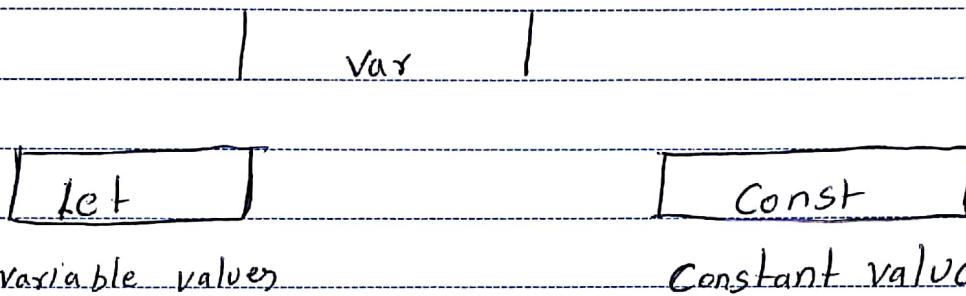


Next - Gen JavaScript

let & Const

They are different varieties of creating variables



var key words works but you're highly encouraged to use let, const

jsbin.com

```
var myName = 'max';
Console.log(myName); // max
```

```
const myName = 'max';
Console.log(myName) // max
```

```
myName = 'Manu';
Console.log(myName); // manu
```

```
myName = "Manu";
console.log(myName) // Manu
```

Arrow Functions

```
function myFunc() {  
    // normal syntax  
}
```

now an arrow function looks like this

```
const myFunc = () => {
```

3

no more issues with the this keyword;

```
function PointMyName(name) {  
    console.log(name);  
}  
PointMyName('max');
```

```
const PrintMyName = (name) => {  
    console.log(name);  
}  
PrintMyName('max');
```

Exports & imports (modules)

Person.js Utility.js

```
const Person = {  
  name: 'max'  
}
```

```
export default person
```

```
export const clear = () => {};
```

```
export const baseData = 10;
```

imports default
and only export
of the filename
in the receiving file
is up to you

app.js

```
import person from './person.js'  
import prs from './Person.js'
```

```
import {baseData} from './utility.js'  
import {clear} from './utility.js'
```

named export

```
{ smith } from  
{ smith as smitho } from  
{ * as bundles } from
```

Classes

Classes are blueprints for objects.

```
class Person {  
    name = 'max' // property  
    call = () => {} // method  
}
```

Class will be instantiated like below

Usage const myPerson = new Person()

myPerson.call()

Console.log (myPerson.name)

inheritance

Prototype anyone Class Person extends Master

→ Comes here
class person ^{extends Human} ~~x~~ inheritance

Constructor () {
 super();
 this.name = 'max';
 this.gender = 'female';

Print myName () {
 console.log(this.name);
}

Const Person = new Person();
Person.printMyName();

Class Human {
 Constructor () {

this.gender = 'male';
 } ~~constructor~~

Print Gender () {
 console.log(this.gender);
}

3

Classes, Properties & methods

Different ways to initialization Properties & methods

Properties are like variables attached to classes/objects

methods are like "functions" attached to classes/objects

ES6
Constructors {

`this.myProperty='value'`

ES6

`mymethod() { ... }`

}

ES7

`myProperty='value'`

ES7

`myMethod = () => { ... }`

Spread & Rest Operators



spread

used to split up array elements OR object properties

```
const newArray = [... oldArray, 1, 2]
```

```
const newObject = { ... oldObject, newProp: 5 }
```

rest

used to merge a list of function arguments into an array

```
function sortArgs(... args) {
    return args.sort();
}
```

Destructuring

easily extract array elements or objects properties and
store them in variables

// Array destructuring //

```
[a, b] = ['Hello', 'max']
```

```
console.log(a)
```

```
console.log(b)
```

// Object destructuring //

```
{name} = {name: 'max', age: 28}
```

```
Console.log(name) // max
```

```
Console.log(age) // undefined
```

The Basics

All core React Concept

using a Build workflow

Recommended for SPAs and MPAs

- optimize code (increase performance app)
 - use next-Gen JS features (less error prone code) ES6/ES7
 - Be more productive
-
- use Dependency management Tool npm (or) Yarn
 - use Bundler Recommended: webpack
 - use Compiler (next-Gen JavaScript)
Babel + Presets
 - use Development server

→ Create-react-app

sudo npm install -g Create-react-app

Password:

installation - - - - -

→ Create-react-app react-complete-guide

installation - - - - -

cd react-complete-guide

npm start

opens a browser with a starting page

Create-react-app react-complete-guide

↳ Enter

Success!

Inside that directory, you can run several commands

npm start

starts the development server

npm run build

Bundles the app into static files for production

npm test

Starts the test runner,

npm run eject

Once everything installed run the below commands

* cd react-complete-guide

* npm start

Folder structure

node_modules folder consist of all dependency injectors

public

↳ index.html

src

↳ js files

index.css - body CSS

app.css - only for app

APP.js - where we write JS

Understanding Component

src

↳ APP.JS

```
import React, {Component} from 'react';
import './APP.css';
```

Class App extends Component {

 render () {

 return (

 <div className="APP">

 <h1> Hi, I'm a React APP </h1>

 </div>

);

 }

 export default APP;

} JSX

Index.JS

```
import React from 'react';
```

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

```
import './index.css';
```

```
import APP from './APP';
```

```
import registerServiceWorker from './registerServiceWorker';
```

```
ReactDOM.render(<APP />, document.getElementById('root'));
registerServiceWorker();
```

Understanding JSX

APP.js

Class APP extends Component {

 render() {

 return React.createElement('div', null, 'hi', 'Hello');

}

}

This will be interpreted
as text



If you want to render
this as an element.

React.createElement('hi', null, 'Hello')

null → { className: 'APP' }

The above process is very hard use & complicated so
this is why we use JSX

return {

 <div className="APP">

 <hi> Hi, i'm a React APP </hi>

 </div>

}

Creating a functional component

we are trying to render a person information to the app
let's Create Person folder in src & file person.js

Person.js

ES6

```
import React from 'react';
const person = () => {
  return <p> I'm a Person! </p>
}
```

export default person;

// save the file

APP.js

```
import Person from './Person/Person';
return (
  <Person />
)
```

Working with Components & Re-Using Them

The person component that we created is reusable & Configurable.

Reusable means we can use it anywhere again & again in the app.

~~Configuring~~

Outputting Dynamic Content

If you have any dynamic content which we want to display normally, we need to wrap it in a single ^{copy} bracket

Person.js

```
const person = () => {
```

```
    return <p> I am a person and I am {  
        floor(Math.random() * 30)} years old  
</p>
```

```
}
```

we can write only one line expressions only not Java ^{Script} classes or function

Working the Props

<Person name="Max" age="28" />

<Person name="Manu" age="29" /> myhobbies:Racing<Person>

<Person name="Stephanie" age="26" />

now we need to update our Person.js

const Person = (props) => {
 ↗ attributes of your component }

return <p> I am {props.name} and I am {props.age} years old </p>

}

// check the output

now we have reusable component which we can use anywhere with dynamic properties.

const Person = (props) => {

return <div>

<p> I am {props.name} and I am {props.age} years old </p>

<p> {props.children} </p>

</div>

}

Understanding & Using State

Some time we don't want to get information from outside but you want it inside the component.

if you want to switch names when you click a button.

<button> switch now </button>

Right now it is hard coded in our HTML, we don't want it.

so we can store information inside a variable.

Class has properties we can use them.

Class APP extends Components {

state = {

persons: [

{name: 'max', age: 28}

{name: 'manu', age: 29}

{name: 'Stepanie', age: 26}

]

render() {

return {

<div><h1>Name=APP</h1>

<h1>Hi, I'm a React APP</h1>

<Person name={this.state.persons[0].name} age={this.state.persons[0].age}>

<Person name={this.state.persons[1].name} age={this.state.persons[1].age}>

</div>

state is managed from inside the component and it can be used only for the component using extends.

State can be changed, if it changes it will re-render that particular <Person>

Handling Events with methods

```
<button onClick={this.switchNameHandler}>switch name</button>
```

we need to create a method inside the class which is a function

```
switchNameHandler = () => {
```

```
    console.log("was clicked");
```

```
}
```

~~Manip~~

Manipulating The state

switchNameHandler = () => {

 won't work // This.state.Person[0].name = 'maximilian';

 this.setState({ Persons: [

 { name: 'maximiliam', age: 28 },

 { name: 'manu', age: 29 },

 { name: 'stephanie', age: 27 }]

] })

}

setState allow us to change state.

Function Component (stateless) vs Class Component (stateful Components)

~~Function~~
we can't use state in the Function, where we return some JSX code.

Passing Method References Between Components

Let's say we want to call the switch name handler which I recognized all changes.

Also that say when clicking any paragraph you're the Paragraph which can say it contains name or age

Ofcourse we can add onclick handler but now what we can't call other class event listener to file it in a different file in a different class.

Well we can actually pass a reference to this handler as a property to our component and this is no fancy hack.

```
<Person name={this.state.person[1].name}  
age={this.state.person[1].age}  
click={this.switchNameHandler}>  
</Person>
```

person.js

```
<P onClick={props.click}>
```

You can pass methods also as props so that you can call a method which might change to state in another.

maybe we all don't want to pass a value to our function.

we can switch name handler should receive the newName
so here where I hard coded maximilian

```
switchNameHandler = (newName) => {
```

Person: {

```
{name: newName, age: 26}
```

} }

```
<button onClick={this.switchNameHandler.bind(this,  
"maximilian")}> switchName </button>
```

Also change it for Person Prop.

<Person

click={this.switchNameHandler.bind(this, 'max')} >

Other way to write.

~~intentional~~
unhoovered arrow function



onClick={() => this.switchNameHandler('maximilian!')}

Adding Two Way Binding

Let's say in the Person Component here we actually also have a number element a normal input element which is of type text and :

In the person component we will have normal input element

<input type="text" onChange={props.changed} />

APP.55

Let add a new handler to the app.55

100

nameChangedHandler = (event) => {

this.setState({

Person: {

{ name: 'max', age: 28 },

{ name: event.target.value, age: 29 },

{ name: 'Stephanie', age: 29 }

})

}

<Person

changed={this.nameChangedHandler} > <Person>

if you want we show the current value in the input

<input type="text" onChange={props.changed} value={props.name}

you will get a warning in console ignore

Adding styling with style sheets

There are two ways of styling Components.

Create a Person.css file to your Person Folder.

• Person {

width: 60%;

margin: 16 auto;

border: 1px solid #ccc;

box-shadow: 0 2px 3px #ccc;

padding: 16px;

text-align: center;

}

we need to import our css file to the Person.js

Add the className to the div of the Person Components.

working ~~with~~ with inline styles.

APP.55

inside the render create a new constant which
a.i name style

```
render () {  
  const style = {  
    backgroundColor: 'white',  
    font: 'inherit',  
    border: '1px solid blue',  
    padding: '8px',  
    cursor: 'pointer'  
  }  
}
```

```
return (  
  <button style={style}>
```

Rendering Lists & Conditional Content

Creating truly dynamic apps

Outputting the content conditionally, lets wrap all the person tags in a div so that we can show ~~not~~ them in a group.

change the method of the button tag

```
<button style={...} onClick={this.togglePersonsHandler}>  
    switch Name</button>
```

~~now we will~~

we need to create a method `togglePersonsHandler` before `render()`.

`togglePersonsHandler() = () => {`

3

11

71

now go to the state & add a property `showPersons: false`
initial value:

Go to div enclose the whole div in {} braces

3

`this.state.showPersons == true?`

`<div>`

3 `<div>: null`

Outputting Lists

Write now we manually hard coded the Persons information in to our JSX syntax. This is not flexible.

We need to convert persons array into something else using .map

Persons = [

<div>

{ this.state.Persons.map((Person) => {

return <Person name={Person.name} age={Person.age}>

)},

)

The above code will output our Persons array we declared in our state. We can remove the other hard coded persons components

Lists & state

Lets see how to manipulate this array we are outputting inside out person components we already have a click property

we will create a new handler

deletePersonHandler = () => {
 personIndex

const Persons = this.state.Persons; // fetching all persons
Persons.splice(personIndex, 1); // simple removes one element
in an array
this.setState({Persons: Persons});

}

— By —

Persons =
<div
 * this.state.Persons.map((person, index) => {
 return <Person
 C()=>
 click = * this.deletePersonHandler(index)
 name = * person.name
 age = * person.age
 * />
 }
</div>
*;

when you pass two parameters put it in a
brackets

Updating state immutably

The above process is a bad practice because in JS objects & arrays are reference type. By doing above method ; actually get original state and mutating ~~the~~ it which is bad.

The best practices is to create copy, How to do it is adding ~~the~~ slice method simply copies an array put it in new one.

```
const persons = this.state.persons.slice();
```

ES6 way

```
const persons = [...this.state.persons];
```

Lists & Keys

Key prop is very important when rendering lists to our Dom.

We need to add key property to track the individual element history, which element change which didn't?

Person.js

<Person

~~key = person.id~~

key = { person.id }

1>

— update state with id hard code —

state = {

persons: [

{ id: 'asfa1', name: 'max', age: 28 },
{ id: 'vasdf1', name: 'manu', age: 20 },
{ id: 'asdf11', name: 'stephanie', age: 29 }

]

{

Flexible List

<Person

After the key attribute

changed = { } => this.nameChangeHandler(event, person.id) }
/>

nameChangeHandler = (event, id) => {

const ^{PersonIndex} person = this.state.Persons.findIndex(p => {
return p.id === id;});

const person = this.state.Persons[personIndex];
↓ best practice replace with below.

{ ...this.state.Persons[personIndex] } }

Person.name = event.target.value;

const persons = [...this.state.Persons];

persons[personIndex] = person;

this.setState({ persons: persons });

Styling React Components & Elements

Q

Setting styles dynamically

Let update the button style if already showing persons
background would be red if not green.

render () {

```
const style = {  
    backgroundColor: 'green',  
    color: 'white',  
    font: 'inherit',  
    border: '1px solid blue',  
    padding: '8px',  
    cursor: 'pointer'  
};
```

if (this.state.showPersons) {

persons = (

```
        J, style.backgroundColor = 'red');
```

}

setting Class names Dynamically

APP.css

• red {

color: red;

}

• bold {

font-weight: bold;

}

now i want to change the color & font-weight dynamically of a paragraph when the elements ^{in persons} are less than 2.

APP.js

Create new variable before the return

let classes = ['red', 'bold'].join(' ');

in the p tag

<p className={classes} > This is really working 21P>

lets classes = [] ; Dynamically setting

if (this.states.persons.length <= 2) {
 classes.push('red');
}

if (this.states.persons.length <= 1) {
 classes.push('bold');
}

<P className={classes.join(' ')}> This is really working </P>

Adding and using Radium

Installing package to use media queries & hover effects in
inlinelinks

npm install --save radium



First import it to app.js

import Radium from 'radium'

in the bottom change export line

export default Radium(APP); // higher order component

render () {

const style = {

keep the remaining properties

' :hover' : {

backgroundColor: 'lightgreen'

color: 'black'

}

}

if (this.state.person) {

Persons = (

);

style[': hover'] = {

backgroundColor: ~~lightgreen~~ ~~lightred~~ 'Salmon';

color: 'black'

}

}

Using Radium for media Queries

Person.js

- person = (props) =>

const style = {

'@media (min-width: 500px)': {
width: '450px'}

},
};

return (

<div className="person" style={style}>

</div>

)

export default Radium(Person);

APP ->
import Radium
{styleRoot}
in 'radium'
<style Root>
<div className="app">
</div>
<style Root>

To

React image is an `` tag replacement for React.js featuring Preloader and multiple image fallback support

`npm install react-image --save`

``