React Key Concepts

-A JS UI library used to build and display HTML views. Renders interfaces in SPA and mobile apps. Useful for very dynamic sites (ex. Facebook).

-Library made of UI components created using pure javascript. Components highly contained (ex. address and zip code elements). Components have both visual representation and dynamic logic. Components can interact with server. Component-based architechture (CBA).

-Note: it is common for people to shorthand web GUI as just UI

React Core Concepts & Uses
-Useful in building and managing complex dynamic UIs. "Building large applications with data that changes over time."

-UIs as functions. Call them with data. Get rendered in view with automatic updates. Renders components in memory instead of re-rendering whole DOM. Much faster for dynamic sites. Powerful abstractions for same view on many browsers.

-Declerative over imperative. Imperetive involves declaring a lot of vars, then interacting with them. Declerative creates using functions, often calls on objects returns from another function call, etc. Shorter, more readable. Less split code/logic. Less local vars.

-Change in view = state change. React updates view. Uses virtual dom running in background to detect diffs in existing and new views.

-Unlike angular, and some others, is all pure JS with no templates in other lang. Split lang and files for single componenet. React can do all in one file in pure JS.

-React abstraction provides wrapper around events. Can render elements on sever.

-Speed. React virtual DOM exists in JS memory. Real Dom rendered in browser. Virtual dom then rendered by react. If virtual dom changes, diff with real dom will show, and react will update real DOM, but only elements of DOM that have changed.
-"Internal state" = virtual dom, "view" = real dom

-Large community with many already made components that you can use

-Note that since react is barebones, need to pack with routing and datamodeling libraries like flux, react router, which can be done with *create-react-app*

-JSX - tiny syntax for writing React object in JS using < >, as in HTML, to define elements. JSX src code compiled into JS for browser.

Building a React Project

React Code 101

-*someArray.map(transformFunction)* - Returns an array of values after applying the transform function

-If creating react app in most barebones way, will need three files: index.html, react.js, and react-dom.js. The JS files come from the react source module library.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="js/react.js"></script>                    //imports react library
    <script src="js/react-dom.js"></script>                //imports reactDom library
  </head>
  <body>
    <script type="text/javascript">
          let someElement = React.createElement(..);       //create react element
          ReactDOM.render(...)                             //render elment(s) onto dom,
    </script>                                                 //ex. place on existing <div>
  </body>
</html>
```

-Note: standard to have create react elements and render methods in a seperate file (ex. App.js) and render onto html files, which imports via <script src="...">

-Can install basic local http server with npm via *npm i -g http-server*, then navigate to folder containing site files, then run *http-server* to start server. ip to use for local host for browser access, etc. will show.

App Creation & Server with *create-react-app*
-install *create-react-app* with npm, then in terminal create app via: *create-react-app appName*, then start host via *npm start* from *appName* dir

-Once run *npm start*, will run the react server locally on your computer, after which the project can be inspected from the browser. Typical is *http://localhost:3000/*

Project Folder with create-reacte
-Basic flow: index.html and index.js are the key points which create-react builds on. index.js imports React core modules and App.js serves as the default component and holds a class which extends a component. It has a a *render()* method which returns react elements. index.js holds the ReactDom.render() method, which renders an instance of App via the App's render() method, on a specified location in the html dom (ex. a * *, which the index.html file contains

APP.JS (component with *render()* method) -->rendered by---index.js---->displayed on----index.html

-Folder contains:
node_modules package.json package-lock.json public README.md src

-node_modules - folder, holds library in node packages, including some defaults if used *create-react-app* to instantiate project
-public - folder, holds assets, html, favicon, images, etc.. Is hosted on *localhost:3000* for access to
-src - folder, holds js, css, etc.
-build - built when build project, for combo of needed src and public files

-index.html file is super basic DOM tree, (React will inject html later) with only two body element.
I) a *<noscript>* element to display an error message is browser doesn't have JS running
II) a *id="root" *, which will have components built onto it by REACT

-index.js - contains various *import* react modules, css, etc. Entry point into react.
-import css via *import './fileLocation.css';*
-Imports JS *import Hello from './components/hello';* //hello is a js file, but you leave .js off

-App.js - main view for intial project. Where you will implement REACT components in REACT rendering of view. Components generally in seperate files, and included in app.js

-App.test.js - for holding program tests

-index.css and App.css

-*ReactDOM.render(<app />, document.getElementById('root'));* - renders program into browser. Does so realtime, so browser will auto-update as soon as any code changed.

-*npm start* - start the app development server. Can then acess in website, default via localhost:3000
-*npm build* - bundles the app into static files for production
-*npm test* - starts the test runner
-*npm eject* - removes npm tool from project and copies build dependencies, config files, and scripts into app dir. Once *eject*, cannot be undone.

-bootstrap - on OS level, the program that initializes the OS during startup. May also see someone use on more local terms, "ex. react is bootstrapped," ie the react environment is up and running on the local host

React Basics

-In react, elements are instances of components (ie component classes)

Create Elements
-*React.createElement('elementTagName', {properties: 'values'}, childrenOrHTML);*
  -ex. *React.createElement('a', {href: 'http://myblog.com'}, 'Webapplog.com'};*

-Can also send instances of React component classes to *createElement()* instead of *'elementTagName'*. Send name of componenet class and instance will be created.

-in *createElement()* { //propertyBox }}, attribute names reference HTML attributes. Inside the component classes code, can access specified properties via *this.props.somePropertyName;*

-Can also give element a someProperty: "theValue", attribute where someProperty is a not a standard attribute. If this happens, attribute will not render on element (how would it?), but you can access it as a key/value pair from inside the class via *this.props.someProperty*

-Could thus create element of *someComponent* with same base, but different properties, and have functions inside component's *render()* method for how to display potential property values

-Could run a test on a property value and return one element type in case1, another element type with different properties in case2, etc.

-Note that class properties for instances of a class are immutable. If you want unique properties, set them at instantion time
-Can pass properties to child elements, by passing *this.props* instead of *{...} value* definitions

Nesting elements

-Can create a single named react element, via React.create(Element), define it more, then apply multiple times within container element by sending name multi times to createElement

-To create parent element with children (ex. * * containing multiple *<img>*, *createElement()* like:
React.createElement('div', null, img, img};

-React dev tools for firefox adds better React support to mozilla dev tools.

Creating Component Classes
-HTML and CSS provide bare structure for. index.html provides the skeleton for the layout of the site, then has component classes added on top of it, similiar to very self contained widgets

-Create component by creating a class that *extends Component* class
-Make sure .js file for compents imports *Component* via:
import React, { Component } from 'react'; //import if using create-react

-Only mandatory method for component is *render()*, which must return a single element, created from an html tagName or another as component. If need multi elements, wrap in * * and return that, etc.

Ex. *class HelloWorld extends Component {*
render() {
return React.createElement('div', null, 'Hello World')
}

}

ReactDom.render(React.createElement(HelloWorld, null, document.getElementById('contentDiv');
//renders onto content in html

-If *return* covers multiple lines and no text following *return* on first line, enclose lines in ( );

-Naming syntax: let ComponentInstance = ...;

-To create an instance of *ComponentType* from componentX JS file, make sure to import *ComponentType* into componentX via *import ComponentType from './ComponentType';*
//JS file with no .js

Intro to JSX

-Introduced largely as a way to create quicker, more reable react code, so you're not writing long nested *createElement* statements. Shorthand sugar to create react elements. A small language with XML-like syntax.

-Allows you to effeciently define and create react elements with complex html definitions, in readable code. Allows you to mimic HTML coding from React, so don't have to ever really touch .html files

-JSX code compiled by various transpilers into standard JS for browser use

-With JSX, can create instance of component by calling *<ComponentName />*

-Can inspect elements in react.dom by *console.log(React.DOM)*

-JSX is supported by default in *create-react-app* as it includes Babel

Creating Elements with JSX

```
<elementName key1='value1' key2='value2' ...>
    <Child1/>                  //create component of class Child1
     <Child2/>                 //create component of class Child2
</elementName>
```

Ex. *let h1Elm = <h1>Hello world</h1>;* //could also pass right side code to RDOM.render(), etc.
//object creation and instantiation

-Can pass expression or value of contained element inside JSX <elementName> creation with ex. *{new*

Date().toLocaleString()}, as you can do in strings via template literals via ${…}. If put {variableName}, {expression}, etc. in braces, value of variable, expression result, etc. will be output

-Can also pass property names in { }, including *this.props.propertyName*, and value will be passed. If object passed within { }, whole object is passed

ex.
```
return (
    <div>
        {helloReactComponent}
        {helloReactComponent}
    </div>
);
```

Properties in JSX

-*keyName="value"* when defining elements represents either HTML attribute or React component property

-When accessing "*/public/images/image.jpg"* in an attribute, in *create-react* apps, leave */public* off

-To make dynamic objects, create the objects based on arguments passed in as properties by JSX < > instantion statements, then use those properties to create the component inside the component class by referencing the < > args via *this.props.propertyName* values

-Ex. create a class with an img and <a> inside a div, then pass in different img urls and links

-If storing any custom html attributes, name attribute in *dataTheName* format, to not accidently override any native attributes by accident

-To pass all properties, which can then be accessed individually by this, props, return *{...this.props}* as a nameless attribute value in the object you are creating

-Reminder for ES6: rest parameter ..., ex. *...someArray*, When last paremeter in args list has a ... before it, that parameter becomes an array of that name and holds any more args that are entered

Methods Inside Components

-Can write component method in form

```
methodName() {
    //logic
    return something;
}
```

-Can call on class methods from with class via *this.methodName()*

-If want to display one of two options (ex. logout or login), ternary op inside <> component creation *propertyValue={ ... }* where one of two values is selected based on a *this.props* flag is a quick way to do so

-ex **
{(this.props.sessionFlag) ? '/logout' : '/login'} //renders different text tool
**

Comments in JSX & Random
*{/*JSX Comment*/}* - same as JS, but wrapped in {}

-Still use html codes (ex. #40;) when called for when creating inner html content (ex. inside <p>) with JSX

CSS Styles
-define style in Javascript object, meaning define inside {...} set for *style="{...}"*

-To access and set CSS styles, use *this.props.style* or. Note naming for css like this background-image becomes backgroundImage when called in JSX via *this.props*.

-To render javascript object inline do so in double curly braces {{...}}

-Example of creating element with inline style

```
<span style={ {border: '3px solid black',
              font-size: '2rem'
            } }
</span>
```

Reserved Words - class, for
-As *class* and *for* are reserved words in JS, when defininging an element using JSX and REACT and giving it a *class="someName"* attribute, use *className="..."* instead. Same with HTML labels that take a *for="someElement"* attribute. When setting attribute via JS, use *htmlFor="..."* instead

Boolean Attribute Values
-For boolean attributes (ex. *loop* for <*video*>), define them as *loop={false}* or *loop={true}*. Can also ommite *{...}* definition and then JS will take as true (ex. <*input disabled*>)

Interactive React with States

-Allows you to store data for react components even with companant instances having immutable properties. Can automately augment views based on data changes.

-React state - A mutable data store held by components. *this.state* object, which is held by components, and its attributes. Ex. *this.state.inputValue*. When a state changes, corresponding parts of view that require changes are updated in DOM by virtual dom.

-*{this.state.inputFile}* - will show on dom if put in *render()*

Instantiating States

-Do via constructor inside component class:

```
constructor(props) {
   super(props)                  //necessary for it to have same properties as parent
   this.state = { keyA: "valueA",
                  keyB: expressioncall(),
                  keyC: [ arrayValues, ... ]
                  …
                }
}
```

-This is necessary because ECMAScript doesn't have support for class variables.…

Updating States

-Avoid setting state directly via *this.state*

-Update states via *this.setState({keyA: "valueA" , …}*

-*setInterval(function, milliseconds)* - a *window* function (can be called just by function name, though), that calls *function* every *milliseconds*.

-Also *setTimeout(function, ms)* - same, but only executes once after *ms*

-Standard to create an update method for updating this way

-Note that calling *setState()* triggers *render()*