Объект Array

Числовой массив (Array) – это упорядоченный набор данных, который имеет имя и является экземпляром объекта Array. Он состоит из элементов, доступ к которым осуществляется с помощью их порядкового номера (индекса).

Как уже говорили раньше, для хранения информации используются переменные. Иногда необходимо хранить несколько значений одного и того же показателя. Например, имена: Вася, Саша и Ваня. В таком случае можно использовать несколько переменных (согласитесь, не очень удобно и легко запутаться) или же числовой массив.

Числовой массив (Array) – это объект, который служит для хранения и обработки множества (нескольких) значений. По своему устройству массив (слово «числовой» опустим. Когда говорят или пишут слово «массив», подразумевают - числовой массив. В JavaScript существует несколько видов массивов и коллекции. С ними познакомитесь позже) похож на шкаф. Массив состоит из элементов, как шкаф из полочек. На каждую полку можно класть предметы, в каждый элемент можно записывать данные (информацию). Если говорить о именах, то раньше пришлось бы создавать три переменные, теперь можно создать один массив, элементы которого будут хранить информацию о количестве посетителей/пользователей и тд. Как и в шкаф, в элементы можно «класть» различные «вещи». Помня, что JavaScript слабо типизированный, то РАЗЛИЧНЫЕ – не преувеличение. Например, в массиве можно хранить почтовый адрес:

• первый элемент – число – почтовый индекс;

• второй – строка – страна;

• третий – строка – город;

• четвертый – строка – улица;

• пятый – число – номер дома.

Вместо пяти переменных разных типов, одна – объект Array (числовой массив).

**Создание**



Чтобы сходу не запутаться в скобках, начнем знакомство с более редкой формой создания массивов. Данный подход вам уже знаком по работе объектом Date. Так будет проще понять другие формы записи.

Начнем.

**var ar = new Array();**

Где, **ar** – имя массива (дается по тем же правилам, что и имя переменной). После такой записи **ar** – это экземпляр объекта, или переменная объектного типа.

Если мы знаем, сколько элементов должен иметь массив, то пишем так

**var n = 10;**

**var ar = new Array(n);**

где **n** – количество элементов массива. Во второй строке вместо n может быть число.

Если мы знаем значения элементов массива, то можно писать так

**var ar = new Array(3,"hello",3.1);**

В скобках перечисляются значения элементов. Не забываем, что они могут быть разных типов.

Чтобы обратится к элементу, используют квадратные скобки. Похоже на работу с параметрами функций, но это другое. Выглядит это так

**Имя\_массива[номер\_элемента]**

Элементы нумеруются с нуля. Если есть массив из **n** элементов, то первый элемент будет иметь номер **0**, а последний – (**n-1**).

 **Пример**

Создать массив. Инициализировать массив в скрипте. Вывести массив в сообщении.



Открываем нашу страницу в браузере. Результат работы скрипта – на рисунке ниже



Как видим, переменная ar содержит не одно, а множество значений (массив значений).

**Работа с элементами массива**

Для того чтобы обратиться к определённому элементу массива, необходимо указать имя этого массива и в квадратных скобках его индекс(То, что мы сделали выше). Данная операция ещё называется **операцией индексации**.

Например, создадим пустой массив и добавим в него 4 текстовых элемента:



Например, выведем в консоль браузера (F12) значения 2 и 4 элемента массива smartphoneColors:





Разобравшись с классической формой создания экземпляров объекта через **new**, перейдем к короткой форме записи. Именно она используется чаще всего.

*В примерах умышлено встречаются все формы объявления массивов. Преобразовывайте их самостоятельно к короткой форме записи.*



**Длина массива (количество элементов в массиве)**

Определение количества элементов в массиве осуществляется с помощью свойства **length**. (с этим свойство вы уже сталкивались во время знакомства со строковым типом данных - String). У каждого массива есть свойство **length**, которое автоматически меняется при каждом обновлении массива. ( В отличие от многих других языков программирования массивы в JavaScript автоматически меняют свой размер, т.е. они изначально являются **динамическими**. Таким массивам не надо задавать какие-либо размеры.) Длина массива **length** - это не количество элементов (как в String), а максимальный целый ключ + 1:

**Пример**



Можно добавить объект под любым индексом, даже превосходящим текущую длину массива. При этом массив хранит (потребляет память) только для тех элементов, которые установлены (не undefined). Такие массивы называются **разреженными.**

**Пример**: разреженный массив хранит два элемента: номер 1 и номер 999.



Следует избегать разреженных массивов. Массивы предназначены именно для работы с упорядоченным (непрерывным) множеством элементов. Это вызвано тем, что современные интерпретаторы стараются оптимизировать их и хранить в оперативной памяти в виде непрерывной области памяти, по которой можно быстро «пробежаться» от начала до конца.

Как только браузер встречает необычное поведение массива, например устанавливается значение arr[0], а потом сразу arr[1000], то он начинает работать с ним, как c более сложной структурой – ассоциативным массивом.

*(Ну, это как хранить список из пары элементов не в блокноте, а в электронной таблице Excel.)*

Последнее чревато снижением скорости обработки числового массива.

**Укорачивание длинны массива и его полная очистка**.

Значение свойства **length** можно изменять в меньшую сторону. Причем этот процесс **необратимый**, т.е. даже если потом вернуть **length** обратно – значения не восстановятся:



**Получение первого элемента массива**

Получение значения первого элемента массива осуществляется с помощью указания в квадратных скобках этого массива числа 0:



**Получение последнего элемента массива**

Получение значения последнего элемента массива осуществляется с помощью указания в квадратных скобках этого массива выражения имя\_массива.length-1:



**Перебор массива (массивы+циклы)**

**Перебор** элементов массива осуществляется **с помощью цикла for**.

**Пример классического прохода по массиву**





**Цикл For in и массивы**

Если требуется осуществлять проход учитывая содержимое элементов, а не их индексы, используется цикл For in.

Пример

Имеется два массива. Необходимо найти значения, которые одновременно присутствуют в обоих массивах.



**Удаление элементов массива**

В JavaScript массивы являются динамическими, то есть их размер может изменятся в ходе выполнения сценария. Теоретически, такой массив может быть бесконечным (все упирается в ресурсы ПК). Так же говорилось, что «кушают» память, только инициализированные элементы (которым присвоено значение). Не инициализированные элементы пустого массива имеют значение **undefined**. Следовательно, чтобы удалить элемент массива, достаточно сделать его неопределенным. При этом, в классическом представлении, сам элемент не удалится. Следует это учитывать при работе со свойством **length.**

**Оператор delete**

Оператор delete используется не для удаления элемента из массива, а для присваивания данному элементу массива значение undefined. После этой операции массив станет **разреженным**.



Выведите в консоль значение свойства **length** до и после применения оператора **delete.**

**Копирование массива**

Иногда возникает необходимость создать копию массива перед рискованными операциями с ним. Как мы убедились выше, внеся изменения в значения элементов массива или манипулируя его длинной, вернуться к его начальному состоянию становится невозможным. Только через полную очистку и повторное создание.

Но есть способ избежать таких трудностей. Реализуйте следующий пример:



Такое поведение вызвано тем, что **arr** является «автономным» экземпляром объекта **Array**. Экземпляры объектов нельзя копировать как переменные.

В примере выше **arr2** содержит не копию экземпляра **arr** объекта **Array**, а ссылку на область памяти уже существующего экземпляра **arr**. В частности, сравнение **arr2 ==** **arr** даст **true**.

Чтобы получить новый экземпляр объекта Array, его нужно создать. Чтобы он стал копией первого, требуется передать в него те же значения элементов.



Такой подход не практичен. Массивы могут содержать большое количество элементов и прописывать их явно в коде не получится. Решить задачу копирования можно с помощью пустого массива и цикла. Такое копирование будет происходить динамически и не зависит от размерности (количества элементов) копируемого массива.



**Поменять элементы местами**

Обмен значениями между двумя элементами одного массива, можно реализовать через переменную (буфер).



**Задание**

Создайте массив arr=['a', 'b', 'c', 'd', 'e']. С помощью одной переменной test сделайте из него массив ['e', 'd', 'c', 'b', 'a'].

**Поиск в массиве**

Мы уже увидели, что доступ к элементам массива (для чтения, изменения или удаления) происходит по их интедексу. А вот поиск, обычно, происходит по значению элемента. Пользователи знают значения, которые они ищут, но никак не индексы элементов или структуры данных в которых эти значения хранятся.

**Пример:**

Создайте функцию поиска в массиве find с двумя параметрами arr и value. Функция ищет в массиве arr значение value и возвращает его индекс, если найдено, или -1, если не найдено.



**Наполнение массива**

Динамическое наполнение массивов появилось в JavaScript совсем недавно (в ES6) , и ещё плохо поддерживается браузерами, но как факт существует.

Метод **fill()** заполняет все элементы массива от начального до конечного индексов одним значением.

Array(1000).fill(1);

[1, 2, 3].fill(4) // [4, 4, 4]

Например, чтобы включить поддержку в chrome://flags активируйте запись «Включить экспериментальный JavaScript».

Ситуацию с этим методом не изменил выход ES7 летом 2016. Например, браузер Опера совсем не поддерживает данный метод.

Существует и классический подход



**Задание**

1. Создайте массив из 1000 элементов. Первые 100 элементов =0, вторая сотня =2 ….последняя сотня =9.
2. Создайте функцию, которая сравнивает два массива на предмет отсутствия одинаковых значений.

Дополнительное задание. Закрепление навыков работы с циклами.

  