

## Правила создания документа JavaScript

1. Любой скрипт в HTML начинается с пары:

```
<script>  
...  
</script>
```

2. JS различает регистр.
3. Текст внутри скобок находится в кавычках, который представляет собой HTML.
4. Двойные и одинарные кавычки равноправны.
5. Внутри двойных кавычек ставятся одинарные.
6. Так как JS – объектно-ориентированный язык программирования, то при составлении скрипта будут присутствовать объекты (object), методы и свойства.

```
<script>  
document.write (“<FONT COLOR = ‘red’>Это красный текст</FONT>”)  
</script>
```

7. Вставка комментариев:

```
// в каждой строке;  
/* в начале и в конце*/
```

8. Прописывая строку JS ее необходимо уместить на одной строке редактора.
9. Каждая строка JS заканчивается «;»
10. Сочетание текста и команд требует «+» между элементами.

## Типы данных в JavaScript

В JavaScript существуют следующие примитивные типы данных:

1. Число «number». Единый тип «число» используется как для целых, так и для дробных чисел.

```
let n = 123;  
n = 12.345;
```

2. Строка «string». Тип «символ» не существует, есть только «строка».

```
let str = "Мама мыла раму";  
str = 'Одинарные кавычки тоже подойдут';
```

3. Булевый (логический) тип «boolean». У него всего два значения: true (истина) и false (ложь). Как правило, такой тип используется для хранения значения типа да/нет.

```
let checked = true; // поле формы помечено галочкой  
checked = false; // поле формы не содержит галочки
```

4. Специальное значение «null». Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null. В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

```
let age = null;
```

В частности, код говорит о том, что возраст age неизвестен.

5. Специальное значение «undefined». Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено». Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть undefined. В явном виде undefined обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого» или «неизвестного» значения используется null.

```
let x;  
alert( x ); // выведет "undefined"
```

Можно присвоить undefined и в явном виде, хотя это делается редко:

```
let x = 123;  
x = undefined;  
alert( x ); // "undefined"
```

6. Объекты «object», не является примитивным. Это отдельный тип. Он используется для коллекций данных и для объявления более сложных сущностей. Объявляются объекты при помощи фигурных скобок {...}.

```
let user = { name: "Вася" };
```

7. В стандарте ECMAScript 6 определен новый тип данных Symbol.

## Переменные в JavaScript

Тип переменной зависит от того, какой тип информации в ней хранится. *JavaScript* не является жестко типизированным языком. Это означает, что не нужно точно определять тип данных переменной, в момент ее создания. Тип переменной присваивается автоматически в процессе выполнения скрипта.

Если во время присвоения значения переменной число заключается в кавычки, то оно рассматривается как строковое, а не числовое значение.

Можно определить переменную следующим образом:

```
let answer = 42
```

А позже, можно присвоить той же переменной следующее значение:

```
answer = "Уже текст..."
```

Так как *JavaScript* не поддерживает никаких методов и свойств для определения типа текущего значения переменной, очень важно внимательно отслеживать типы переменных во избежание неожиданных результатов.

На имя переменной в JavaScript наложены всего два ограничения.

1. Имя может состоять из: букв, цифр, символов \$ и \_.
2. Первый символ не должен быть цифрой.

Целочисленные переменные могут быть десятичными (24), шестнадцатеричными (0x24) или восьмеричными (024), с плавающей запятой (5.14E6), булевы переменные (true, false).

Переменные могут быть локальными и глобальными.

## Основные операторы

### Сложение строк, бинарный +

Обычно при помощи плюса '+' складывают числа. Но если бинарный оператор '+' применить к строкам, то он их объединяет в одну, т.е. выполняет конкатенацию строк:

```
var a = "моя" + "строка";  
alert( a ); // моястрока
```

Если хотя бы один аргумент является строкой, то второй будет также преобразован к строке, причем не важно, справа или слева находится операнд-строка.

```
alert( '1' + 2 ); // "12"  
alert( 2 + '1' ); // "21"
```

Остальные арифметические операторы работают только с числами и всегда приводят аргументы к числу.

```
alert( 2 - '1' ); // 1  
alert( 6 / '2' ); // 3
```

### Преобразование к числу, унарный плюс +

Унарный, то есть применённый к одному значению, плюс ничего не делает с числами. Однако он широко применяется, так как он позволяет преобразовать значения в число.

Например, когда полученные значения в форме строк из HTML-полей или от пользователя необходимо сложить.

```
let apples = "2";  
let oranges = "3";
```

`alert (apples + oranges);` // результат "23", так как бинарный плюс складывает строки

```
let apples = "2";  
let oranges = "3";
```

`alert (+apples + +oranges);` // 5, число, оба операнда предварительно преобразованы в числа.

### Присваивание

Принцип работы такой же, как и в изученных ранее языках программирования. Возможно присваивание по цепочке:

```
let a, b, c;  
a = b = c = 2 + 2;  
alert( a ); // 4  
alert( b ); // 4  
alert( c ); // 4
```

Такое присваивание работает справа-налево, то есть сначала вычисляется самое правое выражение  $2+2$ , присвоится в `c`, затем выполнится `b = c` и, наконец, `a = b`.

### Взятие остатка %

Его результат `a % b` – это остаток от деления `a` на `b`.

### Деление на цело

Для взятия целой части от деления необходимо выполнить округление. Варианты округления:

```
num1 = 10, num2 = 3;  
Math.floor(num1/num2); // 3 (округление в меньшую сторону)  
Math.ceil(num1/num2); // 4 (округление в большую сторону)  
Math.round(num1/num2); // 3 (математическое округление)  
parseInt((num1/num2)); // 3 (приведение к числу, будет отброшена дробная часть)
```

```
<script>  
  //из числа 126 получаем 621  
  let a = 126;  
  a1 = (Math.floor(a/100));  
  a2 = (Math.floor(a/10))%10;  
  a3 = a%10;  
  alert ("Исходное число: " + a);  
  alert ("Результат: " + (a3*100 + a2*10 + a1));  
</script>
```

**Инкремент/декремент: ++, --**

Одной из наиболее частых операций в JavaScript, как и во многих других языках программирования, является увеличение или уменьшение переменной на единицу. Для этого существуют специальные операторы:

Инкремент ++ увеличивает на 1:

```
let i = 2;  
i++; // более короткая запись для i = i + 1.  
alert(i); // 3
```

Декремент -- уменьшает на 1:

```
let i = 2;  
i--; // более короткая запись для i = i - 1.  
alert(i); // 1
```

Инкремент/декремент можно применить только к переменной. Код 5++ даст ошибку.

Вызывать эти операторы можно не только после, но и перед переменной: i++ (называется «постфиксная форма») или ++i («префиксная форма»). Обе эти формы записи делают одно и то же: увеличивают на 1. Постфиксная форма i++ отличается от префиксной ++i тем, что возвращает старое значение, бывшее до увеличения.

```
// префиксная форма  
let i = 1;  
let a = ++i; // увеличит переменную, а затем вернёт ее значение в a  
alert(a); // 2
```

```
// постфиксная форма  
let i = 1;  
let a = i++;  
alert(a); // 1
```

**Сокращённая арифметика с присваиванием**

```
let n = 2;  
n += 5;  
n *= 2;  
alert(n); // 14
```

**Оператор запятая**

Обычно он используется в составе более сложных конструкций, чтобы сделать несколько действий в одной строке. Например:

```
// три операции в одной строке  
for (a = 1, b = 3, c = a*b; a < 10; a++) {  
  ...  
}
```

## Работа с консолью

Консоль даёт разработчику возможность писать JavaScript-код прямо в браузере, наблюдать за тем, что происходит на страницах, и управлять этими процессами.

Чтобы в браузере открыть консоль:

Chrome – Ctrl + Shift + i

Yandex - Ctrl + Shift + j

Или в меню → Дополнительно → Дополнительные инструменты → Консоль

Методы для вывода в консоль:

`console.log(переменная)`

```
<script>
    //из числа 126 получаем 621
    let a = 126;
    a1 = (Math.floor(a/100));
    a2 = (Math.floor(a/10))%10;
    a3 = a%10;
    console.log("Исходное число: " + a);
    console.log("Результат: " + (a3*100 + a2*10 + a1));
</script>
```

`console.table(переменная)`

```
<script>

    // Я в соцсетях
    const mySocial = {
        facebook: true,
        linkedin: true,
        flickr: true,
        instagram: true,
        VKontaktebadoo: false
    };
    console.log("Я в соцсетях");
    console.table(mySocial);
</script>
```

**Задание:** в данный скрипт добавьте еще 2-3 соцсети с соответствующим значением

## КТ № 1 (задания в файле JavaScript КТ.pdf)

## Операторы сравнения и логические значения.

Операторы сравнения следующие:

- Больше/меньше:  $a > b$ ,  $a < b$ .
- Больше/меньше или равно:  $a \geq b$ ,  $a \leq b$ .
- Равно  $a == b$ .
- Не равно  $!=$ .

Логические значения. Как и другие операторы, сравнение возвращает значение. Это значение имеет логический тип.

```
alert( 2 > 1 ); // true, верно  
alert( 2 == 1 ); // false, неверно  
alert( 2 != 1 ); // true
```

Логические значения можно использовать и напрямую, присваивать переменным, работать с ними как с любыми другими:

```
var a = true; // присваивать явно  
  
var b = 3 > 4; // или как результат сравнения  
alert( b ); // false  
  
alert( a == b ); // (true == false) неверно, выведет false
```

Сравнение строк. Строки сравниваются побуквенно. При этом сравниваются численные коды символов. JavaScript использует кодировку Unicode. В кодировке Unicode обычно код у строчной буквы больше, чем у прописной. Для корректного сравнения символы должны быть в одинаковом регистре.

В частности, код у символа Б больше, чем у А, поэтому и результат сравнения такой.

```
alert( 'Б' > 'А' ); // true
```

Если строка состоит из нескольких букв, то сначала сравниваются первые буквы, потом вторые, и так далее, пока одна не будет больше другой.

Если первая буква первой строки больше – значит первая строка больше, независимо от остальных символов. Если одинаковы – сравнение идёт дальше. При этом любая буква больше отсутствия буквы.

```
alert( 'Привет' > 'Прив' ); // true, так как 'е' больше чем "ничего"
```

Обычно значения, получаемые от посетителя, представлены в виде строк. Поэтому числа, полученные таким образом сравнивать нельзя, результат будет неверен.

```
alert( "2" > "14" ); // true, неверно, ведь 2 не больше 14  
2 оказалось больше 14 , потому что строки сравниваются посимвольно, а первый символ 2 больше 1.
```

Правильно было бы преобразовать их к числу явным образом, поставив перед ними +

```
alert( +"2" > +"14" ); // false, теперь правильно
```

Сравнение разных типов. При сравнении значений разных типов, используется числовое преобразование. Оно применяется к обоим значениям.

```
alert( '2' > 1 ); // true, сравнивается как 2 > 1  
alert( '01' == 1 ); // true, сравнивается как 1 == 1  
alert( false == 0 ); // true, false становится числом 0  
alert( true == 1 ); // true, так как true становится числом 1.
```

Строгое равенство. Для проверки равенства без преобразования типов используются операторы строгого равенства === (тройное равно) и !==. Если тип разный, то они всегда возвращают false.

```
alert( 0 === false ); // false, т.к. типы различны
```



## Взаимодействие с пользователем.

**Метод alert.** Выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК». Окно сообщения, которое выводится, является модальным окном. Слово «модальное» означает, что посетитель не может взаимодействовать со страницей, нажимать другие кнопки и т.п., пока не нажмёт на «ОК».

```
alert(сообщение)
```

```
alert ( "Привет" );
```

**Метод prompt** выводит окно со строкой ввода.

```
Prompt ("текст в окне", "текст в строке ввода")
```

Чтобы строка ввода оставалась чистой, не заполнять вторую пару кавычек. Если вторая пара кавычек отсутствует, то в строке появится слово `undefined`. Если текст, имеющийся в строке ввода не изменить и нажать ОК, то он отобразится на странице. Если текст в строке отсутствует и ее не заполнять, то после нажатия Отмена на странице появится слово `null`.

```
<script>
/* Скрипт предназначен для того, чтобы получить
от пользователя информацию и поместить ее на страницу */

let user_name = prompt ("Напишите свое имя", "Здесь");
document.write("Привет, " + user_name + "! Милости просим!");
</script >
```

**Метод confirm** выводит сообщение в модальном окне с двумя кнопками: "ОК" и "ОТМЕНА".

```
<script>
    if (confirm("Сказать привет?")) {
        alert("Привет!")
    } else {
        alert("Вы нажали кнопку отмена")
    }
</script >
```

Конкретное место, где выводится модальное окно с вопросом – обычно это центр браузера, и внешний вид окна выбирает браузер. Разработчик не может на это влиять.

**Задание:** Создайте скрипт, который спрашивает имя и выводит его.

## Условный оператор if

Условный оператор **if** имеет формат

```
If (условие)
    {
        Команда(ы)
    }
else
    {
        Команда(ы)
    }
```

Оператор **if** вычисляет и преобразует выражение в скобках к логическому типу. В логическом контексте: число 0, пустая строка "", null и undefined, а также NaN являются false, остальные значения – true.

```
if (0) { // 0 преобразуется к false, такое условие никогда не выполнится
    ...
}
```

```
if (1) { // 1 преобразуется к true, такое условие выполнится всегда
    ...
}
```

Можно передать уже готовое логическое значение, к примеру, заранее вычисленное в переменной:

*JavaScript. Раздаточный материал №39*

```
let cond = (year != 2011); // true/false
```

```
if (cond) {
    ...
}
```

### **Пример 1**

```
<script>
let year = prompt('В каком году появилась спецификация ECMA-262 5.1?', '');
if (year < 2011) {
    alert('Это слишком рано..');
} else if (year > 2011) {
    alert('Это поздновато..');
} else {
    alert('Да, точно в этом году!');
}
</script>
```

### **Решение задач:**

1. Скрипт из **Пример 1** оформить как внешний скрипт и вызвать его на выполнение.

2. Используя конструкцию if..else, напишите код, который получает значение prompt, а затем выводит alert:  
1, если значение больше нуля,  
-1, если значение меньше нуля,  
0, если значение равно нулю.
3. Напишите код, который будет спрашивать логин (prompt).  
Если посетитель вводит «Админ», то спрашивать пароль, если нажал отмена (escape) – выводить «Вход отменён», если вводит что-то другое – «Я вас не знаю».

Пароль проверять так. Если введён пароль «Чёрный Властелин», то выводить «Добро пожаловать!», иначе – «Пароль неверен», при отмене – «Вход отменён».

## КТ № 2 (задания в файле JavaScript КТ.pdf)

Смотрим видео:

[https://www.youtube.com/watch?v=dcnCI\\_-Uq4E&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=1](https://www.youtube.com/watch?v=dcnCI_-Uq4E&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=1)

<https://www.youtube.com/watch?v=3UEoc5iWZUw&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=2>

## Тема: Логические операторы.

Логические операторы могут применяться к значениям любого типа и возвращают также значения любого типа.

### Логический оператор || (ИЛИ)

Оператор ИЛИ возвращает то значение, на котором остановились вычисления, причём, не преобразованное к логическому типу, т.е. оператор ИЛИ вычисляет ровно столько значений, сколько необходимо – до первого true.

#### Раздаточный материал №43

```
result = a || b;
```

```
alert( true || true ); // true
alert( false || true ); // true
alert( true || false ); // true
alert( false || false ); // false
```

Если значение не логического типа – то оно к нему приводится в целях вычислений.

#### Раздаточный материал №44

```
if (1 || 0) { // сработает как if( true || false ), число 1 будет воспринято как true, а 0 – как false
  alert( 'верно' );
}
```

Обычно оператор ИЛИ используется в if, чтобы проверить, выполняется ли хотя бы одно из условий.

#### Раздаточный материал №45

```
let hour = 12,
    isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
  alert( 'Офис до 10 или после 18 или в выходной закрыт' );
}
```

JavaScript вычисляет несколько ИЛИ слева направо. Если первый аргумент – true, то результат заведомо будет true (хотя бы одно из значений – true), и остальные значения игнорируются. Если все значения «ложные», то ИЛИ возвратит последнее из них

#### Раздаточный материал №46

```
let x;
true || (x = 1);
alert(x); // undefined, x не присвоен

let x;
false || (x = 1);
alert(x); // 1
```

```
let undef; // переменная не присвоена, т.е. равна undefined
let zero = 0;
let emptyStr = "";
let msg = "Привет!";
let result = undef || zero || emptyStr || msg || 0;
alert( result ); // выведет "Привет!" - первое значение, которое является true
```

```
alert( undefined || " || false || 0 ); // 0
```

## Логический оператор && (И)

*Раздаточный материал №47*

```
result = a && b;
```

```
alert( true && true ); // true
alert( false && true ); // false
alert( true && false ); // false
alert( false && false ); // false
```

*Раздаточный материал №48*

```
let hour = 12,
    minute = 30;

if (hour == 12 && minute == 30) {
    alert( 'Время 12:30' );
}
```

В логическом И допустимы любые значения. Если левый аргумент – false, оператор И возвращает его и заканчивает вычисления. Иначе – вычисляет и возвращает правый аргумент. Можно передать и несколько значений подряд, при этом возвратится первое «ложное» (на котором остановились вычисления), а если его нет – то последнее. Оператор && вычисляет операнды слева направо до первого «ложного» и возвращает его, а если все истинные – то последнее значение. Приоритет оператора И больше, чем ИЛИ.

*Раздаточный материал №49*

```
if (1 && 0) { // вычислится как true && false
    alert( 'не сработает, т.к. условие ложно' );
}
```

```
// Первый аргумент - true,
// Поэтому возвращается второй аргумент
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5
```

```
// Первый аргумент - false,
// Он и возвращается, а второй аргумент игнорируется
alert( null && 5 ); // null
alert( 0 && "не важно" ); // 0
```

```
alert( 1 && 2 && null && 3 ); // null
```

```
alert( 1 && 2 && 3 ); // 3
```

```
alert( 5 || 1 && 0 ); // 5
```

**Логический оператор ! (НЕ)** получает один аргумент. Сначала приводит аргумент к логическому типу true/false. Затем возвращает противоположное значение.

*Раздаточный материал №50*

```
let result = !value;
```

```
alert( !true ); // false
```

```
alert( !0 ); // true
```

## Тема: Работа с формами

Форма предназначена для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Документ может содержать любое количество форм, но одновременно на сервер может быть отправлена только одна форма. По этой причине данные форм должны быть независимы друг от друга.

Устанавливают форму на веб-странице теги `<form>` и `</form>`. Тег имеет следующие атрибуты:

**accept-charset** - устанавливает кодировку, в которой сервер может принимать и обрабатывать данные, например Windows-1251, UTF-8 и др.

**action** - адрес программы или документа (обработчик), который обрабатывает данные формы. В качестве обработчика может выступать CGI-программа или HTML-документ, который включает в себя серверные сценарии. После выполнения обработчиком действий по работе с данными формы он возвращает новый HTML-документ. Если атрибут `action` отсутствует, текущая страница перезагружается, возвращая все элементы формы к их значениям по умолчанию. В качестве значения принимается полный или относительный путь к серверному файлу (URL).

HTML.  
Раздаточный  
материал №53

```
<form action="handler.php">  
  <p>...</p>  
</form>
```

HTML.  
Раздаточный  
материал №54

```
<form action="handler.php" method="post">
```

**name** – уникальное имя формы, которое чаще всего используется для обращения к форме из скриптов.

Допускается внутри контейнера `<form>` помещать другие теги, при этом сама форма никак не отображается на веб-странице, видны только ее элементы и результаты вложенных тегов.

## Элементы формы

На форме могут быть определены следующие элементы управления: кнопка, флажок, радиокнопка, меню, строка текста, пароль, скрытое поле, файл, объект. Основной атрибут тега `<input>`, определяющий вид элемента — `type`.

### Строка текста

Позволяет вводить короткие фрагменты текста – имена, адреса и т.п. При его создании используется `name` для указания имени, которое поможет различить его в наборе данных на сервере. Наименование поля может состоять из букв, цифр и `_`.

Атрибуты:

`size` – ширина строки текста.

maxlength – ограничивает емкость строки текста.

value – начальный текст отображаемый в поле.

HTML.

Раздаточный  
материал №55

```
<br> Ключевые слова: <input type="text" name="key" size="30"
maxlength="25">
```

type="password" – строка текста для хранения конфиденциальной информации.

### Многострочный текст

<textarea> – создает область ввода многострочного текста.

name – имя области ввода.

rows, cols – ширина и высота области ввода

wrap= задает правила переноса целых слов текста:

soft – включает функцию переноса, но не воздействует на внешний вид данных, передаваемых на сервер.

hard – форматирование данных как на экране, так и во внутреннем буфере браузера.

HTML.

Раздаточный  
материал №56

```
<textarea name="unit" rows="5" cols="45" wrap="hard">
</textarea>
```

### Флажок

type="checkbox"

name – имя группы элементов-флажков.

value – служит для назначения имени каждому флажку в группе. Имя будет включено в итоговый набор данных формы, если пользователь выберет соответствующий флажок.

В значениях name и value используются только буквы, цифры и \_.

checked – позволяет установить флажок предварительно, чтобы при воспроизведении страницы опция выбиралась автоматически.

HTML.

Раздаточный  
материал №57

```
<br> Установите атрибуты файла
<br><input type="checkbox" name="flag" value="sys" checked>
системный
<br><input type="checkbox" name="flag" value="archive">архивный
<br><input type="checkbox" name="flag" value="read">только чтение
```

### Переключатель

Используется при выборе только одной опции. Создание аналогично флажку. Type="radio".

### Меню

name, value – аналогично флажку.

size – число элементов меню отображаемых одновременно (по умолчанию – один элемент).

<option> – определяет отдельный элемент меню.

selected – автоматический выбор опции.



HTML.  
Раздаточный  
материал №58

```
<br> Какой цвет Вам нравится?
<br><select name="color" size="3">
<option value="blue">Синий
<option value="red" selected >Красный
<option value="yellow">Желтый
</select>
<br><br><br><input type="submit" value="отправить">&nbsp;
<input type="reset" value="очистить">
```

Кнопка submit служит для завершения пользовательского сеанса заполнения формы и отправки данных на сервер.

Кнопка reset очищает все поля формы и возвращает их к исходному состоянию.

Value – определяет текст, отображаемый на поверхности кнопки (по умолчанию submit/reset).

Кроме кнопки Submit для отправки формы на сервер можно нажать клавишу Enter в пределах формы. Если кнопка Submit отсутствует в форме, клавиша Enter имитирует ее использование, но только в том случае, когда в форме имеется только один элемент <input>. Если таких элементов два и более, нажатие на <Enter> не вызовет никакого результата.

Второй способ создания кнопки основан на использовании тега <button>.

Button позволяет отображать разнообразные элементы в виде кнопок. Кнопки могут быть созданы с использованием всех возможностей, доступных в HTML и таблицах стилей (Поддерживается не всеми браузерами).

HTML.  
Раздаточный  
материал №59

```
<button style="font-family:Arial; font-size:16pt; color:navy">добавьте
информацию<span style="font-style:italic; color:green> Пожалуйста!
</span>
</button>
```

HTML. Раздаточный материал №60

Вариант 1	<a href="https://professorweb.ru/my/html/html5/level2/files/img46023.jpg">https://professorweb.ru/my/html/html5/level2/files/img46023.jpg</a>
Вариант 2	<a href="https://cf.ppt-online.org/files/slide/v/vRAIxeaYzgFDVC2krBdPXQtZbW5McoLJ08ln6q/slide-21.jpg">https://cf.ppt-online.org/files/slide/v/vRAIxeaYzgFDVC2krBdPXQtZbW5McoLJ08ln6q/slide-21.jpg</a>
Вариант 3	<a href="https://lh5.googleusercontent.com/-wG_YHAibVZU/Ud696wJg0FI/AAAAAAAAACP4/ealzPTZRixE/w596-h642-no/4_3.png">https://lh5.googleusercontent.com/-wG_YHAibVZU/Ud696wJg0FI/AAAAAAAAACP4/ealzPTZRixE/w596-h642-no/4_3.png</a>
Вариант 4	<a href="https://www.devlounge.net/wp-content/uploads/2011/02/24ways.jpg">https://www.devlounge.net/wp-content/uploads/2011/02/24ways.jpg</a>
Вариант 5	<a href="https://i.pinimg.com/originals/73/c6/0d/73c60def8c55043f9fd27b370530a9cf.jpg">https://i.pinimg.com/originals/73/c6/0d/73c60def8c55043f9fd27b370530a9cf.jpg</a>
Вариант 6	<a href="https://smashinghub.com/wp-content/uploads/2011/08/html5-forums-14.jpg">https://smashinghub.com/wp-content/uploads/2011/08/html5-forums-14.jpg</a>
Вариант 7	<a href="https://soulcompas.com/wp-content/uploads/2020/06/html5-admin-template-free.jpg">https://soulcompas.com/wp-content/uploads/2020/06/html5-admin-template-free.jpg</a>
Вариант 8	<a href="https://www.webasyst.ru/wa-data/public/updates/img/09/209/7355/7355.970.jpg">https://www.webasyst.ru/wa-data/public/updates/img/09/209/7355/7355.970.jpg</a>
Вариант 9	<a href="https://www.zoho.com/creator/images/subpages/htmlforms/workshop_registration.gif">https://www.zoho.com/creator/images/subpages/htmlforms/workshop_registration.gif</a>
Вариант 10	<a href="https://bramus.github.io/ws1-sws-course-materials/assets/03/testform.jpg">https://bramus.github.io/ws1-sws-course-materials/assets/03/testform.jpg</a>
Вариант 11	<a href="https://www.bestfree.ru/uslugi/constructors/SozданиеSaytaNaUcoz_3.png">https://www.bestfree.ru/uslugi/constructors/SozданиеSaytaNaUcoz_3.png</a>
Вариант 12	<a href="http://ip-whois.net/forma-obratnoj-svyazi/img/form1.jpg">http://ip-whois.net/forma-obratnoj-svyazi/img/form1.jpg</a>
Вариант 13	<a href="https://www.digiseller.ru/preview/125077/p1_30116215520413.JPG">https://www.digiseller.ru/preview/125077/p1_30116215520413.JPG</a>
Вариант 14	<a href="https://mob25.com/images/Perl/images/ris150_1.jpg">https://mob25.com/images/Perl/images/ris150_1.jpg</a>
Вариант 15	<a href="https://i2.wp.com/ps.w.org/ultimate-form-builder-lite/assets/screenshot-1.png">https://i2.wp.com/ps.w.org/ultimate-form-builder-lite/assets/screenshot-1.png</a>
Вариант 16	<a href="https://i.stack.imgur.com/px0jW.png">https://i.stack.imgur.com/px0jW.png</a>

## Тема: Функции в JavaScript.

При создании скрипта разумно в нем выделить логически независимые части – функции.

### Раздаточный материал №61

```
function F(V)
{
  S
}
```

где F – имя функции, по которому к ней можно обращаться,  
V – список параметров, разделенных запятыми,  
S – тело функции (операторы).

Описание функции не может быть вложено в описание другой функции. Начальные значения параметрам присваиваются при обращении к функции. Если описание функции имеет вид

### Раздаточный материал №62

Function F(V1,V2,...,Vn) {S},  
то вызов функции должен иметь вид F(e1,e2,...,en),  
где e1,e2,...,en - выражения, задающие фактические значения параметров.

Параметры V1,V2,...,Vn – формальные параметры, которые получают смысл только после задания в функции фактических параметров e1,e2,...,en, с которыми функция затем и работает. Если в функции параметры отсутствуют, то описание функции следующее:

### Раздаточный материал №63

```
Function F()
{
  S
}
```

где F – имя функции, по которому к ней можно обращаться,  
S – тело функции (операторы).

Вызов функции:

1. Указав ее имя в качестве оператора JavaScript.
2. Указав ее имя в HTML-ссылке  
`<a href="javascript:hello()"> ... </A>`
3. В адресной строке  
`javascript:hello()`
4. В форме при помощи атрибута action

### Раздаточный материал №64

```
<html>
<head>
<script>
function hello() {
```

```

document.write("Привет и добро пожаловать");
}
</script>
</head>
<body>
<form action="javascript:hello()">
<input type="submit" value="Отобразить приветствие">
</form>
</body>
</html>

```

#### Решение задач. Раздаточный материал №65

1. Переработать функцию из раздаточного материала № 64 так, чтобы она сначала спрашивала имя пользователя, а затем выводила сообщение "Привет *имя\_пользователя* и добро пожаловать".

2. Создать скрипт, в котором функция спросит имя пользователя и поприветствует его, а если пользователь не ввел имя, т.е. нажал «Отмена», то выдаст сообщение об этом.

### События и обработчики событий.

Кроме объектов и методов в JS имеются события и обработчики событий. Но они «встроены» HTML-код, а не существуют самостоятельно, как скрипты, т.е. не требуют тегов `<script>` и `</script>`. Это область взаимодействия страницы и пользователя.

Обработчики событий представляют собой небольшие подпрограммы, связывающие действия пользователей со сценариями, которые необходимо выполнить в ответ на эти действия. Обработчики событий можно помещать в HTML-теги и в атрибуты.

Событие **onMouseOver** срабатывает при наведении мыши на объект. Например, на гипертекстовую ссылку: используется внутри ее, при этом формат ссылки остается без изменений; обработчик ставится сразу же после адреса URL. Событие приводится в действие, как только будет распознано браузером.

Событие **onMouseOut** срабатывает после ухода мыши с объекта.

#### JavaScript. Раздаточный материал № 66

```

<a href="les10.htm" onMouseOver="alert('Уходи в сторону!')" onMouseOut="alert('Так-то лучше, спасибо')"> Наведите курсор на эту ссылку и уведите обратно</a>

```

Событие **onClick** срабатывает при однократном нажатии мыши.

#### JavaScript. Раздаточный материал № 67

```

<a href="http://www.jsp.newmail.ru" onClick="alert('Уже уходите!');"> Жмите сюда</a>

<input type="button" onclick="this.style.display='none'" value="Нажми, чтобы спрятать"/>

```

Событие **ondblClick** срабатывает при двойном нажатии мыши.

Событие **onMouseDown** срабатывает когда нажимается кнопка мыши.

Событие **onMouseUp** срабатывает когда кнопка мыши отпускается.

Событие **onFocus** срабатывает когда пользователь «фокусируется» на элементе страницы. Эта команда может использоваться для форм: флажков и текстовых полей.

#### JavaScript. Раздаточный материал № 68

```

<form>
<input type="text" size="30" onFocus="alert ('Вы остановились на элементе')">
</form>

```

Событие **onBlur** срабатывает при потере «фокуса», т.е. когда пользователь переходит к другому элементу

*JavaScript.Раздаточный материал № 69*

```
<form>
<input type="text" size="45" value="Впишите свое имя и щелкните по другой строке"
onBlur="alert('Вы изменили ответ — уверены, что он правильный?');">
</form>
```

Событие **onChange** срабатывает при изменении объекта. Обработчик события onChange полезен для проверки правильности заполнения форм, поскольку это событие наступает при каждом изменении элемента формы.

Например, если в форме присутствует текстовое поле, запрашивающее у пользователя целочисленное значение от 0 до 9, можно создать функцию, которая будет проверять правильность значения, введенного в текстовое поле.

*JavaScript.Раздаточный материал № 70*

```
<html>
<head>
<script>
function intCheck() {
val=document.form1.text1.value;
if (val<0||val>9)
alert("Это значение недопустимо. Введите целое число в пределах от 0 до 9");
}
</script>
</head>
<body>
<form name= "form1">
<input type="text" name="text1" onChange="intCheck()">
</form>
</body>
</html>
```

*JavaScript.Раздаточный материал № 71*

```
<form>
<input TYPE="text" size="45" value="Измените текст и щелкните по другой строке"
onChange="alert('Текст был изменен')">
</form>
```

Событие **onSelect** срабатывает при выделении объекта.

Событие **onSubmit** (отослать, отправить) срабатывает при нажатии кнопки Submit.

*JavaScript.Раздаточный материал № 72*

```
<form>
<input TYPE="submit" onSubmit="parent.location='thanksalot.html'">
</form>
```

parent.location – это стандартная схема ссылки на другую страницу.  
parent – свойство окна браузера.  
location – объект, который появиться в этом окне

### Взаимодействие форм и JavaScript.

Часто взаимодействие формы и JavaScript используется для проверки ввода данных. При организации взаимодействия форм и JavaScript необходимо следить за иерархией объектов:

#### JavaScript. Раздаточный материал № 73

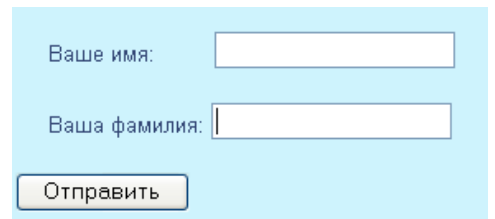
document.myform.fname.value

↑      ↑      ↑      ↑  
объект объект объект свойство  
документ форма поле формы

### Примеры взаимодействия форм и JavaScript

#### JavaScript. Раздаточный материал № 74

```
<html>
<head>
<script>
function doit()
{
  alert("document.myform.fname.value — это " + document.myform.fname.value)
  var greeting="Привет, "
  alert(greeting + document.myform.fname.value + " " + document.myform.lname.value)
  alert("Количество букв в имени " + document.myform.fname.value.length)
}
</script>
</head>
<body>
<form name="myform">
  Ваше имя:
  <input type="text" name="fname"><p>
  Ваша фамилия:
  <input type="text" name="lname"><p>
  <input type="button" value="отправить" onClick="doit()">
</form>
</body>
</html>
```



Ваше имя:

Ваша фамилия:

#### JavaScript. Раздаточный материал № 75

```
<html>
<head>
<script>
function newcolor(color)
{
  alert("Вы выбрали " + color);
}
```

```
document.bgColor=color
}
</script>
</head>
<body>
<p>Выбрать цвет фона</p>
<form>
<input type="button" value="голубой" onClick="newcolor('lightblue')">
<input type="button" value="розовый" onClick="newcolor('pink')">
<input type="button" value="вернуть" onClick="newcolor('white')">
</form>
</body>
</html>
```

### **КТ № 3 (задания в файле JavaScript КТ.pdf)**

## Оператор switch

Используется в тех случаях, когда при решении задачи требуется выбрать один вариант действия из нескольких возможных. Синтаксис оператора:

*Раздаточный материал № 77*

```
switch (n)
{
  case L1: операторы; break;
  case L2: операторы; break;
  ...
  case Ln: операторы; break;
  default: операторы
}
```

Оператор break обеспечивает завершения работы оператора switch после выполнения очередного варианта.

Оператор break может отсутствовать. В этом случае операторы будут выполняться от L1 до конца оператора switch.

*Раздаточный материал № 78*

```
<!DOCTYPE html>
<script>

let a = Number(prompt('Введи число'));

switch (a) {
  case 3:
    alert( 'Маловато' );
    break;
  case 4:
    alert( 'В точку!' );
    break;
  case 5:
    alert( 'Перебор' );
    break;
  default:
    alert( "Нет таких значений" );
}
</script>
```

*Раздаточный материал № 79*

Перепишите код с использованием одной конструкции switch:

```
const number = +prompt('Введите число между 0 и 3, ');

if (number === 0) {
  alert('Вы ввели число 0');
}
```

```
if (number === 1) {  
    alert('Вы ввели число 1');  
}  
  
if (number === 2 || number === 3) {  
    alert('Вы ввели число 2, а может и 3');  
}
```

### **Решение**

```
const number = +prompt('Введите число между 0 и 3, ');  
  
switch (number) {  
    case 0:  
        alert('Вы ввели число 0');  
        break;  
  
    case 1:  
        alert('Вы ввели число 1');  
        break;  
  
    case 2:  
    case 3:  
        alert('Вы ввели число 2, а может и 3');  
        break;  
}
```

### **КТ № 4 часть 1 (задания в файле JavaScript КТ.pdf)**



## Циклы в JavaScript.

JavaScript поддерживает три вида циклов: for, while, do..while.

*Раздаточный материал № 80*

```
For (начальное значение; конечное значение; шаг)
{
  Тело цикла
}
```

*JavaScript. Раздаточный материал № 81*

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Кролик номер " + i);
    }
  }
</script>
</head>
<body>
  <input type="button" onclick="countRabbits()" value="Считать кроликов!"/>
</body>
</html>
```

### Пропуск частей for

Любая часть for может быть пропущена. Например, можно убрать начало.

*Раздаточный материал № 82*

```
let i = 0;

for (; i < 3; i++) {
  alert(i); // 0, 1, 2
}
```

Можно убрать и шаг:

*JavaScript. Раздаточный материал № 83*

```
let i = 0;
for (; i < 3;) {
  alert(i);
  // цикл превратился в аналог while (i<3)
}
```

А можно и вообще убрать всё, получив бесконечный цикл:

#### *Раздаточный материал № 84*

```
for (;;) {  
  // будет выполняться вечно  
}
```

При этом сами точки с запятой ; обязательно должны присутствовать, иначе будет ошибка синтаксиса.

Цикл **while** имеет вид:

#### *Раздаточный материал № 85*

```
while (условие) {  
  // код, тело цикла  
}
```

Пока условие верно – выполняется код из тела цикла. Условие в скобках интерпретируется как логическое значение, поэтому вместо while (i!=0) обычно пишут while (i)

#### *Раздаточный материал № 86*

```
let i = 0;  
while (i < 3) {  
  alert( i );  
  i++;  
}
```

```
let i = 3;  
while (i) { // при i, равном 0, значение в скобках будет false и цикл остановится  
  alert( i );  
  i--;  
}
```

Цикл **do...while** имеет вид:

Цикл сначала выполняет тело, а затем проверяет условие.

#### *Раздаточный материал № 87*

```
do {  
  // тело цикла  
} while (условие);
```

#### *Раздаточный материал № 88*

```
let i = 0;  
do {  
  alert( i );  
  i++;  
} while (i < 3);
```

В цикле также можно определить переменную:

*JavaScript. Раздаточный материал № 89*

```
for (let i = 0; i < 3; i++) {  
  alert(i); // 0, 1, 2  
}
```

Эта переменная будет видна и за границами цикла, в частности, после окончания цикла `i` станет равно 3.

### Прерывание цикла.

Оператор **break** автоматически завершает цикл и выполняет первый оператор, следующий за оператором цикла. При этом все переменные остаются в том состоянии, в котором они были при завершении цикла.

*Раздаточный материал № 90*

```
let sum = 0;  
While (true) {  
  let value = +prompt("Введите число", "");  
  if (!value) break; // (*)  
  sum += value;  
}  
alert( 'Сумма: ' + sum );
```

Директива **continue** прекращает выполнение текущей итерации цикла, когда понятно, что на текущей итерации цикла больше выполнять ничего не нужно.

*JavaScript. Раздаточный материал № 91*

```
for (let i = 0; i < 10; i++) {  
  if (i % 2 == 0) continue;  
  alert(i);  
}
```

Для чётных `i` срабатывает `continue`, выполнение тела прекращается и управление передаётся на следующий проход `for`.

Нельзя использовать `break/continue` справа от оператора „?“

**Метки** для `break/continue` позволяют выйти одновременно из нескольких уровней цикла.

*Раздаточный материал № 92*

```
outer: for (let i = 0; i < 3; i++) {  
  for (var j = 0; j < 3; j++) {  
    let input = prompt("Значение в координатах 'i+', 'j', ");  
    // если отмена ввода или пустая строка -  
    // завершить оба цикла  
    if (!input) break outer; // (*)  
  }  
}  
alert('Готово!');
```

Будет разорван самый внешний цикл и управление перейдёт на `alert`

Метка имеет вид "имя:" (имя должно быть уникальным) и ставится перед циклом или на отдельной строке. Вызов `break outer` ищет ближайший внешний цикл с такой меткой и переходит в его конец.

Директива `continue` также может быть использована с меткой, в этом случае управление перейдет на следующую итерацию цикла с меткой.

Метки не позволяют перейти в произвольное место кода, в JavaScript нет такой возможности.

<https://www.youtube.com/watch?v=oc6HYRgY5uw&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=7>

## **КТ № 4 часть 2 (задания в файле JavaScript КТ.pdf)**

## Одномерные массивы с числовыми индексами

Массив представляет собой набор элементов, доступ к которым осуществляется по индексу. Один массив может включать несколько различных типов переменных. Все элементы массива пронумерованы, начиная с нуля. Для получения значения элемента массива необходимо задать имя массива и в квадратных скобках – порядковый номер элемента.

### Раздаточный материал № 93

```
let land = [ ] //пустой массив

let land = [“Планета”, “24 часа”] // массив land с двумя элементами

alert(land[1]) // результат – 24 часа

land[3] = 6378 //добавлен третий элемент – радиус, теперь
[“Планета”, “24 часа”, 6378]

land[3] = 6371 //замена значения третьего элемента, теперь
[“Планета”, “24 часа”, 6371]

alert(land) // вывести массив целиком, через запятую
Планета, 24 часа, 6371
```

В массиве могут храниться элементы любого типа.

### Раздаточный материал № 94

```
// разные типы значений
let arr = [ 'Яблоко', { name: 'Джон' }, true, function() { alert('привет'); } ];

// получить элемент с индексом 1 (объект) и затем показать его свойство
alert( arr[1].name ); // Джон

// получить элемент с индексом 3 (функция) и выполнить её
arr[3](); // привет
```

### Методы объекта Array

#### Раздаточный материал № 95

sort	сортирует элементы массива
reverse	переставляет элементы в обратном порядке
push	добавляет один или несколько элементов в конец массива
pop	удаляет последний элемент из массива
unshift	добавляет один или несколько элементов в начало массива
shift	удаляет первый элемент массива
join	соединяет элементы массива в одну строку
concat	объединяет два массива в один
split	обеспечивает разделение строк (подробнее см. РМ № 82)
slice	копирует часть массива от начального до конечного элемента. Конечный элемент не копируется. Исходный массив не изменяется.

```
<script>
let land = ["Планета", "24 часа", 6371];
alert(land.pop() ); // удален 6371
alert(land); // Планета, 24 часа
</script>
```

```
<html>
  <body>
    <script>
      //1
      let names = 'Иванов, Петров, Сидоров';
      let fio = names.split(', ');
      console.log('Первый способ');
      for (let f of fio) {
        console.log('Привет, товарищ ' + f);
      }
    </script>

    <script>
      //2
      let names1 = 'Иванов, Петров, Сидоров';
      let fio1 = names1.split(', ');
      console.log('Второй способ');
      for (let i = 0; i < fio1.length; i++) {
        console.log('Привет, товарищ ' + fio1[i]);
      }
    </script>

  </body>
</html>
```

```
<script>
let str = "слово";
alert(str.split(""));
</script>
```

```
<script>
let arr = ['Иванов', 'Петров', 'Сидоров'];
let str = arr.join(';');
alert(str);
</script>
```

```
<script>
let arr = ["воскресенье", "понедельник", "вторник", "среда", "четверг", "пятница", "суббота"];
let arr2 = arr.slice(2, 5);
alert(arr2); //вторник, среда, четверг
</script>
```

Свойство `length` позволяет определить число элементов в массиве.

Найти сумму пяти элементов одномерного массива. Результат вывести в консоль.

```
<html>
  <body>
    <script>
      let number = [1, 2, 3, 4, 5];
      let summ = 0;
      let i;

      for (i = 0; i < 5; i++) {
        summ += number[i];
      }
      console.log('Сумма элементов массива = ' + summ);
    </script>
  </body>
</html>
```

**КТ № 4 часть 3 (задания в файле JavaScript КТ.pdf)**

## Строки

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Внутренний формат для строк — всегда UTF-16, вне зависимости от кодировки страницы.

В JavaScript есть разные типы кавычек. Строку можно создать с помощью одинарных ('), двойных (") либо обратных (`) кавычек:

Раздаточный материал № 97

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

Одинарные и двойные кавычки работают, по сути, одинаково. Одним из преимуществ, обратных кавычек — они могут занимать более одной строки:

Раздаточный материал № 98\*

```
let guestList = `Guests:
* John
* Pete
* Mary
`;
alert(guestList); // список гостей, состоящий из нескольких строк
```

Если попытаться использовать точно так же одинарные или двойные кавычки, то будет ошибка.

Свойство `length` содержит длину строки:

Раздаточный материал № 99\*

```
<script>
  alert("Это строка".length) // выведет 10
</script>
```

*Доступ к символам*

Получить символ, который занимает позицию `pos`, можно с помощью квадратных скобок: `[pos]`. Первый символ занимает нулевую позицию:

Раздаточный материал № 100\*

```
let str = `Hello`;
// получаем первый символ
alert( str[0] ); // H
```

Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине и заменить его. Как только строка создана — она такая навсегда.

Раздаточный материал № 101

```
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.



#### Раздаточный материал № 102\*

```
<script>
let str = 'Hi';
str = 'h' + str[1]; // заменяем строку
alert( str ); // hi
</script>
```

#### *Изменение регистра*

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

#### Раздаточный материал № 103\*

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

Если необходимо перевести в нижний регистр какой-то конкретный символ:

#### Раздаточный материал № 104\*

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

#### *Поиск подстроки*

Существует несколько способов поиска подстроки.

Метод — str.indexOf(substr, pos). Он ищет подстроку substr в строке str, начиная с позиции pos, и возвращает позицию, на которой располагается совпадение, либо -1 при отсутствии совпадений.

#### Раздаточный материал № 105\*

```
let str = 'Widget with id';
alert( str.indexOf('Widget') ); // 0, потому что подстрока 'Widget' найдена в начале
alert( str.indexOf('widget') ); // -1, совпадений нет, поиск чувствителен к регистру
alert( str.indexOf("id") ); // 1, подстрока "id" найдена на позиции 1 (..idget with id)
```

Необязательный второй аргумент позволяет начать поиск с определённой позиции.

#### Раздаточный материал № 106\*

Для того, чтобы найти следующее вхождение "id", начнём поиск с позиции 2:

```
let str = 'Widget with id';
alert( str.indexOf('id', 2) ); // 12
```

Метод str.lastIndexOf(substr, position) - ищет подстроку с конца строки к её началу. Он используется тогда, когда нужно получить самое последнее вхождение: перед концом строки или начинающееся до (включительно) определённой позиции.

#### *Получение подстроки*

В JavaScript есть 3 метода для получения подстроки: substring, substr и slice.

Метод str.slice(start [, end]) - возвращает часть строки от start до (не включая) end.

#### Раздаточный материал № 107\*

```
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. только один символ на позиции 0
```

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

Раздаточный материал № 108\*

```
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:

Раздаточный материал № 109\*

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

Метод str.substring(start [, end]) - возвращает часть строки между start и end. Это — почти то же, что и slice, но можно задавать start больше end.

Раздаточный материал № 110\*

```
let str = "stringify";
// для substring эти два примера — одинаковы
alert(str.substring(2, 6) ); // "ring"
alert(str.substring(6, 2) ); // "ring"
```

```
// ...но не для slice:
alert(str.slice(2, 6) ); // "ring" (то же самое)
alert(str.slice(6, 2) ); // "" (пустая строка)
```

Отрицательные значения substring, в отличие от slice, не поддерживает, они интерпретируются как 0.

Метод str.substr(start [, length]) - возвращает часть строки от start длины length. В противоположность предыдущим методам, этот позволяет указать длину вместо конечной позиции:

Раздаточный материал № 111\*

```
let str = "stringify";
alert(str.substr(2, 4) ); // ring, получаем 4 символа, начиная с позиции 2
```

Значение первого аргумента может быть отрицательным, тогда позиция определяется с конца:

Раздаточный материал № 112\*

```
let str = "stringify";
// gi, получаем 2 символа, начиная с позиции 4 с конца строки
alert( str.substr(-4, 2) );
```

Раздаточный материал № 113\*

Задания:

1. Выполнить все раздаточные материалы, помеченные (\*).
2. Выполнить упражнения:

- Создать строковую переменную, состоящую из нескольких строк (Films: The Terminator, Twilight, The Fast and the Furious, The Lord of the Rings).
- Создать строковую переменную fraza = “Все произведения Достоевского актуальны в настоящее время”
- Найти длину полученной строки fraza.
- Последовательно вывести в консоль символы строковой переменной fraza, стоящие на позициях 1, 7, 14, 17, 21.
- В строковой переменной fraza символы из верхнего регистра перевести в нижний.
- В строковой переменной fraza символы из нижнего регистра перевести в верхний.
- В строковой переменной fraza найти номер позиции слова «Достоевский» (str.indexOf(substr, pos)).
- Из строковой переменной fraza последовательно вывести на экран (alert) все слова, при этом использовать методы substring, substr и slice.

### 3. Просмотр видео (2 шт.) по адресу:

<https://www.youtube.com/watch?v=Rvn5x8vGGGg&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=17>  
[https://www.youtube.com/watch?v=-M\\_10IJV1wk&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=14](https://www.youtube.com/watch?v=-M_10IJV1wk&list=PLA0M1Bcd0w8x9TltCzZDhw0SatK1d10yy&index=14)

#### *Сравнение строк.*

Строки сравниваются побуквенно. При этом сравниваются численные коды символов. Код у строчной буквы больше, чем у прописной. Для корректного сравнения символы должны быть в одинаковом регистре.

В частности, код у символа Б больше, чем у А, поэтому и результат сравнения такой.

Раздаточный материал № 114

```
alert( 'Б' > 'А' ); // true
```

Если строка состоит из нескольких букв, то сначала сравниваются первые буквы, потом вторые, и так далее, пока одна не будет больше другой.

Если первая буква первой строки больше – значит первая строка больше, независимо от остальных символов. Если одинаковы – сравнение идёт дальше. При этом любая буква больше отсутствия буквы.

Раздаточный материал № 115

```
alert( 'Привет' > 'Прив' ); // true, так как 'е' больше чем "ничего"
```

Раздаточный материал № 116

```
<script>
    alert( 'Привет'.localeCompare( 'Прив' ) )
</script>
```

Метод str.codePointAt(pos) возвращает код для символа, находящегося на позиции pos:

Раздаточный материал № 117\*

```
// одна и та же буква в нижнем и верхнем регистре
// будет иметь разные коды
alert( "z".codePointAt(0) ); // 122
alert( "Z".codePointAt(0) ); // 90
```

Метод String.fromCodePoint(code) отображает символ по его коду code.

Раздаточный материал № 118

```
alert( String.fromCodePoint(90) ); // Z
```

## **КТ № 5 часть 1 (задания в файле JavaScript КТ.pdf)**

# Объект Math

(Ссылка на Булгаков

[https://rostov.bulgakov.app/lessons/2439?blockId=content\\_anchor\\_0](https://rostov.bulgakov.app/lessons/2439?blockId=content_anchor_0))

Объект Math включает свойства и методы для работы с математическими формулами, математическими постоянными и для выполнения вычислений. Объект Math не требуется специфицировать как подобъект объектов document или window, но его имя необходимо писать с прописной буквы.

abs()

Функция abs() возвращает абсолютное значение числа:

```
let x = -25;  
console.log(Math.abs(x)); // 25  
let y = 34;  
console.log(Math.abs(y)); // 34
```

ceil()

Функция ceil() округляет число до следующего наибольшего целого числа:

```
let x = Math.ceil(9.2); // 10  
let y = Math.ceil(-5.9); // -5
```

Выражение Math.ceil(9.2) возвращает число 10, так как число 10 следующее наибольшее целое число после 9.2. И также выражение Math.ceil(-5.9) возвращает -5, потому что число -5 следующее наибольшее целое после -5.9

floor()

Функция floor() округляет число до следующего наименьшего целого числа:

```
let x = Math.floor(9.2); // 9  
let y = Math.floor(-5.9); // -6
```

round()

Функция round() округляет число до следующего наименьшего целого числа, если его десятичная часть меньше 0.5. Если же десятичная часть равна или больше 0.5, то округление идёт до ближайшего наибольшего целого числа:

```
let x = Math.round(5.5); // 6  
let y = Math.round(5.4); // 5  
let z = Math.round(-5.4); // -5  
let n = Math.round(-5.5); // -5  
let m = Math.round(-5.6); // -6  
console.log(x);  
console.log(y);  
console.log(z);  
console.log(n);  
pow()
```

Функция `pow()` возвращает число в определенной степени. Например, возведём число 2 в степень 3:

```
let x = Math.pow(2, 3); // 8
```

`sqrt()`

Функция `sqrt()` возвращает квадратный корень числа:

```
let x = Math.sqrt(121); // 11
let y = Math.sqrt(9); // 3
let z = Math.sqrt(20); // 4.47213595499958
```

`log()`

Функция `log()` возвращает натуральный логарифм (по основанию  $e$ ) числа:

```
let x = Math.log(1); // 0
let z = Math.log(10); // 2.302585092994046
```

Тригонометрические функции

Целый ряд функций представляют тригонометрические функции: `sin()` (вычисляет синус угла), `cos()` (вычисляет косинус угла), `tan()` (вычисляет тангенс угла).

```
let x = Math.sin(90); // 0.8939966636005579
let y = Math.cos(0); // 1
let z = Math.tan(45); // 1.6197751905438615
```

Функция `asin()` вычисляет арксинус числа, `acos()` - арккосинус, а `atan()` - арктангенс числа:

```
let x = Math.asin(0.9); // 1.1197695149986342
let y = Math.acos(1); // 1
let z = Math.atan(1); // 0.7853981633974483
```

`min()` и `max()`

Функции `min()` и `max()` возвращают соответственно минимальное и максимальное значение из набора чисел:

```
let max = Math.max(19, 45); // 45
let min = Math.min(33, 24); // 24
```

Эти функции необязательно должны принимать два числа, в них можно передавать и большее количество чисел:

```
let max = Math.max(1, 2, 3, -9, 46, -23); // 46
```

random()

1: Функция random() возвращает случайное число с плавающей точкой из диапазона от 0 до 1:

```
let x = Math.random(); // например, 0.34934820
```

Константы

Кроме методов объект Math также определяет набор встроенных констант, которые можно использовать в различных вычислениях:

Math.PI (число PI): 3.141592653589793

Math.SQRT2 (квадратный корень из двух): 1.4142135623730951

Math.SQRT1\_2 (половина от квадратного корня из двух): 0.7071067811865476

Math.E (число e или число Эйлера): 2.718281828459045

Math.LN2 (натуральный логарифм числа 2): 0.6931471805599453

Math.LN10 (натуральный логарифм числа 10): 2.302585092994046

Math.LOG2E (двоичный логарифм числа e): 1.4426950408889634

Math.LOG10E (десятичный логарифм числа e): 0.4342944819032518

Используем константы в вычислениях:

```
let x = Math.log(Math.E); // 1
```

```
let z = Math.tan(Math.PI/4); // 0.9999999999999999
```

**КТ № 5 часть 2 (задания в файле JavaScript КТ.pdf)**

# Объект Date

Системные часы ПК отсчитывают время в системе GMT, т.е. по Гринвичу. Однако в ОС Windows установлено локальное время (UTC – Всемирное координированное время), соответствующее конкретному часовому поясу. Это время фиксируется при создании и изменении файлов на ПК. UTC является стандартом времени на основе Интернета.

Дата и время, генерируемые в сценариях, сохраняются в памяти в системе GMT, но пользователю выводятся, как правило, в локальном виде.

Для работы с датой и временем необходимо создать экземпляр встроенного объекта Date. Для манипуляций с объектом Date применяются соответствующие методы.

## *JavaScript.Раздаточный материал № 119*

```
переменная = new Date(параметры)      // создаем экземпляр, параметры могут отсутствовать
переменная.метод
переменная.метод(новое значение)      //для задания нового значения
```

При создании объекта даты с помощью выражения new Date() можно указать в качестве параметров, какие дату и время следует установить в этом объекте. Для этого существуют следующие способы:

## *JavaScript.Раздаточный материал № 120*

```
new Date("Месяц дд, гггг чч:мм:сс")
new Date("Месяц дд, гггг")
new Date(гг, мм, дд, чч, мм, сс)
new Date(гг, мм, дд)
new Date(миллисекунды)
```

В первых двух вариантах параметры задаются в виде строки, в которых указаны компоненты даты и времени. Буквенные обозначения определяют шаблон параметров. Запятые и двоеточия обязательны. Если компоненты времени опущены, то устанавливается значение 0 (полночь). Компоненты даты обязательно должны быть указаны. Месяц задается своим полным английским названием. Остальные компоненты указываются в виде чисел. Если число меньше 10, то допускается запись без первого 0.

В третьем и четвертом способах компоненты даты и времени задаются целыми числами.

В пятом варианте задают целое число, которое представляет интервал в миллисекундах, начиная с 1 января 1970 года (с момента 00:00:00, Unix Time).

<https://www.epochconverter.com/>

Количество миллисекунд, отсчитанное от указанной стартовой даты, позволяет вычислить все компоненты даты и времени.

Все методы объекта Date делятся на две категории: для получения значения (их названия имеют префикс get) и для установки новых значений (их названия имеют префикс set). В каждой категории выделяют две группы методов – для локального формата и формата UTC. Методы позволяют работать с отдельными компонентами даты и времени. Часто используемые методы:

## *JavaScript.Раздаточный материал № 121*

```
getDate(), getMonth(), getFullYear() (возвращает четырехзначное значение года), getHours(),
getMinutes(), getSeconds()
```

**ВАЖНО: Нумерация месяцев, дней недели, часов, минут и секунд начинается с нуля.**



```
<script>
//Скрипт отмечает точную дату и время вашего прибытия на страницу
Now = new Date();
document.write("Сегодня " + Now.getDate() + "-" + Now.getMonth() + "-" + Now.getFullYear() + ". Вы
зашли на мою страницу ровно в: " + Now.getHours() + ":" + Now.getMinutes() + " и " +
Now.getSeconds() + " секунд.")
</script>
```

Вычислять разность двух дат или создавать счетчик времени можно с помощью методов как для локального формата, так и для UTC. Формат UTC применяется в расчетах, учитывающих часовой пояс. При отображении значения объекта Date оно автоматически преобразуется в строку методом toString(). Формат этой строки зависит от ОС и браузера. Если коррекция времени с учетом часового пояса не требуется, то для представления даты и времени можно воспользоваться методом toLocaleString().

Метод parse() переводит заданную дату в миллисекунды с момента 00:00:00 (т.е. с 01.01.1970г.).

*JavaScript.Раздаточный материал № 122*

```
<script>
date1=new Date(2022,11,18);
date2=new Date(2022,12,28);
days=(Date.parse(date2) - Date.parse(date1))/1000/60/60/24;
document.write("До окончания 1 семестра осталось " + days + " дней");
document.write("<br>");
document.write("Дата начала отсчета в формате Всеобщего времени: " + date1); // метод toString()
применен автоматически
document.write("<br>");
document.write("Дата начала отсчета без коррекции часового пояса: " + date1.toLocaleString());
</script>
```

*Раздаточный материал № 123. Решение задач.*

1. Создать скрипт, обеспечивающий выполнение следующих действий:

Запрос имени пользователя;

Отображение имени и даты прихода в строке заголовка и в документе.

Переменную даты представить в виде одной строки.

2. Создать скрипт, обеспечивающий выполнение следующих действий:

Расчет и отображение количества дней до Нового года;

Расчет и отображение количества дней до Вашего дня рождения.

Даты отображать без коррекции часового пояса.

## Функции с параметрами

При вызове функции ей можно передать данные, которые она использует по своему усмотрению. Функцию можно вызвать с любым количеством аргументов.

*Раздаточный материал №124*

```
<script>
    function showMessage(from, text) { // параметры from, text
        from = "** " + from + " **"; // здесь может быть сложный код оформления
        alert(from + ': ' + text);
    }
    showMessage('Студент', 'Привет!');
    showMessage('Студент!', 'Как дела?');

</script>
```

Функция может вернуть результат, который будет передан в вызвавший её код.

Например, создадим функцию calcD, которая будет возвращать дискриминант квадратного уравнения по формуле  $b^2 - 4ac$ :

*Раздаточный материал № 125*

```
<script>
    function calcD(a, b, c) {
        return b*b - 4*a*c;
    }

    let test = calcD(-4, 2, 1);
    console.log(test);

</script>
```

Для возврата значения используется директива return. Она может находиться в любом месте функции. Как только до неё доходит управление – функция завершается и значение передается обратно. Одним вызовом return можно передать несколько значений. Вызовов return может быть и несколько, но выполниться только тот, до которого первым дойдет управление, например:

*Раздаточный материал № 126*

```
<script>
    function checkAge(age) {
        if (age > 18) {
            return true;
        } else {
            return confirm('Родители разрешили?');
        }
    }

    let age = prompt('Ваш возраст?');

    if (checkAge(age)) {
        alert( 'Доступ разрешен' );
    } else {
        alert( 'В доступе отказано' );
    }

</script>
```

В качестве параметра, передаваемого функции, может выступать объект, в частности форма со всеми ее элементами (*рассмотрено в РМ 127*).

### Создание HTML-документов с помощью JavaScript

JS позволяет создавать новый HTML-документ в процессе выполнения сценария. Новые документы можно помещать в отдельные окна. В качестве параметра метода `document.write` может использоваться строка, содержащая теги HTML, например, для построения таблицы.

#### *Раздаточный материал № 127*

```
!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

  <script>
    //сумма за квартал, налог и сумма, выдаваемая на руки
    function sum(obj)
    {
      let a1 = Number(obj.num1.value);
      let a2 = Number(obj.num2.value);
      let a3 = Number(obj.num3.value);
      let s = a1+a2+a3;
      let n = 0.13*s;
      let m = s - n;
      let st = "<h4>Расчет за квартал</h4><br><table border=3"
      st = st + "<tr><th>сумма за квартал</th><th>налог</th><th>на
руки</th>"
      t = st +
"<tr><td>"+s+"</td>"+<td>"+n+"</td>"+<td>"+m+"</td></tr></table>"
      document.write(t);
    }
  </script>

  <h4>Определение дохода за квартал</h4>

  <form name = "form1">
  <p>Введите суммы за каждый месяц и нажмите <b>Вычислить</b></p>
  <pre>
январь:<input type="text" name="num1" size="10"><br>
февраль:<input type="text" name="num2" size="10"><br>
март:<input type="text" name="num3" size="10"><br>

  <input type="button" value="Вычислить" onClick = "sum(form1)"><br>
  </pre>
  <input type="reset">
```

## **КТ № 5 часть 3 (задания в файле JavaScript КТ.pdf)**

## Создание таблицы.

`<table>` и `</table>` ограничивают текст описания будущей таблицы.

`<tr>` - используется для описания её строк.

`<th>` и `<td>` позволяют создавать ячейки таблицы. Существует два типа ячеек, из которых может быть построена таблица. `<th>` отвечает за создание ячеек с заголовками, `<td>` дает возможность определения ячеек для хранения данных таблицы. Содержимое ячеек заголовков отображается полужирным шрифтом и выравнивается по центру.

*Раздаточный материал № 128*

```
<table border = "1">
```

```
<tr>
```

```
  <th>название фильма</th>
```

```
  <th>время сеанса</th>
```

```
  <th>стоимость</th>
```

```
</tr>
```

```
<tr align="right" bgcolor="green">
```

```
  <td>аватар</td>
```

```
  <td>10.30</td>
```

```
  <td>250 руб</td>
```

```
</tr>
```

```
</table>
```

## Многомерные массивы

Многомерные массивы можно создавать путем присваивания любому элементу массива нового массива.

*JavaScript. Раздаточный материал № 129*

M[0] = [3, 2, 8];

Обращение к элементу:

J = M[0][1];

*JavaScript. Раздаточный материал № 130*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Максимальный элемент в двумерном массиве</title>
</head>
<body>
  <script>
    let M = [0,0,0];

    M[0] = [3,2,1];
    M[1] = [7,8,9];
    M[2] = [5,6,7];

    Max = M[0][0];
    Ind = [0,0];

    for (i=0; i<M.length; i++) {
      for (j=0; j<M.length; j++) {
        if (M[i][j] > Max) {
          Max = M[i][j];
          Ind[0] = i;
          Ind[1] = j;
        }
      }
    }
    document.write('Максимальный   ' + Max);
    document.write('<br>');
    document.write('Его индекс  [' + Ind[0] + '][' + Ind[1] + ']');
  </script>
</body>
</html>
```

**КТ № 6 (Решение задач, блок 1)**  
**(задания в файле JavaScript КТ.pdf)**

## Подключение файла со скриптами JavaScript

JavaScript код можно писать в отдельном файле, который затем будет подключен к HTML файлу. Сначала создается файл со скриптом. У этого файла должно быть расширение .js.

Для подключения файла со скриптом к HTML файлу нужно в теге script в атрибуте src указать путь к файлу со скриптом:

*JavaScript. Раздаточный материал № 131*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <script src="script.js"></script>
  </head>
  <body>

  </body>
</html>
```

Можно подключать не один, а несколько файлов с помощью нескольких тегов script:

*JavaScript. Раздаточный материал № 132*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <script src="script1.js"></script>
    <script src="script2.js"></script>
  </head>
  <body>

  </body>
</html>
```

В теге script можно либо писать код, либо подключить файл. Попытка сделать это одновременно не будет работать.



## Строгий режим в JavaScript

В современном JavaScript при написании кода первой строчкой нужно включать так называемый строгий режим. Этот режим заставляет браузер использовать все современные возможности языка.

Для включения строгого режима первой строчкой скрипта необходимо поставить команду "use strict".

*JavaScript. Раздаточный материал № 133*

```
<script>  
    "use strict";  
    alert('text!');  
</script>
```

## Многострочность

В JavaScript строках, созданных через одинарные или двойные кавычки, не допустим перенос строки. То есть вот так не будет работать:

*JavaScript. Раздаточный материал № 134*

```
let str = 'abc  
def'; // так не будет работать
```

А вот косые кавычки специально предназначены для создания многострочных строк:

*JavaScript. Раздаточный материал № 135*

```
let str = `abc  
def`; // так будет работать
```

## Регулярные выражения

Регулярные выражения – мощное средство поиска и замены в строке.

Регулярные выражения

Регулярное выражение (оно же «регэксп», «регулярка» или просто «рег»), состоит из шаблона (также говорят «паттерн») и необязательных флагов.

Существует два синтаксиса для создания регулярного выражения.

Длинный синтаксис:

```
regexr = new RegExp("шаблон", "флаги");
```

...И короткий синтаксис, использующий слешы "/":

```
regexr = /шаблон/; // без флагов  
regexr = /шаблон/gmi; // с флагами gmi (будут описаны далее)
```

Слешы `/.../` говорят JavaScript о том, что это регулярное выражение. Они играют здесь ту же роль, что и кавычки для обозначения строк.

Регулярное выражение `regexr` в обоих случаях является объектом встроенного класса `RegExp`.

Основная разница между этими двумя способами создания заключается в том, что слешы `/.../` не допускают никаких вставок переменных (наподобие возможных в строках через `${...}`). Они полностью статичны.

Слешы используются, когда мы на момент написания кода точно знаем, каким будет регулярное выражение – и это большинство ситуаций. А `new RegExp` – когда мы хотим создать регулярное выражение «на лету» из динамически сгенерированной строки, например:

```
let tag = prompt("Какой тег вы хотите найти?", "h2");  
let regexr = new RegExp(`<${tag}>`); // то же, что /<h2>/ при ответе "h2" на  
prompt выше
```

## Флаги

Регулярные выражения могут иметь флаги, которые влияют на поиск.

В JavaScript их всего шесть:

Флаг	Применение
i	С этим флагом поиск не зависит от регистра: нет разницы между А и а
g	С этим флагом поиск ищет все совпадения, без него – только первое.
m	Многострочный режим
s	Включает режим «dotall», при котором точка . может соответствовать символу перевода строки \n (рассматривается в главе Символьные классы).
u	Включает полную поддержку Юникода. Флаг разрешает корректную обработку суррогатных пар (подробнее об этом в главе Юникод: флаг "u" и класс \p{...}).
y	Режим поиска на конкретной позиции в тексте (описан в главе Поиск на заданной позиции, флаг "y")

### Поиск: str.match

Метод `str.match(regex)` для строки `str` возвращает совпадения с регулярным выражением `regex`.

У него есть три режима работы:

1) Если у регулярного выражения есть флаг `g`, то он возвращает массив всех совпадений:

```
let str = "Любо, братцы, любо!";  
alert( str.match(/любо/gi) ); // Любо,любо (массив из 2х подстрок-совпадений)
```

Обратите внимание: найдены и Любо и любо, благодаря флагу `i`, который делает регулярное выражение регистронезависимым.

2) Если такого флага нет, то возвращает только первое совпадение в виде массива, в котором по индексу 0 находится совпадение, и есть свойства с дополнительной информацией о нём:

```
let str = "Любо, братцы, любо!";  
let result = str.match(/любо/i); // без флага g  
alert( result[0] ); // Любо (первое совпадение)  
alert( result.length ); // 1  
// Дополнительная информация:  
alert( result.index ); // 0 (позиция совпадения)
```

```
alert( result.input ); // Любо, братцы, любо! (исходная строка)
```

В этом массиве могут быть и другие индексы, кроме 0, если часть регулярного выражения выделена в скобки (об этом поговорим позже)

3) И, наконец, если совпадений нет, то, вне зависимости от наличия флага g, возвращается null.

Это очень важный нюанс. При отсутствии совпадений возвращается не пустой массив, а именно null. Если об этом забыть, можно легко допустить ошибку, например:

```
let matches = "JavaScript".match(/HTML/); // = null

if (!matches.length) { // Ошибка: у null нет свойства length
    alert("Ошибка в строке выше");
}
```

Если хочется, чтобы результатом всегда был массив, можно написать так:

```
let matches = "JavaScript".match(/HTML/) || [];

if (!matches.length) {
    alert("Совпадений нет"); // теперь работает
}
```

### **Замена: str.replace**

Метод str.replace(regex, replacement) заменяет совпадения с regex в строке str на replacement (все, если есть флаг g, иначе только первое).

Например:

```
// без флага g
alert( "We will, we will".replace(/we/i, "I") ); // I will, we will
```

```
// с флагом g
alert( "We will, we will".replace(/we/ig, "I") ); // I will, I will
```

В строке замены replacement мы можем использовать специальные комбинации символов для вставки фрагментов совпадения:

Спецсимволы	Действие в строке замены
\$&	вставляет всё найденное совпадение
\$`	вставляет часть строки до совпадения
\$'	вставляет часть строки после совпадения
\$n	если n это 1-2 значное число, вставляет содержимое n-й скобочной группы регулярного выражения
\$<name>	вставляет содержимое скобочной группы с именем name
\$\$	вставляет символ "\$"

Пример с \$&:

```
alert( "Люблю HTML".replace(/HTML/, "$& и JavaScript") ); // Люблю HTML и JavaScript
```

### Проверка: `regexp.test`

Метод `regexp.test(str)` проверяет, есть ли хоть одно совпадение, если да, то возвращает `true`, иначе `false`.

```
let str = "Я ЛюБлЮ JavaScript";  
let regexp = /люблю/i;  
alert( regexp.test(str) ); // true
```

### Символьные классы

Символьный класс	Значение
\d	Цифра: символ от 0 до 9.
\s	Пробельные символы: включает в себя символ пробела, табуляции \t, перевода строки \n и некоторые другие редкие пробельные символы, обозначаемые как \v, \f и \r.
\w	Символ «слова», а точнее – буква латинского алфавита или цифра или подчеркивание _. Нелатинские буквы не являются частью класса \w, то есть буква русского алфавита не подходит.

Для примера, `\d\s\w` обозначает «цифру», за которой идёт пробельный символ, а затем символ слова, например 1 а.

Регулярное выражение может содержать как обычные символы, так и символьные классы.

Например, `CSS\d` соответствует строке CSS с цифрой после неё:

```
let str = "Есть ли стандарт CSS4?";  
let regexp = /CSS\d/  
alert( str.match(regexp) ); // CSS4
```

Также мы можем использовать несколько символьных классов:

```
alert( "I love HTML5!".match(/\s\w\w\w\w\d/) ); // ' HTML5'
```

## Обратные символьные классы

«Обратный» означает, что он соответствует всем другим символам, например:

Обратный класс	Значение
<code>\D</code>	Не цифра: любой символ, кроме <code>\d</code> , например буква.
<code>\S</code>	Не пробел: любой символ, кроме <code>\s</code> , например буква.
<code>\W</code>	Любой символ, кроме <code>\w</code> , то есть не буквы из латиницы, не знак подчёркивания и не цифра. В частности, русские буквы принадлежат этому классу.

Точка `.` — это специальный символьный класс, который соответствует «любому символу, кроме новой строки».

Для примера:

```
alert( "Ю".match(/./) ); // Ю
```

Или в середине регулярного выражения:

```
let regexp = /CS.4/;  
alert( "CSS4".match(regexp) ); // CSS4  
alert( "CS-4".match(regexp) ); // CS-4  
alert( "CS 4".match(regexp) ); // CS 4 (пробел тоже является символом)
```

Обратите внимание, что точка означает «любый символ», но не «отсутствие символа». Там должен быть какой-либо символ, чтобы соответствовать условию поиска:

```
alert( "CS4".match(/CS.4/) ); // null, нет совпадений потому что нет символа для точки
```

## **Якоря: начало строки ^ и конец \$**

У символов каретки ^ и доллара \$ есть специальные значения в регулярных выражениях. Они называются «якоря» (anchors).

Каретка ^ означает совпадение с началом текста, а доллар \$ – с концом.

К примеру, давайте проверим начинается ли текст с Mary:

```
let str1 = "Mary had a little lamb";  
alert( /^Mary/.test(str1) ); // true
```

Аналогично можно проверить, кончается ли строка словом snow при помощи snow\$:

```
let str1 = "it's fleece was white as snow";  
alert( /snow$/.test(str1) ); // true
```

## **Экранирование символа**

Чтобы использовать специальный символ как обычный, добавьте к нему обратную косую черту: \.

Это называется «экранирование символа».

Круглые скобки также являются специальными символами, поэтому, если нам нужно использовать именно их, нужно указать \(. В приведённом ниже примере ищется строка "g()":

```
alert( "function g()".match(/g\(\)/) ); // "g()"
```

Если мы ищем обратную косую черту \, это специальный символ как в обычных строках, так и в регулярных выражениях, поэтому мы должны удвоить её.

```
alert( "1\\2".match(/\/) ); // \'
```

## **Наборы**

Несколько символов или символьных классов в квадратных скобках [...] означают «искать любой символ из заданных».

Для примера, [eao] означает любой из 3-х символов: 'a', 'e' или 'o'.

Это называется набором.

Наборы могут использоваться в регулярных выражениях вместе с обычными символами, например:

```
// найти [т или х], после которых идёт "оп"  
alert( "Топ хоп".match(/[тх]оп/gi) ); // "Топ", "хоп"
```

Обратите внимание, что в наборе несколько символов, но в результате он соответствует ровно одному символу.

Так что этот пример не даёт совпадений:

```
alert( "Буаля".match(/B[ya]ля/) ); // null, нет совпадений  
// ищет "B", затем [у или а], потом "ля"  
// а в строке B, потом у, потом а
```

## Диапазоны

Ещё квадратные скобки могут содержать диапазоны символов.

К примеру, [a-z] соответствует символу в диапазоне от a до z, или [0-5] – цифра от 0 до 5.

В приведённом ниже примере мы ищем "x", за которым следуют две цифры или буквы от A до F:

```
alert( "Exception 0xAF".match(/x[0-9A-F][0-9A-F]/g) ); // xAF
```

Здесь в [0-9A-F] сразу два диапазона: ищется символ, который либо цифра от 0 до 9, либо буква от A до F.

Если мы хотим найти буквы и в верхнем и в нижнем регистре, то мы можем добавить ещё диапазон a-f: [0-9A-Fa-f]. Или поставить у регулярного выражения флаг i.

## Исключающие диапазоны

Помимо обычных диапазонов, есть «исключающие» диапазоны, которые выглядят как [^...].



Они обозначаются символом каретки ^ в начале диапазона и соответствуют любому символу за исключением заданных.

Например:

[^aeuo] – любой символ, за исключением 'a', 'e', 'u' или 'o'.

[^0-9] – любой символ, за исключением цифры, то же, что и \D.

[^\s] – любой непробельный символ, то же, что и \S.

Пример ниже ищет любые символы, кроме латинских букв, цифр и пробелов:

```
alert( "alice15@gmail.com".match(/^[^\d\sA-Z]/gi) ); // @ и .
```

## Экранирование в [ ... ]

Обычно, когда мы хотим найти специальный символ, нам нужно экранировать его, например \. А если нам нужна обратная косая черта, тогда используем \\, т.п.

В квадратных скобках большинство специальных символов можно использовать без экранирования:

- 1) Символы . + ( ) не нужно экранировать никогда.
- 2) Тире - не надо экранировать в начале или в конце (где оно не задаёт диапазон).
- 3) Символ каретки ^ нужно экранировать только в начале (где он означает исключение).
- 4) Закрывающую квадратную скобку ], если нужен именно такой символ, экранировать нужно.

## Количество {n}

Чтобы указать количество повторений, нам нужно добавить квантификатор.

Самый простой квантификатор — это число в фигурных скобках: {n}.

Он добавляется к символу (или символьному классу, или набору [...] и т.д.) и указывает, сколько их нам нужно.

Можно по-разному указать количество, например:

- 1) Точное количество: {5}

Шаблон \d{5} обозначает ровно 5 цифр, он эквивалентен \d\d\d\d\d.

Следующий пример находит пятизначное число:

```
alert( "Мне 12345 лет".match(/\d{5}/) ); // "12345"
```

2) Для того, чтобы найти числа от 3 до 5 цифр, мы можем указать границы в фигурных скобках: `\d{3,5}`

```
alert( "Мне не 12, а 1234 года".match(/\d{3,5}/) ); // "1234"
```

3) Верхнюю границу можно не указывать.

Тогда шаблон `\d{3,}` найдёт последовательность чисел длиной 3 и более цифр:

```
alert( "Мне не 12, а 345678 лет".match(/\d{3,}/) ); // "345678"
```

## Короткие обозначения

Для самых востребованных квантификаторов есть сокращённые формы записи:

+

Означает «один или более». То же самое, что и `{1,}`.

Например, `\d+` находит числа (из одной или более цифр):

```
let str = "+7(903)-123-45-67";
```

```
alert( str.match(/\d+/g) ); // 7,903,123,45,67
```

?

Означает «ноль или один». То же самое, что и `{0,1}`. По сути, делает символ необязательным.

Например, шаблон `ou?r` найдёт `o` после которого, возможно, следует `u`, а затем `r`.

Поэтому шаблон `colou?r` найдёт два варианта: `color` и `colour`:

```
let str = "Следует писать color или colour?";
```

```
alert( str.match(/colou?r/g) ); // color, colour
```

\*

Означает «ноль или более». То же самое, что и `{0,}`. То есть символ может повторяться много раз или вообще отсутствовать.

Например, шаблон `\d0*` находит цифру и все нули за ней (их может быть много или ни одного):

```
alert( "100 10 1".match(/\d0*/g) ); // 100, 10, 1
```

Сравните это с + (один или более):

```
alert( "100 10 1".match(/\d0+/g) ); // 100, 10
```

// 1 не подходит, т.к 0+ требует как минимум один ноль

## Скобочные группы

Часть шаблона можно заключить в скобки (...). Это называется «скобочная группа».

У такого выделения есть два эффекта:

- 1) Позволяет поместить часть совпадения в отдельный массив.
- 2) Если установить квантификатор после скобок, то он будет применяться ко всему содержимому скобки, а не к одному символу.

Пример: gogogo

Без скобок шаблон go+ означает символ g и идущий после него символ o, который повторяется один или более раз. Например, goooo или goooooooooo.

Скобки группируют символы вместе. Так что (go)+ означает go, gogo, gogogo и т.п.

```
alert( 'Gogogo now!'.match(/(go)+/i) ); // "Gogogo"
```

Пример: домен

Сделаем что-то более сложное – регулярное выражение, которое соответствует домену сайта.

Например:

mail.com

users.mail.com

smith.users.mail.com

Как видно, домен состоит из повторяющихся слов, причём после каждого, кроме последнего, стоит точка.

На языке регулярных выражений (\w+\.)+\w+:

```
let regexp = /(\w+\.)+\w+/g;
```

```
alert( "site.com my.site.com".match(regex) ); // site.com,my.site.com
```

Поиск работает, но такому шаблону не соответствует домен с дефисом, например, my-site.com, так как дефис не входит в класс `\w`.

Можно исправить это, заменим `\w` на `[\w-]` везде, кроме как в конце: `([\w-]+\.)+\w+`.

## Содержимое скобок в match

Скобочные группы нумеруются слева направо. Поисковый движок запоминает содержимое, которое соответствует каждой скобочной группе, и позволяет получить его в результате.

Метод `str.match(regex)`, если у регулярного выражения `regex` нет флага `g`, ищет первое совпадение и возвращает его в виде массива:

На позиции 0 будет всё совпадение целиком.

На позиции 1 – содержимое первой скобочной группы.

На позиции 2 – содержимое второй скобочной группы.

...и так далее...

Например, мы хотим найти HTML теги `<.*?>` и обработать их. Было бы удобно иметь содержимое тега (то, что внутри уголков) в отдельной переменной.

Давайте заключим внутреннее содержимое в круглые скобки: `<(.*?)>`.

Теперь получим как тег целиком `<h1>`, так и его содержимое `h1` в виде массива:

```
let str = '<h1>Hello, world!</h1>';
```

```
let tag = str.match(/<(.*?)>/);
```

```
alert( tag[0] ); // <h1>
```

```
alert( tag[1] ); // h1
```

## Необязательные группы

Даже если скобочная группа необязательна (например, стоит квантификатор `(...)?`), соответствующий элемент массива `result` существует и равен `undefined`.

Например, рассмотрим регулярное выражение `a(z)?(c)?`. Оно ищет букву "a", за которой идёт необязательная буква "z", за которой, в свою очередь, идёт необязательная буква "c".

Если применить его к строке из одной буквы a, то результат будет такой:

```
let match = 'a'.match(/a(z)?(c)?/);  
  
alert( match.length ); // 3  
alert( match[0] ); // a (всё совпадение)  
alert( match[1] ); // undefined  
alert( match[2] ); // undefined
```

Массив имеет длину 3, но все скобочные группы пустые.

А теперь более сложная ситуация для строки ac:

```
let match = 'ac'.match(/a(z)?(c)?/)  
  
alert( match.length ); // 3  
alert( match[0] ); // ac (всё совпадение)  
alert( match[1] ); // undefined, потому что для (z)? ничего нет  
alert( match[2] ); // c
```

Длина массива всегда равна 3. Для группы (z)? ничего нет, поэтому результат: ["ac", undefined, "c"].

### **Альтернация (или)**

Альтернация – термин в регулярных выражениях, которому в русском языке соответствует слово «ИЛИ».

В регулярных выражениях она обозначается символом вертикальной черты |.

Например, нам нужно найти языки программирования: HTML, PHP, Java и JavaScript.

Соответствующее регулярное выражение: `html|php|java(script)?`.

Пример использования:

```
let regexp = /html|css|java(script)?/gi;  
  
let str = "Сначала появился язык Java, затем HTML, потом JavaScript";  
  
alert( str.match(regexp) ); // Java,HTML,JavaScript
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Регулярные выражения</title>
</head>
<body>
  <script>
    'use strict'
    str = `Регулярные выражения представляют собой похожий, но гораздо более
    сильный инструмент
        для поиска строк, проверки их на соответствие какому-либо шаблону
    и другой подобной
        работы. Англоязычное название этого инструмента – Regular
    Expressions или просто RegExp.
        Строго говоря, регулярные выражения – специальный язык для
    описания шаблонов строк.

        Иванов Иван Иванович
        AAAA aaaa AaAaAaAa 123 123 12345 11223344
        A1B2B3 AA11 BB22BB 33ГГ44

        8966-874-23-15

        QwertyЙцукен

        +-,/[ ](), *** (***), a*(b+[c+d])*e/f+g-h

        !"'"####$%$%$%$%$%$%&&&' '((( )))***++++, , , , ,-----
    ..//:::;;;<<<<<==>>>???'
        @@@@[[[\\]]]]^^^_{{{||||}}}}

        <a href="#10">10: CamelCase -> under_score</a>;
        <a href="#11">11: Удаление повторов</a>;
        <a href="#12">12: Близкие слова</a>;
        <a href="#13">13: Форматирование больших чисел</a>;
        <a href="#14">14: Разделить текст на предложения</a>;
        <a href="#15">15: Форматирование номера телефона</a>;
        <a href="#16">16: Поиск e-mail'ов – 2</a>;`

    //ищем ФИО
    result_FIO = str.match(/[А-Я][а-я]+\s[А-Я][а-я]+\s[А-Я][а-я]+/);
    console.log(result_FIO);

    //ищем номер телефона по формату XXXX-XXX-XX-XX
    result_tel = str.match(/\d{4}[-]\d{3}[-]\d\d[-]\d\d/g);
    console.log(result_tel);
```

```
//или
//result_tel = str.match(/\d\d\d\d[-]\d\d\d[-]\d\d[-]\d\d/g);
//console.log(result_tel);

// ищем все совпадения начинающиеся с "pe"
let result = str.match(/[p][e]/gi);
console.log(result);
console.log("Количество совпадений", result.length); //4

// ищем все совпадения где есть две цифры
let result1 = str.match(/[0-9][0-9]/g);
console.log(result1);
console.log("Количество совпадений", result1.length);
</script>
</body>
</html>
```

## КТ № 7 (Решение задач, блок 2)

(задания в файле JavaScript КТ.pdf)

1. Написать скрипт, содержащий функцию, которая через регулярное выражение определит соответствие введенного номера телефона шаблону XXXX-XXX-XX-XX и правильность ввода ФИО посетителя. Если номер или ФИО введены по шаблону, то показать его в консоли, иначе – вывести сообщения

Форма ввода и проверки данных

Введи номер телефона:

Введи свои ФИО:

Уведомление от сайта 127.0.0.1

Введен некорректный номер. Введи по формату XXXX-XXX-XX-XX

Форма ввода и проверки данных

Введи номер телефона:

Введи свои ФИО:

Уведомление от сайта 127.0.0.1

Введены некорректные ФИО. Введи по формату Иванов Иван Иванович