



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Webbasierte Anwendungen

JavaScript

Prof. Dr. Ludger Martin

Gliederung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Einführung
- Variablen, Werte und Operatoren
- Bedingungen und Schleifen
- Funktionen
- Objektorientierung
- JSON
- Ausnahmebehandlung
- JavaScript Bibliothek
- Dynamic HTML



- Clientseitige Programmiersprache
- Vom Browser in einer Sandbox ausgeführt. Nur Objekte im Browser können angesprochen werden. Zugriff auf das Dateisystem ist nicht möglich.
- 1995 JavaScript ist mit Netscape Navigator 2.0 veröffentlicht worden.
- JavaScript ist plattformunabhängig, aber an einen Browser gebunden.
- ECMAScript ist ein „JavaScript“ durch *Ecma International* standardisiert.
- Es soll jährliche Updates geben.
- ActionScript 3.0 (Flash9) ist ECMAScript 4 konform.



- JavaScript kann direkt in HTML-Dokument notiert oder in einem externen Skript-Dokument definiert werden
- Notierung in HTML-Dokument

```
<script type="text/javascript"> ... </script>
```

- Zur Abwärtskompatibilität mit Browsern, die keine Skripte erlauben, wurden die JavaScript Anweisungen in HTML Kommentare gesetzt.

```
<script type="text/javascript">
```

```
<!--
```

```
...
```

```
-->
```

```
</script>
```

- Einbindung von JavaScript Dokumenten
 - Externe JavaScript Dokumente enthalten beliebige JavaScript Befehle und enden auf `.js`
 - Es können mehrere JavaScript-Dokumente eingebunden werden
 - Einbindung eines JavaScript-Dokuments

```
<script type="text/javascript" src="..."> </script>
```

`src` spezifiziert den Pfad und Namen des JavaScript Dokuments

- In HTML5 kann Typ-Angabe weggelassen werden

Einführung

Das erste JavaScript



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8"/>
    <title>Das erste JavaScript</title>
  </head>

  <body>
    <script>
      console.log("Das erste JavaScript");
    </script>
  </body>
</html>
```

Einführung

Das erste JavaScript



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Besser:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8"/>
    <title>Das erste JavaScript</title>
  </head>

  <body>
    <script src="script.js"></script>
  </body>
</html>
```

script.js:

```
console.log("Das erste JavaScript");
```

Einführung

Kommentare



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Kommentar über eine Zeile:

```
// ...
```

- Kommentare mehrzeilig:

```
/*  
...  
*/
```

Variablen, Werte und Operatoren



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Variablennamen
 - Dürfen nur aus Buchstaben, Ziffern, \$ und _ bestehen
 - 1. Zeichen muss ein Buchstabe oder eines der beiden gültigen Sonderzeichen sein
 - Groß- und Kleinschreibung wird unterschieden
- Deklaration von Variablen
 - Wird eine Variable nicht deklariert, so ist sie immer im globalen Kontext.
 - Mit `var` deklariert ist sie im Kontext der umschließenden Funktion (oder Global).
 - Mit `let` deklariert ist sie im Kontext des Blocks, Befehls oder Ausdrucks.

```
function varTest() {  
    var x = 31;  
    if (true) {  
        var x = 71; // gleiche Variable!  
        console.log(x); // 71  
    }  
    console.log(x); // 71  
}
```

```
function letTest() {  
    let x = 31;  
    if (true) {  
        let x = 71; // andere variable  
        console.log(x); // 71  
    }  
    console.log(x); // 31  
}
```


Variablen, Werte und Operatoren



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Variablen besitzen keine Typisierung
- Wertzuweisung

- `test = 42;`
- `test = 10.5;`
- `test = true;`
- `test = "Guten Tag!";`
- `test = 'Guten Tag!';`

→ Zeichenreihen mit " oder mit ' sind gleichwertig. Es gibt keine Unterscheidung wie z.B. in PHP.

Variablen, Werte und Operatoren



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Berechnungsoperatoren**
 - + Addition
 - - Subtraktion
 - * Multiplikation
 - / Division
 - % Modulo
 - Punkt vor Strich Rechnung
- **Vergleichsoperatoren**
 - === bzw. == gleich
 - !== bzw. != ungleich
 - > größer
 - < kleiner
 - >= größer gleich
 - <= kleiner gleich
- **Logikoperatoren**
 - && logisches UND
 - || logisches ODER
 - ! logisches NICHT

Variablen, Werte und Operatoren



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Zeichenreihenverknüpfung mit + Operator
- Zirkulare Bezüge


- zahl++ bzw. ++zahl	zahl = zahl + 1
- zahl-- bzw. --zahl	zahl = zahl - 1
- zahl += 2	zahl = zahl + 2
- zahl -= 5	zahl = zahl - 5
- zahl *= 3	zahl = zahl * 3
- zahl /= 4	zahl = zahl / 4

Bedingungen und Schleifen



- Bedingte Anweisung

```
if (Ausdruck) {  
    Anweisung  
}  
[ else if (Ausdruck) {  
    Anweisung  
} ]  
[ else {  
    Anweisung  
} ]
```




Die []
gehören nicht
zur Syntax!

Bedingungen und Schleifen



Beispiel:

let n = ...;



Zuweisung
einer Zahl

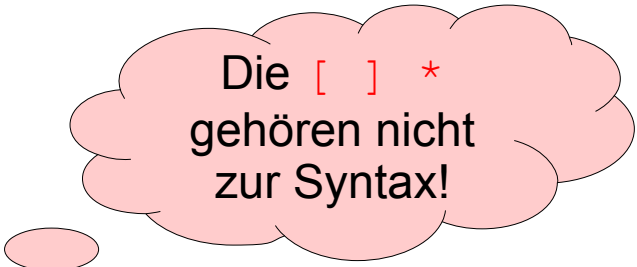
```
if (n === Math.pow(2, 2)) {  
  console.log('22');  
} else if (n === Math.pow(2, 3)) {  
  console.log('23');  
} else {  
  console.log('Kein Ergebnis gefunden');  
}
```

Bedingungen und Schleifen



- Fallunterscheidung switch

```
switch (Ausdruck) {  
  [ case Ausdruck:  
    Anweisung  
    [ break; ] ] *  
  [ default:  
    Anweisung ]  
}
```



Die [] *
gehören nicht
zur Syntax!


→ JavaScript erlaubt hinter `case` beliebige Ausdrücke, nicht nur Konstanten!

Bedingungen und Schleifen



Beispiel:

let n = ...;



Zuweisung
einer Zahl

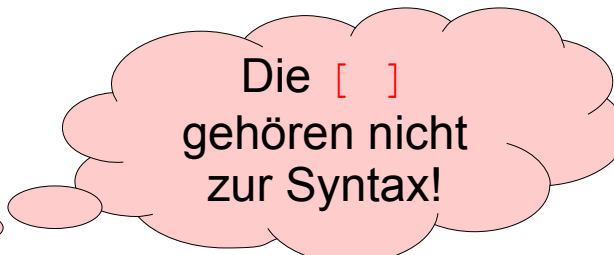
```
switch (n) {  
  case Math.pow(2, 2):  
    console.log('22');  
    break;  
  case Math.pow(2, 3):  
    console.log('23');  
    break;  
  default:  
    console.log('Kein Ergebnis gefunden');  
}
```

Bedingungen und Schleifen



- while Schleife

```
while (Ausdruck) {  
    [ Anweisungsblock ]  
}
```



Die []
gehören nicht
zur Syntax!

- Block solange ausführen, bis die Bedingung nicht mehr erfüllt ist

- do-while Schleife

```
do {  
    [ Anweisungsblock ]  
} while (Ausdruck)
```

- mindestens einmal Ausführen, dann solange bis Bedingung nicht mehr erfüllt ist

Bedingungen und Schleifen



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Beispiel:

```
let i = 0;  
do {  
    console.log(i);  
    i += 1;  
} while (i < 10);
```

Bedingungen und Schleifen



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- for-Schleife

```
for (Initialisierung; Ausdruck; Anweisung) {  
    [ Anweisungsblock ]  
}
```

- Beispiel for-Schleife:

```
for (i = 0; i < 10; i += 1) {  
    console.log(i);  
}
```

Die []
gehören nicht
zur Syntax!

- for/in-Schleife

```
for (Variable in Objekt){  
    [ Anweisungsblock ]  
}
```

Objekte werden
später behandelt

- Beispiel for/in-Schleife:

```
for (p in o) {  
    console.log(o[p]); // den Wert einer Eigenschaft ausgeben  
}
```

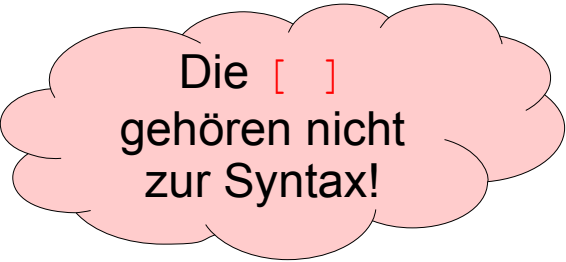
Funktionen



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Definition**

```
function nameDerFunktion() {  
    [ Anweisungsblock ]  
}
```



Die []
gehören nicht
zur Syntax!

- **Aufruf der Funktion**

```
nameDerFunktion();
```

- **Argumente getrennt durch Komma angeben**

```
function nameDerFunktion(param1, param2, ...) {  
    [ Anweisungsblock ]  
}
```

- **Rückgabewert mit Anweisung `return()` bestimmen**

Funktionen



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Beispiel:**

```
function quadrat(x) {  
    let produkt;  
    produkt = x * x;  
    return (produkt);  
}
```

```
console.log(quadrat(4));
```

- Funktionen lassen sich auch schachteln, dürfen aber nicht in Schleifen vorkommen

Beispiel:

```
function hypothenuse(a, b) {  
    function quadrat(x) {  
        return (x * x);  
    }  
    return Math.sqrt(quadrat(a) + quadrat(b));  
}
```



- Funktionen mit variabler Anzahl von Argumenten
 - Eine Funktion muss nicht mit der angegebenen Anzahl an Argumenten aufgerufen werden
 - Werden mehr Argumente angegeben, gelten die zusätzlichen als unbenannt
 - Array zum Zugriff auf *benannte* und *nicht benannte* Argumente
`arguments[]`
 - Anzahl Elemente
`arguments.length`
 - i-te Argument
`arguments[i]`

- Beispiel:

```
function max() {  
    let m = Number.NEGATIVE_INFINITY;  
    let i;  
    // Suche das größte aller Argumente  
    for (i = 0; i < arguments.length; i += 1) {  
        if (arguments[i] > m) {  
            m = arguments[i];  
        }  
    }  
    return (m); // Größtes zurück geben  
}  
console.log(max(1, 5, 2, 67, 23));
```



- Objektorientierung in JavaScript
 - Ein Objekt ist eine ungeordnete Sammlung von Eigenschaften, die jeweils einen Namen und einen Wert haben.
 - Objekte können Eigenschaften eines anderen Objekts, das als *Prototyp* bezeichnet wird, erben.
 - Objekte sind *dynamisch* – Eigenschaften können hinzugefügt und gelöscht werden.
 - Objekte werden mittels einer *Referenz* in einer Variable gespeichert. Durch $y = x$ wird die Referenz auf das Objekt kopiert.
- Drei Schreibweisen
 - Prototypische Objektorientierung
 - Pseudoklassische Objektorientierung
 - Objektorientierung mit Klassensyntax (ab ECMAScript2015)

Prototypische Objektorientierung

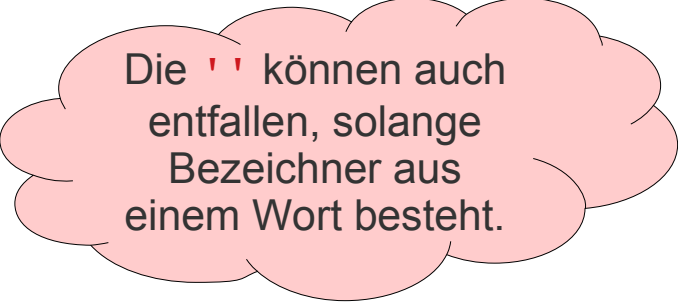


- Bei der prototypischen Vererbung erben Objekte von Objekten (nicht Klassen von Klassen).
- Objekte lassen sich als **Objektliterale** erstellen und initialisieren

Sind kommabasierte Listen von Name/Wert-Paaren.

```
let empty = {};
```

```
let animal = {  
  'name': '',  
  'age': 0,  
  'eat': function (food) {  
    console.log('Mmpf mmpf, ' + food + '!');  
  }  
};
```



Die '' können auch entfallen, solange Bezeichner aus einem Wort besteht.

Prototypische Objektorientierung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Objekte erben von Objekten**
`let cat = Object.create(animal);`
- **Methoden aufrufen**
`cat.eat('mouse');`
- **Eigenschaften zugreifen**
`console.log(cat.age);`
- **Methoden und Eigenschaften im Erben definieren**
`cat.meow = function() {
 console.log('Miauuuuu!');
};`

`cat.meow();`

→ Auf gleiche Weise können Methoden überschrieben werden.

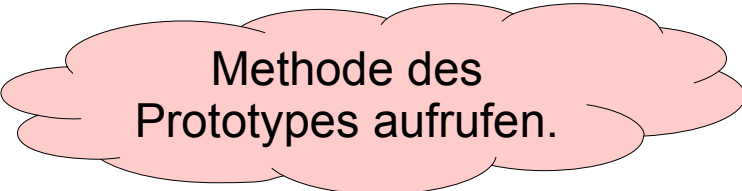
Prototypische Objektorientierung



- **Prototypekette**

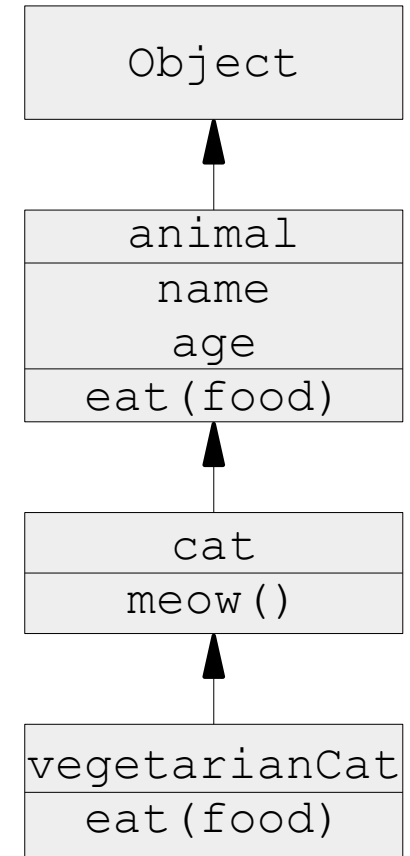
```
let vegetarianCat = Object.create(cat);
```

```
vegetarianCat.eat = function (food) {  
  if (food.indexOf('mouse') >= 0) {  
    console.log('I don't like mice!');  
  } else {  
    this.__proto__.eat(food);  
  }  
};
```



Methode des
Prototypes aufrufen.

```
vegetarianCat.eat('mouse');
```





- Objekte lassen sich mit `Object.create()` erstellen und initialisieren.
 - Erstellt ein neues Objekt und setzt ihr erstes Argument als Prototyp
 - ```
let o1 = Object.create({'x': 1, 'y': 2});
// o1 erbt die Eigenschaften x und y
```
  - ```
let o2 = Object.create(Object.prototype);  
// o2 ist wie {} oder new Object();
```
- Die Möglichkeit, neue Objekte mit frei gewähltem Prototyp zu erstellen, ist äußerst mächtig.

Pseudoklassische Objektorientierung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Eigene Klassen definieren
 - Eigenschaften müssen nicht definiert werden. Ab der ersten Zuweisung existieren sie.
 - Konstruktor
 - Funktion mit Namen der Klasse
 - Referenzierung der Eigenschaften mit `this`
 - Muss kein explizites `return` haben
- Beispiel:

```
function Animal(name, age) {  
    this.name = name;  
    this.age = age;  
}
```

```
let fish = new Animal('Nemo', 2);  
console.log(fish.age);
```



- Methoden
 - JavaScript-Funktionen, die durch ein Objekt aufgerufen werden.
 - Auf die Eigenschaften kann mit Schlüsselwort `this` zugegriffen werden.
 - Eine Methode wird als solche im Prototyp bekannt gemacht.

- Beispiel:

```
// Definiere Konstruktor
function Animal(name, age) {
    this.name = name;
    this.age = age;
}

// Definiere Methode
Animal.prototype.eat = function (food) {
    console.log ('Mmpf mmpf, ' + food + '!');
};

let fish = new Animal('Nemo', 2);
fish.eat('alga');
```



- Objekte haben einen Satz eigener Eigenschaften und erben zusätzlich einen Satz Eigenschaften vom Prototyp.
- Setzen der Oberklasse als Prototyp

```
// Unterklassenkonstruktor
function Cat(name, age, type) {
    // Super-Konstruktor als erstes aufrufen
    Animal.call(this, name, age);
    // Eigenschaften initialisieren
    this.type = type;
}
```

```
// Sub-Klasse erbt von Super-Klasse
Cat.prototype = new Animal();
```

```
// korrigieren des Konstruktors
Cat.prototype.constructor = Cat;
```

Pseudoklassische Objektorientierung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Zusätzliche Methoden des Erben**

```
Cat.prototype.meow = function() {  
    console.log('Miauuuuu!');  
};
```

- **Ausprägen eines Objekts**

```
let cat = new Cat('Kitty', 2, 'Angora');  
cat.meow();
```

- **Methoden der Oberklasse aufrufen**

```
Cat.prototype.eat = function (food) {  
    Animal.prototype.eat.call(this, food);  
};
```

Objektorientierung mit Klassensyntax



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Ab ECMAScript2015

- **Klassendefinition**

```
class Animal {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    eat(food) {  
        console.log ('Mmpf mmpf, ' + food + '!');  
    }  
}
```

- **Ausprägen eines Objekts**

```
let snake = new Animal('Hydra', 4);  
snake.eat('mouse');
```



- Getter und Setter

```
class Animal {  
    constructor(name, age) {  
        this._name = name;  
        this._age = age;  
    }  
  
    get name() {  
        return (this._name);  
    }  
    set name(name) {  
        this._name = name;  
    }  
}
```

Name der Getter / Setter soll
sich von der Eigenschaft
unterscheiden
(Stacküberlauf vermeiden)

- Nutzung von Getter / Setter

```
let snake = new Animal('Hydra', 4);  
console.log(snake.name);
```




- Vererbung

```
class Cat extends Animal {  
    constructor(name, age, type) {  
        // Super-Konstruktor als erstes aufrufen  
        super(name, age);  
        // Eigenschaften initialisieren  
        this._type = type;  
    }  
    meow() {  
        console.log('Miauuuuu!');  
    }  
}
```

- Wird kein Konstruktor im Erbe definiert, wird implizit der Elternkonstruktor mit allen übergebenen Parametern aufgerufen.

Objektorientierung mit Klassensyntax



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Methoden der Elternklasse aufrufen

```
class Cat extends Animal {  
    [...]  
    eat(food) {  
        super.eat(food);  
    }  
}
```



- Statische Methoden (werden direkt auf der Klasse aufgerufen)

```
class Cat extends Animal {  
    [...]  
    static getCatType() {  
        return {  
            ANGORA: 'Angora',  
            SIAM: 'Siam'  
        };  
    }  
}
```

- Nutzung von statischen Methoden

```
let cat = new Cat('Kitty', 2, Cat.getCatType().ANGORA);
```



- Eigenschaften (ab ECMAScript 5)
 - Haben einen Namen und einen Wert.
 - Der **Name** ist ein beliebiger eindeutiger String (auch leer).
 - **Werte** sind beliebige JavaScript-Werte, Methoden, **Getter**- oder **Setter**-Methoden (oder beides).
 - Es wird zwischen *eigenen* und *geerbten* Eigenschaften unterschieden.
- Eigenschaftsattribute:
 - *writable* → Eigenschaft kann gesetzt werden
 - *enumerable* → Eigenschaft durch for/in-Schleife geliefert
 - *configurable* → Eigenschaft löschar und Attribute änderbar



- Objektattribute
 - *prototype* → Referenz auf Objekt, von dem geerbt wird
 - *class* → String, der den Type des Objekts angibt
 - *extensible* → gibt an, ob neue Eigenschaften hinzugefügt werden können
- Kategorien von Objekten
 - *nativ* → in ECMAScript-Spezifikation definiert
 - *Host* → vom Webbrowser definiert/bereitgestellt
 - *benutzerdefiniert* → durch JavaScript Code erstellt



- Eingebaute Klassen können auf die gleiche Weise erweitert werden:

```
String.prototype.endetMit = function (c) {  
    return (c === this.charAt(this.length - 1));  
};
```

- Vererbung
 - Sie fragen Eigenschaft `x` von Objekt `o` an. Hat `o` keine Eigenschaft `x`, so wird der Prototyp von `o` nach Eigenschaft `x` gefragt. Hat dieser auch keine Eigenschaft `x`, so wird der Prototyp des Prototyps gefragt usw. Wird die Eigenschaft `x` nicht gefunden, wird `undefined` zurück gegeben.
 - Prüfen, ob ein Objekt eine Eigenschaft hat oder erbt
`"x" in o; // true: o hat oder erbt Eigenschaft Namens "x"`
 - Prüfen, ob ein Objekt eine Eigenschaft hat
`o.hasOwnProperty("x"); // true: o hat Eigenschaft Namens "x"`



- Getter- und Setter-Funktionen
 - Lesen bzw. Setzen des Wertes einer *Zugriffseigenschaft*.
 - Eine *Dateneigenschaft* wird ohne Getter/Setter angesprochen.
 - *Private Dateneigenschaften* beginnen mit einem \$.
- Eigenschaftsattribute
 - *Dateneigenschaften* haben die Attribute `value`, `writable`, `enumerable` und `configurable`.
 - *Zugriffseigenschaften* haben Attribute `get`, `set`, `enumerable` und `configurable`.



- **Eigenschaftsattribute**

- Eigenschaftsattribute können wie folgt abgefragt werden:

```
var o = { x: 1 };  
Object.getOwnPropertyDescriptor(o, "x");  
  
// Object { configurable: true, enumerable: true,  
//           value: 1, writable: true}
```

- Setzen von Eigenschaftsattributen (es müssen nicht alle Attribute spezifiziert werden)

```
var o = { x: 1 };  
// nicht schreibbare Eigenschaft  
Object.defineProperty(o, "x", { configurable: true, enumerable: true,  
                               value: 1, writable: false}  
                               );  
  
// ab jetzt eine Zugriffseigenschaft  
Object.defineProperty(o, "x", { get: function() {  
                                return this.$x;  
                                }  
                                }  
                                );
```




- Prototypen
 - Ein Objekt erbt alle Eigenschaften seines Prototyps.
 - Mit `new` erstellte Objekte nutzen den Wert der `prototype`-Eigenschaft der Konstruktorfunktion als Prototyp.
 - In der Regel hat jedes Objekt einen Prototyp (selten ist dieser `null`). `Object.prototype` ist eines der wenigen Objekte, das keinen Prototyp hat.
 - Beispiel:
`Date.prototype` erbt Eigenschaften von `Object.prototype`.

Mit `new Date()` erstelltes Objekt erbt Eigenschaften von `Date.prototype` und `Object.prototype`. → **Prototypkette**
- Wird eine Eigenschaft `x` von Objekt `o` gelesen, so wird geprüft, ob `o` die Eigenschaft `x` kennt. Ist dies nicht der Fall, so wird das Prototypobjekt von `o` nach der Eigenschaft `x` gefragt. Ist dies auch nicht der Fall, so erfolgt die Abfrage auf den Prototyp des Prototyps, usw.
- Bei einer **Zuweisung** wird auch die Prototypkette untersucht. Handelt es sich um eine schreibgeschützte Eigenschaft, so wird die Zuweisung untersagt. Anderenfalls wird die Eigenschaft im **Ausgangsobjekt** erstellt oder gesetzt.



- Eigenschaften abfragen und setzen

- Werden erhalten mit Punktoperator (.) oder eckigem Klammeroperator ([])
- Beim *Punktoperator* muss rechtsseitiger Operand ein *Bezeichner* sein.
- Beim *Klammeroperand* muss in der Klammer ein *Ausdruck* sein, der zu einem *String* ausgewertet wird oder umgewandelt werden kann.

- Beispiel Eigenschaft abfragen

```
var name = cat.name;  
var type = cat["type"];
```

- Beispiel Eigenschaft setzen

```
cat.age = 6;  
cat["type"] = "Angora";
```

- In ECMAScript 3 waren keine reservierten Worte (z.B. for) als Bezeichner erlaubt. Es musste die Schreibweise mit dem eckigen Klammeroperator genutzt werden.



- In JavaScript werden Objekte als assoziative Arrays betrachtet.
 - Ein Programm kann eine beliebige Anzahl von Eigenschaften auf ein Objekt erstellen.
- Wird der *eckige Klammeroperator* genutzt, so können Eigenschaften während des Programmlaufs *manipuliert* und *erstellt* werden.
- Beispiel

```
var addr = "";
for (i = 0; i < 4; i += 1) {
    addr += customer["address" + i] + "\n";
}
```



- Objektattribute

- **prototype**-Attribut gibt das Objekt an, von dem das Objekt seinen Prototyp erbt.
 - Wird gesetzt, wenn Objekt erstellt wird (`Literal`, `new`, `create()`)
 - Mit der Methode `.isPrototypeOf(o)` kann der Prototyp eines Objekts ermittelt werden.

```
p.isPrototypeOf(o) // => true: o erbt von p
```
- **class**-Attribut ist ein String des Klassennamens. Kann weder gesetzt noch gelesen werden.
- **extensible**-Attribut legt fest, ob neue Eigenschaften hinzugefügt werden können
 - Kann mit `.isExtensible()` abgefragt werden
 - Kann mit `.preventExtension()` gesetzt werden (kann nicht rückgängig gemacht werden)
 - `.seal()` setzt *configurable* auf `false` (vgl. `.isSealed()`)
 - `.freeze()` schützt zusätzlich auch Dateneigenschaften (vgl. `.isFrozen()`)



- **JavaScript Object Notation**
- Werden Objekte einer Art nur einmal benötigt, so muss keine Klasse definiert werden. Sie werden als Objektliteral geschrieben.
- JSON unterstützt nur enumerierbaren Eigenschaften, Methoden u.a. werden ausgelassen.
- Zugriff auf Eigenschaften normal über `.-Operator`
- JSON-Strings von Objekten können mittels `eval()` ausgewertet werden, das kann aber zu Sicherheitsproblemen führen.

```
let litObj = "{ durchmesser: 12746, umfang: 40041 }";  
let erde = eval("(" + litObj + ")");
```
- Abhilfe schafft das Parsen eines JSON-Strings und gleichzeitiges Erzeugen eines Objekts.

```
let erde = JSON.parse(litObj);
```
- Serialisieren eines Objekts

```
s = JSON.stringify(o);
```

JSON



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Für diverse Skriptsprachen gibt es Bibliotheken, die JSON Notationen erstellen können, z.B. für PHP:
 - `string json_encode(mixed $value)`
 - `mixed json_decode(string $json)`

- Werfen einer Ausnahme:

`throw Ausdruck`

- I.d.R. ein Error-Objekt oder eine Unterklasse

- Abfangen einer Ausnahme

```
try {
```

Code, der normalerweise problemlos abläuft

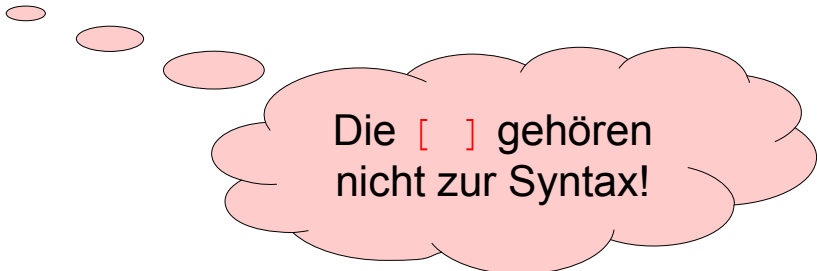
```
} catch (e) {
```

Block, der ausgeführt wird, wenn eine Ausnahme ausgelöst wurde. In `e` steht das Error-Objekt. Es kann immer nur ein `catch`-Zweig angegeben werden.

```
} [ finally {
```

Anweisungen, die unabhängig von einer Ausnahme ausgeführt werden.

```
} ]
```



Die `[]` gehören
nicht zur Syntax!



Beispiel Werfen einer Ausnahme:

```
function fakultaet(x) {  
    // Auf Gültigkeit prüfen  
    if (x < 0) {  
        throw new Error("x darf nicht negativ sein!");  
    }  
    // Berechne Fakultät  
    let f;  
    for (f = 1; x > 1; f *= x, x -= 1) {  
        /* leer */  
    }  
    return(f);  
}
```


Ausnahmebehandlung



- Beispiel Abfangen einer Ausnahme:

```
try {  
    // Berechne Fakultät  
    let f = fakultaet(-5);  
    // Zeige Ergebnis  
    console.log(n + "! = " + f);  
} catch (e) {  
    console.log(e);  
}
```

JavaScript Bibliothek

Zeichenreihen-Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Ausprägung/Zuweisung:**

```
let aString = 'Zeichenreihe';  
let aString = new String('Zeichenr.');
```

- **Länge:**

```
laenge = aString.length;
```

- **Groß- und Kleinschreibung:**

```
aString.toLowerCase();  
aString.toUpperCase();
```

- **Zerlegen einer Zeichenreihe:**

```
aString.split(begrenzer[, grenze])
```

- **begrenzer:** Zeichenreihe, die die Zeichenreihe zerlegt (ab JavaScript 1.2 auch reguläre Ausdrücke – *Perl Syntax*)
- **grenze:** maximale Länge des Rückgabe-Arrays
- **Rückgabewert:** Zeichenreihen-Array der Teile

Die [] gehören nicht zur Syntax!

JavaScript Bibliothek

Zeichenreihen-Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Durchsuchen einer Zeichenreihe

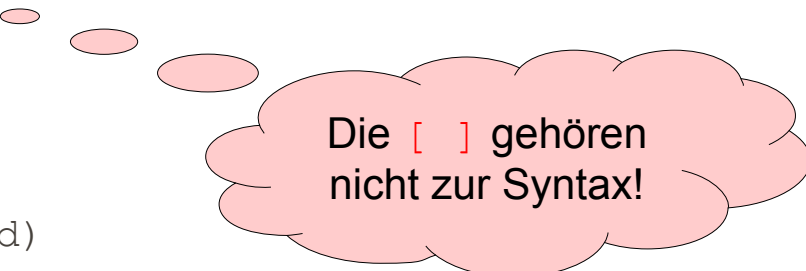
`aString.indexOf(teil[, start])`

- Rückgabewert: Position des nächsten Auftretens oder -1

- Teil-Zeichenreihe

`aString.substring(from[, to])`

- `from`: Index von Anfangsposition
- Rückgabewert: Teil-Zeichenreihe
- Alternativ `aString.slice(start, end)`
(auch negative Werte erlaubt)



Die `[]` gehören
nicht zur Syntax!

JavaScript Bibliothek

Array Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Ausprägung:**

```
aArray = new Array();  
aArray = new Array(groesse);  
aArray = new Array(feld0, feld1,  
                    ..., feldn);  
aArray = [feld0, feld1, ..., feldn];
```
- **Indizes beginnen bei 0**

```
feld1 = aArray[1];
```
- **Zuweisung**

```
aArray[1] = neuerWert;
```

JavaScript Bibliothek

Array Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Verketteten von Arrays**
`aArray.concat(wert,)`
 - `wert`: entweder einzelne Elemente oder ein Array
- **Beispiel:**
`let a = [1,2,3];`
`a.concat(4,5);`
`a.concat([4,5]);`
ergeben jeweils `[1,2,3,4,5]`
- **Array-Länge**
`aArray.length;`

JavaScript Bibliothek

Array Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Letztes Element zurückliefern und entfernen
`aArray.pop()`
 - Rückgabewert: letztes Element
- Anhängen an Array
`aArray.push(wert, ...)`
 - Rückgabewert: Neue Länge des Arrays
- Erstes Element zurückliefern und löschen
`aArray.shift()`
 - Rückgabewert: Erstes Element
- Vorne an Array einfügen
`aArray.unshift(wert, ...)`
 - Rückgabewert: Neue Länge des Arrays

JavaScript Bibliothek

Array Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Sortieren nach UTF-16 Codepoints
`aArray.sort()`
- Sortieren mit gegebener Vergleichsfunktion
`aArray.sort(sortierfunktion)`
- Beispiel Sortieren:

```
function numberorder(a,b) {  
    return(a-b);  
}
```

```
a = new Array(33,4,1111,222);  
a.sort();
```

Ausgabe: 1111, 222, 33, 4

```
a.sort(numberorder);
```

Ausgabe: 4, 33, 222, 1111



- Wenn eine Zeichenreihe in einem numerischen Kontext verwendet wird, wird dieser automatisch umgewandelt.
- Explizite Umwandlung in eine Zahl
`let number = Number(string_value);`
- Wenn Zeichenreihen nicht in eine Zahl umgewandelt werden können, wird der Wert NaN (not a number) zurückgegeben.
- Explizite Umwandlung in eine Zeichenreihe
`let n = 12.3456789;`
`n.toFixed(2); // "12.35"`
`n.toPrecision(3); // "12.3"`
`n.toExponential(4); // "1.2346e+1"`

Es wird automatisch gerundet!


JavaScript Bibliothek

Datums- und Uhrzeit Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Ausprägung:**
`aDate = new Date();`
 Heutige Uhrzeit und Datum
`aDate = new Date(jahr, monat [, tag[, stunde[,`
 `minute[, sekunden[, ms]]]]]);`
- **Datum auslesen**
`aDate.getDay(), aDate.getMonth(), ...`
- **Datum setzen**
`aDate.setDay(), aDate.setMonth(), ...`
- **Umwandlung in Zeichenreihe**
`aDate.toString();`



Die `[]` gehören
nicht zur Syntax!



- Kombination von JavaScript und HTML
- W3C-DOM (Document Object Model) ist ein Standard, wie Elemente eines HTML-Dokuments mit Skriptsprachen anzusteuern sind
- Browser haben ggf. noch zusätzliche proprietäre API
- Elemente werden in einer Baumstruktur abgebildet
- Zum Auffinden eines Elements stehen folgende Methoden bereit:
 - `Element document.getElementById(String id)`
`id` muss dokumentenweit eindeutig sein
 - `Node[] document.getElementsByTagName(String tagname)`
Zurückgegeben wird ein Array mit allen Elementen die dem `tagname` entsprechen

Dynamic HTML



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Beispiel:

```
<input type="text" id="i1" />
```

```
<script>  
    document.getElementById('i1').value = '42';  
    document.getElementById('i1').style.color = '#ff0000';  
</script>
```

Dynamic HTML

Ereignisbehandlung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Ereignisbehandler werden als Attribut von HTML-Elementen notiert und beginnen mit `on`. Als Wert wird i.d.R. eine JavaScript Funktion notiert.
- Ereignisse sind immer die Reaktion auf die Aktion des Benutzers
- Mögliche Einsatzgebiete sind Gültigkeitsprüfungen vor dem Absenden eines Formulars oder das Starten einer Animation beim Laden eines Dokuments

Event-Handler	Beschreibung
<code>abort</code>	Das Laden der Grafik wird abgebrochen
<code>blur</code>	Der Eingabefokus wurde entzogen
<code>change</code>	Der Text eines Elements wurde geändert
<code>click</code>	Mausklick
<code>dblclick</code>	Mausdoppelklick
<code>error</code>	Ein Fehler ist aufgetreten
<code>focus</code>	Der Eingabefocus wird auf Element gesetzt
<code>keydown</code>	Eine Taste wird gedrückt
<code>keypress</code>	Tastendruck
<code>keyup</code>	Taste wird gelöst

Dynamic HTML

Ereignisbehandlung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Event-Handler	Beschreibung
load	Ein Dokument wird geladen *
mousedown	Eine Maustaste wird gedrückt
mousemove	Die Maus wird bewegt
mouseout	Die Maus verlässt ein Element
mouseover	Die Maus steht über einem Element
mouseup	Die Maustaste wird gelöst
reset	Formulardaten werden zurückgesetzt
resize	Die Fenstergröße wird geändert
select	Text wird ausgewählt
submit	Ein Formular wird versendet
unload	Das Dokument wird geschlossen

* Alternativ zu `load` kann das `script`-Tag durch Attribut `defer` ergänzt werden:

```
<script defer>...</script>
```

Die Funktionalität ist ähnlich wie `load`, wartet aber nur auf DOM
(nicht bis Bilder geladen)

Dynamic HTML

Ereignisbehandlung



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Beispiel:** change
Element wurde geändert und wird gerade verlassen

```
<input type="text" onchange="window.alert('change')" value="Text" />
```

- **Beispiel:** mouseover/mouseout

Wird ausgelöst, sobald der Anwender die Maus in den Anzeigebereich eines Elements bewegt bzw. diesen verlässt.

```
<h1 style="color:#000000"  
  onmouseover="this.style.color = '#FF0000'"  
  onmouseout="this.style.color = '#000000'">  
  mouseout und mouseover  
</h1>
```

Dynamic HTML

Ereignisbehandlung



- **Beispiel** onchange/getElementById

```
<input id="s1" type="text"
      onchange="textChanged();" value="0" /> +
<input id="s2" type="text"
      onchange="textChanged();" value="0" /> =
<input id="s3" type="text" />
```

```
function textChanged() {
    let s1, s2, s3;
    s1 = document.getElementById("s1").value;
    s2 = document.getElementById("s2").value;
    s3 = Number(s1) + Number(s2);
    document.getElementById("s3").value = s3;
}
```

Dynamic HTML

Ereignisbehandlung (DOM Level 2)



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Ein Ereignis-Listener zu einem HTML Element hinzufügen:

```
void Element.addEventListener(  
    String type,  
    Function listener,  
    boolean useCapture);
```

 - `type`: Ereignis, bei dem der Listener eingebunden werden soll (jeweils ohne das führende `on`)
 - `listener`: Funktion, die aufgerufen werden soll
 - `useCapture`: `true`, falls ein Listener zu der *capturing Phase* (nächste F.) zugeordnet werden soll, i.d.R. `false`

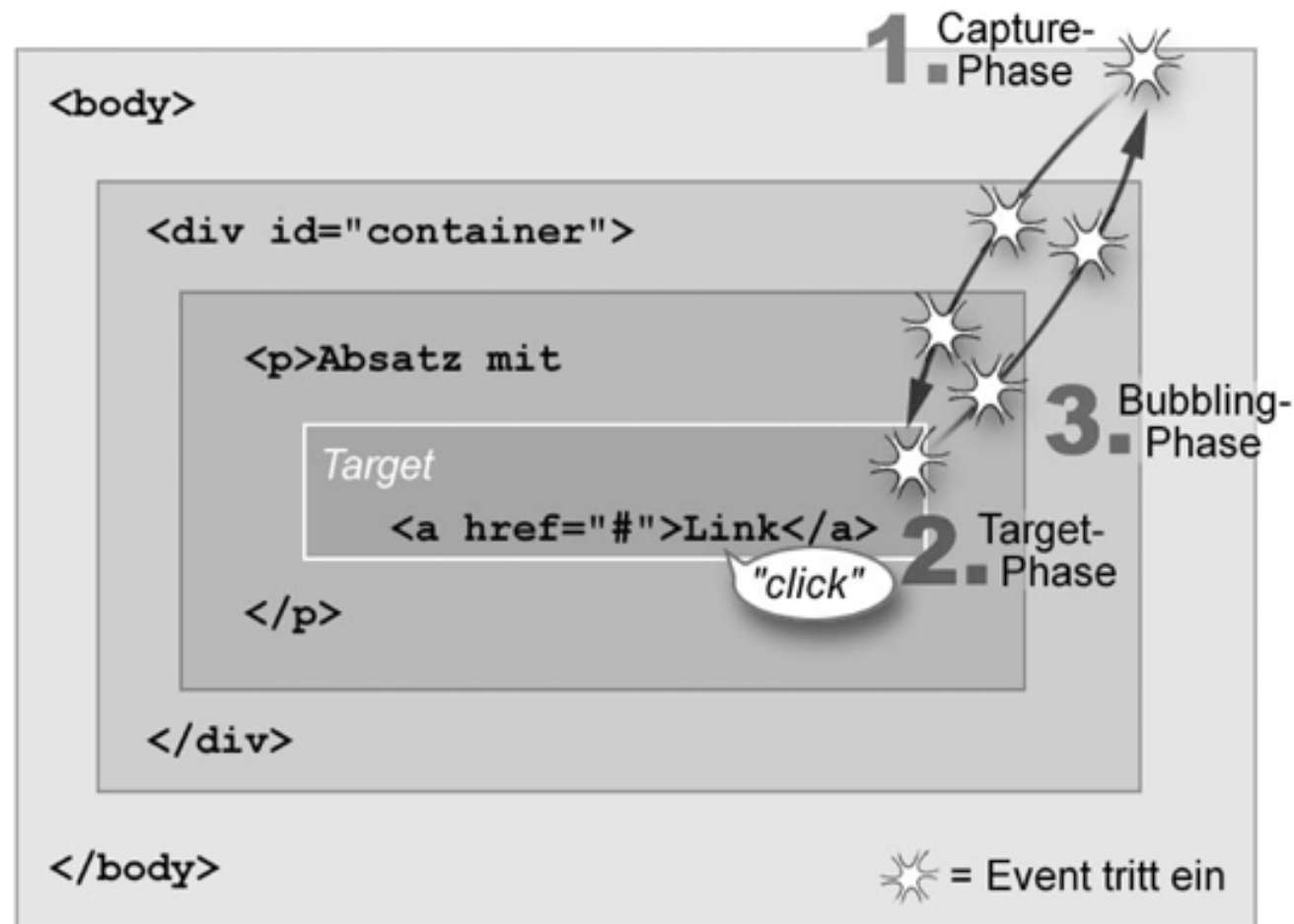
Dynamic HTML

Ereignisbehandlung (DOM Level 2)



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Ereignisse werden in 3 Phasen abgearbeitet:
 - *Capturing Phase*: vom Dokument bis zum Zielobjekt
 - *Target Knoten*: am Zielobjekt selbst
 - *Bubbling Phase*: vom Zielobjekt bis zum Dokument



Dynamic HTML

Ereignisbehandlung (DOM Level 2)



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Es können mehrere Listener pro Ereignis registriert werden, eine Reihenfolge kann nicht bestimmt werden.
- Beispiel:

```
<input id="s1" type="text" />
```

```
let input = document.getElementById("s1");
```

```
input.addEventListener(  
    'change',  
    function (e) {  
        window.alert('Ereignis change');  
    },  
    false  
);
```

Dynamic HTML

Ereignisbehandlung (DOM Level 2)



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Event-Objekt

Eigenschaft	Erklärung
<code>e.target</code>	Das DOM-Element, an dem das Ereignis stattfand
<code>e.currentTarget</code>	Das DOM-Element, das während der Capturing-/Bubbling-Phase Ziel des Ereignisses ist
<code>e.clientX/Y</code>	Mausposition relativ zum Browserfenster
<code>e.timeStamp</code>	Zeitstempel des Ereignisses (an <code>Date()</code> zu übergeben)
<code>e.type</code>	Die Art des Ereignisses
<code>e.keyCode</code>	Die Keyboard-Taste, die betätigt wurde
...	...

- Durch

`void Event.stopPropagation();`

kann eine Verbreitung des Ereignisses angehalten werden.

- Durch

`void Event.preventDefault();`

wird Ereignis abgebrochen, eine Standardverarbeitung findet nicht statt.

Dynamic HTML

Dokument Eigenschaften und Methoden



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- `document.title`: Titel der Seite
- `document.URL`: URI der Seite (ro)
- `document.lastModified`: Datum der letzten Änderung (ro)
- Anhängen von Daten an ein Dokument
 - `document.write(wert, ...);`
 - `document.writeln(wert, ...);`
 - Hinweis: Darf nur verwendet werden, während DOM geladen wird.
- ...
- **Besonderheiten bei unterschiedlichen Browsern!**

Dynamic HTML

Informationen vom Browser



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Nur Kernkompatibilität zwischen verschiedenen Browsern (W3C-DOM)
- Für verschiedenen Browser muss ggf. eine unterschiedliche Programmierung vorgenommen werden
- Abfragen von Browser-Detailinformationen
 - `navigator.appName`
 - `navigator.appVersion`
 - `navigator.cookieEnabled`
 - `navigator.language` (**Mozilla**) / `userLanguage` (**IE**)
 - `navigator.platform`

Dynamic HTML

Bildschirm Informationen



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- `screen.width`
- `screen.height`
- `screen.availWidth` (z.B. ohne Taskleiste)
- `screen.availHeight`
- `screen.colorDepth` (in Bit)

Dynamic HTML

Plugin Informationen



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Informationen über installierte Plugins
 - `navigator.plugins[i];`
 - `navigator.plugins['name'];`
- `navigator.plugins[i].name` - **Name**
- `navigator.plugins[i].description` - **Beschreibung**
- `navigator.plugins[i].length` - **Anzahl von MIME-Typen**
- `navigator.plugins[i][j]` - **MIME-Typ j**
- **Beispiel installierte Plugins abfragen**

```
for (i = 0; i < navigator.plugins.length; i++) {  
    console.log(navigator.plugins[i].name);  
}
```
- **Analog dazu kann geprüft werden, ob ein Plugin installiert ist**

```
if (navigator.plugins["Shockwave Flash"]) ...
```

Dynamic HTML

Methoden für Fenster



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Fenster öffnen**

```
aWindow = window.open(uri[, name[, features]]);
```

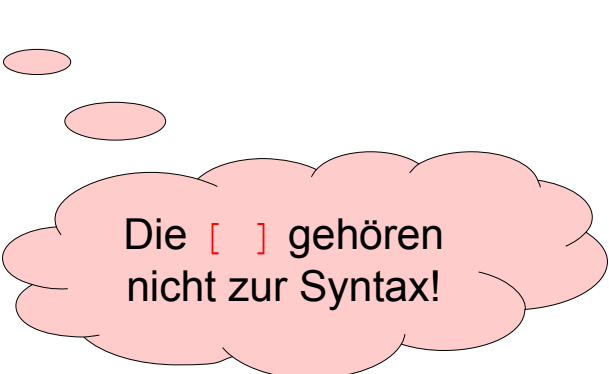
- uri: Anzuzeigendes Dokument
- name: Name für das neue Fenster
- features: Anzeige-Optionen (menubar, ...)
- Rückgabewert: Referenz auf das Fenster

- **Schließen eines Fensters der Referenz aWindow**

```
aWindow.close();
```

- **Größe verändern**

```
aWindow.resizeTo(breite, hoehe);
```



Die [] gehören
nicht zur Syntax!

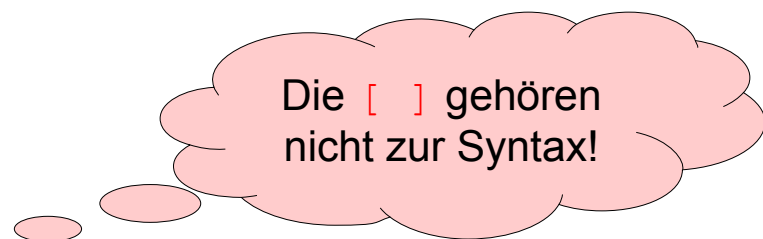
Dynamic HTML

Methoden für Fenster



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Zeigt eine Nachricht mit OK-Knopf an
`window.alert(meldung);`
- Stellt eine Ja/Nein Frage
`antwort = window.confirm(frage);`
 - Rückgabewert: True bei Ja
- Zeigt einen Eingabedialog
`window.prompt(meldung[, default]);`
 - Rückgabe: Eingabe vom Benutzer oder `null`, falls Abbruch gedrückt
- Druckdialog aufrufen
`window.print();`



Die [] gehören
nicht zur Syntax!

Dynamic HTML

Methoden für Fenster



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Einmalige zeitverzögerte Ausführung von Code

```
window.setTimeout(code, aufschub);
```

- code: auszuführender JavaScript Code
- aufschub: Millisekunden, die verstreichen sollen

- Periodische Ausführung von Code

```
window.setInterval(code, intervall);
```

- code: auszuführender JavaScript Code
- intervall: Zeitintervall in Millisekunden

Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Jedes HTML Element wird in JavaScript als Objekt dargestellt.
- Die Objekte besitzen Eigenschaften und Methoden, um diese zu verändern.
- Bereits kennen gelernt haben wir die Eigenschaft `value` vom `<input>`-Tag

```
document.getElementById('i1').value = '42';
```

→ Alle HTML-Eigenschaften haben in JavaScript den gleichen Namen

Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Ein HTML Element erzeugen
`Element Document.createElement(String tagName)`
`throws DOMException;`
- Einen Text-Knoten erzeugen
`Text Document.createTextNode(String text);`
- HTML Dokument wird als Baum abgebildet. Ein Kind-Knoten lässt sich wie folgt einfügen
`Node Node.appendChild(Node newChild)`
`throws DOMException;`

Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Beispiel Element erzeugen

```
<p id="absatz">Textabsatz</p>
```

```
<script type="text/javascript">
  let absatz = document.getElementById('absatz');
  let input = document.createElement('input');
  input.type = 'text';
  absatz.appendChild(input);
  let text = document.createTextNode('Dies ist ein Text!');
  absatz.appendChild(text);
</script>
```

Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Beispielhaft wird hier die Klasse `Table` vorgestellt
- Eigenschaften von `Table`
 - `HTMLElement caption`
Eine Referenz zu dem `<caption>` Elementen der Tabelle oder `null`, falls keins vorhanden
 - `readonly HTMLCollection rows`
Eine `HTMLCollection` von `TableRow` Objekten, die alle Zeilen in der Tabelle repräsentieren.
 - `String border`
Breite des Randes der Tabelle
- Methoden von `Table`
 - `HTMLElement createCaption()`
Gibt eine Referenz zu der `<caption>` der Tabelle zurück. Falls keine existiert, wird eine neue zuerst erzeugt.
 - `void deleteCaption()`
Löscht eine `<caption>` Element von der Tabelle, falls sie eins hat.
 - `HTMLElement insertRow(long index)`
`throws DOMException`
Fügt ein neues, leeres `<tr>` Element an der gegebenen Position in der Tabelle ein. Der Rückgabewert ist die neu eingefügte `TableRow`.

Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Eine Tabellenzeile wird durch die Klasse `TableRow` implementiert
- Sie hat folgende Methoden
 - `HTMLElement insertCell(long index)`
 throws `DOMException`
 Fügt eine neue Zelle an der gegebenen Position ein. Zurückgegeben wird ein `TableCell` Objekt der neu erzeugten Zelle.
 - `void deleteCell(long index)`
 throws `DOMException`
 Löscht die angegebene Zelle

Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Beispiel (HTML-Teil):

```
<table border="1" id="tab1">
  <tr>
    <th>Spalte 1</th>
    <th>Spalte 2</th>
  </tr>
</table>
```


Dynamic HTML

HTML Elemente



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Beispiel (JavaScript-Teil):

```
// Tabelle finden
let tab1 = document.getElementById('tab1');
// Eine Zeile einfügen (Nummerierung von 0 an)
let row2 = tab1.insertRow(1);
//Zellen einfügen (Nummerierung von 0 an)
let cell1 = row2.insertCell(0);
let cell2 = row2.insertCell(1);
// Zelleninhalt setzen
let textNode1 = document.createTextNode('C1');
cell1.appendChild(textNode1);
let textNode2 = document.createTextNode('C2');
cell2.appendChild(textNode2);
```

Dynamic HTML

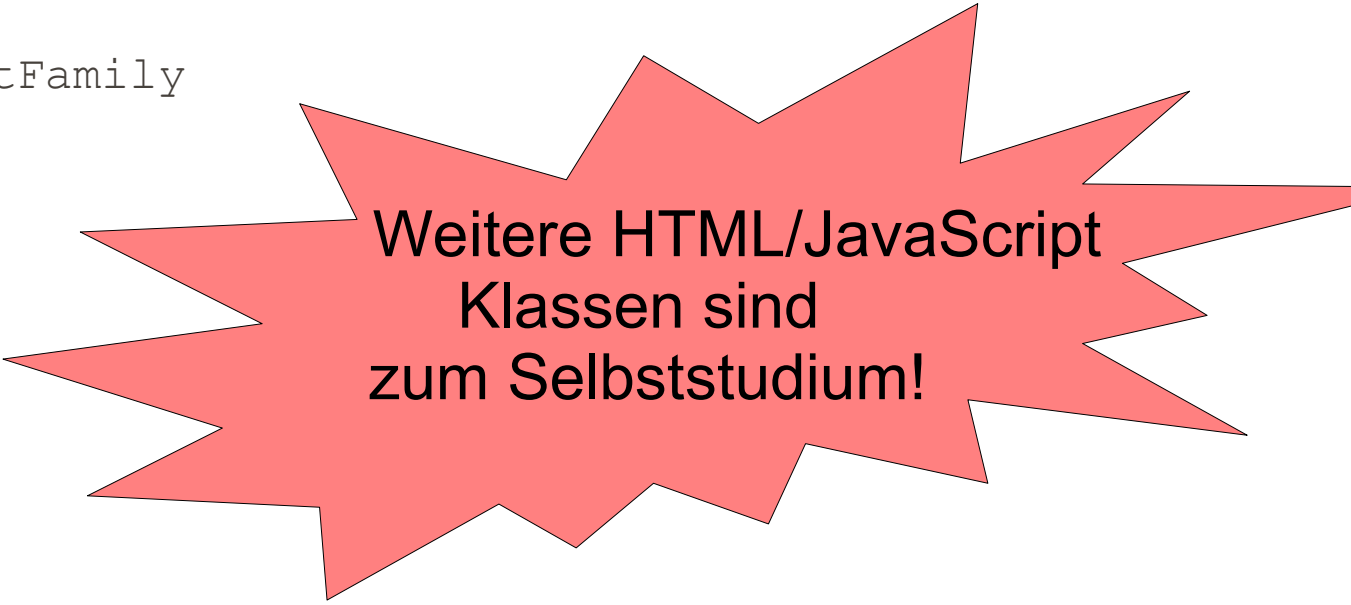
CSS Regeln



Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- CSS Regeln eines HTML Elements können durch die Eigenschaft `CSS2Properties HTMLElement.style` angesprochen werden
- Die Klasse `CSS2Properties` enthält Eigenschaften, die den CSS Regeln entsprechen:
`style.color = '#ff0000'`
 - Regeln, die – enthalten, werden stattdessen zusammen mit Großbuchstaben geschrieben

`font-family` → `fontFamily`



Weitere HTML/JavaScript
Klassen sind
zum Selbststudium!



- Ackermann, P.: JavaScript – Das umfassende Handbuch, Auflage 1, Rheinwerkverlag, 2016
- Flanagan, D.: JavaScript – Das umfassende Referenzwerk, Auflage 6, 1. korr. Nachdruck, O'Reilly, 2014
- Crockford, D.: JavaScript – The Good Parts, O'Reilly, 2008
- Kennedy, B. und Musciano, C.: HTML & XHTML - Das umfangreiche Referenzwerk, Auflage 4, O'Reilly, 2003
- Lubkowitz, M: Webseiten programmieren und gestalten, 2. Auflage, Galileo Press, 2006
- Vallendorf, M. und Bongers, F.: jQuery – Das Praxisbuch, Galileo Computing, 2010