**React**

**References**

**Lists of tutorials:**

* [https://javascript.plainenglish.io/helpful-react-tutorials-for-beginners-832c0d5aab7d](about:blank)   
  This is a *list of react tutorials*. It beqins with the official tutorial (which I didn't particularly like), but also includes others.
* [https://medium.com/@kutnickclose/the-best-free-react-course-in-2023-surprise-its-not-codecademy-4d039e4851c7](about:blank) recommends & compares various other react tutorials.   
    
    
    
  **Basic overview tutorials:**
* [https://www.guru99.com/reactjs-tutorial.html](about:blank) has a nice overview –it will orient you, even if it won’t make you into a senior react developer.
* [https://www.geeksforgeeks.org/react-js-introduction-working/](about:blank)

Started at   [https://www.geeksforgeeks.org/reactjs-tutorials/](about:blank)

* [https://www.tutorialspoint.com/reactjs/reactjs\_architecture.htm](about:blank) has the tutorialspoint tutorial.

**More** **in depth tutorials**

* One of the ‘lists of tutorials’ recommends the one at **Scrimba** [https://scrimba.com/learn/learnreact](about:blank) . (I tired Scrimba’s tutorial on typescript and it was painfully slow, even at higher speed. The react one seems to be a little better, but I I haven’t gotten thru all of it yet.)
* Mosh's video course at [https://www.youtube.com/watch?v=Ke90Tje7VS0](about:blank) or at

[https://www.youtube.com/watch?v=-AbaV3nrw6E&list=PL6gx4Cwl9DGBuKtLgPR\_zWYnrwv-JllpA&index=1](about:blank) . That course is from 2018 or 2019 and is 2 ½ hours long. It also links with a tutorial  
from freecodecamp (which may also be available as text, not video.)  
 There is an *updated version* (also ***fromMosh***) at

[https://www.youtube.com/watch?v=SqcY0GlETPk](about:blank) In it he uses node to install react, and a function approach. The newer version is from 2023 and is only 1 ½ hours long --- so I recommend the newer version. Mosh uses **vite** and typescript.

* [https://www.youtube.com/watch?v=w7ejDZ8SWv8](about:blank) is ***Brad*** ***Traversy’s*** video. He actually has two ‘crash courses’ on react. The earlier video (2019) uses classes and the later one (2021) uses functions. Brad uses **CRA (create react app)** and JS.
* [https://www.youtube.com/watch?v=-AbaV3nrw6E&list=PL6gx4Cwl9DGBuKtLgPR\_zWYnrwv-JllpA&index=1](about:blank) has the tutorial from Bucky Roberts at ***thenewboston***. As usual, this is less rushed that the other videos, but it’s old. (The comments suggest it was from 2017 or earlier.) That said, this is an easy way to get started.
* [https://react.dev/learn](about:blank) has the one from reactjs itself. The reactjs.com site has a lot of ways to get started --- there is a browser version to play with at [https://react.dev/learn/installation#try-react-locally](about:blank) (not for production) and that site also has a Quick Start Guide, for those who know something about react already. That said, these are more useful after you have a little orientation to React.

**Introduction to React:**  
First, we should distinguish between a ***library****and a****framework.***

One of the best explanations of their differences is at <https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/>   it's a 5 minute read - please read it.

Another older explanation is from <https://stackoverflow.com/questions/148747/what-is-the-difference-between-a-framework-and-a-library> I think that the main difference is that frameworks follow the "[Hollywood principle](http://en.wikipedia.org/wiki/Hollywood_Principle)", i.e. "don't call us, we'll call you."

According to [Martin Fowler](http://martinfowler.com/bliki/InversionOfControl.html):

A **library** is essentially a set of functions that you can call, these days usually organized into classes. Each call does some work and returns control to the client.

A **framework** embodies some abstract design, with more behavior built in. In order to use it you need to insert your behavior into various places in the framework either by subclassing or by plugging in your own classes. The framework's code then calls your code at these points.

After we work thru this document on React, we will ask briefly about the differences among React, Vue, and Angular  (Vue, like React, is based on building components. )

As of today, React is the most popular of the three, but ours is a rapidly changing field.  A couple of years ago Angular was the most popular.

A pretty short, good summary of their differences is at <https://www.browserstack.com/guide/angular-vs-react-vs-vue>  A slightly longer and possibly more informative summary is at <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/#who-wins>   A little googling on react vs vue vs angular will get you to many other opinions.  The chances are high that at a first job in web development you will be at a company which has already made that decision --- and that as a professional you will want to learn a little more about all three  (and about typescript.)

One  reliable way to track popularity is on Stack Overflow's annual survey.  The 2022 survey is at<https://survey.stackoverflow.co/2022/>  (notice, for example, that TypeScript has about half as many users as JavaScript overall and about 60% as many among professional developers;  and look at the databases, and the cloud platforms.)  This is also a good way to see what you should be keeping up on.

**Okay – on to React!**

**What is React?**

* From Facebook, which maintains it
* A *library*  which was created to build *sin*g*le pa*g*e apps.*
* React is for *front-end*  work. It focuses on building a *single page application or SPA*, but it does this by building many reuseable components.
* Like Express and Handlebars, React projects will have a substantial set of folders organized in a specified way. You can have this structure created for you (including loading the libraries of functions React uses) in either of two easy ways – with vite or with create-react-app. Both are available on npm.You will download the module vite or create-react-app from npm (no surprise) or you can link to or download the scripts (as we have done so often) from   
  [https://react-cn.github.io/react/downloads.html](about:blank) , which is rarely done these days. (You need 3 scripts – React, React-DOM and the Babel transpiler).
* In building your app you will design components. While the components get realized as pieces of html on your page, they are in fact JS functions which return that html. In fact, they are JSX functions.
* Based on designing *components* of a page --- they may be a nav bar or just one cell in a table. They can contain other components, but think HTML.  
  It is oriented towards builiding *user interfaces.*
* In terms of what they are syntactically: ***React components are JavaScript functions which return markup.***  See [https://react.dev/learn](about:blank)and [https://transform.tools/html-to-jsx](about:blank)
* Mixes together JavaScript and HTML – which it calls JSX. JSX allows you to mix together JavaScript (for behavior) and the HTML it interacts with. Then Babel transpiles it into vanilla JavaScript so that your browser can interpret it.
* To allow you to put the components together, React constructs a ***virtual dom*** for your page and updates the virtual dom as your page manipulates it. Then it updates the actual DOM. The result is that you end up using 3 separate libraries – a react library, a react-dom library, and the Babel transpiler. (You get all 3 automatically when you use either CRA or vite.)
* As you will see when we look at the tutorials, a component may *return* only html or other components. This causes some odd coding – especially if what you return is either from a loop or determined by an if statement.  
  The value of a variable or a value returned from a function may be included in your code with { } around it ---- e.g. <h2>Hi {name}!</h2> where name is a var/let/const.
* Each component can return only one parent element --- so if you have multiple elements (e.g. 2 paragraphs) they get wrapped in a <div> </div> or <Fragment></Fragment> or <></>
* Fast – b/c has a virtual DOM – but large files.
* The ***root*** starts the tree of components.
* Components are implemented as JS classes with a state property (to hold information which may change over time and may determine what they will display). If you use classes to build components, then they will have a render () method ; if you use functions to build components, they the functiosn will return the component.  
  if you expect your component to be nested inside other components then you will *export*  it.
* The state (virtual dom representation) can be changed and then it will update the (real) DOM. This is the antithesis of what we did when we worked with jQuery and we took care of updating the DOM ourselves (indeed, we gloried in doing that.)
* React (JSX) code is translated and compiled by babel. Babel is a *transpilerI*  - was used extensively to turn ES6 into classic JS, when ES6 was first introduced. That use of babel is not so common no, but it is still used to compile JSX into (classic) JS>  
  NOTE: That means that, when you are developing code, the old save and re-run will include the extra step of recompiling (even if it happens automatically when you use the create-react-app).  
  Because JSX is an extension of JS, you can have variables (like template literals, but missing the $) which you use in your page (i.e. values which are displayed in the HTML.)
* Each component has *one* parent element – which makes for some fancy coding --- e.g. you may have a function which returns a lot of html code and then your component just renders the value of that function.  
  For example   
  function SomeSuch(x, y, z) {  
   let message = ‘<ul><li>x</li>  
   <li>y</li>  
   <li>z</li>  
   </ul>’;  
   return message;  
  }  
  Then our “one parent element” can be   
   <h1>{SomeSuch(2, 3, 4)}</h1>
* You can also use template literals (w/o the $) which refer to a variable, so that the contents of your component can change (dynamically) depending on other things.
* Inside JSX there is no if statement (despite the fact that JSX is an extension of JS) so there is a lot of use of the ternary operator   
   what ? truth-response: false-response  
  or functions.
* Like handlebars or express, react has an expected structure of folders and files. You can start this up with either vite or create-react-app (CRA).

There is a whole ecosystem of modules that work with React – e.g. Remix, Redux etc. You can find lists of these at [https://brainhub.eu/library/top-react-libraries](about:blank) and [https://technostacks.com/blog/react-component-libraries/](about:blank) and [https://www.simform.com/blog/react-libraries/](about:blank)

**Order of videos and tutorials  
Explain MVC**

**TheNewBoston – video 2  
Newer Mosh – minute 6  
Traversy – minute 9  
 At minute 12 he writes *npx create-react-app my-app***npx is installed with npm and it allows you to run an npm package w/o installing it---- in other words we are running create-react-app (which sets up our new react app) but not downloading the CRA package. Obviously, if you use CRA a lot, you don’t mind downloading it.  
 He also recommends the React Dev tools for Chrome and FF. Lingo: Tool chain  
Whether you use vite or CRA, you will get gazillions of files installed --- including the libraries for React, React-Dom, and Babel.

The set up from either CRA or vite will give you an index.html page which has the usual doctype etc. and the body will just have **<div id=’root’></div>**  which is what will be the container for your page.  
**Mosh- minute 16.44 & then 23.29 ( after he imports bootstrap).**He shows you the well-know hack of changing class= to className =

He also emphasizes that *in React the name of any component must start with a capital letter.* (Pascal case is PascalCase).  
  
There are other subtleties – e.g. putting () around what you are returning so you can put it on multiple lines. But importantly, b/c you can return only one parent element it is **wrapped in a <div> or a <Fragment> (if you imported Fragment from React) or <>** at minute 29  
  
Please notice that here are hacks for what JSX is missing. For example,   
 multiple items are wrapped in <> </>  
 { } encloses dynamically rendered content  
 There is no ***if*** so we use the ternary operator what?true:false (Mosh minute 33)  
 or we use a variable which holds the result of our JS in some function and put  
 {the function’s return value} in our JSX.  
 There is no loop so we use map (Mosh minute 29)  
At minute 39 Mosh introduces props or properties.

Look at the [https://react.dev/learn](about:blank) examples and then [https://react.dev/learn/thinking-in-react](about:blank) and there are also tutorials about managing state.