CS385 Mobile Application Development (Lecture 3)

Peter Mooney

Lecture 3 and Lecture 4 - menu

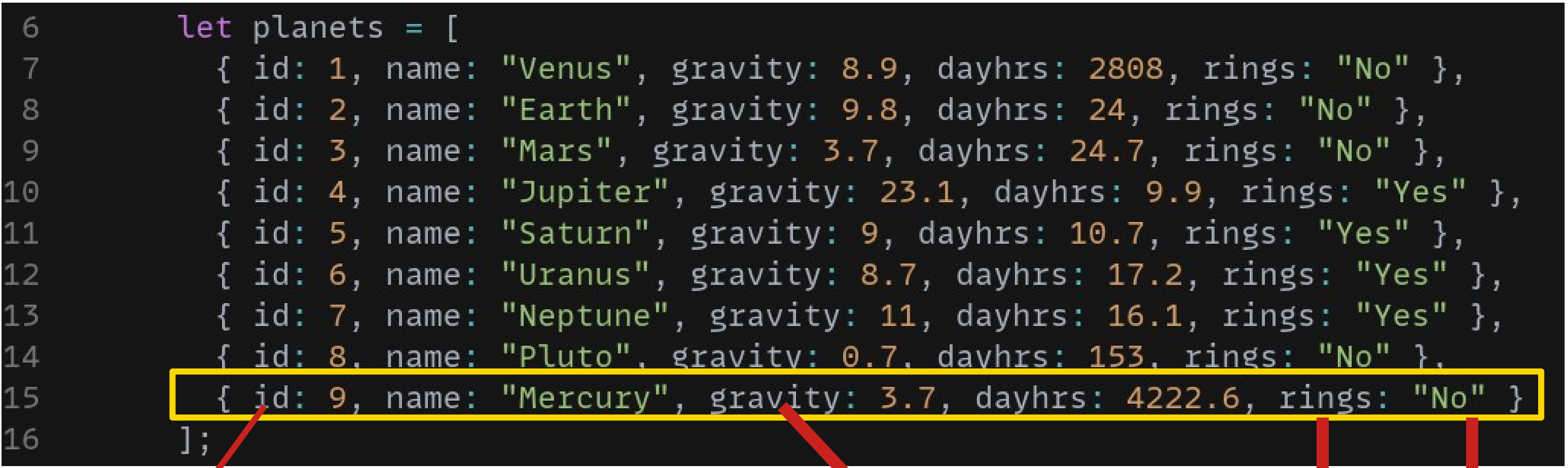
- Working with objects (map function again)
- Storing our object arrays in files.
- Using filtering with the map function when we have large arrays of objects.
- Popup Quiz – Topic 2
- Overview of the CS385 project
- Information about your first CS385 Lab

Some revision from Lecture 1 and Lecture 2 (Arrays of Objects)

A first look at Objects and arrays in Javascript (continued from lecture 2)

- Javascript is natively capable of working with objects (like those you have seen in Java)
- Indeed, Javascript allows us to very easily define objects and manipulate them.
- **One very important concept in Mobile Application Development is working with ARRAYS of Objects. This will become more obvious in a few weeks time.**
- So let's look at some objects and then move to arrays.

Objects – understanding properties and values



```
let planets = [
  { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808, rings: "No" },
  { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24, rings: "No" },
  { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7, rings: "No" },
  { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9, rings: "Yes" },
  { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7, rings: "Yes" },
  { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2, rings: "Yes" },
  { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1, rings: "Yes" },
  { id: 8, name: "Pluto", gravity: 0.7, dayhrs: 153, rings: "No" },
  { id: 9, name: "Mercury", gravity: 3.7, dayhrs: 4222.6, rings: "No" }
];
```

This is a single object

Property

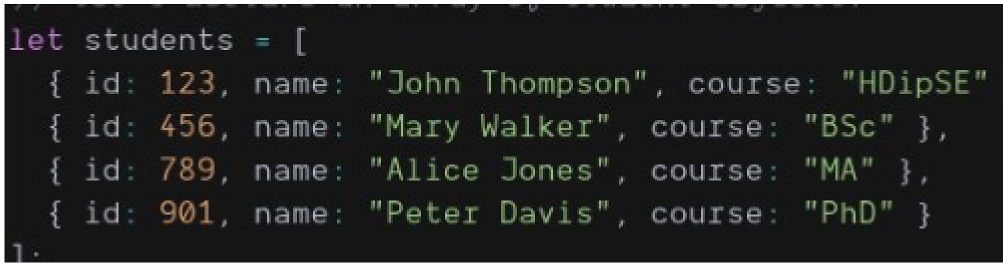
Value

Usually, in an array of objects, there is one property (in this case **`id`**) which is UNIQUE – if you compare it to other objects.

It has 5 PROPERTIES or ATTRIBUTES with the names: **`id, name, gravity, dayhrs, rings`**

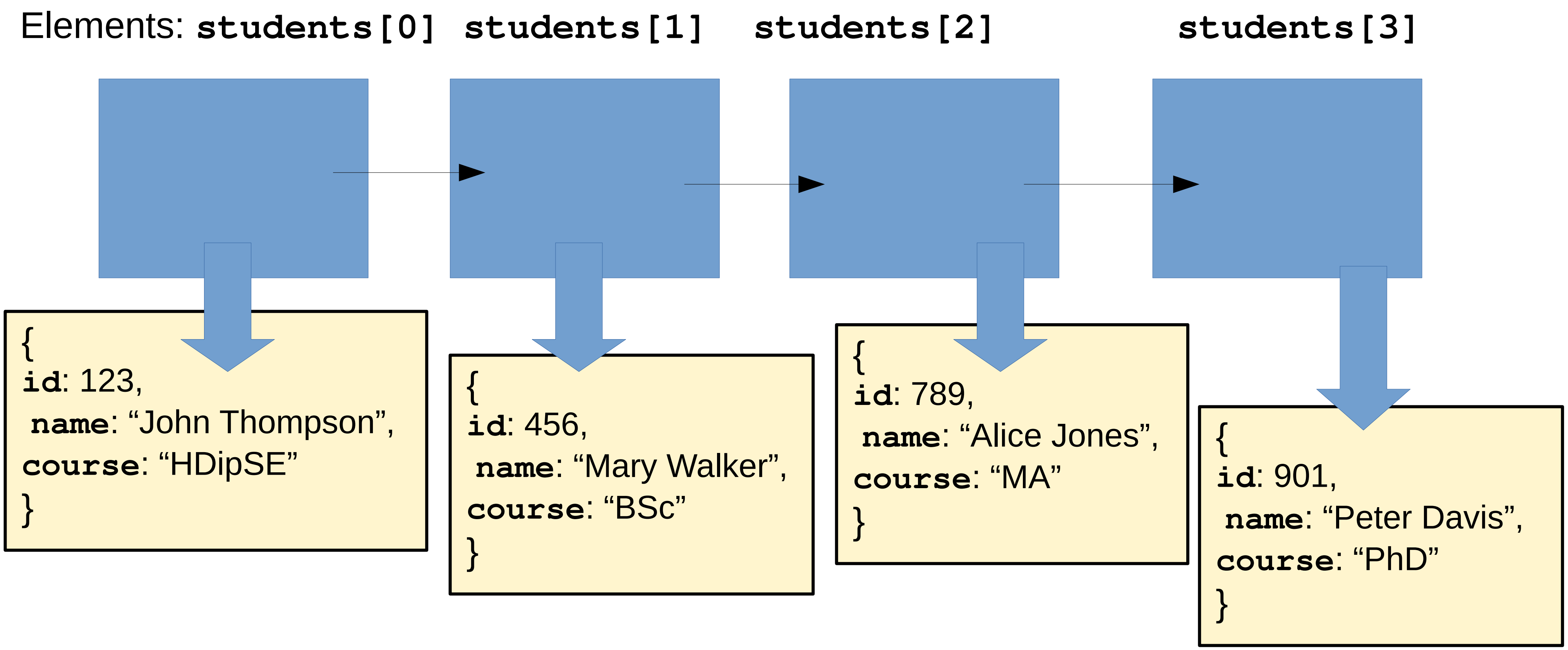
PROPERTY names can be enclosed in double quotes if you wish

Array of objects (visual)



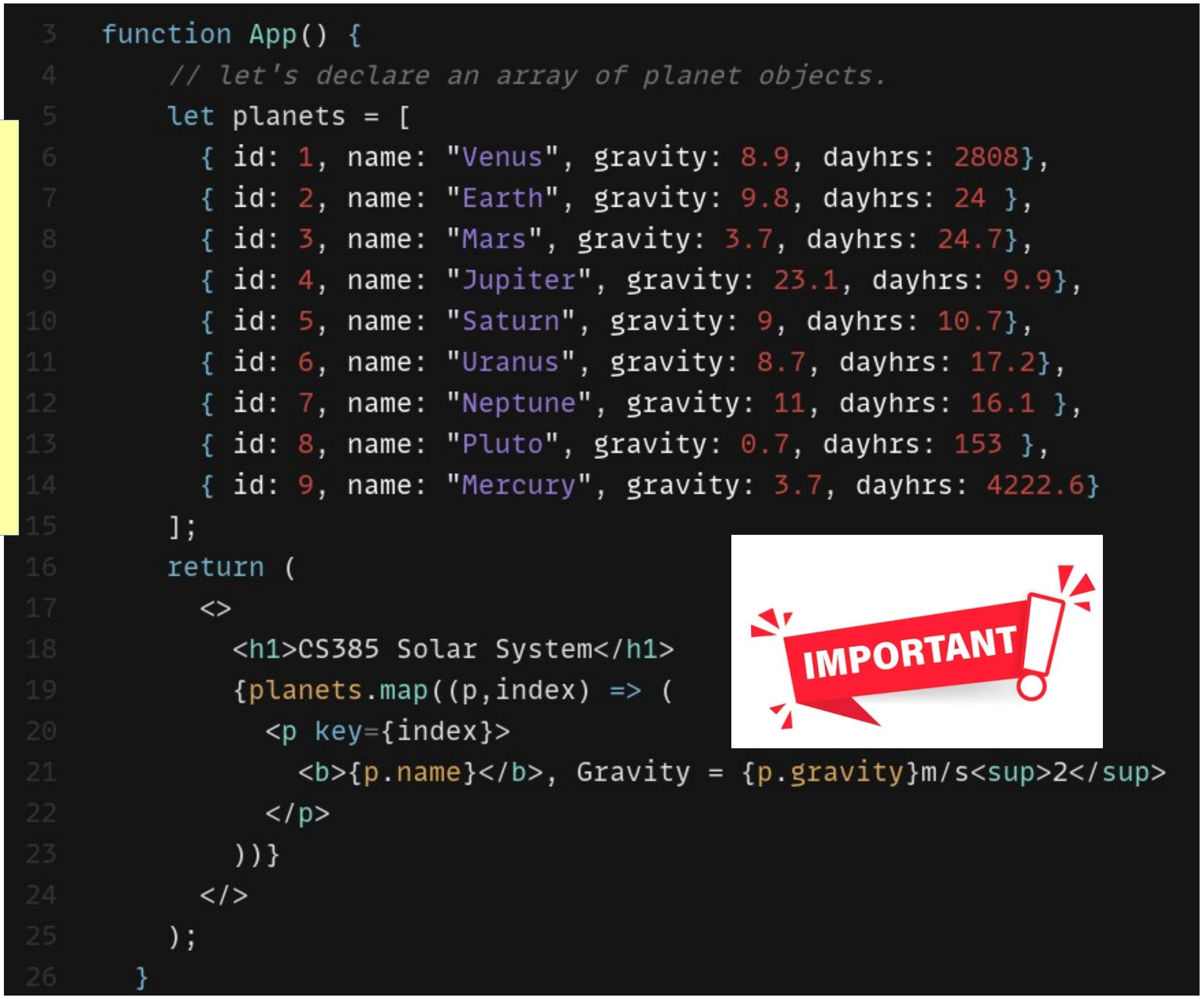
```
let students = [
  { id: 123, name: "John Thompson", course: "HDipSE"
  { id: 456, name: "Mary Walker", course: "BSc" },
  { id: 789, name: "Alice Jones", course: "MA" },
  { id: 901, name: "Peter Davis", course: "PhD" }
];
```

- The array of objects called students – an array is always declared using square brackets.
- Each object is an array element – we can add more if we need


Elements: **students[0]**
students[1]
students[2]
students[3]
{ **id**: 123, **name**: “John Thompson”, **course**: “HDipSE” }
{ **id**: 456, **name**: “Mary Walker”, **course**: “BSc” }
{ **id**: 789, **name**: “Alice Jones”, **course**: “MA” }
{ **id**: 901, **name**: “Peter Davis”, **course**: “PhD” }

Understanding the map function applied to an array

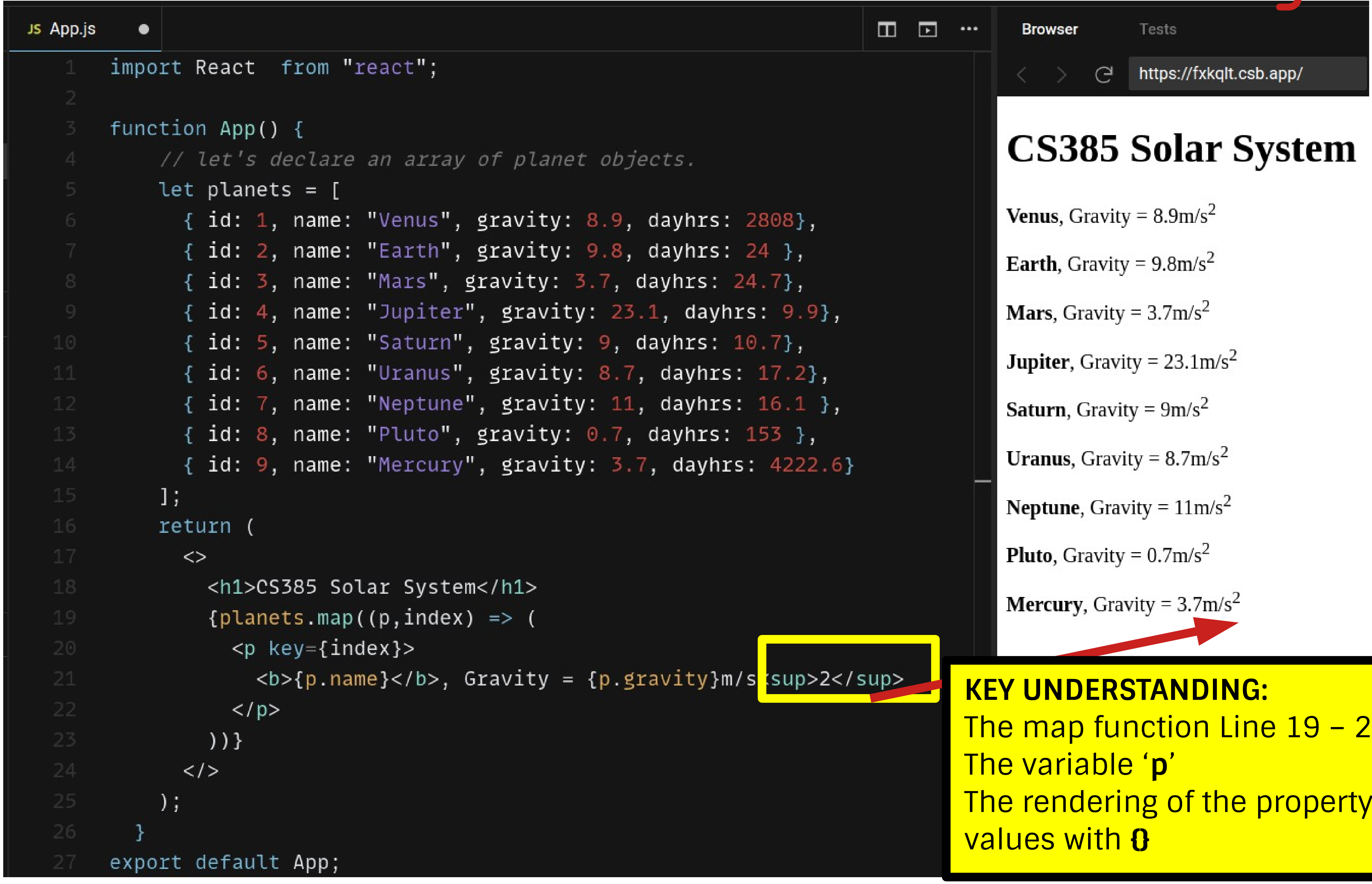
- The map function is used to render the array.
- **It is analogous to a for loop – but we NEVER specify the length of the array. The map function always starts at the first element and moves to the end.**
- Notice how we use the curly bracket notation to print out each **`property`** value of each object.
- **The variable p** is the name we give each object as the map function moves from element to element in the array



```
function App() {
    // let's declare an array of planet objects.
    let planets = [
      { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
      { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
      { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
      { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
      { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
      { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
      { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
      { id: 8, name: "Pluto", gravity: 0.7, dayhrs: 153 },
      { id: 9, name: "Mercury", gravity: 3.7, dayhrs: 4222.6}
    ];
    return (
      <>
        <h1>CS385 Solar System</h1>
        {planets.map((p,index) => (
          <p key={index}>
            <b>{p.name}</b>, Gravity = {p.gravity}m/s<sup>2</sup>
          </p>
        ))}
      </>
    );
  }
```

Array of objects (The planets)

Source code: Lecture3-Planets.js



```
1   import React  from "react";
2
3   function App() {
4       // let's declare an array of planet objects.
5       let planets = [
6         { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
7         { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
8         { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
9         { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
10        { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
11        { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
12        { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
13        { id: 8, name: "Pluto", gravity: 0.7, dayhrs: 153 },
14        { id: 9, name: "Mercury", gravity: 3.7, dayhrs: 4222.6}
15      ];
16      return (
17        <>
18          <h1>CS385 Solar System</h1>
19          {planets.map((p,index) => (
20            <p key={index}>
21              <b>{p.name}</b>, Gravity = {p.gravity}m/s<sup>2</sup>
22            </p>
23          ))}
24        </>
25      );
26    }
27  export default App;
```

CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

Uranus, Gravity = 8.7m/s^2

Neptune, Gravity = 11m/s^2

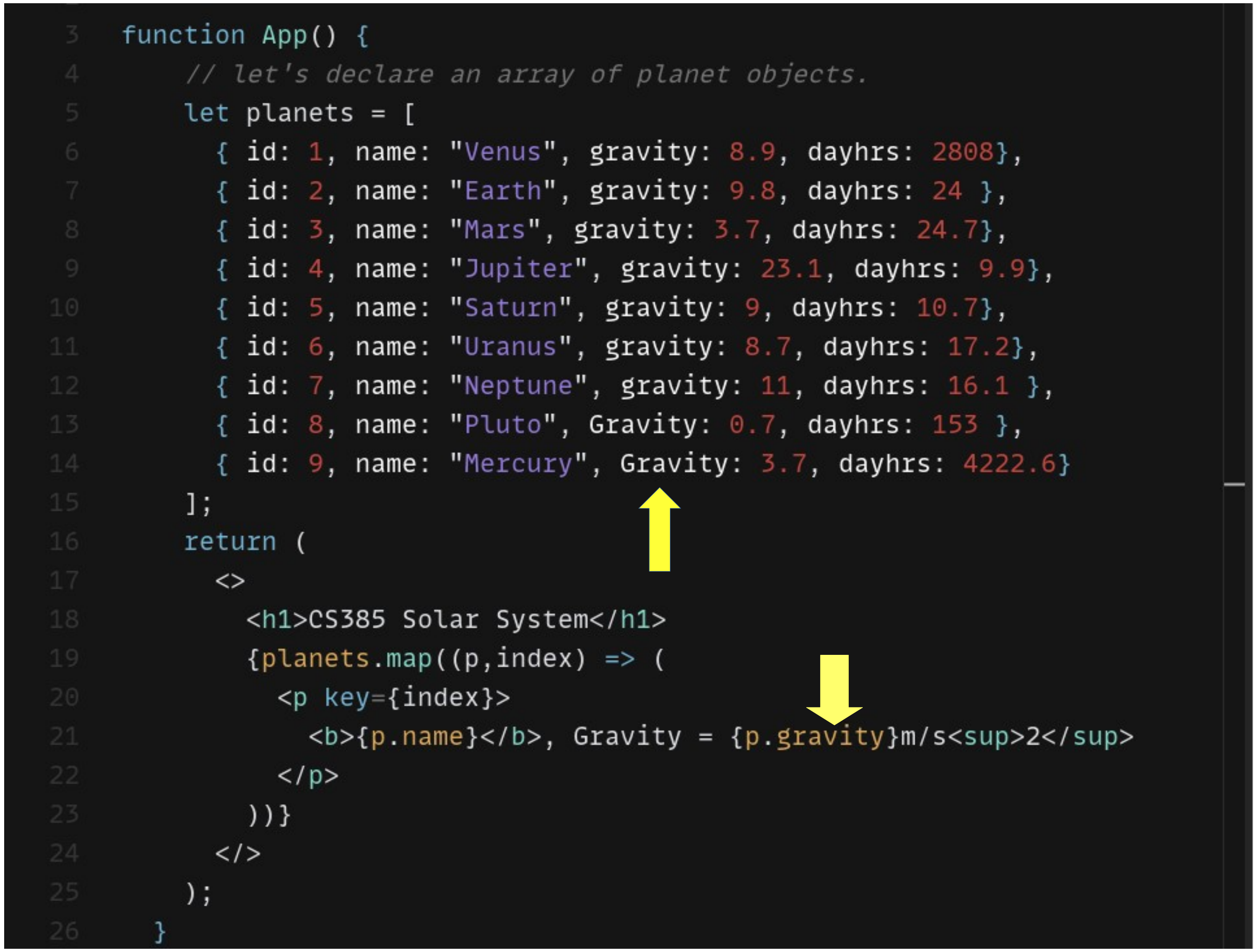
Pluto, Gravity = 0.7m/s^2

Mercury, Gravity = 3.7m/s^2

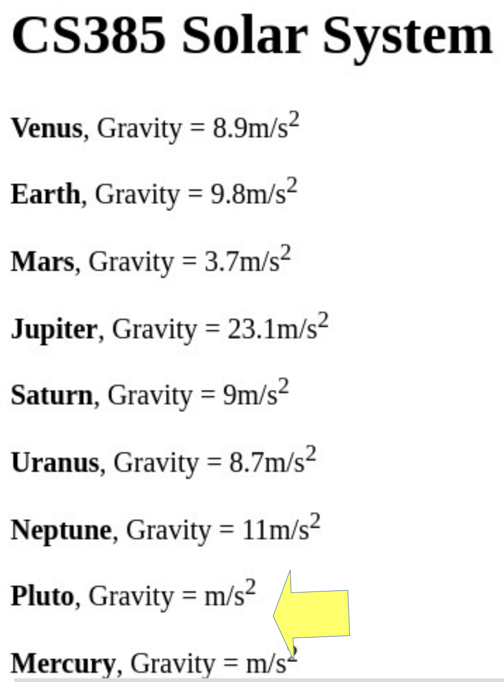
KEY UNDERSTANDING:
The map function Line 19 – 23
The variable ‘**p**’
The rendering of the property values with **{}**

Object property names are always case sensitive.

- Property names are ALWAYS case sensitive. Look at the example of "Gravity" below



```
function App() {
    // let's declare an array of planet objects.
    let planets = [
      { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
      { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
      { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
      { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
      { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
      { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
      { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
      { id: 8, name: "Pluto", Gravity: 0.7, dayhrs: 153 },
      { id: 9, name: "Mercury", Gravity: 3.7, dayhrs: 4222.6}
    ];
    return (
      <>
        <h1>CS385 Solar System</h1>
        {planets.map((p,index) => (
          <p key={index}>
            <b>{p.name}</b>, Gravity = {p.gravity}m/s<sup>2</sup>
          </p>
        ))}
      </>
    );
  }
```



CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

Uranus, Gravity = 8.7m/s^2

Neptune, Gravity = 11m/s^2

Pluto, Gravity = m/s^2

Mercury, Gravity = m/s^2

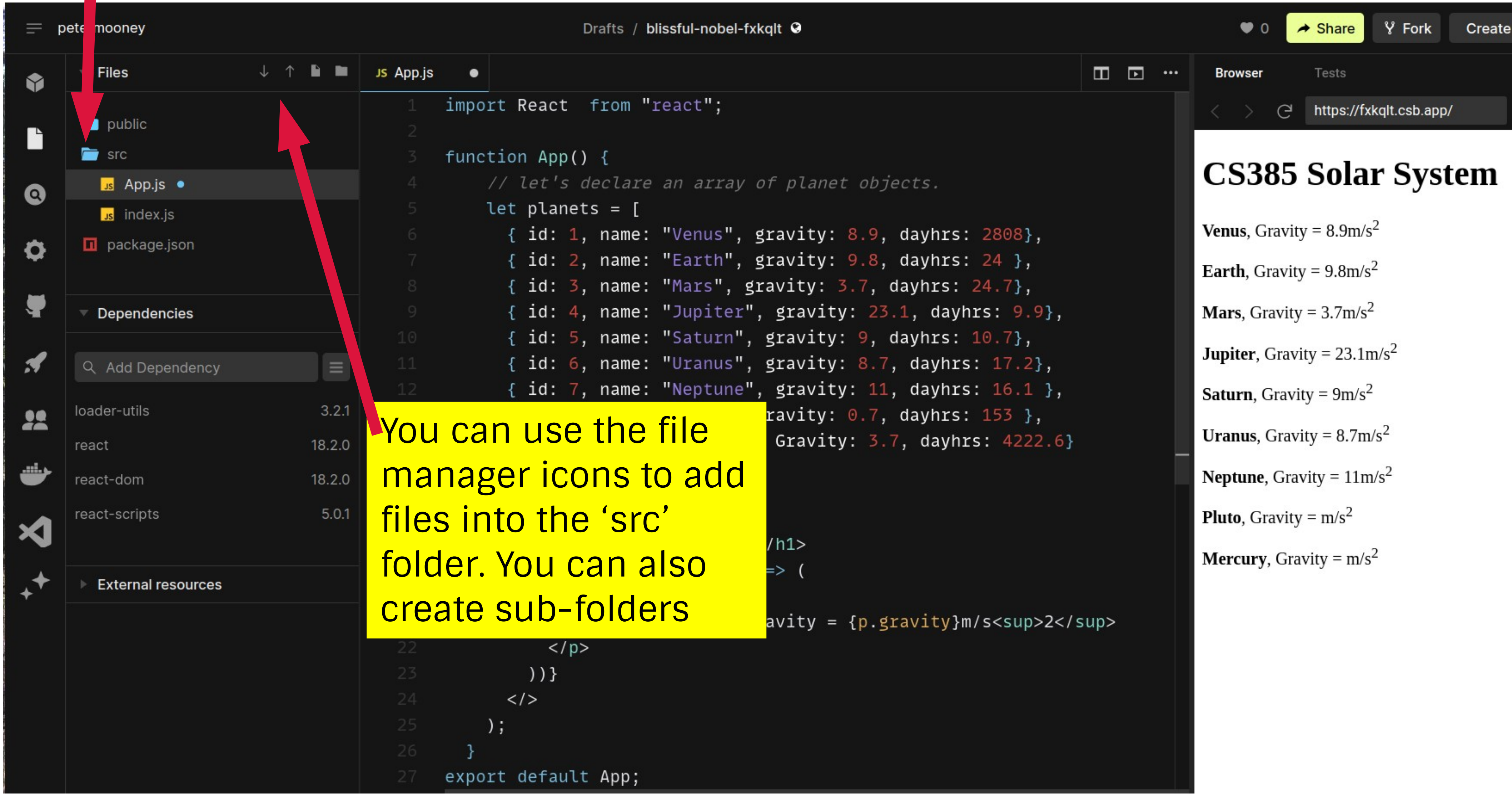
React/Javascript just ignores the Gravity (with capital G) property as it does not exist. Therefore it cannot be printed or rendered

The map function and arrays

- If you can work to understand the previous example then you're well on your way to grasping one of the most useful concepts in Javascript/React. The importance of the map function and arrays cannot be over emphasised
- The map function will work regardless of the number of elements in the array.
- We, as the programmer, decides how we render the values of each property (in our example we put the name property value in bold text)

Understanding the structure of a React class and codesandbox

The ‘src’ folder: This is where ALL of your source code, data, and media files are stored for your ‘App’



loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

```
import React  from "react";

function App() {
    // let's declare an array of planet objects.
    let planets = [
      { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
      { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
      { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
      { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
      { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
      { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
      { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
                                ravity: 0.7, dayhrs: 153 },
                                 Gravity: 3.7, dayhrs: 4222.6}



                              /h1>
                              => (

                              avity = {p.gravity}m/s<sup>2</sup>
              </p>
            ))}
          </>
      );
  }
export default App;
```

CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

Uranus, Gravity = 8.7m/s^2

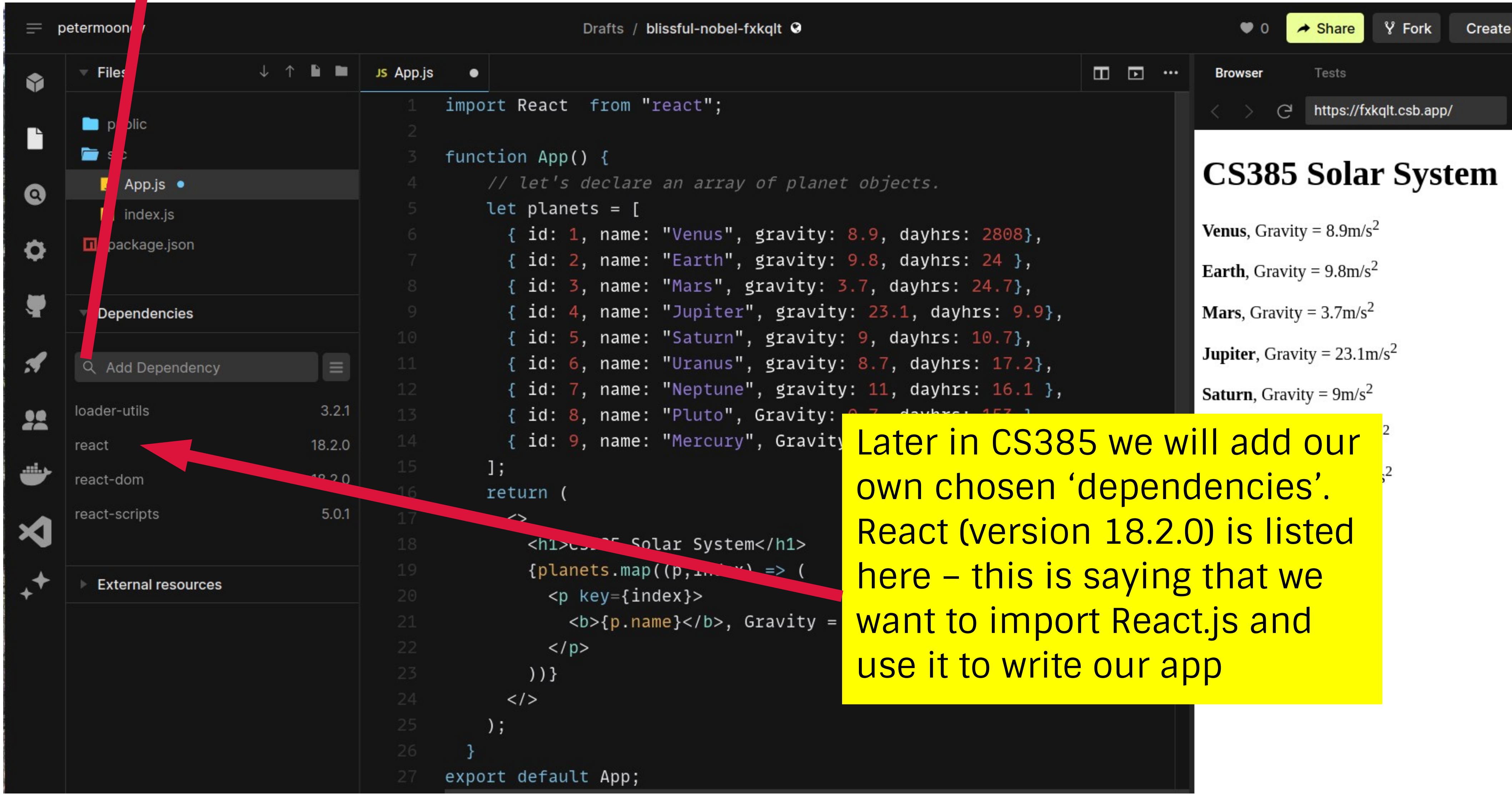
Neptune, Gravity = 11m/s^2

Pluto, Gravity = m/s^2

Mercury, Gravity = m/s^2

You can use the file manager icons to add files into the ‘src’ folder. You can also create sub-folders

Dependencies – this is where the libraries or external source code your app needs are linked to your ‘app’



loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

```
import React  from "react";

function App() {
    // let's declare an array of planet objects.
    let planets = [
      { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
      { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
      { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
      { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
      { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
      { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
      { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
      { id: 8, name: "Pluto", Gravity:
      { id: 9, name: "Mercury", Gravity
    ];
    return (
      <>
        <h1>CS385 Solar System</h1>
        {planets.map((p, index) => (
          <p key={index}>
            <b>{p.name}</b>, Gravity =
          </p>
        ))}
      </>
    );
  }
export default App;
```

CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

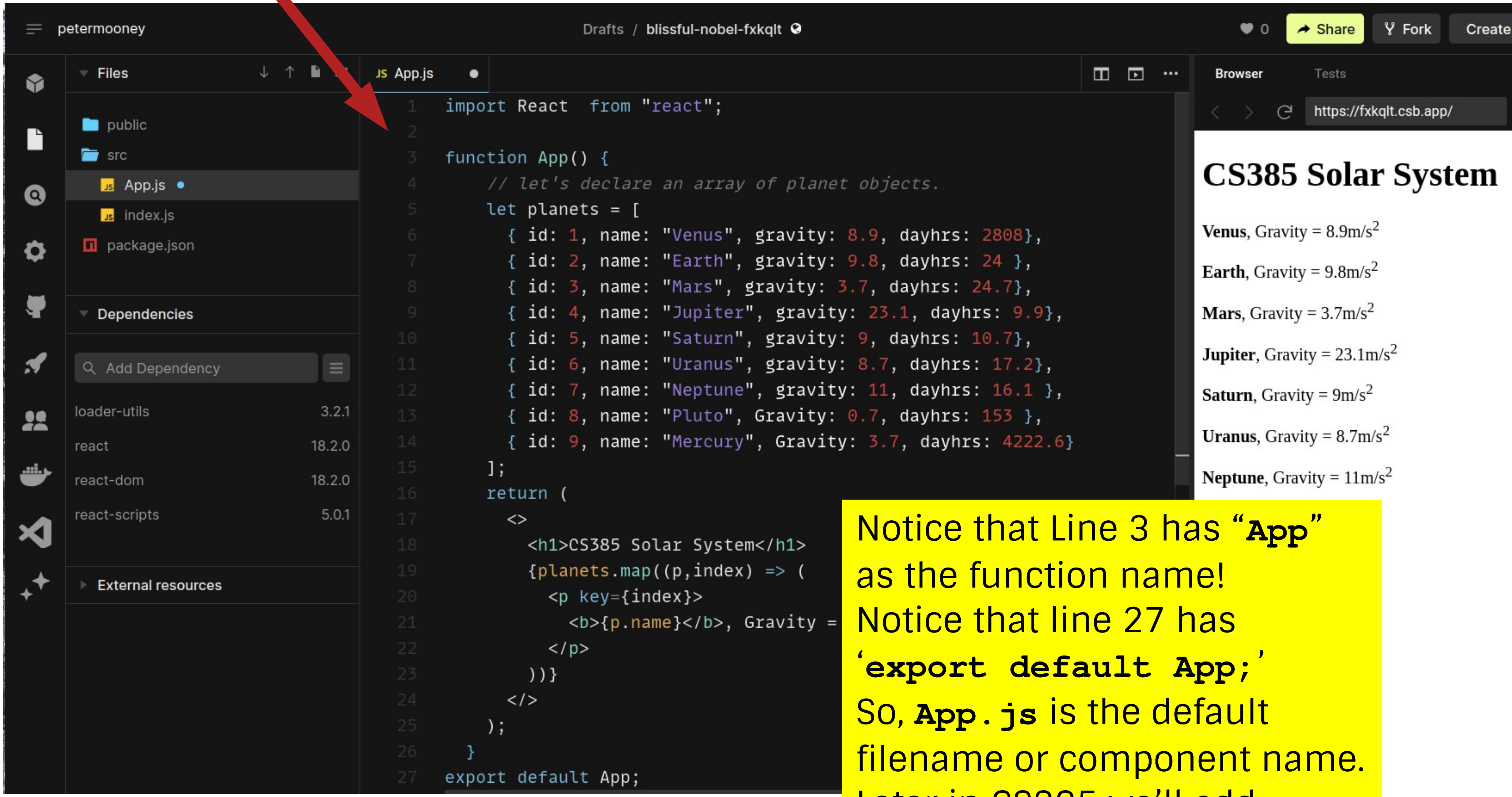
Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

Later in CS385 we will add our own chosen ‘dependencies’. React (version 18.2.0) is listed here – this is saying that we want to import React.js and use it to write our app

“App.js” – this is the filename of the JS (JavaScript) file containing your app code



petermooney — Drafts / blissful-nobel-fxkqlt — 0 — Share — Fork — Create

Files
- public
- src
  - App.js
  - index.js
- package.json

Dependencies

Add Dependency

loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

External resources

App.js

```
1  import React  from "react";
2
3  function App() {
4      // let's declare an array of planet objects.
5      let planets = [
6        { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
7        { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
8        { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
9        { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
10       { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
11       { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
12       { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
13       { id: 8, name: "Pluto", Gravity: 0.7, dayhrs: 153 },
14       { id: 9, name: "Mercury", Gravity: 3.7, dayhrs: 4222.6}
15     ];
16     return (
17       <>
18         <h1>CS385 Solar System</h1>
19         {planets.map((p,index) => (
20           <p key={index}>
21             <b>{p.name}</b>, Gravity =
22           </p>
23         ))}
24       </>
25     );
26   }
27 export default App;
```

Browser — Tests

https://fxkqlt.csb.app/

CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

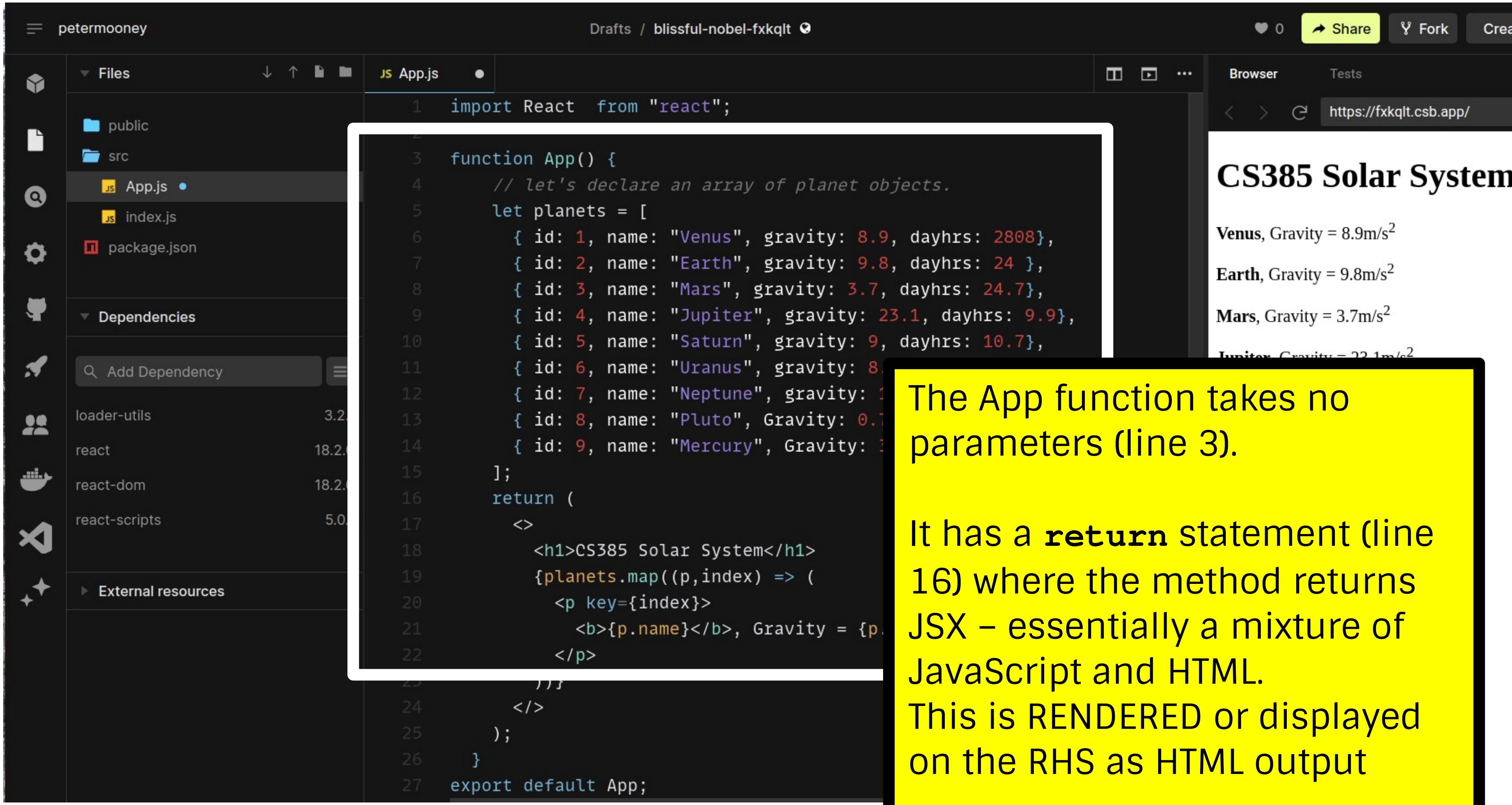
Saturn, Gravity = 9m/s^2

Uranus, Gravity = 8.7m/s^2

Neptune, Gravity = 11m/s^2

Notice that Line 3 has “`App`” as the function name! Notice that line 27 has ‘`export default App;`’ So, `App.js` is the default filename or component name. Later in CS385 we’ll add other `.js` files here.

`function App`: This is a function or component called ‘`App`’.



```
1   import React  from "react";
2
3   function App() {
4       // let's declare an array of planet objects.
5       let planets = [
6         { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
7         { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
8         { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
9         { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
10        { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
11        { id: 6, name: "Uranus", gravity: 8
12        { id: 7, name: "Neptune", gravity: 1
13        { id: 8, name: "Pluto", Gravity: 0.
14        { id: 9, name: "Mercury", Gravity: 3
15      ];
16      return (
17        <>
18          <h1>CS385 Solar System</h1>
19          {planets.map((p,index) => (
20            <p key={index}>
21              <b>{p.name}</b>, Gravity = {p.
22            </p>
23          ))}
24        </>
25      );
26    }
27  export default App;
```

CS385 Solar System

Venus, Gravity = 8.9m/s^2

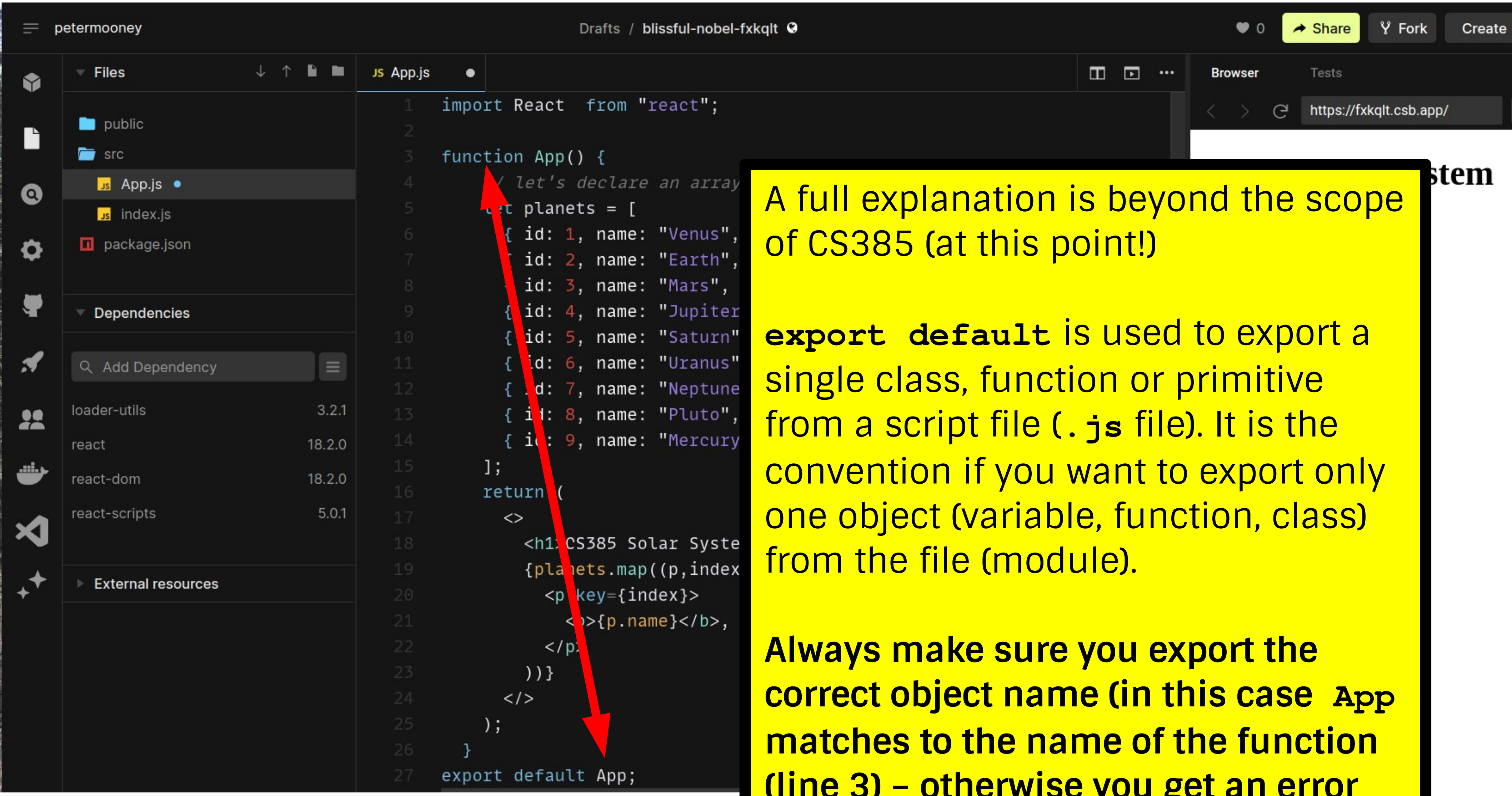
Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

The App function takes no parameters (line 3).

It has a **`return`** statement (line 16) where the method returns JSX – essentially a mixture of JavaScript and HTML.
This is RENDERED or displayed on the RHS as HTML output

`export default App;` Line 27 – essentially "exports" the App function for usage elsewhere



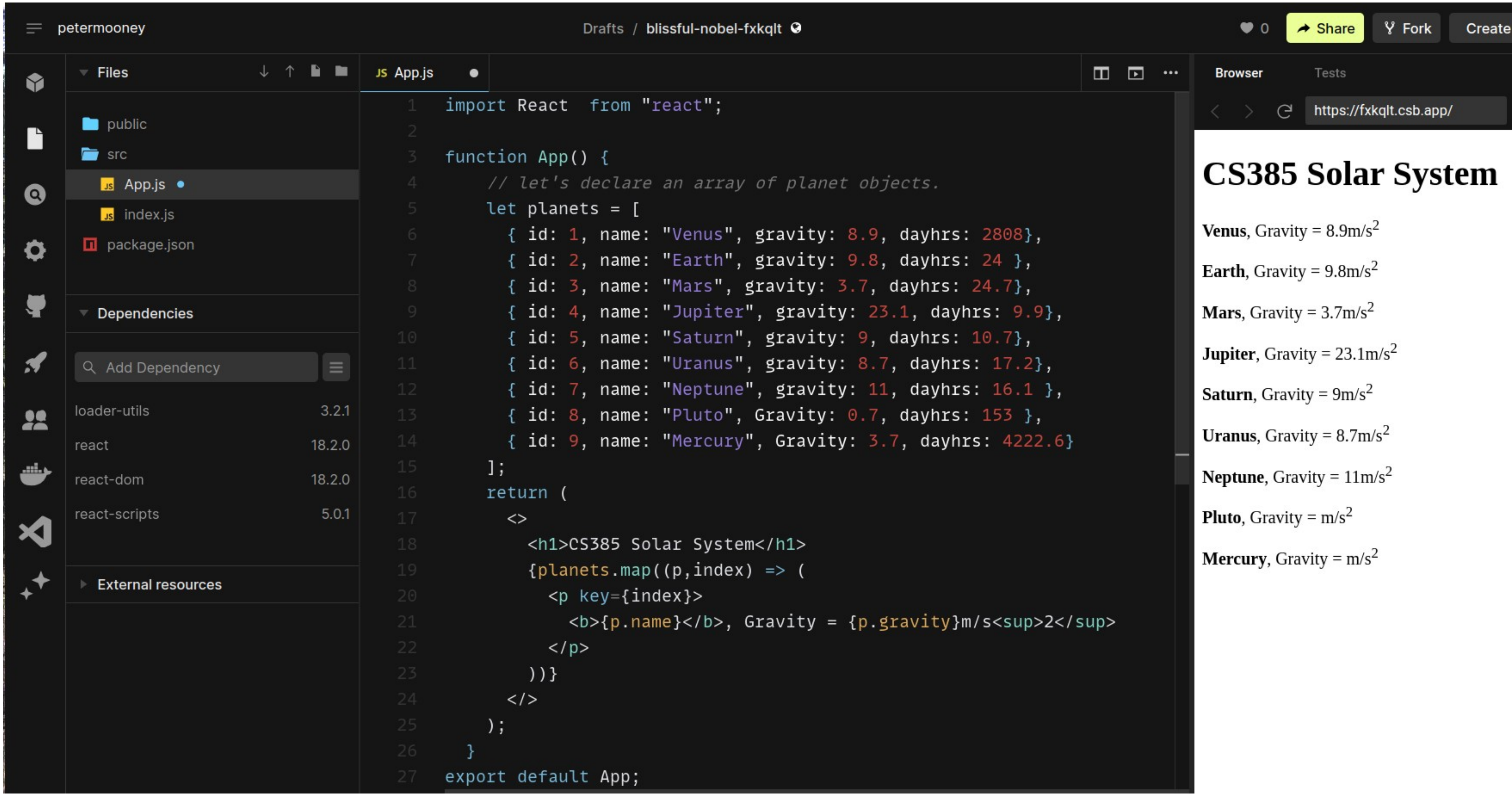
```
import React  from "react";

function App() {
    // let's declare an array
    let planets = [
      { id: 1, name: "Venus",
      { id: 2, name: "Earth",
      { id: 3, name: "Mars",
      { id: 4, name: "Jupiter
      { id: 5, name: "Saturn"
      { id: 6, name: "Uranus"
      { id: 7, name: "Neptune
      { id: 8, name: "Pluto",
      { id: 9, name: "Mercury
    ];
    return (
      <>
        <h1>CS385 Solar Syste
        {planets.map((p,index
          <p key={index}>
            <b>{p.name}</b>,
          </p>
        ))}
      </>
    );
  }
export default App;
```

A full explanation is beyond the scope of CS385 (at this point!)

`export default` is used to export a single class, function or primitive from a script file (`.js` file). It is the convention if you want to export only one object (variable, function, class) from the file (module).

Always make sure you export the correct object name (in this case `App` matches to the name of the function (line 3) – otherwise you get an error

What does the `<p key={index}>` actually mean in the map fuction?



loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

```
import React  from "react";

function App() {
    // let's declare an array of planet objects.
    let planets = [
      { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
      { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
      { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
      { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
      { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
      { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
      { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
      { id: 8, name: "Pluto", Gravity: 0.7, dayhrs: 153 },
      { id: 9, name: "Mercury", Gravity: 3.7, dayhrs: 4222.6}
    ];
    return (
      <>
        <h1>CS385 Solar System</h1>
        {planets.map((p,index) => (
          <p key={index}>
            <b>{p.name}</b>, Gravity = {p.gravity}m/s<sup>2</sup>
          </p>
        ))}
      </>
    );
  }
export default App;
```

CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

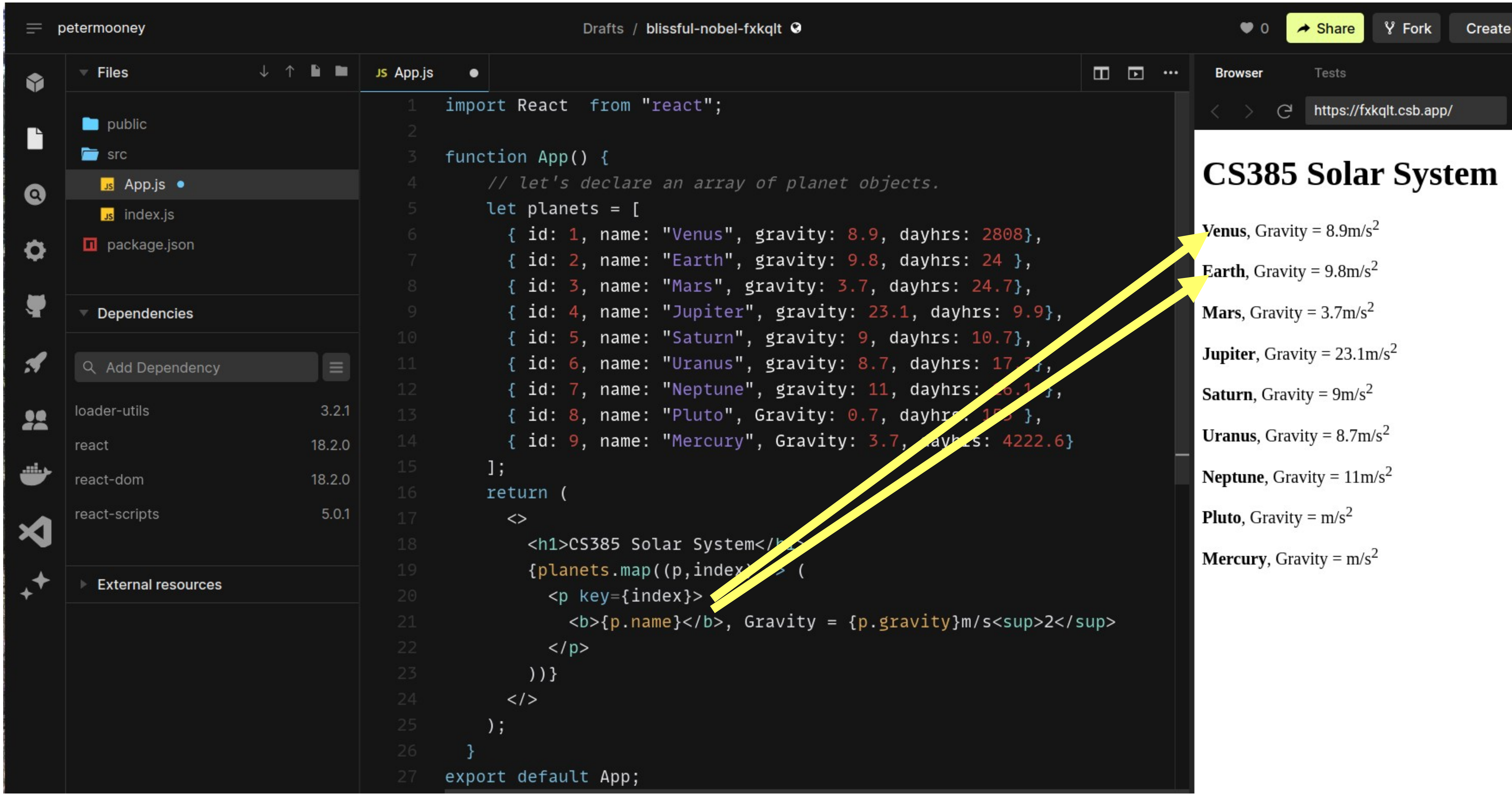
Uranus, Gravity = 8.7m/s^2

Neptune, Gravity = 11m/s^2

Pluto, Gravity = m/s^2

Mercury, Gravity = m/s^2

What does the `<p key={index}>` actually mean in the map fuction?



```
import React  from "react";

function App() {
    // let's declare an array of planet objects.
    let planets = [
      { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
      { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
      { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
      { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
      { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
      { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
      { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
      { id: 8, name: "Pluto", Gravity: 0.7, dayhrs: 153 },
      { id: 9, name: "Mercury", Gravity: 3.7, dayhrs: 4222.6}
    ];
    return (
      <>
        <h1>CS385 Solar System</h1>
        {planets.map((p,index) => (
          <p key={index}>
            <b>{p.name}</b>, Gravity = {p.gravity}m/s<sup>2</sup>
          </p>
        ))}
      </>
    );
  }
export default App;
```

CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

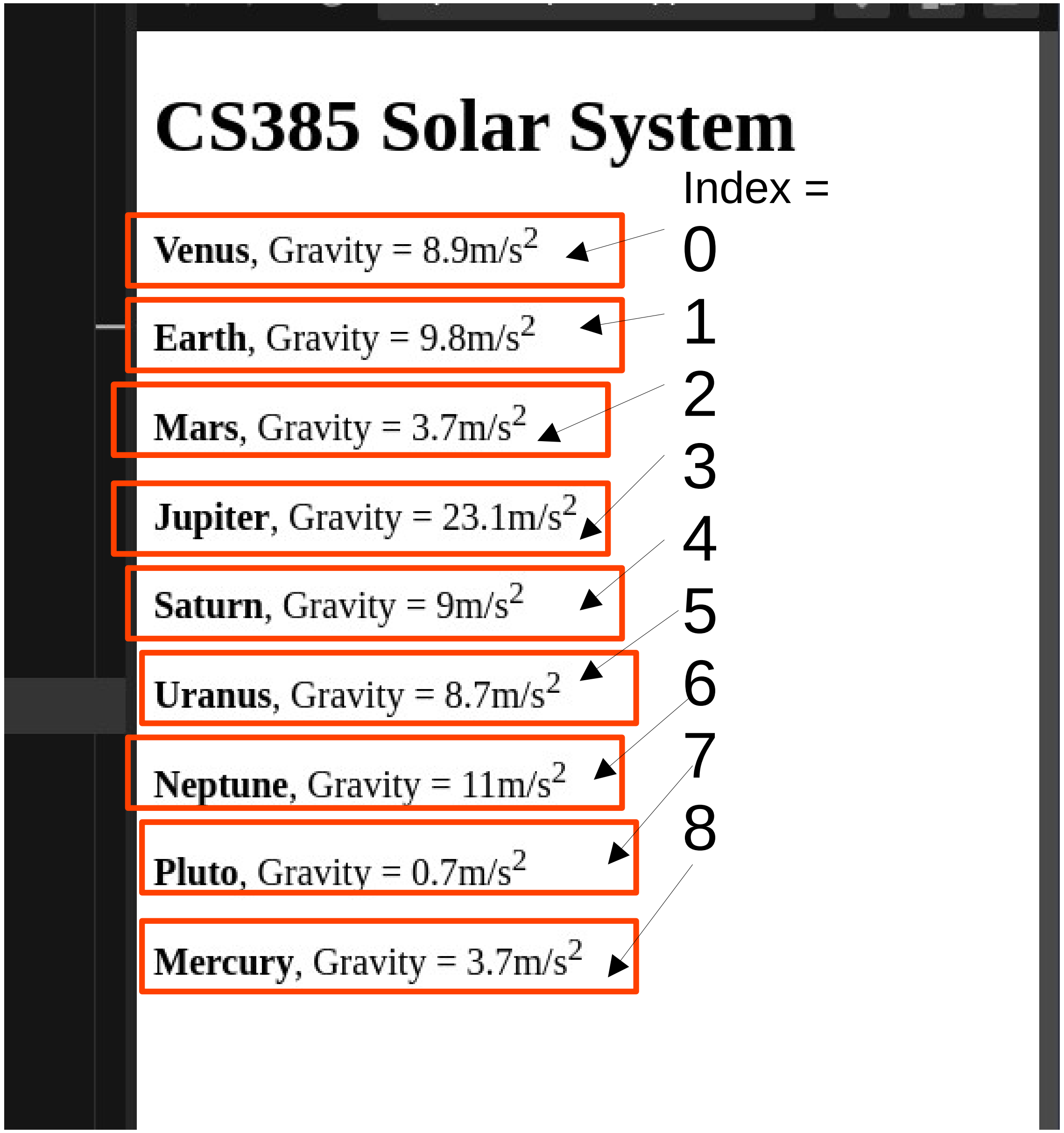
Uranus, Gravity = 8.7m/s^2

Neptune, Gravity = 11m/s^2

Pluto, Gravity = m/s^2

Mercury, Gravity = m/s^2

In the rendered app – think of each <p> (paragraph) as a single block or element



CS385 Solar System

Venus, Gravity = 8.9m/s^2

Earth, Gravity = 9.8m/s^2

Mars, Gravity = 3.7m/s^2

Jupiter, Gravity = 23.1m/s^2

Saturn, Gravity = 9m/s^2

Uranus, Gravity = 8.7m/s^2

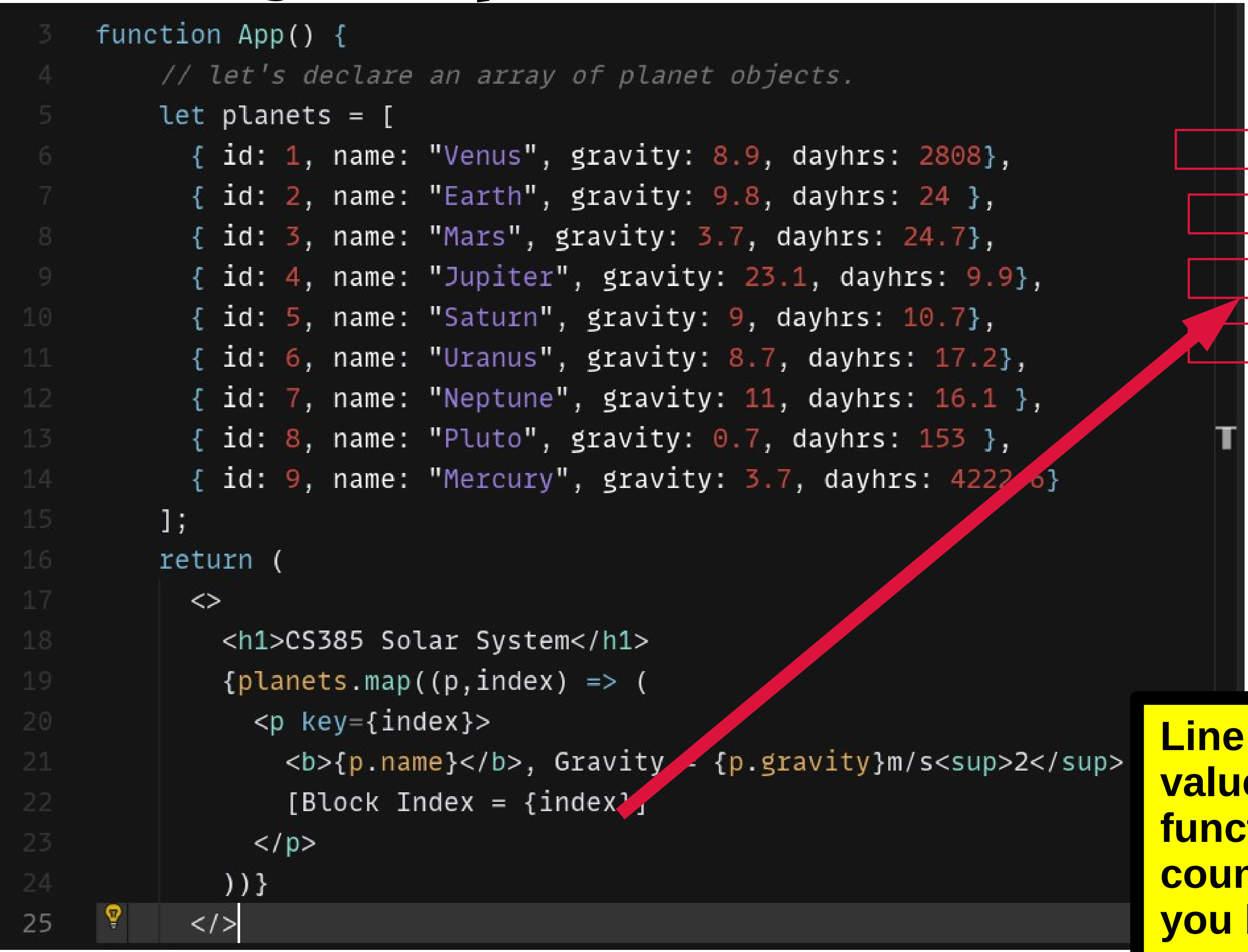
Neptune, Gravity = 11m/s^2

Pluto, Gravity = 0.7m/s^2

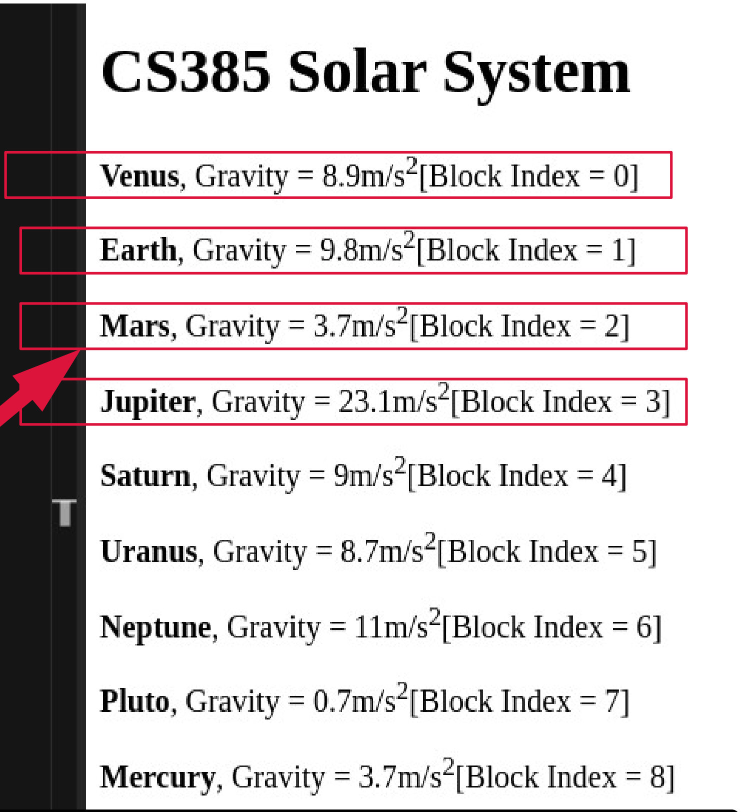
Mercury, Gravity = 3.7m/s^2

- For the browser to be able to control the movement of blocks or element (such as a user clicking to delete one) – **the DOM (Document Object Model) must keep track of the blocks or elements**.
- Hence **the need for a UNIQUE IDENTIFIER for each block**.

The MAP FUNCTION provides use with an index variable – and this just provides a count from 0



```
3   function App() {
4       // let's declare an array of planet objects.
5       let planets = [
6         { id: 1, name: "Venus", gravity: 8.9, dayhrs: 2808},
7         { id: 2, name: "Earth", gravity: 9.8, dayhrs: 24 },
8         { id: 3, name: "Mars", gravity: 3.7, dayhrs: 24.7},
9         { id: 4, name: "Jupiter", gravity: 23.1, dayhrs: 9.9},
10        { id: 5, name: "Saturn", gravity: 9, dayhrs: 10.7},
11        { id: 6, name: "Uranus", gravity: 8.7, dayhrs: 17.2},
12        { id: 7, name: "Neptune", gravity: 11, dayhrs: 16.1 },
13        { id: 8, name: "Pluto", gravity: 0.7, dayhrs: 153 },
14        { id: 9, name: "Mercury", gravity: 3.7, dayhrs: 4222.6}
15      ];
16      return (
17        <>
18          <h1>CS385 Solar System</h1>
19          {planets.map((p,index) => (
20            <p key={index}>
21              <b>{p.name}</b>, Gravity = {p.gravity}m/s<sup>2</sup>
22              [Block Index = {index}]
23            </p>
24          ))}
25        </>
```



CS385 Solar System

Venus, Gravity = 8.9m/s^2[Block Index = 0]

Earth, Gravity = 9.8m/s^2[Block Index = 1]

Mars, Gravity = 3.7m/s^2[Block Index = 2]

Jupiter, Gravity = 23.1m/s^2[Block Index = 3]

Saturn, Gravity = 9m/s^2[Block Index = 4]

Uranus, Gravity = 8.7m/s^2[Block Index = 5]

Neptune, Gravity = 11m/s^2[Block Index = 6]

Pluto, Gravity = 0.7m/s^2[Block Index = 7]

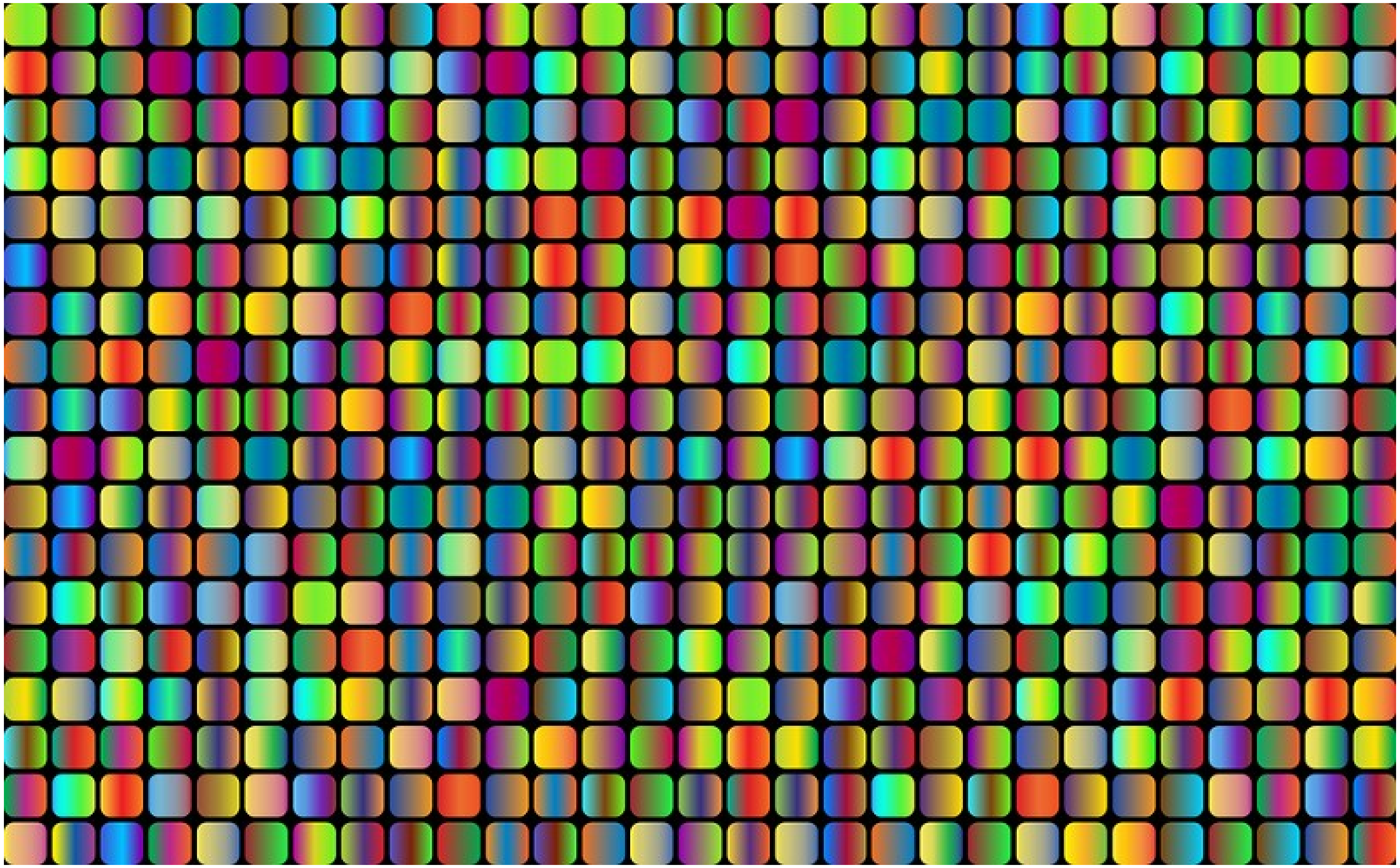
Mercury, Gravity = 3.7m/s^2[Block Index = 8]

Line 22 – actually renders the value of index from the map function. Think of it as just a counter variable provided to you by REACT

Recap

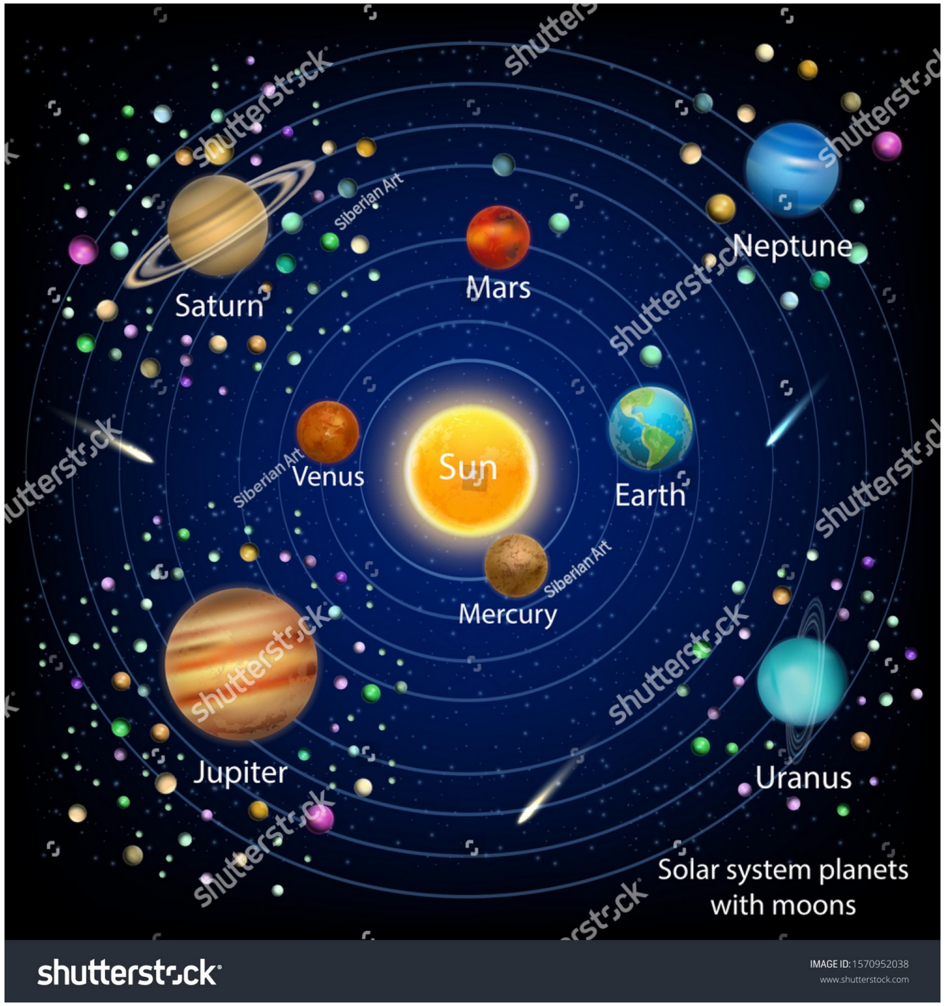
- **Property names MUST stay the same** (it just makes your coding life easier)
- **The map function provides you with its own internal "index"** or counter value for BLOCKs
- **The map function can render or print any valid array of JSON objects**.
- **Try out the Planets code** (available on Moodle)

Working with large arrays of Objects in a mobile application



Large arrays of objects in Javascript – how do we deal with these situations?

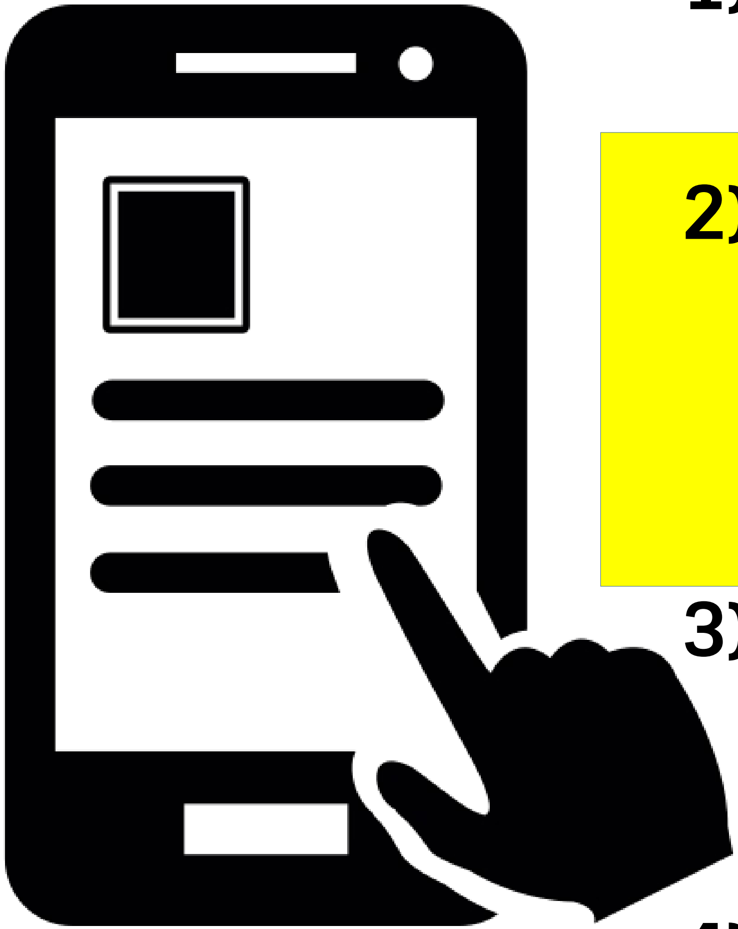
- In the planets example we had 9 objects. This isn't a big array but it takes up a lot of lines of code in our `App.js` file.
- What if we wanted to create an array of objects representing all of the moons or satellites of planets in our solar system?
- This would be over 200 objects – that is a big array and would take 200 lines of code!



We need to be able to process large arrays in our mobile applications

- In our usage of applications on our mobile devices we interact with arrays of objects ALL THE TIME. **For example:**
  - **Scrolling through your contacts**
  - **Scrolling message threads in social apps** such as WhatsApp (each thread is an array of object where each message is an object)
  - **News aggregator feeds** – a constantly updating array of news objects which you can click on …
  - **List of items in an online shop….**

There are FOUR ways (with variations) on how you import data into your mobile app



1) **Use an array of objects within your `App.js` class** (or similar) ✓
2) **Use a local file** (within your app source) to hold or store an array of objects which is accessible to **App.js**
3) **Use an external API** (Application Programming Interface) – later in CS385
4) **Use an external database** - later in CS385

End – Lecture 3