Массивы

Array

Для работы с наборами данных предназначены массивы. Для создания массива применяются квадратные скобки **[]**. Внутри квадратных скобок определяются элементы массива

Массив – это специальная структура данных, которая предназначена для хранения упорядоченных коллекций значений.

Пример

коллекция данных : **[123, 7, 50, -9, 24]**

Каждое значение в массиве имеет свой порядковый номер (**индекс**).
Значения называются **элементами**. Элементами могут быть любые значения – даже другие массивы.

Первый элемент массива имеет индекс 0, второй – 1, третий – 2 и т.д.

пример - журнал

Объявление (создание)

```
let arr = new Array(); // с использованием конструктора Array()

let arr = []; // посредством литерала массива
```

Примеры:

let arr = [123, 7, 50, -9, 24]

let stud = ["Иванов", "Петров", "Сидоров"]

let colors = ['black', 'white', 'grey'];

let fruits = ["Яблоко", "Апельсин", "Слива"];

Доступ к элементам

Получение доступа к элементу массива выполняется через квадратные скобки, внутрь которых нужно поместить индекс.

```
let colors = ['black', 'white', 'grey'];

console.log( colors[0] );  // 'black'

console.log( colors[1] );  // 'white'

console.log( colors[2] );  // 'grey'

console.log( colors[3] ); // undefined  - не существующий элемент
```

Изменение / добавление элементов

Чтобы изменить элемент, ему нужно просто присвоить новое значение:

['black', 'white', 'grey'];

colors[1] = 'yellow'; // ['black', 'yellow', 'grey'] изменили значение

colors[3] = 'red'; // ['black', 'yellow', 'grey', 'red'] добавили значение

colors[10] = 'green'; // ['black', 'yellow', 'grey', 'red', , , , , , , 'green']

//Если при добавлении элемента случайно пропустить индекс, то в массиве появится неопределенный элемент

colors[4] ,colors[5] , ..colors[9] - undefined

Изменение данных массива

```
colors = ['bg_red'];  // ?

console.log(colors); // ?

console.log(colors[0])  //?
```

Length - возвращает количество элементов в массиве

let n = arr.length; // количество элементов

```
colors=['black', 'yellow', 'grey', 'red']
console.log( colors.length ); // 4


 arr = [123,  7,  50,  -9,  24]
console.log( arr .length ); // 5
```

Первый и последний элемент

arr = [123, 7, 50, -9, 24]

```
let elFirst = arr[0];      //первый элемент

let n = arr.length;        // количество элементов в массиве arr

let indLast = n - 1;       //индекс последнего элемента

let elLast = arr[indLast]; //последний элемент
```

spread-оператор (оператор ...) позволяет разложить массив на отдельные значения.

Для этого перед массивом ставится многоточие: …массив

```
let arr = [123, 7, 50, -9, 24]
console.log(arr);  // [123, 7, 50, -9, 24]
console.log(...arr);  //  123  7  50 -9 24
```

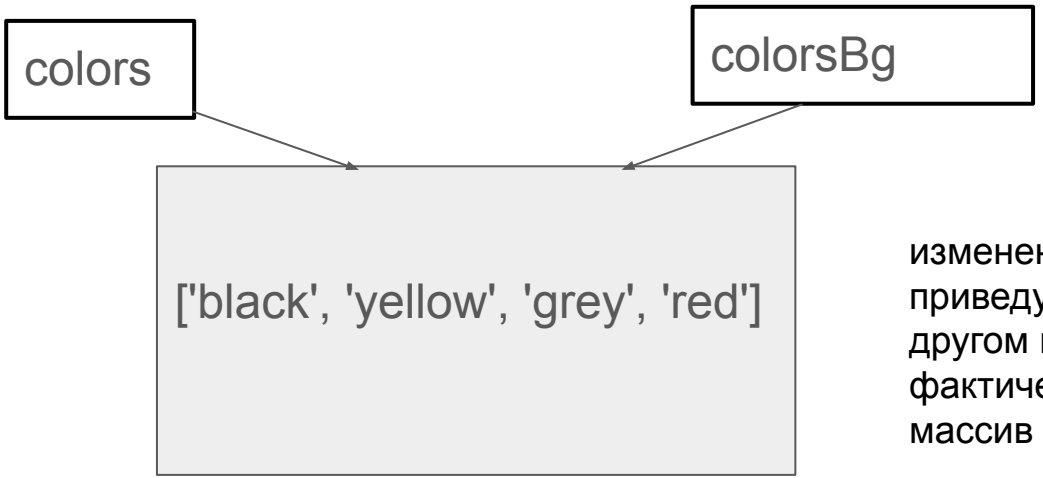
И, применяя этот оператор, можно наполнять один массив значениями из другого массива:

Неглубокое (shallow copy) копирование массива

при копировании массива передаётся не он сам, а ссылка на него

```
let colors=['black', 'yellow', 'grey', 'red']

let colorsBg = colors
```



изменения в одном массиве приведут к изменениям в другом массиве,так как фактически это один и тот же массив

Глубокое (deep copy) копирование массива с помощью оператора …spread

let **colors**=['black', 'yellow', 'grey', 'red']

создание копии массива:

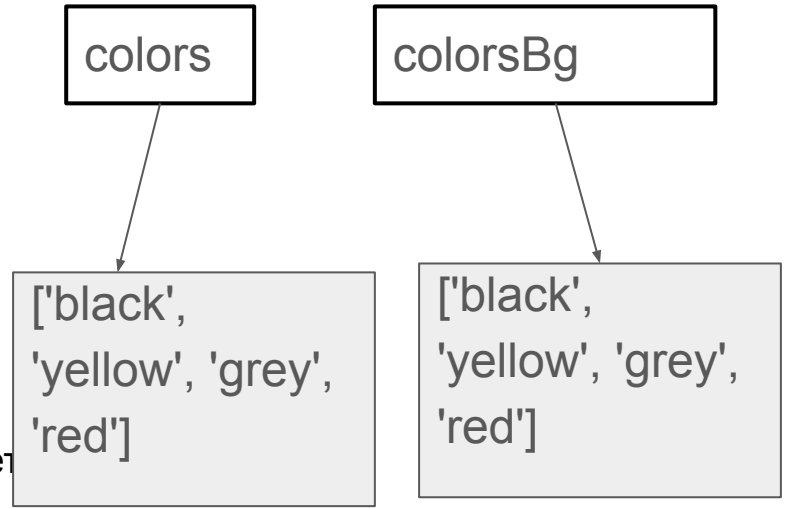
let colorsBg = [...**colors**];

let colorsBg = new Array(...**colors**);

['black', 'yellow', 'grey', 'red'] //colors

colors[0] = ‘green’ // [green, 'yellow', 'grey', 'red'] -поменяет

['black', 'yellow', 'grey', 'red'] //bg_colors - не поменяется



Объединение с помощью spread

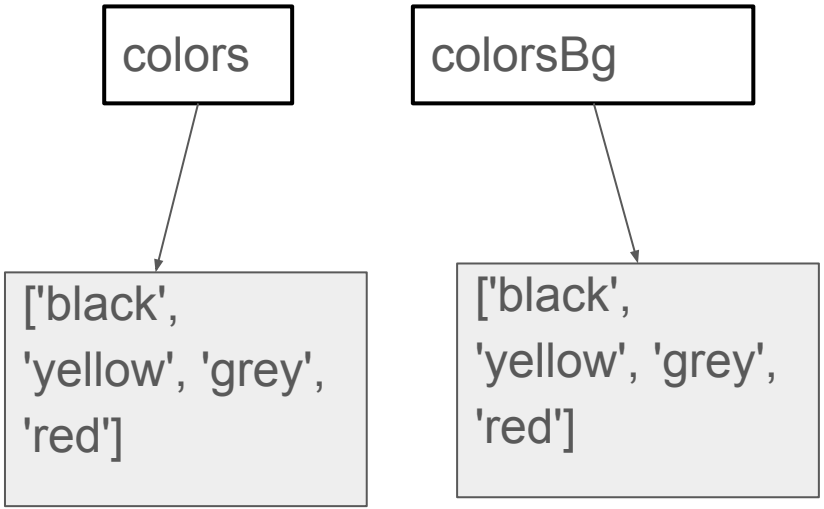
```
let colorsBg = ['yellow', 'grey', 'red']
let colorTxt = ['black', white]
let colorAll = [...colorsBg, ...colorTxt]
console.log( colorAll )     // ['yellow', 'grey', 'red','black', white]
```

Глубокое (deep copy) копирование массива slice()

let **colors**=['black', 'yellow', 'grey', 'red']

создание копии массива:

let colorsBg = **colors**.slice();



Глубокое (deep copy) копирование массива slice()

Также метод slice() позволяет скопировать часть массива. Для этого он принимает два параметра:

slice(начальный_индекс, конечный_индекс)

Первый параметр указывает на начальный индекс элемента, с которого которые используются для выборки значений из массива. А второй параметр - конечный индекс, по который надо выполнить копирование.

```
colors=['black', 'yellow', 'grey', 'red']
```

выберем в новый массив элементы, начиная с 1 индекса по индекс 2 не включая:

```
colorsBg = colors.slice(1,2);   // ['yellow', 'grey'], то есть с индексами 1 и 2
```

Если указан только начальный индекс, то копирование выполняется до конца массива:

```
colorsBg = colors.slice(2);   // ['grey', 'red'], то есть начиная со 2 и до конца
```

Задание

1. Создать массив **user** из 4 элементов и вручную ввести данные (имена пользователей)
2. Создать массив **admin** из 2 данных (имена админов)
3. Добавить пользователя в массив **user**
4. Изменить первого пользователя в массиве **admin**
5. Изменить последнего пользователя в массиве **user**
6. Объединить массивы **user** и **admin** в массив **users**
7. Подсчитать количество элементов массива **users**,
8. Создать массив **people**, состоящий из элементов массива **users** (массивы должны быть независимыми друг от друга)
9. Добавить в массив **people** еще двоих пользователей.
10. Вывести данные массива **users** и **people**

метод forEach() - перебор массива

Метод forEach позволяет последовательно перебрать все элементы массива.

Метод в параметре получает функцию(callback), которая выполнится для каждого элемента массива. В эту функцию можно передавать 3 параметра.

Если эти параметры есть (они не обязательны), то в первый автоматически попадет *элемент массива*, во второй попадет его *номер в массиве (индекс)*, а в третий - *сам массив.*

```
массив.forEach(function(элемент, индекс, массив) {
    //код, который выполнится для всех элементов
})
```

Вывод элементов массива

Для этого в анонимную функцию передадим первый параметр elem (назвать его можно как угодно). В эту переменную последовательно будут попадать элементы массива:

```
let colors=['black', 'yellow', 'grey', 'red']

colors.forEach(function(elem) {

    console.log(elem);

});
```

Вывод элементов и индекса массива

Для этого в анонимную функцию передадим первый параметр elem (назвать его можно как угодно). В эту переменную последовательно будут попадать элементы массива:

```
let colors=['black', 'yellow', 'grey', 'red']

color.forEach(function(elem, i) {

    console.log(`color[${i}] = ${elem}`);

});
```

Вывод элементов, индекса массива и самого массива

Для этого в анонимную функцию передадим первый параметр elem (назвать его можно как угодно). В эту переменную последовательно будут попадать элементы массива:

```
let colors=['black', 'yellow', 'grey', 'red']

color.forEach(function(elem, i,arr) {

   console.log(`color[${i}] = ${elem}`);

   console.log(arr);

});
```

Дан массив

```
arr = [123, 7, 50, -9, 24]
```

- Вывести все элементы массива
- Найти сумму элементов массива

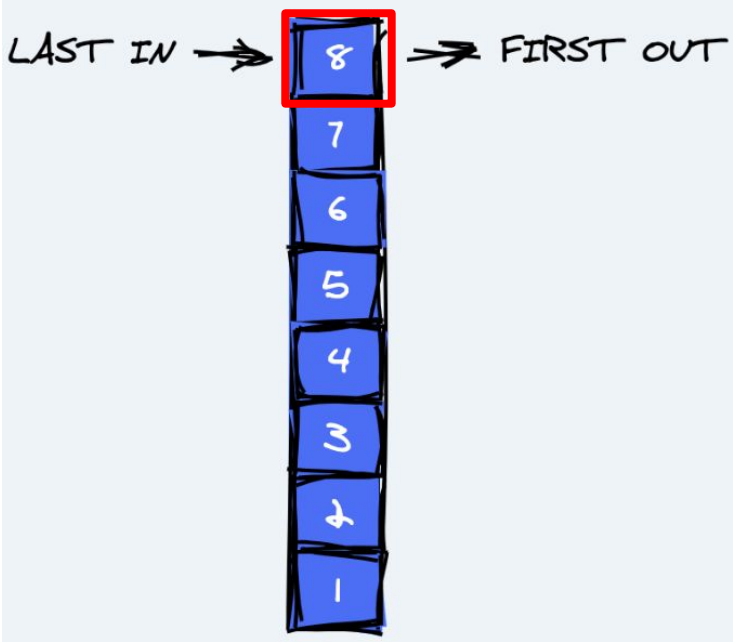
Структуры данных в JS: стек и очередь

Мы можем воспринимать структуры данных как эффективный способ хранения. Существуют разные структуры данных, каждая имеет свой вариант использования, но все они должны иметь одну и ту же цель — добиться максимальной производительности при хранении и работе с ними.

Стек : первый пришел - последний ушел
последний вошел - первым вышел

Этот последовательный порядок можно описать как стопка тарелок, стопка блинов

Например, операция «отменить» текстового редактора использует стек для организации данных;



Операции стека - работают с **концом** массива

- `push()` добавляет элемент в конец.
- `pop()` удаляет последний элемент.

Стек полезен, когда мы хотим добавить данные в последовательном порядке и удалить эти данные. Основываясь на своем определении, стек может удалить только самые последние добавленные данные.

метод `pop` очищает индекс и уменьшает значение `length`.

push() / pop()

let **colors**=['black', 'yellow', 'grey', 'red']

Добавить новый цвет white

colors.push('white'); ////Добавляет последний элемент в массив и возвращает количество элементов нового массива

удалить новый цвет

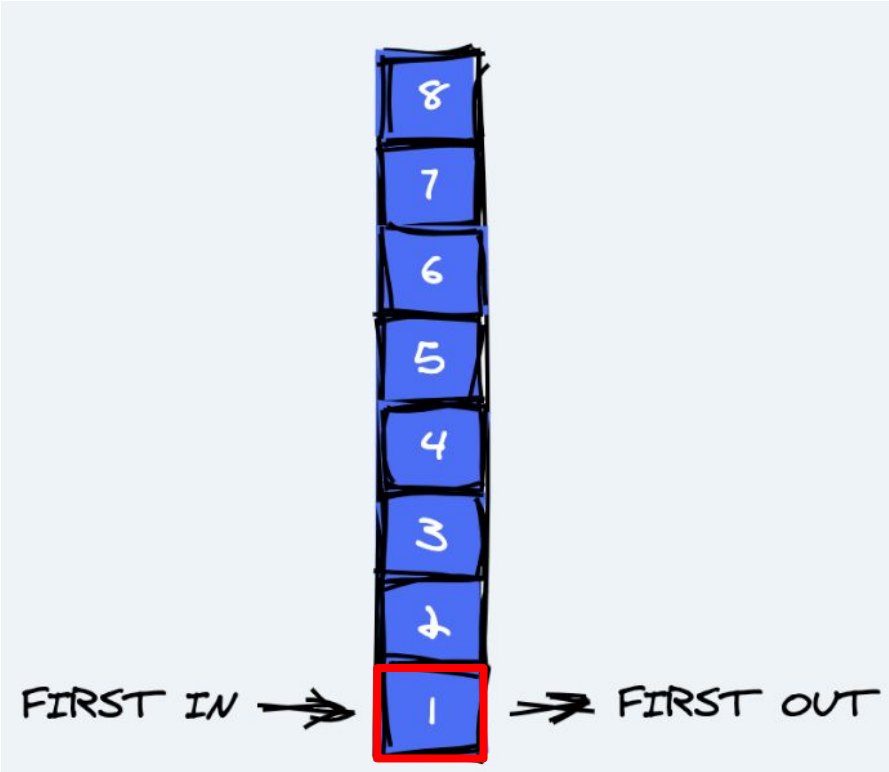
colors.pop() //Удаляет последний элемент из массива и возвращает этот элемент

Задание -
Дан массив 10 чисел.
Создать массив положительных чисел

Очередь: первым пришел - первым ушел

Этот последовательный порядок похож на систему билетов в гастроном. Каждый клиент берет билет и обслуживается при вызове их номера. Сначала нужно обслуживать клиента, который берет первый билет.

Более практичным примером очереди является цикл событий веб-браузера. По мере запуска различных событий, таких как щелчок кнопки, они добавляются в очередь цикла событий и обрабатываются в том порядке, в котором они попали в очередь.



Операции очереди - работают с **началом** массива

- **`unshift()`** добавляет элемент в начало.
- **`shift()`** удаляет первый элемент.

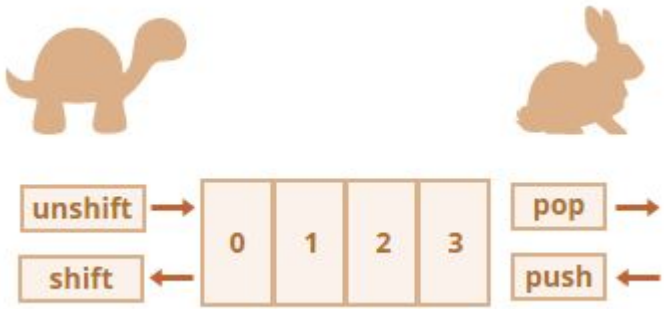
В отличие от стека, очередь удаляет самые старые добавленные данные.

Операция `shift` должна выполнить 3 действия:

1. Удалить элемент с индексом `0`.
2. Сдвинуть все элементы влево, заново пронумеровать их, заменив `1` на `0`, `2` на `1` и т.д.
3. Обновить свойство `length`

Эффективность

Методы `push/pop` выполняются быстро, а методы `shift/unshift` – медленно.



`unshift() / shift()`

let **colors**=['black', 'yellow', 'grey', 'red']

Добавить новый цвет white в начало

colors.unshift('white'); //Добавляет первый элемент в массив и возвращает количество элементов нового массива

удалить цвет в начале

colors.shift() //Удаляет первый элемент из массива и возвращает этот элемент

Задание

1. Ввести n элементов массива(через push)
2. Поменять местами первый и последний элементы
3. Подсчитать количество элементов в массиве
4. Вывести :
    a. положительные элементы (новый массив),
    b. четные элементы (новый массив)
    c. последний элемент
    d. сумму элементов
    e. количество четных элементов
    f. количество простых элементов
    g. те элементы, которых больше среди простых и составных