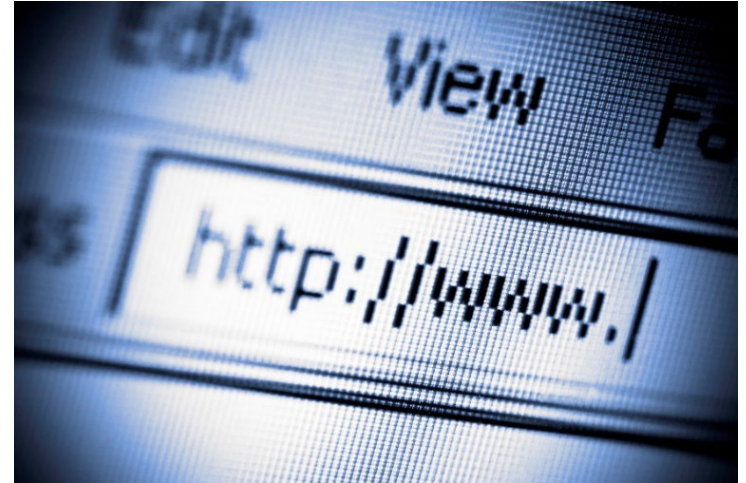


Web-Technologien

Geolocation API und responsive Layouts

- Geolocation API: Abfragen der Geräteposition
- W3.CSS: CSS-Library für responsive Layouts

Letzte Aktualisierung: 28. Oktober 2022



Ziele

- Verstehen der Konzepte und Unterschiede von HTML5 APIs und Libraries.
- Prinzip der Standortbestimmung als Beispiel einer HTML5 API verstehen und Anwenden können.
- Funktionalität des Geolocation HTML5 APIs verstehen und anwenden können.
- Wissen wie man eine CSS-Library einbindet.
- Kennenlernen und Anwenden der W3.CSS Library.
- Wissen was ein Responsive Layout ist und wie man ein solches Layout mittels W3.CSS implementiert.

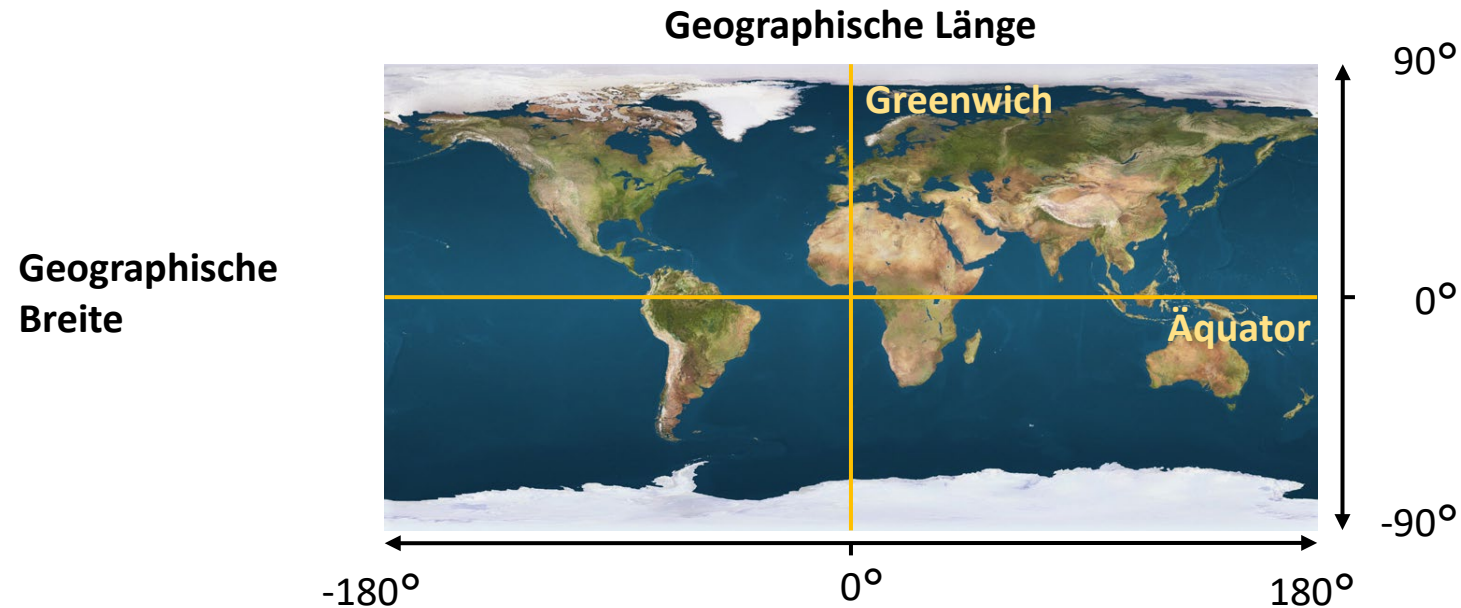
APIs und Libraries

- **APIs** (Programmierschnittstellen) ermöglichen Zugriff auf im Webbrowser eingebaute Funktionalität.
 - Bieten Schnittstellen zum Browser an und erweitern die Möglichkeiten von puren JavaScript.
 - **Beispiele:** Canvas, Geolocation, WebStorage, WebGL, WebSockets, ...
- **Libraries** bieten Funktionalität, welche auf Basis-Funktionalität aufbaut.
 - Alles, was in Libraries möglich ist, ist mit purem HTML/CSS/JavaScript möglich.
 - Typischerweise bieten Sie eine spezifischere Schnittstelle an.
 - **Beispiele:** jQuery, Bootstrap, Vue.js, w3.css, Angular, react ...

Geolocation

- Abfrage des aktuellen Standorts.
- Verwendung (unter Anderem) in:
 - Routenplanern / Navigation
 - Buchungsdienste (Uber, AirBnB, ...)
 - Wetterbericht
 - Sport (Tracking der gelaufenen Strecke)
 - ...
- Typischerweise werden Werte zu einem Webserver übertragen.

Geographische Koordinaten



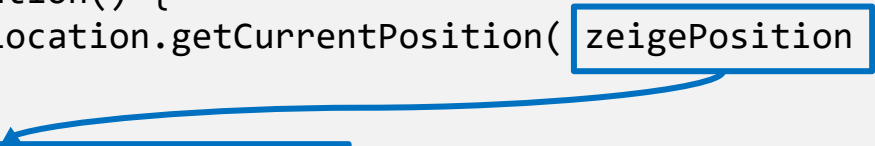
Quelle: https://en.wikipedia.org/wiki/Geographic_coordinate_system

Positionsabfrage mittels Browser-API

```
<body>
  <p id="output"></p>
  <script>
    function getLocation() {
      navigator.geolocation.getCurrentPosition( zeigePosition );
    }

    function zeigePosition(position) {
      document.getElementById("output").innerHTML =
        = "Breite: " + position.coords.latitude + " / " +
        + "Länge: " + position.coords.longitude;
    }

    // Ausgabe
    getLocation();
  </script>
</body>
```

A blue arrow originates from the 'zeigePosition' argument in the 'getCurrentPosition' call within the 'getLocation' function and points to the 'zeigePosition' function definition below it.

Anzeige auf Karte (1)

- Mittels JavaScript-Library oder via Bild-URL.

Beispiel: Google Static Map API (Bild-URL):

```

```



Anzeige auf Karte (2)

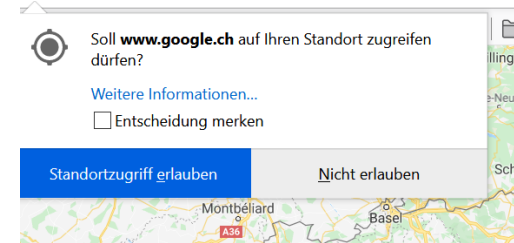
- Ab einer gewissen Anzahl Abfragen **kostenpflichtig**.
- Abrechnung mittels API-KEY.
- Verschiedene Anbieter, z.B.: Google, TomTom, Mapbox, HERE, MapFit, Leaflet, etc.

Achtung: API-KEY (hier: YOUR_KEY) sichtbar in URL integriert. **Absicherung notwendig (abhängig vom Anbieter)**.

- Mittels Beschränkung auf bestimmte URLs oder IP-Adressen (nicht 100%-sicher).
- Besser (aber komplexer): API-KEY nur serverseitig verwenden.

Fehlerbehandlung

- Typische Probleme:
 - Benutzer verweigert Zugriff.
 - Position nicht verfügbar (z.B., kein Satellit).
- Fehlerbehandlungsfunktion als zweiten Parameter der Positionsabfrage:



```
navigator.geolocation.getCurrentPosition(zeigePosition, handleFehler);
```

```
function handleFehler(error) {  
    let text;  
    if      (error.code == error.PERMISSION_DENIED) { text = "Benutzer verweigert Zugriff"; }  
    else if (error.code == error.TIMEOUT)          { text = "Wartezeit überschritten";      }  
    else if (error.code == error.POSITION_UNAVAILABLE) { text = "Position nicht verfügbar"; }  
    else                                           { text = "Unbekannter Fehler";      }  
    document.getElementById("output").innerHTML = text;  
}
```

Kontinuierliches Tracking

Kontinuierliches Tracking der Geräteposition:

Starten: `watchId = watchPosition(showPosition);`

- Funktion `showPosition` wird für jede Positionsänderung aufgerufen.
- Rückgabewert `watchId` identifiziert diese Überwachung.

Stoppen: `clearWatch(watchId)`

- Stoppt die Meldung von Positionsänderungen.
- Benötigt Parameter `watchId`.

Kontinuierliches Tracking (Beispiel)

```
<body>
  <p id="output"></p>
  <button onclick="start()">Start</button>
  <button onclick="stop()">Stop</button>

  <script>
    function start() {
      watchId = navigator.geolocation.watchPosition(showPosition);
    }
    function stop() {
      navigator.geolocation.clearWatch(watchId);
    }
    function showPosition(position) {
      document.getElementById("output").innerHTML =
        = "Breite: " + position.coords.latitude +
        + " / Länge: " + position.coords.longitude;
    }
  </script>
</body>
```

Einführung in W3.CSS

- CSS-Library um "mobile-first"-Projekte im Web zu realisieren.
- Frontend-Library, d.h., gedacht für den Browser und nicht Webserver.
- **Ziel:** Schnellere und einfachere Frontend-Entwicklung.
- Default auf sinnvolle Standardwerte gesetzt und viele "ready-to-use" Elemente wie Buttons, Formulare, Tabellen, Images, Navigation, etc.
- Download: https://www.w3schools.com/w3css/w3css_downloads.asp

Libraries (Funktionsbibliotheken)

- Libraries bieten Funktionalität, welche auf bestehender JavaScript- und Webbrowser-Funktionalität aufbaut.
- Alles, was eine Library bietet, könnte man auch selbst implementieren.
- Die CSS-Library "W3.CSS" (aber auch andere CSS-Libraries) bietet eine Anzahl vorgefertigter Styles an.
- Die Programmierer kann eine beliebige Anzahl dieser Styles in seinem Projekt verwenden, daher die Bezeichnung (Library): Es ist oft mehr da als man braucht.
- CSS-Libraries werden wie ein normales externes Stylesheet eingebunden.

Beispiel W3.CSS:

```
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
```

W3.CSS-Klassen (kleiner Auszug)

w3-container

Setzt gleiche Ränder, Füllung, vert. und horizl. Ausrichtung, Schriften und Farben.

w3-panel

Hervorgehobener Container.

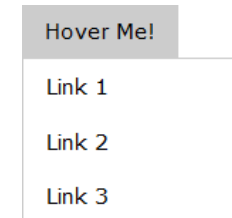
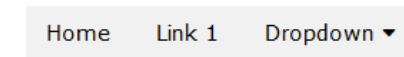
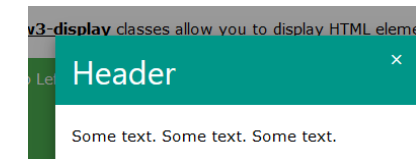
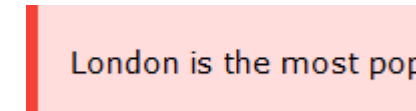
w3-modal

Zeigt Container modal an.

w3-bar

Horizontaler Balken, z.B., Navigationsleiste.

w3-dropdown – Container, welcher als Dropdown fungiert (z.B. in Navigationsleiste).



Verwendung von W3.CSS-Klassen

```
<html><head>
  <title>W3.CSS - Hello world</title>
  <link rel="stylesheet" href="w3.css">
</head>
<body>
  <header class="w3-container w3-black">
    <nav class="w3-bar w3-black">
      <button class="w3-button">News</button>
      <button class="w3-button">About</button>
    </nav>
  </header>
  <main class="w3-container w3-sand">
    <section class="w3-panel w3-blue-grey">Hello World</section>
  </main>
  <footer class="w3-container w3-black">More infos...</footer>
</body></html>
```

Schwarzer Container

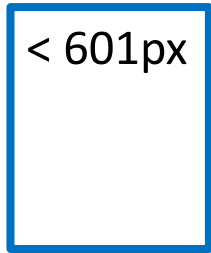
Navigationsleiste

Button

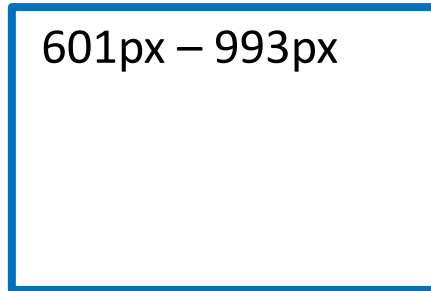
Panel

Konzept: Responsive Layout

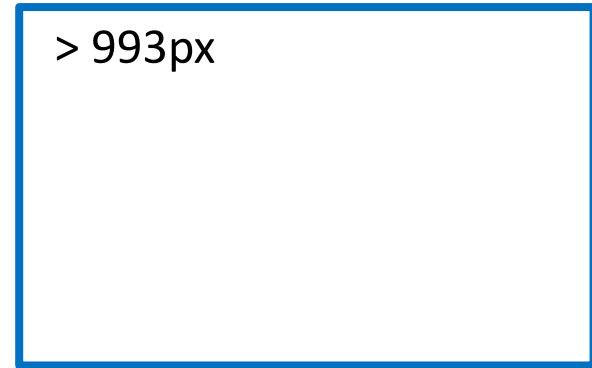
Ein **einzelnes** Layout, welches sich der **Breite** des Ausgabegerätes anpasst:



small
(phone)



medium
(tablet, small laptop)



large
(laptop, desktop)

Umgang mit mehreren Displaygrößen

Responsives Grid mit zwölf Spalten. Ausrichtung aller Elemente nach diesen Spalten:

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

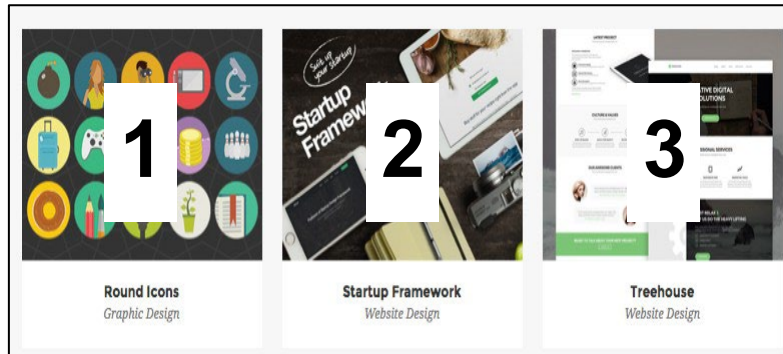
Vorgehen:

1. Definition von Zeilen-Containern (Klasse: `w3-row`).
2. Definition der Spaltenanzahl pro Displaygröße und pro Element (small: `s1-s12`, medium: `m1-m12`, large: `l1-l12`).

Ist für eine Displaygröße keine Spaltenanzahl definiert: Übernimmt Spaltenanzahl aus nächst kleinerer Displaygröße oder 12, falls nichts gesetzt.

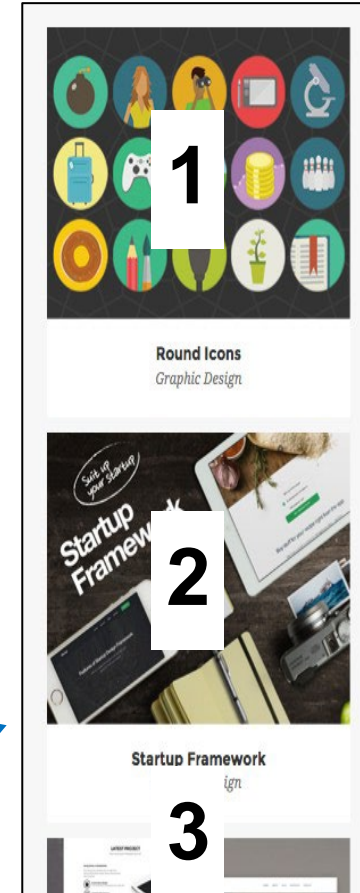
Responsive Grid

Spalten werden in der mobilen Darstellung gestapelt.



Desktop

Mobile



Beispiel eines Responsive Grids

```
<section class="w3-container">
  <article class="w3-row">
    <div class="w3-col m4 w3-blue">
      Box 1
    </div>
    <div class="w3-col m4 w3-red">
      Box 2
    </div>
    <div class="w3-col m4 w3-green">
      Box 3
    </div>
  </article>
</section>
```

Medium und Large:



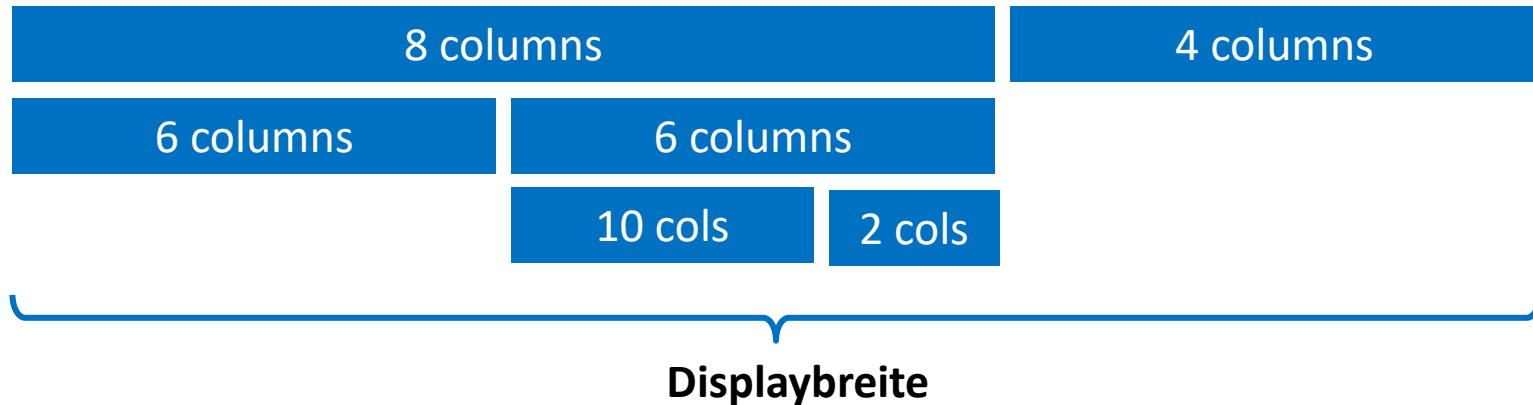
Small:



Verschachteln von Zeilen

- Zum Erzielen bestimmter Layouts ist es u.U. notwendig Zeilen in einander zu verschachteln, d.h, Zeilen weiter zu unterteilen.
- Dabei wird wiederum ein 12 spaltiges Layout verwendet.

Beispiel:



Beispiel mit verschachtelten Zeilen

```
<div class="w3-container w3-grey">
  <div class="w3-row">
    <div class="w3-container w3-col m6">
      <div class="w3-row">
        <div class="w3-col l6">
          <input class="w3-input" placeholder="First">
        </div>
        <div class="w3-col l6">
          <input class="w3-input" placeholder="Last">
        </div>
        <div class="w3-col">
          <input class="w3-input" placeholder="Email">
        </div>
      </div>
    </div>
    <div class="w3-container w3-col m6">
      <textarea class="w3-input" placeholder="Message">
    </div>
  </div>
</div>
```

Small

First

Last

Email

Message

Medium

First

Last

Email

Message

12 Spalten

Large

First

Last

Email

Message

6 Spalten

6 Spalten

12 Spalten

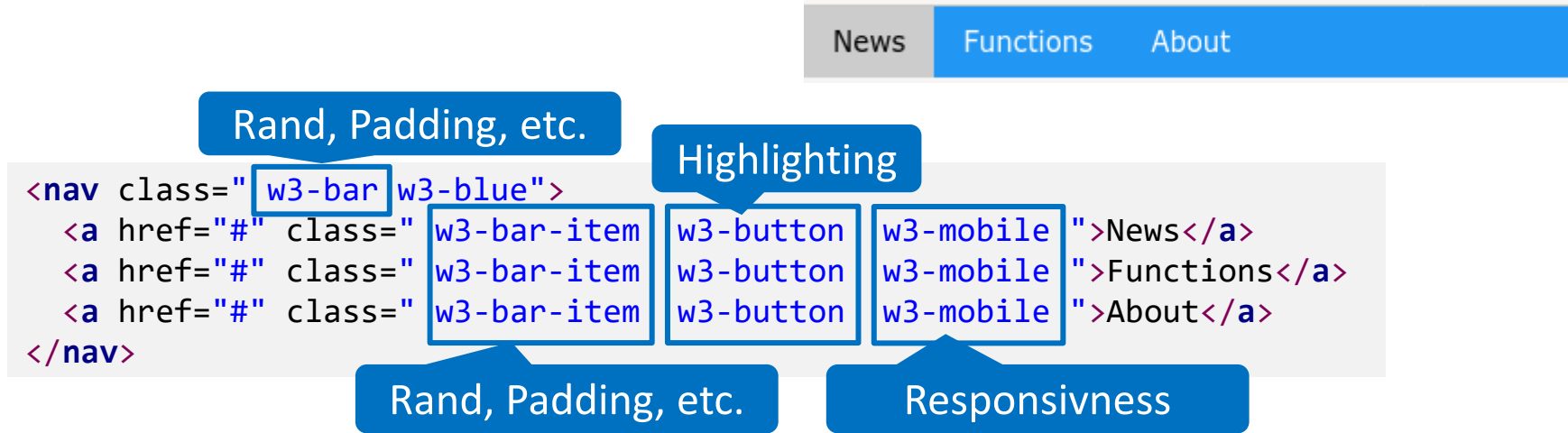
Responsive Navigationsleisten (1)

- Navigation ist ein Kernelement einer Webapplikation.
- Wichtigste Navigationspunkte müssen immer einfach erreichbar sein.
- Responsive Navigation mit W3.CSS ist ohne Responsive-Grid realisierbar.

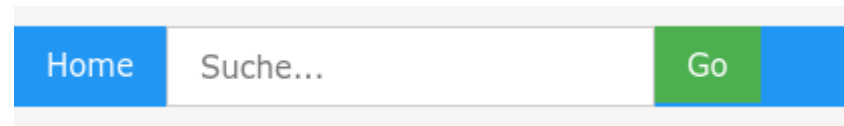
Wichtige Klassen für Navigationsleisten:

- w3-bar: Container um HTML-Element horizontal anzuzeigen (die Leiste).
- w3-bar-item: Element einer Navigationsleiste.
- w3-button: Button-Style ideal für Schaltflächen einer Navigationsleiste.
- w3-mobile: Schaltfläche wird zum Blockelement bei 100% Grösse auf small-Displays.

Responsive Navigationsleisten (2)

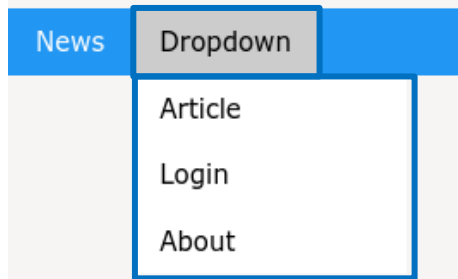


- Auch andere Elemente wie z.B. Buttons, Icons oder Inputs (für Suchleisten) können verwendet werden:



Navigationslisten mit Untermenüs deklarativ erstellen

w3-dropdown-hover

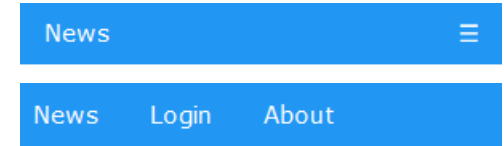


w3-dropdown-content

- Falls Zeiger über Container `w3-dropdown-hover` liegt, wird Container `w3-dropdown-content` sichtbar.
- Container `w3-dropdown-hover` muss eine Fläche haben (wie hier z.B. ein Button).

```
<nav class="w3-bar w3-blue">
  <a href="#" class="w3-bar-item w3-button">News</a>
  <div class="w3-dropdown-hover">
    <button class="w3-button">Dropdown</button>
    <div class="w3-dropdown-content w3-bar-block">
      <a href="#" class="w3-bar-item w3-button">Login</a>
      <a href="#" class="w3-bar-item w3-button">About</a>
    </div>
  </div>
</div>
```


"Hamburger"-Button (1)



- Verstecke auf kleinen Displaygrößen Einträge hinter einem Button.
- **Idee:** Zwei Menüs mit unterschiedlichen Sichtbarkeiten per Displaygröße:
 - small: Einträge in der Leiste teilweise versteckt, Button wird sichtbar.
 - medium, large: Alle Einträge sichtbar, Button versteckt.
- `w3-hide-<size>` versteckt Elemente jeweils auf der entsprechenden Größe.
Beispiel: Element mit Klasse `w3-hide-small` wird auf small-Geräten **nicht** gezeigt.

```
<div class="w3-bar w3-blue">
  <a href="#" class="w3-bar-item w3-button">News</a>
  <a href="#" class="w3-bar-item w3-button w3-hide-small">Login</a>
  <a href="#" class="w3-bar-item w3-button w3-hide-small">About</a>
  <button class="w3-bar-item w3-button w3-right w3-hide-large w3-hide-medium"
    onclick=...>⌵</button>
</div>
```

small: nicht sichtbar

large / medium: nicht sichtbar

Zeige Menü (nächstes Slide)

Zeichencode für ≡ (math. Operation)

"Hamburger"-Button (2)

- Klasse w3-hide versteckt ein Element auf **jeder Grösse**.
- Verstecktes Menü enthält die restlichen Einträge:

```
<div id="sub" class="w3-bar-block w3-blue w3-hide">  
  <a href="#" class="w3-bar-item w3-button">Login</a>  
  <a href="#" class="w3-bar-item w3-button">About</a>  
</div>
```

nicht sichtbar

- und wird sichtbar beim Klick auf den Button:

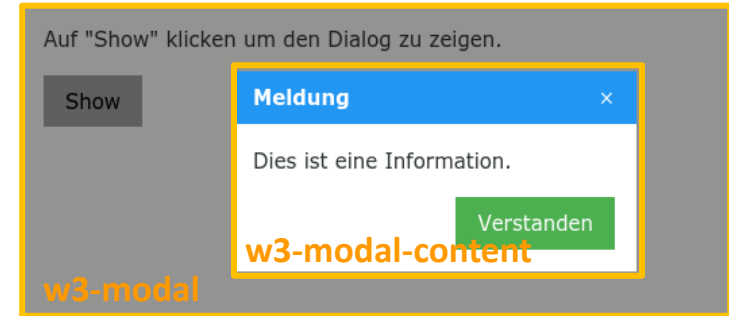
Klasse hinzufügen, falls nicht vorhanden, sonst entfernen

```
onclick="document.getElementById('sub').classList.toggle('w3-hide');"
```



Modale Dialoge für zwingende Benutzerinteraktion

Modal: Applikation bleibt eingefroren bis Dialog geschlossen wird ("Dialog-Modus").



Dialog definieren:

- ID ist beliebig muss aber **unique** sein.
- Default: "hidden".

ID zum Referenzieren

```
<div id="box" class="w3-modal">
  <div class="w3-modal-content" style="max-width:400px">
    <!-- hier steht der Inhalt des Dialogs -->
  </div>
</div>
```

Maximale Breite (Responsive)

Maximale Breite (Responsive)

Dialog öffnen:

```
document.getElementById('box').classList.add('w3-show');
```

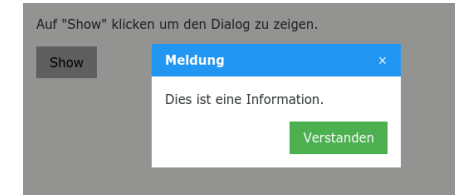
Dialog schliessen:

```
document.getElementById('box').classList.remove('w3-show');
```

Referenzieren via ID

Klasse w3-show übersteuert "versteckt"

Beispiel eines modalen Dialogs



Inhalt wie ein normales HTML-Dokument mit responsiven Techniken.

1. Header erstellen (analog zur Navigationsleiste):

```
<header class="w3-bar w3-blue">  
  <span class="w3-bar-item"><b>Meldung</b></span>  
  <span class="w3-bar-item w3-button w3-right" onclick="...">&times;</span>  
</header>
```

Abbruch und Schliessen

2. Information und Button(s) hinzufügen (auch Inputfelder):

```
<main class="w3-container">  
  <section class="w3-section">Dies ist eine Information.</section>  
  <section class="w3-row w3-section">  
    <button class="w3-button w3-right m3 w3-green" onclick="...">OK</button>  
  </section>  
</main>
```

OK und Schliessen

W3.CSS anpassen

- Libraries nicht ändern: Eigenes Stylesheet erstellen und dazu laden.

Vorgehen:

1. custom.css erzeugen.
2. Eigene CSS-Definitionen in custom.css schreiben.
3. Nach w3.css laden:

```
<link rel="stylesheet" href="w3.css">  
<link rel="stylesheet" href="custom.css">
```

Zusammenfassung

- Unterschied zwischen HTML5 API (neue Funktionalität via Browser Funktionen) und Libraries (auf bestehender Funktionalität aufbauend).
- HTML5 API: Geolocation zur Positionsbestimmung des Gerätes (einmalig und kontinuierlich) .
- Kombination von Geolocation und Karten API.
- Konzept des Responsive-Layout.
- Library: W3.CSS zur einfachen Erstellung eines Responsive Layout.
- Responsive Techniken mit W3.CSS: Responsive-Grid, Navigationsleisten und modale Dialoge.