МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

к практическим занятиям студентов
09.02.07 Информационные системы и программирование
Профессиональный цикл

Разработали:
преподаватели:
Кротова О.В.
Сушкова Ю.А.

Севастополь
2025

МДК 03.01 Проектирование и разработка веб-приложений

Практическая работа № 1

Инструкции языка JavaScript.

Задания:

Изучите материалы лекций по javascript. Отработайте указанные там примеры и выполните следующие действия:

1. В файле index1.html организуйте между двумя абзацами вывод

сообщения в диалоговом окне, задав необходимые команды JavaScript во внешнем файле.

Для этого:

- создайте новый текстовый файл,
- поместите в него код JavaScript,
- сохраните файл с именем main.js следующим образом: укажите тип файла “Все файлы”, кодировку “UTF-8”,
- добавьте ссылку на внешний скриптовый файл из рабочего HTML документа.

2. Добавьте в документ код JavaScript так, чтобы в диалоговом окне появлялось поле с надписью "Введите сюда своё ФИО" и со значением по умолчанию в поле "Введите ФИО". Для этого используйте метод promt(...) объекта window. Для хранения введенного значения заведите новую переменную.

3. Организуйте вывод введенного значения имени в окно браузера в виде: "Ваше ФИО <…..>".

4. Дополните код, чтобы в новом диалоговом окне появилось надпись "Начать заново? " При положительном ответе появлялось диалоговое окно: "Стоп ввод? ", при отказе – "Ввод завершен!". Используйте для написания методы alert(…) и confirm(…) объекта window.

ИИ для подготовки:
https://developer.mozilla.org/ru/docs/Web/JavaScript

Практическая работа № 2

Тип данных массив и его использование в JavaScript.

Обработка массивов

Для хранения набора данных в языке JavaScript предназначены массивы. Массивы в JavaScript представлены объектом Array. Объект Array предоставляет ряд свойств и методов, с помощью которых мы можем управлять массивом и его элементами.

Создание массива

Можно создать пустой массив, используя квадратные скобки или конструктор Array:

```
const lesson = new Array();
const lesson = [];
console.log(lesson); // Array[0]
console.log(lesson); // Array[0]
```

Можно сразу же инициализировать массив некоторым количеством элементов:

```
const lesson = new Array("БД", "СП", "ИСР");
const lesson = ["ОС", "ИСС", "ООП"];

console.log(lesson); // ["БД", "СП", "ИСР"]
console.log(lesson); // ["ОС", "ИСС", "ООП"]
```

Еще один способ инициализации массивов представляет метод Array.of() - он принимает элементы и инициизирует ими массив:

```
const lesson = Array.of("БД", "ИТ", "ОС");
console.log(lesson); // ["БД", "ИТ", "ОС"]
```

Можно определить массив и по ходу определять в него новые элементы:

```
const lesson = [];
lesson[1] = "БД";
lesson[2] = "ООП";
console.log(lesson[1]); // "БД"
console.log(lesson[0]); // undefined
```

При этом не важно, что по умолчанию массив создается с нулевой длиной. С помощью индексов мы можем подставить на конкретный индекс в массиве тот или иной элемент.

Array.from

И еще один способ представляет функция Array.from(). Она имеет много вариантов, рассмотрим самые распространенные:

```
Array.from(arrayLike)
Array.from(arrayLike, function mapFn(element) { ... })
Array.from(arrayLike, function mapFn(element, index) { ... })
```

В качестве первого параметра - arrayLike функция принимает некий объект, который, условно говоря, "похож на массив", то есть может быть представлен в виде набора элементов. Это может быть и другой массив, это может быть и строка, которая по сути предоставляет набор символов. Вообщем какой-то набор элементов, который можно преобразовать в массив. Кроме того, это может и некий объект, в котором определено свойство length. Например:

```
const array = Array.from("Hello");
console.log(array); // ["H", "e", "l", "l", "o"]
```
В данном случае в функцию передается строка и возвращается массив, каждый элемент которого предоставляет один из символов этой строки.

В качестве второго параметра передается функция преобразования, которая через первый параметр получает текущий элемент набора и возвращает некоторый результат его трансформации. Например:

```
const numbers = [1, 2, 3, 4];
const array = Array.from(numbers, n => n * n);
console.log(array); // [1, 4, 9, 16]
```
В данном случае в функцию Array.from() передается массив чисел. Второй параметр - функция (в данном случае в ее роли выступает лямбда-выражение) запускается для каждого числа из этого массива и получает это число через параметр n. В самом лямбда-выражении возвращаем квадрат этого числа. В итоге Array.from() возвратит новый массив, в котором будут квадраты чисел из массива numbers.

И еще одна версия функции Array.from() в качестве второго параметра принимает функцию преобразования, в которую кроме элемента из перебираемого набора передается и индекс этого элемента: Array.from(arrayLike, function mapFn(element, index) { ... }). Используем эту версию и передадим в функцию объект со свойством length:

```
const array = Array.from({length:3}, (element, index) => {
   console.log(element);   // undefined
   return index;
});
console.log(array); // [0, 1, 2]
```
Здесь в функцию передается объект, у которого свойство length равно 3. Для функции Array.from это будет сигналом, в возвращаемом массиве должно быть три элемента. При этом неважно, что функция преобразования из второго параметра принимает элемент набора (параметр element) - в данном случае он будет всегда undefined, тем не менее значение length:3 будет указателем, что возвращаемый массив будет иметь три элемента с соответственно индексами от 0 до 2. И через второй параметр функции преобразования - параметр index мы можем и получить текущий индекс элемента.

Тем не менее мы можем передать объект, где в качестве названий свойств применяются индексы. В этом случае объект превратится в массивоподобный объект, который можно перебрать:

```
const array = Array.from({length:3, "0": "БД", "1": "ОС", "2": "ИТ"}, (element) => {
   console.log(element);
   return element;
});
console.log(array); // ["БД", "ОС", "ИТ"]
```
length

Чтобы узнать длину массива, используется свойство length:

```
const fruit = [];
fruit[0] = "яблоки";
```

```
fruit[1] = "груши";
fruit[2] = "сливы";

console.log("В массиве fruit ", fruit.length, " элемента");
for(let i=0; i < fruit.length; i++)
   console.log(fruit[i]);
```
По факту длиной массива будет индекс последнего элемента с добавлением единицы. Например:

```
const lesson = []; // в массиве 0 элементов
lesson[0] = "БД";
lesson[1] = "ООП";
lesson[4] = "ОС";
for(let i=0; i<lesson.length;i++)
   console.log(lesson[i]);
```
Вывод браузера:

```
БД
ООП
undefined
undefined
ОС
```
Несмотря на то, что для индексов 2 и 3 мы не добавляли элементов, но длиной массива в данном случае будет число 5. Просто элементы с индексами 2 и 3 будут иметь значение undefined.

Язык JavaScript предоставлет богатые возможности для работы с массивами, которые реализуются с помощью методов объекта Array. Рассмотрим применение этих методов

Копирование массива. slice()
Копирование массива может быть поверхностным или неглубоким (shallow copy) и глубоким (deep copy).

При неглубоком копировании достаточно присвоить переменной значение другой переменной, которая хранит массив:

```
const lesson = ["БД", "ОС", "СП"];
console.log(lesson);     //  ["БД", "ОС", "СП"]
const lesson = lesson;       //  неглубокое копирование

lesson[1] = "ДиС";     //  изменяем второй элемент
console.log(lesson);     //  ["БД", "ДиС", "СП"]
```
В данном случае переменная lesson после копирования будет указывать на тот же массив, что и переменная lesson. Поэтому при изменении элементов в lesson, изменятся элементы и в lesson, так как фактически это один и тот же массив.

Такое поведение не всегда является желательным. Например, мы хотим, чтобы после копирования переменные указывали на отдельные массивы. И в этом случае можно использовать глубокое копирование с помощью метода slice():

```
const lesson = ["БД", "ОС", "СП"];
```

```
console.log(lesson);            //  ["БД", "ОС", "СП"]
const lesson = lesson.slice();      //  глубокое копирование

lesson[1] = "ДиС";              //  изменяем второй элемент
console.log(lesson);            //  ["БД", "ОС", "СП"]
console.log(lesson);            //  ["БД", "ДиС", "СП"]
```
В данном случае после копирования переменные будут указывать на разные массивы, и мы сможем изменять их отдельно друг от друга.

Но тут стоит отметить, что то же самое копирование по сути можно выполнить и с помощью spread-оператора ...:

```
const lesson = ["БД", "ОС", "СП"];
console.log(lesson);    //  ["БД", "ОС", "СП"]
const lesson = [...lesson];

lesson[1] = "ДиС";    //  изменяем второй элемент
console.log(lesson);    //  ["БД", "ОС", "СП"]
console.log(lesson);    //  ["БД", "ДиС", "СП"]
```
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.

Например, выберем в новый массив элементы, начиная с 1 индекса по индекс 4 не включая:

```
const lesson = ["БД", "ОС", "СП", "ИСР", "ООП"];
const lesson = lesson.slice(1, 4);
console.log(lesson);        // ["ОС", "СП", "ИСР"]
```
И поскольку индексация массивов начинается с нуля, то в новом массиве окажутся второй, третий и четвертый элемент.

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

```
const m1 = ["БД", "ОС", "СП", "ИСР", "ООП"];
const lesson = m1.slice(2);  // со второго индекса до конца
console.log(lesson);        // ["СП", "ИСР", "ООП"]
```
push()
Метод push() добавляет элемент в конец массива:

```
const lesson = [];
lesson.push("БД");
lesson.push("ОС");
lesson.push("ИТ","ДиС");

console.log("В массиве lesson элементов: ", lesson.length);
console.log(lesson); // ["БД", "ОС", "ИТ", "ДиС"]
```
pop()

Метод pop() удаляет последний элемент из массива:

```
const lesson = ["БД", "ОС", "ИТ", "ДиС"];

const lastLesson = lesson.pop(); // извлекаем из массива последний элемент
console.log(lastLesson );   // ДиС
console.log(lesson);    // ["БД", "ОС", "ИТ"]
```

shift()

Метод shift() извлекает и удаляет первый элемент из массива:

```
const lesson = ["БД", "ОС", "ИТ", "ДиС"];

const firstLesson = lesson.shift(); // извлекаем из массива первый элемент
console.log(firstLesson);   // БД
console.log(lesson);    // ["ОС", "ИТ", "ДиС"]
```

unshift()

Метод unshift() добавляет новый элемент в начало массива:

```
const lesson = ["БД", "ОС", "ИТ"];

lesson.unshift("ИСР");
console.log(lesson);    // ["ИСР", "БД", "ОС", "ИТ"]
```

Удаление элемента по индексу. splice()

Метод splice() удаляет элементы с определенного индекса. Например, удаление элементов с третьего индекса:

```
const lesson = ["БД", "ОС", "СП", "ИСР", "ООП"];
const deleted = lesson.splice(3);
console.log(deleted);       // [ "ИСР", "ООП" ]
console.log(lesson);        // [ "БД", "ОС", "СП" ]
```

Метод splice возвращает удаленные элементы в виде нового массива.

В данном случае удаление идет с начала массива. Если передать отрицательный индекс, то удаление будет производиться с конца массива. Например, удалим последний элемент:

```
const lesson = ["БД", "ОС", "СП", "ИСР", "ООП"];
const deleted = lesson.splice(-1);
console.log(deleted);       // [ "ООП" ]
console.log(lesson);        // ["БД", "ОС", "СП", "ИСР"]
```

Дополнительная версия метода позволяет задать количество элементов для удаления. Например, удалим с первого индекса три элемента:

```
const lesson = ["БД", "ОС", "СП", "ИСР", "ООП"];
const deleted = lesson.splice(1, 3);
console.log(deleted);       // ["ОС", "СП", "ИСР"]
console.log(lesson);        // ["БД", "ООП"]
```

Еще одна версия метода splice позволяет вставить вместо удаляемых элементов новые элементы:

```
const lesson = ["БД", "ОС", "СП", "ИСР", "ООП"];
const deleted = lesson.splice(1, 3, "ТРПО", "ИТ");
```

```
console.log(deleted);       // ["ОС", "СП", "ИСР"]
console.log(lesson);        // ["БД", "ТРПО", "ИТ", "ООП"]
```
В данном случае удаляем три элемента с 1-го индекса и вместо них вставляем два элемента.

concat()
Метод concat() служит для объединения массивов. В качестве результата он возвращает объединенный массив:

```
const m1 = ["БД", "ОС", "ИТ"];
const m2 = ["ИСР", "ООП"];
const lesson = m1.concat(m2);
console.log(lesson);        // ["БД", "ОС", "ИТ", "ИСР", "ООП"]
```
join()
Метод join() объединяет все элементы массива в одну строку, используя определенный разделитель, который передается через параметр:

```
const lesson = ["БД", "ОС", "ИТ"];

const lessonToString = lesson.join("; ");
console.log(lessonToString);        // БД; ОС; ИТ
```
В метод join() передается разделитель между элементами массива. В данном случае в качестве разделителя будет использоваться точка с запятой и пробел ("; ").

sort()
Метод sort() сортирует массив по возрастанию:
```
const lesson = ["БД", "ОС", "ИТ"];

lesson.sort();
console.log(lesson);        // ["ИТ", "ОС", "БД"]
```
Стоит отметить, что по умолчанию метод sort() рассматривает элементы массива как строки и сортирует их в алфавитном порядке. Что может привести к неожиданным результатам, например:

```
const numbers = [200, 15, 5, 35];

numbers.sort();
console.log(numbers);        // [15, 200, 35, 5]
```
Здесь мы хотим отсортировать массив чисел, но результат может нас обескуражить: [15, 200, 35, 5]. В этом случае мы можем настроить метод, передав в него функцию сортировки. Логику функции сортировки мыы определяем сами:

```
const numbers = [200, 15, 5, 35];

numbers.sort( (a, b) =>  a - b);
console.log(numbers);        // [5, 15, 35, 200]
```
Функция сортировки получает два рядом расположенных элемента массива. Она возвращает положительное число, если первый элемент должен находится перед вторым элементом. Если первый элемент должен располагаться после второго, то возвращается отрицательное число. Если элементы равны, возвращается 0.

reverse()

Метод reverse() переворачивает массив задом наперед:

```
const lesson = ["БД", "ОС", "ИТ"];

lesson.reverse();
console.log(lesson);        // ["ИТ", "ОС", "БД"]
```

Поиск индекса элемента

Методы indexOf() и lastIndexOf() возвращают индекс первого и последнего включения элемента в массиве. Например:

```
const lesson = ["БД", "ОС", "ИТ", "БД", "ИСР", "ОС"];

const firstIndex = lesson.indexOf("БД");
const lastIndex = lesson.lastIndexOf("БД");
const otherIndex = lesson.indexOf("ДиС");
console.log(firstIndex); // 0
console.log(lastIndex);  // 3
console.log(otherIndex); // -1
```

firstIndex имеет значение 0, так как первое включение строки "БД" в массиве приходится на индекс 0, а последнее на индекс 3.

Если же элемент отсутствует в массиве, то в этом случае методы indexOf() и lastIndexOf() возвращают значение -1.

Проверка наличия элемента

Метод includes() проверяет, есть ли в массиве значение, переданное в метод через параметр. Если такое значение есть, то метод возвращает true, если значения в массиве нет, то возвращается false. Например:

```
const lesson = ["БД", "ОС", "ИТ", "БД", "ИСР", "ОС"];
console.log(lesson.includes("БД"));    // true - БД есть в массиве
console.log(lesson.includes("ООП"))    // false - ООП нет в массиве
```

В качестве второго параметра метод includes() принимает индекс, с которого надо начинать поиск:

```
const lesson = ["БД", "ОС", "ИТ", "БД", "ИСР", "ОС"];
 console.log(lesson.includes("ИТ", 2)); // true
console.log(lesson.includes("ИТ", 4))  // false
```

В данном случае мы видим, что при поиске со 2-го индекса в массиве есть строка "ИТ", тогда как начиная с 4-го индекса данная строка отсутствует.

Если если этот параметр не передается, то по умолчанию поиск идет с 0-го индекса.

При передаче отрицательного значения поиск идет с конца

```
const lesson = ["БД", "ОС", "ИТ", "БД", "ИСР", "ОС"];
 console.log(lesson.includes("БД", -2)); // false - 2-й индекс с конца
console.log(lesson.includes("БД", -3)) // true - 3-й индекс с конца
```

every()

Метод every() проверяет, все ли элементы соответствуют определенному условию:

```
const numbers = [ 1, -12, 8, -4, 25, 42 ];
```

```
const passed = numbers.every(n => n > 0);
console.log(passed); // false
```
В метод every() в качестве параметра передается функция, которая представляет условие. Эта функция в качестве параметра принимает элемент и возвращает true (если элемент соответствует условию) или false (если не соответствует).

Если хотя бы один элемент не соответствует условию, то метод every() возвращает значение false.
В данном случае условие задается с помощью лямбда-выражения n => n > 0, которое проверяет, больше ли элемент нуля.

some()
Метод some() похож на метод every(), только он проверяет, соответствует ли хотя бы один элемент условию. И в этом случае метод some() возвращает true. Если элементов, соответствующих условию, в массиве нет, то возвращается значение false:

```
const numbers = [ 1, -12, 8, -4, 25, 42 ];
const passed = numbers.some(n => n > 0);
console.log(passed); // true
```
filter()
Метод filter(), как some() и every(), принимает функцию условия. Но при этом возвращает массив тех элементов, которые соответствуют этому условию:

```
const numbers = [ 1, -12, 8, -4, 25, 42 ];
const filteredNumbers = numbers.filter(n => n > 0);
console.log(filteredNumbers); // [1, 8, 25, 42]
```
forEach() и map()
Методы forEach() и map() осуществляют перебор элементов и выполняют с ними определенный операции. Например, используем метод метода forEach() для вычисления квадратов чисел в массиве:

```
const numbers = [ 1, 2, 3, 4, 5, 6];
 numbers.forEach(n =>
    console.log("Квадрат числа", n, "равен", n * n )
);
```
Метод forEach() в качестве параметра принимает функцию, которая имеет один ппараметр - текущий перебираемый элемент массива. А в теле функции над этим элементом можно выполнить различные операции.

Консольный вывод программы:

```
Квадрат числа 1 равен 1
Квадрат числа 2 равен 4
Квадрат числа 3 равен 9
Квадрат числа 4 равен 16
Квадрат числа 5 равен 25
Квадрат числа 6 равен 36
```
Метод map() похож на метод forEach, он также в качестве параметра принимает функцию, с помощью которой выполняются операции над перебираемыми элементами массива, но при этом метод map() возвращает новый массив с результатами операций над элементами массива.

Например, применим метод map к вычислению квадратов чисел массива:

```
const numbers = [ 1, 2, 3, 4, 5, 6];
const squares = numbers.map(n => n * n);
console.log(squares);   //  [1, 4, 9, 16, 25, 36]
```

Функция, которая передается в метод map() получает текущий перебираемый элемент, выполняет над ним операции и возвращает некоторое значение. Это значение затем попадает в результирующий массив squares

Поиск в массиве

Метод find() возвращает первый элемент массива, который соответствует некоторому условию. В качестве параметр метод find принимает функцию условия:

```
const numbers = [1, 2, 3, 5, 8, 13, 21, 34];
// получаем первый элемент, который больше 10
let found = numbers.find(n => n > 10 );
console.log(found); // 13
```

В данном случае получаем первый элемент, который больше 10. Если элемент, соответствующий условию, не найден, то возвращается undefined.

Метод findIndex также принимает функцию условия, только возвращает индекс первого элемента массива, который соответствует этому условию:

```
const numbers = [1, 2, 3, 5, 8, 13, 21, 34];
// получаем индекс первого элемента, который больше 10
let foundIndex = numbers.findIndex(n => n > 10 );
console.log(foundIndex);    // 5
```

Если элемент не найден, то возвращается число -1.

Метод flat и преобразование массива

Метод flat() упрощает массив с учетом указанной вложенности элементов:

```
const lesson = ["БД", "ИТ", ["ИСР", "ООП", ["ОС", "ТРПО"]]];
const flattenLesson = lesson.flat();
console.log(flattenLesson); // ["БД", "ИТ", "ИСР", "ООП", ["ОС", "ТРПО"]]
```

То есть метод flat() фактически из вложенных массивов переводит элементы во внешний массив самого верхнего уровня. Однако мы видим, что элементы массива второго уровня вложенности перешли в массив первого уровня вложенности, но тем не менее по-прежнему находятся во вложенном массиве. Дело в том, что метод flat() по умолчанию применяется только к вложенным массивам первого уровня вложенности. Но мы можем передать в метод уровень вложеноти:

```
const lesson = ["БД", "ИТ", ["ИСР", "ООП", ["ОС", "ТРПО"]]];
const flattenLesson = lesson.flat(2);
console.log(flattenLesson); // ["БД", "ИТ", "ИСР", "ООП", "ОС", "ТРПО"]
```

Если массив содержит вложенные массивы гораздо более глубоких уровней вложенности, или мы даже не знаем, какие уровни вложенности есть в массиве, но мы хотим, чтобы все элементы были преобразованы в один массив, то можно использовать значение Infinity:

```
const lesson = ["БД", "ИТ", ["ИСР", "ООП", ["ОС", "ТРПО"]]];
const flattenLesson = lesson.flat(Infinity);
```

```
console.log(flattenLesson); // ["БД", "ИТ", "ИСР", "ООП", "ОС", "ТРПО"]
```

Задания:

Ознакомиться с теоретическим материалом и выполнить задания

Задание 1

Работа с concat

Даны два массива: **[1, 2, 3]** и **[4, 5, 6]**. Объедините их вместе.

Задание 2

Работа с reverse

Дан массив **[1, 2, 3]**. Сделайте из него массив **[3, 2, 1]**.

Задание 3

Работа с push, unshift

Дан массив **[1, 2, 3]**. Добавьте ему в конец элементы **4, 5, 6**.

Дан массив **[1, 2, 3]**. Добавьте ему в начало элементы **4, 5, 6**.

Задание 4

Работа с shift, pop

Дан массив **['js', 'css', 'jq']**. Выведите на экран **первый** элемент.

Дан массив **['js', 'css', 'jq']**. Выведите на экран **последний** элемент.

Задание 5

Работа с slice

Дан массив **[1, 2, 3, 4, 5]**. С помощью метода **slice** запишите в новый элементы **[1, 2, 3]**.

Дан массив **[1, 2, 3, 4, 5]**. С помощью метода **slice** запишите в новый элементы **[4, 5]**.

Задание 6

Работа с splice

Дан массив **[1, 2, 3, 4, 5]**. С помощью метода **splice** преобразуйте массив в **[1, 4, 5]**.

Дан массив **[1, 2, 3, 4, 5]**. С помощью метода **splice** запишите в новый массив элементы **[2, 3, 4]**.

Дан массив **[1, 2, 3, 4, 5]**. С помощью метода **splice** сделайте из него массив **[1, 2, 3, 'a', 'b', 'c', 4, 5]**.

Дан массив **[1, 2, 3, 4, 5]**. С помощью метода **splice** сделайте из него массив **[1, 'a', 'b', 2, 3, 4, 'c', 5, 'e']**.

Задание 7

Работа с sort

Дан массив **[6, 4, 7, 2, 5]**. Отсортируйте его.

ИИ для подготовки:

https://learn.javascript.ru/array

Практическая работа № 3

Функции языка JavaScript.

Функции представляют собой набор инструкций, которые можно повторно вызывать в различных частях программы по имени функции. В общем случае синтаксис определения функции выглядит следующим образом:

```
function имя_функции(параметры){

    // Инструкции
}
```

Определение функции начинается с ключевого слова function, после которого следует имя функции. Наименование функции подчиняется тем же правилам, что и наименование переменной: оно может содержать только цифры, буквы, символы подчеркивания и доллара ($) и должно начинаться с буквы, символа подчеркивания или доллара.
После имени функции в скобках идет перечисление параметров. Даже если параметров у функции нет, то просто идут пустые скобки. Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

Определим простейшую функцию:

```
function hello(){

    console.log("Hello world ");
}
```

Данная функция называется `hello()`. Она не принимает никаких параметров и все, что она делает, это выводит на консоль браузера строку "Hello World".

Чтобы функция выполнила свою работу, нам надо ее вызвать. Общий синтаксис вызова функции:

```
имя_функции(параметры)
```

При вызове после имени вызываемой функции в скобках указывается список параметров. Если функция не имеет параметров, то указываются пустые скобки.

Например, определим и вызовем простейшую функцию:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>WORLD</title>
```

```
</head>
<body>
<script>
// определение функции
function hello(){

    console.log("Hello world");
}
// вызов функции
hello();
</script>
</body>
</html>
```

В данном случае функция `hello` не принимает параметров, поэтому при ее вызове указываются пустые скобки:

```
hello()
```

Отличительной чертой функций является то, что их можно многократно вызывать в различных местах программы.

Функция в JavaScript может принимать параметры. Параметры представляют способ передачи в функцию данных. Параметры указываются в скобках после названия функции.

Например, определим простейшую функцию, которая принимает один параметр:

```
function print(message){
    console.log(message);
}

print("Hello
JavaScript");
print("Hello WORLD");
print("Function in
JavaScript");
```

Функция `print()` принимает один параметр - message. Поэтому при вызове функции мы можем передать для него значение, например, некоторую строку:

```
print("Hello JavaScript");
```

Передаваемые параметрам значения еще называют аргументами.

При этом в отличие от ряда других языков программирования мы в принципе можем не передавать значения параметрам. Например:

```
function print(message){
    console.log(message);
}
print();
```

Если параметру не передается значение, тогда он будет иметь значение

`undefined`. Если функция принимает несколько параметров, то они

перечисляются через запятую:

```
function sum(a, b){
    const result = a + b;
    console.log(result);
}

sum(2, 6);           // 8
sum(4, 5);           // 9
sum(109, 11);        // 120
```

При вызове функции с несколькими параметрами значения передаются параметрам по позиции. То есть первое значение передается первому параметру, второе значение - второму и так далее.
Например, в вызове:

```
sum(2, 6);
```

Число 2 передается параметру `a`, а число 6 - параметру `b`.

Задание:

№	Вариант
1.	Написать и протестировать функцию, которая параметром будет принимать массив и возвращать сумму его элементов.
2.	Написать и протестировать функцию, которая параметром будет принимать число и возвращать массив его делителей.
3.	Написать и протестировать функцию, которая параметром будет принимать строку и возвращать массив ее символов.
4.	Написать и протестировать функцию, которая параметром будет принимать строку и переворачивать ее символы в обратном порядке.
5.	Написать и протестировать функцию, которая параметром будет принимать строку и делать заглавной первую букву этой строки.
6.	Написать и протестировать функцию, которая параметром будет принимать массив и возвращать количество отрицательных его элементов.
7.	Написать и протестировать функцию, которая параметром будет принимать строку и делать заглавной первую букву каждого слова этой строки.
8.	Написать и протестировать функцию, которая параметром будет принимать массив и возвращать сумму элементов массива больше некоторого числа A, вводимого в текстовое поле формы
9.	Написать и протестировать функцию, заполняющую массив целыми числами от `1` до заданного числа N.
10	Написать и протестировать функцию, которая параметром будет принимать число N и возвращать сумму его цифр
11	Написать и протестировать функцию, у которой параметром будет число N и возвращаемым значение четное оно или нечетное.

12	Написать и протестировать функцию, которая будет возвращать случайный элемент из массива.
13	Написать и протестировать функцию, которая будет возвращать заданный параметром элемент из массива.
14	Написать и протестировать функцию, которая параметром будет принимать число и проверять, простое оно или нет.
15	Написать и протестировать функцию, которая вычисляет сумму квадратов n ее аргументов
16	Даны действительные числа *x*, *y* и *z*. Получить $u = \frac{\max(x, y) + \max(x + y, z)}{(\max(0.5, x + z))^2}$. Использовать функцию поиска максимального значения
17	В ледовом городке построены три снеговика, каждый - из трех ледяных сфер, радиусы нижних сфер соответственно равны R1 = 90см, R2 = 85см, R3 = 75см. Радиус каждой второй сферы в 1,5 раза меньше, чем радиус первой, а радиус третьей – в 2 раза меньше, чем радиус первой. Вычислить, сколько кг льда потребовалось на изготовление снеговиков, если плотность льда 0.9 г/см3. Вычисление объема сферы оформить в виде функции с одним входным параметром (Vсферы = 4/3*π*R3).
18	Написать и протестировать функцию с именем *EXPONENT*, вычисляющую $e^{x+iy} = e^x(\cos y + i \sin y)$, где *x* и *y* — два параметра вещественного типа.
19	Написать и протестировать функцию, позволяющую найти корень *m*-той степени из *x*, применив тождество: $\sqrt[m]{x} = e^{\frac{\ln x}{m}}$
20	Написать и протестировать функцию нахождения координат *(x, y)* точки, делящей отрезок *AB* пополам, если известно, что его концы имеют координаты: $A(x_1, y_1)$, $B(x_2, y_2)$. Применить функцию для трех разных отрезков.
21	Сделайте функцию, которая параметром будет принимать массив и возвращать количество его ненулевых элементов.
22	Сделайте функцию, которая параметром будет принимать массив и возвращать количество его двузначных элементов.
23	Написать и протестировать функцию, которая параметром будет принимать массив и возвращать количество таких элементов массива, первая цифра которых совпадает с последней
24	Написать и протестировать функцию, которая параметром будет принимать строку и удалять все гласные буквы из строки.
25	Написать и протестировать функцию, которая параметром будет принимать строку и удалять повторяющиеся буквы из строки.

ИИ для подготовки: https://developer.mozilla.org/ru/docs/Web/JavaScript

Объект Date. Работа с датами

Объект Date позволяет работать с датами и временем в JavaScript.

Существуют различные способы создания объекта Date. Первый способ заключается в использовании пустого конструктора без параметров:
var currentDate = new Date();
document.write(currentDate);
В этом случае объект будет указывать на текущую дату компьютера.

Второй способ заключается в передаче в конструктор Date количества миллисекунд, которые прошли с начала эпохи Unix, то есть с 1 января 1970 года 00:00:00 GMT:

var myDate = new Date(2359270000000);
document.write(myDate);

Параметром конструктора может быть строка, в которой записана нужная дата:
date1 = new Date(“february 19, 2021, 12:40:00”);

Можно задать список параметров:
Date2 = new Date(2021, 2, 19, 12, 40, 0);

Большинство методов объекта Date вызываются через экземпляр этого объекта. Например:

system.write("Today is: "+date1.toLocalString());

Чаще всего объект Date используется для вычитания значения текущего времени в миллисекундном формате из другого значения времени с целью определения разницы.

Методы класса Date:
getDate() (возвращает число объекта Date в диапазоне от 1 до 31)
getDay() (возвращает день недели объекта Date в диапазоне
от 0 [воскресенье] до 6 [суббота])
getHours() (возвращает значение поля часов объекта Date в диапазоне
от 0 [полночь] до 23)
getMinutes() (возвращает значение поля минут объекта Date в диапазоне от 0 до 59)
getMonth() (возвращает значение поля месяца объекта Date в диапазоне
от 0 [январь] до 11 [декабрь])
getSeconds() (возвращает значение поля секунд объекта Date в диапазоне от 0 до 59)
getTime() (возвращает внутреннее (в миллисекундах) представление объекта Date)
getYear() (возвращает значение поля года объекта Date. Возвращаемое значение
равно году минус 1900 (например, 1997-й выдается как 97)
parse() (выполняет синтаксический анализ строкового представления даты и
возвращает ее значение в миллисекундном формате)
setDate() (устанавливает значение поля числа объекта Date)
дата.setDate(число_месяца) //число_месяца равно 1-31
и т.д.

toLocalString() (преобразует Date в строку, используя местный часовой пояс)
UTC() (преобразует числовую спецификацию даты и времени в миллисекундный формат)

Задание:

Создать страницу с товаром или услугой по вашему варианту сайта или интернет-магазина.

Используя объект date и условный оператор javascript создать функцию sale(…), которая выводит сообщение о грядущей распродаже и позволяет сделать скидку на цену товара/услуги вашего сервиса в зависимости от даты распродажи:

Если до даты распродажи осталось 15-30 дней, то скидка 10%,

если до даты распродажи осталось 1-14 дней, то скидка 25%,

в день даты распродажи, скидка 50%, иначе распродажи нет и товар/услуга стоит полную стоимость.

Указывать старую стоимость зачеркнутой, а новую по скидке рядом со старой, например ~~1000,00руб~~ *500,00 руб.*

Таблица1 – Варианты заданий

Вариант	Предметная область
1	Интернет-магазин игрушек для детей
2	Сайт компании по разработке программных продуктов
3	Сайт автошколы
4	Сайт страховой медицинская компания
5	Интернет-магазин по продаже бытовой техники
6	Сайт агентства недвижимости
7	Сайт агентства по трудоустройству
8	Школьный сайт
9	Интернет-магазин по продаже строительных материалов
10	Интернет-магазин по продаже мебели
11	Интернет-магазин по продаже электронных книг (читалка)
12	Сайт фирмы по установке пластиковых окон ПВХ
13	Интернет-магазин по продаже компьютерной техники
14	Интернет-магазин «Сад и огород»
15	Сайт ветеринарной лечебницы
16	Сайт салона красоты
17	Сайт олимпиады по программированию
18	Интернет-магазин канцелярских товаров
19	Сайт волонтерского движения
20	Интернет-магазин компьютерных игр
21	Интернет-магазин охота и рыбалка
22	Новостной сайт

23	Сайт банка
24	Сайт городской поликлиники
25	Интернет-магазин охранных систем и устройств

ИИ для подготовки:

https://developer.mozilla.org/ru/docs/Web/JavaScript

Практическая работа № 5-6 (4 часа)

Проверка правильности заполнения форм с помощью JavaScript. Окна диалога.

Цель работы: формирование умений использования JavaScript для проверки правильности заполнения форм и создания окон диалога.

Объектная модель браузера

Объектная модель браузера содержит 12 объектов:

1. Document - предоставляющий возможность доступа к компонентам документа HTML.
2. Event - предоставляющий возможность доступа к свойствам событий, когда последние происходят.
3. History - предоставляет информацию об адресах, которые клиент посетил.
4. Location - предоставляет информацию об адресе текущего документа.
5. MimeType - предоставляет информацию о типе MIME (Multipurpose Internet Mail Extensions или MIME тип - является стандартом, который описывает природу и формат документа, файла или набора байтов. Он определён и стандартизирован в спецификации RFC 6838).
6. Navigator - позволяет обращаться к свойствам браузера.
7. Selection - отображает текущее выделение документа.
8. Style - представляет конкретный элемент стиля в таблице стилей.
9. TextRange - отображает разделы текста, формирующего документ HTML.
10. Screen - предоставляет информацию о мониторе и системе вывода, информации клиента.
11. Window - предоставляет свойства, методы и события, связанные с окном браузера.
12. StyleSheet - представляет все элементы стиля внутри таблицы стилей.

которые позволяют обращаться к части браузера или части страницы с помощью языка JavaScript. Кроме объектов в объектной модели вводятся следующие понятия:

Методы - способы работы с объектами. Например: закрыть окно. По сути это функция, ассоциированная с объектом.

object.methodname

События - объект сообщает нам, что нечто произошло. Например: элемент становиться активным.

Свойства - свойства объекта. Например: имя и размеры окна.

Окна диалога. Объект Window

Открытие окна

Синтаксис:

window.open ("URL или URI","имя окна","свойства окна")

Следующий оператор создаёт окно, которое отображает содержимое страницы https://www.sevsu.ru/:

window.open("https://www.sevsu.ru/")

При создании окна вы можете также предоставить имя, в данном случае - myWindow, для обращения к окну как к цели/target при отправке формы или при переходе по гиперссылке.

window.open("https://www.sevsu.ru/", "myWindow")

Имя окна не требуется при создании окна. Но окно обязано иметь имя, если вы хотите обратиться к нему из другого окна.

При открытии окна вы можете специфицировать атрибуты, такие как высота/height и ширина/width, панель /toolbar, адресная строка/location field или полосы прокрутки/scrollbars. Следующий оператор создаёт окно без панели утилит, но с полосами прокрутки:

window.open ("https://www.sevsu.ru/","myWindow","toolbar=no,scrollbars=yes")

Некоторые свойства окна:

directories - Если yes, создаются стандартные кнопки директорий браузера, такие как What's New и What's Cool.
height - Специфицирует высоту окна в пикселах.
innerHeight - Специфицирует высоту области содержимого окна в пикселах. Это свойство заменило height, которое оставлено для обеспечения обратной совместимости.
innerWidth - Специфицирует ширину области содержимого окна в пикселах. Это свойство заменило width, которое оставлено для обеспечения обратной совместимости.
location - Если yes, создаёт поле ввода Location.
menubar - Если yes, создаёт строку меню в верхней части окна.
outerHeight - Специфицирует размер по вертикали в пикселах внешней границы окна.
resizable - Если yes, даёт пользователю возможность изменять размеры окна.
screenX - Специфицирует расстояние, на котором новое окно помещается от левого края экрана.
screenY - Специфицирует расстояние, на котором новое окно помещается от верха экрана.
scrollbars - Если yes, создаются вертикальная и горизонтальная полосы прокрутки, если документ становится больше размеров окна.
status - Если yes, создаётся статусная строка внизу окна.
titlebar - Если yes, создаётся окно со строкой заголовка.
toolbar - Если yes, создаётся стандартная панель браузера с кнопками, такими как Back и Forward.
width - Специфицирует ширину окна в пикселах.

Для того чтобы функция отрабатывалась при нажатии мышкой на элементе документа, будем использовать событие **onClick** объекта Document.

Пример:

Откроем ссылку в новом окне

```
<a href=""

onClick="window.open('https://www.sevsu.ru/','myWindow','left=300,top=300,width=200,hight=400,
toolbar=no,menubar=no,location=no,directories=no')">

Открыть новое окно

</a>
```

Исполнение: Открыть новое окно

Тоже самое можно сделать с помощью кнопки:

```
<input type="button" value="Открыть новое окно"

onClick="window.open('https://www.sevsu.ru/','myWindow','left=300,top=300,width=200,hight=400,
toolbar=no,menubar=no,location=no,directories=no')">
```

Закрытие окна

Следующий оператор закрывает текущее окно:
window.close()

Проверка правильности заполнения форм с помощью JavaScript.

С помощью JavaScript можно контролировать правильность заполнения форм. Рассмотрим пример отправки данных в форме, расположенной ниже, с учетом заполнения полей формы.

Имя

E-mail

Комментарий

Отправить Сброс

Функции для этого примера:

```
<script language="JavaScript">
<!--

function CheckForm(UserForm)
{
 var is_ok = true;

 if (UserForm.name.value == '')
 {
 is_ok = false;
 alert("Введите ваше имя!!");
 UserForm.name.focus();
 }
```

```
 if (UserForm.Comment.value == '')
 {
 is_ok = false;
 alert("Введите ваши комменирии!");
 UserForm.Comment.focus();
 }
 if (UserForm.email.value == '')
 {
 is_ok = false;
 alert("Введите ваш e-mail!");
 UserForm.email.focus();
}

return is_ok;
}
// -->
</script>
```

Вызов функции в теге <FORM> при нажатии на кнопку "Отправить"

```
<form onSubmit = "return CheckForm(this)">
```

Имена полей

```
<input type="text" name="name" size="20">
<input type="text" name="email"  size="20">
<textarea rows="3" name="Comment" cols="20">
```

Чек-лист для проверки поля email

Номер	Проверка	Ожидаемый результат
1	Пустое поле email	Сообщение о незаполненном поле email
2	Email в нижнем регистре	Значение поля принимается
3	Email в верхнем регистре	Значение поля принимается
4	Email с цифрами в имени пользователя	Значение поля принимается
5	Email с цифрами в доменной части	Значение поля принимается
6	Email с дефисом в имени пользователя	Значение поля принимается
7	Email с дефисом в доменной части	Значение поля принимается
8	Email со знаком подчеркивания в имени пользователя	Значение поля принимается
9	Email со знаком подчеркивания в доменной части	Значение поля принимается
10	Email с точками в имени пользователя	Значение поля принимается
11	Email с несколькими точками в доменной части	Значение поля принимается
12	Email без точек в доменной части	Сообщение о некорректном email
13	Превышение длины email (>320 символов)	Сообщение о некорректном email
14	Отсутствие @ в email	Сообщение о некорректном email
15	Email с пробелами в имени пользователя	Сообщение о некорректном email
16	Email с пробелами в доменной части	Сообщение о некорректном email
17	Email без имени пользователя	Сообщение о некорректном email
18	Email без доменной части	Сообщение о некорректном email

19	Некорректный домен первого уровня (допустимо 2-6 букв после точки: .ru)	Сообщение о некорректном email

JavaScript валидация формы

Закрепим на input событие onkeyup, которое сработает при отпускании клавиши. Как только пользователь ввел email и отпустил клавишу, запустится функция validate().

<input type="text" id="email" onkeyup="validate()" placeholder="Enter Email Address">

Написание функции validate()

Найдем в документе элемент form и занесем его в переменную form. Затем по ID получим значение поля email и так же присвоим переменной email. Объявим переменную pattern и присвоим ей регулярное выражение. Атрибут pattern используется только для input элементов. Это позволяет нам задать собственные правила проверки, используя Регулярные Выражения (RegEx). Регулярное выражение - это шаблон, задающий правила, которому должны соответствовать вводимые данные. Еще раз, если величина не будет соответствовать заданному шаблону, то ввод вызовет ошибку.

https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Regular_expressions

```
function validate(){
  var form = document.getElementById("form");
  var email = document.getElementById("email").value;
  var pattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/
  ...
}
```

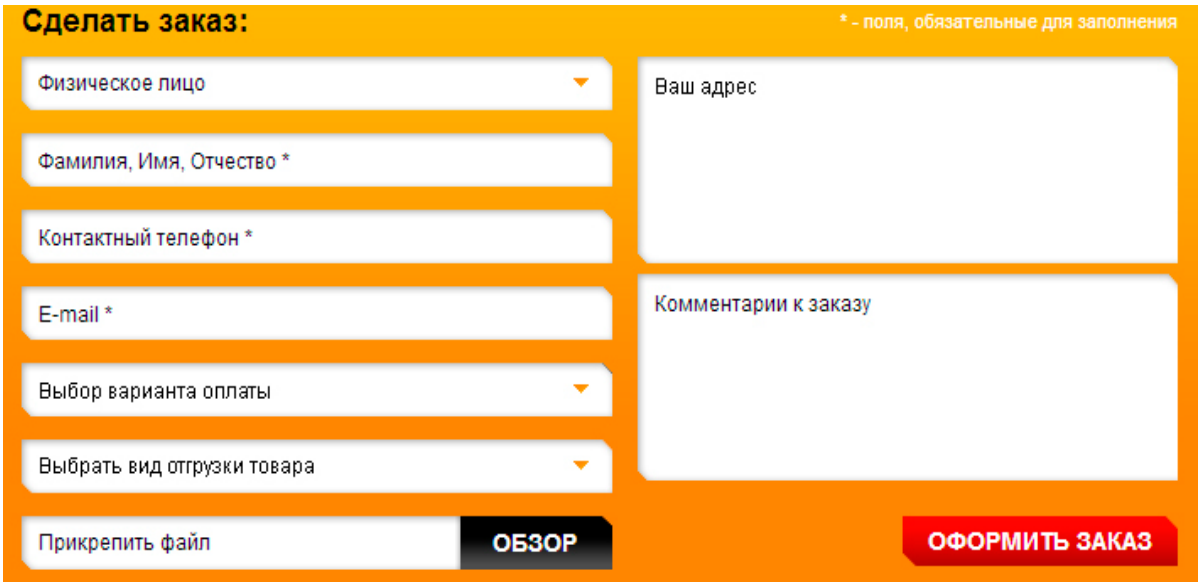
С помощью метода match() сопоставим полученный email с регулярным выражением, применив условную конструкцию if-else. Если (if) email прошел проверку по шаблону, то добавится класс valid (зеленый индикатор), а invalid (красный индикатор) удалится из формы. В противном случае (else), все произойдет с точностью наоборот. Если поле оставить пустым (email == ""), то удалятся оба класса и индикатор снова будет серым.

```
function validate(){
  ...
  if(email.match(pattern))
  {
    form.classList.add("valid");
    form.classList.remove("invalid");
  }
  else{
    form.classList.remove("valid");
    form.classList.add("invalid");
  }
  if (email == "") {
    form.classList.remove("valid");
    form.classList.remove("invalid");
  }
}
```

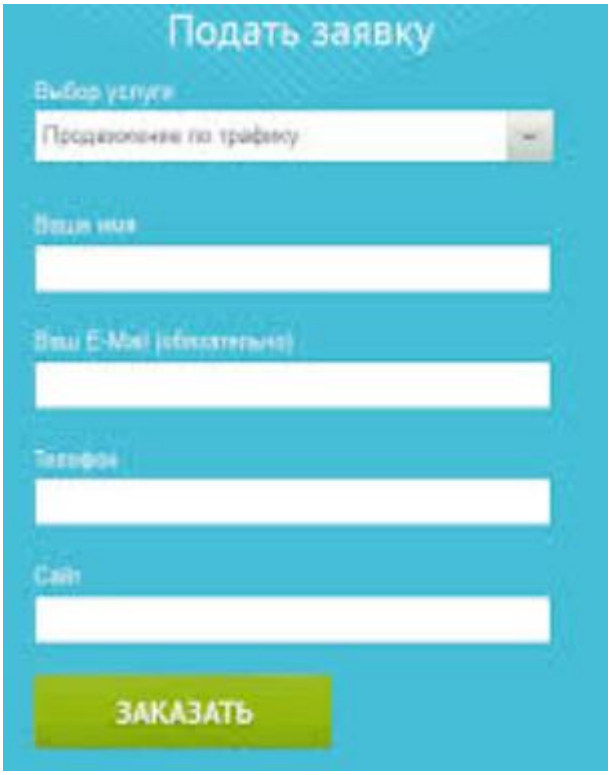
Задание:

1. **Для созданной страницы с товаром или услугой в прошлой практической работе добавьте кнопку купить (для товара)/ заказать (для услуги).**
2. **По нажатию кнопки в новом диалоговом окне должна быть**

спроектирована форма заказа товара (1-2час):



Или форма заказа услуги (список услуг придумать по предметной области):



Нужно обязательно задать:

- отступ слева
- отступ сверху
- ширину
- высоту
- не отображение панели инструментов
- не отображение панели меню
- отображение строки состояния
- не отображение адресной строки

- не отображение кнопок браузера
- отображение полос прокрутки.

3. **Выполните проверку (3-4 час)**

А) заполнения полей формы помеченных *

Б) корректное отображение поля e-mail

Таблица1 – Варианты заданий

Вариант	Предметная область
1	Интернет-магазин игрушек для детей
2	Сайт компании по разработке программных продуктов
3	Сайт автошколы
4	Сайт страховой медицинская компания
5	Интернет-магазин по продаже бытовой техники
6	Сайт агентства недвижимости
7	Сайт агентства по трудоустройству
8	Школьный сайт
9	Интернет-магазин по продаже строительных материалов
10	Интернет-магазин по продаже мебели
11	Интернет-магазин по продаже электронных книг (читалка)
12	Сайт фирмы по установке пластиковых окон ПВХ
13	Интернет-магазин по продаже компьютерной техники
14	Интернет-магазин «Сад и огород»
15	Сайт ветеринарной лечебницы
16	Сайт салона красоты
17	Сайт олимпиады по программированию
18	Интернет-магазин канцелярских товаров
19	Сайт волонтерского движения
20	Интернет-магазин компьютерных игр
21	Интернет-магазин охота и рыбалка
22	Новостной сайт
23	Сайт банка
24	Сайт городской поликлиники
25	Интернет-магазин охранных систем и устройств

ИИ для подготовки:

https://developer.mozilla.org/ru/docs/Web/JavaScript

Практическая работа № 7

События языка JavaScript.

Обработка событий

Для взаимодействия с пользователем в JavaScript определен механизм событий. Например, когда пользователь нажимает кнопку, то возникает событие нажатия кнопки. В коде JavaScript мы можем определить возникновение события и как-то его обработать.

В JavaScript есть следующие типы событий:

- События мыши (перемещение курсора, нажатие мыши и т.д.)
- События клавиатуры (нажатие или отпускание клавиши клавиатуры)
- События жизненного цикла элементов (например, событие загрузки веб-станицы)
- События элементов форм (нажатие кнопки на форме, выбор элемента в выпадающем списке и т.д.)
- События, возникающие при изменении элементов DOM
- События, возникающие при касании на сенсорных экранах
- События, возникающие при возникновении ошибок

Рассмотрим простейшую обработку событий. Например, на веб-странице у нас есть следующий элемент div:

```
<div id="rect" onclick="alert('Нажато')" style="width:50px;height:50px;background-color:blue;"></div>
```

Здесь определен обычный блок div, который имеет атрибут onclick, который задает обработчик события нажатия на блок div. То есть, чтобы обработать какое-либо событие, нам надо определить для него обработчик. Обработчик представляет собой код на языке JavaScript. В данном случае обработчик выглядит довольно просто:

```
alert('Нажато')
```

И при нажатии на кнопку будет выскакивать сообщение:

Также можно было бы вынести все действия по обработке события в отдельную функцию:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
</head>
<body>
<div id="rect" onclick="displayMessage()" style="width:50px;height:50px;background-color:blue;"></div>
<script>
function displayMessage(){
    alert('Нажато');
}
</script>
</body>
</html>
```

Теперь обработчиком события будет выступать функция displayMessage.

Передача параметров в обработчик события
В обработчик можно передавать параметры. Например, мы можем передать текущий объект, на котором возникает событие:

```
<a href="page1.html" onclick="return handler(this)">Станица 1</a>
<script>
function handler(obj){

   alert(obj.href);
   return false;
}
</script>
```

Ключевое слово this указывает на текущий объект ссылки, на которую производится нажатие. И в коде обработчика мы можем получить этот объект и обратиться к его свойствам, например, к свойству href.

Кроме того, надо отметить, что здесь обработчик возвращает результат. Хотя в первом примере с блоком div от обработчика не требовалось возвращения результата. Дело в том, что для некоторых обработчиков можно подтвердить или остановить обработку события. Например, нажатие на ссылку должно привести к переадресации. Но возвращая из обработчика false, мы можем остановить стандартный путь обработки события, и переадресации не будет. Если же возвращать значение true, то событие обрабатывается в стандартном порядке.

Если же мы вовсе уберем возвращении результата, то событие будет обрабатываться, как будто возвращается значение true:

```
<a href="page1.html" onclick="handler(this)">Станица 1</a>
<script>
function handler(obj){
    alert(obj.href);
}
</script>
```

Кроме непосредственно элемента-источника события в обработчик мы можем передавать объект event. Этот объект не определяется разработчиком, это просто аргумент функции обработчика, который хранит всю информацию о событии. Например:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <style>
   #rect{
      width:50px;
      height:50px;
      background-color:blue;
   }
   </style>
</head>
<body>
<div id="rect" onclick="handler(event)"></div>
```

```
<script>
function handler(e){

   alert(e.type); // получаем тип события
}
</script>
</body>
</html>
```

В данном случае с помощью свойства type объекта event получаем тип события (в данном случае тип click).

Пример 1. Использование метода alert() и обработчика событий onBlur.

```
<html>
<head>
<script language="JavaScript">
<!--
function Age(form)
{
if (1*document.form1.a.value <= 18)
{
alert ("Возраст должен быть старше 18");
}
}
<!---->
</script>
</head>
<body>
<form name="form1">
Возраст: <input type="text" name="a" size=5 onBlur="Age()">
</form>
</body>
</html>
```

Пример 2. Использование обработчика событий onFocus.

```
<html>
<head>
<script language="JavaScript">
<!--//
function loan(obj)
{
var a=1*obj.a1.value;
var result=a*a;
obj.sr.value=result;
}
//-->
</script>
</head>
<body>
<form name="Form1">
<font size=3>
Сторона квадрата:  <input type="text" size=6 name="a1"><br>
Площадь квадрата:  <input type="text" size=6 name="sr" onFocus="loan(Form1)">
```

```
</form>
</body>
</html>
```

Пример 3. Использование метода confirm и метода close.

```
<html>
<head>
<script language="JavaScript">
<!--//
function exit()
{
var a=window.confirm("Хотите закрыть окно?")
if (a==true)
window.close()
}
//-->
</script>
</head>
<body>
<form name="Form1">
<font size=3>
<input type="button" value="Close" onClick="exit()">
</form>
</body>
</html>
```

Пример 4. Использование метода prompt.

```
<html>
<head>
<script language="JavaScript">
<!--//
function registration(obj)
{
var a=window.prompt("Введите Ваше имя"," ")
obj.regname.value=a;
}
//-->
</script>
</head>
<body>
<form name="form1">
<input type="button" value="Регистрация" onClick="registration(form1)">
<input type=text name=regname size=50>
</form>
</body>
</html>
```

Задания:

В созданной на предыдущей практической работе странице со скидками на товар или услугу по акции к праздничной дате выполнить создание обработчиков событий о доступности товара/услуги по достижении 18 лет и обработчиков событий по варианту:

№ вари-анта	Создать строку, при наведении курсора на которую меняется			Создать изображение с надпись к нему	
	Шрифт	Цвет шрифта	Цвет фона	При щелчке на изображении меняется	Цвет при щелчке на подписи
1/13	Увеличить до 30 pt	белый	голубой	Изображение и надпись	синий
2/14	Увеличить до 36 pt	синий	белый	Цвет шрифта и надпись	красный
3/15	Увеличить до 42 pt	красный	голубой	Цвет фона и надпись	зеленый
4/16	Увеличить до 48 pt	зеленый	белый	Цвет шрифта и изображ	голубой
5/17	Увеличить до 52 pt	синий	зеленый	Цвет фона и изображ	желтый
6/18	Увеличить до 56 pt	белый	голубой	Фон и цвет надписи	зеленый
7/19	Уменьшить на 6 pt	синий	белый	Изображение и надпись	синий
8/20	Уменьшить на 10 pt	красный	голубой	Цвет шрифта и надпись	желтый
9/21	Уменьшить на 16 pt	зеленый	белый	Цвет фона и надпись	красный
10/22	Уменьшить на 12 pt	синий	зеленый	Цвет шрифта и изображ	голубой
11/23	Уменьшить на 8 pt	красный	голубой	Цвет фона и изображ	зеленый
12/24	Уменьшить на 4 pt	зеленый	белый	Фон и цвет надписи	синий

ИИ для подготовки:

https://developer.mozilla.org/ru/docs/Web/JavaScript

МДК 03.02 Оптимизация веб-приложений

Практическая работа 1
Проведение общего аудита сайта: SEO, юзабилити, тексты

Цель работы: изучить методы общего аудита сайта (SEO, юзабилити, тексты) на практике.

Задание.

1. Выберите адрес сайта, с которым вы будете работать, согласно варианта задания из таблицы 1.

Таблица 1.

№	Адрес сайта
1	https://keytostart.space/
2	https://www.astrozona.ru
3	https://rosastronomy.ru
4	https://www.gctc.ru
5	https://devyatayaplaneta.ru
6	https://sktur.ru/
7	https://rostec.ru/media/news/kak-stat-kosmonavtom/
8	https://kosmodrom.space/
9	https://www.glavkosmos.com/
10	https://spacerussiantour.ru/

Представьте в отчете краткую информацию о том, что это за сайт, чему он посвящен, является ли он многостраничным, какие действия доступны пользователю на сайте. Сделайте скрин страницы сайта для отчета. Если на сайте есть пользовательские формы, то дайте им краткую характеристику (что это за форма, для чего она нужна), сделайте скрин формы для отчета.

2. Оцените валидность кода сайта (адрес сайта выбирается согласно варианта задания из таблицы с помощью сервиса https://validator.w3.org/. Выпишите в отчет общее количество найденных ошибок, общее количество предупреждений. Сделайте скрин для отчета. Рассмотрите любые 5 из них, как их исправить. Заполните таблицу 2.

Таблица 2.

№	Ошибка. Исходный код с ошибкой.	Исправленный фрагмент кода (уже без ошибки)
1		
2		
3		
4		
5		

3. Оцените валидность кода сайта (адрес сайта выбирается согласно варианта задания из таблицы с помощью сервиса https://jigsaw.w3.org/ . Выпишите в отчет общее количество найденных ошибок, общее количество предупреждений. Сделайте скрин для отчета. Рассмотрите любые 5 из них. Посмотрите, как их исправить. Заполните таблицу 3.

Таблица 3.

№	Сообщение об ошибке	Текст CSS с ошибкой
1		
2		
3		
4		
5		

4. Исследуйте адаптивность сайта с помощью одного из сервисов:

-http://iloveadaptive.com/

-google.com/webmasters/tools/mobile-friendly/ Скрины в отчет.

5. Опишите, для чего используется плагин Theme Authenticity Checker в WordPress?
6. Опишите назначение плагина Download Manager в WordPress. Как организовать скачивание файлов с вашего сайта с его помощью. Приведите последовательность действий.
7. Опишите плагин Янлекс Поделиться в WordPress, его назначение. Что такое вид социальных кнопок **Счетчик**, зачем они используются.

Практическая работа 2
Проведение общего аудита сайта: SEO, юзабилити, тексты

Цель работы: изучить методы общего аудита сайта (SEO, юзабилити, тексты) на практике.

Задание.

1. Выберите адрес сайта, с которым вы будете работать, согласно варианта задания из таблицы 1.

Таблица 1.

№	Адрес сайта
1	https://keytostart.space/
2	https://www.astrozona.ru
3	https://rosastronomy.ru
4	https://www.gctc.ru
5	https://devyatayaplaneta.ru
6	https://sktur.ru/
7	https://rostec.ru/media/news/kak-stat-kosmonavtom/
8	https://kosmodrom.space/
9	https://www.glavkosmos.com/

10	https://spacerussiantour.ru/

2. Выберите текст, размещенный на странице сайта и проведите анализ его уникальности с помощью сервиса https://text.ru. Сделайте скрин для отчета. Опираясь, на информацию ниже, сделайте вывод об уровне уникальности размещенного контента.

Уникальный текст на сайте не должен иметь совпадений с текстами на других сайтах Сети.

Принято считать, что если процент уникальности:

1. Выше 90% - высокий уровень уникальности текста;
2. 80-90% - средний уровень уникальности;
3. Ниже 80% - низкий уровень уникальности;
4. Уникальность, близкая к нулю – плагиат.

Допустимый уровень заимствования текста из чужих работ не должен превышать 5-10%.

Процент уникальности страниц сайта должен быть не менее 90%, в противном случае поисковая система может исключить эти страницы из поиска.

3. Выполните автоматический рерайт имеющегося текста с помощью программы синонимайзера https://bname.ru/tools/synonymizer/

Скопируйте полученный текст и проведите анализ его уникальности с помощью сервиса https://text.ru. Сделайте скрин для отчета. Сравните полученные сейчас результаты анализа с теми, что были получены в пункте 2 настоящей работы. Как изменилась уникальность текста? Как изменилось число орфографических ошибок? Сделайте вывод.

4. Раскройте понятие «прогон сайта». Как влияет прогон сайта на индексацию и ссылочную массу?
5. Выполните проверку релевантности страницы сайта с помощью https://ru.megaindex.com/a/textanalysis .

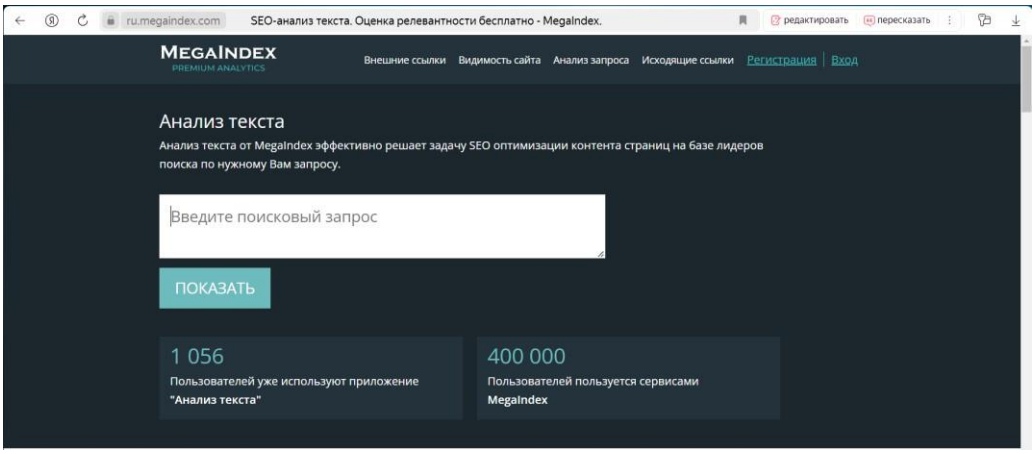


Рисунок 1

На рисунке 1 приведено окно анализатора текста MegaIndex.

В качестве поискового запроса указываем то, что прописано в теге

<title> на странице исследуемого сайта. После этого жмем «Показать».

Во вновь открывшемся окне вводим в поле «URL для сравнения» адрес страницы исследуемого сайта. Жмем кнопку «Обновить».

Делаем скрин экрана для отчета. На нем должен быть отображен показатель релевантности.

Релевантность текста- это его соответствие поисковым запросам, т.е. соответствие статьи на сайте его ключевой фразе.

Чем выше процент релевантности текста, тем эффективнее его индексация поисковыми роботами.

При релевантности свыше 90% текс будет индексироваться идеально. Смотрим на полученный показатель релевантности и делаем вывод:

«По результатам проверки становится ясно, что релевантность нашей страницы достигает %, когда она должна быть не ниже 90%. Следовательно, статью следует/не следует доработать».

Чтобы повысить релевантность страницы, необходимо повысить релевантность каждого элемента страницы, а также провести соответствующую работу над ссылочной структурой сайта. В частности, увеличить релевантность материала позволяет использование анкорных ссылок.

6. Раскройте понятия «Анкорные ссылки» или «Анкор (якорь)», «Безанкорные ссылки», «Перелинковка сайта».

Практическая работа 3
Исследование способов ускорения загрузки сайтов

Цель работы: изучить основные аспекты способов ускорения загрузки сайтов

Задание 1.

Изучите публикацию А.В.Вовк, В.С. Кузнецовой «Методы повышения загрузки сайтов». Создайте презентацию на базе теоретических сведений, приведенных в статье.

Задание 2.

Сжатие и валидность

Говоря о файлах CSS ни в коем случае, **нельзя забывать об их валидности**. Сжатие файла только тогда имеет смысл, если при этом не потеряется его валидность. К сожалению, многие сервисы по сжатию редко обновляются и используют устаревшие алгоритмы и методы сжатия. Получая на них хороший эффект по уменьшению объема, можно потерять главное, валидность файла. В результате вместо улучшения показателя скорости сайта, получим увеличение времени полного открытия сайта или потерю вида сайта.

1. Возьмите CSS-файл шаблона Twenty Seventeen. Укажите в отчете его объем.
2. Осуществите проверку его на валидность с поморщью https://validator.w3.org/ . Запишите в отчет общее число ошибок, общее число предупреждений. Приложите скрин. Обратите внимание, что сервис сам исправляет эти ошибки и возвращает файл исправленным ниже ошибок. Используйте исправленный CSS код без ошибок для дальнейшего сжатия. Какой объем файла после исправлений?
3. Используйте сервисы сжатия из списка ниже согласно варианта задания. Для каждого использованного сервиса в отчете
    - приведите краткое описание, скрин,
    - укажите объем исходного кода,
    - объем оптимизированного кода,
    - коэффициент сжатия,
    - Проверьте валидность CSS после сжатия с помощью https://validator.w3.org/. Внесите в отчет результат проверки (появились ли ошибки, сколько обнаружено ошибок и предупреждений)?
4. Сделайте вывод об эффективности исследованных сервисов сжатия.

Таблица 1 – Варианты заданий

№ варианта	Список номеров сервисов сжатия для исследования
1	1, 5, 7, 9
2	1, 3,4, 8
3	1, 3, 5,10
4	1, 2, 4, 6
5	1, 3, 7,10
6	1, 2, 3,8
7	1, 4, 5, 9
8	1, 6, 10,12
9	1, 3,5, 9
10	1, 7, 9, 10
11	1, 2, 4,11
12	1, 8, 9,12

Сервисы сжатия

1. https://www.cy-pr.com/tools/css/
2. http://www.cssdrive.com/index.php/main/csscompressor/
3. http://www.cleancss.com/
4. http://www.csscompressor.com/
5. https://www.giftofspeed.com/css-compressor/
6. http://www.pagecolumn.com/tool/css_compressor.htm
7. http://www.lotterypost.com/css-compress.aspx
8. http://www.phpinsider.com/compress_css.php
9. http://www.cssportal.com/css-optimize/

10. http://www.generateit.net/css-optimize/
11. http://tools.arantius.com/css-compressor
12. http://refresh-sf.com/

Практическая работа 4-5
Проведение внутренней SEO оптимизация сайта. Создание семантического ядра сайта. Сбор ключевых фраз.

Цель работы: Ознакомиться с термином SEO оптимизации и основами продвижения сайта, применяемыми методиками и алгоритмами. Применить изложенные знания на практике, чтобы получить более детальное представление о сложности создания семантического ядра сайта, трудоемкости этого процесса, о механизме перехода от семантики к логической структуре проектируемого сайта.

Теоретические сведения.

1. Что такое поисковая оптимизация?

Поиско́вая оптимиза́ция (англ. search engine optimization, SEO) — комплекс мер по внутренней и внешней оптимизации для поднятия позиций сайта в результатах выдачи поисковых систем по определённым запросам пользователей, с целью увеличения сетевого трафика (для информационных ресурсов) и потенциальных клиентов (для коммерческих ресурсов) и последующей монетизации (получение дохода) этого трафика. SEO может быть ориентировано на различные виды поиска, включая поиск информации, товаров, услуг, изображений, видеороликов, новостей и специфические отраслевые поисковые системы.

Обычно чем выше позиция сайта в результатах поиска, тем больше заинтересованных посетителей переходит на него с поисковых систем. Все факторы, влияющие на положение сайта в выдаче поисковой системы, можно разбить на внешние и внутренние. К внутренней оптимизации (касающейся исключительно внутренней системы сайта) относится работа, направленная на общее повышение качества сайта, пользы, которую он приносит посетителю. Сюда можно отнести работу над структурой проекта, над облегчением восприятия контента и непосредственно над качеством этого контента. Значение общего количества таких факторов в большинстве источников колеблется в районе 200. Функциональный подход к поисковой оптимизации, направленный на подгонку определённых факторов к их целевым значениям, отошел в прошлое в связи с усложнением алгоритмов поисковых систем — стоимость «балансирования» десятков факторов многократно превышает стоимость создания изначально качественного ресурса.

Внутренняя оптимизация включает в себя работу с заголовками страницы, которые содержатся в коде с тегами <h1>, <h2>, <h3>, надписью, которая высвечивается на вкладке браузера — Title, и созданием уникального текста на этих же страницах. Также важно уделить внимание мета-тегу description, поскольку именно его пользователь чаще всего видит под url сайта в поисковой выдаче.

Методы внутренней поисковой **оптимизации:**

- написание качественного контента;
- увеличение скорости работы сайта;
- адаптация под мобильные устройства;
- HTML, CSS— валидация;
- исследования нужных вам ключевых слов и конкурентов;
- создание правильной структуры сайта;
- оптимизация тегов H1, Title и мета-тегов Description;
- внутренняя перелинковка.

Перелинковка – что это такое и как ее сделать?

Перелинковка— это связывание страниц одного сайта или разных ресурсов гиперссылками. Само слово происходит от английского слова link, что можно перевести как ссылка или связующее звено. Если грамотно сделать внутреннюю перелинковку сайта, можно без лишних денежных затрат увеличить позиции по продвигаемым запросам.

Виды внутренней перелинковки сайта

Контекстная — ссылки располагаются в тексте и ведут на другие страницы сайта (Пример: Википедия).

Сквозная — ссылки на одни и те же страницы располагаются на всех страницах продвигаемого сайта.

Контекстная перелинковка— это самый удобный вариант с точки зрения SEO-эффекта и пользы для посетителей. Ссылки, установленные на страницах вашего сайта, приносят эффект, сравнимый с покупными. Однако, помимо пользы, такие ссылки помогут вашему сайту улучшить и поведенческие метрики. Правильная внутренняя перелинковка страниц сайта способствуют удержанию пользователя на сайте, увеличивая количество просмотренных страниц и время, проведенное на сайте.

Сквозная перелинковка— одна из разновидностей «прокачки» продвигаемых страниц сайта. Суть сквозных ссылок— увеличить вес продвигаемых страниц. Главный минус в том, что в отличие от контекстных ссылок сквозные не несут значительную пользу для посетителей ресурса. Пользователи будут кликать по подобным ссылкам, но не массово.

Правила внутренней перелинковки сайта.

Лучше всего не использовать много ссылок на одной странице (оптимально не больше 50), так как распределение веса происходит по всем исходящим ссылкам. Закрывайте ненужные ссылки от индексации, чтобы они не мешали продвигаемым в топ страницам. Это очень важно для правильной передачи веса. Закрывайте от индексации раздел с тегами, чтобы не плодить дубли страниц и не запутывать роботов поисковых систем.

Соблюдая эти простые правила, вы сможете наиболее грамотно распределить статический

вес между страницами своего сайта.

Структура сайта и SEO-оптимизация

2. Как сделать структуру сайта для продвижения, отталкиваясь от семантического ядра

Структура сайта - это связь документов, принадлежащих ему, между собой. Правильная структура позволит поисковым роботам быстро совершать обход ресурса, а посетителям легко перемещаться между его страницами.

При разработке структуры нового сайта целесообразно отталкиваться от его семантического ядра.

Семантическое ядро – это тщательно подобранный и отсортированный список ключевых слов и фраз, составленный в соответствии с направлением сайта, которые распределяются по страницам. На его основе проводится SEO-продвижение сайта и создается контент.

Главные задачи ядра:

- Расширение базового списка ключей, добавление менее популярных запросов и «хвостов»;
- Помощь в организации грамотной структуры сайта;
- Разбивка на категории и подкатегории.

Семантическое ядро сайта применяется не только для оптимизации, его можно использовать в создании контекстной рекламы и объявлений. Можно отслеживать популярные запросы, по которым приходят пользователи и продвигаются конкуренты.
Как создать семантическое ядро.

Для сбора ядра применяются специальные сервисы и программы. Один из них – бесплатный сервис Google Adwords. Однако в виду введенных санкций в Крыму он не действует. Есть возможность обойти запрет, используя VPN. Так как в нашу задачу пока не входит полная проработка ключевых запросов для всех топовых поисковых систем, ограничимся бесплатным сервисом Wordstat Yandex https://wordstat.yandex.ru . Возможностей этого сервиса вполне достаточно, чтобы осуществить подбор ключевых фраз для составления семантического ядра.

Итак, перейдем на главную страницу сервиса. Затем введем в поле поиска словосочетание или слово, соответствующее тематики нашего сайта, и нажмем кнопку «Подобрать». Ниже отобразиться таблица, содержащая статистику по словам. Скорее всего она займет не один лист. Полученный список следует перенести таблицу Excel.

В таблице с запросами должно быть два столбца:

Статистика по словам (ключевые слова);

Показов в месяц (среднее число запросов за месяц).

По цифрам во второй колонке можно сделать выводы, насколько популярна фраза: чем выше число, тем больший процент пользователей был заинтересован в запросе. По данному показателю ключи подразделяют на три группы: высокочастотные (у них самый большой показатель), средне- и низкочастотные.

Подбор фраз следует осуществлять по нескольким ключевым словам. К примеру. Если сайт посвящен крюинговой компании, то рекомендуется осуществить подбор фраз для «крюинг», «работа морякам», «услуги морякам», «документы морякам» и т.д.

Составление семантического ядра требуется именно для того, чтобы рассортировать запросы по популярности. Для этого потребуется применить фильтр и выставить все значения в порядке убывания, то есть от большего к меньшему. Самый важный ключ – самый высокий, от этого нужно будет отталкиваться в будущем при распределении ключей по страницам.

WordStat пытается составить семантику, исходя из подбора фраз, содержащих заданные слова. Поэтому в выданном варианте есть ключи, совершенно неподходящие тематике сайта (рис.1). К примеру «крюинг в СПб» - совершенно неподходящий ключ, если сайт разрабатывается для другого региона. Такие лишние ключевые фразы придется удалить или с помощью фильтров, или вручную.

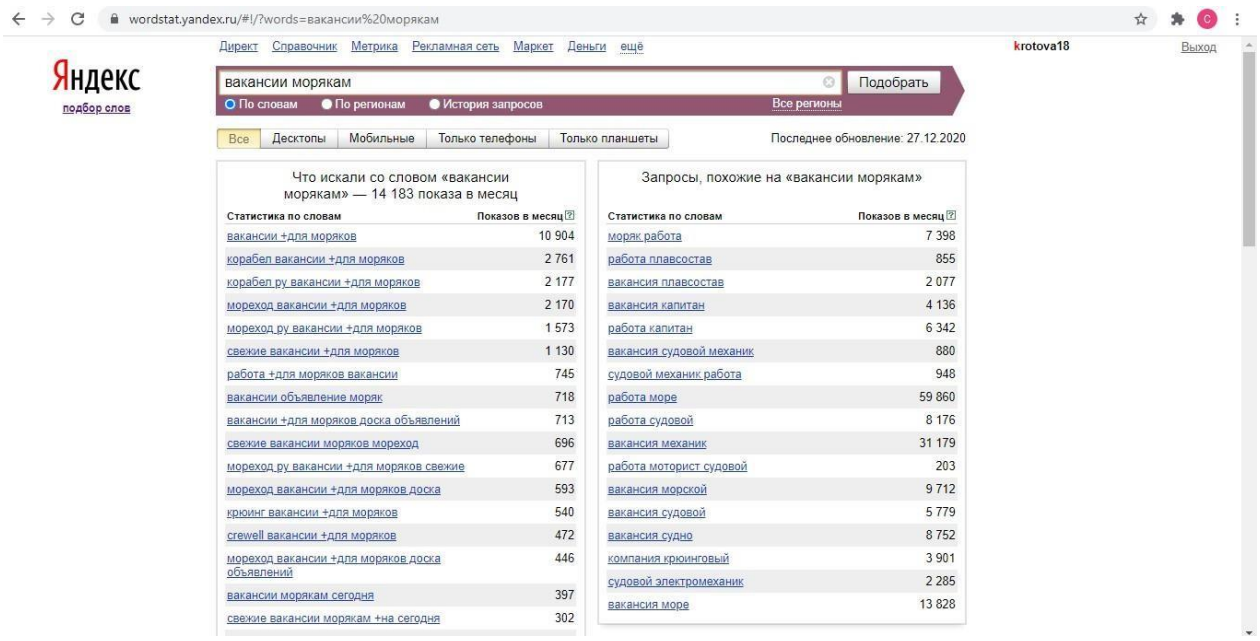


Рисунок 1 – Подбор ключевых фраз в WordStat Yandex.

Если строк в таблице очень много, допускается сократить ядро до 200-300 позиций и убрать низкочастотные варианты.

Следующим важным шагом станет сортировка полученной информации по группам.

Зная, что такое семантическое ядро сайта и на какие группы оно разделено, можно приступать к оптимизации. Для начала потребуется скорректировать структуру каталога,

выделить разделы и подразделы.

Затем для каждой страницы, начиная от главной и заканчивая товаром или статьей, понадобится составить мета-теги: заголовки и описания. Заголовок должен быть коротким и емким, отражая суть содержимого. В нем следует использовать главное ключевое слово, отобранное для данной страницы. В описание вносится небольшой текст рекламного характера, в нем можно применить 2-5 штук ключей.

Таким образом, построение структуры сайта на базе его семантического ядра сводится к следующему:

- собрать семантическое ядро - все поисковые запросы, по которым сайт могут искать посетители;
- рассортировать запросы по родственным;
- сделать наброски структуры, при этом сразу распределяем запросы по страницам;
- дать название страницам;
- продумать навигацию;
- завершить работу.Рассмотрим пример, как правильно распределить запросы по страницам для сайта, осуществляющего продажу платьев.

Из анкор-листа выбираем общие запросы (платье, платья, женские платья и т.д.). Как правило, это высокочастотники.
Группируем отдельно уточняющие среднечастотники (платье повседневное, платье коктейльное, платье вечернее и т.д.).
Такую же работу проводим с низкочастотниками, еще более сужающими круг поиска (платье повседневное мини, платье повседневное миди).
Следующая подгруппа будет типа: платье повседневное мини красное, платье повседневное мини черное и т.д.
Составляем структуру, исходя из логики. У нас получится примерно следующее:

платье, платья, женские платья, женские платья купить
- **платье повседневное, платье повседневное купить**
  - **платье повседневное мини, платье повседневное мини недорого, платье повседневное мини купить**
    - **платье повседневное мини красное, платье повседневное мини красное купить**
    - **платье повседневное мини черное, платье повседневное мини черное купить**
  - **платье повседневное миди**
    - ...
- **платье коктейльное**
  - ...
- **платье вечернее**
  - ...

На основании полученного дерева анкоров создаем структуру самого сайта:

Как видите, одним действием мы решили две важные проблемы: создали правильную структуру сайта и распределили ключи по страницам. (В данном случае речь идет только о фрагменте.)

Задание.

Необходимо разработать и построить структуру небольшого интернет- магазина, осуществляющего продажу песка (морского и речного) с доставкой в Севастополе на основе семантического ядра.

Должны быть выполнены следующие шаги:

С помощью WordStat Yandex осуществить подбор ключевых фраз и выбрать из списка всего 60 подходящих (20 высокочастотных, 20 среднечастотных, 20 низкочастотных) – сформировать семантическое ядро;

1. Рассортировать по родственным группам полученные запросы;
2. С помощью MindMap и анализа полученной информации построить примерную структуру сайта.
3. Составить таблицу, в которой будут указаны название страницы и список запросов для нее. (распределить запросы по страницам).
4. Определить максимальный уровень вложенности страницы. Если он превышает цифру 3 – переделать структуру сайта еще раз.
5. Сделать вывод о проделанной работе.

Результатом выполнения работы является документ World, в котором есть:

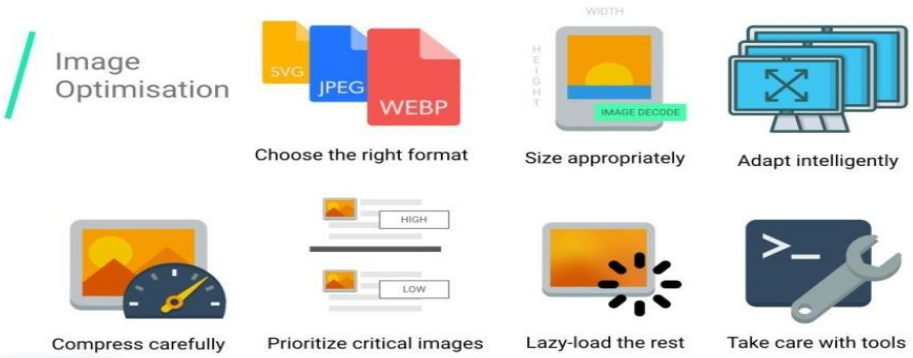
- Титульный лист, где указаны ФИО студента, номер группы, название учебного заведения; ФИО преподавателя, который эту работу будет проверять; номер работы, тема работы;
- Цель работы;
- Ход работы: результаты выполнения каждого пункта раздела «Задание»;
- Вывод.

Практическая работа 6
Исследование способов ускорения загрузки сайтов.

Изображения составляют огромную долю интернет-трафика. Согласно HTTP Archive, 60% объёма веб-страниц — это графика в форматах JPEG, PNG и GIF.

Эксперимент Тэмми Эвертса доказал, что добавление изображений на страницу или увеличение существующих изображений повышает коэффициент конверсии (процент посетителей, которые становятся клиентами
— прим. пер.). Так что картинки никуда не денутся — вот почему важно вложиться в эффективную стратегию по их сжатию.

Оптимизация изображений включает ряд мер. Выбор зависит от того, какую потерю качества считать приемлемой.



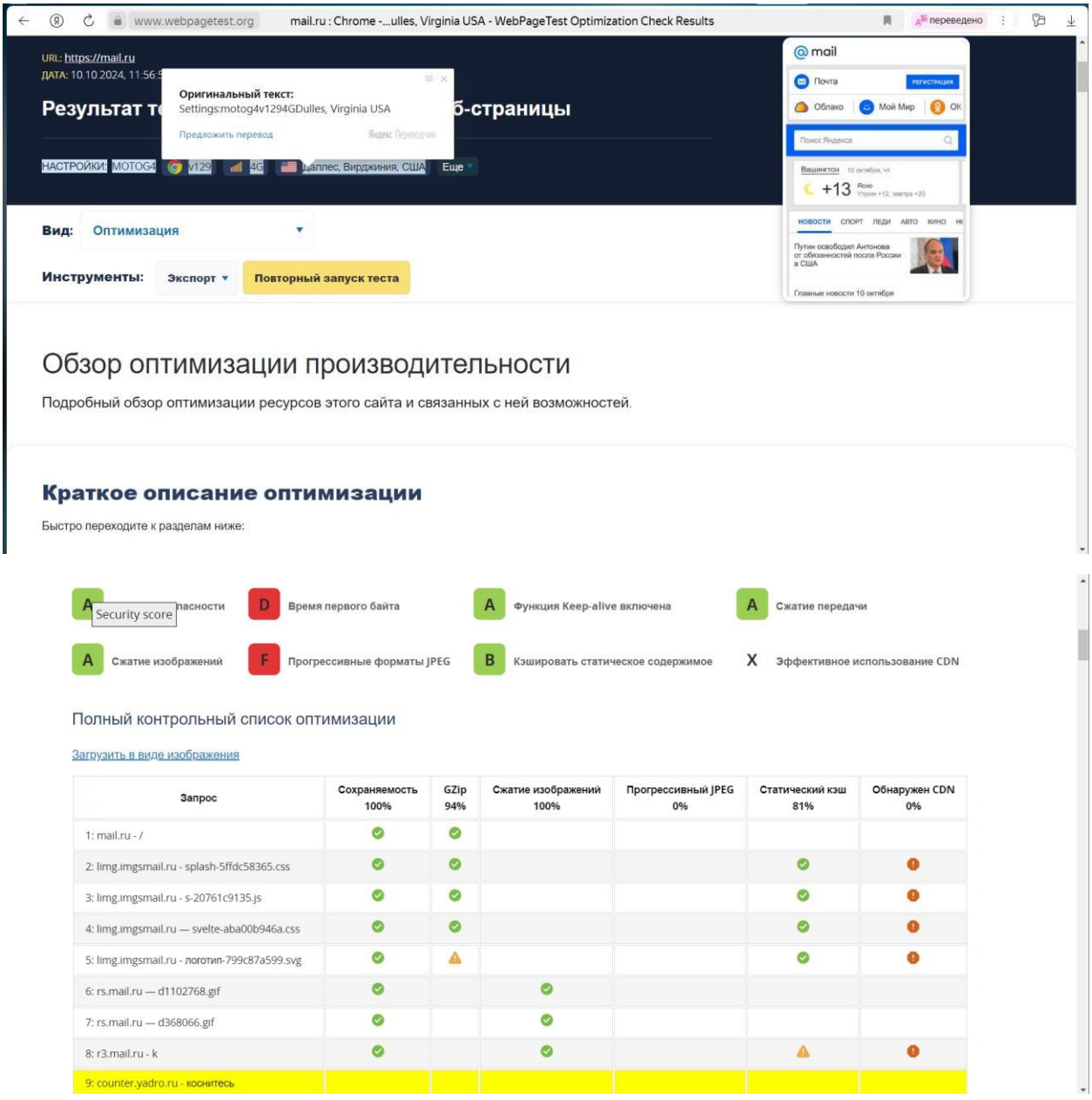
Поясните каждое изображение, приведенное на рисунке выше.

> *Оптимизация: выбрать правильный формат, аккуратно сжать и установить приоритеты загрузки разных изображений*

1. **Чтобы узнать, требуется ли оптимизация изображений, проведите аудит сайта (согласно варианта задания) с помощью WebPageTest.org. Занесите в отчет полученные результаты, скрины, сделайте вывод.**

Ниже приведен пример тестирования сайта mail.ru с помощью этого сервиса.

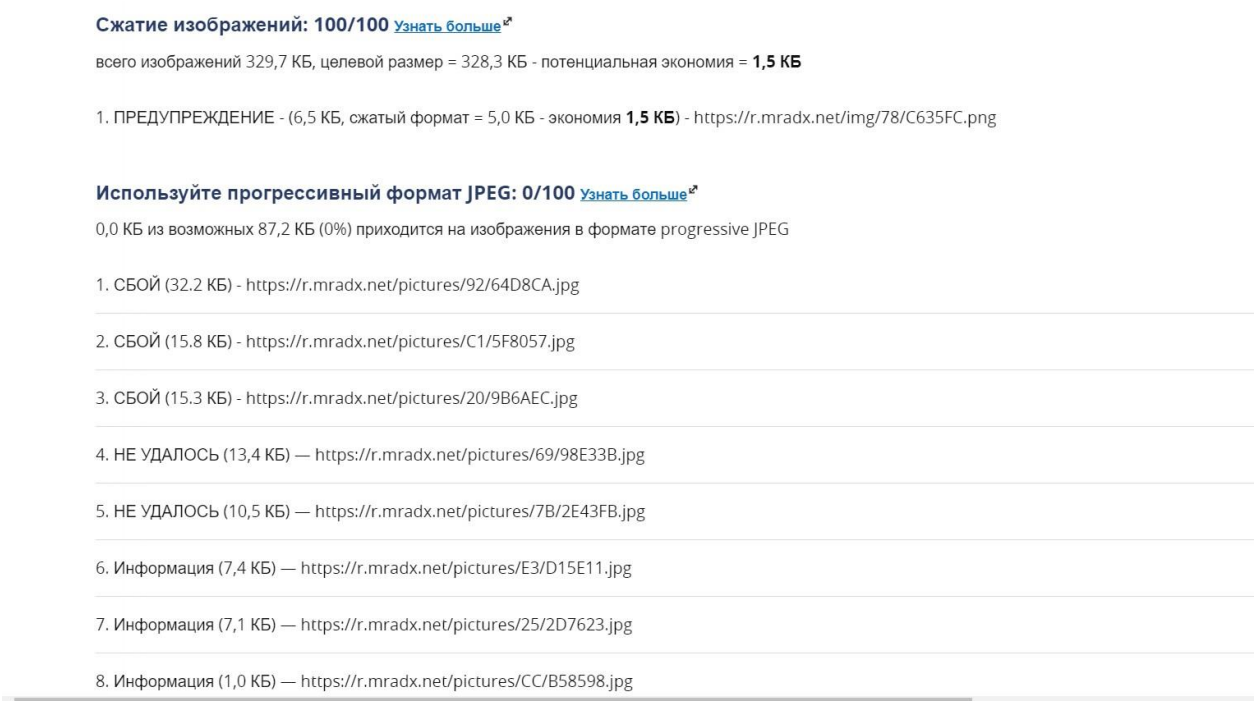
В разделе Compress Images отчёта WebPageTest перечислены картинки, которые можно сжать более эффективно, при этом оценивается потенциальный выигрыш по размеру файлов.



Нас интересует «Сжатие изображений» - зеленый квадратик слева с буквой «А»:

По клику на него откроется раздел Compress Images отчёта WebPageTest, где перечислены картинки, которые можно сжать более эффективно, при этом оценивается потенциальный выигрыш по размеру файлов.

2. Проведите анализ изображений сайта (согласно варианта задания) с помощью https://webspeedtest.cloudinary.com/. Поместите результат анализа и скрины в отчет.
3. Выберите любые 5 изображений с тестируемого сайта, которые нуждаются в оптимизации. Выполните их оптимизацию с помощью



https://imagecompressor.com/ru/. Приложите скрины по результатам сжатия в отчет. Заполните таблицу:

Картинка до сжатия	Картинка после сжатия	Размер до сжатия	Размер после сжатия	Коэффициент сжатия

4. Опишите режимы сжатия jpeg (базовый, прогрессивный и сжатие без потерь).
5. Чем отличаются jpeg и прогрессивный jpeg (PJPEG)?
6. Опишите тэг <picture>. Каким образом его использование, позволяет оптимизировать скорость загрузки сайта?