Računarski praktikum 2
Vježbe 3 – dodatak

Napravili:

Antonia Grbić i Ante Pocedulić

Javascript i petlje

- Dosad smo se susreli s klasičnim pisanjem for petlje

```
for (statement 1; statement 2; statement 3)
 {
   kod koji se izvršava
 }
```

- Uz to smo spominjali i for/in petlju

```
For(element in set_of_elements)
{
  kod koji se izvršava
}
```

Javascript i petlje

- Jedan od prvih problema s kojim se susretnemo pri web developmentu jest sporo loadanje sadržaja web stranice.
- Što biste naveli kao moguće uzroke za sporo loadanje stranice?
- Uzroci mogu biti različiti: Neoptimizirane slike, mnogo Flash-a na web stranici, “glomazan” kod (mnogo externog css-a i javascripta), table-based layout,...
- Jedan od razloga jest i neoptimalan Javascript kod.
- Koji su razlozi neoptimalnog Javascript koda?

Optimalne i neoptimalne petlje

- Jedni od glavnih razloga su:
  - U petlji se procesira mnogo javascipt koda
  - Petlja mora izvesti mnogo iteracija
  - Mnogo rekurzija
- Općenito, Javascript ne poznaje threadove i sav kod se izvodi sinhrono – prema tome, vrijeme izvršanja Javascript koda je direktno vezano uz broj iteracija koje se moraju “izvrtiti”.
- Ukoliko pricamo o neoptimalnom javascript kodu, dva su najcesca razloga zasto bismo zavrsili s blokiranim browserom i mi cemo se fokusirati na njih:
  - Unutar tijela petlje se nalazi preglomazan kod
  - Petlja mora izvesti mnogo iteracija uz ne nužno glomazan kod u tijelu petlje.

Optimalne i neoptimalne petlje

- Problem možemo riješiti postavljajući si dva pitanja:
  - Da li se petlja mora izvršavati sinhrono?
  - Da li je bitan poredak izvršavanja unutar petlje?
- Ukoliko nemamo kod koji se izvršava direktno poslije petlje, a koji ovisi o rezultatu iz te petlje, onda je odgovor na prvo pitanje: NE!
- Odgovor na drugo pitanje bi bio NE u situaciji poput

```
for(var i = 0; i<array.length;i++){
  procesiraj(array[i]);
}
```

- Očigledno i-ta iteracija ne ovisi o nekoj j-toj iteraciji (za i različito od j), pa prema tome zaključujemo da nije bitan poredan izvršavanja unutar petlje.
- Ukoliko je odgovor NE na oba pitanja, tada si možemo dozvoliti određeno manerviranje petlje u svrhu optimizacije.

Optimalne i neoptimalne petlje

- “Stari način”

```
var arr = new Array(“Jedan",“Dva",“Tri");
for(var i=0; i<arr.length; i++) {
   var vrijednost= arr[i]; alert(i =") "+vrijednost);
}
```

- Ovo je najkorišteniji i najčitljivi oblik u kojemu možemo naći for petlju.
- Problem s ovom petljom jest da se u svakoj iteraciji mora pročitati varijabla i, računati duljina niza, evaluirati , testirati da li je i<length istinito, te inkrementirati varijabla i.

Optimalne i neoptimalne petlje

- Pisanje petlje koristeći cachiranje

```
var len=arr.length;
for(var i=0; i<len; i++) {
    var vrijednost= arr[i];
    alert(i =") "+vrijednost); }
```

- Ova verzija je mnogo bolja od prethodne jer cachira duljinu niza i prema tome štedi vrijeme izvršavanja, jer javascript sada više ne mora svaki put tražiti duljinu niza.
- NAPOMENA: Cachiranje duljine niza se može koristiti isključivo ukoliko se duljina niza ne mijenja tijekom izvršavanja petlje.

Optimalne i neoptimalne petlje

- “Revertirana” for petlja

```
for(var i=arr.length-1; i>=0; i--) {
    var vrijednost= arr[i];
    alert(i =") "+vrijednost); }
```

- Ova metoda se ne može koristiti ukoliko je bitan poredak izvršavanja petlje.

```
for(var i=arr.length-1; i--;) {
   var vrijednost= arr[i];
   alert(i =") "+vrijednost); }
```

- Javascript dozvoljava i ovakve zapise što je još i bolje rješenje.
- Samo jednom se izračuna duljina niza, te se nadalje samo izvode dvije operacije: provjeravanje da li je varijabla i dosegla nulu, te dekrementiranje varijable i.

Optimalne i neoptimalne petlje

- Prethodno opisano se može napisati u čitljivijem obliku koristeći while petlju

```
var i = arr.lenght;
while(i--) {
var vrijednost= arr[i];
    alert(i =") "+vrijednost); }
```

- Bitno je napomenuti da while petlja u javascriptu iterira dok varijabla uvjeta ne postane false, 0, prazni string,...
- U ovakvom zapisu for petlje se koristi ta činjenica.
- Ovako zapisana for petlja (u obliku while petlje) se pokazala najboljim izborom.

Sinhrono i asinhrono izvršavanje Javascript koda

- Vratimo se na prvo pitanje:
  - Da li se petlja mora izvršavati sinhrono?
- Browser window ima samo jedan thread koji procesira DOM objekte, na njega se vežu svi događaji, te se na njemu izvršava sav Javascript kod.
- Taj thread procesira jednu po jednu jedinicu koda, što znači – dok se jedna stvar procesira, sve druge čekaju.

Sinhrono i asinhrono izvršavanje Javascript koda

- Takvo ponašanje zna biti veoma neinuitivno, te nerijetko dovodi do blokiranog web browsera.
- Javascript funkcije koje predugo traju uvjerljivo najviše ubijaju user experince zbog efekta blokiranja browsera i nemogućnosti daljnjeg korištenja web stranice.
- Javascript funkcije koje predugo traju će u jednom trenutku dovesti do timeout-a, posebno na mobilnim browserima.
- Problem se može rješiti na dva načina – korištenjem setTimeout funkcije, te korištenjem async tag-a koji još nije u potpunosti podržan na svim browserima.
- Mi ćemo se zato koncentrirati na rješenje koje koristi setTimeout funkciju i demonstrirati na jednom primjeru.