Лабораторная работа 13. Массивы и объекты и функции в JavaScript.

Цель работы. Познакомиться с основными структурами данных: массивами, объектами и функциями в JavaScript, а также способами их применения для изменения отрисовки web-страниц в браузере.

Результатом работы должны быть выполненные задания (12.1–12.6) /или (для студентов с полным доступом к тренажерам) пройденные тренажеры на htmlacademy.ru (https://htmlacademy.ru/courses/213, https://htmlacademy.ru/courses/215 , https://htmlacademy.ru/courses/217)

Конспект «Массивы»

Массив — тип данных, который представляет собой список элементов, у каждого из которых есть свой порядковый номер.

В массиве можно хранить любые данные: строки, булевы значения, числа и даже другие массивы.

Нумерация элементов массива начинается с нуля, поэтому порядковый номер (индекс) первого элемента равен нулю.

В качестве индекса можно использовать переменную.

Используйте команду `[].length`, чтобы узнать длину массива (сколько в нём элементов). С её помощью можно получить последний элемент массива.

```
let numbers = [1, 2, 3, 4, 5];
let index = 3;

console.log(numbers[0]);                    // Выведет: 1
console.log(numbers[index]);                // Выведет: 4
console.log(numbers.length);                // Выведет: 5
console.log(numbers[numbers.length - 1]); // Выведет: 5
```

Массивы можно перебирать в циклах. Например, цикл ниже выводит элементы массива в консоль по очереди и прекращает свою работу, когда i станет равно длине массива.

```
let numbers = [1, 2, 3, 4, 5];

for (let i = 0; i <= numbers.length - 1; i++) {
  console.log(numbers[i]);
}

// Выведет: 1
// Выведет: 2
// Выведет: 3
// Выведет: 4
// Выведет: 5
```

Запись в массив происходит так же, как чтение, через квадратные скобки.

```
let numbers = [];
let index = 1;
```

```
numbers[0] = 1;
numbers[index] = 2;

console.log(numbers); // Выведет: [1,2]
```

Сортировка массива

```
let numbers = [12, 3, 7, 9, 10, 5];

for (let i = 0; i <= numbers.length - 2; i++) {
  let minValue = numbers[i];

  for (let j = i + 1; j <= numbers.length - 1; j++) {
    if (numbers[j] < minValue) {
      minValue = numbers[j];
      let swap = numbers[i];
      numbers[i] = minValue;
      numbers[j] = swap;
    }
  }
}

console.log(numbers); // Выведет: [3, 5, 7, 9, 10, 12];
```

Массив с числами numbers сортируется по возрастанию элементов. На каждой итерации мы сравниваем minValue с остальными элементами массива. Если какой-то из них окажется меньше, чем minValue, он запишется в minValue, перезаписав старое значение, и переместится в начало массива. Переменная swap — вспомогательная переменная, с помощью которой мы можем поменять элементы местами.

Поиск медианы массива

```
let median;
if (usersByDay.length % 2 !== 0) {
  let medianIndex = (usersByDay.length - 1) / 2;
  median = usersByDay[medianIndex];
} else {
  let leftIndex = usersByDay.length / 2 - 1;
  let rightIndex = usersByDay.length / 2;
  median = (usersByDay[leftIndex] + usersByDay[rightIndex]) / 2;
}

console.log(median);
```

Конспект Функция

Функция — кусок кода, который можно написать один раз, а затем многократно использовать. Функция не просто содержит в себе значение, как переменная, а выполняет какое-то действие и решает какую-то задачу: считает, сравнивает, ищет.

Код внутри { } называется «телом функции».

```
let functionName = function () {
  // Тело функции
};
```

Чтобы функция начала свою работу, её надо вызвать. Для этого нужно обратиться к функции по её имени, а затем указать круглые скобки.

```
functionName();
```

Параметры и аргументы функции

Параметры — значения, с помощью которых можно настраивать функции. Так мы можем узнать результат работы функции для разных случаев.

В момент объявления функции, в круглых скобках, мы создаём параметры. Здесь всё, как с переменными: сначала задаём параметрам имена, которые описывают, что за значения будут в них записаны. Если параметров несколько, они записываются через запятую.

Параметры работают так же, как переменные. Мы подставляем их вместо фиксированных значений в операции внутри функции. При выполнении кода вместо каждого параметра подставится его значение.

В момент вызова функции мы указываем в круглых скобках те значения, которые окажутся в параметрах.

```
let showTime = function (hours, minutes) {
  console.log('Текущее время: ' + hours + ':' + minutes);
};

showTime(3, 15);    // Выведет: Текущее время: 3:15
showTime(16, 20);   // Выведет: Текущее время: 16:20
```

Правильно говорить «функция принимает параметры», но при этом мы «передаём функции аргументы».

Если у функции указан параметр, но аргумент не передан, то значение параметра в теле функции будет undefined — то есть «не определено».

Передавать аргументы надо в том же порядке, в котором объявлены параметры функции.

Возвращение из функции

Функции умеют ***возвращать*** результат своей работы. Это значит, что функция может выполнить код и отдать результат операций для дальнейшей работы с этим результатом. Он подставится в то место кода, где мы вызвали функцию.

Чтобы функция вернула значение, мы используем оператор return. После оператора указываем, что именно надо вернуть. Когда программа доходит до строки с return, функция отдаёт результат своей работы и выполнение кода из тела функции останавливается, иными словами ***происходит выход из функции.***

- Код, написанный **на новой строке** после `return`, не выполняется.
- Функция не может вернуть сразу много значений, она возвращает **только один** результат.
- Если внутри функции нет `return` или после `return` не указано, какое значение нужно вернуть, функция вернёт `undefined`, иными словами, **ничего**.

Пример функции:

```
let calculateSum = function (numberFirst, numberSecond) {
  let sum = numberFirst + numberSecond;
  return sum;
};

calculateSum(); // Вернёт NaN
calculateSum(2); // Вернёт NaN
calculateSum(2, 5); // Вернёт 7
calculateSum(9, 5); // Вернёт 14
```

В этом примере:

- calculateSum — имя, по которому можно обратиться к функции.
- numberFirst, numberSecond — параметры функции.
- return sum; — место кода, где происходит возвращение sum и выход из функции.
- calculateSum(2, 5); — аргументы, которые передаются в функции при вызове. Порядок аргументов такой же, как у параметров функции. Первый аргумент 2 записывается в первый параметр numberFirst, аргумент 5 записывается в параметр numberSecond. Важно соблюдать порядок параметров при вызове функции, чтобы избежать неочевидных ошибок.

```
// Функция подсчёта миль
let calculateMiles = function (distance, isBusinessClass) {
  let percent = 0.18;
  if (isBusinessClass) {
    percent += 0.04;
  }
  if (distance > 3500) {
     percent += 0.15;
  }
  return distance * percent;
};

// Функция, которая считает количество полётов
let calculateFlights = function (distance, isBusinessClass, milesTarget) {
// Вызываем одну функцию из другой
  let miles = calculateMiles(distance, isBusinessClass);
  let flights = Math.ceil(milesTarget / miles);
  return flights;
};

// Массив миль, которые нужно накопить
let targets = [1500, 3000, 5000, 7500, 10000, 15000];

// Цикл, в котором выясняется, какими перелётами мили накопятся быстрей
for (let i = 0; i < targets.length; i++) {
  let flightsVariantFirst = calculateFlights(3118, true, targets[i]);
  let flightsVariantSecond = calculateFlights(3617, false, targets[i]);
  console.log('Необходимое количество полётов в бизнес-классе до Валенсии: '
+ flightsVariantFirst);
  console.log('Необходимое количество полётов в экономе до Лиссабона: ' +
flightsVariantSecond);
```

```
  if (flightsVariantFirst > flightsVariantSecond) {
    console.log('Быстрей накопишь полётами в экономе до Лиссабона!
Количество полётов: ' + flightsVariantSecond);
  } else {
  console.log('Быстрей накопишь полётами в бизнесе до Валенсии! Количество
полётов: ' + flightsVariantFirst);
  }
}
```

Конспект «Объект»

Создание объекта

Объект — тип данных, который хранит в себе информацию в виде пар «ключ-значение». Каждый элемент сопоставлен со своим ключом и порядок элементов совсем неважен.

Ключи в объекте следует делать уникальными. Если в одном объекте несколько одинаковых ключей, то используется значение последнего.

Несколько правил синтаксиса:

- Ключ обособляется от значения двоеточием.
- Пары «ключ-значение» отделяются друг от друга запятыми.
- Значениями могут быть данные любого типа (число, строка, массив и так далее).

```
let cat = {
  name: 'Кекс',
  age: 5
};
```

Чтение из объекта

Чтобы получить значение свойства, к нему надо обратиться через точку объект.ключ. Такая запись называется точечной нотацией и возвращает значение свойства объекта, если такое свойство есть.

В противном случае вы получите undefined, то есть ничего.

```
console.log(cat.name); // Выведет в консоль: Кекс
console.log(cat.age); // Выведет в консоль: 5
console.log(cat.color); // Выведет: undefined. Такого ключа в объекте нет
```

Запись в объект

Свойства объектов можно не только читать, но и переопределять, как и обычные переменные. А ещё в объект можно добавлять новые свойства уже после того, как он был создан.

```
cat.age++; // Увеличили возраст кота на 1
console.log(cat.age) // Выведет: 6
cat.name = 'Рокки'; // Заменили снаружи значение свойства name
console.log(cat.name); // Выведет: Рокки
cat.color = 'рыжий'; // Добавили в объект новое свойство
```

```
console.log(cat.color); // Выведет: рыжий
```

Передача по ссылке

Объект всегда один, в памяти не создаётся новое место под копию объекта. Каждая переменная содержит не новую отдельную сущность, а ссылку на один-единственный объект. Поэтому, когда мы меняем что-то в объекте через одну из переменных, в которой содержится ссылка на него, изменения видны во всех других переменных, будь их хоть двадцать или сорок. Это важная особенность объектов, которую надо запомнить. Она так и называется — ***передача объектов по ссылке***.

```
let firstCat = {
  name: 'Кекс',
  age: 5
};

let secondCat = firstCat;
console.log(secondCat); // Выведет: {name: "Кекс", age:5}

firstCat.name = 'Снежок';
console.log(secondCat); // Выведет: {name: "Снежок", age:5}
```

В объектах могут храниться любые типы данных, в том числе и функции. Такие свойства-функции называются методами объектов. Они вызываются так же, как и любые другие функции, через круглые скобки, а обращаемся мы к методам, как и к свойствам объекта. В итоге вызов метода записывается так: **`объект.метод()`**.

Главное в создании метода — придумать подходящее название, описывающее, что делает этот метод. В мире программирования есть устоявшиеся традиции именования. Например, функции, которые что-то возвращают, называются геттерами и начинаются со слова get. Это выглядит так:

```
let cat = {
  name: 'Кекс',
  color: 'рыжий',
  age: 5,
  getGreeting: function() {
    return 'Мяу, привет!';
  }
};
console.log(cat.getGreeting()); // Выведет: Мяу, привет!
```

Раз методы — это те же функции, почему мы вообще записываем их в объект, а не используем привычные внешние функции?

Методы используются для работы с объектами. Они читают свойства, переписывают их и возвращают. Да, можно создать внешнюю функцию, передавать ей объект и обрабатывать внутри этой функции. Но намного удобней держать в объекте всё, что относится именно к этому объекту.

На самом деле мы уже использовали методы. Например, .length для определения длины массива — это тоже метод — стандартная функция, возвращающая число. Метод log объекта console выводит сообщение в консоль.

Метод prompt объекта window (окно браузера) запускает отображение окна с полем и кнопками.

Метод — это та же функция, но привязанная в какому-либо объекту.

Почти всё в JS является объектом. Даже теги и элементы на странице — тоже объекты. И у каждого объекта есть свои заранее созданные для работы методы.

Задание. Напишите 6 программ. Каждую поместите в файл scriptN.js, где N – соответственно, номер программы. Поместите их в отдельную директорию в своем репозитории на гитхабе или на гуглдиске. В качестве ответа разместите в курсе ссылку на эту директорию.

1 Список покупок

Напиши программу, которая составит из элементов массива список покупок.

В результате должна получиться строка с элементами массива через запятую вида “элемент, элемент, элемент”, выведенная в консоль.

Каждый элемент должен быть отделён запятой, точка в конце строки не нужна. И помни про пробелы перед всеми словами, кроме первого.

Элементы должны добавляться в строку последовательно, начиная с самого первого элемента массива, заканчивая последним.

Массив с покупками записан в переменную groceries.

Строку со списком покупок записывай в переменную shoppingList.

Например,

```
let groceries = ['чай', 'шпроты', 'печенье', 'сахар', 'чипсы'];
let shoppingList = '';
/* далее – ваш код */
```

2 Сортировка выбором

2.1 В поисках элемента. Напишите программу, которая будет находить последний индекс определенного элемента в массиве и выводить сообщение об этом в функцию alert().

Что значит последний индекс? Например, в массиве [1, 2, 1, 4, 1] три единицы. Если мы будем искать индекс единицы в этом массиве с первого элемента, то сначала мы получим 0, потому что единица — первый элемент массива. Но в массиве это не единственная единица. Затем мы найдём единицу по индексу 2, но это нам тоже не подходит. Нам нужен индекс последней единицы в массиве. Это индекс 4.

Массив записан в переменную numbers.

Элемент, последний индекс которого надо найти, записан в переменную number.

Создайте переменную **`lastIndex`** и записывайте в неё последний индекс числа **`number`**. Если числа **`number`** в массиве нет, переменная **`lastIndex`** должна быть равна -1.

2.2 В поисках элемента. Массив записан в переменную numbers. Требуется отсортировать его методом «выбора» элементов по возрастанию, от самого маленького значения к наибольшему.

Алгоритм этой сортировки устроен так:

- В массиве находится самый маленький элемент. Он меняется местами с нулевым элементом массива. Таким образом самый маленький элемент стоит в начале.
- Затем мы исключаем из выборки нулевой элемент, ведь он уже найден. Среди элементов с первого (с индексом 1) по последний ищется самый маленький элемент. Когда он найден, он меняется местами с первым элементом. В итоге два самых маленьких значения в массиве стоят в начале.
- Затем поиск наименьшего значения продолжается среди элементов с третьего (с индексом 2) по последний. И так до тех пор, пока массив не будет отсортирован по возрастанию значений.

По завершению сортировки вывести индекс элемента, указанного при вводе. Если этого элемента нет в массиве, вывести строку «*элемент не найден*»

3. Третья программа: «От зарплаты до зарплаты»

Макс хочет, чтобы вы написали программу для онлайн- калькулятора, который автоматизирует самое сокровенное — бухгалтерские расчёты!

Ему понадобилось нанять несколько новых сотрудников, так как местных бездельников уже не хватает. Бухгалтерия выделила бюджет на зарплаты, но он включает налоги, а в вакансии зарплату нужно указать без налогов. Впрочем, вот техническое задание:

Ему нужна программа, которая от «грязной» зарплаты (зарплата до вычета налогов) посчитает примерную «чистую» зарплату (которая выдаётся на лапы).

Оформи программу в виде функции **`calculateSalary`** с одним параметром — величиной грязной зарплаты. Функция должна возвращать чистую зарплату.

В зависимости от того, как вы будете считать проценты, итоговый результат может иметь дробную часть. Используйте **`Math.round`** для округления результатов вычисления.

Для облегчения вашей работы Макс предлагает воспользоваться данной заготовкой (с описанием ТЗ)

```
let calculateSalary = function () {

};

/* Техническое задание

Мне нужна программа, которая от «грязной» зарплаты (зарплата до вычета
налогов) посчитает примерную «чистую» зарплату (которая выдаётся на лапы).
```

```
Оформи программу в виде функции calculateSalary с одним параметром —
величиной грязной зарплаты. Функция должна возвращать чистую зарплату.

Большая точность мне не нужна, просто считаем, что 35% величины грязной
зарплаты составляют налоги, а если грязная зарплата больше или равна 100
тысячам, то налоги составляют уже 45%.
*/
```

4. Четвертая программа: «Сколько стоит ваш дизайн-макет?»

Пока мы считали мили, в дизайн-студию Макса поступили новые заказы. Их так много, что студия не может справиться с таким объемом работ. Придётся выбирать наиболее выгодные предложения. Чтобы не считать стоимости проектов вручную, Макс просит вас написать программу.

Напиши программу для расчёта стоимости проекта.

Назови функцию getPrice. У неё должно быть два параметра:

время (в часах), которое нужно потратить на проект;

булево значение, которое указывает на срочность проекта — true для срочного заказа и false для обычного.

Названия параметров могут быть любыми.

Для каждого проекта есть фиксированная ставка — 1500 рублей в час. Расчёт стоимости проектов выглядит так: время * ставка в час.

Есть несколько нюансов. Если проект срочный, то часы уменьшаются в два раза, а ставка за час повышается в 2.5 раз.

А если время проекта больше 150 часов, ставка в час уменьшается на 250 рублей.

В первую очередь проверяй срочность. Функция должна возвращать стоимость проекта.

Почему нужно сначала проверять срочность? Чтобы программа работала корректно.

Представьте, что пришёл срочный проект на 160 часов. Сначала мы проверяем срочность, увеличиваем ставку и уменьшаем количество часов вдвоё. Часов станет 80. Тогда проверка на количество часов уже не пройдёт и не внесёт свои коррективы в ответ. Так как время часов изменилось, эта проверка и не должна проходить.

Если пришёл срочный проект на 400 часов, то сначала мы сокращаем время в два раза, увеличиваем ставку. Время проекта теперь 200 часов, что всё равно больше 150, поэтому мы снижаем ставку на 250 рублей, раз проект всё равно большой.

Вывести стоимость заказа в alert-функцию.

```
/* Техническое задание
Напиши программу для расчёта стоимости проекта.
```

```
Назови функцию getPrice. У неё должно быть два параметра:
- время (в часах), которое нужно потратить на проект;
- булево значение, которое указывает на срочность проекта — true для
срочного заказа и false для обычного.
Названия параметров могу быть любыми.

Для каждого проекта есть фиксированная ставка — 1500 рублей в час. Расчёт
стоимости проектов выглядит так: время * ставка в час.

Есть несколько нюансов. Если проект срочный, то часы уменьшаются в два раза,
а ставка за час повышается в 2.5 раз.
А если время проекта больше 150 часов, ставка в час уменьшается на 250
рублей.
В первую очередь проверяй срочность. Функция должна возвращать стоимость
проекта в функцию alert().

*/
```

4. Подготовка к созданию формы заказа в интернет-магазине

В новый проект Интернет-магазина, надо добавить корзину для заказа товаров.

Вот небольшое ТЗ:
Пользователь выбирает компьютер, а программа выводит результат заказа.
В сообщении должны быть технические характеристики и итоговая цена товара.
Есть базовая цена любого компьютера, а конечная сумма зависит от каждого технического показателя устройства.

5. Сортировка объектов

В этом задании вам нужно написать сортировку в массиве объектов. Вы можете использовать любой алгоритм сортировки. Например, можете написать сортировку выбором, с ней вы уже встречались во 2-м задании.

Создайте функцию **`getSortedArray`**. У неё должно быть два параметра. Первый — массив, который нужно отсортировать. Второй — имя ключа в объектах. Именно по значению этого ключа нужно будет делать сортировку.

Функция должна возвращать отсортированный массив объектов. Значения в массиве должны увеличиваться от меньшего к большему.

Подсказка! Примерно так будет выглядеть результат работы вашей программы:

```
// Массив, который надо отсортировать
// Сортировать будем по значению в ключе age

[
  {
    name: 'Петя',
    age: 5
  },
  {
    name: 'Лёля',
    age: 2
  },
```

```
  {
    name: 'Сима',
    age: 3
  }
];

// Отсортированный массив
[
  {
    name: 'Лёля',
    age: 2
  },
  {
    name: 'Сима',
    age: 3
  },
  {
    name: 'Петя',
    age: 5
  }
];
```

6. **Собираем массив объектов**

В этом задании вам нужно на основании массивов с данными собрать массив объектов.

Создайте функцию getData. У неё должно быть два параметра. Первый параметр — массив с ключами. Второй — массив с массивами данных. Ниже приведён пример такого массива.

```
// Массив ключей
['имя', 'любимый цвет', 'любимое блюдо'];

// Массив значений
[
  ['Василий', 'красный', 'борщ'],
  ['Мария'],
  ['Иннокентий', 'жёлтый', 'пельмени', '18', 'Азовское']
];
```

Функция должна собрать объект для каждого массива значений. И каждый из этих объектов должен быть записан в массив данных. Именно этот массив должна вернуть функция getData.

Каждому элементу из массива ключей подходит элемент с таким же индексом в массиве значений. Есть один нюанс: значений может оказаться больше или меньше, чем ключей. Если значений не хватает, то создавать пустой ключ не надо. А если значений больше, то их не нужно включать в объект — для них нет ключей.

Примерно так должен выглядеть результат работы вашей программы:

```
// Готовый массив объектов
[
  {
    'имя': 'Василий',
    'любимый цвет': 'красный',
    'любимое блюдо': 'борщ'
  },
  {
    'имя': 'Мария'
  },
  {
    'имя': 'Иннокентий',
    'любимый цвет': 'жёлтый',
    'любимое блюдо': 'пельмени'
  }
];
```

Работать со сложными массивами, элементы которых тоже массивы, не так трудно, как кажется. Они работают так же, как и привычные вам массивы.

```
let array = ['весна', 'лето', 'осень', 'зима'];
console.log(array[2]);
// Выведет: осень

let anotherArray = [
  ['оранжевый', 'синий', 'красный'],
  ['седан', 'универсал', 'внедорожник'],
  ['весна', 'лето', 'осень', 'зима']
];

console.log(anotherArray[2]);
// Выведет: ["весна", "лето", "осень", "зима"];

console.log(anotherArray[2][3]);
// Выведет: зима
```

Полезные ссылки и материалы

1. https://www.w3schools.com/js/default.asp
2. Объекты: основы (javascript.ru)
3. Свойства объекта, их конфигурация (javascript.ru)
4. Массивы (javascript.ru)