heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Front-end frameworks

Component-based development

heig-vd

Building UIs is hard !

Edit in JSFiddle

- JavaScript
- HTML
- Result

```
let direction = 1;
const sortButton = document.getElementById('sort');

function sort(ul, direction) {
  const items = [];
  const newUl = ul.cloneNode(false);

  for (let i = 0; i < ul.childNodes.length; ++i) {
    const node = ul.childNodes[i];
    if (node.nodeName === 'LI') {
      items.push(node);
    }
  }

  items.sort(function(item1, item2) {
    const a = item1.childNodes[0].data;
    const b = item2.childNodes[0].data;
    const result = a > b ? 1 : b > a ? -1 : 0;
    return result * direction;
  });

  items.forEach(function(item) {
    newUl.appendChild(item)
  });

  ul.parentNode.replaceChild(newUl, ul);
}
```

heig-vd

You need an abstraction

- To manage your state
- To be more productive
- To hide complexity
- to scale
- to do more !

heig-vd

How to choose a Framework ?

https://github.com/gothinkster/realworld

heig-vd

How to choose a Framework ?

Things to consider

- ecosystem
- learning curve
- performance
- taste

heig-vd

Meet AngularJS

Hey I'm angularJS.

Nobody wants to use me anymore — except existing large projects that are waiting for migration. So make sure you meet my brother Angular 2.

I introduced a lot of confusing concepts for beginners...

- Dependency injection
- Services, Factories, Providers
- etc..

Things I accomplished really well...

- Controllers
- Template syntax (Directives)

Good bye !
Angularjs

heig-vd

Meet AngularJS

Edit in JSFiddle

- JavaScript
- HTML
- Result

```
function ListController($scope) {
  $scope.reverse = true;
  $scope.orderBy = null;
  $scope.items = [
      { name: 'React' },
      { name: 'Angular' },
      { name: 'Vue.js' },
      { name: 'Polymer' },
      { name: 'Riot' },
      { name: 'Knockout' },
      { name: 'Ember' },
  ]

  $scope.sort = function() {
    $scope.reverse = !$scope.reverse;
    $scope.orderBy = 'name';
  }
}
```

```
<div ng-app ng-controller="ListController">
  <button ng-click="sort()">sort</button>
  <ul>
    <li ng-repeat="item in items | orderBy: orderBy: reverse">
```

heig-vd

Meet React

Hi, I'm React

I'll help you build complex interactive user interfaces.

I'm more a component-based **library** than a framework. This means that you can build your app the way you like. Build your views in a declarative way and let me do the rest.

I really, really like **Javascript** and I hope you feel the same way !

Cheers,
React

heig-vd

Meet React

Edit in JSFiddle

- React
- Result

```
class TodoApp extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
        sort: false,
        sortDirection: -1,
        items: [
          { name: 'React' },
          { name: 'Angular' },
          { name: 'Vue.js' },
          { name: 'Polymer' },
          { name: 'Riot' },
          { name: 'Knockout' },
          { name: 'Ember' },
      ]
    }
  }

  getSortedItems = () => {
        const { sort, sortDirection, items } = this.state
    if (sort) {
        const sorted = items.sort((a, b) => a.name.localeCompare(b.name))
        return sortDirection > 0 ? sorted : sorted.reverse();
      }
      return items;
```

heig-vd

Meet Angular 2+

Hello, I'm Angular 2

I'm a component-base **framework**.

I do things differently compared with React — Instead of letting you mess up with your codebase when having too much freedom, I enforce a specific code structure.

You won't get lost in the javascript eco-system because I'll give you all tools you need (Routing, Data fetching, Forms, etc...) and teach you how to use them.

I'm serious about code quality and robustness. Thank God for such a beautiful language: **Typescript**!

Best regards,
Angular 2

heig-vd

Meet Vue.js

Hi, I'm Vue

I'm a javascript library for building user interfaces. Combined with some other tools, I become a framework.

You might be like: *“Come on ! Yet another Javascript framework”*. But I swear you won't be disappointed !

I've been learning from mistakes and success of others (especially React and Angular). Now, I'm Harder, Better, Faster, Stronger

heig-vd

Meet Vue.js

Edit in JSFiddle

- Vue
- HTML
- Result

```
new Vue({
  el: "#app",
  data: {
    sort: false,
    sortDirection: -1,
    items: [
      { name: 'React' },
      { name: 'Angular' },
      { name: 'Vue.js' },
      { name: 'Polymer' },
      { name: 'Riot' },
      { name: 'Knockout' },
      { name: 'Ember' },
    ]
  },
  computed: {
                sortedItems: function() {
      console.log({ direction: this.sortDirection })
      if (this.sort) {
        const sorted = this.items.sort((a, b) => a.name.localeCompare(b.name))
        return this.sortDirection > 0 ? sorted : sorted.reverse();
      }
      return this.items;
    `
```

heig-vd

Get started with React

First, create a component – It can be an ES6 `class` or a `function` like below.

```
import React from 'react'
import ReactDOM from 'react-dom'

function App() {
  return (
    <div>
      <h1>You think it's HTML ?</h1>
      <p>No, It's Javascript</p>
    </div>
  );
}
```

Then attach your component to a DOM node

```
const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Only Javascript

Use JSX to describe your interface

```
function App() {
  return (
    <div className="App">
      <h1>You think it's HTML ?</h1>
      <p>No, It's Javascript</p>
    </div>
  )
}
```

JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

```
const e = React.createElement

function App() {
  return e('div', { className: "App" },
    e('h1', null, "You think it's HTML ?"),
    e('p', null, "No, It's Javascript"),
  )
}
```

Embedding Expressions in JSX

You can put any valid Javascript expression inside the **curly braces** in JSX.
Because JSX is just Javascript

```
const name = 'Paul Nta';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Embedding Expressions in JSX

JSX expression are **function calls** that return objects — we call them "Elements"

```
const element = React.createElement(
  'h1',
  { className: 'greeting' },
  'Hello, Paul Nta'
);
```

```
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, Nta!'
  }
};
```

Embedding Expressions in JSX

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Paul',
  lastName: 'Nta'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

JSX is an Expression

You can use JSX inside of `if` statements and `for` loops, assign it to variables, accept it as arguments, and return it from functions:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Attributes with JSX

Use quotes to pass string literals as attributes

```
const element = <input tabIndex="0"></input>;
```

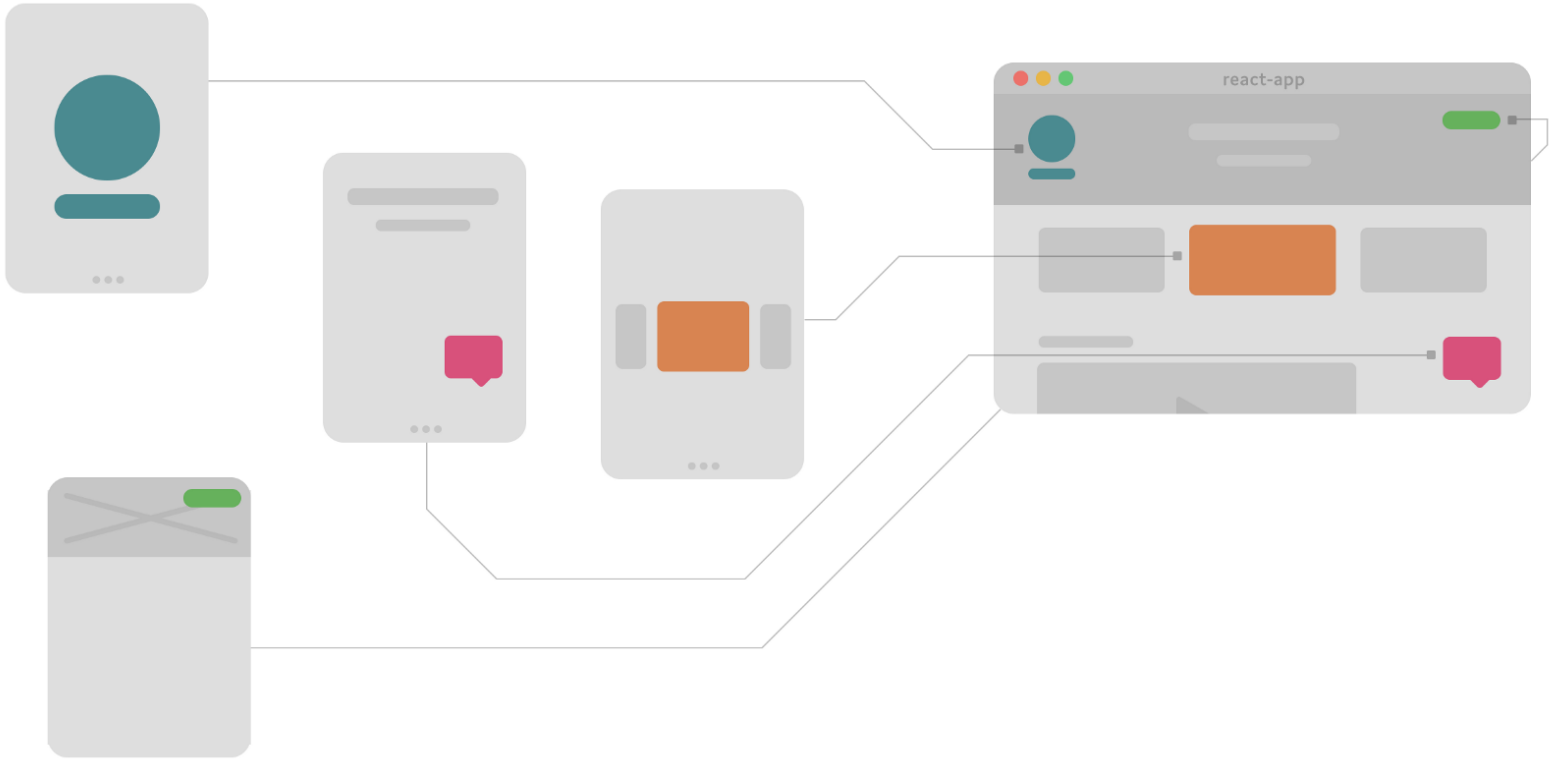
Use curly braces to embed a JavaScript expression in an attribute

```
const element = <img src={user.avatarUrl}></img>;
```

> **Warning:** Since JSX is closer to JavaScript than to HTML, React DOM uses camelCase property naming convention instead of HTML attribute names. For example, `class` becomes `className` in JSX, and `tabindex` becomes `tabIndex`.

Components and Props

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.



https://bitsrc.io/

Components and Props

The simplest way to define a component is to write a JavaScript function

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

You can also use an ES6 class to define a component. Classes have some additional features that we will discuss later

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Use it like this

```
const element = <Welcome name="Paul Nta" />
```

Component State

State is similar to props, but it is private and fully controlled by the component. Components defined as a class can have a locale state

```
class TodoApp extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
        items: [
          { id: 0, text: "Learn JavaScript", done: false },
          { id: 1, text: "Learn React", done: false },
          { id: 2, text: "Build something awesome", done: true }
      ]
    }
  }

  render() {
    return (
      <div>
        <h2>{this.state.items.length} todos</h2>
      </div>
    )
  }
}
```

Component State

```
class TodoApp extends React.Component {
  /*...*/
  render() {
      const { items } = this.state
      return (
        <div>
          <h2>Todos:</h2>
          <ol>
          {items.map(item => (
            <li key={item.id}>
              <label>
                <input type="checkbox" checked={item.done} />
                <span>{item.text}</span>
              </label>
            </li>
          ))}
          </ol>
        </div>
      )
    }
}
```

Edit on CodeSandbox

Updating Component State

Do Not Modify State Directly. The only way to update a component's state is to use `this.setState()`.

This will not re-render a component

```
// Wrong
this.state.items = []

this.state.items.push({ text: '...'})
```

Instead use `setState(obj)` or `setState(function)`

```
// Correct
this.setState({ items: [] })

this.setState((state, props) => ({
  items: [...state.items, newItem]
}))
```

Updating Component State

```
class TodoApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      items: [/*...*/]
    };
  }

  handleRemove = (id) => {
    const { items } = this.state
    this.setState({
      items: items.filter(item => item.id !== id)
    })
  }

  render() {/*...*/}
}
```

Handling Events

```
// class TodoApp.render()
{items.map(item => (
  /*...*/
    <button onClick={() => this.handleRemove(item.id)}>
  /*...*/
))}
```

- React events are named using camelCase, rather than lowercase
- With JSX you pass a function as the event handler
- React normalizes events so that they have consistent properties across different browsers. See supported events

Composing components

```
// components/TodoItem.js
import React from "react";

function TodoItem(props) {
  const { id, text, done, onRemove } = props;
  const className = done ? "done" : "";
  return (
    <li>
      <label>
        <input type="checkbox" checked={done} disabled readOnly />
        <span className={className}>{text}</span>
        <button onClick={onRemove}>X</button>
      </label>
    </li>
  );
}

export default TodoItem;
```

Composing components

```
// index.js
class TodoApp extends React.Component {
  /*...*/
  render() {
    return (
      <div>
        <h2>Todos:</h2>
        <ol>
          {this.state.items.map(item => (
            <TodoItem
              key={item.id}
              text={item.text}
              done={item.done}
              onRemove={() => this.handleRemove(item.id)}
            />
          ))}
        </ol>
      </div>
    );
  }
}
```

Edit on CodeSandbox

Method binding

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary
    // to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Method binding alternative #1

```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

Method binding alternative #2

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    // This syntax ensures `this` is bound within handleClick
    return (
      <button onClick={(e) => this.handleClick(e)}>
        Click me
      </button>
    );
  }
}
```

> The problem with this syntax is that a different callback is created each time the `LoggingButton` renders.

Method binding explained

```
class Foo {
  constructor(name){
    this.name = name
  }
  display() {
    console.log(this.name);
  }
}

const foo = new Foo('paulnta');
foo.display(); // paulnta
```

The assignment operation below simulates loss of context similar to passing a handler as a callback in a React Component

```
const display = foo.display;
display(); // TypeError: this is undefined
```

This is why we need to bind event handlers in Class Components in React

Lists and keys

> Keys should be given to the elements inside the array to give the elements a stable identity

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
const numbers = [1, 2, 3, 4, 5]
const element = <NumberList numbers={numbers} />
```

- keys Must Only Be **Unique** Among **Siblings**
- avoid using array indexes as key. Why?