**JavaScript**

Code-Blöcke und Erklärung am Beispiel des Projekts «Fotogram»

document.addEventListener("DOMContentLoaded", () => {

const images = document.querySelectorAll(".row1\_pic, .row2\_pic, .row3\_pic, .row4\_pic");

**1. Zeile:**

document.addEventListener("DOMContentLoaded", () => {

**Was passiert hier?**

* **Browser lädt zuerst HTML-Dokument:**  
  Diese Zeile sagt dem Browser: „Führe den nachfolgenden Code erst aus, wenn das gesamte HTML-Dokument vollständig geladen wurde.“
* **Event-Listener DOMContentLoaded:**
  + Wird ausgelöst, sobald die HTML-Struktur des Dokuments geladen und geparst wurde (noch bevor Bilder oder andere externe Ressourcen wie CSS vollständig geladen sind).
  + Der Code im Callback (() => { ... }) wird nur ausgeführt, wenn die Bedingung erfüllt ist.

**Warum wichtig?**

* **Verfügbarkeit des DOM:**  
  Sicherzustellen, dass alle Elemente (wie <img> oder <div>) existieren, bevor das JavaScript versucht, auf sie zuzugreifen.
* **Fehler vermeiden:**  
  Ohne diese Zeile könnten Fehler auftreten, wenn das JavaScript auf Elemente zugreift, die noch nicht im DOM existieren.

**Reihenfolge?**

* **Einstiegsbedingung:**  
  Diese Zeile steht ganz oben, weil sie die Grundlage für den nachfolgenden Code bildet.

**2. Zeile:**

const images = document.querySelectorAll(

".row1\_pic, .row2\_pic, .row3\_pic, .row4\_pic"

);

**Funktion:**

* **Was macht diese Zeile?**
  + Sie erstellt eine Konstante images, die eine Sammlung (NodeList) aller DOM-Elemente speichert, die den CSS-Klassen .row1\_pic, .row2\_pic, .row3\_pic oder .row4\_pic entsprechen.

**Detaillierte Erklärung:**

* **document.querySelectorAll(...):**
  + **DOM-Methode**: Sucht nach allen Elementen, die den angegebenen Selektoren entsprechen.
  + **Selektoren:** Mehrere Klassen (oder andere CSS-Selektoren) können mit Kommas getrennt kombiniert werden.
  + **Rückgabewert:** Gibt eine **NodeList** zurück, eine Art Sammlung von DOM-Elementen.
* **const images:**
  + Speichert diese NodeList unter dem Namen images.
  + **const:** Die Variable ist unveränderlich, aber die Inhalte der NodeList können manipuliert werden.

**Warum wird es verwendet?**

* Um mit allen Bildern interagieren zu können, die die angegebenen CSS-Klassen besitzen.
* Z. B. um:
  + **Events** (wie Klicks) auf die Bilder zu registrieren.
  + **Attribute** (wie src) der Bilder zu ändern.

**Beispiel mit HTML-Struktur:**

<img class="row1\_pic" src="image1.jpg">

<img class="row2\_pic" src="image2.jpg">

<img class="row3\_pic" src="image3.jpg">

<img class="row4\_pic" src="image4.jpg">

Nach dem Aufruf von querySelectorAll enthält images:

* images[0]: Das <img>-Element mit der Klasse row1\_pic.
* images[1]: Das <img>-Element mit der Klasse row2\_pic.
* images[2]: Das <img>-Element mit der Klasse row3\_pic.
* images[3]: Das <img>-Element mit der Klasse row4\_pic.

**Wichtig zu beachten:**

* **NodeList:**
  + Ist ähnlich wie ein Array, unterstützt aber keine Array-Methoden wie .map() oder .reduce().
  + Kann mit einer Schleife oder der Methode .forEach() durchlaufen werden.

**Zusammenfassung:**

1. **document.addEventListener("DOMContentLoaded", ...):**
   * Wartet, bis das DOM vollständig geladen ist.
   * Vermeidet Fehler durch vorzeitigen Zugriff auf DOM-Elemente.
2. **const images = document.querySelectorAll(".row1\_pic, ...");**
   * Sammelt alle <img>-Elemente, die in den angegebenen Klassen definiert sind.
   * Bereitet sie vor, damit du später mit ihnen arbeiten kannst (z. B. Event-Listener hinzufügen oder Eigenschaften ändern).

// Holt die Überlagerung des Dialogs (schwarzer Hintergrund)

const dialogOverlay = document.querySelector(".dialog-overlay");

**Was passiert hier?**

1. **const dialogOverlay:**
   * Erstellt eine Konstante namens dialogOverlay.
   * In dieser wird das erste HTML-Element gespeichert, das der CSS-Klasse .dialog-overlay entspricht.
   * Der Name dialogOverlay ist frei wählbar, sollte aber beschreibend sein.
2. **document.querySelector(".dialog-overlay"):**
   * Diese Methode durchsucht das **DOM** (Document Object Model) nach dem ersten Element, das der CSS-Selektor .dialog-overlay beschreibt:
     + **.** steht für eine Klasse in CSS.
     + **dialog-overlay** ist der Name der Klasse, die im HTML definiert ist.
3. **Gefundene Elemente:**
   * Beispielsweise könnte das folgende HTML-Element gefunden werden:
   * <section class="dialog-overlay hidden">
   * <div class="dialog-content">
   * <!-- Dialog-Inhalt -->
   * </div>
   * </section>
4. **Speichern der Referenz:**
   * Die Referenz zum gefundenen HTML-Element wird in der Konstante dialogOverlay gespeichert.
   * Diese Referenz ermöglicht es, später im Code mit diesem Element zu interagieren (z. B. Klassen hinzufügen, entfernen oder den Stil ändern).

**Warum wichtig?**

* Die dialog-overlay-Klasse repräsentiert den schwarzen Hintergrund, der erscheint, wenn der Dialog geöffnet wird.
* Mit dieser Referenz kannst du:
  1. Den Dialog **anzeigen** oder **verbergen**, indem du Klassen hinzufügst oder entfernst (z. B. hidden oder active).
  2. Den Stil des Overlays ändern (z. B. Farbe, Deckkraft, Position).
  3. Event-Listener hinzufügen (z. B. Klick außerhalb des Dialogs schließt das Fenster).

**Beispielanwendung im Code:**

* Aktivieren der Überlagerung:
* dialogOverlay.classList.add("active"); // Zeigt das Overlay an
* Verbergen der Überlagerung:
* dialogOverlay.classList.remove("active"); // Versteckt das Overlay
* Überprüfen, ob das Overlay sichtbar ist:
* if (dialogOverlay.classList.contains("active")) {
* console.log("Das Dialog-Overlay ist sichtbar.");
* }

**Zusammenfassung:**

* **Was passiert?**
  + Ein HTML-Element mit der Klasse .dialog-overlay wird aus dem DOM ausgewählt und in der Konstanten dialogOverlay gespeichert.
* **Warum?**
  + Damit das schwarze Overlay später dynamisch manipuliert werden kann, z. B. für das Öffnen und Schließen des Dialogs.
* **Wie wird es genutzt?**
  + Über Methoden wie classList.add(), classList.remove(), oder durch Event-Listener, die mit dem Overlay interagieren.

// Holt das Bild im Dialog

const dialogImage = document.querySelector(".dialog-image");

**Was passiert hier?**

1. **document.querySelector(".dialog-image"):**
   * Wählt das erste HTML-Element mit der Klasse .dialog-image aus.
   * In deinem Fall handelt es sich um das <img>-Tag im Dialog-Overlay:
   * <img class="dialog-image" src="" alt="Grosses Bild">
2. **const dialogImage:**
   * Speichert das ausgewählte <img>-Element in der Konstante dialogImage.
   * Dadurch kann dieses Element im weiteren JavaScript-Code angesprochen und manipuliert werden (z. B. Bildquelle ändern).

**Warum wird das gebraucht?**

* Dieses Element wird verwendet, um im Dialog das entsprechende Bild aus den Galerie-Bildern anzuzeigen.
* Über JavaScript kannst du:
  + Die **src-Eigenschaft** ändern, um das passende Bild im Dialog zu laden:
  + dialogImage.src = images[index].src;
  + Stil-Anpassungen (optional) oder Animationen hinzufügen.

**Verbindung zur CSS-Klasse .dialog-image:**

Die CSS-Klasse .dialog-image definiert, wie das Bild im Dialog angezeigt wird.

**CSS-Klasse:**

.dialog-image {

max-width: 90%; /\* Verhindert, dass das Bild zu groß wird \*/

max-height: 80vh; /\* Bild bleibt innerhalb der sichtbaren Höhe \*/

width: auto; /\* Bewahrt das Seitenverhältnis \*/

height: auto; /\* Bewahrt das Seitenverhältnis \*/

object-fit: contain; /\* Passt das Bild in den verfügbaren Raum \*/

}

**Was macht jede Zeile?**

* **max-width: 90%;**
  + Das Bild wird höchstens 90% der Breite des Dialogs einnehmen.
  + Verhindert, dass das Bild den Dialog sprengt.
* **max-height: 80vh;**
  + Das Bild darf höchstens 80% der sichtbaren Höhe des Fensters (Viewport Height) ausfüllen.
  + Sicherstellt, dass das Bild auf kleineren Bildschirmen sichtbar bleibt.
* \*\*width: auto; und **height: auto;**
  + Bewahren das Seitenverhältnis des Bildes.
  + Die Höhe und Breite passen sich dynamisch an, um Verzerrungen zu vermeiden.
* **object-fit: contain;**
  + Das Bild wird so skaliert, dass es vollständig innerhalb des verfügbaren Platzes bleibt.
  + Es wird weder abgeschnitten noch verzerrt.

**Zusammenhang zwischen JS und CSS:**

1. **JavaScript:**
   * Wählt das Element <img class="dialog-image"> und setzt dynamisch die src des Bildes.
   * Dadurch wird das passende Bild im Dialog geladen.
2. **CSS:**
   * Regelt, wie das Bild innerhalb des Dialogs dargestellt wird.
   * Verhindert, dass das Bild zu groß oder verzerrt wird, unabhängig von der ursprünglichen Bildgröße.

**Zusammenfassung:**

* Die **JavaScript-Zeile** sorgt dafür, dass das Dialog-Bild manipuliert werden kann.
* Die **CSS-Klasse .dialog-image** regelt das responsive Verhalten und das visuelle Layout des Bildes im Dialog.

// Holt den Titel im Dialog

const dialogTitle = document.querySelector(".dialog-title");

**Was passiert hier?**

1. **document.querySelector(".dialog-title"):**
   * Wählt das erste HTML-Element mit der Klasse .dialog-title aus.
   * In deinem HTML ist das folgendes Element:
   * <h2 class="dialog-title">project</h2>
2. **const dialogTitle:**
   * Speichert das ausgewählte <h2>-Element in der Konstante dialogTitle.
   * Damit kannst du später den Titel des Projekts dynamisch ändern, wenn der Dialog geöffnet wird.

**Warum wird das gebraucht?**

* Du möchtest den Titel (z. B. den Projektnamen) anzeigen, wenn der Dialog geöffnet wird.
* Über JavaScript kannst du den Textinhalt dynamisch anpassen:
* dialogTitle.textContent = data.title;
* data.title ist der Titel, der aus dem **imageData-Array** geladen wird, basierend auf dem aktuell angeklickten Bild.

**Verbindung zur CSS-Klasse .dialog-title:**

Die CSS-Klasse .dialog-title definiert, wie der Titel im Dialog visuell dargestellt wird.

**CSS-Klasse:**

.dialog-title {

font-family: "Fira Sans", sans-serif; /\* Schriftart des Titels \*/

color: #B48A04; /\* Gelbliche Farbe passend zum Design \*/

font-size: 24px; /\* Schriftgröße des Titels \*/

font-weight: bold; /\* Fettschrift für eine markante Darstellung \*/

margin: 10px 0; /\* Abstand oben und unten \*/

text-align: center; /\* Zentriert den Text \*/

}

**Was macht jede Zeile?**

* **font-family: "Fira Sans", sans-serif;**
  + Definiert die Schriftart des Titels.
  + Die primäre Schriftart ist "Fira Sans", die durch eine generische sans-serif-Schriftart ersetzt wird, falls "Fira Sans" nicht verfügbar ist.
* **color: #B48A04;**
  + Legt die Farbe des Texts fest (goldener Farbton passend zur restlichen Designästhetik).
* **font-size: 24px;**
  + Setzt die Schriftgröße des Titels auf 24 Pixel.
* **font-weight: bold;**
  + Macht den Text fett, um ihn hervorzuheben.
* **margin: 10px 0;**
  + Fügt 10 Pixel Abstand oberhalb und unterhalb des Titels hinzu, um ihn von anderen Elementen zu trennen.
* **text-align: center;**
  + Zentriert den Text horizontal innerhalb seines Containers (in diesem Fall des Dialogs).

**Zusammenhang zwischen JS und CSS:**

1. **JavaScript:**
   * Greift auf das <h2>-Element mit der Klasse .dialog-title zu und ändert den Text dynamisch, um den Projektnamen anzuzeigen.
2. **CSS:**
   * Gestaltet die visuelle Darstellung des Titels, einschließlich Schriftart, Größe, Farbe und Ausrichtung.

**Zusammenfassung:**

* Die **JavaScript-Zeile** ermöglicht das dynamische Anzeigen eines Titels im Dialog.
* Die **CSS-Klasse .dialog-title** sorgt für ein einheitliches und ästhetisches Erscheinungsbild des Titels, das sich nahtlos in das Design des Dialogs einfügt.

// Holt das Datum im Dialog

const dialogDate = document.querySelector(".dialog-date");

**Was passiert hier?**

1. **document.querySelector(".dialog-date"):**
   * Wählt das erste HTML-Element mit der Klasse .dialog-date aus.
   * In deinem HTML ist das folgendes Element:
   * <p class="dialog-date">date</p>
2. **const dialogDate:**
   * Speichert das ausgewählte <p>-Element in der Konstante dialogDate.
   * Damit kannst du später das Datum des Projekts dynamisch ändern, wenn der Dialog geöffnet wird.

**Warum wird das gebraucht?**

* Du möchtest das Datum (z. B. den Zeitraum eines Projekts) im Dialog anzeigen, wenn ein Bild ausgewählt wurde.
* Über JavaScript kannst du den Textinhalt des Datums dynamisch anpassen:
* dialogDate.textContent = data.date;
* data.date ist das Datum, das aus dem **imageData-Array** geladen wird, basierend auf dem aktuell angeklickten Bild.

**Verbindung zur CSS-Klasse .dialog-date:**

Die CSS-Klasse .dialog-date (zusammen mit .dialog-role) definiert, wie das Datum visuell dargestellt wird.

**CSS-Klassen:**

.dialog-date,

.dialog-role {

font-family: "Fira Sans", sans-serif; /\* Einheitliche Schriftart für beide Elemente \*/

color: white; /\* Weißer Text, gut sichtbar auf dem dunklen Hintergrund \*/

font-size: 16px; /\* Einheitliche Schriftgröße für Datum und Rolle \*/

margin: 5px 0; /\* Vertikaler Abstand von 5 Pixeln \*/

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

}

**Was macht jede Zeile?**

* **font-family: "Fira Sans", sans-serif;**
  + Verwendet die Schriftart "Fira Sans" oder als Fallback eine generische sans-serif-Schriftart.
* **color: white;**
  + Stellt sicher, dass der Text weiß ist, damit er auf dem dunklen Hintergrund des Dialogs gut sichtbar bleibt.
* **font-size: 16px;**
  + Setzt die Schriftgröße auf 16 Pixel, eine moderate Größe für ergänzende Informationen.
* **margin: 5px 0;**
  + Fügt einen vertikalen Abstand von 5 Pixeln oberhalb und unterhalb der Elemente hinzu, um sie optisch voneinander abzuheben.
* **text-align: center;**
  + Zentriert den Text innerhalb des Dialogs.

**Zusammenhang zwischen JS und CSS:**

1. **JavaScript:**
   * Greift auf das <p>-Element mit der Klasse .dialog-date zu und ändert den Text dynamisch, um das Projektdatum anzuzeigen.
2. **CSS:**
   * Gestaltet das Datum optisch ansprechend, indem es Schriftart, Farbe, Größe und Ausrichtung definiert.

**Zusammenfassung:**

* Die **JavaScript-Zeile** ermöglicht das dynamische Anzeigen eines Datums im Dialog.
* Die **CSS-Klasse .dialog-date** sorgt für eine klare und gut lesbare Darstellung des Datums, passend zum Design des Dialogs.

// Holt die Rolle (Jobbeschreibung) im Dialog

const dialogRole = document.querySelector(".dialog-role");

**Erklärung der Zeile im JavaScript**

// Holt die Rolle (Jobbeschreibung) im Dialog

const dialogRole = document.querySelector(".dialog-role");

**Was passiert hier?**

1. **document.querySelector(".dialog-role"):**
   * Durchsucht das DOM nach dem ersten Element mit der CSS-Klasse .dialog-role.
   * Das gesuchte Element befindet sich in deinem HTML:
   * <p class="dialog-role">role</p>
2. **const dialogRole:**
   * Speichert das gefundene <p>-Element in der Konstante dialogRole.
   * Dieses Element wird später über JavaScript verwendet, um die Jobbeschreibung (oder Rolle) des ausgewählten Projekts im Dialog dynamisch anzuzeigen.

**Warum wird das gebraucht?**

* Wenn der User auf ein Bild klickt, öffnet sich ein Dialog, der detaillierte Informationen zu dem Projekt anzeigt.
* Diese Zeile stellt sicher, dass die Rolle (z. B. "TV Graphics Coordinator") dynamisch aus den Daten des Projekts geladen und angezeigt wird.
* Die Änderung erfolgt später im Code durch:
* dialogRole.textContent = data.role;
  + data.role stammt aus dem **imageData-Array**, das die Projektdaten enthält.

**Verbindung zur CSS-Klasse .dialog-role:**

Die CSS-Klasse .dialog-role definiert, wie die Rolle im Dialog dargestellt wird.

**CSS-Klasse .dialog-role:**

In deinem CSS ist .dialog-role gemeinsam mit .dialog-date definiert:

.dialog-date,

.dialog-role {

font-family: "Fira Sans", sans-serif; /\* Einheitliche Schriftart für Datum und Rolle \*/

color: white; /\* Weißer Text für Kontrast auf dunklem Hintergrund \*/

font-size: 16px; /\* Gleiche Schriftgröße für beide Elemente \*/

margin: 5px 0; /\* Vertikaler Abstand \*/

text-align: center; /\* Zentriert den Text \*/

}

**Was macht die CSS-Klasse?**

1. **font-family: "Fira Sans", sans-serif;**
   * Verwendet die Schriftart "Fira Sans" (oder eine generische sans-serif-Schriftart).
2. **color: white;**
   * Setzt die Textfarbe auf Weiß, damit sie auf dem dunklen Dialoghintergrund lesbar bleibt.
3. **font-size: 16px;**
   * Definiert eine gut lesbare Schriftgröße von 16 Pixeln.
4. **margin: 5px 0;**
   * Sorgt für etwas Abstand über und unter dem Element, um die Lesbarkeit zu erhöhen.
5. **text-align: center;**
   * Zentriert den Text horizontal innerhalb des Dialogs.

**Zusammenhang zwischen JavaScript und CSS:**

1. **JavaScript:**
   * Greift auf das Element mit der Klasse .dialog-role zu.
   * Setzt den Text dynamisch basierend auf den Projektdaten.
2. **CSS:**
   * Stellt sicher, dass die Rolle (Jobbeschreibung) optisch ansprechend und lesbar dargestellt wird.

**Zusammenfassung:**

* Die JavaScript-Zeile holt das Element mit der Klasse .dialog-role, damit die Jobbeschreibung des ausgewählten Projekts im Dialog angezeigt werden kann.
* Die CSS-Klasse .dialog-role sorgt für eine klare und gut lesbare Darstellung der Jobbeschreibung im Dialog.

// Holt den Schließen-Button im Dialog

const closeButton = document.querySelector(".close-button");

**Erklärung der Zeile im JavaScript**

// Holt den Schließen-Button im Dialog

const closeButton = document.querySelector(".close-button");

**Was passiert hier?**

1. **document.querySelector(".close-button"):**
   * Diese Methode durchsucht das DOM nach dem ersten Element mit der Klasse .close-button.
   * Das entsprechende HTML-Element:
   * <span class="close-button">&times;</span>
   * Es handelt sich um ein <span>-Element, das ein &times;-Symbol (ein "x") enthält, um den Dialog zu schließen.
2. **const closeButton:**
   * Speichert das gefundenen Element in der Konstanten closeButton.
   * Dieses Element wird später verwendet, um ein Klick-Event zu binden, das den Dialog schließt:
   * closeButton.addEventListener("click", closeDialog);

**Warum wird es gebraucht?**

* Der Schließen-Button ermöglicht dem Benutzer, den Dialog einfach und intuitiv zu schließen, wenn dieser geöffnet ist.
* Ohne diesen Button müsste der Benutzer möglicherweise außerhalb des Dialogs klicken, was weniger benutzerfreundlich ist.

**Verbindung zur CSS-Klasse .close-button:**

Die CSS-Klasse .close-button definiert das Design und die Position des Schließen-Buttons.

**CSS-Klasse .close-button:**

.close-button {

width: 30px; /\* Breite des Buttons \*/

height: 30px; /\* Höhe des Buttons \*/

background-color: red; /\* Hintergrundfarbe des Buttons \*/

color: white; /\* Farbe des Symbols "x" \*/

border-radius: 50%; /\* Macht den Button rund \*/

display: flex; /\* Aktiviert Flexbox \*/

justify-content: center; /\* Zentriert das "x" horizontal \*/

align-items: center; /\* Zentriert das "x" vertikal \*/

font-size: 18px; /\* Größe des "x" \*/

font-weight: bold; /\* Fettdruck für das "x" \*/

cursor: pointer; /\* Zeigt eine Hand an, wenn der Button schwebt \*/

position: absolute; /\* Positioniert den Button relativ zum Dialog \*/

top: 10px; /\* Abstand vom oberen Rand \*/

right: 10px; /\* Abstand vom rechten Rand \*/

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3); /\* Leichter Schatten für 3D-Effekt \*/

transition: transform 0.3s ease; /\* Animation beim Hover \*/

}

**Was macht die CSS-Klasse?**

1. **width und height:**
   * Definiert die Größe des Buttons auf 30x30 Pixel, damit er kompakt bleibt.
2. **background-color und color:**
   * Roter Hintergrund mit weißem "x" für eine klare Sichtbarkeit.
3. **border-radius:**
   * Wandelt das Quadrat in einen runden Button.
4. **display, justify-content, align-items:**
   * Aktiviert Flexbox, um das "x" horizontal und vertikal im Button zu zentrieren.
5. **font-size und font-weight:**
   * Macht das "x" groß und fett.
6. **cursor:**
   * Zeigt beim Hover den Zeiger an, um Interaktivität zu signalisieren.
7. **position, top, right:**
   * Positioniert den Button in der oberen rechten Ecke des Dialogs.
8. **box-shadow:**
   * Fügt einen subtilen Schatten hinzu, um den Button hervorzuheben.
9. **transition:**
   * Ermöglicht eine weiche Animation, wenn der Button größer wird (siehe .close-button:hover).

**Zusätzlicher CSS-Hover-Effekt:**

.close-button:hover {

transform: scale(1.1); /\* Vergrößert den Button leicht \*/

}

* Verleiht dem Button einen visuellen Feedback-Effekt, wenn der Benutzer darüber schwebt.

**Zusammenhang zwischen JavaScript und CSS:**

1. **JavaScript:**
   * Identifiziert und speichert das Schließen-Button-Element.
   * Bindet ein Klick-Event an den Button, das den Dialog schließt.
2. **CSS:**
   * Definiert das Aussehen, die Größe und die Position des Buttons.
   * Sorgt für Benutzerfreundlichkeit durch visuelle Rückmeldungen und einen sauberen Stil.

**Zusammenfassung:**

* Die JavaScript-Zeile sucht nach dem Schließen-Button und speichert ihn in closeButton.
* Die CSS-Klasse .close-button macht den Button optisch ansprechend und funktional durch zentriertes "x", Schatten und Hover-Effekt.

// Holt den Button für das vorherige Bild

const prevButton = document.querySelector(".prev-button");

**Erklärung der JavaScript-Zeile:**

// Holt den Button für das vorherige Bild

const prevButton = document.querySelector(".prev-button");

**Was passiert hier?**

1. **document.querySelector(".prev-button"):**
   * Sucht im DOM nach dem ersten HTML-Element mit der Klasse .prev-button.
   * In diesem Fall ist es das <button>-Element:
   * <button class="prev-button">←</button>
   * Dieses Button-Element zeigt den Pfeil "←", der für die Navigation zum vorherigen Bild verwendet wird.
2. **const prevButton:**
   * Speichert das gefundene HTML-Element in der Konstanten prevButton.
   * Diese Konstante wird später in der JavaScript-Funktion genutzt, um ein **Klick-Event** zuzuweisen, das die Bildnavigation nach links (zum vorherigen Bild) ermöglicht:
   * prevButton.addEventListener("click", showPrevImage);

**Verbindung zur HTML-Struktur:**

Im HTML ist der Button so definiert:

<button class="prev-button">←</button>

* **class="prev-button":** Die CSS-Klasse, die verwendet wird, um das Element zu identifizieren.
* **←:** Der Inhalt des Buttons (ein visueller Pfeil), der dem Benutzer anzeigt, dass der Button zum vorherigen Bild navigiert.

**Verbindung zur CSS-Klasse .prev-button:**

Die Klasse .prev-button wird in der CSS-Datei wie folgt gestaltet:

.dialog-navigation button {

background-color: #B48A04; /\* Hintergrundfarbe des Buttons \*/

color: white; /\* Farbe des Pfeils \*/

border: none; /\* Kein Rahmen \*/

border-radius: 50%; /\* Rundes Design \*/

width: 50px; /\* Breite des Buttons \*/

height: 50px; /\* Höhe des Buttons \*/

font-size: 20px; /\* Schriftgröße des Pfeils \*/

cursor: pointer; /\* Zeigt eine Hand beim Hover \*/

display: flex; /\* Aktiviert Flexbox \*/

justify-content: center; /\* Zentriert den Inhalt horizontal \*/

align-items: center; /\* Zentriert den Inhalt vertikal \*/

transition: transform 0.3s ease; /\* Sanfte Animation beim Hover \*/

}

* **Visuelle Eigenschaften:**
  + Der Button hat eine runde Form mit goldener Hintergrundfarbe (#B48A04) und weißem Pfeil.
  + Der Button reagiert auf den Hover-Effekt und vergrößert sich leicht:
  + dialog-navigation button:hover {
  + transform: scale(1.1); /\* Button wird beim Hover größer \*/
  + }
* **Positionierung:**
  + Die Position des Buttons wird in der **dialog-navigation-Klasse** gesteuert:
  + .dialog-navigation {
  + position: absolute;
  + width: 100%; /\* Navigation erstreckt sich über die gesamte Breite \*/
  + top: 50%; /\* Vertikale Mitte des Dialogs \*/
  + display: flex;
  + justify-content: space-between; /\* Buttons bleiben an den Rändern \*/
  + transform: translateY(-50%); /\* Zentriert die Navigation vertikal \*/
  + }

**Zusammenhang zwischen HTML, CSS und JavaScript:**

1. **HTML:**
   * Definiert den Button für die Navigation zum vorherigen Bild und gibt ihm die Klasse .prev-button.
2. **CSS:**
   * Gestaltet den Button visuell und positioniert ihn korrekt innerhalb des Dialogs.
3. **JavaScript:**
   * Sucht das Button-Element (prevButton) im DOM und speichert es.
   * Verwendet die Konstante prevButton, um ein Klick-Event zu binden:
   * prevButton.addEventListener("click", showPrevImage);
   * Bei einem Klick auf den Button wird die Funktion showPrevImage() aufgerufen, die das vorherige Bild anzeigt.

**Zusammenfassung:**

1. **Zweck der Zeile:**
   * Findet und speichert den Button, der für die Navigation zum vorherigen Bild verantwortlich ist.
2. **Verbindung zu anderen Code-Bestandteilen:**
   * Der Button wird durch HTML bereitgestellt, durch CSS optisch gestaltet und durch JavaScript interaktiv gemacht.
3. **Interaktion:**
   * Bei einem Klick auf den Button wird der Dialog aktualisiert, um das vorherige Bild anzuzeigen.

// Holt den Button für das nächste Bild

const nextButton = document.querySelector(".next-button"); same above

// Holt den Zähler, der die aktuelle Bildposition anzeigt (z. B. 1/20)

const imageCounter = document.querySelector(".image-counter");

**Erklärung der Zeile:**

// Holt den Zähler, der die aktuelle Bildposition anzeigt (z. B. 1/20)

const imageCounter = document.querySelector(".image-counter");

**Was passiert hier?**

1. **document.querySelector(".image-counter"):**
   * Sucht im DOM nach dem HTML-Element mit der Klasse .image-counter.
   * In diesem Fall:
   * <span class="image-counter">1/20</span>
   * Das gefundene Element wird in der Konstanten imageCounter gespeichert, um es später im JavaScript manipulieren zu können.
2. **const imageCounter:**
   * Speichert das <span>-Element, welches die aktuelle Bildposition darstellt.
   * Dieses Element wird genutzt, um während der Navigation den angezeigten Text dynamisch zu aktualisieren (z. B. von "1/20" zu "2/20").

**Verbindung zur HTML-Struktur:**

Das Zähler-Element ist so definiert:

<span class="image-counter">1/20</span>

* **class="image-counter":** Die Klasse, die verwendet wird, um das Element zu identifizieren.
* **1/20:** Der anfängliche Text, der anzeigt, dass das erste von insgesamt 20 Bildern aktiv ist.

**Verbindung zur CSS-Klasse .image-counter:**

Die Klasse .image-counter sorgt für die visuelle Darstellung des Zählers:

.image-counter {

font-family: "Fira Sans", sans-serif; /\* Schriftart für konsistentes Design \*/

font-size: 18px; /\* Schriftgröße des Zählers \*/

color: white; /\* Textfarbe \*/

text-align: center; /\* Zentriert den Text \*/

margin-top: 10px; /\* Abstand zur vorherigen Beschreibung (z. B. Rolle) \*/

margin-bottom: 20px; /\* Abstand zum unteren Rand des Dialogs \*/

}

* **Visuelle Eigenschaften:**
  + Die Schriftart und -größe sind konsistent mit dem restlichen Design des Dialogs.
  + Die weiße Farbe sorgt für gute Lesbarkeit auf dem dunklen Hintergrund.
* **Positionierung:**
  + Der Zähler wird unterhalb der Bildbeschreibung angezeigt, mit einem kleinen Abstand nach oben (margin-top) und unten (margin-bottom).

**Verbindung zu JavaScript:**

1. **imageCounter wird dynamisch aktualisiert:**
   * Bei der Navigation zwischen den Bildern wird der Inhalt des Zählers geändert, um die aktuelle Bildposition anzuzeigen.
   * Beispiel:
   * imageCounter.textContent = `${currentIndex + 1}/${images.length}`;
   * Wenn das zweite Bild angezeigt wird, ändert sich der Text von 1/20 zu 2/20.
2. **Nutzung in der updateDialog-Funktion:**
   * Der Zähler wird in der Funktion updateDialog aktualisiert, die immer dann aufgerufen wird, wenn ein neues Bild angezeigt wird:
   * function updateDialog(index) {
   * imageCounter.textContent = `${index + 1}/${images.length}`;
   * }

**Zusammenhang zwischen HTML, CSS und JavaScript:**

1. **HTML:**
   * Stellt das <span>-Element bereit, das den Zähler enthält.
2. **CSS:**
   * Sorgt für die visuelle Gestaltung und Positionierung des Zählers innerhalb des Dialogs.
3. **JavaScript:**
   * Findet das Element (querySelector) und aktualisiert seinen Inhalt dynamisch während der Navigation.

**Zusammenfassung:**

1. **Zweck der Zeile:**
   * Speichert das Zähler-Element, das die aktuelle Bildposition im Dialog anzeigt.
2. **Verbindung zu anderen Code-Elementen:**
   * Der Zähler wird bei jeder Navigation (vor/zurück) aktualisiert.
3. **Interaktion:**
   * Der Benutzer sieht immer die aktuelle Position des Bildes im Verhältnis zur Gesamtanzahl (z. B. "5/20").

// Speichert den Index des aktuell angezeigten Bildes

let currentIndex = 0;

**Erklärung der Zeile:**

// Speichert den Index des aktuell angezeigten Bildes

let currentIndex = 0;

**Was passiert hier?**

1. **Deklaration der Variable:**
   * Die Variable currentIndex wird mit dem Schlüsselwort let deklariert.
   * let ermöglicht es, den Wert der Variable später zu ändern, was hier notwendig ist, da sich der Index bei der Navigation zwischen den Bildern ändert.
2. **Initialisierung:**
   * Der Anfangswert von currentIndex wird auf 0 gesetzt.
   * Das bedeutet, dass zu Beginn das erste Bild in der Liste der Bilder angezeigt wird, da Arrays und NodeLists in JavaScript nullbasiert indiziert sind:
     + images[0] ist das erste Bild.
     + images[1] ist das zweite Bild, usw.

**Warum wird currentIndex benötigt?**

* **Tracking des aktuellen Bildes:**
  + currentIndex speichert, welches Bild gerade aktiv ist.
  + Diese Information wird benötigt, um:
    - Das richtige Bild, den Titel, das Datum und die Rolle im Dialog anzuzeigen.
    - Die Position des Bildzählers (z. B. 1/20) zu aktualisieren.
    - Zu bestimmen, welches Bild als nächstes oder vorheriges angezeigt werden soll.
* **Dynamische Navigation:**
  + currentIndex wird bei der Navigation geändert:
    - Wenn der Benutzer auf "Weiter" klickt, erhöht sich der Index.
    - Wenn der Benutzer auf "Zurück" klickt, verringert sich der Index.

**Wie wird currentIndex verwendet?**

1. **In der Funktion openDialog(index):**
   * Beim Öffnen des Dialogs wird currentIndex auf den Index des ausgewählten Bildes gesetzt:
   * function openDialog(index) {
   * currentIndex = index;
   * updateDialog(currentIndex); // Zeigt das ausgewählte Bild an
   * dialogOverlay.classList.add("active");
   * }
2. **In den Navigationsfunktionen:**
   * **Weiterblättern (Nächstes Bild):**
   * function showNextImage() {
   * currentIndex = (currentIndex + 1) % images.length; // Springt bei Ende zum Anfang
   * updateDialog(currentIndex);
   * }
   * **Zurückblättern (Vorheriges Bild):**
   * function showPrevImage() {
   * currentIndex = (currentIndex - 1 + images.length) % images.length; // Springt bei 0 zum letzten Bild
   * updateDialog(currentIndex);
   * }
3. **In der Funktion updateDialog(index):**
   * Der Index wird genutzt, um das richtige Bild und die zugehörigen Informationen anzuzeigen:
   * function updateDialog(index) {
   * const data = imageData[index];
   * dialogImage.src = images[index].src;
   * dialogTitle.textContent = data.title;
   * dialogDate.textContent = data.date;
   * dialogRole.textContent = data.role;
   * imageCounter.textContent = `${index + 1}/${images.length}`;
   * }

**Warum wird let statt const verwendet?**

* let erlaubt es, den Wert der Variable zu ändern.
* const wäre hier nicht geeignet, da currentIndex bei der Navigation immer wieder aktualisiert werden muss.

**Zusammenhang:**

1. **HTML:**
   * Enthält die Bilder, die durch images referenziert werden.
2. **CSS:**
   * Stellt die visuelle Darstellung des Dialogs und der Navigation bereit.
3. **JavaScript:**
   * currentIndex dient als zentrale Steuerung, um zu bestimmen, welches Bild gerade angezeigt wird.

**Zusammenfassung:**

* **Zweck der Zeile:**
  + Initialisiert den Index des aktuell angezeigten Bildes.
  + Beginnt standardmäßig mit dem ersten Bild (Index 0).
* **Verbindung:**
  + Wird in Funktionen verwendet, um die Navigation zwischen Bildern zu ermöglichen.
* **Dynamische Nutzung:**
  + Ändert sich, wenn der Benutzer durch die Bilder navigiert (vor/zurück).

// Array mit Bilddaten, einschließlich Titel, Datum und Rolle für jedes Bild

const imageData = [

{

title: "Challenge League",

date: "2021-03 - current",

role: "TV Graphics Coordinator",

},

{

title: "Bachelor Thesis",

date: "2023-02 - 2023-08",

role: "Student Business Economics Major Sportmanagement",

},

{

title: "Super League",

date: "2021-03 current",

role: "TV Graphics Coordinator",

},

{

title: "FIFA Youth League",

date: "2023-05",

role: "TV Graphics Coordinator",

},

{

title: "Arosa Snow Football",

date: "2024-01",

role: "TV Graphics Coordinator",

},

{

title: "UEFA Youth League",

date: "2024-04",

role: "TV Graphics Coordinator",

},

{

title: "UBS Kids Festival",

date: "2022",

role: "Event Manager Sportsmarketing",

},

{

title: "International Event",

date: "2024-03 - current",

role: "TV Graphics Coordinator",

},

{

title: "Radsporttage Gippingen",

date: "2024-06",

role: "TV Graphics Coordinator",

},

{

title: "CCB Europe Top 16 Cup Montreux",

date: "2022-02",

role: "TV Graphics Coordinator",

},

{

title: "Champions Hockey League",

date: "2024-09",

role: "TV Graphics Coordinator",

},

{

title: "Season Opening CHL",

date: "2024-09",

role: "TV Graphics Coordinator",

},

{

title: "Fritschi Umzug",

date: "2024-02",

role: "TV Graphics Coordinator",

},

{ title: "Monstercorso", date: "2024-02", role: "TV Graphics Coordinator" },

{

title: "Goldentracks Awards",

date: "2021-10",

role: "TV Graphics Coordinator",

},

{ title: "LVW", date: "2024-06", role: "Graphic Operator" },

{

title: "UBS Kids Cup",

date: "2022",

role: "Event Manager Sportsmarketing",

},

{

title: "Weltklasse Zürich",

date: "2022",

role: "Event Manager Sportsmarketing",

},

{

title: "NEP Workstation",

date: "2021-03 current",

role: "TV Graphics Coordinator",

},

{

title: "SFL Workstation",

date: "2019-09 - 2023-01",

role: "Speaker Supervisor",

},

];

**Erklärung der Codezeile**

// Array mit Bilddaten, einschließlich Titel, Datum und Rolle für jedes Bild

const imageData = [

{

title: "Challenge League",

date: "2021-03 - current",

role: "TV Graphics Coordinator",

},

...

];

**Was passiert hier?**

1. **Deklaration der Konstante:**
   * const imageData: Eine Konstante wird deklariert, die ein **Array** speichert.
   * Dieses Array enthält mehrere **Objekte** mit spezifischen Informationen zu jedem Bild.
2. **Inhalt des Arrays:**
   * Jedes Element des Arrays ist ein **Objekt**.
   * Die Objekte enthalten drei Schlüssel (Eigenschaften):
     + title: Der Titel oder Name des Projekts.
     + date: Das Datum oder der Zeitraum, in dem das Projekt stattfand.
     + role: Die Rolle oder Position, die der Benutzer im Projekt hatte.
3. **Datenstruktur:**
   * Das Array ermöglicht eine strukturierte Speicherung aller Bilddaten.
   * Beispiel eines Objekts:
   * {
   * title: "Challenge League",
   * date: "2021-03 - current",
   * role: "TV Graphics Coordinator"
   * }
   * Jedes Objekt beschreibt ein Bild, das in der Galerie angezeigt wird.
4. **Beziehung zu den Bildern:**
   * Jedes Objekt im Array imageData korrespondiert mit einem Bild, das mit querySelectorAll ausgewählt wurde.
   * Die Daten im Array werden verwendet, um die Dialogelemente (title, date, role) dynamisch mit den richtigen Informationen zu füllen.

**Warum wird es benötigt?**

* **Zentrale Datenhaltung:**
  + Alle Informationen zu den Bildern (Titel, Datum, Rolle) werden an einem Ort gespeichert.
  + Dadurch wird der Code übersichtlich und wartungsfreundlich.
* **Dynamische Verwendung:**
  + Die Daten können mit der Funktion updateDialog(index) abgerufen und angezeigt werden.
  + Dadurch wird die Anzeige automatisch aktualisiert, wenn ein anderes Bild ausgewählt wird.
* **Flexibilität:**
  + Änderungen an einem Bild (z. B. neuer Titel oder geändertes Datum) müssen nur im imageData-Array vorgenommen werden.
  + Diese Änderungen werden dann automatisch überall angewendet, wo die Daten genutzt werden.

**Wie wird imageData verwendet?**

1. **Füllen des Dialogs:**
   * Wenn ein Bild angeklickt wird, nutzt die Funktion updateDialog(index) das Array, um die Daten des entsprechenden Bildes zu laden:
   * function updateDialog(index) {
   * const data = imageData[index]; // Holt das Objekt an der Position "index"
   * dialogTitle.textContent = data.title; // Setzt den Titel
   * dialogDate.textContent = data.date; // Setzt das Datum
   * dialogRole.textContent = data.role; // Setzt die Rolle
   * }
2. **Verknüpfung mit den Bildern:**
   * Das Array imageData und die NodeList images sind synchron:
     + images[0] korrespondiert mit imageData[0].
     + images[1] korrespondiert mit imageData[1], usw.
   * Diese Synchronität ermöglicht eine einfache Navigation und Anzeige.

**Warum wird const verwendet?**

* Das Array imageData wird als Konstante deklariert, weil:
  + Die Referenz auf das Array nicht geändert wird.
  + Der Inhalt des Arrays (die Objekte und deren Eigenschaften) kann jedoch bei Bedarf bearbeitet werden.

**Zusammenfassung der Funktion:**

1. **Speichert strukturierte Bildinformationen:**
   * Jedes Bild hat Titel, Datum und Rolle.
2. **Ermöglicht dynamische Anzeige:**
   * Die Daten werden in den Dialog eingefügt, wenn ein Bild ausgewählt wird.
3. **Verbindung zu anderen Variablen:**
   * imageData korrespondiert direkt mit den Bildern (images).
4. **Flexibel und wartungsfreundlich:**
   * Änderungen an den Bildinformationen sind zentral und einfach vorzunehmen.

**Zusammenhang mit HTML:**

* Jedes Objekt im Array repräsentiert ein Bild in der Galerie.
* Beispiel für ein Bild in HTML:
* <img class="row1\_pic" src="./img/challenge\_league.png" alt="Challenge League">
* Die Daten für dieses Bild stammen aus:
* {
* title: "Challenge League",
* date: "2021-03 - current",
* role: "TV Graphics Coordinator"
* }

// Funktion zum Aktualisieren des Dialoginhalts basierend auf dem aktuellen Bildindex

function updateDialog(index) {

const data = imageData[index];

dialogImage.src = images[index].src;

dialogImage.style.width = "auto"; // Passt die Breite automatisch an

dialogImage.style.height = "auto"; // Passt die Höhe automatisch an

dialogTitle.textContent = data.title;

dialogDate.textContent = data.date;

dialogRole.textContent = data.role;

imageCounter.textContent = `${index + 1}/${images.length}`;

}

// Öffnet den Dialog und zeigt das ausgewählte Bild

function openDialog(index) {

currentIndex = index; // Speichert den aktuellen Index

updateDialog(currentIndex); // Aktualisiert den Dialoginhalt

dialogOverlay.classList.add("active"); // Zeigt die Dialogüberlagerung an

}

// Schließt den Dialog

function closeDialog() {

dialogOverlay.classList.remove("active"); // Versteckt die Dialogüberlagerung

}

// Zeigt das nächste Bild im Dialog

function showNextImage() {

currentIndex = (currentIndex + 1) % images.length; // Erhöht den Index und geht bei Ende zurück zu 0

updateDialog(currentIndex); // Aktualisiert den Dialoginhalt

}

// Zeigt das vorherige Bild im Dialog

function showPrevImage() {

currentIndex = (currentIndex - 1 + images.length) % images.length; // Reduziert den Index und springt bei 0 zurück ans Ende

updateDialog(currentIndex); // Aktualisiert den Dialoginhalt

}

// Fügt Klick-Event-Listener zu jedem Bild hinzu, um den Dialog zu öffnen

images.forEach((image, index) => {

image.addEventListener("click", () => openDialog(index)); // Öffnet den Dialog für das angeklickte Bild

});

// Fügt ein Klick-Event zum Schließen-Button hinzu

closeButton.addEventListener("click", closeDialog);

// Schließt den Dialog, wenn außerhalb des Dialogs geklickt wird

dialogOverlay.addEventListener("click", (e) => {

if (e.target === dialogOverlay) closeDialog(); // Schließt nur, wenn auf die Überlagerung geklickt wird

});

// Fügt ein Klick-Event zum vorherigen Button hinzu

prevButton.addEventListener("click", showPrevImage);

// Fügt ein Klick-Event zum nächsten Button hinzu

nextButton.addEventListener("click", showNextImage);

});