

Zusammenfassung:

URL:

Beispiel:

- **Webserveradresse:** www.meinserver.de
- **Verzeichnis:** Script
- **Skript:** processform

Absolute URL:

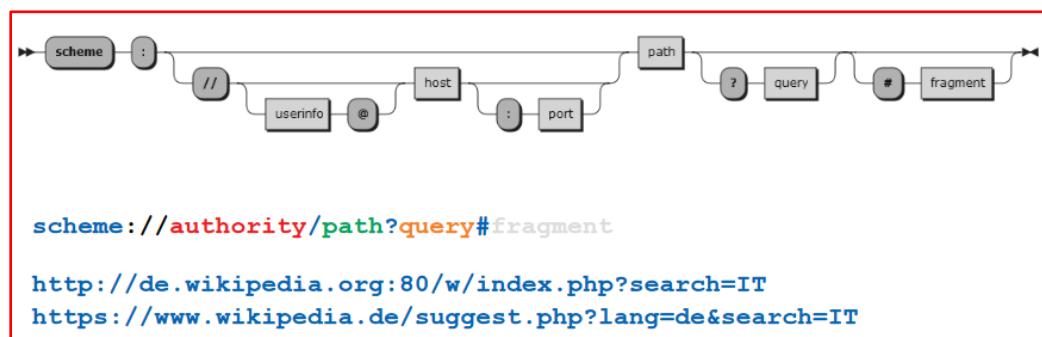
- Absolut sind mit Domäne Host und Protokoll (HTTP/HTTPS)
- Absolut: <http://www.meinserver.de/script/processform.php>

Relative URL:

- Sozusagen nur der Pfad zur Datei
- script/processform.php
 - o Pfad zur datei aus aktuellem Verzeichnis
- /script/processform.php
 - o Pfad zur datei vom Root Verzeichnis aus

Links:

- klicke diesen Link
- Falls man einen Link hinzufügen will, muss man diesen immer in <a>packen



Bedeutung der einzelnen Attribute in der URL:

Uniform Resource Identifier	URL Encoding
<ul style="list-style-type: none">- scheme<ul style="list-style-type: none">▪ Type of the URI, e.g.<ul style="list-style-type: none">▪ http HyperText Transfer Protocol▪ ftp File Transfer Protocol- authority<ul style="list-style-type: none">▪ userinfo for logins▪ host▪ port- path<ul style="list-style-type: none">▪ segments separated by /▪ e.g. index.php▪ web servers often supply a missing index.php or index.html	<ul style="list-style-type: none">- query<ul style="list-style-type: none">▪ attribute=value pairs▪ sequenced by #▪ e.g. ?search=IT&lang=en- fragment<ul style="list-style-type: none">▪ Reference to a place within the resource▪ HTML uses this to address id tags

Wie Link ändern dass er funktioniert mit Leertaste:

Schreiben Sie mit möglichst wenig Änderungen die nachfolgende URL so um, dass sie funktioniert!

<http://www.example.com/Zweite + Aufgabe.html>

- Mit – kann man Dinge verbinden, mit + kann man Leertaste ersetzen

Beispiel absolute und relative Pfade files on the server

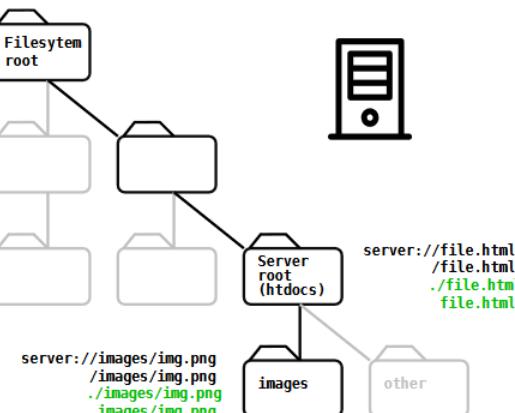
Web Server References

- absolute
 - scheme + path
 - <http://www.ct.de>
- relative
 - without starting slash
 - relative path reference
 - [index.html](#)
 - single starting slash
 - absolute path reference within server
 - </files/images/beach.jpg>
 - double starting slash
 - network path reference
 - <//www.ct.de/index.php>

These are references as seen in a browser.

The files on a web server are stored in a different tree – the web servers directory tree!

Files on the Server



Kann die URL ein HTML Document returnen?

- 1.1 Kann die nachfolgende URL ein HTML-Dokument zurückliefern und warum (nicht)?

Can the following URL return a HTML document and why (not)?

<http://www.example.com/style>

- Ja, die URL kann ein HTML-Dokument zurückliefern, wenn der Server den Content-Type als text/html in den HTTP-Header angibt und das Dokument das richtige HTML-Format hat, inklusive <!DOCTYPE html>.
- Erklärung:
 - URL selbst: Die URL an sich bestimmt nicht den Inhalt. Der Server entscheidet, welcher Inhalt zurückgegeben wird.
 - Content-Type Header: Der Server muss den richtigen Content-Type (text/html) im HTTP-Header senden, um dem Browser mitzuteilen, dass es sich um ein HTML-Dokument handelt.
 - HTML-Format: Das Dokument sollte korrektes HTML sein und mit einem gültigen <!DOCTYPE html> beginnen, um als HTML-Dokument erkannt zu werden.

Wie weiß Browser welche Datei?

Ein Browser ruft die Seite www.example.com/getit.php?id=20 auf.

Woher weiß der Browser, ob html-Text oder ein Bild zurückkommt und das Datei-Format des Bildes?

4 Pun

- Der "Content-Type"-Header in der HTTP-Antwort gibt an, welche Art von Daten zurückgesendet werden, z.B. text/html für HTML-Text oder image/png für ein PNG-Bild.
- Browser verwendet diese Informationen, um den Inhalt entsprechend zu verarbeiten und darzustellen.
- **HTTP-Anfrage:**
 - o Der Browser sendet eine Anfrage an die URL www.example.com/getit.php?id=20.
- **Server-Antwort:**
 - o Der Server verarbeitet die Anfrage und sendet eine HTTP-Antwort zurück an den Browser.
 - o Diese Antwort enthält einen Header namens "Content-Type", der den Typ der zurückgesendeten Daten spezifiziert.

Query Parameter:

- Bestandteil von URLs, zur Übermittlung von Daten an einen Webserver
- Befinden sich nach einem Fragezeichen ? Und bestehen aus Schlüssel-Wert-Paaren
- Werden mit & verknüpft
- Basis-URL:<http://www.meinserver.de/script/processform.php>
- Query-String: ?useride=Steve&password=secret&captcha=1337
- Mögliche Query Parameter:
 - o **name**: Der Name des Benutzers.
 - o Beispiel: name=Max
 - o **email**: Die E-Mail-Adresse.
 - o Beispiel: email=max@example.com
 - o **age**: Das Alter des Benutzers.
 - o Beispiel: age=25
 - o **city**: Die Stadt des Benutzers.
 - o Beispiel: city=Berlin
 - o **gender**: Das Geschlecht des Benutzers.
 - o Beispiel: gender=male
 - o **userid**=Steve
 - o **password**=secret
 - o **captcha**=1337

Wie kann der Browser das Format des Captcha-Bild erkennen welches im Formular vorliegt?

- Normalerweise erkennt er es am tag
- Browser erkennt das Format also anhand der Dateiendung im src-Attribut (zB .jpg)
 - o Zum Beispiel gibt es einen image tag
- Wenn kein Image Tag gegeben ist und eine Datei geöffnet werden muss:
 - o -> dieses captcha.php Skript generiert das Bild
 - o Bildinhalt wird zusammen mit entsprechenden MIME-Typ im HTTP-Header zurück.
 - MIME-Typ teilt dem Browser mit, welches Format das Bild hat, das Skript selbst fügt keine Dateiendung wie .jpg hinzu

Woran Erkennt Browser Format von php Bild Datei?

1.2 Woran erkennt der Browser, in welchem Format das Bild in „headerline“ vorliegt? (z.B. .gif, .jpg oder .png)

-
 1. **Server liefert Header aus:** Der Server sendet HTTP-Header zusammen mit der Bilddatei.
 2. **Enthält MIME-Type:** Diese Header enthalten den MIME-Typ, der das Format des Bildes angibt (z.B., image/gif, image/jpeg, image/png).

Request Response:

Request-Response

- Request-Message
 - from client (browser)
 - starts with a request line: method, URI, protocol version
 - followed by header fields
 - request modifiers
 - client information
 - representation metadata
 - an empty line ends the header
 - the message body may contain payload data

- Response-Message(s)
 - status line: protocol version, success or error code, textual reason
 - possibly followed by
 - server information
 - resource metadata
 - representation metadata
 - an empty line ends the header
 - the message body may contain payload data

DNS:

Domain Name System DNS

- a hierarchical and decentralized naming system for resources connected to the internet
 - Domain Name Server translate names of resources to IP addresses
 - and vice versa
 - **blocking of websites is often achieved by deleting the record in the official DNS servers**
 - the DNS consists of a tree structure
 - the root node at top “.” is often not written
 - from then the tree subdivides into zones
 - hierachies are written from bottom up
 - highest hierarchy at the end
- www.hs-esslingen.de

HTTP Status Codes:

HTTP Status Codes

- Return codes from the web server

1xx Information
 2xx Success
 3xx Forward
 4xx Client error
 5xx Server error

Examples

200 OK
 301 Moved permanently
 302 Found
 303 See other
 403 Forbidden
 404 Not found
 418 I'm a teapot
 500 Internal server error
 503 Service unavailable



- often accompanied by a clear text explanation

Auf welchem Weg liefert ein Server HTTP Statuscodes zurück?

- Server sendet Statuscode als Teil der HTTP-Antwort
- HTTP-Statuscode ist in Kopfzeile der Antwort enthalten
- Statuscodes geben den Erfolg Misserfolg der Anfrage an
- **Von 1xx-5xx**

Woher erhält Browser Informationen über Zeichensatz?:

2.1 Woher erhält der Browser die Information, welchen **Zeichensatz** eine HTML-Datei verwendet?

How does the browser get the information, which **character set** a HTML-File uses? 2 Pu

Entweder im Content-Type oder in der HTML Datei z. B. <meta charset="UTF-8">

Formularübertragung URL an Server:

1. Get:

- a. Daten werden in der URL als sichtbare Query Parameter gesendet (zB search requests)
- b. Transer key=value pairs
- c. Mehrere key=value pairs werden mit & zusammen übergeben
- d. Separated by question mark ?
- e. Geeignet für kleine Datenmengen.
- f. Weniger sicher, da Daten in der URL sichtbar sind.
- g. <form action="http://www.meinserver.de/script/processform.php" method="get">

2. Post:

- a. Daten werden im Anfragekörper gesendet, teil des requests nicht der URL
- b. Nicht sichtbar in der URL, nicht sichtbar für User deshalb sicherer
- c. Geeignet für größere Datenmengen
- d. Sicherer, da Daten im Anfragekörper verborgen sind.
- e. URL ohne Daten, daher nicht direkt buchmarkierbar.
- f. <form action="http://www.meinserver.de/script/processform.php" method="post">

Welches Feld benötigt die Form zum Absenden der Daten noch?

- Einen Submit Button <input type="submit" value="Submit">

Was ist ein Formular in HTML und wie ist es abgebaut?

Name:

Email:

- Man erkennt hier die Eingabefelder und den Submit button.
- <button type="submit">Submit</button>

Formular in HTML Aufbau:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>Beispiel-Formular</title>
</head>
<body>
    <form action="/script/processform.php" method="post">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email"><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Weiteres Beispiel Form erstellen:

2.2 Geben Sie den Code für eine Form und ihren Inhalt an, die die Felder **model** und **quantity** hat und beim Drücken des Knopfes **BESTELLEN** diese Werte per POST an <https://www.example.com/order.php> schickt!

```
<form action="https://www.example.com/order.php" method="post">
    <label for="model">Model:</label>
    <input type="text" id="model" name="model"><br>
    <label for="quantity">Quantity:</label>
    <input type="text" id="quantity" name="quantity"><br>
    <input type="submit" value="BESTELLEN">
</form>
```

Tables

- Tables structure tabular data within rows and columns
 - the are enclosed in **table** tags
 - rows use the table row tag **tr**
 - columns use the table data tag **td**
 - header columns use the table header tag **th**
 - table header tags **thead** and table body tags **tbody**, and **tfoot** structure the table further

```
<table>
    <thead>
        <tr>
            <th>Date</th>
            <th>Weight</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>May 30th</td>
            <td>72.3 kg</td>
        </tr>
    </tbody>
</table>
```

Listen in HTML:

- :
- Erstellt eine ungeordnete Liste, die standardmäßig Aufzählungspunkte (Bullets) verwendet.
- :
- Erstellt eine geordnete Liste, die standardmäßig nummerierte Listenelemente enthält.
- :
- Definiert ein Listenelement innerhalb einer ungeordneten () oder geordneten () Liste.
- <dl>:
- Erstellt eine Definitionsliste, die Begriffe und deren Beschreibungen enthält.
- <dt>:
- Definiert einen Begriff innerhalb einer Definitionsliste (<dl>).
- <dd>:
- Definiert die Beschreibung eines Begriffs innerhalb einer Definitionsliste (<dl>)

Ungeordnete Liste mit Unterliste:

Science
Art
Video
Feature Film
Short Film
Audio
Society

```
<nav>
  <ul>
    <li>Science</li>
    <li>Art
      <ul>
        <li>Video
          <ul>
            <li>Feature Film</li>
            <li>Short Film</li>
          </ul>
        </li>
        <li>Audio</li>
      </ul>
    </li>
    <li>Society</li>
  </ul>
</nav>
```

```
<head>
  <style>
    nav ul{
      list-style-type: none;
    }
  </style>
</head>
```

- Wichtig für die richtige Hierarchie mit Unterpunkten darauf achten, dass die nächste unordered List innerhalb des Listenelements ist

Geordnete Liste mit Unterliste:

```
<ol>
  <li>Apple</li>
  <li>banana</li>
  <li>berries
    <ul>
      <li>blackberry</li>
      <li>blueberry</li>
    </ul>
  </li>
  <li>Pear</li>
</ol>
```

1. Apple
2. Banana
3. Berries
 ◦ Blackberry
 ◦ Blueberry
4. Pear

Klassen, IDs:

- <section id="contact" class="highlight">
- Es können HTML Elementen Klassen hinzugefügt werden, jedem Element können klassen hinzugefügt werden und mehrere Elemente können die gleiche Klasse besitzen
- Eine ID kann nur ein einziges Element besitzen sie kann nicht mehreren Elementen zugeordnet werden

Unterschied zwischen Div und Span:

- **Div:**
 - o Verwendung:
 - Gruppert größere Block-Elemente und Layout-Segmente
 - Strukturierung und Layout der Website
 - Container für andere Block- und Inline-Elemente
 - Gestaltung von Layouts mit CSS (z.B 'flexbox' oder 'grid')
 - o Display: Block-Level (nimmt die volle Breite des übergeordneten Containers ein)
- **Span:**
 - o Verwendung:
 - Gruppert kleinere-Elemente und Text, sowie Inline Elemente
 - Formatieren von Textteilen
 - o Display; Inline-Level (nimmt nur so viel Platz ein wie der Inhalt)
- **Unterschiede:**
 - o **Struktur:**
 - div: Block-Level, für größere Layout-Abschnitte.
 - span: Inline-Level, für kleinere Textabschnitte.
 - o **Breite:**
 - div: Nimmt die volle verfügbare Breite ein.
 - span: Nimmt nur die Breite des Inhalts ein.
 - o **Verwendung:**
 - div: Hauptsächlich für Layout und Struktur.
 - span: Hauptsächlich für Textformatierung und Inline-Stilisierung.

Aufgabe zu Span und Div wozu kann es verwendet werden?

2.2 Wozu kann man span verwenden im Vergleich zu div?

- span ist ein Inline-Element ohne semantische Bedeutung. Es wird verwendet, um Textbereiche innerhalb eines Absatzes zu formatieren, ohne den Textfluss zu unterbrechen. Im Gegensatz dazu ist div ein Block-Element, das einen ganzen Block von Inhalt umschließt und auf einer neuen Zeile beginnt.
 - a. **Inline-Element:** span beeinflusst den Textfluss nicht, während div einen neuen Block startet.
 - b. **Formatierung:** span ermöglicht die Anwendung von CSS-Stilen auf bestimmte Textbereiche innerhalb eines Absatzes, während div größere Inhaltsblöcke strukturiert.

Wie Element markieren dass einen Zeilenumbruch haben soll?

- **Span ist inline**, d.h es erzeugt keinen Zeilenumbruch
- Deshalb den **
** tag verwenden, dieser muss nicht geschlossen werden
- **<p>**Dies ist ein ****inline-Element**** und das nächste Wort folgt direkt danach.**</p>**
- **<p>**Hier ist ein Text mit einem ****Zeilenumbruch**
innerhalb eines span-Tags.</p>**
- **<div>**Dies ist ein Text innerhalb eines div-Tags.**
Hier beginnt eine neue Zeile.</div>**
- Div erzeugt dagegen immer ein Zeilenumbruch, kann auch in div verwendet werden

HTML Seite Layout bauen:



```
<!DOCTYPE html>
<html lang="en"> <!-- Deklariert das Dokument als HTML5 und legt die Sprache auf Englisch fest -->
<head>
    <meta charset="UTF-8"> <!-- Stellt die Zeichencodierung auf UTF-8 ein -->
    <!-- Stellt sicher, dass die Seite auf verschiedenen Geräten korrekt angezeigt wird -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>it_20WS</title> <!-- Titel der Webseite, der im Browser-Tab angezeigt wird -->
    <style>
        /* Setzt den Standardstil für alle Elemente zurück */
        * {
            margin: 0; /* Entfernt alle Standardränder */
            padding: 0; /* Entfernt alle Standardabstände */
            box-sizing: border-box; /* Ändert das Box-Modell, sodass Padding und Border in die Höhe und Breite einbezogen werden */
        }
        /* Setzt die Höhe für html und body auf 100%, sodass sie die gesamte Höhe des Viewports einnehmen */
        body, html {
            height: 100%;
        }
        /* Stellt den Container mit Flexbox dar und legt die Höhe fest */
        .container {
            display: flex; /* Aktiviert Flexbox für das Container-Element */
            flex-direction: column; /* Ordnet die Kinder des Containers vertikal an */
            height: 750px; /* Stellt die Höhe des Containers auf 750px ein */
        }
        /* Stellt den Header-Bereich mit Flexbox dar */
        .header {
            display: flex; /* Aktiviert Flexbox für den Header */
            justify-content: space-between; /* Verteilt den Platz zwischen den Kindern gleichmäßig */
            align-items: center; /* Zentriert die Kinder vertikal */
            background-color: #lightgrey; /* Setzt die Hintergrundfarbe auf Hellgrau */
            height: 10%; /* Setzt die Höhe auf 10% des Containers */
            padding-right: 0 20px; /* Fügt rechts einen Abstand von 20px hinzu */
        }
        /* Stellt den Navigationsbereich mit Flexbox dar */
    </style>
```

```
.container {
    display: flex; /* Aktiviert Flexbox für das Container-Element */
    flex-direction: column; /* Ordnet die Kinder des Containers vertikal an */
    height: 750px; /* Stellt die Höhe des Containers auf 750px ein */
}
/* Stellt den Header-Bereich mit Flexbox dar */
.header {
    display: flex; /* Aktiviert Flexbox für den Header */
    justify-content: space-between; /* Verteilt den Platz zwischen den Kindern gleichmäßig */
    align-items: center; /* Zentriert die Kinder vertikal */
    background-color: #lightgrey; /* Setzt die Hintergrundfarbe auf Hellgrau */
    height: 10%; /* Setzt die Höhe auf 10% des Containers */
    padding-right: 0 20px; /* Fügt rechts einen Abstand von 20px hinzu */
}
/* Stellt den Navigationsbereich mit Flexbox dar */
.nav {
    display: flex; /* Aktiviert Flexbox für die Navigation */
    justify-content: flex-end; /* Richtet die Kinder am rechten Rand aus */
    padding: 10px; /*fügt innen Abstand hinzu*/
}
```

```

/* Stellt die Navigationspunkte dar */
.nav-item {
    width: 30px; /* Setzt die Breite auf 30px */
    height: 30px; /* Setzt die Höhe auf 30px */
    background-color: #grey; /* Setzt die Hintergrundfarbe auf Grau */
    margin-left: 10px; /* Fügt links einen Abstand von 10px hinzu */
}

/* Stellt den Hauptinhalt mit Flexbox dar */
.main {
    display: flex; /* Aktiviert Flexbox für den Hauptbereich */
    justify-content: center; /* Zentriert die Kinder horizontal */
    align-items: center; /* Zentriert die Kinder vertikal */
    background-color: #grey; /* Setzt die Hintergrundfarbe auf Grau */
    flex-grow: 1; /* Lässt den Hauptbereich den verbleibenden Platz einnehmen */
}

/* Stellt den Inhalt im Hauptbereich dar */
.content {
    width: 200px; /* Setzt die Breite auf 200px */
    height: 200px; /* Setzt die Höhe auf 200px */
    background-color: #lightgrey; /* Setzt die Hintergrundfarbe auf Hellgrau */
}

/* Stellt den Footer-Bereich dar */
.footer {
    height: 5%; /* Setzt die Höhe auf 5% des Containers */
    background-color: #black; /* Setzt die Hintergrundfarbe auf Schwarz */
}

</style>
</head>

```

```

</head>
<body>
<div class="container" ><!-- Hauptcontainer, der die gesamte Seite umschließt -->
    <header class="header"><!-- Header-Bereich -->
        <h1>Klausur IT20WS</h1> <!-- Überschrift im Header -->
    </header>
    <div class="nav"><!-- Navigationsbereich -->
        <div class="nav-item"></div> <!-- Navigationspunkt -->
        <div class="nav-item"></div> <!-- Navigationspunkt -->
        <div class="nav-item"></div> <!-- Navigationspunkt -->
    </div>
    <main class="main" ><!-- Hauptinhalt der Seite -->
        <div class="content" ><!-- Inhalt im Hauptbereich -->
            <form action="https://www.example.com/order.php" method="post" ><!-- Formular zum Senden von Daten -->
                <input type="text" name="model" > <!-- Eingabefeld für das Modell -->
                <input type="text" name="quantity" > <!-- Eingabefeld für die Menge -->
                <input type="submit" name="order" value="BESTELLEN" > <!-- Absende-Button -->
            </form>
        </div>
    </main>
    <footer class="footer"></footer> <!-- Footer-Bereich -->
</div>
</body>
</html>

```

HTML Grundgerüst (essenzielle Bestandteile):

```

<!DOCTYPE html>
<html>
<head>
    <title>Grundgerüst</title>
</head>
<body>
</body>
</html>

```

- Für einen lauffähigen HTML Code ist nur das notwendig, aber es kann mehr vorkommen:
- In Head kann folgendes vorkommen:
 - title, meta, link, style, base
- In Body dinge die angezeigt werden auf Website:
- header beinhaltet banner, main beinhaltet main content,
- footer für author und **Copyright: © 1821 Kev** (**merken für Footer Fehleranalyse!**)

HTML Beispiel Code mit href links, forms:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <!-- Zeichensatz auf UTF-8 setzen -->
    <meta charset="UTF-8">

    <!-- Responsive Design aktivieren -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- Titel der Seite -->
    <title>Meine erste Webseite</title>

    <!-- Verknüpfung mit externem Stylesheet -->
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <!-- Kopfbereich der Webseite -->
    <header>
        <!-- Hauptüberschrift der Seite -->
        <h1>Willkommen auf meiner Webseite</h1>

        <!-- Navigationsbereich -->
        <nav>
            <ul>
                <!-- Navigationslinks -->
                <li><a href="#home">Home</a></li>
                <li><a href="#about">Über mich</a></li>
                <li><a href="#contact">Kontakt</a></li>
            </ul>
        </nav>
    </header>

    <!-- Hauptinhalt der Webseite -->
    <main>
        <!-- Abschnitt Startseite -->
        <section id="home">
            <h2>Startseite</h2>
            <p>Dies ist die Startseite meiner Webseite</p>
        </section>

        <!-- Abschnitt Über mich -->
        <section id="about">
            <h2>Über mich</h2>
            <p>Mein Name ist Kevin</p>
        </section>

        <!-- Abschnitt Kontakt -->
        <section id="contact">
            <h2>Kontakt</h2>
        </section>
    </main>
    <!-- Formular zur Eingabe von Kontaktdataen -->
    <form action="/submit-form" method="post">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email"><br>

        <label for="message">Nachricht:</label><br>
        <textarea id="message" name="message" rows="4" cols="50"></textarea><br>

        <input type="submit" value="Absenden">
    </form>
    </section>
</main>

<!-- Ein <div>-Element, das als Container für zusätzlichen Inhalt dient -->
<div class="additional-info">
    <h2>Zusätzliche Informationen</h2>
    <p>Hier finden Sie weitere Informationen zu verschiedenen Themen.</p>

    <!-- Ein <span>-Element, das Text innerhalb eines Absatzes hervorhebt -->
    <p>Dieses <span class="highlight">hervorgehobene</span> Wort.</p>

    <!-- Ein zusätzlicher Link -->
    <p>Weitere Informationen<a href="https://www.example.com">Seite</a>.</p>
</div>

<!-- Fußbereich der Webseite -->
<footer>
    <p>© 2024 Max Mustermann. Alle Rechte vorbehalten.</p>
</footer>
</body>
</html>
```

HTML Code der richtig auf Dateipfad zugreift:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="utf-8">
    <title>Website</title>
    <link rel="stylesheet" href="right.css">
</head>
<body>
    <header>
        <nav>
            <ul>
                <li>Seite 1
                    <ul>
                        <li>mit Extras</li>
                    </ul>
                </li>
                <li>Seite 2</li>
                <li>Seite 3</li>
            </ul>
        </nav>
    </header>
    <main>
        <aside>Lorem ipsum dolor sit amet?</aside>
        <div id="content">
            
        </div>
    </main>
    <footer>
        <p>2020 Fakultät IT</p>
    </footer>
</body>
</html>
```

HTML Code Divs und Span erstellen Link zu Bild angeben

1.1 Geben Sie auf der nächsten Seite den kompletten und validen HTML5-Code für eine deutschsprachige Webseite an, die

- ein Div mit der ID „page“ für die Seiteninhalte enthält, darin
- drei Divs mit den IDs „headerline“, „content“ und „footerline“
- das Div „headerline“ soll das durch das PHP-Script „randomimg.php“ erzeugte Bild enthalten, das relativ zum HTML-Dokument im Verzeichnis „script“ liegt.
- das „content“-Div soll enthalten:
 - ein Nav mit einer unsortierten Liste mit 4 Einträgen
 - einer der Einträge soll einen Unter-Eintrag erhalten
 - vier Divs der Klasse „newsblock“
 - jedes davon enthält einen Span mit seiner Ordnungsnummer, also „Eins“, „Zwei“, „Drei“ oder „Vier“, je nach Div

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>Seite</title>
</head>
<body>
    <div id="page">
        <div id="headerline">
            
        </div>
        <div id="content">
            <nav>
                <ul>
                    <li>Eintrag 1
                        <ul>
                            <li>Eintrag untergeordnet</li>
                        </ul>
                    </li>
                    <li>Eintrag 2</li>
                    <li>Eintrag 3</li>
                    <li>Eintrag 4</li>
                </ul>
            </nav>
            <div class="newsblock"><span>Eins</span></div>
            <div class="newsblock"><span>Zwei</span></div>
            <div class="newsblock"><span>Drei</span></div>
            <div class="newsblock"><span>Vier</span></div>
        </div>
        <div id="footerline"></div>
    </div>
</body>
</html>
```

HTML Codeanalyse Beispiel und Vergleich:

2.4 Gegeben ist der folgende fehlerhafte HTML-Code, der eine Seite mit einem Bild anzeigen soll, das relativ zum HTML-Dokument im Unterordner **images** liegt.

The following HTML code should display a page with an image, that is located in the folder **images** - relative to the HTML document

```
01: <!DOCTYPE>
02:   <html lang="de">
03:
04:     <head>
05:       <meta charset="utf-8">
06:       <title></title>
07:       <link rel="stylesheet" src="right.css">
08:     </head>
09:
10:    <body>
11:      <header>
12:        <nav>
13:          <ul>
14:            <li>Seite 1
15:            <li>mit Extras</li>
16:            </li>
17:            <li>Seite 2
18:            </li>
19:            <li>Seite 3</li>
20:          </ul>
21:        </nav>
22:        <main>
23:          <aside />Lorem ipsum dolor sit amet?
24:        </aside>
25:        <div id="content">
26:          </div>
27:        </main>
28:        <footer>
29:          &copy; 2020 Fakultät IT
30:        </footer>
31:      </body>
32:
33:    </html>
34:  </DOCTYPE>
```

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="right.css">
</head>
<body>
  <header>
    <nav>
      <ul>
        <li>Seite 1</li>
        <li>mit Extras</li>
        <li>Seite 2</li>
        <li>Seite 3</li>
      </ul>
    </nav>
  </header>
  <main>
    <aside>Lorem ipsum dolor sit amet?</aside>
    <div id="content">
      
    </div>
  </main>
  <footer>
    &copy; 2020 Fakultät IT
  </footer>
</body>
</html>
```

Zeile Line	Fehlerbeschreibung Error Description
01	<!DOCTYPE> anstatt <DOCTYPE>
06	no title
07	src="right.css" ist falsch, muss href="right.css" sein
11	</header> fehlt (wird nicht geschlossen)
14	 fehlt
15	"mit Extra" sollte in anstatt stehen
26	img src=images anstatt img src=/images wegen relativen Pfad
23	<aside/> ist falsch, muss <aside> stehen
29	© anstatt ©
34	</DOCTYPE> ist falsch, muss entfernt werden

Möglicherweise sinnvolle Lecture Slides zu erstem Kapitel und HTML:

Cookies:

- Session Cookies
- Permanent Cookies
- Secure Cookies
- Third Party Cookies

Overview

- Cookies are small text files that store user data on the user side
 - up to 4k bytes
 - up to 50 cookies per domain
 - stored in a cookie jar
 - managed by the browser

- A remote website can
 - set cookie data
 - retrieve cookie data that it has set later
 - e.g. when the user has moved to another page of the same domain
- a website can only retrieve cookies of its own domain

Cookie Structure: Server → Client

- In the Response-Header:
 - Set-Cookie: <cookie>
- The Cookie <cookie>:
 - a name-value pair
 - followed by zero or more attribute-value pairs
 - separated by a semicolon

```
Set-Cookie: MoodleSession=4lbb72bo98pj1fvjhr67991m8v; path=/moodle/; secure
```

Attribute-Value Pairs

- **Expires**
 - maximum lifetime given as date and time
- **Max-Age**
 - maximum lifetime given in seconds
 - has priority over Expires
- **Domain**
 - specifies the domain wherin all hosts get the cookie sent
- **Path**
 - browser sends cookie only, when the path part of the requested URL matches

- **HttpOnly**
 - don't expose cookie to JavaScript
- **Secure**
 - only send via HTTPS
- Cookies without **Expires** or **Max-Age** are session cookies

Cookie Example

```
MoodleSession=4lbb72bo98pj1fvjhr67991m8v; path=/moodle/; secure
```

- **Name**
 - relative Path (browser supplies absolute part)
- **Type**: Secure cookie

Cookie Communication

- the web server sends a **Set-Cookie** header with the response
 - can even be in image requests
- the browser sends the cookies with every new request

```
HTTP/2.0 200 OK
Content-type: text/html
Set-Cookie: sweets=chocolate
Set-Cookie: taste=almond
[page content]
```

Cookie Structure: Client → Server

- In the Request-Header:
 - **Cookie**: <cookie>

- The Cookie <cookie>:
 - a name-value pair

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: sweets=chocolate; taste=almond
```

Session Cookies

- used to keep track of a visitor
 - e.g. when she is moving through several pages of a shop
 - collect items in a basket
 - track page views
 - track order progress
- required due to the stateless nature of HTTP

- expire, when the browser is closed
 - unless "start with last tabs" is selected

Third-Party Cookies

- installed by third parties
 - customer analysis
 - preference detection
 - behavior tracking
 - demographics

- have an expiration date
 - track user
 - last visit
 - recognize user
 - automatic re-login

- save user preferences on web-site
 - sort order
 - color scheme
 - preferred delivery address

- expire, when their expiration date is exceeded

Secure Cookies

- only accessible by HTTP or HTTPS
- contains an **HttpOnly** flag
- instructs browser to restrict access to the cookie
- developed by Microsoft for the IE6 to add some security

- can not be accessed by Javascript

- installation difficult, because the browser only hands out cookies to a web server of a domain they came from

Cookie Handling

- Erase Cookies
 - delete the content of the local cookie folder
 - clear browser cache including cookies
- Block Cookies
 - configure browser to block all cookies
 - prevents shopping (or seeing your moodle class!)

- Selectively Allow Cookies
 - configure browser to ask, when a web site wants to set a cookie
- Disallow third party cookies
 - as third party cookies are nearly exclusively used for behavioral tracking: block

Flash Cookies aka Super Cookies

- Flash Cookies are independent of the browser
- written by Adobe Flash
 - called Locally Shared Object LSO
- stored permanently on the computer
 - can save up to 100k
- requires Adobe Flash to be used on a web page and being enabled within the browser
- deleting ordinary cookies leaves Flash Cookies intact

CSS:

->Stripe, gradient, hatch, kreuz pattern gestalten können, repeating patterns/sich wiederholende patterns gestalten üben

Ansprechen von Klassen, IDs, und HTML Elementen:

- Beispiel <section id="contact" class="highlight">
- **Ansprechen einer ID:**
- #section{
 Font-size: 14px;
}
- **Ansprechen einer Klasse:**
- .section{
 Border: 4px;
}
- **Ansprechen soezifischer Attribute:**
- .personalBar div[href="#profil"]:hover {
 font-weight: bold; /* Setzt die Schrift auf fett beim Hover */
}
- **Ansprechen eines festen HTML Elements zB Header**
- Header{
 Background-image: url('header.jpg');
}

Relative und Absolute Positionierung:

- Relativ:
- Wird von ursprünglicher Position aus verschoben, behält Platz im Dokumentfluss bei
 - o Beeinflusst weiterhin die Positionierung nachfolgender Elemente
- Absolut:
- Entfernt Element aus dem normalen Dokumentenfluss und positioniert es relativ zu seinem nächsten positioned ancestor (Vorfahr-Element mit position:relative, absolute oder fixed, wenn nicht existent relativ zu viewport)
 - o Beeinflusst die Positionierung nachfolgender Elemente nicht

Div Stylen Beispiel: Personal Bar soll oben Rechts sein:

FAs Nachrichten Profil

```
<header>
    <!-- PersonalBar -->
    <div class="personalBar">
        <ul>
            <li><a href="#faqs">FAs</a></li>
            <li><a href="#nachrichten">Nachrichten</a></li>
            <li><a href="#profil">Profil</a></li>
        </ul>
    </div>
</header>
```

```
/* Stile für die PersonalBar */
.personalBar {
    position: absolute; /* Positioniert die PersonalBar relativ zum
                        /nächsten positionierten Vorfahren (dem Header) */
    top: 0px; /* Abstand vom oberen Rand des Headers */
    right: 0px; /* Abstand vom rechten Rand des Headers */
    background-color: white; /* Hintergrundfarbe der PersonalBar */
    border: 2px solid black; /* Rahmen der PersonalBar */
    z-index: 1;
    padding: 10px; /* Innenabstand der PersonalBar */
}
```

```
/* Stile für die Links in der PersonalBar */
.personalBar ul {
    list-style-type: none; /* Entfernt Aufzählungszeichen */
    margin: 0; /* Entfernt Außenabstände */
    padding: 0; /* Entfernt Innenabstände */
    display: flex; /* Anordnung der Listenelemente in einer Zeile*/
    gap: 10px; /* Abstand zwischen den Listenelementen */
}
.personalBar a {
    text-decoration: none; /* Entfernt Unterstreichung der Links */
    color: black; /* Setzt die Textfarbe auf schwarz */
    font-weight: bold; /* Setzt den Text fett */
}
```

Div Stylen: Hover Effekt wenn man drüber fährt und als zweites hover effekt

```
.personalBar a {  
    text-decoration: none; /* Entfernt Unterstreichung der Links */  
    color: black; /* Setzt die Textfarbe auf schwarz */  
    font-weight: bold; /* Setzt den Text fett */  
}  
  
.personalBar a:hover{  
    background-color: gray;  
    color: white;  
}
```

FAs Nachrichten Profil

```
/* Spezifischer Stil für den "Profil"-Link im Hover-Zustand */  
.personalBar div[href="#profil"]:hover {  
    font-weight: bold; /* Setzt die Schrift auf fett beim Hover */  
}
```

FAs Nachrichten Profil

```
/* Spezifischer Stil für den "Profil"-Link im Hover-Zustand */  
.personalBar li+li+li:hover {  
    font-weight: bold; /* Setzt die Schrift auf fett beim Hover */  
}
```

```
<element class="personalBar">  
    ...  
    <li>  
    <li>  
    <li :hover>  
Selector Specificity: (0, 2, 3)
```

-es ist auch so möglich überli+li+li dieses spezifische Profil anzusprechen

Beispiel: Beim Hovern ändert sich der Text zu einem anderen Text

```
.secretText {  
    position: relative; /* Positioniert das Pseudo-Element relativ zum Original-Element */  
}  
  
.secretText:hover::after/* ::after, ::before -> Pseudoelemente immer mit :: */  
content: "Dieser Text enthält keine Geheimnisse";  
position: absolute;  
top: 0;  
left: 0;  
visibility: visible;  
}  
  
.secretText:hover/* :hover, :focus -> Pseudoklassen immer mit einem : */  
visibility: hidden;  
}  
  
-- -- --  
<p class="secretText">Dieser Text enthält Geheimnisse</p>
```

- :hover ist eine Pseudoklasse, hinzugefügt ändert es den Style während dem hovern
- ::after ist der Zustand nachdem etwas passiert ist, zB :hover::after ist der Zustand nach dem hovern und die entsprechenden Styles werden danach angewandt

Selektoren in CSS:

- **Element-Selektor:**
 - Wählt alle HTML-Elemente eines bestimmten Typs aus.
 - Beispiel: `p { color: red; }` (Wählt alle `<p>`-Elemente aus)
- **Klassenselektor:**
 - Wählt alle Elemente mit einer bestimmten Klasse aus.
 - Beispiel: `.className { color: blue; }` (Wählt alle Elemente mit der Klasse `className` aus)
- **ID-Selektor:**
 - Wählt das Element mit einer bestimmten ID aus.
 - Beispiel: `#idName { color: green; }` (Wählt das Element mit der ID `idName` aus)
- **Universalselektor:**
 - Wählt alle Elemente aus.
 - Beispiel: `*` { `margin: 0;` } (Wählt alle Elemente aus)
- **Attributselektor:**
 - Wählt alle Elemente mit einem bestimmten Attribut aus.
 - Beispiel: `[type="text"] { border: 1px solid black; }` (Wählt alle Elemente mit `type="text"` aus)
- **Pseudoklassen-Selektor:**
 - Wählt Elemente basierend auf ihrem Zustand aus.
 - Beispiele:
 - o `a:hover { color: red; }` (Wählt alle Links aus, über die der Mauszeiger fährt)
 - o `:first-child { margin-top: 0; }` (Wählt das erste Kind eines Elements aus)
- **Pseudoelement-Selektor:**
 - Stellt Teile eines Elements dar, die nicht explizit im HTML-Dokument enthalten sind.
 - Beispiele:
 - o `::before { content: "Prefix"; }` (Fügt Inhalt vor einem Element hinzu)
 - o `::after { content: "Suffix"; }` (Fügt Inhalt nach einem Element hinzu)
- **Nachkommenschaftsselektor:**
 - Wählt alle Nachkommen eines bestimmten Elements aus.
 - Beispiel: `div p { color: red; }` (Wählt alle `<p>`-Elemente innerhalb eines `<div>` aus)
- **Kindselektor:**
 - Wählt alle direkten Kinder eines bestimmten Elements aus.
 - Beispiel: `ul > li { color: blue; }` (Wählt alle ``-Elemente aus, die direkte Kinder eines `` sind)
- **Adjacent-Sibling-Selektor:**
 - Wählt ein Element aus, das unmittelbar nach einem bestimmten Element folgt.
 - Beispiel: `h1 + p { margin-top: 0; }` (Wählt das `<p>`-Element aus, das direkt nach einem `<h1>`-Element folgt)
- **General-Sibling-Selektor:**
 - Wählt alle Geschwister eines bestimmten Elements aus.
 - Beispiel: `h1 ~ p { color: green; }` (Wählt alle `<p>`-Elemente aus, die Geschwister eines `<h1>`-Elements sind)

Selektoren Stylen und Zugriff auf Kinder eines Divs mit > Selector:

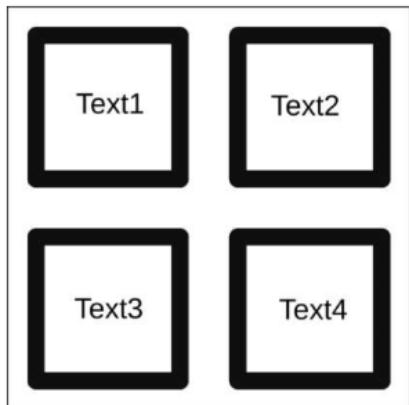
Gegeben sei folgender HTML-Code-Ausschnitt:

```
<div class="taskcontainer">
<div>Task 1</div>
<div>Task 2</div>
<div>Task 3</div>
<div>Task 4</div>
</div>
```

mit dem CSS-Styling für den **taskcontainer**:

```
.taskcontainer {
  width: 200px;
  height: 200px;
  border: 1px solid black;
}
```

so dass sich der folgenden Aufbau ergibt:



```
<title>Website</title>
<style>
  .taskcontainer{
    width: 200px;
    height: 200px;
    border: 1px solid black;
  }
```

```
.taskcontainer > div {
  width: 60px;
  height: 60px;
  border: 10px solid black;
  border-radius: 5px;
  margin: 10px;
  float: left;
  text-align: center;
  line-height: 60px;
}
```

3.1 Geben Sie das Styling für **task** an (inclusive Selektor), das die Texte zentriert in die **task**-Divs ausgibt. Ränder etc. sind alle **10px**. 8 Punkte

```
.taskcontainer > div {
  margin: 10px; /* Fügt außen um jedes div-Element herum einen Abstand von 10 Pixeln hinzu. */
  width: 60px; /* Setzt die Breite jedes div-Elements auf 60 Pixel. */
  height: 60px; /* Setzt die Höhe jedes div-Elements auf 60 Pixel. */
  border: 10px solid black; /* Fügt jedem div-Element einen 10 Pixel breiten, schwarzen, festen Rahmen hinzu. */
  float: left; /* Lässt jedes div-Element nach links fließen, wodurch sie nebeneinander angezeigt werden, wenn genügend Platz vorhanden ist. */
  text-align: center; /* Zentriert den Text horizontal innerhalb jedes div-Elements. */
  border-radius: 5px; /* Fügt jedem div-Element abgerundete Ecken mit einem Radius von 5 Pixeln hinzu. */
  line-height: 60px; /* Setzt die Zeilenhöhe auf 60 Pixel, was dem Wert der Höhe entspricht, um den Text vertikal zu zentrieren. */
}
```

Selektoren Div und Span Stylen Childs auswählen:

Stylen Sie die „newsblock“-Divs:

6 |

- quadratisch 100px
- grau mit schwarzem Rand – das zweite ausnahmsweise rot
- alle Divs nebeneinander (falls Platz ist)
- Der Text mit der Ordnungsnummer soll UNTER die Divs kommen

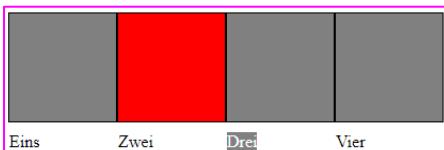


```
<div class="newsblock"><span>Eins</span></div>
<div class="newsblock"><span>Zwei</span></div>
<div class="newsblock"><span>Drei</span></div>
<div class="newsblock"><span>Vier</span></div>
```

```
.newsblock{
    width: 100px;
    height: 100px;
    background-color: grey;
    border: 1px solid black;
    float: left;
}
div.newsblock+div.newsblock{
    background-color: red;
}
div.newsblock+div.newsblock+div.newsblock{
    background-color: grey;
}
.newsblock span{
    position: relative;
    top: 110px;
}
```

Hover Effekt hinzufügen:

Beim Überfahren der Divs soll der zugehörige Text weiß auf grauem Untergrund dargestellt werden.



```
div.newsblock:hover span{
    color: white;
    background-color: gray;
}
```

Margin-bottom:

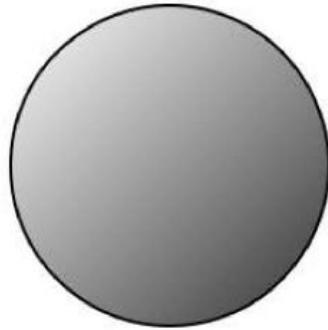
Falls die Divs untereinander kommen, soll der Text sichtbar bleiben.



```
.newsblock{
    margin-bottom: 30px;
}
```

Gradient in CSS:

Stylen ein leeres **div** mit der Class **circle** so:



```
.circle{  
    width: 200px;  
    height: 200px;  
    border-radius: 50%;  
    background: linear-gradient(-45deg, ■rgb(0, 0, 0) 0, □rgb(255, 255, 255) 100%);  
    border: 2px solid ■black;  
}
```

Div Pattern Stylen:

Dieses **div** ist quadratisch mit je 150px Kantenlänge. Die Linienbreite ist 5px und die Farben sind weiß und schwarz. Geben Sie die Styles an!



```
.quadrat{  
    border: 5px;  
    width: calc(150px-2*5px);  
    height: calc(150px-2*5px);  
    border-bottom-left-radius: 50%;  
    background: linear-gradient(45 deg, □white 0 50%, ■black 50% 100%);  
}
```

- Auf Border achten diese soll 5cm groß sein, deshalb von width und height abziehen!

CSS Selektor Slides (klausurrelevant):

Element Selectors

- Selectors select the elements to be styled
 - they precede the curly braces "{}"
 - they use the Document Object Model DOM of the browser
 - an erroneous HTML code may lead to unexpected results in style

- Classification:
 - simple element selectors
 - a combination of simple selectors for more precise selection
 - pseudo elements
 - pseudo classes
 - structural
 - dynamic

- by tag name
 - e.g. body, h1

```
body {  
    background-color:#ccf;  
}
```

- by dot . and class name
 - several elements may have the same class name

```
.attention {  
    color: white;  
    background-color:red;  
}
```

- by crosshatch # and id
 - unique identifier
 - also usable as jump target

```
#sidebar {  
    color:black;  
    background-color:#3c0;  
}
```

Attribute Selectors

Multiple Simple Selectors

- a comma , separates several element selectors that all get the same style

- a star * matches all elements

- Attribute a exists:

- [a]

- Attribute a has a certain value v:
 - [a=v]

- Attribute contains a certain word w:
 - [a~="w"]

- Attribute a starts with a certain word part w, followed be a minus
 - [a|=w]

```
html, body {  
    margin:0;  
    padding:0;  
}
```

```
* {  
    margin:0;  
    padding:0;  
}
```

```
[title] {background-color:green;}
```

```
[title=Lecture]
```

```
[title~=Homework]
```

```
[title|=Chapter]
```

Attribute Selectors

- Attribute value of attribute a starts with a certain string s:
 - [a^=s]

- Attribute value of attribute a ends with a certain string s:
 - [a\$=s]

- Attribute value of attribute contains a certain string s:
 - [a*=s]

```
[title^=Forum]
```

```
[title$=message]
```

```
[title*=german]
```

Klausurrelevant um Divs in Div anzusprechen!:

Descendant Selectors

- the Descendant Selector (space) selects elements that are descendants of a specified element
 - a b

```
p .wichtig {color:red;}
```

- the Child Selector ">" selects all elements, that are immediate children of the specified element
 - a>b

```
a>img {border-style:none;}
```

Sibling Selectors

- the Adjacent Sibling Selector selects the element that immediately follows a specified element on the same level
 - a+b

```
h1 + p { color:black;}
```

- the General Sibling Selector selects **all** elements that follow the specified element within the same parent
 - there may be other elements on the same level in between
 - a~b

```
h1 ~ p {background-color: gray;}
```

Pseudo Elemente:

Pseudo Elements

- the first line of the enclosing block
 - = ::first-line

```
.news::first-line {color: blue;}
```

- the first letter of the enclosing block
 - = ::first-letter

```
.news::first-letter {font-weight: bold;}
```

Pseudo Elements

- ::before and ::after generate Elements before or after the given element
 - = used to generate content automatically

```
.news p::before {content:'NEWS: '}
```

- ::selection selects the selected text;
 - = only some styles may be changed
 - = color, background, cursor, outline

```
::selection {color: white; background-color: black;}
```

Pseudo Klassen:

Structural Pseudo Classes

- :root the Document
- :empty empty Elements
 - = e.g. table cells
- :first-child, :last-child
- :nth-child(n), :nth-last-child(n)
 - = :nth-child(3) the 3rd child element
 - = :nth-child(even) all even positions
 - = :nth-child(odd) all odd positions

- :first-of-type, :last-of-type
 - = p:first-of-type {color: green} colorizes the text of the first paragraph
- :nth-of-type(n), :nth-last-of-type(n)
- :only-of-type if single child
- :not() all besides the simple selector
 - = :not(p)

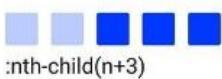
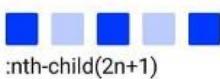
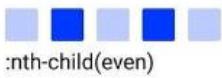
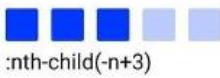
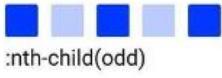
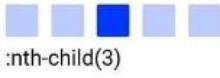
Wichtig für Verständnis falls man so spezifische Divs auswählen möchte klausurrelevant:

nth-child(arg)

- nth child takes a single argument with the pattern of the match
- keywords:
 - = odd, even
- functions
 - = An+B with
 - = A step size
 - = B offset

```
tr:nth-child(odd)  
tr:nth-child(2n+1)  
:nth-child(7)  
:nth-child(3n+4)  
:nth-child(-n+3)
```

:nth-child Properties



Dynamic Pseudo Classes

- :link, :visited unvisited and visited links
 - = only color changes
- :hover mouse over element
- :active actually pressed
- :focus has focus
 - = e.g. by tab
- :target target of actual link

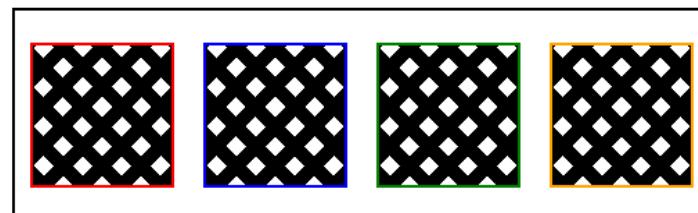
- :disabled, :enabled, :checked for check boxes
- :valid, :invalid for entry fields

```
input:invalid {color: red; font-weight: bold;}
```

```
<input type="number" min="1" max="12" />
```

CSS weitere Beispiele:

Hatch Pattern mit Div Childs:



```
<div class="hatch-container">
  <div class="hatch"></div>
  <div class="hatch"></div>
  <div></div>
  <div></div>
</div>

/* Container für die Hatch-Elemente, sorgt für horizontale Anordnung */
.hatch-container {
  display: flex; /* Flexbox Layout für horizontale Anordnung */
  justify-content: center; /* Horizontale Zentrierung */
  justify-content: space-around; /*jedes div gleichen Borderabstand*/
  align-items: center; /* Vertikale Zentrierung */
  width: 500px;
  height: 150px;
  border: 2px solid ■black;
}

/* Allgemeine Stile für die Hatch-Elemente */
.hatch-container > div {
  width: 100px; /* Breite des Divs */
  height: 100px; /* Höhe des Divs */
  background: repeating-linear-gradient(
    45deg, /* Erster diagonaler Linienwinkel */
    ■black, /* Erste Farbe der Linie */
    ■black 10px, /* Endet nach 10px in schwarz */
    transparent 10px, /* Startet nach 10px in transparent */
    transparent 20px /* Endet nach 20px in transparent */
  ),
  repeating-linear-gradient(
    -45deg, /* Zweiter diagonaler Linienwinkel */
    ■black, /* Erste Farbe der Linie */
    ■black 10px, /* Endet nach 10px in schwarz */
    transparent 10px, /* Startet nach 10px in transparent */
    transparent 20px /* Endet nach 20px in transparent */
  );
}

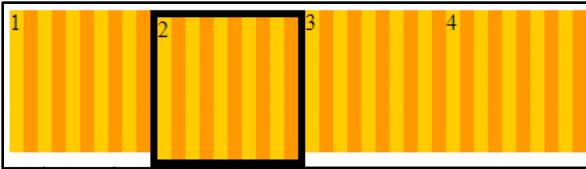
/* Stile für das erste Hatch-Element */
.hatch:nth-child(1) {
  border: 2px solid ■red; /* Rote Umrandung */
}

/* Stile für das zweite Hatch-Element */
.hatch:nth-child(2) {
  border: 2px solid ■blue; /* Blaue Umrandung */
}

/* Stile für das dritte Hatch-Element */
.hatch-container div:nth-child(3) {
  border: 2px solid ■green; /* Grüne Umrandung */
}

/* Stile für das vierte Hatch-Element */
.hatch-container div+div+div+div {
  border: 2px solid ■orange; /* Orange Umrandung */
}
```

Repeating Stripe Pattern und spezifisches Div auswählen:



```
<div class="repeating-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

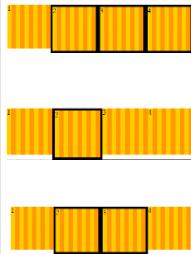
```
.repeating-container {
  display: flex; /*für horizontale Ausrichtung*/
}

.repeating-container > div {
  width: 100px;
  height: 100px;
  background: repeating-linear-gradient(
    90deg,
    #ffcc00,
    #ffcc00 10px,
    #ff9900 10px,
    #ff9900 20px
  );
}

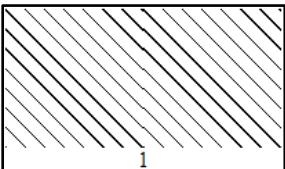
/* Wählt alle Divs ab dem 2. Div aus also Div 2, 3 und 4 ausgewählt */
.repeating-container > div+div {
  border: 5px solid black;
}

/* wählt alle divs ab dem 3. div aus und entfernt die Border */
.repeating-container > div+div+div{
  border: 0px solid black;
}

/* wählt das letzte div aus und entfernt die Border */
.repeating-container > div+div+div{
  border: 0px solid black;
}
```



Span in bestimmten Div ansprechen:



```
<div class="repeating-container">
  <div> <span>1</span></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

```
.repeating-container {
  width: 200px;
  height: 200px;
  margin: 5px;
  position: relative;
}

.repeating-container > div {
  width: 100px;
  height: 100px;
  float: left;
  background: repeating-linear-gradient(
    45deg,
    #000,
    #000 1px,
    #fff 1px,
    #fff 10px
  ),
  repeating-linear-gradient(
    135deg,
    #000,
    #000 1px,
    #000 1px,
    #000000 1px,
    #110101 10px
  );
}

/*span ansprechen der sich in einem div befindet*/
.repeating-container >div>span {
  display: block;
  width: 100%;
  text-align: center;
  position: absolute;
  top: 100px;
}
```

Cross Container mit mehreren Divs:

```
<div class="cross-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

1	2	3	4

```
.cross-container {
  display: flex; /* Display flex richtet flex Elemente automatisch horizontal aus*/
}
.cross-container > div {
  width: 100px;
  height: 100px;
  background: linear-gradient(90deg, transparent 49%, #000 49%, #000 51%, transparent 51%),
              linear-gradient(0deg, transparent 49%, #000 49%, #000 51%, transparent 51%);
  background-size: 50px 50px;
  text-align: center; /*der text in den Divs wird zentriert*/
}
/* Spezifisches div am Anfang */
.cross-container > div:first-child {
  background-color: lightblue;
}
/* Spezifisches div am Ende */
.cross-container > div:last-child {
  background-color: lightgreen;
}
/* Spezifisches div in der Mitte */
.cross-container > div:nth-child(3) {
  background-color: lightcoral;
}
/* Spezifisches div in der Gruppe */
.cross-container > div + div {
  border: 2px solid black;
}
```

Radial Gradient Beispiel:



```
body, html {
  margin: 0;
  padding: 0;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #f0f0f0;
}
/* Stile für die Box mit dem Radial Gradient */
.gradient-box {
  width: 300px;
  height: 300px;
  background: radial-gradient(circle, rgba(255,255,255,1) 0%, rgba(255,0,0,1) 50%, rgba(0,0,255,1) 100%);
  border: 2px solid #000;
  box-shadow: 0 4px 8px rgba(0,0,0,0.2);
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Repeating Radial Gradient:



```
/* Stile für die Box mit dem wiederholenden Radial Gradient */
.repeating-gradient-box {
  width: 300px;
  height: 300px;
  background: repeating-radial-gradient(
    circle,
    #ffcc00,
    #ffcc00 10px,
    #ff9900 10px,
    #ff9900 20px
  );
  border: 2px solid #000;
  box-shadow: 0 4px 8px rgba(0,0,0,0.2);
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Übersicht Folien aus der Vorlesung:

Text Stlyen:

Text Color

- Color is the **color** property with a color value
 - otherwise inherited from the enclosing element

```
h1 {  
    color: #c00; /* HSE red */  
}
```

Text Alignment

- Text can be aligned with **text-align** at
 - left
 - center
 - right
- the enclosing element must have enough space to align center or right

```
.left {  
    text-align: left;  
}  
  
.center {  
    text-align: center;  
}  
  
.right {  
    text-align: right;  
}
```

Text Decoration

- The **text-decoration** property sets or removes decorations from text
- it is a shorthand for
 - text-decoration-line**
 - text-decoration-color**
 - text-decoration-style**

```
a {  
    text-decoration: none;  
}  
  
.ol {  
    text-decoration: overline;  
}  
  
.it {  
    text-decoration: line-through;  
}  
  
.ul {  
    text-decoration: underline;  
}
```

Text Decoration

- Line
 - none
 - underline
 - overline
 - line-through
 - initial
 - inherit

```
ul {  
    text-decoration-line: underline;  
}  
  
li {  
    text-decoration-line: line-through;  
}  
  
ol {  
    text-decoration-line: overline underline;  
}
```

- Color
 - a CSS color value

```
p {  
    text-decoration: underline;  
}
```

- Style
 - solid
 - double
 - dotted
 - dashed
 - wavy
 - initial
 - inherit

```
.wavy {  
    text-decoration-line: underline;  
    text-decoration-style: wavy;  
}  
  
.double {  
    text-decoration-line: underline;  
    text-decoration-style: double;  
}
```

Font Style

- The **font-style** property specifies, whether a text is straight or slanted
- normal
- italic
- oblique
 - "poor man's italic"

```
p.normal {  
    font-style: normal;  
}  
  
p.italic {  
    font-style: italic;  
}  
  
p.oblique {  
    font-style: oblique;  
}
```

Border Styling:

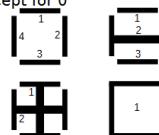
Separately styled Box Sides

- the four sides of the box elements can be styled separately
- Here shown for border:
 - complete
 - border-top
 - border-right
 - border-bottom
 - border-left
- single parameters
 - border-top-style
 - border-right-style
 - border-bottom-style
 - border-left-style

```
p {  
    border-right: 1px dotted gray;  
    border-bottom: 3px solid black;  
}
```

Several different Dimensions

- the browser accepts 4 different sizes for the thickness of the box outlines
- every number MUST have a unit
 - except for 0



- The browser supplies missing arguments in case it are less than four
- 4: top right bottom left
 - clock-wise
- 3: top (right+left) bottom
- 2: (top+bottom) (left+right)
- 1: (top+right+bottom+left)

Border Radius

- The **border-radius** is roundness of one or more corners
 - supported by newer browsers
 - older browsers just ignore this

```
.post-it {  
    width: 320px;  
    padding: 30px;  
    border: 5px solid gray;  
    border-radius: 5px;  
    margin: 50px;  
}
```

Box Model Styling:

Sizes that fit all

- = px: Pixel
 - = one pixel is one dot on the display
 - = leads to very small elements on high resolution displays
 - = fixed
- = %: Percent
 - = percent of the width of the actual enclosing element
 - = regular fonts have 100%
 - = variable
- = vw and vh:
 - = viewport width or height in percent

- = em: corresponds to the actual font size
 - = at a 12 point font 1em is 12pt
 - = it is the size of the character m
 - = grows and shrinks with the magnification
 - = variable
- = pt: a printers point
 - = 1/72 of an inch
 - = the browser scales
 - = fixed

The Box Model

- = HTML elements can be considered as boxes
- = The CSS box model is essentially a box that wraps around every HTML element.
- = It consists of:
 - = margins
 - = borders
 - = padding
 - = the actual content.

Margin

Border

Padding



Content

- = The content of the box, where text and images appear
- = The **width** and **height** parameters refer to the content
 - = in addition to:
 - = padding
 - = border
 - = margin

```
div {  
    width: 320px;  
    height: 200px;  
    padding: 30px;  
    border: 5px solid gray;  
    margin: 50px;  
}
```

- = the example div has a width of 350px and a height of 200px

Border

- = The border has 3 main properties
 - = **border-width**, which needs a unit after the number
 - = **border-style**
 - = solid, double, dashed, dotted, none
 - = **border-color**
- = these properties can be aggregated to the **border** property

```
div {  
    width: 320px;  
    height: 200px;  
    padding: 30px;  
    border: 5px solid gray;  
    margin: 50px;  
}  
  
p.l {  
    border-width: 3px;  
    border-style: dashed;  
    border-color: gray;  
}  
p.s { border: 3px dashed gray; }
```

Margin

- = The **margin** is the space around the border
 - = it separates the element from the next element
 - = has the background of the underlying element

```
div {  
    width: 320px;  
    height: 200px;  
    padding: 30px;  
    border: 5px solid gray;  
    margin: 50px;  
}
```

- = the example div has a margin of 50px

Padding

- = The **padding** is the space around the content
 - = within the border
 - = has the same background as the content

```
div {  
    width: 320px;  
    height: 200px;  
    padding: 30px;  
    border: 5px solid gray;  
    margin: 50px;  
}
```

- = the example div has a padding of 30px

Different Box Sizing

- = Normally, the size parameters width and height relate to the content
 - = **box-sizing: content-box**
- = with a different styling, the parameters relate to the
 - = content plus
 - = padding plus
 - = border
 - = **box-sizing: border-box**



Positioning Unterschied Absolute und Relative:

Absolute Positioning

- = Absolute Positioning removes the element from the layout flow
 - = and positions it at the given place
 - = the element covers other elements below
 - = can be overridden by using the z-index property

- = **position: absolute**
 - = positions absolute oriented on the next higher level positioned element
 - = also scrolls
- = **position: fixed**
 - = positions on the window
 - = does not scroll
- = **top:, bottom:, left:, right:**
 - = position parameters

Relative Positioning

- = Absolute Positioning positions absolute to the next higher level positioned element

- = Relative Positioning is a way to set a position (for lower level positioned elements) that does not change the position of **this** element
 - = but builds an anchor point for included positioned elements

Block Level und Inline Elements klausurrelevant:

Block and Inline Elements

- Block elements have their own rectangular block
 - `div, p, h1, ...`
- Inline elements occur within text lines
 - `span, strong, ...`
- CSS can switch these properties
 - `display: inline`
 - `display: block`
 - `display: inline-block`

- Hide elements with
 - `display: none`

Arranging Block Level Elements

- a Block Level Element normally causes a break
 - so consecutive block elements are positioned one under the other

- if Block Level Elements should be positioned side-by-side one uses
 - `float:left` or `float:right`
 - for images besides text
 - for multi column layout
- in this case the blocks need a `width`
- `float` takes the element out of the regular layout flow

End of Float

- `clear` ends the floating and continues after the cleared element
 - `clear: left;`
 - `clear: right;`
 - `clear: both;`

Flexbox klausurrelevant für Layout der Container:

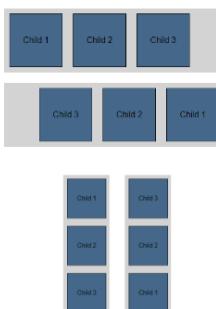
Flexbox

- Layout
 - Alignment
 - Distribution of space
- even with unknown or dynamic sizes

- uses a container
 - which alters the width/height of its children
 - to best fill the available space
- good for small-scale layouts

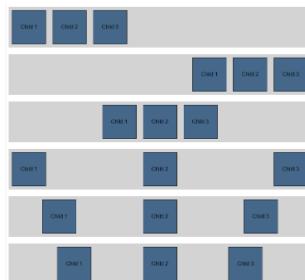
Parent Flex Direction

- flex-direction** establishes the direction of flow of flex items
 - `row`
 - `row-reverse`
 - `column`
 - `column-reverse`



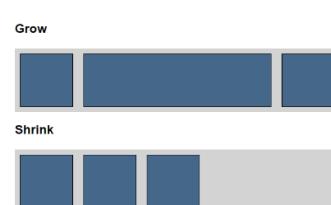
Parent Justify Content

- justify-content**
 - justifies the child items
 - distributes space between them
 - most common:
 - `flex-start`
 - `flex-end`
 - `center`
 - `space-between`
 - `space-around`
 - `space-evenly`



Child Grow and Shrink

- flex-grow** and **flex-shrink** define the proportion, in which a child element grows (or shrinks) with regard to its neighbor children



Child Flex Basis

- flex-basis** is the width (or height) for flex children
 - it defines the default size

```
.item {  
    flex-basis: 200px;  
    background-color: #456789;  
}
```

Parent Display

- `display: flex` switches flexbox on for the child elements

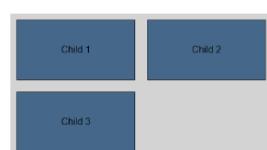
```
div.container {  
    display: flex;  
    width: 360px;  
    background-color: #456789;  
}
```



Parent Flex Wrap

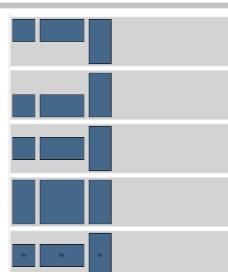
- flex-wrap** determines
 - whether the child items will shrink or wrap
 - the direction of wrap, if applicable
 - `nowrap`
 - `wrap`
 - `wrap-reverse`

```
div.fw {  
    flex-wrap: wrap;  
}
```



Parent Align Items

- align-items** determines how the item are aligned perpendicular to their flow
 - `flex-start`
 - `flex-end`
 - `center`
 - `stretch`
 - `baseline`



Child Order

- order** allows to rearrange the order of children relative to each other
 - default is 0

```
div.egoistic {  
    order: -1;  
}
```



Gradients und Patterns klausurrelevant:

Color Gradients

- CSS allows for smooth transition between two or more colors
 - they are used where an image can be used
- Positioning Color Stops

- without further ado the transition points between colors are equally distributed
- in addition they can be specified
 - by percent
 - by CSS units

```
.multicolor-linear {
  background: linear-gradient(to left,
    orange 26px, red 77%, blue);
}
```

Gradient Hints

- gradient hints specify the transition point between two colors
- default is the middle

```
.linear-without-hint {
  background: linear-gradient(red, blue);
}
.linear-with-hint {
  background: linear-gradient(red, 80%,
    blue);
}
```

Stacked Gradients

- Gradients can be stacked atop of each other
 - stacking order is from top top bottom

```
.stacked-linear {
  background:
    linear-gradient(217deg, rgba(255,0,0,.8), rgba(255,0,0,0) 70.71%),
    linear-gradient(127deg, rgba(0,255,0,.8), rgba(0,255,0,0) 70.71%),
    linear-gradient(336deg, rgba(0,0,255,.8), rgba(0,0,255,0) 70.71%);
}
```

Stacked Radial Gradients

- Like linear gradients, radial gradients can be stacked

```
.stacked-radial {
  background:
    radial-gradient(circle at 50% 0,
      rgba(255,0,0,.5),
      rgba(255,0,0,0) 70.71%),
    radial-gradient(circle at 6.7% 75%,
      rgba(0,0,255,.5),
      rgba(0,0,255,0) 70.71%),
    radial-gradient(circle at 93.3% 75%,
      rgba(0,0,255,.5),
      rgba(0,0,255,0) 70.71%) beige;
  border-radius: 50%;
}
```

Linear Gradients

- Linear gradients have
 - an optional direction
 - preceded by `to`
 - e.g. `to right`
 - default is `to bottom`
 - diagonal by two sides
 - e.g. `to bottom right`
 - can use angles
 - e.g. `90deg`
 - two or more color stops

```
.simple-linear {
  background: linear-gradient(blue, pink);
}

.diagonal-linear {
  background: linear-gradient(to bottom
    right, blue, pink);
}

.angled-linear {
  background: linear-gradient(70deg, blue,
    pink);
}
```

Hard Transitions

- a hard transition between colors occurs, if two adjacent color stops share the same position

```
.striped-linear {
  background: linear-gradient(to bottom
    left, blue 0 50%, pink 50% 100%);
}
.striped-linear-simplified {
  background: linear-gradient(to bottom
    left, blue 50%, pink 50%);
}
```

Overlaying Gradients

- gradients can be overlaid on images, one atop of another
 - from top to bottom
- gradient colors include transparency
- so gradients can partly cover underlying images and other gradients

```
.linear-layered-image {
  background:
    linear-gradient(transparent, 80%, white),
    url("ada-byron.png");
}
```

Radial Gradients

- Radial gradients go from a center point in all directions
- default center point is the center of the element
 - can be changed with `at`

```
.radial-simple {
  background: radial-gradient(red, blue);
}

.radial-offset {
  background:
    radial-gradient(at 0% 30%, red, blue);
```

Conic Gradients

- A conic gradient changes colors **around a center**
- Start is at 0 deg
- This can be used for pie charts

```
.conic-simple {
  background: conic-gradient(red, blue);
}
```

Conic Multicolor

- conic-gradient also allows multiple colors

```
.conic-cone {  
    background: conic-gradient(  
        red,  
        orange,  
        yellow,  
        green,  
        blue,  
        blueviolet  
    );  
}
```

Conic Pie Chart

- pie charts are color gradients, where the stop position of one color is the start position of the next color

```
.conic-piechart {  
    background: conic-gradient(  
        red 60deg,  
        orange 60deg 120deg,  
        yellow 120deg 180deg,  
        green 180deg 240deg,  
        blue 240deg 300deg,  
        blueviolet 300deg  
    );  
}
```

Repeating Gradients

- Repeating gradients repeat the given gradient in the specified direction
- gradient must have a width in CSS units (not percent)
- otherwise the gradient will span the whole element

```
repeating-linear-gradient()  
repeating-radial-gradient()  
repeating-conic-gradient()  
  
.repeating-linear {  
    background: repeating-linear-gradient(  
        to bottom right,  
        white,  
        red 20px,  
        white 40px  
    );  
}
```

CSS Übersicht aller Styling Elemente:

Textausrichtung:

```
text-align: center; /* Zentriert den Text horizontal */
```

Schriftstil:

```
font-size: 14px; /* Setzt die Schriftgröße auf 14 Pixel */
```

```
color: red; /* Setzt die Textfarbe auf rot */
```

```
text-decoration: none; /* Entfernt die Unterstreichung von Links*/
```

```
font-family: Arial, sans-serif; /* Setzt die Schriftfamilie für das gesamte Dokument auf Arial */
```

```
font-style: italic; /* Setzt den Schriftstil auf kursiv */
```

```
font-weight: bold; /* Setzt das Schriftgewicht auf fett */
```

```
text-shadow: 2px 2px 5px gray; /* Fügt einen 2px horizontalen und vertikalen grauen Schatten hinzu */
```

```
line-height: 1.5; /* Setzt die Zeilenhöhe auf das 1,5-fache der Schriftgröße */
```

```
letter-spacing: 2px; /* Setzt den Abstand zwischen den Buchstaben auf 2 Pixel */
```

```
word-spacing: 5px; /* Setzt den Abstand zwischen den Wörtern auf 5 Pixel */
```

Textüberlauf:

Verhindert Zeilenumbrüche und zeigt Auslassungspunkte bei Überlauf

```
white-space: nowrap; /* Kein Zeilenumbruch innerhalb des Elements */
```

```
overflow: hidden; /* Versteckt den Überlaufenden Text */
```

```
text-overflow: ellipsis; /* Zeigt Auslassungspunkte bei Überlauf */
```

```
overflow: auto; /* Fügt Scrollleisten hinzu, wenn der Inhalt größer als der Container ist */
```

Farben und Hintergründe :

```
background-color: #f0f0f0; /* Setzt die Hintergrundfarbe auf ein helles Grau */
```

```
background-image: url('image.jpg'); /* Setzt das Hintergrundbild */
```

```
background-size: cover; /* Deckt das gesamte Element mit dem Bild ab */
```

```
background-repeat: no-repeat; /* Verhindert die Wiederholung des Bildes */
```

```
background-position: center; /* Zentriert das Hintergrundbild im Element */
```

```
background: linear-gradient(to right, red, yellow); /* Farbverlauf von links nach rechts */
```

```
background-color: rgba(0, 0, 0, 0.5); /*Deckkraft Schwarze Farbe mit 50% Deckkraft */
```

```
Background: linear-gradient(45 deg, white 0 50%, black 50% 100%); /*diagonal schwarz oben rechts, weiß unten links*/
```

Box-Modell

Breite und Höhe:

Setzt die Breite und Höhe von <div> Elementen

```
width: 100px; /* Breite auf 100 Pixel setzen */
```

```
height: 200px; /* Höhe auf 200 Pixel setzen */
```

Innenabstand:

Fügt Innenabstand innerhalb der Box hinzu

```
padding: 20px; /* Setzt 20 Pixel Innenabstand an allen Seiten */
```

```
padding-top: 10px; /* Spezifischer Innenabstand oben */
```

```
padding-right: 15px; /* Spezifischer Innenabstand rechts */
```

```
padding-bottom: 20px; /* Spezifischer Innenabstand unten */
```

```
padding-left: 5px; /* Spezifischer Innenabstand links */
```

Außenabstand:

Fügt Außenabstand außerhalb der Box hinzu

```
margin: 10px; /* Setzt 10 Pixel Außenabstand an allen Seiten */
```

```
margin-top: 5px; /* Spezifischer Außenabstand oben */
```

```
margin-right: 20px; /* Spezifischer Außenabstand rechts */
```

```
margin-bottom: 15px; /* Spezifischer Außenabstand unten */
```

```
margin-left: 0px; /* Spezifischer Außenabstand links */
```

Rahmen/Borders:

Fügt einen Rahmen hinzu und setzt Eckenradius

```
border: 2px solid black; /* Setzt einen 2 Pixel breiten schwarzen Rahmen */
```

```
border-radius: 10px; /* Setzt den Radius der Ecken auf 10 Pixel */
```

```
border-top-left-radius: 5px; /* Spezifischer Radius für die obere linke Ecke */
```

```
border-top-right-radius: 5px; /* Spezifischer Radius für die obere rechte Ecke */
```

```
border-bottom-right-radius: 15px; /* Spezifischer Radius für die untere rechte Ecke */
```

```
border-bottom-left-radius: 15px; /* Spezifischer Radius für die untere linke Ecke */
```

```
box-shadow: 5px 5px 10px rgba(0, 0, 0, 0.5); /* Setzt einen 5px horizontalen und vertikalen Schatten mit 50% Deckkraft */
```

```
box-sizing: border-box; /* Bezieht den Innenabstand und den Rahmen in die Gesamtbreite und -höhe ein */
```

Layout und Positionierung:

Sichtbarkeit:

```
display: none; /* Versteckt das Element */  
visibility: visible; /* Macht das Element sichtbar */  
z-index: 10; /* in Layern bedeutet, höherer Z Wert überlagert niedrigere, Setzt den Stapelindex eines Elements */
```

Positionierung:

```
position: absolute; /* Position absolut relativ zum nächstgelegenen positionierten Vorfahren */
```

```
top: 50px; /* 50 Pixel von oben */
```

```
left: 100px; /* 100 Pixel von links */
```

```
max-width: 100%; /* Setzt die maximale Breite auf 100% */
```

```
min-height: 100vh; /* Setzt die minimale Höhe auf die volle Höhe des Ansichtsfensters */
```

```
top: 2vw; /* 2% der Viewport-Breite als Abstand von oben */
```

```
/* Clear */
```

```
clear: left; /* Verhindert, dass das Element neben links fließenden Elementen steht */
```

```
clear: right; /* Verhindert, dass das Element neben rechts fließenden Elementen steht */
```

```
clear: both; /* Verhindert, dass das Element neben fließenden Elementen steht */
```

```
/* Display */
```

```
display: block; /* Element als Block-Element darstellen */
```

```
display: inline; /* Element als Inline-Element darstellen */
```

```
display: inline-block; /* Element als Inline-Block-Element darstellen */
```

```
display: flex; /* Flexbox-Layout aktivieren */
```

```
display: grid; /* Grid-Layout aktivieren */
```

```
/* Position */
```

```
position: static; /* Standardpositionierung */
```

```
position: relative; /* Relativ zur normalen Position */
```

```
position: absolute; /* Absolut relativ zum nächsten positionierten Vorfahren */
```

```
position: fixed; /* Fest relativ zum Viewport */
```

```
position: sticky; /* Klebend relativ zum Scrollen */
```

```
align-items: start; /* Elemente am Anfang der Zelle ausrichten */
```

```
align-items: end; /* Elemente am Ende der Zelle ausrichten */
```

```
align-items: center; /* Elemente in der Mitte der Zelle ausrichten */
```

```
align-items: stretch; /* Elemente über die gesamte Höhe der Zelle strecken */
```

Float (Positionierung):

```
float: left; /* Fließt das Element nach links */
```

```
clear: both; /* Hebt das Fließen auf, sowohl links als auch rechts */
```

```
float: left; /* Element nach links fließen lassen */
```

```
float: right; /* Element nach rechts fließen lassen */
```

Flexbox (Positionierung):

```
display: flex; /* Aktiviert Flexbox */  
justify-content: center; /* Horizontale Ausrichtung der Kinder */  
align-items: center; /* Vertikale Ausrichtung der Kinder */  
flex-direction: column; /* Anordnung der Kinder in Spalten */  
flex-wrap: wrap; /* Erlaubt das Umbrechen der Kinder */  
align-content: center; /* Ausrichtung der Zeilen im Container */  
order: 1; /* Reihenfolge des Elements */  
flex-grow: 1; /* Wachstumsfaktor des Elements */  
flex-shrink: 1; /* Schrumpffaktor des Elements */  
flex-basis: auto; /* Basisgröße des Elements */  
flex: 1; /* Kurzform für flex-grow, flex-shrink, und flex-basis */  
align-self: flex-start; /* Ausrichtung eines einzelnen Elements */  
justify-content: flex-start; /* Kinder am Anfang des Containers ausrichten */  
justify-content: flex-end; /* Kinder am Ende des Containers ausrichten */  
justify-content: center; /* Kinder in der Mitte des Containers ausrichten */  
justify-content: space-between; /* Platz zwischen den Kindern verteilen */  
justify-content: space-around; /* Platz um die Kinder verteilen */  
align-items: flex-start; /* Kinder am Anfang des Containers ausrichten (vertikal) */  
align-items: flex-end; /* Kinder am Ende des Containers ausrichten (vertikal) */  
align-items: center; /* Kinder in der Mitte des Containers ausrichten (vertikal) */  
align-items: baseline; /* Kinder entlang der Textbasislinie ausrichten */  
align-items: stretch; /* Kinder über die gesamte Höhe des Containers strecken */
```

Grid Layout (Positionierung):

Definiert ein Grid-Layout mit drei Spalten

```
display: grid; /* Aktiviert das Grid-Layout */  
grid-template-columns: repeat(3, 1fr); /* Erstellt drei gleich breite Spalten */  
grid-gap: 10px; /* Setzt einen Abstand von 10 Pixeln zwischen den Grid-Zellen */  
justify-items: start; /* Elemente am Start der Zelle ausrichten */  
justify-items: end; /* Elemente am Ende der Zelle ausrichten */  
justify-items: center; /* Elemente in der Mitte der Zelle ausrichten */  
justify-items: stretch; /* Elemente über die gesamte Breite der Zelle strecken */
```

Überspannt mehrere Spalten und Zeilen

```
grid-column: span 2; /* Überspannt zwei Spalten */  
grid-row: span 2; /* Überspannt zwei Zeilen */
```

Animation:

Definiert eine Animation von rot zu gelb

```
@keyframes example {  
  from {background-color: red;} /* Anfangszustand: Hintergrundfarbe rot */  
  to {background-color: yellow;} /* Endzustand: Hintergrundfarbe gelb */  
}
```

Wendet die definierte Animation auf ein Element an */

```
.animation {  
  animation: example 5s infinite; /* Spielt die Animation über 5 Sekunden unendlich oft ab */  
}
```

Übergänge:

Definiert einen Übergang für die Hintergrundfarbe

```
.transition {  
  transition: background-color 0.5s ease; /* Übergang der Hintergrundfarbe über 0,5 Sekunden mit einer ease-Funktion */  
}
```

Ändert die Hintergrundfarbe bei Hover

```
.transition:hover {  
  background-color: blue; /* Setzt die Hintergrundfarbe auf blau bei Hover */  
}
```

Transformationen:

```
transform: rotate(45deg); /* Dreht das Element um 45 Grad */  
transform: scale(1.5); /* Skaliert das Element auf 1,5-fache Größe */  
transform: translate(10px, 20px); /* Verschiebt das Element um 10 Pixel nach rechts und 20 Pixel nach unten */
```

Hover-Effekt:

Ändert die Textfarbe von Links bei Hover

```
a:hover {  
  color: green; /* Setzt die Textfarbe auf grün bei Hover */  
}
```

Nth-Child:

Stilt gerade Zeilen in einer Tabelle */

```
tr:nth-child(even) {  
  background-color: #f2f2f2; /* Setzt die Hintergrundfarbe auf ein helles Grau für gerade Zeilen */  
}
```

Before und After:

Fügt Inhalt vor und nach einem Element hinzu

```
.quote::before {  
    content: ""; /* Fügt ein öffnendes Anführungszeichen vor dem Inhalt ein */  
}  
  
.quote::after {  
    content: ""; /* Fügt ein schließendes Anführungszeichen nach dem Inhalt ein */  
}
```

Listen

```
list-style-type: none; /* Entfernt Aufzählungszeichen */  
display: inline; /* Ordnet die Elemente horizontal an */  
list-style-type: disc; /* Verwendet Aufzählungszeichen für ungeordnete Listen */  
margin-left: 20px; /* Fügt einen linken Außenabstand von 20 Pixeln hinzu */  
list-style-type: decimal; /* Verwendet Dezimalzahlen für geordnete Listen */
```

Responsive design

```
/* Medienabfragen: Definiert das Aussehen für verschiedene Bildschirmgrößen */  
@media (max-width: 600px) {  
    .responsive {  
        flex-direction: column; /* Ändert die Anordnung der Kinder zu einer Spalte bei kleinen Bildschirmen */  
    }  
}
```

Beispiel Buttons:

```
button {  
    background-color: #4CAF50; /* Setzt die Hintergrundfarbe auf grün */  
    color: white; /* Setzt die Textfarbe auf weiß */  
    padding: 15px 32px; /* Fügt einen Innenabstand von 15 Pixeln oben/unten und 32 Pixeln links/rechts hinzu */  
    text-align: center; /* Zentriert den Text horizontal */  
    text-decoration: none; /* Entfernt die Textdekoration */  
    display: inline-block; /* Stellt sicher, dass die Schaltfläche als Inline-Block-Element dargestellt wird */  
    font-size: 16px; /* Setzt die Schriftgröße auf 16 Pixel */  
    margin: 4px 2px; /* Fügt einen Außenabstand von 4 Pixeln oben/unten und 2 Pixeln links/rechts hinzu */  
    cursor: pointer; /* Setzt den Cursor auf einen Zeiger */  
}  
  
button:hover {  
    background-color: #45a049; /* Ändert die Hintergrundfarbe bei Hover */  
}
```

Beispiel Tabellen:

Definiert das Layout und das Aussehen von Tabellen

```
table {  
    width: 100%; /* Setzt die Tabellenbreite auf 100% */  
    border-collapse: collapse; /* Führt benachbarte Rahmen zusammen */  
}  
  
th, td {  
    border: 1px solid black; /* Setzt einen 1 Pixel breiten schwarzen Rahmen */  
    padding: 8px; /* Fügt einen Innenabstand von 8 Pixeln hinzu */  
    text-align: left; /* Richtet den Text linksbündig aus */  
}  
  
th {  
    background-color: #f2f2f2; /* Setzt die Hintergrundfarbe der Tabellenköpfe auf ein helles Grau */  
}
```

Beispiel Listen:

Definiert das Aussehen von Links

```
a {  
    color: blue; /* Setzt die Textfarbe auf blau */  
    text-decoration: underline; /* Unterstreicht den Text */  
}  
  
a:visited {  
    color: purple; /* Setzt die Textfarbe auf lila für besuchte Links */  
}  
  
a:hover {  
    color: red; /* Setzt die Textfarbe auf rot bei Hover */  
}  
  
a:active {  
    color: green; /* Setzt die Textfarbe auf grün für aktive Links */  
}
```

Beispiel Navigation:

```
nav ul {  
    list-style-type: none; /* Entfernt Aufzählungszeichen */  
    padding: 0; /* Entfernt den Innenabstand */  
}  
  
nav li {  
    display: inline; /* Ordnet die Elemente horizontal an */  
    margin-right: 10px; /* Fügt einen rechten Außenabstand von 10 Pixeln hinzu */  
}
```

Java Script:

Allgemeines:

Grundgerüst:

- Variablen Definieren:
 - o var: Funktionaler oder globaler Gültigkeitsbereich, hoisted, mehrfach deklarierbar.
 - o let: Block-skopiert nur in der funktion wo es definiert wird verfügbar
 - o const: kann innerhalb oder außerhalb des Blocks definiert werden.

Klasse:

- Blaupause, die sowohl Struktur als auch die Implementierung (Methoden) enthält.
- Aus einer Klasse können Objekte erstellt werden

```
class Car {  
    constructor(brand, model) { //Konstruktor der Klasse  
        this.brand = brand; // Initialisiert die Eigenschaft 'brand' mit dem übergebenen Wert  
        this.model = model; // Initialisiert die Eigenschaft 'model' mit dem übergebenen Wert  
    }  
  
    // Methode, die die Werte der Eigenschaften 'brand' und 'model' ausgibt  
    displayInfo() {  
        console.log(`Brand: ${this.brand}, Model: ${this.model}`);  
    }  
}  
  
// Erstellen einer Instanz der Klasse Car  
const myCar = new Car('Toyota', 'Corolla');  
  
// Aufrufen der Methode 'displayInfo' auf der Instanz 'myCar'  
myCar.displayInfo(); // Ausgabe: Brand: Toyota, Model: Corolla
```

Objekte:

- Kann direkt erstellt werden oder aber als Instanz einer Klasse
- Funktionen können einfach so geschrieben werden ohne explizit function() zu schreiben

```
const car = {  
    // Eigenschaften des Objekts  
    brand: 'Toyota', // Marke des Autos  
    model: 'Corolla', // Modell des Autos  
    year: 2020, // Baujahr des Autos  
  
    // Methode zur Ausgabe der Eigenschaften des Autos  
    displayInfo: function() {  
        // 'this' bezieht sich auf das aktuelle Objekt 'car'  
        console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);  
    },  
  
    // Methode zur Berechnung des Alters des Autos  
    calculateAge: function(currentYear) {  
        // Berechnung des Alters durch Subtraktion des Baujahrs vom aktuellen Jahr  
        const age = currentYear - this.year;  
        return age; // Rückgabe des berechneten Alters  
    }  
};  
  
// Aufrufen der Methode 'displayInfo' auf dem Objekt 'car'  
car.displayInfo(); // Ausgabe: Brand: Toyota, Model: Corolla, Year: 2020  
  
// Aufrufen der Methode 'calculateAge' auf dem Objekt 'car' mit dem aktuellen Jahr als Argument  
const carAge = car.calculateAge(2023); // Berechnet das Alter des Autos  
console.log(`The car is ${carAge} years old.); // Ausgabe: The car is 3 years old.
```

Globale Funktion:

- Kann überall im Javascript Dokument aufgerufen werden

```
// Globale Funktion zur Ausgabe von Auto-Informationen  
function displayCarInfo(car) {  
    console.log(`Brand: ${car.brand}, Model: ${car.model}`);  
}  
// Objekt, das die Eigenschaften eines Autos enthält  
const car = {  
    brand: 'Toyota',  
    model: 'Corolla'  
};  
// Aufrufen der globalen Funktion mit dem Auto-Objekt als Argument  
displayCarInfo(car); // Ausgabe: Brand: Toyota, Model: Corolla
```

Objekt erstellen mit zweidimensionalem Array:

Erstellen Sie ein Objekt namens **Lotto** mit einer Funktion **init**. **init** soll in das Lotto-Objekt ein zweidimensionales Feld der Größe 7x7 namens **Spielfeld** einbauen und alle Positionen des Spielfelds mit **false** vorbesetzen.

```
var Lotto = { //Objekt namens lotto erstellen
    Spielfeld: [], //leeres Array initialisieren

    Complexity is 3 Everything is cool!
    init(){ //Funktion die Spielfeld erstellt
        for(let i = 0; i < 7; i++){//Erste Schleife erstellen i steht für die y-Achse
            //Hier nochmal neue Zeile im Array initialisiert (also y-Ebene)
            this.Spielfeld[i] = [];
            //Hier werden die Spalten in den Zeilen aufgefüllt (also x-Ebene)
            for(let j = 0; j < 7; j++){ //j steht für die x-Achse
                this.Spielfeld[i][j] = false;
            }
        }
    },
}
```

- Kann mit `Lotto.init();` aufgerufen werden
- Mit `console.log(Lotto.Spielfeld);` wird es ausgegeben
- Attribute des Objekts werden durch , getrennt

Bestimmte Werte auf True setzen innerhalb des Arrays:

Erstellen Sie im Lotto-Objekt eine Funktion **kreuz(x,y)**, die als Parameter die Koordinaten eines Felds (von 0,0 an) enthält und das Spielfeld an dieser Position auf **true** setzt.
Im Fall ungültiger Koordinaten soll die Funktion nichts verändern.

```
kreuz(x,y){ //Überprüfen ob die übergebenen Koordinaten
    //im gültigen Bereich liegen
    if(x>=0 && x < 7 && y >= 0 && y < 7){
        //Setze gegebenes Feld auf True
        this.Spielfeld[x][y]=true;
    }
}
```

- `Lotto.kreuz(1,2);` so könnte man die Funktion aufrufen

Funktion zum überprüfen ob Werte true sind und speichern in Array:

Erstellen Sie im Lotto-Objekt eine Funktion **getippt()**, die ein Array mit allen im Spielfeld angekreuzten Zahlen zurückliefert. Die Zahlen liegen dabei jeweils zwischen 1 und 49, mit 1 am Anfang des Spielfelds und 49 am Ende.

```
getippt(){ //Array erstellen für getippte
    var tipps = []; //Array erstellen für getippte
    for(let i = 0; i<7; i++){ //Schleife für i durchgehen
        for(let j = 0; j<7; i++){ //Schleife für j durchgehen
            //da boolische Werte kann direkt geprüft werden ob beide Werte auf True sind
            if(this.Spielfeld[i][j]){
                //An Array die Zahl anhängen, i*7 da 7 Zeilen und +1 da Array bei 0 anfängt
                tipps.push(i*7+j+1); //push um anzuhängen, in der Klammer wird Wert übergeben
            }
        }
    }
    return tipps;
},
```

Funktion zum überprüfen ob genauer Wert vorhanden ist:

Erstellen Sie im Lotto-Objekt eine Funktion **voll()**, die **true** zurückliefert, wenn 6 Felder angekreuzt sind.

```
voll(){
    let count = 0; //counter erstellen
    for(let i = 0; i < 7; i++){
        for(let j = 0; j < 7; j++){
            if(this.Spielfeld[i][j]){ //alle Indexe durchgehen
                count++; //count inkrementieren
                if(count==6){ //überprüfen ob es 6 angekreuzte sind
                    return true;
                }
            }
        }
    }
    return false; //false, wenn es keine 6 sind
}
```

Beispiel 4 gewinnt:

Zurückgeben welche Farbe Stein in Spalte x,y Zweidimensionales Array immer **Array[Zeile][Spalte]**:

Erstellen Sie im VierGewinnt-Objekt eine Funktion **stein(x,y)**, die zurückliefert, welche Farbe ein Stein in Spalte x und in der Höhe y hat, oder **false**, falls das Feld leer ist.
Positionen außerhalb des Spielfeldes sollen ebenfalls **false** liefern.

```
stein(x, y) { //x steht für horizontale position (7 Spalten breit) y steht für vertikale position (6 Zeilen hoch)
    // Überprüfen, ob x und y innerhalb des Spielfeldes liegen
    if (x < 0 || x >= 7 || y < 0 || y >= 6) {
        return false;
    }
    // Farbe des Steins zurückgeben oder false, falls das Feld leer ist
    return this.Spielfeld[y][x]; //Erster Array Index steht für die Zeile (y) und der zweite steht für die Spalte (x)
}
```

Setzen der Spielfeld Steine:

Erstellen Sie im **VierGewinnt**-Objekt eine Funktion **einwurf(x,farbe)**, die als Parameter die x-Koordinate (Spalte) des Einwurfs und die Farbe des einzuwerfenden Spielsteins enthält.
Ist in dieser Spalte schon ein Stein enthalten, kommt der neue Stein darüber an eine höhere Position.
Ist die Spalte schon voll, soll nichts passieren.

3 Punkt

```
// Funktion zum Einwerfen eines Spielsteins
Complexity is 7 It's time to do something...
einwurf(x, farbe) {
    // Überprüfen, ob die Spalte x innerhalb des gültigen Bereichs liegt
    if (x < 0 || x >= 7) {
        return; // Nichts tun, wenn x außerhalb des Spielfeldes liegt
    }

    // Über die Zeilen der Spalte x iterieren, um die erste leere Position zu finden
    for (var y = 5; y >= 0; y--) { // Von unten nach oben durch die Zeilen iterieren
        if (this.Spielfeld[y][x] === false) { // Überprüfen, ob die Position leer ist
            this.Spielfeld[y][x] = farbe; // Spielstein in der gefundenen Position platzieren
            return; // Beenden der Funktion nach Einfügen des Spielsteins
        }
    }
    // Wenn die Schleife ohne Einfügen eines Spielsteins endet, ist die Spalte voll
},
```

Gewinnermittlung durch testen der horizontalen, vertikalen, diagonalen Richtung:

Zur Gewinnermittlung reicht es, herauszubekommen, ob der zuletzt eingeworfene Stein zu einer 4er-Reihe geführt hat.
Dazu sind Tests in horizontaler, vertikaler und diagonaler Richtung nötig.
Implementieren Sie nur den horizontalen Test, die Funktion **winH(x,y)** im VierGewinnt-Objekt:
Finden Sie heraus, ob der Stein an der gegebenen Position in einer horizontalen Reihe der Mindestlänge 4 liegt und liefern in diesem Fall **true** zurück.

5 Punk

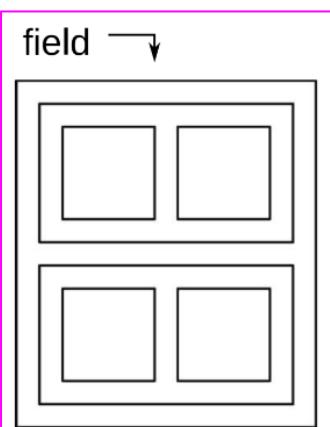
```
winH(x, y) { █
    // Überprüfen, ob die Position innerhalb des Spielfeldes liegt
    if (x < 0 || x >= 7 || y < 0 || y >= 6) {
        return false; // false zurückgeben, wenn die Position außerhalb des Spielfeldes liegt
    }
    // Farbe des Steins an der gegebenen Position
    const farbe = this.Spielfeld[y][x];
    if (farbe === false) {
        return false; // false zurückgeben, wenn das Feld leer ist
    }
    // Zähler für die Länge der gefundenen Kette gleicher Steine
    let count = 0;

    // Überprüfen nach links von der gegebenen Position
    for (let i = x; i >= 0; i--) {
        if (this.Spielfeld[y][i] === farbe) {
            count++;
            if (count >= 4) {
                return true; // true zurückgeben, wenn mindestens 4 gleiche Steine in Folge gefunden wurden
            }
        } else {
            break; // Schleife beenden, wenn ein anderer Stein oder ein leeres Feld gefunden wurde
        }
    }
    // Überprüfen nach rechts von der gegebenen Position
    for (let i = x + 1; i < 7; i++) {
        if (this.Spielfeld[y][i] === farbe) {
            count++;
            if (count >= 4) {
                return true; // true zurückgeben, wenn mindestens 4 gleiche Steine in Folge gefunden wurden
            }
        } else {
            break; // Schleife beenden, wenn ein anderer Stein oder ein leeres Feld gefunden wurde
        }
    }
    // false zurückgeben, wenn keine 4 gleichen Steine in horizontaler Richtung gefunden wurden
    return false;
}
```

Funktion um Divs zu erzeugen:

Schreiben Sie die Funktion **createDivs** innerhalb eines Objekts, die in das umschließende Div mit der id **field** (das äußere in der Zeichnung) die weiteren Divs einbaut.

3 Punkte

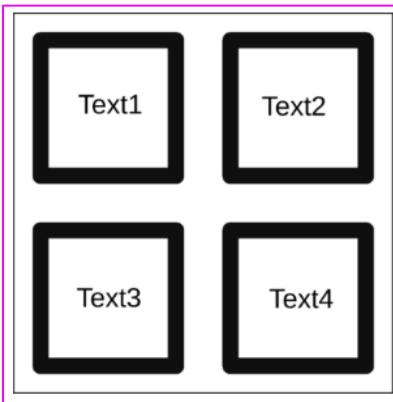


```

// Definieren eines Objekts namens "allFields"
const allFields = {
    // Definition einer Methode namens "createDivs" innerhalb des Objekts
    Complexity is 3 Everything is cool!
    createDivs() {
        // Holen des Elements mit der ID 'field' aus dem DOM und speichern in der Variablen 'field'
        const field = document.getElementById('field');
        // Eine Schleife, die 2-mal durchläuft
        for(let i = 0; i < 2; i++) {
            // Erstellen eines neuen 'div'-Elements und speichern in der Variablen 'innerField'
            const innerField = document.createElement('div');
            // Hinzufügen der Klasse 'innerField' zu dem gerade erstellten 'div'-Element
            innerField.classList.add('innerField');
            // Eine innere Schleife, die ebenfalls 2-mal durchläuft
            for(let j = 0; j < 2; j++) {
                // Erstellen eines weiteren neuen 'div'-Elements und speichern in der Variablen 'innerinnerField'
                const innerinnerField = document.createElement('div');
                // Hinzufügen der Klasse 'innerinnerField' zu dem gerade erstellten 'div'-Element
                innerinnerField.classList.add('innerinnerField');
                // Anhängen des 'innerinnerField'-Elements als Kind von 'innerField'
                innerField.appendChild(innerinnerField);
            }
            // Anhängen des 'innerField'-Elements als Kind von 'field'
            field.appendChild(innerField);
        }
    }
}
// Aufrufen der Methode 'createDivs' des Objekts 'allFields'
allFields.createDivs();

```

Divs erzeugen zweites Beispiel:



```

<div class="taskcontainer">
<div>Task 1</div>
<div>Task 2</div>
<div>Task 3</div>
<div>Task 4</div>
</div>

```

mit dem CSS-Styling für den **taskcontainer**:

```

.taskcontainer {
    width: 200px;
    height: 200px;
    border: 1px solid black;
}

```

```

<script>
    //Eventlistener ruft Funktion createDivs() auf sobald Seite vollständig geladen
    window.addEventListener('load', ()=> page.createDivs());
    const page = { //Objekt erstellen
        createDivs() { //Objekt beinhaltet Funktion createDivs()
            //Konstante Body erstellt welche auf body Element des HTML Dokuments verweist
            const body = document.body;
            //document.createElement('div') erstellt neues div ordnet es Konstante taskcontainer hinzu
            const taskcontainer = document.createElement('div');
            //der Variable taskcontainer wird auch noch die Klasse Taskcontainer hinzugefügt
            taskcontainer.classList.add('taskcontainer');
            //Schleife die 4 Divs erstellt
            for(let i = 0; i < 4; i++){
                const div = document.createElement('div'); //div werden konstante div zugeordnet
                div.innerText= 'Task'+(i+1); //div wird innerer Text hinzugefügt mit entsprechendem Index
                //erstellten Divs werden hier dem Taskcontainer hinzugefügt
                taskcontainer.appendChild(div);
            }
            //Taskcontainer wird body des HTML Documents hinzugefügt
            body.appendChild(taskcontainer);
        }
    }
</script>

```

Wann JS referenzieren?:

Warum benötigen Sie **window.onload()**, wenn Sie das JavaScript im Kopf referenzieren und nicht, wenn Sie es am Ende des Bodys referenzieren?

- Im Kopf (Head): window.onload() ist erforderlich, um sicherzustellen, dass das JavaScript nach dem Laden der gesamten Seite ausgeführt wird, da ansonsten javascript vor dem body lädt und die elemente wo es darauf zugreifen will noch garnicht existieren
- Am Ende des Bodys (Body): window.onload() ist nicht notwendig, da das JavaScript erst ausgeführt wird, wenn die gesamte Seite bereits geladen ist.

Wo JS referenzieren, damit **window.onload()** überflüssig?:

Wie müssen sie Javascript-Code von einem HTML-Dokument referenzieren, um **ohne** den Window **load**-Event auszukommen?

- Wenn man das Skript am Ende des Bodys einfügt. Da das Skript von oben nach unten geladen wird, benötigt man den window.onload() nicht mehr, wenn es nach dem Body eingefügt wird.
- **Erläuterung:** Durch das Platzieren des **<script>**-Tags direkt vor dem schließenden **</body>**-Tag stellt man sicher, dass das gesamte HTML-Dokument und dessen Elemente vollständig geladen sind, bevor das JavaScript ausgeführt wird. Dies macht das **window.onload**-Event überflüssig.

Javascript Fehlercode Analyse Vergleich:

```
01: document.addEventListener('load', () => page.start());
02:
03: const page = {
04:   colors: [ 'red', 'orange', 'yellow', 'green', 'blue' ],
05:   start() {
06:     const body = document.querySelector('body');
07:     for (const color in colors) {
08:       const innerDiv = document.createElement('div');
09:       innerDiv.classList.add('color');
10:       innerDiv.backgroundColor = color;
11:
12:       body.appendChild(innerDiv);
13:       innerDiv.addEventListener('click', evt =
14:         clickHandler(evt));
15:     }
16:     clickHandler(evt) {
17:       alert(`#${evt.srcElement.backgroundColor}`);
18:     },
19:   };

```

```
document.addEventListener('DOMContentLoaded', () => page.start());

const page = {
  colors: [ 'red', 'orange', 'yellow', 'green', 'blue' ],
  Complexity is 3 Everything is cool!
  start() {
    const body = document.querySelector('body');
    for (const color of this.colors) {
      const innerDiv = document.createElement('div');
      innerDiv.classList.add('color');
      innerDiv.style.backgroundColor = color;
      body.appendChild(innerDiv);
      innerDiv.addEventListener('click', (evt) => this.clickHandler(evt));
    }
  },
  clickHandler(evt) {
    alert(`#${evt.target.style.backgroundColor}`);
  }
};
```

01	.window anstatt document
04	eckige Klammern anstatt geschweifte Klammern
07	this.color anstatt color
07	of anstatt in
08	.createDiv gibt es nicht, .createElement muss benutzt werden
09	classes.add Funktion gibt es nicht, Classlist.add muss benutzt werden
10	.backroundColor hat keine Property, man braucht style .['background-color']

- **Zeile 13:** innerDiv.addEventListener('click', evt = clickHandler(evt));
 - o Fehler: Die Syntax für die Event-Listener-Funktion ist falsch. Es sollte nur die Funktion referenziert werden.
 - o Korrektur: innerDiv.addEventListener('click', clickHandler);
- **Zeile 16:** clickHandler(evt) {
 - o Fehler: Die Methode sollte eine Methode im page Objekt sein.
 - o Korrektur: clickHandler: function(evt) {
- **Zeile 17:** alert(\${evt.srcElement.backgroundColor});
 - o Fehler: Der Zugriff auf das Style-Attribut ist falsch. Es sollte evt.target.style.backgroundColor sein.
 - o Korrektur: alert(\${evt.target.style.backgroundColor});
- **Zeile 19:** ;
 - o Fehler: Das Semikolon ist hier nicht notwendig.
 - o Korrektur: }

Liste aller Dom Elemente:

```
// Wählt ein Element anhand seiner ID aus
const elementById = document.getElementById('exampleId');
// Wählt alle Elemente mit einem bestimmten Klassennamen aus
const elementsByClassName = document.getElementsByClassName('exampleClass');
// Wählt alle Elemente mit einem bestimmten Tag-Namen aus
const elementsByTagName = document.getElementsByTagName('div');
// Wählt das erste Element aus, das einem CSS-Selektor entspricht
const elementByQuery = document.querySelector('.exampleClass');
// Wählt alle Elemente aus, die einem CSS-Selektor entsprechen
const elementsByQueryAll = document.querySelectorAll('.exampleClass');
// Erstellen und Modifizieren von DOM-Elementen
// Erstellt ein neues Element
const newElement = document.createElement('div');
// Setzt oder holt den Inhalt eines Elements
newElement.innerHTML = 'Hallo Welt';
// Setzt oder holt Textinhalt eines Elements
newElement.textContent = 'Hallo Welt';
// Fügt ein Kind-Element hinzu
document.body.appendChild(newElement);
// Fügt ein Element vor einem anderen ein
document.body.insertBefore(newElement, existingElement);
// Entfernt ein Kind-Element
document.body.removeChild(existingElement);
// Austausch eines Kindes
document.body.replaceChild(newElement, oldElement);
// Klassen und Attribute
// Fügt eine Klasse zu einem Element hinzu
newElement.classList.add('neueKlasse');
// Entfernt eine Klasse von einem Element
newElement.classList.remove('alteKlasse');
// Überprüft, ob ein Element eine bestimmte Klasse hat
newElement.classList.contains('neueKlasse'); // gibt true oder false zurück
// Setzt ein Attribut für ein Element
newElement.setAttribute('id', 'neueId');
// Holt den Wert eines Attributs
const idValue = newElement.getAttribute('id');
// Entfernt ein Attribut von einem Element
newElement.removeAttribute('id');
// CSS-Stile
// Setzt einen CSS-Stil für ein Element
newElement.style.color = 'blue';
// Ereignisse
// Fügt einem Element einen Ereignis-Listener hinzu
newElement.addEventListener('click', () => {
  console.log('Element wurde geklickt');
});
// Entfernt einen Ereignis-Listener von einem Element
newElement.removeEventListener('click', eventHandler);
```

Liste aller Event Listener:

```
// Einfache Ereignis-Listener
// Klick-Event: wird ausgelöst, wenn ein Element angeklickt wird
element.addEventListener('click', function() {
  console.log('Element wurde geklickt');
});

// Maus-Events
// Mausbewegung: wird ausgelöst, wenn die Maus über ein Element bewegt wird
element.addEventListener('mousemove', function(event) {
  console.log('Maus bewegt');
});

// Mouseover: wird ausgelöst, wenn die Maus über ein Element bewegt wird
element.addEventListener('mouseover', function() {
  console.log('Maus über dem Element');
});

// Mouseout: wird ausgelöst, wenn die Maus das Element verlässt
element.addEventListener('mouseout', function() {
  console.log('Maus hat das Element verlassen');
});

// Tastatur-Events
// Tastendruck: wird ausgelöst, wenn eine Taste gedrückt wird
document.addEventListener('keydown', function(event) {
  console.log(`Taste ${event.key} wurde gedrückt`);
});

// Taste loslassen: wird ausgelöst, wenn eine Taste losgelassen wird
document.addEventListener('keyup', function(event) {
  console.log(`Taste ${event.key} wurde losgelassen`);
});

// Formular-Events
// Fokus: wird ausgelöst, wenn ein Eingabeelement den Fokus erhält
inputElement.addEventListener('focus', function() {
  console.log('Eingabeelement fokussiert');
});

// Blur: wird ausgelöst, wenn ein Eingabeelement den Fokus verliert
inputElement.addEventListener('blur', function() {
  console.log('Eingabeelement verliert den Fokus');
});

// Senden: wird ausgelöst, wenn ein Formular gesendet wird
formElement.addEventListener('submit', function(event) {
  event.preventDefault(); // Verhindert das Standardverhalten des Formulars
  console.log('Formular gesendet');
});

// Lade-Events
// Laden: wird ausgelöst, wenn die Seite vollständig geladen ist
window.addEventListener('load', function() {
  console.log('Seite vollständig geladen');
});

// Vor dem Entladen: wird ausgelöst, bevor die Seite entladen wird
window.addEventListener('beforeunload', function(event) {
  event.returnValue = 'Sind Sie sicher, dass Sie die Seite verlassen möchten?';
});

// Fokus- und Blur-Events
// Fokus auf dem Fenster: wird ausgelöst, wenn das Fenster den Fokus erhält
window.addEventListener('focus', function() {
  console.log('Fenster fokussiert');
});

// Fenster verliert Fokus: wird ausgelöst, wenn das Fenster den Fokus verliert
window.addEventListener('blur', function() {
  console.log('Fenster verliert den Fokus');
});

// Scroll-Event
// Scroll: wird ausgelöst, wenn das Fenster oder ein Element gescrollt wird
window.addEventListener('scroll', function() {
  console.log('Fenster wird gescrollt');
});

// Resize-Event
// Resize: wird ausgelöst, wenn die Größe des Fensters geändert wird
window.addEventListener('resize', function() {
  console.log('Fenstergröße geändert');
});

// Kontextmenü-Event
// Kontextmenü: wird ausgelöst, wenn das Kontextmenü (Rechtsklick) aktiviert wird
element.addEventListener('contextmenu', function(event) {
  event.preventDefault(); // Verhindert das Standard-Kontextmenü
  console.log('Kontextmenü aktiviert');
});
```

Javascript Beispielaufgaben zu Labor 2 (generiert mit Chatgpt):

Aufgabe 1: Spielfeld initialisieren

Frage: Erstellen Sie eine Funktion initializeBoard(), die das Spielfeld als ein zweidimensionales Array von Objekten initialisiert. Jedes Objekt repräsentiert eine Zelle und hat die Attribute hasMine (boolean) und revealed (boolean).

```
function initializeBoard(size) {  
    // Erstelle ein leeres Array für das Spielfeld  
    let board = [];  
  
    // Initialisiere jede Zelle des Spielfeldes  
    for (let y = 0; y < size; y++) {  
        let row = [];  
        for (let x = 0; x < size; x++) {  
            // Jede Zelle ist ein Objekt mit den Attributen 'hasMine' und 'revealed'  
            row.push({  
                hasMine: false, // Initial keine Mine  
                revealed: false // Zelle ist nicht aufgedeckt  
            });  
        }  
        board.push(row);  
    }  
  
    return board; // Rückgabe des initialisierten Spielfeldes  
}
```

Aufgabe 2: Zellen als div-Elemente generieren

Frage: Erstellen Sie eine Funktion generateCells(), die Zellen als div-Elemente erstellt und in ein übergebenes Container-Element einfügt.

```
function generateCells(board, container) {  
    container.innerHTML = ''; // Leere den Container  
  
    // Iteriere über jede Zelle im Spielfeld  
    for (let y = 0; y < board.length; y++) {  
        for (let x = 0; x < board[y].length; x++) {  
            let cell = document.createElement('div'); // Erstelle ein div-Element  
            cell.classList.add('cell'); // Füge die Klasse 'cell' hinzu  
            cell.setAttribute('data-x', x); // Speichere die x-Koordinate  
            cell.setAttribute('data-y', y); // Speichere die y-Koordinate  
            container.appendChild(cell); // Füge die Zelle in den Container ein  
        }  
    }  
}
```

Aufgabe 3: Minen platzieren

Frage: Erstellen Sie eine Funktion placeMines(), die eine bestimmte Anzahl von Minen zufällig im Spielfeld platziert. Dabei soll die Zelle, auf die der Spieler klickt, keine Mine enthalten.

```

function placeMines(board, mineCount, initialX, initialY) {
    let placedMines = 0;
    let size = board.length;

    while (placedMines < mineCount) {
        let x = Math.floor(Math.random() * size);
        let y = Math.floor(Math.random() * size);

        // Überprüfen, ob die zufällig ausgewählte Zelle bereits eine Mine hat oder die Startzelle ist
        if (!board[y][x].hasMine && !(x === initialX && y === initialY)) {
            board[y][x].hasMine = true; // Platziere die Mine
            placedMines++; // Erhöhe den Zähler
        }
    }
}

```

Aufgabe 4: Spielfeld in einem Container anordnen

Frage: Erstellen Sie eine Funktion arrangePlayfield(), die das Spielfeld im Container so anordnet, dass es korrekt angezeigt wird.

```

function arrangePlayfield(container, size) {
    container.style.display = 'grid'; // Verwende CSS Grid Layout
    container.style.gridTemplateColumns = `repeat(${size}, 1fr)`; // Definiere die Anzahl der Spalten
    container.style.gridTemplateRows = `repeat(${size}, 1fr)`; // Definiere die Anzahl der Reihen
}

```

Aufgabe 5: Zellenzustände verwalten

Frage: Erstellen Sie eine Funktion revealCell(), die den Zustand einer Zelle aufgedeckt (revealed) setzt und den aktuellen Zustand zurückgibt.

```

function revealCell(board, x, y) {
    // Überprüfen, ob die Koordinaten innerhalb des Spielfeldes liegen
    if (x < 0 || x >= board.length || y < 0 || y >= board.length) {
        return null;
    }

    // Setze den Zustand der Zelle auf 'revealed'
    board[y][x].revealed = true;

    // Rückgabe des aktuellen Zustandes der Zelle
    return board[y][x];
}

```

Aufgabe 6: Benachbarte Minen zählen

Frage: Erstellen Sie eine Funktion countAdjacentMines(), die für eine gegebene Zelle die Anzahl der benachbarten Minen berechnet und zurückgibt.

```

function countAdjacentMines(board, x, y) {
    let mineCount = 0;
    let size = board.length;

    // Durchlaufe alle benachbarten Zellen
    for (let dx = -1; dx <= 1; dx++) {
        for (let dy = -1; dy <= 1; dy++) {
            let nx = x + dx;
            let ny = y + dy;

            // Überprüfen, ob die Nachbarzelle innerhalb des Spielfeldes liegt und eine Mine hat
            if (nx >= 0 && nx < size && ny >= 0 && ny < size && board[ny][nx].hasMine) {
                mineCount++;
            }
        }
    }

    return mineCount;
}

```

Aufgabe 7: Minenzahlen auf den Zellen anzeigen

Frage: Erstellen Sie eine Funktion `setMineCounts()`, die für jede Zelle des Spielfeldes die Anzahl der benachbarten Minen berechnet und als Attribut speichert.

```
function setMineCounts(board) {  
    let size = board.length;  
  
    for (let y = 0; y < size; y++) {  
        for (let x = 0; x < size; x++) {  
            if (!board[y][x].hasMine) {  
                board[y][x].adjacentMines = countAdjacentMines(board, x, y);  
            }  
        }  
    }  
}
```

Aufgabe 8: Zellen rekursiv aufdecken

Frage: Erstellen Sie eine Funktion `revealAdjacentCells()`, die rekursiv alle benachbarten Zellen aufdeckt, wenn eine Zelle keine benachbarten Minen hat.

```
function revealAdjacentCells(board, x, y) {  
    // Überprüfen, ob die Koordinaten innerhalb des Spielfeldes liegen  
    if (x < 0 || x >= board.length || y < 0 || y >= board.length) {  
        return;  
    }  
    // Überprüfen, ob die Zelle bereits aufgedeckt ist  
    if (board[y][x].revealed) {  
        return;  
    }  
    // Zelle aufdecken  
    board[y][x].revealed = true;  
    // Wenn keine benachbarten Minen vorhanden sind, rekursiv die Nachbarzellen aufdecken  
    if (board[y][x].adjacentMines === 0) {  
        for (let dx = -1; dx <= 1; dx++) {  
            for (let dy = -1; dy <= 1; dy++) {  
                revealAdjacentCells(board, x + dx, y + dy);  
            }  
        }  
    }  
}
```

Aufgabe 9: Spielstatus überprüfen

Frage: Erstellen Sie eine Funktion `checkGameOver()`, die überprüft, ob das Spiel verloren ist (eine Mine aufgedeckt wurde) oder gewonnen ist (alle nicht-minierten Zellen aufgedeckt wurden).

```
function checkGameOver(board) {  
    let size = board.length;  
    let isGameOver = false;  
    let allCellsRevealed = true;  
    for (let y = 0; y < size; y++) {  
        for (let x = 0; x < size; x++) {  
            if (board[y][x].hasMine && board[y][x].revealed) {  
                isGameOver = true;  
            }  
            if (!board[y][x].hasMine && !board[y][x].revealed) {  
                allCellsRevealed = false;  
            }  
        }  
    }  
    if (isGameOver) {  
        return 'lose';  
    } else if (allCellsRevealed) {  
        return 'win';  
    } else {  
        return 'continue';  
    }  
}
```

Aufgabe 10: Markierung von Zellen

Frage: Erstellen Sie eine Funktion `toggleFlag()`, die eine Zelle als verdächtig (mit einer Flagge) markiert oder die Markierung entfernt.

```
function toggleFlag(board, x, y) { █
    // Überprüfen, ob die Koordinaten innerhalb des Spielfeldes liegen
    if (x < 0 || x >= board.length || y < 0 || y >= board.length) {
        return;
    }
    // Überprüfen, ob die Zelle bereits aufgedeckt ist
    if (board[y][x].revealed) {
        return;
    }
    // Markierung umschalten
    board[y][x].flagged = !board[y][x].flagged;
}
```

Aufgabe 11: Event-Handler für Klicks

Frage: Erstellen Sie eine Funktion `setupEventHandlers()`, die Event-Handler für das Klicken und das Kontextmenü (Rechtsklick) der Zellen im Spielfeld einrichtet.

```
function setupEventHandlers(board, container) { █
    // Füge einen Event-Listener für 'click'-Events zum Container hinzu
    container.addEventListener('click', function(event) {
        // Überprüfe, ob das angeklickte Element die Klasse 'cell' enthält
        if (event.target.classList.contains('cell')) {
            // Hole die x-Koordinate aus dem 'data-x' Attribut des angeklickten Elements
            let x = parseInt(event.target.getAttribute('data-x'));
            // Hole die y-Koordinate aus dem 'data-y' Attribut des angeklickten Elements
            let y = parseInt(event.target.getAttribute('data-y'));
            // Rufe die Funktion revealCell() auf, um die Zelle an den Koordinaten (x, y) aufzudecken
            revealCell(board, x, y);
        }
    });

    // Füge einen Event-Listener für 'contextmenu'-Events (Rechtsklick) zum Container hinzu
    container.addEventListener('contextmenu', function(event) {
        // Verhinder das Standard-Kontextmenü, das bei einem Rechtsklick angezeigt wird
        event.preventDefault();
        // Überprüfe, ob das angeklickte Element die Klasse 'cell' enthält
        if (event.target.classList.contains('cell')) {
            // Hole die x-Koordinate aus dem 'data-x' Attribut des angeklickten Elements
            let x = parseInt(event.target.getAttribute('data-x'));
            // Hole die y-Koordinate aus dem 'data-y' Attribut des angeklickten Elements
            let y = parseInt(event.target.getAttribute('data-y'));
            // Rufe die Funktion toggleFlag() auf, um eine Flagge an der Zelle an den Koordinaten (x, y) zu setzen oder zu entfernen
            toggleFlag(board, x, y);
        }
    });
}
```

Aufgabe 12: Anzeige des Spielstatus

Frage: Erstellen Sie eine Funktion `updateGameStatusDisplay()`, die den aktuellen Spielstatus (gewonnen, verloren, läuft) anzeigt.

```
function updateGameStatusDisplay(status) { █
    // Wähle das HTML-Element mit der ID 'game-status' aus
    let statusElement = document.querySelector('#game-status');

    // Überprüfe den Spielstatus und aktualisiere den Textinhalt des Status-Elements entsprechend
    if (status === 'win') {
        // Wenn der Status 'win' ist, setze den Textinhalt auf 'Gewonnen!'
        statusElement.textContent = 'Gewonnen!';
    } else if (status === 'lose') {
        // Wenn der Status 'lose' ist, setze den Textinhalt auf 'Verloren!'
        statusElement.textContent = 'Verloren!';
    } else {
        // Für alle anderen Status (z.B. 'continue'), setze den Textinhalt auf 'Spiel läuft...'
        statusElement.textContent = 'Spiel läuft...';
    }
}
```

Aufgabe 13: Spiel zurücksetzen

Frage: Erstellen Sie eine Funktion resetGame(), die das Spiel zurücksetzt und ein neues Spielfeld erstellt.

```
function resetGame(size, mineCount, container) {  
    // Initialisiere das Spielfeld als zweidimensionales Array von Objekten  
    board = initializeBoard(size);  
    // Erstelle die Zellen als div-Elemente und füge sie in den Container ein  
    generateCells(board, container);  
    // Ordne das Spielfeld im Container an, damit es korrekt angezeigt wird  
    arrangePlayfield(container, size);  
    // Platziere die Minen zufällig auf dem Spielfeld, ohne einen initialen Klick zu berücksichtigen  
    placeMines(board, mineCount, -1, -1);  
    // Berechne und setze die Anzahl der benachbarten Minen für jede Zelle  
    setMineCounts(board);  
    // Aktualisiere die Anzeige des Spielstatus auf 'Spiel läuft...'  
    updateGameStatusDisplay('continue');  
}
```

Aufgabe 14: Zellen aufdecken bei Klick

Frage: Erstellen Sie eine Funktion handleClick(), die bei einem Klick auf eine Zelle diese aufdeckt und die angrenzenden Zellen aufdeckt, wenn sie keine benachbarten Minen hat.

```
function handleClick(x, y) {  
    // Decke die Zelle an den Koordinaten (x, y) auf und speichere den Zustand der Zelle  
    let cellState = revealCell(board, x, y);  
    // Überprüfen, ob die Zelle eine Mine enthält  
    if (cellState && cellState.hasMine) {  
        // Aktualisiere die Anzeige des Spielstatus auf 'Verloren'  
        updateGameStatusDisplay('lose');  
    } else if (cellState && cellState.adjacentMines === 0) {  
        // Wenn die Zelle keine benachbarten Minen hat, decke die benachbarten Zellen auf  
        revealAdjacentCells(board, x, y);  
    }  
    // Überprüfe, ob das Spiel vorbei ist (gewonnen oder verloren)  
    let status = checkGameOver(board);  
    // Aktualisiere die Anzeige des Spielstatus basierend auf dem aktuellen Status  
    updateGameStatusDisplay(status);  
}
```

Aufgabe 15: Benutzeroberfläche aktualisieren

Frage: Erstellen Sie eine Funktion updateUI(), die die Benutzeroberfläche basierend auf dem aktuellen Zustand des Spielfeldes aktualisiert.

```

function updateUI(board, container) {
    // Iteriere über jede Zeile des Spielfeldes
    for (let y = 0; y < board.length; y++) {
        // Iteriere über jede Spalte der aktuellen Zeile
        for (let x = 0; x < board[y].length; x++) {
            // Finde das entsprechende div-Element für die Zelle im Container
            let cell = container.querySelector(`.cell[data-x="${x}"][data-y="${y}"]`);
            // Überprüfe, ob die Zelle aufgedeckt ist
            if (board[y][x].revealed) {
                // Füge die Klasse 'revealed' zur Zelle hinzu
                cell.classList.add('revealed');
                // Überprüfe, ob die Zelle eine Mine enthält
                if (board[y][x].hasMine) {
                    // Füge die Klasse 'mine' zur Zelle hinzu
                    cell.classList.add('mine');
                } else {
                    // Setze den Textinhalt der Zelle auf die Anzahl der benachbarten Minen
                    // Wenn keine benachbarten Minen vorhanden sind, bleibt der Textinhalt leer
                    cell.textContent = board[y][x].adjacentMines > 0 ? board[y][x].adjacentMines : '';
                }
            } else if (board[y][x].flagged) {
                // Wenn die Zelle nicht aufgedeckt ist, aber markiert wurde
                // Füge die Klasse 'flag' zur Zelle hinzu
                cell.classList.add('flag');
            } else {
                // Wenn die Zelle nicht aufgedeckt und nicht markiert ist
                // Entferne die Klasse 'flag' von der Zelle, falls sie vorhanden ist
                cell.classList.remove('flag');
            }
        }
    }
}

```

Aufgabe 17: Zellen markieren und entmarkieren

Frage: Erstellen Sie eine Funktion markCell(), die eine Zelle markiert oder die Markierung entfernt. Die Funktion sollte auch die Anzeige entsprechend aktualisieren.

```

function markCell(board, x, y) {
    // Überprüfen, ob die Zelle bereits aufgedeckt ist
    if (board[y][x].revealed) return; // Wenn die Zelle aufgedeckt ist, tue nichts
    // Umschalten der Markierung (Flagge) der Zelle
    board[y][x].flagged = !board[y][x].flagged; // Wenn die Zelle markiert war, entferne die Markierung und umgekehrt
    // Aktualisiere die Benutzeroberfläche, um den neuen Zustand der Zelle anzuzeigen
    updateUI(board, document.querySelector('.playfield'));
}

```

Aufgabe 18: Sieg überprüfen

Frage: Erstellen Sie eine Funktion checkWin(), die überprüft, ob der Spieler gewonnen hat, indem alle nicht-minierten Zellen aufgedeckt wurden.

```

function checkWin(board) {
    // Iteriere über jede Zeile des Spielfeldes
    for (let y = 0; y < board.length; y++) {
        // Iteriere über jede Spalte der aktuellen Zeile
        for (let x = 0; x < board[y].length; x++) {
            // Überprüfe, ob die aktuelle Zelle keine Mine enthält und noch nicht aufgedeckt ist
            if (!board[y][x].hasMine && !board[y][x].revealed) {
                // Wenn eine nicht-minierte Zelle nicht aufgedeckt ist, ist das Spiel noch nicht gewonnen
                return false;
            }
        }
    }
    // Wenn alle nicht-minierten Zellen aufgedeckt sind, ist das Spiel gewonnen
    return true;
}

```

Aufgabe 19: Spielfeld mit Zellen generieren Divs erstellen und durch Button generieren mit Eventlistener

Erstellen Sie ein HTML-Dokument und ein dazugehöriges JavaScript, das folgendes realisiert:

1. Eine **Schaltfläche** mit der Beschriftung "Generiere Spielfeld".
2. Beim **Klick** auf die Schaltfläche soll ein **Spielfeld (ein div-Container)** mit **mehreren Zellen** (ebenfalls div-Elemente) erzeugt und angezeigt werden.
3. Das Spielfeld soll ein **3x3 Gitter** von Zellen darstellen.

```
<body>
    <!-- Schaltfläche zum Generieren des Spielfelds -->
    <button id="generateButton">Generiere Spielfeld</button>

    <!-- Container für das Spielfeld -->
    <div id="spielfeld"></div>

    <script>
        // Funktion zum Generieren des Spielfelds
        function generateSpielfeld() {
            const spielfeld = document.getElementById('spielfeld');
            spielfeld.innerHTML = ''; // Leert den Container

            // Schleife zum Erstellen von 9 Zellen (3x3 Gitter)
            for (let i = 0; i < 9; i++) {
                const cell = document.createElement('div'); // Erstellt ein div-Element für die Zelle
                cell.classList.add('cell'); // Fügt die Klasse 'cell' hinzu
                cell.textContent = i + 1; // Setzt den Textinhalt der Zelle
                spielfeld.appendChild(cell); // Fügt die Zelle dem Spielfeld-Container hinzu
            }
        }

        // Event Listener für den Klick auf die Schaltfläche
        document.getElementById('generateButton').addEventListener('click', generateSpielfeld);
    </script>
</body>
```

Generiere Spielfeld		
1	2	3
4	5	6
7	8	9

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>Spielfeld Generator</title>
    <style>
        /* Grundstil für den Body */
        body {
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            height: 100vh;
            margin: 0;
        }

        /* Stil für die Schaltfläche */
        #generateButton {
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
        }

        /* Stil für den Spielfeld-Container */
        #spielfeld {
            display: grid;
            grid-template-columns: repeat(3, 100px);
            grid-template-rows: repeat(3, 100px);
            gap: 5px;
            margin-top: 20px;
        }

        /* Stil für die Zellen im Spielfeld */
        .cell {
            width: 100px;
            height: 100px;
            background-color: #f0f0f0;
            display: flex;
            align-items: center;
            justify-content: center;
            font-size: 24px;
            border: 1px solid #000;
        }
    </style>
</head>
```

Aufgaben zu Formular erstellen und mit Get und Post senden:

Formular erstellen und auch erstellen der einzelnen Buttons:

Aufgabe: Erstellen Sie ein Formular mit JavaScript, das zwei Eingabefelder (model und quantity) und einen Submit-Button enthält. Das Formular soll per POST an <https://www.example.com/order.php> gesendet werden. Fügen Sie das Formular in einen div-Container mit der ID formContainer ein. Kommentieren Sie den Code ausführlich.

Model:

Quantity:

BESTELLEN

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8"> <!-- Setzt die Zeichencodierung des Dokuments auf UTF-8 -->
    <title>Formular mit JavaScript</title> <!-- Titel der Webseite -->
</head>
<body>
    <!-- Container für das Formular -->
    <div id="formContainer"></div> <!-- Platzhalter-Div, in das das Formular eingefügt wird -->

    <script>
        // Funktion, um das Formular zu erstellen und in den Container einzufügen
        functioncreateForm() {
            // Erstellt das Formular-Element
            const form = document.createElement('form');
            form.action = "https://www.example.com/order.php"; // Setzt die Action-URL für das Formular
            form.method = "post"; // Setzt die Methode des Formulars auf POST

            // Erstellt das Label und Eingabefeld für "Model"
            const labelModel = document.createElement('label'); // Erstellt ein Label-Element
            labelModel.for = "model"; // Verknüpft das Label mit dem Eingabefeld für Model
            labelModel.textContent = "Model:"; // Setzt den Text des Labels
            form.appendChild(labelModel); // Fügt das Label dem Formular hinzu

            const inputModel = document.createElement('input'); // Erstellt ein Eingabefeld
            inputModel.type = "text"; // Setzt den Typ des Eingabefelds auf Text
            inputModel.id = "model"; // Setzt die ID des Eingabefelds
            inputModel.name = "model"; // Setzt den Namen des Eingabefelds
            form.appendChild(inputModel); // Fügt das Eingabefeld dem Formular hinzu
            form.appendChild(document.createElement('br')); // Fügt einen Zeilenumbruch ein

            // Erstellt das Label und Eingabefeld für "Quantity"
            const labelQuantity = document.createElement('label'); // Erstellt ein Label-Element
            labelQuantity.for = "quantity"; // Verknüpft das Label mit dem Eingabefeld für Quantity
            labelQuantity.textContent = "Quantity:"; // Setzt den Text des Labels
            form.appendChild(labelQuantity); // Fügt das Label dem Formular hinzu

            const inputQuantity = document.createElement('input'); // Erstellt ein Eingabefeld
            inputQuantity.type = "text"; // Setzt den Typ des Eingabefelds auf Text
            inputQuantity.id = "quantity"; // Setzt die ID des Eingabefelds
            inputQuantity.name = "quantity"; // Setzt den Namen des Eingabefelds
            form.appendChild(inputQuantity); // Fügt das Eingabefeld dem Formular hinzu
            form.appendChild(document.createElement('br')); // Fügt einen Zeilenumbruch ein

            // Erstellt den Submit-Button
            const submitButton = document.createElement('input'); // Erstellt ein Eingabeelement
            submitButton.type = "submit"; // Setzt den Typ des Eingabefelds auf Submit
            submitButton.value = "BESTELLEN"; // Setzt den Text des Buttons
            form.appendChild(submitButton); // Fügt den Submit-Button dem Formular hinzu

            // Fügt das Formular in den Container ein
            // Sucht den Container mit der ID 'formContainer' und fügt das Formular hinzu
            document.getElementById('formContainer').appendChild(form);
        }
        // Ruft die Funktion auf, um das Formular zu erstellen, sobald das Dokument geladen ist
        // Wartet, bis das DOM vollständig geladen ist, und führt dann die createForm-Funktion aus
        document.addEventListener('DOMContentLoaded', createForm);
    </script>
</body>
</html>
```

Aufgabe Formular erstellen und senden an Server per Post-Anfrage:

In HTML erstellst du die Grundstruktur deines Formulars. Mit JavaScript fügst du dann Funktionalitäten hinzu, um die Daten per POST oder GET an den Server zu senden. Hier ist eine ausführlich kommentierte Version des HTML- und JavaScript-Codes:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>Beispiel-Formular</title>
</head>
<body>
    <!-- Das Formular mit den Eingabefeldern für Name und Email -->
    <form id="meinFormular">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email"><br>
        <input type="submit" value="Submit">
    </form>

    <!-- JavaScript-Code zum Verarbeiten des Formulars -->
    <script>
        // Event Listener für das 'submit'-Ereignis des Formulars hinzufügen
        Complexity is 4 Everything is cool!
        document.getElementById('meinFormular').addEventListener('submit', function(event) {
            // Verhindert das Standard-Absendeverhalten des Formulars
            event.preventDefault();
            // Sammeln der Formulardaten
            const formData = new FormData(this);
            // Senden der Formulardaten per POST-Anfrage an den Server
            fetch('/script/processform.php', {
                method: 'POST',
                body: formData
            })
            .then(response => response.text()) // Verarbeiten der Antwort des Servers als Text
            .then(result => {
                console.log(result); // Ausgabe der Serverantwort in der Konsole
            })
            .catch(error => {
                console.error('Fehler:', error); // Fehlerbehandlung
            });
        });
    </script>
</body>
</html>
```

Aufgaben zu Labor 3 mit fetchen vom Server (Chatgpt generiert):

Aufgabe 1: Spiel initialisieren

Frage: Erstellen Sie eine Methode initGame(userid, size, mines), die das Spiel durch eine Anfrage an den Server initialisiert und den Token speichert.

```
var remoteLogic = {
  baseURL: 'https://www2.hs-esslingen.de/~melcher/it/minesweeper/?',
  token: '',

  // Methode zur Initialisierung des Spiels
  Complexity is 4 Everything is cool!
  async initGame(userid, size, mines) { █
    // Erstellen der URL für die Anfrage
    const url = `${this.baseURL}request=init&userid=${userid}&size=${size}&mines=${mines}`;

    // Senden der Anfrage an den Server
    const response = await fetch(url);

    // Umwandeln der Antwort in JSON
    const data = await response.json();

    // Überprüfen des Status der Antwort
    if (data.status === 'ok') {
      // Speichern des Tokens, wenn die Initialisierung erfolgreich war
      this.token = data.token;
      return data; // Rückgabe der Daten
    } else {
      // Fehlerbehandlung, wenn die Initialisierung fehlgeschlagen ist
      throw new Error(data.message);
    }
  }
};

// Beispieldruck
remoteLogic.initGame('userid123', 9, 10)
  .then(data => console.log('Game initialized:', data))
  .catch(error => console.error('Error:', error));
```

Aufgabe 2: Zelle aufdecken

Frage: Erstellen Sie eine Methode revealCell(x, y), die eine Anfrage an den Server sendet, um eine Zelle aufzudecken, und das Ergebnis zurückgibt.

```
var remoteLogic = {
  baseURL: 'https://www2.hs-esslingen.de/~melcher/it/minesweeper/?',
  token: '',

  // Methode zum Aufdecken einer Zelle
  async revealCell(x, y) {
    // Erstellen der URL für die Anfrage
    const url = `${this.baseURL}request=sweep&token=${this.token}&x=${x}&y=${y}`;

    // Senden der Anfrage an den Server
    const response = await fetch(url);

    // Rückgabe des JSON-Promises
    return response.json();
  }
};

// Beispieldruck
remoteLogic.revealCell(2, 3)
  .then(result => console.log('Sweep result:', result))
  .catch(error => console.error('Error:', error));
```

Aufgabe 3: Spielstatus abfragen

Frage: Erstellen Sie eine Methode getStatus(), die eine Anfrage an den Server sendet, um den aktuellen Spielstatus zu erhalten.

```

var remoteLogic = {
  baseURL: 'https://www2.hs-esslingen.de/~melcher/it/minesweeper/?',
  token: '',
  // Methode zur Abfrage des Spielstatus
  async getStatus() {
    // Erstellen der URL für die Anfrage
    const url = `${this.baseURL}request=status&token=${this.token}`;
    // Senden der Anfrage an den Server
    const response = await fetch(url);
    // Rückgabe des JSON-Promises
    return response.json();
  }
};

// Beispieldruck
remoteLogic.getStatus()
  .then(status => console.log('Game status:', status))
  .catch(error => console.error('Error:', error));

```

Aufgabe 4: Neues Spiel starten

Frage: Erstellen Sie eine Methode startNewGame(userid, size, mines), die ein neues Spiel startet, indem sie eine Initialisierungsanfrage an den Server sendet.

```

var remoteLogic = {
  baseURL: 'https://www2.hs-esslingen.de/~melcher/it/minesweeper/?',
  token: '',

  // Methode zum Starten eines neuen Spiels
  Complexity is 4 Everything is cool!
  async startNewGame(userid, size, mines) { █
    // Erstellen der URL für die Initialisierungsanfrage
    const url = `${this.baseURL}request=init&userid=${userid}&size=${size}&mines=${mines}`;
    // Senden der Anfrage an den Server
    const response = await fetch(url);
    // Umwandeln der Antwort in JSON
    const data = await response.json();
    // Überprüfen des Status der Antwort
    if (data.status === 'ok') {
      // Speichern des Tokens, wenn die Initialisierung erfolgreich war
      this.token = data.token;
      return data; // Rückgabe der Daten
    } else {
      // Fehlerbehandlung, wenn die Initialisierung fehlgeschlagen ist
      throw new Error(data.message);
    }
  }
};

// Beispieldruck
remoteLogic.startNewGame('userid123', 9, 10)
  .then(data => console.log('New game started:', data))
  .catch(error => console.error('Error:', error));

```

Aufgabe 5: Spiel beenden

Frage: Erstellen Sie eine Methode endGame(), die eine Anfrage an den Server sendet, um das aktuelle Spiel zu beenden.

```

var remoteLogic = {
  baseURL: 'https://www2.hs-esslingen.de/~melcher/it/minesweeper/?',
  token: '',

  // Methode zum Beenden des aktuellen Spiels
  async endGame() {
    // Erstellen der URL für die Anfrage zum Beenden des Spiels
    const url = `${this.baseURL}request=end&token=${this.token}`;

    // Senden der Anfrage an den Server
    const response = await fetch(url);

    // Rückgabe des JSON-Promises
    return response.json();
  }
};

// Beispieldruck
remoteLogic.endGame()
  .then(result => console.log('Game ended:', result))
  .catch(error => console.error('Error:', error));

```

Wissensfragen zum Fetching

Frage 1: Was ist der grundlegende Unterschied zwischen einer lokalen Funktion und einer, die Fetch verwendet?

Antwort:

- Lokale Funktion: Verarbeitet Daten intern ohne externe Kommunikation.
- Fetch-Funktion: Sendet HTTP-Anfragen an einen Server und verarbeitet die Antwort.

Frage 2: Welches JavaScript-Objekt wird verwendet, um HTTP-Anfragen zu senden?

Antwort:

- fetch-API.

Frage 3: Was gibt die fetch-Methode zurück?

Antwort:

- Ein Promise.

Frage 4: Wie wird eine URL für eine Fetch-Anfrage erstellt?

Antwort:

- Durch String-Konkatenation oder Template-Literale.

Frage 5: Welche Methode wird verwendet, um die Antwort einer Fetch-Anfrage in JSON umzuwandeln?

Antwort:

- response.json().

Frage 6: Was passiert, wenn der Fetch-Aufruf fehlschlägt?

Antwort:

- Das Promise wird abgelehnt und ein Fehler wird ausgelöst.

Frage 7: Wie wird ein erfolgreicher Fetch-Aufruf verarbeitet?

Antwort:

- Mit .then().

Frage 8: Wie wird ein fehlgeschlagener Fetch-Aufruf behandelt?

Antwort:

- Mit .catch().

Frage 9: Welche HTTP-Methode wird standardmäßig von Fetch verwendet?

Antwort:

- GET.

Frage 10: Wie können Fetch-Anfragen konfiguriert werden, um andere HTTP-Methoden zu verwenden?

Antwort:

- Durch Übergabe eines Konfigurationsobjekts als zweiten Parameter zu fetch.

Frage 11: Wie wird ein Token in einer Fetch-Anfrage verwendet?

Antwort:

- Als Teil der URL oder im Header.

Frage 12: Was sind die Hauptkomponenten einer Fetch-Anfrage?

Antwort:

- URL, Method, Headers.

Frage 13: Was bedeutet es, wenn die Fetch-Antwort den Statuscode 200 hat?

Antwort:

- Anfrage war erfolgreich.

Frage 14: Wie wird eine Fetch-Anfrage beendet, wenn der Server keine Antwort gibt?

Antwort:

- Mit einem Timeout oder durch Abbrechen des Promises.

Frage 15: Wie wird JSON in ein JavaScript-Objekt umgewandelt?

Antwort:

- Mit JSON.parse().

Lokale zu Fetch-Abfragen (Beispieländerungen)

Frage 16: Wie würde eine lokale Funktion zum Aufdecken einer Zelle in eine Fetch-Funktion geändert werden?

Antwort:

- Lokale Logik entfernen.
- Fetch-Anfrage zum Server hinzufügen.

Frage 17: Wie wird eine Initialisierung des Spiels lokal und mit Fetch durchgeführt?

Antwort:

- Lokal: Array initialisieren.

- Fetch: Anfrage an den Server zum Initialisieren.

Frage 18: Was muss beim Ändern einer lokalen Methode in eine Fetch-Methode berücksichtigt werden?

Antwort:

- Asynchrone Verarbeitung.
- Fehlerbehandlung.

Frage 19: Wie wird der Spielstatus lokal und mit Fetch überprüft?

Antwort:

- Lokal: Array-Daten analysieren.
- Fetch: Anfrage an den Server senden.

Frage 20: Wie kann ein Benutzerereignis (z.B. Klick) lokal und mit Fetch behandelt werden?

Antwort:

- Lokal: Direkte Array-Manipulation.
- Fetch: Senden einer Anfrage und Verarbeiten der Antwort.