04 JavaScript

Multimedia Engineering (PS siehe 36609b)

Nils Hellwig, M.Sc.
Lehrstuhl für Medieninformatik
FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE

Universität Regensburg

Universität Regensburg

Überblick Themen

- Was ist JavaScript?
- Variablen und Datentypen
- Funktionen
- Comparisons und logische Operatoren
- Objekte
- Klassen
- Arrays
- DOM-Manipulation

Universität Regensburg

Was ist JavaScript?

- JavaScript ist eine Programmiersprache für die Entwicklung interaktiver Websites
- JavaScript ermöglicht es, dynamische Webinhalte zu erstellen, indem es den HTML- und CSS-Code einer Webseite manipuliert
- JavaScript wurde 1995 von Netscape entwickelt (Originalname: „LiveScript“)
- Obwohl beide Sprachen eine ähnliche Syntax aufweisen und der Name ähnlich klingt, sind Java und JavaScript grundlegend unterschiedliche Programmiersprachen

ECMAScript

- ECMAScript (kurz ES) ist der offizielle Standard, auf dem JavaScript basiert
- ECMAScript definiert die Syntax, Struktur und Funktionen der Programmiersprache JavaScript, die von allen modernen Browsern unterstützt wird
- Neue Versionen von ECMAScript werden regelmäßig veröffentlicht, um die Sprache zu verbessern und neue Funktionen hinzuzufügen
- Die aktuelle Version von ECMAScript ist ECMAScript 2024 (Edition 15), die im Juni 2024 veröffentlicht wurde (Versionsgeschichte: https://en.wikipedia.org/wiki/ECMAScript_version_history)

JavaScript in HTML einbinden

Skripte können in den `<body>` - oder in den `<head>` - Abschnitt einer HTML-Seite oder in beide eingefügt werden.

Beispiel:

```
<!DOCTYPE html>
<html>
 <head>
  <script>
    function myFunction() {
      document.getElementById("demo").innerHTML = "Paragraph changed.";
    }
  </script>
 </head>
 <body>
   <h2>Demo JavaScript in Head</h2>
   <p id="demo">A Paragraph</p>
   <button type="button" onclick="myFunction()">Try it</button>
 </body>
</html>
```

Universität Regensburg

JavaScript in HTML einbinden

Skripte können auch mithilfe des `src`– Attributs eingebunden werden (beliebig viele):

Beispiel:

```
<script src="myScript.js"></script>
```

- Separation von HTML und Code (bessere Wartbarkeit/Übersichtlichkeit)
- Alternativ kann JavaScript-Code auch über eine URL geladen werden:

  ```
  <script src="https://www.w3schools.com/js/myScript.js"></script>
  ```

Universität Regensburg

JavaScript Statements

- Ein Computerprogramm ist eine Liste von "Anweisungen", die von einem Computer "ausgeführt" werden sollen.
- Ein JavaScript-Programm ist eine Liste von Programmieranweisungen.
- Die Anweisungen werden nacheinander in der gleichen Reihenfolge ausgeführt, in der sie geschrieben wurden.

Universität Regensburg

JavaScript Statements

- Semikolons trennen JavaScript-Anweisungen
- Am Ende jeder Anweisung sollte ein Semikolon sein:

```
let a, b, c;   // 3 Variablen deklarieren
a = 5;         // a den Wert 5 zuweisen
b = 6;         // b den Wert 6 zuweisen
c = a + b;     // c die Summe aus a und b zuweisen
```

- Im Internet findet man häufig Code ohne Semikolon
- Das Beenden von Anweisungen mit Semikolon ist nicht erforderlich, aber empfehlenswert um unerwartete Verhaltensweisen zu vermeiden

Universität Regensburg

JavaScript Funktionen

- JavaScript-Anweisungen können in geschweifter Klammern `{...}` gruppiert werden
- Der Zweck von Codeblöcken besteht darin, Anweisungen zu definieren, die gemeinsam ausgeführt werden sollen
- Eine Möglichkeit dafür sind JavaScript-Funktionen:

```
function myFunction() {
   document.getElementById("demo1").innerHTML = "Hello Max!";
   document.getElementById("demo2").innerHTML = "How are you?";
}
myFunction();
```

Universität Regensburg

JavaScript Funktionen

- Funktionen können optional Parameter akzeptieren und Werte zurückgeben (mit return), die von anderen Code-Blöcken im Programm wiederverwendet werden können:

```
// Definition einer benannten Funktion
function addNumbers(a, b) {
   return a + b;
}

// Aufruf der Funktion und Speicherung des Rückgabewerts in einer Variablen
let sum = addNumbers(2, 3);

// Ausgabe des Ergebnisses
console.log(sum); // Output: 5
```

Universität Regensburg

JavaScript Funktionen

- Funktionen in JavaScript können sowohl benannte als auch anonyme Funktionen sein
- Benannte Funktionen werden mit einem Namen definiert und können über den Namen aufgerufen werden, während anonyme Funktionen direkt einer Variable zugewiesen werden können und durch den Variablennamen aufgerufen werden

Universität Regensburg

JavaScript Funktionen: Anonyme Funktion

Beispiel für eine anonyme Funktion:

```
// Definition einer anonymen Funktion
let multiplyNumbers = function(a, b) {
  return a * b;
}

// Aufruf der Funktion und Speicherung des Rückgabewerts in einer Variablen
let product = multiplyNumbers(2, 3);

// Ausgabe des Ergebnisses
console.log(product); // Output: 6
```

Universität Regensburg

JavaScript Funktionen: Default Parameter

- Es ist möglich, Default-Werte für Parameter in einer JavaScript-Funktion zu setzen
- Wenn die Funktion aufgerufen wird und ein Parameterwert nicht bereitgestellt wird, wird der Default-Wert anstelle des fehlenden Wertes verwendet:

```
function greet(name = "World") {
  console.log("Hello " + name);
}

// Aufruf der Funktion ohne Argument
greet(); // Output: "Hello, World!"

// Aufruf der Funktion mit Argument
greet("John"); // Output: "Hello, John!"
```

Universität Regensburg

JavaScript Arrow Functions

```
let hello = function() {
  return "Hello World!";
}
```

Arrow functions ermöglichen es, eine (noch) kürzere Funktionssyntax zu schreiben:

```
let hello = () => {
  return "Hello World!";
}
```

Wenn die Funktion nur eine Anweisung hat und die Anweisung einen Wert zurückgibt, können Sie die Klammern und das Schlüsselwort `return` entfernen:

```
let hello = () => "Hello World!";
```

Universität Regensburg

JavaScript Variablen

- In einer Programmiersprache werden Variablen verwendet, um Datenwerte zu speichern
- JavaScript verwendet die Schlüsselwörter `var`, `let` und `const`, um Variablen zu deklarieren
- Ein "=" wird verwendet, um Variablen Werte zuzuweisen

```
let x;
x = 6;
```

Universität Regensburg

JavaScript Datentypen

- String

```
let x = "Volvo";
let y = 'VW';
```

- Number

```
let x = 16;
let y = 16.5;
```

- Boolean

```
let x = true;
let y = false;
```

- null / undefined:

  `null` bedeutet, dass eine Variable bewusst auf einen "leeren" Wert gesetzt wurde.

  `undefined` bedeutet dagegen, dass eine Variable noch nicht initialisiert wurde oder keinen Wert hat.

- Object
- Weitere Datentypen: Symbol, BigInt

Universität Regensburg

JavaScript Variablen: var

- In JavaScript steht `var` für "Variable" und es wird verwendet, um eine Variable zu deklarieren
- Existiert seit der ersten Version von JavaScript (1995)
- Wenn man eine Variable mit `var` deklariert, schaut JavaScript, dass Speicherplatz für einen bestimmten Wert reserviert wird

Universität Regensburg

JavaScript Variablen: var

- Die Sichtbarkeit von Variablen, die mit `var` deklariert werden, hängt davon ab, wo die Variable innerhalb des Codes deklariert wird
- Wenn die Variable innerhalb einer Funktion deklariert wird, ist sie nur innerhalb dieser Funktion sichtbar und kann von anderen Funktionen oder vom Hauptcode außerhalb der Funktion nicht verwendet werden
- Beispiel: Es wird eine Variable namens "zahl" innerhalb der Funktion "beispielFunktion" deklariert. Diese Variable ist nur innerhalb der Funktion sichtbar und kann von außerhalb der Funktion nicht zugegriffen werden:

```
function beispielFunktion() {
  var zahl = 10;
  console.log(zahl); // würde den Wert 10 ausgeben
}

beispielFunktion();
console.log(zahl); // würde einen ReferenceError ausgeben
```

Universität Regensburg

JavaScript Variablen: var

- Wenn die Variable außerhalb einer Funktion deklariert wird, ist sie im gesamten Code sichtbar, einschließlich innerhalb von Funktionen
- Beispiel:

```
var zahl = 10;
function beispielFunktion() {
   console.log(zahl); // würde den Wert 10 ausgeben
}

beispielFunktion();
console.log(zahl); // würde den Wert 10 ausgeben
```

Universität Regensburg

JavaScript Variablen: let vs. var

Gültigkeitsbereich:

- `var:` Gültigkeitsbereich, der sich auf die Funktion bezieht, in der sie deklariert wurde.
- Wenn die Variable außerhalb einer Funktion deklariert wird, hat sie einen globalen Gültigkeitsbereich und ist im gesamten Code sichtbar.
- `let:` Block-Gültigkeitsbereich, die Variable ist nur innerhalb des Blocks (geschweifte Klammern, { }), in dem sie deklariert wurde, sichtbar.
- Wenn die Variable außerhalb dieses Blocks verwendet wird, wird ein ReferenceError erzeugt.

Universität Regensburg

JavaScript Variablen: let vs. var

Gültigkeitsbereich:

- Beispiel `let`:

```
function myFunction() {
    let x = 10; // Variable x ist nur innerhalb dieser Funktion gültig
    if (x == 10) {
        let y = 20; // Variable y ist nur innerhalb dieser if-Anweisung (=Block) gültig
        console.log(x + y); // Ausgabe: 30
    }
    console.log(x); // Ausgabe: 10
    // ReferenceError: y ist nicht definiert (da y nur innerhalb des if-Blocks gültig ist)
    console.log(y);
}

myFunction();
```

Universität Regensburg

JavaScript Variablen: let vs. var

Gültigkeitsbereich:

- Beispiel `var`:

```
function myFunction() {
    var x = 10; // Variable x ist nur innerhalb dieser Funktion gültig
    if (x == 10) {
        // Variable y ist innerhalb dieser if-Anweisung (=Block) gültig
        // sowie innerhalb der Funktion (anders als bei let)
        var y = 20;
        console.log(x + y); // Ausgabe: 30
    }
    console.log(x); // Ausgabe: 10
    console.log(y); // Ausgabe: 20
}

myFunction();
```

Universität Regensburg

JavaScript Variablen: let vs. var

Art der Deklaration:

- Eine Variable, die mit `var` deklariert wird, kann auch ohne eine explizite Zuweisung deklariert werden und erhält den Wert `undefined`
- Eine Variable, die mit `let` deklariert wird, muss immer einen Wert zugewiesen bekommen, sonst wird ein Fehler generiert

```
var zahl1; // Die Variable wird ohne Zuweisung deklariert
console.log(zahl1); // Ausgabe: undefined

let zahl2; // Die Variable wird ohne Zuweisung deklariert, was jedoch einen Fehler generiert
console.log(zahl2); // Uncaught ReferenceError: zahl2 is not defined
```

JavaScript Variablen: let vs. var

Vermeidung von Fehlern:

- `let` hilft dabei, Fehler in deinem Code zu vermeiden, da es einen zwingt, die Variablen explizit zu initialisieren

Universität Regensburg

JavaScript Variablen: const (neu seit ECMAScript 6, 2015)

- `const` wird verwendet, um eine Konstante zu deklarieren, die sich während der Programmausführung nicht ändern kann
- Eine Variable, die mit `const` deklariert wurde, muss sofort mit einem Wert initialisiert werden
- Eine Variable, die mit `const` deklariert wurde, kann nicht später neu zugewiesen werden

JavaScript Variablen: const (neu seit ECMAScript 2015)

Beispiel 1:

```
const PI = 3.14159; // PI ist eine Konstante, die nicht überschrieben werden kann
console.log(PI); // Ausgabe: 3.14159

PI = 3.14; // Fehlermeldung: Uncaught TypeError: Assignment to constant variable.
console.log(PI); // Diese Zeile wird nicht ausgeführt, da es zu einem Fehler kommt
```

Beispiel 2:

```
// Fehlermeldung: Uncaught SyntaxError:
// Missing initializer in const declaration
const MY_CONSTANT;
```

Universität Regensburg

JavaScript Variablen: Benennung

- Variablennamen sollten aussagekräftig und beschreibend sein, um ihren Zweck und ihre Verwendung im Code klar zu machen
- Variablennamen sollten in camelCase geschrieben werden, das bedeutet, dass das erste Wort kleingeschrieben wird und jedes folgende Wort mit einem großen Buchstaben beginnt (z.B. myVariableName)
- Variablennamen dürfen keine Leerzeichen enthalten und müssen aus Buchstaben, Zahlen, Unterstrichen (_) oder Dollarzeichen ($) bestehen
- GROßSCHREIBUNG bei Konstanten.

Universität Regensburg

JavaScript Variablen: Operatoren

- In JavaScript können arithmetische Operatoren (`+ - * /`) zur Berechnung von Werten verwendet werden:

```
let x = (5 + 6) * 10;
```

Universität Regensburg

JavaScript Variablen: Operatoren

- Addition: **+**
- Subtraktion: **-**
- Multiplikation: ***
- Potenzierung: ****
- Division: **/**
- Modulus: **%**
- Inkrementieren: **++**
- Dekrementieren: **--**

Universität Regensburg

JavaScript Variablen: Assignment Operatoren

- Assignment Operatoren (+=) weisen JavaScript-Variablen Werte zu

```
let x = 10;
x = x + 5;
```



```
let x = 10;
x += 5;
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

JavaScript Variablen: Strings

- Ein String ist eine Sequenz von Zeichen, die in Anführungszeichen (entweder einfache oder doppelte) eingeschlossen sind
- Strings können Buchstaben, Zahlen, Leerzeichen und Sonderzeichen enthalten (Unicode Zeichenkodierung)

JavaScript Variablen: Strings

- JavaScript verfügt über eine Reihe von integrierten Methoden/Properties, mit denen Eigenschaften abgefragt werden können, wie u.a.:

  `.length`-Methode: Anzahl an Zeichen in String

  `.toUpperCase()` - Methode: String in Großbuchstaben konvertiert

  `.toLowerCase()` - Methode: String in Kleinbuchstaben konvertiert

Weitere Methoden: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Universität Regensburg

JavaScript Variablen: Strings

- `.length`-Methode: Anzahl an Zeichen in String

```
let message = "Hello, world!";
let length = message.length;

console.log(length);  // gibt 13 aus
```

Universität Regensburg

JavaScript Variablen: Strings

- Ein String kann entweder kann mit einfachen oder doppelten Anführungszeichen definiert werden:

```
let carName1 = "Volvo XC60";  // Double quotes
let carName2 = 'Volvo XC60';  // Single quotes
```

- Sie können Anführungszeichen innerhalb einer Zeichenkette verwenden, solange sie nicht mit den Anführungszeichen übereinstimmen, die die Zeichenkette umgeben:

```
let answer1 = "It's alright";
let answer2 = "He is called 'Max'";
let answer3 = 'He is called "Max"';
```

Universität Regensburg

JavaScript Variablen: Strings

- Da Zeichenketten in Anführungszeichen geschrieben werden müssen, wird JavaScript diese Zeichenkette falsch verstehen:

```
let text = "We are the so-called "Vikings" from the north.";
```

- Die Lösung, um dieses Problem zu vermeiden, ist die Verwendung des Backslash-Zeichens. Das Escape-Zeichen "Backslash" (\) verwandelt ein Sonderzeichen in ein String-Zeichen:

```
let text = "We are the so-called \"Vikings\" from the north.";
```

```
let text= 'It\'s alright.';
```

Universität Regensburg

JavaScript Variablen: Template Literals

- Template Literals sind eine syntaktische Erweiterung in JavaScript, die es ermöglicht, Strings mit eingebetteten Ausdrücken und Variablen zu erstellen
- Template-Literals werden in Backticks geschrieben (`…`) und können Ausdrücke innerhalb von ${ } - Zeichen enthalten
- Innerhalb der ${ }-Zeichen können Variablen, Funktionen oder beliebiger Code ausgeführt werden

Universität Regensburg

JavaScript Variablen: Template Literals

Beispiel:

```
const name = "Max Mustermann";
const age = 30;

// Verwenden von Template Literals
const message = `Mein Name ist ${name} und ich bin ${age} Jahre alt.`;

// Ausgabe: "Mein Name ist Max Mustermann und ich bin 30 Jahre alt."
console.log(message);
```

Universität Regensburg

JavaScript Variablen: Type Conversion

String zu Number: Die globale Methode `Number()` wandelt eine Variable (oder einen Wert) in eine Zahl um

```
Number("3.14") // returns 3.14
Number(" ") // returns 0
Number("") // returns 0
```

```
Number("99 88") // returns NaN (Datentyp Number)
Number("John") // returns NaN
```

Eine nicht numerische Zeichenfolge (wie "John") wird in NaN (Not a Number) umgewandelt

Universität Regensburg

JavaScript Variablen: Type Conversion

Number zu String: Die globale Methode `String()` wandelt eine Variable (oder einen Wert) in eine Zeichenfolge um:

```
String(x) // returns "123"
String(123) // returns "123"
String(100 + 23) // returns "123"
```

Boolean zu Number: Die globale Methode `Number()` kann eine Variable (oder einen Wert) auch in eine Variable des

Typs number umwandeln:

```
Number(false) // returns 0
Number(true) // returns 1
```

Universität Regensburg

JavaScript Objects

- JavaScript-Objekte bestehen aus Schlüssel-Wert-Paaren, wobei der Schlüssel eine Zeichenkette ist, die die Eigenschaft identifiziert, und der Wert den tatsächlichen Wert oder die Funktion enthält
- Objekte können jederzeit neue Eigenschaften hinzugefügt oder vorhandene Eigenschaften aktualisiert oder entfernt werden

Universität Regensburg

JavaScript Objects

Beispiel:

Schlüssel → firstName: "John", ← Wert

```
let person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

console.log(person.firstName) // "John"
console.log(person["lastName"]) // "Doe"
```

Universität Regensburg

JavaScript Objects: this Keyword

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

- `this` ist ein Schlüsselwort in JavaScript, das sich auf das Objekt bezieht, in dem es gerade verwendet wird
- Es ermöglicht den Zugriff auf die Eigenschaften und Methoden des aktuellen Objekts, in dem der Code ausgeführt wird

Universität Regensburg

JavaScript Objects: this Keyword

- Wenn `this` außerhalb eines Objekts aufgerufen wird, wird es normalerweise auf das globale Objekt verweisen: `window` im Browser-Umfeld (oder `global` in Node.js ist)
- Das `window` - Objekt stellt eine Schnittstelle für die Interaktion mit dem Browserfenster bereit
- Es enthält viele integrierte Eigenschaften und Methoden wie z.B. `document`, `console`, `setTimeout`, `setInterval` und vieles mehr

JavaScript Arrays

- Arrays in JavaScript sind geordnete Listen von Werten
- Sie können Werte verschiedener Datentypen enthalten
- Beispiel:

```
var cars = ["Saab", "Volvo", "BMW"];
```

Universität Regensburg

JavaScript Arrays

- Man greift auf ein Array-Element zu, indem man sich auf die Indexnummer mit `[index]` bezieht:

```
var cars = ["Saab", "Volvo", "BMW"];
let car = cars[0]; // Wert von cars[0] ist "Saab"
```

- Man kann ein Array-Element ändern, indem man sich auf die Indexnummer (beginnend bei 0) bezieht:

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Universität Regensburg

JavaScript Arrays: Properties

- JavaScript-Arrays bieten einige eingebaute Array-Eigenschaften und Methoden:

```
var numberOfCars = cars.length   // Gibt Anzahl an Elementen zurück
cars.sort()   // Array wird aufsteigend sortiert
```

- Man kann ein Array-Element auslesen, indem man sich auf die Indexnummer bezieht:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

// Erstes Element im Array
var firstFruit = fruits[0];

// Letztes Element im Array
var lastFruit = fruits[fruits.length - 1];
```

JavaScript Arrays: Methods

- Die Methode `pop()` entfernt das letzte Element aus einem Array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop(); // Entfernt "Mango"
```

- Die Methode `push()` fügt einem Array (am Ende) ein neues Element hinzu:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

Universität Regensburg

JavaScript Arrays: Elemente löschen

- Array-Elemente können mit dem JavaScript-Operator `delete` gelöscht werden
- Die Verwendung von `delete` hinterlässt als Wert im dem zu löschenden Index den Wert `undefined`

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0]; // fruits danach: [undefined, "Orange", "Apple", "Mango"]
```

- Stattdessen verwendet man lieber `splice()`:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1); // fruits danach: ["Orange", "Apple", "Mango"]
```

JavaScript Comparisons: == vs. ===

- In JavaScript konvertiert der == Operator Werte, bevor er sie vergleicht
- Wenn der Operator zwei Werte vergleicht, die unterschiedliche Datentypen haben, wird JavaScript versuchen, einen oder beide Werte in einen gemeinsamen Datentyp umzuwandeln, bevor er den Vergleich durchführt
- Es ist wichtig zu beachten, dass die Verwendung des == Operators in JavaScript oft zu unerwarteten Ergebnissen führen kann, da die Typumwandlung manchmal zu unvorhersehbaren Ergebnissen führen kann
- Es ist oft besser, den strikteren === Operator zu verwenden, der keine Typumwandlung durchführt und nur dann `true` zurückgibt, wenn die beiden Werte gleich sind **und** denselben Datentyp haben

JavaScript Comparisons

Unter der Voraussetzung, dass `x = 5` ist, werden in der folgenden Tabelle die Vergleichsoperatoren erläutert:

Operator	Description	Comparing	Returns
==	equal to	x == 8	
		x == 5	
		x == "5"	
===	equal value and equal type	x === 5	
		x === "5"	
!=	not equal	x != 8	
!==	not equal value or not equal type	x !== 5	
		x !== "5"	
		x !== 8	

Operator	Description	Comparing	Returns
>	greater than	x > 8	
<	less than	x < 8	
>=	greater than or equal to	x >= 8	
<=	less than or equal to	x <= 8	

Universität Regensburg

JavaScript Comparisons

Unter der Voraussetzung, dass `x = 5` ist, werden in der folgenden Tabelle die Vergleichsoperatoren erläutert:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	
		x === "5"	
!=	not equal	x != 8	
!==	not equal value or not equal type	x !== 5	
		x !== "5"	
		x !== 8	

Operator	Description	Comparing	Returns
>	greater than	x > 8	
<	less than	x < 8	
>=	greater than or equal to	x >= 8	
<=	less than or equal to	x <= 8	

JavaScript Comparisons

Unter der Voraussetzung, dass `x = 5` ist, werden in der folgenden Tabelle die Vergleichsoperatoren erläutert:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	
!==	not equal value or not equal type	x !== 5	
		x !== "5"	
		x !== 8	

Operator	Description	Comparing	Returns
>	greater than	x > 8	
<	less than	x < 8	
>=	greater than or equal to	x >= 8	
<=	less than or equal to	x <= 8	

Universität Regensburg

JavaScript Comparisons

Unter der Voraussetzung, dass `x = 5` ist, werden in der folgenden Tabelle die Vergleichsoperatoren erläutert:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	
		x !== "5"	
		x !== 8	

Operator	Description	Comparing	Returns
>	greater than	x > 8	
<	less than	x < 8	
>=	greater than or equal to	x >= 8	
<=	less than or equal to	x <= 8	

Universität Regensburg

JavaScript Comparisons

Unter der Voraussetzung, dass `x = 5` ist, werden in der folgenden Tabelle die Vergleichsoperatoren erläutert:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true

Operator	Description	Comparing	Returns
>	greater than	x > 8	
<	less than	x < 8	
>=	greater than or equal to	x >= 8	
<=	less than or equal to	x <= 8	

Universität Regensburg

JavaScript Comparisons

Unter der Voraussetzung, dass `x = 5` ist, werden in der folgenden Tabelle die Vergleichsoperatoren erläutert:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true

Operator	Description	Comparing	Returns
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Universität Regensburg

JavaScript if-Statement

- Man verwendet `if`, um einen Codeblock anzugeben, der ausgeführt wird, wenn eine bestimmte Bedingung (vom Datentyp boolean) erfüllt ist:

```
if (condition) {
  //  Codeblock, der ausgeführt wird, wenn die Bedingung erfüllt ist
}
```

- Man verwendet die `else` - Anweisung, um einen Codeblock anzugeben, der ausgeführt wird, wenn die Bedingung falsch ist:

```
if (condition) {
  //  Codeblock, der ausgeführt wird, wenn die Bedingung erfüllt ist
} else {
  //  Codeblock, der ausgeführt wird, wenn die Bedingung nicht erfüllt ist
}
```

Universität Regensburg

JavaScript if-Statement

- Beispiel:

```
var time = 5;
var greeting;

if (time < 10) {
   greeting = "Good morning";
} else if (time < 20) {
   greeting = "Good day";
} else {
   greeting = "Good evening";
}
```

Universität Regensburg

JavaScript: Ternäre Operator

- Der Ternäre Operator ist eine kurze Möglichkeit, um eine If-Else-Anweisung in JavaScript zu schreiben
- Der Operator besteht aus drei Teilen:

  (1) Bedingung

  (2) Wert, der zurückgegeben werden soll, wenn die Bedingung wahr ist

  (3) Wert, der zurückgegeben werden soll, wenn die Bedingung falsch ist.
- Beispiel:

```
const age = 18;
const message = age >= 18 ? "Sie sind volljährig" : "Sie sind minderjährig";
console.log(message); // Ausgabe: "Sie sind volljährig"
```

Universität Regensburg

JavaScript: Nullish Coalescence Operator

- Der Nullish Coalescence Operator ist ein Operator in JavaScript, der es einfacher macht, einen Standard- oder Ersatzwert zurückzugeben, wenn eine Variable **`null`** oder **`undefined`** ist.
- Beispiel:

```
let foo = null;
let bar = "Ersatzwert";
let result = foo ?? bar;
console.log(result); // Ausgabe: "Ersatzwert"
```

Universität Regensburg

JavaScript: for-Loop

- Schleifen (bzw. Loops) sind praktisch, wenn Sie denselben Code immer wieder ausführen wollen, z.B.

  jedes Mal mit einem anderen Wert

- Dies ist oft der Fall, wenn man mit Arrays arbeitet:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```



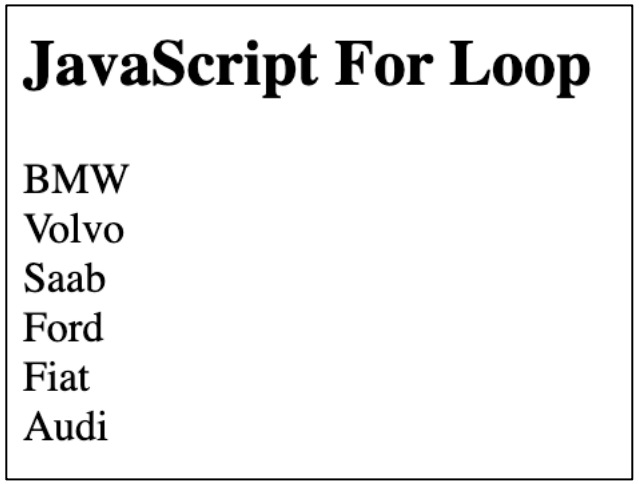
```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Universität Regensburg

JavaScript: for-Loop

- Beispiel:

```
<html>
  <body>
    <h2>JavaScript For Loop</h2>
    <p id="demo"></p>
    <script>
      const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
      let text = "";
      for (let i = 0; i < cars.length; i++) {
         text += cars[i] + "<br>";
      }
      document.getElementById("demo").innerHTML = text;
    </script>
  </body>
</html>
```



Universität Regensburg

JavaScript: for-Loop

```
for (Ausdruck 1; Ausdruck 2; Ausdruck 3) {
  // Codeblock, der ausgeführt werden soll
}
```

```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Ausdruck 1 wird (einmalig) vor der Ausführung des Codeblocks ausgeführt

Ausdruck 2 definiert die Bedingung für die Ausführung des Codeblocks

Ausdruck 3 wird (jedes Mal) ausgeführt, nachdem der Codeblock ausgeführt wurde

Universität Regensburg

JavaScript: for-of-Loop

```
const fruits = ["apple", "banana", "cherry"];

let text = "";
for (let x of fruits) {
  text += x + " ";
}

console.log(text) \\ Ausgabe: "apple banana cherry "
```

- for...of dient der Iteration über iterierbare Objekte wie Arrays oder Strings
- Jede Iteration gibt einen Wert zurück

Universität Regensburg

JavaScript: for-in-Loop

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x] + " ";
}

console.log(text) \\ Ausgabe: "John Doe 25"
```

- Die for-in-Schleife kann verwendet werden um über ein Objekt zu iterieren
- Jede Iteration gibt einen Schlüssel (x) zurück
- Der Schlüssel wird für den Zugriff auf den Wert des Schlüssels verwendet
- Der Wert des Schlüssels ist `person[x]`

Universität Regensburg

Objekt-Iterationen mit `Object.keys()`

- Methode zur Rückgabe der Schlüssel eines Objekts
- Gibt ein Array aller Schlüssel zurück, die dem Objekt zugeordnet sind

```
const obj = { name: "John", age: 30, city: "New York" };
for (let key of Object.keys(obj)) {
    console.log(key);
}

// Output:
// name
// age
// city
```

Universität Regensburg

Objekt-Iterationen mit `Object.values()`

- Methode zur Rückgabe der Werte eines Objekts
- Gibt ein Array aller Werte zurück, die dem Objekt zugeordnet sind

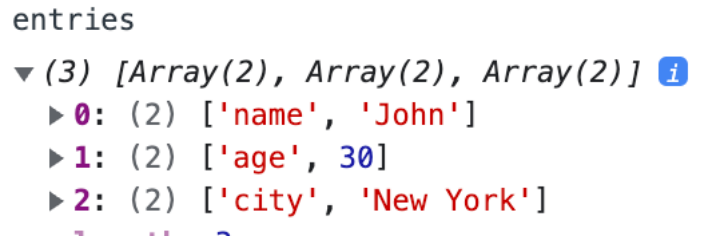
```
const obj = { name: "John", age: 30, city: "New York" };
for (let value of Object.values(obj)) {
    console.log(value);
}

// Output:
// John
// 30
// New York
```

Universität Regensburg

Objekt-Iterationen mit `Object.entries()`

- Methode zur Rückgabe von Schlüssel-Wert-Paaren eines Objekts
- Gibt ein Array von Arrays zurück, wobei jedes innere Array ein Schlüssel-Wert-Paar darstellt

```
const obj = { name: "John", age: 30, city: "New York" };
const entries = Object.entries(obj);

for (let i = 0; i < entries.length; i++) {
   const key = entries[i][0];
   const value = entries[i][1];
   console.log(key + ": " + value);
}

// Output:
// name: John
// age: 30
// city: New York
```



Universität Regensburg

JavaScript: Array-Transformation mit `map()`

- Die Methode `.map()` kann auf Arrays in JavaScript angewendet werden, um ein neues Array zu erstellen, das auf einer Transformation jedes Elements des ursprünglichen Arrays basiert
- Das ursprüngliche Array wird dabei nicht verändert
- Ein Beispiel für eine Array-Transformation mit `.map()` wäre die Verwendung einer Methode, um alle Elemente in einem Array zu verdoppeln. Dazu könnte der folgende Code verwendet werden:

```
const originalArray = [1, 2, 3, 4];
const doubledArray = originalArray.map((element) => {
   return element * 2
});
console.log(doubledArray); // Ausgabe: [2, 4, 6, 8]
```

Universität Regensburg

JavaScript: Array-Filterung mit `filter()`

- `.filter()` ist eine weitere Methode, die verwendet werden kann, um ein neues Array zu erstellen, das nur aus den Elementen des ursprünglichen Arrays besteht, die bestimmte Kriterien erfüllen.
- Das ursprüngliche Array wird dabei nicht verändert
- Ein Beispiel für die Verwendung von `.filter()` wäre die Erstellung eines neuen Arrays, das nur aus den geraden Zahlen eines ursprünglichen Arrays besteht. Dazu könnte der folgende Code verwendet werden:

```
const originalArray = [1, 2, 3, 4, 5, 6, 7, 8, 9];
const evenArray = originalArray.filter((element) => {
  return element % 2 === 0
});
console.log(evenArray); // Ausgabe: [2, 4, 6, 8]
```

JavaScript: Array-Aggregation mit `reduce()`

- Die Methode `.reduce()` kann auf Arrays in JavaScript angewendet werden, um eine einzige aggregierte Ausgabe basierend auf der Kombination aller Elemente des ursprünglichen Arrays zu erstellen.
- Anders als `.map()` und `.filter()` verwendet `.reduce()` nicht nur die Elemente des ursprünglichen Arrays, sondern auch eine sogenannte **Akkumulatorvariable**, die **bei jedem Schleifendurchlauf aktualisiert wird** und die aggregierte Ausgabe enthält.

JavaScript: Array-Aggregation mit `reduce()`

- Ein Beispiel für die Verwendung von `.reduce()` in JavaScript wäre die Erstellung einer einzigen Summe aller Zahlen in einem ursprünglichen Array. Dazu könnte der folgende Code verwendet werden:

```
const originalArray = [1, 2, 3, 4, 5];
const sum = originalArray.reduce((accumulator, currentValue) => {
   return accumulator + currentValue
}, 0); // Initialer Wert ist 0
console.log(sum); // Ausgabe: 15
```

- Das erste Argument von `.reduce()` ist eine Funktion, die den Akkumulator und das aktuelle Element des Arrays als Argumente erhält. Die Funktion gibt das aktualisierte Ergebnis zurück, das als Akkumulator für den nächsten Durchlauf verwendet wird.

Universität Regensburg

JavaScript: Array-Aggregation mit `reduce()`

- Das zweite Argument von `.reduce()` ist der anfängliche Wert des Akkumulators. In unserem Beispiel setzen wir ihn auf 0.
- Es ist wichtig zu beachten, dass `.reduce()` ebenfalls das ursprüngliche Array nicht verändert, sondern stattdessen eine einzige aggregierte Ausgabe zurückgibt.

Universität Regensburg

JavaScript: while-Loop

```
let i = 0;

while (i < 10) {
  console.log("Hello World")
  i++;
}
```

- Ein while-Loop ist eine Schleifenstruktur in JavaScript, die eine Bedingung überprüft und solange eine bestimmte Anweisung ausführt, bis die Bedingung falsch wird
- Im gegebenen Beispiel wird die while-Loop ausgeführt, solange der Wert von i kleiner als 10 ist, und in jeder Iteration wird der Wert von i um eins erhöht und an den Text angehängt, bis i den Wert 10 erreicht hat

Universität Regensburg

JavaScript: `break` – Statement bei Loops

```
for (let i = 0; i < 10; i++) {
  if (i === 3) {
     break;
  }
  text += "The number is " + i + "<br>";
}
```

JavaScript Loops

A loop with a **break** statement.

The number is 0
The number is 1
The number is 2

`break` **– Statement:**

Im obigen Beispiel beendet die `break` - Anweisung die Schleife ("unterbricht" die Schleife),

wenn der Schleifenzähler (i) 3 ist

Universität Regensburg

JavaScript: try and catch

- `try` und `catch` ist ein Konstrukt in JavaScript, das verwendet wird, um Fehler oder Ausnahmen abzufangen, die während der Ausführung von Code auftreten können
- Der Code innerhalb des `try` - Blocks wird ausgeführt und falls dabei ein Fehler auftritt, wird der Code innerhalb des `catch` - Blocks ausgeführt, der den Fehler abfängt und eine benutzerdefinierte Aktion ausführt:

```
try {
  // Block of code to try
}
catch(err) {
  // Block of code to handle errors
}
```

Universität Regensburg

JavaScript DOM-Manipulation

- DOM: Das Document Object Model (DOM) ist eine Schnittstelle, die das gesamte HTML-Dokument als Baumstruktur repräsentiert. Mit JavaScript kann man auf das DOM zugreifen und es manipulieren, um Änderungen an der Webseite vorzunehmen.
- Das `document` - Objekt in JavaScript stellt das HTML-Dokument dar und bietet Methoden, um auf das Dokument zuzugreifen und es zu ändern

Universität Regensburg

JavaScript DOM-Manipulation: Elemente auswählen

Um auf ein HTML-Element zuzugreifen, um dieses zu manipulieren, gibt es mehrere Möglichkeiten:

Methode	**Zweck**
`document.getElementById(`*`id`*`)`	Element über id auswählen
`document.querySelector(`*`name`*`)`	name = `"p"` -> wählt Element mit tag `"p"` aus name = `".list"` -> wählt **erstes** Element mit Klasse `"list"` aus name = `"#list"` -> wählt Element mit Id `"list"` aus

Universität Regensburg

JavaScript DOM-Manipulation: Mehrere Elemente auswählen

Um auf mehrere HTML-Elemente zuzugreifen, um diese zu manipulieren, gibt es zwei Möglichkeiten:

- **HTMLCollection**: Wird automatisch aktualisiert, wenn Elemente im DOM hinzugefügt oder entfernt werden, die der ursprünglichen Auswahlkriterien entsprechen

Methode	**Zweck**
`document.getElementsByClassName(`*name*`)`	Elemente über Klassennamen auswählen
`document.getElementsByTagName(`*name*`)`	Elemente über tag-Namen auswählen

Universität Regensburg

JavaScript DOM-Manipulation: Mehrere Elemente auswählen

Um auf mehrere HTML-Elemente zuzugreifen, um diese zu manipulieren, gibt es zwei Möglichkeiten:

- **NodeList**: Im Gegensatz zur HTMLCollection wird eine NodeList nicht automatisch aktualisiert, wenn Elemente im DOM hinzugefügt oder entfernt werden, die der ursprünglichen Auswahlkriterien entsprechen.

Methode	**Zweck**
`document.querySelectorAll(`*name*`)`	name = `"p"` -> wählt **alle** Elemente mit tag `"p"` aus name = `".list"` -> wählt **alle** Elemente mit Klasse `"list"` aus

JavaScript DOM-Manipulation: Elemente manipulieren

Es gibt einige Möglichkeiten, Eigenschaften von HTML-Elementen zu ändern:

Methode	**Zweck**
`element.innerHTML = new html content`	Den Content eines HTML-Elements ändern
`element.setAttribute(Name der Eigenschaft, Wert der Eigenschaft)`	Ein Attribut eines HTML-Elements ändern
`element.style.property = new style`	Stil eines HTML-Elements ändern

JavaScript DOM-Manipulation: Elemente hinzufügen/löschen/ersetzen

Methode	Zweck
`childElement = document.createElement(`*`tagName`*`)`	Die Methode `createElement()` erstellt ein neues Element. Ist der tagName beispielsweise "h1", so hat das Element einen h1-Tag ( ).
`element.appendChild(childElement)`	Child-Element zu Parent-Element hinzufügen
`element.removeChild(child`*`Element`*`)`	Die Methode `removeChild()` entfernt ein Child-Element aus einem Parent-Element und gibt das entfernte Element zurück
`document.replaceChild(`*`newChildElement`*`,` `  `*`oldChildElement`*`)`	HTML-Elemente können ersetzt werden

JavaScript Classes

- Klassen in JavaScript sind wie Blaupausen oder Vorlagen, die uns helfen, ähnliche Objekte mit **gemeinsamen Eigenschaften und Methoden** zu erstellen
- In JavaScript werden Klassen mithilfe von Prototypen definiert, wobei der Prototyp als Blaupause für alle Instanzen der Klasse dient

Universität Regensburg

JavaScript Classes

- Um ein Objekt zu erstellen, müssen wir eine Instanz der Klasse erstellen
- Eine Klasse kann Vererbung unterstützen, was bedeutet, dass sie Eigenschaften und Methoden von einer anderen Klasse erben kann.

Universität Regensburg

JavaScript Classes

- Man verwendet das Schlüsselwort `class`, um eine Klasse zu erstellen.

  Beispiel:

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}

let myCar1 = new Car("Ford", 2014); // Car Object: {name: "Ford", year: 2014}
let myCar2 = new Car("Audi", 2019); // Car Object: {name: "Audi", year: 2019}
```

  Im obigen Beispiel wird die Klasse Car verwendet, um zwei Car-Objekte zu erstellen.

Universität Regensburg

JavaScript Classes: Konstruktor

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

- Die Konstruktormethode wird automatisch aufgerufen, wenn ein neues Objekt erstellt wird
- Es wird verwendet, um Objekteigenschaften zu initialisieren
- Wenn Sie keine Konstruktormethode definieren, fügt JavaScript eine leere Konstruktormethode hinzu

Universität Regensburg

JavaScript Classes: Methoden

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  getCarInformation() {
    return "A " + this.name + " from " + this.year;
  }
}

let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML = myCar.getCarInformation();
```

Universität Regensburg

JavaScript Classes: Vererbung

- Klassen in JavaScript können von anderen Klassen erben.
- Die erbende Klasse erhält alle Eigenschaften und Methoden der vererbenden Klasse (auch Basisklasse genannt)
- Die erbende Klasse kann die Methoden und Eigenschaften der Basisklasse überschreiben oder neue hinzufügen.

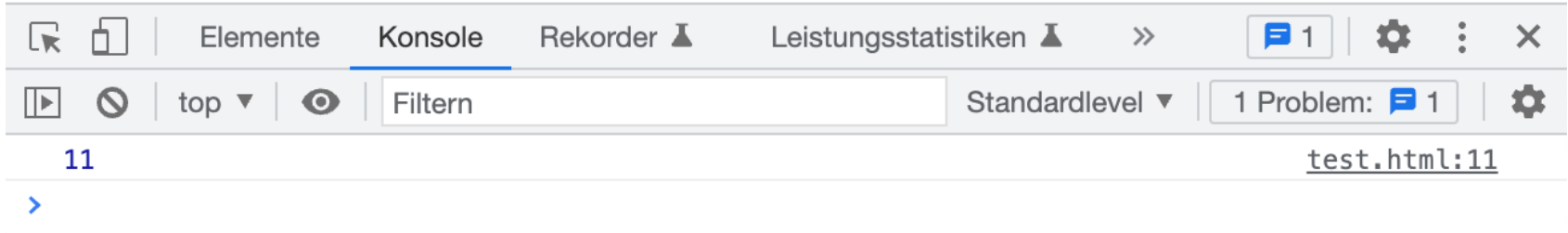
Universität Regensburg

JavaScript Classes: Vererbung

```
class Tier {
  constructor(name) {
    this.name = name;
  }

  sprechen() {
    console.log("Ein Tier spricht.");
  }
}

class Hund extends Tier {
  sprechen() {
    console.log("Der Hund bellt.");
  }
}

const fido = new Hund("Fido");
console.log(fido.name); // Output: "Fido"
fido.sprechen(); // Output: "Der Hund bellt."
```

JavaScript Debugging: `console.log()`

```
<!DOCTYPE html>
<html>
  <body>
  <script>
     console.log(5 + 6);
  </script>
  </body>
</html>
```



Universität Regensburg

JavaScript Kommentare

- Code nach doppelten Schrägstrichen `//` oder zwischen `/*` und `*/` wird wie ein Kommentar behandelt

```
let x = 5;    // wird ausgeführt
// x = 6;    wird nicht ausgeführt

/*
 Dies ist ein mehrzeiliger Kommentar
*/
```

Universität Regensburg

Weiterführende Literatur / Quellen

- Literatur: Eloquent JavaScript:
  https://eloquentjavascript.net/
- Ausführliches Tutorial zu JavaScript mit anschaulichen Beispielen:
  https://www.w3schools.com/js/
- Daten im Browser speichern:
  https://www.w3schools.com/html/html5_webstorage.asp
- JavaScript
  javascript.info