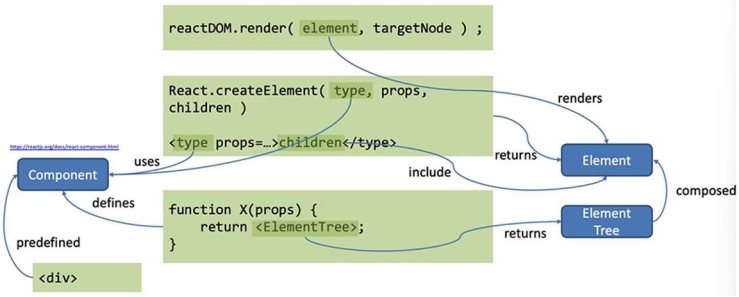
04_02 ELEMENTI E JSX

Quando creiamo un elemento in JSX, associamo quell'elemento al suo nodo. L'elemento usa un componente che viene definito da una funzione. Nel render di un elemento inseriamo un Element tree.



Un elemento è un oggetto che descrive un'istanza di un componente o un nodo del DOM (h1, h2…) con tutte lse sue proprietà.

Un ElementoReact è una rappresentazione di un elementoDOM nel virtualDom e contiene:

- Type componente (Button, ..)
- Proprietà (colore, …)
- Eventuali figli

CREAZIONE elemento

- React.createElement(type, props, children)
- Type
  - String: a DOM node identified by the tag name (e.g., 'div')
  - React component class/function: a user-defined component

- **Tipo**: qual è il componente da cui si sta realizzando l'elemento
- **Props**: oggetto semplice che può contenere:
  - Attributi semplici del DOM (type, src, hrev, alt…)
  - Valori arbitrari dei componenti react
  - Represented as object properties (not strings like HTML attributes)
    - Exceptions (reserved words): `class` → `className`, `for` → `htmlFor`
- **Figli**:
  - a ReactNode object, that may be:
    - A string or number: text content of the nodes
    - A ReactElement (that may contain a tree of Elements)
    - An array of ReactNodes
  - nested Elements to be rendered as children of the element

Convenzioni:

- Elementi DOM (div, p, li, img)→ lowercase
- Componenti React (WarningButton, LoginForm, TaskList) →UpperCase
- I due tipi di elementi possono essere mischiati, incapsulati, combinare
  - React è dichiarativo → usa composition
- L'albero degli elementi descrive la pozione del DOM virtuale che si vuole visualizzare

JSX

Modo più simile all'html per scrivere componenti react in maniera snella e leggibile, componenti che sono espansi durante il rendering della pagina html.

Può usare:

- <tag> … </tag>
- <tag/>

Nota: si può usare per componenti ma anche in Array/Oggetti, in quest'ultimo caso va racchiuso tra parentesi tonde per una questione di leggibilità

```
const element = <div className="main">Hello world</div>;
```

```
const element2 =  (<Message text="Hello world" />);
```

Nota: se il componente non è presente, va importato

```
import CustomButton from './CustomButton';

function WarningButton() {
  return <CustomButton color="red" />;
}
```

I tag html vengono convertiti in props.

- String attributes become string-valued props
  - color="blue" -> {color: 'blue'}
- Other objects may be specified as a JS expression, enclosed in {}
  - shadowSize={2} -> {shadowSize: 2}
  - log={true}
  - color={warningLevel === 'debug' ? 'gray' : 'red'}
- Any JS expression is accepted

I **figli di un componente** sono passati come figli nelle props

```
<MyComponent>Hello world!</MyComponent>
```

```
<MyContainer>
  <MyFirstComponent />
  <MySecondComponent />
</MyContainer>
```

Posso avere un jsx dentro un js dentro un jsx dentro un js

- JS expressions in {} may be used to specify element children
- One child (or an array of children) are generated by an expression

```
const Menu = (<ul>{loggedInUser ? <UserMenu /> : <LoginLink />}</ul>)
```

  - <JSX> inside {JS} inside <JSX> inside JS. Totally Legit.
- undefined, null or Booleans (true, false) are not rendered
  - Useful for conditionally including children

```
return (<ul>
  <li>Menu</li>
  {userLevel === 'admin' && renderAdminMenu()}
</ul>)
```

→usati per decidere se visualizzare o meno un componente ma niente di più

In Jsx, un **valore booleano** può essere assegnato

- True, for the presence of the attribute (optional in recent React versions)
- False (or nothing) for the absence of the attribute
- JSX: <option value='WA' selected={true}>Washington</option>
- JSX : <input name='Name' disabled={true} />

In Jsx non esistono commenti, bisogna inserire un **commento javascript** {/*… */}

Nome degli attributi del DOM modificati per uniformità:

- Attributi in camelCase
- Attributo style accetta oggetti
  - <div style={{color: 'white'}}>Hello World!
  - Object keys are CSS Properties, and are camelCase (e.g., margin-top → marginTop)
  - Object values are CSS values, represented as strings

Sintassi spread utile per passare tutte le proprietà

```
const welcome = {msg: "Hello", recipient:
"World"} ;


<Component {...welcome} />
```

Class → className
for → htmlFor

Componenti React:

funzioni pure (no side-effect), proprie, idempotenti che:

- Prendono **props** opzionale come input
- Hanno un **return** obbligatorio → albero degli elementi
    - Deve avere un'unica radice/root
        - The special node `<React.Fragment>` may be used to wrap a list of element into a single root.
            - React.Fragment will not generate any node at the DOM level
        - A shortcut syntax for fragments is `<>` … `</>`

Si possono usare metodi di manipolazione degli array:

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map(
    (number) => <li>{number}</li> );
  return (<ul>{listItems}</ul>);
}

Function App(props) {
  const numbers = [1, 2, 3, 4, 5];
  return <NumberList
numbers={numbers}/>;
}
```

quando si usano insiemi di elementi di contenuti univoci/distinguibili, ogni elemento di una lista deve avere un attributo univoco che si chiama key
→ react usa questa chiave nel VirtualDom per capire cosa è cambiato

- Always assign to each item in the list a special 'key' attribute, with unique values
    - `<li key={number}>{number}</li>`
- Most likely, we may reuse unique IDs from the data itself
    - `<li key={todo.id}>{todo.text}</li>`
- Keys **must** be specified when building the array of components
    - Usually in the `.map()` call, in the 'container' component
- Uniqueness is only required within *the same list*
    - *Not globally* on the page
- Keys are *not* available as `props` in the component

→ unicità richiesta all'interno della lista (non globalmente)