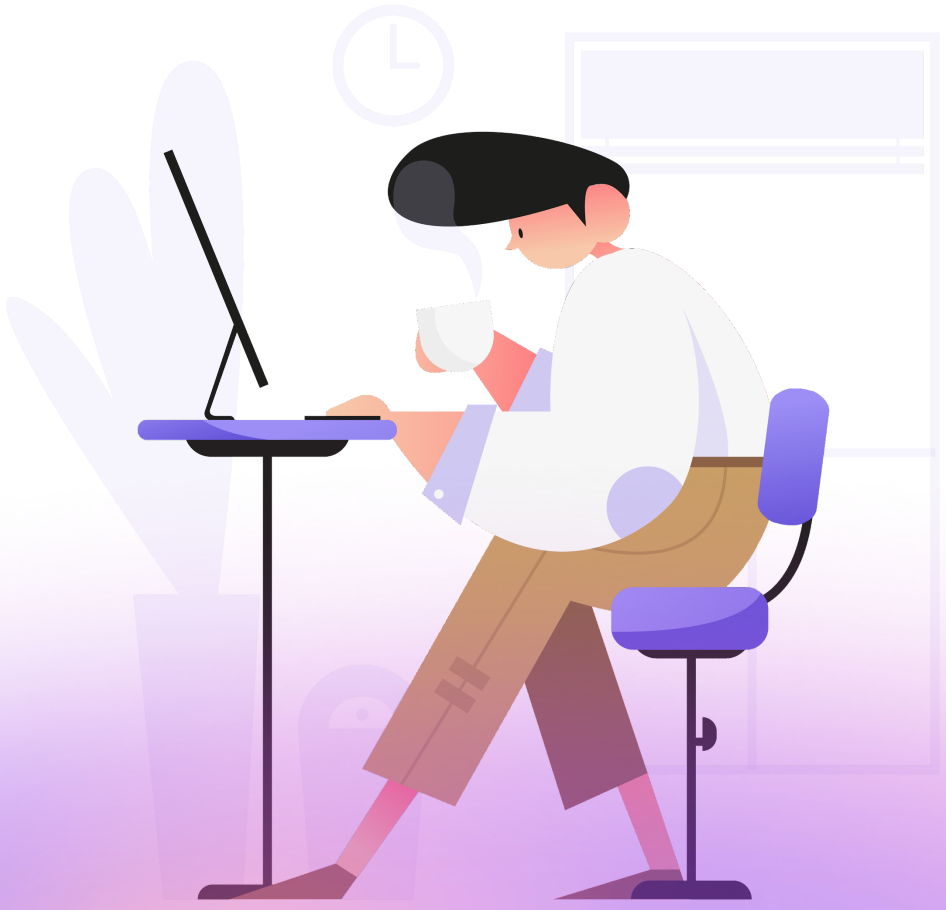
GeekBrains

Основы программирования

Массивы

[В какой версии ПО был написан код. Если актуально]



Иллюстрации для обложки:

https://www.dropbox.com/sh/lzla84g77kbk851/AAASYvRaXE-4idVbxkXaC4LYa/Illustration?dl=0&subfolder_nav_tracking=1

Иллюстрация должна быть отцентрирована, уменьшать в зависимости от размера заголовка

Акцентный для выделений: #6654D9

На этом уроке

1. Узнаем что такое массивы и какие задачи можно решать с помощью них;
2. Научимся создавать массивы, добавлять, удалять и получать элементы массивов.

Оглавление

Введение

На прошлых уроках мы научились работать с данными с помощью переменных. Но в программировании часто бывают задачи, когда нужно обрабатывать большие объёмы или массивы данных. Например, представьте что вам необходимо решить следующую задачу: на вход вам поступают цены на определённый товар в виде списка цен на каждый день за последние 60 дней, от вас требуется определить растёт цена, в целом, или падает. Мы могли бы хранить информацию о цене за каждый день в отдельной переменной, но нам бы понадобилось для этого объявить 60 переменных. Кроме того, что это потребовало бы написания большого количества повторяющегося кода, работать с таким большим числом переменных затруднительно.

Создание массива

В JavaScript для создания пустого или заполненного массива существует несколько способов, мы рассмотрим наиболее простой с точки зрения синтаксиса:

```
let array = [];
```

Пока непонятно что произошло, потому что назначение созданного нами массива неизвестно. Давайте создадим заполненный числами массив:

```
let array = [17, 9, 15, 4, 2, 12, 1, 6];
```

Методы массива

Поскольку массив хранит в себе множество элементов, логично что у нас должна быть возможность узнать количество этих элементов, добавить новые, удалить ненужные и изменить существующие. Для этих операций у массива есть соответствующие свойства и методы, посмотрим их на примере:

```
let array = [17, 9, 15];

console.log('Так массив выглядит изначально: ' + array); // 17,9,15,4,2,12,1,6
console.log('Длина массива ' + array.length); // 8
array.push(100); // Добавляем в конец массива элемент 100
console.log('А теперь он выглядит так: ' + array); // 17,9,15,100
console.log('Длина массива ' + array.length); // 9
array.shift(); // Удаляем первый элемент массива
console.log('Теперь массив выглядит так: ' + array); // 9,15,100
array.unshift(50); // А теперь добавим элемент в начало
console.log('В массиве снова 4 элемента: ' + array); // 50,9,15,100
array.pop(); // Удаляем последний элемент массива
console.log('Снова три элемента: ' + array); // 50,9,15
array[0] = 100; // Изменим первый элемент массива
```

```
array[1] = 200; // Изменим второй элемент массива
array[2] = 300; // Изменим третий элемент массива
console.log(array); // 100,200,300
```

На примере этого кода видно, что у массива есть свойство length, которое содержит информацию о длине массива. Заметьте, что это именно свойство, его не нужно вызывать, как функцию, то есть обращение array.length() приведёт к ошибке. Кроме length мы использовали следующие методы:

- **push** — добавляет новый элемент в конец массива;
- **pop** — обратный **push** метод, удаляет последний элемент массива;
- **unshift** — добавляет новый элемент в начало массива;
- **shift** — обратный **unshift** метод, удаляет первый элемент массива.

Для того, чтобы изменить значение какого-то элемента массива, нужно обратиться к этому значению по индексу элемента, при этом нужно учитывать, что индексация элементов массива начинается не с единицы, а с нуля — это нужно всегда помнить, чтобы избежать логических ошибок в программах. Если *array[0]* — это всегда первый элемент массива, то последний — *array[array.length - 1]*:

```
let array = [1, 2, 3];

console.log(array[0]); // 1
console.log(array[array.length - 1]); // 3
```

Если же мы выйдем за пределы массива, обращаясь к элементу по индексу, ни к чему хорошему это не приведёт, как в этом случае, когда мы попытались получить первый элемент массива, хотя такого элемента нет:

```
let array = [];

console.log(array[0]); // undefined
```

undefined — это ещё один тип в языке JavaScript, он обозначает, что значение не задано (undefined с английского так и переводится — «не задано»).

Избегать таких ситуации можно с помощью условия:

```
let array = [];

if (array.length) {
  console.log(array[0]);
} else {
  console.log('Массив пустой');
}
```

До сих пор мы хранили в массиве только числа, но на самом деле там можно хранить любые данные:

```
let array = ['Василий', 'съел', 5, 'яблок', 'это', true, 'или', false, '?'];

console.log(array); // Василий,съел,5,яблок,это,true,или,false,?
```

Самое время узнать ещё о нескольких полезных методах массива:

```
let array = ['Василий', 'съел', 5, 'яблок', 'это', true, 'или', false, '?'];

console.log(array.join(' ')); // Василий съел 5 яблок это true или false ?
```

Метод **join** «склеивает» все элементы массива в одну строку, а в качестве разделителя использует переданный в него аргумент.

Метод **reverse** зеркально изменяет порядок элементов в массиве:

```
let array = ['Василий', 'съел', 5, 'яблок', 'это', true, 'или', false, '?'];

console.log(array.reverse().join(' ')); // ? false или true это яблок 5 съел Василий
```

Заметьте, что **reverse** и **join** можно вызвать последовательно через точку. Это происходит потому, что **reverse** возвращает этот же самый массив, так что мы можем обратиться к нему как обычно.

Теперь у нас достаточно знаний о массивах, чтобы перейти к решению практической задачи.

Нахождение максимального числа из ряда

На втором уроке мы решали задачу нахождения максимального числа, но наше решение предусматривало ввод пользователем только двух чисел:

```
let a = prompt('Введите число a');
let b = prompt('Введите число b');

if (a > b) {
  console.log('Максимальное число: a');
} else {
  console.log('Максимальное число: b');
}
```

Что если бы нам пришлось найти максимальное из трёх чисел? Попробуем решить эту задачу:

```
let a = prompt('Введите число a');
let b = prompt('Введите число b');
let c = prompt('Введите число c');

if (a > b && a > c) {
  console.log('Максимальное число: a');
} else if (b > a && b > c) {
  console.log('Максимальное число: b');
} else if (c > a && c > b) {
  console.log('Максимальное число: c');
}
```

В новом коде можно заметить оператор **&&**. Это логический оператор **И**, а выражение if (a > b && a > c) можно перевести на русский язык как «если *a* больше *b* и *a* больше *c*». В разделе дополнительных материалов есть ссылка на подробное объяснение логический операторов в JavaScript, в рамках этого курса логические операторы рассматриваться не будут.

Нетрудно представить как разрастётся кодовая база, как усложнится код, если мы с таким подходом будем находить максимальное из 10 чисел, введённых пользователем. Решить эту проблему нам поможет массив.

```
let userInput = prompt('Введите числа через запятую');
let array = userInput.split(',');
let numbers = array.map(
  function (element) {
      return parseInt(element);
  }
);
let maxNumber = 0;

for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] > maxNumber) {
      maxNumber = numbers[i];
  }
}

console.log('Максимальное число ' + maxNumber);
```

На второй строке мы встречаем незнакомый вызов *userInput.split(',')*, разберёмся для чего он нужен. Но сперва нам надо разобраться с тем, какое значение возвращает prompt. Несмотря на то, что мы не первый раз используем prompt для получения чисел от пользователя, в действительности он возвращает строку, это легко проверить с помощью оператора **typeof**, который возвращает тип объекта:

```
let userInput = prompt('Введите число');

console.log(typeof userInput); // string
console.log(typeof 'какая-то строка'); // string
console.log(typeof 10); // number
console.log(typeof false); // boolean
```

Но до этого мы производили операции вычитания и сравнения полученных от пользователя значений, так, как будто это были числа, а не строки, и всё работало, почему?

```
let a = '5';
let b = '3';

console.log(a - b); // 2
```

Неявное приведение типов

Дело в том, что интерпретатор JavaScript, когда видит операторы, свойственные для чисел, пытается привести операнды к числовому значению автоматически, поэтому в нашем случае всё работало корректно, но надеяться на такое поведение JavaScript не стоит, и вот почему:

```
let a = '5';
let b = '3';

console.log(a + b); // 53
```

В данном случае вместо 8 мы получили 53, а всё потому, что JavaScript в случае оператора сложения (**+**) вместо приведения операндов к числовым значениям и сложения этих значений, просто склеил две строки ‘5’ и ‘3’, что в результате дало строку 53. То есть, мы получили не только не ожидаемый нами результат, но и не ожидаемый тип (строковый вместо числового):

```
let a = '5';
let b = '3';

console.log(typeof (a - b)); // number
console.log(typeof (a + b)); // string
```

Поэтому прежде чем использовать полученные от пользователя числовые значения, лучше их привести к типу number, для этого можно воспользоваться одним из следующих способов:

```
let a = +'5';
let b = Number('5');
let c = parseInt('5');
```

```
console.log(typeof a); // number
console.log(typeof b); // number
console.log(typeof c); // number
```

Разбиение строки с использованием split

Метод **split** доступен для строкового типа и выполняет он операцию, обратную методу **join** у массивов: разбивает строку на элементы и создаёт из них массив, при этом в качестве аргумента передаётся «разделитель», по которому произойдёт разбивка строки на элементы:

```
let array = ['Василий', 'съел', 5, 'яблок', 'это', true, 'или', false, '?'];
let str = array.join(' '); // Склеиваем элементы массива в строку
let newArray = str.split(' '); // Обратно создаём массив
```

Вернёмся к коду нахождения максимального числа и строке, на которой мы остановились:

```
let userInput = prompt('Введите числа через запятую');
let array = userInput.split(',');
let numbers = array.map(
  function (element) {
      return parseInt(element);
  }
);
```

Мы видим, что после получения от пользователя строки, которая состоит из чисел, перечисленных через запятую, происходит формирование массива. Поскольку метод **split** формирует массив из строк, нам для дальнейшей работы было бы удобно преобразовать все эти строки в массиве в числовой тип, что и происходит на этом участке кода:

```
let numbers = array.map(
  function (element) {
      return parseInt(element);
  }
);
```

У массива вызывается метод **map**, который создаёт новый массив из результатов выполнения переданной в этот метод функции. То есть, если в массиве 3 элемента, функция будет вызвана последовательно три раза, при этом в функцию в качестве аргумента будет передаваться обрабатываемый элемент массива.

Давайте рассмотрим несколько примеров, чтобы лучше понять как это работает:

```
let array = ['Василий', 'Пётр', 'Валерий'];
let capitalized = array.map(
  function(name) {
      return name.toUpperCase();
  }
);
let ordered = array.map(
  function(name, index) {
      return name + ' ' + (index + 1);
  }
)
let zeros = array.map(
  function() {
      return 0;
  }
);

console.log(capitalized); // ВАСИЛИЙ,ПЁТР,ВАЛЕРИЙ
console.log(ordered); // Василий 1,Пётр 2,Валерий 3
console.log(zeros); // 0,0,0
```

В первом случае с помощью метода toUpperCase у строки мы перевели все буквы во всех строках массива в верхний регистр. Во втором случае мы использовали ещё одну возможность **map**, добавив к именам порядковые номера, используя индекс текущего элемента, передаваемого вторым аргументом в нашу функцию. В третьем случае, мы просто всегда возвращали 0 и получили массив заполненный нулями.

Узнав как работает метод **map** у массивов, и как преобразовать строковый тип в числовой, мы можем понять как работает весь алгоритм нахождения максимального числа из множества, в этом коде не осталось белых пятен:

```
let userInput = prompt('Введите числа через запятую');
let array = userInput.split(',');
let numbers = array.map(
  function (element) {
      return parseInt(element);
  }
);
let maxNumber = 0;

for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] > maxNumber) {
      maxNumber = numbers[i];
  }
}

console.log('Максимальное число ' + maxNumber);
```

В комментарии нуждается только использование уже знакомого нам цикла **for**, здесь он используется для того, чтобы перебрать все возможные индексы массива. И действительно, если *numbers.length* равно, например, 3, то итерации будут выполнены для i == 0, i == 1 и i == 2. А в случае, если массив пустой (т.е. Его длина равно 0), а такую ситуацию тоже нужно рассматривать, цикл не выполнится ни разу, так как условие 0 < 0 не будет выполнено.

На каждой итерации мы проверяем является ли текущий элемент массива больше *maxNumber*, который мы нашли до этого, если является, то обновляем значение *maxNumber*.

Осталась лишь одна проблема, если пользователь ввёл только отрицательные числа, программа выдаст результат 0, потому что ни одно из чисел не больше 0, а это значение мы присваиваем maxNumber изначально. Исправить эту ситуацию можно следующим образом:

```
let maxNumber = array[0];

for (let i = 1; i < numbers.length; i++) {
  if (numbers[i] > maxNumber) {
       maxNumber = numbers[i];
  }
}
```

Мы изначально присваиваем переменной *maxNumber* значение первого элемента массива, а цикл **for** начали не с нулевого индекса, а с первого. В случае, если массив будет пустым, ответ будет undefined, что в нашем случае корректно, потому что в пустом списке не может быть максимального значения.

График изменения цены

Решим задачу отображения графика изменения цены на какой-то товар или ценную бумагу. Поскольку график представляет собой ломаную линию, нам понадобится функция рисования линии. И мы добавили такую функцию в тренажёр.

Не забывайте, что все функции для работы с графикой, представленные в этих материалах, не являются стандартными для JavaScript, они используются только в рамках этого курса для ознакомления с основами программирования графики.

Функция drawLine принимает следующие аргументы:

- xFrom — X-координата начальной точки;
- yFrom — Y-координата начальной точки;
- xTo — X-координата конечной точки;
- yTo — Y-координата конечной точки;
- color — цвет линии (необязательный параметр);
- width — толщина линии (необязательный параметр).

Например, чтобы нарисовать линию от левого верхнего угла холста к центру, нужно вызвать drawLine со следующими аргументами:

```
drawLine(0, 0, 100, 100);
```

Теперь мы знакомы со всем необходимым инструментарием, чтобы нарисовать график:

```
let prices = [100, 120, 90, 100, 130, 115, 210, 230, 60, 80, 90, 100, 103, 107, 114,
150];
let prev = [0, 200];

for (let i = 0; i < prices.length; i++) {
  let current = [i * 10, 200 - prices[i]];
  drawLine(prev[0], prev[1], current[0], current[1]);
  prev = current;
}
```

На каждой итерации цикла мы рисуем линию от предыдущей точки (*prev*) к следующей (*current*). При этом по оси X мы каждый раз делаем приращение на 10, а значение по оси Y вычисляем по формуле *200 - [значение цены на текущий день]*, где 200 — это максимальная высота нашего графика. Отнимать от максимальной высоты нужно потому, что y == 0 — это самая вершина области рисования, а y == 200 находится на 200 точек ниже, то есть, если бы мы вместо *200 - prices[i]* указывали просто *prices[i]*, то график получился бы перевёрнутым и рос сверху вниз.

Обратите внимание что мы используем массивы для хранения координат предыдущей и следующей точки, это удобнее, чем создавать отдельные переменные для каждой координаты.

Остаётся одна проблема, в ряде цен есть значения больше 200, из-за этого график не влезает по высоте в область рисования:

Для того, чтобы решить эту проблему нам нужно вычислить максимальное значение цены из всего ряда цен. Затем, при вычислении координаты Y необходимо текущее значение цены делить на максимальное найденное значение и умножать на высоту графика, который мы хотим нарисовать. Алгоритм нахождения максимального значения из ряда чисел у нас уже есть, для удобства вынесем его в отдельную функцию и обновим наш код:

```
function getMaxFromArray(numbers) {
  let maxNumber = 0;
  for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] > maxNumber) {
      maxNumber = numbers[i];
    }
  }
  return maxNumber
}

let prices = [100, 120, 90, 100, 130, 115, 210, 230, 60, 80, 90, 100, 103, 107, 114,
150];
let max = getMaxFromArray(prices);
let prev = [0, 200];

for (let i = 0; i < prices.length; i++) {
  let current = [i * 10, 200 - prices[i] / max * 200];
  drawLine(prev[0], prev[1], current[0], current[1]);
  prev = current;
}
```

Как видно из этого решения, массивы, как и любой другой тип, могут быть переданы в качестве аргумента вызываемой функции. Чтобы график смотрелся интереснее, наполним массив цен случайными числами, функцию генерации случайных чисел возьмём из прошлого урока:

```
function getMaxFromArray(numbers) {
  let maxNumber = 0;
  for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] > maxNumber) {
      maxNumber = numbers[i];
    }
  }
  return maxNumber
}
function generateRandomNumber(to) {
  return Math.round(Math.random() * to);
}

let prices = [];
for (let i = 0; i < 60; i++) {
  prices.push(generateRandomNumber(300));
}

let max = getMaxFromArray(prices);
let prev = [0, 200];

for (let i = 0; i < prices.length; i++) {
  let current = [i * 10, 200 - prices[i] / max * 200];
  drawLine(prev[0], prev[1], current[0], current[1]);
  prev = current;
```

```
}
```

Домашнее задание

Реализуйте калькулятор общей стоимости покупок: пользователь вводит суммы покупок из чека через запятую, а программа выдаёт результат «Сумма ваших покупок n рублей».

Вынесите алгоритм рисования графика изменения цены из урока в отдельную функцию, принимающую следующие аргументы:

- prices — массив с ценами;
- height — высота графика (чтобы вместо фиксированных 200 можно было указать любую высоту).

Дополнительно

Усовершенствуйте функцию рисования графика, добавив в неё ещё один аргумент:

- width — ширина графика, чтобы шаг по оси X не был фиксированным (сейчас это 10), а вычислялся по формуле *[количество дней] / [ширина графика]*;

Нарисуйте сетку под графиком, чтобы его удобнее было читать. Для рисования сетки в функцию drawLine можно в качестве аргумента color передавать 'gray' или любой другой, например сгенерированный с помощью сервиса https://www.w3schools.com/colors/colors_picker.asp.

Глоссарий

- Массив — структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона;
- Неявное приведение типов — автоматическое приведение одного типа к другому, которое выполняется транслятором (компилятором или интерпретатором) по правилам, описанным в стандарте языка.

Дополнительные материалы

- Документация по массивам в JavaScript — https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array;

Используемые источники

- https://wikipedia.org