CALLBACK

Funzione che viene passata in un'altra funzione come argomento.
A quel punto viene poi invocata dalla funzione stessa per fare operazioni.

Può avere comportamento asincrono o sincrono.

```
function logQuote(quote) {
  console.log(quote);
}

function createQuote(quote,
callback) {
  const myQuote = `Like I always
say, '${quote}'`;
  callback(myQuote);
}

createQuote("WebApp I rocks!",
logQuote);
```

Callback sincrone: sono eseguite interamente quando vengono richiamate (programma aspetta fine esecuzione)

```
let numbers = [4, 2, 5, 1, 3];

numbers.sort(function(a, b) {
  return a - b;
});

console.log(numbers);
```

```
let numbers = [4, 2, 5, 1, 3];

numbers.sort((a, b) => a - b);

console.log(numbers);
```

Filter → metodo funzionale che accetta un criterio e filtra secondo quello

```
const market = [
  { name: 'GOOG', var: -3.2 },
  { name: 'AMZN', var:  2.2 },
  { name: 'MSFT', var: -1.8 }
];

const bad = market.filter(stock => stock.var < 0);
// [ { name: 'GOOG', var: -3.2 }, { name: 'MSFT', var: -1.8 } ]

const good = market.filter(stock => stock.var > 0);
// [ { name: 'AMZN', var: 2.2 } ]
```

Programmazione funzionale

Sviluppatori costruiscono e strutturano il codice principalmente usando funzioni.
Stile dichiarativo → maggior visibilità programma

```
new_array =
  array.filter ( filter_function ) ;
```

```
new_array = [] ;
for (const el of list)
      if ( filter_function(el) )
            new_array.push(el) ;
```

- Funzioni sono cittadini di prima classe in quanto oggetti
  - Posso farci tante cose
- Le funzioni sono di ordine alto
  - Può operare su altre funzioni
  - Prende come parametro delle funzioni
  - Ritorna una funzione
- Funzioni si possono comporre
- Possono essere chiamate in catena
  - Risultato di una è l'input dell'altra

NOTA: si evitano modifiche in place
- Se devo modificare un array, non modifco l'array ma una sua copia

METODI FUNZIONALI:

- **forEach**(f): processa ogni elemento con callback f
  - chiama la callback una volta per ogni elemento
  - currentValue: valore corrente
  - index: indice del valore corrente
  - array: copia dell'array corrente
  - ritorna un undefined, non contatenabile
    - non posso fare .altro
  - non si può bloccare ad un certo punto del for

```
const letters = [..."Hello world"] ;
let uppercase = "" ;
letters.forEach(letter => {
  uppercase += letter.toUpperCase();
});
console.log(uppercase); // HELLO WORLD
```

- every(f), some(f): verofoca se tutti/alcuni elementi nell'array soddisfano la callback booleana f
  - **every**(f)
    - true: se tutti gli elementi superano il test
    - false: se c'è almeno uno che non lo supera
  - **some**(f):
    - true: se c'è almeno uno che supera il test
    - false: se tutti non superano il test

```
let a = [1, 2, 3, 4, 5];
a.every(x => x < 10); // => true: all values are < 10
a.every(x => x % 2 === 0); // false: not all even values
```

```
let a = [1, 2, 3, 4, 5];
a.some(x => x%2===0); // => true; a has some even numbers
a.some(isNaN);  --> false
```

- **map**(f) → costruire un nuovo array con i valori tornati dalla callback
  - callback deve restituire qualcosa
- **filter**(f) → crea un nuovo array con tutti gli elementi che passano i test implementati dalla funzione
- **reduce**: combina gli elementi di un array, usando la callback specificata
  - **restituisce un valore singolo**
    - si calcola progressivamente un risultato
  - 1 parametro obbligatorio→ callback
    - Dentro la callback ha 2 parametri
      - Accumulatore → risultato accumulato fino a quel momento specifico
  - 1 parametro opzionale → da quale ID partire per fare combinazione elementi
    - Se non passato, usa il primo valore come iniziale e iterazione parte da secondo

```
const a = [1, 2, 3];

const b = a.map(x => x*x);

console.log(b); // [1, 4, 9]
```

```
const letters = [..."Hello world"];

const uppercase = letters.map(letter
=> letter.toUpperCase());

console.log(uppercase.join(''));
```

```
const a = [5, 4, 3, 2, 1];

a.filter(x => x < 3); // generates [2, 1], values less than 3

a.filter((element, index) => index%2 == 0); // [5, 3, 1]
```

```
const a = [5, 4, 3, 2, 1];

a.reduce( (accumulator, currentValue) =>
accumulator + currentValue,     0);
// 15; the sum of the values

a.reduce((acc, val) => acc*val,  1);
// 120; the product of the values

a.reduce((acc, val) => (acc > val) ? acc
: val);
// 5; the largest of the values
```

```
import dayjs from 'dayjs';

function Answer(text, username, date, score=0) {
    this.text = text;
    this.username=username;
    this.score=score;
    this.date= dayjs(date);
    this.toString = () => {
        return `${this.username} replied '${this.text}' on ${this.date.format('YYYY-MM-DD')} con score ${this.score}`;
    }
}

function Question(text, username, date){
    this.text = text;
    this.username=username;
    this.date=dayjs(date);
    this.answers=[];

    this.add = (answer) => {
        this.answers.push(answer);
    }

    this.find = (username) => {
        const foundAnswers = [];
        for(const ans of this.answers){
            if(ans.username===username)
                foundAnswers.push(ans);
        }
        return foundAnswers;
    }
    this.afterDate = () => { //TO DO
    }

    this.listByDate = () => { //TO DO
    }
    this.listByScore = () => { //TO DO
    }
}

const question= new Question('Is JS better then python?', 'Peppe Arbo', '2024-02-27');
const firstAnswer= new Answer('Yes', 'Pluto Mannella', '2024-02-18', -10);
const secondAnswer= new Answer('Not in a million year', 'Pino Van Rossen', '2024-03-01', 5);
const thirdAnswer= new Answer('Yes', 'Albert Einstein', '2024-03-01', 1);
const fourthAnswer= new Answer('Yes', 'Pluto Mannella', '2024-03-10');
question.add(firstAnswer);
question.add(secondAnswer);
question.add(thirdAnswer);
question.add(fourthAnswer);
const answerByPluto = question.find('Pluto Mannella');
console.log(question);
console.log("Risposte di Pluto" + answerByPluto);
```

con metodi funzionali:

```
import dayjs from 'dayjs';

function Answer(text, username, date, score=0) {
    this.text = text;
    this.username=username;
    this.score=score;
    this.date= dayjs(date);
    this.toString = () => {
        return `${this.username} replied '${this.text}' on ${this.date.format('YYYY-MM-
DD')} con score ${this.score}`;
    }
}

function Question(text, username, date){
    this.text = text;
    this.username=username;
    this.date=dayjs(date);
    this.answers=[];

    this.add = (answer) => {
        this.answers.push(answer);
    }
    this.find = (username) => {
        return this.answers.filter(answer => answer.username === username);
    }
    this.afterDate = (date) => {
        return this.answers.filter(ans => ans.date.isAfter(dayjs(date)));
    }
    this.listByDate = () => {
        return [...this.answers].sort((a,b) => (a.date.isAfter(b.date) ? 1 : -1));
//quadre aggiunte per fare spread ... in modo tale da non avere un ordinamento in place
    }
    this.listByScore = () => {
        return [...this.answers].sort((a,b) => (b.score-a.score)); //ordine decrescente
    }
}

const question= new Question('Is JS better then python?', 'Peppe Arbo', '2024-02-27');
const firstAnswer= new Answer('Yes', 'Pluto Mannella', '2024-02-18', -10);
const secondAnswer= new Answer('Not in a million year','Pino Van Rossen', '2024-03-01',5);
const thirdAnswer= new Answer('Yes', 'Albert Einstein', '2024-03-01', 1);
const fourthAnswer= new Answer('Yes', 'Pluto Mannella', '2024-03-10');
question.add(firstAnswer);
question.add(secondAnswer);
question.add(thirdAnswer);
question.add(fourthAnswer);
const answerByPluto = question.find('Pluto Mannella');
console.log(question);
console.log("\nRisposte di Pluto" + answerByPluto);
console.log(question.listByDate());
console.log(question.listByScore());
console.log(question.afterDate('2024-02-29'));
```

Callback asincrone: programma non aspetta la fine dell’esecuzione