Początkujący programiści bardzo szybko spotykają się z operacjami na tablicach. Operacje te są często nieodłączną częścią kodu naszej aplikacji. Aby zrozumieć je lepiej (i ułatwić sobie trochę życie) warto zapoznać się z wbudowanymi metodami jakie oferuje JavaScript.
W tym artykule skupię się na operacjach dodawania, usuwania oraz sposobów iteracji tablicy oraz wytłumaczę jak działają metody mutujące i nie mutujące tablicę.

Zacznijmy od tego czym różnią się metody mutujące od tych nie mutujących. Otóż te pierwsze zmieniają tablicę, na której działają. Po zastosowaniu metody mutującej i ponownym wywołaniu referencji danej tablicy, zauważymy że jej struktura została zmieniona.

Z kolei nie mutujące metody, jak łatwo się domyślić, nie zmieniają struktury tablicy. Zwracając nową tablicę. W związku z tym, jeśli chcemy mieć dostęp do nowej tablicy musimy ją przypisać do zmiennej.

DODAWANIE WARTOŚCI DO TABLICY

Metody mutujące:

- `Array.prototype.push()` dodaje wartość na koniec tablicy.
  Przykład:
  ```
  let arr = [1,2,3,4];
  arr.push(5);
  console.log(arr);    // [1,2,3,4,5]
  ```

- `array unshift()` dodaje wartość na początek tablicy.
  Przykład:
  ```
  arr.unshift(5);
  console.log(arr); //[5,1,2,3,4,5]
  ```

  Zauważ, że operujemy ciągle na tej samej tablicy (arr), której długość została zmieniona przez poprzednią metodę `.push()` .

Metody nie mutujące:

- `Array.prototype.concat()` zwraca tablicę, która składa się z tablicy, na której została wywołana metoda, połączonej z innymi tablicami lub wartościami
  Przykład:
  ```
  const arr2 = [10,20,30];
  const newArr = arr.concat(arr2);
  console.log(newArr)   //[5,1,2,3,4,5,20,30,40]
  ```

  Jak widzisz na przykładzie, metoda .concat() dodała do tablicy, na której została wywołana, wartości z tablicy arr2. Warto zapamiętać, że w przypadku podania referencji do obiektu, metoda skopiuję tą referencję. Oznacza to, że każda zmiana w obiekcie będzie widoczna w nowej tablicy.

- operator spread (...) może być wykorzystywany do dodawania wartości, spójrz na poniższy przykład:

```
const spreadArr = [...newArr, 'burek', 'reksio'];
console.log(spreadArr);
//[5,1,2,3,4,5,20,30,40, 'burek', 'reksio']
```

USUWANIE WARTOŚCI Z TABLICY

Metody mutujące:

- Array.proptotype.pop() usuwa ostatnią wartość z tablicy i zwraca ją. W przypadku pustej tablicy zwraca `undefined`.
  Przykład:

```
let arr2 = [10,20,30];
arr2.pop()        //30
console.log(arr2)        //[10,20]
```

  Istnieje możliwość aby użyć metody pop() na obiektach tablicopodonych. W tym celu należy użyć metody call() lub apply(), które tworząc kontekst, mogą umożliwić np. użycie na obiekcie tablicopodobnym metod przeznaczonych dla tablic.
  Przykład:

```
function justFunc(arg1, arg2){
        const popValue = Array.prototype.pop.call(arguments);
        return popValue
}
justFunc('Tymon', 'Pumba'); //Pumba
```

  Funkcja przyjęła dwa argumenty 'Tymon' i 'Pumba', następnie do zadeklarowanej zmiennej popValue przypisany zostaje wynik operacji z użyciem metod pop() i call(). Czyli -> Wywołujemy metodę obiektu Array, pop(). Zamiast nawiasów, zaraz po metodzie pop wywoływana jest metoda call, która tworzy kontekst arguments dla metody pop. Uruchamiamy metodę typową dla tablic dla obiektu tablicopodobnego. Bez użycia metody call() zwrócony zostałby błąd.

```
arguments.pop()  //Uncaught TypeError: arguments.pop is not a function
```

  Dlatego przekazujemy interpreterowi aby wywołał metodę pop() w kontekście obiektu arguments.

- Array.prototype.shift() działa podobnie jak metoda pop() z tą różnicą, że usuwa pierwszy element z tablicy.

W przypadku metod mutujących tablice warto zwrócić uwagę na metodę splice(), która może zarówno zmieniać, usuwać jak i dodawać elementy. Metoda zwraca tablicę z usuniętymi elementami.

Jako pierwszy argument, metoda przyjmuje indeks od którego rozpoczynamy modyfikację tablicy, następnie liczbę elementów które mają zostać usunięte a jako trzeci argument, przyjmowane są elementy, które mają zostać dodane do tablicy.
Np.

```
let simpleArray = [1,2,3,4,5];
simpleArray.splice(1,2,10)
```

Z tablicy simpleArray, zaczynając od elementu z indeksem 1, usuwamy dwa elementy i w miejscu elementu z indeksem 1 umieszczony zostaje element integer o wartości 10.

Metody niemutujące:
`Array.prototype.filter()` - metoda ta ta tworzy nową tablice z elementami, które przeszły test w postaci funkcji.
Funkcja przejmuje cztery argumenty -

- **callback** - funkcja, która określa warunki jakim zostanie poddany każdy element tablicy. W przypadku przejścia testu, tzn. jeśli funkcja zwróci wartość true, element jest dodawany do nowej tablicy zwracanej przez metodę filter().
  Funkcja callback przyjmuje trzy argumenty:
  - **element** - element tablicy do przetworzenia.
  - **index** - wartość indeksu obecnie testowanego elementu w tablicy.
  - **array** - tablica, na której wywoływana jest metoda filter()

- **this** - obiekt, na który będzie wskazywał this podczas wywoływania funkcji callback.

Poniższy zwraca nową tablice z elementami string (z tablicy, na której została wywołana metoda), które nie zawierają “u”.

```
[“Kuba”, “Andrzej”, “Janusz”].filter(element=>!element.includes("u"))  //[“Andrzej”]
```

*w funkcji strzałkowej w przypadku jednego parametru możemy zrezygnować z nawiasów. Podobnie w przypadku podania tylko jednej instrukcji w ciele funkcji, możemy zrezygnować z nawiasów klamrowych i wyrażenia *return. Jest ono wtedy domyślne.*

`Array.prototype.slice()` - wydobywa określony element z tablicy i zwraca go jako tablicę.
Funkcja przyjmuje dwa argumenty -

- **begin** - indeks , od którego zaczyna się wydobywanie.
- **end** - indeks przed którym kończy się wydobywanie.

Poniższy przykład wydobywa ostatni element z tablicy.

```
[“Kuba”, “Andrzej”, “Janusz”].slice(-1)  //[“Janusz”]
```

W kolejnym przykładzie wydobywamy tylko zwierzę, które nie żyje w Polsce tj. jeżozwierz.

```
[“jeleń”, “wydra”, “ryjówka”, “łoś”, “żubr”, “bóbr”, “jeżozwierz”, “lis”, “wilk”, “kuna”, “koń”].slice(6,7)  //[“jeżozwierz”]
```

Co jeśli chcielibyśmy pokazać tylko zwierzęta, które (zgodnie z klasykiem kinowym) żyją w Polsce? Moglibyśmy użyć np. metodyy filter()

```
["jeleń", "wydra", "ryjówka", "łoś", "żubr", "bóbr", "jeżozwierz", "lis", "wilk", "kuna", "koń"].filter(element=>element != "jeżozwierz")
```

Zamiana wartości tablicy - metody mutujące
`Array.prototype.splice()` - Zmienia wartość tablicy, dodając nowe elementy i opcjonalnie usuwając starsze. Zwraca tablicę usuniętych elementów.
Przyjmuje trzy argumenty:

- start - indeks, od którego zaczynamy modyfikację tablicy.
- deleteCount - liczba całkowita określająca liczbę starych elementów, które nalezy usunąć. W przypadku przekazania argumentu o wartości 0, należy w kolejnym argumencie podać elementy, które mają zostać dodane. Jeśli nie podamy parametru deleteCount, wszystkie elementy od elementu startowego zostaną usunięte.
- item - elementy, które mają zostać dodane do tablicy.

W poniższym przykładzie dodajemy element *String* jako trzeci (indeks 2) element tablicy.

```
let rosliny = ["korzeń", "gałąź", "liść", "nasiona"];
rosliny.splice(2,0,"sadzonka");
console.log(rosliny);  //[“korzeń”, “gałąź”, “sadzonka”, “liść”, “nasiona]
```

Tutaj z kolei podmieniamy “liść” na “szyszka”.

```
rosliny.splice(3,1,”szyszka”);
console.log(rosliny)  //[“korzeń”, “gałąź”, “sadzonka”, “szyszka”, “nasiona”]
```

Metody niemutujące
`Array.prototype.map()` - Zwraca tablicę zawierającą wyniki wywołania funkcji dla każdego elementu tablicy, na której metoda została wykonana. Callback jest wywoływany tylko na elementach posiadających wartości, włącznie z *undefined*.
Przyjmuje dwa argumenty:

- **callback** - funkcja tworząca element nowej tablicy.
    - **element** - element, na którym obecnie jest wykonywane działanie.
    - **index** - indeks elementu.
    - **array** - tablica, dla której została wywołana metoda map().

- **thisArg** - wartość jaką użyć jako *this* podczas wywołania callback.

Jeśli element tablicy zostanie dodany po wywołaniu metody map(), element nie jest odwiedzany. Jeśli wartość elementu tablicy zostanie zmieniona (po wywołaniu metody map), ich wartość będzie wartością równą wartości kiedy funkcja callback sprawdza właśnie ten element (w tym danym momencie). Elementy usunięte po wywołaniu metody map oraz przed tym jak trafi na nie callback, nie zostaną sprawdzone (przez callback).
Jest to podobne działanie jak w przypadku metody filter().

W poniższym przykładzie deklarujemy zmienną miasta, gdzie nazwa każdego miasta, zaczyna się małą literą. Następnie deklarujemy zmienną miastaPoprawione, gdzie wywołujemy metodę map() na tablicy miasta. Metoda ta przyjmuje funkcję, która dla każdego elementu tablicy miasta, pobiera

pierwszą literę elementu (przy użyciu metody chartAt), następnie konwertuje go do postaci dużej litery i łączy z pozostałą częścią łańcucha zwracaną przez metodę slice(), o której pisałem już wcześniej.

```
let miasta = ["wrocław", "warszawa", "poznań", "gdańsk", "kraków", "łódź"];
let miastaPoprawione = male.map(element=>element.charAt(0).toUpperCase()+element.slice(1))
console.log(miastaPoprawione);  //["Wrocław", "Warszawa", "Poznań", "Gdańsk", "Kraków", "Łódź"]
```

Istnieje wiele metod, które mogą pomóc nam w operacjach na tablicach. Nie wspomniałem tu m.in. o metodzie reduce(), która ma ogromne możliwości.
Mam nadzieję, że ten artykuł pozwoli Ci zrozumieć lepiej część metod operujących na tablicach oraz zapamiętać, które z nich zmieniają stan tablicy (mutują), a które tworzą nową tablicę (nie mutują).

Źródła/Bibliografia:
1. Dokumentacja MDN. https://developer.mozilla.org/pl/
2. “Tajniki języka JavaScript” - Kyle Simpson
3. “JavaScript: Array Methods” - Aniket Jha
https://vtechguys.medium.com/javascript-array-methods-mutating-vs-non-mutating-8606d9b78c77