

# JAVASCRIPT - LE BASI

# CONST

Il valore di una costante non può essere modificato

```
const invariabile = 5;  
invariabile = 6; // NO!  
// TypeError: Assignment to constant variable.
```

# CONST

Se la costante è un oggetto o un array, le sue proprietà o i suoi elementi possono essere modificati!

```
const nomeCostante = {};  
// Oggetto  
  
nomeCostante.proprieta = 1; // OK:  
// ho modificato una proprietà dell'oggetto,  
// ma non l'oggetto in sè
```

# LET

Può cambiare nel tempo

```
let nomeLet = "valore";  
  
nomeLet = {  
    proprietà: "valore",  
    altra: 123  
};
```

Si può fare, ma è sconsigliato modificare il tipo di dato  
di una variabile!

# VAR

Sintassi valida ma non più in uso

```
var nomeVariabile = "valore";
```

al suo posto, utilizza const e let

# OGGETTI

```
const myObject = {  
    propriet1: "valore",  
    propriet2: "altro valore"  
};
```

Accedere al valore di una proprietà:

```
// sintassi equivalenti  
const propriet1 = myObject["propriet1"];  
  
const propriet2 = myObject.proprieta2;
```

# {DESTRUCTURING}

Estrae una o più proprietà di un oggetto e le assegna a delle variabili

```
const { inputPath, url, postId, tags, ...postContent } = apiPo
```

[Approfondisci su MDN](#)

# ...SPREAD

```
const oldObject = {  
  "key1": "first value",  
  "key2": "second value",  
  "key3": "third value"  
};  
  
const myObject = {  
  "key0": "zeroth value",  
  ...oldObject  
};  
  
// > myObject: Object { key0: "zeroth value", key1: "first val
```

Approfondisci su MDN

# FUNZIONE *OBJECT.ENTRIES()*

```
const utente = { nome: "Mario", cognome: "Rossi" };

const proprietauTente = Object.entries(utente);

/*
proprietauTente = [
  ["nome", "Mario"],
  ["cognome", "Rossi"]
]
*/
```

Genera un array di coppie chiave/valore

# ESEMPIO DI UTILIZZO DI *OBJECT.ENTRIES()*

```
const utente = { nome: "Mario", cognome: "Rossi" };

Object.entries(utente).forEach(([chiave, valore]) => {
  // esegue qualcosa su ciascuna coppia chiave/valore dell'oggetto
  console.log("- ", chiave, " -> ", valore);
}) ;

/*
- nome -> Mario
- cognome -> Rossi
*/
```

# ARRAY

```
const myArray = [  
    "stringa",  
    123,  
    true,  
    { "proprietàOggetto": "valore" },  
    ["array"]  
] ;
```

## Accedere al valore in una determinata posizione:

```
const primoElementoDellArray = myArray[0]; // "stringa"  
const secondoElementoDellArray = myArray[1]; // 123
```

# [DESTRUCTURING]

Estrae uno o più elementi di un array e li assegna a variabili

```
const myArray = ["stringa", 2, 3];

// destructuring
const [primo, secondo, terzo] = myArray;

// > primo === "stringa";
// > secondo === 2;
// > terzo === 3
```

(puoi nominare liberamente le variabili)

# [DESTRUCTURING]

```
const myArray = ["stringa", "secondo", 3];  
  
// non devi per forza estrarre tutti gli elementi  
const [primo, , terzo] = myArray;  
  
// > primo === "stringa"; terzo === 3
```

# ...SPREAD

```
const myArray = [ 4, 5, 6 ];
const myMergedArray = [ 1, 2, 3, ...myArray ];

// > Array(6) [ 1, 2, 3, 4, 5, 6 ]
```

# ...SPREAD

```
function concatStrings(...args) {  
  return args.join(' ');  
}  
  
concatStrings("Hello", "world")
```

# METODI DEGLI ARRAY

## forEach(callback)

```
myArray.forEach((el) => {  
  el = el + "stringa";  
});  
  
myArray === myArray;  
// vengono modificati gli elementi contenuti nell'array  
// non l'array in sè
```

Modifica ogni elemento dell'array, applicando la funzione di callback

# METODI DEGLI ARRAY

## map(callback)

```
const newArray = myArray.map((el) => {
  el = el + "stringa";
  return el;
} )

newArray !== myArray;
// il metodo map() genera un nuovo array
```

Ritorna un nuovo array, con gli elementi modificati  
dalla callback

# ALTRI METODI DEGLI ARRAY

- Sort: riordina gli elementi dell'array
- Find: trova un elemento
- Filter: elimina alcuni elementi
- Reduce: elabora tutti gli elementi in un unico risultato
- Join: unisce due o più array

# APPROFONDISCI

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops\\_and\\_iteration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration)

# FUNZIONI

```
const miaFunzione = function nomeFunzione() {  
    // ...  
}  
  
const miaArrowFunction = () => {  
    // ...  
}  
  
const funzioneConParametri = (param1, param2) => {  
    return param1 + " " + param2;  
}
```

# FIRST CLASS FUNCTIONS

Si può assegnare una funzione a una variabile

```
const foo = () => {
  console.log("foobar");
};
foo();
```

# FIRST CLASS FUNCTIONS

Si può passare una funzione come argomento di un'altra funzione

```
function eseguiOperazione(operazione, a, b) {  
    return operazione(a, b);  
}  
  
function somma(a, b) {  
    return a + b;  
}  
  
eseguiOperazione(somma, 5, 3); // Restituisce 8
```

# FIRST CLASS FUNCTIONS

Una funzione può ritornare una nuova funzione

```
function creaFunzioneGreeting(saluto) {  
  return function(nome) {  
    return `${saluto}, ${nome}!`;  
  };  
}  
  
const salutaInItaliano = creaFunzioneGreeting("Ciao");  
salutaInItaliano("Marco"); // Restituisce "Ciao, Marco!"
```

# FIRST CLASS FUNCTIONS

Le funzioni possono avere proprietà e metodi come gli altri oggetti

```
function test() {}  
test.descrizione = "Questa è una proprietà della funzione 'tes
```

# APPROFONDISCI

[https://developer.mozilla.org/en-US/docs/Glossary/First-class\\_Function](https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function)

# CURRYING

Funzioni che ritornano altre funzioni, con una sintassi fluida

```
const chiama = (nome) => (functionCognome) => {
  return nome + " " + cognome;
}

// posso chiamarla così:
chiama("Mario")("Bianchi");

// oppure in più step,
// assegnando a una variabile la funzione generata:
const chiamaPaolo = chiama("Paolo");
chiamaPaolo("Rossi");
```

# APPROFONDISCI

<https://it.javascript.info/currying-partials>

# FUNZIONI ASINCRONE

```
1 function saluta() {
2     setTimeout(() => {
3         console.log("Ciao");
4     }, 1000);
5 }
6
7 const chiama = (nome) => {
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
```

```
6
7 const chiama = (nome) => {
8   setTimeout(() => {
9     console.log("Ehi, " + nome + "!");
10  }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
```

```
9     console.log( Ehi, + nome + ! );
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
21 Ehi, Mario!
22 Ciao
23 */
```

```
7 const chiama = (nome) => {
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
21 Ehi, Mario!
```

```
8  setTimeout(() => {
9    console.log("Ehi, " + nome + "!");
10   }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
21 Ehi, Mario!
22 Ciao
```

```
9      console.log("Ehi, " + nome + "!");
10     }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
21 Ehi, Mario!
22 Ciao
23 */
```

```
9     console.log( Ehi, + nome + ! );
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
21 Ehi, Mario!
22 Ciao
23 */
```

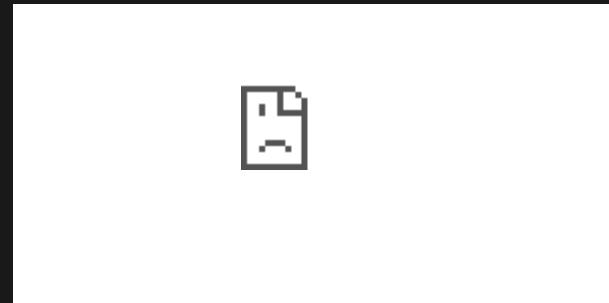
```
2  setTimeout(() => {
3      console.log("Ciao");
4  }, 1000);
5 }
6
7 const chiama = (nome) => {
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
```

```
9     console.log(`Ehi, ${name}!`);  
10    }, 0); // timeout = zero  
11 }  
12  
13 console.log("Inizio");  
14 saluta();  
15 chiama("Mario");  
16 console.log("Fine");  
17  
18 /* Output:  
19 Inizio  
20 Fine  
21 Ehi, Mario!  
22 Ciao  
23 */
```

```
1 function saluta() {
2     setTimeout(() => {
3         console.log("Ciao");
4     }, 1000);
5 }
6
7 const chiama = (nome) => {
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
```

```
9     console.log( Ehi, + nome + ! );
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 saluta();
15 chiama("Mario");
16 console.log("Fine");
17
18 /* Output:
19 Inizio
20 Fine
21 Ehi, Mario!
22 Ciao
23 */
```

# APPROFONDISCI



JavaScript Event Loop

COME POSSO CONTROLLARE  
L'ORDINE DI ESECUZIONE DELLE  
FUNZIONI ASINCRONE?

# CALLBACK

```
1 function saluta(cb) {  
2     setTimeout(() => {  
3         console.log("Ciao");  
4         cb(); // <- esegue la callback  
5     }, 1000);  
6 }  
7  
8 const chiama = (nome) => {  
9     setTimeout(() => {  
10        console.log("Ehi, " + nome + "!");  
11    }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15
```

# CALLBACK

```
5      } , 1000) ;
6  }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + " !");
11   } , 0) ;
12 }
13
14 saluta(() => chiama("Paolo")) ;
15
16 /* Output:
17 Ciao
18 Ehi, Paolo!
19 */
```

# CALLBACK

```
1 function saluta(cb) {  
2     setTimeout(() => {  
3         console.log("Ciao");  
4         cb(); // <- esegue la callback  
5     }, 1000);  
6 }  
7  
8 const chiama = (nome) => {  
9     setTimeout(() => {  
10        console.log("Ehi, " + nome + "!");  
11    }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15
```

# CALLBACK

```
5      } , 1000) ;
6  }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + " !");
11   } , 0) ;
12 }
13
14 saluta(() => chiama("Paolo")) ;
15
16 /* Output:
17 Ciao
18 Ehi, Paolo!
19 */
```

# CALLBACK

```
1 function saluta(cb) {  
2     setTimeout(() => {  
3         console.log("Ciao");  
4         cb(); // <- esegue la callback  
5     }, 1000);  
6 }  
7  
8 const chiama = (nome) => {  
9     setTimeout(() => {  
10        console.log("Ehi, " + nome + "!");  
11    }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15
```

# CALLBACK

```
3     console.log("Ciao");
4     cb(); // <- esegue la callback
5   }, 1000);
6 }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + "!");
11   }, 0);
12 }
13
14 saluta() => chiama("Paolo"));
15
16 /* Output:
17 Ciao
```

# CALLBACK

```
5      } , 1000) ;
6  }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + " !");
11   } , 0) ;
12 }
13
14 saluta(() => chiama("Paolo")) ;
15
16 /* Output:
17 Ciao
18 Ehi, Paolo!
19 */
```

# ATTENZIONE

La funzione `chiama()` si aspetta un parametro.

Poiché la funzione `saluta()` non passa nessun  
parametro alla callback,

`chiama("Paolo")` viene eseguita in una callback che  
non ha parametri!

```
saluta( function cb(/* nessun parametro */) { chiama("Paolo") }
```

# PROMISE

- è un oggetto che rappresenta un'operazione asincrona
- ha 3 metodi:
  - *then(fn)* -> esegue *fn()* come callback in caso di success
  - *catch(fn)* -> esegue *fn()* in caso di errore
  - *finally(fn)* -> esegue *fn()* in entrambi i casi

# PROMISE: VANTAGGI

- permette di concatenare molte funzioni asincrone una dopo l'altra, invece che una dentro l'altra, come avveniva con le callback
- permette di gestire gli errori in modo standardizzato

# PROMISE: COME SI COSTRUISCE

```
1 function saluta() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             console.log("Ciao");  
5             resolve();  
6         }, 1000);  
7     } );  
8 }  
9  
10 const chiama = (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + "!");  
14             resolve();  
15         } );  
16     } );  
17 }
```

# PROMISE: COME SI COSTRUISCE

```
1 function saluta() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             console.log("Ciao");  
5             resolve();  
6         }, 1000);  
7     } );  
8 }  
9  
10 const chiama = (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + "!");  
14             resolve();  
15         } );  
16     } );  
17 }
```

# PROMISE: COME SI COSTRUISCE

```
1 function saluta() {
2     return new Promise((resolve) => {
3         setTimeout(() => {
4             console.log("Ciao");
5             resolve();
6         }, 1000);
7     });
8 }
9
10 const chiama = (nome) => {
11     return new Promise((resolve) => {
12         setTimeout(() => {
13             console.log("Ehi, " + nome + " !");
14             resolve();
15         }, 0);
```

# PROMISE: COME SI COSTRUISCE

```
1 function saluta() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             console.log("Ciao");  
5             resolve();  
6         }, 1000);  
7     } );  
8 }  
9  
10 const chiama = (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + " !");  
14             resolve();  
15         } );  
16     } );  
17 }
```

# PROMISE: COME SI COSTRUISCE

```
5      setTimeout(() => {
4          console.log("Ciao");
5          resolve();
6      }, 1000);
7  });
8 }
9
10 const chiama = (nome) => {
11     return new Promise((resolve) => {
12         setTimeout(() => {
13             console.log("Ehi, " + nome + " !");
14             resolve();
15         }, 0);
16     });
17 }
```

# PROMISE: COME SI COSTRUISCE

```
5      setTimeout(() => {
4          console.log("Ciao");
5          resolve();
6      }, 1000);
7  });
8 }
9
10 const chiama = (nome) => {
11     return new Promise((resolve) => {
12         setTimeout(() => {
13             console.log("Ehi, " + nome + " !");
14             resolve();
15         }, 0);
16     });
17 }
```

# PROMISE: COME SI USA

```
saluta()
  .then(() => chiama("Paolo"))
  .then(saluta)
  .then(() => { console.log("Fine") }) ;

/* Output:
Ciao
Ehi, Paolo!
Ciao
Fine
*/
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# PROMISE: COME SI GESTISCONO GLI ERRORI

```
1 saluta()
2   .then(() => chiama("Paolo"))
3   .then(() => { throw new Error("Ops, qualcosa è andato storto") })
4   .then(saluta)
5   .then(() => { console.log("Fine") })
6   .catch((err) => console.error(err.message));
7
8 /* Output:
9 Ciao
10 Ehi, Paolo!
11 Ops, qualcosa è andato storto
12 */
```

# APPROFONDISCI

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Pro](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Pro)

# ASYNC/AWAIT

E' una sintassi ancora più comoda per scrivere ed eseguire Promise

# ASYNC/AWAIT

```
1  async function saluta() {  
2      return new Promise((resolve) => {  
3          setTimeout(() => {  
4              console.log("Ciao");  
5              resolve();  
6          }, 1000);  
7      });  
8  }  
9  
10 const chiama = async (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + "!");  
14             resolve();  
15         }, 0);  
16     });  
17 }
```

# ASYNC/AWAIT

```
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
30 Ciao
31 Ehi, Paolo!
32 Fine
33 */
```

# ASYNC/AWAIT

```
12     setTimeout(() => {
13         console.log("Ehi, " + nome + " !");
14         resolve();
15     }, 0);
16 }
17 }
18
19 const main = async () => {
20     console.log("Inizio");
21     await saluta();
22     await chiama("Paolo");
23     console.log("Fine");
24 }
25
26 main();
```

# ASYNC/AWAIT

```
13     console.log("Ehi, " + nome + " !") ;
14     resolve() ;
15 }, 0) ;
16 }) ;
17 }
18
19 const main = async () => {
20   console.log("Inizio") ;
21   await saluta() ;
22   await chiama("Paolo") ;
23   console.log("Fine") ;
24 }
25
26 main() ;
27
```

# ASYNC/AWAIT

```
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
30 Ciao
31 Ehi, Paolo!
32 Fine
33 */
```

# ASYNC/AWAIT

```
14         resolve() ;
15     }, 0) ;
16   ) ) ;
17 }
18
19 const main = async () => {
20   console.log("Inizio") ;
21   await saluta();
22   await chiama("Paolo") ;
23   console.log("Fine") ;
24 }
25
26 main() ;
27
28 /* Output:
```

# ASYNC/AWAIT

```
1 async function saluta() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             console.log("Ciao");  
5             resolve();  
6         }, 1000);  
7     });  
8 }  
9  
10 const chiama = async (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + " !");  
14             resolve();  
15         }, 0);  
16     });  
17 }
```

# ASYNC/AWAIT

```
1 async function saluta() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             console.log("Ciao");  
5             resolve();  
6         }, 1000);  
7     });  
8 }  
9  
10 const chiama = async (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + " !");  
14             resolve();  
15         }, 0);  
16     });  
17 }
```

# ASYNC/AWAIT

```
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
30 Ciao
31 Ehi, Paolo!
32 Fine
33 */
```

# ASYNC/AWAIT

```
1 async function saluta() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             console.log("Ciao");  
5             resolve();  
6         }, 1000);  
7     });  
8 }  
9  
10 const chiama = async (nome) => {  
11     return new Promise((resolve) => {  
12         setTimeout(() => {  
13             console.log("Ehi, " + nome + " !");  
14             resolve();  
15         }, 0);  
16     });  
17 }
```

# ASYNC/AWAIT

```
15      }, 0);
16  });
17 }
18
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
```

# ASYNC/AWAIT

```
~      , , + ~ ~ ~ , ,  
7      } ) ;  
8  }  
9  
10 const chiama = async (nome) => {  
11   return new Promise((resolve) => {  
12     setTimeout(() => {  
13       console.log("Ehi, " + nome + " !") ;  
14       resolve() ;  
15     }, 0) ;  
16   } ) ;  
17 }  
18  
19 const main = async () => {  
20   console.log("Inizio") ;  
21   await chiama("John") ;  
22 }
```

# ASYNC/AWAIT

```
6      }, 1000);
7  );
8 }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + " !");
14       resolve();
15     }, 0);
16   );
17 }
18
19 const main = async () => {
20   console.log("Inizio");
```

# ASYNC/AWAIT

```
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
30 Ciao
31 Ehi, Paolo!
32 Fine
33 */
```

# ASYNC/AWAIT

```
7      } ) ;
8  }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + " !");
14       resolve();
15     }, 0);
16   });
17 }
18
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
```

# ASYNC/AWAIT

```
16      } ) ;
17  }
18
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
30 Ciao
```

# ASYNC/AWAIT

```
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22   await chiama("Paolo");
23   console.log("Fine");
24 }
25
26 main();
27
28 /* Output:
29 Inizio
30 Ciao
31 Ehi, Paolo!
32 Fine
33 */
```

# MAPPING CON FUNZIONI ASINCRONE

# Problema: abbiamo un array da mappare su una funzione asincrona

```
const result = [1, 2, 3, 4].map(async (id) => {  
  return await getById(id);  
})
```

Questa funzione ritorna:

```
[Promise, Promise, Promise, Promise]
```

mentre ci aspettavamo:

```
["valuea", "value2", "value3", "value4"]
```

## Soluzione: Promise.all()

```
const result = Promise.all(  
  [1, 2, 3, 4].map(async (id) => {  
    return await getById(id);  
  })  
) ;
```

Approfondisci su MDN

