JAVASCRIPT TO KNOW FOR REACT

QCC TECH WORKS

COURSE OUTLINE

- Goals:
  - Gain a better understanding of JavaScript features you'll be using with React.
  - Dive further into ES6
  - Practice Like a boss.

TEMPLATE LITERALS

- These are strings with super powers! They allow embedded expressions & can use multi-line strings. They also allow String interpolation features.
- They are enclosed by back-ticks (` `) character instead of double or single quotes.
- They can contain placeholders - these are indicated by the dollar sign and curly braces (`String text ${expression} string text`)
- Template Literals MDN

TEMPLATE LITERALS

```
/*
  Template Literals
*/

const myName = "Captain Fancy";
const campus = "Queensborough Community College";

console.log(`${myName} works at ${campus}.`)
//these are the same
console.log(myName + "works at " + campus + ".")

// In React
const box = ({className, ...props}) => {
  return <div className={`${className}`} />
}
```

ARROW FUNCTIONS

- Arrow Functions are another way to write functions in JavaScript. They do have a few semantic differences than regular functions - they have an implicit returns - and allows for anonymous functions.
- They are used in React to preserve the “this” context with-in a class.
- Arrow Functions MDN

ARROW FUNCTIONS

```
/*
  Arrow Functions
*/

const multiply = (a, b) => a * b;
const divide = (a, b) => a / b;
const addTen = a => a + 10;

// this is the same as

function multiply(a, b) {
  return a * b;
}

function divide(a, b) {
  return a / b;
}

function addTen(a) {
  return a + 10;
}
```

ARROW FUNCTIONS IN REACT

```
16 export default class Carousel extends Component {
17   constructor(props) {
18     super(props);
19
20     this.state = {
21       currentImageIndex: 0,
22     };
23   }
24
25   //@preLoadImages Loads all the images on the inital load to reduce the
     transistion flicker
26   preLoadImages = () =>
27     ImgUrls.map((image, idx) => {
28       return <img src={image} alt="" key={idx} />;
29     });
```

DESTRUCTURING

- The restructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. - MDN Docs
- You'll see this used heavily in React when passing props into functional components & using hooks.

DESTRUCTURING ARRAYS

```
40  //Array Destructuring
41  var a, b, rest;
42  [a, b] = [10, 20];
43
44  console.log(a);
45  // expected output: 10
46
47  console.log(b);
48  // expected output: 20
49
50  [a, b, ...rest] = [10, 20, 30, 40, 50];
51
52  console.log(rest);
53  // expected output: [30,40,50]
```

DESTRUCTURING OBJECTS

```
58   //Object Destructuring
59   var o = {p: 42, q: true};
60   var {p, q} = o;
61
62   console.log(p); // 42
63   console.log(q); // true
```

DESTRUCTURING IN REACT

Calling the functional component ImageSlider in a parent class component and passing the prop of url

```
65        <ImageSlider url={ImgUrls[this.state.currentImageIndex]} />
```

Then we destructure the prop in the Imageslider functional component

```
1   import React from 'react';
2
3   const Imageslider = ({url}) => {
4     const styles = {
5       backgroundImage: `url(${url})`
6     }
7     return (
8       <div className="image-slider" style={styles} id='img1'></div>
9     );
10  };
11
12  export default Imageslider;
```

REST / SPREAD OPERATORS

- Think of the ... syntax as a “collection” syntax. The rest or spread operator operates on a collection of values.
- The spread syntax allows an iterable such as an array expression or string to be expanded in place where zero or more arguments (for functions calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-values pairs (for object literals) are expected. -MDN Docs

REST / SPREAD OPERATORS

- Here we can use the spread operator to have the remaining values in the object be assigned to the value z.

```
66  //Destructuring using the spread operator
67
68  let { x, y, ...z } = { x: 1, y: 2, a: 3, b: 4 };
69  console.log(x); // 1
70  console.log(y); // 2
71  console.log(z); // { a: 3, b: 4 }
```

- We can also use the spread operator to make a copy of an array. In the past we would've use the Array.prototype.slice higher order array method.

```
73  //Copying an array using the spread operator.
74  var arr = [1,2,3];
75  var arr2 = [...arr]; // like arr.slice()
76  arr2.push(4)
77  console.log(arr2) //[1,2,3,4]
```

REST / SPREAD OPERATORS

- We can use this operator as well to combine arrays.

```
80   arr1.push(...arr2) // Adds arr2 items to end of array
81   arr1.unshift(...arr2) //Adds arr2 items to beginning of array
```

CONDITIONAL TERNARY OPERATOR

- The conditional ternary operator is the only JavaScript operator that takes three operands. This operator is frequently used as a shortcut for the if statement. - MDN Docs

```
84  //Conditional Ternary operator
85  const isTheDogHappy = flash.isHappy ? "Yes the pouchie is happy" : "No the doggie
    needs a belly rub"
86  // create a variable that will store the message if the dog is happy or not.
87
88  //This is the same as
89
90  const message;
91  if(flash.isHappy) {
92      isTheDogHappy = "Yes the pouchie is happy";
93  } else {
94      isTheDogHappy = "No the doggie needs a belly rub"
95  }
96
```

CONDITIONAL TERNARY OPERATOR IN REACT

- The ternary operator is great in react when you're conditionally waiting for some data. This functional component will only produce the list of the data when the puppies array has some length.

```
98  function PuppyList({puppies}) {
99    return (
100     <React.Fragment>
101       {puppies.length ? (
102         <ul>
103           {puppies.map(puppy => (
104             <li key={puppy.id}>
105               <span>{puppy.name}</span>
106             </li>
107           ))}
108         </ul>
109       ) : (
110         <div>There are no puppies. The sadness continues.</div>
111       )}
112     </React.Fragment>
113   )
114 }
```

RESOURCES

https://reactjs.org/docs/getting-started.html
https://reactjs.org/docs/introducing-jsx.html

GitHub Link to the workshop and files

https://github.com/CaptainKRS/lessons