ДАТЫ И СТРОКИ

JavaScript

Модуль 1. Урок 3.

LUXOFT

LXFT LISTED NYSE

РАБОТА С ДАТАМИ. КЛАСС DATE

Создает объект Date с текущей датой и временем:

```
var now = new Date();
```

Создает объект Date, значение которого равно количеству миллисекунд (1/1000 секунды), прошедших с 1 января 1970 года GMT+0.

```
var Jan02_1970 = new Date(3600 * 24 * 1000);
```

new Date(datestring)
Если единственный аргумент – строка, используется вызов Date.parse (см. далее) для чтения даты из неё.

new Date(year, month, date, hours, minutes, seconds, ms)
Дату можно создать, используя компоненты в местной временной зоне. Для этого формата обязательны только первые два аргумента. Отсутствующие параметры, начиная с hours считаются равными нулю, а date – единице.

ПОЛУЧЕНИЕ КОМПОНЕНТОВ ДАТЫ

Для доступа к компонентам даты-времени
объекта Date используются следующие методы:

getFullYear()
Получить год (из 4 цифр)

getMonth()
Получить месяц, **от 0 до 11**.

getDate()
Получить число месяца, от 1 до 31.

getHours(), getMinutes(), getSeconds(), getMilliseconds()
Получить соответствующие компоненты.

УСТАНОВКА КОМПОНЕНТОВ ДАТЫ

- setFullYear(year [, month, date])
- setMonth(month [, date])
- setDate(date)
- setHours(hour [, min, sec, ms])
- setMinutes(min [, sec, ms])
- setSeconds(sec [, ms])
- setMilliseconds(ms)
- setTime(milliseconds) (устанавливает всю дату по миллисекундам с 01.01.1970 UTC)

DATE.PARSE

Все современные браузеры, понимают даты в упрощённом формате ISO 8601 Extended.
Этот формат выглядит так: **YYYY-MM-DDTHH:mm:ss.sssZ**, где:

- YYYY-MM-DD – дата в формате год-месяц-день.
- Обычный символ T используется как разделитель.
- HH:mm:ss.sss – время: часы-минуты-секунды-миллисекунды.
- Часть 'Z' обозначает временную зону – в формате +-hh:mm, либо символ Z, обозначающий UTC.

По стандарту её можно не указывать, тогда UTC, но в Safari с этим ошибка, так что лучше указывать всегда.
Также возможны укороченные варианты, например YYYY-MM-DD или YYYY-MM или даже только YYYY.

Метод Date.parse(str) разбирает строку str в таком формате и возвращает соответствующее ей количество миллисекунд. Если это невозможно, Date.parse возвращает NaN.
Например:

```
var msUTC = Date.parse('2012-01-26T13:51:50.417Z'); // зона UTC
console.log( msUTC ); // 1327571510417 (число миллисекунд)
```

С таймзоной -07:00 GMT:

```
var ms = Date.parse('2012-01-26T13:51:50.417-07:00’);
console.log( ms ); // 1327611110417 (число миллисекунд)
```

DATE - РЕЗЮМЕ

- Дата и время представлены в JavaScript одним объектом: Date. Создать «только время» при этом нельзя, оно должно быть с датой. Список методов Date вы можете найти в справочнике Date или выше.
- Отсчёт месяцев начинается с нуля.
- Отсчёт дней недели (для getDay()) тоже начинается с нуля (и это воскресенье).
- Объект Date удобен тем, что автокорректируется. Благодаря этому легко сдвигать даты.
- При преобразовании к числу объект Date даёт количество миллисекунд, прошедших с 1 января 1970 UTC. Побочное следствие – даты можно вычитать, результатом будет разница в миллисекундах.

СОЗДАНИЕ СТРОКИ

```
var text = "моя строка";
var anotherText = 'еще строка’;
var str = "012345";
```

Строки могут содержать специальные символы. Самый часто используемый из таких символов – это «перевод строки».
Он обозначается как \n, например:

Специальные символы	
Символ	Описание
\b	Backspace
\f	Form feed
\n	Новая строка
\r	Возврат каретки
\t	Tab
\uNNNN	Символ в кодировке Юникод с шестнадцатеричным кодом `NNNN`. Например, `\u00A9` -- юникодное представление символа копирайт ©

```
alert( 'Привет\nМир' ); // выведет "Мир" на новой строке
```

ЭКРАНИРОВАНИЕ СПЕЦИАЛЬНЫХ СИМВОЛОВ

Если строка в одинарных кавычках, то внутренние одинарные кавычки внутри должны быть *экранированы*, то есть снабжены обратным слешем \', вот так:

```
var str = 'I\'m a JavaScript programmer';
```

В двойных кавычках – экранируются внутренние двойные:

```
var str = "I'm a JavaScript \"programmer\" ";
alert( str ); // I'm a JavaScript "programmer"
```

Экранирование служит исключительно для правильного восприятия строки JavaScript. В памяти строка будет содержать сам символ без '\'. Вы можете увидеть это, запустив пример выше.

Сам символ обратного слэша '\' является служебным, поэтому всегда экранируется, т.е пишется как \\:

```
var str = ' символ \\ ';
```

ДЛИНА СТРОКИ, ДОСТУП К СИМВОЛАМ

Длина строки length

Одно из самых частых действий со строкой – это получение ее длины:

```
var str = "My\n"; // 3 символа. Третий - перевод строки
alert( str.length ); // 3
```

Чтобы получить символ, используйте вызов charAt(позиция). Первый символ имеет позицию 0:

```
var str = ”JavaScript"; alert( str.charAt(0) ); // ”J"
```

Также для доступа к символу можно использовать квадратные скобки:

```
var str = "Я - современный браузер!"; alert( str[0] ); // "Я"
```

ИЗМЕНЕНИЕ СТРОК

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

Можно лишь создать целиком новую строку и присвоить в переменную вместо старой, например:

```
var str = "пример";
str = str[3] + str[4] + str[5]; alert( str ); // мер
```

Смена регистра

Методы toLowerCase() и toUpperCase() меняют регистр строки на нижний/верхний:

```
alert( "Интерфейс".toUpperCase() ); // ИНТЕРФЕЙС
```

Пример ниже получает первый символ и приводит его к нижнему регистру:

```
alert( "Интерфейс" [0].toLowerCase() ); // 'и'
```

ПОИСК ПОДСТРОКИ

Для поиска подстроки есть метод indexOf(подстрока[, начальная_позиция]).
Он возвращает позицию, на которой находится подстрока или -1, если ничего не найдено.

Например:

```
var str = "Widget with id";
alert( str.indexOf("Widget") ); // 0, т.к. "Widget" найден прямо в начале str
alert( str.indexOf("id") ); // 1, т.к. "id" найден, начиная с позиции 1
alert( str.indexOf("widget") ); // -1, не найдено, так как поиск учитывает регистр
```

Необязательный второй аргумент позволяет искать, начиная с указанной позиции. Например, первый раз "id"появляется на позиции 1. Чтобы найти его следующее появление – запустим поиск с позиции 2:

```
var str = "Widget with id"; alert(str.indexOf("id", 2)) // 12, поиск начат с позиции 2
```

Также существует аналогичный метод lastIndexOf, который ищет не с начала, а с конца строки.

ПОИСК ВСЕХ ВХОЖДЕНИЙ

Чтобы найти все вхождения подстроки, нужно запустить indexOf в цикле. Как только получаем очередную позицию – начинаем следующий поиск со следующей.

Пример такого цикла:

```
var str = "Ослик Иа-Иа посмотрел на виадук"; // ищем в этой строке
var target = "Иа"; // цель поиска
var pos = 0;
while (true) {
  var foundPos = str.indexOf(target, pos);
  if (foundPos == -1) break;
  alert( foundPos ); // нашли на этой позиции
  pos = foundPos + 1; // продолжить поиск со следующей
}
```

ВЗЯТИЕ ПОДСТРОКИ: SUBSTRING, SUBSTR, SLICE

В JavaScript существуют целых 3 (!) метода для взятия подстроки, с небольшими отличиями между ними.

substring(start [, end])

Метод substring(start, end) возвращает подстроку с позиции **start** до, но не включая **end**.

```
var str = "stringify"; alert(str.substring(0,1)); // "s", символы с позиции 0 по 1 не включая 1.
```

Если аргумент end отсутствует, то идет до конца строки:

```
var str = "stringify"; alert(str.substring(2)); // ringify, символы с позиции 2 до конца
```

substr(start [, length])

Первый аргумент имеет такой же смысл, как и в substring, а второй содержит не конечную позицию, а количество символов.

```
var str = "stringify"; str = str.substr(2,4); // ring, со 2-й позиции 4 символа alert(str)
```

Если второго аргумента нет – подразумевается «до конца строки».

slice(start [, end])

Возвращает часть строки от позиции start до, но не включая, позиции end. Смысл параметров – такой же как в substring.

ОТРИЦАТЕЛЬНЫЕ АРГУМЕНТЫ В SUBSTRING И SLICE

Различие между substring и slice – в том, как они работают с отрицательными и выходящими за границу строки аргументами:

substring(start, end)

Отрицательные аргументы интерпретируются как равные нулю. Слишком большие значения усекаются до длины строки:

```
alert( "testme".substring(-2) ); // "testme", -2 становится 0
```

Кроме того, если start > end, то аргументы меняются местами, т.е. возвращается участок строки *между* start и end:

```
alert( "testme".substring(4, -1) ); // "test"
// -1 становится 0 -> получили substring(4, 0)
// 4 > 0, так что аргументы меняются местами -> substring(0, 4) = "test”
```

slice Отрицательные значения отсчитываются от конца строки:

```
alert( "testme".slice(-2) ); // "me", от 2 позиции с конца
alert( "testme".slice(1, -1) ); // "estm", от 1 позиции до первой с конца.
```

Это гораздо более удобно, чем странная логика substring.

ES2015: СТРОКИ-ШАБЛОНЫ (ИНТЕРПОЛЯЦИЯ СТРОК)

Специальный вид кавычек для строк: апостроф

```
let str = `обратные кавычки`;
```

Основные отличия от двойных "…" и одинарных '…' кавычек:

В них разрешён перевод строки.

Например:

```
alert(`моя
многострочная
строка`);
```

Заметим, что пробелы и, собственно, перевод строки также входят в строку, и будут выведены.

Можно вставлять выражения при помощи ${…}.

```
let apples = 2; let oranges = 3;
alert(`${apples} + ${oranges} = ${apples + oranges}`); // 2 + 3 = 5
```

Как видно, при помощи ${…} можно вставлять как и значение переменной ${apples}, так и более сложные выражения, которые могут включать в себя операторы, вызовы функций и т.п. Такую вставку называют «интерполяцией».

ДОПОЛНИТЕЛЬНЫЕ МЕТОДЫ ДЛЯ РАБОТЫ СО СТРОКАМИ

str.includes(s) – проверяет, включает ли одна строка в себя другую, возвращает true/false.
str.endsWith(s) – возвращает true, если строка str заканчивается подстрокой s.
str.startsWith(s) – возвращает true, если строка str начинается со строки s.
str.repeat(times) – повторяет строку str times раз.