želi uporabljati namizno aplikacijo, si jo mora najprej namestiti na svoj računalnik. Mogoče to za »domačega« uporabnika ne predstavlja velike ovire, administratorju v večjem podjetju pa jo, saj mora vsakemu uporabniku posebej namestiti aplikacijo, kar pa je lahko zelo zamudno. Poleg tega je odveč skrb za predhodno shranjene podatke in posodobitve, do nje lahko dostopamo iz različnih naprav, pa še bi lahko naštevali. Za uporabo spletnih aplikacij pa mora imeti uporabnik na svojem računalniku ali mobilni napravi po navadi nameščen le spletni brskalnik.

Spletne aplikacije so v osnovi centralizirane, kar pomeni, da so nameščene na eni sami lokaciji (spletni strežnik), ki pa mora biti povezana v omrežje. Torej, ko uporabnik (klient) uporablja spletno aplikacijo, pošlje zahtevek preko brskalnika spletnemu strežniku po določenem omrežju (ponavadi internet). Spletni strežnik prestreže zahtevo, jo obdela, pridobi določene podatke od podatkovnega strežnika in pošlje podatke za prikaz spletnemu brskalniku. Ta jih seveda pretvori v uporabniku vidno obliko in jih prikaže na zaslonu.

Spletne aplikacije so vse bolj dinamične in zahtevne, zato so tudi programski jeziki, ki jih opisujejo, vse bolj kompleksni. Stalno se pojavljajo novi principi in načini razmišljanja, zato se tudi razvoj spletnih aplikacij spreminja. Konkurenca na trgu ponudnikov spletnih aplikacij je precejšnja, zato podjetja vedno težijo k časovni optimizaciji razvoja spletnih aplikacij. Prav zaradi tega so se pojavila števila ogrodja in knjižnice, ki olajšajo in pohitrijo razvoj. Razvoj spletne aplikacije je v bistvu razvoj informacijskega sistema, zato se je razvoja spletne aplikacije potrebno lotiti premišljeno in organizirano. Razvoj spletne aplikacije poteka skozi razvojne faze, ki so prisotne tudi pri razvoju klasičnih aplikacij oz. razvoju programske opreme. V zadnjem času so se predvsem uveljavile agilne metodologije, ki so bolj prilagodljive in primerne predvsem za manjša podjetja.

Razvoj programske opreme se ponavadi začne z opredelitvijo problema, ki ga bo programska oprema reševala. Sledi faza analize zahtev in specifikacije sistema, kjer zajamemo vse zahteve in določimo natančne specifikacije sistema. Sledi faza načrtovanja, v kateri odgovorimo na vprašanje, kako bomo to naredili. Nato sledita fazi implementacija in testiranje sistema; slednjega ne smemo zanemariti. Na koncu preostaneta še faza lansiranja programske opreme na trg in njeno vzdrževanje

ZGODOVINA INTERNETA

Ob koncu šestdesetih let je ameriška vojska odkrila prednosti in možnosti računalniških povezav, a tedaj prav gotovo še nihče ni slutil, da se bo to omrežje tako hitro razširilo. Internet *(ang. Interment*) izvira iz omrežja **ARPANET** (Advance Research Projects Agency NETwork), ki so ga ustanovili leta 1969 z namenom, da bi pomagalo raziskovalcem s hitrejšo izmenjavo različnih informacij. ARPANET je postal pomembno orodje, ki je omogočalo delo na oddaljenih računalniških sistemih, prenos datotek, elektronsko pošto in izmenjavo informacij po interesnih skupinah t.i. distribucijskih seznamih.

Internet je začel delovati leta 1983, ko se je omrežje ARPANET razdelilo na dve sestavni internetovi omrežji **MILNET** (MILitary NETwork) in ARPANET. Vsako je dobilo številko omrežja in z namestitvijo pretvornikov je bilo omogočeno usmerjanje paketov med njima. Po nalogu DCA (Defence Communication Agency) so morali vsi računalniki priključeni na ARPANET uporabljati **protokolni sklad TCP/IP** za izmenjavo sporočil. Tako so obstoječemu omrežju brez problemov priključili nova omrežja in pretvornike.

Sestavna dela Interneta MILNET in ARPANET sta se večala, priključena so bila tudi ostala omrežja, ki opravljajo funkcijo hrbtenice (*ang. network backbone*). Eno izmed njih je bilo CSNET (*ang.Computer and Science Network*) ustanovljeno leta 1981 z namenom, da se izboljša sodelovanje med raziskovalci inženirji in računalniškimi strokovnjaki v ZDA. CSNET je omogočal dostop do interneta tistim, ki jim nista bila dostopna ARPANET ali MILNET.

Danes se je CSNET razširil in vključuje inštitucije povezane z znanstveno raziskovalnim delom in ameriškimi univerzami in je eno izmed mednivojskih omrežij, ki sestavljajo NSFNET. Julija 1986 je NSFNET prevzel vlogo storitve omrežne hrbtenice in je omogočal komunikacijo med superračunalniškimi centri.

Uporaba storitev, ki imajo uporabljajo za prenos podatkov protokolni sklad TCP/IP (storitve telnet, ftp, gopher e-pošta) se je razširilo na akademsko in komercialno področje, kajti prej so bile te storitve dostopne le za vladne raziskave.

NSFNET je eden izmed večjih delov interneta in je omrežje omrežij, ki povezuje univerzitetna in komercialna omrežja. Od leta 1983 je število priključenih omrežij v internet rastlo eksponentno. Leta 1985 jih je bilo približno sto, 1987 dvesto, 1989 petsto, januarja 1990 pa je to število narastlo na 2218 omrežij. 31.12. 2009 je bilo v Internet vključenih preko svojih lokalnih računalniških omrežij že več kot 1.802.330.000 uporabnikov.

(vir World Internet Users and Population Stats -[http:// www. internetworldstats. com/ stats.htm](http://www.egradiva.net/moduli/spletne_aplikacije/splet/01_internet/03_datoteka.html), obiskano 21. 5. 2010).

# INTERNET DANES

Brez interneta si ne moremo več zamisliti nakupov, našega dodatnega izobraževanja, doseganja za nas zanimivih informacij, plačevanja računov, načrtovanja potovanj, poslušanja radijskih programov in glasbe, gledanja televizijskih novic in videa, komuniciranja s prijatelji, igranja računalniških iger itd. Internet je začel spreminjati celo smisel dela, da ne rečemo življenja.

Bistvo vključevanja v omrežje je v našem aktivnem soustvarjanju dogodkov, saj uporabnik informacije soustvarja (npr. Wikipedia, spletni dnevniki, spletni forumi, klepetalnice, brezplačni sistemi za publiciranje vsebine (npr. FaceBook, Myspace, live.com, spaces.msn.com, flickr, YouTube...)), Uporabnik lahko tudi priporoča informacije drugim (Blink, Del.icio.us, Digg, Furl, Google, Simpy, Spurl, MyYahoo, MyWeb...).

V tem prostoru so dobili možnost tudi časopisi, saj se je njihova dosedanja dvodimenzionalna informacija s statičnim besedilom in statično grafiko, v elektronski obliki tako imenovanih „on-line elektronskih časopisov“ razširila z možnostmi animacije, videa in avdio posnetkov. Danes že skoraj vse radijske in televizijske postaje ponujajo svoj program preko interneta.

Vse več dogodkov pa se tudi neposredno prenaša preko interneta (zelo dober primer so direktni prenosi različnih športnih tekmovanj, koncertov ali izobraževalnih dogodkov ). Preko interneta se lahko pogovarjamo s prijatelji (sistemi takojšnjega sporočanja, videokonferenčni sistemi, telefonija IP, IRC).

# ORGANIZACIJE INTRANETA

Na razvoj in delovanje Interneta pomembno vplivajo naslednje organizacije:

* **Internet Engineering Task Force** (**IETF**) – organizacija, ki se ukvarja s standardi na internetu - <http://www.ietf.org/>. Podlaga za razvoj novih storitev so poenotene specifikacije protokolov za izmenjavo informacij med sodelujočimi v komunikaciji. IETF zbira in usklajuje predloge specifikacij in po končnem usklajevanju izda priporočila (RFC standardi).
* **The Internet Society** (**ISOC**) – organizacija, ki se ukvarja s promocijo interneta -<http://www.isoc.org/>.
* **Internet Assigned Numbers Authority (IANA)** - organizacija, ki skrbi za globalno dodeljevanje imen in IP številk. Sedež ima v Kaliforniji ZDA - <http://www.iana.org/>.

# INTRANET

Predstavljajte si, da ste uslužbenec nekega podjetja in da ste večino časa na terenu, kjer se pogovarjate s strankami. Kako lahko vaše vodstvo komunicira z vami ali vi z njimi? Možen način komunikacije je prek telefona. Ta način ni najboljši, saj so lahko vaši nadrejeni zaposleni z drugimi telefonskimi pogovori ali pa niso v pisarni. Dostikrat se tudi zgodi, da potrebujete podatke o vaših poslovnih partnerjih, prodaji izdelkov, trenutni zalogi v skladiščih.... Ti podatki so shranjeni v centralnem informacijskem sistemu vašega podjetja.

Kako torej komunicirati z vašimi kolegi v službi ali pridobivati informacije o vaših poslovnih partnerjih? Odgovor na to vprašanje je **intranet**. Intranet je dejansko informacijski sistem  podjetja, ki je dostopen preko  spletnega brskalnika ali s pomočjo drugih storitev, ki jih omogoča internet.

Intranet mora omogočati zaposlenim **enostaven** in **varen** dostop do podatkov in dokumentov podjetja ter medsebojno komuniciranje med zaposlenimi. Razlika med internetom in intranetom je v tem, da je "omrežje vseh omrežij" (internet) namenjeno posredovanju informacij vsem, ki do njega dostopajo, medtem ko je intranet namenjen **samo zaposlenim** določenega podjetja ali organizacije.

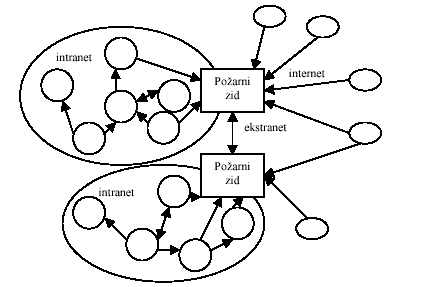
V obeh primerih uporabljamo iste tehnologije in storitve. Namen obeh je omogočiti možnost dostopa do podatkov in medsebojno komuniciranje njunih uporabnikov. Internet je predvsem namenjen predstavitvi posameznikov, podjetij ali organizacij. Podjetja predstavljajo svoje izdelke in storitve. Izdelke ali storitve lahko vsakdo naroči in plača. Intranet pa je namenjen samo uporabi znotraj podjetja. Večina podjetij pa omogoča omejen dostop do podatkov in storitev, ki tečejo na internetu, tudi svojim poslovnim partnerjem.

# EKSTRANET

Ekstranet je **povezava** **intranetov** več podjetij z namenom, da bi lahko le ta med seboj lažje poslovala. Primer uporabe ekstraneta je omogočanje direktnega plačevanja podjetij prek bank. Več podjetij se z bankami dogovori, da bodo njihove transakcije potekale direktno med njihovimi bančnimi računi. Drug primer je npr. pošiljanje naročil po izdelkih med podjetji. Tretji primer bi bil skupni rezervacijski sistem hotelov.

Poglejmo si primer bolnišnice, ki ima svojo predstavitev na internetu in svoj intranet. Na internetu ima predstavljene svoje oddelke s podatki o zaposlenih. Prek interneta omogoča svojim pacientom naročanje na preglede. Zdravniki in pacienti lahko med seboj komunicirajo po elektronski pošti. Intranet bolnišnice pa omogoča komunikacijo med posameznimi oddelki in zaposlenimi na teh oddelkih (npr. pošiljanje izvidov, konzultiranje drugih zdravnikov ali sester...).

Intranet bi moral zdravnikom ali sestram omogočati, da bi lahko od doma preučevali izvide pacientov ali si pogledali kdaj so dežurni. Na strežnikih v intranetu so lahko shranjeni tudi interni dokumenti, dopisi, obvestila. Intranet lahko tudi omogoča pošiljanje sporočil SMS zdravnikom, ki so na dopustu. Predstojniki oddelkov bi lahko svojim podrejenim prek intraneta dodeljevali naloge, sklicevali sestanke ali z njimi neposredno komunicirali. Vodstvo bolnišnice lahko preko intraneta ugotavlja razlike med planirano izvedbo operacij in dejansko izvedbo operacij. Ugotavljajo lahko tudi porabo zdravil in ostalih materialnih sredstev v bolnišnici. Bolnišnica se lahko poveže z zdravstvenimi zavarovalnicami, zdravstvenimi domovi, zasebnimi zdravniki in drugimi bolnišnicami v ekstranet. Vsi podatki o opravljenih preiskavah na pacientih, diagnozah in terapijah so dostopni vsem, ki so v takšen ekstranet priključeni. Z intranetom in ekstranetom lahko torej izboljšamo kvaliteto poslovanja organizacij, izognemo se odvečnemu pošiljanju papirja med njihovimi uporabniki, zmanjšamo pa tudi stroške njihovega poslovanja.



Prvi pogoj za izvedbo intraneta ali ekstraneta (slika) je zagotovitev **varnosti** dostopa do podatkov. Vsakemu uporabniku intraneta ali ekstraneta moramo najprej določiti vlogo v takšnem sistemu. Ugotoviti moramo, do katerih podatkov lahko dostopa oz. katere funkcije takšnega sistema lahko uporablja.

Dostop do intraneta od zunaj omogočimo samo prek požarnega zidu (ang. Firewal), ki mora onemogočati vdore od zunaj. Vsak uporabnik intraneta ali ekstraneta se mora identificirati.

Požarni zid mora zavračati vse neavtorizirane dostope do podatkov ali posameznih storitev znotraj intraneta ali ekstraneta. Vsi prenosi podatkov preko požarnega zidu morajo biti šifrirani, da jih nepooblaščeni uporabniki ne morejo prebrati. Znotraj intraneta se lahko podatki prenašajo nešifrirano, če jih lahko berejo vsi uporabniki. Za dostop do ostalih podatkov in storitev sistema, ki zahtevajo identifikacijo uporabnikov, pa uporabimo šifriran prenos podatkov.

# INTERNETOVA IMENA IN NASLOVI IP

Ker lahko na internetu komunicirajo vsi računalniki med seboj, je potreben enoten koncept identifikacije vsakega računalnika v omrežju. Vsaka organizacija ali podjetje ima enolično določeno ime, ki mu pravimo domena (domain name). Primer domene računalnikov, ki so na Univerzi v Mariboru je **uni-mb.si** (**uni-mb** pomeni univerzo v Mariboru, **si**pa pomeni Slovenijo). Znotraj vsake domene imamo lahko poljubno število računalnikov. Vsak računalnik mora imeti svoje ime. Primer imena računalnika je **mario.uni-mb.si**. Lahko pa ima računalnik tudi več namišljenih imen (vzdevkov, angl. **alias**), npr. mario.uni-mb.si ima vzdevek maribor.uni-mb.si. Vsak računalnik ima določen tudi svoj naslov IP (npr. mario.uni-mb.si ima naslov IP 164.8.252.9).

Splošna oblika domene skupine računalnikov je **nekje.področje**. Del imena nekje pove sistem, lokacijo ime podjetja ali ime organizacije. Področje pove za omrežja v ZDA, katera vrsta organizacije uporablja to omrežje, za ostala omrežja, ki so izven ZDA pa državo, v kateri je to omrežje. Možni so naslednji tipi organizacije:

* **com** podjetja ali profitna organizacija, kot npr. Convex Computers - convex.com,
* **edu** izobraževalna organizacija, kot  npr. New York Universtity - nyu.edu,
* **gov** vladna ameriška organizacija, kot npr. NASA - nasa.gov,
* **mil** ameriška vojaška organizacija, kot npr. Air Force - af.mil,
* **net** podjetja, ki se ukvarjajo s ponudbo omrežnih storitev, kot npr siol.net,
* **org** neprofitne organizacije kot npr. acm.org.

DVOČRKOVNE DOMENE

Področja com, org in net so mednarodna in jih lahko uporabljajo podjetja in organizacije v vseh državah sveta. Za registracijo domen, ki se končajo com, org in net je zadolžena organizacija **INTERNIC**. Če si hočete registrirati svojo domeno s temi končnicami, morate najprej preveriti, če takšna domena že ne obstaja, nato pa uporabite enega izmed naslovov, ki omogočajo proti plačilu registracijo domen.

Izven ZDA je področje dvočrkovna koda države, ki je določena pri mednarodni organizaciji za standardizacijo (ISO), kot npr.:

* au– Avstralija,
* ca– Kanada,
* fr– Francija,
* uk– Velika Britanija. Tu so definirana tudi podpodročja kot npr. ac.uk za akademske lokacije in co.uk za komercialne organizacije.
* ch– Švica,
* de– Nemčija,
* fi– Finska,
* jp– Japonska,
* il– Izrael,
* si– Slovenija,
* tv– Tuvalu

# DOMENE IN PRESLIKAVE NASLOVOV IP

Za registracijo domen s končnico **si** skrbi ARNES. Vse informacije o registraciji teh domen najdete na naslovu [http://www.arnes.si/](http://www.egradiva.net/moduli/spletne_aplikacije/splet/02_internet/04_datoteka.html). Za internetna imena ni nobenih posebnih omejitev, prepovedana je le uporaba nekaterih posebnih znakov ($, \*, %, ...), posamezno ime, ločeno od drugega s piko, ne sme biti daljše od 63 znakov, skupaj pa je lahko največ 255 znakov. Internetni naslov v obliki nekje.področje označujemo tudi s kratico **FQDN** (angl. Fully Qualified Domain Name) in je navadno izbran tako, da nam pove nekaj smiselnega o samem podjetju, organizaciji ali produktu (npr. coca-cola.com).

Internetni naslov, oziroma naslov IP, je implicitno uporabljen za usmerjanje omrežnega prometa do ustreznega vira. Vsakemu računalniku je prirejen enoumni 32-bitni naslov.

Ker naslov IP določa omrežje in računalnik v tem omrežju, nam v bistvu pove povezavo oziroma pot do tega računalnika. To je osnovna dobra lastnost tega naslavljanja. Slabost takšnega naslavljanja pa se pokaže v primeru ko računalnik preselimo v drugo omrežje. Če to naredimo, moramo spremeniti tudi njegov naslov IP in njegovo ime. To se bo rešilo, ko se bo uvedel protokol IPv6 (sedanji protokol se imenuje IPv4), kjer bo imel naslov IP naslov 128 bitov in bodo naslovi neodvisni od geografske lokacije.

Za preslikavo imena v naslov obstaja porazdeljena podatkovna baza imenovana Domain Name System (**DNS**). Vsako manjše omrežje ima enega ali več imenskih strežnikov (name serverjev), ki nam za to omrežje iz imena računalnika določajo naslov IP računalnika.

# SVETOVNI SPLET

**WWW** (angl. Word Wide Web) – svetovni splet je zbirka hipertekstovnih dokumentov, ki so med seboj povezani preko interneta. Večina dokumentov je narejenih v jeziku **HTML** (Hypertext Markup Language), ki ga je leta 1991 na **CERN**-u v Švici razvil **Tim Berners-Lee**. Izdelal je spletni strežnik, urejevalnik dokumentov HTML in spletni brskalnik. Za prenos podatkov med spletnim strežnikom in brskalnikom je uporabil protokol HTTP (angl. Hypertext Transfer Protocol). Specifikacije jezika HTML, protokola HTTP, programsko kodo za spletni strežnik, spletni brskalnik in urejevalnik dokumentov HTML je objavil na Internetu in omogočil vsem, ki so to želeli, da so programsko opremo uporabljali in naprej razvijali.



Slika 1: Videz na sliki prikazanega dokumenta HTML v spletnem brskalniku Okno spletnega brskalnika Mosaic

Vir: Wikipedia

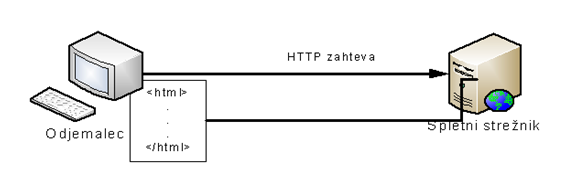
# SVETOVNI SPLET

Leta 1993 je **Marc Andreessen** skupaj s sodelavci izdelal na NCSA prvi grafični spletni brskalnik **MOSAIC**. Tudi Marc Andersen je objavil programsko kodo svojega spletnega brskalnika na internetu in omogočil vsem, ki so to želeli njegovo brezplačno uporabo. S pojavom tega brskalnika se je začel hiter razvoj svetovnega spleta, saj je bilo od takrat možno enostavno prikazati tudi večpredstavne vsebine, kar je omogočilo uporabo svetovnega spleta tudi v poslovne namene (predstavitev podjetij, virtualne trgovine, rezervacije hotelov...).

Število uporabnikov spleta se je začelo zelo hitro povečevati, saj so si ta brskalnik namestili skoraj vsi, ki so imeli dostop do interneta. Pojavljati se je začelo tudi vedno večje število podjetij, ki so ponujala dostop do interneta, prav tako pa se je internet začel še hitreje širiti med mladimi in v akademskih krogih, saj se je število informacij skokovito povečevalo.

Maja 1994 se je izvedla prva WWW konferenca, kjer so se sestali znanstveniki, ki so razvijali tehnologije svetovnega spleta. Tega leta je bil na MIT v ZDA ustanovljen WWW konzorcij. Ta organizacija skrbi za standarde na področju tehnologij povezanih s spletnimi aplikacijami (CSS, HTML, RDF, OWL, XML itd.). Njihov spletni naslov je: <http://www.w3.org/>.

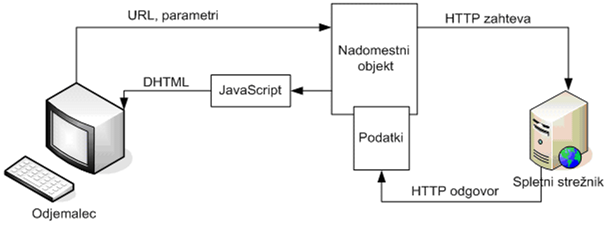
# PRINCIP KOMUNIKACIJE ODJEMALEC (BRSKALNIK) IN SPLETNI STREŽNIK



Večinoma spletne aplikacije delujejo na naslednji način.

Ko uporabnik izda novo zahtevo, bodisi preko skripte na odjemalski strani ali s pomočjo klika na gumb, se podatki iz spletne strani pošljejo po protokolu HTTP (»HyperText Transfer Protocol«) na spletni strežnik. Strežnik procesira zahtevo (npr. izvede ustrezno funkcijo, shrani podatke v podatkovno bazo...) in vrne odjemalcu dokument HTML, ki ga spletni brskalnik naloži in tako nadomesti prejšnjo vsebino.

# PRISTOP AJAX



Značilnost pristopa AJAX v nasprotju s tradicionalnim je v tem, da omogoča izvedbo ločenih zahtev HTTP in tako predstavlja vmesni sloj za komunikacijo s strežnikom.

Izvedbo ločene zahteve HTTP sprožimo preko dogodka na strani HTML(npr. klik na gumb), kjer s pomočjo nadomestnega objekta (v splošnem je toXmlHttpRequestobjekt) izdamo običajno zahtevo HTTP, bodisi sinhrono ali asinhrono. Ko prispe odgovor iz strežnika, nadomestni objekt izvede funkcijo JavaScript, preko katere nato osvežimo samo del strani, ki potrebuje posodobitev oz. za katerega smo preko nadomestnega objekta pridobili podatke.

# PROTOKOL HTTP

* Dogovarjanje pošiljanja vsebine med odjemalcem in strežnikom (vsak izmed njiju pove kako želi sprejemati podatke (content negotiation).
* Avtentifikacija.
* Izmenjava podatkov ob uporabi **MIME** (angl. Multiporpouse Internet Mail Extension) (npr. image/gif, text/html, text/plain…).

# vrstice glave HTTP

* **Odjemalec**: glave zahteve - odvisne so od:
  + - odjemalca,
    - verzije HTTP,
    - jezika.
* **Strežnik**: glave odgovora - odvisne so od
  + - zahteve,
    - verzije HTTP,
    - konfiguracije strežnika.

# METODE HTTP

GET izmenjava podatkov poteka prek argumentov.

POST izmenjava poteka prek standardnega vhoda oz. izhoda.

HEAD podobno kot GET, vendar ne vrača rezultata.

TRACE uporaba za diagnostiko – strežnik vrne sporočilo, ki smo ga poslali nazaj.

PUT (HTTP 1.1) nalaganje na strežnik.

DELETE (HTTP 1.1) brisanje s strežnika.

# STATUS HTTP

1xx: informativne kode (100: Continue);

2xx: uspešno (200: OK);

3xx: preusmeritev (301: Moved Permanently- prestavljen dokument + vrne novi naslov);

4xx: napake pri odjemalcu (400: Bad Request; 404: Not Found – ni dokumenta);

5xx: napake na strežniku (500: Internal Server Error; 505 HTTP Version Not Supported).

EVOLUCIJA SPLETNIH APLIKACIJ

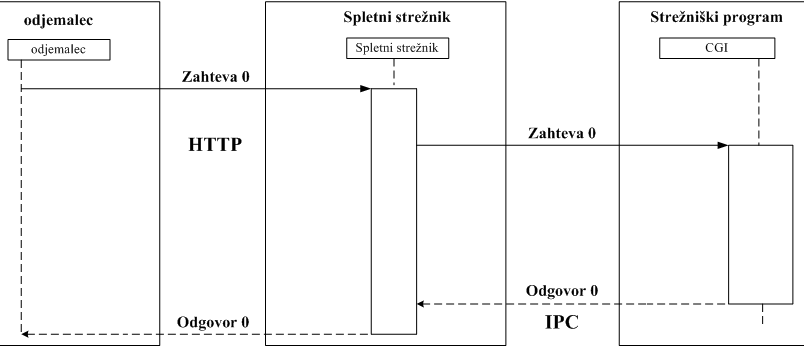
* Statične spletne aplikacije (*Document centric Web sites).*
* Interaktivne spletne aplikacije (*Interactive Web applications).*
* Transakcijske spletne aplikacije (*Transactional Web applications).*
* Spletne aplikacije, ki se izvedejo s pomočjo množice med seboj tokovno povezanih spletnih storitev (*Workflow-based Web applications)*.
* Spletne aplikacije za podporo skupinskemu delu (C*ollaborative Web applications).*
* Uporabniško bogate spletne aplikacije (*Rich Internet Applications*).
* Vseprisotne spletne aplikacije (*Ubiquitous Web applications).*
* Vključitev uporabnikov v gradnjo spletnih vsebin (Web 2.0 spletne aplikacije -*Social Web*).
* Pomensko in storitveno usmerjen splet (Web 3.0 spletne aplikacije –*semantic Web*).

STATIČNE SPLETNE APLIKACIJE

Zbirka dokumentov na strežniku HTML.

* **Prednost**: kratek čas za posredovanje dokumentov iz strežnika.
* **Slabost**: težko vzdrževanje konsistentnosti vsebine in hiperpovezav.

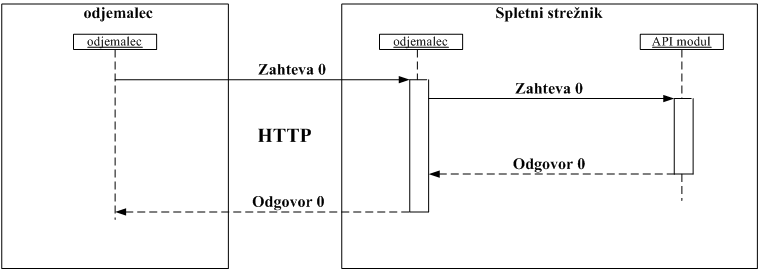
STREŽNIŠKI PROGRAMI CGI



Slika 2: Potek komunikacije med odjemalcem in strežniškim programom CGI

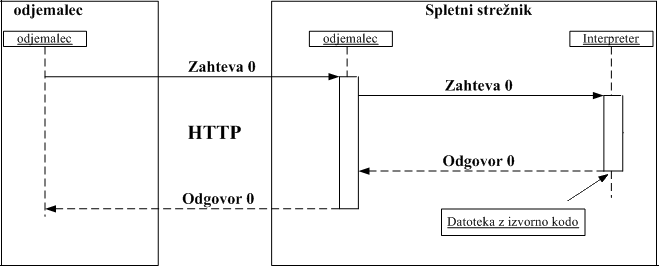
* Programi so običajno shranjeni na direktoriju /cgi-bin
* Rezultati izvedbe programa se izpisujejo na stdout (standardni izhod)
* Odjemalec lahko pošlje sporočilo s pomočjo metod:
  + **GET** – podatki se pošljejo v sistemski spremenljivki QUERY STRING,
  + **POST** – spletni strežnik posreduje podatke prek standardnega vhodastdin
* Vsaka instanca programa se izvaja v ločenem naslovnem prostoru
* **Prednost** – visoka stopnja izolacije med spletnim strežnikom in strežniškim programom
* **Slabosti**:
  + Za vsak klic CGI programa se ustvari nova instanca programa
  + Počasnost izvedbe, saj se mora ob zagonu CGI programa ustvariti novo izvajalno okolje.
  + Ob velikem število zahtev po izvedbi strežniških CGI programov se zelo zniža propustnost strežnika.

STREŽNIŠKI MODUL



* Izvajajo se v naslovnem prostoru spletnega strežnika.
* Na ta način so odpravljeni presežki zaradi pogostega nalaganja izvedljivih modulov, inicializacije procesov in medprocesne komunikacije.
* Morebitne napake v implementaciji strežniškega modula zaradi nizke stopnje izolacije med spletnim strežnikom, ogrožajo stabilnost spletnega strežnika.
* Običajno so pisani v visokih programskih jezikih (C, Java, C++, Visual Basic…) in so implementirani kot DLL datoteke (Dynamic Link Library) (NSAPI, ISAPI) ali v obliki java servletov (JSAPI)
* Prednost strežniških modulov JSAPI je njihova prenosljivost, saj jih prevedemo enkrat, nato pa jih namestimo na katerikoli strežnik, ki zagotavlja ustrezno okolje za njihovo izvajanje (JVM).

INTERPRETIRANI STREŽNIŠKI PROGRAM



* PERL, tcl, ASP, PHP.
* Preprostost, velika izrazna moč teh jezikov.
* Nekoliko počasnejše izvajanje teh programov od strežniških modulov.
* Implementacija in vzdrževanje programske opreme bistveno cenejše kot pri implementaciji strežniških modulov.

# Označevalni jezik HTML5

**HTML** (angl. Hyper Text Markup Language) je jezik, ki opisuje vsebino spletnega dokumenta. Uporablja posebno sintakso, ki vsebuje oznake (elemente), ki se ovijajo okrog besedila v dokumentu in brskalniku interpretirajo ta del besedila . HTML predstavlja osnovo vsake spletne strani. Lahko ga napišemo kar v beležnici ali v kakšnem drugem urejevalniku besedila.

Primer preprostega HTML stavka:

**<p id='primer'>To je naš prvi odstavek</p>**

Oznaka <p> predstavlja oznako za odstavek, kar pomeni, da tisti del, ki sledi, predstavlja vsebino odstavka. Ker je ta oznaka vedno na začetku vsebine, jo imenujemo začetna oznaka. Oznaka </p> predstavlja končno oznako, v našem primeru se nahaja na koncu odstavka, pomeni pa, da zaključuje ta odstavek. Del, ki se nahaja zraven začetne oznake, v našem primeru id='primer', predstavlja atribut, ki HTML elementu zagotavlja dodatne informacije. Del, ki se nahaja znotraj oznake (To je naš prvi odstavek), predstavlja element. Kot element lahko dodamo tudi novo oznako, torej lahko oznake gnezdimo.

Obstajajo pa tudi oznake, ki nimajo elementa, torej so prazne (npr. <img>). Te oznake imajo samo začetno oznako. Jezik HTML so zasnovali konec osemdesetih let prejšnjega stoletja v Evropskem centru za fiziko osnovnih delcev (CERN), pod vodstvom Tima Berners-Leeja. Tim Berners-Lee je delal na razvoju informacijskega sistema, v katerem bi lahko besedilo vsebovalo povezave in reference na druga dela, kar bi omogočalo bralcu hitro preskakovanje z dokumenta na dokument. Ustvaril je strežnik za objavo teh dokumentov (imenovan Hypertext) in tudi program za njihovo branje, ki je bil imenovan WordWideWeb. Ta oprema je izšla leta 1991, vendar je pravo eksplozijo doživela šele dve leti kasneje. Z razvojem jezika HTML se je potem začelo ukvarjati kar nekaj proizvajalcev brskalnikov.

Leta 1994 se je oblikovalo združenje **WC3** (angl. World Wide Web Consortium), ki je eno najpomembnejših združenj na področju spleta. WC3 se ukvarja z oblikovanjem standardov za izgradnjo spleta, razvojem semantičnega spleta, s priporočili in novimi tehnologijami. Vanj so vključeni vsi največji proizvajalci programske opreme (Microsoft, IBM, Adobe, Apple, Google ) in je sestavljeno iz približno 400 organizacij. Z ustanovitvijo WC3 je delovna skupina uspela najti skupni jezik med različnimi razvijalci in v letu 1996 pričela z standardizacijo jezika HTML. Rezultat je bil standard HTML 3.2. Leta 1999 je bil izdan standard HTML 4.01, ki še danes velja za uraden HTML standard. Leta 2000 se je standard HTML preimenoval v standard XHTML. Do danes se je na področju spleta ogromno spremenilo, zato je že nekaj let v pripravi novi standard HTML5. Danes vsi večji brskalniki že podpirajo elemente standarda HTML5. HTML5 v primerjavi z njegovimi predhodniki prinaša ogromno novosti:

* oznaki <video> in <audio> za predvajanje multimedijskih vsebin,
* oznaka <canvas> za risanje na podlago,
* novi elementi za boljše določanje strukture dokumenta HTML (npr. <section>,

<article>, <header>, <nav>),

* novi elementi za obrazce (npr.: <calendar>, <date>, <time>, <email>, <url>,

<search> ...),

* oznake za matematične funkcije MathML.

# ELEMENTI HTML 5

## Head

* [doctype](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/doctype.html)
* [html](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_html.html)
* [head](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html)
* [title](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_title.html)
* [base](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_base.html)
* [link](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_link.html)
* [meta](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta.html)
* [style](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_style.html)
* [script](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_script.html)
* [noscript](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_noscript.html)

# <!DOCTYPE HTML>

Definicija in uporaba:

<!DOCTYPE> deklaracija mora biti na začetku dokumenta HTML5, pred značko html .

Deklaracija <!DOCTYPE> ni značka, ampak navodila brskalniku v katerem HTML je zapisana stran.

V HTML 4 in XHTML1 vse <!DOCTYPE> deklaracije zahtevajo referenco DTD, ker sta HTML4 in XHTML1 zasnovana na SGML.

HTML5 ni zasnovan na SGML in ne zahteva reference DTD kot XHTML1 in HTML4.

**Nasvet:**Vedno dodajte deklaracijo <!DOCTYPE> svojim dokumentom HTML, da bo brskalnik vedel katero različico dokumenta naj pričakuje.

## Razlike med HTML 4.01, XHTML1 in HTML5.

V HTML 4.01 so tri različne <!DOCTYPE> deklaracije. V HTML5 je samo ena:

<!DOCTYPE html>

ELEMENT HTML

Atributi:

globalni & manifest

Otroci:

[body](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_body.html) & [head](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html)

Pomen:

Značka **html** informira brskalnik, da je dokument kodiran po pravilih HTML. Začetna in končna značka določata začetek in konec dokumenta.HTML 5 dovoljuje, da začetne značke ni, če se dokument ne prične s komentarjem. Tudi zaključno značko lahko izpustimo, če ima dokument neprazen element body ali če ima začetno značko html. Vseeno je bolje, da znački uporabljamo. Pred uvodno značko html je lahko samo še značka **DOCTYPE**. Ta določa po katerih pravilih in sintaksi je sestavljen dokument.

Dodatno:

Dokument HTML običajno vsebuje pravilno gnezdene značke **html**, **head**, **title**, in **body**.

HTML 5 ne zahteva elementov **head**, **title**, in**body**.

**CSS**

body {display: block;}

body:focus {outline: none;}

ELEMENT HEAD

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

[html](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_html.html)

Otroci:

en element [title](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_title.html) & [& base](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_base.html) & [link](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_link.html) & [meta](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta.html) &[script](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_script.html) & [noscript](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_noscript.html) & [style](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_style.html) & [command](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_command.html)

Pomen:

Element **head** predstavlja glavo dokumenta HTML. V glavi dokumenta je zbirka metapodatkov

CSS

head {display: none; }

Takoj za korenskim elementom nastopi del, imenovan glava dokumenta. Služi za vsebovanje vseh elementov, ki nadzorujejo potek telesa v dokumentu – element **body**. Glava vsebuje element naslov in vso vsebino, ki bi naj bila shranjena v meta podatkih. Določimo lahko naslov (le-ta se prikaže v naslovni vrstici brskalnika), ključne besede (te uporabljajo razni iskalniki), način kodiranja (ta določa, na kakšen način smo zapisali znake, ki jih ni v tabeli ASCII), obliko posameznih elementov, vključimo lahko pomožne datoteke in drugo. Kar napišemo v glavi, prikazovalnik ne prikaže v svojem oknu ali na drugačen način. Do teh podatkov lahko pridemo samo, če to posebej zahtevamo.

ELEMENT TITLE

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

[head](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html)

Otroci:

[replaceable character data](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_character_data.html)

Pomen:

Element **title** določa ime dokumentu HTML. Samo **en** element te vrste se lahko nahaja v celotnem dokumentu. Element je v večini brskalnikov prikazan kot besedilo v naslovni vrstici brskalnika (v nekaterih pa tudi v statusni vrstici). V kolikor je element prazen, brskalnik prikazuje določen naslov ali ime datoteke.

Dodatno:

Title se **mora** biti v glavi (**head**) dokumenta HTML. Ne more vsebovati drugih elementov (in tudi ne komentarjev), ampak samo besedilo ali prazen niz (kar ni priporočljivo). Lahko se uporabi samo znake, ki so v določenem dokumentu dovoljeni in seveda **znakovne entitete**, npr. &nbsp; za presledek.

Naslove prepoznavajo tudi internetni **iskalnik**i, zato je pomembna pravilna izbira naslova. Avtorji spletnih strani bi naj izbirali naslov zelo previdno. Kot pri naslovu knjige, bi naj izražal naslov na bežen pogled vsebino dokumenta, brez uporabe dolgih stavkov ali gesel. Prva beseda v naslovu je zelo pomembna, bolje je zapisati **br - primer HTML** kot Primer HTML - br.

Če je okno brskalnika razdeljeno na okvirje, naslovi posameznih oken niso prikazani v brskalniku (tudi to je eden izmed razlogov, zakaj se odsvetuje uporaba okvirjev).

CSS:

title { display: none; }

ELEMENT BASE

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html) & [href](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_href.html) & [target](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_target.html)

Starši:

[head](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html)

Pomen:

Element **base** določa osnovni **URL** dokumenta HTML. Na ta način je mogoča uporaba relativnih naslovov kjerkoli v dokumentu. In zanimivo – osnovni **URL** dokumenta ni nujno, da je absolutni naslov - lahko je tudi relativni.

Element **base** se nahaja znotraj elementa **head** in to pred elementi na katere se nanaša - na zunanje slike, predmete, slogovne predloge.

Primer:

Relativni naslov

uri="../datoteke/moja.gif " in

<base   
href="http://www.bb.com/n/pac.html>

se bo obravnaval kot

uri="http://www.bB.com/datoteke/moja.gif"

# ELEMENT LINK

Atributi:

globalni & [href](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_href.html) & [rel](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_rel.html) & [hreflang](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_hreflang.html) & [media](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_media_a.html) & sizes & [type](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_type.html)

Starši:

[any element that can contain metadata elements, noscript](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html)

Otroci:

[nima (prazen element)](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_void_element.html)

Pomen:

Značka **link** se uporablja za vzpostavljanje povezave med tekočim dokumentom in enim ali več odvisnimi dokumenti. Lahko je samo v glavi dokumenta. Obvezen atribut je **href**.

Določanje zunanje slogovne predloge

Zunanjo slogovno predlogo se določi z elementom **link** v glavi dokumenta (element head), ki se mu določi naslednje atribute:

* Atribut **rel** se nastavi na vrednost **stylesheet**.
* Atribut **type**, ki določa jezik, se v povezani slogovni predlogi običajno nastavi na vrednost **text/css**. S tem se prepreči brskalniku, da bi naložil slogovno predlogo v neznanem jeziku. V HTML5je privzeta vrednost "text/css" in atributa ni potrebno pisati.
* Atribut **href** določa lokacijo, na kateri je slogovna predloga. Vrednost je **URI**.

Za pravilno delovanje spletnih strani na več platformah se priporoča uporaba izključno zunanjih pravil CSS.

CSS:

link {display: none; }

# ELEMENT META

Atributi:

content & http-equiv & name & charset

Starši:

[head](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html)

Otroci:

[nima (prazen element)](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_void_element.html)

Pomen:

Glava dokumenta vsebuje meta podatke. Preprosto povedano so meta podatki podatki, ki nam povedo splošne podatke o XHTML dokumentu. Ti podatki so lahko karkoli, kar bi lahko uporabnika zanimalo o dokumentu samem. Pogosto so to avtorjevo ime, čas izdelave, informacije o avtorskih pravicah in druge podrobnosti. Meta podatke uporabljajo pri svojem delu spletni iskalniki (npr. Google, Bing, Najdi si). V dokumentu je lahko več elementov **meta**.

Privesek "meta" vsebuje dva primarna atributa, ime (name) in vsebino (content).

Vrh obrazca

Večnamenski element za predstavitev meta podatkov.

Podrobnosti meta elementa so napisane v naslednjih sklicih:

* [meta name="application-name"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_application-name.html)
* [meta name="author"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_author.html)
* [meta name="description"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_description.html)
* [meta name="generator"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_generator.html)
* [meta name="keywords"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_keywords.html)
* [meta http-equiv="refresh"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_refresh.html)
* [meta http-equiv="default-style"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_default-style.html)
* [meta http-equiv="content-type"](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_content-type.html)
* [meta charset](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_meta_charset.html)

Dno obrazca

ELEMENT STYLE

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)&[media](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_media.html)&[scoped](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_scoped.html)& type

Starši:

Vsak element, ki lahko vsebuje meta elemente [div](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_div.html), [noscript](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_noscript.html), [section](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_section.html), [article](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_article.html), [aside](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_aside.html).

Otroci:

Nezamenljivi znakovni podatki

Pomen:

Element **style** ustvari pravila za prekrivne sloge v samem dokumentu in za dodanje datotek s pravilom **@import**.

Notranja pravila prekrivnih slogov veljajo **izključno** za dokument, v katerem so definirana. Značka se nahaja v glavi in ne vsebuje pa ničesar drugega kot golo besedilo.

Koda, ki se nahaja med začetno in končno značko ni koda HTML, ampak so pravila **slogovnih predlog** (CSS, Cascading Style Sheets). Za več informacij, si oglejte strani o slogovnih predlogah.

Uporaba notranjih slogov ni priporočljiva. Popolno združljivost slogovnih predlog s HTML se lahko doseže samo z uporabo zunanjih prekrivnih slogov.

# ELEMENT SCRIPT

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html) &[async](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_async.html)&[defer](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_defer.html)&[type](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_type.html)&[charset](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_charset.html)&[src](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_src_script.html)

Starši:

any element that can contain [metadata elements](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html), any element that can contain [phrasing elements](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html)

Otroci:

non-replaceable character data

Pomen:

Element **script** je uporabljen za vstavljanje kode v glavi in/ali v telesu dokumenta HTML. Praktično to pomeni, da se uporablja v glavnem za izvajanje kode JavaScript v HTML. Zelo redko pa se uporablja drug jezik.

Uporabniški skript (ang. Client-Side Script) je program, ki je vključen v dokument HTML ali pa je z njim neposredno povezan. Program se izvede, ko je stran naložena ali pa v primeru nekega dogodka.

Skripti kot razširitev HTML dovoljujejo razvijalcev spletnih strani aktivne in interaktivne možnosti:

* Skripti lahko dinamično spreminjajo vsebino strani.
* V kombinaciji z obrazci se lahko vnaša razna vsebina. Vsebina se lahko uporabi na strani, lahko pa se prenese strežniku.
* Skripti se lahko odzivajo na razne dogodke kot je nalaganje in zapiranje strani, klikanje in premikanje miške.

Skripti se lahko izvedejo samo enkrat (npr. ob nalaganju strani) ali pa vsakokrat, ko to zahteva nek **dogodek**.

Vsebina skripta je lahko navedena kot vsebina elementa **script** ali pa v zunanji datoteki. V kolikor atribut **src** ni naveden, brskalnik interpretira, da se vsebina elementa skript. V primeru, da je atribut **src**navedena kot nek **URI**, se brskalnik ne posveča vsebini elementa, temveč poišče skript v zunanji datoteki.

Skriptne jezik se mora navede kot atribut značke.

<script type="text/javascript" src="skript.js"></script>

V primeru, da skriptni jezik ni naveden, brskalniki obravnavajo privzet skriptni jezik kot **text/javascript**.

**Opomba:** Poznamo več smeri, kako je skript lahko izvršen.

* Če je async="async": Skript je izvršen asihrono s preostalo vsebino strani.
* Če async ni prisoten in defer="defer": se stran izvrši, ko stran neha z napakami pri razčlenjevanju.
* če ni prisoten ne async ne defer je privzeto: skript je pripeta in izvršena takoj.

**CSS:**

script { display: none; }

# NOSCRIPT

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Katerikoli element, ki lahko vsebuje [metadata elemente](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html), katerikoli element, ki lahko vsebuje [phrasing elemente](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html). Katerikoli element, ki lahko vsebuje [flow elemente](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html)

Otroci:

Noben ali eden od: en element [link](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_link.html) , ali meta http-equiv="default-style" element, ali meta http-equiv="refresh" element, ali element [style](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_style.html)

ali transparenten (ali [phrasing content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html) ali [flow content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html))

Pomen:

Element **noscript** se uporablja se prikazovanje nadomestne vsebine v brskalnikih:

* ki ne prepoznajo elementa **script**
* ki ne prepoznajo skriptnega jezika
* kadar je izvajanje skriptov onemogočeno.

Po pravilih za dostopnost je potrebno **vsako** vsebino skripta prikazati tudi z **noscript**, zaradi uporabnikov s posebnimi potrebami.

<script type="text/javascript">

alert("Halo!")

</script>

<noscript>

<p>Pojdi na <a href="http://pojdi.halo.com/data">na neko stran</a></p>

</noscript>

## Sections

# body

# article

# nav

# aside

# section

# header

# footer

# h1-h6

# hgroup

# address

# ELEMENT BODY

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

[html](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_html.html)

Pomen:

Znački elementa **<body/>** in **</body>**označujeta začetek in konec vsebine dokumenta HTML. Pravilno oblikovan dokument HTML mora vsebovati ta element.

V telesu se napiše vse, kar se želi, da bo brskalnik prikazal na zaslonu. Vsebino različni brskalniki na različnih platformah različno interpretirajo.

Pri tolmačenju besedila brskalniki tabulatorje in znake za novo vrstico obravnavajo kot presledke, več zaporednih presledkov pa obravnavajo kot enega samega (obstajajo tudi izjeme, a o tem drugje). Ko mu zmanjka prostora v vrstici, samodejno skoči v novo vrsto.

Grafični vmesniki pa znajo prikazati tudi slike, fotografije, risbe. Nekateri tudi zvok in video. Za oblikovanje vsebine se uporabljajo slogovne predloge.

CSS:

body { display: block;  
margin: 8px; }

body:focus {   
outline: none; }

# ELEMENT ARTICLE

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

katerikoli element ki vključuje [flow elements](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html)

Otroci:

Article element se ne sme pojaviti kot otrok elementa adress.

Pomen:

<article> določa neodvisno vsebino.

članek moral bi biti smiselen sam po sebi in omogočen, da se razdeli neodvisno od preostalega območja.

Primeri možnih člankov:

* pogovori na forumah
* članek o novicah
* blog
* komentar uporabnika

CSS

article { display: block;}

# ELEMENT NAV

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Katerikoli element, ki lahko vsebuje [flow elements](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html)

Otroci:

[flow content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html)

Pomen:

Element nav označuje področje navigacijskih povezav.

Ni potrebno, da so vse povezave v elementu nav. Namenjen je prevsem za področje navigacijskih povezav.

Brskalniki, kot so bralniki ekrana za uporabnike sposebnimi potrebami lahko uporabljajo ta element, da ugotovijo ali lahko izpustijo to področje pri tolmačenju vsebine.

CSS:

nav { display: block; }

# ELEMENT ASIDE

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Katerikoli element ki vsebuje flow elemente

Otroci:

Nič ali več slogovnih elementov, ki sledijo flow vsebini.

Element aside se nes me pojaviti kot potomec elementa adress .

Pomen:

Aside mora biti povezan z okolico.

Vsebina elementa aside je lahko vstavljena kot stranska vsebina v članku.

Element aside predstavlja vsebino, ki je v povezvezi z vsebino, ki predstavlja tekstovni pretok dokumenta.

Na tiskanih dokumentih je vrsta vsebine, ki jo predstavlja aside včasih pokazana kot navpični trek ali opomba v nogi.

CSS

aside { display: block; }

# ELEMENT SECTION

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Vsak element, ki lahko vsebuje [flow elemente](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html).

Element section ne more biti potomec elementa adress.

Otroci:

Nič ali več elementov [style](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_style.html) , ki jim sledi [flow vsebina](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html).

Pomen:

Element section ločuje besedilo, oz. dokument na dele. Kot na primer: poglavja, glave, noge ali pa kakšen drug del.

Element section predstavlja del besedila, običajno s svojim naslovom.

CSS:

section { display: block; }

# ELEMENT HEADER

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Vsak element, ki lahko vsebuje "flow elements" .

Element **header** ne more biti vstavljen v elemente footer, address ali drugi header.

Otroci:

[flow content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html)

Pomen:

Element **header**označuje uvod ali skupino navigacijskih elementov dokumenta.

Element header lahko vsebuje naslove, podnaslove, različice, navigacijske elemente in drugo.

Element header ponavadi vsebuje naslove za odsek (elementi h1-h6 ali element hgroup ), kot tudi uvodni material ali navigacijsko pomoč za odsek.

CSS:

header {display: block; }

# ELEMENT FOOTER

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Vsak element, ki lahko vsebuje "flow elements" .

Element **header** ne more biti vstavljen v elemente footer, address ali drugi header.

Otroci:

[flow content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html)

Pomen:

Element **header** označuje uvod ali skupino navigacijskih elementov dokumenta.

Element header lahko vsebuje naslove, podnaslove, različice, navigacijske elemente in drugo.

Element header ponavadi vsebuje naslove za odsek (elementi h1-h6 ali element hgroup ), kot tudi uvodni material ali navigacijsko pomoč za odsek.

CSS:

header {display: block; }

# ELEMENT H#

#=1,12,3,4,5,6

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

katerikoli element, ki vsebuje [flow elements](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_common_content_models.html),[hgroup](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_hgroup.html)

Otroci:

[phrasing content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html)

Uporaba:

Elementi **h1**, **h2**, **h3**, **h4**, **h5**, **h6** so namenjeni za prikaz naslovov v dokumentu ali sekciji. Brskalniki prikažejo naslove z različnimi velikostmi pisave. Element h1 prikaže z največjo pisavo, element h6 pa z najmanjšo. Ta je običajno celo manjša od običajnega besedila. Seveda se lahko za oblikovanje uporabijo slogovne predloge.

Elementi so obravnavani kot odstavki. Samodejno dodajo prehod v novo vrstico, dodajo presledek pred in za odstavkom. Element **h1** določa naslov na najvišjem nivoju v sekciji. Dokument naj bi vseboval en element h1 za označitev naslova na najvišjem nivoju.

Ti odstavki so namenjeni za naslove. Grda navada nekaterih oblikovalcev je, da uporabljajo elemente **h#** za oblikovanje povdarjenega besedila. Tudi izpuščanje elementov po hierarhiji ni dovoljeno. Če je v dokumentu element **h3**, potem morata biti že prej elementa **h1** in **h2**.

CSS:

h1 { display: block;  
font-size: 2em;  
margin-before: 0.67em;  
margin-after: 0.67em;  
margin-start: 0;  
margin-end: 0;   
font-weight: bold; }

# ELEMENT HGROUP

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

Katerikoli element lahko vsebuje "flow elements".

Otroci:

En ali več elementov: [h1, h2, h3, h4, h5, h6](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_h.html)

Pomen:

Element hgroup predstavlja skupino naslovov.

Element hgroup je običajno uporabljen, da združi niz enega ali več elementov [h1 – h6](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_h.html) — da združi, na primer, naslov in podnaslov.

CSS:

header {display: block; }

# ELEMENT ADDRESS

Atributi:

[globalni](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atributi_globalni.html)

Starši:

[blockquote](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_blockquote.html) | [body](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_body.html) | [button](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_button.html) | [dd](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_dd.html) | [del](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_del.html) | [div](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_div.html) |[fieldset](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_fieldset.html) | [form](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_form.html) | [ins](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_ins.html) | [li](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_li.html) | [map](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_map.html) | [noscript](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_noscript.html) | [object](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_object.html)| [td](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_td.html) | [th](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_th.html)

Otroci:

[flow content](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/definicija_content.html)

Pomen:

Element **address** določa kontaktne informacije dokumenta (ime in priimek avtorja, e–pošta, naslov...). Brskalniki prikazujejo element v drugačni obliki kot ostalo besedilo, običajno v ležeči pisavi (seveda je mogoče uporabiti slogovne predloge).

Besedilo v address lahko vsebuje tudi druge elemente HTML, ne more pa address.

če je element znotraj elementa, predstavlja informacijo o dokumentu. Če je element znotraj elementa, predstavlja kontaktno informacijo za ta članek.

CSS

address {   
font-style: italic;  
display: block; }

ZNAČKE HTML 5

[<!--...-->](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_comment.html) Komentar

[<!DOCTYPE>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_doctype.html)  Tip Dokumenta

[<a>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_a.html) Hiperpovezava

[<abbr>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_abbr.html) an abbreviation

[<acronym>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_acronym.html) Ni podprto v HTML5

[<address>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_address.html) Kontakt o lastniku/avtorju dokumenta

[<applet>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_applet.html) Ni podprto v HTML5

[<area>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_area.html) Prostor v mapi slike

[<article>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_article.html) Članek

[<aside>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_aside.html) Vsebina na strani glavne vsebine

[<audio>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_audio.html) Audio vsebina

[<b>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_b.html) Krepki text

[<base>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_base.html) Določa osnovni naslov / cilj za vse relativne naslove URL v dokumentu

[<bdi>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_bdi.html) Izolira del besedila, ki bi lahko bil oblikovan v drugo smer od drugega zunanjega besedila

[<bdo>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_bdo.html) Prepiše smer texta

[<blockquote>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_blockquote.html) Odsek, ki je naveden iz drugega vira

[<body>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_body.html) Telo dokumenta

[<br>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_br.html) Prelom vrstice

[<button>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_button.html) Stiskajoči gumb

[<canvas>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_canvas.html) Narejeno za pripravo grafike..

[<caption>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_caption.html) Zajem tabele

[<cite>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_cite.html) Naslov dela

[<code>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_phrase_elements.html) Del računalniške kode

[<col>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_col.html) Določa lastnosti stolpcev za vsak stolpec z <colgroup> elementom

[<colgroup>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_colgroup.html) Določa skupino enega ali več stolpcev v tabeli za oblikovanje

[<command>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_command.html) Ukazni gumb, ki ga uporabnik lahko sklicuje

[<datalist>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_datalist.html) Določa seznam vnaprej določenih možnosti za vhodne kontrole

[<dd>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_dd.html) Opis elementa v opredelitvenem seznamu

[<del>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_del.html) Text ki je bil izbrisan iz dokumenta

[<details>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_details.html) Dodatne podrobnosti, ki jih lahko uporabnik prikaže ali skrije

[<dfn>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_phrase_elements.html) Opredelitev izraza

[<div>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_div.html) Sektor v dokumentu

[<dl>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_dl.html) Opredelitveni seznam

[<dt>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_dt.html) Nek izraz v odpredelitvenem seznamu

[<em>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_phrase_elements.html) Poudarjeno besedilo

[<embed>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_embed.html) shramba za uporabo zunanjih interaktivnih vsebin (plug-in)

[<fieldset>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_fieldset.html) Skupine ki so povezane z elementi v obliki

[<figcaption>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_figcaption.html) Zajetek za <figure> element

[<figure>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_figure.html) Določa samoshrambeno vsebino

[<footer>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_footer.html) noga za dokument ali sektor

[<form>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_form.html) HTML obrazec za uporabnike

[<h#>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_h.html) HTML naslovi

[<head>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_head.html) Informacije o dokumentu

[<header>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_header.html) Določa uvod ali skupinsko navigacijo elementov za dokument

[<hgroup>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_hgroup.html) Skupine na intervalu od <h1> do <h6>

[<hr>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_hr.html) Tematična sprememba v vsebini

[<html>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_html.html) Koren dokumenta HTML

[<i>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_i.html) Del besedila za alternativni glas ali razpoloženje

[<iframe>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_iframe.html) Inline frame

[<img>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_img.html) Slika

[<input>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_input.html) nadzor vnosa

[<ins>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_ins.html) Text ki je bil vstavljen v dokument

[<keygen>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_keygen.html) Ključni generator polj, obrazcov

[<kbd>](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/element_phrase_elements.html) Vnos iz tipkovnice

GLOBALNI ATRIBUTI HTML 5

* [accesskey](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_accesskey.html)*(character)* shortcut key to access an element
* [class](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_class.html) *(classname*) class specified in a style sheet
* [contenteditable](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_contenteditable.html) (true | false | inherit) whether a user can edit the content of an element or not
* [contextmenu](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_contextmenu.html) *(menu\_id*) context menu for an element. The value must be the id of a element < menu>
* [dir](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_dir.html) (ltr | rtl | auto) text direction for the content in an element
* [draggable](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_draggable.html) (true | false |auto) whether a user is allowed to drag an element or not
* [dropzone](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_dropzone.html) (copy | move | link) what happens when dragged items/data is dropped in the element
* [hidden](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_hidden.html) (hidden) that an element should be hidden
* [id](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_id.html) (*id*) a unique id for an element
* [lang](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_lang.html) *(koda jezika)* the language of the element's content
* [spellcheck](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_spellcheck.html) (true | false) if the element must have its spelling and grammar checked
* [style](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_style.html) *(pravila slogov)* medvrstično slogovno pravilo ali pravila
* [tabindex](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_tabindex.html) *(število*) tabulatorski vrstni red
* [title](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_title.html) *(besedilo)* dodatne informacije elementa

DOGODKI DOKUMENTOV

Events triggered for the window object (applies to the tag):

[onafterprint](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onafterprint.html) Uporabnik je tiskal dokument.

[onbeforeprint](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onbeforeprint.html) Uporabnik je zahteval tiskanje dokumenta.

[onbeforeunload](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onbeforeunload.html) Dokument je v fazi zapiranja.

[onblur](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onblur.html) Dokument je izgubil fokus.

[onerror](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onerror.html) Dokument ni bil uspešno naložen.

[onfocus](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onfocus.html) Dokument je v žarišču.

[onhashchange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onhashchange.html) Spemenil se je fragmentni naslov, ki sledi po znaku #.

[onload](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onload.html) Dokument je končal nalaganje.

[onmessage](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmessage.html) Dokument je prejel sporočilo.

[onoffline](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onoffline.html) Povezava ne dela.

[ononline](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ononline.html) Povezava se je vrnila.

[onpopstate](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onpopstate.html) Uporabnik je navigiral zgodovino seje.

[onresize](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onresize.html) Pogled dokumenta je bil povečan/pomanjšan.

[onstorage](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onstorage.html) Shranitveno območje se je spremenilo.

[onunload](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onunload.html) Dokument odhaja.

DOGODKI S TIPKOVNICO

Dogodek je sprožen iz tipkovnice (vključeno v vse elemente HTML5):

[onkeydown](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onkeydown.html) Skripto zaženemo ko pritisnemo tipko dol.

[onkeypress](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onkeypress.html) Skript se začne ko pritisnemo in spustimo tipko.

[onkeyup](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onkeyup.html) Skript se začne, ko spustimo tipko.

DOGODKI OBRAZCEV

Dogodki sproženi z akcijami znotraj HTML obrazcev (vključeni v vse elemente html5, vendar so bolj pogosti pri obrazcih elementih).

[onblur](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onblur.html) Skript se zažene, ko element zgubi pozornost.

[onchange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onchange.html) Skript se začne, ko se spremeni element.

[oncontextmenu](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_oncontextmenu.html) Skript se začne, ko se sproži ozadje.

[onfocus](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onfocus.html) Skript se zažene, ko element dobi fokus.

[onformchange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onformchange.html) Skript se zažene, ko se obrazec spremeni.

[onforminput](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onforminput.html) Skript se zažene, ko obrazec dobi neko vsebino.

[oninput](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_oninput.html) Skript se zažene ko element dobi neke zunanje podatke.

[oninvalid](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_oninvalid%20.html) Skript se zažene, ko je element neveljaven.

[onselect](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onselect.html) Skript se zažene ko je izbran element.

[onsubmit](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onsubmit.html) Skript se zažene, ko je obrazec oddan.

MIŠKINI DOGODKI

Events triggered by a mouse, or similar user actions (vključeno v vse elemente HTML5):

[onclick](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onclick.html) Skript se zažene na miškin klik.

[ondblclick](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondblclick.html) Skript se zažene na miškin dvojni klik.

[ondrag](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondrag.html) Skript ki se zažene, ko potegnemo nek element.

[ondragenter](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondragenter.html) Skript se zažene, ko je element potegnjen do neke ciljne točke.

[ondragend](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondragend.html) Skript se zažene, ko končamo z premikanjem.

[ondragleave](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondragleave.html) Skript, ki se začne ko element zapusti ciljno točko.

[ondragover](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondragover.html) Skript, ki se začne, ko element potegnemo skozi ciljno točko.

[ondragstart](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondragstart.html) Skript se začne takoj, ko začnemo z vlečenjem elementa.

[ondrop](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondrop.html) Skript, ki se začne, ko spustimo element, ki smo ga povlekli.

[onmousedown](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmousedown.html) Skript se začne, ko stisnemo miškin gumb.

[onmousemove](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmousemove.html) Skript se začne ko premaknemo miškin kazalec.

[onmouseout](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmouseout.html) Skript se začne, ko se s kazalcem pomaknemo iz elementa.

[onmouseover](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmouseover.html) skripta se začne, ko se z kazalcem pomaknemo skozi element.

[onmouseup](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmouseup.html) Skript se začne, ko se spusti miškin gumb.

[onmousewheel](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onmousewheel.html) Skript se začne, ko zavrtimo kolešček na miški.

[onscroll](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onscroll.html) Skript se začne, ko zavrtimo kolešček od miške.

MEDIJSKI DOGODKI

Dogodki sproženi preko medijev kot so videii, fotografije in zvok (velja za vse elemente HTML5, a najbolj pogosto in uporabno je v medijskih elementih kot so audio, embed, img, object in video):

[onabort](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onabort.html) Script se izvede ob prekinitvi

[oncanplay](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_oncanplay.html) Script se izvede, ko je datoteka pripravljena za predvajanje (ko je dovolj naložena za začetek).

[oncanplaythrough](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_oncanplaythrough.html) Script se izvede, ko se celotna datoteka lahko predvaja ne, da bi se vmes ustavljala.

[ondurationchange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ondurationchange.html) Script se izvede, ko se spremeni dolžina medija.

[onemptied](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onemptied.html) Script se izvede, ko je datoteka nedostopna.

[onended](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onended.html) Script se izvede, ko je medij dosegel konec.

[onerror](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onerror.html) Script se izvede, ko nastane napaka med nalaganjem datoteke.

[onloadeddata](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onloadeddata.html) Script se izvede, ko so medijski podatki naloženi.

[onloadedmetadata](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onloadedmetadata%20.html) Script se izvede, ko so meta podatki (podatki o dolžini in trajanju) naloženi.

[onloadstart](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onloadstart.html) Script se izvede, ko se datoteka prične nalagati, preden je karkoli naloženo.

[onpause](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onpause.html) Script se izvede, ko je medij zaustavljen ali programsko ali preko uporabnika.

[onplay](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onplay.html) Script se izvede, ko je medij pripravljen za predvajanje.

[onplaying](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onplaying.html) Script se izvede, ko se je medij začel predvajati.

[onprogress](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onprogress.html) Script se izvede, ko je brskalnik v procesu pridobivanja medijskih podatkov.

[onratechange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onratechange.html) Script se izvede vedno, ko se način predvajanja spremeni.

[onreadystatechange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onreadystatechange.html) Script se izvede vedno, ko se spremeni stanje pripravljenosti.

[onseeked](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onseeked.html) Script se izvede, ko je iskalna lastnost nastavljena na napačno.

[onseeking](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onseeking.html) Script se izvede, ko je iskalna lastnost nastavljena na pravilno.

[onstalled](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onstalled.html) Script se izvede, ko brskalnik ne more prenesti podatkov zaradi kakršnihkoli razlogov.

[onsuspend](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onsuspend.html) Script se izvede, ko se prenašanje medijskih podatkov ustavljeno preden so naloženi zaradi kakršnikoli razlogov.

[ontimeupdate](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_ontimeupdate.html) Script se izvede, ko se predvajalna pozicija spremeni.

[onvolumechange](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onvolumechange.html) Script se izvede vedno, ko se glasnost spremeni.

[onwaiting](http://www.egradiva.net/moduli/spletne_aplikacije/html/90_html5_referenca/atribut_onwaiting.html) Script se izvede vedno, ko se medij čaka, vendar se pričakuje nadaljevanje (npr. ko se nalagajo podatki).

# KAJ JE HTML?

**HTML** je kratica za **HyperText** **Markup Language** in je jezik za objavljanje na svetovnem spletu. Je podmnožica mednarodnega standarda za izmenjavo elektronskih dokumentov imenovan **SGML** (Standard Generalized Markup Language). Računalniki potrebujejo programsko opremo, da lahko ta jezik razumejo. Ta programska oprema vključuje grafične brskalnike, tekstovne in govorne naprave.

HTML je programski jezik. Vendar za programiranje ni potrebno matematično znanje in sploh ni potrebno znanje, ki je potrebno za druge proramske jezike (kot so C++, Java, Pascal in drugi). HTML je označevalni jezik (ang. Markup Language).

S posebnimi elementi se označi besedilo, ki pove brskalniki, kako naj strukturo dokumenta prikaže. Obstaja način, kako označiti naslove, odstavke, sezname tabele in še druge elemente. HTML je hipertekstovni jezik (ang. Hypertext Language). Hiperttekst upoorablja način na sklicevanje drugih spletnih dokumentov, ki so med seboj povezani. Ko se v spletnem dokumentu klikne povezavo, se v resnici kliče hiperpovezavo.

HTML ponuja:

* elektronsko objavo dokumentov z besedili, tabelami, fotografijami...
* preusmeritev in napotitev na druge informacije s pomočjo hiperpovezav,
* uporabo obrazcev, ki omogočajo iskanje po spletu, naročanje izdelkov...
* dodajanje video izrezkov, zvoka in drugih aplikacij v dokumente

# KAJ JE XML?

XML je kratica za Extensible Markup Language. XML je bil narejen za povečanje zmogljivosti in prilagodljivosti jezika SGML, ki izključuje zapletenost le-tega. Kljub omejenosti pa ohranja večino sposobnosti in moči SGML.

# KAJ JE XHTML?

XHTML je naslednji korak v razvoju jezikov, ki se uporabljajo za objavljanje v spletu. Je nadomestilo za odhajajoči jezik HTML. Je jezik, ki zadostuje vsem pravilom jezika XML. Dokumente XHTML 1 je mogoče napisati tako, da so popolnoma združljivi z različicami jezika HTML 4. V pripravi pa je različica HTML5, W3C je opustil razvoj XHTML 2.

* Uporabljamo ga predvsem zaradi lažje strojne obdelave in analize hiperteksta. Dele hiperteksta lahko vstavljamo kot atributne multimedijske podatke v podatkovno bazo.
* Iz XHTML hipertekstov lažje naredimo izvlečke in določimo pomen teksta.
* Prihodnost XHTML je predvsem v povezavi s semantičnim spletom (Sematic Web).
* Specifikacije trenutnega stanja razvoja verzije HTML 5 najdete na <http://www.w3.org/>

# ZGODOVINA HTML

HTML je iznašel Tim Berners-Lee, ko je delal v [CERN](http://www.cern.ch/). Popularen je postal šele po izdelavi brskalnika, ki je bil razvit v NCSA. Po skromnem začetnem obdobju (1990-93) je prišlo nato do eksplozivne rasti svetovnega spleta. V toku razvoja je prišlo do mnogih razširitev v raznih smereh. Svetovni splet ponuja avtorjem več možnih načinov za prikaz vsebin v spletu. To še spodbuja motivacijo za iskanje skupnih rešitev in standardov.

HTML 2.0 (november 1995) je bil razvit pod iniciativo Internet Engineering Task Force (IETF) in se je pričel uporabljati konec leta 1994. HTML+ (1993) in HTML 3.0 (1995) sta precej razširila jezik HTML v primerjavi s prejšnjimi različicami. Kljub temu, da ni bil dosežen dogovor o standardizaciji in so obstajale različice jezikov, se je pričel zaradi svojih zmožnosti široko uporabljati. Z ustanovitvijo World Wide Web Consortium (W3C) je delovna skupina za HTML uspela najti skupen jezik med raznimi razvijalci in je v letu 1996 pričela s standardizacijo HTML. Rezultat je bil HTML 3.2 (januar 1997).

Večina uporabnikov se strinja, da se mora HTML pravilno prikazovati v različnih brskalnikih in na različnih platformah. To je pa tudi želja razvijalcev, saj na ta način za nižjo ceno izdelajo strani - ni potrebno razvijati različic.

Vsaka različica HTML je kompromis med industrijo računalniške opreme, internetnimi ponudniki, razvijalci spletnih strani, Internetnimi trgovci in drugimi, ki vlagajo svoja sredstva v Internet. Vsak si želi, da bi si s kar najnižjimi stroški uspel čim daljši čas igrati svojo vlogo.

HTML je bil razvit z vizijo, da bo lahko uporabljen na mnogih napravah, ki so priključene v splet: računalnike z različnimi resolucijami in barvnimi globinami, telefone, razne industrijske naprave...

HTML, ki ji bil standardiziran 24.12.1999, nosi oznako 4.01. Naslednja različica jezika HTML ima ime XHTML1.

W3C je bil pravzaprav popolnoma zaskrbljen kar se tiče prehoda, ki ga je bilo potrebno namestiti med HTML in XML. V maju leta 1998 je bila sklicana javna delavnica s predmetom pogovora o XHTML blizu San Francisca, California. Med dvodnevnim dogodkom so razvijalci svetovnega spleta, prodajalci programske opreme in avtorji priporočil za HTML in XML razpravljali, kako bi lahko prenesli HTML v svet XML. Rešitev, ki je bila predlagana, je bil prehodni jezik , ki bi zagotovil most med HTML in XML. Ta jezik je danes imenovan XHTML 1.

Ko smo pričakovali novo različico XHTML2, ki jo propravljajo že več kot 10 let, se je pojavil še en predlog z imenom HTML 5 in bo v skladu z smernicami XHTML.

# HTML 5

V HTML je prva vrstica:

<!doctype html>

## Novi elementi

Novi pomensko usmerjeni bloKOVNI elementi v HTML 5 so:

* **section** – predstavlja del vsebine, ki ne vsebuje navigacije in je setavljena iz več člankov
* **hgroup**  - predstavlja glavo sekcije
* **header** – predstavlja glavo dokumenta
* **footer** – predstavlja nogo dokumenta
* **nav** – Predstavlja del, namenjen navigaciji
* **article** – članek znotraj sekcije
* **aside**  – za opis opomb, komentarjev...
* **dialog**   - uporablja se za opis konverzacije
* **figure**  – uporablja se za vsebino večpredstavnosti
* **video**  - uporablja se za  prikaz video vsebine
* **audio**  - uporablja se  za predvajanje avdio vsebine
* **embed** - uporablja se za vsebine, ki jih lahko prikazujemo z vstavki (plugins) v brskalniku
* **canvas**  - uporablja se za prikaz (renderiranje) animiranih ( 2D, 3D) vsebin ( igre, grafika, interaktivne vsebine)
* **ruby, rt, rp**   - uporablja se za prikaz vsebin v vzhodno azijskih jezikih  (japonščina,  kitajščina, korejščina).
* **output**  - uporablja se za izpise rezultatov (npr. pri izračunih, ki se izvedejo na podlagi vhodnih podatkov v obrazcu).

# HTML 5

## Novi pomensko usmerjeni medvrstični elementi v HTML 5.0

* **mark** – označevanje besedila (poudarjanje),
* **time** – opis časovno opisanih vsebin,
* **meter** – opis kvantitativno opisanih vsebin (lastnosti value, min, low, high, max, optimum),
* **progress** – opis napredka pri nalaganju vsebine.

## Interaktivnost

* **details  -** izpis podrobnosti
* **menu** – izpis menija
* **command –** izpis alineje v meniju

# HTML 5

## Dodatne lastnosti pri znački <input> v obrazcih

* datalist  - predstavlja seznam elementov, ki jih lahko opcijsko izberemo v obrazcu
* tel  - telefonska številka
* search – iskalni niz
* url
* email
* datetime
* date
* month
* week
* time
* datetime-local
* number
* range
* color

## Vključevanje MathML in SVG

* V HTML 5 lahko direktno pišemo MathML in elemente SVG.
* V HTML 5 so izločili značke basefont, big, center, font, s, strike, tt, u, frame, frameset, noframes, acronym, abbr, applet (namesto njega se uporablja object), isindex in dir.
* Lastnosti , dir, id, lang, style, tabindex in title se lahko uporabljajo pri vseh elementih.

# Element

Elementi so strukture, ki določajo **zgradbo** HTML dokumentov. Elementom lahko predpišemo dodatne atribute (lastnosti).

Vsak element je sestavljen iz treh delov: začetne značke, vsebine in končne oznake.

Značka (ang. tag) je besedilo, ki se nahaja med znakoma **<** in **>**. **Zaključna** značka vsebuje znak / takoj za znakom <. Npr. **element span** element vsebuje začetno značko, **<span>**, in zaključno značko, **</span>**.

Začetna in zaključna značka omejujeta območje vsebine elementa span:

**<span>**To je oblikovano malo drugače.**</span>**

Vsebina elementa se nahaja med začetno in končno značko.

## Male črke

Značke v XHTML lahko zapišemo izključno malimi črkami.

<IMG onclick="fx()">

// to je narobe

<IMG ONCLICK="fx()">

// to je narobe

<img onclick="fx()">

// to je narobe

<img onclick="fx()">

// to je pravilno

V HTML uporaba malih in velikih črtk ni pomebna in so **vsi** primeri **pravilni**.

## **Gnezdenje in odprte značke**

## Gnezdenje

Posamezni elementi se smejo gnezditi. Če se element span pojavi v elementu p, potem se mora začetna značka <p> pojaviti pred začetno značko <span>. Zaključna značka </span> se mora pojaviti pred zaključno značko </p>. V podanem primeru je element span otrok starša p. Za vsak element je določeno, kdo so lahko njegovi starši in njegovi otroci. Ne moremo gnezditi elemente znotraj drug drugega, če to preprečuje DTD, vrednosti atributov morajo ustrezati določeni zalogi vrednosti in dokument mora ustrezati vsem drugim omejitvam podanim v DTD.

<p> <span> To je to </p> </span> // to je narobe, span je znotraj p

<span> <p> To je to </p> </span> // narobe, p ne more biti otrok span <p>

<span> To je to </span> </p> // to je pravilno

## Odprte značke

Nezaključeni elementi v XHTML niso dovoljeni. Posamezni elementi omogočajo, da lahko izpustimo končno značko. To se elementi, ki nimajo vsebine. V tem primeru se v začetno značko na koncu napiše znak za zaključek značke **</>**. Pri tem je neznano zakaj nastal problem v Netscape Navigatorju z napačno razumljenim ukazom <br/>. Nihče ne ve zakaj, ampak če je zapisano kot **<br />** (s presledkom) deluje pravilno.

<br>

// to je narobe, manjka zaključek

<br> </br>

// to je narobe, prazni elementi nimajo ločenih značk

<br/>

// to je narobe, manjka presledek

<br /> // to je pravilno

V HTML zaključevanje ni pomebno in je pravilno tudi:

<br> // V HTML je to dovoljeno, ne pa v XHTML

## Vrste elementov

V HTML4/XHTML1

Poznamo blokovne in medvrstične elemente HTML.

## Blokovni elementi

**Blokovni** elementi so podobni odstavkom, kar pomeni, da se njihova vsebina vedno prične na začetku nove vrstice in lahko zasede več vrstic. Običajno še dodajo prazen prostor pred in za elementom.

Takšni elementi so na primer: h1, h2, h3, h4, h5, h6, p, div, dl, ol, ul, div, p, pre, hr, blockquote, address, script, noscript form, fieldset.

## Medvrstični elementi

Elementi so **medvrstični** (ang. inline, text-level), če se prikaz nadaljuje v isti vrstici. Takšni elementi ne dodajo prehod v novo vrsto niti praznega prostora okoli elementom. Torej so bolj podobni znakom, saj lahko nastopajo kjerkoli v vrstici.

Takšni elementi so na primer: input, textarea, select, button, label a, strong, em, tt, span, br, big, small, sub, sup, img, object, abbr, acronym, cite, code, dfn, kbd, samp, var, ins, del, bdo, script, noscript

## Drugi elementi

Obstaja tudi nekaj elementov, ki ne pripadajo nobeni od naštetih skupin. To so tisti elementi, za katere je točno določeno, kje lahko nastopijo. Sem spadajo korenski element, elementi, ki lahko nastopijo samo v glavi dokumenta, samo v tabeli, seznamu in drugje.

Ti elementi so na primer: html, head, body, title, meta, link, style, base, caption, col, colgroup, thead, tfoot, tbody, tr, td, th, dt, dd, param, area, option, optgroup, legend.

## Atributi

Atributi podrobneje določajo različne lastnosti elementov. Lahko so izključno znotraj uvodne značke elementa HTML. Atribut ima svoje ime, ki je določeno s pravili W3C in vrednostjo, ki je v primeru XHTML vedno nahaja med narekovaji. Atributi so obvezni in opcijski.

Na primer, element img vsebuje **obvezni** atribut src, ki določa lokacijo (mesto) slike na pomnilniškem mediju, atribut alt pa določa nadomestno besedilo, če slika ne obstaja ali je ni mogoče prikazati.

Opcijski atribut width določa širino slike:

**<img src="logo.gif" alt="logo" width="320" height="200" />**

Atribute se vnaša samo v začetne oznake elementov za ime elementa v obliki:

**ime\_atributa="vrednost atributa".**

Atributi **morajo** imeti svojo vrednost v XHTML:

**<input checked="checked" />**

V HTML je dolovljeno vrednost logičnih atributov izpustiti. Na primer

**<input checked />**

je enakovedno prej napisanemu.

## Dostopanje in spreminjanje z JavaScriptom

Vrednosti atributov lahko preberemo in v nekaterih primerih spremenimo. Potrebno je uporabiti eno izmed metod DOM:

getElementById

getElementsByTagName

getElementsByClassName

getElementsByName

Vrednost se bere na način:

**element=document.getElementById(id) lastnost=element.getAttribute(atribut)**

nastavi se z metodo:

**element=document.getElementById(id)**

**element.setAttribute(atribut,vrednost)**

## Pravilno pisanje atributov

## Velikost črk

Imena atributov XHTML morajo biti pisana z malimi črkami:

<table WIDTH="100px"> // to je narobe

<table Width="100px"> // to je narobe

<table width="100px"> // to je pravilno

V HTML velikost črk ni pomembna in so vsi primeri **pravilni**.

## Okrajšave

Okrajšave atributov v XHTML so prepovedane. Atributi vrednosti logičnega tipa morajo biti "raztegnjeni". Povedano malo drugače pomeni to, da mora imeti vsak atribut svojo vrednost. Atribut logičnega tipa pa je atribut, ki je vklopljen samo v njegovi prisotnosti. Pogosto se uporabljajo okrajšave, ki pa so dovoljene samo v HTML.

<input checked />

// to je narobe

<input checked="checked" />

// to je pravilno

<option selected>

// to je narobe

<option selected="selected">

// to je pravilno

# HTML - STRUKTURA

## Deklariranje dokumenta

Kako so urejeni medsebojni odnosi v našem življenju določajo zakoni in norme. V primeru spletnih strani je to določeno z DTD.

Spletne strani morajo vsebovati deklaracijo tipa dokumenta (DTD) DOCTYPE, če se želi, da bodo predstavljale pravilne dokumente.

Kakorkoli, zavedati se je potrebno, da bodo morda novejši brskalniki (na primer Internet Explorer 6, Netscape Navigator 7, Opera 7) obravnavali vaš dokument različno, odvisno od deklaracije <!DOCTYPE>. Če brskalnik prebere dokument z DOCTYPE, ga bo morda obravnaval kot pravilnega. Napačno formiran XHTML je lahko prikazan drugače, kot brez deklaracije DOCTYPE.

## Pregledovanje kode HTML z DTD

Spletni dokument je pregledan zglede na definicijo DTD. V DTD so napisana pravila za posamezne tipe dokumentov. Za preverjanje pravilnosti spletne strani se lahko uporabi TIDY. Dave Raggettov HTML TIDY je brezplačno orodje za čiščenje kode HTML. Lahko se ga uporabi tudi za težko berljive, dokumente ustvarjene z HTML urejevalniki in prirejevalniki. Lahko pa se obišče preverjevalnik na spletni strani validator.w3.org

## Tipi dokumentov XHTML

Preden začnemo ustvarjati XHTML dokument moramo izbrati tip dokumenta. Kako se odločiti kateri tip dokumenta bomo uporabljali?

Če se lotite ustvarjanja novega spletišča, je edina prava rešitev XHTML 1.0 Strict ali pa HTML 5, ki je v razvojni fazi.

## XHTML 1.0 Strict

XHTML 1.0 Strict ne vsebuje nobenih predstavitvenih informacij. Če ima opravka s slogom, videzom in prikazom, so navodila podana brskalniku preko CSS ali XSL. V mislih lahko imamo nekaj preprostega, kot je izbrati barvo za tekst ali naslov, lahko pa tudi položaj slik v samem dokumentu. Za temi nastavitvami pa se skrivajo veliko bolj zapleteni ukazi, ki jih navajamo v CSS ali XSL. To so npr. velikost, poravnanost, slogi, vizualni pogledi tabel kot tudi zamiki, razmiki in pomiki. Vsi ti ukazi so bili odstranjeni iz različice jezika Strict, kar pomeni da v jeziku ni več elementov in atributov, ki bi upravljali z zgoraj omenjenimi lastnostmi besedila. Ti ukazi so namesto v brskalniku oziroma dokumentu zapisani v CSS.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

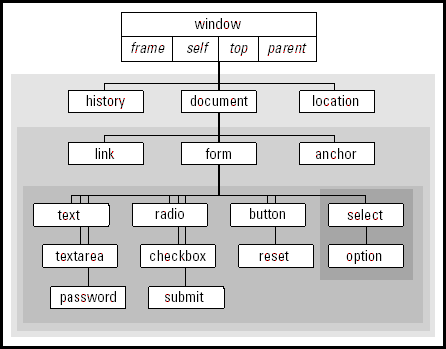
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

## HTML5/XHTML5

<!DOCTYPE html>

## **DOM**

JavaScript lahko dostopa tudi do predmetov, ki se nahajajo v brskalniku (npr. window, navigator, history, screen). Za to pa potrebuje posebno zbirko, imenovano **BOM** (Browser Object Model). Vsak brskalnik ima svojo zbirko. Ta je pri četrti generaciji spletnih brskalnikov povzročila neskladje med brskalniki in tudi pri pisanju programov v JavaScript. Netscape je poskušal vsiliti s svoj model, imenovan **JSOM** (JavaScript Object Model). Microsoft je predlagal standardizacijo svojega modela, ki je bil boljši od Nescapovega. Netscape dolgo ni hotel tega prevzeti in je zato morda izgubil bitko brskalnikov. World Wide Web Consortium (**W3C**) je ustavil razvoj različnih predmetnih modelov brskalnikov in možnosti dostopanja do posameznih predmetov. Predpisal je standardni model, imenovan **DOM** (Document Object Model). Več informacij lahko poiščete na naslovu W3 DOM. Med različnimi organizacijami so potrebna usklajevanja, vsak načrtovalec brskalnikov pa si izmišljuje nove spremembe, ki jih nato poskuša uveljaviti kot standard.



DOM da vsakemu odstavku, vsaki podobi, tudi vsakemu posameznemu členu v vašem dokumentu različno, individualno entiteto z možnostjo naslavljanja, manipuliranja ali celo celotne zamenjave kadarkoli in kakorkoli želite.

Objekti so, enostavno povedano, individualni delčki, ki tvorijo spletno stran. Seveda pa mora imeti vsak predmet določene postopka, ki definirajo kako se predmet obnaša v določeni situaciji. Tako postane predmet del DHTML z možgani.

Model pa je povezava teh predmetov v celoten predmet oziroma dokument kot tak. Vsak predmet določuje svoj pomen, svojo lastnost katero lahko povežemo z drugo lastnostjo, ki se nahaja v drugem predmetu. Objekte lahko spretno povežemo in tako dobimo splet vseh lastnosti dokumenta, ki nam določijo stran.

# SORODSTVO

Ogledali si bomo naslednja razmerja:

* prednik, potomec, starš, otrok, sorojenec

Za pregledovanje dokumenta ima Firefox vgrajem DOM Inspektor.

## POTOMCI

Elementi, ki so vgnezdeni v starša, se imenujejo**potomci**(descendant) tega elementa.

## PREDNIKI

Če element vsebuje vgnezdene elemente, se imenuje **prednik** (ancestor) teh elementov.

## STARŠI

Če element vsebuje vgnezdene elemente, se imenuje **starš** (parent) teh elementov. Starši so predniki v prvem kolenu ali neposredniki predniki.

## OTROCI

Elementi, ki so vgnezdeni neposredno v starša, se imenujejo**otroci**(children) tega elementa.

## SOROJENCI

Elementi, ki si zaporedno sledijo in niso med seboj vgnezdeni, se imenujejo sorojenci (sibling). Element A se imenuje soreojnec elementa B, samo takrat, če ima skupaj z elementom B istega starša C. Element A je predhodni sorodnik (preceding sibling) elementa B, če se nahaja v dokumentnem drevesu pred elementom B. Element B je naslednji sorodnik (following sibling) elementa A, če se v dokumentnem drevesu nahaja za elementom A.

# ZNAKI

## KODNE TABELE

Nekateri znaki v HTML imajo torej poseben pomen. Ti znaki so < (manjši), > (večji), " (dvojni narekovaj) in & (in). če jih želimo uporabiti v besedilu, moramo uporabiti njihova imena, določena s standardom SGML, na primer takole: &lt;, &gt;, &quot; in &amp;.

Znakovna referenca so številke ali simbolična imena za znake, ki se vključujejo v dokumente HTML. Uporabljajo se za redko uporabljene znake, ki bi jih avtor strani težko vnesel. Znakovna entiteta se prične z znakom &, oznako in se zaključi s podpičjem ;.

Nekaj primerov:

* &lt; predstavlja znak <
* &gt; predstavlja znak >
* &quot; predstavlja znak "
* &#268; (decimalno) predstavlja veliko črko C s kljukico nad njo: Č.
* &#1048; (decimalno) predstavlja veliko črko I v cirilici: И.
* &#x6C34; (šestnajtiško) predstavlja kitajski znak za vodo.

Vsak dokument SGML (in tudi vsak dokument HTML) je zaporedje znakov nekega nabora. Računalniški sistem prepozna vsak znak po njegovem položaju. Na primer v naboru ASCII, so kode posicij 65, 66, and 67 obravnajajo in prikažejo kot znaki 'A', 'B', and 'C'.

Znakovni nabor ASCII ni popoln za prikaz velikih števila informacijskih sistemov kot je na primer svetovni splet. Tako HTML uporablja več zbirk znakovnih naborov, ki jih imenuje Universal Character Set (UCS) (določeni so s standardom [ISO10646]). Ta standard določa repertoar tisočerih znakov, ki so uporabljeni za komunikacijo po celem svetu.

## KODIRANJE ZNAKOV

Specifikacija, ki določa kodiranje znakov je poznana z različnimi imeni na različnih področjih (kar pogosto povzroča zmedo).

Atribut charset določa nabor znakov za kodiranje, ki jih zna postopek pretvarjanja spremeniti iz zaporedja bitov v zaporedje znakov. Ta postopek je podoben kot pri predstavitvi spletnih strani: strežnik pošlje brskalniku zaporedje bitov in ta jih mora predstaviti kot zaporedje znakov. Način pretvarjanja je odvisen od metode kodiranja, od enostavnega preklapljanja do posebnih algoritmov.

## Določanje znakov za kodiranje

Kako strežnik ve katero kodiranje je uporabljeno za dokument, ki ga nudi? Nekateri strežniki preverijo nekaj prvih bytov v dokumentu, preverijo zbirko poznanih načinov kodiranja. Mnogo modernih spletnih strežnikov ponuja spletnim mojstrom več nadzora nad določanjem znakovnih naborov.

Kako brskalnik ve, katere vrste kodiranje je bilo uporabljeno? Strežnik lahko pošlje to informacijo. Zelo pogosto je podatke o kodiranju v privesku meta, ki se nahaja v glavi dokumenta html. V atributu Content-Type je navedena vrsta kodiranja, npr.:

**<meta http-equiv="content-type" content="text/html; charset=EUC-JP" />**

V HTML 5 je to zapisano enostavneje:

**<meta charset="EUC-JP" />**

S korensko značko dokumenta html je mogoče določiti jezik in s tem posledično tudi vrsto kodiranja.

**<html lang="si" xml:lang="si" xmlns="http://www.w3.org/1999/xhtml">**

Kodiranje je lahko določeno tudi z deklaracijo XML. Ker je dokument XHTML tudi dokument XML, se lahko zapiše v prvo vrstico dokumenta:

**<?xml version="1.0" encoding="si" ?>**

## KODIRANJE ZNAKOV V URI

Moderni opearcijski sistemi dovoljeujejo dolga imena, ki lahko vsebujejo "čudne znake" kot so predsledki, ę, ĭ, p, š in druge.

## KODIRANJE ZNAKOV V URI

Moderni opearcijski sistemi dovoljeujejo dolga imena, ki lahko vsebujejo "čudne znake" kot so predsledki, д, Ġ, Σ, š in druge. Pri tem morate paziti na razlikovanje malih in velikih črk.

Kako zapisati URI za datoteko z imenom

San Ġiljan.jpg

Recimo, da se datoteka nahaja na strežniku www.malta.com v mapi images.

Če boš zapisal:

[http://www.malta.com/images/San Ġiljan.jpg](http://www.malta.com/images/San%20Ġiljan.jpg)

to ni pravilni zapis, čeprav imajo datoteke takšno ime. Čudna znaka je potrebno nadomestiti s številskima ekvivalentoma po tabeli Unicode:

<http://www.malta.com/images/San%20%120iljan.jpg>.

Pri uporabi imen se raje izogibajte "čudnim znakom. Manj težav bo, če boste sliki dali ime

san\_giljan.jpg.

## RELATIVNI URI

Enostavno povedano, je relativni URI tisti, ki potrebuje nek proces, da se izračuna. To je lokalni URI, ki mu nekatere informacije manjkajo. Običajno so naslovi v istem imeniku ali pa se uporabljajo posebne sekvence, npr. ../. Relativni se imenuje zato, ker se URI izračuna glede na URI trenutnega vira.

## Izračun relativnega URI

Relativni URI potrebuje za pravilen izračun le-ta URI trenutnega vira. Z nekaj operacijami se relativni naslov pretvori v absolutnega, ki je nato uporabljen za iskanje novega vira.

Relativni URI je običajno lokalni URI. Prvi vir za izračun je trenutni URI. Nato se uporabijo naslednji postopki:

* Iz naslova trenutnega lokalnega URI, se odstrani ime datoteke, razen če ni to direktorij URI.
* Za vsako zaporedje ../ se na desni začetni (desni) strani odreže en imenik od trenutnega imenika URI
* Doda se relativni lokalni URI in se sestavi v celoto.

# CSS

## ZAKAJ UPORABLJAMO PREKRIVNE SLOGE

Prekrivni slogi (ang. Cascading Style Sheets - CSS) je mehanizem, ki pove spletnemu brskalniku, kako naj prikaže HTML dokument. CSS struktuira vsebino HTML dokumenta in ji da želeno obliko.

V prazgodovini spleta se je z večino značk, besedilo in druge elemente dokumenta HTML oblikovalo logično. Določilo se je, kaj določen del predstavlja (slika, tabela, vrstica v tabeli, polje v tabeli, seznam, točka seznama, odstavek, črta, indeks, eksponent, naslov, aktivna povezava ...), nič pa se ni ukvarjalo z obliko teh elementov. Dovolili s(m)o, da jih je brskalnik oblikoval po svoje.

Če ste sredi zadnjega desetltja preteklega tisočletja izdelovali spletne strani, ste uporabljali značke <font> in druge, da ste lahko oblikovali spletne strani. Te so vam omogočile, da so dokumenti videti tako, kot ste si želeli. Če pa ste se kasneje odločili zamenjati videz vseh strani v spletišču (četudi samo barvo ozadja), ste morali popraviti vsako stran posebej.

Prekrivni slogi predstavljajo rešitev tega problema. Kadar določite slogovno predlogo in jo povežete z vsemi stranmi v spletišču, imate lahek nadzor nad celotnim spletiščem. Če želite spremeniti videz, je potrebno spremeniti samo datoteko s prekrivnim slogom.

Z uporabo slogov lahko določamo, kako naj pregledovalnik oblikuje posamezne elemente našega dokumenta. Določiti je mogoče celo vrsto oblikovnih lastnosti, med katere spadajo ozadje, robovi, razmiki, odmiki, pisava, poravnava, barva

## KAKO BO VIDETI STRAN

Dandanes si ne bi smeli delati problemov ali brskalniki podpirajo prekrivne sloge ali ne. Prva priporočila za CSS je W3C izdal že davnega leta 1996. Prvi jih je podprl Internet Explorer, nato Opera, Netscape se je izogibal CSS in hotel uveljaviti svoje JSS.

Danes vsi brskalniki podpirajo CSS. CSS Level 1 je uveljavljen v vseh. Z redkimi izjemami velja tudi za CSS Level 2. Že nekaj let je tik pred izidom CSS3. Brsklaniki podpirajo že precej novosti.

Kot je običajno pa se v vseh brskalnikih pojavljajo napake in brskalniki posameznih pravil ne tolmačijo pravilno.

To predstavlja velik problem za spletne oblikovalce. Nekateri oblikovalci uporabljajo JavaScript, da prilagodijo strani posameznemu brskalniku in različici in tako izkoristijo prednosti posameznih brskalnikov. Vendar pa je običajno boljši konservativnejši pristop. Pazljivo se izdela predloga strani in se preskusi v vseh brskalnikih. V kolikor kateri brskalnik kakšnega pravila ne podpira, ga pri oblikovanju strani ne uporabimo.

## Različice

Trenutna različica CSS je 2.1. Brskalniki, ki podpirajo to različico, morajo pravilno prebrati tudi pravila CSS 1.0. Brskalniki, ki podpirajo CSS 1.0, morajo prebrati dokumente v CSS 2.1, pri tem pa preskočijo vsa pravila, ki jih ne prepoznajo.

Veljavna slogovna predloga temelji na plasti CSS, ki je uporabljena. Vsak veljaven dokument CSS1, je tudi veljaven dokument CSS 2.1, čeprav se nekatera pravila CSS1 malenkostno razlikujejo od tistih v CSS 2.1. Dokument CSS2 pa ni nujno veljaven dokument CSS 2.1 in obratno, saj so nekatera pravila CSS 2.1 dodana in nekatera odvzeta.

Mnogo pravil razvojne različice **CSS3** podpira že precej brskalnikov. Nekatera so še v razvojni fazi, zato jih v datoteki CSS pišemo s predponami -moz-, -webkit-, -ms in drugimi, odvisno od razvojnega orodja.

# KAJ JE CSS

* CSS je prekrivni slog.
* CSS je kratica za Cascading Style Sheets.
* Slog lahko določamo za vsak element v HTML.
* Slogi so običajno shranjeni v slogovnih predlogah (datoteka \*.css).
* Slogi so bili dodani v različico HTML 4.0 za boljšo preglednost spletnih dokumentov.
* Zunanje prekrivni slogi vam lahko prihranijo veliko dela.

## Prekrivni Slogi lahko prihranijo veliko časa

Slogi v HTML določajo, kako so elementi v HTML oblikovani. Prekrivni slogi so običajno shranjeni v posebnem dokumentu, zaradi lažje preglednosti. Eno zunanjo slogovno predlogo lahko uporabimo potem na vseh drugih straneh, če jih je več. Vsi, ki smo že kdaj skušali spreminjati recimo pisavo v vseh elementih HTML, vemo koliko truda lahko prihrani slogovna predloga.

Prekrivni slogi CSS nam omogočajo, da nadzorujemo slog strani samo preko enega dokumenta, in to je ravno največja prednost. Ker so elementi običajno globalno povezani med sabo, s spremembo le enega pravila zamenjamo na primer ozadje vseh spletnih strani spletišča.

# SINTAKSA

prekrivni slogi so zgrajena iz niza pravil. Pravilo je zgrajeno iz dveh delov: iz selektorja in iz deklaracije. Deklaracija je sestavljena iz lastnosti in vrednosti.

selektor {lastnost: vrednost}

Selektor je lahko katerikoli element (značka) uporabljen v kodi HTML. Tu je primer spremembe barve pisave na rdeče za celoten dokument.

body {color: red}

Vrednosti se **ne pišejo** z narekovaji. V narekovajih so nizi. Le če je vrednost sestavljena iz dveh ali**več besed,** potem jo je potrebno pisati med narekovaji:

p {font-family: "Times New Roman"}

Če želimo selektorju prirediti **več** **lastnosti**, potem jih ločimo s **podpičjem**. Priporoča se tudi pisanje podpičaja na koncu, saj se pogosto zgodi, da dodamo še kakšno pravilo, pri tem pa se pozabi na vejico.

p {text-align: center; color: red; font-size: 12pt;}

Če se selektorju priredi več **alternativnih** lastnosti, potem se jih loči z vejico. Brskalnik bo v seznamu izbral prvo, ki jo prepozna.

p {cursor: text, pointer, wait}   
blockquote {font-family : Verdana, Geneva, Arial, Helvetica, sans-serif}

Zaradi boljše preglednosti, je priporočljivo pisanja eno lastnosti pod drugo.

p { text-align: center; color: red; font-size: 12pt}

p {   
text-align: center;   
color: red;   
font-size: 12pt  
}

# KOMENTARJI V CSS

Če želimo, lahko vstavimo komentar v kodo prekrivnega sloga, tako da začetek komentarja označimo z /\* , napišemo komentar, konec pa naznanimo z \*/:

/\* To je komentar \*/

p { text-align: center;

/\* Tukaj je drugi komentar \*/

color: black; font-family: Arial; }

# VKLJUČITEV V HTML

Opise slogov lahko vključimo v HTML na tri različne načine:

1. Opise pripravimo na posebni datoteki (običajno ji damo končnico \*.css), ter jo z elementom **link** vključimo v **glavo** dokumenta HTML. Na tak način lahko vključimo tudi več datotek z opisi slogov. Na ta način zelo enostavno spremenimo videz vseh dokumentov, ki so povezani s to datoteko

<head> ...

<link rel="stylesheet" type="text/css" href="slog.css" />

... </head>

 Celoten opis slogov vključimo v glavo dokumenta HTML. Vse, kar bi v prejšnjem načinu napisali na posebno datoteko, tu napišemo kot vsebino elementa **style** v glavi dokumenta.

<head> ...

<style type="text/css">  
**pravila CSS**  
</style>

... </head>

 Sloge lahko opišemo tudi sproti (med besedilom). Večini elementom namreč lahko predpišemo **lastnost style**, ki določa obliko elementa.

<p style="..."> ... </p>

Seveda lahko te tri možnosti opisov slogov tudi kombiniramo. Nekaj slogov opišemo ločeno na datotekah CSS, nekaj opisov vključimo kar v glavo dokumenta, preostanek pa definiramo sproti.

Glavni namen slogov je ločiti vsebino in obliko.

**Edini strogo zružljiv način je uporaba zunanjih prekrivnih slogov.**

# PREKRIVANJE SLOGOV

# PREKRIVANJE

Posredovanje informacij o slogu brskalniku je možno na več načinov. Slog elementa je lahko vsebovan neposredno v elementu HTML, lahko ga uporabimo v glavi dokumenta HTML (element**style**) ali pa v zunanji datoteki. Seveda lahko te tri možnosti opisov slogov tudi kombiniramo. Nekaj jih opišemo v zunanjih datotekami, nekaj opisov vključimo kar v glavo dokumenta, preostanek pa definiramo sproti. Glavni namen slogov je ločiti oblikovanje od vsebine, zato naj se najbolj uporabljajo zunanje datoteke.

## Kateri slog ima prednost

Kateri slog bo uporabljen, če je v dokumentu HTML le-teh več? W3C določa kar komplesen način za določanje prioritete. Poenostavljeno in za večino uporabnikov popolnoma ustrezno lahko rečemo, da brskalnik uporabi neki slog po takem vrstnem redu:

1. slog kot atribut elementa (style="")
2. slogovna predloga v glavi (<style>)
3. zunanja slogovna predloga (<link rel="stylesheet">)
4. privzeta vrednost v brskalniku

Slog znotraj elementa največjo prioriteto, sledi mu slogovna predloga v glavi dokumenta na koncu, če ni nikjer določen slog tega elementa, vzame privzeto vrednost, ki jo ima brskalnik.

V primeru, da je neko pravilo napisano dvakrat z isto prioriteto, velja, da prevlada **zadnje** zapisano.

# 1. PRIMER

V zunanji datoteki (datoteke/zunanji.css) je napisano:

p {color: red;}

V elementu style je:

p {color: blue;}

Kakšna bo barva odstavka? Preveri na primeru desno.

## 2. Primer

Dopolni primer še tako, da boš določil zeleno barvo prvemu odstavku (ima atribut style).

# VEČKRAT DOLOČENO

V primeru, da pravilo določimo večkrat, se uporabi zadnje napisano.

## Primer

p, h1{color: red;}

p, blockquote {color: maroon;}

Če je **p** določen tudi z drugom pravilom, se prekrije barva, ki je določena najprej.

## Aktivnost

Kakšna bo barva odstavka za gornji primer CSS in podani HTML.

Za podani primer napiši pravilo CSS in se prepričaj.

# PREKRIVANJE IN HTML

Prekrivanje slogov je tudi odvisno od zgradbe dokumenta HTML. Pri pravilih CSS je potrebno vedeti, katera se dedujejo in katera ne.

V splošnem velja, da imajo otroci enake lastnosti kot starši. V primeru, da otrokom pripišemo lastnosti, ki so drugačne od starševih, jih le te prekrijejo.

## Primer

div{color: red;}

p {color: maroon}

Če je **p** otrok **div**, se prekrije barva, ki je določena potomcu.

## Aktivnost

Kakšna bo barva odstavka za gornji primer CSS in podani HTML.

Za podani primer napiši pravilo CSS in se prepričaj.

# GLAVA DOKUMENTA HTML

# ZAKAJ POTREUJEMO GLAVO

V HTML 5 sicer ni več obvezno deliti spletne strani na glavo in telo, vendar še vedno priporočam, da to storite.

V glavi med značkama **<head>** in **</head>**opišemo dokument. Z elementom **<title>** moramo določiti naslov, ki se prikaže v naslovni vrstici brskalnika.

Z drugimi elementi pa lahko določimo še način kodiranja, ki določa, na kakšen način smo zapisali znake, ki jih ni v tabeli ASCII, ključne besede, ki raznim iskalnikom pomagajo pri uvrščanju spletne strani v ustrezne skupine, obliko posameznih elementov, vključimo lahko pomožne datoteke ... Kar napišemo v glavi, v oknu brskalnika ni vidno. Do teh podatkov pridemo le, če to posebej zahtevamo (na primer z ukazom Tools, Page Infov brskalniku Mozilla Firefox).

DELI GLAVE

<head>

Glava HTML dokumenta

<title>

Določa naslov dokumenta.

<meta>

Določa meta podatke dokumenta.

<base>

Definira začetni URL sklicevanih dokumentov.

<script>

Del dokumenta, ki vsebuje skript (JavaScript, VBScript) ali povezavo do zunanjega skripta.

<noscript>

Del v katerem v katerem ni skriptov.

<link>

Določimo, kje se nahaja zunanja datoteka s prekrivnim slogom CSS za tekoči dokument.

<style>

Definira prekrivni slog CSS tekočega dokumenta HTML.

ELEMENT META

description

Meta podatek description vsebuje splošen**opis** spletne strani.  
<meta name="description" content="Carska spletna stran">

keywords

Meta podatek keyword vsebuje **ključne besede** določene spletne strani, za lažje pravilno razvrščanje po iskalnikih.  
<meta name="keywords" content="SERŠ, RSO, Seminarska, HTML">

author

Meta podatek author vsebuje **avtorja**oziroma avtorje spletne strani.  
<meta name="author" content="Klemen Šauperl">

refresh

Opisuje osvežitev spletne strani  
<meta http-equiv="refresh" content="10">

# SEZNAMI

# NEUREJEN SEZNAM

Neurejen seznam ustvarimo z elementom **ul** .

Za začetek napišemo znalko **<ul>**, zaključimo pa z značko **</ul>**.

Znotraj značk <ul> in </ul> pa napišemo posamezne točke seznama z obveznimi elementi**li**, vsak seznam mora imeti vsaj en element li. Med začetmo značko <li> in končno </li> se nahaja besedilo, lahko pa tudi večima medvrstičnih elementov.

# OBLIKOVANJE OZNAK SEZNAMA

Seznam lahko oblikujemo tako, da spremenimo slog besedila.

To storimo v CSS in lahko spremenimo vrstično oznako v krog, kvadrat ali v nek drug znak. V CSS za določitev vrste seznama napišemo **list-style-type**.

Za neurejene sezname sedanji standard dovoljuje naslednje možnosti:

* circle (krožnica),
* disc (zapolnjen krog),
* square (kocka)

Če ne želimo oznake, pa uporabimo:

* none

# UREJEN SEZNAM

Urejen seznam ima v vsaki vrstici neko vrednost, ki narašča. Ta vrednost je lahko številska ali črkovna.

Za urejen seznam pa namesto značk <ul> in </ul> uporabimo znački **<ol>** in **</ol>**.

Znotraj še vedno pišemo enako, posamezne alineje označimo z <li> in z</li>.

# DODATNE MOŽNOSTI

Urejen seznam ima lahko v HTML 5 dva atributa.

## reversed

Logični atribut reversed določa ali bodo števila naraščala ali padla. Analogno velja za urejevanje po abecedi.

reversed="reversed"

## start

Atribut start določa vrednost, ki začenja seznam.

start="5"

# OBLIKOVANJE OZNAK SEZNAMA

Seznam lahko oblikujemo tako, da spremenimo slog besedila.

To storimo v CSS in lahko spremenimo predznak iz pomišljaja v krog, karo ali kateri drugi znak. V CSS tako napišemo kateri tip seznama hočemo.

Za urejene sezname je:

* decimal (desetiška števila)
* lower-roman (male rimske številke)
* upper-roman (velike rimske številke)
* lower-alpha (male črke)
* upper-alpha (velike črke)

Če ne želimo oznake, pa uporabimo:

* none

# LEGA OZNAKE

Zamik besedila glede na položaj oznake določimo z list-style-position:

inside

Z vrednostjo inside postavimo besedilo vzporedno z znakom seznama.

outside

Z vrednostjo outside pa pomaknemo besedilo za znak

# Skriptni jeziki

Kaj so skriptni jeziki

V pisanju programov lahko v zadnjih petnajstih letih opazimo bistvene spremembe - prihaja do prehoda od programiranja v sistemskih programskih jezikih k programiranju v skriptnih programskih jezikih.

Skriptni jeziki so namenjeni drugačnim nalogam, kot so tiste, za katere uporabljamo sistemske programske jezike. Z njimi želimo predvsem povezati že obstoječe komponente, ki pa so ponavadi narejene v enem izmed sistemskih programskih jezikov.

Obe vrsti programskih jezikov obstajata vzporedno in sta komplementarni (zasledimo ju na skoraj vseh računalniških platformah že v šestdesetih letih), vendar lahko spremljamo večanje pomembnosti vloge skriptnih jezikov, ki se dogaja v zadnjem času, predvsem po razmahu uporabe grafičnih uporabniških vmesnikov, komponentnih arhitektur in interneta. Ne gre pa niti zanemariti dejstva, da se je tudi kakovost samih skriptnih programskih jezikov zelo povečala. Programiranje v skriptnih jezikih se od "navadnega" programiranja razlikuje v pomembnem dejstvu, da skriptni jeziki predvidevajo, da že obstaja množica uporabnih komponent, ki jih nato lahko uporabimo v lastnih programih. Na tak način lahko razširimo uporabnost komponent in jih uporabimo tudi za namene, za katere prvotno niso bile predvidene. Kot primer lahko navedem skripte napisane v raznih UNIX lupinah.

Druga pomembna razlika je, da skriptni jeziki ne uporabljajo tipov, kot jih uporabljajo sistemski programski jeziki - eni spremenljivki lahko priredimo vrednosti različnih tipov. Spremenljivk ponavadi ni potrebno niti deklarirati, saj je večina skriptnih jezikov interpretiranih, tako da se prostor za spremenljivke rezervira med samim izvajanjem programa. Takšna uporaba spremenljivk povečuje ponovno uporabnost in zelo olajšuje povezovanje komponent, saj jih lahko uporabimo na načine, ki prvotno niso bili predvideni. Kljub temu, da skriptni jeziki niso strogo tipizirani, napake vseeno ne ostanejo neopažene, saj so tipi preverjeni tik preden je neka vrednost uporabljena.

Takšno preverjanje je uporabljeno, ker je večina skriptnih jezikov interpretiranih. Tako so cikli pisanja programa in preverjanja delovanja lahko krajši kot pri prevajanih jezikih. Pri takem načinu programiranja pa žal zelo trpi hitrost izvajanja programov. K sreči je uporaba skriptnih jezikov naravnana tako, da gre večina časa izvajanja za izvajanje komponent, ki jih skripti povezujejo.

## Uporaba skriptnih jezikov

Skriptni jezik je programski jezik, Ki je uporabljen za rokovanje, prilagajanje in samodejno opravljanje določenih opravil v obstoječem sistemu. V mnogih sistemih je običajno že na voljo uporabniški vmesnik in skriptni jezik je mehanizem za vodenje uporabnosti programa. V tem sistemu obstaja množica predmetov, ko dopolnjujejo sposobnost skriptnega jezika. Skriptne jezike uporabljajo profesionalci in občani programerji.

Z uporabo skriptnih jezikov si lahko zelo olajšamo in pospešimo delo pri razvoju nekaterih aplikacij. Zasluga za to gre tudi nenehnim izboljšavam, ki so jih bili deležni skriptni jeziki. Da pa je prišlo tudi do velikih izboljšav pri hitrosti izvajanja programov, ki so napisani v skriptnih jezikih, je tudi zasluga vse hitrejše strojne opreme. Skriptni jeziki se danes uporabljajo predvsem pri:

* **Grafičnih uporabniških vmesnikih**: Grafični uporabniški vmesniki so se razširili ob začetku devetdesetih let. Cilj takšnega vmesnika ni zagotavljanje novih funkcionalnosti, ampak čim učinkovitejše povezovanje elementov grafičnega uporabniškega vmesnika z internimi funkcijami raznih aplikacij.
* **Internetu**: Internet je pravzaprav le veliko orodje za povezovanje aplikacij - z njim neposredno ne nastajajo informacije ampak nam omogoča enostaven dostop do množice raznih podatkov in aplikacij. Idealno orodje, ki omogoča čim lažjo povezavo vseh obstoječih komponent, je skriptni jezik.
* **Uporabi komponent**: Komponente, ki jih kreiramo s pomočjo sistemskih programskih jezikov, najlažje povežemo s pomočjo skriptnih orodij.
* **Nedeljskih programerjih**: Z množično uporabo računalnikov je tudi vse več ljudi želelo včasih kaj spremeniti pri aplikaciji, ki so jo uporabljali. Tudi vse več aplikacij je podpiralo vsaj neko obliko skriptnega programiranja. Tako si danes lahko skoraj vsakdo napiše kakšen makro v najljubši preglednici ali urejevalniku besedil.

## Različice

Trenutna različica CSS je 2.1. Brskalniki, ki podpirajo to različico, morajo pravilno prebrati tudi pravila CSS 1.0. Brskalniki, ki podpirajo CSS 1.0, morajo prebrati dokumente v CSS 2.1, pri tem pa preskočijo vsa pravila, ki jih ne prepoznajo.

Veljavna slogovna predloga temelji na plasti CSS, ki je uporabljena. Vsak veljaven dokument CSS1, je tudi veljaven dokument CSS 2.1, čeprav se nekatera pravila CSS1 malenkostno razlikujejo od tistih v CSS 2.1. Dokument CSS2 pa ni nujno veljaven dokument CSS 2.1 in obratno, saj so nekatera pravila CSS 2.1 dodana in nekatera odvzeta.

Mnogo pravil razvojne različice **CSS3** podpira že precej brskalnikov. Nekatera so še v razvojni fazi, zato jih v datoteki CSS pišemo s predponami -moz-, -webkit-, -ms in drugimi, odvisno od razvojnega orodja.

# Jezik CSS

Z jezikom **CSS** (angl. Cascading Style Sheets) definiramo stil elementov HTML v smislu pravil, kako naj se ti prikažejo na strani. Določamo lahko torej barve, velikosti, odmike, pozicije, obrobe, poravnave, pisavo in celo vrsto drugih atributov, prav tako pa lahko upravljamo določene aktivnosti, ki jih izvaja uporabnik na strani, recimo prekritje povezave z miško. CSS so torej predloge, zapisane v obliki preprostega slogovnega jezika, ki se uporablja za prezantacijo spletnih strani in ostalih dokumentov (XML, SVG, XUL)

Glavni namen uporabe CSS je poleg definiranja pravil, ločitev strukture strani, ki je zapisana v HTML dokumentu, od njene predstavitve. S tem omogočimo večjo preglednost dokumentov HTML in zmanjšamo ponavljanje kode, kar lahko bistveno zmanjša njihovo velikost in nalaganje. Specifikacija in vzdrževanje je podobno kot pri jeziku HTML, v rokah organizacije W3C. Prva uradna specifikacija s strani W3C je izšla leta 1996. Ta verzija je vsebovala ogromno pomanjkljivosti, zato so leta 1998 izdali priporočilo za drugo verzijo, t. i. CSS2. Ta specifikacija je veljala za najnovejše priporočilo vse do junija 2011, ko je izšlo priporočilo CSS2.1. Septembra 2011 je izšlo priporočilo CSS3, ki trenutno velja za najnovejše priporočilo. Vse trenutne in tudi prihajajoče verzije bodo uporabljale isti temeljni koncept sintakse, ki je precej preprost.

Primer deklaracijskega bloka, ki določa lastnost posameznemu elementu:

selektor1 [, selektor2, selektor3 ...][:psevdo-razred ali

:psevdo-element]{

lastnost1: vrednost1;

[lastnost2: vrednost2;

lastnost3: vrednost3;

...]

}

* **Selektor**: predstavlja element ali več elementov, na katerega se nanašajo stilska pravila. Selektor je lahko element HTML določenega tipa (npr. div, p, img ) ali pa glede na njihov atribut (id ali class).
* **Psevdo-razred**: predstavlja način identifikacije elementov na podlagi aktivnosti uporabnika, psevdo-element pa nabor okoliščin, v katerem se element nahaja. Tipični primer psevdo-razreda je prekritje povezave z miško (:hover), tipični primer psevdo-elementa pa je prvi element znotraj več vrstičnega zapisa (:first-line).
* **Telo deklaracijskega bloka**: se nahaja med zavitima oklepajema. Sestavljeno je iz lastnosti, ki ji definiramo neko vrednost (npr. color: blue;). Sledi podpičje in lahko dodamo novo lastnost.

CSS lahko v HTML vključimo na različne načine:

* **Vrstični**, ki je definiran znotraj HTML dokumenta in ima najvišjo prioriteto. Primer:

<h1 style='color: red;'>Pozdravljen svet</h1>

* **Vgrajen**, ki je definiran znotraj HTML dokumenta pod atributom style. Primer:

<style type="text/css">

h1 {

color: blue;

}

</style>

* **Zunanji**, ki je vključen v HTML dokument tok zunanji dokument. Primer:
* **o vsebina dokumenta stil.css**

h1 {

color: green;

}

* **o vsebina dokumenta test.html**

<head>

<link rel="stylesheet" href=" stil.css" type="text/css"/>

</head>

Pri uporabi določenih stilov se velikokrat pojavijo tudi problemi. Določene lastnosti niso podprte ali pa delujejo drugače. To predstavlja problem predvsem v starejših brskalnikih, ki so še vedno v uporabi (IE 6 in IE7). Oblikovalcem spletnih strani in razvijalcem to dejstvo predstavlja zelo resen problem in tudi dodatno delo z usklajevanjem prikaza za vse brskalnike. Lahko se torej odločimo, da bomo določene stile, ki predstavljajo problem, določili za določen tip brskalnika, lahko pa se poskušamo tem stilom izogniti s t. i. CSS triki. Vsekakor priporočamo spletnim oblikovalcem oz. razvijalcem, da že v času samega »razreza« grafične predloge (transformacije iz grafične predloge v HTML in CSS kodo) testirajo prikaz spletne strani oz. aplikacije v vseh brskalnikih, ki so trenutno v uporabi (Chrome, Firefox, IE 6+, Opera, Safari). V zadnjem času je uporaba IE 6 in IE7 že zelo upadla in znaša skupaj za oba brskalnika 0,8 % (podatek iz julija 2013). To je precej nizek odstotek; lahko se nehamo ozirati na ta dva brskalnika, gleda na to, da povzročata največ problemov.

# Kaj je CSS

* CSS je prekrivni slog.
* CSS je kratica za Cascading Style Sheets.
* Slog lahko določamo za vsak element v HTML.
* Slogi so običajno shranjeni v slogovnih predlogah (datoteka \*.css).
* Slogi so bili dodani v različico HTML 4.0 za boljšo preglednost spletnih dokumentov.
* Zunanje prekrivni slogi vam lahko prihranijo veliko dela.

## Prekrivni Slogi lahko prihranijo veliko časa

Slogi v HTML določajo, kako so elementi v HTML oblikovani. Prekrivni slogi so običajno shranjeni v posebnem dokumentu, zaradi lažje preglednosti. Eno zunanjo slogovno predlogo lahko uporabimo potem na vseh drugih straneh, če jih je več. Vsi, ki smo že kdaj skušali spreminjati recimo pisavo v vseh elementih HTML, vemo koliko truda lahko prihrani slogovna predloga.

Prekrivni slogi CSS nam omogočajo, da nadzorujemo slog strani samo preko enega dokumenta, in to je ravno največja prednost. Ker so elementi običajno globalno povezani med sabo, s spremembo le enega pravila zamenjamo na primer ozadje vseh spletnih strani spletišča.

# Sintaksa

prekrivni slogi so zgrajena iz niza pravil. Pravilo je zgrajeno iz dveh delov: iz selektorja in iz deklaracije. Deklaracija je sestavljena iz lastnosti in vrednosti.

**selektor {lastnost: vrednost}**

Selektor je lahko katerikoli element (značka) uporabljen v kodi HTML. Tu je primer spremembe barve pisave na rdeče za celoten dokument.

**body {color: red}**

Vrednosti se **ne pišejo** z narekovaji. V narekovajih so nizi. Le če je vrednost sestavljena iz dveh ali **več besed,** potem jo je potrebno pisati med narekovaji:

**p {font-family: "Times New Roman"}**

Če želimo selektorju prirediti **več** **lastnosti**, potem jih ločimo s **podpičjem**. Priporoča se tudi pisanje podpičaja na koncu, saj se pogosto zgodi, da dodamo še kakšno pravilo, pri tem pa se pozabi na vejico.

**p {text-align: center; color: red; font-size: 12pt;}**

Če se selektorju priredi več **alternativnih** lastnosti, potem se jih loči z vejico. Brskalnik bo v seznamu izbral prvo, ki jo prepozna.

**p {cursor: text, pointer, wait}**blockquote {font-family : Verdana, Geneva, Arial, Helvetica, sans-serif}

Zaradi boljše preglednosti, je priporočljivo pisanja eno lastnosti pod drugo.

**p { text-align: center; color: red; font-size: 12pt}**

p {   
text-align: center;   
color: red;   
font-size: 12pt  
}

# Komentarji v CSS

Če želimo, lahko vstavimo komentar v kodo prekrivnega sloga, tako da začetek komentarja označimo z /\* , napišemo komentar, konec pa naznanimo z \*/:

/\* To je komentar \*/

p { text-align: center;

/\* Tukaj je drugi komentar \*/

color: black; font-family: Arial; }

**Vključitev v HTML**

Opise slogov lahko vključimo v HTML na tri različne načine:

1. Opise pripravimo na posebni datoteki (običajno ji damo končnico \*.css), ter jo z elementom **link** vključimo v **glavo** dokumenta HTML. Na tak način lahko vključimo tudi več datotek z opisi slogov. Na ta način zelo enostavno spremenimo videz vseh dokumentov, ki so povezani s to datoteko

<head> ...

<link rel="stylesheet" type="text/css" href="slog.css" />

... </head>

 Celoten opis slogov vključimo v glavo dokumenta HTML. Vse, kar bi v prejšnjem načinu napisali na posebno datoteko, tu napišemo kot vsebino elementa **style** v glavi dokumenta.

<head> ...

<style type="text/css">  
**pravila CSS**  
</style>

... </head>

Sloge lahko opišemo tudi sproti (med besedilom). Večini elementom namreč lahko predpišemo **lastnost style**, ki določa obliko elementa.

<p style="..."> ... </p>

Seveda lahko te tri možnosti opisov slogov tudi kombiniramo. Nekaj slogov opišemo ločeno na datotekah CSS, nekaj opisov vključimo kar v glavo dokumenta, preostanek pa definiramo sproti.

Glavni namen slogov je ločiti vsebino in obliko.

**Edini strogo združljiv način je uporaba zunanjih prekrivnih slogov.**

# Prekrivanje

Posredovanje informacij o slogu brskalniku je možno na več načinov. Slog elementa je lahko vsebovan neposredno v elementu HTML, lahko ga uporabimo v glavi dokumenta HTML (element **style**) ali pa v zunanji datoteki. Seveda lahko te tri možnosti opisov slogov tudi kombiniramo. Nekaj jih opišemo v zunanjih datotekami, nekaj opisov vključimo kar v glavo dokumenta, preostanek pa definiramo sproti. Glavni namen slogov je ločiti oblikovanje od vsebine, zato naj se najbolj uporabljajo zunanje datoteke.

## Kateri slog ima prednost

Kateri slog bo uporabljen, če je v dokumentu HTML le-teh več? W3C določa kar komplesen način za določanje prioritete. Poenostavljeno in za večino uporabnikov popolnoma ustrezno lahko rečemo, da brskalnik uporabi neki slog po takem vrstnem redu:

1. slog kot atribut elementa (style="")
2. slogovna predloga v glavi (<style>)
3. zunanja slogovna predloga (<link rel="stylesheet">)
4. privzeta vrednost v brskalniku

Slog znotraj elementa največjo prioriteto, sledi mu slogovna predloga v glavi dokumenta na koncu, če ni nikjer določen slog tega elementa, vzame privzeto vrednost, ki jo ima brskalnik.

V primeru, da je neko pravilo napisano dvakrat z isto prioriteto, velja, da prevlada **zadnje** zapisano.

# Večkrat določeno

V primeru, da pravilo določimo večkrat, se uporabi zadnje napisano.

## Primer

p, h1{color: red;}

p, blockquote {color: maroon;}

Če je **p** določen tudi z drugom pravilom, se prekrije barva, ki je določena najprej.

## Aktivnost

Kakšna bo barva odstavka za gornji primer CSS in podani HTML.

# Prekrivanje in HTML

Prekrivanje slogov je tudi odvisno od zgradbe dokumenta HTML. Pri pravilih CSS je potrebno vedeti, katera se dedujejo in katera ne.

V splošnem velja, da imajo otroci enake lastnosti kot starši. V primeru, da otrokom pripišemo lastnosti, ki so drugačne od starševih, jih le te prekrijejo.

## Primer

div{color: red;}

p {color: maroon}

Če je **p** otrok **div**, se prekrije barva, ki je določena potomcu.

# Pravilo !important

Pravilo !important uporabimo, kadar želimo, da neka deklaracija velja vedno. Če npr. v zunanji slogovni predlogi ob deklaraciji uporabimo pravilo !important, potem ta deklaracija velja vedno, tudi če jo v kakšni drugi slogovni predlogi spremenimo. Če uporabimo več pravil !important so si vsa pravila enakovredna.

## Primer

p{color: red !important;}

p {color: maroon}

Kakšna bo sedaj barva odstavka.

# Zakaj potrebujemo glavo

V HTML 5 sicer ni več obvezno deliti spletne strani na glavo in telo, vendar še vedno priporočam, da to storite.

V glavi med značkama **<head>** in **</head>** opišemo dokument. Z elementom **<title>** moramo določiti naslov, ki se prikaže v naslovni vrstici brskalnika.

Z drugimi elementi pa lahko določimo še način kodiranja, ki določa, na kakšen način smo zapisali znake, ki jih ni v tabeli ASCII, ključne besede, ki raznim iskalnikom pomagajo pri uvrščanju spletne strani v ustrezne skupine, obliko posameznih elementov, vključimo lahko pomožne datoteke ... Kar napišemo v glavi, v oknu brskalnika ni vidno. Do teh podatkov pridemo le, če to posebej zahtevamo (na primer z ukazom Tools, Page Info v brskalniku Mozilla Firefox).

# Deli Glave

**<head>** Glava HTML dokumenta

**<title>** Določa naslov dokumenta.

**<meta>** Določa meta podatke dokumenta.

**<base>** Definira začetni URL sklicevanih dokumentov.

**<script>** Del dokumenta, ki vsebuje skript (JavaScript, VBScript) ali povezavo do zunanjega skripta.

**<noscript>** Del v katerem v katerem ni skriptov.

**<link>** Določimo, kje se nahaja zunanja datoteka s prekrivnim slogom CSS za tekoči dokument.

**<style>** Definira prekrivni slog CSS tekočega dokumenta HTML.

# Element Meta

**description**

Meta podatek description vsebuje splošen **opis** spletne strani.  
<meta name="description" content="Carska spletna stran">

**keywords**

Meta podatek keyword vsebuje **ključne besede** določene spletne strani, za lažje pravilno razvrščanje po iskalnikih.  
<meta name="keywords" content="SERŠ, RSO, Seminarska, HTML">

**author**

Meta podatek author vsebuje **avtorja** oziroma avtorje spletne strani.  
<meta name="author" content="Klemen Šauperl">

**refresh**

Opisuje osvežitev spletne strani  
<meta http-equiv="refresh" content="10">

# Neurejen seznam

Neurejen seznam ustvarimo z elementom **ul** .

Za začetek napišemo znalko **<ul>**, zaključimo pa z značko **</ul>**.

Znotraj značk <ul> in </ul> pa napišemo posamezne točke seznama z obveznimi elementi **li**, vsak seznam mora imeti vsaj en element li. Med začetmo značko <li> in končno </li> se nahaja besedilo, lahko pa tudi večima medvrstičnih elementov.

**Oblikovanje oznak seznama**

Seznam lahko oblikujemo tako, da spremenimo slog besedila.

To storimo v CSS in lahko spremenimo vrstično oznako v krog, kvadrat ali v nek drug znak. V CSS za določitev vrste seznama napišemo **list-style-type**.

Za neurejene sezname sedanji standard dovoljuje naslednje možnosti:

* circle (krožnica),
* disc (zapolnjen krog),
* square (kocka)

Če ne želimo oznake, pa uporabimo:

* none

# Urejen seznam

Urejen seznam ima v vsaki vrstici neko vrednost, ki narašča. Ta vrednost je lahko številska ali črkovna.

Za urejen seznam pa namesto značk <ul> in </ul> uporabimo znački **<ol>** in **</ol>**.

Znotraj še vedno pišemo enako, posamezne alineje označimo z <li> in z</li>.

# Dodatne možnosti

Urejen seznam ima lahko v HTML 5 dva atributa.

## reversed

Logični atribut reversed določa ali bodo števila naraščala ali padla. Analogno velja za urejevanje po abecedi.

reversed="reversed"

## start

Atribut start določa vrednost, ki začenja seznam.

start="5"

**Oblikovanje oznak seznama**

Seznam lahko oblikujemo tako, da spremenimo slog besedila.

To storimo v CSS in lahko spremenimo predznak iz pomišljaja v krog, karo ali kateri drugi znak. V CSS tako napišemo kateri tip seznama hočemo.

Za urejene sezname je:

* decimal (desetiška števila)
* lower-roman (male rimske številke)
* upper-roman (velike rimske številke)
* lower-alpha (male črke)
* upper-alpha (velike črke)

Če ne želimo oznake, pa uporabimo:

* none

# Lega oznake

Zamik besedila glede na položaj oznake določimo z list-style-position:

**inside** Z vrednostjo inside postavimo besedilo vzporedno z znakom seznama.

**outside** Z vrednostjo outside pa pomaknemo besedilo za znak,

# Dodajanje slik

Slike ali fotografijo v HTML vključimo z značko <img />. Značka mora **obvezno** vsebovati tudi atribut **src**, ki je okrajšava za vir (ang. source). Brskalniku torej pove, kjer se slika nahaja.

Značka je odprta in je ne smemo napisati s začetno in končno značko (~~<img>...</img>~~). Slike tudi ne more biti ugnezdene.

Slika mora biti shranjena na datoteki v obliki GIF, JPEG ali PNG.

Slika ne more biti neposredno v telesu dokumenta, v katerih elementih je lahko, si oglejte v referenci HTML.

# Atribut alt

Atribut alt pove bralcu oz. uporabniku, ki si ogleduje kakšno stran, kaj v bistvu manjka tej spletni strani, če brskalnik ne more naložiti slike, ki je povezana s to stranjo.

Brskalnik v tem primeru nadomesti sliko s tako imenovanim zamenljivim besedilom. Vsekakor je zelo dobro uporabiti alt z opisom za vsako sliko, ki se nahaja na kakšni spletni strani, da izboljšamo prikaz in ohranimo koristnost dokumenta, predvsem za ljudi, ki imajo le tekstovne brskalnike (npr. slepi imajo naprave, ki preberejo vsebino alt).

Če je slika nadomestilo za besedilo, kot je ime podjetja ali logotip, gumbi za navigacijo, potem ustrezne informacije vpišite v alt. Če je slika nekaj, česar ni mogoče nadomestiti z besedilom, označite vsebino slike s kratko oznako, npr. "fotografija oddelka".

V novejših brskalnikih se pojavi kot kratek opis slike, če z miško nekaj sekund počivamo na njej in če napis ni določen z atributom title. Po standardu je navedba te lastnosti **obvezna**.

# Velikost slike

Sliki lahko spremenimo širino (width) in/ali višino (height).

Velikost slike je v CSS-pikslih (brez navedbe enote).

Če podani širina in višina ne ustrezata dejanski velikosti slike, bo pregledovalnik sliko ustrezno raztegnil ali skrčil. Tega početja pa ne svetujem predvsem pri bitnih slikah. Če je originalna slika večja, se bo brskalnik "namučil" za prikaz pomanjšane slike, kvaliteta pa ne bo boljša.

Uporaba atributov width in height je priporočljiva z navajanjem dejanske velikosti slike. Če širine in višine ne podamo, pregledovalnik ne more rezervirati prostora za sliko, dokler slike ne dobi s strežnika

# Optimizacija slike

Za **optimizacijo slike** sta pomembna atributa **title=" "** in pa obvezni **alt=" "**. Z atributom **title** določimo **ime slike**, atribut **alt** pa vstavimo **ključne besede**, ki opisujejo sliko.

Z lastnostjo **alt** na kratko opišemo, kaj se nahaja na sliki. Opis slike se prikaže, če pregledovalnik slike ne more prikazati (prikaže se namesto slike). To je pomembno za aplikacije, ki so namenjene slepim in slabovidnim

# Odzivna velikost

Odzivni sistemi prilagajajo velikost elementov spletne strani velikosti vidne površine (ang. Viewport)

Sliko ustvarimo tako, da jo vključimo v nek vsebovalnik (npr. v element div ali figure).

Sliki določimo največjo vrednost v tem elementu:

img{

width:100%;

height: auto;

}

# Slika kot povezava

Sliko lahko uporabimo tudi kot povezavo. Ob kliku na sliko bomo preusmerjeni na stran, ki je v povezavi navedena.

Povezavo ustvarimo tako, da ustvarimo sliko znotraj elementa **a**.

Pogosto je v tem primeru moteča obroba okoli slike.

Če se želimo obrobe znebiti, to storimo s slogovnimi predlogami:

**border:none;**

# Uporaba barv

Za prikaz barv na ekranu se uporablja barvni zapis RGB. Človeško oko lahko loči med okoli 10.000.000 odtenki. Zato za dokaj realen zapis barv zadostuje 24-bitna barvna globina, z njo dobimo **16.777.216 različnih barv** in njihovih odtenkov. To veliko številko smo dobili z množenjem 256×256×256 (Zakaj ne 255×255×255? Ker moramo upoštevati še 0).

Barve lahko spletnim stranem stranem dodamo samo z uporabo slogovnih predlog. To lahko storimo na več načinov:

**barvna trojica**

Barva je določena s posameznimi koponentami **RGB**. Obstaja več načinov za prikaz barv.

**barvna trojica**

Barva je določena s posameznimi koponentami **RGB** in prosojnostjo.

**ime barve**

Barva je določena z imenom.

**transparent**

Element nima barve.

## Faktor Gama

Barva je na monitorju pravilno prikazana, če je le-ta nastavljen na naravne barve (faktor gama ima vrednost 2,2). To pravilo velja samo za CSS, posamezni formati slik ima svoje definicije naravnih barv.

## Kaj se oblikuje

Barvo ima lahko:

* pisava
* obroba
* drugi vidni elementi

# Poimenovane barve

V CSS2.1 je 17 poimenovanih barv, v CSS3 je podana barva transparent:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| aqua |  |  | black |  |
| blue |  |  | fuchsia |  |
| gray |  |  | green |  |
| lime |  |  | maroon |  |
| navy |  |  | olive |  |
| orange |  |  | purple |  |
| red |  |  | silver |  |
| teal |  |  | white |  |
| yellow |  |  | transparent |  |

## Barve v SGML

V SGML obstaja še cela množica poimenovanih barv. Večina brskalnikov zmore prikazati te poimenovane barve, čeprav jih standard CSS ne dovoljuje. Ne uporabljajte jih.

# šestnajstiški zapis

Koda barve je sestavljena **iz treh** števil, za vsako izmed osnovnih barv: rdečo, zeleno, modro (ang. red, green, blue). V kodi RGB navedemo, koliko je katere od teh barv v mešanici.

V CSS zapišemo barvo na naslednji način:

* simbol za število **#**.
* Prvo dvomestno šestnajstiško število predstavlja **rdečo** barvo,
* drugo **zeleno** barvo in
* tretje **modro** barvo.

Najnižja vrednost števila je **00** (0), najvišja pa **FF** (255).

## Primer:

**Modra barva** ne vsebuje odtenkov rdeče in zelene, zato je njena koda #0000FF.

**Rumena barva** vsebuje rdečo in zeleno a ne vsebuje modre, zato je koda za rumeno barvo #FFFF00.

**Črna** ne vsebuje nobenih odtenkov, zato ima kodo **#000000**, **bela** pa vsebuje vse barvne odtenke, zato ima kodo **#FFFFFF**.

# Ena števka za barvo

Starejši zapis kode barve je sestavljen **iz treh** števk, za vsako izmed osnovnih barv: rdečo, zeleno, modro (ang. red, green, blue).

V CSS zapišemo barvo na naslednji način:

* simbol za število **#**.
* po tri **enomestna** šestnajstiška števila za rdečo, zeleno in modro barvo.

Najnižja vrednost števila je **0** (0), najvišja pa **F** (15).

## Primer:

**Rdeča barva** ne vsebuje odtenkov modre in zelene, zato je njena koda #F00.

## Pretvorba

Pretvorbo iz zapis s 3 števkami v zapis s 6 števkami naredimo tako, da vsako števko podvojimo: **#624** je **enakovreden** **#662244**.

Pogosta napaka pri pretvorbi iz enomestnega v dvomestno, je dodajanje ničel.

# Zapis rgb( )

Zapis **rgb()** ima obliko:

**rgb(rdeča, zelena, modra)**.

Vrednosti posamesne barve so med **0** in **255** ali v odstotkih med **0%** in **100%**.

## Primer

**Zelena barva** ne vsebuje rdeče in modre, zato ima zapis rgb(0,255,0) ali rgb(0,100%,0)

|  |  |  |  |
| --- | --- | --- | --- |
| ime | RGB | # |  |
| aqua | rgb(0,255,255) | #00ffff |  |
| blue | rgb(0,255,255) | #0000ff |  |
| gray | rgb(0,255,255) | #808080 |  |
| lime | rgb(0,255,255) | #00ff00 |  |
| navy | rgb(0,0,128) | #000080 |  |
| orange | rgb(255,165,0) | #ffA500 |  |
| red | rgb(255,0,0) | #ff0000 |  |
| teal | rgb(0,128,128) | #008080 |  |
| yellow | rgb(255,255,0) | #ffff00 |  |

## Aktivnost

Za podani seznam določi naslednje barve:

* Naslov - temno zelena,
* besedilo v odstavku - škrlatno,
* elementi seznama - srebrna.

# Zapis rgbA( )

Zapis **rgb()** ima obliko:

**rgb(rdeča, zelena, modra, alfa)**.

Vrednost alfa je med 0.0 (popolna prozornost) in 1.0 (popolna neprosojnost).

Barvne vrednosti RGBA so razširitev barv TGB z dodano vrednostjo alfa, ki določa prozornost predmeta.

## Primer

**Zelena barva** ne vsebuje rdeče in modre, zato ima zapis rgb(0,255,0,0.) ali rgb(0,100%,0)

# Barve HSL

HSL je kratica za hue, saturation, lightness (barva, nasičenost, svetlost) in predstavlja valnji barvni koordinatini sistem

**H** je barva na obroču. 0 je rdeča, 120 je zelena in 240 je modra.

**S** predstavlja odstotek barve. 0% je sivina, 100% je polna barva.

**L** je tudi podana v odstotkih. 0% je črna, 100%je bela.

## Primer

ul {background-color:hsl(120,65%,75%); }

## Barve HSLA

Tudi modelu HSL je lahko dodan faktor alfa.

## Primer

ul {color:hsla(240,45%,75%,0.4); }

# Ustvarjanje naslovov

Naslovi so v HTML ustvarjeni z elementi od <h1> do <h6>.

V HTML5 ločimo logično in fizično hierarhijo naslovov. h1 se ne uporablja več samo za glavni naslov celotne spletne strani, temveč je to glavni naslov **sekcije**.

Pri pravilnem oblikovanju dokumenta HTML, za glavne naslove sekcij uporabljamo h1, nato za podnaslove h2, še za nižji nivo h3. Posameznih nivojev **ne izpuščamo**.

Naslovi se obnašajo podobno kot odstavki. Z ustvarjenim novim naslovom bo izveden avtomatski skok v novo vrstico, prav tako razmik pri naslovu.

Naslovov **ne** uporabljamo za **oblikovanje** velikosti pisave, temu služijo slogovne predloge CSS.

# Velikost pisave

Naslovov **ne** uporabljamo za **oblikovanje** velikosti pisave, temu služijo slogovne predloge CSS.

Naslovi pa se med seboj ločijo po velikosti pisave v HTML. h1 je največji, h6 pa najmanjši.

Za oblikavanje velikosti pisave se običajno uporablja enota **em**. S tem je omogočeno uporabniku spreminjati velikost pisave v brskalniku. Ko računamo velikost pisave, si zamislimo, da ima večina uporabnikov nastavljeno 1em na 16px. Če smo si zamislili velikost h1 35px izračunamo velikost pisave v em:

35px/16px = 2.1875em

Nobene potrebe ni, da bi števila zaokroževali na manjše število decimalnih mest.

# Velike črke

Naslovov **ne** pišite v HTML samo z velikimi črkami. Če želite velike črke, uporabite za to CSS:

**h1{ text-transform: uppercase;}**

V angleških besedilih se pogosto uporabljajo v naslovih velike začetnice:  
**h2 { text-transform: capitalize:}**

**Barva naslovov**

Barvo lahko določimo v CSS s color. Barvo je mogoče določiti na različne načine. Nekateri izmed teh so:

red rdeča

#00ff00 zelena

rgb(0, 0, 255) modra

rgba(0, 0, 255, 0.7) prosojna, modra

hsla(200, 20%, 55%, 0.7) prosojna

# Krepkost naslovov

Debelost pisave določamo s atributom font-width, (določimo mu lahko tudi) normal, bold, bolder, lighter. oz s število od 100 do 900.

## Druge možnosti

Seveda lahko dodamo pisavi tudi druge možnosti, npr. spodnjo obrobo:

**h1, h2 {border-bottom: #236 2px dotted;}**

# Povezave

Element **a** z atributom **href** tvori hiperpovezavo. S klikom na povezavo se uporabnik "preseli" na drugo mesto v spletu. Na ta nač se seli ("surfa") s strani na stran. Vrednost atributa href je URI ciljne povezave.

Povezave so lahko:

* v istem dokumentu
* v zunanjem elementu

Povezave so

* absolutne ali
* relativne.

# Notranje povezave

Povezave pa lahko uporabljamo tudi na spletni strani, ki jo trenutno pregledujemo. Če je stran prenatrpana, lahko povsem preprosto dodamo povezave in tako povečamo preglednost.

Z atributom **id** se enolično določi nek element. V atribut href dodamo kot vrednost pred imenom povezave znak # in na mesto, kjer želimo, da nas poveže zamenjamo href z id.

# Povezave

Del besedila lahko naredimo aktivnega. To pomeni, da se bo uporabniku, če bo kliknil nanj, odprla neka druga datoteka ali pa drug del istega dokumenta. Ta druga datoteka je običajno spletna stran. Lahko pa je tudi slika ali druge različne vrste dokumentov (DOC, XLS, PDF, PS, ZIP ...) ali pa celo program, ki ga lahko poženemo na strežniku. Z atributom type damo uporabniku namig o vsebimi npr:

**type="image/jpeg"**

# AbsolutnE povezavE

Povezavo lahko nastavimo tudi v sliko tako, da jo ustvarimo znotraj slike. Povezave znotraj slike se uporabljajo predvsem na logotipih podjetji oziroma ustanov za vrnitev na prvo stran. HTML uporablja hiperpovezavo za povezavo na druge dokumente v spletu. Za povezavo obstajata v HTML element a (ang. anchor, slov. sidro), ki ima atribut href, ki določa URI cilja. Povezava lahko kaže na katerikoli predmet v spletu: lahko je dokument HTML, slika, zvočni dokument, film.

# Relativni URI

Enostavno povedano, je relativni URI tisti, ki potrebuje nek proces, da se izračuna. To je lokalni URI, ki mu nekatere informacije manjkajo. Običajno so naslovi v istem imeniku ali pa se uporabljajo posebne sekvence, npr. ../. Relativni se imenuje zato, ker se URI izračuna glede na URI trenutnega vira.

## Izračun relativnega URI

Relativni URI potrebuje za pravilen izračun le-ta URI trenutnega vira. Z nekaj operacijami se relativni naslov pretvori v absolutnega, ki je nato uporabljen za iskanje novega vira. Relativni URI je običajno lokalni URI. Prvi vir za izračun je trenutni URI. Nato se uporabijo naslednji postopki:

* Iz naslova trenutnega lokalnega URI, se odstrani ime datoteke, razen če ni to direktorij URI.
* Za vsako zaporedje ../ se na desni začetni (desni) strani odreže en imenik od trenutnega imenika URI
* Doda se relativni lokalni URI in se sestavi v celoto.

# Povezava za e-pošto

Omogočena pa je tudi povezava do elektronske pošte, ki ponuja pošiljanje le te. To dosežemo tako, da za vrednost href napišemo **mailto:** in dodamo želeni elektronski naslov. Na računalniku je **potrebno** imeti program za pošiljanje elektronske pošte

**Oblikovanje povezav**

Psevdorazredi se uporabljajo za določene dogodke. S psevdorazredi lahko naredimo določene stvari lažje in hitreje kot pa z JavaScript.

Psevdorazredi za povezavo:

**link** običajna oblika povezave

**hover** z miško nad povezavo

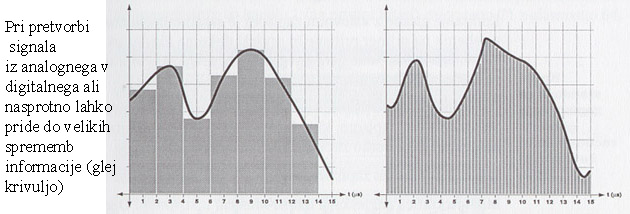
**visited** obiskana povezava

**active** aktivna povezava

V podanem primeru so povezave obarvane modro. Če greste z miško na povezavo, se bo obarvala na svetlo modro in črke so bodo razmaknile za 3px. Ko boste pritisnili na povezavo, se bo povezava obarvala vijoličasto in se bo povečala. Če je ta povezava že bila obiskana, bo obarvana zeleno in bo prečrtana.

**Digitalni zvok**

Človek sliši frekvence med 20 Hz in 20 kHz. V realnosti je zvok analogno valovanje, katerega amplituda je zvezna časovna funkcija. Narava digitalnega signala je drugačna, saj je definiran le v določenih točkah na časovni osi, njegovo amplitudo pa lahko merimo s končnim naborom vrednosti. Zaradi lastnosti digitalne predstavitve je treba analogni signal vzorčiti.

Vzorčenje in kvantizacija (sampling and quantization)

Vzorčimo običajno v enakih časovnih intervalih, katerih pogostost imenujemo vzorčevalna frekvenca.

**Osnovne lastnosti digitalnega zvoka**

Lastnosti so:

**Število kanalov**

Večina aplikacij potrebuje dva kanala (stereo), v profesionalnih aplikacijah pa se jih uporablja tudi več (vse do 64).

**Prepletanje**

Kadar imamo več kanalov, so lahko vzorci za vsak kanal shranjeni ločeno - neprepleteno ali pa prepleteno. V drugem primeru je sinhronizacija posameznih kanalov že zagotovljena, prav tako pa je učinkovitejše stiskanje. Vseeno pa pride do razsipavanja prostora na nosilcu (CD-ROM), kadar niso uporabljeni vsi kanali.

**Kodiranje**

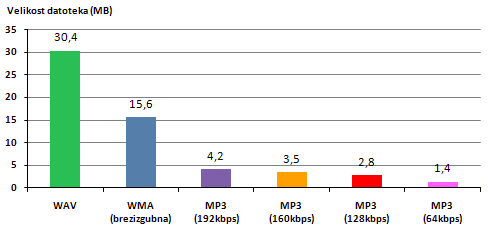
Obstajajo različne metode za predstavitev digitalnega zvoka. Danes se najboljši rezultati dosegajo z uporabo standarda, ki ga je razvila Moving Picture Expert Group (MPEG). Standard pokriva tako digitalni zvok kot tudi video. MPEG uporablja štiri različne načine za predstavitev zvoka: en sam kanal, dualni kanal (dva neodvisna kanala, npr. za dva jezika), stereo (dva prepletena kanala) in sestavljeni stereo (le-ta izkorišča redundanco med levim in desnim kanalom).

# Formati zvočnih datotek

Kadar se srečamo s formati zvočnih datotek, moramo najprej poznati razliko med formatom datoteke in med kodekom (ang. codec). Kodek izvaja kodiranje in dekodiranje čistega zvočnega signala, medtem ko so podatki shranjeni v datoteki s specifičnem formatu. Medtem ko format zvočne datoteke podpira samo en kodek, lahko, na primer, format AVI podpira več kodekov.

Obstajata dve glavne skupine formatov zvočnih datotek:

* **nestisnjeni zvočni forma**t, kot so:
  + Waveform Audio (WAV). Se uporablja za Windows in IBM okolja. Je najbolj pogosto uporabljen format za zajemanje in obdelavo zvoka na osebnih računalnikih.
  + Audio Interchange File Format (AIFF). Podoben format kot WAV, samo da ga uporablja Apple.
  + Free Lossless Audio Codec (FLAC). Je odprtokodni format.
* **format z izgubnim stiskanjem**, kot so:
  + Windows Media Audio (WMA), format, razvit pri Microsoftu,
  + format MPEG (MP3), najbolj priljubljen format, ki omogoča močno stiskanje brez večjih izgub kakovosti zvoka,
  + Advanced Audio Coding (AAC),
  + odprtokodni format Vorbis (OGG),
  + Monkey's Audio (s končnico APE),
  + WavPack (s končnico WV).



Slika 3: Velikost datoteke glede na različni format dokumenta

Slika prikazuje razmerja velikosti datoteke glede na izbran format. Format WAV je nestisnjen in zavzema največ prostora, priljubljeni MP3 pri 64 kb/s zavzame skoraj 30 krat manj prostora, seveda na račun slabše kakovosti zvoka.

# Vstaljanje v HTML

Do sedaj ni bilo standarda za predvajanje zvočnih datotek na spletni strani. Danes se večina zvocnih datotek predvaja preko vtičnikov (najbolj popularen je flash). Vendar pa lahko potrebujejo različni brskalniki različne vtičnike.

HTML5 definira nov element, ki določa enoten način za uporabo zvočne datoteko v spletni strani, to je **audio**.

Trenutno so podprti trije formati glasbenih datotek: mp3, wav in ogg

# Digitalni video

Tako kot analogni video (celuloidni filmski trak) tudi digitalni sestoji iz okvirjev - zaporedja slik. V digitalnem videu so okvirji digitalne slike. Prednosti digitalnega videa nasproti analognemu so očitne. Na primer: digitalni video ne izgublja kakovosti s številom predvajanj, kopiranje in distribucija sta hitra, nudi močna orodja za urejanje.

# Osnovne lastnosti

## Gostota okvirjev

Število okvirjev na sekundo je ključnega pomena pri ustvarjanju občutka gibanja. Pri 25 ali več okvirjih na sekundo govorimo o videu s popolnim vtisom gibanja (ang. full-motion video).

## Vzorčenje

Digitalni video lahko izdelamo na dva načina: z računalniškim programom (animacijo) ali z vzorčenjem analognega video signala. V drugem primeru je treba analogni signal pretvoriti v digitalnega s procesom, imenovanim vzorčenje. Teorija vzorčenja določa spodnjo mejo frekvence vzorčenja, ki je dvakratna najvišja frekvenca znotraj signala (Nyquistova hitrost). Vsak vzorec je treba še kvantizirati.

# Osnovne lastnosti

## Gostota podatkov

Glede na frekvenco vzorčenja in kvantizacijo razdelimo današnje formate digitalnega videa v dve skupini: formati z visoko in formati z nizko gostoto podatkov. Primeri formatov iz prve skupine so:

* International Consultative Committee on Broadcasting Recommendation 601 - CCIR 601 (frekvenca vzorčenja 3.375 MHz in 8 (ali 10) kvantizacijskih bitov za vsako barvno komponento).
* Digital composite video (frekvenca vzorčenja za PAL = 4.43 MHz in 8 (ali 10) bitov kvantizacije).
* Common Intermediate Format - CIF (frekvenca vzorčenja je 6.75 MHz z 8-bitno kvantizacijo).

Pri formatih z visoko gostoto podatkov je treba prenašati okoli 20 Mbajtov/s (HDTV). Tega na žalost današnja strojna oprema (omrežje in enote DVD ) še ne zmore, zato te formate s stiskanjem, zmanjšanjem ločljivosti in zniževanjem gostote okvirjev pretvarjamo v formate z nižjo gostoto podatkov. Na ta način pade zahtevana hitrost prenosa na 1 Mbit/s, kar zmorejo tudi današnji cenejši računalniki.

## Stiskanje

Za zmanjšanje ogromne množine zahtevanega pomnilniškega prostora potrebujemo zmogljivo tehniko stiskanja z visoko stopnjo stiskanja. Za stiskanje digitalnega videa lahko uporabljamo tehnike z izgubami, saj izgubljena kvaliteta posameznega okvirja pogosto ni opazna.

Stopnjo stiskanja dvigujejo s tako imenovanim stiskanjem med okvirji (ang. interframe compression). Najdejo se razlike med zaporednimi slikami in potem stiskajo le te. Restavriranje nekompresiranega videa mora biti opravljeno v realnem času. Najuspešnejši standard za stiskanje digitalnega videa je MPEG.

# FORMATI datotek

## format AVI



Format **AVI** (angl. Audio Video Interleave) je razvil Microsoft in ga podpira večina računalnikov, ki deluje v Windows operacijskem sistemu. Je sicer eden od najbolj priljubljenih formatov, vendar ga pogosto ne moremo uporabiti na drugih sitemih, ki nimajo operacijskega sistema Windows. Končnica za dokumente v AVI formatu je .avi.

## format Windows Media format



Format **Windows Media** je prav tako pogost, vendar te vrste filmov ni moč predvajati na drugih računalnikih, ki nimajo operacijskega sistema Windows, ne da bi dodali (sicer brezplačne) predvajalne komponente. Končnica za dokumente v formatu Windows Media je .wmv ali .asf.

## format Adobe (Flash)



Format **flash** je trenutno v lasti Adoba. Zahteva dodatne komponente za predvajanje video posnetkov na računalniku. Končnica za dokumente v Flash formatu je .flv ali .swf.

## format MPEG



**MPEG** (angl. Movie Pictures Expert Group) format je standardiziran format, ki je tudi najbolj pogost na internetu in ga lahko uporabljamo na vseh platformah in ga samodejno podpira tudi večina brskalnikov. Končnica za dokumente v MPEG formatu je .mpeg.

## Format QuickTime



**QuickTime format** je razvilo podjetje Apple in je splošno znan format na Internetu, ki zahteva poseben predvajalnik. Končnica za dokumente v QuickTime formatu je .mov.

## format RealVideo



Format **RealVideo** je bil razvilo podjetje Real Media in omogoča sprotni pretok videoposnetkov (video v živo, internetna TV) pri majhnih hitrostih prenosa podatkov v omrežju. Zaradi majhne hitrosti prenosa, je tudi kvaliteta video posnetka ustrezno slabša. Končnica za dokumente formatu v RealTime je .rm ali .ram.

## format Matroska



Format **Matroska** je odprt format, ki je namenjen za kompresijo video posnetkov visoke resolucije. Končnica za dokumente v formatu Matroska je .mkv.

## Zaključek

Če želimo torej predvajati video, moramo imeti instaliran predvajalnik, ki lahko predvaja filme ne glede na standard kot na format dokumenta. To pomeni, da na primer na iPhone ne moremo gledati filma v standardu H.264, ker je zavit v dokument v formatu .mkv, ki ga iPhone morda ne podpira.

# Obroba

Lastnost **border** nam omogoča popolnoma spreminjati obrobe, ki se pojavljajo okoli elementov html.

**Slog obrobe**

Lastnost **border-style** določa slog obrobe.

none (CSS 1) Ne bo obrobe.

hidden (CSS 1) Isto kot none, razen za primere konfliktov z robovi za elemente tabel.

dotted (CSS 1) Definira pikčasto obrobo.

dashed (CSS 1) Definira črtkasto obrobo.

solid (CSS 1) Definira polno obrobo.

double (CSS 1) Definira dvojno obrobo. Širina obeh obrob je takšna, kot jo določa lastnost border-width.

groove (CSS 1) Definira 3D-žlebčasti rob. Učinek je odvisen od barve, ki se določi obrobi.

ridge (CSS 1) Definira rob tako, da je prikazan kot izbočen. Učinek je odvisen od barve, ki se jo določi obrobi.

inset (CSS 1) Definira 3D-vbočeni okvir. Učinek je odvisen od barve, ki se določi obrobi.

outset (CSS 1) Definira 3D-izbočeno obrobo. Učinek je odvisen od barve, ki se jo določi obrobi.

**Širina obrobe**

Lastnost **border-width** določi širino obrobe. Širina je podana v eni od veljavnih enot, ali pa z uporabo ene izmed v naprej določenih vrednosti, ki so odvisne od brskalnika.

Lastnost **border-width** ne učinkuje, če ne določimo tudi lastnosti **border-style**.

***širina*** vrednost za dolžino lahko vnesete v katerikoli od dolžinskih enot.

**thin** tanka obroba

**medium** srednje debela obroba, privzeta vrednost.

**thick** debela obroba

**inherit** širina obrobe je podedovana od starša

**barva obrobe**

Lastnost border-color se uporablja za določevanje barve obrobe. Barva se lahko določi na tri načine:

[*barva*](http://www.egradiva.net/moduli/spletne_aplikacije/primeri_html_css/109_border/index_color.html)  Barva je določena s posameznimi komponentami **RGB**. Barve se lahko določi na različne načine.

transparent Obroba nima barve.

inherit Barva obrobe se deduje.

# Skrajšani zapis

Kot vidimo na prejšnjih primerih, je veliko lastnosti, ki jih je treba napisati, ko želimo primerno obrobo. Da bi zmanjšali in poenostavili kodo, lahko določimo vse posamezne lastnosti obrobe v eni lastnosti. Ta lastnost je bližnjica za naslednje individualne lastnosti:

* border-width (širina obrobe),
* border-style (stil obrobe),
* border-color (barva obrobe).

# Osnovne animacije

Z CSS3 lahko ustvarjamo animacije, ki lahko zamenjajo animacije Flash, JavaScript.

Da lahko ustvarimo animacijo v CSS3 se moramo naučiti uporabljati @keyframe. Keyframe je tam kjer se ustvari animacija. Znotraj keyframe definiramo lastnosti animacije.

# Animacija

Ko je animacija v **@keyframe** ustvarjena, jo moramo povezati s selektorjem, drugače ne bo imela nobenega učinka. To lahko storimo tako, da definiramo najmanj dve lastnosti animacij:

* ime animacije
* trajanje animacije

Ime in trajanje je obvezno potrebno določiti. Če to ni storjeno, ne bo animacije.

# Kaj so animacije?

Animacija je efekt, ki dopušča elementom, da se iz enega izgleda spreminjajo v drugega. Videz lahko spremenimo kolikokrat želimo.

Določiti moramo, kdaj se bo sprememba zgodila. To zapišemo v odstotkih, ki pa so odvisni od trajanja efekta. Lahko uporabimo tudi ključne besede "from" in "to".

**Lastnosti animacij**

**@keyframes:** S ključnimi okvirji ustvarjamo animacije. V njih definiramo obnašanje animacije.

**animation name** Določa ime ključnih okvirjev @keyframe animacije.

**animation duration** Določa koliko sekund ali milisekdun se bo izvajal en cikel pri animaciji.

**animation timing function** Določa hitrost animacije; to je čas, ki ga animacija porabi pri prehodu iz enega stanja v drugega.

**animation delay** Določa začetek izvajanja animacije. Podajamo v sekundah ali milisekundah.

**animation iteration count** Definira število predvajanj animacije.

# Prehodi

**Osnovno**

S CSS3 lahko v besedilu dodajamo efekte (prehodi, kar pomeni, da nam pri "okraševanju" strani ni potrebno animacij Flash. V brskalnikih so prehodi različno definirani in podprti. Nekateri brskalniki zahtevajo za posamezna pravila predpono:

-moz- Mozzila Firefox

-webkit- Google Chrome in Safari

Opera -ms-

Interent Explorer

# Delovanje

CSS3 prehodnost dovoljuje elementom, da prehajajo iz enega videza v drugega.

Da je to možno morata biti izpolnjena 2 pogoja:

* definirana mora biti lastnost CSS kateri želimo dodati efekt
* definirano mora biti trajanje efekta.

**POMNI:** Če trajanje efekta ne bo določeno potem tudi sam efekt ne bo deloval.  
Če širina diva ni določena efekt ne bo viden v našem primeru.

Efekt bo začel delovati, ko bo definirana lastnost CSS spremenila vrednost. Pri tem največkrat uporabimo možnost, ko je kazalec nad besedilom (hover).

# Dodatne spremembe

Da lahko dodamo več efektov moramo tudi dodati več lastnosti, ki jih definirajo. Dodamo še lastnosti height in transform.

**Transition lastnosti**

**transition property**

Definira ime lastnosti CSS efekta. Vedno je potrebno definirati čas delovanje, ker drugač obdrži vrednost 0 in je trajanje efekta 0s.

**transtion durration**

Ta lastnost definira trajanje efekta v sekundah ali milisekundah.

**transition timing function**

Ta lastnost definira hitrost efekta. Dovoli spremembo hitrosti med samim izvajanjem.

**transition delay**

Določa, kdaj se bo izvajanje efekta začelo. Definiran je v sekundah in milisekundah.

# Border-radius - uvod

Border-radius je nova funkcija v CSS3.

Je ena izmed najbolj omenjenih novost v CSS3, saj lahko na preprost način zaoblimo robove.

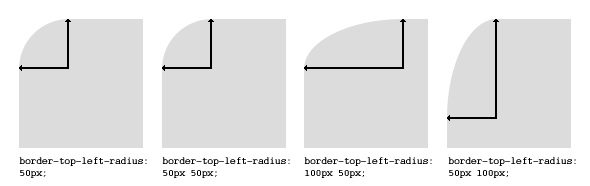
To je bila že dolgo zaželjena možnost v CSS, ker zdaj lahko zaoblimo robove z zelo malo kode.

**Kako deluje**

Lahko zaoblimo vsak rob posebej (border-bottom-left-radius, border-top-left-radius,...), lahko pa vse naenkrat (border-radius).

Zaželjeno dolžino lahko pišemo v procentih ali pikslih.

V treh primerih imamo dve vrednosti. Če je ena izmed vrednosti nič, rob ne bo zaobljen.



# Box-shadow

Box-shadow nam omogoča določitev senc. Senco lahko zamegljimo ter določimo velikost in barvo.

Sence lahko tudi dodamo našim likom, ki smo jih ustvarili s pomočjo "border-radius".

**Osnove lastnosti clip**

Lastnost clip je ena izmed manj uporabljenih lastnosti v CSS.

"Clip" omogoča odstranitev nezaželjenih robov slike. Trenutno je na voljo samo rect() funkcija, v novejčih različicah pa bo mogoče več možnosti, kot recimo circle().

**Sintaksa**

Lastnost **clip** ima učinek samo, če ima element lastnost "position:absolute" ali "position:fixed".

lastnost **clip** ima lahko tri vrednosti:

**auto**

Obdrži trenutno velikost slike, če imamo clip nastavljen na auto je isto, kot da ne bi imeli lastnosti clip.

**shape**

Obreže sliko, edina veljavna vrednost je "rect (top, right, bottom, left)".

**inherit**

Določa, da mora biti vrednost lastnosti "clip" podedovana od nadrejenega elementa

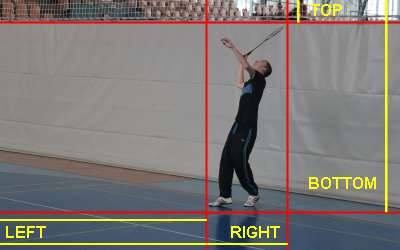
# Rect() funkcija

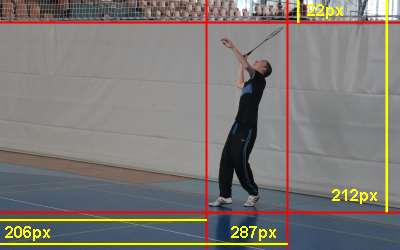
Rect() tudi omogoča, da uporabimo "auto" kot vrednost: rect(0, auto, auto, 0). Ne moremo pa napisati odstotkov namesto pikslov.

Funkcija rect() potrebuje štiri dolžine, tako kot margin in padding. Vendar se dolžine računajo malo drugače:

**clip:rect(<top>, <right>, <bottom>, <left>);**

Lahko je zelo čudna funkcija za začetnike, ampak ko vidite kako deluje je čisto preprosta.





# Skriptni jezik JavaScript

JavaScript je skriptni programski jezik. Vsak sodobni spletni brskalnik ima že vgrajeno podporo za izvajanje le-tega. Javascript je eden izmed najboljših načinov za dodajanje interaktivnosti na spletno stran. Je edini programski jezik, ki deluje direktno v spletnem brskalniku (angl. Crossbrowser language). Je precej preprost za uporabo in ne potrebuje nobene dodatne strojne ali programske opreme, tudi spletnega strežnika ne. Potrebujete le brskalnik in urejevalnik besedila. Vse modernejše spletne strani HTML danes uporabljajo JavaScript za dodajanje dodatnih funkcionalnosti, preverjanje veljavnosti vnesenih podatkov, komunikacijo s spletnimi servisi, odpiranje dodatnih oken itd.

JavaScript je razvilo podjetje Netscape leta 1995 z namenom, da bi spletnim programerjem pomagali pri ustvarjanju interaktivnosti spletnih strani. Danes ga podpirajo vsa velika podjetja in vsi modernejši brskalniki. Razvit je bil neodvisno od programskega jezika Java, vendar si z njim deli številne lastnosti in strukture. Podpira tudi objektno orientirani pristop programiranja. Uporablja se tudi izven spleta, in sicer v različnih orodjih:

Recimo Adobe Acrobat ga podpira v datotekah PDF, podpirata pa ga tudi operacijska sistema Microsoft Windows in Mac OS X. V zadnjem času je jezik pridobil na priljubljenosti, ker so se pojavili VMs (angl. virtual machines) in ogrodja (predvsem Node.js), ki omogočajo izvajanje »server-side« spletnih aplikacij. Pojavilo se je tudi nekaj ogrodij, ki omogočajo hitrejšo izvedbo določenih interaktivnosti (jQuery). Trenutna različica JavaScript-a je 1.8.5 in ustreza specifikaciji

ECMAScript 5 compliance.

Primer kode v jeziku Javascript:

<script type="text/javascript">

alert("Pozdravljen svet!");

</script>

Ta primer bo izpisal v pogovornem oknu besedilo »Pozdravljen svet«. Na tak način lahko vključimo JavaScript kodo v datoteko HTML. Obstaja pa tudi druga možnost, da celotno JavaScript kodo pišemo v posebno zunanjo datoteko in jo nato pripnemo v dokument HTML. Primer:

<script type="text/javascript" src="primer.js"></script>

## Zgradba JavaScript

Jedro vsebuje vse skupne elemente jezika - sintakso, operatorje in stavke pa tudi nekatere ključne skupne predmete in funkcije. Vseeno pa je treba opozoriti, da so nekateri elementi uporabljeni drugače na strežniški in drugače na odjemalčevi strani. Jedro je zgrajeno v skladu s standardom ECMA-262, standardnim skriptnim jezikom, z nekaterimi dodatki.

Odjemalsko stran predstavlja internetni brskalnik. JavaScriptu so dodani predmeti, ki jezik povezujejo z brskalnikom. Tukaj je JavaScript zapisan kar v HTML dokumentih in ga brskalnik v celoti interpretira, ko uporabnik odpre HTML dokument, ki vsebuje JavaScript program.

Ko uporabnik naloži stran v brskalnik, le-ta interpretira izgled strani, kot ga določa HTML in tudi morebitne JavaScript programčke. V programčkih se lahko odzivamo na različne dogodke, ki jih sproži uporabnik (npr. miškin klik in podobno). To nam omogoča, da recimo preverjamo vnos že pri odjemalcu in ne šele ko bi podatki potovali preko mreže do strežnika.

Strežniška stran je tudi zapisana v obliki dokumenta HTML, dostopne pa so nam tudi številne možnosti, ki jih na odjemalski strani ne moremo početi - lahko dostopamo do podatkovnih baz različnih proizvajalcev, do datotek, ki so shranjene na strežniku, izmenjujemo podatke med uporabniki istega JavaScript programa in pa tudi komuniciramo z drugimi aplikacijami s pomočjo LiveConnect oziroma Jave. V dokumentu, ki vsebuje JavaScript za strežnik, je lahko tudi program za odjemalca, ki se bo seveda izvajal v odjemalčevem brskalniku. Program, ki naj se izvaja na strežniku, zapišemo med oznaki <server> in </server> . Zaradi večje kompleksnosti dokumentov HTML na strežniški strani, je potrebno le-te prevesti s posebnim prevajalnikom v vmesno obliko:

### Prevajanje dokumentov na strežniku

Rezultat prevajanja je binarna datoteka, ki vsebuje tako dokument HTML kot tudi programčke. Ko odjemalec zahteva tako preveden dokument, ga mora strežnik ustrezno interpretirati (ob tem tudi izvaja programčke, ki so bili zapisani med značkama <server> in </server> ), rezultate pa pošilja odjemalcu.

JavaScript je programski jezik, ki spadata v skupino tolmačev. To pomeni, da se napisana koda programa ne prevede v program, ki bi ga lahko shranili v obliki izvršilne datoteke, pač pa poseben tolmač po vrsti razčlenjuje in izvršuje ukaze, ki smo jih napisali. JavaScript je predmetno usmerjen programski jezik, na prvi pogled precej podoben programskima jezikoma Java in C++. Kdor je že kdaj napisal kakšen program v enem od teh dveh jezikov, z JavaScriptom ne bo imel posebnih težav. Precej predmetov je v JavaScriptu že definiranih in jih lahko takoj uporabimo v svojih programih.

JavaScript je programski jezik, ki ima več uporab. Največkrat ga srečamo v spletnih brskalnikih (Client-Side JavaScript), pri Netscapu pa so ga vgradili tudi v svoj omrežni strežnik (Server-Side JavaScript). Poleg tega lahko JavaScript vključujemo v različne vrste dokumentov, kot so HTML, XML, VRML, in celo PDF. Elemente jezika JavaScript lahko razdelimo na tri nivoje:

### Jedro (Core)

Jedro vsebuje tiste elemente jezika, ki so skupne vsem različnim uporabam. Sem se uvrščajo pravila o zgradbi programa (sintaksa), pravila o sestavljanju izrazov in stavkov, vgrajeni predmeti in funkcije...

### Gostitelj (Host)

Glede na to kje se uporablja JavaScript, imamo na razpolago dodatne predmete, ki opisujejo trenutno stanje okolja. Če na primer uporabljamo JavaScript pri pregledovanju dokumentov HTML, imamo na razpolago predmete, ki opisujejo okno brskalnika, zgradbo dokumenta, evidenco o obiskanih straneh ...

### Uporabnik (User)

Vsak uporabnik lahko v JavaScriptu definira svoje funkcije in predmete, ki jih nato uporablja, kot da so del jezika samega.

## Jedro

Jedro skriptnega jezika JavaScripta predstavlja standard ECMA-262 z nekaterimi dodatki. Standard določa sintakso, nekatere osnovne predmete, njihove postopke in lastnosti in še nekatere globalne funkcije. Tako mora vsaka implementacija skriptnega jezika, ki bo v skladu s standardom ECMA-262, podpirati naslednje predmete:

globalni predmet (lahko je tudi brez imena),

Object,

Function,

Array,

String,

Boolean,

Number,

Math in

Date.

Ti predmeti imajo določene tudi lastnosti in postopke, ki tudi morajo biti implementirani. Standard je tukaj zelo prilagodljiv, saj določa le osnovni nabor funkcionalnosti, ki jih mora podpirati skriptni jezik, dovoljuje pa poljubno razširjanje - lahko se doda lastne predmete, dodatne postopke in lastnosti predmetov, lahko se celo razši sintakso (standardna sintaksa seveda mora ostati nespremenjena).

Pri Netscapu so standardni implementaciji dodali le predmet RegExp (iz jezika Perl), ki se uporablja za delo z regularnimi izrazi, in seveda posameznim predmetom mnogo dodatnih lastnosti in postopkov. Nekoliko je bila razširjena tudi sintaksa.

### Sintaksa

Sintaksa JavaScripta je zelo podobna sintaksi Jave. Oba jezika poznata predmete, v nobenem ni kazalcev, v obeh je večina entitet, s katerimi delamo, predmet. Do razlik pride pri naprednejših lastnostih, ki jih v JavaScriptu ni (npr. napredna obravnava izjem, izpopolnjen dogodkovni model, veliko število že definiranih gradnikov uporabniškega vmesnika), oziroma pri lastnostih, ki so neposredno povezane z dejstvom, da je JavaScript interpretiran (dinamično dodajanje razrednih atributov, dinamično ustvarjanje razredne hierarhije). Primerjava z Javo je v veliki meri neumestna, saj bi primerjali jabolka in hruške. Na eni strani Java poskuša zgraditi nadomestek operacijskega sistema, JavaScript pa je le delček internetnega brskalnika, sicer zelo pomemben, a le delček.

### Stavki

Stavek JavaScripta je sestavljen iz rezervirane besede, ki jih uporabimo z določeno sintakso. En stavek se lahko razteza preko več vrstic. V eni vrstici lahko zapišemo več stavkov, ki morajo v tem primeru biti ločeni s podpičjem. Podpičja na koncu stavka so sicer dodana po želji. Priporočljivo pa jih je uporabiti. Mogoče bo v naslednji različici uporaba obvezna.

## Aplikacije JavaScript

JavaScript močno orodje za pisanje aplikacij na uporabniški strani. Toda vsak jezik ima svoje omejitve. Tipična uporaba JavaScript je:

* pošiljanje obrazcev
* posebni učinki na spletnih straneh
* navigacija
* matematične funkcije
* generiranje dinamičnih lastnosti

JavaScript je bil narejen prav za preverjanje in pošiljanje obrazcev strežniškim aplikacijam, ko so npr. programi CGI. To je poznano kot preverjanje obrazcev (form validation). Po prvih začetkih, so brskalniki pričeli dopuščati izvajanje raznih akcij ob miškinih dogodkih in s tem omogočali razne animacije. Tipično so z Javascript napisani kompleksni navigacijski sistemi, kot so padajoči meniji, razni seznami. JavaScript je bil načrtovan za izračune na uporabniški strani in je uporaben za obiskovalce spletnih strani. Ta koncept je poznan kot dinamični HTML (DHTML). Za njegovo razumevanje je potrebno poznati DOM (Document Object Model). Netscape se je šele z Navigatorjevo različico 6 odločil prevzeti DOM (predpisuje ga tudi W3C) namesto trmastega vztrajanja pri lastnem nestandardiziranem modelu JSOM (JavaScript Object Model). To bo povečalo možnosti programiranja z JavaScript, saj ne bo potrebno izdelovati različic za več brskalnikov.

# Ajax tehnologija

**Ajax** (asinhroni JavaScript in XML) je skupina medsebojno povezanih razvojnih tehnik, ki

omogočajo ustvarjanje interaktivnih spletnih aplikacij. Prednost tehnologije Ajax je v tem, da omogoča spletnim aplikacijam, da prenašajo podatke iz spletnega strežnika **asinhrono v ozadju**, brez potrebe po osveževanju celotne strani. To omogoča spletnim aplikacijam **hitrejše in tekoče spreminjanje vsebine**, saj ne prenašajo celotne strani, ampak samo del strani. Podatki se prenašajo s pomočjo objektov **XMLHttpRequest**. Uporaba teh tehnologij je značilna za Web 2.0. Čeprav se v imenu nahajata tehnologija JavaScript in XML, prisotnost teh ni pogoj za izvajanje tehnologije Ajax. JavaScript lahko nadomestimo s jezikom VBScript, namesto jezika XML pa recimo uporabimo JSON (angl. *Java Script Object Notification*).

Tehnologija Ajax se prvič omenja leta 2005, ko jo je v svojem članku prvič omenil Jesse James Gerrett. Vendar temelji segajo že v leto 1996, ko je podjetje Microsoft za svoj brskalnik Internet Explorer razvilo konstrukt IFrame. Leta 1999 je Microsoft ustvaril objekt XMLHttpRequest kot kontrolnik ActiveX. Temu so sledili drugi razvijalci brskalnikov (Mozilla, Opera in Safari). Aprila 2006 je nadzor nad tehnologijo Ajax prevzel konzorcij W3C in pripravili so prvi osnutek specifikacije XMLHttpRequest. Pozneje je to postal uradni standard. Vidno vlogo pri razvoju ima tudi podjetje Google, ki je kot prvo podjetje predstavilo zmogljivejši spletni aplikaciji, ki temeljita na tehnologiji Ajax, in sicer Gmail (2004) in Google Maps (2005).

Preprosti primer napisan v JavaScript-u:

//inicializacija Ajax zahtevka

var xhr = XMLHttpRequest();

xhr.open('get', 'pridobi\_podatke.php');

//pošljemo zahtevek za pridobitev podatkov od skripte pridobi\_podatke.php

xhr.send();

//pridobimo besedilo od skripte pridobi\_podatke.php

var response = xhr.responseText;

//izpis

alert(response);

Na začetku torej ustvarimo objekt XMLHttpRequest in ga inicializiramo (open). V prvem atributu določimo HTTP metodo (GET ali POST), v drugem pa URL naslov skripte, ki bo vračala podatke. V zgornjem primeru bo skripta pridobi\_podatke.php preprosto izpisala del besedila. V nadaljevanju pošljemo zahtevek (send) za pridobitev podatkov od skripte in dobimo odgovor v obliki besedila (responseText). V zadnji vrstici izpišemo dobljeno besedilo. Danes v praksi redko uporabljamo Ajax na način, kot je zapisan v zgornjem primeru, saj obstaja kar nekaj ogrodij JavaScript (JQuery, Mootools), ki omogočajo uporabo tehnologije Ajax na preprostejši način. V nadaljevanju bomo tudi predstavili ogrodje JQuery in si ogledali, kako bi isti primer zapisali v knjižnici JQuery. Ajax sigurno velja za prelomno tehnologijo na področju razvoja spletnih aplikacij, saj je prav ta tehnologija približala spletne aplikacije namiznim aplikacijam. glede same uporabniške izkušnje. Omogoča tudi hitrejše delovanje spletnih aplikacij in prepreči nepotreben prenos celotne vsebine.

# Knjižnica JQuery za skriptni jezik JavaScript

**JQuery** je knjižnica, napisana v jeziku JavaScript, in vsebuje zbirko funkcij, ki omogoča hitrejši razvoj spletnih aplikacij. Zaseda zelo malo prostora, je zelo hitra in omogoča ogromno dodatnih funkcionalnosti. Omogoča stvari, kot so prikaz in manipulacija elementov HTML, upravljanje z dogodki, animacijo in uporabo tehnologije Ajax na precej

preprostejši in hitrejši način v primerjavi s »čistim« jezikom JavaScript. Deluje v vseh novejših brskalnikih in to brez prilagajanja, kajti isto stvar v različnih brskalnikih implementira drugače, torej na način, ki podpira brskalnik. Prva verzija se je pojavila januarja 2006, njen avtor je bil John Resig. Danes je daleč najbolj priljubljena knjižnica JavaScript, uporablja jo več kot 65 % od 1000 najbolj obiskanih strani.

Knjižnica JQuery je bila razvita z namenom, da bi skrajšala čas dodajanja dinamičnih funkcionalnosti v spletno aplikacijo. Osnovni slogan knjižnice je, da z manj kode naredimo

več (angl. Whrite Less, Do More).

Primer, ki vsaki sodi vrstici v tabeli spremeni CSS stil. Rešitev bi v jeziku JavaScript obsegala sigurno več kot 10 vrstic kode, rešitev v JQuery-u pa bi lahko zapisali v eni vrstici, in sicer:

$("table tr:nth-child(even)").addClass("striped");

JQuery knjižnica je brezplačna odprtokodna rešitev in temelji na licenci MIT. Sintaksa je precej preprosta in hitro učljiva. Knjižnico preprosto snamemo iz uradne spletne strani (<http://jquery.com/> ) in je vstavimo v svoj HTML dokument na sledeči način:

<script type="text/javascript" src="jquery.js"></script>

Na voljo imamo dve različici, in sicer v izvorni različici in minimizirano. Ponavadi uporabljamo slednjo, saj zasede manj prostora, razen če nas zanima izvorna koda, kajti »min« različica je zapisana v nerazumljivi obliki. Knjižnico si lahko naložimo tudi preko Google Libraries API. Prednost tega načina je, da se knjižnica v primeru, če je uporabnik že obiskal kakšno spletno stran, ki uporablja isto knjižnico, do njega ne prenaša ponovno.

Sintaksa uporabe knjižnice je sila preprosta. Ko želimo uporabljati JQuery, vedno začnemo z rezerviranim znakom $ (dolar). JQuery uporabljamo na dva načina:

1. Dostop do HTML elementov in manipulacija z njimi.

Oglejmo si primer, ki izbriše vrednost vsem input elementom:

$("input").val("");

Funkcija $() vrne poseben Javascript objekt, ki vsebuje polje DOM elementov, ki se ujemajo z izbiro (selektor). Torej v našem primeru ta funkcija vrne objekte vseh input elementov. Sintaksa določitve elementov je identična CSS sintaksi.

Recimo, da funkcija $('div.test p') vrne objekte vseh odstavkov, ki se nahajajo znotraj označbe div in z atributom class='test'. Torej $() funkcija vrne objekte, nad katerimi potem manipuliramo; torej v našem primeru nastavimo vsem objektom prazno vrednost.

2. Direktna uporaba funkcij

V tem primeru ne izvajamo manipulacije nad elementi HTML, ampak kličemo direktno JQuery funkcijo. Sintaksa je torej sledeča:

**$.ime\_funkcije(atributi\_funkcije);**

Oglejmo si še primer iz prejšnjega poglavja, in sicer uporaba tehnologije Ajax:

**$.get('pridobi\_podatke.php', function(data) {**

**alert(data);**

**});**

Obstaja tudi dodatna knjižnica, ki je zgrajena na osnovi ogrodja JQuery, omogoča pa interaktivno gradnjo spletnih aplikacij; to je knjižnica JQuery UI. To dodatno knjižnico bi lahko poimenovali tudi CSS ogrodje, saj je predvsem osredotočena na uporabniški vmesnik spletnih aplikacij. Na voljo imamo celo vrsto komponent: koledar za izbiro dneva, zavihki, pojavno okno, razni efekti, spustni seznami, drsniki. Na voljo je tudi ogromno pripravljenih grafičnih predlog, ki jih lahko tudi po želji modificiramo.

# Odzivni spletni dizajn

V zadnjih letih se je enormno povečalo število lastnikov pametnih telefonov in tablic. Vse več ljudi uporablja svoje pametne telefone in tablice tudi za brskanje po internetu in posledično tudi uporabo spletnih aplikacij. Ker so zasloni na teh napravah v primerjavi s klasičnimi računalniškimi zasloni še vedno dokaj majhni, je uporaba klasičnih spletnih aplikacij precej otežena in nerodna. Če imamo recimo zaslon na pametnem telefonu dimenzije 480 slikovnih pik, klasične spletne aplikacije pa širine vsaj okrog 1000 slikovnih pik, pomeni, da privzeto vidimo manj kot polovico aplikacije. Sicer lahko sliko pomanjšamo, ampak potem nastane problem, da je slika premajhna, besedilo pa neberljivo. Vsekakor je to lahko zelo nerodno. Zato so se pojavile rešitve, ki izgled spletnih aplikacij prilagajajo glede na širino zaslona uporabljene naprave.

Odziven spletni dizajn (angl. Responsive web design - **RWD**) je pristop, ki zagotovi optimalno spletno izkušnjo in maksimalno izkoristi medij, ki ga uporabljamo. Omogoči lažje branje in pomikanje po straneh (navigacijo) brez spreminjanja velikosti, neodvisno od naprave oz. velikosti zaslona.

Izraz RWD je skoval Ethan Marcotte leta 2010 v svojem članku z imenom »A List Apart«, leta 2011 pa je izdal knjigo »Responsive Web Design«, kjer je podrobneje opisana tehnologija, podani so tudi primeri. RWD dosežemo s pomočjo treh stvari Prilagodljiva postavitev spletne strani, ki temelji na t. i. mreži. To v praksi pomeni, da vsem elementom določamo relativne velikosti, namesto v slikovnih pikah (procentualne vrednosti).

Primer:

.blog { margin: 0 auto 53px; width: 900px;}

.blog { margin: 0 auto 53px; width: 93.75%; /\* 900px / 960px \*/}

• Prilagodljiva velikost slik in drugih medijev na spletni strani. Slikam in ostalim medijem določimo relativne (procentualne) velikosti, tako se izognemo, da je slika vedno prikazana v celoti.

Primer:

img { max-width: 100%;}

• Uporaba »media poizvedb«, ki je del CSS3 specifikacije. Media poizvedbe omogočajo spremembo CSS lastnosti glede na lastnosti naprave, ki jo uporabljamo pri prikazu spletne strani oz. aplikacije.

Primer:

/\* Small devices (tablets, 768px and up) \*/

@media (min-width: @screen-sm) { ... }

/\* Medium devices (desktops, 992px and up) \*/

@media (min-width: @screen-md) { ... }

/\* Large devices (large desktops, 1200px and up) \*/

@media (min-width: @screen-lg) { ... }

RWD je vsekakor odlična zadeva, saj prikazuje želeno spletno vsebino na vseh medijih na optimalen način. Ima pa tudi nekatere slabosti. Problem predstavljajo starejši brskalniki, ki še ne podpirajo standarda CSS3. Če se želimo izogniti tem nevšečnostim, moramo prvo pametno zgraditi spletno stran oz. aplikacijo, ki deluje na vseh napravah. Šele nato jo nadgradimo za naprednejše mobilne naprave in računalnike, ki novejše tehnologije razumejo. Tukaj velja omeniti, da omejitve, ki jih prinašajo manj zmogljive naprave in brskalniki, moramo upoštevati na strani strežnika in ne na strani odjemalca. S tem pridobimo predvsem to, da ne prenašamo nepotrebnih podatkov iz strežnika, ki jih ne bomo rabili za prikaz. Drug problem sta uporaba pasice Flash in banerji, ki so fiksne širine in višine, torej jih ni mogoče prilagajati. Ogromno strani je financiranih samo iz oglasov, ki se nahajajo na spletni strani, ti oglasi pa so ponavadi v formatu Flash.

Obstaja pa tudi cela vrsta ogrodij, ki omogočajo RWD (Bootstrap, Foundation, Gumby, Unsemantic, Skeleton).

# Bootstrap

**Bootstrap** je brezplačna zbirka orodij, ki olajša grajenje spletnih strani oz. aplikacij. S pomočjo jezikov HTML, CSS in JavaScript omogoča enostavno in robustno grajanje izgleda spletne strani oz. aplikacije in to z minimalno količino kode. S pomočjo ogrodja Bootstrap lahko enostavno in hitro uporabimo elemente, kot so elementi vnosnega obrazca, gumbki, navigacija, tabele. Omogoča, da vse elemente porazdelimo v obliki mreže, kar omogoča zelo pregleden izgled strani in dizajn, odziven na širino zaslona (RWD). Omogoča pa tudi vtičnike JavaScript, ki stran naredijo interaktivno (spustni seznami, modalna okna, zavihki, opozorila).

Če ste se že kdaj spopadli z izdelavo spletnih aplikacij, vam je verjetno ogromno časa vzel sam izgled aplikacije. V praksi je ponavadi tako, da se na osnovi koncepta in žičnih diagramov najprej naredi grafični dizajn v programu za grafično oblikovanje (PhotoShop, CorelDraw). Potem je potrebno ta dizajn spremeniti v HTML in CSS izvorno kodo (razrez), za kar spet porabimo ogromno časa. Ker večje aplikacije vsebujejo ogromno elementov, se pogosto zgodi, da določenih elementov nimamo oblikovanih; potem se programerji moramo znajti sami, pri tem pa izgubimo tudi ogromno časa. Tudi pri dodajanju interaktivnosti spletni aplikaciji izgubimo precej časa, kajti poiskati moramo prave module oz. knjižnice, poleg tega pa moramo poznati tudi uporabniške izkušnje, kateri modul oz. knjižnico in na kakšen način sploh uporabiti. In ravno to omogoča ogrodje Bootstrap.

Bootstrap sta razvila Mark Otto in Jacob Thornton leta 2011. Ko sta izdala prvo verzijo, sta bila oba zaposlena v podjetju Twitter. Potrebovali so standardno orodje za razvoj aplikacij znotraj podjetja. Bootstrap je hitro postal zelo popularno orodje in februarja 2012 je postal najbolj popularen razvojni projekt na portalu GitHub.

Bootstrap si preprosto snamemo z njihove spletne strani (http://getbootstrap.com/). Trenutna verzija je 3.0.0 (12. september 2013) in spada pod licenco Apache Licence 2.0. Ko razpakiramo sneto verzijo, vsebuje naslednjo strukturo Datoteke so smiselno razporejene v tri mape, in sicer: css, js in fonts. Paket vsebuje izvorno kodo v normalni in minimalistični obliki (.min). Preprosto jih dodamo na koren (root) projekta in vključimo v dokument HTML. Oglejmo si preprosti primer dokumenta HTML, v katerem bomo uporabili Bootstrap.

Vsekakor je ogrodje Bootstrap odličen pripomoček za pohitritev razvoja spletnih aplikacij in tudi za zmanjšanje stroškov. Edini problem, ki ga velja izpostaviti, je priljubljenost ogrodja: čez nekaj let se lahko zgodi, da bo ogromno spletnih aplikacij in spletnih strani grafično precej podobnega izgleda.

# Programski jezik PHP

Jezik **PHP** je odprtokodni programski jezik, ki se uporablja za razvoj dinamičnih spletnih vsebin in aplikacij. V začetku razvoja je okrajšava PHP pomenila Personal Home Page Tolls (orodja za razvoj osebnih spletnih strani), pozneje pa se je okrajšava morala preimenovati, ker več ni ustrezala funkcionalnosti jezika. Pravi pomen okrajšave je danes Hypertext Preprocesor. Gre za enega izmed prvih strežniških skriptnih jezikov, ki vstavi kodo HTML v izvorni dokument, namesto da bi direktno klical zunanjo datoteko. PHP torej primarno teče na spletnem strežniku, kjer s pomočjo PHP modula kreira izvorno kodo za prikaz na spletu. PHP pa se lahko uporablja tudi na druge načine, in sicer preko ukazne vrstice ali pa ga lahko uporabljamo v samostojnih grafičnih aplikacijah. Mogoče ga je namestiti na večino spletnih strežnikov, na večino operacijskih sistemov in je na voljo brezplačno.

Če primerjamo PHP z ostalimi strukturnimi programskimi jeziki, je najbolj podoben jezikoma C in Perl. Je pa PHP vsekakor enostavnejši za uporabo, saj ne vsebuje številnih nizkonivojskih zmogljivosti programiranja in je oblikovan za izdelavo spletnih strani. PHP je daleč najbolj priljubljen in uporabljen jezik za razvoj spletnih strani, saj je uporabljen že na več kot 244 milijonov spletnih strani in je nameščen na več kot 2,1 milijona spletnih strežnikov (podatek 29.10.2013).

# MySQL

Danes si več ne predstavljamo spletne aplikacije, ki ne uporablja podatkovne baze za shranjevanje podatkov. Podatkovno bazo si lahko predstavljamo kot shrambo vseh podatkov v elektronski obliki, ki omogoča preprost in hiter dostop do podatkov. MySQL je sistem za upravljanje z relacijskih podatkovnih baz - **RDBMS** (angl. Relational DataBase Managment System). Je implementacija podatkovne baze, ki za delo s podatki uporablja jezik SQL (angl. Structured Query Language) .

MySQL si lahko brezplačno snamemo njihove uradne spletne strani <http://www.mysql.com>. Brezplačna različica (MySQL Community Edition) je najbolj popularen odprtokodni sistem za upravljanje relacijskih podatkovnih baz na svetu, še posebej v povezavi s programskim jezikom PHP. Brezplačna različica je dosegljiva pod licenco GNU; če izpolnjujete pogoje, lahko uporabite to verzijo, namenjena je tudi za profitne spletne strani in aplikacije. Če vam licenca GNU ne ustreza, imate na voljo tudi tri komercialne plačljive različice, in sicer MySQL Standard Edition, MySQL Enterprise Edition in MySQL Cluster Carrier Grade Edition. Brezplačna različica ima veliko in aktivno skupnost za pomoč razvijalcem.

Izpostavimo nekatere dobre lastnosti MySQL:

* Možna namestitev na več kot 20 platformah in operacijskih sistemih (Linux, Unix, Mac, Windows ).
* Prenosljivost med različnimi platformami.
* Podpora različnim načinom shranjevanja podatkov - Storage Engines (InnoDB,
* MyISAM, NDB, Memory, Archive, CSV). MyISAM uporablja zelo hitre tabele s kompresijo indeksa.
* Napisan v C in C++.
* Testiran z različnimi orodji.
* Aplikacijski programski vmesnik (API) je na voljo za različne programske jezike (C,
* C++, Java, Perl, PHP, Python, Ruby, TCL in Eiffel).
* Zelo hiter omrežni sistem za razporeditev pomnilnika.
* Strežnik je na voljo kot ločen program za uporabo v omrežnem okolju strežnik - odjemalec.
* Podpora za upravljanje velikih podatkovnih baz.

MySQL je del zelo popularne izbire za spletne aplikacije, še posebej v povezavi s programskim jezikom PHP. Je tudi centralna komponenta zelo razširjenega in priljubljenega paketa odprtokodne programske opreme LAMP (Linux, Apache, MySql in Perl/PHP/Python). LAMP torej skupaj tvori popolnoma delajoči spletni strežnik za gostovanje spletnih strani in aplikacij. Namestitev je zelo preprosta in vzame nekaj minut časa. Obstaja pa tudi verzija za operacijski sistem Windows, in sicer se imenuje WAMP.

MySQL podatkovne baze uporabljajo tudi v najbolj znanih in razširjenih spletnih aplikacijah, in sicer v Facebook-u, Twitter-ju, YouTube-u, Flickr-ju, Wikipedi-i, pa tudi v najbolj razširjenih CMSsistemih, in sicer so to TYPO3, Joomla, WordPress, Drupal, Magetno in drugi.

Zraven brezplačnega podatkovnega strežnika si lahko z uradne spletne strani snamete tudi nekaj drugih odprtokodnih brezplačnih orodij, ki so prav tako del DPL licence. Tukaj bi posebej izpostavili MySQL Workbench, ki je razvit s strani podjetja Oracle. Omogoča uporabniški vmesnik za administracijo in grafično načrtovanje podatkovnih baz (E-R diagram).

# Ogrodje CodeIgniter

**CodeIgniter** (CI) je odprtokodno ogrodje (angl. Framework) za izdelavo dinamičnih spletnih strani in spletnih aplikacij v programskem jeziku PHP. Glavni namen CI je, da omogoči razvijalcem hitrejši razvoj, predvsem pri razvoju aplikacij od začetka, saj vsebuje ogromno število knjižnic, ki razvoj zelo olajšajo. Prva javna verzija za razvijalce se je pojavila februarja 2006.

Trenutno najbolj priljubljeno in razširjeno ogrodje PHP. Njegov uspeh se skriva v preprostosti in hitri krivulji učenja. Ustvarja ga podjetje EllisLab. Po namestitvi že vsebuje generirano MVC strukturo. Sama izvorna koda ogrodja zavzema zelo malo prostora (če odstranimo uporabniška navodila, manj kot 2MB). Preprosto je dodajati zunanje knjižnice tretjih oseb. V primerjavi z ostalimi ogrodji PHP ima najbolj dejavno skupnost, prav tako pa ima odlično dokumentacijo za razvijalce. Vsekakor je ogrodje priporočljivo za manj izkušene uporabnike. Največja pomanjkljivost tega ogrodja pa je, da omogoča ogromno svobode pri pisanju kode, kar ni vedno dobro. V nekaterih primerih je ogrodje precej dovzetno za napake in lahko podaljša proces razvoja.

# Prednosti in namen uporabe ogrodij

Ogrodje PHP je pravzaprav množica datotek, ki določajo pravila glede kodiranja in lokacije določenih datotek znotraj projekta. Datoteke so torej razvrščene v vnaprej določenih mapah, kar omogoča boljše upravljanje projekta. Poleg tega pa večina ogrodij zagotavlja kup knjižnic, ki omogočijo hitrejši razvoj.

Ko gradimo določeno aplikacijo od začetka (angl. From Strach), se zna zgoditi, da ogromno časa vzame sama priprava. Pripraviti si moramo strukturo datotek in map, navigacijo po strukturi, napisati osnovne funkcije (povezava s podatkovno bazo, pošiljanje pošte, razne funkcije za delo z datumi ). Če pa uporabimo pripravljeno ogrodje, ki ima že vključene vse te funkcionalnosti, prihranimo ogromno časa.

Pri razvoju spletnih aplikacij enostavno ni prostora za napake v fazi načrtovanja. Če ustvarjamo nekaj brez celotne zasnove sistema in ne gledamo na prihodnost sistema, se nam v poznejših fazah razvoja zna zgoditi, da bomo imeli kar nekaj problemov, čas razvoja pa se zna bistveno podaljšati. Prav zaradi tega razloga so se pojavila ogrodja, ki take probleme rešujejo.

Značilnosti PHP ogrodij:

* **Generiranje poizvedb** - Danes skoraj ni več spletnih aplikacij, ki bi delovale brez podatkovne baze. Vedno je potrebno določene podatke shranjevati, delati poizvedbe po določenih podatkih, povezovati podatke po tujih ključih itd. Če naredimo napako pri kreiranju poizvedbe, nam ta napaka prinese precej nevšečnosti. Ogrodja PHP pomagajo rešiti ta problem.
* **Upravljanje s formo** - Velikokrat se zgodi, da potrdimo vnosno formo za vnos podatkov v podatkovno bazo in se pojavi napaka. Začnemo ugotavljati v čem je problem in preden ga rešimo, preteče že deset minut. V teh primerih lahko pomaga uporaba ogrodij. Ogrodja zmanjšajo možnost take napake, kajti morebitne napake sproti preverjajo, poleg tega pa z njimi napake dosti lažje odkrijemo. Upravljanje s formo vsebuje tudi preverjanje vpisanih podatkov znotraj forme (angl. Form Validation).
* **Upravljanje s stanji** - Stanja na spletnih straneh in aplikacijah uporabljamo zato, da si zapomnijo določene stvari, ki jih uporabnik upravlja na strani. Najbolj tipičen primer stanja je prijava v sistem. Ogrodja torej omogočajo lažje upravljanje teh stanj.
* **MVC arhitektura** - Model-view-controller (MVC) je programska arhitektura, ki se uporablja pri razvoju programske opreme . V bistvu gre za arhitekturni vzorec, ki ločuje domensko oz. poslovno logiko od uporabniškega vmesnika (pogleda), kar omogoča neodvisen razvoj, lažje testiranje in kasnejše vzdrževanje oz. nadgradnjo programske opreme.
* **Organizacija datotek in map** - Ko namestimo ogrodje PHP, se ustvari določena struktura map. Od nas pa se pričakuje, da se držimo določenih standardov in vse organiziramo na enak način. Ko bomo začeli uporabljati ta model, se bomo tega vedno držali. To nam omogoča lažje iskanje določenih delov kode oz. funkcionalnosti, saj bomo takoj vedeli, kam določen del kode spada.
* **Velik nabor knjižnic in orodij** - Vsekakor je PHP odličen programski jezik. Vsebuje velik nabor osnovnih orodij in knjižnic. Če smo že kdaj razvijali kakšno spletno aplikacijo ali spletno stran, se nam je verjetno že zgodilo, da smo uporabili kakšne obstoječe knjižnice oz. funkcije od drugih oseb (angl. Third Party). Načeloma lahko vse napišemo sami, ampak za to bi potrebovali ogromno časa.
* Ogrodja PHP imajo torej že v osnovno velik nabor knjižnic in funkcij, ki rešijo zgornji problem. Vsebujejo torej funkcije in knjižnice, ki skrbijo za validacijo forme, filtriranje vhodnih iz izhodnih podatkov, rešujejo abstrakcijo podatkovne baze, skrbijo za upravljanje seje in piškotkov. Vsebujejo tudi funkcije za pošiljanje e- pošte, za pomikanje po straneh
* **Varnost** - PHP sam po sebi že vsebuje funkcije, ki omogočajo filtriranje vhodnih in izhodnih podatkov. Se pa zgodi, da mestoma pozabimo uporabljati to filtriranje podatkov, zato je lahko spletna aplikacija ranljiva. Za filtriranje podatkov porabimo tudi nekaj časa. Ogrodja PHP že sama po sebi rešujejo te probleme.
* **Manj kode, hitrejše programiranje** - ogrodja PHP že po namestitvi vsebujejo večino knjižnic, ki jih bomo uporabili. Zato ne bo potrebno vedno znova iskati knjižnice od drugih. To pomeni, da bomo napisali manj kode in s tem prihranili ogromno časa. Seveda obstaja za vsako ogrodje druga krivulja učenja, ampak večinoma bomo uživali v hitrem razvoju aplikacij.
* **Razširjena skupnost za podporo** - Za vsa priljubljena PHP ogrodja obstaja zelo razširjena aktivna skupnost (angl. communities). Lahko se pogovarjamo z drugimi razvijalci, dobimo pomoč in povratne informacije, lahko pa pomoč in izkušnje tudi sami nudimo drugim. Obstajajo pa tudi številne knjige, portali z raznimi vodiči, lahko se prijavimo na mailing liste. Ogromno znanja in nasvetov lahko dobimo že z brskanjem po forumih in prebiranjem tem.
* **Večje zaposlitvene možnosti** - Verjetno smo v zaposlitvenem oglasu za programerja že zasledili, da je pogoj za zaposlitev znanje katerega izmed ogrodij. Če smo poznavalec katerega izmed ogrodij, imamo boljše možnosti za zaposlitev na delovnem mestu spletnega razvijalca.
* **Primerno za timsko delo** - Če imamo projekt s pomočjo ogrodja lepo strukturiran in organiziran, je mogoče z delom nadaljevati v skupini. Tako lahko oblikovalec dela samo na pogledih (views), specialist za podatkovne baze na modelih (models), glavni programer pa razvija knjižnice in vtičnike. Prav tako lahko nekdo čisto neodvisno razvija module za testiranje.

Seveda ogrodje predstavlja samo neko osnovo, ki jo lahko po želji nadgradimo in prilagodimo. Vsekakor je ogrodje prava rešitev za vse manj izkušane programerje, saj je že v osnovi poskrbljeno za varnost in stabilnost sistema. Velika prednost je tudi v tem, da če v ekipi dela več programerjev in ti poznajo določeno ogrodje, se vsi držijo določenih načel; zato sta tudi razumevanje in preglednost kode dosti boljši.

# Arhitektura MVC

**Model-view-controller (MVC)** je programska arhitektura, ki se uporablja pri razvoju programske opreme. V bistvu gre za arhitekturni vzorec, ki ločuje domensko oz. poslovno logiko od uporabniškega vmesnika (pogleda), kar omogoča neodvisen razvoj, lažje testiranje in kasnejše vzdrževanje oz. nadgradnjo programske opreme.

Kot že samo ime pove, je MVC vzorec sestavljen iz treh segmentov:

* **Model** (angl. Model) - model je del, ki vsebuje poslovno logiko, torej del, ki vsebuje neke operacije ali transformacije nad podatki. Prav tako model upravlja vedenje aplikacije, odgovarja na zahtevke za informacije o stanju in se odziva na navodila za spremembo stanj.
* **Pogled** (angl. View) - pogled predstavlja različne načine predstavitve podatkov, ki jih prejme od modela. Ponavadi gre za predloge, ki so napojeni s podatki. Ker ponavadi gre za več različnih pogledov, more kontroler določiti, katerega uporabiti.
* **Krmilnik** (angl. Controller) - ta skrbi za upravljanje zahtev, ki prihajajo od uporabnika (prejete kot HTTP GET ali POST zahtevke, ko uporabnik klikne na element v uporabniškem vmesniku). Glavna naloga krmilnika je, da pokliče in usklajuje vse objekte in vire, ki so potrebni za izvedbo uporabnikovega zahtevka.

Ponavadi krmilnik zahteva ustrezen model in izbere ustrezen pogled za prikaz.

Na sliki je prikazano, kako se obnaša MVC vzorec, ko uporabnik zaprosi za določeno zahtevo. Za lažjo predstavitev si oglejmo dejanski primer. Ogledamo si spletno trgovino s knjigami. Uporabnik ima na voljo različne opcije: ogled seznama knjig po kategorijah, ogled podrobnosti o knjigah, dodajanje knjig v košarico, registracijo v spletni trgovini

Kaj se zgodi, ko uporabnik izbere kategorijo fantazijski romani? Po vsej verjetnosti imamo krmilnik z imenom »books\_controller«, ki skrbi za upravljanje vseh aktivnosti, povezanih s knjigami (pregled seznama knjig, prikaz podrobnosti ). Prav tako imamo model z imenom »books\_model«, ki skrbi za podatke in logične povezave med elementi spletne trgovine. Na koncu še imamo pogled, ki skrbi za predstavitev: npr. za prikaz seznama knjig (view\_books\_list).

Zdaj pa si oglejmo, kako navedeni primer deluje v praksi. Krmilnik (books\_controller) prejme uporabniško zahtevo preko HTTP GET ali POST zahtevka. Krmilnik preuči zahtevo in parametre, kliče model (books\_model) in od njega zahteva seznam vseh fantazijskih romanov. Model je odgovoren za pridobivanje informacij iz podatkovne baze in za upoštevanje filtrov ali logike (če je potrebno). Vrne podatke, ki predstavljajo seznam knjig. Krmilnik na koncu poskrbi za predstavitev pridobljenih podatkov na uporabniku prijazen način.

# Krmilniki

Naloga krmilnika je, da prevzame zahtevo HTTP, pokliče določene modele in vse želene podatke združi v pogledu. Zdaj pa si še oglejmo, kako krmilnik izgleda v praksi.

Oglejmo si welcome.php krmilnik, ki se nahaja v osnovni verziji CI.

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Welcome extends CI\_Controller {

public function index()

{

$this->load->view('welcome\_message');

}

}

/\* End of file welcome.php \*/

/\* Location: ./application/controllers/welcome.php \*/

Edina dostopna točka v aplikacijo je index.php. Krmilnik je razred, do katerega lahko dostopamo preko URI-ja. Do zgornjega krmilnika bi dostopali na naslednji način:

<http://ime_domene/index.php/welcome/index>

Prvi segment za imenom index.php predstavlja ime krmilnika (v našem primeru welcome), drugi segment predstavlja ime metode znotraj krmilnika (index), vsi nadaljnji segmenti pa predstavljajo atribute metode. Ime metode index je prizveto ime metode; to ime lahko preprosto izpustimo. Lahko pa v naslovu URL odstranimo tudi ime index.php, in sicer tako, da to pravilo zapišemo v datoteko .htaccess. Torej lahko ta krmilnik pokličemo tudi na krajši način: <http://ime_domene/welcome>

Če bi odstranili še ime krmilnika, potem bi imeli samo ime domene, bi nas sistem preusmeril na privzeti krmilnik. Ime privzetega krmilnika nastavimo vmapi application/config in sicer v datoteki routes.php. Nastavimo na sledeči način:

$route['default\_controller'] = "welcome";

Če bi sedaj vpisali samo ime domene, bi nas sistem takoj preusmeril na krmilnik welcome in metodo index.

Če si še enkrat podrobneje ogledamo primer krmilnika, vidimo, da razred Welcome deduje od razreda CI\_Controller. Pomeni, da vsebuje krmilnik vse lastnosti, kot ga ima krmilnik CI\_Controller. Ta pa je osnovni krmilnik v CI in vsebuje vse funkcionalnosti, ki jih omogočajo krmilniki v CI. Seveda bi lahko krmilnik dedoval tudi od kakšnega drugega razreda. Recimo, če imamo prijavo v sistem, kjer moramo za vsak krmilnik preverjati, če je uporabnik prijavljen v sistem, je vsekakor najenostavnejše, da vsi krmilniki, ki so vezani na prijavo v sistem, dedujejo od nekega dodatnega razreda (recimo My\_Controller), ta krmilnik pa deluje od CI krmilnika.

# Pogledi

Pogled je preprosta spletna stran ali del spletne strani; recimo glava, noga, stranska navigacija. V bistvu lahko poglede poljubno vstavljamo znotraj drugih pogledov, če želimo imeti poljubno hierarhijo.

Pogleda ne moremo nikoli direktno poklicati, ampak mora biti naložen preko krmilnika. Pa si oglejmo primer pogleda in postopek, kako ga naložimo v krmilniku:

Pogled, shranjen v datoteki pozdravljen\_svet.php:

<html>

<head>

<title>My Blog</title>

</head>

<body>

<h1>Pozdravljen svet</h1>

</body>

</html>

Primer krmilnika, v katerem naložimo pogled:

<?php

class Pozdravljen\_svet extends CI\_Controller {

function index()

{

$this->load->view('pozdravljen\_svet');

}

}

?>

Pogled preprosto naložimo tako, da v metodo vpišemo ime datoteke pogleda, ki se nahaja v mapi views, s tem da izpustimo pripono .php. Ponavadi so tudi pogledi dinamično zgrajeni, torej jih ustvarjamo na podlagi pridobljenih podatkov, recimo iz podatkovne baze. Zato moramo podatke, pridobljene iz modela, v krmilniku posredovati v pogled.

Oglejmo si primer za novico:

<?php

class Novica extends CI\_Controller {

function index()

{

$data['ime'] = 'Ime naše novice';

$data['vsebina\_novice'] = 'Vsebina naše novice';

$this->load->view('novica');

}

}

?>

Še primer pogleda novica.php

<html>

<head>

<title><?php echo $ime;?></title>

</head>

<body>

<h1><?php echo $ime;?></h1>

<p><?php echo $vsebina;?></h1>

</body>

</html>

Vse dinamične podatke, ki jih želimo prikazati v pogledu, preprosto shranimo v polje in jih posredujemo pogledu. Elementi polja pa v pogledu postanejo spremenljivke, tako da jih lahko preprosto izpišemo.

Kot smo že omenili, lahko znotraj enega krmilnika uporabimo več pogledov, lahko pa posamezni pogled ugnezdimo v drug pogled.

Primer:

<?php

class Stran extends CI\_Controller {

function index()

{

//posamezni deli strani

$data['glava'] = $this->load->view('glava','',TRUE);

$data['meni'] = $this->load->view('meni','',TRUE); $data['noga'] = $this->load->view('noga','',TRUE);

$data['vsebina'] = $this->load->view('vsebina','',TRUE);

//glavna predloga stani

$this->load->view('stran', $data);

}

}

?>

Pridobimo poglede in jih shranimo v skupno spremenljivko, to spremenljivko pa potem posredujemo glavnemu pogledu, kjer izpišemo posamezne dele strani. Ko shranjujemo poglede v spremenljivko, moremo paziti, da tretji atribut metode view damo na TRUE.

# Modeli

Modeli so PHP razredi, ki jih uporabljamo za delo podatki, ponavadi v povezavi s podatkovno bazo. Recimo, ko potrebujemo podatke, ki so zapisani v podatkovni bazi, pokličemo model, v njem naredimo poizvedbe in pridobimo podatke. Podobno kot pri pogledih, tudi modela ne moremo klicati direktno, ampak vedno preko krmilnika. Modeli se nahajajo v mapi application/models.

Primer modela:

class Ime\_modela extends CI\_Model {

function \_\_construct()

{

parent::\_\_construct();

}

function ime\_metode()

{

$query = $this->db->get('novice');

return $query->result();

}

}

Pri imenu modela moramo paziti, da prvo črko imena zapišemo z veliko začetnico, ostali del pa mora biti obvezno zapisan z malimi črkami. Pri imenu datoteke modela pa lahko vse zapišemo z malimi črkami. Za primer bi datoteko poimenovali tako:

**application/models/ime\_modela.php**

Kot smo že omenili, model moramo najprej naložiti v krmilniku, šele nato ga uporabimo. To naredimo tako:

**$this->load->model('Ime\_modela');**

Metodo v modelu pa potem pokličemo na naslednji način:

**$this->Ime\_modela->ime\_metode();**

Modela v CI lahko tudi samodejno naložimo, tako da ni potrebno v vsakem krmilniku nalagati vedno isti model. To nam je v pomoč predvsem takrat, ko isto model uporabljamo v večini krmilnikov. Modele, ki jih želimo samodejno naložiti, določimo v nastavitvah, in sicer v mapi application/config, v datoteki autoload.php.

# Podatkovna baza

Danes si ne predstavljamo spletne aplikacije brez podatkovne baze. CI precej olajša delo s podatkovno bazo. Omogoča povezavo na različne tipe podatkovnih baz (MySQL, MySQLi, MS SQL, Postgres, Oracle, SQLite, and ODBC).

Povezavo s podatkovno bazo vzpostavimo tako, da uredimo datoteko database.php, ki se nahaja v mapi application/config/, še prej pa moramo dodati samodejno nalaganje razreda knjižnice database v konfiguracijski datoteki autoload.php, ki se prav tako nahaja v mapi application/config/.

Primer datoteke database.php:

$active\_group = 'default';

$active\_record = TRUE;

$db['default']['hostname'] = 'localhost';

$db['default']['username'] = 'root';

$db['default']['password'] = '';

$db['default']['database'] = 'codeigniter';

$db['default']['dbdriver'] = 'mysql';

$db['default']['dbprefix'] = '';

$db['default']['pconnect'] = TRUE;

$db['default']['db\_debug'] = TRUE;

$db['default']['cache\_on'] = FALSE;

$db['default']['cachedir'] = ''; $db['default']['char\_set'] = 'utf8';

$db['default']['dbcollat'] = 'utf8\_general\_ci';

$db['default']['swap\_pre'] = '';

$db['default']['autoinit'] = TRUE;

$db['default']['stricton'] = FALSE;

Gre za dvodimenzialno polje, tako da lahko nastavimo več povezav do podatkovnih baz, recimo eno za lokalni strežnik in drugo za produkcijski strežnik. Katere nastavitve oz. katero povezavo s podatkovno bazo bomo uporabili, določimo na naslednji način:

$active\_group = 'default';

Razred database v CI vsebuje ogromno funkcij, ki zelo olajšajo delo s podatkovno bazo. Zagotovo najbolj odstopa razred Active Record. CI uporablja modificirano verzijo vzorca Active Record, ki omogoča pridobivanje, vstavljanje in posodabljanje v podatkovni bazi na preprostejši način, z minimalno kodo. Namesto, da bi pisali dolge select, insert in update stavke, lahko ta problem rešimo v nekaj vrsticah. Primer poizvedbe z Active Record:

$query = $this->db->get('mytable', 10, 20);

//Poizvedba brez Active Record: SELECT \* FROM mytable LIMIT 20, 10

Uporabo razreda Active Record vključimo na naslednji način (v datoteki database.php):

**$active\_record = TRUE;**

Pomanjkljivost CI pred nekaterimi PHP ogrodji pa je, da privzeto ne omogoča koncepta ORM. To je koncept, ki razvijalcem v objektno orientiranih okoljih omogoča, da v svojih aplikacijah upravljajo s podatki na povsem domač način - kar preko objektov, kot da bi delali z objektno podatkovno bazo in ne z relacijsko. To omogoča abstrakcijski sloj ORM, ki poskrbi za brezšivno in transparentno preslikovanje iz okolja objektov v relacijsko podatkovno bazo in nazaj.

# Razredi in funkcije

CI vsebuje ogromno število razredov in funkcij, ki olajšajo in skrajšajo čas razvoja spletnih aplikacij. Izpostavili bomo najbolj pogosto uporabljane razrede, ki jih bomo tudi mi uporabljali v razvoju aplikacije. Razred Email je razred, ki vsebuje robustne metode, ki omogočajo pošiljanje e- pošte. Razred Email omogoča tudi podporo za različne protokole (mail, smtp, sendmail), podporo za več prejemnikov, možnost pošiljanja golega ali HTML besedila, pripenjaje priponk, nastavitev prioritet:

* **Razred Email** uporabimo najlažje tako, da nastavimo samodejno nalaganje razreda (autoload) in v konfiguracijski datoteki (config.php) določimo nastavitve za povezavo na strežnik ter naredimo inicializacijo teh lastnosti. To pomeni, da bomo vedno, ko bomo želeli pošiljati e-pošto, poklicali samo metodo za pošiljanje, določili prejemnike, pošiljatelja, zadevo in vsebino sporočila.
* **Razred Encryption** je razred, ki omogoča dvostanstransko enkripcijo podatkov. Enkripcija poteka na podlagi zasebnega ključa, ki ga določimo v konfiguracijski datoteki (config.php). Omogoča uporabo različnih enkripciskih algoritmov. Oglejmo si primer, kjer dvostranska enkripcija podatkov pride v poštev. Uporabili ga bomo tudi v aplikaciji. Takšen primer je preklic rezervacije s strani stranke. Oglejmo si primer podrobneje: Stranka rezervira termin in v povratnem e-sporočilu prejme povezavo, do preklica rezervacije. Seveda mora ta povezava vsebovati tudi identifikator rezervacije, ki jo želimo odjaviti. Ta identifikator je zaporedna številka. Kaj hitro bi se lahko zgodilo, da bi nekdo želel škodovati, pa bi preprosto spremenil zaporedno številko in preklical rezervacijo nekoga drugega. Če pa identifikator kriptiramo, pa nekdo spremeni nekaj znakov v nizu kriptiranega besedila, je skoraj nemogoče, da bi ustvaril celoštevilčno vrednost; dobil bi nek čuden nabor znakov.
* **Razred Form Validation** je razred, ki ga uporabimo pri validiranju spletnih obrazcev. To omogoča na zelo preprost način in z malo napisane programske kode. Spletno aplikacijo si težko predstavljamo brez spletnega obrazca. Seveda pa lahko znotraj obrazca uporabniki marsikaj spreminjajo. Zato je potrebno vnosni obrazec zaščititi do te mere, da uporabniku dovolimo vpis samo takšnih podatkov, kot jih sistem pričakuje. Torej šele po tem, ko uporabnik pravilno vpiše zahtevane podatke, se izvede neka operacija (ponavadi je to zapis vnesenih podatkov v podatkovno bazo). Mi bomo v aplikaciji imeli registracijski vnosni obrazec, kjer se bodo registrirali novi uporabniki. Imeli bomo vnosna polja: ime in priimek, ime podjetja, e-naslov, geslo in ponovi geslo. Z razredom Form Validation bomo preverili, ali so uporabniki izpolnili vsa vnosna polja, ob tem pa jih bomo omejili na maksimalno število znakov. Za vpisan e-naslov bomo preverili, ali ustreza sintaksi, ki je značilna za e-naslov, in če ta e-naslov v bazi uporabnikov še ne obstaja. Za geslo in ponovi geslo pa bomo preverili, če se ujemata med seboj in če je vpisanih vsaj 5 znakov.
* **Razred Input** je razred, ki ga uporabljamo v dva namena. Skrbi za predhodno obdelavo globalnih vhodnih podatkov zaradi varnosti in vsebuje funkcije za pridobivanje vhodnih podatkov. Vhodne podatke v spletnih aplikacijah lahko pridobivamo na različne načine, in sicer POST, GET, COOKIE in SERVER; prav te funkcije vsebuje ta razred. Če vpišemo podatke v vnosni obrazec in ga potrdimo, se ponavadi izvede metoda post, ki pridobi podatke iz vnosnih polj. Seveda pa moramo tudi preprečiti napade XSS; ravno ta razred prepreči to vrsto napadov.
* **Razred Pagination** je razred, ki ga uporabimo za oštevilčevanje strani in pomikanje po njih; npr. če imamo v podatkovni bazi shranjeno 10000 uporabnikov in želimo prikazati ta seznam v spletni aplikaciji. Vsekakor ni priporočljivo tega seznama prikazati naenkrat, ampak je smiselno uporabiti oštevilčene strani in na eni strani prikazati npr. 50 uporabnikov. Ravno ta razred rešuje ta problem na zelo enostaven in popolnoma prilagodljiv način.
* **Razred Session** je razred, ki nam dovoljuje, da ohranjamo uporabnikovo stanje in spremljamo aktivnosti, medtem ko uporablja našo spletno aplikacijo. Razred shranjuje podatke za vsakega uporabnika posebej v piškotek, ki je shranjen na vsakem lokalnim računalniku. Ti podatki so ponavadi šifrirani. Prav tako imamo možnost, da se podatki shranjujejo direktno v podatkovno bazo. Na primer: Ko se uporabnik uspešno prijavi v spletno aplikacijo (uspešno vpiše uporabniško ime in geslo), se ustvari seja (angl. Session), ki hrani podatke, na podlagi katerih aplikacija ve, če je se je uporabnik uspešno prijavil, kakšen je njegov id in kakšni so njegovi drugi podatki, ki jih zapišemo v sejo.
* **Razred Unit Testing** je razred, ki omogoča testiranje vsake funkcije v aplikaciji. V MVC pristopu je predvsem smiselno testirati modele, kjer lahko točno predvidimo, kaj lahko posamezna metoda v modelu vrača. Razred Unit Testing vsebuje tudi metodo, ki omogoča samodejno kreiranje poročila.

# Dodatne funkcionalnosti

Opisali bomo še nekaj funkcionalnosti, za katere menimo, da so pomembne in vsekakor zelo uporabne pri razvoju spletne aplikacije.

* **Hooks** - Ta funkcionalnost omogoča, da spremenimo potek delovanja ogrodja brez poseganja v jedro ogrodja. Če se CI izvaja normalno, sledi procesu, ki ga prikazuje Slika v prejšnjem poglavju. Včasih pa si želimo, da bi na določeni točki izvedli določene operacije, zato ta funkcionalnost lahko v določenih situacijah zelo pomaga; recimo, če želimo izvesti skripto preden se naložijo krmilniki. Na voljo so različne točke, ki omogočajo spremembo normalnega izvajanja CI, in sicer: pred izvajanjem sistema, pred izvajanjem krmilnikov, po izvedbi konstruktorjev v krmilniku, po izvedbi krmilnikov in po izvedbi sistema. Na voljo sta pa tudi točki, ki omogočata »prebris« funkcije za izpis strani v brskalniku in »prebris« funkcije za prikaz strani v predpomnilniku.
* **Routing** - V MVC pristopu nadzor nad izvajanjem vedno prevzamejo krmilniki glede na zahtevani URL. Torej CI vedno sledi vzorcu ime\_domene/razred/metoda/atribut1/atribut2 (razred je ime krmilnika). Velikokrat se zgodi, da želimo imeti ta URL zapisan v lepši obliki (bolj prijazni za uporabnika in tudi za iskalen robote). Ravno ta funkcionalnost omogoča prijaznejšo uporabo URL naslovov.
* **Caching** - Ta funkcionalnost omogoča predpomljenje strani. Velikokrat se zgodi, da spletni strežnik po nepotrebnem obremenjujemo. S tem tudi povečamo čas, da se uporabniku izriše spletna stran v celoti. Če ne uporabljamo predpomljenja, na spletni strani pa pride do novega zahtevka, mora spletni strežnik obdelati vse operacije, ki jih potrebujemo za prikaz strani. Če pa uporabljamo predpomljnenje, se ob prvi zahtevi celotna izvorna koda strani shrani v datoteko HTML, ki se ob ponovnem zahtevku prikaže uporabniku. Strežniku torej ni potrebno izvajati nobenih dodatnih operacij in tudi poizvedb ne, ampak preprosto prikaže stran, kot bi bila statična. Obstaja ogromno strani, ki se osvežujejo enkrat mesečno, tako da lahko predpomljenje strežnik precej razbremeni. V CI določimo čas v minutah, na podlagi katerega določimo, kdaj se izvede novo predpomljenje strani.
* **Profiling** - Razred, ki omogoča prikaz podrobnosti, ki se izvajajo v ozadju in omogočajo hitrejšo odpravo napak in večjo učinkovitost. Omogoča prikaz časa izvajanja določenih operacij, seznam vseh poizvedb in trajanje le-teh, informacije o uporabljenih krmilnikih, konfiguracijske spremenljivke, podatke posredovane preko GET in POST metod
* **Error in logs** - CI omogoča hitro grajenje poročil napak znotraj aplikacije. To pomeni, da se za vsak dan posebej ustvari datoteka, v katero se zapišejo vse napake, ki so se zgodile v aplikaciji. Poleg napak lahko še zapisujemo obvestila, ki se izvajajo pri iskanju napak (angl. debug) in informacijskih sporočil. Poleg tega imamo na voljo tudi preprosto funkcijo, ki jo kličemo, da prikažemo napako znotraj spletne aplikacije v bolj prijaznejši obliki za uporabnika.

# Pregled in primerjava ostalih php ogrodij

Trenutno obstaja več kot 20 odprtokodnih PHP ogrodij. Mi se poleg CodeIgniter-ja osredotočamo na še na štiri ogrodja, in sicer na Zend Framework, CakePhp, Symfony in Yiiframework. Gre za največkrat iskana PHP ogrodja v letu 2012 preko iskalnika Google (podatki iz Google Trends). Na podlagi tega sklepamo, da gre tudi za najpopularnejša oz. največkrat uporabljena ogrodja. V nadaljevanju bomo na kratko opisali PHP ogrodja in naredili manjšo primerjavo ter predstavili dejstva, zakaj smo se odločili za izbrano ogrodje in na katere parametre smo se osredotočili.

## Zend Framework

Razvija ga podjetje Zend, ki je sodelovalo tudi pri razvoju motorja (angl. Engine) programskega jezika PHP, zato vsekakor velja to ogrodje jemati resno. Zend velja za zelo robustno ogrodje, zraven pa ponujajo druge programske rešitve, kot je njihov lastni spletni strežnik, podpora in njihov lastni IDE (angl. Interface development editor). Kombinacija vseh teh orodij da izredno močno orodje za razvoj spletnih aplikacij . Zend je kaj precej podoben CodeIgniter-ju in CakePhp-ju, vendar njegova celotna dokumentacija temelji na oblikovnih vzorcih (angl. Desing patterns), ki pa so kar precej težki za razumevanje. Ogrodje ni priporočljivo za tistega, ki začenja z razvoje spletnih aplikacij. Pri iskanju zaposlitve je sigurno najboljša referenca Zend, kajti če ga hočete razumeti, morete poznati dosti lastnosti.

## CakePhp

To ogrodje se zgleduje po konceptih ogrodja Ruby on Rails. Struktura ogrodja je preprosta za razumevanje, vsebuje mape Controllers, Models in Views. Najmočnejši del tega ogrodja je »pekarna« (angl. Bakery), kjer lahko zunanji razvijalci naložijo svojo kodo in jo delijo preko skupnosti. CakePhp je za razumevanje precej preprost, sicer ne tako kot CodeIgniter, ampak ni tako kompleksen kot Zend in Symfony. Slabost tega ogrodja je, da nima samodejnega nalaganja dodatkov in podpore za predloge (angl. Template engine). Ni primerno za bolj zahtevne uporabnike, dokumentacija je precej komplicirana. V ozadju ni nobenega podjetja, torej je last avtorjev in je licencirana z licenco MIT (Massachusetts Institute of Technology).

## Symfony

To ogrodje sponzorira francosko podjetje SensioLabs in je licencirano z licenco MIT. Največje prednosti tega ogrodja so, da spodbuja uporabo ORM (object-relational mapping), kar je velika prednost pri delu z večjimi podatkovnimi bazami, in uporablja svoj lastni pogon za kreiranje predlog. Je kar težek za razumevanje, ampak ima zelo dobro skupnost, kjer hitro najdemo pravo pomoč. Na voljo je tudi ogromno vodičev in knjig. Struktura modela se od ostalih ogrodij precej razlikuje, saj spodbuja uporabo Doctrine knjižnic, ki je v bistvu vstopna točka do poslovne logike modela. Uporablja ukazno vrstico za generiranje, s pomočjo katere lahko hitro generiramo kodo. To je zagotovo dobrodošlo pri vzpostaviti novih projektov. Ni pa jasno, zakaj to ogrodje (v primerjavi z ostalimi) uporablja drugo strukturo.

## Yii framework

Yii ogrodje (Yes It Is) je najmlajše od predstavljenih ogrodij, saj njegov razvoj sega v začetek leta 2008. Trenutno velja za enega izmed najboljših in najbolj sposobnih ogrodij, saj rešuje nekatere probleme, ki jih druga orodja ne. Če ste novi razvijalec, je to ogrodje zagotovo za vas zelo primerno, saj ne obremenjuje z nepotrebnimi stvarmi in vam prihrani ogromno časa. Njegovi največji prednosti sta hitrost, saj je najhitrejši od predstavljenih ogrodij, in samodejno generiranje osnovne kode. Čeprav ima ogromno prednosti, ima tudi nekaj slabosti. To je pa predvsem posledica tega, da gre za precej novo ogrodje. Na voljo ni toliko knjig in vodičev, zato je razumevanje tega ogrodja precej zahtevno. Tudi dokumentacija je precej obširna in zahtevna.

To ogrodje vsebuje vse osnovne funkcionalnosti ogrodij (MVC arhitektura, abstrakcija podatkovnih baz ), poleg tega pa ponuja še ogromno dodatnih funkcionalnosti: podporo spletnim storitvam (Web services), dobro podporo za internacionalizacijo in lokalizacijo, različne načine predpomljenja, samodejno generiranje kode in generiranje kode preko ukazne vrstice, dobro podporo za testiranje, pa še bi lahko naštevali.

Vsekakor gre za zelo sposobno ogrodje, ki vsebuje ogromno funkcionalnosti, zato ni primerno za majhne in manj zahteven projekte.

## Kako narediti spletno stran, del 1

### Korak 1: Pomislite na Privlačen ime

To, V vseh odkritosti, Najtežji del, in najdaljši del (Žal!). Vaša spletna stran ime, znan tudi kot vaš ime domene, opredeljuje vaša spletna stran vedno. Brez pritiska.

(Tehnična opomba: ko sem rekel, ime spletne strani, Jaz vključujejo. Com ali karkoli [v tehniki znane kot ccTLD] kot tudi dejansko ime, e.g. hathix.com).

Prva, bo ožji seznam imen. Vam bom dal nekaj nasvetov, potem lahko greš naprej in da bo vaše ožji izbor.

Sem ugotovila, da najboljši imena spletnih mest, so kratke in so izdelani iz dveh zlogov, s poudarkom na prvi. Ne verjamete? Vprašajte Google, Apple, Twitter, Facebook, Yahoo, Tumblr, Foursquare, YouTube, ali Pinterest (če se izgovarja kot 2 zlogov). Seveda, imena ste izbrali morate biti izgovorljivo in enostavno pojasniti, ko jih slišiš.

Zdaj, če delaš spletno stran za obstoječega podjetja ali skupine, Vaše delo je precej enostavno. Samo pomislite na nekaj načinov, da dajo svoje ime skupaj. Na primer, Pennsylvania (PA) Slon Lovers’ Klub lahko uporabite pennsylvaniaelephantloversclub.com, elephantloverspa.com, elephantloversclub.com, paelc.com, in tako naprej. Just izognili preveč kratic (kot paelc.s) ker oni so težje, da se spomnimo in težje povezati s svojo skupino. Vendar, Ne bo vaše ime predolgo bodisi (kot prvo ime). Karkoli narediš, ne dajo Spletna stran ali na spletu Vaše ime – to zveni neprofesionalno.

Če delaš spletno stran za app, novo podjetje, ali si, postane malo težje, saj boste morali začeti iz nič. Dober vir imen je svoje ime (zagotavlja, da je dovolj redka in izgovorljivo), občasni beseda (twitter pomeni ptičji vrišč), ali nekaj kratkega in kreativno (hathix prihaja iz hindujščini Hathi, kar pomeni, slona, plus x za polje; ne sprašujte o celotnem slona v škatli stvar). Če vse drugo odpove, samo povem, kaj se bo vaša spletna stran je približno. Če ste Cole in ste blogging o tehnologiji, colestechnologyblog.com je preprosta in enostavna rešitev.

Priporočam uporabo. Com saj to je najbolj strokovno in nepozabno, ampak, če si res želite, da vas lahko poskusite. org, .net, ali .[Oznaka države] – podobno. ca, .od, .co.uk, itd. V ZDA, karkoli drugega. com ali. org izgleda slabo, ampak tudi v drugih državah, si verjetno lahko uporabite kodo države.

Torej,, vaš izziv je, da narediti ožji izbor približno 10 Imena. Bodite kreativni in gredo vsem spektru. Zapomnite si,: Skratka, preprost, nepozabno. Pisati ta seznam navzdol na vašem telefonu, papirja ali če ste občutek Hipster. Uvrstitev teh imen iz favorita za najmanj priljubljena.

A tutorial: using a website to find if a website name isn't taken

Vnesite imena na vaši ožji izbor v polju v zgornjem desnem kotu te strani. Kliknite / pipe sliko, da gredo na spletno stran.

Zdaj, moramo vidim, če so dejansko na voljo vaše spletne strani imena. Vnesite vsako od vaših imen “Vnesite domeno” polje na vrhu te strani. Če je vaše ime na voljo, boste dobili zeleno preverite. Če ne, boste dobili predloge – ampak jih ignorirati, ker oni grozno. (Druge spletne strani, ki jih lahko poskusite, so FatCow in GoDaddy – si ne kupujejo od njih, samo z uporabo svoje orodje.)

Če ste res všeč ime, in to ne deluje, poskusite swapping ven. Com. org ali druge tiste ki sem jih omenil zgoraj. Vendar bodite previdni, saj ti niso tako dobri.

OK, zato upajmo, da nekaj imen na seznam izbranih preživela. Izberite svojo najljubšo ime preživelega – da ti je ime spletne strani. Čestitke! Ste pravkar končali najtežji del oblikovanja spletne strani! Pojdi naprej in da majice, darila, pisalne potrebščine, in celotna krogla voska okrašen z vašim imenom spletne strani. Put “webmaster” na vaše socialne mreže profile. Očarati svoje prijatelje. In tako naprej.

### Korak 2: Najti gostitelja

Obstajajo 2 veliki deli za pridobivanje vaše spletne strani na spletu: nakup domene (To je kot pridobivanje pravice do zemlje) in pridobivanje gostovanje (to je kot pridobivanje gradbenega podjetja). Vsaka spletna stran ima svoj strežnik, in vsak webmaster vprašate vam bo povedal, da je njihova največja stvar, saj rezan kruh. Si prost za uporabo karkoli gostitelj vam je všeč, ampak jaz priporočam WebHostingHub (moj gostitelj; Všeč mi je), FatCow, in DreamHost (Slišal sem dobre stvari o teh zadnjih dveh).

Tako odpirajo tiste 3 spletne strani in poglej okoli njihove spletne strani. Iščete najboljši načrt, v smislu stroškov in funkcij. Preveril sem, in ti 3 gostitelji ponujajo precej vse, kar potrebujete za razumno ceno ($5-$10 na mesec). Oni so precej podobni, ampak tu je moja analiza:

* DreamHost izgleda dobro, vendar je njihova nadzorna plošča (kaj uporabljate za upravljanje vaše spletne strani) je po meri, tako deli tega priročnika ne bo delo za vas, če boste uporabili DreamHost.
* FatCow je nekoliko dražja (Prvo leto ni zastonj, za razliko od ostalih) ampak vse ostalo je videti brezhibno. Verjetno ima največ funkcij.
* WebHostingHub ponuja prvo leto brezplačno in je precej poceni, po tem. Vse ostalo je precej dobro, in njihove storitve za stranke, zlasti je fantastičen.

Screenshot image photo of FatCow, a web host and domain registrar

FatCow je odličen spletni strežnik (Gostitelji so fantje, ki vodijo vaše spletne strani na spletu.) Kliknite / pipe sliko, da obiščete njihovo spletno stran.

Res ne more iti narobe s katerim koli od njih. Samo izberi enega gostitelja, obiščite njihovo spletno stran, in kliknite velika “Da” ali “prijavite” Gumb (Ne morem dati povezave, ker so morali spremeniti veliko). Nasveti: V WebHostingHub, uporabite polje, ki pravi “registrirati novo domeno”, in FatCow potisnite “registrirati novo domeno” Gumb.

Vnesti podatke o kreditni kartici, , da račun, in tako naprej. Želel priporočamo nakup najboljšo možno paket – v primeru WebHostingHub, dobili Nitro – in pridobivanje večletni dogovor, ker so daljše pogodbe cenejši. (Za WebHostingHub, to je $ 7.99/year v času pisanja za 3-letni dogovor, vendar 9.99/year $ za 1-letno pogodbo.) Vendar to ni obvezno.

# Vprašanja za 2. izpitno enoto poklicne mature Strokovni predmet NPA

**POZOR: Nekatera vprašanja so delno spremenjena in dopolnjena!**

**Vprašanja *Visual C#* (4. letnik)**

1. **Uporabniški vmesnik razvojnega okolja *Visual C#***

Pomen posameznih oken uporabniškega vmesnika, urejevalnik programske kode, okno *Form Designer*. Označevalci v *Visual* C# (*C#*-ovi, uporabnikovi, poimenovalna pravila, poimenovalni dogovori, sestavljeni označevalci). Okna, ki jih uporabljamo pri izdelovanju projekta.

* pomen posameznih oken uporabniškega vmesnika 40
* označevalci v *C#*, urejevalnik programske kode, okno *Code Designer*  30
* okna, ki jih uporabljamo pri izdelovanju projekta, pomen okna Solution Explorer. 30

1. **Projekt v *Visual C#***

Moduli/datoteke, ki sestavljajo projekt v *Visual C#*, razlaga privzetega konstruktorja osnovnega obrazca. Obstoječi in dodatni imenski prostori. Dodajanje novih modulov in obrazcev v projekt. Lastnosti obrazca, parameter *this*. Pogled *Code/Design*.

* projektni modul, imenski prostor i, privzeti konstruktor osnovnega obrazca 40
* lastnosti obrazca, parameter *this*, pogled *Code/Design* 30
* dodajanje novih modulov in obrazcev v projekt. 30

1. **Osnovni postopki pri izdelavi projekta v *Visual C#***

Začenjanje novega projekta. Shranjevanje projekta. Prevajanje. Ustvarjanje in odstranjevanje odzivnih metod okenskih dogodkov. Postavljanje gradnikov na obrazec. Poimenovanje, preimenovanje in odstranjevanje gradnikov in obrazcev. Pomen datoteke Designer.cs.

* začenjanje novega projekta, shranjevanje projekta, prevajanje 40
* ustvarjanje in odstranjevanje odzivnih metod okenskih dogodkov 30
* poimenovanje, preimenovanje in odstranjevanje gradnikov . Datoteka Designer.cs. 30

1. **Osnovni gradniki v *Visual C#***

Osnovni gradniki (*TextBox*, *ListBox*, *ComboBox*, *Button*, ***CheckBox***, *RadioButton*, *GroupBox*, *Panel*, *TabControl*) in njihove glavne lastnosti in dogodki. Privzete nastavitve gradnikov. Ustvarjanje nove palete gradnikov in dodajanje novih gradnikov v obstoječo paleto.

* osnovni gradniki v *Visual C#* 30
* lastnosti in dogodki gradnikov, privzete nastavitve (dogodki) gradnikov 30
* ustvarjanje nove palete, dodajanje gradnikov v paleto. 40

1. **Programski modul brez lastnega obrazca v *C#*, pisanje lastnih metod.**

Pomen oz. namen programskega modula brez lastnega obrazca. Ustvarjanje modula brez lastnega obrazca. Pisanje lastnih metod v *Visual C#.*

* pomen oz. namen programskega modula brez lastnega obrazca 40
* ustvarjanje modula brez lastnega obrazca 30
* pisanje lastnih metod v *Visual C#*, klic lastnih metod. 30

1. **Projekt z več obrazci v C#, gradnik *LinkLabel*.**

Pomen glavnega obrazca in določanje glavnega obrazca projekta, v primeru, da je v projektu več obrazcev. Vključevanje že izdelanega obrazca v projekt in izključevanje obrazca iz projekta, gradnika *LinkLabel* in *ErrorProvider* - uporaba, primer.

* pomen glavnega obrazca, določanje glavnega obrazca projekta 40
* vključevanje že izdelanega obrazca v projekt in izključevanje obrazca iz projekta 30
* gradnika *LinkLabel* in *ErrorProvider*- uporaba, primer*.* 30

1. **Zgradba metod/funkcij v *Visual C#*.**

Lastne metode. Odzivne metode in razlaga oz. pomen obeh parametrov odzivnih metod, poimenovanje odzivnih metod, izbira odzivne metode v primeru, da ta že obstaja.

* lastne metode 40
* odzivne metode, 30
* odzivna parametra odzivnih metod, primer uporabe enega in drugega. 30

1. **Dostopnost označevalcev – spremenljivk in objektov v *Visual C#*.**

Dostop do spremenljivk, objektov in metod podrejenih obrazcev iz glavnega obrazca. Dostop do gradnikov glavnega obrazca iz drugih modulov in obrazcev. Pomen in uporaba (primer) odzivnih metod dogodkov *Load* in *Shown*.

* dostop do spremenljivk, objektov in metod podrejenih obrazcev iz glavnega obrazca 40
* pomen in uporaba odzivnih metod dogodkov *Load* in *Shown* 30
* dostop do gradnikov glavnega obrazca iz drugih modulov in obrazcev. 30

1. **Obrazec v *Visual C#*.**

Videz obrazca – lastnosti *FormBorderStyle* in *ControlBox*. Izdelava obrazca s poljubno obliko (nepravokotno). Preklapljanje med pogledom *Designer* in *Code*. Konstruktor obrazca. Glavne lastnosti obrazca, dogodki obrazca.

* videz obrazca – lastnosti *BorderStyle* in *ControlBox*, nepravokotni obrazci 40
* *DesignervView* in *Code view*, konstruktor obrazca 30
* lastnosti in dogodki obrazca. 30

1. **Modalno odprti obrazci v *Visual C#*.**

Pomen modalno odprtega obrazca in modalnih gumbov v Visual C#. Uporaba metode *ShowDialog* in testiranje vrnjenih vrednosti.

* pomen modalno odprtega obrazca in modalnih gumbov v *Visual C#* 40
* klic metode *ShowDialog* 30
* testiranje vrnjenih vrednosti. 30

1. **Meniji v C#.**

Izdelava glavnega menija in podmenijev, ločevalna črta v meniju, pomen znaka ***&*** v lastnosti *Caption*. Zlivanje menijev. Lebdeči meni.

* izdelava glavnega menija in podmenijev 40
* ločevalna črta v meniju, pomen znaka *&* v lastnosti *Text*, zlivanje menijev 30
* lebdeči meni. 30

1. **MDI – vmesnik z več dokumenti.**

Starševski in otroški obrazci v *Visual C#*. Odpiranje in zapiranje otroških obrazcev. Razporejanje odprtih obrazcev.

* starševski in otroški obrazci v *Visual C#* 40
* odpiranje in zapiranje otroških obrazcev, razporejanje odprtih obrazcev 30
* zapiranje in uničenje takih obrazcev. 30

1. **Pogovorna okna v *Visual C#*.**

Skupine pogovornih oken in njihov namen. Uporaba pogovornih oken.

* skupine pogovornih oken in njihov namen 30
* standardno pogovorna okna za delo z datotekami, imeniki, pisavo in barvami,… 30
* primer uporaba pogovornega okna za delo z datotekami 40

1. **Sporočilna okna v *Visual C#*.**

Osnovno sporočilno okno, odločitveno sporočilno okno, uporaba sporočilnega okna s tremi gumbi, dodatne oblike sporočilnih oken.

* osnovno sporočilno okno 30
* odločitveno sporočilno okno 30
* uporaba sporočilnega okna s tremi gumbi, dodatne oblike sporočilnih oken. 40

1. **Prekinitvene točke v *Visual C#*, razhroščevanje.**

Nastavljanje in pomen prekinitvenih točk v *Visual C#* in iskanje napak. Preverjanje trenutnih vrednosti spremenljivk med izvajanjem programa. Razhroščevanje. Razlika med opcijami *Run*, *Step Over* in *Step Into*.

* nastavljanje in pomen prekinitvenih točk, koračno izvajanje programa, iskanje napak 40
* spremljanje poteka programa in vrednosti spremenljivk med izvajanjem programa 30
* razhroščevanje (Visualizer, okno Watch, Immediate, Locals, Call Stack) 30

1. **Delo s tiskalnikom**

Dialogi in objekti za delo s tiskalnikom, pravilna uporaba dialogov, odzivna metoda dogodka *PrintPage*, tiskanje besedila, slik in likov, izdelava predogleda

* dialogi za nastavitev tiskalnika, uporaba dialogov, dogodek *PrintPage* 40
* tiskanje besedila, slik in likov 30
* izdelava predogleda 30

1. ***Garbage Collector***

Pomen *Garbage Collector*-ja, življenjska doba spremenljivk in objektov. Kako deluje *Garbage Collector*, upravljanje s pomnilnikom, uporaba *Using* stavka za kontrolo življenjske dobe nekaterih posebno pomembnih objektov.

* pomen, življenjska doba objektov 40
* kako deluje *Garbage Collector*, upravljanje s pomnilnikom 30
* *Using* stavek za kontrolo življenjske dobe nekaterih posebno pomembnih objektov 30

1. **Sporočilno okno *AboutBox*, lastnosti projekta**

Pomen in namen sporočilnega okna *AboutBox*. Dodajanje sporočilnega okna *AboutBox* v projekt in njegovo odpiranje. Nastavitve vrednosti, ki jih prikazuje sporočilno okno *AboutBox*.

* namen sporočilnega okna *AboutBox* 40
* dodajanje sporočilnega okna *AboutBox* v projekt in njegovo odpiranje 30
* nastavitve v *Assembly Information* in *Startup Object* 30

1. **Knjižnice razredov in datoteka z razredom v okolju *Visual C#***

Pomen knjižnic. Ustvarjanje nove knjižnice, prevajanje knjižnice. Ustvarjanje in uporaba datoteke z razredom.

* Pomen lastne knjižnice, ustvarjanje lastne knjižnice (**.dll**) 40
* Prevajanje in uporaba lastne knjižnice 30
* ustvarjanje in uporaba datoteke z razredom (**.cs**) 30

1. **Podatkovna skladišča**

Pomen podatkovnih skladišč. Ustvarjanje nove baze kar znotraj razvojnega okolja Visual C#. Prikaz in pomen oken *DataBase Explorer* in *Data Sources in* prikaz vsebine podatkovne tabele v gradniku *DataGridView*. Izdelava projekta nad obstoječo bazo podatkov.

* pomen podatkovnih skladišč, ustvarjanje nove *SQL Baze* in nove tabele v bazi 40
* prikaz in pomen oken *DataBase Explorer* in *Data Sources,* prikaz vsebine podatkovne tabele v gradniku *DataGridView* 30
* izdelava projekta nad obstoječo bazo podatkov – vključitev obstoječe baze v projekt 30

1. ***SqlConnection, Dataset*, *TableAdapter*, gradnika *BindingNavigatori* in *TableAdapterManager.***

Pomen objektov tipa *SqlConnection*, *Dataset*, *TableAdapter*. Uporaba gradnika *BindingNavigator* in gradnika *TableAdapterManager*.

* pomen objetov tipa *SqlConnection, DataSet* in *TableAdapter* 40
* vključitev gradnika *DataSet* in *TableAdapter*-jev v projekt s pomočjo čarovnika 30
* gradnik *BindingNavigator* in gradnik *TableAdapterManager* 30

1. **Programski dostop do baze**

Imenski prostor za dostop do objektov za delo za bazo pdatkov. Objekti, ki so potrebni za programski dostop do baze. Prikaz vsebine poljubne tabele v gradnikih tipa *DataGridView* in *ComboBox*. Objekt tipa *SqlCommand*.

* imenski prostor, objekti potrebni za programski dostop do baze 40
* prikaz podatkov v gradniku *DataGridView* in gradniku *ComboBox* (določanje podatkov za prikaz in določanje vrednosti, ki jo spustni seznam vrne) 30
* obdelava (poizvedba) tabele v bazi podatkov s pomočjo objekta tipa *SqlCommand* 30

1. **Transakcije**

Pomen in razlaga transakcij. Opis posameznih delov transakcije. Pomen in vloga parametrov v transakcijah.

* pomen in vloga transakcij 40
* opis transakcije in pomen parametrov v transakcijah 30
* pomen in vloga parametrov v transakcijah. 30

1. **Prikaz sporočil, namigov in brezpogojni varovalni blok v Visual C#.**

Programski namig – *ToolTip*, sporočilna vrstica – *StatusBar*, brezpogojni varovalni blok

* programski namig – *ToolTip* 40
* sporočilna vrstica – *StatusBar* 30
* brezpogojni varovalni blok 30

1. **Dogodki tipkovnice v Visual C#.**

Dogodki, ki jih proži tipkovnica. Vrstni red dogodkov. Kontrola uporabnikovega vnosa podatkov (npr. kako uporabniku preprečiti vnos vseh znakov razen cifer). Razlaga stavka ***e.Handled = true;***

* dogodki tipkovnice, vrstni red dogodkov 40
* vrstni red dogodkov 20
* kontrola uporabnikovega vnosa 40

1. **Dogodki ki jih proži miška v Visual C#, parameter *sender*.**

Najpomembnejših dogodki, ki jih proži miška. Podrobna razlaga najpomembnejših dogodkov miške. Parameter s*ender* – primer.

* najpomembnejših dogodki, ki jih proži miška 40
* podrobna razlaga najpomembnejših dogodkov miške 20
* parameter *sender* – primer 40

1. **Projekt z več obrazci, lastnost *DialogResult*, uničenje obraza. Sporočilno okno *MessageBox*.**

Projekt z več obrazci, modalni in nemodalni obrazci, vključevanje obrazcev v projekt, zapiranje obrazca, lastnost *DialogResult*, uničenje obraza. Sporočilno okno *MessageBox*.

* projekt z več obrazci, modalni in nemodalni obrazci 40
* lastnost *DialogResult* 20
* Sporočilno okno *MessageBox* 40

1. **Objekti tipa *DateTime* v C#, gradnika *Timer* in *DateTimePicker*.**

Kreiranje objekta *DateTime*, namen, uporaba, .

* kreiranje objekta *DateTime*, namen, uporaba datotek 40
* gradnik *Timer*, namen, uporaba 30
* gradnik *DateTimePicker*, namen, uporaba 30

1. **Razred *Object*, *Boxing* in *Unboxing*, prekrivne metode**

Razlaga razreda *Object* in pojmov *Boxing* in *Unboxing*. Razlaga prekrivnih metod

* razred *Object* 30
* *Boxing* in *Unboxing* 30
* razlaga prekrivnih metod 40

1. **Gradnik DataGridView**

Ustvarjanje stolpcev, vstavljanje podatkov, dostop do celic, obdelava podatkov v gradniku *DataGridView*

* ustvarjanje stolpcev, tipi, dostop do odzivne metode določenega stolpca 30
* vstavljanje podatkov, dostop do celic, 30
* obdelava podatkov v gradniku *DataGridView* 40

1. **Objektno programiranje**

Razloži pojem Objektno programiranje. Razred, objekt, instanca. Polja, objektne metode, privzeti konstruktor. Konstruktor glavnega obrazca v projektu in dostop do lastnosti glavnega obrazca.

* objektno programiranje, razred, objekt, instanca 30
* konstruktor glavnega obrazca v projektu in dostop do lastnosti glavnega obrazca 30
* polja, objektne metode, privzeti konstruktor 40

1. **Izdelava novega projekta v Visual C#.**

Izdelava novega projekta v *Visual C#*. Shranjevanje. Vključevanje že izdelanega obrazca v projekt. Izdelava datoteke z razredom in vključitev v projekt. Izdelava knjižnice.

* izdelava novega projekta v *Visual C#*. Shranjevanje 30
* vključevanje že izdelanega obrazca v projekt 30
* izdelava datoteke z razredom in vključitev v projekt. Izdelava knjižnice. 40

1. **Postopki pri izdelavi projekta.**

Spreminjanje imen obrazca, gradnikov, metod. Odstanjevanje odzivnih metod gradnikov. Datoteka *.Designer.cs*. Izključevanje obrazca iz projekta in brisanje obrazca.

* spreminjanje imen obrazca, gradnikov, metod 30
* odstanjevanje odzivnih metod gradnikov. Datoteka *.Designer.cs* 30
* izključevanje obrazca iz projekta in brisanje obrazca. 40

1. **Prevajanje projekta in odprava napak.**

Prevajanje in odpravljanje sintaktičnih napak. Odpravljanje semantičnih napak. Poravnavanje gradnikov na obrazcu – okno *Layout*

* prevajanje in odpravljanje sintaktičnih napak. 30
* odpravljanje semantičnih napak, uporaba razhroščevalnika 30
* poravnavanje gradnikov na obrazcu – okno *Layout* 40

1. **Dinamično ustvarjanje okenskih gradnikov.**

* Dinamično ustvarjanje obrazcev. Dinamično ustvarjanje okenskih gradnikov in postavljanje takih objektov na obrazce. Dinamično ustvarjanje odzivnih metod okenskih dogodkov.
* dinamično ustvarjanje okenskih gradnikov in postavljanje na obrazce 40
* dinamično ustvarjanje obrazca 30
* dinamično ustvarjanje odzivnih metod okenskih dogodkov 30

1. **Dedovanje vizuelnih gradnikov.**

Pojem dedovanja. Dedovanje lastnega in poljubnega obrazca. Dedovanje poljubnega gradnika, ter njegova nadgradnja. Izdelava novega uporabniškega gradnika, dedovanje lastnega gradnika, vključitev na paleto gradnikov ter uporaba.

* pojem dedovanja, dedovanje lastnega obrazca 40
* dedovanje poljubnega gradnika, ter nadgradnja 30
* Izdelava novega gradnika in uporaba le-tega. 30

1

**Zanke v C#**

Vrste zank v C#. Testiranje pogoja za nadaljevanje in izstop iz zanke. Razlike med posameznimi zankami.

* vrste zank
* testiranje pogoja
* razlike med zankami

**Modalno odprti obrazci v Visual C#**

Pomen modalno odprtega obrazca v Visual C#. Klic metode *ShowDialog* in testiranje vrnjenih vrednosti.

* pomen modalno odprtega obrazca v Visual C#
* klic metode *ShowDialog*
* testiranje vrnjenih vrednosti

2

**Podatkovni tip *string* v C#**

Ustvarjanje spremenljivk tipa niz, pristopanje k posameznemu znaku v nizu, osnovne lastnosti in metode za delo z nizi: dolžina niza, vstavljanje in odstranjevanje znakov, iskanje podniza v nizu, velike in male črke, spajanje nizov, metoda Split.

* deklaracija in dolžina niza
* pristop k posameznemu znaku v nizu
* osnovne metode za delo z nizi

**Objekti tipa DateTime v C#, gradnika Timer in DateTimePicker [15 točk]**

Kreiranje objekta DateTime, namen, uporaba, .

* kreiranje objekta *DateTime*, namen, uporaba datotek
* gradnik *Timer*, namen, uporaba
* gradnik *DateTimePicker*, namen, uporaba

3

**Varovalni bloki v C#**

Pomen varovalnih blokov, vrste varovalnih blokov. Večkratni varovalni blok. Primer varovalnega bloka za obdelavo napake.

* pomen varovalnih blokov, vrste varovalnih blokov
* večkratni varovalni blok
* primer varovalnega bloka za obdelavo napake

**Pogovorna okna v Visual C#**

Skupine pogovornih oken in njihov namen. Osnovno sporočilno okno, pogovorna okna za delo z datotekami, imeniki, fonti in barvami,…

* skupine pogovornih oken in njihov namen
* osnovno sporočilno okno: standardno pogovorna okna za delo z datotekami, imeniki, fonti in barvami,…

4

**Algoritem**

Definicija algoritma, osnovni pogoji, ki jih mora izpolnjevati vsak algoritem. Načini zapisov algoritma.

* definicija
* osnovni pogoji, oblike zapisa algoritma

**MDI – vmesnik z več dokumenti**

Starševski in otroški obrazci v Visual C#. Kreiranje starševskega obrazca. Odpiranje in zapiranje otroških obrazcev. Razporejanje odprtih obrazcev

* starševski in otroški obrazci v Visual C#, kreiranje starševskega obrazca
* odpiranje in zapiranje otroških obrazcev, razporejanje odprtih obrazcev zapiranje in uničenje takih obrazcev

5

**Pseudokoda in sled algoritma**

Definicija pseudokode. Razlaga in namen pseudokode. Primer pseudokode za izračun vsote prvih 100 naravnih števil.

* definicija
* razlaga in namen
* primer pseudokode

**Knjižnice razredov in datoteka z razredom v okolju Visual C#**

* Ustvarjanje lastne knjižnice (.dll)
* Uporaba lastne knjižnice razredov
* Ustvarjanje in uporaba datoteke z razredom (.cs)

6

**Zanka *foreach* v C#**

Sintaksa *foreach* zanke. Razlike med *for* stavkom in zanko *foreach*. Primer zanke *foreach* za izpis tabele stavkov

* sintaksa *foreach* zanke
* razlike med *for* stavkom in zanko *foreach*
* primer zanke *foreach* za izpis tabele stavkov

**Dinamično ustvarjanje okenskih gradnikov.**

* Dinamično ustvarjanje okenskih gradnikov
* Dinamično ustvarjanje obrazca
* Dinamično ustvarjanje okenskih dogodkov

7

**Osnovni podatkovni tipi v C#**

Naštej osnovne podatkovne tipe v C# in njihovo delitev, njihove zaloge vrednosti. Koliko prostora zasedajo v pomnilniku? Metoda *Sizeof()*. Osnovne aritmetične operacije in funkcije nad njimi.

* osnovni podatkovni tipi
* zaloga vrednosti osnovnih podatkovnih tipov, metoda *sizeof*
* osnovne aritmetične operacije in funkcije nad osnovnimi podatkovnimi tipi

**Sporočilno okno MessageBox, lastnost DialogResult**

Sporočilno okno *MessageBox*: oblike (načini uporabe) sporočilnega okna, lastnost *DialogResult*, vračanje vrednosti

* Sporočilno okno *MessageBox*: oblike
* lastnost *DialogResult*, vračanje vrednosti

8

**Zapis podatkov v tekstovne datoteke v C#**

Ustvarjanje oz. odpiranje tekstovne datoteke za potrebe zapisovanja podatkov. Metode za zapisovanje. Zapis podatkov v datoteko. Ločilni znaki.

* ustvarjanje oz. odpiranje tekstovne datoteke za potrebe zapisovanja podatkov
* metode za zapisovanje, ločilni znak
* zapis podatkov v datoteko

**Gradnik DataGridView**

Kreiranje stolpcev, vstavljanje podatkov, dostop do celic, obdelava podatkov v gradniku DataGridView

* Kreiranje stolpcev (programsko in s pomočjo okna Properties)
* Vstavljanje podatkov, dostop do celic
* Obdelava podatkov v gradniku DataGridView

9

**Napake pri programiranju, testiranje programov**

Vrste napak pri programiranju. Iskanje napak, orodja za iskanje napak. Razloži pri katerih napakah lahko prevajalnik pomaga programerju? Katero orodje pa vsaj delno pomaga pri odpravljanju druge vrste napak?

* vrste napak pri programiranju , iskanje napak.
* testiranje programov, orodja za odkrivanje napak pri programiranju

**Objekti tipa DateTime in TimeSpan , dogodki ure**

Kreiranje objekta TimeSpan, namen, uporaba. Delo z uro, lastnosti gradnika za delo u uro, dogodki ure. Kontrola uporabnikovega vnosa podatkov v vnosna polja.

* Kreiranje objekta tiap DateTime in TimeSpan, uporaba
* Delo z uro, lastnosti gradnika za delo u uro, dogodki ure, merjenje časa

10

**Enodimenzionalna polja/tabele v C#**

Ustvarjanje tabelaričnega podatkovnega tipa. Indeksi, inicializacija tabele, dostop do posameznega elementa tabela, prednosti in pomanjkljivosti uporabe tabel.

* ustvarjanje tabelaričnega podatkovnega tipa,
* pomen indeksov, primer deklaracije in inicializacije tabele
* prednosti in pomanjkljivosti uporabe tabel

**Gradniki ProgressBar, ToolTip, StatusStrip, PictureBox in ImageList**

* Namen, opis in uporaba gradnika gradnika ProgressBar
* Izdelava menija in orodjarne – gradnika!
* Delo s slikami – gradnika ImageList in PictureBox

11

**Deklaracija in definicija spremenljivk**

Razlika med deklaracijo in definicijo spremenljivk. Napiši primer za eno in drugo ter še primer za oboje hkrati. Pri zadnjem primeru razloži še pojem inicializacija spremenljivke. Življenjska doba spremenljivke. Sklad in kopica.

* deklaracija in definicija spremenljivke, primer
* inicializacija in življenjska doba spremenljivke
* sklad in kopica

**Dogodki tipkovnice v Visual C#**

Dogodki, ki jih proži tipkovnica. Vrstni red dogodkov. Kontrola uporabnikovega vnosa podatkov (npr. kako uporabniku preprečiti vnos vseh znakov razen cifer). Razlaga stavka **e.Handled = true;**

* dogodki tipkovnice, vrstni red dogodkov
* vrstni red dogodkov
* kontrola uporabnikovega vnosa

12

**Razred Console, tehnologija IntelliSense in vrste pomoči pri pisanju kode, prevajanje in zagon projekta, komentarji v C#**

Pomen razreda *Console*, uporaba in vrsta pomoči s pomočjo tehnologije *IntelliSense*, *using* stavki, prevajanje in zagon novega projekta

* pomen razreda *Console*
* uporaba in vrsta pomoči s pomočjo tehnologije *IntelliSense*, *using* stavki
* prevajanje in zagon novega projekta, komentarji

**Osnovni gradniki v Visual C#**

Osnovni gradniki (TextBox, ListBox, ComboBox, Button, CheckBox, RadioButton, GroupBox, Panel, TabControl) in njihove glavne lastnosti in dogodki.

* osnovni gradniki v Visual C#
* lastnosti in dogodki gradnikov

13

**Enostavne vhodno - izhodne operacije v C#**

Naštej osnovne vhodno – izhodne operacije ( branje in izpis) v C#, uporaba blokov {} v stavku *WriteLine*. Formatiranje izpisov s pomočjo metode *WriteLine*.

* metode za branje in izpis
* uporaba blokov {} v stavku *WriteLine*
* formatiranje izpisov

**Pomen dogodkov okenskih gradnikov, odzivne metode v Visual C#**

Lastne metode. Odzivne metode, Klicatelj metode – parameter *Sender*.

* Dogodki okenskih gradnikov
* odzivne metode
* parameter *Sender*

14

**Krmilni stavki v C#**

Opiši osnovni krmilni stavek v C#, razloži razliko med pogojno izvršitvijo in razvejitvijo, razloži testiranje enostavnih in sestavljenih pogojev, relacijski operatorji v pogojih.

* krmilni stavek, pogojna izvršitev in razvejitev
* testiranje enostavnih in sestavljenih pogojev
* relacijski operatorji v pogojih

**Dogodki ki jih proži miška v Visual C#, parameter sender**

Najpomembnejših dogodki, ki jih proži miška. Podrobna razlaga najpomembnejših dogodkov miške. Ugotavljanje položaja miške in stisnjenega gumba.

* najpomembnejših dogodki, ki jih proži miška
* podrobna razlaga najpomembnejših dogodkov miške
* Ugotavljanje položaja miške in stisnjenega gumba

15

**Pretvarjanje oz. konverzije med osnovnimi podatkovnimi tipi z metodo *Convert*. Eksplicitne konverzije. Standardne oznake za formatiranje števil in njihova uporaba v metodi *ToString*, metoda *Parse***

* Pretvarjanje oz. konverzije med osnovnimi podatkovnimi tipi
* Eksplicitne konverzije
* Standardne oznake za formatiranje, metoda *Parse*

**Dostopnost označevalcev – spremenljivk in objektov v Visual C#**

Dostop do spremenljivk/metod glavnega modula in do gradnikov glavnega obrazca v drugih modulih/obrazcih. Dostop do spremenljivk in objektov iz drugih modulov/obrazcev v glavnem modulu. Pomen in uporaba (primer) dogodka *Load*.

* dostop do spremenljivk glavnega modula in do gradnikov glavnega obrazca
* pomen in uporaba dogodka *Load*
* dostop do spremenljivk in objektov v drugih modulih

16

***Switch* stavek v C#**

Kdaj uporabljamo stavek *switch*, sintaksa ***switch*** stavka, popoln in nepopoln *switch* stavek, kakšen je lahko izraz, ki nastopa kot pogoj v *switch* stavku. Pomen stavka *break* v *switch* stavku.

* namen uporabe *switch* stavka
* sintaksa popolnega in nepopolnega *switch* stavka
* pomen stavka *break*

**Izdelava novega projekta v Visual C#**

Izdelava novega projekta v Visual C#. Shranjevanje. Vključevanje že izdelanega obrazca v projekt. Izdelava datoteke z razredom in vključitev v projekt.

* Izdelava novega projekta v Visual C#. Shranjevanje.
* Vključevanje že izdelanega obrazca v projekt
* Izdelava datoteke z razredom in vključitev v projekt.

17

**For zanka v C#**

Sintaksa *for* zanke v C#, diagram poteka. Poimenovanje posameznih stavkov v glavi *for* zanke. Uporaba dveh ali več števcev v zanki, števec naštevnega tipa v *for* zanki, sestavljanje pogojev.

* Sintaksa *for* zanke v C#, diagram poteka, poimenovanje posameznih stavkov
* Uporaba dveh ali več števcev v zanki
* števec naštevnega tipa v v *for* zanki, sestavljanje pogojev

**Podatkovna skladišča**

* Ustvarjanje nove SQL Baze in nove tabele v bazi
* Prikaz vsebine podatkovne tabele v gradniku DataGridView
* Izdelava projekta nad obstoječo bazo podatkov – vključitev obstoječe baze v projekt

18

**Operatorji v C#**

Najpomembnejši operatorji v C#. Aritmetični operatorji, logični operatorji, operatorji prirejanja, primerjalni operatorji.

* najpomembnejši operatorji v C++, krajši zapis operatorjev
* pomen preinkrementa in poinkrementa predekrementa in podekrementa
* logični operatorji, operatorji prirejanja, primerjalni operatorji

**Obrazec v Visual C#**

Videz obrazca – lastnosti *FormBorderStyle* in *ControlBox*. Preklapljanje med pogledom Designer in Code. Konstruktor obrazca. Glavne lastnosti obrazca, dogodki obrazca

* videz obrazca – lastnosti *BorderStyle* in *ControlBox*
* Designer view in Code view, konstruktor obrazca
* lastnosti in dogodki obrazca

19

**Tekstovne datoteke v C#**

Delo s tekstovnimi datotekami. Kako preverimo obstoj datoteke? Kako kreiramo tekstovno datoteko? Metode za branje in pisanje ter dodajanje podatkov. S katero metodo bi prebral celotno datoteko in prepisal njeno vsebino v drugo datoteko?

* delo s tekstovnimi datotekami, preverjanje obstoja datoteke
* operacije pri delu z z datotekami, obdelava tekstovne datoteke
* metoda za prepis celotne datoteke v drugo

**Dedovanje, ustvarjanje novih gradnikov in dedovanje okenskih gradnikov.**

* Dedovanje – pomen.
* Dedovanje lastnega obrazca
* Ustvarjanje novih gradnikov, ter dedovanje poljubnega gradnika in dodajanje novih lastnosti in dogodkov

20

**While zanka v C#**

Sintaksa *while* zanke, diagram poteka. Sestavljanje pogojev, primer pogoja za neskončno zanko in izhod iz take zanke:

* sintaksa *while* zanke, diagram poteka
* sestavljanje pogojev
* neskončna zanka

**Meni, lebdeči meni, sporočilna vrstica na dnu obrazca, gradnik LinkLabel**

* meni in lebdeči meni
* sporočilna vrstica
* gradnik LinkLabel

21

**Do while zanka v C#**

Sintaksa *do while* zanke, diagram poteka. Sestavljanje pogojev, razlika med zanko *while* in *do while*.

* sintaksa *do while* zanke, diagram poteka
* sestavljanje pogojev
* razlika med zanko *while* in *do while*

**Delo s tiskalnikom**

* dialogi za delo s tiskalnikom
* nastavitve za delo s tiakalnikom: dogodek za tiskanje, tiskanje besedil, slik in gradfike
* predogled tiskanja.

22

**Razred, sestavni deli razreda, pomen konstruktorjev**

Razloži pomen konstruktorjev in destruktorjev v C#, pravila pri pisanju konstruktorjev in destruktorjev.

* Razred in sestavni deli razreda
* razloži pomen konstruktorjev
* pravila pri pisanju konstruktorjev, privzeti konstruktor

**Razvojno okolje za izdelavo okenskih aplikacij Visual C#**

Paleta z gradniki, nadzornik lastnosti in dogodkov predmeta, obrazec, urejevalnik programske kode, okno *Form Designer*. Označevalci v *Visual* C# (C#-ovi, uporabnikovi, poimenovalna pravila, poimenovalni dogovori, sestavljeni označevalci). Okna, ki jih uporabljamo pri izdelovanju projekta.

* paleta z gradniki, nadzornik lastnosti in dogodkov predmeta
* obrazec in urejevalnik programske kode, okno Code Designer
* označevalci v C#, okna, ki jih uporabljamo pri izdelovanju projekta.

23

**Naključna števila, razred Math v C#, osnovne matematične metode in njih uporaba, število PI, kotne funkcije, formatiranje decimalnih števil na poljubno število decimalk**

Uporaba metod razreda *Math*: absolutna vrednost, kvadriranje, kvadratni koren, zaokroževanje, celi del števila, kotne funkcije.

* Delo z naključnimi števili
* Razred Math
* najpomembnejše metode razreda *Math*

**Osnovni postopki pri izdelavi projekta v Visual C#**

Začenjanje novega projekta. Shranjevanje projekta. Kreiranje in odstranjevanje dogodkov. Postavljanje gradnikov na obrazec. Kreiranje in odstranjevanje dogodkov. Poimenovanje in preimenovanje in odstranjevanje gradnikov, obrazcev, odzivnih dogodkov..

* začenjanje novega projekta, shranjevanje projekta
* kreiranje in odstranjevanje dogodkov.
* poimenovanje, preimenovanje in odstranjevanje gradnikov, obrazcev, odzivnih dogodkov.

24

**Metode v C#**

Kaj so to metode, vrste metod in razlike med njimi, klic. Kaj so parametri metode. Na lastnem primeru prikaži njihovo sintakso in uporabo!

* kaj so to metode, vrste, razlike, klic
* parametri
* primer dveh različnih lastnih metod

**Projekt v Visual C#**

Moduli/datoteke, ki sestavljajo projekt v Visual C#. Imenski prostor. Dodajanje novih modulov in obrazcev v projekt. Lastnosti obrazca, parameter *this*. Pogled *Code/Design*.

* projektni modul, imenski prostor
* lastnosti obrazca, parameter *this*, pogled *Code/Design*
* *do*dajanje novih modulov in obrazcev v projekt.

25

**Večdimenzionalna polja/tabele v C#**

Deklaracija večdimenzionalne tabele. Inicializacija tabele. Kako dostopamo do posameznega elementa tabele? Primer: kako se imenuje element v tabeli Tab, ki je v tretji vrstici in petem stolpcu? Ali morajo biti vsi elementi tabele istega podatkovnega tipa ali ne?

* deklaracija večdimenzionalne tabele, primer deklaracije in inicializacije tabel
* dostop do posameznega elementa tabele
* podatkovni tipi v tabeli

**Projekt z več obrazci v C#, dostop do gradnikov, spremenljivk in metod na glavnem obrazcu**

* pomen glavnega obrazca
* vključevanje že izdelanega obrazca v projekt in izključevanje obrazca iz projekta
* dostop do gradnikov, spremenljivk in metod na glavnem obrazcu

26

**Sestavljeni(strukturirani) podatkovni tipi v C#**

Naštej strukturirane( sestavljene) podatkovne tipe v C# (štiri tipe). Primeri deklaracij. Kakšne so zaloge vrednosti teh tipov. Deklariraj sestavljeno podatkovno strukturo, s katerim bi lahko opisal metalca krogle, elementi tega tipa pa naj bodo naziv tekmovalca in dolžine šestih zaporednih metov krogle!

* naštej strukturirane podatkovne tipe v C#
* primeri deklaracij, zaloge vrednosti
* deklaracija tipa za metalca krogle

**Delo z bazami podatkov: objekti tipa Dataset, TableAdapter, BindingNavigator**

* Pomen objetov tipa DataSet in TableAdapter
* Vključitev gradnika DataSet in ustreznih TableAdapterjev v projekt s pomočjo čarovnika
* Gradnik BindingNavigator in gradnik TableAdapterManager

27

**Datoteke v C#**

Pomen razredov *File*, *Directory* in *Path*. Razlika med tekstovnimi in binarnimi datotekami, glavne operacije nad datotekami. Ustvarjanje in pomen objektov razreda *FileStream* in *StreamReader*

* pomen razredov *File*, *Directory* in *Path*
* razlike med tekstovnimi in binarnimi datotekami, glavne operacije nad datotekami
* ustvarjanje in pomen objektov razreda *FileStream* in *StreamReader*

**Prekinitvene točke v Visual C#, razhroščevanje**

Nastavljanje prekinitvenih točk v Visual C# in iskanje napak. Preverjanje trenutnih vrednosti spremenljivk med izvajanjem programa. Razlika med opcijami Run, Step Over in Step Into.

* nastavljanje prekinitvenih točk v Visual C#, koračno izvajanje programa
* preverjanje trenutnih vrednosti spremenljivk med izvajanjem programov
* razhroščevanje (Visualizer, okno Watch, Immediate, Locals, Call Stack)

28

**Stavka break, continue in pomen stavkov Environment.Exit(0);** **in stavka** **return**  **v C#**

Pomen stavkov *break* in *continue*. Navedi dva primera in nariši diagram poteka zanju. Pomen stavka Environment.Exit(0); in stavka return  **v C#**

* pomen stavkov *break* in *continue*
* primer uporabe stavkov *break* in *continue*
* staveki za zapiranje (zaključek projekta)

**Polja in metode razreda, kapsulacija, parameter *this*, statični in nestatični elementi razredov**

Skrivanje polj in metod razreda, nivoji skrivanja (kapsulacija), parameter *this*, statični in nestatični elementi razredov – kdaj uporabimo statične metode in polja

* skrivanje polj in metod razreda, nivoji skrivanja (kapsulacija),
* parameter *this*
* statični in nestatični elementi razredov, pomen, uporaba

29

**Zbirke – Collections v C#**

Pojem zbirke. Razlike med tabelami in zbirkami. Vrste zbirk: netipizirane in tipizirane zbirke. Primer netipizirane in tipizirane zbirke. Metode za dodajanje podatkov v zbirko, število podatkov v zbirki, odstranjevanje podatkov iz zbirke, obdelava podatkov v zbirki.

* pojem zbirke, razlike med tabelami in zbirkami
* vrste zbirk: netipizirane in tipizirane zbirke, metode
* primer netipizirane in tipizirane zbirke

**Povezovanje gradnikov z zbirkami podatkov**

Povezovanje z netipiziranimi in tipiziranimi zbirkami, zbirka tipa BindingList, Povezava s tabelo v bazi podatkov, ter povezava z gradnikom ComboBox

* Povezovanje z netipiziranimi in tipiziranimi zbirkami
* Povezava z zbirko tipa BindingList
* Povezava s tabelo v bazi podatkov, ter povezava z gradnikom ComboBox (Display Member in Value Member)

30

**Preobložene funkcije v C#**

Pojem preobloženih funkcij v C#. Napiši primer preobloženih funkcij za izračun ploščine kroga in pravokotnika. Funkcija *Main* in uporaba njenih parametrov.

* pojem preobloženih funkcij v C#
* napiši primer preobloženih funkcij za izračun ploščine kroga in pravokotnika
* funkcija *Main* in uporaba njenih parametrov

**Gradnik ErrorProvider, gradnik TabControl**

* pomeg gradnika ErrorProvider
* uporaba gradnika ErrorProvider
* pomen in uporaba gradnika TabControl

<http://www.s-sers.mb.edus.si/gradiva/rac/drugo/javascript/01_datoteka.html>

<http://nsa.zpanel.sckr.si/css/zgledi/css-04-programska-oprema.php>