



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 (European Computer Manufactures Association) standardisiert (ISO/IEC 16262:1998) .
- 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
 - === gleich
 - !== 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

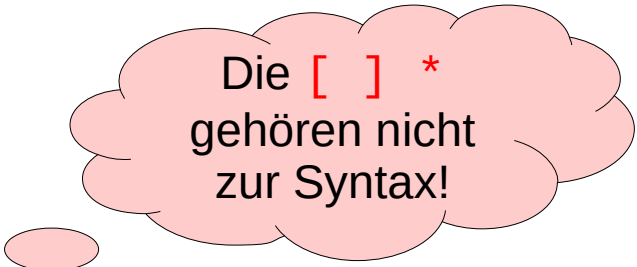
```
if (n === Math.pow(2, 2)) {  
  console.log('22');  
} else if (n === Math.pow(2, 3)) {  
  console.log('23');  
} else {  
  console.log('Kein Ergebnis gefunden');  
}
```

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

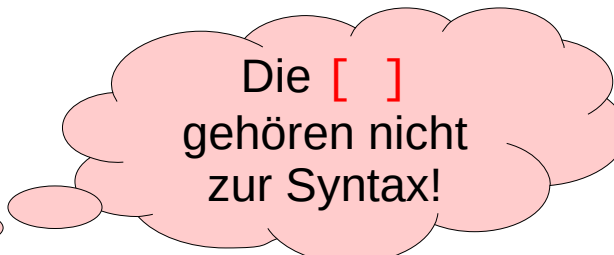
```
switch (n) {  
  case Math.pow(2, 2):  
    console.log('22');  
    break;  
  case Math.pow(2, 3):  
    console.log('23');  
    break;  
  default:  
    console.log('Kein Ergebnis gefunden');  
}
```

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



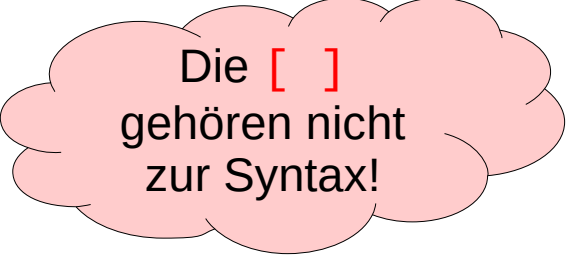
Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Definition

```
function nameDerFunktion() {  
    [ Anweisungsblock ]  
}
```
- Aufruf der Funktion

```
nameDerFunktion();
```
- Argumente getrennt durch Komma angeben

```
function nameDerFunktion(param1, param2, ...) {  
    [ Anweisungsblock ]  
}
```
- Rückgabewert mit Anweisung `return()` bestimmen



Die []
gehören nicht
zur Syntax!

Funktionen



- 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

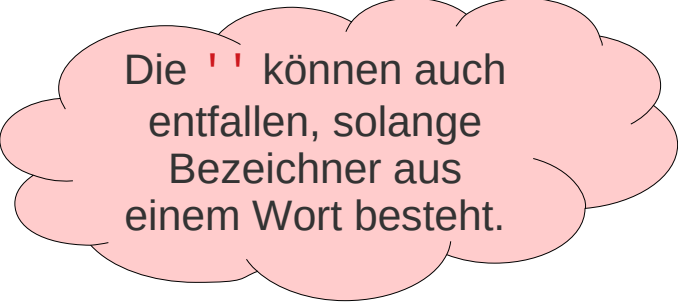


- Bei der prototypischen Vererbung erben Objekte von Objekten (nicht Klassen von Klassen).
- Objekte lassen sich als **Objektliterale** erstellen und initialisieren

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.

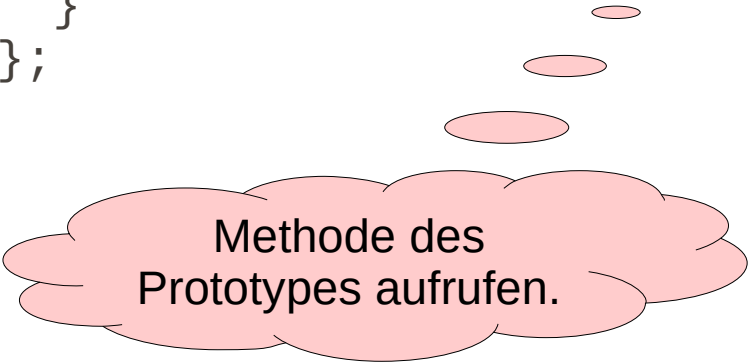
Prototypische Objektorientierung



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

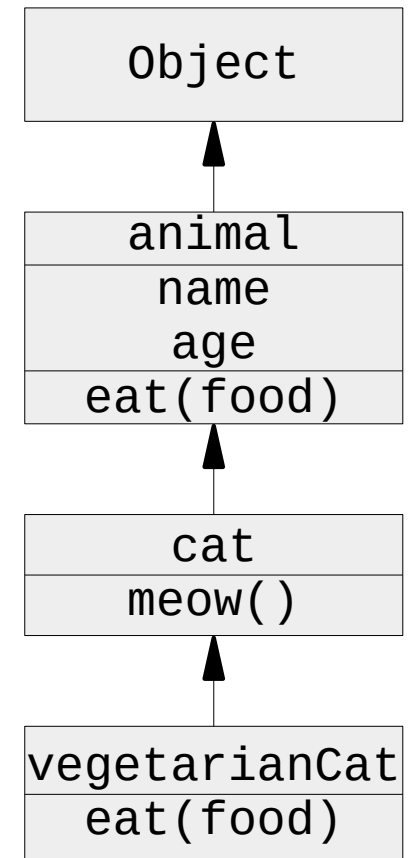
- Prototypekette
`let vegetarianCat = Object.create(cat);`

```
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);
```

Pseudoklassische Objektorientierung



- 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');
```

Pseudoklassische Objektorientierung



- 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



- 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');
```

Objektorientierung mit Klassensyntax



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

- 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



- 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);`