Introduktion till JavaScript

Gustaf Cele @ Mejsla, Feb. 2013

Agenda

- Motivering och bakgrund
- Språkets grunder
  - Vad som finns
  - Vad som saknas (och "saknas")
- Datatyper
- Funktioner
- Räckvidder
- Objektorientering

Vem är Gustaf?

- Började med Java och JavaScript 1999
- Fil. mag, data- och systemvetenskap
- Jobbat på riktigt sedan 2004
  - Med Java, JavaScript, C#, C
  - På Mejsla sedan 2008
- Just nu: JEE + Swing på Databyrån

Atwood's Law

Any application that *can* be written in JavaScript, *will* eventually be written in JavaScript.

Historia

Brendan Eich hamnar på Netscape

- Mål: "Lim" mellan dyra Java/backend-utvecklare och simpla frontendare

a. *"Scheme was the bait I went for in joining Netscape"*

b. *"The* diktat *from upper engineering management was that the language must 'look like Java'"*

c. *"I spent about ten days in May 1995 developing the interpreter, including the built-in objects"*

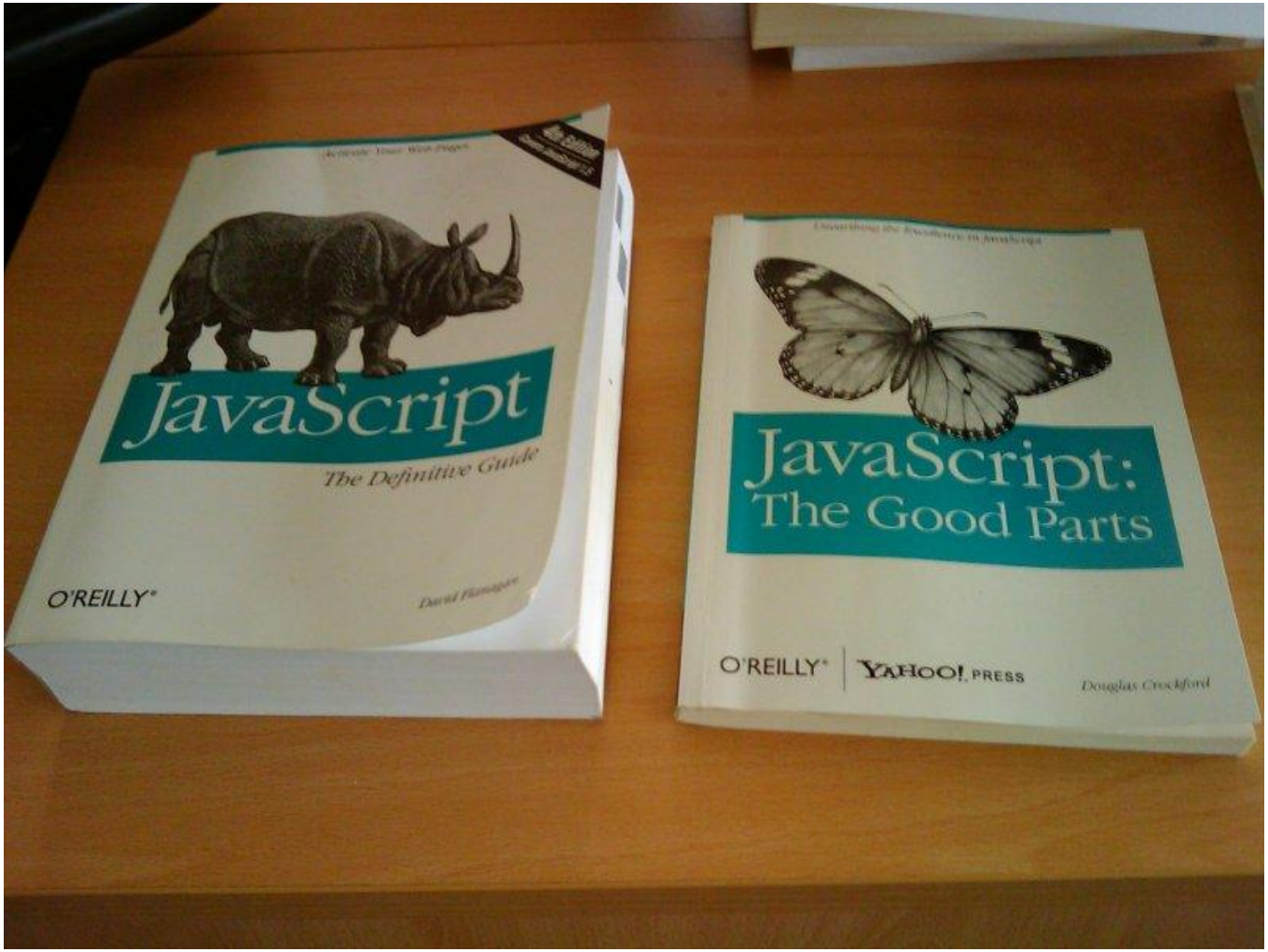
ECMA

- Standardiserar språket sedan 1996
- Nuvarande version 5

Designmål

- Målgrupp: "Nybörjare"
  - Förlåtande - "inte orsaka några fel"
  - Genom det, lätt att använda(?)
- Se ut som Java
  - Syntax, bibliotek
- Bete sig som LISP/Scheme
  - Första klassens funktioner
- Objektmodell som Self
  - Prototypbaserad OO

Resultat:


JavaScript
The Definitive Guide
O'REILLY
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford

Hello, world!

```
$ jrunscript
js> println("Hello, world!");
Hello, world!
js>
```

Hello World, enterprise edition

```
var name = read("What's your name? ");
var message = new String("Hello, "
    + name + "!");
for (var i = 0; i < message.length; i++) {
    var ch = message[i] // <-- Hm?
    print(ch);
}
println("");
```

Javaliknande syntax

- Variabler deklareras med `var`
  - eller inte alls, för globalt scope
- Kontrollstrukturer som i Java
  - `if`/`else`: "Truthy" värden
  - `switch`: Tal, strängar
  - `for`, `while`, `do`/`while`: Som i Java
- Fristående funktioner
  - Deklareras med `function(p1, p2){ ... }`
  - Kan även användas som värden/uttryck
- Exceptions: Allt flyger
  - Inget direkt stöd för att fånga specifika exceptions
- Semikolon "valfritt". Bra eller dåligt?

Saker vi får klara oss utan

Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter...

- Standardbibliotek, förutom:
  - Strängar
  - Regexp
  - Arrayer (≈ listor)
  - Matte (≈ `java.util.Math`)
  - Datum (kopia av `java.util.Date`, tyvärr)
  - JSON-parser (på senare tid)
- Namespaces/paket
- Modulhantering
- Klasser

Vi saknar dock inte allt på listan.

Dynamisk, stark och svag typning

Dynamisk typning:

Variabler (etc) har ingen specifik typ.

Stark typning:

"*Jag begriper typsystemet och tycker om det."*

Svag typning:

"*Jag tycker inte om typsystemet."*

JavaScript är **dynamiskt** och **svagt** typat.

JavaScripts typer

- `boolean`
- `number` (hel- och flyttal)
- `string`
- `function`
- `object` (även `null`)
- `undefined`

Går att kolla med `typeof`:

```
if (typeof val == "string") val = parseInt(val);
```

Literaler

- `true` och `false`
- `undefined` och `null`
- Tal: `123`, `1.23`, `1.2e3`, `0xABCD`, `0123`
  - `Infinity` och `NaN`
- Strängar: `"ett"`, `'tv\u00e5'`
- Array: `[1, true, "three"]`
  - Muterbar - mer som Javas `List<T>` än arrayer
- Regexpar: `/https?:\/\/\S+/`
- Objekt: `{ a: 1, "foo bar": 2 }`
  - `assert(o.a * 2 == o['foo bar'])`
  - Mer om objekt senare!

Typkonvertering - jämförelser

```
"true" == true          false
"42" == 42              true
"052" == 052            false
"0x2A" == 0x2A          true
"052" == 52             true
false == "false"        false
false == ""             true
"0" == false            true
"" == 0                 true
0 == "0"                true
"0" == ""               false
```

Lärdom:

Använd *alltid* de icke typkonverterande operatorerna, === och !==.

Istället för:

```
if (foo == bar && alfa != beta)
```

Använd:

```
if (foo === bar && alfa !== beta)
```

Funktioner

Statement:

```
function f(a1, a2) {
   return a1 + a2;
}
```

Expression:

```
var f = function(a1, a2) {
   return a1 + a2;
}
```

Argumentlistor

Aritet är bara en riktlinje:

```
function hello(name) {
   println("Hello, " +
                (name || "World"));
}
hello();
hello("JavaScript");
```

Variadiska funktioner

En magisk variabel, `arguments`, symboliserar den faktiska argumentlistan:

```
function printAll() {
   for (var i = 0; i < arguments.length; i++)
      println(arguments[i]);
}
printAll("alfa", "beta", "delta");
```

`arguments` är dock inte en array, tekniskt sett.

Nästlade funktioner

```
function factorial(n) {
   if (n < 0) throw new Error(n);
   return facTc(n, 1);
   function facTc(n, acc) {
      if (n <= 1) return acc;
      return facTc(n - 1, n * acc);
   }
}
```

Funktionell programmering:
Closures & variable capture

En funktion har tillgång till allt som är i räckvidd där den deklareras:

```
function printThrice(message) {
    function printIt() { println(message); }
    printIt();
    printIt();
    printIt();
}
```

Funktionell programmering:
First-class functions

Funktioner är värden som tal, strängar, etc., och kan hanteras på samma sätt:

```
function writeToConsole(s, appendNewline) {
    var outputFunc = print;
    if (appendNewline) outputFunc = println;
    outputFunc(s);
}
writeToConsole("Hello, 1st-class functions!", true);
```

Funktionell programmering:
Mutable variable capture

Man kan inte bara använda, utan också ändra på variabler från en funktion som redan returnerat:

```
function makeSerialGenerator() {
   var current = 1;
   return function() {
      return current++;
   };
}
var nextSerial = makeSerialGenerator();
assert(nextSerial() === 1);
assert(nextSerial() === 2);
```

Räckviddsregler

En variabel som inte blir deklarerad hamnar automatiskt i det *globala* scopet.

En `var`-deklarerad variabel kan nås från hela funktionen där den deklareras:

```
function id(x) {
    r = x;
    var r;
    return r;
}
```

Räckviddsregler

- "Block scope" förekommer inte i JS
- Alla variabler "hoistas" till början av funktionen
- Vad händer här?

```
function printSquaresTo(end) {
    for (var i = 1; i <= end; i++) {
        var sq = i * i;
        println(sq);
    }
}
```

Hoisting in action

Eftersom alla variabler hoistas upp...

```
function printSquaresTo(end) {
    var i, sq;
    for (i = 1; i <= end; i++) {
        sq = i * i;
        println(sq);
    }
}
```

När blir detta ett problem, då?

Objektorientering

Enkla objekt kräver inget förarbete:

```
var circle = {
    radius: 12,
    position: { x: 145, y: 32 },
    toString: function(){
        return "Circle(r=" + this.radius + ")";
    }
};
```

Objektorientering - enterprise edition

```
function Point(x, y) {
   this.x = x;
   this.y = y;
}
function Circle(radius, position) {
   this.radius = radius;
   this.position = position;
};
Circle.prototype.toString = function(){
      return "Circle(r=" + this.radius + ")";
}
var circle=new Circle(12, new Point(145, 32));
```

Objekt 101

- Alla objekt ärver från `Object`
  - `toString` - som i Java
  - `constructor` - funktionen som objektet `new`ades med
- Alla funktioner kan `new`as
  - ... men returneras något ignoreras `new`
- `new` kan utelämnas vid konstruktoranrop
  - ... med kaos som följd
- Nya fält kan läggas till godtyckligt...
  - `o.x = y;`
  - `o["wt" + f(a + 3)] = "bananer";`
- ... och tas bort:
  - `delete o.x;`

Prototyper

Man kan använda en konstruktors prototyp för att ge gemensamma egenskaper till alla instanser:

```
function Point(x, y) {
    this.x = x;
    this.y = y;
}
Point.prototype.distanceTo = function(other){
    var dx = this.x - other.x, dy = this.y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
}
var p = new Point(0, 0);
assert(p.distanceTo(p) == 0);
```

Prototyper för arv

Genom att tilldela prototypen kan vi åstadkomma ett slags arv.

```
function Shape(){}
Shape.prototype.distanceTo = function(otherShape){
    return this.pos.distanceTo(otherShape.pos);
}
function Circle(pos, radius) {
    this.pos = pos; this.radius = radius;
}
Circle.prototype = new Shape;
Circle.prototype.constructor = Circle;
```

Prototyper, del 2

För skicka argument till superklasskonstruktorn kan vi använda `Function.call`:

```
function Shape(pos){ this.pos = pos; }
Shape.prototype.distanceTo = function(otherShape){
    return this.pos.distanceTo(otherShape.pos);
}
function Circle(pos, radius) {
    Shape.call(this, pos); this.radius = radius;
}
Circle.prototype = new Shape;
Circle.prototype.constructor = Circle;
```

Metoder

- Funktion som används genom fält på objekt
- Värdet av `this` bestäms beroende på hur man refererar till funktionen
  - Har dock *alltid* ett värde - om inte annat, det globala objektet
- Olika klasser kan dela på samma funktion
- Går att manipulera för att få `this` till vadhelst man vill mha metoder på `funktioner`
  - `function.call(thisArg, arg1, arg2, ..., argN);`
  - `function.apply(thisArg, [a1, a2, ..., aN]);`

Metoder: Enkla fall

```
js> var consultant = { name: "Gustaf" };
js> var cheese = { name: "Stilton" };
js> function f() { return this.name; }
js> consultant.toString = f;
js> cheese.toString = consultant.toString;
js> println(consultant.toString());
Gustaf
js> println(cheese.toString());
Stilton
js> println(f()); // WTF?
undefined
```

Metoder: `this` och nästlade funktioner

```
order.printAsCsv = function(){
    // Prints order in CSV format:
    // Order ID, quantity, product ID
    this.items.forEach(function(item){
        println([this.id, item.quantity,
                 item.product.id].join(","))
    })
}
```

Resultat:

```
,1,234
,3,345
,1,456
```

Workaround: Temporär variabel

```
order.printAsCsv = function(){
    // Prints order in CSV format:
    // Order ID, quantity, product ID
    var self = this;
    this.items.forEach(function(item){
        println([self.id, item.quantity,
                 item.product.id].join(","))
    })
}
```

Förslag på hemuppgift: Terminalbaserad telefonbok

- Grundfunktionalitet:
  - Lägga in poster
  - Visa poster: Söka enskilda, lista alla
  - Redigera poster
  - Ta bort
- Möjliga utökningar:
  - Olika kontakttyper: Telefon, adress, mail, ...
  - Hardcore OO - bryt ut person, plats, organisation, ...
  - "Fuzzy search", m.h.a. t.ex. Levenshteinavstånd
  - Taggade relationer mellan olika entiteter
    - "Kalle är chef till Gustaf", "Bart är son till Homer"
  - Skapa grupper av poster ("bandylaget", "bandet", ...)

Hemuppgift:
Råd för implementering

- ***Versionshantera din kod!!!***
- Föreslagen miljö: Rhino (`jrunscript`)
- Undvik helst externa bibliotek
  - Slåss mot språket, inte ramverken!
  - Spara till fil?
- Perfektion förväntas ej
  - Ej önskvärt!
- Alla rekommendationer är just rekommendationer
  - Kreativitet uppmuntras!

Länkar

Mozilla Developer Center - allmän JS-dokumentation

https://developer.mozilla.org/en-US/docs/JavaScript

w3schools - smaka på JS som den ser ut i det vilda

http://www.w3schools.com/js/js_intro.asp

Rhino Shell - inbyggda funktioner m.m. i Rhino

https://developer.mozilla.org/en-US/docs/Rhino/Shell

Överkurs

Inkapsling

Motsvarighet till Javas private finns inte.
Lösning: Closures!

```
function Point(x, y) {
    this.getX = function(){ return x; }
    this.getY = function(){ return y; }
    this.distanceTo = function(other){
        var dx = x-other.getX(), dy = y-other.getY();
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

Syntaktiskt socker (salt?):

Property accessors

```
function Point(x, y){
   this.__defineGetter__("x", function(){
      return x;
   });
   this.__defineGetter__("y", function(){
      return y;
   });
}
assert(new Point(123, 456).x === 123)
```

Syntaktiskt socker (salt?):
Property accessors för literaler

```
function p(x, y){
   return {
      get x(){ return x},
      get y(){ return y}
   };
}
assert(p(123, 456).x === 123)
```

Quiz! Vad blir fel här?

```
function processTable(table){
    for (i = 0; i < table.rows.length; i++)
        processRow(table.rows[i]);
}

function processRow(row) {
    for (i = 0; i < row.cells.length; i++)
        processCell(row.cells[i]);
}

function processCell(cell) { /* gör't */ }
```

Mutable variable capture + hoisting: Försök 1 - vad händer?

```
function disableButtonsOnClick() {
    for (var i = 0; i < buttons.length; i++) {
        buttons[i].onclick = function(){
            buttons[i].disable();
        };
    }
}
```

TypeError: Cannot call method "disable" of undefined

Mutable variable capture + hoisting: Försök 1 - vad hände?

```
function disableButtonsOnClick() {
    var i;
    for (i = 0; i < buttons.length; i++) {
        buttons[i].onclick = function(){
            buttons[i].disable();
        };
    }
}
```

Aj då: i == buttons.length

Mutable variable capture + hoisting: Försök 2 - vad händer?

```
function disableButtonsOnClick() {
    for (var i = 0; i < buttons.length; i++) {
        var btn = buttons[i];
        btn.onclick = function(){
            btn.disable();
        };
    }
}
```

Sista knappen släcks oavsett vilken man klickar på.

Mutable variable capture + hoisting: Försök 2 - vad hände?

```
function disableButtonsOnClick() {
    var i, btn;
    for (i = 0; i < buttons.length; i++) {
        btn = buttons[i];
        btn.onclick = function(){
            btn.disable();
        };
    }
}
```

btn ändras under varje varv i loopen, tills vi kommer till den sista.

Mutable variable capture + hoisting: Försök 3 - rätt sätt

```
function disableButtonsOnClick() {
    for (var i = 0; i < buttons.length; i++) {
        (function (btn){
            btn.onclick = function(){
                btn.disable();
            };
        })(buttons[i]);
    }
}
```

Mutable variable capture + hoisting: "Funktionelligt" alternativ

```
function disableButtonsOnClick() {
    buttons.forEach(function(btn){
        btn.onclick = function(){
            btn.disable();
        };
    });
}
```

Var ska man deklarera variablerna? Där de finns "på riktigt", eller det scope man vill använda dem i?

Funktionell programmering:

Skriv ut alla kunder som köpte något under 2012

```
println(
    customers.filter(function(c){
        return c.orders.some(function(o){
            return o.datePlaced.getFullYear() === 2012
        })
    }).map(function(c){return c.name})
    .reduce(function(p, n){
        return p ? p + ", " + n : n;
    }, ""));
```

`filter`	Välj alla värden i arrayen som motsvarar ett kriterium
`map`	Transformera alla värden i arrayen (`flatMap` saknas)
`reduce`	Reducera (aggregera) alla värden i en array till ett

Semikolon

- Obligatoriska enligt grammatiken
- Tekniskt sett valfria i programmen
- Om utelämnade produceras de vid parsning
  - Kan ge kniviga fel när de stoppas in oväntat
  - Google: "Semicolon insertion"
- Rekommenderas ofta att man använder dem
  - "För att Crockford säger så"
  - Dock få fall där utelämnande orsakar riktiga problem
- *Den riktiga ondskan är hanteringen av utelämnade semikolon, inte utelämnandet*