**Promenljive, tipovi podataka i operatori**

**00:30:49**

Jedinica: 21 od 30

**Rezime**

* Promenljive služe kao kontejneri za vrednosti koje se koriste prilikom izvršavanja koda.
* Deklaracija predstavlja kreiranje promenljive (varijable).
* Incijalizacija je dodeljivanje početne vrednosti promenljivoj.
* JavaScript je slabo tipizirani jezik, što znači da se tipom podataka rukuje automatski.
* JavaScript poznaje sledeće tipove podataka: Number, String, Boolean, Null, Undefined i Object.
* Objekti se mogu doživeti kao kontejneri za veći broj podataka.
* Nizovi omogućavaju da se unutar jedne promenljive smesti više vrednosti.
* Operatori su specijalne jezičke konstrukcije za vršenje operacija nad operandima.
* Najznačajniji operatori u jeziku JS su: aritmetički operatori, operatori dodeljivanja, operatori poređenja, logički operatori, String operatori, specijalni operatori.

U prethodnoj lekciji izneta su neka osnovna sintaksna pravila jezika JavaScript. Rečeno je da je osnovna jedinica obrade izjava, i da se izjave u jeziku JavaScript završavaju karakterom tačka sa zapetom. Takođe, rečeno je da se izvorni kod JS-a sastoji iz mnoštva elemenata: ključnih reči, identifikatora, komentara, praznih mesta…

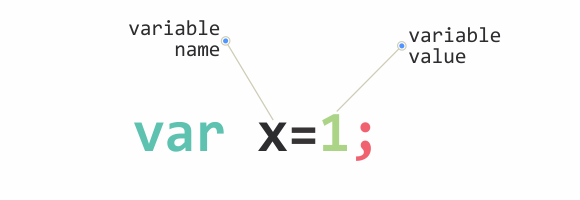
Podsetimo se linije iz prethodne lekcije, koja je poslužila za definisanje osnovne sintakse jezika:

|  |  |
| --- | --- |
| 1 | var x=1; |

Rečeno je da je u prikazanoj liniji koda, slovom x označen naziv promenljive, a da je vrednost promenljive 1.

**Šta su promenljive?**

Promenljive, kako im i samo ime kaže, služe kao kontejneri za vrednosti koje se koriste prilikom izvršavanja koda. Promenljive se sastoje iz naziva, koji se drugačije naziva identifikator, i vrednosti (slika 21.1).



*Slika 21.1 – Promenljiva*

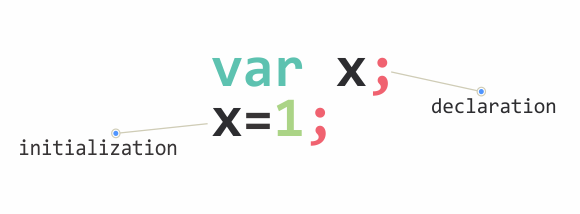
Da bi se neka promenljiva kreirala, baš kao što se to postiže u prikazanoj liniji, pribegava se procesu koji se naziva deklaracija.

**Deklaracija i inicijalizacija**

*Deklaracija predstavlja kreiranje promenljive (varijable)*. Kodom iz primera, kreirana je varijabla sa nazivom x.

*Incijalizacija je dodeljivanje početne vrednosti promenljivoj*. Tako promenljiva x iz primera ima početnu vrednost 1.

Iz navedenog se može zaključiti, da je u sledećoj liniji obavljena i deklaracija i inicijalizacija. Deklaracija i inicijalizacija su odvojene operacije, a moguće ih je obaviti u okviru jedne ili dve izjave. Razdvajanje deklaracije i inicijalizacije na dve izjave ilustrovano je slikom 21.2.



*Slika 21.2 – Deklaracija i inicijalizacija*

Deklaraciju je u JavaScript jeziku moguće postići korišćenjem ključnih reči prikazanih u tabeli 21.1.

|  |  |
| --- | --- |
| **Ključna reč** | **Opis** |
| var | deklariše promenljivu |
| let | deklariše lokalnu promenljivu |
| const. | deklariše konstantu čiju vrednost je moguće samo čitati |

*Tabela 21.1 – Ključne reči za kreiranje promenljivih*

Već je prikazan efekat ključne reči var za deklarisanje promenljive. Da bi se razumela razlika između ključnih reči var i let, neophodno je poznavati opseg važenja promenljive. O tome će biti više reči već u narednom poglavlju.

Ključna reč const omogućava da se definiše konstanta. Konstanta je vrednost koja se ne može menjati nakon inicijalizacije. Primer deklarisanja konstante je sledeći:

|  |  |
| --- | --- |
| 1 | const PI = 3.14; |

Na ovaj način je deklarisana konstanta sa nazivom PI i vrednošću 3.14. Time je osigurano da su i naziv a i vrednost konstante rezervisani. Drugim rečima, ukoliko se napiše sledeća linija koda, doći će do greške:

|  |  |
| --- | --- |
| 1 | PI = 6.54; |

Kada se definiše konstanta određenog imena, zabranjeno je ponovno deklarisanje promenljive sa istim imenom. Tako da i sledeća linija proizvodi grešku:

|  |  |
| --- | --- |
| 1 | const PI = 6.54; |

**Tipovi podataka**

Kako bi parser bio u stanju da izvršava neke operacije nad promenljivama, neophodno je da poznaje njihov tip. Pod tipom se prevashodno misli na tip vrednosti promenljive. Vrednost promenljive u bilo kom jeziku može biti tekst (string), broj, niz, logička vrednost itd.

Koncept tipova podataka je nešto što postoji u svakom programskom jeziku, ali se različiti jezici na različite načine odnose prema tipovima. JavaScript je, za razliku od svojih glavnih uzora, slabo tipizirani jezik. To praktično znači da JavaScript ne podržava striktno definisanje tipova prilikom kreiranja promenljive. Ukoliko još jednom pogledamo ovu liniju:

|  |  |
| --- | --- |
| 1 | var x=1; |

možemo zaključiti da ni na koji način tip podatka promenljive nije naznačen. Razlog je taj što JavaScript poseduje funkcionalnost dinamičkih tipova. To praktično znači da jedna promenljiva može da prihvati vrednost bilo kojeg tipa.

JavaScript poznaje sledeće tipove podataka:

* Number
* String
* Boolean
* Null
* Undefined
* Object
* Array

**Brojevi (Numbers)**

Broj je najjednostavniji tip podatka u JavaScript jeziku. Svi primeri do sada, ilustrovali su kreiranje jedne promenljive koja je tipa Number. Iako većina jezika poseduje nekoliko različitih numeričkih tipova podataka (int, float, double…), JavaScript sve numeričke podatke objedinjuje pod tipom Number.

Kada se deklariše neka promenljiva i njoj dodeli vrednost nekog broja, može se reći da je kreirana promenljiva tipa Number.

|  |  |
| --- | --- |
| 1 | var x = 10; |

U primeru, promenljiva sa nazivom x je tipa Number.

Numeričke vrednosti se u JavaScriptu mogu pisati na nekoliko načina: sa decimalama, bez decimala i u [eksponencijalnom obliku](https://www.link-elearning.com/linkdl/opisPojma.php?id=144317).

|  |  |
| --- | --- |
| 1  2  3  4 | var x1 = 56.00; // with decimals  var x2 = 56; // without decimals  var y = 154e5; // exponential 15400000  var z = 154e-5; // exponential 0.00154 |

JavaScript poznaje i dve specijalne numeričke vrednosti:

* NaN
* Infinity

NaN je skraćenica za pojam Not-a-Number, zato što predstavlja vrednost koja se koristi da ukaže da vrednost nije validan broj. Sledeći primer proizvodi NaN vrednost:

|  |  |
| --- | --- |
| 1 | var x = 0 / 0; // x is NaN |

**NaN** je vrlo karakteristična vrednost, koja je, i pored toga što ukazuje na to da neka vrednost nije broj, i sama tipa Number. Takođe, NaN je jedina vrednost u JavaScript jeziku koja, prilikom poređenja sa samom sobom, daje vrednost false.

Druga specijalna numerička vrednost u JavaScriptu jeste Infinity. Ova vrednost ukazuje na broj koji teži beskonačnosti. Tako će deljenje nekog broja nulom proizvesti ovu vrednost:

|  |  |
| --- | --- |
| 1 | var x = 5 / 0; // x is Infinity |

Infinity može označavati i negativnu beskonačnu vrednost:

|  |  |
| --- | --- |
| 1 | var x = -5 / 0; // x is –Infinity |

**Stringovi**

JavaScript promenljiva može kao svoju vrednost imati i neki tekst. Tekstualna vrednost se, za razliku od numeričkih, piše između navodnika, i na taj način JS parser zna da je reč o tekstu. Dozvoljeno je pisanje tekstualnih vrednosti između jednostrukih i dvostrukih navodnika:

|  |  |
| --- | --- |
| 1  2 | var x = 'text 1';  var y = "text 2"; |

Na ovaj način, kreirane su dve promenljive x i y, i obe su tipa String.

Navođenje navodnika unutar vrednosti stringa je moguće, sve dok se takvi navodnici razlikuju od onih koji su iskorišćeni za definisanje string vrednosti:

|  |  |
| --- | --- |
| 1  2  3 | var answer = "It's alright";             // Single quote inside double quotes  var answer = "He is called 'Johnny'";    // Single quotes inside double quotes  var answer = 'He is called "Johnny"';    // Double quotes inside single quotes |

**Boolean – Logičke vrednosti**

Boolean je specijalan tip podatka, koji poznaje samo dve predefinisane vrednosti: true i false. Ove vrednosti se obično koriste za kreiranje uslovnih izraza, i vrlo često za kontrolu toka izvršavanja koda.

Primer promenljive koja će imati vrednost tipa Boolean može da izgleda ovako:

|  |  |
| --- | --- |
| 1 | var x = 5==6; |

Za vrednost promenljive x, postavljen je jedan logički izraz, unutar koga je upotrebljen operator jednakosti. Jednostavnije rečeno, prikazana linija koda može da se pročita: *proveri da li je 5 jednako 6, i informaciju o tome upiši u promenljivu x*. S obzirom da 5 nije jednako 6, ovakav izraz će proizvesti logičku vrednost false. To je vrednost Boolean tipa.

**Undefined**

Kada se u JavaScriptu deklariše nova promenljiva, bez inicijalizacije, ona automatski dobija početnu vrednost. Reč je o vrednosti undefined.

|  |  |
| --- | --- |
| 1 | var x;  // Value is undefined, type is undefined |

Na ovaj način, deklarisana je promenljiva sa nazivom x. S obzirom da nije inicijalizovana, imaće vrednost undefined. A pošto ima vrednost undefined, ni njen tip neće biti poznat, pa se i za njega može reći da je undefined.

Korišćenjem undefined vrednosti, moguće je neku promenljivu efikasno resetovati na početno stanje:

|  |  |
| --- | --- |
| 1  2 | var x = 13;  x = undefined; //value is undefined, type is undefined |

**Null**

Null je jedna od specijalnih vrednosti koje JS promenljive mogu imati. Ne može se nazvati tipom, zato što je tip null vrednosti zapravo Object, za razliku od undefined vrednosti koja je tipa undefined.

Uglavnom, vrednost null je zamišljena kao indikator odsustva vrednosti. Ukoliko je potrebno definisati odsustvo vrednosti neke promenjive, može se napisati:

|  |  |
| --- | --- |
| 1 | var x = null; // Value is null, type is Object |

Razlika između pojmova Null i Undefined je više nego očigledna. Pored toga što je Undefined tip, a Null nije, undefined vrednost označava da je promenljiva deklarisana, ali da joj još nije dodeljena vrednost. Pošto je null zapravo vrednost, promenljiva sa ovom vrednošću se smatra inicijalizovanom, a njen tip je postavljen na Object.

**Koji tip podatka se u JavaScriptu koristi za predstavljanje brojeva:**

 Number

 String

 Boolean

 Undefined

**Object**

Sve vrednosti sa kojima smo se do sada upoznali bile su takozvane proste ili primitivne vrednosti. Promenljive sa vrednostima primitivnih tipova podataka zapravo mogu sadržati samo jednu vrednost u jednom trenutku:

|  |  |
| --- | --- |
| 1 | var car = "Ford"; |

Jedan automobil može biti definisan brojnim podacima: proizvođač, model, boja, zapremina motora… Hajde da vidimo kako bismo sve te podatke predstavili tradicionalno korišćenjem JavaScripta:

|  |  |
| --- | --- |
| 1  2  3  4 | var make = "Ford";  var model = "Mondeo";  var color = "black";  var cubic\_capacity = 2499; |

Bilo bi stvarno sjajno kada bi bilo moguće sve navedene vrednosti smestiti u jednu promenljivu. Tako nešto je zapravo moguće korišćenjem objekata.

Objekti se mogu doživeti kao kontejneri za veći broj podataka. Tako je prethodni primer moguće transformisati na sledeći način:

|  |  |
| --- | --- |
| 1 | var car = {make:"Ford", model:"Mondeo", color:"black", cubic\_capacity:2499}; |

Na ovaj način, kreiran je jedan objekat, koji sadrži sve informacije o jednom vozilu.

|  |
| --- |
| **Napomena**    *O objektima će biti mnogo više reči u jednoj od narednih lekcija.* |

**Array**

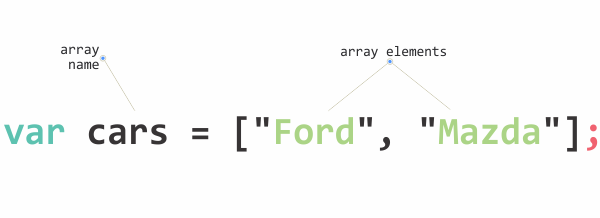
Specijalan tip objektnog podatka u jeziku JavaScript jeste Array. Array je zapravo objekat koji je u JavaScript jeziku dostupan na globalnom nivou, a koristi se za kreiranje nizova. Podsetimo se primera iz prethodnog poglavlja:

|  |  |
| --- | --- |
| 1 | var car = "Ford"; |

Ovo je jedna promenljiva sa nazivom car i vrednošću *Ford*. Kako bi bilo kada bi promenjiva car, pored vrednosti *Ford*, mogla da sadrži i neke druge vrednosti koje bi predstavljale i ostale proizvođače automobila. Upravo takvo ponašanje može da se postigne upotrebom nizova.

|  |  |
| --- | --- |
| 1 | var cars = ["Ford", "Mazda"]; |

Nizovi se formiraju korišćenjem uglastih zagrada, unutar kojih se navode vrednosti razdvojene zapetama. Struktura jednog niza prikazana je na slici 21.3.

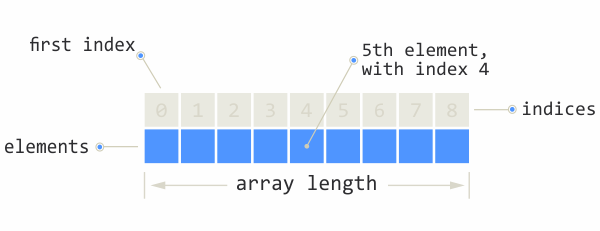


*Slika 21.3 – Struktura jednog niza*

Opšta sintaksa za kreiranje nizova u JavaScript jeziku je sledeća:

|  |  |
| --- | --- |
| 1 | var array-name = [item1, item2, ...]; |

Nizovi omogućavaju da se unutar jedne promenljive smesti više vrednosti. Slika 21.4 dočarava logiku jednog niza.



*Slika 21.4 – Niz*

Niz je sastavljen iz elemenata koji imaju svoje pozicije izražene kroz indekse. Indeksi počinju od nule. Dužina niza predstavlja ukupan broj elemenata u nizu.

|  |
| --- |
| **Napomena**    *Niz u JavaScriptu je moguće kreirati na još jedan način, koji podrazumeva upotrebu ključne reči new:*    var cars = new Array("Ford", " Mazda");    *Oba načina za kreiranje nizova proizvode identičan efekat, a zbog jednostavnosti, brzine i preglednosti se preporučuje prvi pristup.*    *S obzirom da su nizovi objekti u JavaScript jeziku, njihovi elementi su svojstva takvog objekta.* |

**Čitanje pojedinačnih elemenata niza**

Elementima jednog niza pristupa se korišćenjem indeksa koji svaki član niza poseduje. Indeksi elemenata unutar niza počinju od nule, tako da prvi element niza ima indeks nula.

|  |  |
| --- | --- |
| 1  2  3 | var cars = ["Ford", "Mazda"];  console.log(cars[0]);  console.log(cars[1]); |

Primer ilustruje proces čitanja elemenata niza, korišćenjem njihovih indeksa. Kada se izvrši, prikazani kod će unutar konzole browsera upisati:

|  |  |
| --- | --- |
| 1  2 | Ford  Mazda |

**Čitanje svih elemenata niza**

JavaScript omogućava da se sve vrednosti jednog niza pročitaju odjednom. Dovoljno je napisati:

|  |  |
| --- | --- |
| 1  2 | var cars = ["Ford", "Mazda"];  console.log(cars); |

**Promena vrednosti elemenata niza**

Na veoma lak način je moguće izvršiti promenu vrednosti nekog elementa u nizu:

|  |  |
| --- | --- |
| 1  2  3  4 | var cars = ["Ford", "Mazda"];  cars[1] = "BMW";  console.log(cars[0]);  console.log(cars[1]); |

**Dodavanje elementa u niz**

JavaScript nizovi su dinamičke prirode, što znači da nemaju fiksnu, unapred definisanu dužinu *(kao kod jezika C, npr*). To značajno olakšava izmenu strukture niza nakon inicijalizacije početnih elemenata.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var cars = ["Ford", "Mazda"];    cars[2] = "BMW";    console.log(cars[0]);  console.log(cars[1]);  console.log(cars[2]); |

Na početku primera, kreiran je niz sa dva elementa. Nakon kreiranja, javila se potreba za dodavanjem još jednog elementa. Jednostavnim korišćenjem indeksa, moguće je definisati novi član niza.

JavaScript nizovi nude veliku slobodu pri radu, pa čak nije neophodno ni elemente smeštati sukcesivno. Ipak, tako nešto uglavnom može stvoriti probleme, s obzirom da je potrebno znati na kojim indeksima se elementi nalaze.

**Svojstva i metode objekta Array**

Objekat Array poseduje set ugrađenih svojstava i metoda za rad sa nizovima. Tako je na lak način moguće obaviti neke osnovne intervencije nad vrednostima jednog niza.

**Čitanje dužine niza**

Čitanje dužine niza postiže se na veoma jednostavan način, korišćenjem svojstva length:

|  |  |
| --- | --- |
| 1  2  3 | var cars = ["Ford", "Mazda"];  var length = cars.length;  console.log(length); |

Prikazani kod proizvodi rezultat:

|  |  |
| --- | --- |
| 1 | 2 |

**Sortiranje niza**

Sortiranje niza postiže se korišćenjem metode sort:

|  |  |
| --- | --- |
| 1  2  3 | var cars = [1, 2, 55, 23, 0, 11, 123, -33];  cars.sort();  console.log(cars); |

Kod proizvodi:

|  |  |
| --- | --- |
| 1 | -33, 0, 1, 11, 123, 2, 23, 55 |

**Dodavanje elementa nizu**

Kada je dodavanje elementa u pitanju, pored bazičnog pristupa koji podrazumeva da se poznaje indeks na koji će element biti ubačen, moguće je koristiti i ugrađene metode push i unshift.

Metoda push dodaje element na kraj niza:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var cars = ["Ford", "Mazda"];  var newLength = cars.push("BMW");    console.log(newLength);  console.log(cars); |

Efekat je:

|  |  |
| --- | --- |
| 1  2 | 3  "Ford", "Mazda", "BMW" |

Metoda unshiftdodaje element na početak niza:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var cars = ["Ford", "Mazda"];  var newLength = cars.unshift("BMW");    console.log(newLength);  console.log(cars); |

Efekat je:

3  
"BMW", "Ford", "Mazda"

Obe metode, kao svoju povratnu vrednost imaju novu dužinu niza. U prikazanim primerima, ta nova družina niza dodaje se kao vrednost promenljivoj newLength, a taj podatak se zajedno sa elementima niza ispisuje unutar konzole.

**Uklanjanje elemenata iz niza**

I za uklanjanje elemenata iz niza postoje dve metode, u zavisnosti od lokacije sa koje će se element ukloniti.

Za uklanjanje elementa sa kraja niza koristi se metoda pop:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var cars = ["Ford", "Mazda"];  var last = cars.pop();    console.log(last);  console.log(cars); |

Metoda pop kao svoju povratnu vrednost ima element koji je uklonila iz niza. Tako će prikazani kod proizvesti efekat:

|  |  |
| --- | --- |
| 1  2 | Mazda  "Ford" |

Da bi se element uklonio sa početka niza, koristi se metoda shift:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var cars = ["Ford", "Mazda"];  var last = cars.shift();    console.log(last);  console.log(cars); |

Efekat je:

|  |  |
| --- | --- |
| 1  2 | Ford  "Mazda" |

**Prolazak kroz niz**

Prolazak kroz niz podrazumeva sukcesivno čitanje svakog elementa pojedinačno, i to uglavnom redom. Nekada nizovi mogu imati zaista puno elemenata, a može se javiti potreba za sprovođenjem izvesnih intervencija na svakom od elemenata niza. U takvim situacijama, ručno čitanje svakog elementa ne dolazi u obzir, već se mora pribeći nešto drugačijim rešenjima.

Osnovno rešenje ovakvog problema jeste upotreba petlji. Petlje su predmet jedne od narednih lekcija, kada će biti objašnjen osnovni pristup za prolazak kroz niz, korišćenjem petlji.

Pored petlji, prolazak kroz niz se može obaviti i korišćenjem specijalne ugrađene metode forEach:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var cars = ["Ford", "Mazda"];    cars.forEach(function (item, index, array) {   console.log(item, index);  }); |

Efekat je:

|  |  |
| --- | --- |
| 1  2 | Ford 0  Mazda 1 |

Metoda forEach prihvata kao parametar funkciju koja se poziva jednom za svaki element. Ulazni parametri funkcije su item, index, array, odnosno, tekući element koji se čita, njegov indeks i kompletan niz, respektivno.

|  |
| --- |
| **Napomena**    *Funkcije su poseban tip bloka u jeziku JavaScript, kojima će biti posvećena posebna pažnja u nastavku kursa.* |

**Čitanje indeksa elementa**

Svaki element u nizu poseduje svoj indeks. Ukoliko znamo vrednost nekog elementa, a želimo da saznamo njegov indeks, u pomoć možemo pozvati metodu indexOf:

|  |  |
| --- | --- |
| 1  2  3 | var cars = ["Ford", "Mazda"];  var pos = cars.indexOf("Mazda");  console.log(pos); |

Rezultat je:

|  |  |
| --- | --- |
| 1 | 1 |

**Konverzija tipova podataka**

S obzirom da je JavaScript jezik dinamičkih tipova, odnosno da se tipovi prilikom kreiranja promenljivih ne moraju navoditi, konverzija između tipova se takođe obavlja automatski. Ipak, iako se takav posao obavlja automatski, potrebno je poznavati osobine procesa automatskog konvertovanja, kako bi se on na najbolji način iskoristio.

U JavaScript jeziku moguće je jednoj istoj promenljivoj dodeljivati vrednosti različitog tipa, bez ikakvih problema:

|  |  |
| --- | --- |
| 1  2 | var x = 42;  x = "some text"; |

U prvoj liniji, promenljivoj x dodeljena je numerička vrednost 42. Na taj način promenljiva x postaje tipa Numeric. U narednoj liniji, promenljivoj x je promenjena vrednost, dodeljivanjem tekstualne vrednost. Tako, promenljiva x menja svoj tip u String, bez ikakvog problema.

Nad Numeric i String vrednostima može se koristiti operator sabiranja, a JavaScript će automatski obaviti konverziju.

|  |
| --- |
| **Napomena**    *Operatori su specijalni elementi jezika, koji se koriste za vršenje operacija nad operandima (promenljivama). Operatori su predmet sledećeg poglavlja.* |

|  |  |
| --- | --- |
| 1 | x = "Total length in km is " + 365 // " Total length in km is 365" |

Kao što se u primeru može videti, numerička vrednost *365*se konvertuje u vrednost tipa String i nadovezuje na vrednost sa leve strane izraza.

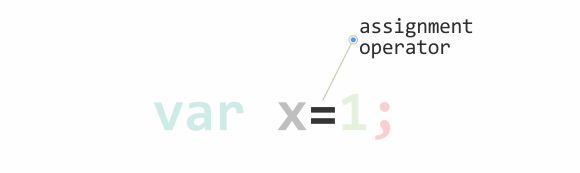
Ponekad se može javiti potreba za konvertovanjem String vrednosti u numeričku, a za obavljanje takvog posla JavaScript poseduje dve ugrađene metode, prikazane u tabeli 21.2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Potpis metode** | **Opis** | **Primer** | **Rezultat** |
| parseInt(*string*, *radix*); | parsira string vrednost koja je prosleđena kao prvi parametar korišćenjem osnove koja je prosleđena kao drugi parametar; drugi parametar definiše osnovu matematičkog brojčanog sistema koji će se koristiti prilikom konverzije; podrazumevana vrednost je 10; konverzija vrednosti se obavlja u celobrojnu numeričku vrednost | parseInt("015",10);  parseInt(15.99,10);  parseInt("15,123",10); | 15  15  15 |
| parseFloat(*string*); | Parsira string vrednost koja je prosleđena kao parametar i vraća vrednost sa decimalama | parseFloat("3.14");  parseFloat("6.18");  parseFloat("5"); | 3.14  6.18  5 |

*Tabela 21.2 – Metode za konverziju String podataka u tip Numeric*

**Operatori**

Nad promenljivama JavaScript jezika mogu se vršiti različite operacije. To se postiže upotrebom operatora. Čak je i u liniji koda koja je korišćena u prethodnim primerima upotrebljen jedan operator. Reč je o operatoru dodeljivanja.



*Slika 21.5 – Operator dodeljivanja*

Operator dodeljivanja, kao što i samo ime sugeriše, vrši dodeljivanje vrednosti promenljivoj. JavaScript poznaje nekoliko vrsta operatora:

* aritmetički operatori
* operatori dodeljivanja
* operatori poređenja
* logički operatori
* String operatori
* specijalni operatori

**Aritmetički operatori**

Aritmetički operatori su oni koji izvršavaju aritmetičke operacije nad promenljivama. Tabela 21.3 prikazuje aritmetičke operatore jezika JavaScript.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Opis** | **Primer** | **Vrednost** |
| + | sabiranje | var x = 5;  var y = 2;  var z = x + y; | 7 |
| - | oduzimanje | var x = 5;  var y = 2;  var z = x - y; | 3 |
| \* | množenje | var x = 5;  var y = 2;  var z = x \* y; | 10 |
| / | deljenje | var x = 5;  var y = 2;  var z = x / y; | 2.5 |
| % | ostatak pri deljenju (moduo) | var x = 5;  var y = 2;  var z = x % y; | 1 |
| ++ | uvećanje | var x = 5;  x++; | 6 |
| -- | smanjenje | var x = 5;  x--; | 4 |

*Tabela 21.3 – Aritmetički operatori u jeziku JavaScript*

S obzirom na dinamičku prirodu JavaScript tipova, aritmetičke operatore je moguće koristiti nad vrednostima različitih tipova, a JavaScript će obaviti automatski konverziju podataka. Iako ovaj pristup omogućava veliku slobodu prilikom rada sa vrednostima, potrebno je poznavati neka osnovna pravila kojih se JS pridržava prilikom obavljanja automatske konverzije.

Kao što je rečeno, nad Numeric i String vrednostima može se koristiti operator sabiranja, a JavaScript će automatski obaviti konverziju.

|  |  |
| --- | --- |
| 1 | x = "Total length in km is " + 365 // " Total length in km is 365" |

U primeru se može videti da je JS automatski obavio konverziju broja*365* u String tip podatka, i da je takvu string vrednost nadovezao na string sa leve strane izraza. Ista situacija bi se dogodila i u sledećem primeru:

|  |  |
| --- | --- |
| 1 | x = 365 +  " is the total length in km" // "365  is the total length in km" |

Iz prikazanog se može zaključiti da prilikom upotrebe operatora za sabiranje, nad vrednostima Numeric i String tipova, JS automatski konvertuje Numeric vrednosti u String.

Ipak, u izrazima u kojima se upotrebljavaju ostali aritmetički operatori, JS ne vrši konverziju Numeric vrednosti u String.

|  |  |
| --- | --- |
| 1 | "45" - 5 // 40 |

**Operatori dodeljivanja**

Operatorima dodeljivanja izvršava se dodeljivanje neke vrednosti promenljivoj. JavaScript poznaje nekoliko operatora dodeljivanja, prikazanih u tabeli 21.4. Osnovni operator dodeljivanja jeste karakter jednako (=), ali JS poznaje i nekoliko drugih operatora dodeljivanja, koji predstavljaju kombinaciju operatora dodeljivanja i aritmetičkih operatora.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Opis** | **Primer** | **Značenje** |
| = | dodeljuje vrednost sa desne strane promenljivoj sa leve | x = y | x = y |
| += | uvećava vrednost promenljive x za vrednost y i novodobijenu vrednost dodeljuje promenljivoj x | x += y | x = x + y |
| -= | umanjuje vrednost promenljive x za vrednost y i novodobijenu vrednost dodeljuje promenljivoj x | x -= y | x = x - y |
| \*= | množi vrednost promenljive x sa vrednošću y i novodobijenu vrednost dodeljuje promenljivoj x | x \*= y | x = x \* y |
| /= | deli vrednost promenljive x sa vrednošću y i novodobijenu vrednost dodeljuje promenljivoj x | x /= y | x = x / y |
| %= | računa ostatak deljanja vrednosti x vrednošću y, i dobijeni rezultat dodeljuje promenljivoj x | x %= y | x = x % y |

*Tabela 21.4 – Operatori dodeljivanja u jeziku JavaScript*

**Operatori poređenja**

Operatori poređenja koriste se u logičkim izjavama, kako bi utvrdili jednakost ili nejednakost između promenljivih. Tabela 21.5 prikazuje operatore poređenja JavaScript jezika.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Opis** | **Primer** | **Rezultat** |
| == | jednakost | 6 == 8  6 == 6  6 == "6" | False  True  true |
| === | jednakost po vrednosti i po tipu | 6===6  6==="6" | True  false |
| != | nejednakost | 6 != 8 | true |
| !== | nejednakost po vrednosti ili po tipu | 6 !== 6  6 !== "6"  6 !== 8 | False  True  true |
| > | veće | 6 > 8  6 > 4 | False  true |
| < | manje | 6 < 8  6 < 4 | True  false |
| >= | veće ili jednako | 6 >= 8  6 >= 6 | False  true |
| <= | manje ili jednako | 6 <= 8  6 <= 6 | True  true |

*Tabela 21.5 – Operatori poređenja u jeziku JavaScript*

Baš kao što je to slučaj i sa aritmetičkim operatorima, JS omogućava poređenje vrednosti koje su različitog tipa. Kada se vrši poređenje String vrednosti sa Numeric vrednošću, JavaScript će konvertovati String vrednost u broj. Pritom se prazan string konvertuje u broj nula. String vrednost koja se ne može konvertovati u Numeric vrednost dobija vrednost NaN. Na kraju, u logičkim izrazima i izrazima poređenja NaN uvek predstavlja false.

Primeri u nastavku ilustruju poređenje vrednosti različitih tipova:

|  |  |
| --- | --- |
| 1  2  3  4 | 3 < "13" //true  3 < "Ben" //false  "3" > "13" //true  "2" == "12" //false |

Vrednost „13“ se konvertuje u broj. Broj 3 je manji od broja 13, pa je rezultat true.

Vrednost *Ben* se konvertuje u broj, ali konverzija ne uspeva, s obzirom da se Ben ne može konvertovati u broj; tako se *Ben* konvertuje u NaN. Izrazi poređenja sa NaN uvek vraćaju false.

Ipak, kada se porede dva stringa, kao u trećem i četvrtom primeru, takve vrednosti se ne konvertuju u Numeric, pa se poređenje obavlja na osnovu alfabetnog reda. Tako je karakter 3 u alfabetnom redu veći od broja 1, pa na kraju ispada da je vrednost 3 veća od vrednosti 13.

Poslednji, četvrti primer, ilustruje poređenje dve string vrednost. Samo u slučaju apsolutne podudarnosti poređenje će rezultirati vrednošću true.

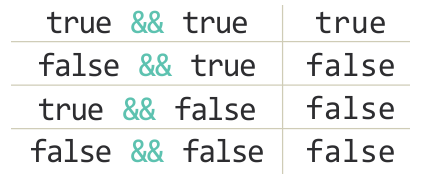
**Logički operatori**

Logički operatori se koriste kako bi utvrdili logičku povezanosti između promenljivih, odnosno vrednosti. Najčešće se koriste u kombinaciji sa malopre prikazanim operatorima poređenja. Tako se logičkim operatorima mogu kreirati različiti složeni logički izrazi, koji uvek moraju kao svoju finalnu vrednosti imati true ili false. Tabela 21.6 prikazuje logičke operatore.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Opis** | **Primer** | **Rezultat** |
| && | AND | (6 < 11 && 5 > 1)  (6 < 5 && 5 > 1) | True  false |
| || | OR | (6 < 5 || 5 > 1) | True |
| ! | NOT | !(6==6) | false |

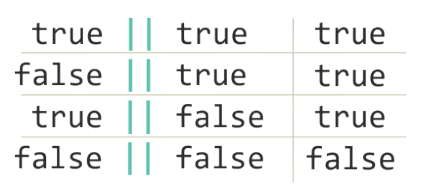
*Tabela 21.6 – Logički operatori u jeziku JavaScript*

Analizom primera upotrebe logičkih operatora, može se mnogo toga zaključiti. Operator AND zahteva ispunjenje svih navedenih uslova, kako bi proizveo vrednost true. U protivnom, proizvodi vrednost false. Slika 21.6 ilustruje različite situacije upotrebe logičkog operatora AND.



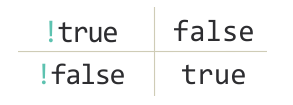
*Slika 21.6 – Logički operator AND*

S druge strane, OR operator zahteva ispunjenje samo jednog od uslova, kako bi kao svoju finalnu vrednost proizveo true (slika 21.7).



*Slika 21.7 – Logički operator OR*

Na kraju, operator NOT je operator negacije, koji true vrednost pretvara u false, a false u true, što je ilustrovano slikom 21.8



*Slika 21.8 – Logički operator NOT*

|  |
| --- |
| **Napomena**    *Logički operatori posebno važnu upotrebnu vrednost imaju prilikom formiranja uslova za kontrolu toka izvršavanja koda. Kontrola toka izvršavanja koda predmet je naredne lekcije.* |

**Ternarni operator**

Ternarni operator je obavezni operator svakog popularnijeg jezika. Ni JavaScript nije izuzetak. Ovaj operator je nešto teži za razumevanje, ali je u osnovi veoma jednostavan, i posebno koristan kada se jednom ovlada njegovom sintaksom.

Ternarni operator se koristi za dodelu vrednosti promenljivoj, ali na osnovu nekog predefinisanog uslova. Sintaksa ternarnog operatora je sledeća:

|  |  |
| --- | --- |
| 1 | myvariable = (condition) ? value1:value2 |

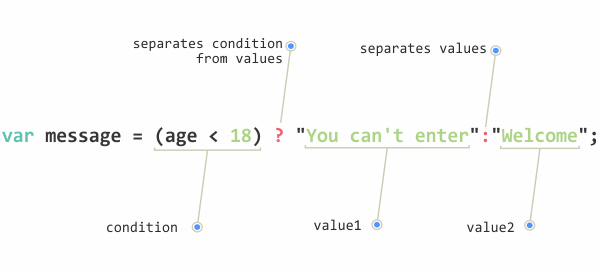
Napisano prostim jezikom, značenje navedene sintakse bi bilo: *ukoliko je* condition *ispunjen, promenjivoj* myvariable *dodeli* value1,*a u protivnom* value2.

Ternarni operator se gradi korišćenjem karaktera ? i :. Karakter upitnik (?) razdvaja uslov od mogućih vrednosti, a karakter dvotačka (:) razdvaja ponuđene vrednosti.

Primer upotrebe ternarnog operatora je sledeći:

|  |  |
| --- | --- |
| 1 | var message = (age < 18) ? "You can’t enter":"Welcome"; |

Prikazana linija koja sadrži ternarni operator može se raščlaniti na elemente prikazane na slici 21.9.



*Slika 21.9 – Ternarni operator*

Ukoliko je vrednost promenljive age manja od 18, promenljiva message dobija vrednost *You can’t enter*. U protivnom, promenljiva message dobija vrednost *Welcome*.

**String operatori**

Operator koji ima osnovnu funkciju sabiranja (+) može se u JavaScriptu koristiti i za nadovezivanje string vrednosti. Nadovezivanje stringova se drugačije naziva konkatenacija. Primer ilustruje nadovezivanje stringova:

|  |  |
| --- | --- |
| 1  2  3 | txt1 = "Java";  txt2 = "Script";  txt3 = txt1 + " " + txt2; |

Rezultat prikazanog koda je sledeći:

|  |  |
| --- | --- |
| 1 | JavaScript |

[**Povratak na vrh strane**](https://www.link-elearning.com/linkdl/preview/index.php?pk=beaf8efdcc1353ee6e5c01512d60e54b_ita_17&IDJedinica=1068970&c=jedinice)