

Interimmobili s.r.l.

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S. Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

Cosa e' l'HTML?

HyperText Markup Language → e' il linguaggio usato per strutturare ~~contenuto~~ la pagina web, cioè a dire al browser cosa c'e' nella pagina. 1^a versione nel 1991 di Al

2014 HTML5

Cosa e' CSS?

Cascading Style Sheets → linguaggio usato per dare stile alla pagina HTML (colori, dimensioni, font, position ecc.)

- C'e' il CSS inline → dentro un singolo tag HTML
- Internal CSS → dentro il tag style dentro l'head
- External CSS → in un file separato (+ usato x ordine e) stabilita'

INLINE PREVALICA
SUL INTERNAL

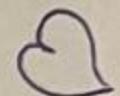
TAG BLOCK

l'elemento occupa tutta la larghezza disponibile e va a capo automaticamente prima e dopo es. div, p, h, ul, li

TAG INLINE

l'elemento occupa solo lo spazio necessario al suo contenuto e non va a capo. esempi: span, a, img

- SELETTORI DI CLASSE → applicano lo stile a tutti gli elementi di ~~un certo tipo~~ una classe specifica
es. < p class = "aiuto" > Giao </ p >
• aiuto { color: red; } → posso usare la stessa classe + volte molto utile



- SELETTORI DI ID # → applicano lo stile ad un elemento unico, identificato da un id (l'id non si ripete)
< p id = "titolo" > Giao </ p >
titolo { color: green; }



DOM → Document Object Model → fornisce una rappresentazione del documento come una composizione gerarchica di oggetti.

→ E' un insieme di funzioni, metodi e proprietà che i programmi possono richiamare

**X ACCEDERE AGLI ELEMENTI DELLA PAGINA
FACCIAMO GETELEMENTBYID**

EVENTI → Gli eventi DOM consentono a Javascript di registrare diversi gestori di eventi su elementi in un documento HTML.

→ Sono usati in ~~una~~ combinazione con funzioni che saranno eseguite finché l'evento non si verifica

ES ONCLICK

può essere sia dentro il tag sia dentro lo script associandolo all'id e richiamandolo

ADD EVENT LISTENER → serve per ascoltare un evento e far partire una funzione, si può usare per + eventi diversi su un singolo id

- POSITION: STICKY → Si comporta come relative finché non si scrolla oltre un certo punto, poi si blocca ~~non~~ come fixed.

COSA E' LO Z-INDEX?

Si applica solo agli elementi posizionati (\neq static)

DECIDE L'ORDINE DEI NIVELLI DI PROFONDITÀ

Più è alto il numero + è vicino all'osservatore

ATRIBUTI DEI TAG HTML

servono a modificare
e rendere + facile

l'esecuzione da parte del browser del valore/azione
associato al tag es. <~~el~~ p align="right">

ELENCHI

- ORDINATI → gli elementi sono numerati 1, 2, 3...
 ogni voce è dentro un

- NON ORDINATI → sono puntati
 e dentro i

- DI DEFINIZIONE → usati x termini e definizioni —
definition <dd> <dt> e <dd>
list term description

TABEULE

→ mostrano dati organizzati in righe e colonne

<table>

<tr> → table rows

<th> → table header

<tr>

<td> → table data → cella normale

PSEUDOCLASSE CSS

→ serve x modificare lo stile in base a uno stato o condizione es button: hover
hover pseudo classe, poiché aggiunge uno stile quando il mouse passa sopra il bottone

COSA E' POSITION?

Serve a controllare come e dove un elemento viene posizionato nella pagina.

VALORI PRINCIPALI DI POSITION

- POSITION: STATIC ➔ default, le proprietà top, left, bottom, right non funzionano
- POSITION: RELATIVE ➔ l'elemento resta nella posizione naturale, ma si può spostare un po' top, left, bottom, up
nel spazio oriale Resta riservato
Permette di ~~sovrapporre~~ posizionare l'elemento nello spazio in relazione alla sua posiz. oriale
- POSITION: ABSOLUTE ➔ dipende dal contenitore con POSITION: relative + vicino, oppure a static se non c'è nessuno. L'elemento viene rimosso dal flusso normale della pagina in base a top, bottom, right e left inseriti
- POSITION: FIXED ➔ si comporta uguale ad absolute tranne che se con abs l'elemento ~~rimane bloccato~~
dove l'abbiamo messo se scorriamo con fixed rimane bloccato anche scollando

COSA E' DISPLAY?

Proprietà CSS che dice come un elemento HTML deve comportarsi nel layout della pagina.

DISPLAY: BLOCK → ogni elemento va a capo da solo

DISPLAY: INLINE → ~~solo~~ ogni elemento sta sulla stessa riga

DISPLAY: INLINE-BLOCK → stanno sulla stessa riga ma scelgo

dimensioni con width e height

DISPLAY: FLEX → trasforma un elemento in un contenitore flessibile, dove i suoi figli (i contenuti interni) si dispongono automaticamente in modo adattabile e ordinato.

• ~~contenitore~~ container {display: flex;} → tutti i figli del contenitore si mettono uno accanto all'altro sulla stessa riga.

PROPRIETÀ PRINCIPALI DEI FLEXBOX

• ~~box~~ hanno direzione sull'asse X e Y

row column

- **FLEX-DIRECTION:** row / column → decide la direzione dei box

- **JUSTIFY-CONTENT** → allinea gli elementi lungo l'asse principale (:center, centrati orizzontalmente)

- **ALIGN-ITEMS** → allinea gli elementi lungo l'asse trasversale (:center, centrati verticalmente)

• i contenuti responsive Flexbox si usa insieme a media query che permette di applicare stili diversi in base alle diverse tipologie di dispositivo



INTERIMMOBILI

Interimmobili s.r.l.

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S.Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

FORM → servono a raccogliere dati dall'utente e inviarli ad un server o gestirli con Java script definito dal tag `<form>..</form>`

→ `<input>` elemento + usato nel form es. `type = "submit"`
 sia `x:input` che per ~~button~~ `x:inniare i dati`
 → oltre a type c'è value che definisce il valore dell'input

- CSS → BORDER, MARGIN, PADDING

è possibile agire su `d`
tutti e 4 i valori del box
→ SPESORE
`BORDER-C WIDTH`, `BORDER-STYLE`,
`BORDER-COLOR`

↓
Crea spazio esterno dai box,
uso x
distanziare i vari elementi della pagina tra loro

↓
Crea spazio intorno al contenuto del box
↓
Spazio di distanza all'interno dei bordi
NON fuori come MARGIN

COSA SONO I SELETTORI? → è la parte del CSS che dice a quale elemento HTML applicare uno stile.

IL SELETTORE "SELEZIONA" UNO O + ELEMENTI HTML

- SELETTORI DI TIPO → applicano lo stile a tutti gli elementi di un certo tag es. `p {color:blue;}`



Interimmobili s.r.l.

INTERIMMOBILI

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S. Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

BOOTSTRAP → utilizzando tutto x creare contenuto responsive
scrive il tag META "viewport"

CONTAINER → contenitore a barra fissa

CONTAINER-FLUID → contenitore a barra fissa intera, ricopre l'intera barra della finestra a prescindere dalla risoluzione

COSA VUOL DIRE RESPONSIVE? → Si adatta

automaticamente alle dimensioni dello schermo
SINGOLO THREAD sincrono

→ gestisce l'asincronia grazie a callback, promise e async/await

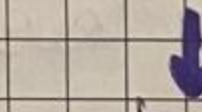
JAVASCRIPT →

- singolo thread
- funziona su browser
- linguaggio di programmazione
- orientato agli oggetti e eventi

LINGUAGGIO SINGLE THREAD → unico thread d'esecuzione

ASSEGNAZIONE PER VALORE O RIFERIMENTO

Si applica ai tipi di dato
primitivi: numeri, stringa, bool,
undefined



a tipi di dato
complessi: array,
funzioni, oggetti

Quando si assegna una variabile
ad un'altra viene copiata una
copia indipendente del valore

let a = 5 b = a let b = 10

log(b) = 10

Quando associo la
variabile ad un'altra
non copio l'oggetto ma
solo il riferimento

alla stessa
area di
memoria

NODE.JS → RUNTIME di JavaScript Open Source
multipiattaforma orientato agli eventi x l'esecuzione di codice JavaScript

VARIABILI → spazio di memoria che serve x salvare un valore
abbiamo VAR, LET e CONST

- **VAR** → vecchio modo → ha scope funzionale ovvero vive solo nelle funzioni dove è dichiarata.
- Può essere riassegnata e redefinita nella stessa area
- viene hoistata → ha l'hoisting → ovvero viene spostata in alto nel motore di JS, anche se scritta dopo

• **LET** → moderno

- scope d'istruzione → vive solo dentro il blocco {} dove è dichiarata
- si può riassegnare ma non redefinire nella stessa area

• **CONST** → COSTANTE → block-scoped come LET

- Non può essere riassegnata dopo la dichiarazione
- Se CONST contiene un'oggetto o array il riferimento non può cambiare, ma il contenuto interno sì.

ARRAY → liste di valori anche diversi

OPERATORI → ++ incremento di 1

- | | |
|------------------|---|
| = = Valore Utile | = = = Strettamente Utile, tipo e valore |
| '30' = = 30 True | '30' = = = 30 False |

ESEMPI DI GESTIONE DI VARIABILI

Var Val1 = "10";

Var Val2 = "14";

Var Val3 = Val1 * Val2; → 140 moltiplica

SENNÒ Var Val5 = Val1 + Val2 => 1014 concatena

Var Val7;

Var Val6 = Val7 + 30 → Nan



INTERIMMOBILI

gli oggetti sono
array associativi

Interimmobili s.r.l.

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S.Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

CREO NUOVI OGGETTI CON NEW OBJECT()

OGGETTI → servono x raggruppare + valori sotto

una unica entità. Contengono coppie chiave-valore e possono contenere funzioni.

RIPRESA ARRAY → particolare tipo di oggetto con

possentisi trovare

- METODI DEGLI ARRAY:

- array.push('qualcosa') → aggiunge in fondo all'array qualcosa

- array.map(item => item + '!') **TRASFORMA GLI ELEMENTI DELL'ARRAY**
oppure trasforma elementi in oggetti

DESTRUTTURAZIONE

→ di array e oggetti

[] { }

↳ ci permette di estrarre i valori di un array e assegnarli a nuove variabili in maniera + fluida

x gli array scegliamo noi i nomi delle nuove variabili,

x gli oggetti dobbiamo usare i nomi delle proprietà

CON ARRAY: const user = ['Rob', 'Del'];

destruttura. → const [firstname, lastname] = user;

CON OGGETTO: const user = { name: 'Rob', lastname: 'Del' };

const { children } = props; console.log(name) → Rob

FOR-IN → × ~~per~~ lavorare meglio con gli array (anche il for-of)
come quello di Python

FOR-OFF → ~~per~~ assegna alla variabile prima del
of array ogni volta un elemento dell'array.

FUNZIONI → blocco di codice in cui sono eseguite
+ istruzioni

Può ricevere parametri e ritornare un valore
esempio.

function testProva() {
 console.log('aiuto');
 return 'ciao';
}

→ si crea in automatico
una variabile arguments
oggetto simile ad un
array

var risultato = testProva(); → aiuto
console.log(risultato) → ~~bewerkoet~~ ciao

VARIABILI LOCALI e GLOBALE

Le variabili chiamate dentro una funzione hanno
uno scope locale

ARROW FUNCTION → var test = () => {}

le parentesi tonde stanno a significare che posso
passare dei parametri alla funzione

LE FUNCTION CLASSICHE HANNO THIS E ARRAY ASSOCIAZIVI

↓
CHE FA
RIFERIMENTO
ALLE FUNZIONI
STESSE

LE ARROW FUNCTION HANNO
IL THIS ~~OPERATO~~ RELATIVO
AL CONTESTO DI RIFERIMENTO
DOVE VENGONO CHIAMATE

const SOMMA(a, b) => {
 return a + b
};

Interimmobili s.r.l.

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S.Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

- **PROMPT()** • funzione JS x far apparire una finestra di dialogo con un campo di testo all'interno o nel quale l'utente può inserire del contenuto

RESTITUISCE UNA STRINGA

~~Confezione~~

- **IF (CONDIZIONE) {**

ISTRUZIONI

} ELSE {

ISTRUZIONI

}

- **FOR** → esecuzione di un blocco di codice ~~per~~ per un numero determinato di volte

FOR(INIZIAZIONE; CONDIZIONE;
MODIFICA) { ISTRUZIONI }

- prima JS esegue l'istruzione specificata in INIZIAZIONE.

- al termine di ciascuna iterazione viene eseguita l'istruzione MODIFICA

- **WHILE (CONDIZ.) {**

ISTRUZIONI }

finché la condizione sarà vera verranno eseguite le istruzioni

DO - WHILE

DO {

ISTRUZIONI

}

WHILE (CONDIZIONE)

la condizione viene valutata dopo aver eseguito le istruzioni



INTERIMMOBILI

Interimmobili s.r.l.

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S.Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

CALLBACK → si usano per ~~eseguire~~ rinviare l'esecuzione di un'azione finché un'altra non è finita **1)** Personalizzare il comportamento di una funzione **2)** Gestire operazioni asincrone

setTimeout(() => {

console.log('Ciao dopo 2 secondi');
}, 2000);

MAP → metodo degli array che crea un nuovo array, applicando una funzione ad ogni elemento dell'array originale. es const numeri = [1, 2, 3, 4]
const doppi = numeri.map(num => num * 2)
↓
[2, 4, 6, 8]

FILTER → crea un nuovo array con solo gli elementi che soddisfano una condizione
const numeri = [-1, 2, 3, 4]
const pari = numeri.filter(num => num % 2 === 0)
↓
[2, 4]

PROMISE

→ introdotte per risolvere i problemi di callback

es. cuociPasta()

- then(() => preparaSugo())
- then(() => apparecchiaTavola())
- then(() => console.log('Tutti a tavola!'))
- catch(() => console.log('Qualcosa è andato storto!'))

3 STATI DELLA PROMISE

- PENDING → operazione non ancora completata
- RISOLTA → operazione completata con successo
- RIFIUTATA → operazione fallita

NON PUOI USARE UN RISULTATO ASINCRONO IN UN

FLUSSO ASINCRONO → non puoi usare il risultato di una funzione asincrona subito!

es. SBAGLIATO

```
function prendiDati() {  
    setTimeout(() => {  
        return 'Dati dal server';  
    }, 2000);  
}
```

```
const dati = prendiDati();  
console.log(dati) → NULLA!
```

ESEMPIO GIUSTO

```
function prendiDati() {  
    return new Promise(resolve => {  
        setTimeout(() => {  
            resolve('Dati dal server');  
        }, 2000);  
    });  
}
```

```
→ const dati = await  
    prendiDati();  
    console.log('Dati  
    ricevuti', dati);  
}  
mostraDati();
```

```
async function mostraDati() {
```

```
    console.log('Sto aspettando i dati...'); →
```



Interimmobili s.r.l.

00198 ROMA
Piazza Ungheria, 6
Tel. 06.88.42.347
Fax 06.85.53.839
roma@interimmobili.it

20122 MILANO
Galleria S.Babila, 4B
Tel. 02.76.02.83.95
Fax. 02.76.02.84.05
milano@interimmobili.it

10121 TORINO
Corso Matteotti, 32/A
Tel. 011.50.96.081
Fax. 011.50.96.475
torino@interimmobili.it

16121 GENOVA
Via C.R. Ceccardi, 4
Tel. 010.56.70.81
Fax 010.58.32.64
genova@interimmobili.it

50122 FIRENZE
Via Por Santa Maria, 8
Tel. 055.21.81.40
Fax 055.21.39.82
firenze@interimmobili.it

ALTRÒ ESEMPIO DI CHIAMATA

```
fetch('https://api.example.com')
```

```
.then(res => res.json())
```

```
.then(dati => console.log(dati));
```

servono x gestire le PROMISE

DIFFERENZA TRA .THEN() ASYNC / AWAIT

• THEN → usato direttamente con Promise (occorri gestiti da .catch)

• ASYNC / AWAIT → permette di scrivere codice asincrono come se fosse sincrono (stesso risultato di then) occorri gestiti da (try... catch)

PROMISE ALL → metodo che serve x eseguire più operazioni asincrone in parallelo, e aspettare che tutte siano completate prima di proseguire

PROMISE ALL accetta ~~un~~ un array di promise e restituisce una nuova promise che si risolverà solo quando tutte le promise fornite saranno risolte

ESEMPIO SBAGLIATO

```
let a = '';
fetch(url).then(res => a =
= res);
console.log(a); → NO!
```

GIUSTO

```
let a = '';
fetch(url).then(res => res.json())
.then(dati => {
  a = dati;
  console.log(a); → QUESTO È OK
})
console.log(a); → QUESTO NO PERCHE'
FUORI DAL THEN
```

REACT

→ libreria Javascript.

→ le applicazioni vengono create usando moduli di codice indipendenti e riutilizzabili che gestiscono il proprio stato.
Lo stato viene mantenuto fuori dal DOM.

DOM VIRTUALE → rappresentazione visiva del DOM, quando lo stato dell'app cambia, viene aggiornato il DOM virtuale anziché quello reale, riducendo il costo delle prestazioni.

APPROCCIO ASINCRONO DI NODE JS → accedere alle risorse del sistema operativo in modalità event-driven → 'PROGRAMMAZIONE ad eventi' → si lancia un'azione ogni volta che accade qualcosa → in questo modo ogni azione risulta asincrona.

SCAFFOLDING DI REACT → il file che viene effettivamente

caricato quando facciamo partire la nostra applicazione è

index.html → punto di ingresso della pagina che è una

single page → l'unico createElementById di tutta l'app

↳ base statica <div id='root'>

index.js → punto di partenza del Javascript, collega React con l'html grazie a ReactDOM.createRoot(document.getElementById('root'));

Root.render(<App/>)

App.js = componente principale

COMPONENTE
FUNZIONALE
↑

COSA E' UN COMPONENTE REACT? funzione Javascript
che ritorna in JSX → cioè codice HTML "finto" che React traduce in vero HTML.

**FUNZIONE, LETTERA MAIUSCOLA, JSX DI RITORNO,
EXPORT DEFAULT LETTERA MAIUSCOLA**

Il componente ha uno stato proprio, può ricevere dati da fonti esterne usando le props e può rendere dinamica l'interfaccia utente grazie a queste cose.

PROPS E CHILDREN

→ + PARAMETRI LI POSSO PASSARE CON LO SPREAD OPERATOR
[PROPS] → properties → parametri che si passano a un componente dall'esterno

es. function Saluto(props) {
 return <h1> Ciao, {props.nome} </h1> }

< Saluto nome='Luca' />

React lo traduce in

Saluto({ nome: 'Luca' }); Quindi props è un oggetto

[CHILDREN] → prop speciale di React → contiene tutto ciò che scrivi tra i tag di apertura e chiusura del componente
<Card> <p> contenuto </p> </Card>

COMPONENTE REACT CREAZIONE

SINTASSI JSX

- bisogna wrappare tutto in un singolo tag <div>
sempre React ci dà la possibilità di usare il fragment <></>
- NON + class MA CLASSNAME className (camel syntax)
- style={{ color: red }} perché la griglia singola serve per richiamare le variabili.
- volendo passare + props puoi usare lo spread operator

cosa vuol dire RENDER/RENDERING?

Significa trasformare un componente React in elementi visibili sullo schermo.

SPREAD OPERATOR → x unire 2 array. Usando... i valori vengono estratti dagli array originali e aggiunti come elementi separati nel nuovo array.

Si può usare anche con gli oggetti

ARRAY:

```
const hobby1 = ['Nuoto', 'Tennis'];
const hobby2 = ['Gdismo'];
const unione = [...hobby1, ...hobby2];
output: ['Nuoto', 'Tennis', 'Gdismo']
```

OGGETTI: const utente = { nome: 'Mar', eta: 30 };
const aggiunta = { isAdmin: true };
const finale = { ...utente, ...aggiunta }
output: { nome: 'Mar', eta: 30, ~~isAdmin~~ isAdmin: true };

CAPACITÀ DI JAVASCRIPT DI PASSARE FUNZIONI COME ARGOMENTI AD ALTRE FUNZIONI.

```
esempio ottieniUtente(id, callback) {
  setTimeout(() => {
    callback({ id: id, nome: 'Mario' });
  }, 1000);
}
```

DEFINIRE FUNZIONI ALL'INTERNO DI ALTRE FUNZIONI → la funzione definita ~~sia~~ dentro l'altra funzione ~~può~~ essere richiamata fuori dall'altra funzione. È disponibile solo corre variabile dentro lo scope della funzione principale

OPERATORE TERNARIO

Condizione ? valoreSeVero : ValoreSeFalso
In React si usa per rendere condizionale il rendering.
cioè x mostrare qualcosa solo se la condizione è vera

IN REACT → RENDERING DI USTE

Se proviamo a fare un render dell'oggetto all'interno dell'array

```
const persone = [ {  
    nome: 'Ric',  
    cogn: 'Mac',  
    eta: 25  
}, {  
    nome: 'Chiara',  
    cogn: 'Mac',  
    eta: 27  
}];
```

da errore! Dovremmo prima accedere ad ogni singola proprietà.

ci aiuta .MAP

↓ dentro il return nelle { persone.map(pers => {
 return <h1>{pers.cogn}</h1> }
 MA OGNI ELEMENTO DI UN
 ARRAY RITORNATO DINAMICAMENTE
 DEVE AVERE UN ATTRIBUTO KEY

OPPURE AGGIUNGERE id ALL'OGGETTO
E USARE QUELLO COME KEY

map [↓] ne crea uno lui
~~pers.map((pers, index)~~
persone.map((pers, index) =>
{ return <h1 key={index}>
 {pers.cogn}</h1> }

EVENTI

Sono per rispondere alle azioni degli utenti

<button onClick={saluta()}> Clicca qui </button> SBAGLIATO
<button onClick={saluta}> Clicca qui </button> GIUSTO

CALLBACK FUNCTION

Una callback è una funzione che il genitore passa al figlio
affinché il figlio possa avvisare il genitore quando succede qualcosa to

```
es. function App () {  
    const stampaNome = (nome) => {  
        alert(nome)  
    }  
    return (  
        <Componente1 onStampa={stampaNome}></Componente1>  
    )  
}
```

onStampa == stampaNome

ALTRÒ FILE

```
const Componente1 = ({onStampa}) => {  
    return (  
        <div>  
            <h2>Componente 1 </h2>  
            <button onClick={()=> onStampa('Pippo')}>  
                Clicca qui </button>  
        </div>  
    )  
}
```

HOOKS

Quindi nell'esempio:

```
function App() {  
    let saluto = 'ciao';  
    const CambiaSaluto = () => {  
        saluto = 'arrivederci';  
        console.log(saluto);  
    }  
    return (  
        <div className='App'>  
            <h1>Componente Principale </h1>  
            <h2>{saluto}</h2>  
            <button onClick={CambiaSaluto}>Cambia </button>  
        </div>  
    );  
};
```

IMPORTANTE

React NON aggiorna il rendering quando cambia una variabile normale GUARDA SOLO LO STATE

Quando clicchi sul bottone in memoria Javascript il valore Cambia e in console compare 'ARRIVEDERCI' MA React NON riesegue App e NON ridisegna la UI

React ridisegna solo quando cambia lo stato o le props

USE STATE → Crea e Gestisce uno stato all'interno di un componente funzionale
2 funzionalità principali

- permette di avere un render della variabile
- tiene traccia dello stato che viene mutato

SINTASSI USE STATE

1. Importarlo da React

se si usa setValore ridisegna tutto

```
const [valore, setValore] = useState(valoreIniziale)
```

restituisce un ARRAY x consentire la destrutturaz. in ordine fisso

Lo stato è immutabile! Modificare direttamente un oggetto non cambia il riferimento e non provoca il re-render del componente

ES. SBAGLIATO

const [utente, setUtente] = useState({ nome: 'luca', eta: 25 });

```
function CambiaNome() {  
  utente.nome = 'Marco';  
  setUtente(utente);  
}
```

QMI NON CAMBIA NULLA
L'OGGETTO E' LO STESSO

ES. GIUSTO

```
function CambiaNome() {  
  setUtente({  
    ...utente,  
    nome: 'Marco'  
  });  
}
```

... utente copia l'oggetto
crea un nuovo oggetto
React fa il rendering

FUNCTIONAL RETURN

```
setCount(prev => prev + 1);
```

→ Passiamo una funzione al setter che
riceve come parametro il valore preced.
dello stato e restituisce il valore aggiornato.
GARANTENDO L'USO DEL NUOVO VALORE

USEEFFECT

→ Gestisce tutte le azioni che avvengono al di fuori del componente.

1. Viene chiamato dopo il render del componente
2. L'array delle dipendenze controlla quando l'effetto viene riesecuito

```
function App() {  
  const [count, setCount] = useState(0);  
  useEffect(() => {  
    console.log('Count cambiato:', count);  
  }, [count]);  
  return (  
    <button onClick={() => setCount(prev => prev + 1)}>  
      {count}  
    </button>  
  );  
}
```

Al click cambia count

React fa il re-render

UseEffect parte

Serve x fetch, timer

ESEMPIO DI FETCH E USEEFFECT E USESTATE

```
function App() {  
  const [post, setPost] = useState(null);  
  useEffect(() => {  
    fetch(url)  
      .then(res => res.json())  
      .then(data => setPost(data));  
  }, []);  
  return (  
    <div>  
      <h1> Post </h1> →  
    </div>  
  );  
}
```

{ Post ? (

<>

<h3> {post.title} </h3>
<p> {post.body} </p>

</>

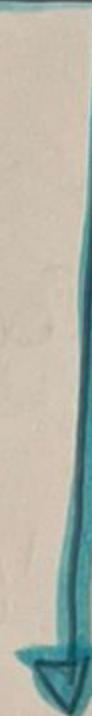
)::(

<p> Caricamento... </p>

)}

</div>

RENDER
CONDIZIONALE



Mostrare elementi diversi
nell'interfaccia in base alla condizione

CLEANUP FUNCTION

FUNZIONE restituita da useEffect che viene eseguita quando
l'effetto viene rimosso oppure prima della sua riesecuzione.

```
useEffect(()=> {  
    // fai qualcosa  
    RETURN ()=> {  
        // disfa quella cosa  
    };  
});
```

la cleanup function NON PARTE SUBITO
MA:

- PRIMA del prossimo effect
- OPPURE quando il COMPONENTE viene smontato

N.B.

Se metti un
AddEventListener
metti anche x for za
RemoveEventListener

ESEMPIO USE EFFECT FETCH ASYNC

```
const FetchComponent = () => {
  const [photos, setPhotos] = useState([]);
  const getData = async () => {
    const photos = await fetch(url).then(res => res.json());
    setPhotos(data.slice(0, 20)); → tiene solo i primi 20
  } catch (err) {
    console.error('Errore:', err);
  }
};
```

```
useEffect(() => {
  getData();
}, []); → fetch solo all'avvio, mettiamo getData
dentro lo useEffect perché ci serve una
funzione a parte x gestire async/await
return (
  <div>
    <h2> Photos </h2>
    <ul>
      {photos.map((p) =>
        <li key={p.id}> {p.title} </li>
      )}
    </ul>
  </div>
);
```

RICORDA:

Perché nel return c'è il DIV e NON IL
FRAGMENT?

React richiede che un componente ritorni
UN SOLO elemento padre; questo può essere
UN DIV o un FRAGMENT, scelto in base alle
esigenze del DOM.

Il FRAGMENT è un contenitore logico (lo usi quando non vuoi un div int.)
il DIV un contenitore reale (usato quando vuoi style o container ecc)

HOOKS → funzioni fornite da React che permettono ai componenti funzionali di utilizzare lo stato, il ciclo di vita e altre funzionalità di React senza dover usare le classi.

USE STATE → gestisce lo stato

USE EFFECT → esegue effetti collaterali

USE CONTEXT → condivide dati

USEREF → lavora con il DOM

RENDER CONDIZIONALE (detto prima cosa fa)

const url = '...'

const RenderCondizionale = () => {

const [isLoading, setLoading] = useState(true);

const [isError, setError] = useState(false);

const [users, setUsers] = useState([]);

const getData = async () => {

setError(false);

setLoading(true);

try {

const response = await fetch(url);
~~SET USER & ERROR~~

const dati = await response.json();

setUsers(dati);

} catch(error) {

console.log(error);

setError(true);

setLoading(false);

};

useEffect(() => {

setTimeout(() => {

getData();

, 3000); }, []);

→

→ if (!loading) {
RETURN <loading>
} </loading>;
if (isError) {
RETURN <error>
} </error>;

OPERATORE TERNARIO → scocciatoia di if che a sua differenza ritorna SEMPRE UN valore

non ha da verificare? 'sono vero': 'sono falso'

SHORT CIRCUIT EVALUATION → ritorna un valore in base allo stato di una costante o di una variabile che vogliamo valutare

|| → const esempio = parola || 'paperino' → se parola ha un valore allora esempio vale quello sennò paperino

&& → const esempio = ~~parola~~ && 'paperino' → se parola ha un valore allora esempio assume 'paperino'

FORM

↓
Gestione eventi
 onClick
 onSubmit
e.preventDefault

NB

class → className
for → htmlFor

USEREF → hook che permette di conservare un valore o un riferimento al DOM senza causare un re-render del componente.

const myRef = useRef(valore iniziale);

const contatore = useRef(0);

const aumenta = () => {

 contatore.current += 1;

 console.log(contatore.current);

};

Il VALORE CAMBIA LA UI NO!
TOP PER TIMER INTERNI O VALORI TEMPORANEI

USECONTEXT

→ serve a condividere dati tra componenti
senza passarsi props a catena

senza Context

```
<App utente={utente}>
  <Header utente={utente}>
    <Menu utente={utente}>
      </Header>
    </App>
```

{ devi passare
utente anche dae
non serve!

come si crea lo useContext

```
import { createContext } from react;
export const UserContext = createContext();
<UserContext.Provider value={utente}>
  <App />
</UserContext.Provider>
```

REACT ROUTER

→ permette di associare URL diversi a componenti
React senza ricaricare la pagina

BrowserRouter Componente principale

```
import { Routes, Route } from 'react-router-dom';
<Routes>
  <Route path='/' element={<Home />} />
  <Route path='/about' element={<About />} />
</Routes>
```

Link serve a navigare senza ricaricare la pagina

```
<Link to='/about'>About</Link>
```