

REACT

ITS 2024-25

LE BASI

```
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="content">Content</div>
  </body>
</html>
```

```
body {  
  padding: 0;  
  margin: 0;  
  background: #fff;  
}  
  
.content {  
  max-width: 900px;  
  margin: 0 auto;  
}
```

```
const el = document.querySelector('.content');  
el.innerHTML = "Contenuto aggiornato";
```

- **innerHTML** è altamente sconsigliato per ragioni di sicurezza

OBIETTIVI

- modificare il contenuto della pagina
- reagire agli eventi che si verificano (es: click)

REACT

- sintassi facile e sicura per scrivere HTML usando javascript
- **reactive**: permette di modificare l'HTML in seguito ad eventi

JSX

- sintassi simile ad HTML
- in realtà è **javascript** “mascherato”


```
<MyComponent textVariable="Contenuto da inserire" />
```

- Componente con l'iniziale maiuscola
- attributi camelCase
- gli attributi possono contenere anche numeri, booleani, funzioni, array... {tra parentesi graffe}
 - *(in HTML gli attributi sono sempre stringhe)*

Un componente è una funzione javascript
che ritorna un altro componente

```
function MyComponent(props) {  
  return (  
    <div className="content">  
      {props.textVariable}  
    </div>  
  )  
}
```

Gli attributi in react si chiamano **props**

```
const Page = () => (  
  <Layout>  
    <Header>  
      <Logo img="./company.png" />  
      <Navigation activePage="home" />  
      <Search />  
    </Header>  
    <Container>  
      <MyComponent textVariable="Contenuto principale" />  
    </Container>  
  </Layout>  
) ;
```

Composition: per ottenere applicazioni complesse,
combina diversi componenti in una struttura
gerarchica, come in HTML

REACTIVE

Facciamo in modo che l'applicazione si comporti in
modo dinamico

Passiamo una funzione come parametro a un componente:

```
1 <Search
2   onSearch={async (searchQuery) => {
3     const searchResults = await
4       fetch('http://api.com/search?' + searchQuery);
5     setContent(searchResults);
6   }}
7 />
```

- le funzioni sono first class objects
- posso passare una funzione come parametro di un'altra

Passiamo una funzione come parametro a un componente:

```
1 <Search
2   onSearch={async (searchQuery) => {
3     const searchResults = await
4       fetch('http://api.com/search?' + searchQuery);
5     setContent(searchResults);
6   }}
7 />
```

- le funzioni sono first class objects
- posso passare una funzione come parametro di un'altra

Ora vediamo come quella funzione viene utilizzata all'interno del componente:

```
1  const Search = ({ onSearch }) => {  
2    return (  
3      <form onSubmit={ () => onSearch(what) }>  
4        Cerca:  
5        <input type="text" name="what" />  
6      </form>  
7    )  
8  }
```

pseudo-code: il form non funziona così

Ora vediamo come quella funzione viene utilizzata all'interno del componente:

```
1  const Search = ({ onSearch }) => {  
2    return (  
3      <form onSubmit={ () => onSearch(what) }>  
4        Cerca:  
5        <input type="text" name="what" />  
6      </form>  
7    )  
8  }
```

pseudo-code: il form non funziona così

Ora vediamo come quella funzione viene utilizzata all'interno del componente:

```
1 const Search = ({ onSearch }) => {  
2   return (  
3     <form onSubmit={ () => onSearch(what) }>  
4       Cerca:  
5       <input type="text" name="what" />  
6     </form>  
7   )  
8 }
```

pseudo-code: il form non funziona così

Un passo indietro: cos'è quel setContent?

```
1 <Search
2   onSearch={async (searchQuery) => {
3     const searchResults = await
4       fetch('http://api.com/search?' + searchQuery);
5     setContent(searchResults);
6   }}
7 />
```

Un passo indietro: cos'è quel setContent?

```
1 <Search
2   onSearch={async (searchQuery) => {
3     const searchResults = await
4       fetch('http://api.com/search?' + searchQuery);
5     setContent(searchResults);
6   }}
7 />
```

E' un hook di React, che permette di gestire una variabile di stato.

```
1  const Page = () => {
2    const [content, setContent] = useState("Contenuto inizial
3    return (
4      <Layout>
5        <Search onSearch={async (searchQuery) => {
6          const searchResults = await
7            fetch('http://api.com/search?' + searchQuery);
8          setContent(searchResults);
9        }}/>
10     <MyComponent textVariable={content} />
11   </Layout>
12 );
13 }
```

E' un hook di React, che permette di gestire una variabile di stato.

```
1  const Page = () => {
2    const [content, setContent] = useState("Contenuto inizial
3    return (
4      <Layout>
5        <Search onSearch={async (searchQuery) => {
6          const searchResults = await
7            fetch('http://api.com/search?' + searchQuery);
8          setContent(searchResults);
9        }}/>
10     <MyComponent textVariable={content} />
11   </Layout>
12 );
13 }
```

E' un hook di React, che permette di gestire una variabile di stato.

```
1  const Page = () => {
2    const [content, setContent] = useState("Contenuto inizial
3    return (
4      <Layout>
5        <Search onSearch={async (searchQuery) => {
6          const searchResults = await
7            fetch('http://api.com/search?' + searchQuery);
8          setContent(searchResults);
9        }}/>
10     <MyComponent textVariable={content} />
11   </Layout>
12 );
13 }
```

E' un hook di React, che permette di gestire una variabile di stato.

```
1  const Page = () => {
2    const [content, setContent] = useState("Contenuto inizial
3    return (
4      <Layout>
5        <Search onSearch={async (searchQuery) => {
6          const searchResults = await
7            fetch('http://api.com/search?' + searchQuery);
8          setContent(searchResults);
9        }}/>
10     <MyComponent textVariable={content} />
11   </Layout>
12 );
13 }
```

E' un hook di React, che permette di gestire una variabile di stato.

```
1  const Page = () => {
2    const [content, setContent] = useState("Contenuto inizial
3    return (
4      <Layout>
5        <Search onSearch={async (searchQuery) => {
6          const searchResults = await
7            fetch('http://api.com/search?' + searchQuery);
8          setContent(searchResults);
9        }}/>
10     <MyComponent textVariable={content} />
11   </Layout>
12 );
13 }
```


Quando cambia lo stato di un componente, la funzione che lo rappresenta viene eseguita nuovamente (rendering) con il valore di stato aggiornato.

APPROFONDISCI

- [React.dev](https://react.dev)

GESTISCI LO STATO

Lo stato è la “memoria interna” di un componente

```
function MyButton() {  
  const [count, setCount] = useState(0);  
  // ...  
  return <div></div>  
}
```

```
function MyButton() {  
  const [count, setCount] = useState(0);  
  // ...  
  return <div></div>  
}
```

- `useState(initialValue)`: imposta un valore iniziale (zero, nell'esempio)
- `count`: il valore corrente dello stato
- `setCount`: una funzione che serve per modificare il valore di `count`

```
function MyButton() {  
  const [count, setCount] = useState(0);  
  const handleClick = () => setCount(count + 1);  
  return <button onClick={handleClick}>  
    Clicked {count} times  
  </button>  
}
```

Ad ogni click di button, il counter viene incrementato e il suo valore viene visualizzato all'interno del pulsante (variabile di stato {count})

Prova ad usare le funzionalità di **stato** sulla
documentazione ufficiale di React

STATO CONDIVISO

A volte vogliamo che lo stato di un componente sia utilizzabile anche in un altro componente.

Individua il componente comune più vicino

```
1 <div class="container">
2   <div class="column">
3     <HandleState /> {/* Modifica lo stato */}
4   </div>
5   <div class="column">
6     <DisplayState /> {/* Visualizza o stato */}
7   </div>
8 </div>
```


Individua il componente comune più vicino

```
1 <div class="container">
2   <div class="column">
3     <HandleState /> {/* Modifica lo stato */}
4   </div>
5   <div class="column">
6     <DisplayState /> {/* Visualizza lo stato */}
7   </div>
8 </div>
```

Trasformiamo il `<div>` in un Componente

```
1 <Container>
2   <div class="column">
3     <HandleState /> {/* Modifica lo stato */}
4   </div>
5   <div class="column">
6     <DisplayState /> {/* Visualizza o stato */}
7   </div>
8 </Container>
```

In quel componente puoi gestire lo stato a cui
accedono i componenti figli

```
1  const Container = () => {  
2    const [state, setState] = useState('initial');  
3  
4    return (  
5      <div class="container">  
6        <div class="column">  
7          <HandleState onClick={setState} />  
8        </div>  
9        <div class="column">  
10         <DisplayState value={state} />  
11        </div>  
12      </div>  
13    )  
14  }
```

In quel componente puoi gestire lo stato a cui
accedono i componenti figli

```
1  const Container = () => {  
2    const [state, setState] = useState('initial');  
3  
4    return (  
5      <div class="container">  
6        <div class="column">  
7          <HandleState onClick={setState} />  
8        </div>  
9        <div class="column">  
10         <DisplayState value={state} />  
11        </div>  
12      </div>  
13    )  
14  }
```

In quel componente puoi gestire lo stato a cui
accedono i componenti figli

```
1  const Container = () => {  
2    const [state, setState] = useState('initial');  
3  
4    return (  
5      <div class="container">  
6        <div class="column">  
7          <HandleState onClick={setState} />  
8        </div>  
9        <div class="column">  
10         <DisplayState value={state} />  
11        </div>  
12      </div>  
13    )  
14  }
```

In quel componente puoi gestire lo stato a cui
accedono i componenti figli

```
1  const Container = () => {  
2    const [state, setState] = useState('initial');  
3  
4    return (  
5      <div class="container">  
6        <div class="column">  
7          <HandleState onClick={setState} />  
8        </div>  
9        <div class="column">  
10         <DisplayState value={state} />  
11        </div>  
12      </div>  
13    )  
14  }
```

Prova ad usare lo **stato condiviso** nell'esempio sulla documentazione ufficiale di React

REACT CONTEXT

Quando lo stato di un componente deve essere utilizzato da un altro, che si trova molto lontano nell'alberatura di componenti

Esempio

```
<ComponentWithState> // questo componente contiene uno stato
  <Nested>
    <Nested>
      <Nested>
        <Nested>
          <TargetComponent /> // ...che deve essere usato qui
        </Nested>
      </Nested>
    </Nested>
  </Nested>
</ComponentWithState>
```

```
function ComponentWithState() {  
  const [state, setState] = useState();  
  return <Nested state={state}/>  
}
```

Usando la tecnica delle props, il componente Nested
fa solo da passacarte

```
function Nested({state}) {  
  return <TargetComponent state={state}/>  
}
```

Context permette di scavalcare i passacarte e rendere disponibile lo stato a qualsiasi livello innestato

```
1 export const Context = createContext();
2 function ComponentWithState() {
3   const [state, setState] = useState();
4   return (
5     <Context.Provider value={state}>
6       <Nested /> // non fa più da passacarte
7     </Context.Provider>
8   )
9 }
```

```
1 function TargetComponent() {
2   const state = useContext(Context);
3   // ...utilizza lo stato del componente superiore
4   return "...";
5 }
```

SIDE EFFECTS

Hai notato che per modificare lo stato di un componente abbiamo sempre utilizzato `onClick`?

Questa funzione è un **event handler**, che ci permette di modificare lo stato quando l'utente esegue un'azione.

Ma talvolta lo stato deve essere modificato in base ad altri *eventi*, non generati dall'utente.

Chiameremo questi eventi **side effects**.

Oppure, quando avviene un cambiamento di stato deve essere invocata una funzione esterna: anche questo è un **side effect**.

Per gestire i side effects
utilizzeremo un nuovo *hook* di React:

```
1  const Component = () => {  
2  
3    useEffect(() => {  
4      // side effect code  
5    }, [dependencies]);  
6  
7    return <>...qualcosa</>  
8  }
```


Un utilizzo tipico: visualizzare il contenuto di una pagina

```
const Component = ({ pageName }) => {  
  const [pageContent, setPageContent] = useState("");  
  
  useEffect(() => {  
    fetch('/api/' + pageName)  
      .then(res => res.json())  
      .then(setPageContent);  
  }, [pageName]);  
  
  return <>{pageContent}</>  
}
```

Quando cambia `pageName` viene invocata l'api che va a modificare la variabile `pageContent`.

TIPS

```
<div>
  {isLoggedIn ? (
    <AdminPanel />
  ) : (
    <LoginForm />
  )}
</div>
```

- operatori ternari per il conditional rendering

```
<div>  
  {isLoggedIn && <AdminPanel />}  
</div>
```

- se non ti serve `else`, usa l'operatore logico `&&`

```
const listItems = products.map(product =>
  <li key={product.id}>
    {product.title}
  </li>
);

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

- usa i metodi degli array

```
function MyButton() {  
  const [count, setCount] = useState(0);  
  return (  
    <button onClick={() => setCount(count + 1)}>  
      Clicked {count} times  
    </button>  
  );  
}  
  
export default function MyApp() {  
  return (  
    <MyButton />  
    <MyButton />  
  );  
}
```

- ogni componente ha il suo stato

```
1 function MyButton({count, setCount}) {  
2   return (  
3     <button onClick={() => setCount(count + 1)}>  
4       Clicked {count} times  
5     </button>  
6   );  
7 }  
8  
9 export default function MyApp() {  
10   const [count, setCount] = useState(0);  
11   return (  
12     <MyButton count={count}, setCount={setCount}/>  
13     <MyButton count={count}, setCount={setCount}/>  
14   );  
15 }
```

- stato condiviso tra più componenti

```
1 function MyButton({count, setCount}) {
2   return (
3     <button onClick={() => setCount(count + 1)}>
4       Clicked {count} times
5     </button>
6   );
7 }
8
9 export default function MyApp() {
10   const [count, setCount] = useState(0);
11   return (
12     <MyButton count={count}, setCount={setCount}/>
13     <MyButton count={count}, setCount={setCount}/>
14   );
15 }
```

- stato condiviso tra più componenti

ERRORI DA EVITARE


```
1 function MyButton() {  
2   const [count, setCount] = useState(0);  
3   return (  
4     <button onClick={() => { count = count + 1 }}> // NO!  
5       Clicked {count} times  
6     </button>  
7   );  
8 }
```

- **mai** modificare lo stato direttamente, utilizza sempre la funzione `setState` (qui chiamato `setCount`)

```
1 function MyButton({count}) {  
2   return (  
3     <button onClick={() => { count = count + 1 }}> // NO!  
4       Clicked {count} times  
5     </button>  
6   );  
7 }
```

- **rispetta** il flusso di dati, sempre dall'alto verso il basso. Non modificare una proprietà di un componente di livello superiore (in questo esempio: una prop)

```
1 function MyButton({count, setCount}) {  
2   return (  
3     <button onClick={() => {setCount(count + 1)}}>  
4       Clicked {count} times  
5     </button>  
6   );  
7 }
```

Un componente figlio può modificare lo stato di un genitore, se gli viene fornita la funzione per modificarlo (tramite props o context)

THINKING IN REACT

Ovvero: come procedere quando realizzi
un'applicazione React

☐ Only show products in stock

Name	Price
Fruits	
Apple	\$1
Dragonfruit	\$1
Passionfruit	\$2
Vegetables	
Spinach	\$2
Pumpkin	\$4
Peas	\$1

1. FilterableProductTable

2. SearchBar

3. ProductTable

4. ProductCategoryRow

5. ProductRow

- **disegna una gerarchia di componenti**

- costruisci una versione **statica** in React

- progetta una rappresentazione minima ma completa dello stato
- identifica dove implementare lo stato
 - quali componenti devono gestire lo stato?
 - qual è il primo genitore comune di due componenti che devono condividere lo stesso stato?
 - **lift state up**

- implementa le prop che contengono le funzioni di callback (**inverse data flow**)

APPROFONDISCI

[React.dev](https://react.dev)

INIZIALIZZARE L'APPLICAZIONE

```
1 <html>
2   <head></head>
3   <body>
4     <div id="app"></div>
5   </body>
6 </html>
```

```
1 import { createRoot } from 'react-dom/client';
2 import { Page } from './Page.jsx'
3 const root = createRoot(document.getElementById('app'));
4 root.render(<Page />);
```

LIBRERIE

Utilizzare una libreria di componenti (UI Kit): MUI

```
export const Header: React.FC = () => {  
  return (  
    <Stack  
      direction="row"  
      spacing={2}  
      sx={{  
        justifyContent: "space-between",  
        alignItems: "center",  
        paddingTop: "16px",  
      }}  
    >  
        
      <Stack direction="row" spacing={2}>  
        <List role="menubar" orientation="horizontal">  
          <ListItem role="none">
```

Utilizzare una libreria per il routing: React Router

```
function App() {  
  return (  
    <BrowserRouter basename="/">  
      <Routes>  
        <Route path="/" element={<Layout />}>  
          <Route index element={<Root />} />  
  
          { /* Teacher */}  
          <Route path="teacher" element={<TeacherGuard />}>  
            <Route index element={<Exams />} />  
            <Route path="exam" element={<AddExam />} />  
            <Route path="exam/:id" element={<EditExam />} />  
            <Route path="subscriptions" element={<Subscription />} />  
            <Route path="subscriptions/:date" element={<Session />} />  
            <Route path="subscriptions/:date/:id" element={<Session />} />  
          </Route>  
        </Routes>  
      </BrowserRouter>  
    )  
  }  
}
```

