Razlika promenljivih koje su deklarisane ključnom reči `var` u odnosu na one koje su deklarisane sa `let` je u oblasti važenja.

Promenljiva deklarisana sa **var** ima oblast važenja u celoj funkciji, a dostupna je u kodu i pre same deklaracije.

Promenljive deklarisane sa **let** nisu dostupne pre nego što se deklarišu a oblast važenja je blok koda označen vitičastim zagradama.

Oba su globalna ako su van funkcije.

```
function testFja1() {
    //vLet NIJE vidljiv
    for (let vLet = 0; vLet < 5; vLet++) {
        //vLet JE vidljiv
    }
    //vLet NIJE vidljiv
}
function testFja2() {
    //vGlb JE vidljiv
    for (var vGlb = 0; vGlb < 5; vGlb++) {
        //vGlb JE vidljiv
    }
    //vGlb JE vidljiv
}
```

Naredni primer prikazuje oblast važenja lokalno definisanih promenljivih:

```
// ime1, ime2, ime3: error
function fja() {
    // ime1: undefined; ime2, ime3: error
    // lokalne promenljive
    var ime1 = "Perica";
    let ime2 = "Perica";
    // automatski globalna
    ime3 = "Perica";
    // ime1,ime2, ime3: "Perica"
}
fja();
// ime1, ime2: error; ime3: "Perica"
```

Kada su deklarisane globalne promenljive njihov opseg važenja je identičan, osim što je promenljiva deklarisana sa var dobila vrednost *undefined* i pre definisanja vrednosti. Dakle:

```
// prezime1: undefinded; prezime2, prezime3: error
// globalne promenljive
var prezime1 = "P";
let prezime2 = "P";
prezime3 = "P";
function fja() {
    // prezime1, prezime2, prezime3: "P"
}
fja();
// prezime1, prezime2, prezime3: "P"
```

Preklapanje promenljivih

Promenljive u funkcijama mogu da svojim imenom preklope globalne. Pogledajte sledeći primer:

```
// Globalna var.
var pocetak = "Globalna ";

// Lokalna var. in fja1
function fja1() {
    var pocetak = "Lokalna ";
}

pocetak += "promenljiva. Nastavak zapocetog teksta";

document.write(pocetak); // Globalna promenljiva. Nastavak
zapocetog teksta
```

Važno. Dakle, treba obratiti pažnju da postoji bitna razlika u odnosu na neke druge programske jezike u kojima lokalna oblast važenja nastupa sa blokom vitičastih zagrada. U JavaScriptu cela funkcija predstavlja jednu oblast važenja pa promenljiva deklarisana sa **var** bilo gde u funkciji ima oblast važenja u celoj funkciji. Da bi se doprinelo jasnoći koda obično se promenljive deklarišu na početku funkcije.

```
function fja1() {
    if (true)
    {
        var pocetak = "pocetak";
    }
    var novipocetak = "Novi " + pocetak;
```

```
    return novipocetak;
}
document.write(fja1());
```

Funkcije mogu sadržati ne samo promenljive već i druge funkcije – **ugnježdene funkcije**. Oblast važenja ugnježdenih funkcija je takođe ugnježdena. Na primer:

```
function myFunction() {
    var povrsina =
document.getElementById("povrsinaZida").value;
    var dimX = document.getElementById("dimXcm").value;
    var dimY = document.getElementById("dimYcm").value;
    alert("Potrebno je kupiti: " + brojPločica(dimX, dimY,
povrsina));
}
function brojPločica(p_Xcm, p_Ycm, pov) {
    return pov * 1.05 / povPlocice(p_Xcm, p_Ycm);
    function povPlocice(Xcm, Ycm) {
        return Xcm * Ycm * 0.0001;
    }
}
```

Životni vek promenljivih u JS počinje sa njihovom deklaracijom.
Lokalne promenljive se brišu kada se funkcija, u kojoj su definisane, izvrši.
Globalne promenljive se brišu kada se zatvori prozor (tab) veb čitača, ali ako se otvara nova stranica u istom prozoru ostaju dostupne.
Promenljive koje su definisane kao argumenti funkcije ponašaju se kao lokalne promenljive unutar funkcije.

Promene na elementima i atributima

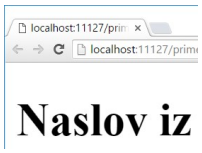
Koristeći objekat document, JS pristupa sadržaju HTML stranice. Već smo u primerima koristili metodu **`getElementById()`** koja omogućava preuzimanje kontrole nad određenim elementom HTML dokumenta i rad sa njim. Element se identifikuje pomoću atributa **`id`**. Ono što ova metoda vraća je objekat sa pratećim svojstvima, pomoću koji se može izvršiti nekoliko tipičnih operacija:

1. promena sadržaja
2. promena atributa

3. promena važećeg stila
4. provera ispravnosti sadržaja (validacija)

Promena sadržaja

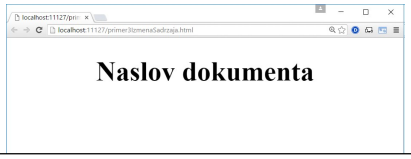
```
<!DOCTYPE html>
<html>
<body>
  <h1 id="naslov">Naslov dokumenta</h1>
  <script>
    document.getElementById('naslov')
            .innerHTML = 'Naslov iz JS!';
  </script>
</body>
</html>
```



Promena atributa

Pošto se element identifikuje, atributu se pristupa imenom atributa koji sledi nakon tačke. Dakle, ime atributa je svojstvo vraćenog objekta.

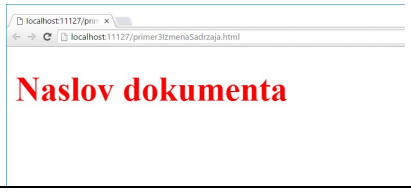
```
<!DOCTYPE html>
<html>
<body>
  <h1 id="naslov">Naslov dokumenta</h1>
  <script>
    document.getElementById('naslov').align = 'center';
  </script>
</body>
</html>
```



Promena tj. definisanje stila

Koristi se ključna reč `style` nakon koje sledi stil koji se primenjuje. Pristupanje stilu nekog elementa slično je atributu tj. radi se o svojstvu objekta elementa, a pristup određenom stilu se vrši kao svojstvo objekta stila.

```
<!DOCTYPE html>
<html>
<body>
    <h1 id="naslov">Naslov dokumenta</h1>
    <script>
    document.getElementById('naslov').style.color
        = 'red';
    </script>
</body>
</html>
```



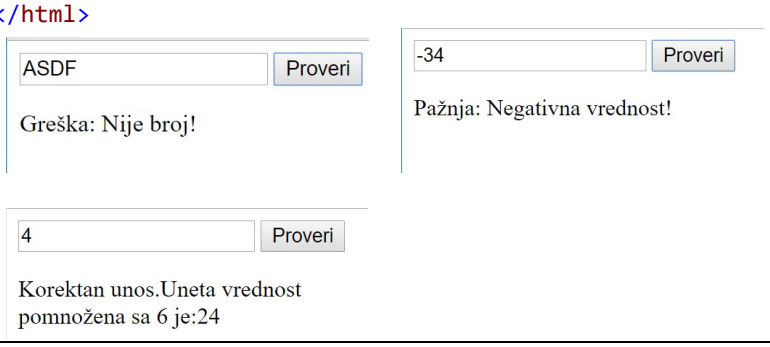
Validacija podataka

Validacija podataka znači proveru ispravnosti unetih podataka. Validacija se odnosi na HTML elemente koji omogućavaju unos podataka i primenjuju se na HTML formama. JavaScript ima jaku podršku za validaciju podataka. Na ovom mestu upoznaćemo se sa osnovnim pristupom.

Vrednost elementa za unos podataka dobija se preko svojstva `value`. Isto svojstvo može se koristiti i pri izmeni vrednosti.

```
<!DOCTYPE html>
<html>
<body>
  <input id="arg1" placeholder="unesi vrednost" />
  <button type="button" onclick="proveriArg1()">
  Proveri</button> <br/>
  <p id="poruka">...</p>
  <script>
   function proveriArg1() {
     console.log("provera");
     var ctrPoruka = document.getElementById("poruka");
     var temp = document.getElementById('arg1').value;
      if (isNaN(temp))
        ctrPoruka.innerHTML = "Greška: Nije broj!";
```

```
      else if (temp < 0)
        ctrPoruka.innerHTML = "Pažnja: Negativna
 vrednost!";
      else
        ctrPoruka.innerHTML = "Korektan unos." +
        "Uneta vrednost pomnožena sa 6 je:" +
                                        temp * 6;
    }
  </script>
</body>
</html>
```



Funkcije kao vrednosti

Funkcija se može samostalno koristiti i kao promenljiva i tako prosleđivati drugim funkcijama ili svojstvima nekog objekta, na primer:

```
function knjiga(naslov, autor, fjacene) {
    this.naslov = naslov;
    this.autor = autor;
    this.racunajCenu = fjacene;
}
function knjigaCena(osnovica)
{
    this.cena = osnovica*0.95;
}
var primerak1 = new knjiga("Pesme", "Ršumović",
knjigaCena);
```

```
primerak1.racunajCenu(300);
```

Funkcija može biti imenovana ili anonimna.

```
var x = function (osnovica) {   // anonimna
    this.cena = osnovica * 0.95;
}
ili
var x =(osnovica)=>this.cena = osnovica * 0.95; // anonimna

var x1 = function cenaOdOsnovice_Imenovana(osnovica) {
    this.cena = osnovica * 0.95;
}
x(100);
console.log(cena);
x1(200);
console.log(cena);
```

Anonimna funkcija nema ime, ali se u praksi često koristi. Na primer, može se pridružiti nekoj promenljivoj koja ima osobine promenljive, ali istovremeno se koristi kao funkcija, kao argument za drugu funkciju ili samo za dobijanje vrednosti koju funkcija vraća.
Funkcija se može pisati tako da se njeno izvršavanje izvede odmah.

```
(function () {
    console.log("dobijena cena je: " + cena);
})();
```

ili

```
var cena = 999.9;
( () => console.log("dobijena cena je: " + cena) )();
```

Objekti

Ponovimo još jednom. U JavaScript jeziku sve promenljive su objekti osim prostih tipova. Prosti tipovi su: string, number, boolean, null i undefined. Čak se string, boolean i number mogu kreirati i koristiti kao objekti: `var tmp=new String();`

Jedan objekat može da sadrži više promenljivih koje same mogu biti prostog tipa ili objekti. Promenljive u objektu imaju svoj naziv, a odgovarajuća vrednost se pridružuje koristeći naziv. Kaže da objekti predstavljaju kolekciju imenovanih vrednosti ili parova `name:value`. Ovi parovi se nazivaju još i svojstvima objekata. Osim svojstava objekti mogu da sadrže i metode tj. pripadajuće funkcije. Na primer:

```
<!DOCTYPE html>
<html>
<body>
    podaci o osobi:
    <p id="osoba1"></p>
  <script>
    var osoba = {
        ime: "Perica",
        prezime : "P",
        id: 666,
        "adresa stanovanja": "Nepoznata",
        // this!!!
        toString : function() {
           return this.prezime + " " + this.ime;
        }
    };
    document.getElementById("osoba1").innerHTML =
osoba.toString();
  </script>
</body>
</html>
```

this

Zapazite da se pristup elementima objekta iz metode u objektu obavlja pomoću ključne reči `this`. Ključna reč `this` označava tekući (nadređeni) objekat. Ako se ključna reč `this` koristi u objektu označava taj objekat, ako se koristi u funkciji označava nadređeni objekat funkcije. U prethodnom primeru objekat `this` je iskorišćen za pristup svojstvima objekta iz jedne funkcije istog objekta:

```
return osoba.prezime + " " + osoba.ime;
ili
return this.prezime + " " + this.ime;
```

Pristup do vrednosti jednog svojstva vrši se pomoću imena svojstva: `obj[imeSvojstva]` ili `obj.imeSvojstva`. Pristup do svih svojstava može da se izvrši pomoću petlje `for.in`: `for(let p in objekat){.}`.

Na primer:

```
for (let key in osoba) {
    alert(key + ': ' +osoba[key]);
}
```