skillcrush

USING ES6 TO KEEP YOUR CODE ORGANIZED *cheatsheet*

INTRO

ES6 is here to make your coding life easier! This cheatsheet walks through the two main ways you'll use ES6 syntax as you write React code. Go on, dig in, and feel the ES6 love!

ARROW FUNCTIONS

ES6 **arrow functions** let you write functions using less code, by using a fat arrow to accept arguments. The fat arrow replaces the need to use curly braces and the word "return" for functions with a single expression, which are called implicit returns.

Arrow functions are particularly helpful when your function only takes one argument, because you don't even need to use parentheses around the parameter, like in the example below!

A FUNCTION IN ES5:

```
  /* An ES5 function */
function es5Function(name) {
  return "Hello, " + name + "
from the functions of JavaScript
past.";
}
```

AND A FUNCTION IN ES6:

```
 /* An ES6 function */
const es6Function = name =>
"Hello,
from the future " + name + ".";
```

Arrow Functions with No Parameters

When a function has no parameters, you need to use empty parentheses to signify the lack of parameters.

```
const noParameters = () => "I accept no arguments";
// pass no arguments when calling this function
```

Arrow Functions with More Than One Parameter

And when you have more than one parameter, you must use parentheses and commas in between the parameters to show that you're using multiple parameters.

```
const manyParameters = (param1, param2, param3) =>
  "I have " +
  param1 +
  ", " +
  param2 +
  ", and " +
  param3 +
  " living in my home!";
// pass three arguments when calling this function
```

Arrow Functions with More Than One Expression

If you write a function that has more than one expression, you'll need to use curly braces and add the word "return." Functions with more than one expression are called explicit returns.

```
const manyParameters = (param1, param2) => {
  let statement = "I have " + param1;
  statement += ", and " + param2;
  statement += " living in my home!";
  return statement;
};
```

MODULES

Modules are individual files that hold each component definition you write. Modules help you separate each part of your code, making it easier to test, reuse, and debug.

There are three aspects to modules you'll need for this class: default modules exports, named modules exports, and third-party modules!

Default Exports

Default exports only allow one export per module file.

To write a default export, write “export default,” followed by the name of the variable that you want to be exportable, such as youRockSkillcrusher.

To import a default export, write “import,” followed by the name you want to call the variable (which can be its original name from its source file or a new name), and the file path, AKA where the file is located.

```
// Example code to export a default export
export default extraSpecialFunction;
```

```
// Example code to import a default export
import extraSpecialFunction from "./myFunction.js";
```

Named Exports

Named exports allow you to use multiple functions. For example, you can pull in youRockSkillcrusher, youRockSkillcrusher2, and youRockSkillcrusher3.

Named exports also allow you to rename functions, so you could import youRockSkillcrusher into a file, then rename it as iAlsoRock by writing “youRockSkillcrusher as iAlsoRock.”

To import a named export, you need to add curly braces after the keyword “import,” but otherwise named exports function similarly to default exports!

```
// Example code to export a named export
export const favoriteNumber = PI;
```

```
// Example code to import a named export
import { favoriteNumber } from "./math/helpers.js";
```

Third-Party Modules

Modules also allow you to import third-party code easily!

When you want to import a third-party module, you'll just write an import request, but instead of signifying the file path, just write "import," the name of the module, "from," and then the name of the library, like the example below.

```
// import React from the node-modules folder

import React from "react";
```