

WMNFE 2410

FRONT-END DEVELOPMENT

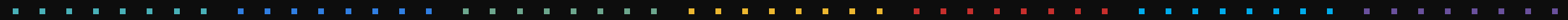


React #2 | Components





ΠΕΡΙΕΧΟΜΕΝΑ



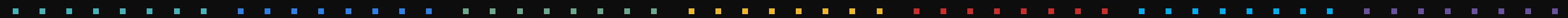


Περιεχόμενα

- Components
 - JSX
 - Props
- Rendering
 - Conditionals
 - Loops
- Events



COMPONENTS





Components

Composition & Re-usability

Η **React** βασίζεται στη φιλοσοφία της σύνθεσης (*composition*) και όχι σε αυτή της κληρονομικότητας (*inheritance*).

Τα **Components** επιτρέπουν το διαχωρισμό του *UI* σε μικρά, ανεξάρτητα, επαναχρησιμοποιούμενα κομμάτια κώδικα.

Παρέχουν ένα βαθμό ενθυλάκωσης (*encapsulation*), όπως και τα *Web Components*, αν και δεν είναι αυτός ο βασικός σκοπός της **React**.

Βασικός σκοπός της **React** είναι να αποτυπώνει τα δεδομένα μιας εφαρμογής στο **DOM**.



Components

Virtual DOM

Το **Virtual DOM** είναι ένα μια ιδεατή αναπαράσταση του **UI**, στη μνήμη, η οποία συγχρονίζει με το πραγματικό **DOM** μέσω μιας βιβλιοθήκης όπως η **ReactDOM**.

Η παραπάνω διαδικασία ονομάζεται *reconciliation*.

Η προσέγγιση αυτή επιτρέπει την *declarative* φύση της **React**. Δηλώνεις την κατάσταση στην οποία θες να βρίσκεται το **UI** και η βιβλιοθήκη αναλαμβάνει να το κάνει.

Αντί να αναφέρεται απευθείας στο **DOM**, δημιουργεί μια αφηρημένη έκδοχή του.



Components

Function vs Class Components

Ο ενδεδειγμένος τρόπος για να δηλωθεί ένα **Component** στη **React**, από την έκδοση **16.8** και έπειτα, είναι μέσω μιας **JS** συνάρτησης (*function component*).

Πριν την έκδοση αυτή, για τον ίδιο σκοπό, χρησιμοποιούνταν κλάσεις (*class component*).

Τα *function component* υπήρχαν και νωρίτερα από την έκδοση **16.8**, αλλά περιορίζονταν σε απλές χρήσεις, καθώς δεν μπορούσαν να διαχειριστούν *internal state* κ.λπ.

Αυτό άλλαξε με την εισαγωγή των *React hooks*.



Components

Κανόνες

Ένα *function component* πρέπει:

- Να επιστρέφει ένα *React element*, είτε μέσω της *createElement*, είτε με χρήση της **JSX**.
 - Αν δεν υπάρχει ένα μοναδικό *root element* αλλά περισσότερα, μπορεί να γίνει χρήση του **<React.Fragment>**.
- Να έχει όνομα που ξεκινά με κεφαλαίο γράμμα.



Components

Παράδειγμα

```
const styles = { color: '#61DAFB', backgroundColor: 'black' };

function Header() {
  return React.createElement(
    'h1',
    {
      id: 'main-header',
      style: styles
    },
    'Hello, from JSX! Time is ',
    new Date().toLocaleTimeString()
  );
}

const root = ReactDOMClient.createRoot(document.getElementById('app'));
root.render(React.createElement(Header));
```



Components

JSX

Είναι μια **XML** επέκταση της **ECMAScript** που επιτρέπει να συνδυάζουμε **markup** με κώδικα **JavaScript** με έναν ευπαρουσίαστο τρόπο.

```
const element = <h1>Hello, world!</h1>;
```

Η **JSX** θυμίζει αλλά δεν είναι ούτε **HTML**, ούτε κάποια *template engine*.

Ως μη επίσημη επέκταση της **ECMAScript**:

- δεν μπορεί να εκτελεστεί απευθείας από το **browser**, χρειάζεται είτε τη βοήθεια μιας βιβλιοθήκης (**Babel**), είτε *transpilation*.
- μπορεί να εμπεριέχει οποιονδήποτε κώδικα **JavaScript**.



Components

Props

Εννοιολογικά, τα **Components** είναι σαν οποιαδήποτε άλλη συνάρτηση σε **JavaScript**.

Δέχονται είσοδο (**props**) και επιστρέφουν **React element**, τα οποία περιγράφουν τι πρέπει να αποτυπωθεί στο **UI**.

Όλα τα *attributes* που υπάρχουν στη **JSX** περνούν, μέσω του αντικειμένου **props**, στο αντίστοιχο **Component**.

Επιπλέον, αν ένα **Component** περικλείει άλλα **Component** ή **HTML tag**, αυτά είναι διαθέσιμα μέσω του **props.children**.



Components

JSX & Curly Braces

Αν ένα **property/attribute** παίρνει τιμή "δυναμικά", τότε πρέπει να χρησιμοποιηθούν τα **{ }** αντί των **" "** / **' '**.

Το ίδιο ισχύει και όταν θέλουμε να περάσουμε μια τιμή που δεν είναι αλφαριθμητική (**string**).

Τα **{ }** μπορούν να χρησιμοποιηθούν και για εκφράσεις. Αυτό σημαίνει ότι μπορούμε να ενσωματώσουμε οποιαδήποτε έγκυρη έκφραση **JavaScript** (όπως μεταβλητές, κλήσεις συναρτήσεων ή υπολογισμούς) απευθείας μέσα στην **JSX**.

Η έκφραση αυτή "αποτιμάται" και το αποτέλεσμα της εμφανίζεται στην τελική , επιτρέποντας έτσι τη δυναμική δημιουργία περιεχομένου.



Components

Παράδειγμα #1

```
const title = 'This is a title';

function App() {
  return (
    <Header title={title} size={2} />
  );
}

const root = ReactDOMClient.createRoot(document.getElementById('app'));
root.render(<App />);
```



Components

Παράδειγμα #2

```
const styles = { color: '#61DAFB', backgroundColor: 'black' };

function Header() {
  return (
    <h1 id="main-header" style={styles}>
      Hello, from JSX! Time is {new Date().toLocaleTimeString()}
    </h1>
  );
}
```



Components

JSX & HTML Attributes

Η **JSX** δεν είναι απλά *markup*, για αυτό και έχει κάποιες ιδιότητες που την κάνουν να ξεχωρίζει από την *HTML*.

- Αντί της ιδιότητας **class**, χρησιμοποιούμε **className**.
- Αντί της ιδιότητας **for** χρησιμοποιούμε **htmlFor**.
- Τα ονόματα των ιδιοτήτων των στυλ είναι γραμμένα σε *camelCase*, π.χ. **fontSize** και **backgroundColor**.



Components

Παράδειγμα #1

```
const cssAttrs = { color: '#61DAFB', backgroundColor: 'black' };

function Header(props) {
  return (
    <h1 id="main-header" style={props.style}>
      Hello, from {props.greeter}!
      Time is {new Date().toLocaleTimeString()}
    </h1>
  );
}
```




Components

Παράδειγμα #2

```
function StyledComponent() {
  return (
    <div
      className="container"
      style={{
        fontSize: '16px',
        backgroundColor: '#f0f0f0'
      }}>
      <label htmlFor="nameInput">Όνομα:</label>
      <input id="nameInput" type="text" />
    </div>
  );
}
```



Components

Prop destructuring

Τα **function components**, όπως προαναφέρθηκε, δέχονται το πολύ μία παράμετρο.

Η παράμετρος αυτή, συνήθως, έχει το όνομα **props**, χωρίς αυτό να είναι δεσμευτικό.

Για να αποφευχθεί η συνεχής αναφορά στην παράμετρο αυτή, πολύ συχνά χρησιμοποιείται η τεχνική του *destructuring*.

```
const Header = (props) => (<h1>{props.title}</h1>);  
  
const Header = ({ title }) => (<h1>{title}</h1>);
```



Components

Παράδειγμα #1 (με τη χρήση *arrow function*)

```
const Header = () => (
  <h1>Box Office</h1>
);

const Movie= ({ title, sum, ratingh }) => (
  <div>`${title} | ${sum}m | ${rating}★`</div>
);

const App = () => (
  <>
    <Header />
    <Movie title="The Godfather" sum={134} rating={9.2} />
    <Movie title="The Shawshank Redemption" sum={58} rating={9.3} />
  </>
);

const root = ReactDOMClient.createRoot(document.getElementById('app'));
root.render(<App />);
```



Components

Παράδειγμα #2

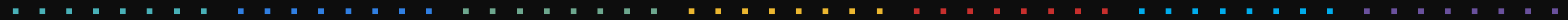
```
function CodeFormatter({ children }) {
  const style = {
    backgroundColor: "lightgrey",
    padding: "1em",
  };
  return (<pre style={style}>{children}</pre>);
}

function App() {
  return (
    <CodeFormatter>
      let a = 1;
    </CodeFormatter>
  );
}

const root = ReactDOMClient.createRoot(document.getElementById('app'));
root.render(React.createElement(Header));
```



RENDERING





Rendering

Loops & Conditionals

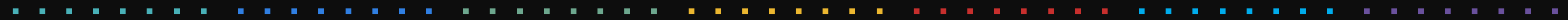
Η **React** ακολουθεί τη φιλοσοφία του *Functional Programming*.

Σε αυτή επικρατεί η *declarative* προσέγγιση, έναντι της *imperative*. Μέσα στα *curly brackets* της **JSX** ενσωματώνουμε *expression* και όχι *statement*.

Για το λόγο αυτό, σε κώδικα **React** θα δούμε να χρησιμοποιείται συχνά η μέθοδος **map**, στη θέση εντολών όπως οι **for**, **while**, κ.λπ.

Επίσης, η *conditional* λογική αναπαριστάται, όχι με τις εντολές **if** ή **switch**, αλλά με τους *ternary*, *logical* και *nullish coalescing* τελεστές.

Αυτά αφορούν κυρίως στη **JSX**.





Rendering

Παράδειγμα #1 (*conditionals*)

```
// user = { name: 'John', age: 25, isAdmin: true };

function UserDetails({user}) {
  const isAdmin = user.isAdmin ? 'Admin' : 'User';
  const age = user.age || 'Unknown';
  const name = user.name && <strong>{user.name}</strong>;

  return (
    <div>
      <p>{isAdmin}</p>
      <p>{age}</p>
      <p>{name}</p>
    </div>
  );
};
```



Rendering

Παράδειγμα #2 (inline *conditionals*)

```
// user = { name: 'John', age: 25, isAdmin: true };

function UserDetails({user}) {
  return (
    <div>
      <p>{user.isAdmin ? 'Admin' : 'User'}</p>
      <p>{user.age || 'Unknown'}</p>
      {user.name &&
        <p><strong>{user.name}</strong></p>
      }
    </div>
  );
};
```




Rendering

Παράδειγμα #3 (*conditionals*)

```
function App() {
  ...
  return (
    <>
      <div>
        {user.name ?? user.username}
      </div>
      <div>
        {user.isLoggedIn ? (
          <button>Logout</button>
        ) : (
          <button>Login</button>
        )}
      </div>
      {user.isAdmin && <AdminPanel />}
    </>
  );
}
```



Rendering

Παράδειγμα #4 (*loops*)

```
const products = [
  { id: '0001', name: 'Awesome dress', price: 100 },
  { id: '0002', name: 'Cool shoes', price: 50 },
  { id: '0003', name: 'Nice hat', price: 20 },
];

function ProductList({ products }) {
  const items = [];
  for (const product of products) {
    items.push(
      <li key={product.id}>
        {product.name} (${product.price})
      </li>
    );
  }

  return (<ul>{items}</ul>);
}
```



Rendering

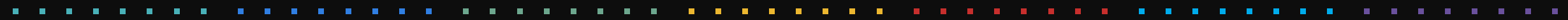
Παράδειγμα #5 (*loops* with *map*)

```
const products = [
  { id: '0001', name: 'Awesome dress', price: 100 },
  { id: '0002', name: 'Cool shoes', price: 50 },
  { id: '0003', name: 'Nice hat', price: 20 },
];

function ProductList({ products }) {
  return (
    <ul>
      {products.map(product => (
        <li key={product.id}>
          {product.name} (${product.price})
        </li>
      ))}
    </ul>
  );
}
```



EVENTS





Events

Responding to events

Η εκτέλεση κώδικα, ως "απάντηση" σε κάποιο *UI event*, γίνεται με την προσθήκη *event handler* στα στοιχεία του *UI*.

Στην **JSX**, οι *event handler* προστίθενται ως *attributes* με την προσθήκη του προθέματος **on**.

```
const handleClick = () => { alert('Button clicked!'); };
```

```
<button onClick={handleClick}>Click me!</button>
```

```
<button onClick={() => { alert('Button clicked!'); }}>Click me!</button>
```



Events

Παράδειγμα #1

```
function TextInput() {  
  function handleChange(event) {  
    console.log('Input changed to:', event.target.value);  
  }  
  
  return (  
    <input type="text"  
      placeholder="Type here..."  
      onChange={handleChange} />  
  );  
}
```



Events

Παράδειγμα #2

```
function SelectListLibrary({ items }) {  
  return (  
    <select onChange={(event) => { alert(event.target.value); }}>  
      {items.map(item => (  
        <option key={item.id} value={item.id}>{item.name}</option>  
      ))}  
    </select>  
  );  
}
```



Χρήσιμα links

⚙ Your First Component – React

<https://react.dev/learn/your-first-component>

⚙ Importing and Exporting Components – React

<https://react.dev/learn/importing-and-exporting-comp...>

⚙ Writing Markup with JSX – React

<https://react.dev/learn/writing-markup-with-jsx>

⚙ JavaScript in JSX with Curly Braces – React

<https://react.dev/learn/javascript-in-jsx-with-curly-braces>

⚙ Passing Props to a Component – React

<https://react.dev/learn/passing-props-to-a-component>

⚙ Conditional Rendering – React

<https://react.dev/learn/conditional-rendering>

⚙ Rendering Lists – React

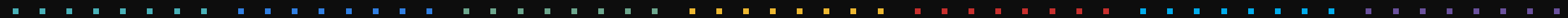
<https://react.dev/learn/rendering-lists>

⚙ Responding to Events – React

<https://react.dev/learn/responding-to-events>

⚙ React Function Components


<https://www.robinwieruch.de/react-function-component>






Extra info

 Conditional (ternary) operator - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

 Nullish coalescing operator (??) - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing_operator

 Destructuring assignment - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

 A Beginner's Guide to Loops in React JSX
<https://www.telerik.com/blogs/beginners-guide-loops-in-react-jsx>



THANK YOU!