


Datenvisualisierung mit Chart.js

- freie Bibliothek (JavaScript)

① Einbinden in das Projekt \nearrow CDN (Content Delivery Network)
 \searrow lokal

② HTML-Element `<canvas>` `</canvas>` \rightarrow  `width` `height`

③ Zugriff auf Canvas (JS) \Rightarrow `getElementById`

④ 2d-Zeichenkontext „rausziehen“ `getContext`

⑤ JavaScript-Objekt erzeugen \rightarrow `new Chart()`

⑥ labels: beschriftung x-Achse
datasets:

label: Titel
data: Werte

Chart.js: Eine umfassende JavaScript-Datenvisualisierungsbibliothek

Chart.js ist eine leistungsstarke und äußerst vielseitige JavaScript-Bibliothek, die es Entwicklern ermöglicht, Daten auf ansprechende und interaktive Weise in Diagrammen und Grafiken darzustellen. Mit ihren zahlreichen Funktionen und Anpassungsmöglichkeiten ist diese Bibliothek besonders wertvoll für Webentwickler und Datenanalysten.

Chart.js Demo

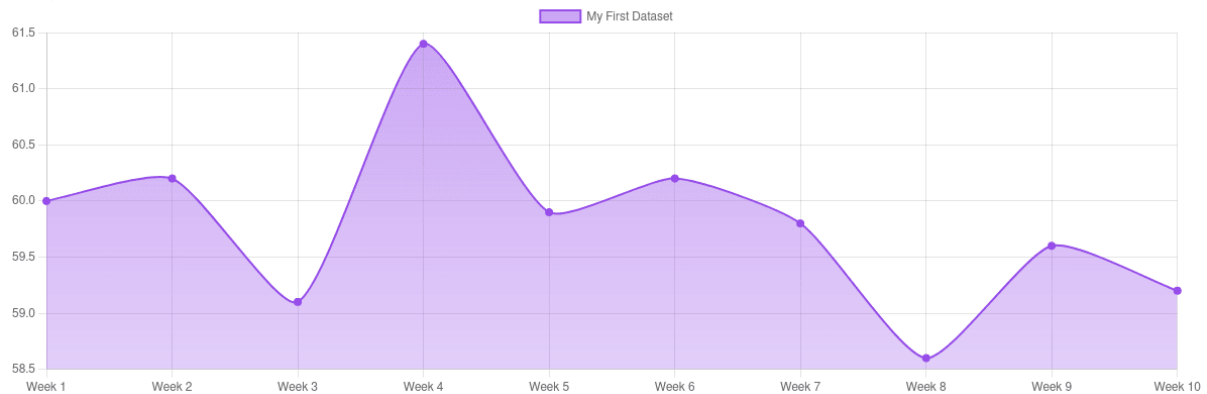
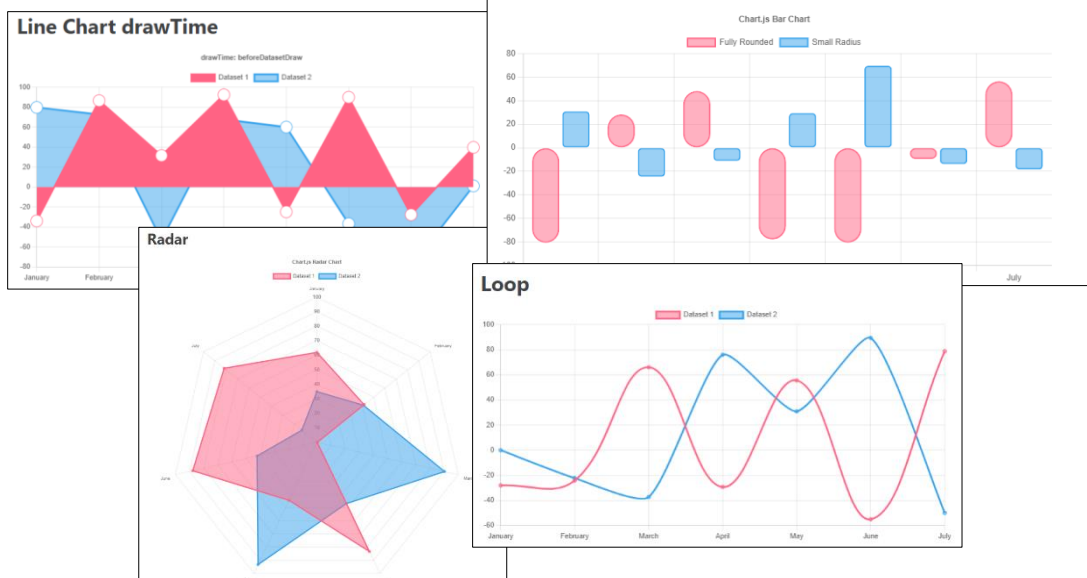


Chart.js ist eine Open-Source-JavaScript-Bibliothek, die von Nick Downie entwickelt wurde und auf HTML5 Canvas-Elementen basiert. Sie bietet eine einfache Möglichkeit, Daten in Diagrammen und Grafiken zu visualisieren, ohne auf zusätzliche Technologien wie Flash oder Silverlight zurückzugreifen. Diese Bibliothek zeichnet sich durch ihre geringe Einstiegshürde und die schnelle Integration in bestehende Webanwendungen aus.

Die Hauptkomponente von Chart.js ist ein Canvas-Element, das in eine HTML-Seite eingefügt wird. Mit Hilfe von JavaScript kann dann der gewünschte Diagrammtyp, die Daten und Anpassungsoptionen definiert werden. Chart.js kümmert sich anschließend darum, die Daten in visuell ansprechende und interaktive Diagramme zu verwandeln.

Beispieldiagramme mit Chart.js



Eine vollständige Übersicht aller möglichen Diagramme mit Chart.js finden Sie auf:

<https://www.chartjs.org/docs/latest/samples/information.html>

Hauptmerkmale von Chart.js

Chart.js verfügt über eine Vielzahl von Funktionen und Anpassungsmöglichkeiten, die es zu einer beliebten Wahl für Datenvisualisierung in Webanwendungen machen:

- **Vielfältige Diagrammtypen:**
Chart.js unterstützt eine breite Palette von Diagrammtypen, darunter:
- **Linien- und Flächendiagramme:**
Ideal zur Darstellung von Trends und Entwicklungen über einen Zeitraum. Diese Diagramme sind nützlich, um beispielsweise Umsatzentwicklungen oder Temperaturschwankungen darzustellen.
- **Balkendiagramme:**
Perfekt, um Vergleiche zwischen verschiedenen Kategorien oder Gruppen darzustellen. Sie werden oft für Umsatzvergleiche nach Produktkategorien oder demographischen Daten verwendet.
- **Kreis- und Tortendiagramme:**
Diese Diagramme sind hervorragend geeignet, um den Anteil von Teilen an einem Ganzen darzustellen. Zum Beispiel den Marktanteil verschiedener Unternehmen in einer Branche.
- **Radar- und Polarflächendiagramme:**
Diese Diagramme sind nützlich, um mehrere Datenpunkte mit verschiedenen Attributen zu vergleichen. Beispielsweise können sie verwendet werden, um die Leistung von Sportlern in verschiedenen Disziplinen zu vergleichen.
- **Anpassbarkeit:**
Die Bibliothek bietet umfangreiche Anpassungsoptionen für Farben, Schriftarten, Achsentypen und -skalen sowie Tooltips. Entwickler können Diagramme perfekt an die Anforderungen ihrer Anwendungen anpassen.
- **Interaktivität:**
Chart.js ermöglicht die Interaktion mit Diagrammen. Benutzer können beispielsweise auf Balken oder Punkte klicken, um weitere Informationen anzuzeigen. Diese Funktion ist besonders nützlich für datenreiche Anwendungen.
- **Animationsunterstützung:**
Die Bibliothek unterstützt Animationen, die die Datenvisualisierung ansprechender gestalten. Diagramme können bei Aktualisierungen sanft animiert werden, was die Benutzererfahrung verbessert.
- **Responsive Design:**
Chart.js ermöglicht es Entwicklern, Diagramme an verschiedene Bildschirmgrößen anzupassen. Dies ist entscheidend, um sicherzustellen, dass Diagramme auf Desktops, Tablets und mobilen Geräten gleichermaßen gut funktionieren.
- **Plugin-System:**
Entwickler können Chart.js mit verschiedenen Plugins erweitern, um zusätzliche Funktionen und Diagrammtypen hinzuzufügen. Dies macht die Bibliothek äußerst flexibel und erweiterbar.

Anpassung und Erweiterung

Chart.js bietet zahlreiche Anpassungsmöglichkeiten und Erweiterungen. Sie können Farben, Schriftarten, Achsentypen, Tooltips und vieles mehr anpassen. Darüber hinaus können Sie Plugins verwenden, um zusätzliche Funktionen und Diagrammtypen hinzuzufügen.

Nützliche Ressourcen:

Offizielle Chart.js-Dokumentation	https://www.chartjs.org/docs/latest/	Die Dokumentation ist eine hervorragende Ressource, um die verschiedenen Funktionen und Optionen von Chart.js zu erlernen.
Chart.js GitHub-Repository	https://github.com/chartjs/Chart.js	Hier finden Sie den Quellcode von Chart.js sowie eine aktive Community, die an der Weiterentwicklung der Bibliothek arbeitet.
Chart.js Beispiele	https://www.chartjs.org/samples/latest/	Die Beispiele auf der offiziellen Website helfen Ihnen dabei, verschiedene Diagrammtypen und Anpassungsmöglichkeiten in Aktion zu sehen.



Welches HTML-Element wird in Chart.js verwendet, um Diagramme in Webanwendungen darzustellen?

`<canvas>` `</canvas>`



Nennen Sie mindestens zwei verschiedene Diagrammtypen, die von Chart.js unterstützt werden, und beschreiben Sie kurz ihre möglichen Einsatzzwecke in der Praxis.

Tortendiagramm: Anteil an ganzen

Liniendiagramm: Trends

Der Aufbau eines Diagramms mit Chart.js

1. Die Chart.js Bibliothek einbinden:

Der erste Schritt besteht darin, Chart.js in das Projekt einzubinden. Dies kann durch das Hinzufügen des entsprechenden Skript-Tags im HTML-Code erfolgen:

```
<head>
  <link href="style.css" rel="stylesheet">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chart.js Übung</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
```

In diesem Beispiel wurde Chart.js über ein CDN eingebunden. Beachten Sie, dass es dafür unterschiedliche Links zu CDNs gibt. Recherchieren Sie bei Bedarf nach einem gültigen CDN-Link im Internet. Wahlweise kann Chart.js natürlich auch heruntergeladen und lokal in ein Projekt eingebunden werden.

2. HTML-Canvas-Element:

Das `canvas`-Element ist ein HTML5-Element, das dazu dient, Grafiken, Zeichnungen und visuelle Inhalte direkt in einer Webseite zu rendern. Es bietet eine leere Fläche (Canvas), auf der JavaScript-Code ausgeführt werden kann, um Grafiken dynamisch zu zeichnen. Das `canvas`-Element ermöglicht es Entwicklern, interaktive Grafiken, Diagramme, Spiele und andere visuelle Elemente in Webseiten zu integrieren.

- Das `canvas`-Element wird durch den HTML-Tag ``<canvas>`` erstellt.
- Die `id` dient dazu, das Element eindeutig zu identifizieren.
- Die `width` und `height` Attribute legen die Breite und Höhe des Canvas in Pixeln fest.

Beispiel-Syntax zum Anlegen eines Canvas-Elements in HTML:

```
<canvas id="meinCanvas" width="800" height="400"></canvas>.
```

3. JavaScript Zugriff auf das Canvas-Element:

Der Zugriff auf das `canvas`-Element erfolgt über JavaScript, in dem ein sogenannter "2D-Kontext" erstellt wird. Dieser Kontext ermöglicht es, Zeichenoperationen durchzuführen.

```
const canvas = document.getElementById('meinCanvas');
const context = canvas.getContext('2d');
```

- `getElementById` wird verwendet, um das `canvas`-Element zu referenzieren.
- `getContext('2d')` erstellt einen 2D-Zeichenkontext, der zum Zeichnen auf das Canvas verwendet wird.

4. Daten für das Diagramm:

Die Daten, die in einem Diagramm dargestellt werden sollen, werden in einem JavaScript-Objekt strukturiert. Dieses Objekt enthält Informationen wie Labels, Datensets und Konfigurationsoptionen.

Beispiel für Daten in einem Diagramm:

```
const daten = {
  labels: ['Jan', 'Feb', 'Mar', 'Apr', 'Mai'], // Beschriftung der X-Achse
  datasets: [
    {
      label: 'Umsatz', //Titel der dargestellten Daten
      data: [1500, 2000, 1800, 2500, 2100], //Werte der Daten
      backgroundColor: 'rgba(75, 192, 192, 0.2)', // Hintergrundfarbe der Balken
      borderColor: 'rgba(75, 192, 192, 1)', // Randfarbe der Balken
      borderWidth: 1 // Breite des Balkenrands
    }
  ]
};
```

- **`labels`**: Ein Array von Beschriftungen für die Achsen (x-Achse bei horizontalen Diagrammen, y-Achse bei vertikalen Diagrammen).
- **`datasets`**: Ein Array von Datensätzen, wobei jeder Datensatz Informationen über die Datenpunkte und das Aussehen enthält.

Info zur Angabe von Farben in Diagrammen (siehe Code-Beispiel oben):

Die Angabe **`backgroundColor: 'rgba(75, 192, 192, 0.2)'`** in Chart.js definiert die Hintergrundfarbe eines Diagramm-Elements, wie zum Beispiel eines Balkens in einem Balkendiagramm. Hier ist eine Aufschlüsselung der einzelnen Teile dieser Farbangabe:

- ✓ **`rgba`**: Steht für "Red, Green, Blue, Alpha" und ist ein Farbmodell, das die Farben basierend auf den Werten für Rot, Grün, Blau und den Alphakanal (Transparenz) beschreibt.
- ✓ **`(75, 192, 192, 0.2)`**: Dies sind die Werte für Rot, Grün, Blau und den Alphakanal, die in diesem Fall eine Türkisfarbe mit einer Transparenz von 0,2 (20%) ergeben.
- ✓ **`75`**: Rotanteil (0-255)
- ✓ **`192`**: Grünanteil (0-255)
- ✓ **`192`**: Blauanteil (0-255)
- ✓ **`0.2`**: Alphakanal (0.0 - 1.0, wobei 0.0 transparent und 1.0 undurchsichtig ist)

Die Hintergrundfarbe wird auf verschiedene Arten verwendet, abhängig vom Diagrammtyp. Im Falle eines Balkendiagramms wird diese Farbe für die Füllung des Balkens verwendet.

Es ist wichtig zu beachten, dass diese Farbangaben variieren können, abhängig von der Art des Diagramms, den Designpräferenzen und den Anforderungen des Projekts. Sie können auch andere Formate wie **Hexadezimalwerte oder CSS-Farbnamen** verwenden. (z.B. **backgroundColor: blue**)

5. Das Diagramm-Objekt erstellen:

Das Diagramm wird durch die Verwendung des Canvas-Elements und der Chart.js-Bibliothek erstellt. Dabei wird der Typ des Diagramms (z. B. Linien-, Balken-, Kreisdiagramm) angegeben.

```
const ctx = document.getElementById('meinDiagramm').getContext('2d');
const meinDiagramm = new Chart(ctx, {
  type: 'bar', // Typ des Diagramms
  data: daten, // Datenobjekt
  options: {} // Optionen für das Diagramm (optional)
});
```

- **`type`**: Gibt den Typ des Diagramms an, z. B. 'bar' für ein Balkendiagramm.
- **`data`**: Das Datenobjekt, das die strukturierten Informationen über das Diagramm enthält.
- **`options`**: Konfigurationsoptionen, die das Aussehen und Verhalten des Diagramms beeinflussen können (optional).

6. Konfigurationsoptionen (optional):

Die **`options`** im Diagramm-Objekt ermöglichen es, verschiedene Aspekte des Diagramms zu konfigurieren. Dies umfasst Einstellungen für Achsen, Legenden, Farben und vieles mehr.

Beispiel für Options:

```
const options = {
  responsive: true,
  maintainAspectRatio: false,
  scales: {
    y: {
      beginAtZero: true
    }
  }
};
```

- **`responsive`**: Legt fest, ob das Diagramm auf verschiedene Bildschirmgrößen reagieren soll.
- **`maintainAspectRatio`**: Steuert, ob das Seitenverhältnis des Canvas-Elements beibehalten werden soll.
- **`scales`**: Konfiguration der Achsen, hier speziell der y-Achse.



Auf der nächsten Seite sehen Sie den vollständigen Beispiel-Code für die Erstellung eines einfachen Balkendiagramms, inkl. Darstellung im Web.

```
<!DOCTYPE html>
<html lang="de">
<head>
  <link href="style.css" rel="stylesheet">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chart.js Übung</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
```

Schritt 1: Einbinden der Chart.js Bibliothek über ein CDN

```
<body>
  <div class="heading">
    <h1>Chart.js Diagramm</h1>
  </div>
```

Schritt 2: Ein HTML-Canvas-Element anlegen mit eindeutiger ID. Größe je nach Wahl anpassen.

```
  <div class="chart">
    <canvas id="meinDiagramm" style="max-width:70%; max-height:400px"></canvas>
  </div>
```

Schritt 6: Das Diagramm wird mit Daten befüllt.
labels = Werte auf X-Achse
datasets: Inhalte des Diagramms;
Farben nach Belieben abändern oder Standardfarben belassen.

```
<script>
  var daten = {
    labels: ["Januar", "Februar", "März", "April"],
    datasets: [{
      label: 'Umsatz in Euro',
      data: [12600, 13400, 23444, 24444],
      //backgroundColor: 'rgba(75, 192, 192, 0.2)',
      //borderColor: 'rgba(75, 192, 192, 1)',
    }],
  }
</script>
```

Schritt 3: Das Canvas-Element aus dem DOM-Tree in einer Variable speichern. Inkl. `getContext('2d')`;

```
var canvas = document.getElementById("meinDiagramm").getContext('2d');
```

```
var meinDiagramm = new Chart(canvas, {
  type: 'bar',
  data: daten,
});
</script>
```

Schritt 4: Ein Chart-Objekt erzeugen. → **new Chart()** (Mit dem Canvas-Element als Parameter)

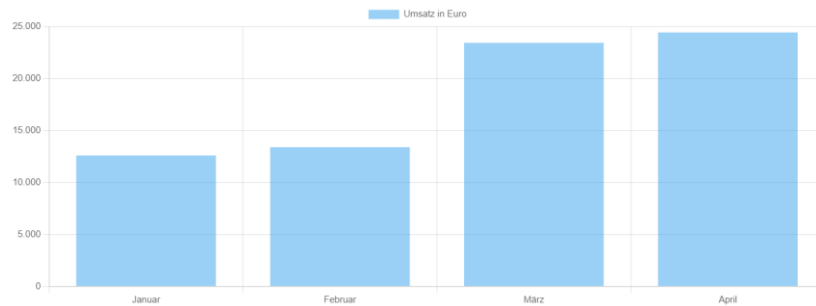
```
</body>
</html>
```

Schritt 5: Den Diagramm-Typ mit **type: 'XXX'** angeben. (z.B. `'bar'`)

Die Daten für das Diagramm mit **data: daten** angeben und anschließend separat initialisieren. (siehe Schritt 6)

Das fertige Diagramm würde in diesem Beispiel so aussehen:

Chart.js Diagramm



Balken-Diagramm mit mehreren Labels

Das Balkendiagramm kann bei Bedarf auch weitere Labels enthalten. (wie zum Beispiel ‚Umsatz in Euro‘ und ‚verkaufte Stückzahlen‘) – Die einzelnen Labels lassen sich bei einem Chart.js-Diagramm per Mausklick ein- und ausblenden.

Beispiel-Code für ein Balkendiagramm mit zwei Labels:

```
var daten = {
  labels: ["Januar", "Februar", "März", "April"],
  datasets: [{
    label: 'Umsatz in Euro',
    data: [12600, 13400, 23444, 24444, 4532, 200],
  },
  {
    label: 'verkaufte Stückzahlen',
    data: [8000, 16000, 15800, 22010],
  }
],
}
```

