Array

ARRAY

Un pattern tipico è quello di dover “filtrare” un array, ovvero produrre un nuovo array in cui si decide di mantenere ogni elemento a seconda di algoritmo.

Ipotizziamo di avere un array con tutti gli utenti della nostra applicazione:

```
const users = UsersStorage.getUsers();
```

Vogliamo avere un nuovo array in cui siano presenti solo gli utenti minorenni. Il modo “tradizionale” dei linguaggi imperativi (C, C++, Java…) è:

```
const underAgeUsers = [];

for (let user of users) {
  if (isUnderAge(user)) {
      // Mutazione!!
      underAgeUsers.push(user);
  }
}

function isUnderAge(user) {
  return user.age < 18;
}
```

ARRAY

- Però, se avessimo la seguente utility:

```
function filter(original, iteratee) {
const result = [];

for (let i = 0; i < original.length; ++i) {
  if (iteratee(original[i], i, original)) {
    result.push(original[i]);
  }
}

return result;
}
```

Allora potremmo fare:

```
const users = UsersStorage.getUsers();
const underAgeUsers = filter(users, user => isUnderAge(user));
```

1. Più conciso
2. Più chiaro (stiamo "dicendo" che vogliamo **filtrare** l'array)
3. Nessuna mutazione necessaria! (nel **nostro** codice)

ARRAY

Addirittura `user => isUnderAge(user)` non è necessaria:

```
const users = UsersStorage.getUsers();
const underAgeUsers = filter(users, isUnderAge);
```

ARRAY

Questa “utility” è disponibile come metodo degli array:

```
const users = UsersStorage.getUsers();
const underAgeUsers = users.filter(isUnderAge);
```

ARRAY

Un altro pattern molto utilizzato è quello di voler "mappare" gli elementi di un array. Ovvero ottenere un array della stessa dimesione in cui ogni elemento che ha un valore calcolato a partire al relativo della lista di partenza.

Supponiamo per esempio di voler ottenere di ogni utente il nome di un genitore.

Il modo "tradizionale" che potremmo implementare in un liguaggio imperativo è il seguente.

```
const parents = [];

for (let user of users) {
  parents.push(getParentName(user));
}

function getParentName(user) {
  return user.parent.name;
}
```

ARRAY

- Se avessimo la seguente utility:

```
function map(original, iteratee) {
const result = [];
for (let i = 0; i < original.length; ++i) {
  result.push(iteratee(original[i], i, original));
}
return result;
}
```

Allora potremmo fare:

```
const users = UsersStorage.getUsers();
const parents = map(users, user => getParentName(user));
```

1. Più conciso
2. Più chiaro (stiamo "dicendo" che vogliamo **mappare** l'array)
3. Nessuna mutazione necessaria! (nel **nostro** codice)

ARRAY

Addirittura `user => getParentName(user)` non è necessaria:

```
const users = UsersStorage.getUsers();
const parents = map(users, getParentName);
```

ARRAY

Questa “utility” è disponibile come metodo degli array:

```
const users = UsersStorage.getUsers();
const parents = users.map(getParentName);
```

ARRAY

Supponiamo di volere i genitori degli utenti miorenni

Siccome `map` e `filter` ritornano l'array è possibile effettuare le operazioni a cascata.

```
const parents = users
  .filter(isUnderAge)
  .map(getParentName)
  .map(name => users.find(user => user.name === name));
```