React JS 16.13.1

Team Members

- Disha Dhingra
- Shubhra Kushal
- Ayush Jain
- Himanshi Gulati

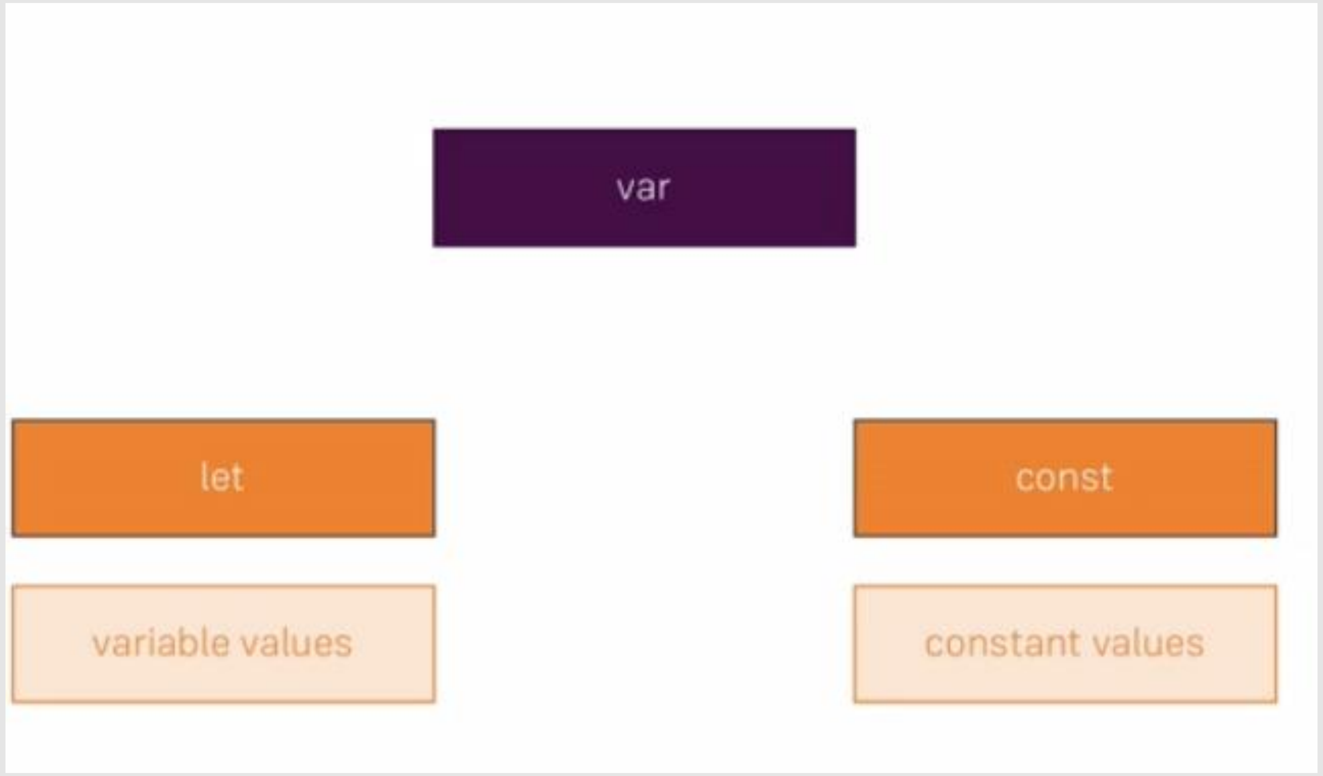
Agenda

1. Next-Gen JavaScript
2. Single Page Applications vs Multi page Applications.
3. What is React?
4. Local Setup of React application
5. Initial setup code structure walkthrough
6. Class based components & Functional Components
7. Stateful & Stateless components
8. JSX
9. Props
10. States
11. Lifecycle Components
12. Pure Components
13. Virtual DOM

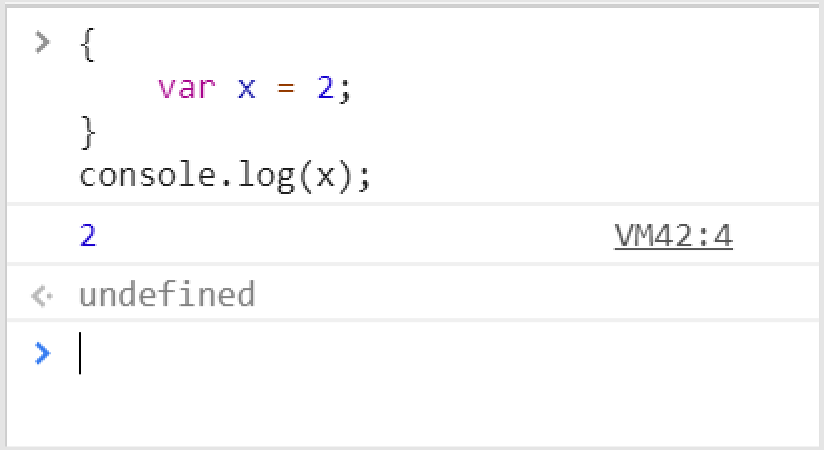
Next-Gen JavaScript

let & const

These
two keywords provide **Block Scope.**
Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.



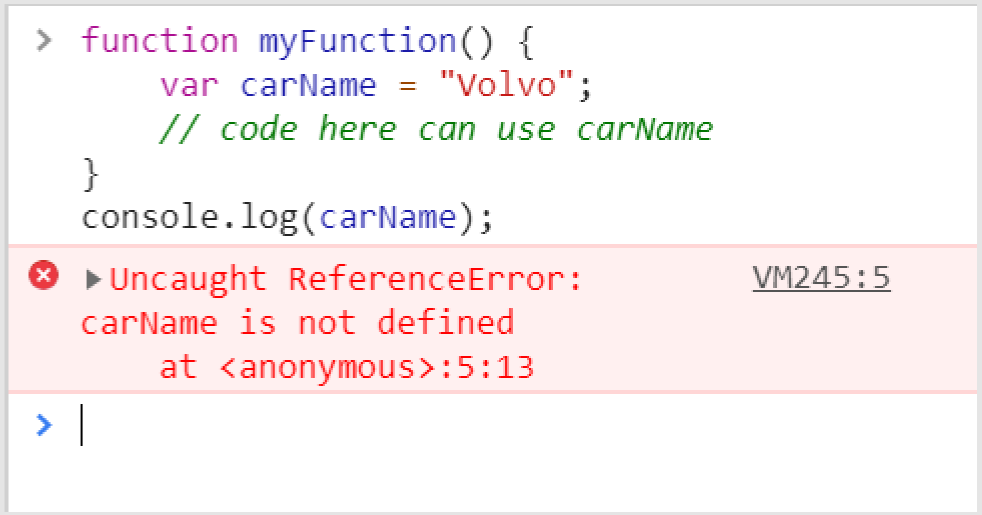
Global Scope



```
> {
      var x = 2;
  }
  console.log(x);
  2                                  VM42:4
<· undefined
> |
```

Variables declared **Globally** (outside any function) have **Global Scope**.

Function Scope

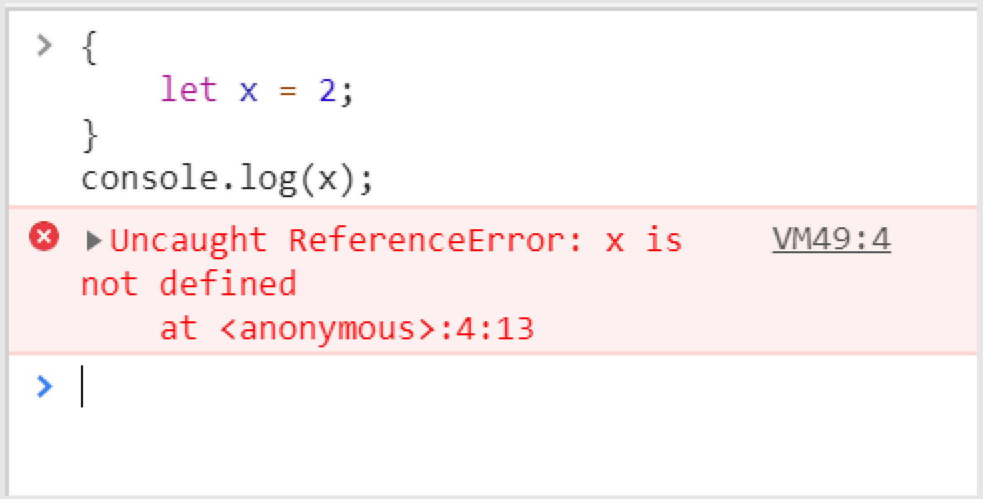


```
> function myFunction() {
      var carName = "Volvo";
      // code here can use carName
  }
  console.log(carName);
```

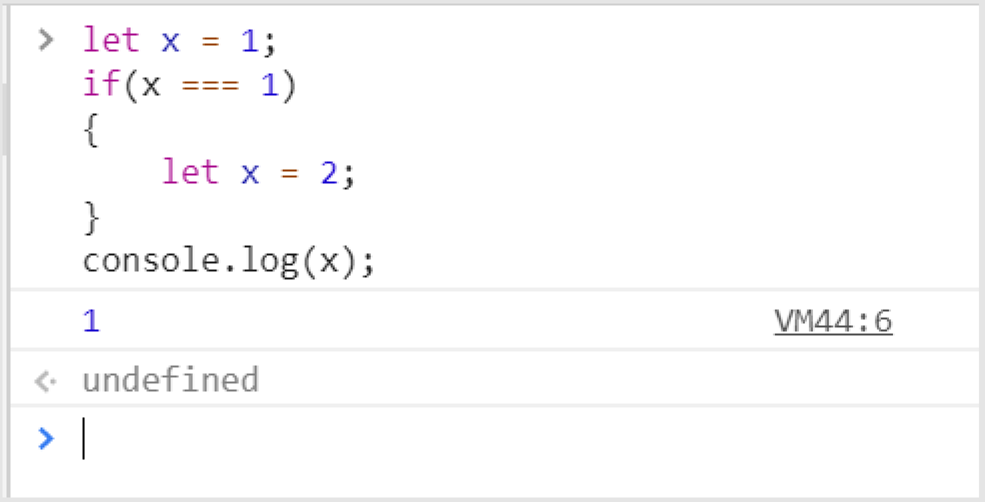
```
✖ ▶Uncaught ReferenceError:                VM245:5
  carName is not defined
      at <anonymous>:5:13
>
```

Variables declared **Locally** (inside a function) have **Function Scope**.

Block Scope



```
> {
      let x = 2;
  }
  console.log(x);
✖ ▶Uncaught ReferenceError: x is      VM49:4
  not defined
      at <anonymous>:4:13
> |
```

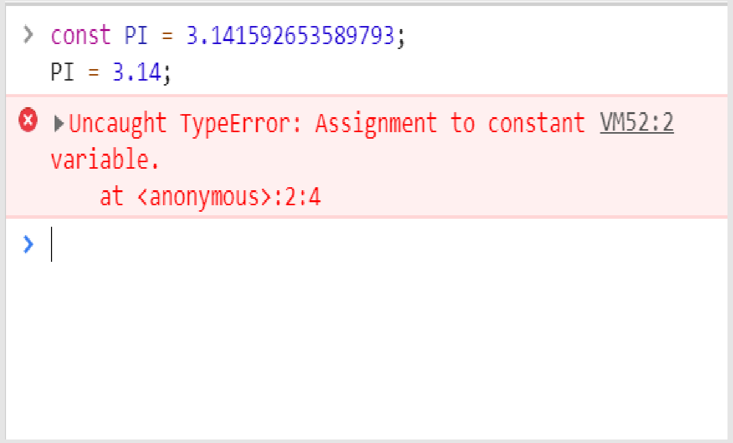


```
> let x = 1;
  if(x === 1)
  {
      let x = 2;
  }
  console.log(x);
  1                                   VM44:6
< undefined
> |
```

let has block scope.

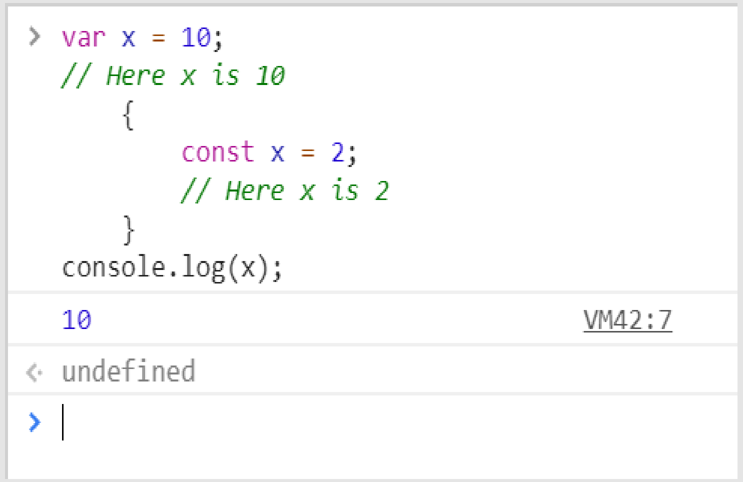
Const

Variables defined with const behave like let variables, except they cannot be reassigned:



```
> const PI = 3.141592653589793;
  PI = 3.14;
⊗ ▶Uncaught TypeError: Assignment to constant VM52:2
  variable.
      at <anonymous>:2:4
> |
```

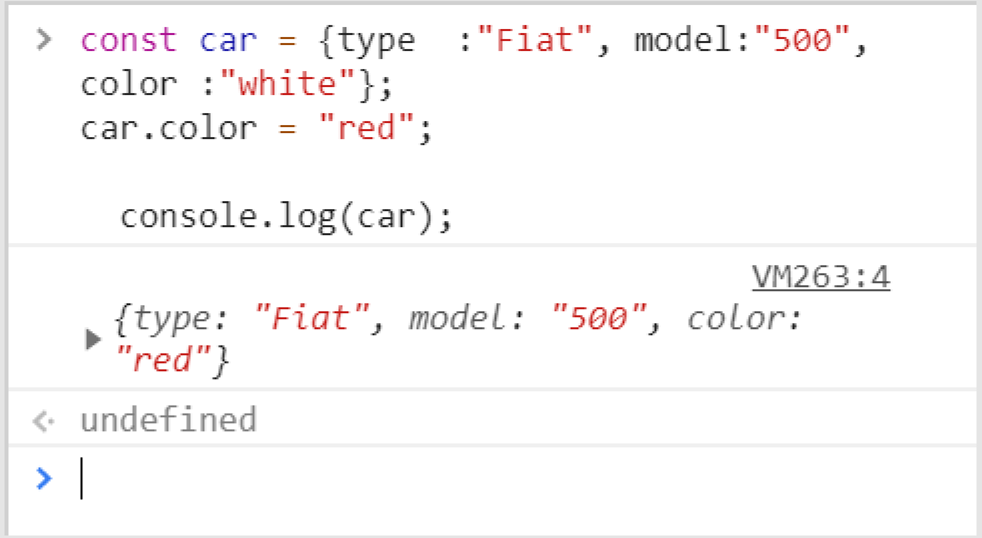
Cannot reassign a value to a constant variable.



```
> var x = 10;
  // Here x is 10
      {
          const x = 2;
          // Here x is 2
      }
  console.log(x);
  10                                        VM42:7
< undefined
> |
```

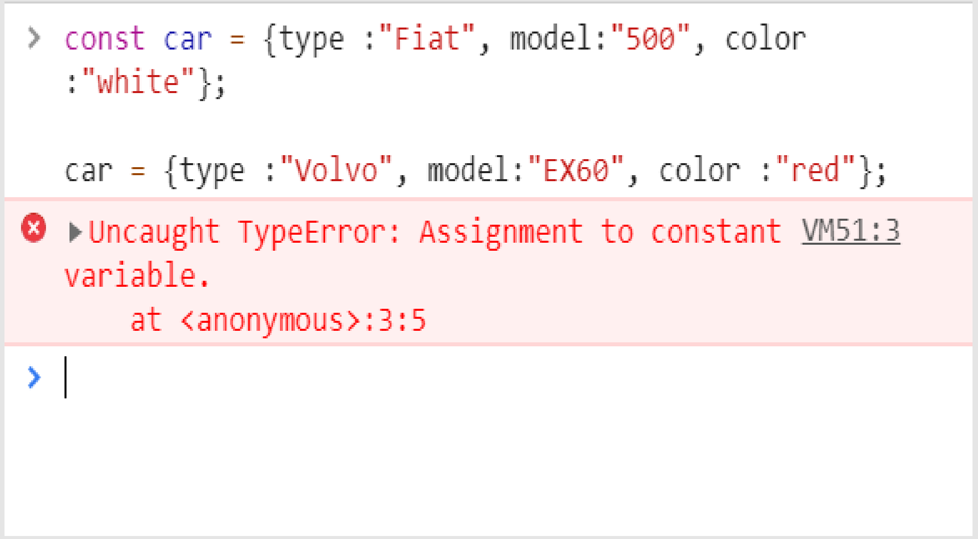
Declaring a variable with const is similar to let when it comes to **Block Scope**.

Const Objects



```
> const car = {type  :"Fiat", model:"500",
  color :"white"};
  car.color = "red";

   console.log(car);
                                              VM263:4
  ▶ {type: "Fiat", model: "500", color:
    "red"}
<· undefined
> |
```

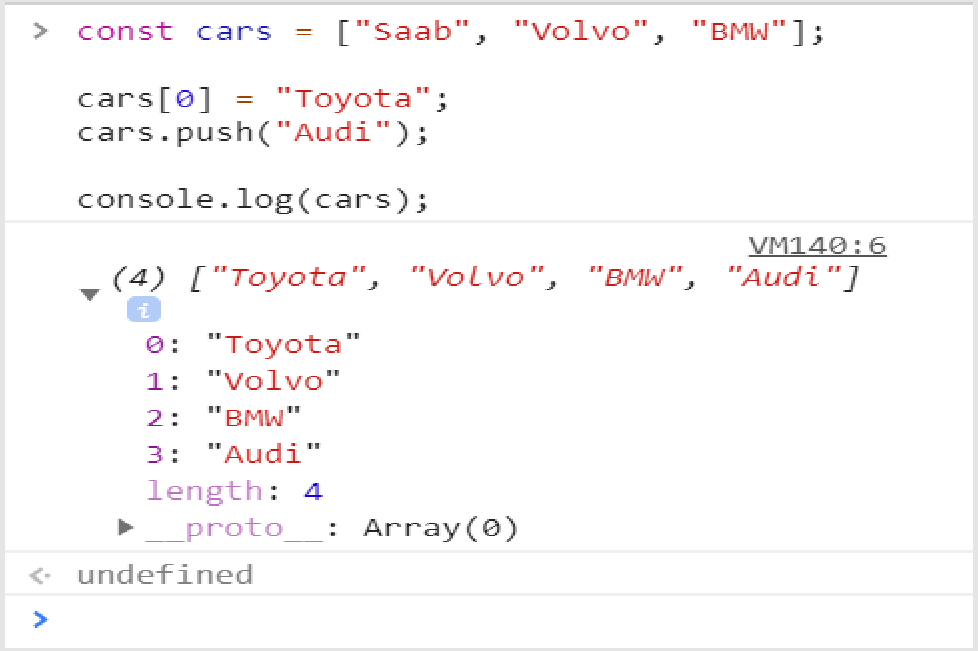
constant objects properties can be changed.



```
> const car = {type :"Fiat", model:"500", color
  :"white"};

  car = {type :"Volvo", model:"EX60", color :"red"};
ⓧ ▶Uncaught TypeError: Assignment to constant VM51:3
  variable.
      at <anonymous>:3:5
> |
```

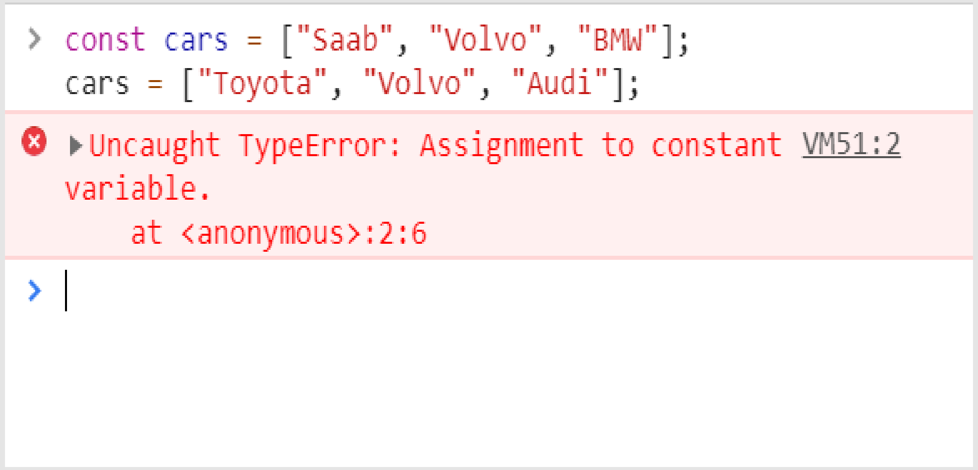
But you can NOT reassign a constant object:

Const Arrays



```
> const cars = ["Saab", "Volvo", "BMW"];

  cars[0] = "Toyota";
  cars.push("Audi");

  console.log(cars);
                                              VM140:6
  ▼ (4) ["Toyota", "Volvo", "BMW", "Audi"]
      0: "Toyota"
      1: "Volvo"
      2: "BMW"
      3: "Audi"
      length: 4
    ▶ __proto__: Array(0)
<· undefined
>
```

const Arrays can Change



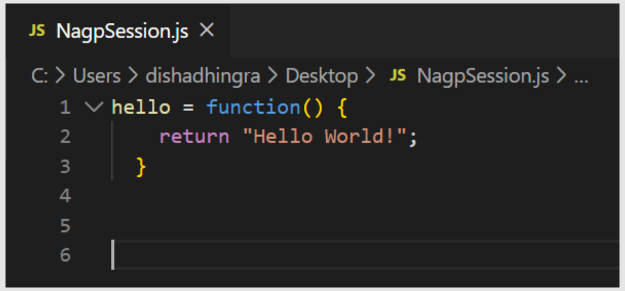
```
> const cars = ["Saab", "Volvo", "BMW"];
  cars = ["Toyota", "Volvo", "Audi"];
⊗ ▶Uncaught TypeError: Assignment to constant VM51:2
  variable.
      at <anonymous>:2:6
> |
```

But you can NOT reassign a constant array:

Arrow Functions

Arrow functions allow us to write shorter function syntax

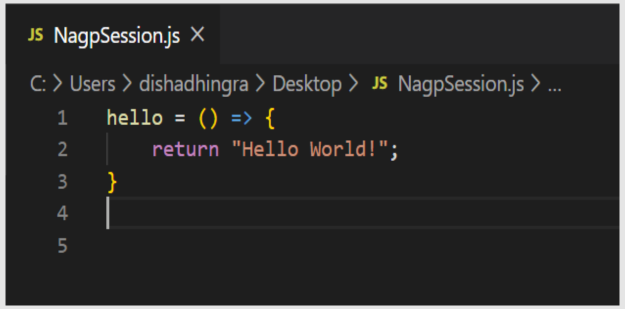
Before:


NagpSession.js ×

C: > Users > dishadhingra > Desktop > JS NagpSession.js > ...

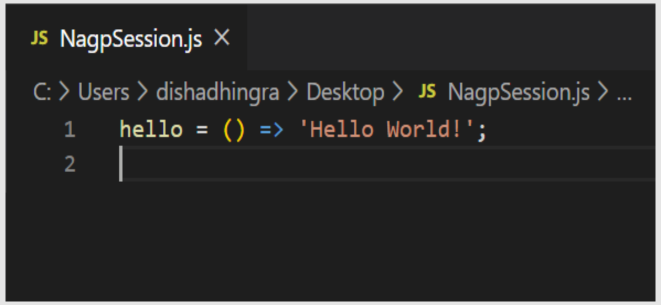
```
hello = function() {
    return "Hello World!";
}


```

With Arrow Function:


NagpSession.js ×

C: > Users > dishadhingra > Desktop > JS NagpSession.js > ...

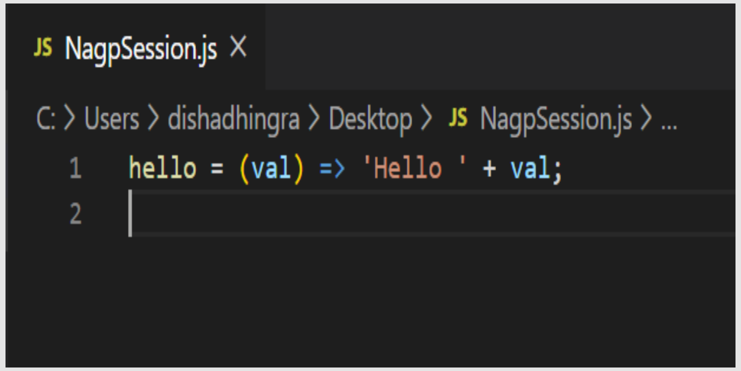
```
hello = () => {
    return "Hello World!";
}

```


NagpSession.js ×

C: > Users > dishadhingra > Desktop > JS NagpSession.js > ...

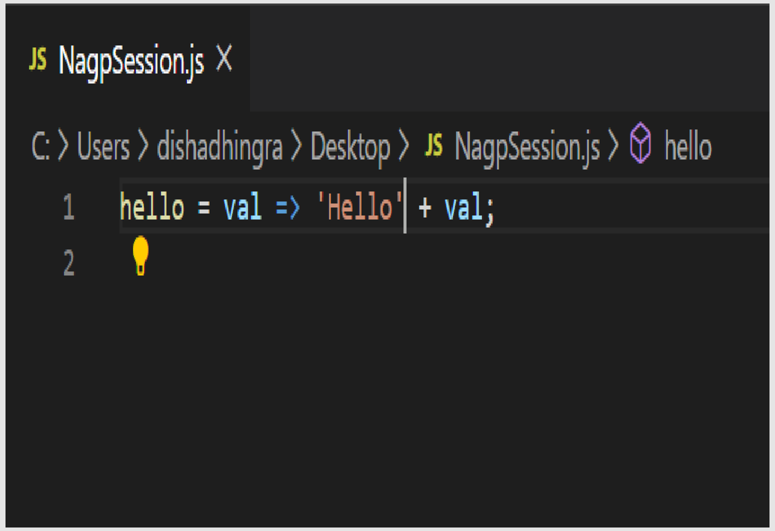
```
hello = () => 'Hello World!';
```

Arrow Function With Parameters



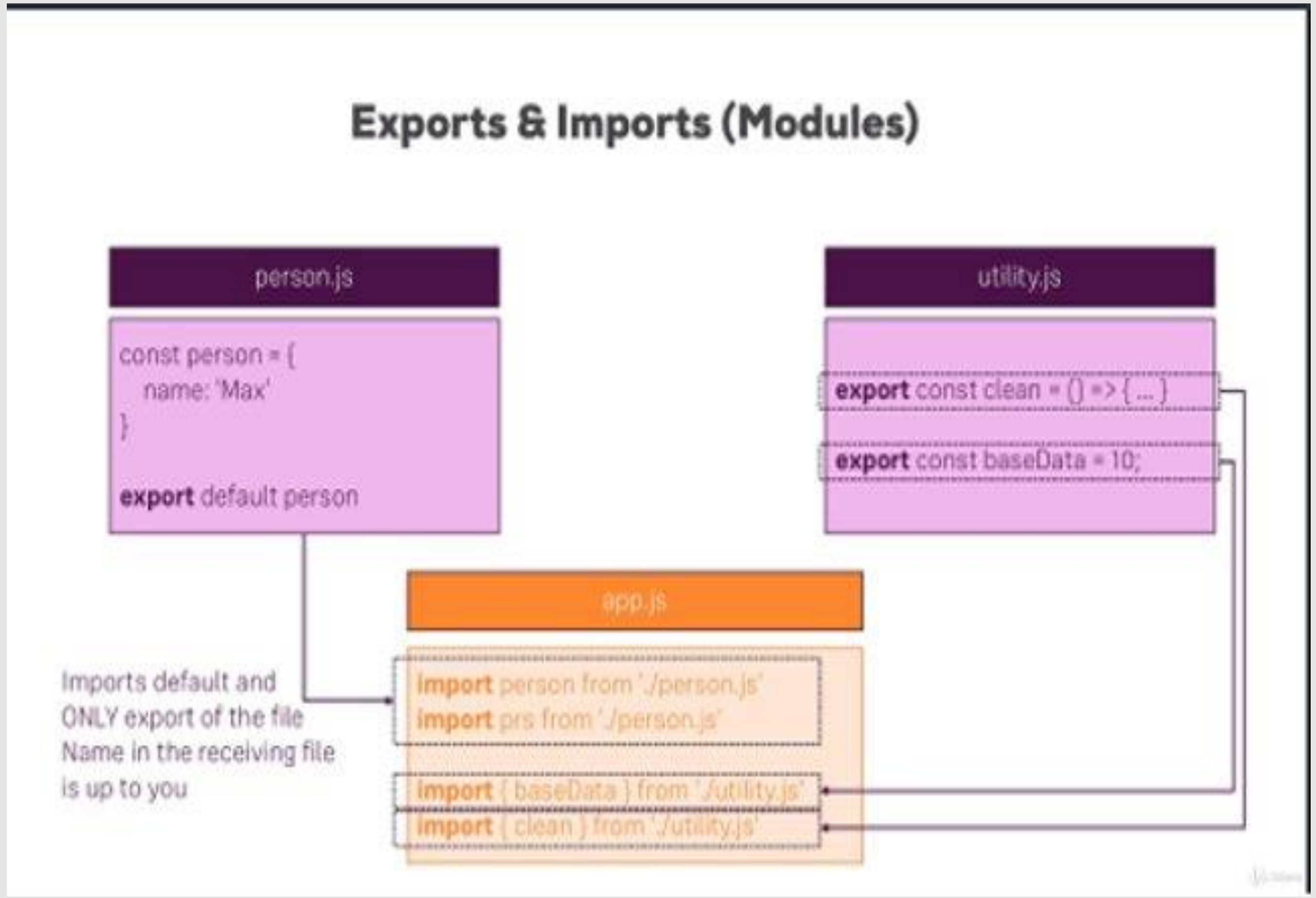
```
hello = (val) => 'Hello ' + val;
```

Arrow Function Without Parentheses:


NagpSession.js
C: > Users > dishadhingra > Desktop > NagpSession.js > hello

```
hello = val => 'Hello' + val;
```

Exports & Imports

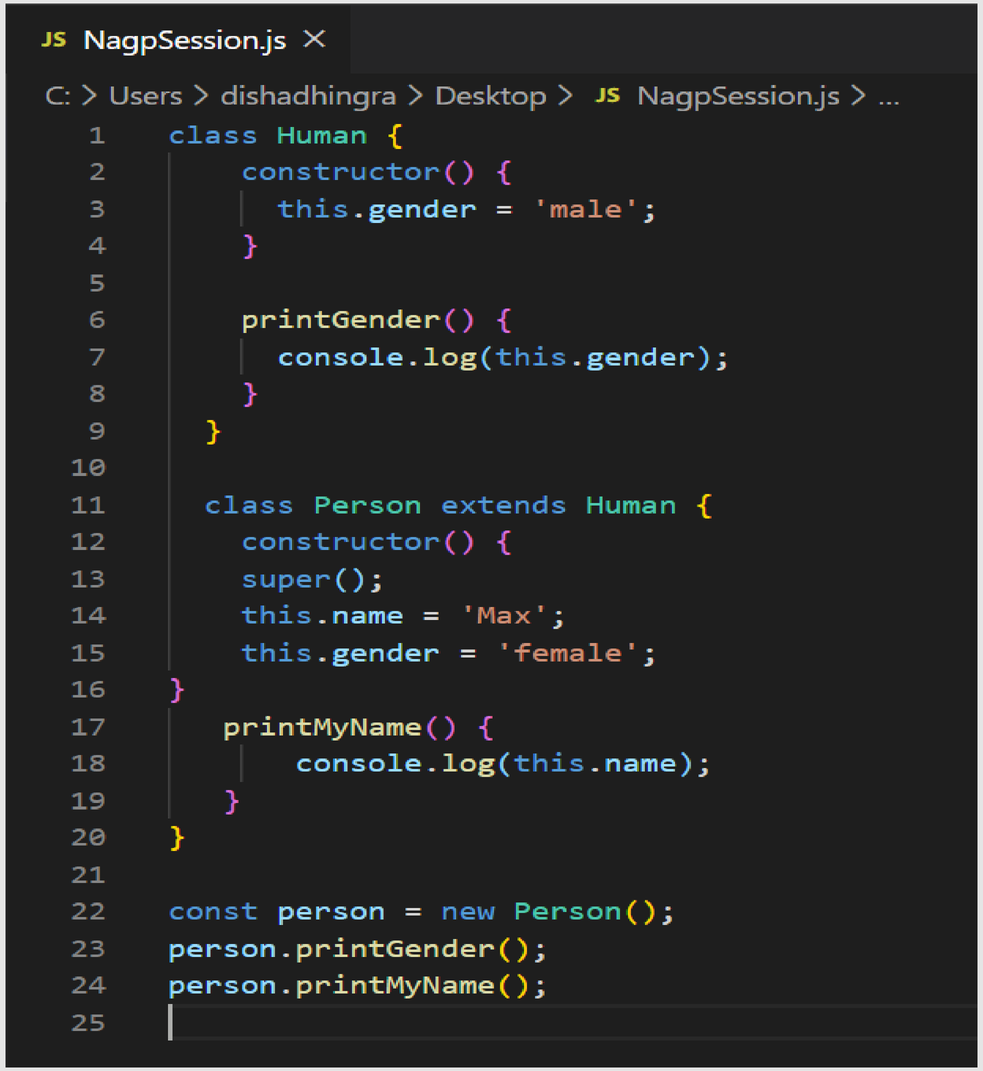

Exports & Imports (Modules)
person.js
utility.js
app.js
Imports default and ONLY export of the file Name in the receiving file is up to you

```
const person = {
   name: 'Max'
}

export default person
```

```
export const clean = () => {...}
export const baseData = 10;
```

```
import person from './person.js'
import prs from './person.js'
import { baseData } from './utility.js'
import { clean } from './utility.js'
```

Classes Example



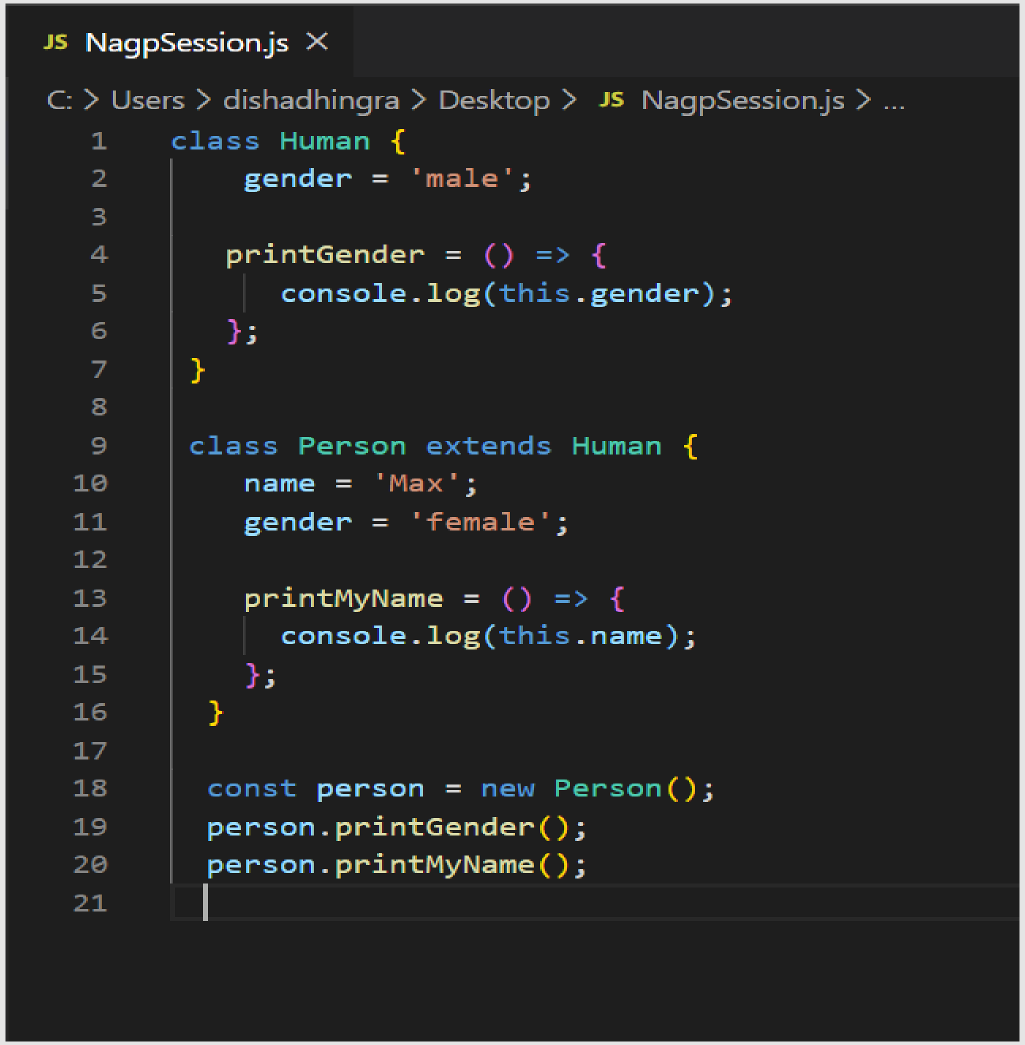
NagpSession.js ×

C: > Users > dishadhingra > Desktop > NagpSession.js > ...

```
class Human {
    constructor() {
      this.gender = 'male';
    }

    printGender() {
      console.log(this.gender);
    }
  }

  class Person extends Human {
    constructor() {
    super();
    this.name = 'Max';
    this.gender = 'female';
}
   printMyName() {
     console.log(this.name);
   }
}

const person = new Person();
person.printGender();
person.printMyName();

```

Output
‘female’
‘Max'

Classes ES7 Example


NagpSession.js ×

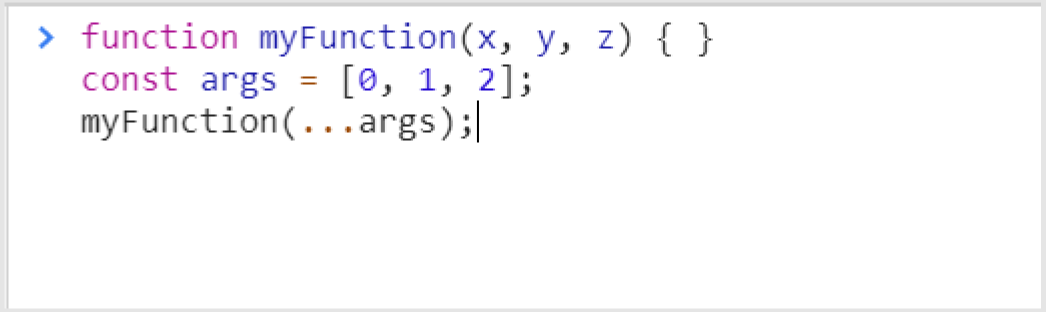
C: > Users > dishadhingra > Desktop > NagpSession.js > ...

```
class Human {
    gender = 'male';

   printGender = () => {
      console.log(this.gender);
   };
 }

 class Person extends Human {
    name = 'Max';
    gender = 'female';

    printMyName = () => {
      console.log(this.name);
    };
  }

  const person = new Person();
  person.printGender();
  person.printMyName();
```

Output
‘female’
‘Max'

Spread Operator (...)

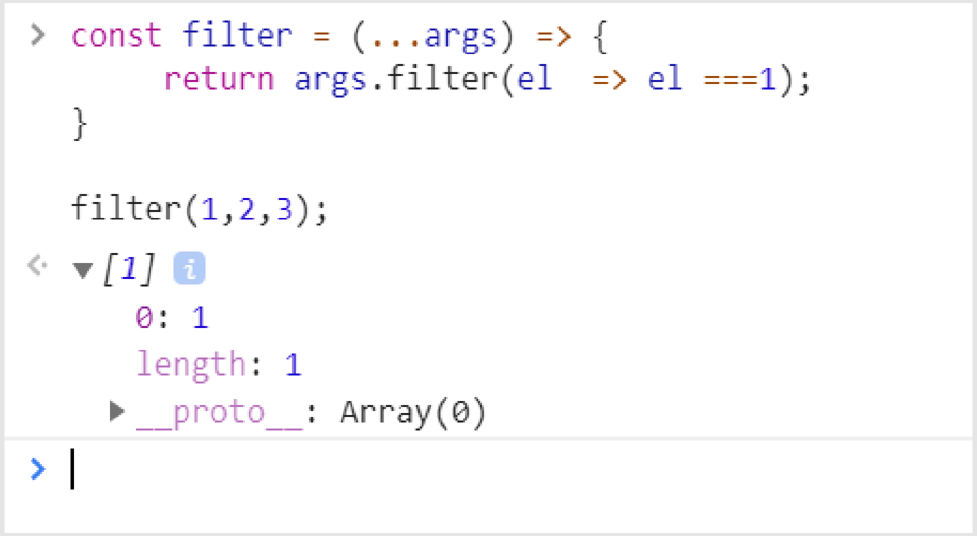
Spread syntax allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.



```
> function myFunction(x, y, z) { }
  const args = [0, 1, 2];
  myFunction(...args);
```

Rest Operator (...)

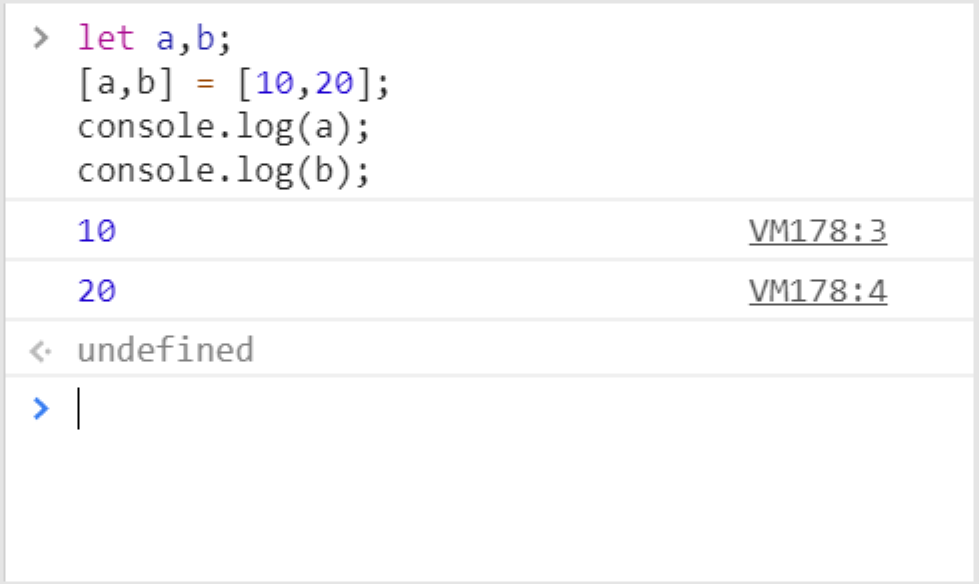
The **rest parameter** syntax allows us to represent an indefinite number of arguments as an array.



```
> const filter = (...args) => {
      return args.filter(el  => el ===1);
  }

  filter(1,2,3);
<· ▼[1] i
      0: 1
      length: 1
     ▶__proto__: Array(0)
> |
```

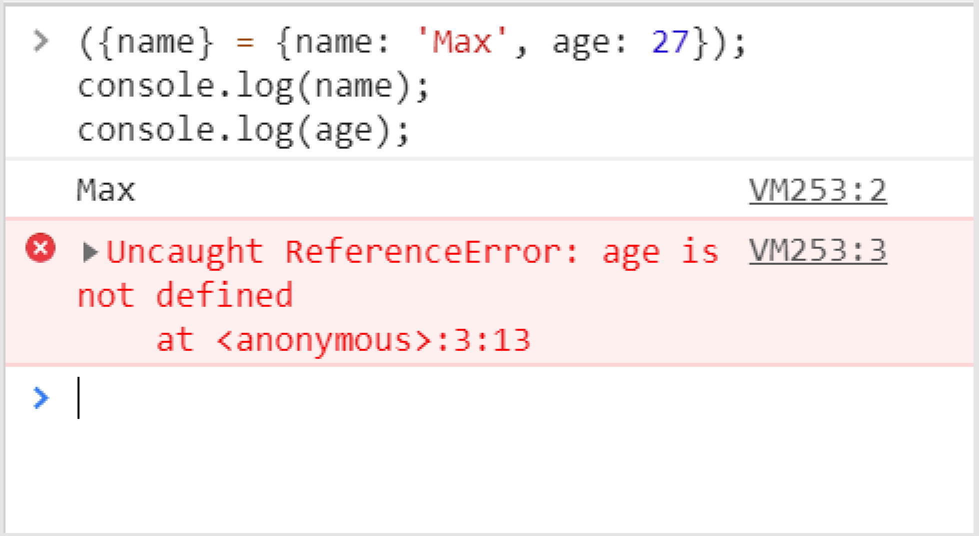
Destructuring

Easily extract array elements or properties objects and store them in variables.



```
> let a,b;
  [a,b] = [10,20];
  console.log(a);
  console.log(b);
  10                                        VM178:3
  20                                        VM178:4
<· undefined
>
```

Array Destructuring



```
> ({name} = {name: 'Max', age: 27});
  console.log(name);
  console.log(age);
  Max                                       VM253:2
✖ ▶Uncaught ReferenceError: age is  VM253:3
  not defined
      at <anonymous>:3:13
>
```

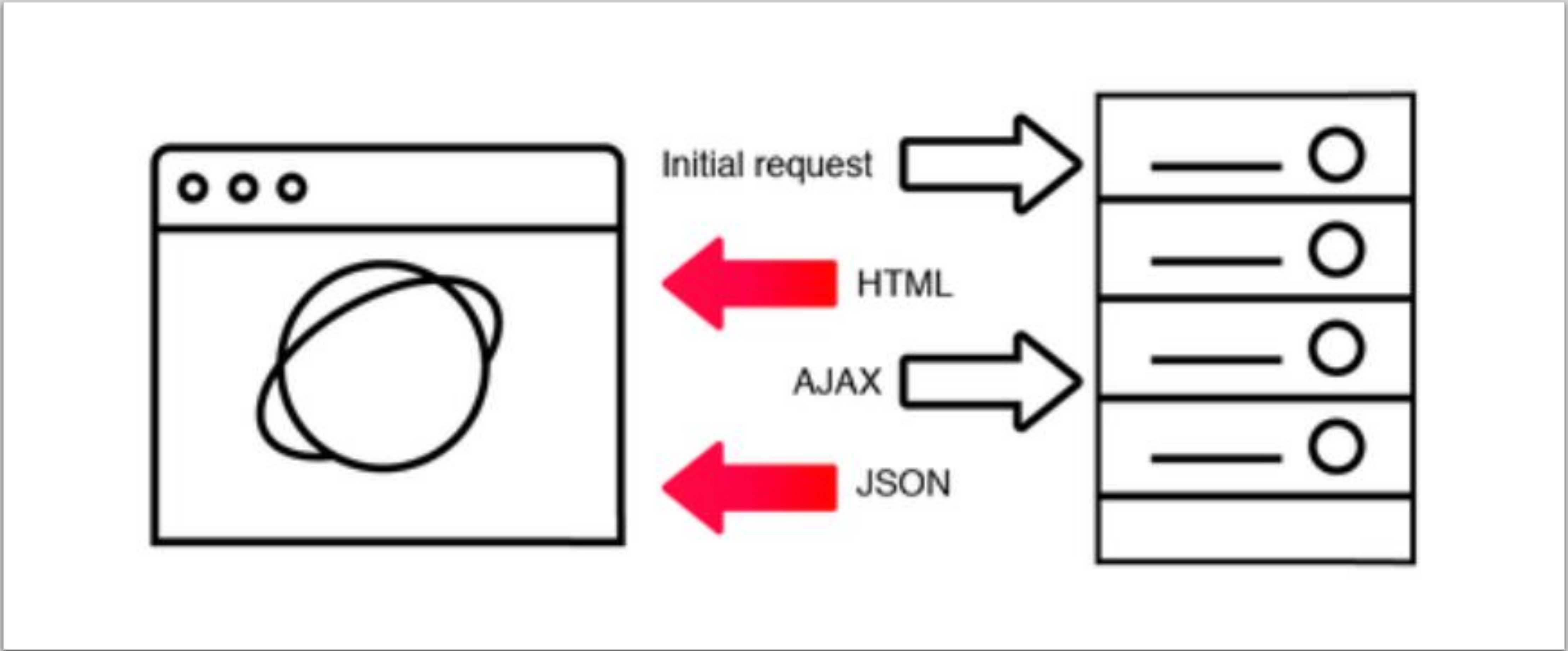
Object Destructuring

Single Page Applications Vs Multi Page Applications

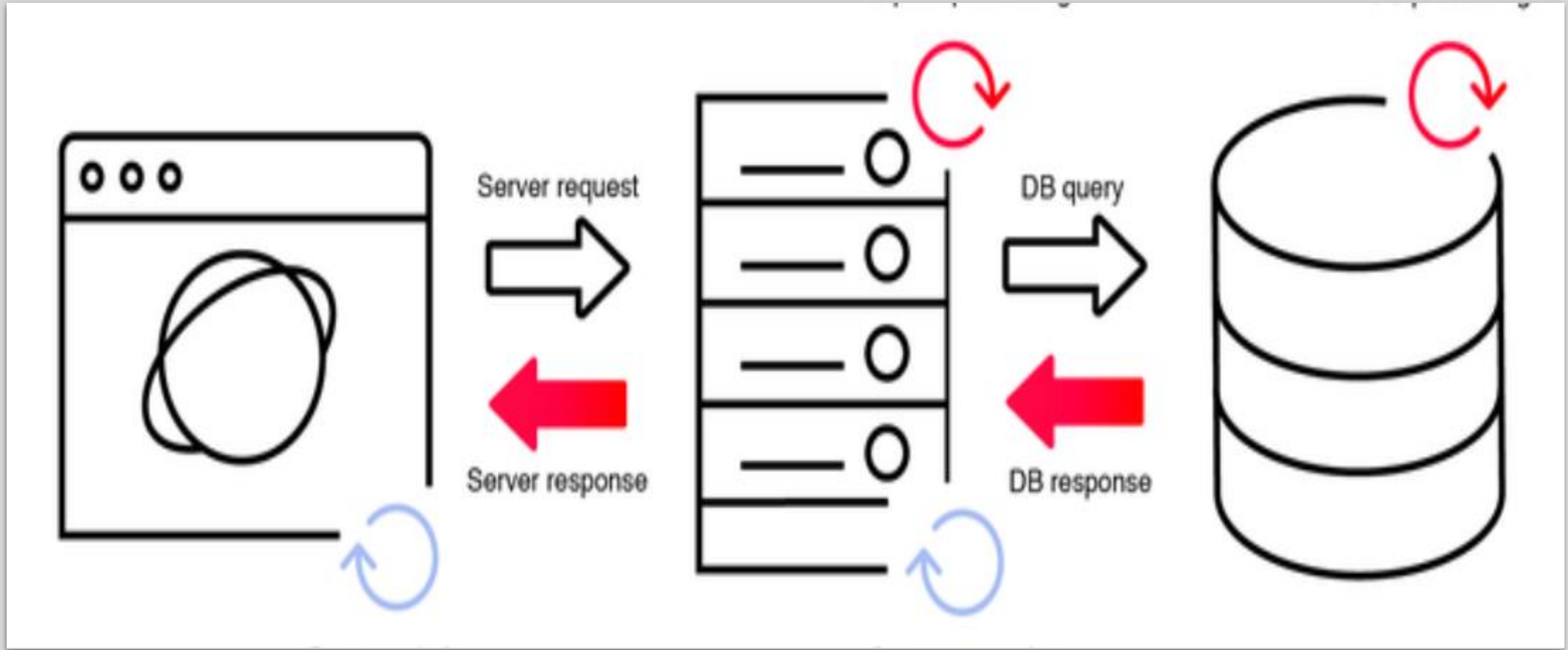
Only one HTML Page, content is (re)-rendered on Client.

Multiple HTML Pages. Content is rendered on server.

Single Page Applications



Multi -Page Applications

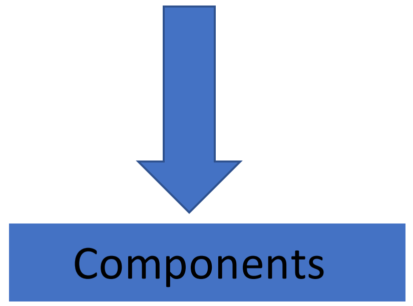


Examples of Single Page Applications

- Gmail
- Google Maps
- Facebook
- Twitter
- Google Drive

What is React?

- Developed by Facebook
- A JavaScript library for building user interfaces.



- Renders your UI and respond to events.

NAGP
Complex is simple

Who uses React?

NAGP
Complex is simple

facebook

TERADEK

Instagram

NETFLIX

KHANACADEMY

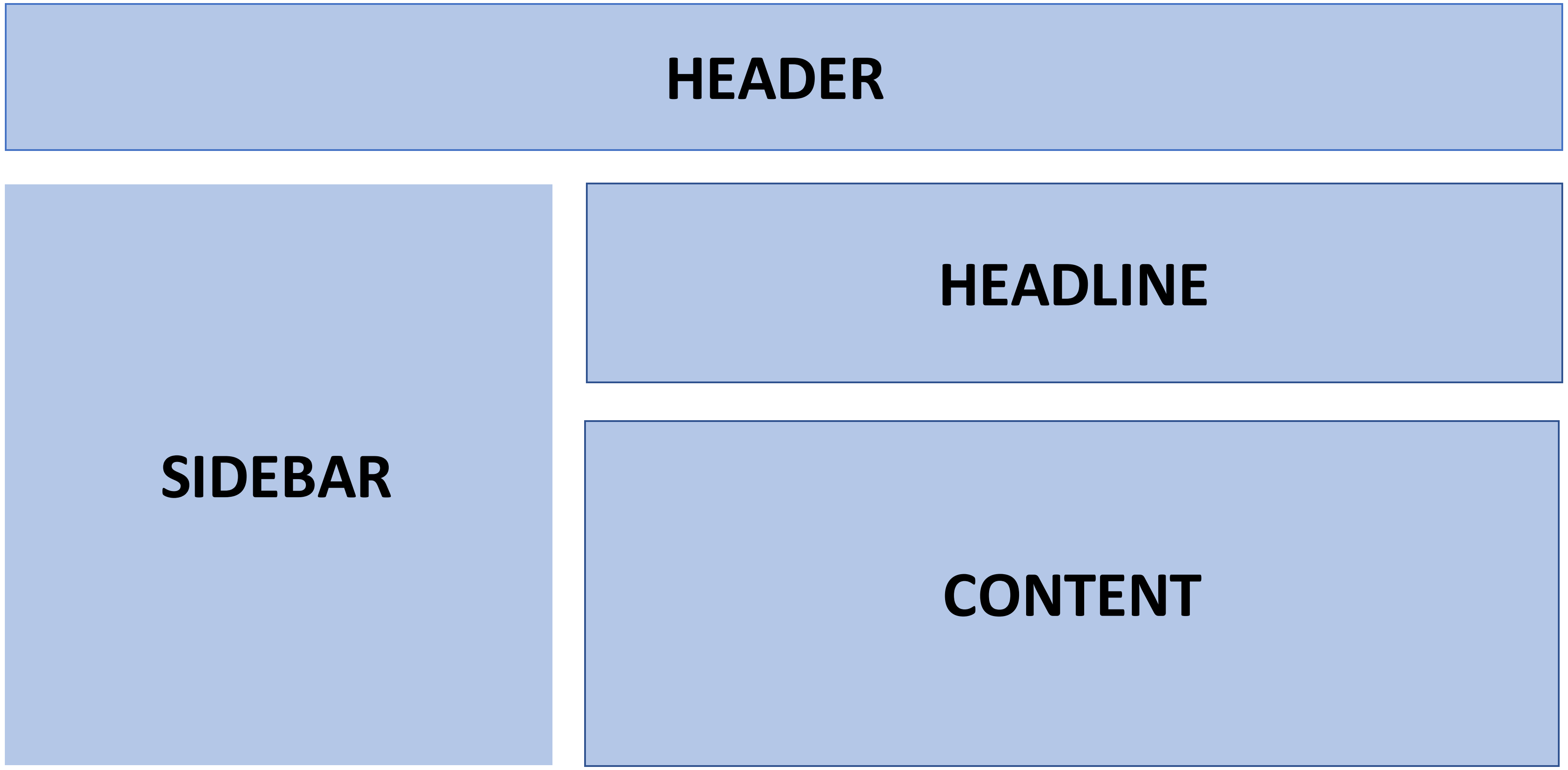
reddit

What are React Components?

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and returns HTML via a render function.

NAGP
Complex is simple

Components



NAGP
Complex is simple

Advantages of React

- UI state becomes difficult to handle with vanilla JavaScript.
- Focus on Business logic on preventing your app from exploding.

NAGP
Complex is simple

Set-Up React Application

- **Prerequisites**

- Node.js should be installed. Node version >=8.10 and npm >=5.6

- Make sure you have set the environment variables for nodejs and npm package manager

- **Create Application**

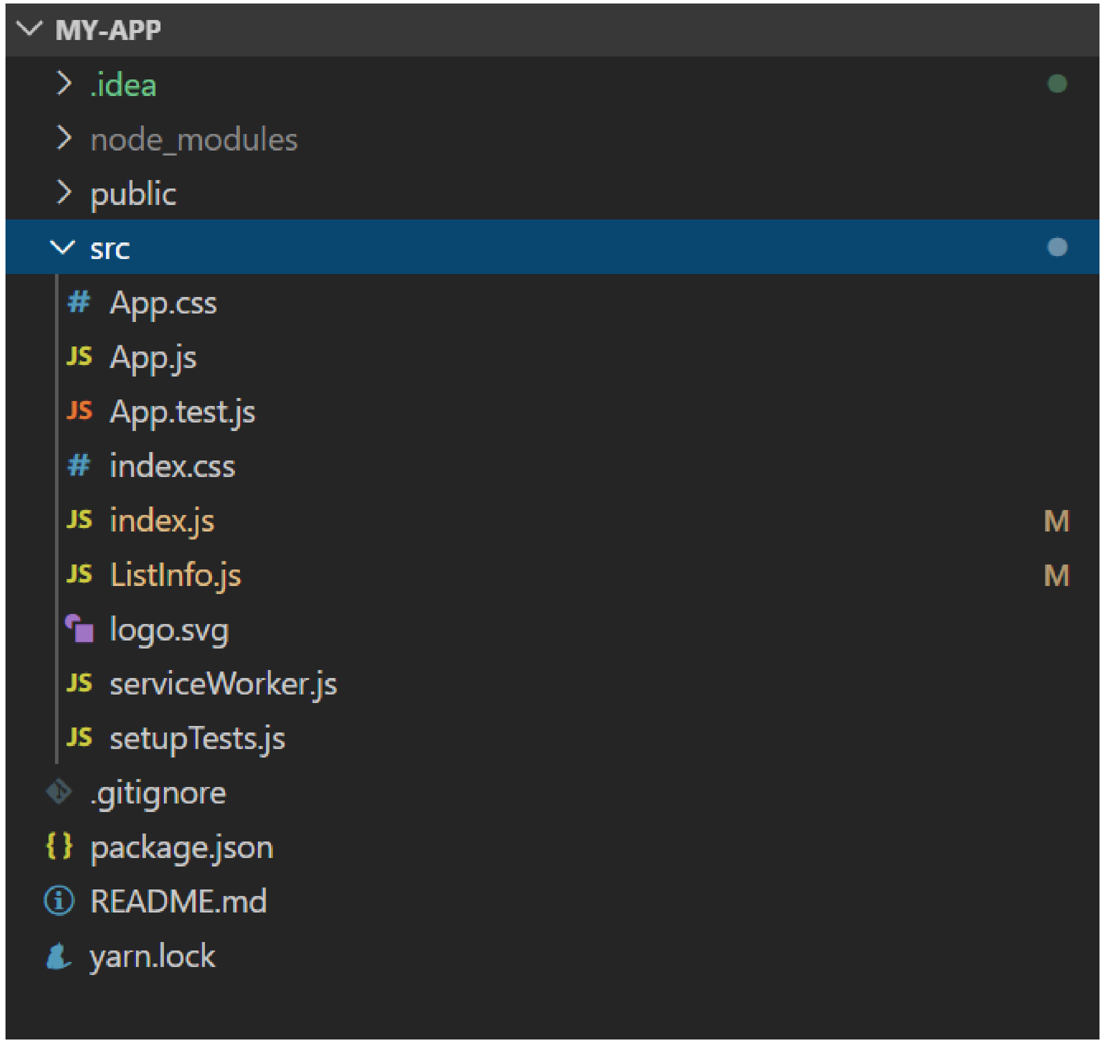
- npx create-react-app my-app
- https://github.com/reactjs/reactjs.org

- **Running app locally**

- cd my-app
- npm start

NAGP
Complex is simple

Code Structure



```
MY-APP
  > .idea
  > node_modules
  > public
  v src
      App.css
      App.js
      App.test.js
      index.css
      index.js            M
      ListInfo.js         M
      logo.svg
      serviceWorker.js
      setupTests.js
    .gitignore
    package.json
    README.md
    yarn.lock
```

JSX

- JSX is short for JavaScript XML.
- It is a syntax extension to JavaScript.
- JSX is an expression which uses valid HTML statements within JavaScript.
- JavaScript expressions and JSX within these HTML statements by placing them within braces ({}). Babel further compiles JSX into an object of type `React.createElement()`.
- We can use it with react to describe what the UI should look like.

Single-line & Multi-line expressions

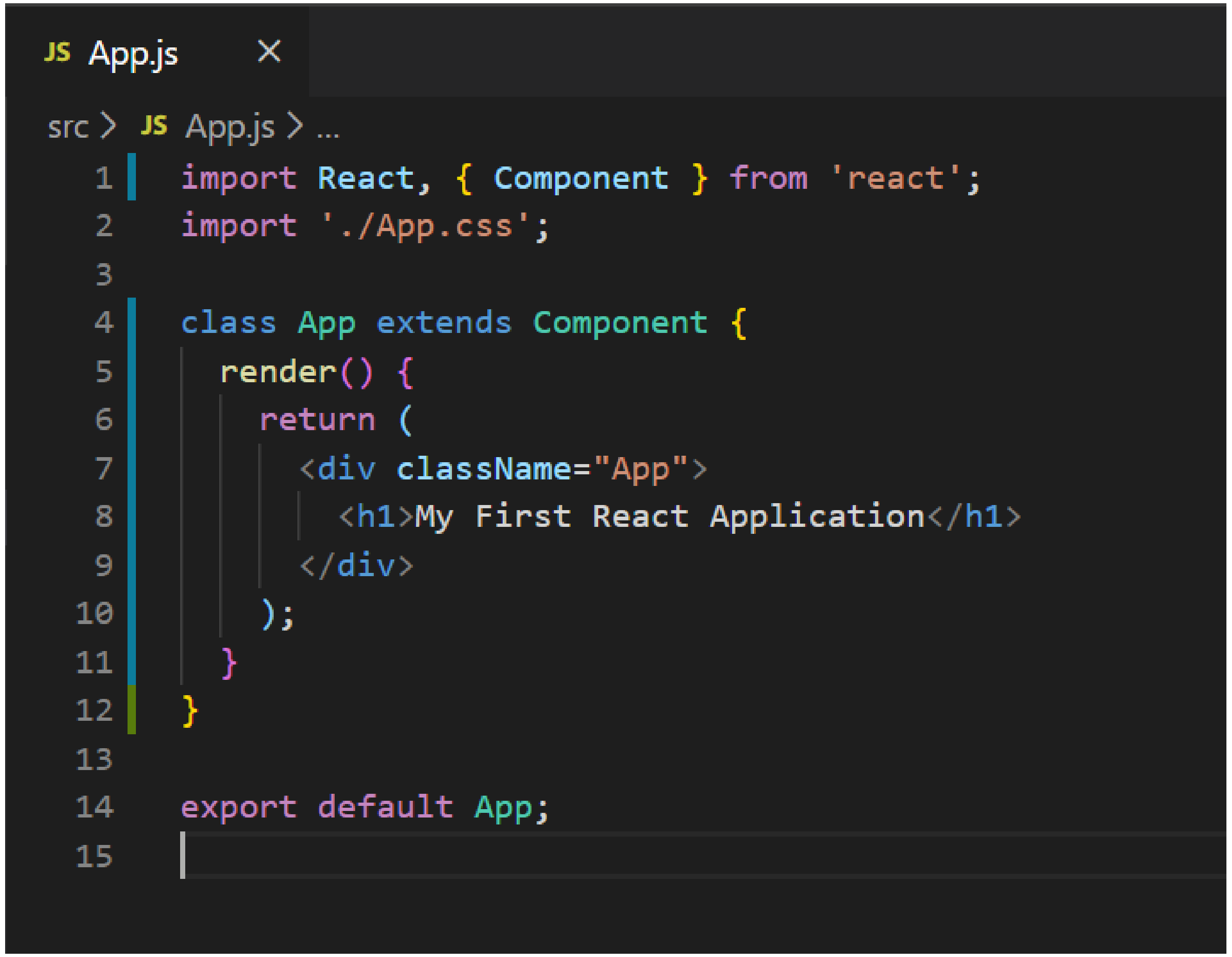
Single-line expression are simple to use.

```
const one = <h1>Hello World!</h1>;
```

When you need to use multiple lines in a single JSX expression, write the code within a single parenthesis.

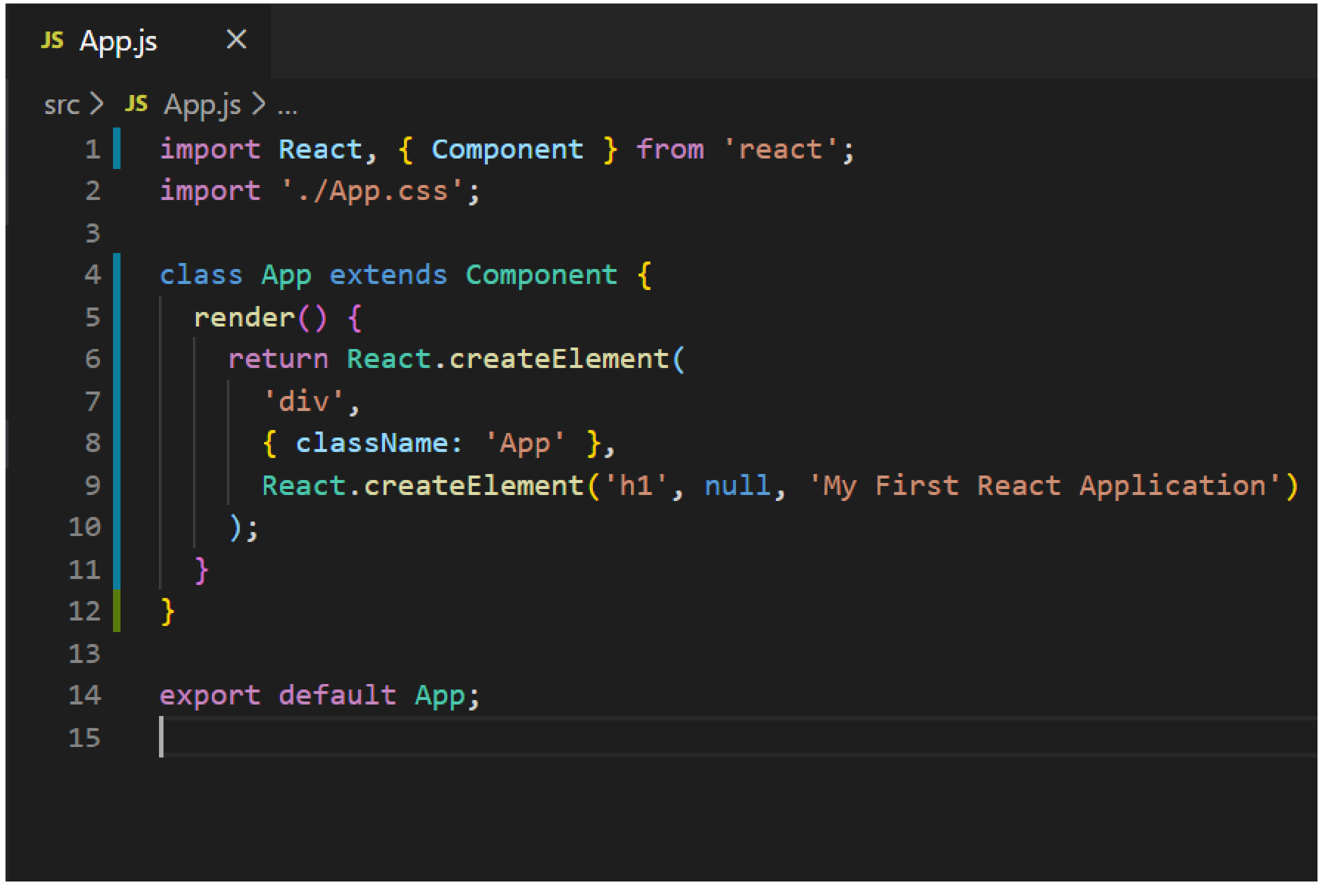
```
const two = (
  <ul>
    <li>Once</li>
    <li>Twice</li>
  </ul>
);
```

JSX
Example



```
import React, { Component } from 'react';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>My First React Application</h1>
      </div>
    );
  }
}

export default App;

```

Understanding JSX


App.js

src > App.js > ...

```
import React, { Component } from 'react';
import './App.css';

class App extends Component {
  render() {
    return React.createElement(
      'div',
      { className: 'App' },
      React.createElement('h1', null, 'My First React Application')
    );
  }
}

export default App;
```

JSX Restrictions

A JSX expression must have only one parent tag. We can add multiple tags nested within the parent element only.

- **// This is valid.**
```
const tags = (
  <ul>
    <li>Once</li>
    <li>Twice</li>
  </ul>
);
```

- **// This is not valid.**
```
const tags = (
  <h1>Hello World!</h1>
  <h3>This is my special list:</h3>
  <ul>
    <li>Once</li>
    <li>Twice</li>
  </ul>
);
```

Outputting Dynamic Content

Dynamic content within our JSX.

{Dynamic Content}

What are Props?

Props are managed from outside the component. It allows you to pass the data from the parent(wrapping) component to the child component.

Changes in props trigger React to re-render your components and potentially update the DOM in the browser.

Understanding the “children” props

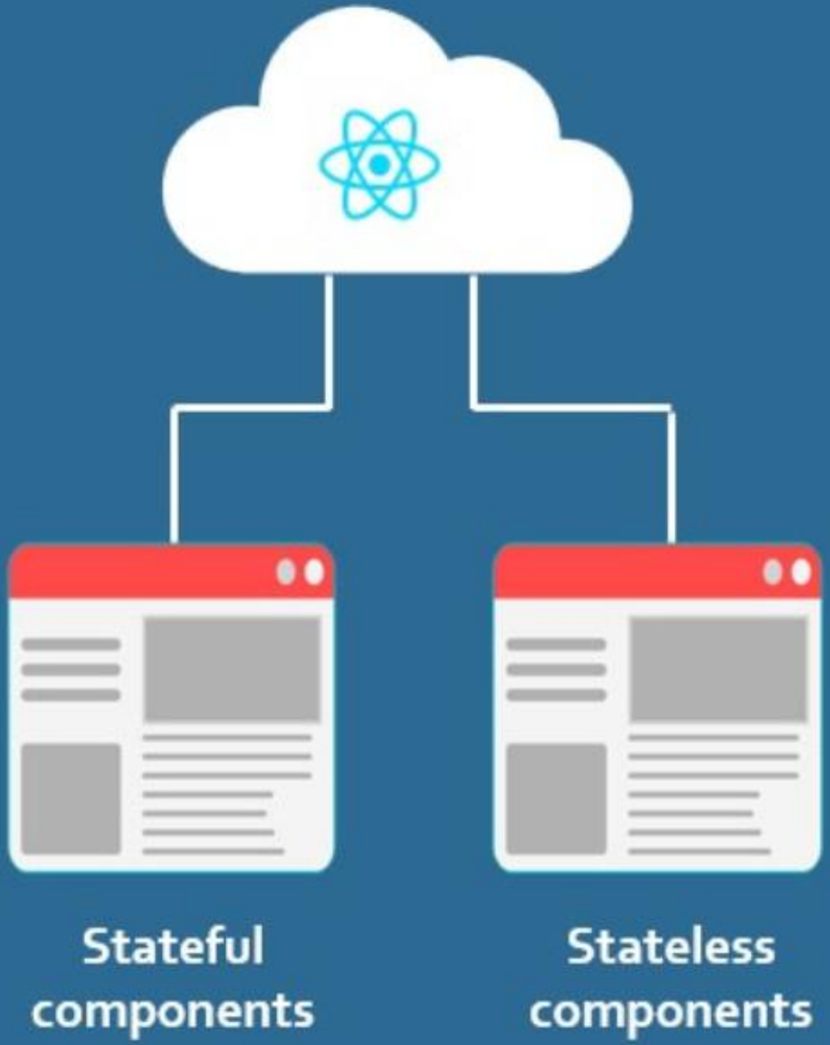
States

States are managed inside the component. It is used to change the component, well the state from within. Changes to state also trigger an UI state.

Manipulation of states

Passing Method References between Components.

React Components



Lifecycle Components

Available only in class-based components

constructor()

getDerivedStateFromProps()

getSnapshotBeforeUpdate()

componentDidCatch()

componentWillUnmount()

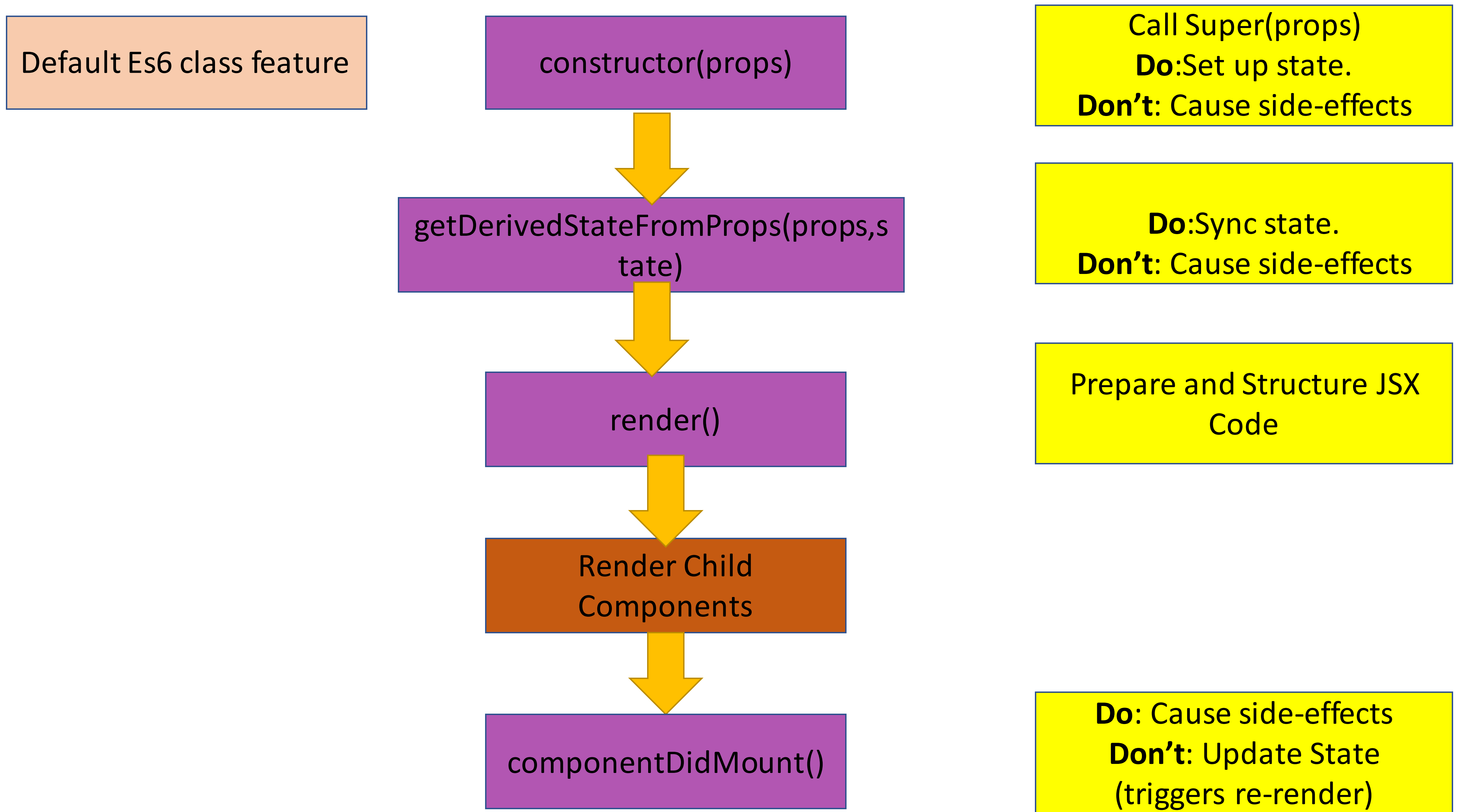
shouldComponentUpdate()

componentDidUpdate()

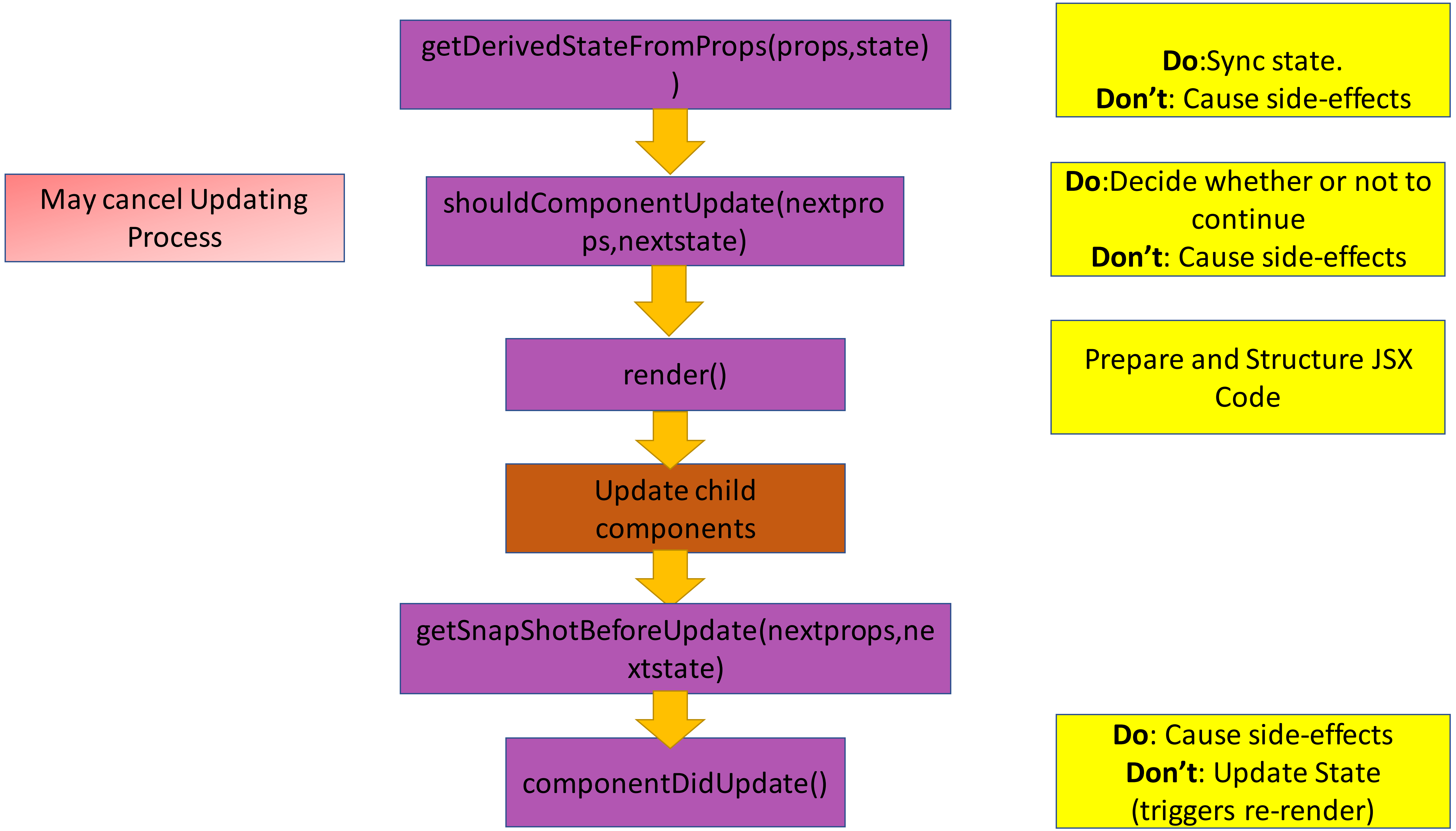
componentDidMount()

render()

Lifecycle Components-Creation


Default Es6 class feature
constructor(props)
Call Super(props)
Do:Set up state.
Don't: Cause side-effects
getDerivedStateFromProps(props,state)
Do:Sync state.
Don't: Cause side-effects
render()
Prepare and Structure JSX Code
Render Child Components
componentDidMount()
Do: Cause side-effects
Don't: Update State (triggers re-render)

Lifecycle Components-Update


getDerivedStateFromProps(props,state)
shouldComponentUpdate(nextprops,nextstate)
render()
Update child components
getSnapShotBeforeUpdate(nextprops,nextstate)
componentDidUpdate()
May cancel Updating Process

Do:Sync state.
Don’t: Cause side-effects

Do:Decide whether or not to continue
Don’t: Cause side-effects

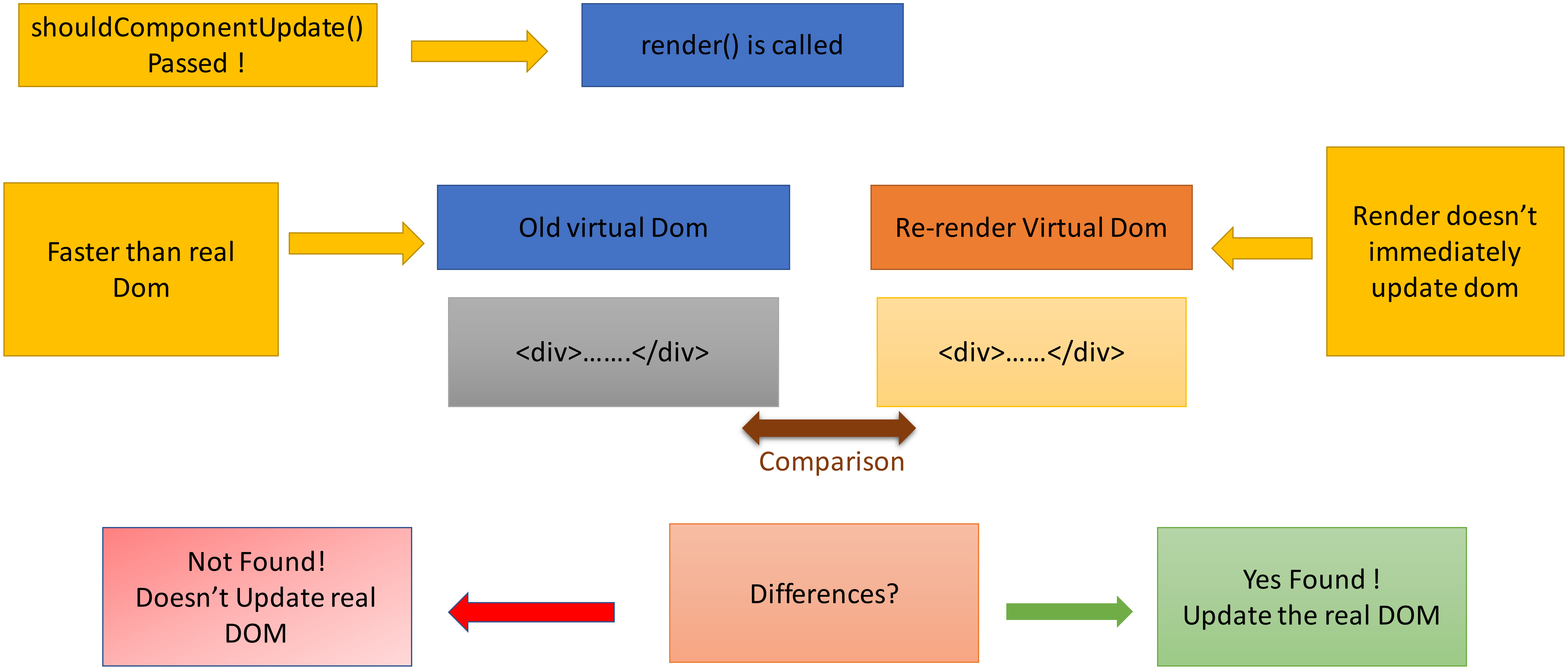
Prepare and Structure JSX Code

Do: Cause side-effects
Don’t: Update State (triggers re-render)

Lifecycle Components-Clean-up

componentWillUnmount()

How DOM Updates In React?


shouldComponentUpdate() Passed !
render() is called
Faster than real Dom
Old virtual Dom
Re-render Virtual Dom
Render doesn’t immediately update dom
<div>…….</div>
<div>……</div>
Comparison
Not Found! Doesn’t Update real DOM
Differences?
Yes Found ! Update the real DOM

End
Demo

nagarro

THANK YOU!!