Tablice (kolekcje) w JS

W wielu przypadkach przechowywanie wielu wartości w ramach kolekcji o jednej nazwie to coś, co znacznie ułatwia nam życie podczas programowania. Właśnie do tego służą w JS tablice (*ang. arrays*). Są one jednym z fundamentalnych struktur danych, jakie powinniśmy znać i rozumieć, przygotowując się do egzaminu INF.03.

Tablice pozwalają przechowywać w swoich “szufladach” wiele wartości oraz operować na nich w wygodny sposób. W porównaniu do definiowania bardzo wielu pojedynczych zmiennych, tablice pozwalają nam grupować dane logicznie powiązane – np. listę produktów, imion czy liczb.

Tworzenie tablicy

Najprostszy sposób na stworzenie tablicy w JS to użycie nawiasów kwadratowych

`[]`. W nawiasach tych możemy od razu umieścić elementy, oddzielone przecinkami.

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
```

W tym momencie stworzyliśmy tablicę o zbiorczej nazwie

`owoce`, zawierającą 3 elementy (tutaj łańcuchy). Każdy z tych elementów posiada w tablicy przypisany numer – tzw. *indeks*, który umożliwia dostęp do wartości w zadanej “szufladzie”. Pamiętajmy, że indeksowanie w JavaScript (jak zresztą w wielu językach programowania) rozpoczyna się od wartości `0`.

Omówmy jeszcze gwoli formalności inne metody definiowania tablic.

Konstruktor Array – mniej popularny sposób, ale możliwy:

```
let owoce = new Array("Jabłko", "Banan", "Gruszka");
```

Działa to tak samo jak z nawiasami kwadratowymi, jednak uwaga: jeśli przekażemy tylko jedną liczbę, JS potraktuje ją jako rozmiar tablicy, a nie element:

```
let pusta = new Array(3);
console.log(pusta);  // [ <3 empty items> ]
```

Definicja pustej tablicy + dodawanie elementów później:

```
let owoce = [];
owoce[0] = "Jabłko";
owoce[1] = "Banan";
owoce.push("Gruszka"); // tę metodę omówimy za chwilę w lekcji!
```

Tablica z ustaloną długością + metoda fill (wypełnij wartościami):

```
let liczby = new Array(5).fill(0);  // [0, 0, 0, 0, 0]
```

Oczywiście na egzaminie najlepiej i najczytelniej używać składni z nawiasami kwadratowymi, ale rzecz jasna warto było omówić inne możliwości.

Pora teraz pokazać odczyt zawartości poszczególnych szuflad – dokonujemy tego podając indeks (numer szuflady) w nawiasach kwadratowych:

```
console.log(owoce[0]); // "Jabłko"
console.log(owoce[1]); // "Banan"
console.log(owoce[2]); // "Gruszka"
```

Dodawanie oraz usuwanie elementów

Tablice w JS są dynamiczne – możemy do nich dodawać nowe elementy lub usuwać istniejące. JavaScript udostępnia nam gotowe metody do wykonywania tych operacji!

Dodawanie elementów do tablicy

Metoda

`push()` oznacza dodanie elementu na końcu tablicy:

```
owoce.push("Śliwka");
console.log(owoce);
```

A zatem stan tablicy po tej operacji to:

```
["Jabłko", "Banan", "Gruszka", "Śliwka"]
```

Z kolei metoda

`unshift()` – wstawia element na jej początku:

```
owoce.unshift("Wiśnia");
console.log(owoce);  // ["Wiśnia", "Jabłko", "Banan", "Gruszka", "Śliwka"]
```

Dlaczego taki jest wynik operacji? Ponieważ metoda

`unshift()` dodaje nowy element na początek tablicy, przesuwając pozostałe elementy o jeden indeks dalej. To oznacza, że: “Wiśnia” trafiła na indeks 0, a wszystkie pozostałe owoce przesunęły się o jeden indeks w prawo. Widzimy zatem, że dzięki tym metodom możemy stosunkowo łatwo zwiększać zawartość tablicy.

Usuwanie elementów z tablicy

Pora zatem teraz omówić operacje przeciwne – usuwające dane. Metoda

`pop()` usuwa ostatni element z tablicy (o największym dostępnym indeksie):

```
let owoce = ["Wiśnia", "Jabłko", "Banan", "Gruszka", "Śliwka"];
owoce.pop();
console.log(owoce);  // ["Wiśnia", "Jabłko", "Banan", "Gruszka"]
```

Metoda

`shift()` – usuwa pierwszy element (czyli w praktyce zerowy indeks):

```
let owoce = ["Wiśnia", "Jabłko", "Banan", "Gruszka"];
owoce.shift();
console.log(owoce);   // ["Jabłko", "Banan", "Gruszka"]
```

Usunięta została więc szuflada z wartością “Wiśnia”, a wszystkie pozostałe przesunęły się o jeden indeks w lewo.

Najbardziej elastyczna metoda tablicowa: splice()

Teraz porozmawiajmy jeszcze o metodzie

`splice()`, która jest jedną z najbardziej uniwersalnych i potężnych metod tablicowych w JavaScript. Pozwala ona usuwać elementy, dodawać nowe, a nawet robić jedno i drugie jednocześnie!

Podstawowa składnia metody:

```
tablica.splice(start, ile, [noweElementy...]);
```

- `start` – indeks, od którego zaczynamy operację,
- `ile` – ile elementów chcemy usunąć,
- `[noweElementy...]` – (opcjonalne) elementy, które chcemy wstawić w miejsce usuwanych.

Przykład 1 – Usuwanie jednego elementu:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
owoce.splice(0, 2);
console.log(owoce);  // ["Gruszka"]
```

Co się wydarzyło? Zapis

`splice(0, 2)` oznacza, iż zaczynamy od indeksu `0`, czyli w praktyce od jabłka, po czym usuwamy kolejno dwa elementy – `"Jabłko"` oraz `"Banan"`.

Przykład 2 – Wstawianie elementu bez usuwania:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
owoce.splice(1, 0, "Malina");
console.log(owoce);  // ["Jabłko", "Malina", "Banan", "Gruszka"]
```

Co się wydarzyło? Zaczynamy od indeksu

`1`, usuwamy `0` elementów (czyli żadnego), po czym dodajemy `"Malina"` dokładnie w to miejsce.

Przykład 3 – Zamiana jednego elementu na inny:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
owoce.splice(2, 1, "Wiśnia");
console.log(owoce);  // ["Jabłko", "Banan", "Wiśnia"]
```

Krok po kroku – zaczynamy od indeksu

`2`, czyli trafiamy na `"Gruszka"`, usuwamy `1` element, czyli właśnie gruszkę, po czym w miejsce usuniętej wartości wstawiamy `"Wiśnia"`.

Iterowanie po tablicy

Często musimy “przejść” po wszystkich elementach tablicy celem sprawdzenia jakiegoś warunku albo zliczenia liczby wystąpień czegoś konkretnego w tablicy.

W JavaScript mamy na to kilka sposobów:

- Klasyczna pętla `for`.
- Metoda `forEach()` – nowoczesny, czytelny sposób.
- Alternatywnie `for...of` – również bardzo przejrzysty zapis.

Podajmy przykłady iterowania po wszystkich szufladach w tablicy celem prostego wypisania ich kolejnych wartości do logu konsoli:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
for (let i = 0; i < owoce.length; i++) {
    console.log(owoce[i]);
}
owoce.forEach(function(owoc) {
    console.log(owoc);
});
for (let owoc of owoce) {
    console.log(owoc);
}
```

Każdy z powyższych sposobów daje ten sam efekt, ale różni się oczywiście składnią. Koniecznie zwróćmy też uwagę na zapis

`owoce.length` – dzięki tej właściwości możemy uzyskać “długość” tablicy, czyli w praktyce liczbę jej elementów. Zależnie od własnych preferencji, zdecyduj się na jeden ze sposobów iterowania po tablicy w JS, którego być może użyjesz podczas egzaminu INF.03.

Sprawdzanie, czy element istnieje w tablicy

W wielu przypadkach chcemy sprawdzić, czy dana wartość znajduje się już w tablicy. Używamy do tego metod:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
console.log(owoce.includes("Banan"));   // true
console.log(owoce.includes("Mango"));   // false
console.log(owoce.indexOf("Gruszka"));  // 2
console.log(owoce.indexOf("Mango"));    // -1
```

- Jeśli `includes()` zwróci `true`, to oznacza, że element znajduje się w tablicy (klasyczne sprawdzenie z jego wynikiem logicznym).
- Z kolei `indexOf()` zwraca numer indeksu pierwszej znalezionej wartości lub `-1`, jeśli elementu nie znaleziono (wyszukiwanie pozycji w tablicy).

Łączenie i modyfikowanie tablic

JavaScript pozwala nam też na łączenie dwóch tablic oraz na ich modyfikowanie – np. odwracanie kolejności lub sortowanie:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
let warzywa = ["Marchew", "Ziemniak"];
let produkty = owoce.concat(warzywa);
console.log(produkty);
```

```
// ["Jabłko", "Banan", "Gruszka", "Marchew", "Ziemniak"]
owoce.reverse();
console.log(owoce);
// ["Gruszka", "Banan", "Jabłko"]
owoce.sort();
console.log(owoce);
// ["Banan", "Gruszka", "Jabłko"]
```

Jak nietrudno zgadnąć – metoda

`concat()` łączy tablice (nazwa wzięła się od słowa: konkatenacja), `reverse()` odwraca kolejność elementów, a `sort()` sortuje alfabetycznie.

Konwersja tablicy na łańcuch i odwrotnie

Tablice możemy także przekształcić w JS w łańcuch znaków (string), np. do wyświetlenia na stronie:

```
let owoce = ["Jabłko", "Banan", "Gruszka"];
let napis = owoce.join(" ");
console.log(napis);   // "Jabłko Banan Gruszka"
let tekst = "Ala ma kota";
let slowa = tekst.split(" ");
console.log(slowa);   // ["Ala", "ma", "kota"]
```

Zobaczyliśmy akcji dwie metody:

- `join(" ")` – łączy elementy tablicy w jeden tekst, wstawiając spację między nimi,
- `split(" ")` – rozdziela ciąg tekstowy na elementy tablicy, rozcinając go tam, gdzie pojawia się spacja.

Oczywiście w obu tych metodach argumentem nie musi być spacja, a może być dowolny inny znak, np. przecinek.

Podsumowanie wiedzy o tablicach w JS

- Tablice w JavaScript pozwalają przechowywać wiele wartości w jednej strukturze danych. Tworzymy je najprościej za pomocą nawiasów kwadratowych:
  ```
  let owoce = ["Jabłko", "Banan", "Gruszka"];
  ```
- Każdy element tablicy ma swój indeks (przy czym numerujemy od 0).
- Do odczytu używamy notacji z nawiasami kwadratowymi, np: `owoce[2]`.
- Do dodawania elementów używamy `push()` (na końcu tablicy) lub `unshift()` (na początku z przeindeksowaniem).
- Do usuwania: `pop()` (ostatni element) i `shift()` (pierwszy element z przeindeksowaniem).
- Metoda `splice()` umożliwia jednocześnie usuwanie i wstawianie elementów w dowolnym miejscu tablicy.
- Iterowanie po tablicy można wykonać za pomocą: `for`, `forEach()` lub `for...of`.
- Do sprawdzania obecności elementów używamy: `includes()` (czy tablica zawiera tę wartość?) i `indexOf()` (zwracany jest indeks pierwszego wystąpienia lub -1 przy braku).
- Tablice można łączyć: `concat()`, odwracać: `reverse()` i sortować: `sort()`.

- Metoda `join()` pozwala zamienić tablicę w łańcuch znaków (string), a `split()` działa odwrotnie – tworzy tablicę z tekstu.