JavaScript

Deveti dio

Pregled

- Closure, drugi dio
  - Shared scope
  - Nested scope
  - Petlje i blokovi
- Module pattern
- PP
- ByRef, ByVal
- Errors
- Dates

Obnavljanje i napomene

- Selidba
- CSS, dva dodatna predavanja
- Osnovni model
  - Thread of execution
  - Memory/variable environment
  - Call stack
- Prošireni model
  - Web Browser APIs background threads
  - Callback/Message queue
  - Event loop

Još primjera

```
1 function foo() {
2     var bar = "bar";
3
4     setTimeout(function() {
5         console.log(bar);
6     },1000);
7 }
8
9 foo();
```

```
1 function foo() {
2     var bar = "bar";
3
4     $("#btn").click(function(evt) {
5         console.log(bar);
6     });
7 }
8
9 foo();
```

Closures, shared scope

```
1  function foo() {
2      var bar = 0;
3
4      setTimeout(function(){
5          console.log(bar++);
6      },100);
7      setTimeout(function(){
8          console.log(bar++);
9      },200);
10 }
11
12 foo();  // 0 1
```

Closures, nested scope

```
1  function foo() {
2      var bar = 0;
3
4      setTimeout(function(){
5          var baz = 1;
6          console.log(bar++);
7
8          setTimeout(function(){
9              console.log(bar+baz);
10         },200);
11     },100);
12 }
13
14 foo();  // 0 2
```

Closures, petlje i blokovi

```
for (var i=1; i<=5; i++) {
    setTimeout(function(){
        console.log("i: " + i);
    },i*1000);
}
```

```
for (var i=1; i<=5; i++) {
    (function(i){
        setTimeout(function(){
            console.log("i: " + i);
        },i*1000);
    })(i);
}
```

```
for (var i=1; i<=5; i++) {
    let j = i;
    setTimeout(function(){
        console.log("j: " + j);
    },j*1000);
}
```

```
for (let i=1; i<5; i++) {
    setTimeout(function(){
        console.log("i: " + i);
    },i*1000);
}
```

Closures, zaključak

- Asinhroni callbacks, Web APIs, Callback Queue i Event loop omogućavaju odlaganje akcija dok se ‘posao’ (npr. API request, timer, itd.) ne završi, a da se pri tome naš JS i dalje izvršava liniju po liniju
- Asinhronost je oslonac modernog Web-a koji omogućava kreiranje neblokirajućih (non-blocking) aplikacija
- U nekim od narednih časova još detaljnije ćemo raditi asinhrono programiranje gdje ćemo uvesti još novih pojmova:
    - Asinhrone funkcije (async/await)
    - Promises
    - API requests
        - Ajax, Axios/Fetch

Module pattern

- Za JSu postoji razni dizajn paterni. Pod paternom možete podrazumijavati način/strukturu pisanja koda. Tokom razvoja jezika javljali su se novi paterni.
- Jedan od najbolje prihvaćenih je module pattern
- Postoje razne knjige koje se bave JS patternima, a jedna od njih se nalazi na sledećem linku
  - Link
- Obnavljanje

```
var foo = {
    o: { bar: "bar" },
    bar() {
        console.log(this.o.bar);
    }
};

foo.bar();      // "bar"
```

Module pattern, primjeri

```
var foo = (function(){

    var o = { bar: "bar" };

    return {
        bar: function(){
            console.log(o.bar);
        }
    };

})();

foo.bar();      // "bar"
```

```
var foo = (function(){
    var publicAPI = {
        bar: function(){
            publicAPI.baz();
        },
        baz: function(){
            console.log("baz");
        }
    };
    return publicAPI;
})();

foo.bar();      // "baz"
```

Modul patterns, modern (ES6)

Foo.js

```
1 var o = { bar: "bar" };
2
3 export function bar() {
4     return o.bar;
5 };
```

```
1 import { bar } from "foo.js";
2
3 bar();           // "bar"
4
5 import * as foo from "foo.js";
6
7 foo.bar();       // "bar"
```

- Imamo predviđena predavanja za ES6

Pair Programming

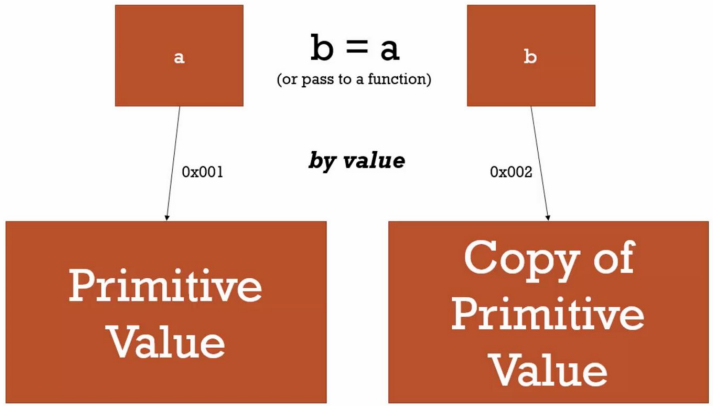
- Odraditi do kraja
  - http://csbin.io/closures

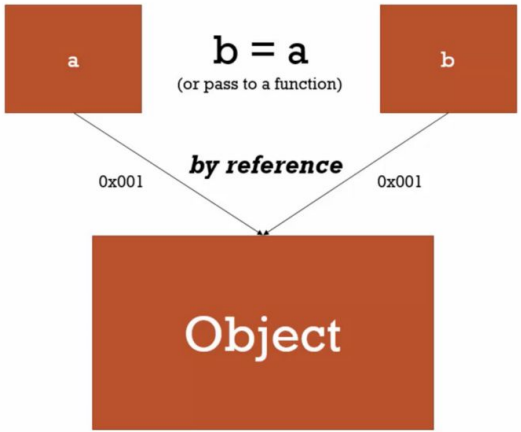
Closures, obnavljanje

- Šta je closure i kako se kreira?
- Koliko scope ostaje “živ”
- Kako se kreira closure za kreiranje encapsulated modula
  - Koji je benefit
- Primjer(code-1)

By value vs By reference

- Prosleđivanje može biti po vrijednosti i po referenci. Svi primitivni tipovipodataka su prosleđuju po vrijednosti, a svi objekti se prosleđuju po referenci





Primjeri

- Koji tipovi podataka spadaju u primitivne tipove?
- Objekte ćemo raditi mnogo detaljnije kada budemo radili OOP
- Primjer (code-2)

Errors

- Kao i većina programskih jezika, tako i JS sadrži mogućnost obrade grešaka
- Četiri ključna izraza za obradu grešaka su
  - try - testirao dio bloka ima li grešaka
  - catch - omogućava obradu grešaka (error handler)
  - throw - kreiranje custom grešaka
  - Finally - izraz koji omogućava izvršavanje koda nakon try i catch bloka, bez obzira na rezultat
- Tokom razvoja aplikacije greška će vam se sigurno javiti u određenom momentu
- Primjeri (code-3)

```
try {
    Block of code to try
}
catch(err) {
    Block of code to handle errors
}
finally {
    Block of code to be executed regardless of the try / catch result
}
```

Errors, nastavak

- Error objekat obezbjeđuje informacije o erroru koji je nastao, a sadrži dva property-a
  - **name** - setuje ili vraća ime greške
  - **message** - setuje ili vraće error poruku

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

- Za više informacije (neke od primjera) pogledati link

Dates

- Datumi u JSu su predstavljeni kao objekti
- Kreiraju se sa **new Date(args)**
  - Predstavlja broj milisekundi od 1. Januara 1970
- Po default-u JS koristi browser time zone i prikazuje podatke kao full text string
  - Isprobajte u console
  - U nastavku ćemo vidjeti koji sve načini formatiranja datuma postoje
- U new Date(), Date konstruktor može da ima do 7 argumenata i to definisanih ovim redosledom
  - godina, mjesec, dan, sat, minut, sekund, milisekund
  - Mjeseci počinju od 0 (0 - 11)
  - Napomena:
    - Ako stavite jedan argument, taj argument će da predstavlja broj milisekundi, a ne godinu
  - new Date(1000000) - milion milisekundi od 1. Januara 1970 (minus ispred znači nazad milion miliseknd od 1. Januara 1970)

Dates, metodi i formati

- Postoji više metoda na Date objektima, a neki od njih:
  - toString() -konvertuje datum u string (ovaj metod se poziva po defaultu)
  - toUTCString() - konvertuje datum u string na 0 vremensku zonu
  - toDateString() - konvertuje datum u čitljiviji format
- ISO jedini standardizovan, ostali zavise od pretraživača

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"

Dates, formati (nastavak)

- **ISO date format** je dosta zastupljen
  - ISO 8601 sintaksa
    - (YYYY-MM-DD) - ("2015-03-25")
      - U zavisnosti od vremenske zone rezultat će biti 24. Ili 25. Mart
    - (YYYY-MM) - ("2015-03")
      - U zavisnosti od vremenske zone rezultat će da varira između 28./29. Februara i 1. Marta
    - (YYYY) - (“2015”)
      - U zavisnosti od vremenske zone rezultat će da varira između 31. Decembra 2014. I 1. Januara 2015
    - (YYYY-MM-DDTHH:MM:SSZ) - ("2015-03-25T12:00:00Z")
      - T separator za vrijeme i datum
      - Z definiše UTC vrijeme
      - I ovo je moguće link

Dates, formati (nastavak 2)

- Pri setovanju datuma, kada se ne definiše vremenska zona, JS koristi vremensku zonu pretraživača
  - Drugim riječima ako se nalazite u GMT zoni, datum i vrijeme će biti konvertovani u CDT zonu ako je na browseru podešena CDT zona
- **Short Dates** imaju format “MM/DD/YYYY”
  - Format (YYYY-MM-DD) na nekim pretraživačima može da bude problematičan ako se ne stave vodeće nule za mjesec i datum
  - Format (YYYY/MM/DD) je nedefinisan format, neki pretraživači će pokušati da pogode datum, dok će drugi da vrate NaN
  - Format (DD-MM-YYYY) je nedefinisan format, neki pretraživači će pokušati da pogode datum, dok će drugi da vrate NaN
- **Long Dates** imaju format “MMM DD YYYY”
  - Mjesec i datum mogu da izmijene redosled
  - Zarezi se ignorišu, imena su case insensitive

Dates metodi

- Metod za konvertovanje u datume Date.parse("string")

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

Pitanja