

WBE: UI-BIBLIOTHEK

TEIL 2: IMPLEMENTIERUNG

ÜBERSICHT

- Interne Repräsentation und das DOM
- Komponenten und Properties
- Darstellung von Komponenten
- Defaults und weitere Beispiele

ÜBERSICHT

- Interne Repräsentation und das DOM
- Komponenten und Properties
- Darstellung von Komponenten
- Defaults und weitere Beispiele

RÜCKBLICK

- Ziel: eigene kleine Bibliothek entwickeln
- Komponentenbasiert und datengesteuert
- An Ideen von React.js und ähnlicher Systeme orientiert
- Motto: „Keep it simple!”
- Bezeichnung:

SuiWeb

Simple User Interface Toolkit for Web Exercises

RÜCKBLICK

- Notation für den Aufbau der Komponenten
 - **JSX**: in React.js verwendet
 - **SJDON**: eigene Notation
- SuiWeb soll beide Varianten unterstützen

```
// jsx
const element = (<h1 title="foo">Hello</h1>)

// sjdon
const element = ["h1", {title: "foo"}, "Hello"]
```

ANSTEHENDE AUFGABEN

- Interne Repräsentation der Komponenten
- Konvertierung von JSX und SJDON in diese Repräsentation
- Abbildung interne Repräsentation ins DOM
- Daten steuern Komponenten: Properties
- Hierarchie von Komponenten
- Komponenten mit Zustand

Anregungen und Code-Ausschnitte aus:

Rodrigo Pombo: Build your own React

<https://pomb.us/build-your-own-react/>

AUSGANGSPUNKT

```
// jsx
/** @jsx createElement */
const element = (<h1 title="foo">Hello</h1>)

// jsx babel output (React < 17)
const element = createElement(
  "h1",
  { title: "foo" },
  "Hello"
)

// sjdon
const element = ["h1", {title: "foo"}, "Hello"]
```

INTERNE REPRÄSENTATION

```
// jsx babel output
const element = createElement(
  "h1",
  { title: "foo" },
  "Hello"
)
```

```
// internal representation
const element = {
  type: "h1",
  props: {
    title: "foo",
    children: ["Hello"],
  },
}
```


INTERNE REPRÄSENTATION

```
{
  type: "h1",
  props: {
    title: "foo",
    children: ["Hello"],    /* noch anzupassen */
  },
}
```

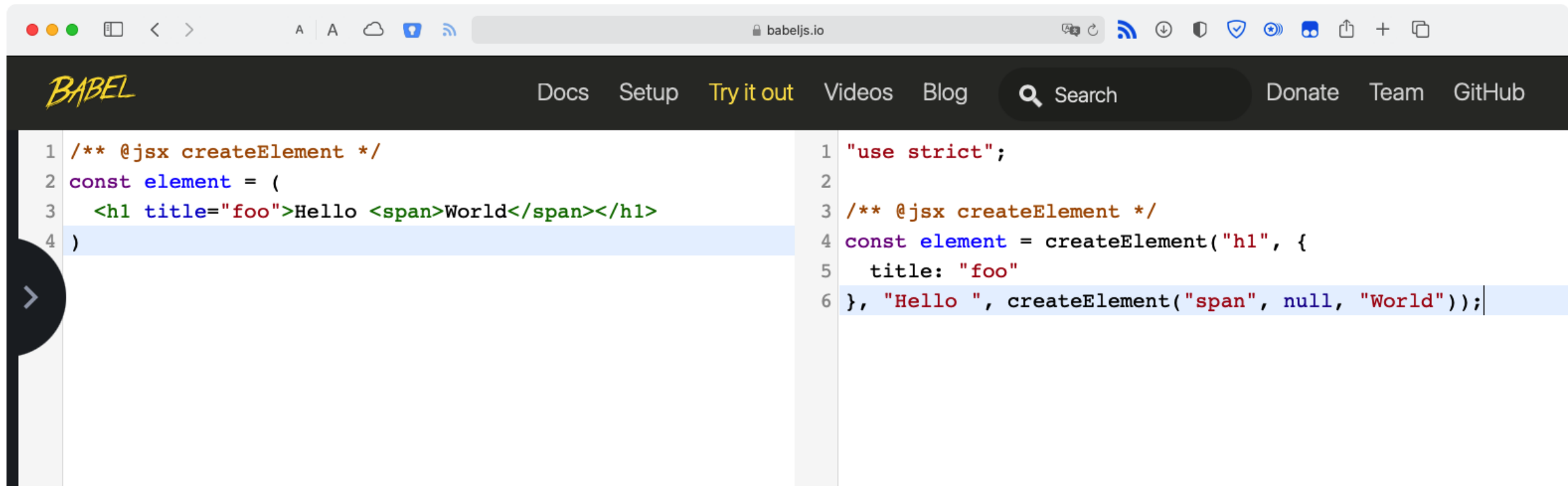
- Element: Objekt mit zwei Attributen, `type` und `props`
- `type`: Name des Elements ("body", "h1", ...)
- `props`: Attribute des Elements
- `props.children`: Kindelemente (Array)

TEXT-ELEMENT

```
{  
  type: "TEXT_ELEMENT",  
  props: {  
    nodeValue: "Hello",  
    children: [],  
  },  
}
```

- Aufbau analog zu anderen Elementen
- Spezieller Typ: `"TEXT_ELEMENT"`

VERSCHACHTELTE ELEMENTE



The screenshot shows the Babel website interface. The left pane contains the input JSX code, and the right pane shows the transformed code using `createElement`.

```
1 /** @jsx createElement */
2 const element = (
3   <h1 title="foo">Hello <span>World</span></h1>
4 )
```

```
1 "use strict";
2
3 /** @jsx createElement */
4 const element = createElement("h1", {
5   title: "foo"
6 }, "Hello ", createElement("span", null, "World"));
```

- Mehrere Kindelemente:
ab drittem Argument von `createElement`
- Verschachtelte Elemente:
rekursive Aufrufe von `createElement`

KONVERTIERUNG VON JSX

```
function createElement (type, props,
                        ...children) {
  return {
    type,
    props: {
      ...props,
      children: children.map(child =>
        typeof child === "object"
          ? child
          : createTextElement(child)
      ),
    },
  }
}
```

```
function createTextElement (text) {
  return {
    type: "TEXT_ELEMENT",
    props: {
      nodeValue: text,
      children: [],
    },
  }
}
```

CREATEELEMENT: BEISPIEL

```
// <div>Hello<br></div>
createElement("div", null, "Hello", createElement("br", null))

// returns
{
  type: 'div',
  props: {
    children: [
      {
        type: 'TEXT_ELEMENT',
        props: { nodeValue: 'Hello', children: [] }
      },
      { type: 'br', props: { children: [] } }
    ]
  }
}
```

KONVERTIERUNG VON SJDON

```
1 function parseSJDON ([type, ...rest]) {  
2   const isObj = (obj) => typeof(obj)==='object' && !Array.isArray(obj)  
3   const children = rest.filter(item => !isObj(item))  
4  
5   return createElement(type,  
6     Object.assign({}, ...rest.filter(isObj)),  
7     ...children.map(ch => Array.isArray(ch) ? parseSJDON(ch) : ch)  
8   )  
9 }
```

- Abbildung auf `createElement`-Funktion
- Attribute in einem Objekt zusammengeführt
- Kindelemente bei Bedarf (Array) ebenfalls geparst

ZWISCHENSTAND

- Einheitliche Repräsentation für Elemente unabhängig von der ursprünglichen Syntax (JSX or SJDON)
- Baumstruktur von Elementen
- Text-Elemente mit leerem Array `children`
- DOM-Fragment im Speicher repräsentiert (**virtuelles DOM?**)

Zu tun:

- Abbildung der Baumstruktur ins DOM

RENDER TO DOM

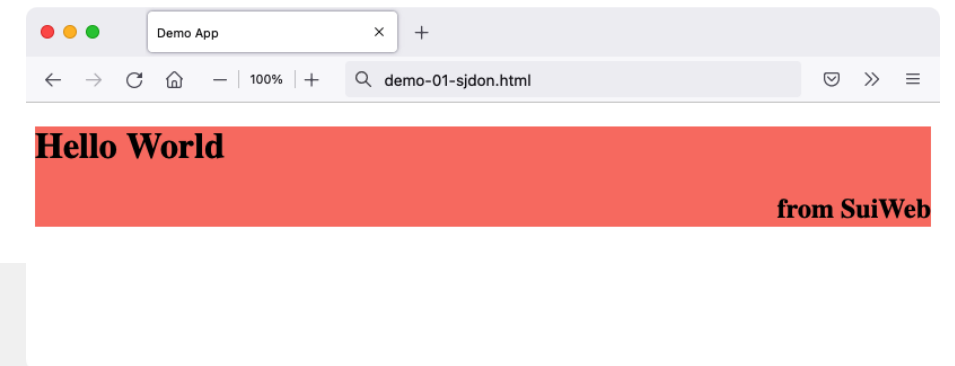
```
1 function render (element, container) {
2   /* create DOM node */
3   const dom =
4     element.type == "TEXT_ELEMENT"
5       ? document.createTextNode("")
6       : document.createElement(element.type)
7
8   /* assign the element props */
9   const isProperty = key => key !== "children"
10  Object.keys(element.props)
11    .filter(isProperty)
12    .forEach(name => { dom[name] = element.props[name] })
13
14  /* render children */
15  element.props.children.forEach(child => render(child, dom))
16  /* add node to container */
17  container.appendChild(dom)
18 }
```


HTML-ELEMENTE

- Komponenten können HTML-Elemente verwenden
- Tagnamen in Kleinbuchstaben
- Gross-/Kleinschreibung ist relevant
- Übliche Attribute für HTML-Elemente möglich
- Wenig Ausnahmen: `className` statt `class`

BEISPIEL

```
1 import { createElement } from './lib/fiber.js'
2 import { parseSjdon } from './lib/sjdon.js'
3 import { render } from './lib/render.js'
4
5 const element =
6   ["div", {style: "background:salmon"},
7     ["h1", "Hello World"],
8     ["h2", {style: "text-align:right"}, "from SuiWeb"] ]
9
10 const container = document.getElementById("root")
11 render(parseSjdon(element, createElement), container)
```



ZWISCHENSTAND

- Interne Struktur aufbauen
- Ins DOM rendern

SuiWeb: JSX, SJDON

demo-01-jsx.html →

demo-01-sjdon.html →

Didact: (Rodrigo Pombo)

<https://codesandbox.io/s/didact-2-k6rbj?file=/src/index.js>

ÜBERSICHT

- Interne Repräsentation und das DOM
- Komponenten und Properties
- Darstellung von Komponenten
- Defaults und weitere Beispiele

FUNKTIONSKOMPONENTEN

```
1 const App = (props) =>  
2   ["h1", "Hi ", props.name]  
3  
4 const element =  
5   [App, {name: "foo"}]
```

- App ist eine Funktionskomponente
- Die zugehörige Repräsentation erzeugt keinen DOM-Knoten
- Ergebnis des Funktionsaufrufs wird als *child* eingehängt
- Konvention: eigene Komponenten mit grossen Anfangsbuchstaben

PROBLEM

- Komponenten können sich auf `props.children` beziehen
- Unter SJDON werden diese in SJDON-Form erwartet
- Diese steht in der internen Repräsentation aber nicht mehr zur Verfügung (bereits mit `createElement` umgeformt)
- Interne Repräsentation daher um ein Attribut `sjdon` erweitert

SJDON-KONVERTIERUNG ERWEITERT

```
1 function parseSJDON ([type, ...rest]) {
2   const isObj = (obj) => typeof(obj)=== 'object' && !Array.isArray(obj)
3   const children = rest.filter(item => !isObj(item))
4
5   const repr = createElement(type,
6     Object.assign({}, ...rest.filter(isObj)),
7     ...children.map(ch => Array.isArray(ch) ? parseSJDON(ch) : ch)
8   )
9   repr.sjdon = children
10  return repr
11 }
```

FUNKTIONSKOMPONENTEN

```
1 function render (element, container) {  
2   if (Array.isArray(element)) {  
3     element = parseSJDON(element)  
4   }  
5  
6   if (element.type instanceof Function) {  
7     updateFunctionComponent(element, container)  
8   } else {  
9     updateHostComponent(element, container)  
10  }  
11 }
```

- Bearbeiten des aktuellen Elements
- Funktionskomponenten separat behandelt

FUNKTIONSKOMPONENTEN

```
1  /* function component
2  /* - run function to get child node
3  /* - render child node
4  */
5  function updateFunctionComponent (element, container) {
6    let children = element.sjdon ?? element.props.children
7    let node = element.type({...element.props, children})
8    renderElem(node, container)
9  }
```

- Funktion wird mit `props` aufgerufen
- Ergebnis ggf. als SJDON geparst
- Resultierendes Kindelement gerendert

BEISPIEL

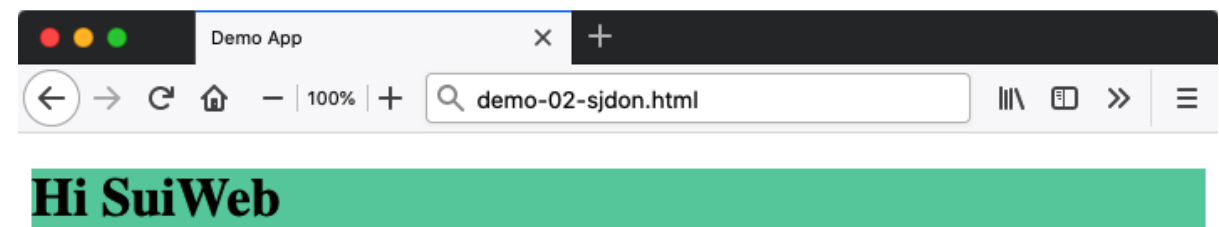
```
const App = (props) =>
  ["h1", {style: "background: mediumaquamarine"}, "Hi ", props.name]

const element =
  [App, {name: "SuiWeb"}]

const container = document.getElementById("root")
render(parseSjdon(element, createElement), container)
```

demo-02-jsx.html →

demo-02-sjdon.html →



WERTE STEUERN UI-AUFBAU

```
const App = () => {  
  const enabled = false  
  const text = 'A Button'  
  const placeholder = 'input value...'  
  const size = 50  
  
  return (  
    ["section",  
      ["button", {disabled: !enabled}, text],  
      ["input", {placeholder, size, autofocus: true}] ]  
  )  
}
```

demo-03-values →

ARRAY ALS LISTE AUSGEBEN

```
const List = ({items}) =>  
  ["ul", ...items.map((item) => ["li", item]) ]  
  
const element =  
  [List, {items: ["milk", "bread", "sugar"]}]
```

- Die `props` werden als Argument übergeben
- Hier interessiert nur das Attribut `items`

demo-04-liste →

OBJEKT ALS TABELLE

```
const ObjTable = ({obj}) =>
  ["table", {style},
    ...Object.keys(obj).map((key) =>
      ["tr", ["td", key], ["td", obj[key]]])]

const style = {
  width: "8em",
  background: "lightblue",
}

const element =
  [ObjTable, {obj: {one: 1111, two: 2222, three: 3333}}]
```

demo-05-object →

VERSCHACHTELN VON ELEMENTEN

```
/* JSX */  
<MySection>  
  <MyButton>My Button Text</MyButton>  
</MySection>
```

- Eigene Komponenten können verschachtelt werden
- `MyButton` ist mit seinem Inhalt in `props.children` von `MySection` enthalten

VERSCHACHTELN VON ELEMENTEN

```
1  const MySection = ({children}) =>
2    ["section", ["h2", "My Section"], ...children]
3
4  const MyButton = ({children}) =>
5    ["button", ...children]
6
7  const element =
8    [MySection, [MyButton, "My Button Text"]]
```

demo-06-nested →

TEILBÄUME WEITERGEBEN

```
1  const Main = ({header, name}) =>
2    ["div",
3     [...header, name],
4     ["p", "Welcome to SuiWeb"] ]
5
6  const App = ({header}) =>
7    [Main, {header, name: "web developers"}]
8
9  const element = [App, {header: ["h2", "Hello "]}]
```

demo-07-subtree →

ÜBERSICHT

- Interne Repräsentation und das DOM
- Komponenten und Properties
- **Darstellung von Komponenten**
- Defaults und weitere Beispiele

DARSTELLUNG

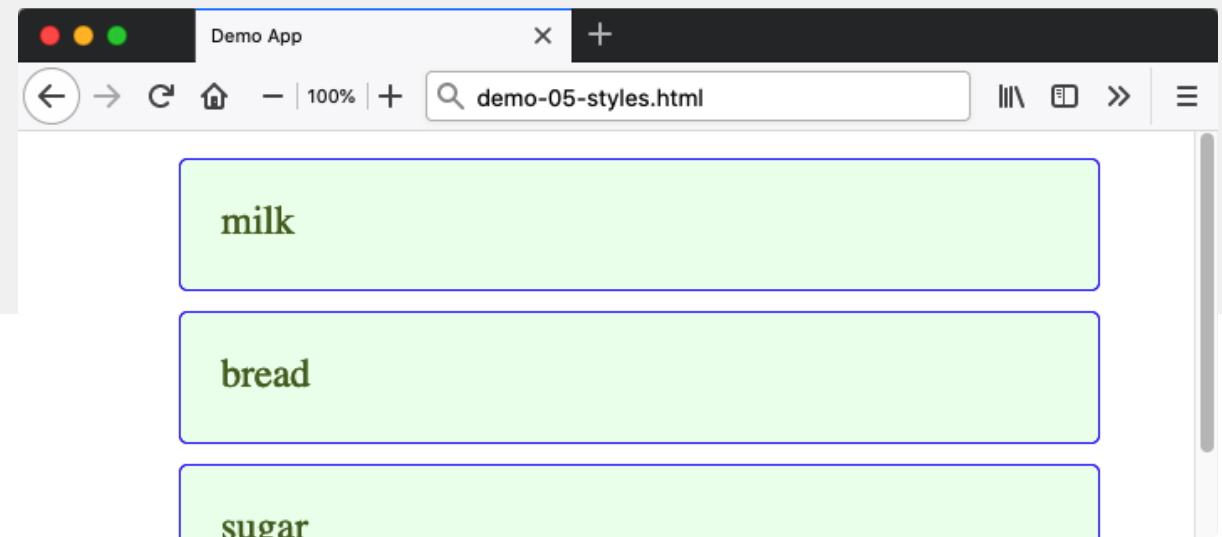
- Komponenten müssen ggf. mehrere Styles mischen können
- Neben Default-Darstellung auch via `props` eingespeist
- Daher verschiedene Varianten vorgesehen:
 - CSS-Stil als String
 - Objekt mit Stilangaben
 - Array mit Stil-Objekten

DARSTELLUNG

```
1 function updateStyleAttribute (dom, styles) {  
2   if (typeof(styles)=="string") {  
3     dom.style = styles  
4   } else if (Array.isArray(styles)) {  
5     Object.assign(dom.style, ...styles)  
6   } else if (typeof(styles)=="object") {  
7     Object.assign(dom.style, styles)  
8   }  
9 }
```

BEISPIEL

```
1 const StyledList = ({items}) => {
2   let style = [styles.listitem, {color: "#556B2F"}]
3   return (
4     ["ul", ...items.map((item) => ["li", {style}, item]) ]
5   )
6 }
7
8 const element =
9   [StyledList, {items: ["milk", "bread", "sugar"]}]]
10
11 const styles = {
12   listitem: {
13     padding: "1em",
14     margin: "0.5em 2em",
15     fontSize: "1.5em",
16     ... }
17 }
```



demo-08-styles →

ÜBERSICHT

- Interne Repräsentation und das DOM
- Komponenten und Properties
- Darstellung von Komponenten
- Defaults und weitere Beispiele

DEFAULT PROPERTIES

```
1 const App = () => (  
2   ["main",  
3     [MyButton, {disabled: true, text: 'Delete'}],  
4     [MyButton] ]  
5 )  
6  
7 const MyButton = ({disabled=false, text='Button'}) => (  
8   ["button", disabled ? {disabled} : {}, text]  
9 )
```

demo-09-defaultprops →

DEFAULT PROPERTIES

- Übergebene Properties überschreiben Defaults
- Selbst zu implementieren (ist einfach, s. Beispiel)
- In React.js können Defaults an Funktion gehängt werden:
(in SuiWeb nicht umgesetzt, wäre aber möglich)

```
const MyButton = (props) => { ... }
```

```
MyButton.defaultProps = {  
  text: 'My Button',  
  disabled: false,  
}
```

WEITERES BEISPIEL

```
1  const MyButton = ({children, disabled=true}) =>
2    ["button", {style: "background: khaki", disabled}, ...children]
3
4  const Header = ({name, children}) =>
5    ["h2", "Hello ", name, ...children]
6
7  const App = (props) =>
8    ["div",
9      [Header, {name: props.name}, " and", ["br"], "web developers"],
10     [MyButton, "Start", {disabled:false}],
11     [MyButton, "Stop"] ]
12
13  const element = [App, {name: "SuiWeb"}]
```

demo-10-children →

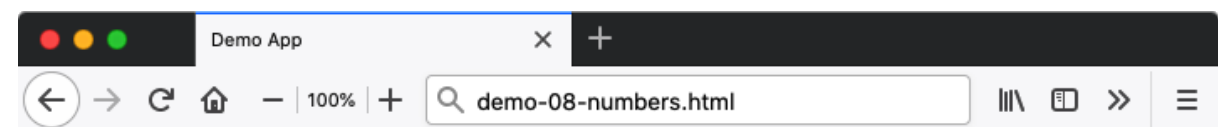
ZAHLEN IN PROPS

```
const App = ({num1, num2}) =>
  ["h1", num1, " * ", num2, " = ", num1*num2]

const element = [App, {num1: 3, num2: 9}]
```

- Beim Funktionsaufruf als Zahlen behandelt
- Beim Rendern in Textknoten abgelegt

demo-11-numbers →



3 * 9 = 27

AKTUELLER STAND

- Notationen, um Komponenten zu definieren: **JSX, SJDON**
- Funktionen zur Anzeige im Browser: **render**-Funktion
- Daten können Komponenten steuern: Argument **props**
- Ausserdem: Verarbeiten von Styles, Default-Properties
- Also: UI-Aufbau mit Komponenten
- Was noch fehlt: Mutation, Zustand
→ nächste Woche 😊

VERWEISE

- Rodrigo Pombo: Build your own React
<https://pomb.us/build-your-own-react/>
- SuiWeb - An Educational Web Framework (Inspired by React)
<https://github.com/suiweb/suiweb>

