BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Computergrafik II
- JavaScript Crashkurs -

Bachelor Medieninformatik

Prof. Dr.-Ing. Kristian Hildebrand
http://hildebrand.beuth-hochschule.de
khildebrand@beuth-hochschule.de

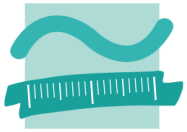
Inhalt

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Einführung
- Funktionen, Gültigkeitsbereiche, Closure
- Objekte
- Generatoren und Konstruktoren
- Module und RequireJS
- 2D-Grafik im Canvas
- jQuery und HTML
- Prototypen und Vererbung

Heute nur bis hier!

Hausaufgabe für Morgen:
Skript durchschauen



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript: Einführung

Wie ist JavaScript enstanden?

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JS had to look like Java only less so, be Java's dumb kid brother or boy-hostage sidekick. Plus, I had to be done in ten days or something worse than JS would have happened."

Brendan Eich [1]

- 1995: Brendan Eich entwickelt *Mocha / LiveScript* bei Netscape
- Ziel: Konkurrenz zu *Visual Basic,* für *semiprofessionelle* Frontend-Entwickler
- Java war gerade das „hot thing for the web", daher Umbenennung in *JavaScript*

1) http://en.wikipedia.org/wiki/Brendan_Eich

Mehr Rückblick unter https://brendaneich.com/2011/06/new-javascript-engine-module-owner/

Was ist JavaScript?

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript is a hybrid

dynamically typed,

single-threaded,

non-blocking,

asynchronous, concurrent

programming language.

Was ist JavaScript?

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript is a hybrid

dynamically typed,

single-threaded,

non-blocking,

asynchronous, concurrent

programming language.

JavaScript ist eine hybride, dynamische Skriptsprache

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- **funktional:** mit Funktionen als First-Class-Citizens, Closures, ...
- **objektorientiert:** aber nicht mittels Klassen, sondern mittels Prototypen.
- **prinzipiell unstrukturiert:** JavaScript gibt keine Strukturen vor*; diese müssen mittels Patterns und Disziplin vom Entwickler geschaffen werden. (prozedural)

*) mangels strenger Typisierung und Modulsystem

Was ist JavaScript?

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript is a hybrid

dynamically typed,

single-threaded,

non-blocking,

asynchronous, concurrent

programming language.

JavaScript ist eine hybride, dynamische Skriptsprache

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- **funktional:** mit Funktionen als First-Class-Citizens, Closures, ...
- **objektorientiert:** aber nicht mittels Klassen, sondern mittels Prototypen.
- **prinzipiell unstrukturiert:** JavaScript gibt keine Strukturen vor*; diese müssen mittels Patterns und Disziplin vom Entwickler geschaffen werden. (prozedural)
- **kompromisslos dynamisch:** Objekte können zur Laufzeit um Methoden und Attribute erweitert werden, Quellcode kann zur Laufzeit hinzugefügt werden, ...

*) mangels strenger Typisierung und Modulsystem

JavaScript: Primitive Typen

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- **undefined**
- Zahl (Number)
- Zeichenkette (String)
- Wahrheitswert (Boolean)

```
var test; // undefined
```

Primitiv-Typen: unveränderlich (immutable)

```
var n = 3;
var b = true;
var s = "Hi!";
```

Nach der Zuweisung verweist der Bezeichner auf das konstante primitive Objekt.

- Typfestlegung bzw. -prüfung findet erst zur Laufzeit statt

JavaScript: Objekt-Typen

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- **Object**
- **Array**
- **Function**
- Number
- Boolean
- String
- RegExp
- Date

Es existiert auch ein unveränderliches Objekt **null**.

```
var test = new String("foo");
test = {0:f, 1:o, 2:o, length:3...};
test[0] = f
```

```
var o = new Object();
var o = {};

var a = new Array();
var a = [];

var b = new Boolean(true);
var s = new String("Hi");
```

äquivalent

äquivalent

Objekt-Wrapper
für Primitivtypen

Das Typsystem von JavaScript

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Implizite Typisierung: Typ muss nicht deklariert werden.

```
var x = 5.6;
```

- Schwache Typisierung: keine strenge Prüfung,
  viele implizite Umwandlungsregeln

```
var x = 4 + 3.2 + "7.3";
```
→ Ergebnis: "7.27.3"

- Dynamische Typisierung: Typ kann erst zur Laufzeit bekannt werden

```
var x = undefined;
if <user clicks on button A> {
  x = 5;
} else {
  x = "hello world";
};
```

Vergleichsoperator

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Ein konkretes Beispiel für die Effekte der schwachen Typisierung ist die Semantik des Vergleichs-Operators == in JavaScript:

```
var a = 1;
var b = "1";
typeof(a)      → Number
typeof(b)      → String"
a == b         → true
```

Beim Vergleich mittels == werden die Typen der Argumente ggf. implizit umgewandelt !

Der tatsächliche Algorithmus ist ziemlich komplex:
http://es5.github.io/#x11.9.3

- Daher wurde in JS auch ein *strikter* Vergleichsoperator eingeführt, der bei ungleichen Typen auf jeden Fall `false` zurückliefert

```
a === b        → false
```

Vergleichsoperator, Primitive Typen vs. Objekte

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Bei primitiven Typen werden Werte verglichen, bei Objekten hingegen die Identität (handelt es sich um das gleiche Objekt):

```
var a = "hallo";
var b = "hallo";
typeof(a)          → String
a == b             → true
```

```
var a = new String("hallo");
var b = new String("hallo");
typeof(a)                    → Object
a == b                       → false
var c = a;   (Referenz auf gleiches Objekt)
a == c                       → true
```

- `null` (Obj.) und `undefined` (primitiv) sind gleich, nicht identisch:

```
null == undefined    → true
null === undefined   → false
```

Was ist JavaScript?

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript is a hybrid

dynamically typed,

single-threaded,

non-blocking,

asynchronous, concurrent

programming language.

Non-blocking I/O

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- JavaScript Interpreter läuft in einem Thread pro Browserfenster

```
window.console.log('This is the start.');

setTimeout(function callbackFunction() {
        window.console.log('This is a msg from callback.');
}, 0);
```

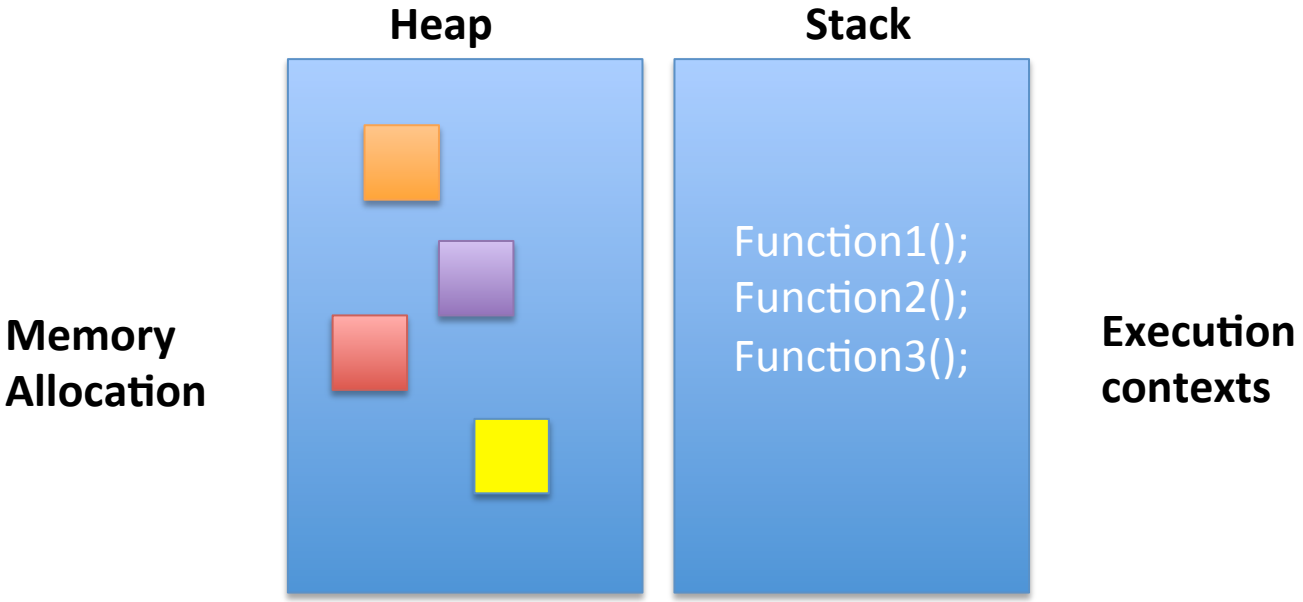
← 0 Millisekunden

```
window.console.log('This is just a message.');
```

Non-blocking I/O

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

JavaScript Runtime

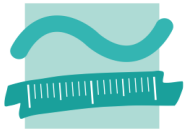


Memory Allocation

Execution contexts

- Neben der Runtime gibt es:
  - WebAPIs (DOM, XMLHTTPRequest, setTimeout ...)
  - EventLoop
  - Callback Queue

http://latentflip.com/loupe



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript:
Funktionen und Gültigkeitsbereiche

Funktionen als Objekte erster Ordnung

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Allgemein:
  - *Eine funktionale Programmiersprache ist eine Sprache, die Sprachelemente zur Kombination und Transformation von Funktionen anbietet.* (wikipedia.de 09/2012)

- In JavaScript:
  - Funktionen sind Objekte;
  - können Variablen zugewiesen werden;
  - können als Funktionsargumente übergeben werden;
  - können zur Laufzeit erzeugt und gelöscht werden;
  - können ihre eigenen Eigenschaften und Methoden besitzen.

Funktionen definieren und verwenden

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
function f(a,b,c) {
  return (a+b)*c;
}
var x = f(1,2,3);
```

99%
äquivalent

```
var f = function(a,b,c) {
 return (a+b)*c;
};
var x = f(1,2,3);
```

- Funktions*deklaration* vs. Zuweisung eines Funktions*ausdrucks*
  - der rechte Ausdruck zeigt schön, dass Funktionen Objekte erster Klasse sind.

- Parameter
  - Für Parameter wird kein Typ deklariert.
  - Parameter verhalten sich in der Funktion wie eine lokale Variable.
  - Parameter, zu denen kein Argument übergeben werden, sind `undefined`

http://stackoverflow.com/questions/2717949/javascript-when-should-i-use-a-semicolon-after-curly-braces

Closure (1)

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var makeTransform = function( incr ) {
  var factor = 3;
  var f = function(arg) {
    return arg * factor + incr;
  };
  return f;
};
```

Bei Aufruf der äußeren Funktion wird die innere Funktion `f` erzeugt. Sie merkt sich dabei die aktuellen Werte von `incr` und `factor`

Die äußere Funktion „exportiert“ die innere und liefert sie als Ergebnis zurück

- Closure = Abschluss, Hülle, Kapsel
  - deutscher Fachbegriff: Funktionsabschluss
  - eines der wichtigsten Konzepte funktionaler Sprachen
  - Eine Funktion merkt sich ihren Erstellungskontext, also insbesondere alle Variablen und Argumente der umgebenden Funktion

Closure (2)

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var makeTransform = function( incr ) {
  var factor = 3;
  var f = function(arg) {
    return arg * factor + incr;
  };
  return f;
};
```

```
var t = makeTransform(5);
```

→ t ist eine neu erzeugte Funktion mit einem Argument

In ihrem Inneren sind die folgenden lokalen Bezeichner bekannt:

```
incr = 5
factor = 3
```

Closure (3)

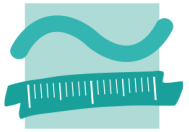
BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var makeTransform = function( incr ) {
  var factor = 3;
  var f = function(arg) {
    return arg * factor + incr;
  };
  return f;
};
```

```
var t = makeTransform(5);
```

→ Ähnlich wie der Aufruf eines Konstruktors: es entsteht ein Funktionsobjekt mit innerem gekapseltem Zustand

```
var x = t(7);   → 7*3+5 = 26
var y = t(9);   → 9*3+5 = 32
```



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Objekte in JavaScript

Das Objektmodell von JavaScript

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Was ist ein Objekt in JavaScript?
  - Menge von *Attributen* (Name, Wert)
  - *Keine Klassen:* keine statischen Vorgaben für Attribut-Namen oder -Typen
  - Prototyp, Delegation von Eigenschaften *(später)*

- Einfache Erzeugung mittels *Literal*-Notation

```
var e = {};
var o = { x:1, y:true, z:"super simpel!" };
var h = { name:"Helmut",
          adresse: {
            strasse:"Limburger Str. 7",
            plz:"13353"
          }};
```

→ neues leeres Objekt

→ Obj. mit drei Attributen **`x,y,z`**

→ geschachteltes Objekt **`h.addresse.strasse`**

JSON: JavaScript Object Notaton

Zugriff auf Objekt-Attribute

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Zugriff über `.` oder `[]`
  - `.` kompakter und lesbarer
  - `[]` auch für Attribute, deren Namen keine zulässigen JavaScript-Bezeichner sind

```
var obj = { a:1,
            b:false,
            "f-n":"Beuth"
          };
```

```
obj.a          → 1
obj["a"]       → 1
obj.f-n        → interpretiert als obj.f - n → Exception: n is undefined
obj["f-n"]     → "Beuth"
obj.test       → undefined
obj.test.x     → Exception: obj.test is undefined
```

→ im wesentlichen können Objekte mit `[]` wie assoziative Arrays verwendet werden!

Objekte werden stets „by reference“ kopiert

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Objekte werden immer „by reference“ behandelt und niemals kopiert

```
var a = { name:"Otto" };
var b = a;
a.name = "Olga";
b.name;
```

→ **b** verweist auf **a**

→ **"Olga"**

- Jedes *als Literal* notierte Objekt ist eine neue Kopie

```
var a = {}, b = {};
var a = b = {};
```

→ zwei separate leere Objekte

→ zwei Verweise auf das gleiche leere Objekt.

Lies von rechts nach links,

entspricht **`var a = ( b = {} );`**

Arrays

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Objekt `x` mit den Attributen `x[0], x[1], x[2], ...`
- *Attribut* `x.length`
- `push(), pop(), indexOf(), slice(), splice(), ...`

```
var a = new Array();
var a = [];                      → a.length == 0
a.push("Hello");                 → a.length == 1, a[0] == "Hello"
a.push("again");                 → a.length == 2, a[1] == "again"
var x = a[35];                   → x == undefined, a.length == 2
a[35] = "out of scope?";         → a.length == 36
var x = a[35];                   → x == "out of scope?"
var x = a[34];                   → x == undefined
```

Siehe auch http://www.w3schools.com/jsref/jsref_obj_array.asp

Fragen: JavaScript Objekte und Arrays

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

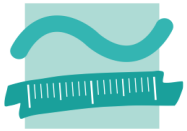
- **Was tut folgender Code?**

```
var x = {my:3, your:5};
var y = {my:x, your:x};
y.my.my = 4;      → x?      → {my:4, your:5}
y.my = 4;         → x?      → x bleibt unverändert, in y wird der
                              Verweis auf x durch eine Zahl ersetzt
```

- **Erzeugt folgender Code ein Array?**

```
var x = { 0:"hallo", 1:"echo" };
```

Nein, sieht nur so aus. Aber die Methoden fehlen.



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript Patterns: Generatoren und Konstruktoren

Generatorfunktion erzeugt neues Objekt

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var MakeAddress = function(name, street, zip, city) {
  var obj = {};
  obj.name   = name;
  obj.street = street;
  obj.zip    = zip;
  obj.city   = city;
  return obj;
};
```

← erzeuge ein neues, leeres Objekt

← definiere Attribute des neuen Objekts

← liefere Objekt (mit Attributen) als Ergebnis zurück

```
var erika = MakeAddress("Erika Mustermann",
                        "Hauptstr. 1",
                        "10111", "Berlin");
var eStreet = erika.street;
erika.zip = "12345";
```

← Der Variablen `erika` wird ein neu erzeugtes Objekt zugewiesen

← Attribute sind öffentlich lesbar

← Attribute sind öffentlich schreibbar

Neues Objekt mit eigenen Methoden

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var MakeAddress = function(name, street, zip, city) {
  var obj = {};
  ...
  obj.zip = zip;
  obj.city = city;
  obj.zipAndCity = function() {
    return zip + " " + city;
  };
  return obj;
};
```

Methode setzt ZIP-Code und Stadt zusammen

Problem: `zip` und `city` sind die Parameter der Funktion, nicht die Attribute von `obj` !

```
var erika = MakeAddress("Erika Mustermann", "Hauptstr. 1",
                        "10111", "Berlin")
erika.zip = "12345";
erika.zipAndCity();
```

Ergebnis: "10111 Berlin"
Warum funktioniert die Methode nicht?

Objekt-Attribute vs. lokale Variablen

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var MakeAddress = function(name, street, zip, city) {
  var obj = {};
  ...
  obj.zip = zip;
  obj.city = city;
  obj.zipAndCity = function() {
    return obj.zip + " " + obj.city;
  };
  return obj;
};
```

← So werden die Attribute des Objekts `obj` angesprochen.

```
var erika = MakeAddress("Erika Mustermann", "Hauptstr. 1",
                        "10111", "Berlin")
erika.zip = "12345";
erika.zipAndCity();
```

← Ergibt gewünschtes Ergebnis: "12345 Berlin"

Konstruktoren: `new` und `this`

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Aufruf einer Funktion mit dem Operator `new()`
  - Neues Objekt wird implizit erzeugt
  - Bezeichner **`this`** wird innerhalb der aufgerufenen Fkt. an dieses Objekt gebunden

```
var Address = function(n,s,z,c) {
    this.name = n;
    this.street = s;
    this.zip = z;
    this.city = c;
    this.cityAndStreet = function() {
      return this.zip + " " + this.city;
    };
};

var a = new Address(...);
```

Konstruktorfunktion

`this` referenziert **dank `new`** ein frisch erzeugtes Objekt

`return this;` wird hier implizit eingefügt (!)

durch `new()` wird eine normale Funktion zur Konstruktorfunktion! Der Name der Funktion liest sich wie der „Typ“ des zu erzeugenden Objekts

Definition von Methoden über den `prototype`

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Wird ein neues Objekt mittels **`new ConstructorFunc()`** erzeugt, erbt es alle Eigenschaften von **`ConstructorFunc.prototype`**

```
var Address = function(n,s,z,c) {
    this.name = n;
    ...
};
Address.protoype.cityAndStreet = function() {
     return this.zip + " " + this.city;
};
var a = new Address(...);
var x = a.cityAndStreet();
```

← Definiere Attribute innerhalb der Konstruktorfunktion

← Definiere Attribut / Methode über den **Prototyp** der Konstruktorfunktion

Wann / warum den `prototype` verwenden?

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var Address = function(n,s,z,c) {
  ...
  this.cityAndStreet = function() {
    return this.zip + " " + this.city;
  };
};
```

← Hier wird **bei jedem Aufruf von `new Address()`** eine neue Funktion `cityAndStreet()` erzeugt.

```
var Address = function(n,s,z,c) {
   ...
};
Address.protoype.cityAndStreet = function() {
    return this.zip + " " + this.city;
};
```

← Hier wird nur **eine** Funktion `cityAndStreet()` erzeugt, die über den Prototypen allen Instanzen zu Verfügung steht.

Weiterer Unterschied: in der unteren Variante hat die Methode keinen Zugriff auf die lokalen Bezeichner der Konstruktor-Funktion (z.B. **`n`** oder **`s`**)!

Vorsicht mit `.prototype`

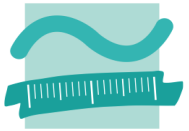
BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Achtung: Der tatsächliche Prototyp von Objekten ist nicht über `.prototype` zugreifbar – dieses Attribut ist nur für Konstruktorfunktionen zu verwenden!

Genauer Nachlesen:
http://pivotallabs.com/javascript-constructors-prototypes-and-the-new-keyword/
http://javascriptweblog.wordpress.com/2010/06/07/understanding-javascript-prototypes/

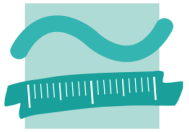
Mehr zum Thema Prototypen und Objekte später.

JavaScript EventLoop



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Loupe zeigen



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript Patterns: Module und RequireJS

Modulares JavaScript

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Wie teile ich mein JS-Projekt übersichtlich in Dateien auf?
- Sind viele einzelne Dateien nicht langsam beim Laden?
- Wie löse ich gegenseitige Abhängigkeiten?
  - Reihenfolge: Modul A braucht Modul B und C, C braucht A, ...?
- Vorteile:
  - Separation of Concerns
  - Bessere Möglichkeiten Tests zu schreiben
  - Leichter wiederverwertbare Komponenten

Kapselung mittels anonymer Funktionen

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Kapselung in anonyme Funktion:

```
(function () {
    var x = 5;
    var y = function(a,b) { ... };
}());
```

→ anonyme Funktion

Lokale Variablen , existieren nur innerhalb der Funktion

→ hier wird die anonyme Funktion sofort ausgeführt

D.h. eine Funktion kann Dinge wie ein Modul kapseln.

Jetzt fehlt noch:

- Definition des Modul-Interfaces: welche Objekte / Funktionen stellt das Modul zu Verfügung?
- Definition der Abhängigkeiten: welche externen Objekte / Funktionen benötigt das Modul, um funktionsfähig zu sein?

Module: Definition / Export des Modul-Interfaces

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

http://www.adequatelygood.com/2010/3/JavaScript-Module-Pattern-In-Depth

- Schaffen eines Namensraums für ein Modul
- Expliziter Export eines Interfaces

```
var vec2 = (function () {
    var v = {};
    v.add = function (v1,v2) {...};
    v.sub = function (v1,v2) {...};
    return v;
}());

var x = vec2.add(...);
var y = vec2.sub(...);
```

→ Modul in Namensraum `vec2`

Konstruiere ein Interface als Menge von Methoden eines Objekts

→ Liefere Interface-Objekt als Modul-Ergebnis zurück

→ Aufruf der Funktionen unter dem Namensraum des Moduls

Kapselung mittels anonymer Funktionen

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Expliziter Import externer Objekte

```
(function ($, FOO) {
    // $ und FOO sind hier bekannt
}($, FOO));
```

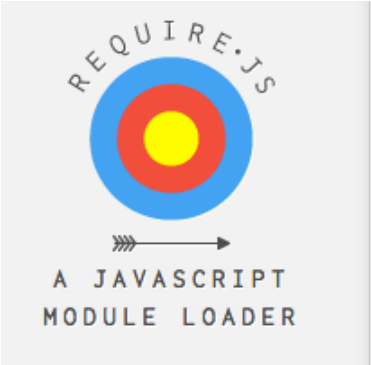
→ Objekte, die die Funktion benötigt, werden als Parameter deklariert

→ Übergabe der entsprechenden Objekte bei Aufruf der Funktion

Anstatt die Existenz „globaler“ Objekte vorauszusetzen, deklariert jedes Modul die von ihm benötigten Objekte als Parameter.

Der Nutzer des Moduls übergibt dem Modul die benötigten Objekte bei der Erzeugung des Moduls (beim Aufruf der Modul-Funktion).

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences



http://requirejs.org/

RequireJS

- Kompakte und unabhängige Lösung
- *Async. Module Definintion* (AMD),
  inzwischen sehr häufig im Einsatz
- Nachladen von Quelldateien (asynchron)
- Beschreiben und Auflösen von Abhängigkeiten

Außerdem: „Packen“ eines Projektes in eine einzige minimierte Datei.
Mittels eines Skripts lässt sich das gesamte JS-Projekt in eine einzige minimierte Datei verwandeln. Dort sind Kommentare und Leerzeichen entfernt und Bezeichner-Namen gekürzt, etc....

Beispiel: Definition eines RequireJS-Moduls

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Einfache Modul-Definition

```
define([], (function() {
  // Inhalt des Moduls...
}));
```

Einfaches Modul ohne Abhängigkeiten

Namen von vorausgesetzten Modulen (`.js`-Dateien)

Komplexeres Modul mit Abhängigkeiten

```
define(["util", "vec2", "scene", "point_dragger"],
       (function(util,vec2,Scene,PointDragger) {

  var Line = function(...) {
    ...
    var d = new PointDragger(...);
  };
  return Line;

})); // define
```

Namen, unter denen die Module in dieses Modul importiert werden

Verwendung eines importierten Moduls

Export des Modulinterfaces

Einbinden von RequireJS

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- In der HTML-Datei wird nur eine einzige JS-Datei referenziert
- Asynchrones Laden dieses „Hauptmoduls“ mittels RequireJS

HTML

```
<head>
    <meta charset="UTF-8">
    <title>CG2 A1: Lines, Curves &amp;amp; Canvas</title>
    <script data-main="main.js" src="../lib/require.js"></script>
    ...
```

Haupt-Modul

RequireJS-Bibliothek

Striktes JavaScript

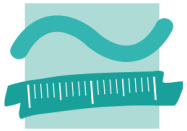
BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
define(..., (function(...) {

    "use strict";

    ...
}));
```

- `"use strict"` verbietet einige JavaScript-Konstrukte, die in Zukunft nicht mehr im Sprachstandard sein werden
- Meldet z.B. (in manchen Fällen) ein vergessenes `var`
- Nicht global verwenden (nicht alle Module vertragen das), sondern im Inneren der eigenen Module!



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

2D Grafik im <canvas>

Überblick: <canvas>

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Canvas = „Leinwand“ oder „Zeichenfläche“
  - Rechteckiger Bereich festgelegter Größe
  - Rasterbilder in einem Teil des Canvas darstellen
  - 2D-Grafik-Funktionen (Linien, Kurven, Rechtecke, Kreisbögen, ...)
  - WebGL-Funktionen (GPU-basierte 3D-Grafik, später!)
  - Transformationen
  - Inhalt als Rasterbild abspeichern
  - ...

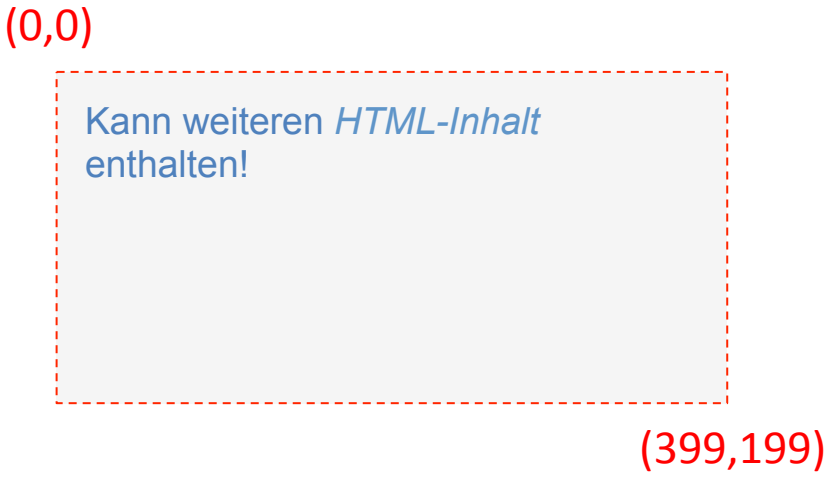
Einführung / Überblick: http://www.w3schools.com/html5/html5_canvas.asp

Canvas: Erzeugung, Koordinatensystem

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

HTML

```
<canvas id="myCanvas" width="400px" height="200px">
  Kann weiteren <i>HTML-Inhalt</i> enthalten!
</canvas>
```



CSS

```
#myCanvas {
  border-style: dashed;
  border-width: 1px;
  border-color: red;
}
```

Canvas: 2D Grafik erzeugen

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Zu jedem Canvas kann ein 2D-Rendering-Kontext erzeugt werden

JS

```
var canvas = $("#myCanvas").get(0);

var context = canvas.getContext("2d");

context.fillRect(...);
```

Das erste Element mit der ID `myCanvas` als natives HTML-Element

Erzeuge 2D-Rendering-Kontext

Alle weiteren Grafik-Befehle beziehen sich auf das Kontext-Element

Einführung / Überblick: http://www.w3schools.com/html/html5_canvas.asp

Linienzüge im Canvas

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

JS

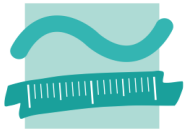
```
context.beginPath();
context.moveTo(20,20);
context.lineTo(30,40);
context.lineTo(10,70);
...
context.closePath();

context.strokeStyle = "#f00";
context.lineWidth = 1px;
context.fillStyle = "#0f0"

context.stroke();
context.fill();
```

- Neuen Pfad starten („Reset“)
- Erster Punkt des Pfads,
- zweiter Punkt des Pfads,
- und so weiter...
- Optional: verbinde ersten und letzten Punkt
- Linienfarbe
- Liniendicke
- Füllfarbe
- Linie tatsächlich zeichnen.
- Inneres füllen (bei geschlossenen Pfaden)

Gutes Tutorial: https://developer.mozilla.org/en-US/docs/Canvas_tutorial/Drawing_shapes
Canvas 2D Referenz: http://www.w3schools.com/html/html5_canvas.asp



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

HTML und jQuery
(nur ganz kurz)

HTML-Elemente und das DOM

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Ausschnitt aus einem HTML-Dokument

```
<canvas id="drawing_area" width="500" height="400”>
</canvas>

<div id="param_area" >

   <h3>Create Objects:</h3>
   <button id="btnNewLine" type="button”>New Line</button>

   <h3>Parameters</h3>
   <input id=“lineWidth” class=“objParam” ...>

</div>
```

← canvas-Element mit ID

← button-Element mit ID

← input-Element mit ID und class

Alle Elemente und ihre Attribute werden in JS über die DOM-API verfügbar gemacht.
DOM = Document Object Model

jQuery

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Sehr weit verbreitete JavaScript-Bibliothek
- Macht alle DOM-Elemente effizient von JavaScript aus zugreifbar
- Definiert nur eine einzige Funktion `$()` (machmal auch `jQuery()`)
- Ist auch ein AMD-Modul, welches mit RequireJS geladen werden kann

http://jquery.com/

- Beispiel: Funktion, die erst dann ausgeführt wird, wenn der DOM-Baum vollständig konstruiert ist:

```
$(document).ready(function(){
  // Your code here
});
```

Zugriff auf DOM-Elemente mittels jQuery

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Event-Handler für benanntes Objekt

HTML
```
<button id="btnNewLine" type="button”>New Line</button>
```

JS
```
$("#btnNewLine").click(...);
```
← Setzt einen Event-Handler für den Button-Klick auf. Selektor: #

- Event-Handler für *alle Objekte* einer bestimmten Klasse

HTML
```
<input id="lineWidth” class="objParam" ...>
```

JS
```
$(".objParam").change(...);
```
← Setzt einen Event-Handler für alle Elemente der Klasse objParam auf. Selektor: .

Zugriff auf DOM-Elemente mittels jQuery (2)

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Ein Attribut eines `<input>`-Elements lesen/schreiben

HTML
```
<input id="lineWidth" type"number" value="3">
```

JS
```
var x = $("#lineWidth").attr("value");
$("#lineWidth").attr("value", 5);
```

← Liest Attribut `value`

← Schreibt Attribute `value`

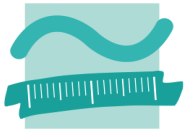
- Bestimmtes Element verstecken oder anzeigen

HTML
```
<div id="div_radius"> ... </div>
```

JS
```
$("#div_radius").hide();
$("#div_radius").show();
```

← Das `div` und sein Inhalt werden versteckt

← ... und wieder angezeigt



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript:
Weitere Typen und Konstrukte

Strings

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
var str = 'Say "hello" to my new BELLO!';
var str = "Say 'hello' to my new BELLO!";
```

```
str.length                  → 28
str.slice(5,5)              → "'hello'"
str.slice(-6)               → "BELLO!"
str.split(" ")              → [ "Say", "'hello'", "to", "my",
                                "new", "BELLO!" ]
str.match(/.ello/gi)        → [ "hello", "BELLO" ]
str.replace("BELLO","dog")
```

Regular Expression

http://www.w3schools.com/jsref/jsref_obj_string.asp

undefined, null und Co.: “falsy values“

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Alle diese Werte gelten als *falsy*
  - `false`
  - `0`
  - `""`
  - `NaN`
  - `null`
  - `undefined`

```
x = NaN;
if(!x) {
  // this could be undefined, NaN, 0, ...
};
```

- Was wann verwenden?
  - `0` = reguläres Ergebnis numerischer Berechnungen
  - `NaN` = irreguläres Ergebnis, z.B. Division durch 0, Ergebnis von `parseInt()`
  - `null` eher für explizites „verweist auf kein Objekt“ (seltener)
  - `undefined` steht für „Wert in diesem Kontext nicht definiert“

Objekt-Attribute und die Operatoren || und &&

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- „Auffüllen“ eines Objekts mit Standard-Werten

```
var middle = person.middle_name || "(none)";
```

entspricht:

```
var middle = person.middle_name;
if(!middle) {
  middle = "(none)";
};
```

VORSICHT bei numerischen Werten:
0 gilt als `false` !

- Exception wegen eines nicht definierten Attributs vermeiden:

```
// obj.model is undefined
var id = obj.model.id;                  → Exception: obj.model is undefined
var id = obj.model && obj.model.id;     → id === undefined
```

Code einfügen mittels `eval()`

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- `eval(str)` evaluiert einen beliebigen String so, als würde er in der aktuellen Closure im Code stehen.

```
var Func = function(a,b,formula) (
  var result = eval(formula);       // Alternative 1
  eval("var result = " + formula); // Alternative 2
  return result;
);
Func(3,5,"a+b*a");
```

- Natürlich ist `eval()` in echten Webanwendungen **böse**!
- Aber wirklich nützlich beim Skripting. Beispiel: Kurvenplotter

Exceptions – String vs. Error

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

```
try {
  // ...
  throw "just throw a string";
} catch(err) {
  console.log("caught it: " + err)
};
```

```
try {
  // ...
  throw new Error("message");
} catch(err) {
  console.log("caught it: " + err.message)
};
```

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

JavaScript
Prototypen, Vererbung und Co.

Im Detail recht „vertrackt“:
http://javascriptweblog.wordpress.com/2010/06/07/understanding-javascript-prototypes/

Prototypen

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Jedes Objekt* besitzt eine Referenz auf ein **Prototyp-Objekt**
  - welches wiederum ein Objekt ist → Prototyp-Kette *(prototype chain)*
  - abfragen mittels `Object.getPrototypeOf(<obj>)`
  - der Prototyp ist eine interne/versteckte Eigenschaft; er ist etwas anderes als das `.prototype`-Attribut einer Konstruktorfunktion!

```
var A = function() {};
A.prototype.x = 5;
```

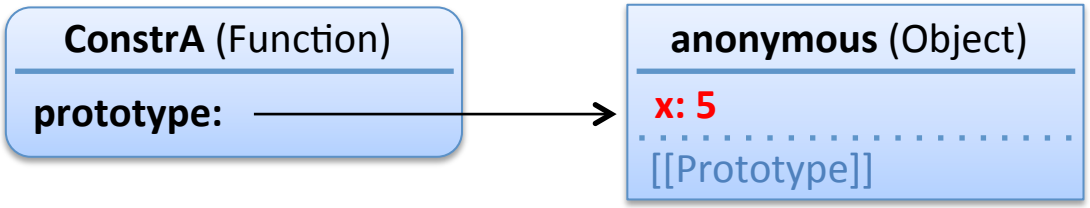
```
A.prototype                    → Object {x: 5}
```

```
Object.getPrototypeOf(A)       → function Empty() {}
```

*) Nur das oberste `Object` selbst hat keinen weiteren Prototypen.

Das Prototyp-Objekt

```
var ConstrA = new function() {};
ConstrA.prototype.x = 5;
```

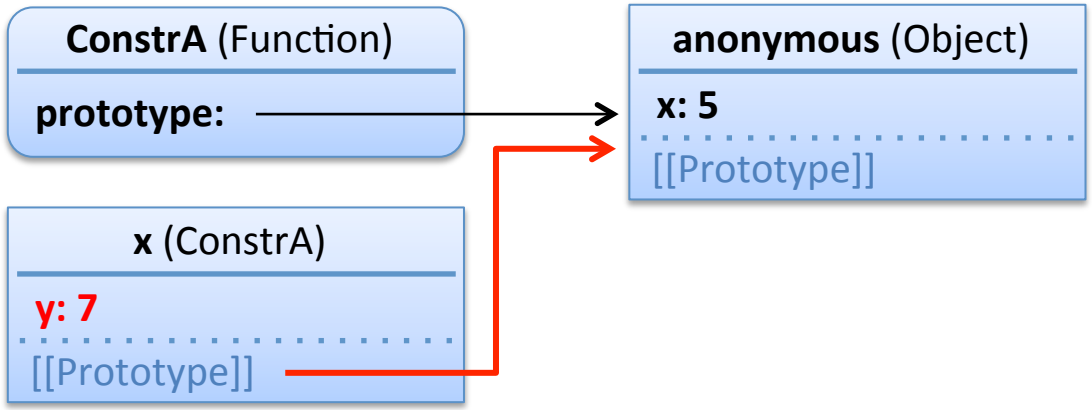


Das Attribut `.prototype` der Funktion
referenziert anfangs ein leeres neues Objekt.

Verwendung von .prototype beim Aufruf von new()

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

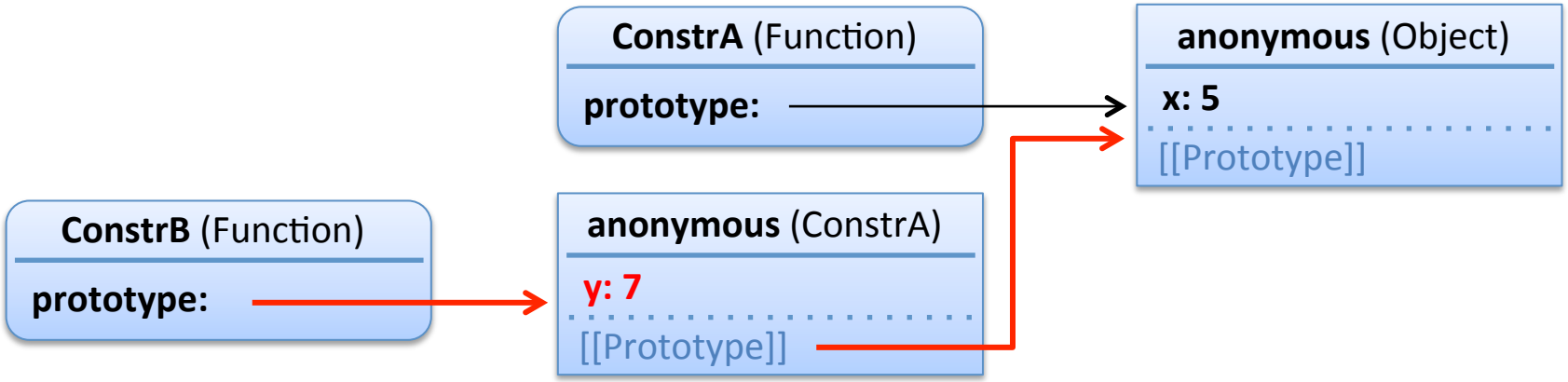
```
var x = new ConstrA();
x.y = 7;
```



mit **`new ConstrA()`** wird eine Instanz des "Typs" **`ConstrA`** erzeugt. Der Prototyp dieses Objekts ist eine Referenz auf den Wert von **`.prototype`** der verwendeten Konstruktorfunktion.

Vererbung

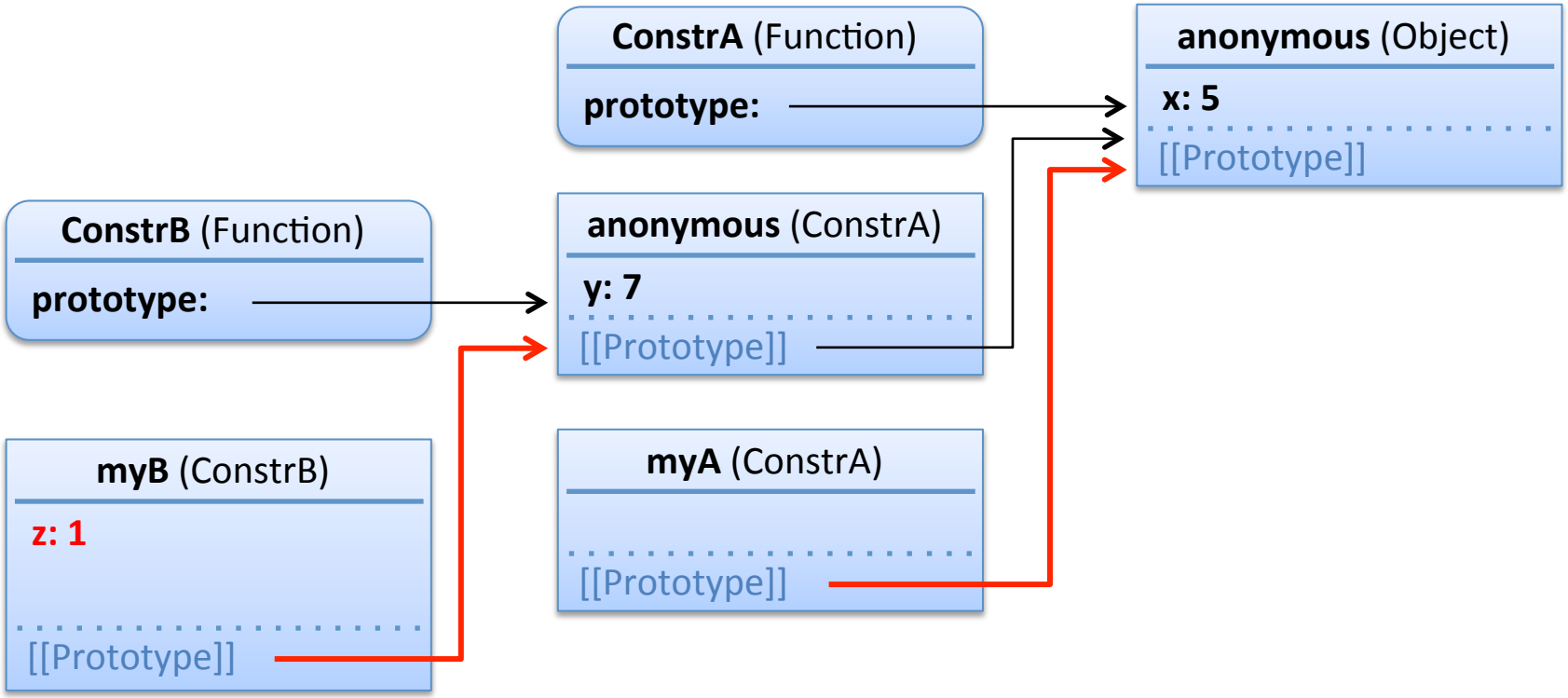
```
var ConstrB = new function() {};
ConstrB.prototype = new ConstrA();
ConstrB.prototype.y = 7;
```



Wenn ConstrB von ConstrA erben soll, muss ConstrB.prototype auf eine Instanz eines mittels ConstrA erzeugten Objekts verweisen.

Prototyp-Kette

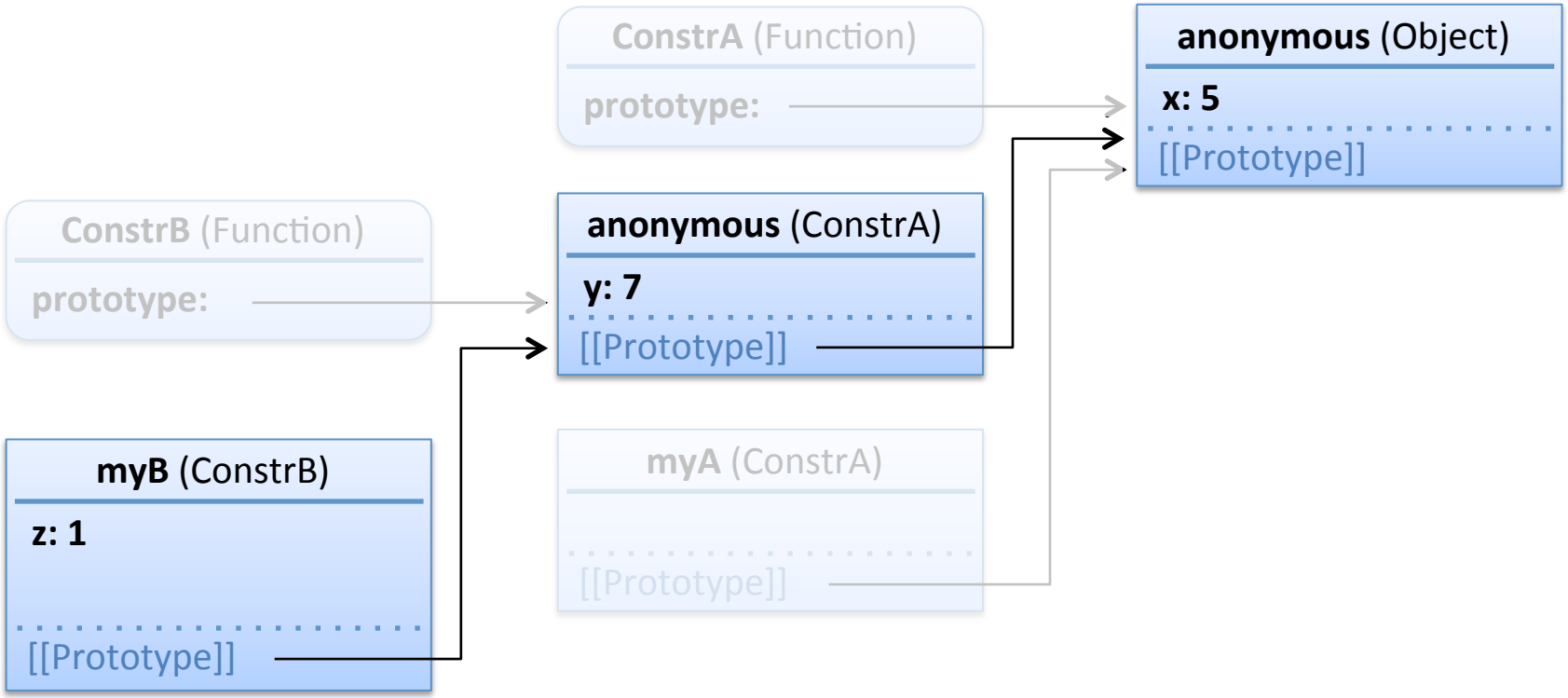
```
var myA = new ConstrA();
var myB = new ConstrB();
myB.z = 1;
```



Typ- und Ahnenforschung

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

```
typeof(myB)                 → object
myB instanceof ConstrB      → true
myB instanceof ConstrA      → true
```



Lesender Zugriff auf ein Attribut

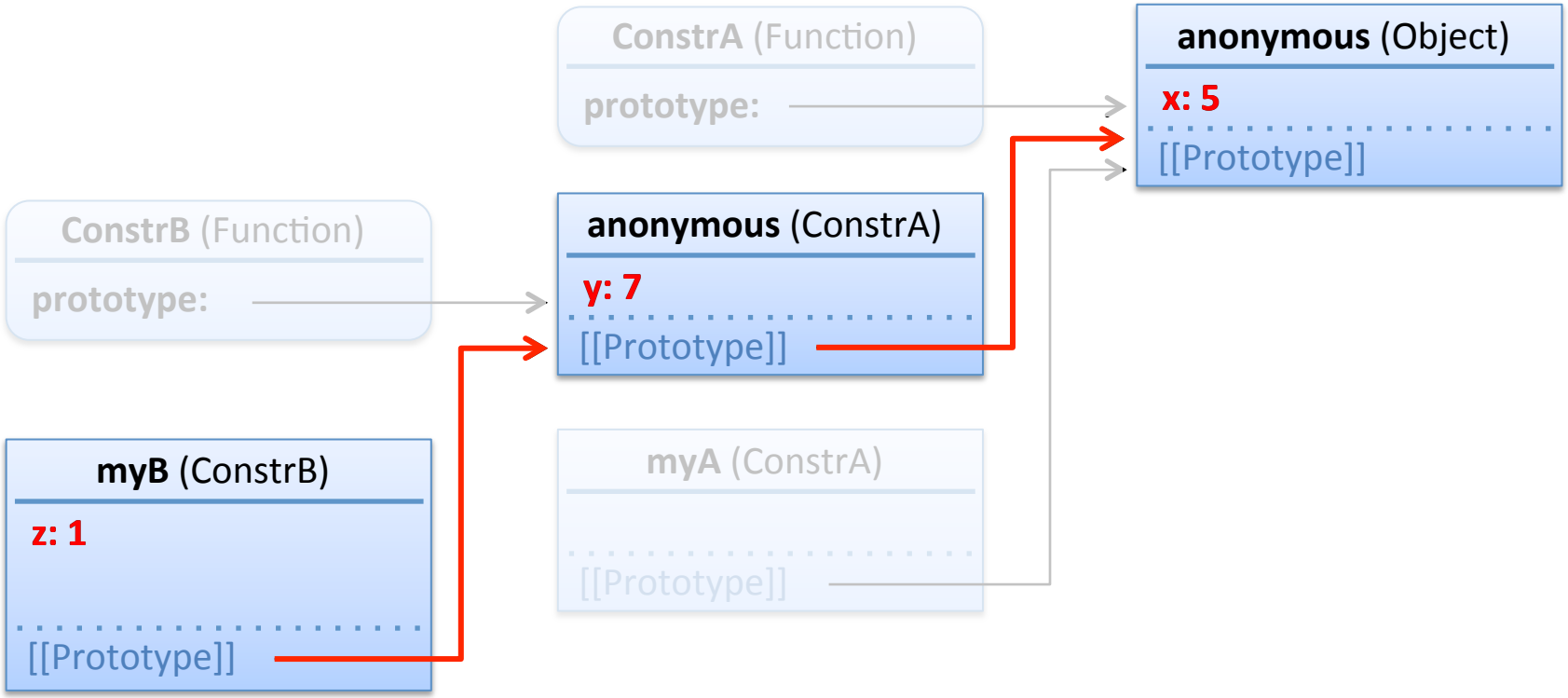
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

```
var z = myB.z;
var y = myB.y;
var x = myB.x;
```

→ **z=1** (Attribut von **myB**)

→ **y=7** (via *Prototype Chain*)

→ **x=5** (via *Prototype Chain*)



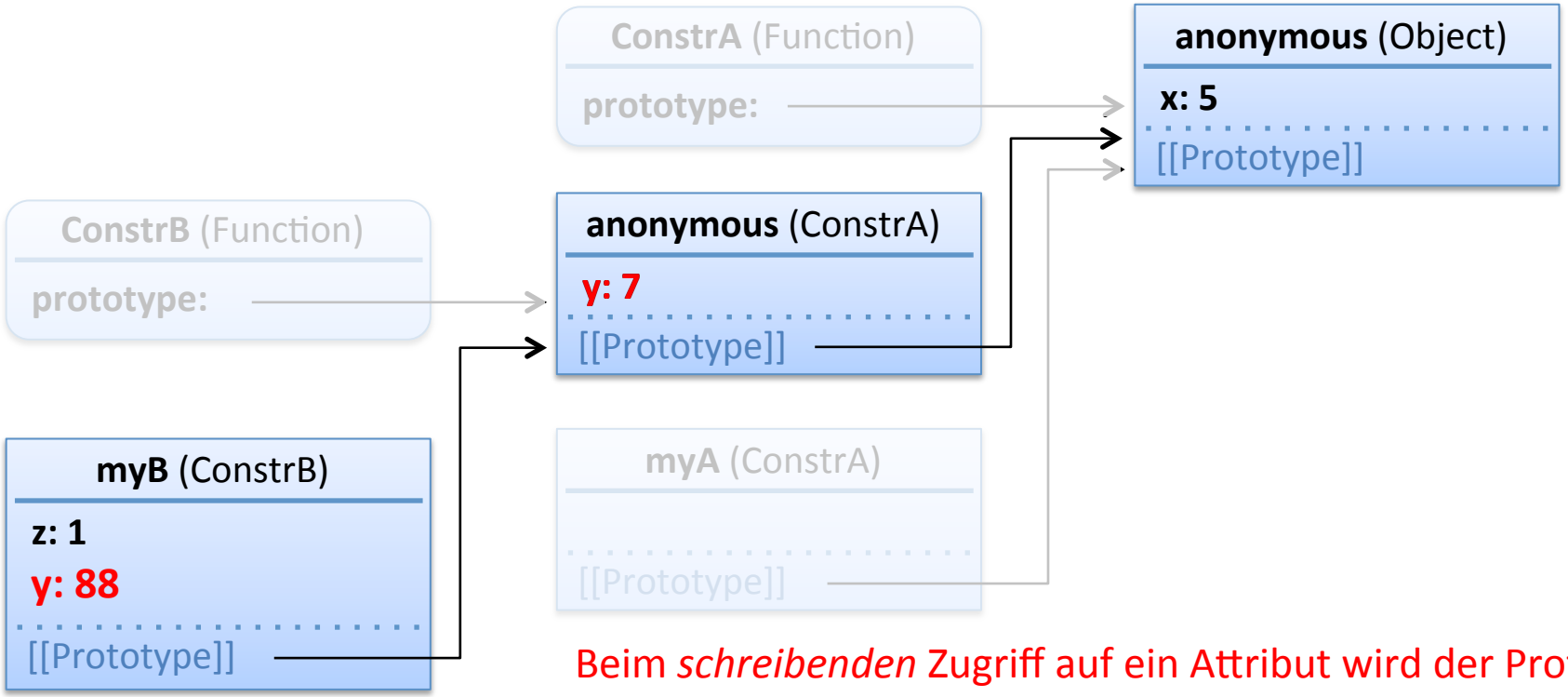
Beim *lesenden* Zugriff auf ein Attribut durchsucht JavaScript potentiell die gesamte Prototyp-Kette.

Schreibender Zugriff und Verdeckung von Attributen

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

```
myB.y = 88;
myB.y
```

→ **88** (`y: 7` im Prototypen ist verdeckt)



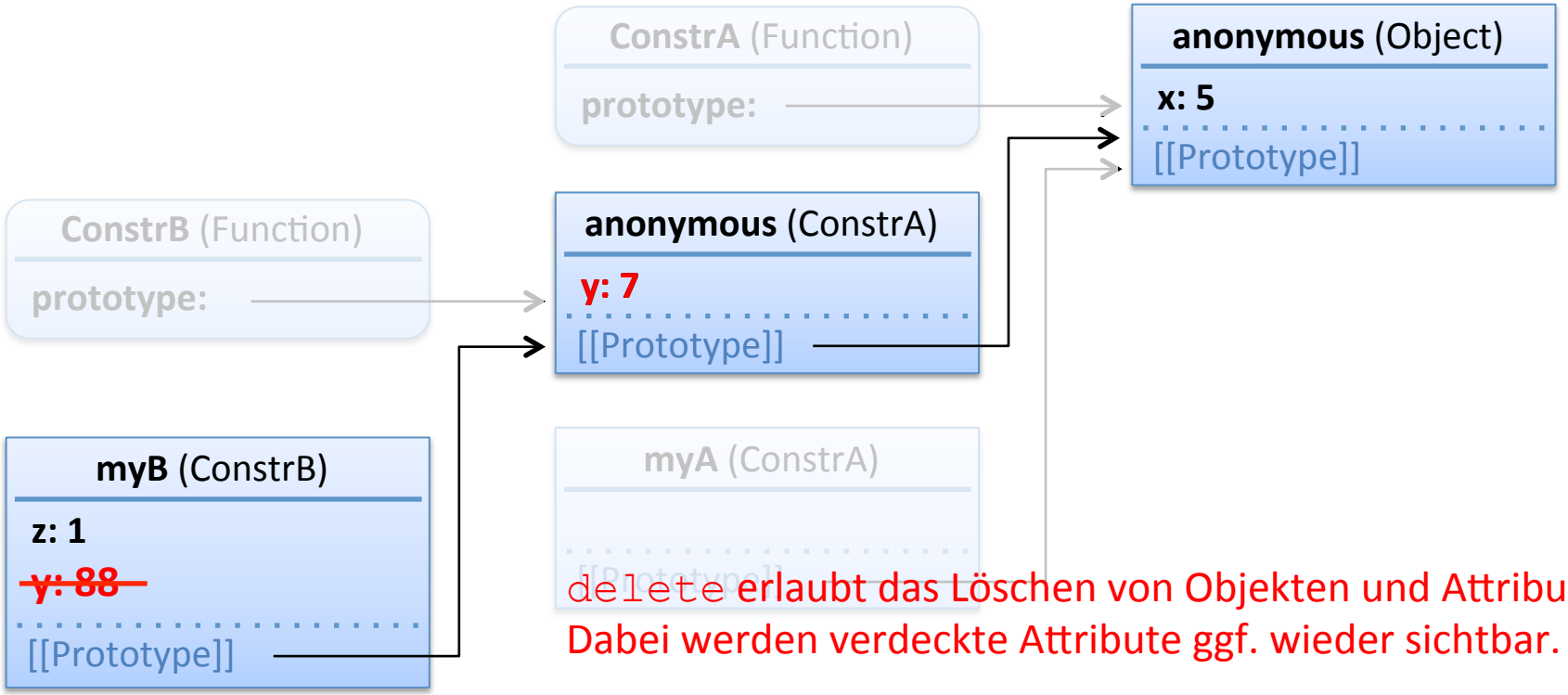
Beim *schreibenden* Zugriff auf ein Attribut wird der Prototyp niemals verändert; das Attribut wird ggf. neu angelegt.

Attribute gleichen Namens verdecken Attribute in der *Prototype Chain*.

Löschen von Attributen und Objekten

```
delete myB.y;
myB.y
```

→ 7 (über den Prototypen)



`delete` erlaubt das Löschen von Objekten und Attributen. Dabei werden verdeckte Attribute ggf. wieder sichtbar.

Objekte können nicht gelöscht werden, man kann lediglich die Referenzen auf die Objekte (z.B. durch `x=undefined` oder `x=0`) entfernen. JS verwendet Garbage Collection.

Eigenes Attribut vs. Prototyp-Attribut?

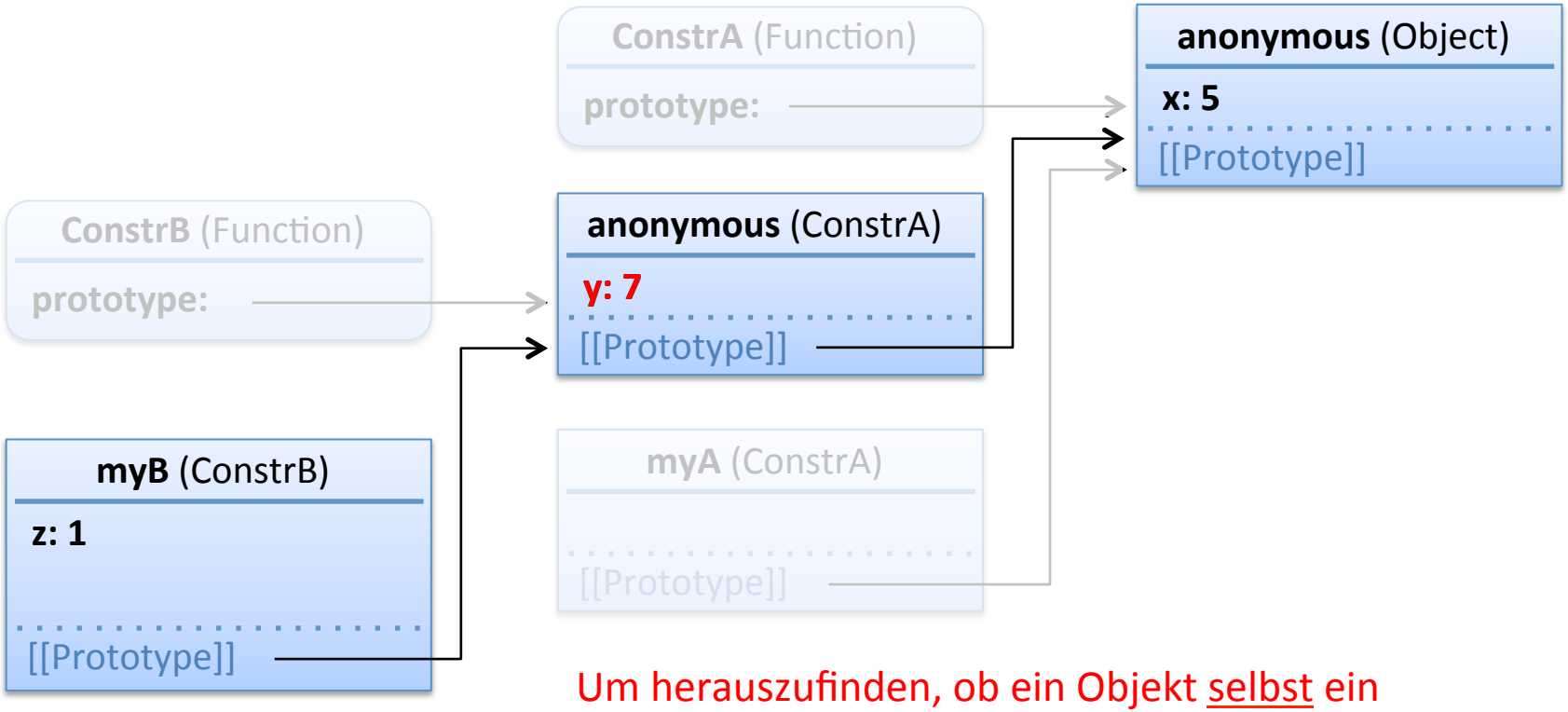
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

```
myB.x != undefined
myB.hasOwnProperty("z")
myB.hasOwnProperty("y")
```

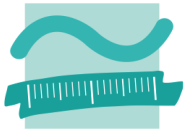
→ **true** (lesender Zugriff über Prototyp)

→ **true**

→ **false**



Um herauszufinden, ob ein Objekt selbst ein bestimmtes Attribut besitzt, ist die Methode **`hasOwnProperty()`** zu verwenden.



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Anhang: Sonstiges zu JavaScript / WebGL

Benannte Parameter / Konfigurations-Objekte

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- Normale Übergabe von Parametern

```
var ring = new Ring(context, 10, 1);
```

was bedeutet hier 10, 1?

-

```
var ring = new Ring(context, { radius: 10, height: 1 } );
```

Objekt mit benannten Attributen übergeben

  - ermöglicht die Verwendung „sprechender“ Parameter!

Benannte Parameter / Konfigurations-Objekte (2)

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Aufruf

```
var ring = new Ring(context, { radius: 10, height: 1 } );
```

- Definition

```
var Ring = function(gl, config) {
    config = config || {};          ← falls kein config-Parameter übergeben wird
    this.radius   = config.radius   || 1.0;
    this.height   = config.height   || 0.1;
    this.segments = config.segments || 20;  ← segments wurde beim Aufruf ausgelassen
    ...
};
```

Weiterführende Literatur: JavaScript

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

- D. Crockford, *JavaScript – The Good Parts*, O'Reilly 2011, 180 Seiten

  Gute Einführung, macht klar, welche Konzepte man einsetzen sollte.
- S. Stefanov, *JavaScript Patterns*, O'Reilly 2010, 232 Seiten

  Sehr gute Sammlung von Mustern und Tricks, die sich für JavaScript bewährt haben
- D. Flanagan, *JavaScript – The Definitive Guide*, O'Reilly 2011, 1077 Seiten

  Umfassendes Werk, mit guter aber sehr ausführlicher Einführung. Eher als Referenz.