Objekte für strukturierte Daten

Was sind Objekte?

Ein **Objekt** fasst mehrere Informationen zusammen, die zu einer Sache gehören. Jede Information erhält dabei einen eindeutigen Namen.

Stell dir ein Benutzerprofil vor: Es hat einen Namen, ein Alter und vielleicht eine E-Mail-Adresse. Statt für jede Information eine eigene Variable anzulegen, kannst du sie gemeinsam in einem Objekt speichern.

```
const person = {
  name: "Mira",
  alter: 24,
  email: "mira@example.de"
};
```

Das Objekt `person` beschreibt eine Person mit drei Eigenschaften.

- `name`, `alter` und `email` heißen **Eigenschaften** oder *Properties*.
- Die Werte rechts vom Doppelpunkt gehören jeweils zu einer Eigenschaft.
- Die Eigenschaften werden durch Kommata getrennt.
- Das gesamte Objekt steht zwischen geschweiften Klammern `{}`.

Eigenschaften auslesen

Um auf eine Eigenschaft zuzugreifen, verwendest du meist die **Punktnotation**:

```
const person = {
  name: "Mira",
  alter: 24,
  email: "mira@example.de"
```

```
};

console.log(person.name);
console.log(person.alter);
```

Die Konsole gibt aus:

```
Mira
24
```

Du kannst Eigenschaften auch direkt in Texten verwenden:

```
console.log("Hallo, " + person.name + "!");
```

Ausgabe:

```
Hallo, Mira!
```

Mit einem Template-String wirkt derselbe Code oft übersichtlicher:

```
console.log(`Hallo, ${person.name}! Du bist ${person.alter} Jahre alt.`);
```

Eigenschaften verändern und ergänzen

Die Werte eines Objekts können verändert werden:

```
const spieler = {
  name: "Noah",
  punkte: 10
};

spieler.punkte = 15;

console.log(spieler.punkte);
```

Ausgabe:

```
15
```

Auch neue Eigenschaften lassen sich nachträglich hinzufügen:

```
spieler.level = 2;

console.log(spieler);
```

Das Objekt enthält nun `name`, `punkte` und `level`.

> **Wichtig:** Ein Objekt kann mit `const` erstellt werden, obwohl seine Eigenschaften später noch geändert werden dürfen. `const` verhindert nur, dass die Variable selbst auf ein vollständig anderes Objekt zeigt.

Das ist erlaubt:

```
const buch = {
  titel: "JavaScript leicht gemacht",
  seiten: 180
};

buch.seiten = 200;
```

Das wäre dagegen ein Fehler:

```
const buch = {
  titel: "JavaScript leicht gemacht"
};

buch = {
  titel: "Ein anderes Buch"
};
```

Eigenschaften mit eckigen Klammern auswählen

Neben der Punktnotation gibt es die **Klammernotation**:

```
const wetter = {
  stadt: "Hamburg",
  temperatur: 18
};

console.log(wetter["stadt"]);
```

Das Ergebnis ist dasselbe wie bei `wetter.stadt`.

Die Klammernotation ist besonders nützlich, wenn der Name einer Eigenschaft in einer Variablen gespeichert ist:

```
const profil = {
  name: "Lea",
  hobby: "Zeichnen"
};

const eigenschaft = "hobby";

console.log(profil[eigenschaft]);
```

Ausgabe:

```
Zeichnen
```

Mit Punktnotation würde JavaScript nach einer Eigenschaft namens `eigenschaft` suchen:

```
console.log(profil.eigenschaft);
```

Da diese Eigenschaft nicht existiert, erhältst du `undefined`.

Fehlende Eigenschaften

Wenn du auf eine Eigenschaft zugreifst, die nicht vorhanden ist, entsteht normalerweise kein Fehler. JavaScript gibt stattdessen `undefined` zurück.

```
const tier = {
  art: "Katze",
```

```
  name: "Luna"
};

console.log(tier.alter);
```

Ausgabe:

```
undefined
```

Du kannst das in einer Bedingung prüfen:

```
if (tier.alter === undefined) {
  console.log("Das Alter ist nicht bekannt.");
}
```

Objekte und Arrays vergleichen

Ein **Array** speichert mehrere Werte in einer Reihenfolge. Du greifst über einen Zahlenindex darauf zu.

```
const farben = ["Rot", "Grün", "Blau"];

console.log(farben[0]);
```

Ein **Objekt** speichert Werte unter aussagekräftigen Namen.

```
const auto = {
  marke: "Volkswagen",
  modell: "ID.3",
  baujahr: 2023
};

console.log(auto.marke);
```

Verwende Arrays, wenn die **Reihenfolge** wichtig ist oder wenn du eine Liste gleichartiger Werte hast:

```
const lieblingsObst = ["Apfel", "Banane", "Erdbeere"];
```

Verwende Objekte, wenn verschiedene Informationen eine Sache **beschreiben**:

```
const lieblingsFilm = {
  titel: "Die Reise zum Mond",
  jahr: 2024,
  gesehen: true
};
```

Objekte mit verschiedenen Datentypen

Die Werte eines Objekts können unterschiedliche Datentypen haben:

```
const kurs = {
  titel: "JavaScript für Anfänger",
  lektionen: 20,
  aktiv: true,
  beschreibung: null
};
```

In diesem Objekt ist:

- `titel` ein String,
- `lektionen` eine Zahl,
- `aktiv` ein Boolean,
- `beschreibung` hat momentan den Wert `null`.

Auch Arrays und andere Objekte können als Werte gespeichert werden.

```
const rezept = {
  name: "Obstsalat",
  portionen: 2,
  zutaten: ["Apfel", "Banane", "Trauben"]
};

console.log(rezept.zutaten[1]);
```

Ausgabe:

```
Banane
```

Hier greifst du zuerst auf die Eigenschaft `zutaten` zu. Sie enthält ein Array. Anschließend wählst du mit `[1]` das zweite Element des Arrays aus.

Verschachtelte Objekte

Ein Objekt kann wiederum ein anderes Objekt enthalten. Das nennt man ein **verschachteltes Objekt**.

```
const benutzer = {
  name: "Sofia",
  kontakt: {
    email: "sofia@example.de",
    stadt: "Köln"
  }
};
```

Um die Stadt auszugeben, verbindest du die Zugriffe:

```
console.log(benutzer.kontakt.stadt);
```

Ausgabe:

```
Köln
```

Die Struktur lässt sich so lesen:

1. Nimm das Objekt `benutzer`.
2. Daraus nimm die Eigenschaft `kontakt`.
3. Daraus nimm die Eigenschaft `stadt`.

Eine Liste von Objekten

Sehr häufig werden **Arrays und Objekte kombiniert**. Zum Beispiel kann ein Array mehrere Produkte enthalten. Jedes Produkt ist dabei ein eigenes Objekt.

```
const produkte = [
  {
    name: "Notizbuch",
    preis: 4.99
```

```
  },
  {
    name: "Stift",
    preis: 1.5
  },
  {
    name: "Rucksack",
    preis: 39.99
  }
];
```

Du kannst nun ein bestimmtes Produkt auswählen:

```
console.log(produkte[0].name);
console.log(produkte[2].preis);
```

Ausgabe:

```
Notizbuch
39.99
```

Mit einer `for`-Schleife kannst du alle Produkte ausgeben:

```
for (let i = 0; i < produkte.length; i++) {
  console.log(`${produkte[i].name}: ${produkte[i].preis} Euro`);
}
```

Ausgabe:

```
Notizbuch: 4.99 Euro
Stift: 1.5 Euro
Rucksack: 39.99 Euro
```

Methoden in Objekten

Ein Objekt kann nicht nur Daten speichern, sondern auch eine Funktion enthalten. Eine Funktion innerhalb eines Objekts heißt **Methode**.

```
const hund = {
  name: "Bello",
  begruessen: function () {
    console.log("Wuff! Ich heiße " + this.name + ".");
  }
};

hund.begruessen();
```

Ausgabe:

```
Wuff! Ich heiße Bello.
```

Das Wort `this` bedeutet hier: *dieses Objekt*. In der Methode steht `this.name` also für `hund.name`.

Für den Einstieg genügt es, sich zu merken:

- Eigenschaften speichern Daten.
- Methoden führen eine Aktion aus, die zu diesem Objekt gehört.

Häufige Fehler

Eigenschaftsnamen vergessen

Zwischen Eigenschaft und Wert muss ein Doppelpunkt stehen:

```
const person = {
  name: "Mira",
  alter: 24
};
```

Das wäre falsch:

```
const person = {
  name = "Mira"
};
```

Komma zwischen Eigenschaften vergessen

```
const person = {
  name: "Mira",
  alter: 24
};
```

Zwischen den Eigenschaften steht ein Komma. Nach der letzten Eigenschaft ist kein Komma erforderlich.

Punktnotation bei einer Variablen verwenden

Wenn der Eigenschaftsname in einer Variablen steckt, brauchst du eckige Klammern:

```
const konto = {
  inhaber: "Emil",
  kontostand: 250
};

const auswahl = "kontostand";

console.log(konto[auswahl]);
```

Nicht:

```
console.log(konto.auswahl);
```

Auf ein nicht vorhandenes Objekt zugreifen

Dieser Code führt zu einem Fehler, weil `adresse` nicht existiert:

```
const person = {
  name: "Mira"
};

console.log(person.adresse.stadt);
```

Prüfe zunächst, ob die Eigenschaft vorhanden ist:

```
if (person.adresse !== undefined) {
  console.log(person.adresse.stadt);
}
```

Übung: Ein Filmobjekt erstellen

Erstelle ein Objekt namens `film`. Es soll diese Eigenschaften enthalten:

- `titel` mit einem Filmtitel
- `jahr` mit dem Erscheinungsjahr
- `genre` mit einem Genre
- `gesehen` mit dem Wert `true` oder `false`

Gib anschließend einen Satz in dieser Form aus:

```
Der Film ... erschien im Jahr ....
```

Ändere danach den Wert von `gesehen` auf `true` und ergänze eine Eigenschaft `bewertung`.

Mögliche Lösung

```
const film = {
  titel: "Sternenreise",
  jahr: 2022,
  genre: "Science-Fiction",
  gesehen: false
};

console.log(`Der Film ${film.titel} erschien im Jahr ${film.jahr}.`);
```

```
film.gesehen = true;
film.bewertung = 8;


console.log(film);
```

Das Wichtigste auf einen Blick

- Objekte speichern mehrere zusammengehörige Werte in einer Struktur.
- Eigenschaften bestehen aus einem Namen und einem Wert, etwa `name: "Mira"`.
- Mit `objekt.eigenschaft` liest oder änderst du Eigenschaften.
- Mit `objekt["eigenschaft"]` kannst du Eigenschaften dynamisch auswählen.
- Objekte können Zahlen, Texte, Booleans, Arrays, Funktionen und weitere Objekte enthalten.
- Arrays mit Objekten sind ideal für Listen wie Produkte, Benutzer oder Filme.

Revision #1
Created 2026-07-24 13:44:22 UTC by art10m
Updated 2026-07-24 13:44:23 UTC by art10m