React

React.js è una libreria javaScript per la creazione di interfacce utente Web.

Come lavora?

React crea un DOM “virtuale” in memoria. Invece di manipolare direttamente il DOM del browser, React crea questo “Virtual DOM” dove fa tutte le modifiche del caso, prima di fare i cambiamenti sul browser DOM (*che avvengono confrontando proprio i due DOM*).

React cambia solo ciò che deve essere cambiato

L’obiettivo di React è eseguire in vari modi il rendering della pagina web HTML, tramite una funzione `createRoot()` e il suo metodo `render()`.

```
const container = document.querySelector("#root");
ReactDOM.createRoot(container).render(<p> Hello ! </p>);
```

Definiamo JSX il linguaggio “*unione*” tra JS e XML, che ci permette di scrivere in HTML su pagine React.

Con questo linguaggio possiamo scrivere inoltre espressioni all’interno dell’HTML, all’interno di { }

React Component

I componenti React sono indipendenti e riutilizzabili “pezzi di codice”. Hanno lo stesso scopo delle funzioni JS, ma ritornano dell’HTML. Possono essere di due tipi: **componenti classe** e **componenti funzioni**.

Class Component

Un componente classe deve includere lo statement `extends React.Component` attraverso il quale viene “creata” l’ereditarietà dello stesso e gli viene dato accesso alle funzioni dei React.Component. Esso deve inoltre dichiarare un metodo `render()` che appunto ritorna il codice HTML.

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Function Component

Un componente funzione ritorna sempre dell’HTML, e ci permette di scrivere più o meno le stesse cose del componente classe ma con molto meno codice.

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

Per effettuare il rendering:

```
const root = ReactDOM.createRoot(document.getElementById('root'
root.render(<Car />);
```

Props

I componenti possono essere anche passati come **props**, che sta per *proprietà*. Le props sono come gli argomenti delle funzioni:

```
function Car(props) {
  return <h2>I am a {props.color} Car!</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'
root.render(<Car color="red"/>);
```

State

Esiste un altro modo per rappresentare le proprietà di un componente di tipo classe, detto **state**. A differenza di props, le proprietà definite all'interno l'oggetto state NON sono immutabili, anzi *state* è pensato apposta per contenere proprietà che nel tempo cambieranno (*in seguito al verificarsi di determinati eventi*).

Quando una delle proprietà all'interno di state cambia valore, viene invocata la ri-renderizzazione del relativo componente.

Nota: *le componenti function sono stateless, ovvero NON hanno un oggetto state*.

```
class Car extends React.Component{
    constructur(){
    super();
    this.state ={ brand:"Ford", model:"Mustang", color:"red"};
    }

render(){
        return(
        <div>
        <h1> My {this.state.brand} </h1>
```

```
          <P> It is {this.state.color}{this.state.model} </p>
        </div>
        );
}
}
```

Modifichiamo l'oggetto state attraverso la funzione `setState()`. L'invocazione di tale funnzione scatena una reazione da parte della libreria React, la quale provvederà a modificare lo stato e a re-invocare la funzione `render()` della classe in modo del tutto trasparente.

ATTENZIONE: l'oggetto state è incapsulato all'interno di un componente, il quale è l'unico ad avere diritto e responsabilità di modificarlo.

Raccomandazioni di uso: NON tutti i componenti dovranno avere uno state, al contrario è consigliato costruire componenti stateless.

Se dobbiamo svolgere un'operazione al cambiamento di un certo stato, non scriviamola sotto la proprietà `setState()`, bensi dentro come secondo paramentro:

```
this.setState(
    {
     count: this.state.count + 1
    },
    () =>{handlerFunction()}
)
```

Inoltre, se l'operazione va fatta sulla base dell'utlimo valore, ricordiamo di passarlo:

```
his.setState( prevState => (
    {
     count: this.state.count + 1
    }))
```