

## HANDS-ON (1): NPM

```
npm --version
```

```
(sudo) npm install npm@latest -g
```

```
npm view npm
```

## Übung: NPM-Abhängigkeiten – Wir programmieren einen ASCII-Generator

CMD + UP + P → Konsole

```
mkdir my-ascii
```

```
code my-ascii
```

- In Visual Studio Code das Terminal öffnen
- `npm search ascii-text`
- Listet eine ganze Reihe von Vorschläge auf, darunter auch das Modul „ascii-text-generator“ welches wir nachfolgend verwenden wollen
- Nochmal in den Browser und auf [npmjs.org](https://npmjs.org), dort nach „ascii-text“ suchen ...
- `npm install ascii-text-generator`
- `ls -al`
- Im Verzeichnis „node\_modules“ werden die heruntergeladenene Packages vorgehalten (dieses Verzeichnis daher besser nicht ins Quellcode-Repository mit einchecken, sollte man ignorieren)
- Die Datei „**package-lock.json**“ wurde von npm angelegt und wird von npm für die Synchronisation genutzt, alle Abhängigkeiten die im node\_modules-Verzeichniss stehen, sind in dieser Datei gelistet! Diese Datei wird nicht vom Entwickler gepflegt!
- Probieren wir es aus:  
`cat package-lock.json`  
`npm list`
- Anzeigen, welche Versionen von unserer Abhängigkeit bestehen und downgraden:  
`npm view ascii-text-generator versions`  
`npm install ascii-text-generator@1.0.2`  
`cat package-lock.json`  
`npm list`  
`npm install ascii-text-genertor@latest`
- Unterschied zur „**package.json**“:  
Diese Datei wird manuell vom Benutzer angelegt und widerspiegelt die Projektkonfiguration wie sich der Entwickler diese sich vorstellt, die Quelle der Wahrheit. Was hier steht ist Gesetz. Vergleichbar mit pom.xml aus Maven.

Um die Datei anzulegen:

```
npm init
```

- Über Code in der package.json die Versions-Nummer ändern
- `npm list` zeigt, dass wir immer noch auf Version 1.0.5 stehen, auf 1.0.2 ändern (Caret stehen lassen)
- `npm install` sorgt dafür, dass die package.json auf das Projekt angewendet wird
- Wenn `npm list` immer noch die Versions-Nummer 1.0.5 anzeigt, dann liegt das am Caret
- Neue Datei `my.js` anlegen
- Folgenden Code eintippen:

```
const gen = require('ascii-text-generator')
```

Bemerke: Visual Studio bietet Codevervollständig an!

`require` → Um das heruntergeladene Modul nachladen zu können wird ein sog. Modul-System benötigt. Node lehnt sich an CommonJS an und stellt für das Laden von Modules den `require`-Befehl zur Verfügung.

Mit EcmaScript 2015 wurde aber offizielles Modul-System (ES Modules) im Sprachkern verabschiedet, dieses verwendet nicht den `require`-Befehl sondern arbeitet ähnlich wie in Java mit `imports`. Kommen wir später noch darauf zurück.

```
console.log( gen('scc rules', '2'))
```

- `node my` → läuft

→ Folien **JavaScript Versionen** zeigen erklären

→ Folien wie in der 1-Std-Session durchgehen bis zum Thema Web Components

## HANDS-ON (2): WEB COMPONENTS

- ZIP herunterladen und in neuen **Unter-Ordner** /flags entpacken
- code .



- **flags.html** im Chrome öffnen:  
Plugin „Watch in Chrome“ installieren, dann kann man mit SHIFT + CMD + P  
Und „watch in chrome“ Chrome direkt aus der IDE heraus starten...

Erklären:

**assets**-Ordner → Leere Grafik als Platzhalter (blank.gif)

flags.png Alle Flaggen in einer zusammenfassenden Grafik

flags.css CSS-Klassen um eine einzelne Landesflagge zu identifizieren und diesen Teilbereich aus dem Gesamt-PNG zu extrahieren

flags.html → nutzt Image-Tag und CSS-Klassen

Doof:

- Viel Boilerplate-Code (wen interessiert die Platzhalter-Grafik?)
- Es ist kompliziert das Land festzulegen
- Und überhaupt passt doch das <img>-Tag so garnicht zu Flaggen

Besser wäre:

(Auf Flipchart oder Tafel schreiben)

```
<flag-icon country="de" />
```

Und das hierfür realisieren wir jetzt eine Web Component, genauer ein sog. Custom Element.

Folgende Schritte sind zu tun:

- Neue Datei flags.js anlegen und Klasse „Flag“ anlegen:

```
class FlagIcon extends HTMLElement
```

- Wir legen initial einen Konstruktor an, in dem Konstruktor bauen wir uns mit Hilfe des Shadow-DOM unsere Komponente aus bestehenden HTML-Komponenten im Hintergrund zusammen:

Wichtig hier: wir müssen den Shadow-DOM in den OPEN-Modus stellen, damit wir überhaupt im Shadow-DOM operieren können...

```
constructor() {  
  super();  
  this._shadowRoot = this.attachShadow({  
    mode: 'open'  
  });  
  this._shadowRoot.innerHTML = `  
    <link href="assets/flags.css" rel=stylesheet type="text/css">  
    `;  
}
```

- Durch die Backticks können wir HTML-Code über mehrere Zeilen schreiben
- Zuerst die updateFlag und dann die connectedCallback-Methode implementieren, letztere ist eine sogenannte Lifecycle-Methode die immer dann aufgerufen wird, wenn die Komponente zum mit dem DOM verbunden wird.

```
connectedCallback() {  
  this.updateFlag ( this.getAttribute("country") );  
}  
  
updateFlag(country) {  
  this._img = this._shadowRoot.querySelector("#flag-image");  
  this._img.className = 'flag flag-' + country;  
}
```

- Außerdem die Klasse am Browser als Custom-Element anmelden:

```
customElements.define('flag-icon', FlagIcon);
```

- Jetzt ändern wir noch die HTML-Seite flags.html wie folgt:

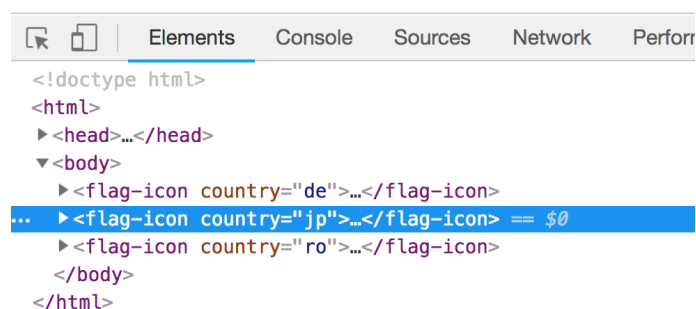
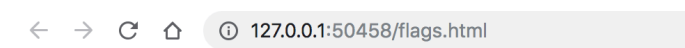
```
<!DOCTYPE html>
<html>
  <head>
    <script src="flags.js"></script>
  </head>
  <body>
    <flag-icon country="de"></flag-icon>
    <flag-icon country="jp"></flag-icon>
    <flag-icon country="ro"></flag-icon>
  </body>
</html>
```

- Danach im Browser starten, jetzt sollten die Flaggen sauber dargestellt werden.

Herzlichen Glückwunsch zur ersten eigenen Komponente!

**Aber... es gibt noch ein Problem... die Komponente reagiert noch nicht auf Veränderungen....**

Mit ALIT + CMD + I die Developer Tools im Chrome einschalten und im DOM-Baum mal zu unserer Komponente navigieren. Manuell z.B. „jp“ in „gb“ für Großbritannien ändern. Komponente reagiert leider nicht.



Lösung: **attributeChangedCallback()**

Folgende Methode hinzufügen:

```
attributeChangedCallback(name, oldValue, newValue) {  
    this.updateFlag(newValue);  
}
```

**Wichtig:** Die `connectedCallback` müssen wir jetzt ändern weil sonst das Land ja zwei mal gesetzt wird. Am besten eine `console.log` einbauen!

Außerdem muss in der Komponente noch festgelegt werden, welche Attribute auf Veränderungen überwacht werden sollen:

```
static get observedAttributes() {  
    return ['country'];  
}
```

Nochmal testen: Jetzt reagiert die Komponente auch auf Veränderungen.

## ES MODULES Demo .....

Jetzt stellen wir uns vor, wir wollen einen einfachen Logger einbinden, diesen aber auslagern und über das Module-System verwenden.

Dazu legen wir zunächst eine neue JavaScript-Datei mit dem Namen `SimpleLogger.js` an und schreiben folgenden Code:



```
1 export default function(logMessage) {  
2     console.log(logMessage)  
3 }
```

Wir importieren anschließend in `flags.js` mit folgender Code-Zeile

```
import log from './SimpleLogger.js'
```

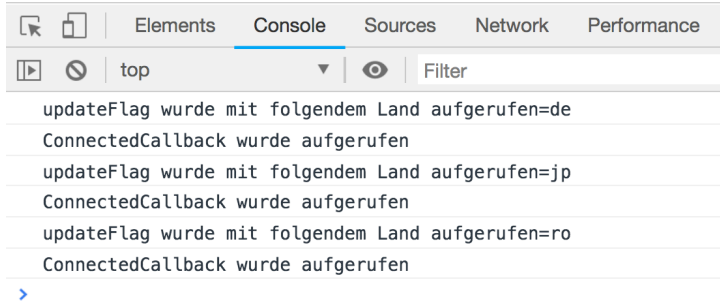
Anschließend die log-Methode auch verwenden:

```
connectedCallback() {  
  log('ConnectedCallback wurde aufgerufen')  
}  
  
updateFlag(country) {  
  this._img = this._shadowRoot.querySelector("#flag-image");  
  this._img.className = 'flag flag-' + country;  
  log('updateFlag wurde mit folgendem Land aufgerufen=' + country)  
}  
  
attributeChangedCallback(name, oldValue, newValue) {  
  this.updateFlag(newValue);  
}
```

Wird nicht funktionieren, da das Modul-System nicht aktiviert ist, siehe Fehlermeldung in den Developer Tools. Das Modul-System wie folgt aktivieren:

```
<head>  
  <script type="module" src="flags.js"></script>  
</head>
```

Ergebnis:



## TEMPLATES....

Warum Templates?

- Wenn ein und dasselbe HTML-Fragment wiederholt verwendet werden soll, ist es sinnvoll eine Art Vorlage (Template) zu verwenden
- Das war zwar vorher durch gezielte DOM-Manipulation mittels JavaScript auch schon möglich, geht jetzt aber viel effizienter und leichter (keine CSS-Visible Hacks und dergleichen)
- Da HTML-Templates nicht im DOM gerendert werden, sondern erst dann wenn Sie hinzugefügt werden, sind Templates auch schneller ....

### Beispiel:

**Wir wollen Gruppen mit jeweils 3 Flaggen bilden und diese auf Knopfdruck darstellen.**

Um dies zu erreichen brauchen wir zunächst mal einen Container, in denen wir unsere Gruppe von 3 Flaggen platzieren können:

```
<div id='flagHolder'></div>
```

Als nächstes definieren wir ein Template mit Hilfe des Template-Tags eine Vorlage in die wir 3 Flaggen positionieren:

```
<template id='flagTemplate'>
  <flag-icon id='flag0' country='de'></flag-icon>
  <flag-icon id='flag1' country='de'></flag-icon>
  <flag-icon id='flag2' country='de'></flag-icon>
</template>
```

Und dann noch zwei Knöpfe über das Template, über die wollen wir die Gruppen von Flaggen aktivieren:

```
<button>Gruppe 1</button>
<button>Gruppe 2</button>
```



Wenn wir die Anwendung jetzt laufenden Betrieb betrachten, sehen wir das Template, aber es wird nichts Sichtbares gerendert:

```
<html>
  <head>
    <script type="module" src="flags.js"></script>
  </head>
  <body> == $0
    <div id="flagHolder"></div>
    <button>Gruppe 1</button>
    <button>Gruppe 2</button>
    <template id="flagTemplate">
      <#document-fragment
        <flag-icon id="flag0" country="de"></flag-icon>
        <flag-icon id="flag1" country="de"></flag-icon>
        <flag-icon id="flag2" country="de"></flag-icon>
      </#document-fragment>
    </template>
  </body>
</html>
```

Jetzt brauchen wir noch eine Prise JavaScript:

```
<script>
  function showFlags(arr) {
    let templateNode = document.getElementById('flagTemplate').content.cloneNode(true)
    arr.forEach( (country, index) => {
      templateNode.getElementById('flag' + index).setAttribute('country', country)
    });

    let holder = document.getElementById('flagHolder')
    holder.innerHTML = ''
    holder.append(templateNode)
  }
</script>
```

Und wir müssen die Funktion im Button natürlich noch aufrufen:

```
<button onclick="showFlags( [ 'de','gb','fr' ] )">Gruppe 1</button>
<button onclick="showFlags( [ 'jp','us','ro' ] )">Gruppe 2</button>
```

- Insbesondere in Web Components machen Templates noch mehr Sinn:
  - Im Beispiel vorher wird die Flagge immer wieder neu im (Shadow-)DOM (innerHTML) aufgebaut
  - Wenn man hier Templates verwendet, werden Web Components schneller, da man das Template nur einmal definiert und dann immer wieder neu verwenden kann
  - Mit sog. Slots <slot> kann man im Template zusätzlich Platzhalter markieren, an denen später zur Laufzeit definierte Inhalte gesetzt werden.

Code wie folgt ändern und erklären:

```
class FlagIcon extends HTMLElement {

  constructor() {
    super();
    const template = document.createElement('template')
    template.innerHTML = `<link href="assets/flags.css" rel=stylesheet type="text/css">
    `
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.appendChild(template.content.cloneNode(true))
  }

  connectedCallback() {
    log('ConnectedCallback wurde aufgerufen')
  }

  updateFlag(country) {
    this._img = this.shadowRoot.querySelector("#flag-image");
    this._img.className = 'flag flag-' + country;
    log('updateFlag wurde mit folgendem Land aufgerufen=' + country)
  }

  attributeChangedCallback(name, oldValue, newValue) {
    this.updateFlag(newValue);
  }

  static get observedAttributes() {
    return ['country'];
  }
}
```

→ Folie „Wer echte Web Components ....“

→ Folie „Angular Vue React“ folgen nicht dem Web Component Standard zeigen

**Bevor wir aber jetzt ins Vue-Framework eintauchen, nochmal ein ganz wichtiger Hinweis:**

→ Foliensatz „Nicht das Framework ist entscheidend...“ durchgehen

## Einstieg in VUE

→ Vue wie in 1 Std. vorstellen, Folien „vuejs“ und „Wichtige Vue-Bestandteile“ zeigen...

## HANDS-ON (3): VUE BASICS

### Ziel: Vue ins Projekt bringen, aktivieren und eine erste Expression verwenden

In Vue dreht sich alles um ein zentrales Vue-Objekt.

- Dieses Vue-Objekt bindet man an HTML-Elemente.
- Alle HTML-Elemente die unter Kontrolle eines Vue-Objekts gesetzt werden sollen sollten daher in einem <div> zusammengefasst werden. Dieses <div> muss unter die Kontrolle des Vue-Objekts gestellt werden.
- Es kann mehr als ein Vue-Objekt geben.

<flag-icon> in ein <div> einpacken mit id="app"

Anwendung läuft immer noch problemlos

---

Jetzt legen wir ein Vue-Objekt an

#### Neue Datei my.js anlegen

```
var app = new Vue ({  
  el: '#app',  
  data: {  
    country: 'ro'  
  }  
});
```

Ein Vue-Objekt wird über den Konstruktor konfiguriert und erhält ein Objekt-Array.

Mit **el** schaffen wir die Verbindung zum HTML-Fragment das wir unter Kontrolle bringen wollen

Im Bereich **data** können wir Variablen deklarieren die wir dann als sog. Expression in geschweiften Klammern oder über Data Binding auf die Oberfläche bringen

Das Herunterladen von Vue gestaltet sich ironischerweise schwieriger als gedacht. Schauen wir uns auf der Webseite von Vue um, werden wir sehr schnell feststellen, dass hier nirgendwo ein Download angeboten wird.

Auf dem zweiten Blick findet sich dann der Verweis zu Github, wo die Sourcen von Vue eingesehen und heruntergeladen werden können und auch entsprechende Releases bereitstehen. Lädt man sich das neueste Release herunter, finden sich die einzubindenden Bibliotheken im Unterordner dist.

Das manuelle Herunterladen solcher SDKs, wie man diese Art von Releases früher auch einmal nannte, ist insbesondere in der modernen JavaScript-Welt nicht mehr zeitgemäß.

Es gibt viel effizientere Möglichkeiten um Vue zu beziehen und ins eigene Projekt zu bringen.

Wir ein sogenanntes Content Delivery Network (CDN) nutzen. Hierbei handelt es sich laut Wikipedias um ein Netz verteilter und über das Internet verbundener Server die Inhalte, meistens große Mediendaten und in unserem Fall JavaScript-Bibliotheken, ausliefern.

Speziell für JavaScript empfiehlt sich der Anbieter jsDelivr.

Hierbei handelt es sich um ein CDN mit dem man beliebige JavaScript-Bibliotheken die über NPM oder auf Github verfügbar sind beziehen und direkt über den `<script>` Tag in eigene Programm einbinden kann.

**Folie „Vue über ein CDN einbinden“ zeigen**

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

---

Jetzt brauchen wir die JavaScript-Datei mit dem Vue-Objekt:

```
<script src="my.js"></script>
```

Wichtig: Muss ans Ende, da zur Laufzeit der Code in dieser Skriptdatei sofort ausgeführt wird.

Würde er weiter oben eingebunden, würde Vue das zu verbindende div nicht finden.

---

{{ country }} ausserhalb des <div> schreiben

Hierbei handelt es sich um eine sog. Expression, d.h. ich kann auf die Variablen im Vue-Objekt hier zugreifen und zwar in Echtzeit

Laufen lassen – funktioniert nicht.

Muss natürlich an die richtige Stelle geschoben werden

---

## Ziel: Attribute Binding – Attribute in HTML-Tags von JavaScript Seite setzen

Expression wieder entfernen.

Als nächstes wollen wir das Land auch von der Vue-Instanz setzen lassen. Dazu nutzen wir das sog. Attribute-Binding.

Vue stellt uns bestimmte Erweiterungen, die wir an Attribute hängen können um diese Attribut an ein Vue-Objekt zu binden. In Angular heißen diese Elemente Direktiven.

```
<flag-icon v-bind:country="country"></flag-icon>
```

Damit wird die Variable „country“ aus dem Vue-Objekt an das country-Attribut unseres Flag Icons gebunden.

Für häufig genutzte Direktiven wie v-bind gibt es auch sog. „Shorthands“ so reicht ein „:“ anstelle v-bind aus um exakt das gleiche zu erreichen.

---

## Ziel: Reaktives Data Binding demonstrieren

In laufender Anwendung in die Entwickler-Konsole

app. → Vollzugriff auf die Vue-Instanz

app.country → Land umstellen → Reaktives Data Binding → Änderung wirkt sich unmittelbar und sofort aus

Das Vue-Objekt liefert noch eine Reihe eigener, Hilfs-Methoden → beginnen mit \$

app.\$el

app.\$root

---

Wenn Vue gestartet wird durchläuft es einen Lebenszyklus...

→ Folie „**Lebenszyklus**“ zeigen...

---

Wir wollen uns in mitkriegen, wenn die Vue-Instanz fertig gemounted ist, und wir wollen mitkriegen, wenn sich eine Property geändert hat und die Oberfläche deshalb aktualisiert wurde:

```
var app = new Vue ({
  el: '#app',
  data: {
    country: 'ro'
  },
  updated() {
    console.log('updated')
  },
  mounted() {
    console.log('mounted')
  }
});
```

Im nächsten Schritt wollen wir das Property „country“ beobachten und über Änderungen in diesem Property informiert werden:

```
app.$watch('country', function (newValue, oldValue) {
  console.log('old=' + oldValue + ' ==> new=' + newValue);
});
```

## Ziel: Computed Properties – Variablen definieren, die sich durch Logik ergeben

```
var countries = ["de", "gb", "fr", "be", "ro", "us", "cz"];

function getRandom(max) {
  return Math.floor(Math.random() * Math.floor(max));
}

var app = new Vue( {
  el: '#app',
  computed: {
    country: function () {
      return countries[getRandom(countries.length)]
    }
  }
});
```

Führt dazu dass immer ein neues Land gesetzt wird. Mit **Computed Properties** können wir Variablen definieren, die sich durch Logik ergeben.

---

## Ziel: Event Handling – JavaScript-Methoden von HTML-Seite anstoßen

```
<div id="app">
  <button v-on:click="changeCountry">Change Country</button> <br />
  <flag-icon :country="country"></flag-icon>
</div>
```

Und im Java-Script Code direkt an eine Methode binden:

```
var app = new Vue( {
  el: '#app',
  data: {
    country: 'de'
  },
  methods: {
    changeCountry: function() {
      this.country = countries[getRandom(countries.length)];
    }
  }
});
```

→ Folien “Warum Komponenten” ... zeigen

## HANDS-ON (3): VUE COMPONENTS

Ausgangsbasis:

- Projekt kopieren, damit wir keine Beispiele verlieren
- flags.js wird nicht mehr benötigt deshalb wird es jetzt gelöscht
- Aus my.js wird erstmal alles Vue-bezogene entfernt
- in flags.html wollen wir jetzt eine Komponente **<vue-flag-icon>** verwenden

```
<div id="app">
  <vue-flag-icon country="de"></vue-flag-icon>
</div>
```

Wenn man dies startet funktioniert das natürlich noch nicht

---

Als erstes muss die Komponente bei Vue registriert werden

Komponenten kann man global mittel Vue.component anmelden und werden genauso konfiguriert wie Vue-Objekte – Vue Komponenten sind in nämlich Vue Objekte !

```
Vue.component('vue-flag-icon', {
  template: `<div>
    <link href="assets/flags.css" rel=stylesheet type="text/css">
    
  </div>`
});
```

Einfügen und erklären. Zwischen dem Backtick bauen wir unsere Komponente aus Standard HTML-zusammen. Wichtig, die einzelnen HTML-Bestandteile müssen in einem sog. Root-Element zusammengefasst sein.

Der : bei class zeigt an, dass wir man in Vue getreu dem Motto „eat your own dog foot“ Vue-Prinzipien einsetzen kann. Das Class-Attribut soll später also über Data Binding dynamisch gesetzt werden können

---

Als nächstes brauchen wir das Attribut „country“, dieses konfigurieren wir wie folgt:

```
props: ['country']
```



---

Außerdem müssen wir irgendwie dafür sorgen, dass der Wert von Property ausgelesen wird und dafür gesorgt wird, dass die CSS-Klassen entsprechend eingestellt werden und die richtige Flagge gerendert werden kann:

Dafür brauchen wir die Konfiguration von **data**

```
data: function() {  
  return {  
    css: 'flag flag-' + this.country  
  }  
}
```

Hier muss eine Funktion geschrieben werden die ein Array von Variablen zurückgibt. In unserem Fall „css“ die mit : ja gebunden ist !

Grund dass hier eine Funktion benötigt ist einfach:

Wenn mehrere Instanzen dieser Komponenten verwendet würden, dürfen die Daten zwischen diesen Instanzen ja nicht geteilt werden, daher müssen sie mit Hilfe einer Funktion erzeugt werden !

Um Vue anzustarten müssen wir natürlich noch den div mit der id=app mit einem Vue-Objekt verbinden (Root- Element setzen)

```
var app = new Vue( {  
  el: '#app'  
});
```

**Jetzt funktioniert die Komponente bereits !!!**

**ABER: Im DOM wird sie doof dargestellt, da haben Web Components ihren Vorteil, die werden auch richtig dargestellt.**

**Abhilfe: Vue Development Tools installieren! Zeigen!**

**Übung mit props:** Sicherstellen, dass das Land nur mit 2 Zeichen eingegeben werden:

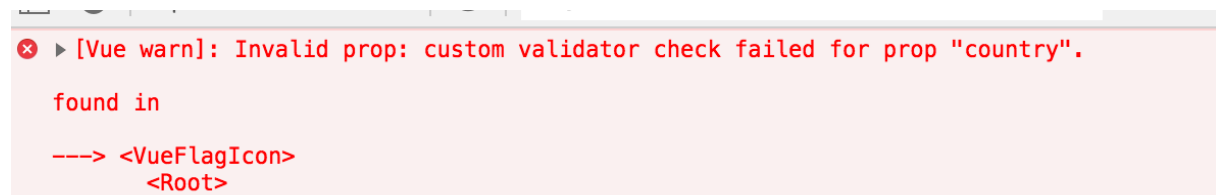
```
props: {
  country: {
    type: String,
    default: 'de',
    validator: function (value) {
      return value.length == 2;
    }
  }
},
```

Im Html wie „gb“ einstellen – geht

Im Html „usa“ einstellen:

```
<div id="app">
  <vue-flag-icon country="usa"></vue-flag-icon>
</div>
```

Führt zu folgendem Warning:



Jetzt wollen wir noch das Land auf Knopfdruck verändern, wir wollen also das Attribut dynamisch verändern:

```
var app = new Vue( {
  el: '#app',
  data: {
    selectedCountry: 'de'
  },
  methods: {
    changeCountry: function() {
      this.selectedCountry = countries[getRandom(countries.length)];
    }
  }
});
```

Und im HTML:

```
<div id="app">
  <button @click="changeCountry" >Change Country</button> <br />
  <vue-flag-icon :country="selectedCountry"></vue-flag-icon>
</div>
```

Da wir als initiales Land im Vue-Objekt „de“ angegeben haben, wird dieses jetzt korrekt gerendert – der Flaggenwechsel auf Knopfdruck funktioniert aber noch nicht.

Warum?

Um auf Attributsänderungen reagieren zu können, müssen wir sog. „**Computed Properties**“ verwenden!

Wir ersetzen data durch computed:

```
computed: {
  css: function() {
    return 'flag flag-' + this.country;
  }
}
```

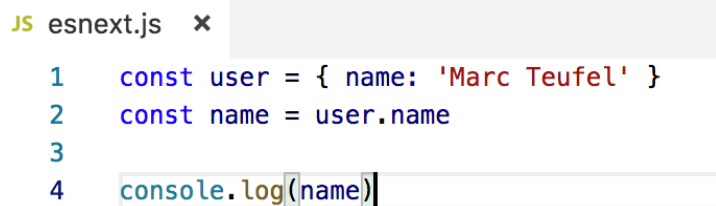
## HANDS-ON (4): ZUKÜNFTIGE ECMASCRIPT-VERSIONEN HEUTE SCHON BENUTZEN

### ERSTE SCHRITTE MIT BABELJS

Grundlegende Projektstruktur anlegen:

```
mkdir my-esnext  
cd my-esnext  
npm init -y  
cd ..  
code my-esnext
```

Neue Java-Script-Datei **src/esnext.js** anlegen und folgenden Code eintippen:



```
JS esnext.js x  
1  const user = { name: 'Marc Teufel' }  
2  const name = user.name  
3  
4  console.log(name)
```

Speichern und über das integrierte Terminal ausführen:



```
PROBLEME  AUSGABE  DEBUGGING-KONSOLE  TERMINAL  
marcteufel@Marcs-iMac:~/my-esnext$ node esnext.js  
Marc Teufel  
marcteufel@Marcs-iMac:~/my-esnext$
```

#### Tipp:

console.log ist zwar cool, aber trotzdem aber immer noch produktiv genug!  
Es gibt mittlerweile Werkzeuge die uns die Möglichkeit geben, Ausdrücke direkt während der Entwicklung auswerten, Funktionen direkt aus der Entwicklungsumgebung heraus direkt aufzurufen und auszuwerten.

Hierzu empfehl ich das Tool Quokka, welches auch in einer kostenlosen Community Variante für VSCode und Webstorm zur Verfügung steht!

Quokka kann direkt aus VSCode heraus installiert werden!

`console.log(name)` entfernen!

Quokka starten: SHIFT + CMD + P → "Quokka.js: Start on current file"

```
const user = { name: 'Marc Teufel' }  
const name = user.name  
  
user { name: 'Marc Teufel' }  
name Marc Teufel
```

Quokka stoppen: SHIFT + CMD + P → "Quokka.js: Stop Current"

Zurück zu ES.NEXT...

Wir wollen nun Optional Chaining ausprobieren, ein Feature das erst in einer der nächsten JavaScript-Versionen offiziell enthalten sein wird, **aber schon heute genutzt werden kann, wie das klären wir gleich**

Wenn wir versuchen auf tiefergeschachtelte Werte zuzugreifen, die es im Original-Objekt nicht gibt, gibt es einen Laufzeitfehler....

```
JS esnext.js ●  
1  const user = { name: 'Marc Teufel' }  
2  const name = user.name  
3  
4  const plz = user.address.plz  Cannot read property 'plz' of undefined
```

In heutigen JavaScript-Versionen müssten wir also entweder total kompliziert mit ternären Blocken oder mit Try...Catch überprüfen, ob die Struktur vorhanden ist um den Laufzeitfehler zu umgehen:

```
JS esnext.js ●
1  const user = { name: 'Marc Teufel', address: { plz: 91792 } }
2  const name = user.name
3
4  // Logisch
5  const plz1 = user && user.address && user.address.zip
6
7  // Terniär
8  const plz2 = user === undefined ? undefined
9  |           |           |           : user.address === undefined ? undefined
10 |           |           |           : user.address.plz
11
12 // Try...Catch
13 let plz3
14 try {
15     plz3 = user.address.plz
16 } catch (error) {
17     plz3 = undefined
18 }
19
20 plz3  91792
```

**Alles Scheisse!** Optional Chaining hilft uns hier, wir können nämlich in Zukunft folgenden syntaktischen Zucker hinschreiben:

```
JS esnext.js ✕
1  const user = { name: 'Marc Teufel' }
2  const name = user.name
3
4  const plz = user?.address?.plz
5
6  console.log(plz)
```

PROBLEME 3 AUSGABE DEBUGGING-KONSOLE TERMINAL

```
marcteufel@Marcs-iMac:~/my-esnext$ node esnext.js
Marc Teufel
marcteufel@Marcs-iMac:~/my-esnext$ node esnext.js
undefined
marcteufel@Marcs-iMac:~/my-esnext$ node esnext.js
/Users/marcteufel/my-esnext/esnext.js:4
const plz = user?.address?.plz
               ^
SyntaxError: Unexpected token .
    at createScript (vm.js:80:10)
    at Object.runInThisContext (vm.js:152:10)
    at Module._compile (module.js:605:28)
    at Object.Module._extensions..js (module.js:652:10)
    at Module.load (module.js:560:32)
    at tryModuleLoad (module.js:503:12)
    at Function.Module._load (module.js:495:3)
    at Function.Module.runMain (module.js:682:10)
    at startup (bootstrap_node.js:191:16)
    at bootstrap_node.js:613:3
marcteufel@Marcs-iMac:~/my-esnext$
```

Aber wie die Fehler im Editor anzeigen, ist der “?.”-Operator noch nicht bekannt. Zur Laufzeit gibt es einen Fehler.

Um zukünftige JavaScript-Versionen zu nutzen, können wir einen sogenannten Transpiler einsetzen. Dieser transpilirt/kompiliert den Quellcode in eine JavaScript-Version die von den gängigsten Browser/Node-Versionen verstanden werden. Auf die gleiche Weise funktioniert auch TypeScript, TypeScript-Code wird mit Hilfe eines Transpilers in stinknormales JavaScript umgewandelt....

Wir wollen nun den JavaScript-Compiler **babeljs** einsetzen.

Wir installieren im Verzeichnis `my-esnext` mit folgendem Befehl:

```
npm install --save-dev @babel/core @babel/cli @babel/node
@babel/plugin-proposal-optional-chaining nodemon
```

Danach erweitern wir die `package.json` wie folgt:

```
"scripts": {
  "start": "nodemon --exec babel-node src/esnext.js"
},
"babel": {
  "plugins": [
    "@babel/plugin-proposal-optional-chaining"
  ]
},
"keywords": [],
```

**Wichtig: Die Datei `esnext.js` muss im Unterverzeichniss `src` liegen!**

```
npm run start
```

Gibt `undefined` aus, jede Änderung an der Datei wird jetzt überwacht und der Transpile mit Hilfe von Nodemon und Babel automatisch ausgeführt....

Wie man Visual Studio Code jetzt auf ES.NEXT bringt habe ich leider noch nicht herausgefunden

```
sudo npm install -g @vue/cli
```

```
vue create flags-sfc
```

→ Manually select features

→ nur Babel

```
npm run serve
```

Anwendung läuft im Browser auf Port 8080

Arbeitsordner flags-src in Code öffnen...

Änderung im Code machen, wird sofort im Browser angezeigt

Erklären wie die Projekt aufgebaut ist

Gemeinsam versuchen unsere Flaggen in diese Struktur zu bringen!