Next - Gen JavaScript

let & Const

They are different varites of creating variables

| var |

let		const
variable values		Constant value

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

js.bin.com

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

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

```
Const myName = 'max'
Console.log(myName) //max

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

Arrow Functions

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

now an arrow function looks like this

```
Const myFnc = () => {

}
```

no more issues with the this keyword;

```
function PrintMyName(name) {
    Console.log(name);
}
PrintMyName('max');
```

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

Exports & imports (modules)

person.js

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

export default person
```

utility.js

```
export Const clean = () => {...}

export Const baseData = 10;
```

app.js

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

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

import {baseData} from './utility.js'
import {clean} 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 instanciated like below

Usage

```
Const myPerson = newPerson()
myPerson.Call()
Console.log (myperson.name)
```

inheritance

Prototype anyone Class Person extends Master

inheritance

```
Comes here
class Person extends Human {

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

    PrintmyName() {
        console.log(this.name);
    }

}

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

class Human {
    Constructor() {
        this.gender = 'male';
    }

    PrintGender() {
        Console.log(this.gender);
    }
}
```

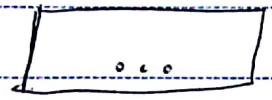
Classes, Properties & methods

Different ways to intialization Properties & methods

Properties are like variables attached to classes/objects	methods are like "functions attached to classes/objects

ES6
Constructor() {
this.myProperty = 'value'
}
 	
ES7
myProperty = 'value'
	ES6
	myMethod() {...}
	
	ES7
	myMethod = () => {...}

Spread & Rest Operators



Spreed — 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
Cosole.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 Pron code) ES6/ES5
- Be more Productive

- use Dependency management Tool npm (or) Yarn
- use Bundler Recommended : webPack
- use comiler (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-guid

npm start

opens a brower 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 comands

*/ cd • react-complete-guide

*/ npm start

Folder structure

node_modules folder consist of all dependancy 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 intrepted as text

if you want to render this as an element.

React.CreateElement ('hi', null, mess)

null → { ClassName: 'APP' }

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

```
return (
    <div ClassName = "APP" >
        <h1> Hi, i'm a React APP </h1>
    </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

Esl

```
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 any where again & again in the app.

Outputting Dynamic content

if you have any dinamic content which we want to display normaly. we need to rap it in a single curly bracket

Person.js

```
Const person=() => {
    return <P> I am a person and I am {math.floor(math.random() x30} years old
    </P>
}
```

we can write only one line expressions only not JavaScript classes or function

working the Props

```
<Person   name = "Max" age = "28" />
<Person   name = "Manu" age = "29" /> myhobbis: Racing </Person
<Person   name = "Stephanie" age = "26" />
```

now we need to update our Person.js

attributs of your component

```
Const Person = (Props) => {

    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 any where with a 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 htms, we don't want it,

So we can store information inside a variable.

State is an object that determines how the Component renders and behave. It is place where the data comes from.

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

        <Person  name={this.state.persons[0].Name} age={this.state.per
<Person name={this.state.person[1].name  age={this.state.person[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 Changed it will rerender that Perticular <Person>

Handling Events with methods

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

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

```
switchNameHandler = () => {

Console.log("was Clicked");
}
```

~~Manup~~

Manipulating the state

```
switchNameHandler = () => {
// dontwork this.state.person[0].name='maximulian';
    this.setState({ persons: [
        { name: 'maximiliam', age: 28},
        {name: 'manu', age: 29},
        {name: 'stephanie' age: 27}.
    ]})
}
```

setState allow us to change state.

Function (stateless) vs Class (stateful Components)

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

Passing Method References Between Components

Lets 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 othe class event listner to file it in a diffrent file ina diffrent Class.

Well we can actvally 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.

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

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

```
switchNameHandler = (newName) => {
    person: [
        {name: newName, age = 26 }
    ]
})
```

```
<button onclick={this.switchNamehandler.bind(this,
        "maximilien")}>switchName</button>
```

Also Change it for Person Prop

```
<Person
    click={this.switchNameHandler.bind(this, 'max')} />
```

Other way to write.

~~anonymous~~ unnamed or 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.JS

Let add a new handler to the app.js

```
nameChangedHandler = (event) => {
  this.setState({
    persons: [
      {name: 'max', age: 28},
      {name: event.target.value, age: 29},
      {name: 'Stephanie', age: 27}
    ]})
}
```

```
<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 consoler ignore

Adding styling with stylesheets

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 #eee;
    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 inline styles.

App.ss

inside the render creat 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

Outputing the content conditionaly, lets wrap all the person tags in a dive so that we an show them in a group.

Change the method of the button tag

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

we need to create a method toogglePersonsHandler before render().

togglePersonsHandler = () => {

}

now go to the state & adda property showPersons: false intial value:

Go to dive enclose the whole dive in { braces }

{
this.state.showPersons===true?

 : null
}

Outputting Lists

Write now we manualy 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 out put our Persons arry we declared in our State. we can remove the other hard coded persons componets

Lists & state

Lets see how to manuplate this array we are outputing inside out Person components we alread 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});
}
```

——— // ———

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

when yo pass two parameters put it in a brackets

Updating state immutably

The above process is a bad Practice because in JS object & arrays are reference type. By doing above method i actually get orignal state and mutating it which is bad.

The best practes is to Create copy, How to do it is adding slice method simply copys an array put it in new one.

```
Const  Persons = this.state.Persons.slice();
```

Esb 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 indival element History, which element change which didn't?

Person.js

```
<Person
    after the age attribute
    key={person.id}
/>
```

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

nameChangeHandler = (event, id) => {

const personIndex = 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

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.stet.showPersons) {
    Persons = C

    } ;
style.backgroundColor='red';
}
```

setting Class Names Dynamicaly

APP.css

```
.red {
  Color: red;
}
.bold {
  font-weight: bold;
}
```

now i want to Chang the color & font weight dinamically of a Pharagres when the elements, in Persons are less than 2.

APP.JS

Creat new variable before the return

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

in the p tag

```
<p className={Classes}> This is relly working 21P</p>
```

```
let classes = [];   Dynamicaly setting

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

```
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~~ 'Salmon',
      color : 'black
    }
}
```

Using Radium for media Queries

Person.js

```
- Person = (Props) => {
    Const style = {
      '@media (min-width : 500px)' : {
          width : '450px'
      }
    };
    return (
      < div Class Name = "person" style = {style}>

      <\div>
    )
```

```
export default Radium(Person);
```

app. js

```
import Radium
  '{styleRoot}
   in 'radium'

<style Root>
  <div Class Name="App">


   <\div>
  <\style Root>
```

To

React image is an <img> tag replacement for React.js featuring Preloader and multiple image fallback support

npm install react-image --save

