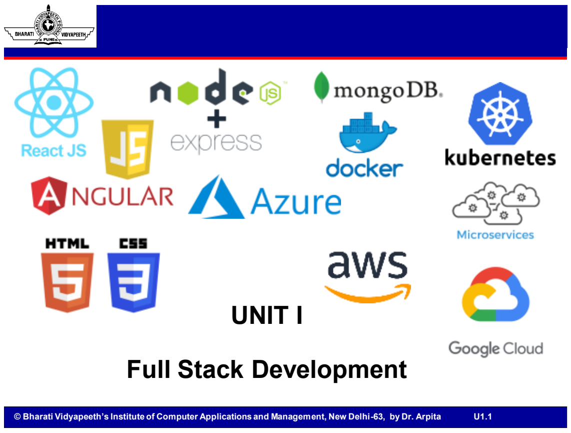
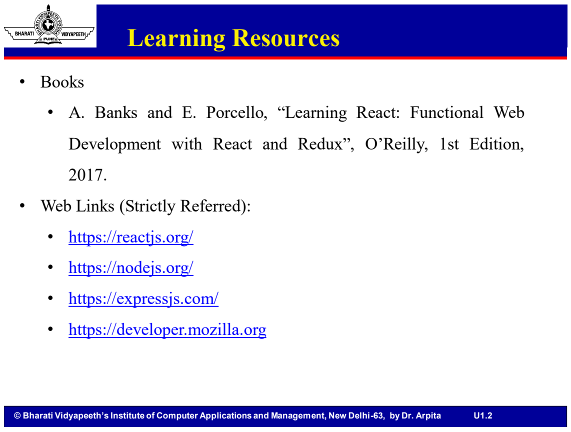

React JS
node js
express
mongoDB
docker
kubernetes
ANGULAR
Azure
Microservices
HTML
CSS
aws
Google Cloud

UNIT I

Full Stack Development

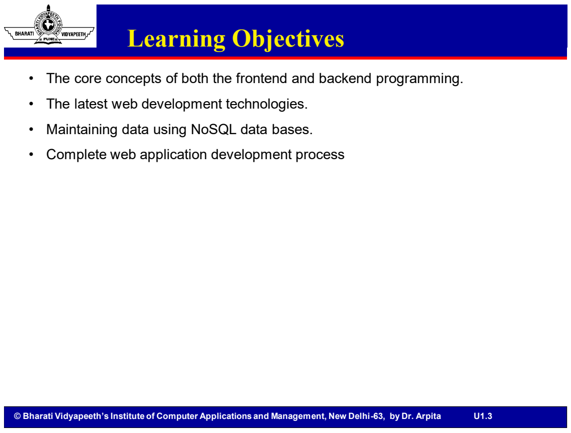
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita U1.1



Learning Resources

- Books
  - A. Banks and E. Porcello, "Learning React: Functional Web Development with React and Redux", O'Reilly, 1st Edition, 2017.
- Web Links (Strictly Referred):
  - https://reactjs.org/
  - https://nodejs.org/
  - https://expressjs.com/
  - https://developer.mozilla.org

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita U1.2



Learning Objectives

- The core concepts of both the frontend and backend programming.
- The latest web development technologies.
- Maintaining data using NoSQL data bases.
- Complete web application development process

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita U1.3

Course Outcome

- CO1: Relate the basics of Javascript (JS) and ReactJS
- CO2: Apply the concepts of props and State Management in React JS
- CO3: Examine Redux and Router with React JS
- CO4: Appraise Node JS environment and modular development
- CO5: Develop full stack applications using MongoDB

Overview

UNIT-1

- Introduction to React
  - Obstacles and Roadblocks
    - React Library, React Developer tools
  - Introduction to ES6
    - Declaring variables in ES6, Arrow Functions, ES6 Objects and Arrays, ES6 modules
  - Introduction to AJAX
- Pure React
  - Page setup, virtual DOM
  - React Element, React DOM, Constructing Elements with Data, React Components, DOM Rendering, First React Application using Create React App, React with JSX, React Element as JSX
- Props, State and Component Tree
  - Property Validation, Validating Props with createClass, Default Props, ES6 Classes and stateless functional components, React state management, State within the component tree, state vs. props, Forms in React

Overview (cont..)

UNIT-2

- Enhancing Components
  - Component Lifecycle, JavaScript library integration
  - Higher-Order Components, Managing state outside the react
  - Introduction to Flux
- Redux and Router
  - State, Actions, Reducers, The Store
  - Middleware
  - React Redux
  - React Router, Incorporating the router, Nesting Router, Router parameters
- JSON
  - Objects
  - Schema
- REST API
  - WRML, REST API Design
  - Identifier Design with URIs, Interaction Design with HTTP, Representation Design, Caching, Security

Overview (cont..)

UNIT-3

- Introduction to Angular
  - Angular architecture; introduction to components, component interaction and styles; templates, interpolation and directives; forms, user input, form validations; data binding and pipes; retrieving data using HTTP; Angular modules
- Node.js
  - Introduction, Features, Node.js Process Model
  - Environment Setup: Local Environment Setup, The Node.js Runtime, Installation of Node.js
  - Node.js Modules: Functions, Buffer, Module, Modules Types
  - Node Package Manager: Installing Modules using NPM, Global vs Local Installation, Attributes of Package.js on, Updating packages, Mobile-first paradigm, Using twitter bootstrap on the notes application, Flexbox and CSS Grids
- File System: Synchronous vs Asynchronous, File operations
- Web Module: Creating Web Server, Web Application Architecture, Sending Requests, Handling http requests
- Express Framework: Overview, Installing Express, Request / Response Method, Cookies Management

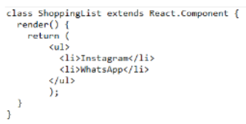
Overview (cont..)

UNIT-4

- MongoDB:
  - Introduction to NoSQL
  - Understanding MongoDB datatypes
  - Building MongoDB Environment (premise and cloud based)
  - Administering Databases and User accounts
  - Configuring Access Control, Managing Collections
  - connecting to MongoDB from Node.js
  - Accessing and Manipulating Databases and Collections
  - Manipulating MongoDB documents from Node.js
  - Understanding Query objects,
  - sorting and limiting result sets

Introduction

- React is a JavaScript library for building user interface
  - Declarative
  - Efficient
  - Flexible
- Complex UIs Problem
  - Components
    - small and isolated pieces of code
    - use to tell React what will beon the screen
    - On every data change, update and re-render the component.
  - React has a few different kinds of components
  - Component
    - Takes some parameters called Properties (props).
    - Returns a hierarchy of views via render method
  - Render()
    - Returns a description to the client for presentation
    - Returns a react element, a lightweight element
    - Use a special syntax called "JSX" which makes these structures easier to write



```
class ShoppingList extends React.Component {
  render() {
    return (
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
        </ul>
        );
  }
}
```

Introduction

•Node.js is an open-source and cross-platform JavaScript runtime environment to create all kinds of server-side tools and applications

•Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser
•JavaScript is a programming language that was created at Netscape as a scripting tool to manipulate web pages inside their browser

•Netscape was to sell Web Servers, called Netscape *LiveWire.*
  •create dynamic pages using server-side JavaScript (FAILURE)

•Web 2.0 applications (such as Flickr, Gmail, etc.) that showed the world what a modern experience on the web could be like

•2009: Node.js is born
•2010:
  •Socket.io: Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server.
  •Express: Fast, unopinionated, minimalist web framework for Node.js
•2011: Hapi: Build powerful, scalable applications, with minimal overhead

Introduction

•2013
  •First big blogging platform using Node.js: Ghost
  •Koa is born
•2014:
  •The Big Fork: io.js is a major fork of Node.js, with the goal of introducing ES6 support and moving faster
•2015:
  •The Node.js Foundation is born
  •IO.js is merged back into Node.js
  •npm introduces private modules
  •Node.js 4 (versions 1, 2 and 3 never previously released)
•2016:
  •The leftpad incident
  •Yarn is born
  •Node.js 6
•2017:
  •npm focuses more on security, HTTP/2
  •V8 introduces Node.js in its testing suite, officially making Node.js a target for the JS engine, in addition to Chrome

Introduction

- Express is the most popular Node web framework, and is the underlying library for a number of other popular Node web frameworks
  - Write handlers for requests with different HTTP verbs at different URL paths (routes).
  - Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
  - Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
  - Add additional request processing "middleware" at any point within the request handling pipeline.
- MongoDB
  - Open source No-SQL Database
  - Light weighted, useful for working with large sets of distributed data.
  - Contains a structure of data, composed of field (key) and value pairs
  - Similar to JSON, called Binary JSON.
  - Sets of these documents are called collections, like table in relational DB

Full Stack Development

- Software development has three main layers :
  - Frontend or Presentation layer
  - Backend or Business layer
  - Database layer
- MERN (React)
- MEAN (Angular)
- LAMP (Apache)
- LEMP (nginX)
- Full-Stack Elixir
- Full-Stack Python
- Full-Stack Django
- Full-Stack Java
- Full-Stack Ruby on Rails

Full Stack Development

- Full-Stack Developers can do the following:
  - Ability to code programs, web applications, or mobile applications.
  - Coordinate the development process with other developers and team members (including product managers, project managers, and C-level executives).
  - Troubleshoot technical issues at every layer.
  - Outline testing techniques for various applications.
  - Analyzing and debugging database queries.
  - Testing codes for app validation and compatibility across required devices for quality assurance.
  - Keeping a tab on important KPIs and taking initiatives as needed.
  - Laying blueprint for future requirements and communicating the same with upper management.

Full Stack Development

- Roles:
  - Tech Lead (or CTO)
  - Product Manager
  - Database Administrator
  - Senior Developer
- Benefits:
  - Delegate Everything Technical
  - Make the Team Flexible
  - Cost-Effectiveness

Full Stack Development

- Skill Set and Qualification required:
  - Front-end fundamentals
  - Server-side fundamentals
  - User Experience design
  - Database architecture and design
  - Business Logic
  - Project Management
  - Multitasking
  - Agile Development
  - Independence

Introduction

Wikipedia

• React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.

Microsoft

• React.js is the most popular front-end JavaScript framework. Developers use JSX, a combination of HTML and JavaScript, to create views in a natural way. Developers can also create components for blocks that can be reused across their applications. This module introduces React and the core skills developers need to use this powerful framework.

Obstacles and Roadblocks

- React Is a Library
  - Small
  - Used for one part of job
  - Restricted than traditional JavaScript
  - New tools emerged and old are cast aside
- New ECMAScript Syntax
  - The ECMA used to release specifications infrequently
- Popularity of Functional JavaScript
  - React emphasizes functional programming over object-oriented programming
- JavaScript Tooling Fatigue
  - Conventional JavaScript added into your source page
  - In react you have to built JavaScript

React Library

- Routing and Navigation in Reactjs
  - React Router : a group of navigational components synchronizes the components of the UI with browsers address
- React CLIs and Boilerplates
  - Create-React-App: Create-React-App is a CLI tool that requires no building configuration
- React UI Component Libraries
  - Ant-design: a consolidated development framework of NPM, Webpack, Babel, Dora, and DVA
  - Tailwind UI: utility classes (in-built classes) instead of inbuilt components that relieves you from overriding styles
  - Semantic UI React
  - React Bootstrap
  - Fabric React
  - Styled components
  - React DnD

React Library

- Notable UI libraries
  - OnsenUI
  - Rebass
  - Material UI
- Animation Libraries
  - React-motion
  - Animated (React Native)
  - React Spinner
- Form Libraries
  - React Hook Form

React Library

- Code formatting
  - ES-Lint
  - React Intl
  - Prettier
- Testing
  - Enzyme
- State Management
  - Redux
  - Mobx
- Augment Reality/ Virtual Reality (AR/ VR)
  - React 360
  - Viro React

React Developer Tools

- react-detector
  - react-detector is a Chrome extension that lets you know which websites are using React and which are not.
- show-me-the-react
  - This is another tool, available for Firefox and Chrome, that detects React as you browse the internet.
- React Developer Tools
  - This is a plugin that can extend the functionality of the browser's developer tools. It creates a new tab in the developer tools where you can view React elements.

Source: https://youteam.io/blog/full-stack-software-developer-hiring/

Introduction of ES6

- ECMA Script 6 (ES6):
  - ECMA Script 2015
  - based on specification and standardization by ECMA International Company
- ES6 is the ECMA script programming language
  - used to create the standards for JavaScript language such that it can bring multiple independent implementations.
- ES6 converted into the production server code, bundled, and minifies with webpack using Babel compiler.

Introduction of ES6

- const: variable that cannot be changed.
- let: JavaScript now has lexical variable scoping.

```
var topic = "JavaScript"

if (topic) {
  var topic = "React"
  console.log('block', topic)     // block React
}

console.log('global', topic)      // global React
```

```
var topic = "JavaScript"

if (topic) {
  let topic = "React"
  console.log('block', topic)     // React
}

console.log('global', topic)      // JavaScript
```

- Template Strings: an alternative to string concatenation
- Traditional string concatenation uses plus signs:

```
console.log(lastName + ", " + firstName + " " + middleName)
```

- With a template, we can create one string and insert the variable values by surrounding them with ${ }:

```
console.log(`${lastName}, ${firstName} ${middleName}`)
```

Introduction of ES6

- Default Parameters: a value is not provided for the argument, the default value will be used

```
function logActivity(name="Shane McConkey", activity="skiing") {
  console.log( `${name} loves ${activity}` )
}
```

```
var defaultPerson = {
    name: {
        first: "Shane",
        last: "McConkey"
    },
    favActivity: "skiing"
}

function logActivity(p=defaultPerson) {
    console.log(`${p.name.first} loves ${p.favActivity}`)
}
```

Introduction of ES6

- Arrow Functions: create functions without using the function keyword

As a traditional function

```
var lordify = function(firstname) {
  return `${firstname} of Canterbury`
}

console.log( lordify("Dale") )      // Dale of Canterbury
console.log( lordify("Daryle") )    // Daryle of Canterbury
```

As an arrow function

```
var lordify = firstname => `${firstname} of Canterbury`
```

```
// New
var _lordify = (firstName, land) => {

  if (!firstName) {
    throw new Error('A firstName is required to lordify')
  }

  if (!land) {
    throw new Error('A lord must have a land')
  }

  return `${firstName} of ${land}`
}
```

- Less Code
- Optimized
- Faster than traditional function

Introduction of ES6

- Arrow Functions: create functions without using the function keyword

```
var tahoe = {
  resorts: ["Kirkwood","Squaw","Alpine","Heavenly","Northstar"],
  print: function(delay=1000) {

    setTimeout(function() {
      console.log(this.resorts.join(","))
    }, delay)

  }
}

tahoe.print() // Cannot read property 'join' of undefined
```

```
var tahoe = {
  resorts: ["Kirkwood","Squaw","Alpine","Heavenly","Northstar"],
  print: function(delay=1000) {

    setTimeout(() => {
      console.log(this.resorts.join(","))
    }, delay)

  }
}

tahoe.print() // Kirkwood, Squaw, Alpine, Heavenly, Northstar
```

```
var tahoe = {
  resorts: ["Kirkwood","Squaw","Alpine","Heavenly","Northstar"],
  print: (delay=1000) => {

    setTimeout(() => {
      console.log(this.resorts.join(","))
    }, delay)

  }

}

tahoe.print() // Cannot read property resorts of undefined
```

Introduction of ES6

- Transpiling ES6
  - all web browsers support ES6
  - Transpiling is a process to convert ES6 code to ES5 before running it in the browser.
  - In past any update in JavaScript, force to wait till weeks or months until the browser update themselves to adapt the new version of JavaScript.
  - Transpiling is not compiling: our code isn't compiled to binary
  - transpiled into syntax that can be interpreted by a wider range of browsers
  - JavaScript now has source code, meaning that there will be some files that belong to your project that don't run in the browser

Introduction of ES6

- Transpiling Example

ES6 code before Babel transpiling

```
const add = (x=5, y=10) => console.log(x+y);
```

After we run the transpiler on this code, here is what the output will look like:

```
"use strict";

var add = function add() {
    var x = arguments.length <= 0 || arguments[0] === undefined ?
        5 : arguments[0];
    var y = arguments.length <= 1 || arguments[1] === undefined ?
        10 : arguments[1];
    return console.log(x + y);
};
```

- transpiler added a "use strict" declaration to run in strict mode

ES6 Objects and Arrays

- JavaScript data types and data structures
  - Dynamic typing
    - JavaScript is a loosely typed and dynamic language
      - Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types

```
let foo = 42;    // foo is now a number
foo     = 'bar'; // foo is now a string
foo     = true;  // foo is now a boolean
```

ES6 Objects and Arrays

- JavaScript data types and data structures
  - JavaScript types
    - Primitive values (immutable datum represented directly at the lowest level of the language)
      - Boolean type
      - Null type
      - Undefined type
      - Numeric type
        - » Number type
        - » BigInt type
        - » NaN type
      - String type
    - Objects (collections of properties)

ES6 Objects and Arrays

- JavaScript data types and data structures
  - Object Type
    - is a value in memory which is possibly referenced by an identifier
    - can be seen as a collection of properties
    - a limited set of properties are initialized; then properties can be added and removed
    - Property values can be values of any type, including other objects, which enables building complex data structures
    - Properties are identified using key values
    - A key value is either a String value or a Symbol value.

ES6 Objects and Arrays

- Object Type
  - two types of object properties
    - data property
      - Associates a key with a value, and has the following attributes:

Attribute	Type	Description	Default value
[[Value]]	Any JavaScript type	The value retrieved by a get access of the property.	undefined
[[Writable]]	Boolean	If false, the property's [[Value]] cannot be changed.	false
[[Enumerable]]	Boolean	If true, the property will be enumerated in for...in loops.	false
[[Configurable]]	Boolean	If false, the property cannot be deleted, cannot be changed to an accessor property, and attributes other than [[Value]] and [[Writable]] cannot be changed.	false

ES6 Objects and Arrays

- Object Properties
  - Accessor property
    - Associates a key with one of two accessor functions (get and set) to retrieve or store a value.

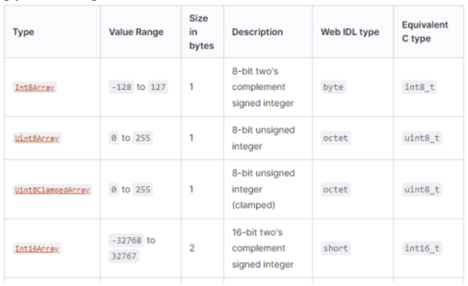
Attribute	Type	Description	Default value
[[Get]]	Function object or undefined	The function is called with an empty argument list and retrieves the property value whenever a get access to the value is performed. See also get.	undefined
[[Set]]	Function object or undefined	The function is called with an argument that contains the assigned value and is executed whenever a specified property is attempted to be changed. See also set.	undefined
[[Enumerable]]	Boolean	If true, the property will be enumerated in for...in loops.	false
[[Configurable]]	Boolean	If false, the property can't be deleted and can't be changed to a data property.	false

ES6 Objects and Arrays

- Indexed collections: Arrays and typed Arrays
  - Arrays are regular objects for which there is a particular relationship between integer-keyed properties and the length property
  - arrays inherit from Array.prototype, which provides to them a handful of convenient methods to manipulate arrays e.g. IndexOf(), Push() etc.
  - a perfect candidate to represent lists or sets
  - Typed Arrays are new to JavaScript with ECMAScript 2015, and present an array-like view of an underlying binary data buffer

ES6 Objects and Arrays

- Typed Arrays



Type	Value Range	Size in bytes	Description	Web IDL type	Equivalent C type
Int8Array	-128 to 127	1	8-bit two's complement signed integer	byte	int8_t
Uint8Array	0 to 255	1	8-bit unsigned integer	octet	uint8_t
Uint8ClampedArray	0 to 255	1	8-bit unsigned integer (clamped)	octet	uint8_t
Int16Array	-32768 to 32767	2	16-bit two's complement signed integer	short	int16_t

[Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures]

ES6 Objects and Arrays

- Keyed collections: Maps, Sets, WeakMaps, WeakSetsArrays
  - Set and WeakSet represent a set of objects, while Map and WeakMap associate a value to an object.
  - Map
    - A Map is a new collection object in JavaScript that functions like an object.

```
const map = new Map();
// Map(0) {}
```

```
map.set('name', 'john');
// Map(1) {"name" => "john"}
```

```
map.set('phone', 'iPhone');
// Map(2) {"name" => "john", "phone" => "iPhone"}

map.set('phone', 'iPhone');
// Map(2) {"name" => "john", "phone" => "iPhone"}
```

- Map is an iterable object
- remove properties from object

```
map.delete('phone');
// true

map.delete('fake');
// false
```

```
for (const item of map) {
  console.dir(item);
}

// Array(2) ["name", "john"]
// Array(2) ["phone", "Galaxy"]
```

ES6 Objects and Arrays

- WeakMaps :WeakMap originated from Map, so they are very similar to each other
  - Differences
    - The key must be an object

```
const John = { name: 'John' };
const weakMap = new WeakMap();

weakMap.set(John, 'student');
// WeakMap {{...} => "student"}

weakMap.set('john', 'student');
// Uncaught TypeError: Invalid value used as weak map key
```

- Not all methods from Map are supportive (support only)
  - delete()
  - get()
  - has()
  - set()

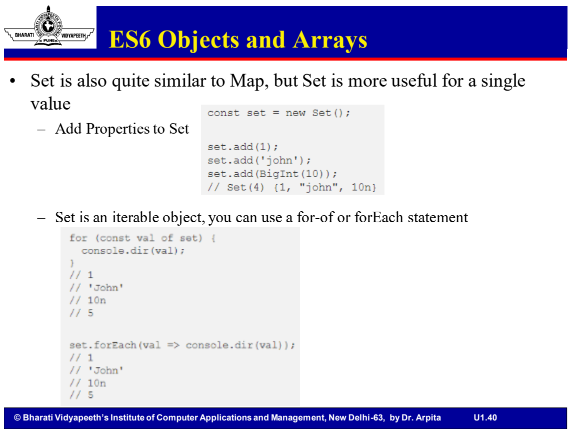
ES6 Objects and Arrays

- Set is also quite similar to Map, but Set is more useful for a single value
  - Add Properties to Set

```
const set = new Set();

set.add(1);
set.add('john');
set.add(BigInt(10));
// Set(4) {1, "john", 10n}
```

  - Set is an iterable object, you can use a for-of or forEach statement

```
for (const val of set) {
  console.dir(val);
}
// 1
// 'John'
// 10n
// 5

set.forEach(val => console.dir(val));
// 1
// 'John'
// 10n
// 5
```



ES6 Objects and Arrays

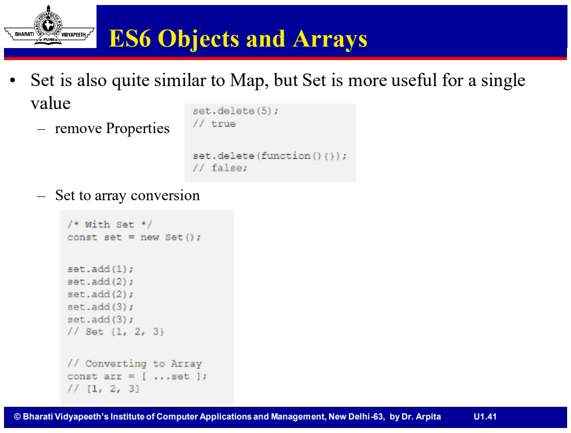
- Set is also quite similar to Map, but Set is more useful for a single value
  - Add Properties to Set

```
const set = new Set();

set.add(1);
set.add('john');
set.add(BigInt(10));
// Set(4) {1, "john", 10n}
```

  - Set is an iterable object, you can use a for-of or forEach statement

```
for (const val of set) {
  console.dir(val);
}
// 1
// 'John'
// 10n
// 5

set.forEach(val => console.dir(val));
// 1
// 'John'
// 10n
// 5
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita U1.40



ES6 Objects and Arrays

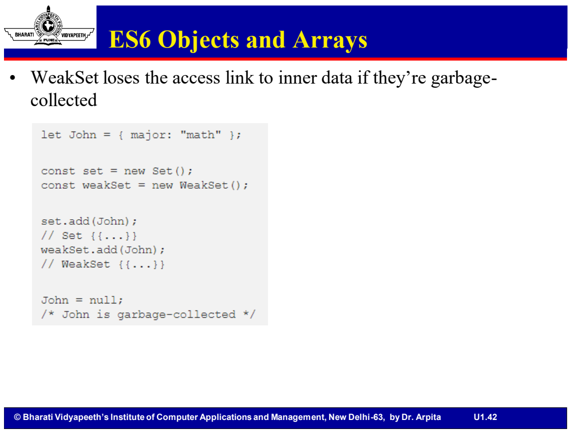
- Set is also quite similar to Map, but Set is more useful for a single value
  - remove Properties

```
set.delete(5);
// true

set.delete(function(){});
// false;
```

  - Set to array conversion

```
/* With Set */
const set = new Set();

set.add(1);
set.add(2);
set.add(2);
set.add(3);
set.add(3);
// Set {1, 2, 3}

// Converting to Array
const arr = [ ...set ];
// [1, 2, 3]
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita U1.41



ES6 Objects and Arrays

- WeakSet loses the access link to inner data if they're garbage-collected

```
let John = { major: "math" };

const set = new Set();
const weakSet = new WeakSet();

set.add(John);
// Set {{...}}
weakSet.add(John);
// WeakSet {{...}}

John = null;
/* John is garbage-collected */
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita U1.42

Some Important Topics

- Destructuring Assignment : allows you to locally scope fields within an object and to declare which values will be used

```
var sandwich =  {
      bread: "dutch crunch",
      meat: "tuna",
      cheese: "swiss",
      toppings: ["lettuce", "tomato", "mustard"]
}

var {bread, meat} = sandwich

console.log(bread, meat) // dutch crunch tuna
```

```
var lordify = regularPerson => {
  console.log(`${regularPerson.firstname} of Canterbury`)
}

var regularPerson = {
  firstname: "Bill",
  lastname: "Wilson"
}

lordify(regularPerson)      // Bill of Canterbury
```

Some Important Topics

- Object Literal Enhancement: the process of restructuring or putting back together or opposite of destructuring

```
var name = "Tallac"
var elevation = 9738

var funHike = {name,elevation}

console.log(funHike) // {name: "Tallac", elevation: 9738}
```

```
// OLD
var skier = {
    name: name,
    sound: sound,
    powderYell: function() {
        var yell = this.sound.toUpperCase()
        console.log(`${yell} ${yell} ${yell}!!!`)
    },
    speed: function(mph) {
        this.speed = mph
        console.log('speed:', mph)
    }
}
```

```
// NEW
const skier = {
    name,
    sound,
    powderYell() {
        let yell = this.sound.toUpperCase()
        console.log(`${yell} ${yell} ${yell}!!!`)
    },
    speed(mph) {
        this.speed = mph
        console.log('speed:', mph)
    }
}
```

Some Important Topics

- The Spread Operator (… three dots):
  - spread operator allows us to combine the contents of arrays

```
var peaks = ["Tallac", "Ralston", "Rose"]
var canyons = ["Ward", "Blackwood"]
var tahoe = [...peaks, ...canyons]

console.log(tahoe.join(', '))  // Tallac, Ralston, Rose, Ward, Blackwood


var peaks = ["Tallac", "Ralston", "Rose"]
var [last] = [...peaks].reverse()

console.log(last) // Rose
console.log(peaks.join(', '))  // Tallac, Ralston, Rose


var lakes = ["Donner", "Marlette", "Fallen Leaf", "Cascade"]

var [first, ...rest] = lakes

console.log(rest.join(", ")) // "Marlette, Fallen Leaf, Cascade"
```

Some Important Topics

- Promises:
  - Promises give us a way to make sense out of asynchronous behavior
  - When making an asynchronous request, one of two things can happen: everything goes as we hope or there's an error
  - Promises give us a way to simplify back to a simple pass or fail.

```
const getFakeMembers = count => new Promise((resolves, rejects) => {
  const api = `https://api.randomuser.me/?nat=US&results=${count}`
  const request = new XMLHttpRequest()
  request.open('GET', api)
  request.onload = () =>
       (request.status === 200) ?
        resolves(JSON.parse(request.response).results) :
        reject(Error(request.statusText))
  request.onerror = (err) => rejects(err)
  request.send()
})
```

ES6 Modules

- A JavaScript module is a piece of reusable code that can easily be incorporated into other JavaScript files
  - There are two options when creating and exporting a module: you can export multiple JavaScript objects from a single module, or one JavaScript object per module

./text-helpers.js

```
export const print(message) => log(message, new Date())

export const log(message, timestamp) =>
    console.log(`${timestamp.toString()}: ${message}`}
```

./mt-freel.js

```
const freel = new Expedition("Mt. Freel", 2, ["water", "snack"])

export default freel
```

```
import { print as p, log as l } from './text-helpers'

p('printing a message')
l('logging a message')
```

AJAX

- Asynchronous JavaScript And XML (AJAX)
  - use of the XMLHttpRequest object to communicate with servers
  - send and receive information in various formats, including JSON, XML, HTML, and text files
  - Nature is asynchronous
  - two major features of AJAX
    - Make requests to the server without reloading the page
    - Receive and work with data from the server

Step 1 – How to make an HTTP request

- To make an HTTP request to the server with JavaScript
  - Create instance of XMLHttpRequest

```
// Old compatibility code, no longer needed.
if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+ ...
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 6 and older
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
```

  - After making a request, you will receive a response back and tell the XMLHttp request object which JavaScript function will handle the response, by setting the **onreadystatechange** property of the object

```
httpRequest.onreadystatechange = nameOfTheFunction;
```

```
httpRequest.onreadystatechange = function(){
    // Process the server response here.
};
```

```
httpRequest.open('GET', 'http://www.example.org/some.file', true);
httpRequest.send();
```

Step 2 – Handling the server response

- When you sent the request, you provided the name of a JavaScript function to handle the response, the function needs to check the request's state

```
if (httpRequest.readyState === XMLHttpRequest.DONE) {
    // Everything is good, the response was received.
} else {
    // Not ready yet.
}
```

- readyState values
  - 0 (uninitialized) or (request not initialized)
  - 1 (loading) or (server connection established)
  - 2 (loaded) or (request received)
  - 3 (interactive) or (processing request)
  - 4 (complete) or (request finished and response is ready)

Step 2 – Handling the server response

- When you sent the request, you provided the name of a JavaScript function to handle the response, the function needs to check the request's state

```
if (httpRequest.readyState === XMLHttpRequest.DONE) {
    // Everything is good, the response was received.
} else {
    // Not ready yet.
}
```

- readyState values
  - 0 (uninitialized) or (request not initialized)
  - 1 (loading) or (server connection established)
  - 2 (loaded) or (request received)
  - 3 (interactive) or (processing request)
  - 4 (complete) or (request finished and response is ready)

```
if (httpRequest.status === 200) {
    // Perfect!
} else {
    // There was a problem with the request.
    // For example, the response may have a 404 (Not Found)
    // or 500 (Internal Server Error) response code.
}
```

Example

```
<button id="ajaxButton" type="button">Make a request</button>

<script>
(function() {
  var httpRequest;
  document.getElementById("ajaxButton").addEventListener('click', makeRequest);

  function makeRequest() {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
      alert('Giving up :( Cannot create an XMLHTTP instance');
      return false;
    }
    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', 'test.html');
    httpRequest.send();
  }
```

```
  function alertContents() {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
      if (httpRequest.status === 200) {
        alert(httpRequest.responseText);
      } else {
        alert('There was a problem with the request.');
      }
    }
  }
})();
</script>
```

Working with the XML response

- test.xml contains the following:

```
<?xml version="1.0" ?>
<root>
    I'm a test.
</root>
```

```
httpRequest.open('GET', 'test.xml');
```

```
var xmldoc = httpRequest.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);
```

Page Setup

- two libraries required
  - React :
    - React is a JavaScript library for building user interfaces
    - **you can use as little or as much React as you need**
  - ReactDOM
    - The react-dom package provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside the React model if you need to.

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

    - The react-dom package also provides modules specific to client and server apps:
      - react-dom/client
      - react-dom/server

Virtual DOM

- DOM (Document Object Model)
  - document structure, style, and content
  - A web page is a document
  - As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript
- The virtual DOM (VDOM) is a programming concept
  - where an ideal, or "virtual", representation of a UI is kept in memory
  - synced with the "real" DOM by a library such as ReactDOM
- VDOM ensure the DOM matches the state, if developer change the state
  - attribute manipulation
  - event handling
  - manual DOM updating
- VDOM is usually associated with React elements
- React internal objects "fiber" hold the additional information about the component tree [React Fiber more details: https://github.com/acdlite/react-fiber-architecture]

Virtual DOM

- Shadow DOM
  - The Shadow DOM is a browser technology
  - primarily designed for scoping variables and CSS in web components
- More differences between Shadow DOM and Virtual DOM

Virtual DOM	Shadow DOM
It revolves around solving performance issues.	It revolves around the concept of encapsulation.
It is a complete representation of an actual DOM.	It is not a complete representation of the entire DOM.
It groups together several changes and does a single re-render instead of many small ones.	It adds a subtree of DOM elements into the rendering of a document, instead of adding it to the main document's DOM tree.
It creates a copy of the whole DOM object.	It creates small pieces of the DOM object having their own, isolated scope.
It does not isolate the DOM.	It isolates the DOM.
It does not help with CSS scoping.	It helps with CSS scoping.

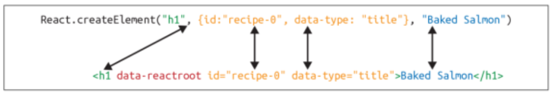
[Source: https://www.geeksforgeeks.org/what-is-the-difference-between-shadowdom-and-virtualdom/]

React Elements

- React DOM is made up of React
- React elements are the instructions for how the browser DOM should be created
- Create a React element

```
React.createElement("h1", null, "Baked Salmon")
```

  - first argument defines the type of element
  - second argument represents the element's properties
  - third argument represents the element's children



```
React.createElement("h1", {id:"recipe-0", data-type: "title"}, "Baked Salmon")

<h1 data-reactroot id="recipe-0" data-type="title">Baked Salmon</h1>
```

ReactDOM

- ReactDOM contains the tools necessary to render React elements in the browser
- ReactDOM is where we will find the render method as well as the **renderToString** and **renderToStaticMarkup** methods
- Children

```
React.createElement(
    "ul",            Main Element
    null,
    React.createElement("li", null, "1 lb Salmon"),        Child Element
    React.createElement("li", null, "1 cup Pine Nuts"),
    React.createElement("li", null, "2 cups Butter Lettuce"),
    React.createElement("li", null, "1 Yellow Squash"),
    React.createElement("li", null, "1/2 cup Olive Oil"),
    React.createElement("li", null, "3 cloves of Garlic")
)
```

ReactDOM

- Constructing Elements with Data

```
React.createElement("ul", {"className": "ingredients"},
    React.createElement("li", null, "1 lb Salmon"),
    React.createElement("li", null, "1 cup Pine Nuts"),
    React.createElement("li", null, "2 cups Butter Lettuce"),
    React.createElement("li", null, "1 Yellow Squash"),
    React.createElement("li", null, "1/2 cup Olive Oil"),
    React.createElement("li", null, "3 cloves of Garlic")
);
```

Mapping an array to li elements

```
React.createElement(
    "ul",
    { className: "ingredients" },
    items.map(ingredient =>
        React.createElement("li", null, ingredient)
)
```

```
var items = [
    "1 lb Salmon",
    "1 cup Pine Nuts",
    "2 cups Butter Lettuce",
    "1 Yellow Squash",
    "1/2 cup Olive Oil",
    "3 cloves of Garlic"
]
```

```
⊗ ▶Warning: Each child in an array or iterator should have a       runner-3.36.10.min.js:1
unique "key" prop. Check the top-level render call using <ul>. See https://fb.me/react-
warning-keys for more information.
```

ReactDOM

- Constructing Elements with Data

```
React.createElement("ul", {"className": "ingredients"},
    React.createElement("li", null, "1 lb Salmon"),
    React.createElement("li", null, "1 cup Pine Nuts"),
    React.createElement("li", null, "2 cups Butter Lettuce"),
    React.createElement("li", null, "1 Yellow Squash"),
    React.createElement("li", null, "1/2 cup Olive Oil"),
    React.createElement("li", null, "3 cloves of Garlic")
);
```

Mapping an array to li elements

```
React.createElement("ul", {className: "ingredients"},
    items.map((ingredient, i) =>
        React.createElement("li", { key: i }, ingredient)
)
```

```
var items = [
    "1 lb Salmon",
    "1 cup Pine Nuts",
    "2 cups Butter Lettuce",
    "1 Yellow Squash",
    "1/2 cup Olive Oil",
    "3 cloves of Garlic"
]
```

React Components

- Reduce the complexities of UI
- re-use the same DOM for different purpose or different sets of data
- Identify the react components
  - Try to break down your elements into reusable pieces
  - How scalable it is?
- How to create a Component?
  - In 2013, React.createClass

React Components

```
const IngredientsList = React.createClass({
  displayName: "IngredientsList",
  render() {
    return React.createElement("ul", {"className": "ingredients"},
        React.createElement("li", null, "1 lb Salmon"),
        React.createElement("li", null, "1 cup Pine Nuts"),
        React.createElement("li", null, "2 cups Butter Lettuce"),
        React.createElement("li", null, "1 Yellow Squash"),
        React.createElement("li", null, "1/2 cup Olive Oil"),
        React.createElement("li", null, "3 cloves of Garlic")
    )
  }
})

const list = React.createElement(IngredientsList, null, null)

ReactDOM.render(
  list,
  document.getElementById('react-container')
)
```

React Components

- Create Components
  - React.Component

```
class IngredientsList extends React.Component {

  renderListItem(ingredient, i) {
    return React.createElement("li", { key: i }, ingredient)
  }

  render() {
    return React.createElement("ul", {className: "ingredients"},
        this.props.items.map(this.renderListItem)
    )
  }

}
```

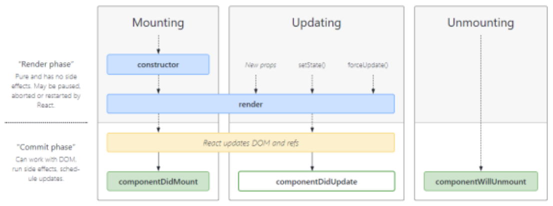
React Components

- Stateless Functional
  - Stateless functional components are functions, not objects
  - Don't have "this" scope
  - Pure functions, we can use as much as possible
  - Stateless functional components are functions that take in properties and return a DOM element
  - keep your application architecture simple
  - Not in use to encapsulate functionality or have a this scope

```
const IngredientsList = props =>
    React.createElement("ul", {className: "ingredients"},
        props.items.map((ingredient, i) =>
            React.createElement("li", { key: i }, ingredient)
        )
    )
```

React Component Lifecycle

- The Component Lifecycle



[Source: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/]

React Component Lifecycle

- The Component Lifecycle
  - Mounting : called in the following order when an instance of a component is being created and inserted into the DOM
    - **constructor()**
    - static getDerivedStateFromProps()
    - **render()**
    - **componentDidMount()**

React Component Lifecycle

- Updating : called in the following order when a component is being re-rendered
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - **render()**
  - getSnapshotBeforeUpdate()
  - **componentDidUpdate()**
- Unmounting
  - **componentWillUnmount()**
- Error Handling
  - static getDerivedStateFromError()
  - componentDidCatch()

DOM Rendering

- Using react we are able to pass data to our components as props
- Isolate the application's data from the logic, used to create UI
- In change of any isolated data, change the state
- Plan to make a light weighted components to lighten the process of component render.
- For the heavy lift of DOM component, react works smartly and make only the minimal required changes to optimize the processing time.

```
["smile", "smile", "frown", "smile", "frown"];

["frown", "frown", "frown", "frown", "frown"];
```

DOM Rendering

- How we can update the DOM to reflect these changes?
  - Inefficient solution
    - Empty the current data
    - Begin looping through data and build the first list item
    - Build and add the second list item
    - Build and append the third list item
    - .. And so on
  - ReactDOM.render makes changes by leaving the current DOM in place and simply updating the DOM elements that need to be updated

```
<ul>
  <li class="smile">smile</li>
  <li class="smile">smile</li>
  <li class="frown">frown</li>
  <li class="smile">smile</li>
  <li class="frown">frown</li>
</ul>
```

```
<ul>
  <li class="frown">frown</li>
  <li class="frown">frown</li>
  <li class="frown">frown</li>
  <li class="frown">frown</li>
  <li class="frown">frown</li>
</ul>
```

Factories

- Another way to create a React element is to use factories
- A factory is a special object that can be used to abstract away the details of instantiating objects
- React has built-in factories for all commonly supported HTML and SVG DOM elements

Using createFactory to create an h1

```
React.DOM.h1(null, "Baked Salmon")
```

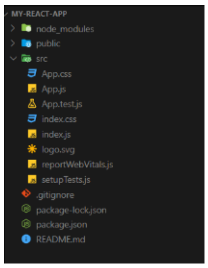
```
React.DOM.ul({"className": "ingredients"},
    React.DOM.li(null, "1 lb Salmon"),
    React.DOM.li(null, "1 cup Pine Nuts"),
    React.DOM.li(null, "2 cups Butter Lettuce"),
    React.DOM.li(null, "1 Yellow Squash"),
    React.DOM.li(null, "1/2 cup Olive Oil"),
    React.DOM.li(null, "3 cloves of Garlic")
)
```

Using Factories with Components

```
const { render } = ReactDOM;

const IngredientsList = ({ list }) =>
    React.createElement('ul', null,
        list.map((ingredient, i) =>
            React.createElement('li', {key: i}, ingredient)
        )
    )

const Ingredients = React.createFactory(IngredientsList)

const list = [
    "1 lb Salmon",
    "1 cup Pine Nuts",
    "2 cups Butter Lettuce",
    "1 Yellow Squash",
    "1/2 cup Olive Oil",
    "3 cloves of Garlic"
]

render(
    Ingredients({list}),
    document.getElementById('react-container')
)
```

Steps to create first reactjs app

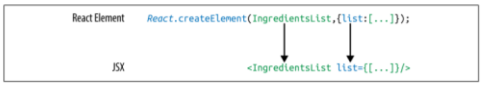
- Install NVM
  - https://github.com/coreybutler/nvm-windows
- Install NodeJS
  - https://nodejs.org/en/download/
- Command to check node version
  - node --version
  - npm --version
- To create first app
  - npx create-react-app my-react-app



```
MY-REACT-APP
> node_modules
> public
v src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

React Elements as JSX

- JSX, a simpler way to creating complex DOM trees with attributes
- JSX is as readable as HTML, XML
- JSX elements can be added as children
- Array of the elements can be pass into JSX


React Element
React.createElement(IngredientsList,{list:[...]});
JSX
<IngredientsList list={[...]}/>

React Elements as JSX

- Nested components
- className
- JavaScript expressions `<h1>{this.props.title}</h1>`
- Evaluation `<h1>{"Hello" + this.props.title}</h1>`
- Mapping arrays to JSX

```
<ul>
    {this.props.ingredients.map((ingredient, i) =>
        <li key={i}>{ingredient}</li>
    )}
</ul>
```

Babel

- To transpile code in the browser, use Babel v. 5.8.
- Babel 6.0+ will not work as an in-browser transformer.
- transpile our code from JSX to pure React, Babel will also convert ES6 into common ES5 JavaScript that is readable by all browsers
- Babel Presets
  - babel-preset-es2015
  - babel-preset-env
  - babel-preset-react

Props, State, and the Component Tree

Property Validation

- JavaScript is a loosely typed language
- React components provide a way to specify and validate property types
  - reduce the amount of time spent debugging
  - incorrect property types triggers warnings

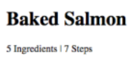
```
import PropTypes from 'prop-types';
```

Type	Validator
Arrays	React.PropTypes.array
Boolean	React.PropTypes.bool
Functions	React.PropTypes.func
Numbers	React.PropTypes.number
Objects	React.PropTypes.object
Strings	React.PropTypes.string

Validating Props with createClass

- Consider the following two examples

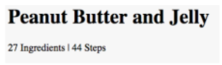
```
    const Summary = createClass({
        displayName: "Summary",
        render() {
            const {ingredients, steps, title} = this.props
            return (
                <div className="summary">
                    <h1>{title}</h1>
                    <p>
                        <span>{ingredients.length} Ingredients</span> |
                        <span>{steps.length} Steps</span>
```



Baked Salmon

5 Ingredients | 7 Steps

```
render(
  <Summary title="Peanut Butter and Jelly"
           ingredients="peanut butter, jelly, bread"
           steps="spread peanut butter and jelly between bread" />,
  document.getElementById('react-container')
)
```



Peanut Butter and Jelly

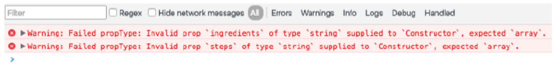
27 Ingredients | 44 Steps

Validating Props with createClass

- The following example demonstrate the propTypes

```
const Summary = createClass({
    displayName: "Summary",
    propTypes: {
        ingredients: PropTypes.array,
        steps: PropTypes.array,
        title: PropTypes.string
    },
```

- propTypes are used to define the property types (strictly), raise an error while not belongs to given propTypes



```
Warning: Failed propType: Invalid prop `ingredients` of type `string` supplied to `Constructor`, expected `array`.
Warning: Failed propType: Invalid prop `steps` of type `string` supplied to `Constructor`, expected `array`.
>
```

Default Props

- assign default values for properties
- the default values you establish will be used if other values are not provided

```
const Summary = createClass({
    displayName: "Summary",
    propTypes: {
        ingredients: PropTypes.number,
        steps: PropTypes.number,
        title: PropTypes.string
    },
    getDefaultProps() {
        return {
            ingredients: 0,
            steps: 0,
            title: "[recipe]"
        }
    },
```

Custom Property Validation

- React's built-in validators are great for making sure that your variables are required and typed correctly
- Custom validation in React is implemented with a function

```
propTypes: {
    ingredients: PropTypes.number,
    steps: PropTypes.number,
    title: (props, propName) =>
        (typeof props[propName] !== 'string') ?
            new Error("A title must be a string") :
            (props[propName].length > 20) ?
                new Error(`title is over 20 characters`) :
                null
}
```

ES6 Classes and Stateless Functional Components

- The class component
  - a stateful/container component
  - regular ES6 class that extends the component class of the React library
  - it controls how the state changes
  - the implementation of the component logic
- The functional component
  - simply JavaScript functions
  - stateless or presentational components
  - returned data to be rendered to the DOM
  - with hooks, possible to implement of state and other features in React

ES6 Classes and Stateless Functional Components

- ES6 classes, propTypes and defaultProps declarations are defined on the class instance, outside of the class body

```
Summary.defaultProps = {
    ingredients: 0,
    steps: 0,
    title: "[recipe]"
}
```

- With a stateless functional component, you also have the option of setting default properties directly in the function arguments

```
const Summary = ({ ingredients=0, steps=0, title='[recipe]' }) => {
  return <div>
    <h1>{title}</h1>
    <p>{ingredients} Ingredients | {steps} Steps</p>
  </div>
}
```

When I need to use ES6 Classes

- If your component needs to access this, use ES6 Classes
- If your components need lifecycle methods, use ES6 classes

ES6 Classes and Stateless Functional Components

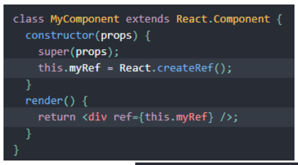
- Class Static Properties

Refs

- Refs provide a way to access DOM nodes or React elements created in the render method
- When to Use Refs
  - Managing focus, text selection, or media playback.
  - Triggering imperative animations.
  - Integrating with third-party DOM libraries.

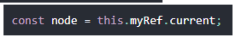
Refs

- Creating Refs



```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

- Accessing Refs



```
const node = this.myRef.current;
```

  - When the ref attribute is used on an HTML element, the ref created in the constructor with React.createRef()
  - not use the ref attribute on function components

Refs

- Adding a Ref to a DOM Element
  - Define under the class

```
this.textInput = React.createRef();
```

  - Explicitly focusing the text input

```
this.textInput.current.focus();
```

  - Attached in DOM element

```
<input
  type="text"
  ref={this.textInput} />
```

Refs

- Refs and Function Components

```
function MyFunctionComponent() {
  return <input />;
}

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }
  render() {
    // This will *not* work!
    return (
      <MyFunctionComponent ref={this.textInput} />
    );
  }
}
```

Refs

- React State Management
  - Properties are immutable
  - State for managing data that will change within a component
  - User navigate, search, filter, select, add, update, and delete
  - After every interaction with an application
    - State might be changed
    - On state change, changes are reflected back to the user in the UI

State Management

- Initializing State from Properties
- State Within the Component Tree
  - Each React components can have their own state
  - React comes from building scalable applications
  - possible to group all state data in the root component
  - State data can be passed down the component tree via properties
  - data can be passed back up the tree to the root via two-way function binding
  - Referred as "single source of truth".

State Management

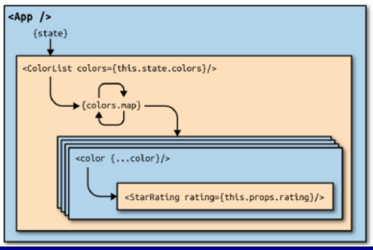
- Passing Properties Down the Component Tree
  - Presentational components are only concerned about the look in the application
  - render DOM elements or other presentational components
  - All data is sent to the components via properties and passed out of these components via callback functions
  - Presentational components only use props
- Since we are removing state from this component, when a user changes the rating, that data will be passed out of this component via a callback function.

State Management

```
const StarRating = ({starsSelected=0, totalStars=5, onRate=f=>f}) =>
    <div className="star-rating">
        {[...Array(totalStars)].map((n, i) =>
            <Star key={i}
                  selected={i<starsSelected}
                  onClick={() => onRate(i+1)}/>
        )}
        <p>{starsSelected} of {totalStars} stars</p>
    </div>
```

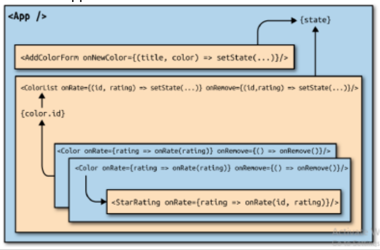
State Management

- State in Reusable Components
  - create stateful UI components for distribution
  - reuse across many different applications


<App />
{state}
<ColorList colors={this.state.colors}/>
{colors.map}
<color {...color}/>
<StarRating rating={this.props.rating}/>

State Management

- Passing Data Back Up the Component Tree
  - create stateful UI components for distribution
  - reuse across many different applications


<App />
{state}
<AddColorForm onNewColor={(title, color) => setState(...)}/>
<ColorList onRate={(id, rating) => setState(...)} onRemove={(id,rating) => setState(...)}/>
{color.id}
<Color onRate={rating => onRate(rating)} onRemove={() => onRemove()}/>
<Color onRate={rating => onRate(rating)} onRemove={() => onRemove()}/>
<StarRating onRate={rating => onRate(id, rating)}/>

Props vs. State

State	Props
The Data is passed from one component to another.	The Data is passed within the component only.
It is Immutable (cannot be modified).	It is Mutable (can be modified).
Props can be used with state and functional components.	State can be used only with the state components/class component (Before 16.0).
Props are read-only.	State is both read and write.

Forms in react

- HTML form elements work a bit differently from other DOM elements in React, because form elements naturally keep some internal state
- Controlled Components
  - <input>, <textarea>, and <select> typically maintain their own state and update it based on user input
    - The textarea Tag
    - The select Tag
    - The file input Tag

Forms in react

- Handling Multiple Inputs
- Controlled Input Null Value