Лабораторная работа №2. Методы примитивов и массивов

Цель работы: освоить основные принципы работы с функциями примитивных типов данных (строк и чисел) и массивов.

Краткие теоретические сведения по теме лабораторной работы:

Одной из часто используемых операций при работе с числами является округление. Есть несколько функций (методов чисел), выполняющих округление:

Math.floor – Округление в меньшую сторону: **3.1** становится **3**, а **–1.1** – **–2**.

Math.ceil – Округление в большую сторону: **3.1** становится **4**, а **–1.1** – **–1**.

Math.round – Округление до ближайшего целого: **3.1** становится **3**, **3.6** – **4**, а **–1.1** – **–1**

Math.trunc (не поддерживается в IE) – Производит удаление дробной части без округления: **3.1** становится **3**, а **–1.1** – **–1**.

Функция **Math.random()** возвращает случайное дробное число в диапазоне [0,1). Для генерации случайного целого числа в диапазоне от **min** до **max** (включительно) можно создать функцию **randomInteger(min, max)**:

```
function randomInteger(min, max) {
  let rand = min + Math.random() * (max + 1 - min);
  return Math.floor(rand);
}

alert( randomInteger(1, 3) );
```

Строку можно создать с помощью одинарных, двойных либо обратных кавычек:

```
let single = 'single-quoted';
let double = "double-quoted";

let backticks = `backticks`;
```

Одинарные и двойные кавычки работают одинаково. Обратные кавычки позволяют вставлять в строку произвольные выражения за счет использования **${…}**:

```
function sum(a, b) {
  return a + b;
}

alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
```

Свойство **length** содержит длину строки:

```
alert( `My\n`.length ); // 3
```

Отметим, что **\n** – это один спецсимвол, поэтому длина строки равна 3.
Получить символ, который занимает позицию **pos**, можно с помощью квадратных скобок: **[pos]**.
Также можно использовать метод **charAt**: **str.charAt(pos)**. Первый символ занимает нулевую позицию:

```
let str = `Hello`;

// получаем первый символ
alert( str[0] ); // H
alert( str.charAt(0) ); // H

// получаем последний символ
alert( str[str.length - 1] ); // o
```

Можно перебрать строку посимвольно, используя **for...of**:

```
for (let char of "Hello") {
  alert(char); // H,e,l,l,o (char – сначала "H", потом "e", потом "l" и т. д.)
}
```

Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине строки и заменить его:

```
let str = 'Hi';

str[0] = 'h'; // ошибка
alert( str[0] ); // не работает
```

Можно создать новую строку и записать её в ту же самую переменную вместо старой:

```
let str = 'Hi';

str = 'h' + str[1]; // заменяем строку

alert( str ); // hi
```

Методы **toLowerCase()** и **toUpperCase()** меняют регистр символов:

```
alert( 'Interface'.toUpperCase() ); // INTERFACE
alert( 'Interface'.toLowerCase() ); // interface
```

Если нужно перевести в нижний регистр определенный символ строки, необходимо обратиться к нему (например, через квадратные скобки) и вызвать метод **toLowerCase()**:

```
alert( 'Interface'[0].toLowerCase() ); // 'i'
```

Метод **slice(start, [end])** возвращает часть строки от **start** (включительно) до **end** (не включительно).

```
let str = "stringify";

alert( str.slice(0, 5) );
// 'strin', символы от 0 до 5 (не включая 5)
alert( str.slice(0, 1) )
// 's', от 0 до 1, не включая 1, т. е. один символ на позиции 0
```

Если аргумент **end** отсутствует, **slice** возвращает символы до конца строки:

```
let str = "stringify";
alert( str.slice(2) ); // ringify, с позиции 2 и до конца
```

Также для **start/end** можно задавать отрицательные значения. Это означает, что позиция определена как заданное количество символов с конца строки:

```
let str = "stringify";

// начинаем с позиции 4 справа, а заканчиваем на позиции 1 справа
alert( str.slice(-4, -1) ); // gif
```

Существует 2 варианта синтаксиса для создания пустого массива:

```
let arr = new Array();
let arr = [];
```

Чаще используется второй вариант синтаксиса. В квадратных скобках можно указать начальные значения элементов.

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

Элементы массива нумеруются, начиная с нуля. Можно получить элемент, указав его номер в квадратных скобках:

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

alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива
```

В отличие от строк, заменить элемент массива возможно:

```
fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]
```

Можно также добавить новый элемент к существующему массиву:

```
fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]
```

Общее число элементов массива содержится в его свойстве **length**:

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

alert( fruits.length ); // 3
```

Метод **pop()** – удаляет последний элемент из массива и возвращает его:

```
let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удаляем "Груша" и выводим его

alert( fruits ); // Яблоко, Апельсин
```

Метод **push() –** добавляет элемент в конец массива:

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

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша
```

Метод **shift()** – удаляет из массива первый элемент и возвращает его:

```
let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удаляем Яблоко и выводим его

alert( fruits ); // Апельсин, Груша
```

Метод **unshift()** – добавляет элемент в начало массива:

```
let fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша
```

Классический способ перебора элементов массива осуществляется через цикл **for** по цифровым индексам:

```
let arr = ["Яблоко", "Апельсин", "Груша"];

for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

Для массивов возможен и другой вариант цикла – **for...of**:

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

// проходит по значениям
for (let fruit of fruits) {
  alert( fruit );
}
```

Массивы могут содержать элементы, которые являются массивами. Это используется для создания многомерных массивов, например, для хранения матриц:

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

alert( matrix[1][1] ); // 5, центральный элемент
```

Метод **arr.concat()** создаёт новый массив, в который копирует данные из других массивов и дополнительные значения. Его синтаксис:

```
arr.concat(arg1, arg2...)
```

Он принимает любое количество аргументов, которые могут быть как массивами, так и простыми значениями. В результате мы получаем новый массив, включающий в себя элементы из **arr**, а также **arg1**, **arg2** и т.д. Если аргумент **argN** – массив, то все его элементы копируются. Иначе скопируется сам аргумент:

```
let arr = [1, 2];

// создать массив из: arr и [3,4]
alert( arr.concat([3, 4]) ); // 1,2,3,4

// создать массив из: arr и [3,4] и [5,6]
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// создать массив из: arr и [3,4], потом добавить значения 5 и 6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

arr.sort() сортирует массив на месте, меняя в нём порядок элементов. По умолчанию элементы сортируются как ***строки***. Для строк применяется лексикографический порядок:

```
let arr = [ 1, 2, 15 ];

// метод сортирует содержимое arr
arr.sort();

alert( arr );  // 1, 15, 2
```

Чтобы использовать наш собственный порядок сортировки, нам нужно предоставить функцию в качестве аргумента **arr.sort()**. Например, для сортировки элементов как чисел передадим функцию **compareNumeric(a,b)**:

```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a == b) return 0;
  if (a < b) return -1;
}

let arr = [ 1, 2, 15 ];

arr.sort(compareNumeric);

alert(arr);  // 1, 2, 15
```

Метод **arr.reverse()** меняет порядок элементов в **arr** на обратный:

```
let arr = [1, 2, 3, 4, 5];
arr.reverse();

alert( arr ); // 5,4,3,2,1
```

Метод **str.split(delim)** «разбивает» строку на массив по заданному разделителю **delim**. В примере ниже таким разделителем является строка из запятой и пробела:

```
let names = 'Вася, Петя, Маша';

let arr = names.split(', ');

for (let name of arr) {
  alert( `Сообщение получат: ${name}.` );
// Сообщение получат: Вася (и другие имена)
}
```

Вызов **split(s)** с пустым аргументом **s** разбил бы строку на массив букв:

```
let str = "тест";

alert( str.split('') ); // т,е,с,т
```

arr.join(glue) выполняет операцию, противоположную **split(delim)**. Метод создаёт строку из элементов **arr**, вставляя **glue** между ними:

```
let arr = ['Вася', 'Петя', 'Маша'];

let str = arr.join(';'); // объединить массив в строку через ;

alert( str ); // Вася;Петя;Маша
```

Задания для самостоятельной работы:

1. Создать функцию, принимающую в качестве параметров 2 массива (длины массивов могут отличаться) и возвращающую новый массив, элементы которого равны сумме соответствующих элементов двух передаваемых в функцию массивов.

   Пример работы:
   arr_A = [1, 3, 5, 7, 9, 11];
   arr_B = [2, 4, 6, 8];
   func(arr_A, arr_B); // [3, 7, 11, 15, 9, 11]

2. Создать функцию, принимающую в качестве параметра строку и меняющую регистр каждого элемента этой строки на противоположный.

   Пример работы:
   func('HeLlO wOrLd') // 'hElLo WoRlD'

3. Создать функцию, которая принимает строку в качестве параметра и переводит первый символ каждого слова этой строки в верхний регистр.

   Пример работы:
   func('hello world how are you') // ' Hello World How Are You '

4. Создать функцию, которая принимает в качестве параметра строку и меняет порядок букв в каждом слове строки на обратный.

   Пример работы:
   func('hello world how are you') // olleh dlrow woh era uoy

5. Создать функцию **func(num, quantity, pm)**, которая добавляет к числу (параметр **num**):
   1) нули в таком количестве, чтобы число содержало **quantity** цифр;
   2) знак (параметр **pm**). Параметр **pm** является необязательным.

   Функция должна возвращать получившееся число в виде строки. Параметр **quantity** должен быть больше количества цифр в числе **num**.

   Пример работы:
   func(12345, 7, +); // '+0012345'

6. Создать функцию, которая принимает в качестве параметра положительное число и возвращает количество операций перемножения цифр этого числа, необходимых для получения однозначного числа (т.е. состоящего из одной цифры).

   Пример работы:
   func(39); // 3 т.к. 3*9 = 27, 2*7 = 14, 1*4=4, 4 – однозначное число
   func(999); // 4 т.к. 9*9*9 = 729, 7*2*9 = 126, 1*2*6 = 12, 1*2 = 2, 2 – однозначное число
   func(4); // 0 т.к. 4 уже является однозначным числом

7. Создать первую функцию, которая принимает в качестве параметра число и возвращает квадратную матрицу (кол-во строк = кол-ву столбцов = **num**), каждый элемент которой – случайное целое число в диапазоне **[0, 9]**.

   Создать вторую функцию, принимающую в качестве параметров 2 квадратные матрицы и возвращающую результат перемножения этих матриц.

   С помощью первой функции создать 2 квадратные матрицы **mat_A** и **mat_B** размерности 3×3, передать эти матрицы во вторую функцию и вывести результат их перемножения на экран.

   Пример работы:
   mat_A = first_Func(3); // создали первую матрицу 3×3
   mat_B = first_Func(3); // создали вторую матрицу 3×3
   console.log(second_Func(mat_A, mat_B)) // перемножили созданные матрицы

8. Создать функцию, подсчитывающую количество одинаковых элементов в массиве. Функция должна возвращать объект, в котором ключами являются элементы массива, а значениями – количество данных элементов в массиве

   Пример работы:
   func(['a' , 'a', 'a', 'b', 'b', 'c']); // {a : 3, b: 2, c: 1}

9. Создать функцию, которая принимает в качестве параметра массив и возвращает другой массив, содержащий только уникальные значения исходного массива. При решении нельзя использовать коллекцию **Set**.

   Пример работы:
   func([1, 1, 1, 2, 2, 5, 5, 8, 8]); // [1, 2, 5, 8]