

ITERATORI I GENERATORI



POWERED BY



COMTRADE

ITERATORI I GENERATORI

- ***“Iterable protokol”*** je dogovoren API na nivou JavaScript-a kao jezika, za kreiranje objekata koji se mogu koristiti za iteraciju nad kolekcijom podataka.
- Ključna karakteristika ovoga protokola je **“sekvencijalanost”** tj. osobina da pri iteraciji vraća vrednost iterabilne strukture jednu po jednu.
- Tipovi podataka koji zadovoljavaju *iterable protokol* se nazivaju **“iterable kolekcije”**.
- Poštujući iterable protokol potrebno je da funkcija vraća objekat koji sadrži specijalnu metodu **“next()”**.



POWERED BY



COMTRADE

ITERATORI I GENERATORI

- **Iterable protokol omogućava programeru da definiše ponašanje nekog objekta prilikom iterativnog poziva.**
- Drugim rečima, moguće je definisati iterativno ponašanje i podatke koje objekat vraća prilikom svake iteracije, čak i za objekte koji inače nemaju ovo ponašanje ugrađeno.
- Isto tako, moguće je promeniti rezultate iteracije i onim objektima koji već imaju ovo ponašanje



POWERED BY



COMTRADE

ITERATORI I GENERATORI

- Objekti kroz koje je moguće vršiti iteracije imaju (ili nasleđuju) interni **@@iterator** metod.
- Ovom metodu ne pristupamo direktno, već preko konstante **Symbol.iterator**.
- Ovo svojstvo objekta je referenca na **iteratorsku funkciju**

```
obj[Symbol.iterator] = function() {...};
```



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Iteratorska funkcija je specifična samo po tome što vraća rezultat prema **iterator protokolu**.

Ovaj protokol prosto zahteva da najpre **funkcija kreira objekat koji zovemo "iterator"** i takođe služi da **obezbedi closure** za iteratorov metod **next()** koji se poziva u svakoj iteraciji.

Dakle **iterator** ima obavezno makar **metod next()**, od koga se očekuje da **pri svakom pozivu obezbedi (nov) rezultat u tačno određenoj formi**.

Drugim rečima, **podatak** koji tražimo u svakoj iteraciji, ne dobijamo samo tako, kao "suv podatak" već se nalazi **"spakovan" unutar objekta**, koji u stvari predstavlja **vraćeni rezultat metoda next()**.

Ovaj objekat ima dva svojstva:

- **done** - logička vrednost; ako je *false* (ili ne postoji), znači da ima još podataka, dok vrednost *true*, označava da podataka više nema
- **value** - sam podatak, ne mora da se navede ako je *done == true*



POWERED BY



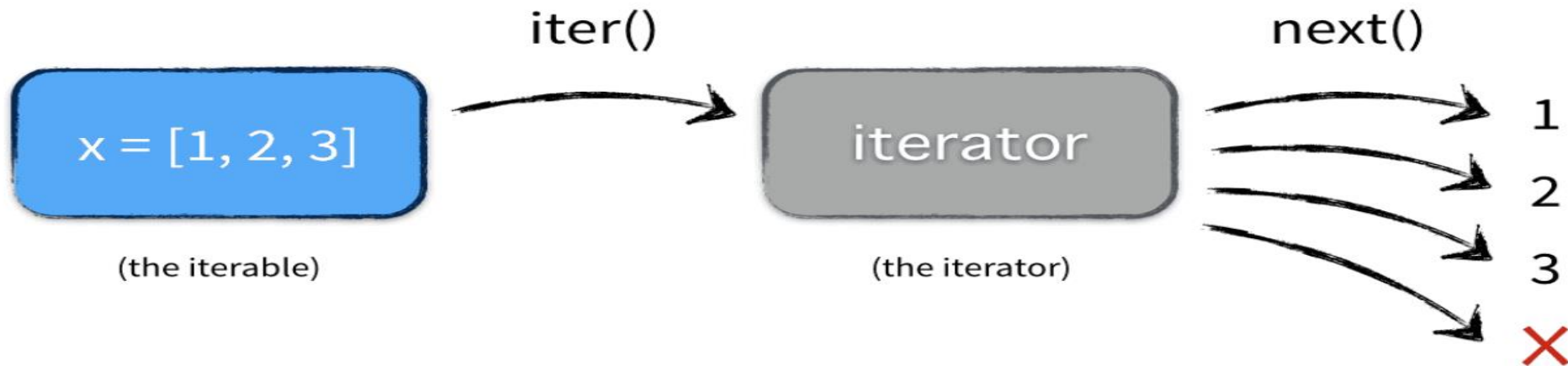
COMTRADE

ITERATORI I GENERATORI

```
{  
  [Symbol.iterator]() {  
  
    // Telo funkcije koje vraća iterator objekat  
  
  }  
}
```

Iterator objekat nastaje pozivajući metodu iterable objekta pod nazivom: "[Symbol.iterator]()".

```
const noviIterator = nekiIterableTipPodataka[Symbol.iterator]();
```



ITERATORI I GENERATORI

```
// Iterabilna struktura podataka
```

```
const nekiNiz = ['a', 'b', 'c'];
```

```
// Kreiranje Iterator objekta sa iterable metodom koristeći "Symbol.iterator" ključ
```

```
const nekiIterator = nekiNiz[Symbol.iterator]();
```

```
// Pristupanje članovima itrable kolekcije
```

```
nekiIterator.next() // { value: 'a', done: false }
```

```
nekiIterator.next() // { value: 'b', done: false }
```

```
nekiIterator.next() // { value: 'c', done: false }
```

```
nekiIterator.next() // { value: undefined, done: true }
```



POWERED BY



COMTRADE

ITERATORI I GENERATORI

```
function iteratorska_funkcija() {  
    ...  
    // iteratorska funkcija vraća iterator  
    return {  
        next: function() {  
            // iteratorov metod next() vraća objekat sa podatkom  
            return {  
                done: logička_vrednost,  
                value: podatak  
            };  
        }  
    };  
}
```

Primer 1 - Iteratori.html



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Kada želimo da **iterativno prođemo kroz neki objekat**, možemo koristiti specijalan tip ciklusa **for..of**.


Za razliku od **for..in** ciklusa koji nam služi da "nabrojimo" svojstva nekog objekta, **for..of** petlju možemo koristiti **samo nad objektom koji ima definisano iterativno ponašanje**.

Na primer, **nizovi** i **stringovi** su po defaultu iterable objekti, pa tako nad njima možemo koristiti i **for..of** ciklus.

Rezultati, odnosno podaci koje dobijamo u ovom ciklusu će zavisiti od samog objekta. **Array objekat**, po definiciji, kroz svaku iteraciju vraća po jedan elemenat niza.

Sa druge strane, objekat **String** kroz svaku iteraciju vraća po jedan znak stringa.

Evo kako se zadaje **for..of** petlja:



```
let iterator = obj[Symbol.iterator]();  
  
do {  
    let rezultat = iterator.next();  
    if (!rezultat.done)  
        console.log( rezultat.value );  
} while (!rezultat.done);
```

✗

```
for (let podatak of obj) {  
    console.log( podatak );  
}
```



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Zahvaljujući iteratoru i metodi next() je omogućena iteracija kroz iterabilnu kolekciju za for..of petljom:

```
1 // Iterabilna struktura podataka
2 const nekiNiz = ['a', 'b', 'c'];
3
4 // Kreiranje Iterator objekta sa iterable metodom koristeći "Symbol.iterator" ključ
5 const nekiIterator = nekiNiz[Symbol.iterator]();
6
7 for (let n of nekiIterator) {
8   console.log(n)
9 }
10 // a
11 // b
12 // c
```



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Generatorska funkcija može prekinuti svoje izvršavanje i kasnije ga nastaviti "po pozivu".

Tako ova funkcija može raditi "u nastavcima". Interpreter zna da je neka funkcija generatorska, ako je obeležena simbolom *

```
DEKLARACIJA FUNKCIJE: function* fun(parametri) {...}  
FUNKCIJSKI IZRAZ:      function* (parametri) {...}  
METOD OBJEKTA:        ..., fun: function* (parametri) {...}  
METOD OBJEKTA:        ..., * fun(parametri) {...}
```

Kako funkcioniše ta stvar sa prekidanjem i nastavljanjem izvršavanja funkcije? Svi znamo da funkcija vraća a rezultat direktivom **return**.

Međutim, u generatorskim funkcijama se može koristiti i specijalna direktiva **yield**, koja takođe **vraća vrednost i prekida rad funkcije**, ali isto tako **predstavlja tačku kasnijeg povrata i nastavka rada funkcije**.



POWERED BY



COMTRADE

ITERATORI I GENERATORI

U generatorskoj funkciji **možemo imati više puta zadatu direktivu yield.**

Pošto suštinski **generatorska funkcija predstavlja poseban tip iteratorske funkcije**, i ona funkcioniše po iterator protokolu.

To znači da **pozivanjem generatorske funkcije dobijamo generator objekat** (koji je praktično specijalna verzija iteratora), sve sa **next()** metodom koju možemo uzastopno pozivati.

Svaki **generator.next()** poziv nas "vraća" u generatorsku funkciju na poziciju posle poslednje **yield** direktive.



POWERED BY



COMTRADE

ITERATORI I GENERATORI

```
function* generatorska_funkcija(parametri) {  
    ...  
    yield vrednost;  
    ...  
    yield vrednost;  
    ...  
}  
  
let generator = generatorska_funkcija(parametri);  
  
generator.next().value;  
generator.next().value;  
...
```



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Kao i svaka druga funkcija, i **generatorska može imati direktivu return**, s tim što posle izvršavanja **return**, **nema više vraćanja u funkciju**.

*yield izraz može imati vrednost. Znači, pri povratku u funkciju, možemo da joj "dobacimo" neku vrednost kroz parametar metoda **next()**, a ta vrednost će biti istovremeno i vrednost poslednjeg izvršenog **yilda**.*

yield može koristiti samo unutar generatorske funkcije



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Osim **next()**, generator ima **još dva metoda** za poziv, odnosno povratak u generatorsku funkciju. To su pomenuti **return()** i **throw()**.

Metod **return()** nas izbacuje iz generatora, kao da je u funkciji momentalno naleteo na **return**. Vraća objekat **{done:true, value:undefined}**. Posle toga, nema više iteracija.

Metod **throw()** radi to isto, s tim što ne vraća rezultat već podiže **izuzetak** (*exception*), kao da smo naleteli na grešku u izvršavanju programa pri pozivu metoda **next()**.

Kao da to nije dosta, metod **next()** iteratorskog objekta može imati i **neku vrednost zadatu kao parametar**. Ova vrednost se tada prosleđuje generatoru na mestu gde se nalazio **poslednji izvršeni yield**, što onda postaje vrednost **yield** izraza.

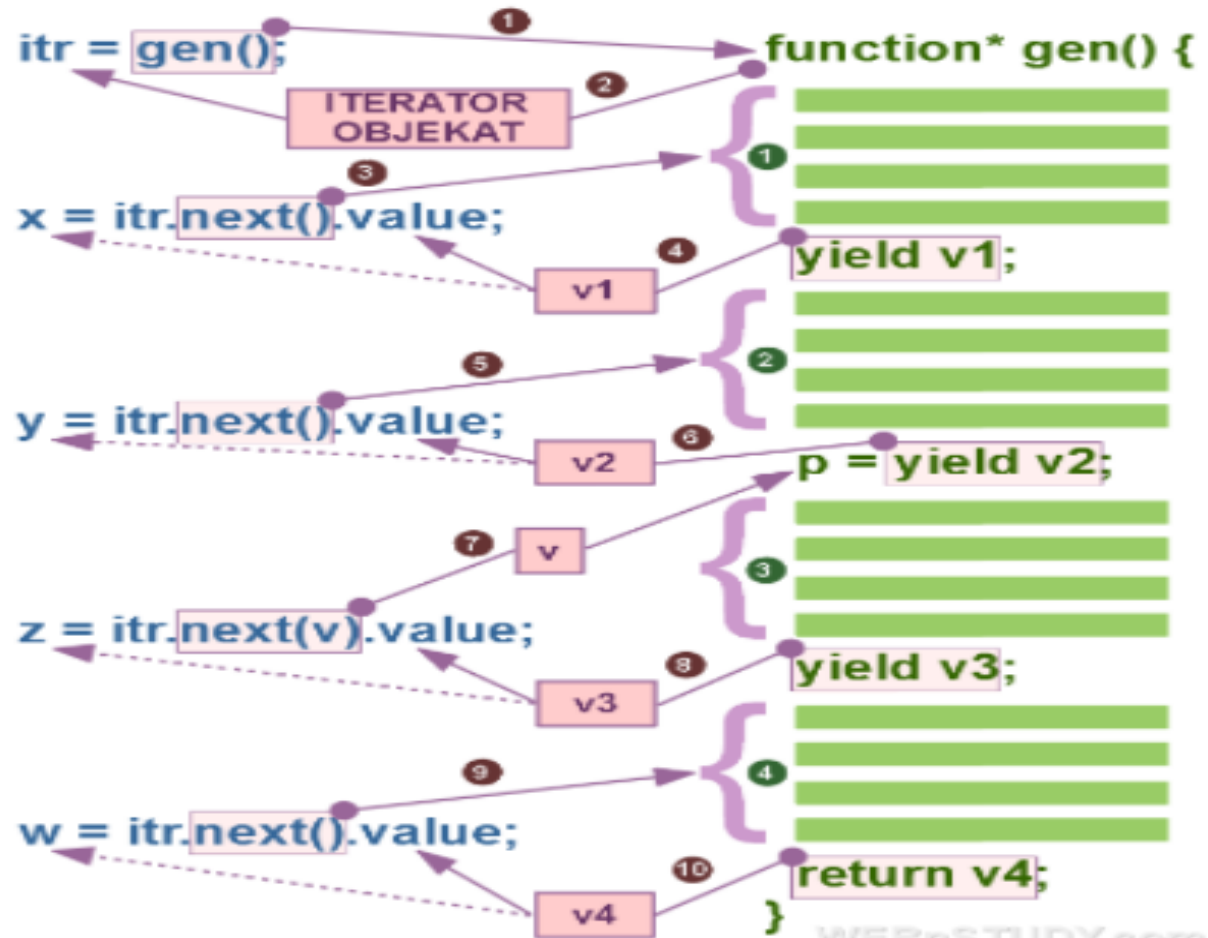


POWERED BY



COMTRADE

ITERATORI I GENERATORI



POWERED BY

COMTRADE

ITERATORI I GENERATORI

1. Na početku **pozivamo generatorsku funkciju `gen()`**.
2. Rezultat tog poziva je **generator objekat**, na koji će pokazivati referenca **`itr`**.
3. Sada koristimo metod **`next()`** objekta **`itr`** kako bismo **izvršili prvi deo koda iz generatorske funkcije**.
4. Kada naiđe na **`yield`**, "iskače" iz funkcije, a kao **rezultat se vraća vrednost `v1`**, koja se nalazi unutar objekta i to pod svojstvom **`value`**, što se smešta u promenljivu **`x`**.
5. Ponovo pozivamo metod **`next()`** generatora, čime se **vraćamo u generatorsku funkciju i nastavljamo sa drugim delom koda**.
6. Kada stigne do sledećeg **`yield`**, ponovo će prekinuti izvršavanje funkcije i vratiće vrednost **`v2`**, koja će na kraju završiti u promenljivoj **`y`**.
7. Ponovo pozivamo metod **`next()`** kako bismo nastavili izvršavanje generatora, ali **ovaj put prosledujemo i neku vrednost `v`**, koja će **predstavljati vrednost poslednjeg `yield` izraza** i biće smeštena u lokalnu promenljivu **`p`**, da bi se odmah potom **nastavilo izvršavanje trećeg dela funkcije**.
8. Ponovo nailazi na **`yield`**, prekida funkciju i vraća vrednost **`v3`**, koja se smešta u promenljivu **`z`**.
9. Poslednji put pozivamo metod **`next()`**, da bi se **vratio u funkciju i izvršio četvrti deo koda**.
10. Konačno, **generatorska funkcija vraća poslednju vrednost `v4` direktivom `return`**, koja se smešta u promenljivu **`w`**. Posle **`return`** (ili kad se stigne do kraja funkcije) nema više povratka u funkciju, gotovo je sa iteracijama. Svaki sledeći poziv **`next()`** metode vratiće vrednost **`undefined`**



POWERED BY



COMTRADE

ITERATORI I GENERATORI

Materijal na sledecoj stranici:

<http://www.webnstudy.com/tema.php?id=js-iteratori-i-generatori>

Dodatni materijal na stranici:

<https://www.webprogramiranje.org/iteratori-generatori/>



POWERED BY



COMTRADE