WBE: JAVASCRIPT

OBJEKTE UND ARRAYS

ÜBERSICHT

- Objekte
- Spezielle Objekte: Arrays
- Werte- und Referenztypen
- Vordefinierte Objekte
- JSON

ÜBERSICHT

- Objekte
- Spezielle Objekte: Arrays
- Werte- und Referenztypen
- Vordefinierte Objekte
- JSON

OBJEKTE UND ARRAYS

- Objekte: Werte zu Einheiten zusammenfassen
- Arrays: Objekte mit speziellen Eigenschaften

Was	Objekt	Array
Art	Attribut-Wert-Paare	Sequenz von Werten
Literalnotation	werte = { a: 1, b: 2 }	liste = [1, 2, 3]
Ohne Inhalt	werte = { }	liste = [ ]
Elementzugriff	werte["a"] oder werte.a	liste[0]

OBJEKTLITERALE

```
let person = {
  name: "John Baker",
  age: 23,
  "exam results": [5.5, 5.0, 5.0, 6.0, 4.5]
}
```

- Sammlung von Attributen und Werten
- Attributname und Wert durch Doppelpunkt getrennt
- Attribut-Wert-Paare durch Kommas getrennt
- Attributname als String, wenn es kein gültiger Name ist

ZUGRIFF AUF ATTRIBUTE

```
let person = {
  name: "John Baker",
  age: 23,
  "exam results": [5.5, 5.0, 5.0, 6.0, 4.5]
}

console.log(person.name)      /* → John Baker  */
console.log(person["age"])    /* → 23          */
console.log(person["exam"])   /* → undefined   */
```

- Punkt- oder Klammernotation zum Zugriff
- Punktnotation: Attribut muss gültiger Name sein
- Zugriff auf nicht vorhandenes Attribut liefert `undefined`

OPTIONAL CHAINING

```
const adventurer = {
  name: 'Alice',
  cat: { name: 'Dinah' }
}

console.log(adventurer.dog.name)                    /* → TypeError */
console.log(adventurer.dog && adventurer.dog.name)  /* → undefined */
console.log(adventurer.dog?.name)                   /* → undefined */
```

- Eingeführt mit ECMAScript 2020
- Verschiedene weitere Möglichkeiten
  (s. Optional Chaining)

ATTRIBUTE HINZUFÜGEN

- Objekte sind dynamische Datenstrukturen
- Sie können jederzeit erweitert werden

```
> let obj = { message: "not yet implemented" }
> obj.ready = false

> obj
{ message: 'not yet implemented', ready: false }

> obj.attr
undefined
```

ATTRIBUTE ENTFERNEN

- Objekte können jederzeit verkleinert werden
- Mit `delete` wird ein Attribut entfernt
- Mit `in` kann überprüft werden, ob ein Attribut existiert

```
> let obj = { message: "ready", ready: true, tasks: 3 }
> delete obj.message
> obj.tasks = undefined

> obj
{ ready: true, tasks: undefined }

> "message" in obj
false
> "tasks" in obj
true
```


METHODEN

- Attribute, deren Werte Funktionen sind, werden Methoden genannt
- Sie werden über das Objekt aufgerufen

```
> let cat = { type: "cat", sayHello: () => "Meow" }

> cat.sayHello
[Function: sayHello]

> cat.sayHello()
'Meow'
```


METHODEN

```
let player = {
  sayHello: function () { return  "Hello" },  /* ausführlich    */
  sayHi() { return  "Hi" },                   /* abgekürzt      */
  sayBye: () => "Bye",                        /* Pfeilnotation  */
}
```

- Verschiedene Notationen für Methoden
- Abgekürzte Variante seit ECMAScript 2015 möglich

METHODEN

```
let cat = {
  type: "cat",
  say1() { return  "Meow from " + this.type },
  say2: () => "Meow from " + this.type,
}
console.log( cat.say1() )    /* → Meow from cat         */
console.log( cat.say2() )    /* → Meow from undefined   */
```

- `this` ist das Objekt, über das die Methode aufgerufen wird
- Das gilt nicht für Funktionen in Pfeilnotation
- Mehr dazu in einer anderen Lektion

OBJEKT ANALYSIEREN

- Methode `keys` von `Object`
- Liefert Array aller Attributnamen
- Analog liefert `values` alle Werte

```
> let obj = {a: 1, b: 2}

> Object.keys(obj)
[ 'a', 'b' ]

> Object.values(obj)
[ 1, 2 ]
```


OBJEKTE ZUSAMMENFÜHREN

- Methode `assign` von `Object`
- Erstes Argument ist das Zielobjekt
- Attribute der weiteren Argumente ins Zielobjekt kopiert
- Referenz auf Ergebnis (erstes Arg.) zurückgegeben

```
> let objectA = {a: 1, b: 2}

> Object.assign(objectA, {b: 3, c: 4})
{ a: 1, b: 3, c: 4 }

> Object.assign(objectA, {m: 10}, {n: 11})
{ a: 1, b: 3, c: 4, m: 10, n: 11 }
```


SPREAD-SYNTAX

```
> let objectA = { a: 1, b: 2 }
> let objectB = { c: 100, d: 200 }

> {...objectA, ...objectB, c: 3}
{ a: 1, b: 2, c: 3, d: 200 }

> {...objectA}
{ a: 1, b: 2 }

> {...objectA} == objectA
false
```

- Inhalte eines Objekts in ein anderes Objekt einfügen
- Spread-Operator `...`

OBJEKTE DESTRUKTURIEREN

```
1 let bar = 87
2 let obj = { foo: 12, bar, baz: 43 }
3
4 let {foo, baz} = obj
5 console.log(foo)                    /* → 12 */
```

- Teile aus (möglicherweise grossen) Objekten extrahieren
- Auch in Funktionsparametern möglich (spätere Lektion)

ÜBERSICHT

- Objekte
- Spezielle Objekte: Arrays
- Werte- und Referenztypen
- Vordefinierte Objekte
- JSON

ARRAYS

- Sequenzen von Werten
- Zugriff über Index (erstes Element hat Index 0)
- Nicht jede Position muss besetzt sein
- Nicht besetzte Positionen liefern `undefined`

```
let a = [1, 2, 3]
a[10] = 99

console.log( a )         /* → [ 1, 2, 3, <7 empty items>, 99 ]  */
console.log( a.length )  /* → 11                                */
console.log( a[1000] )   /* → undefined                         */
```


ARRAYS

- Array-Elemente können von beliebigem Typ sein
- Typen können problemlos gemischt werden
- Hier ist das letzte Element des Arrays eine Funktion:

```
> let data = [41, 3.14, "pi", [1, 2, 3], n => 2*n]
undefined

> data[4](3)
6
```


ARRAYS

- Arrays sind Objekte mit speziellen Eigenschaften
- Sie haben Attribute und Methoden
- Test auf Array: `Array.isArray()`

```
> let data = [1, 2, 3]

> typeof(data)
'object'
> Array.isArray(data)
true
> data.length
3
```

ARRAY-METHODEN

- Für Arrays stehen zahlreiche Methoden zur Verfügung
- Zum Beispiel `push` und `pop`

```
> let data = [1, 2, 3]

> data.push(10)
4
> data.push(11, 12)
6
> data.pop()
12

> data
[ 1, 2, 3, 10, 11 ]
```


ARRAY-METHODEN

- `shift`, `unshift`: Einfügen und Entfernen am Array-Anfang
- `indexOf`, `lastIndexOf`: Element im Array finden
- `slice`: Bereich eines Arrays ausschneiden
- `concat`: Arrays zusammenhängen
- `at`: Zugriff auf Index (ECMAScript 2022)

```
> let data = [ 1, 2, 3, 10, 11 ]

> data.slice(1, 3)
[ 2, 3 ]

> data.concat([100, 101])
[ 1, 2, 3, 10, 11, 100, 101 ]
```


SCHLEIFEN ÜBER ARRAYS

```
1 /* Standard for-Schleife */
2 for (let i = 0; i < myArray.length; i++) {
3   doSomethingWith(myArray[i])
4 }
5
6 /* einfachere Variante für Arrays */
7 for (let entry of myArray) {
8   doSomethingWith(entry)
9 }
```


SPREAD-SYNTAX

```
> let parts = ['shoulders', 'knees']
> ['head', ...parts, 'and', 'toes']
["head", "shoulders", "knees", "and", "toes"]

> [...parts]
['shoulders', 'knees']

> [...parts] == parts
false
```

- Inhalte eines Arrays in ein anderes Array einfügen
- Spread-Operator `...`

ARRAYS DESTRUKTURIEREN

- Mehrere Parameter oder Variablen aus einem Array zuweisen
- Vermeidet das spätere Zugreifen über den Array-Index

```
let numbers = [1, 2, 3]
let [a, b, c] = numbers
console.log(c)                          /* → 3     */
```


ÜBERSICHT

- Objekte
- Spezielle Objekte: Arrays
- Werte- und Referenztypen
- Vordefinierte Objekte
- JSON

WERTE-DATENTYPEN

- Zahlen, Strings und Wahrheitswerte sind *Wertetypen*
- Sie sind unveränderlich
- Zuweisung kann wie Kopieren behandelt werden

```
let msg = "Hello developers!"
let greeting = msg
greeting += "!!"

console.log(greeting)     /* → 'Hello developers!!!' */
console.log(msg)          /* → 'Hello developers!'   */
```


REFERENZ-DATENTYPEN

- Objekte und Arrays sind *Referenz-Datentypen*
- Sie sind jederzeit veränderbar
- Es werden Referenzen zugewiesen

```
let obj = { message: "loading..." }
let anotherObj = obj
anotherObj.message = "ready"

console.log(obj)          /* → { message: 'ready' } */
```

WERTE- UND REFERENZTYPEN

- Objekt- und Array-Literale legen neue Objekte an
- `==` und `===` vergleichen die Referenzen
- `const` heisst: Referenz kann nicht geändert werden

```
> const a = [1, 2, 3]
> const b = [1, 2, 3]
> const c = a

> a == b
false
> a == c
true
> c[0] = 99
> a
[ 99, 2, 3 ]
```

```
> const obj = { message: "loading..." }

> obj.message = "ready"
'ready'

> obj = {}
Uncaught TypeError: Assignment to
constant variable.
```


WERTE- UND REFERENZTYPEN

- Objekte sind also Referenztypen
- Das gilt auch für Arrays und Funktionen
- Referenzen vs. Werte vergleichen:

```
> [ []==[], {}=={}, (()=>{})==(()=>{}) ]
[ false, false, false ]

> [ 3.5==3.5, "abc"=="abc", false==false ]
[ true, true, true ]
```


ATTRIBUTE

- Wie Objekte und Arrays können auch Werte in JavaScript Attribute (bzw. Methoden) haben
- Ausnahmen: `null`, `undefined`

```
> "Zeichenkette".length
12
> "Zeichenkette"["length"]
12
> "Zeichenkette".toUpperCase()
'ZEICHENKETTE'
```

```
> 1.5.toString()
'1.5'
> (1/3).toPrecision(4)
'0.3333'
```


ATTRIBUTE

- Attribute von Wertetypen sind unveränderlich
- Zuweisung neuer Attribute zu Wertetypen wird ignoriert
- Objekten (auch Arrays, Funktionen) können aber jederzeit Attribute zugewiesen werden

```
> const square = n => n*n

> square.doc = "Quadratfunktion"
'Quadratfunktion'

> square(3)
9

> square.doc
'Quadratfunktion'
```

ARRAYS UND ATTRIBUTE

- Die (normalen) Attribute eines Arrays sind ganze Zahlen >=0
- Wird etwas anderes als Index angegeben, wird ein Attribut hinzugefügt

```
> let arr = [1, 2, 3]

> arr[-1] = 4
> arr['key'] = 'value'

> arr
[ 1, 2, 3 ]
> arr.key
'value'
```


ÜBERSICHT

- Objekte
- Spezielle Objekte: Arrays
- Werte- und Referenztypen
- Vordefinierte Objekte
- JSON

String

- Strings sind in JavaScript ein primitiver Datentyp
- Erzeugt durch String-Literale `"..."`, `'...'`, `` `...` ``
- String-Methoden sind in `String.prototype` definiert (mehr zu Prototypen später)

```
> String.prototype.slice
[Function: slice]

> "Hello World".slice(0, 5)
'Hello'
```


STRING-METHODEN

- `slice`: Ausschnitt aus einem String (vgl. Arrays)
- `indexOf`: Position eines Substrings (vgl. Arrays)
- `trim`: Whitespace am Anfang und Ende entfernen
- `padStart`: vorne Auffüllen bis zu bestimmter Länge
- `split`: Auftrennen, liefert Array von Strings
- `join`: Array von Strings zusammenfügen
- `repeat`: String mehrfach wiederholen

MDN: String

Number

- Methoden und Konstanten von `Number`
- Methoden von `Number.prototype` können auf einzelne Zahlen angewendet werden

```
> Number.MAX_VALUE
1.7976931348623157e+308

> Number.isInteger(1.5)
false

> 3500.75.toLocaleString('de-DE')
'3.500,75'
```

MDN: Number

Math

- Methoden und Konstanten als Attribute des `Math`-Objekts
- Objekt als Namensraum für Methoden und Konstanten
- Gleich wie `String` und `Number` ist `Math` built-in

```
> Math.E
2.718281828459045

> Math.exp(1)
2.718281828459045

> Math.min(5, 2, 7, -4, 12)
-4
```

```
> Math.sin(0.5)
0.479425538604203

> Math.round(3.7)
4

> Math.random()
0.04802545627381716
```

MDN: Math

Date

```
let now = new Date()

console.log(now)                    /* → 2021-10-09T15:43:21.753Z */
console.log(now.getHours())         /* → 17                       */
console.log(now.getUTCHours())      /* → 15                       */
console.log(now.getTime())          /* → 1633794201753            */

now.setFullYear(now.getFullYear()+1)
console.log(now.toString())
// 'Sun Oct 09 2022 17:43:21 GMT+0200 (Mitteleuropäische Sommerzeit)'
```

MDN: Date

WEITERE VORDEFINIERTE OBJEKTE

- `Map`, `Set`: Schlüssel/Wert Zuordnung bzw. Menge
- `RegExp`: reguläre Ausdrücke
- `Object`: allgemein Objekte

Zahlreiche weitere vordefinierte Objekte...
MDN: Standard built-in objects

ÜBERSICHT

- Objekte
- Spezielle Objekte: Arrays
- Werte- und Referenztypen
- Vordefinierte Objekte
- JSON

JSON

- JavaScript Object Notation
- Daten-Austauschformat, nicht nur für JavaScript
- Orientiert an Notation für JavaScript-Objektliterale

```
> JSON.stringify({ type: "cat", name: "Mimi", age: 3})
'{"type":"cat","name":"Mimi","age":3}'

> JSON.parse('{"type":"cat","name":"Mimi","age":3}')
{ type: 'cat', name: 'Mimi', age: 3 }
```

https://www.json.org/json-en.html

QUELLEN

- Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
  https://eloquentjavascript.net/

LESESTOFF

Geeignet zur Ergänzung und Vertiefung

- Kapitel 4 von:
  Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
  https://eloquentjavascript.net/