REACT

ITS 2024-25

LE BASI

```
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"

- 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

Gli attributi in react si chiamano props

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:

- 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:

pseudo-code: il form non funziona così

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

pseudo-code: il form non funziona così

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

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 />
```

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

```
const [content, setContent] = useState("Contenuto inizial
     const searchResults = await
      setContent(searchResults);
```

```
const searchResults = await
setContent(searchResults);
```

```
const [content, setContent] = useState("Contenuto inizial
     const searchResults = await
      setContent(searchResults);
```

```
const searchResults = await
           setContent(searchResults);
10
         <MyComponent textVariable={content} />
```

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

APPROFONDISCI

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

Individua il componente **comune** più vicino

Trasformiamo il <div> in un Componente

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

```
const Container = () => {
     const [state, setState] = useState('initial');
     return (
       <div class="container>
         <div class="column">
           <HandleState onClick={setState} />
         </div>
         <div class="column">
           <DisplayState value={state} />
        </div>
12 </div>
13 )
14 }
```

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

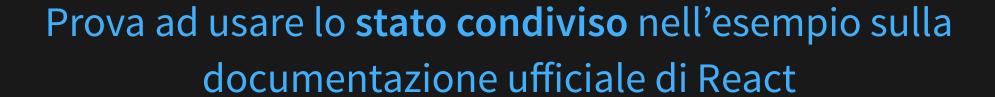
```
const [state, setState] = useState('initial');
      <HandleState onClick={setState} />
```

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

```
const [state, setState] = useState('initial');
      <HandleState onClick={setState} />
```

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

```
const [state, setState] = useState('initial');
      <HandleState onClick={setState} />
     <DisplayState value={state} />
```



REACT CONTEXT

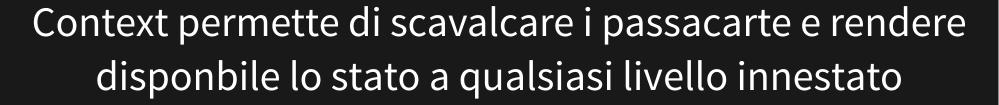
Quando lo stato di un componente deve essere utilizzato da un altro, che si trova molto lontano nell'alberatura di componenti

Esempio

```
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}/>
}
```



```
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 &&

• usa le funzioni 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

```
function MyButton({count, setCount}) {
     return (
       <button onClick={() => setCount(count + 1)}>
         Clicked {count} times
       </button>
     );
   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
     );
```

• stato condiviso tra più componenti

```
const [count, setCount] = useState(0);
```

• stato condiviso tra più componenti

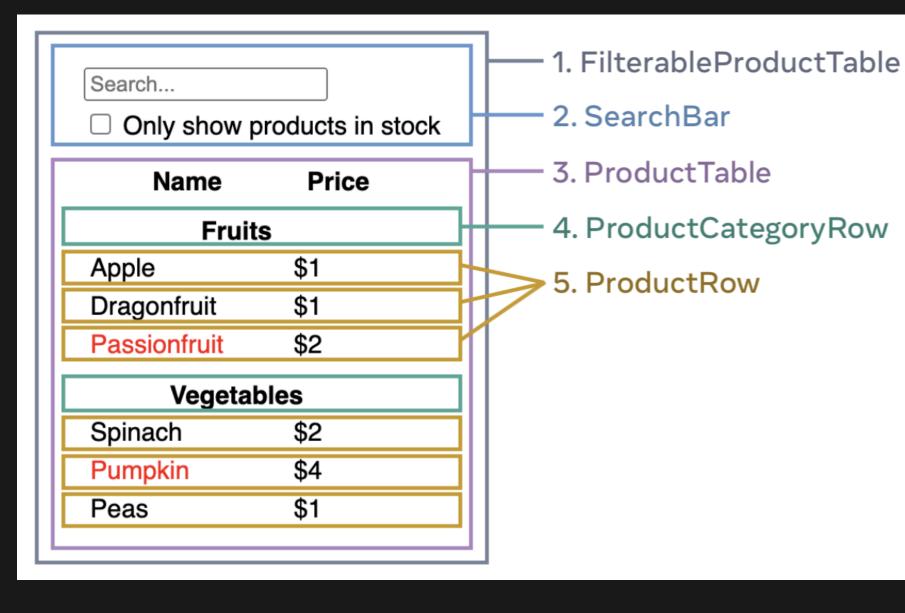
ERRORI DA EVITARE

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

 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)

THINKING IN REACT

Ovvero: come procedere quando realizzi un'applicazione React



• 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

INIZIALIZZARE L'APPLICAZIONE

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",
      <imq src="/logo.svq" height={50} />
      <Stack direction="row" spacing={2}>
        <List role="menubar" orientation="horizontal">
          <TistTtem 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</pre>
            <Route path="subscriptions/:date" element={<Session</pre>
            <Route
```