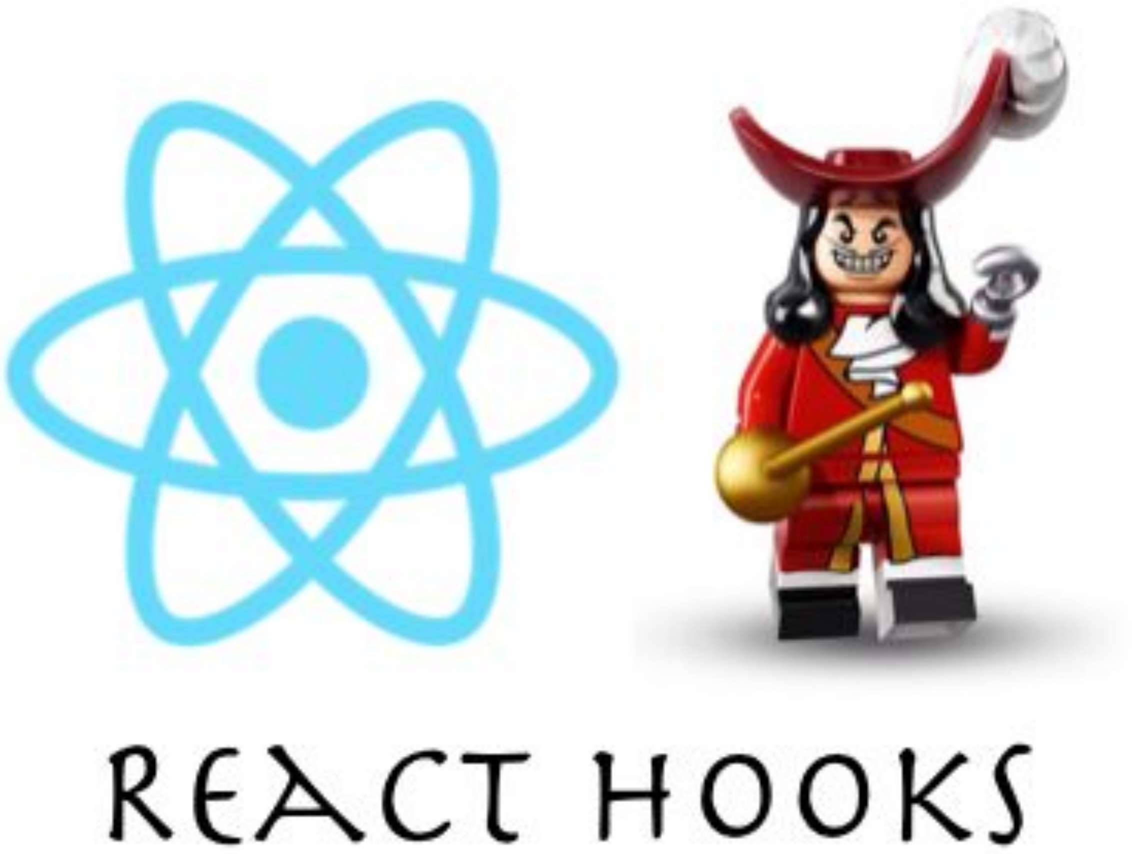
React Hooks

2019-06-16

Content



Introducing Hooks

- ➔ Hooks allow you to reuse stateful logic without changing your component hierarchy.
- ➔ Hooks let you split one component into smaller functions based on what pieces are related (such as setting up a subscription or fetching data).
- ➔ Hooks let you use more of React’s features without classes.

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count +
1)}>
        Click me
      </button>
    </div>
  );
}
```

Most used Hooks

State Hook

```
import React, { useState } from 'react';
function Example() {
  // Declare a new state variable
  const [count, setCount] =  useState(0);
  // ...
```

It's similar to `this.setState` in a class, except it doesn't merge the old and new state together.

Effect Hook

```
useEffect(() => {
    // Update the document title
    document.title = `${count} times`;
  });
// ...
```

It serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in React classes, but unified into a single API.

Using State Hook

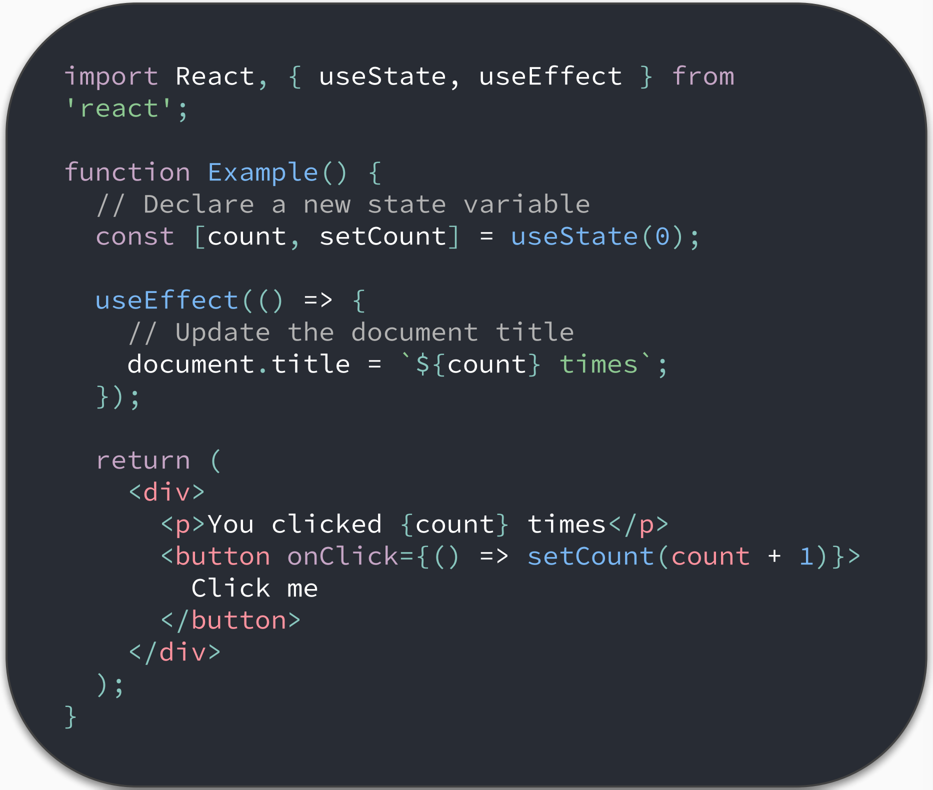
```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() =>
setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Tips for State Hook

```
const [fruit, setFruit] = useState('banana');
```

```
const fruitStateVariable = useState('banana');
const fruit = fruitStateVariable[0];
const setFruit = fruitStateVariable[1];
```

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```



```
import React, { useState, useEffect } from
'react';

function Example() {
  // Declare a new state variable
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Update the document title
    document.title = `${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Using Effect Hook

Tips for Effect Hook

```
const [count, setCount] = useState(0);
useEffect(() => {
  document.title = `You clicked ${count} times`;
});

const [isOnline, setIsOnline] = useState(null);
useEffect(() => {
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }

  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

Tips for Effect Hook

Optimization by skipping Effects

```
useEffect(() => {
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }

  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
}, [props.friend.id]); // Only re-subscribe if props.friend.id changes
```

Other Hooks

useContext lets you subscribe to React context without introducing nesting:

```
function Example() {
  const locale = useContext(LocaleContext);
  const theme = useContext(ThemeContext);
  // ...
}
```

useReducer lets you manage local state of complex components with a reducer:

```
function Todos() {
  const [tds, dispatch] = useReducer(tdsReducer);
  // ...
```

useCallback, useMemo, useRef, useImperativeHandle, useLayoutEffect, useDebugValue

Rules of Hooks

Only Call Hooks at the Top Level

Don’t call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function. By following this rule, you ensure that Hooks are called in the same order each time a component renders.

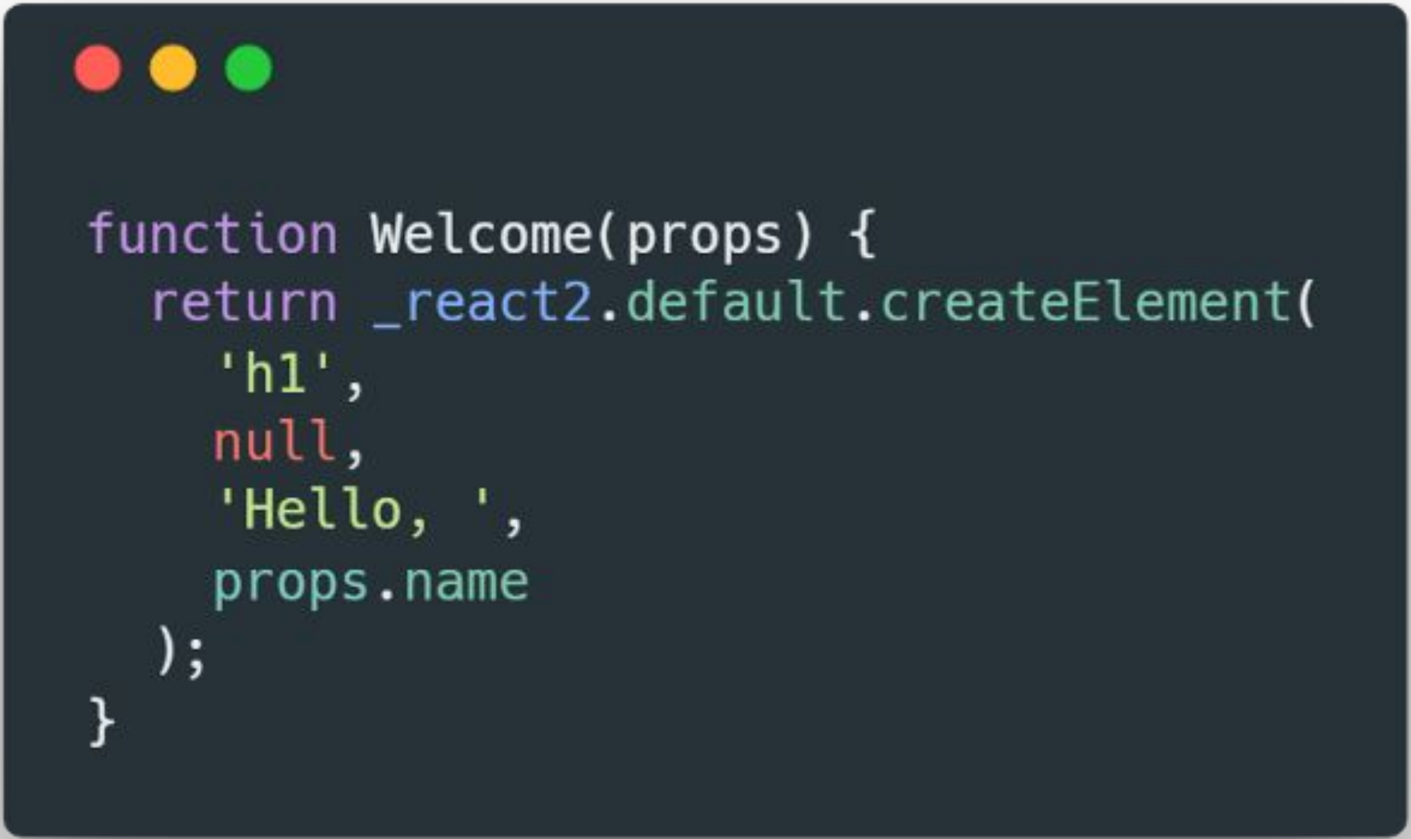
Only Call Hooks from React Functions

Don’t call Hooks from regular JavaScript functions. Instead:

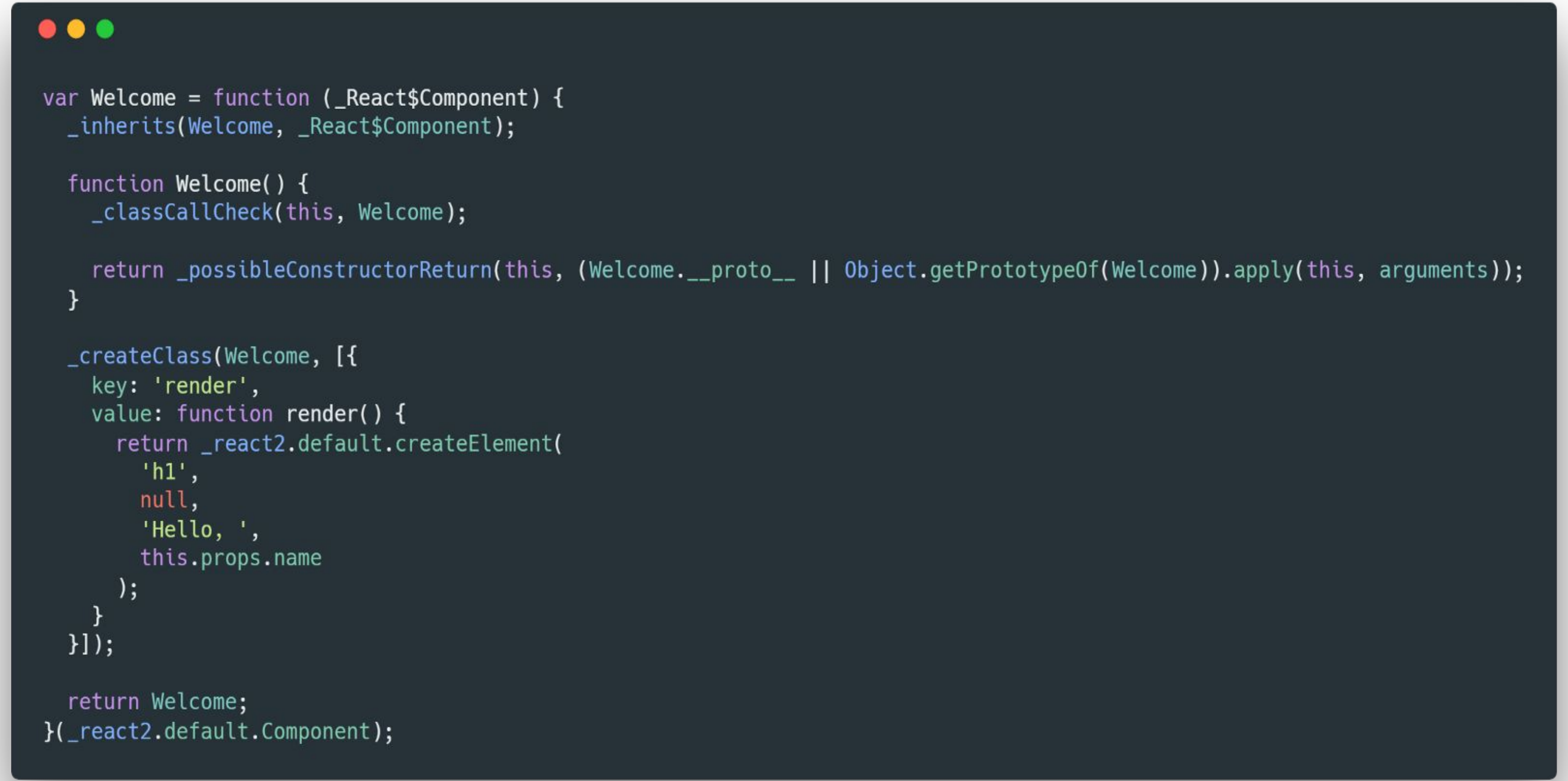
✓ Call Hooks from React function components.

✓ Call Hooks from custom Hook

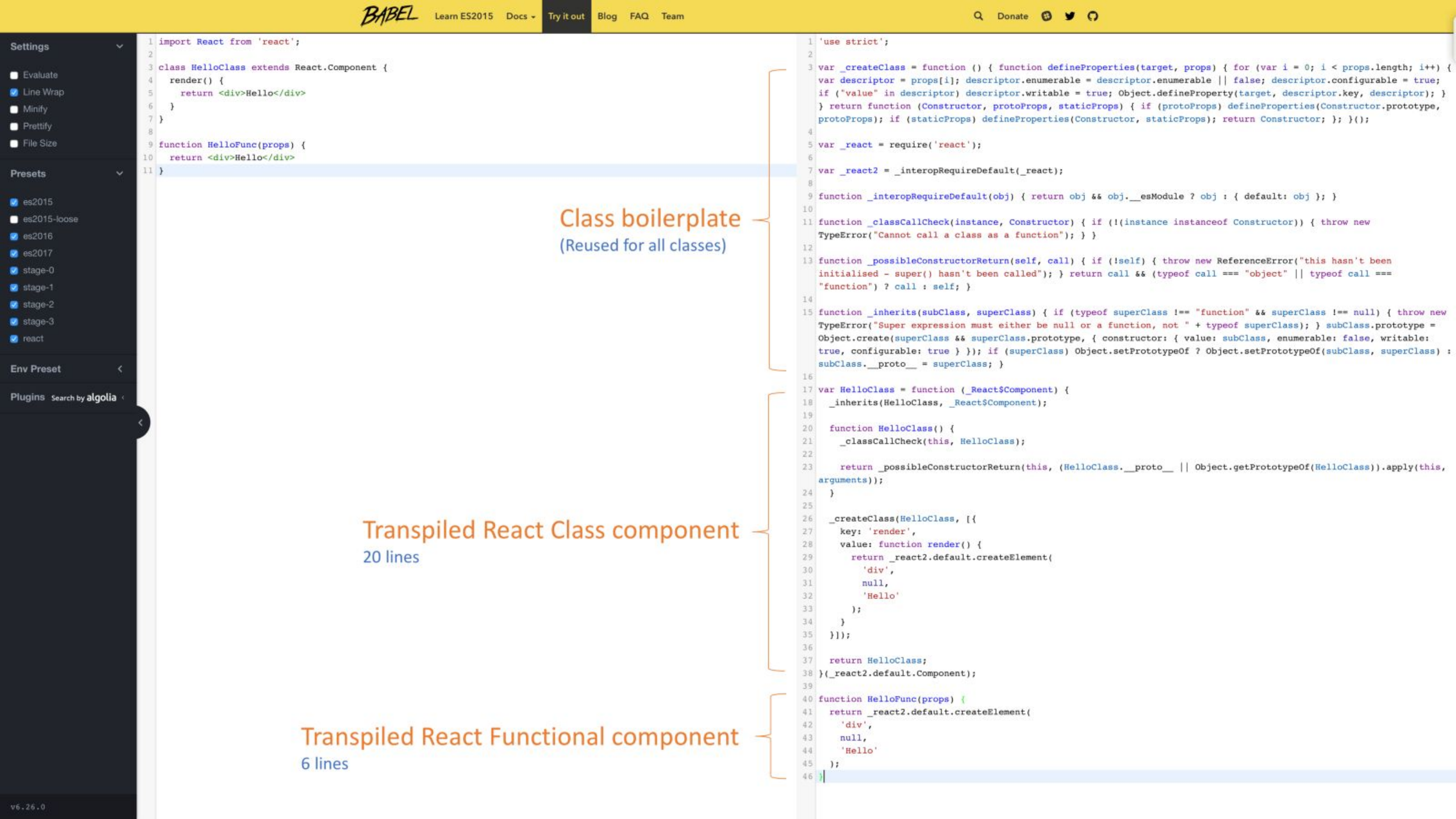
Compare between Functional Components and Class Components



```
function Welcome(props) {
  return _react2.default.createElement(
    'h1',
    null,
    'Hello, ',
    props.name
  );
}
```



```
var Welcome = function (_React$Component) {
  _inherits(Welcome, _React$Component);

  function Welcome() {
    _classCallCheck(this, Welcome);

    return _possibleConstructorReturn(this, (Welcome.__proto__ || Object.getPrototypeOf(Welcome)).apply(this, arguments));
  }

  _createClass(Welcome, [{
    key: 'render',
    value: function render() {
      return _react2.default.createElement(
        'h1',
        null,
        'Hello, ',
        this.props.name
      );
    }
  }]);

  return Welcome;
}(_react2.default.Component);
```



Settings

- [ ] Evaluate
- [✓] Line Wrap
- [ ] Minify
- [ ] Prettify
- [ ] File Size

Presets

- [✓] es2015
- [ ] es2015-loose
- [✓] es2016
- [✓] es2017
- [✓] stage-0
- [✓] stage-1
- [✓] stage-2
- [✓] stage-3
- [✓] react

v6.26.0

```
import React from 'react';

class HelloClass extends React.Component {
  render() {
    return <div>Hello</div>
  }
}

function HelloFunc(props) {
  return <div>Hello</div>
}
```

```
'use strict';

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value" in descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(Constructor, staticProps); return Constructor; }; }();

var _react = require('react');

var _react2 = _interopRequireDefault(_react);

function _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { default: obj }; }

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

function _possibleConstructorReturn(self, call) { if (!self) { throw new ReferenceError("this hasn't been initialised - super() hasn't been called"); } return call && (typeof call === "object" || typeof call === "function") ? call : self; }

function _inherits(subClass, superClass) { if (typeof superClass !== "function" && superClass !== null) { throw new TypeError("Super expression must either be null or a function, not " + typeof superClass); } subClass.prototype = Object.create(superClass && superClass.prototype, { constructor: { value: subClass, enumerable: false, writable: true, configurable: true } }); if (superClass) Object.setPrototypeOf ? Object.setPrototypeOf(subClass, superClass) : subClass.__proto__ = superClass; }

var HelloClass = function (_React$Component) {
  _inherits(HelloClass, _React$Component);

  function HelloClass() {
    _classCallCheck(this, HelloClass);

    return _possibleConstructorReturn(this, (HelloClass.__proto__ || Object.getPrototypeOf(HelloClass)).apply(this, arguments));
  }

  _createClass(HelloClass, [{
    key: 'render',
    value: function render() {
      return _react2.default.createElement(
        'div',
        null,
        'Hello'
      );
    }
  }]);

  return HelloClass;
}(_react2.default.Component);

function HelloFunc(props) {
  return _react2.default.createElement(
    'div',
    null,
    'Hello'
  );
}
```

Class boilerplate
(Reused for all classes)

Transpiled React Class component
20 lines

Transpiled React Functional component
6 lines

Thank You