Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебная дисциплина «Программные средства создания Internet-приложений»

**Инструкция**
по выполнению лабораторной работы
«Работа со стандартными объектами JavaScript: объект Date, объект Math»

Минск
2021

## Лабораторная работа № 21

## Тема работы: Работа со стандартными объектами JavaScript: объект Date, объект Math

### 1. Цель работы

Формирование умений использования стандартных объектов и их методов при написании сценариев на языке JavaScript.

### 2. Задание

Выполнить задания в соответствии с порядком выполнения лабораторной работы с использованием встроенных объектов JavaScript.

### 3. Оснащение работы

ПК, редактор исходного кода, браузер.

### 4. Основные теоретические сведения

В языке JavaScript имеется ряд предопределенных объектов, которыми можно пользоваться при написании сценариев. К ним относятся такие объекты, как **Array**, **Boolean**, **Date**, **Function**, **Math**, **Number**, **RegExp** и **String**, а так же примитивный объект **Object**. Кроме того, имеются встроенные конструкторы для нативных объектов:

```
let x1 = new Object();    // Новый объект Object
let x2 = new String();    // Новый объект String
let x3 = new Number();    // Новый объект Number
let x4 = new Boolean();   // Новый объект Boolean
let x5 = new Array();     // Новый объект Array
let x6 = new RegExp();    // Новый объект RegExp
let x7 = new Function();  // Новый объект Function
let x8 = new Date();      // Новый объект Date
```

Объект Math() отсутствует в списке. Math является глобальным объектом. Ключевое слово new не может использоваться на Math.

Каждый из них используется для определённых целей: для работы со строками, регулярными выражениями, числами, датами и т.п. Для этого каждый объект обладает конкретными свойствами и методами.

Вместо конструкторов рекомендуется использовать литералы:

```
let x1 = {};            // новый объект
let x2 = "";            // новая примитивная строка
let x3 = 0;             // новое примитивное число
let x4 = false;         // новый примитивный Boolean
let x5 = [];            // новый объект массива
let x6 = /()/           // новый объект регулярного выражения
let x7 = function(){};  // новый объект функции
```

В отличие от других языков программирования, JavaScript не имеет такого типа данных, как массив. Но это ограничение обходится благодаря тому, что можно использовать предопределенный объект массива – **Array**.

**Boolean**– это объект, представляющий значение **true** (истина) или **false** (ложь).

```
var myBoolean = true;
```

Не следует путать примитивные логические значения **true** и **false** с типами данных **true** и **false** объекта **Boolean**.

*Например, если объявить переменную* ***x*** *и присвоить ей значение объекта* ***Boolean****, инициализированного при помощи значения* ***false****, она все равно при сравнении будет представлять собой значение истины (****true****):*

```
x = new Boolean(false);
 // при сравнении
if (x) получим true
```

Для работы с датами в языке JavaScript не предусмотрено специального типа данных, однако, как и в случае с массивами, имеется специальный объект **Date**. Для создания объекта даты можно воспользоваться любым из следующих способов:

```
new Date();
new Date(value);
new Date(dateString);
new Date(year, month[, day[, hour[, minute[, second[, millisecond]]]]]);
```

*Например, можно создать объект без аргументов:*

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

Если **Date** вызывается без аргументов, то создаётся объект с текущим временем.

```
alert(new Date()); /* покажет текущее время */
```

Если передаётся один числовой аргумент, то он интерпретируется как количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу.

Если функция Date вызывается в качестве конструктора с более, чем одним аргументом, значения, большие логического диапазона (например, 13 в качестве номера месяца или 70 для значения минут) «переметнутся» на соседние значения.

```
new Date(2013, 13, 1)  /* 2014-02-01 */
оба вызова создадут одинаковую дату
new Date(2014, 1, 1)   /* 2014-02-01 */
```

Нумерация месяцев начинается с нуля. Тоже самое действует и для других значений:

```
new Date(2013, 2, 1, 0, 70)    /* 2013-03-01T01:10:00 */
new Date(2013, 2, 1, 1, 10)    /* 2013-03-01T01:10:00 */
```

Для работы с объектом **Date** предусмотрено 2 свойства – **constructor** и **prototype**.

Свойство **length** объекта **Date** всегда имеет значение 7. Это количество принимаемых аргументов.

Методы наследников **Date** разделяются следующим образом:

- методы, начинающиеся с **«set»**, предназначены для установки даты и времени в объектах **Date**;

- методы, начинающиеся с «**get**», предназначены для извлечения даты, времени или их частей из объектов **Date**;

- методы, начинающиеся с **«to»**, возвращают дату и время (или их части) в виде строковых значений.

Методы объекта **Date** позволяют выделять нужную часть даты (год, месяц, число, время), выводить ее в том или ином формате и т.д.

Методы объекта **Date**, предназначенные для получения отдельных значений, приведены в таблице 12.1.

| Метод | Описание |
|---|---|
| **getFullYear()** | Год из указанной даты по местному времени. |
| **getMonth()** | Месяц из указанной даты по местному времени. Число от 0 до 11. |
| **getDate()** | День месяца из указанной даты по местному времени. Число от 1 до 31. |

| Метод | Описание |
|---|---|
| **getDay()** | День недели из указанной даты по местному времени. Число от 0 до 6. (0 - воскресенье). |
| **getHours()** | Часы из указанной даты по местному времени. Число от 0 до 23. |
| **getMinutes()** | Минуты из указанной даты по местному времени. Число от 0 до 59. |
| **getSeconds()** | Секунды из указанной даты по местному времени. Число от 0 до 59. |
| **getMilliseconds()** | Миллисекунды из указанной даты по местному времени. Число от 0 до 999. |
| **getUTCFullYear()** | Год из указанной даты по всемирному времени. |
| **getUTCMonth()** | Месяц из указанной даты по всемирному времени. Число от 0 до 11. |
| **getUTCDate()** | День месяца из указанной даты по всемирному времени. Число от 1 до 31. |
| **getUTCDay()** | День недели из указанной даты по всемирному времени. Число от 0 до 6. (0 - воскресенье). |
| **getUTCHours()** | Часы из указанной даты по всемирному времени. Число от 0 до 23. |
| **getUTCMinutes()** | Минуты из указанной даты по всемирному времени. Число от 0 до 59. |
| **getUTCSeconds()** | Секунды из указанной даты по всемирному времени. Число от 0 до 59. |
| **getUTCMilliseconds()** | Миллисекунды из указанной даты по всемирному времени. Число от 0 до 999. |
| **getTime()** | Количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу до указанной даты. |
| **getTimezoneOffset()** | Смещение местного времени от всемирного (UTC+0) в минутах. Например, для часового пояса UTC+3 смещение равно -180 минут. |

UTC-варианты, возвращающие день, месяц, год для временной зоны UTC+0. Если местный часовой пояс смещён относительно UTC, то будут выводиться разные часы.

```
// текущая дата
let date = new Date();

// час в текущем часовом поясе
alert( date.getHours() );

// час в часовом поясе UTC+0 (лондонское время без перехода на летнее время)
alert( date.getUTCHours() );
```

Методы объекта **Date**, предназначенные для изменения отдельных значений, приведены в таблице 12.2.

| Метод | Описание |
|---|---|
| **setFullYear()** | Год из указанной даты по местному времени. Дополнительно можно указать месяц и день месяца. |
| **setMonth()** | Месяц из указанной даты по местному времени. Дополнительно можно указать день месяца. |
| **setDate()** | День месяца из указанной даты по местному времени. |
| **setHours()** | Часы из указанной даты по местному времени. Дополнительно можно указать минуты, секунды и миллисекунды. |
| **setMinutes()** | Минуты из указанной даты по местному времени. Дополнительно можно указать секунды и миллисекунды. |
| **setSeconds()** | Секунды из указанной даты по местному времени. Дополнительно можно указать миллисекунды. |
| **setMilliseconds()** | Миллисекунды из указанной даты по местному времени. |

| Метод | Описание |
|---|---|
| **setUTCFullYear()** | Год из указанной даты по всемирному времени. Дополнительно можно указать месяц и день месяца. |
| **setUTCMonth()** | Месяц из указанной даты по всемирному времени. Дополнительно можно указать день месяца. |
| **setUTCDate()** | День месяца из указанной даты по всемирному времени. |
| **setUTCHours()** | Часы из указанной даты по всемирному времени. Дополнительно можно указать минуты, секунды и миллисекунды. |
| **setUTCMinutes()** | Минуты из указанной даты по всемирному времени. Дополнительно можно указать секунды и миллисекунды. |
| **setUTCSeconds()** | Секунды из указанной даты по всемирному времени. Дополнительно можно указать миллисекунды. |
| **setUTCMilliseconds()** | Миллисекунды из указанной даты по всемирному времени. |
| **setTime()** | Количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу до указанной даты. |

Некоторые методы могут устанавливать сразу несколько компонентов даты.

```
setFullYear(year, [month], [date])
setMonth(month, [date])
setHours(hour, [min], [sec], [ms])
setMinutes(min, [sec], [ms])
setSeconds(sec, [ms])
```

```
let today = new Date();

today.setHours(0);
today.setHours(0, 0, 0, 0);
```

Автоисправление – это очень полезная особенность объектов Date. Можно устанавливать компоненты даты вне обычного диапазона значений, а объект сам себя исправит.

```
let date = new Date(2016, 1, 28);
date.setDate(date.getDate() + 2);

alert( date ); // 1 Mar 2016
```

Если объект Date преобразовать в число, то получим таймстамп по аналогии с date.getTime():

```
let date = new Date();
alert(+date); // количество миллисекунд, то же самое, что date.getTime()
```

Для измерения времени предназначен особый метод Date.now(), возвращающий текущую метку времени. Семантически он эквивалентен new Date().getTime(), однако метод не создаёт промежуточный объект Date. Так что этот способ работает быстрее и не нагружает сборщик мусора.

```
let start = Date.now(); // количество миллисекунд с 1 января 1970 года
let end = Date.now(); // заканчиваем отсчёт времени
alert( `Цикл отработал за ${end - start} миллисекунд` ); // вычитаются числа,
                                                         // а не даты
```

Метод Date.parse(str) считывает дату из строки. Формат строки должен быть следующим: YYYY-MM-DDTHH:mm:ss.sssZ, где:

YYYY-MM-DD – это дата: год-месяц-день;

Символ "T" используется в качестве разделителя;

HH:mm:ss.sss – время: часы, минуты, секунды и миллисекунды.

Необязательная часть 'Z' обозначает часовой пояс в формате +-hh:mm. Если указать просто букву Z, то получим UTC+0.

Возможны и более короткие варианты, например, YYYY-MM-DD или YYYY-MM, или даже YYYY.

Вызов Date.parse(str) обрабатывает строку в заданном формате и возвращает таймстамп (количество миллисекунд с 1 января 1970 года UTC+0). Если формат неправильный, возвращается NaN.

```
let ms = Date.parse('2012-01-26T13:51:50.417-07:00');

alert(ms); // 1327611110417 (таймстамп)
```

В отличие от других глобальных объектов, **Math** не является конструктором. Все свойства и методы **Math** статичны. А поскольку все свойства объекта **Math** являются предопределенными константами, то создавать другие объекты типа **Math** не только не требуется, но и недопустимо, а обращаться к ним надо всегда одним и тем же способом:

```
var CircleLength = diameter * Math.PI;
```

Свойства объекта Math приведены в таблице 12.3.

| Свойство | Описание |
|---|---|
| **Math.PI** | Число π (Пи). Примерно равно 3.14 |
| **Math.E** | Число e (число Эйлера). Примерно равно 2.72 |
| **Math.LN2** | Натуральный логарифм из 2. Примерно равен 0.69 |
| **Math.LN10** | Натуральный логарифм из 10. Примерно равен 2.30 |
| **Math.LOG2E** | Логарифм e по основанию 2. Примерно равен 1.44 |
| **Math.LOG10E** | Логарифм e по основанию 10. Примерно равен 0.43 |
| **Math.SQRT1_2** | Квадратный корень из 1/2. Примерно равен 0.71 |
| **Math.SQRT2** | Квадратный корень из 2. Примерно равен 1.41 |

Все константы определены с максимально возможной в JavaScript точностью (double).

Методы объекта Math приведены в таблице 12.4.

| Метод | Описание |
|---|---|
| **Math.random()** | Случайное число в диапазоне **[0 … 1)** (от **0** включительно до **1** не включительно). |
| **Math.abs(x)** | Модуль числа **x** или **NaN**, если передано не число. |
| **Math.ceil(x)** | Округляет число **x** в большую сторону. |
| **Math.floor(x)** | Округляет число **x** в меньшую сторону. |
| **Math.round(x)** | Округляет число **x** в ближайшую сторону. |
| **Math.exp(x)** | Число, равное **e** в степени **x**. **e** - число Эйлера. |
| **Math.pow(x, y)** | Число, равное **x** в степени **y**. |
| **Math.log(x)** | Натуральный логарифм числа **x**. |
| **Math.sqrt(x)** | Квадратный корень числа. |
| **Math.max(x1, …)** | Наибольшее число из переданных аргументов. |
| **Math.min(x1, …)** | Наименьшее число из переданных аргументов. |

| Метод | Описание |
|---|---|
| **Math.cos(x)** | Косинус угла **x** (в радианах). |
| **Math.acos(x)** | Арккосинус числа **x**. Результат в радианах. |
| **Math.sin(x)** | Синус угла **x** (в радианах). |
| **Math.asin(x)** | Арксинус числа **x**. Результат в радианах. |
| **Math.tan(x)** | Тангенс угла **x** (в радианах). |
| **Math.atan(x)** | Арктангенс числа **x**. Результат в радианах. |
| **Math.atan2(y, x)** | Угол для точки с координатами **(x, y)**. Результат в радианах. |

Все тригонометрические функции значения углов принимают и возвращают в радианах. Градусы и радианы связаны следующей формулой: угол_в_радианах = угол_в_градусах * π / 180.

```
var numberRandom = Math.round((Math.random()*5+1));
```

Как и большинство встроенных объектов в Javascript, объект Math может быть расширен пользовательскими способами и методами. Чтобы расширить объект Math, не используют prototype. Вместо этого, расширяют Math напрямую:

```
Math.propName = propValue;
Math.methodName = methodRef;
Новый метод возвращает наибольший общий делитель списка аргументов */
Math.gcd = function() {
    if (arguments.length == 2) {
        if (arguments[1] == 0)
            return arguments[0];
        else
            return Math.gcd(arguments[1], arguments[0] % arguments[1]);
    } else if (arguments.length > 2) {
        var result = Math.gcd(arguments[0], arguments[1]);
        for (var i = 2; i < arguments.length; i++)
            result = Math.gcd(result, arguments[i]);
        return result;
    }
};
```

Для работы со строками предназначен объект **String**, поэтому также можно использовать конструктор String:

```
let name = new String("Tom");
```

Помимо обычных символов JavaScript предоставляет возможность вводить в строку специальные символы, которые нельзя ввести напрямую с клавиатуры, для этого используются управляющие последовательности (escape sequences)? Nf,kbwf 12.5.

| Код | Символ |
|---|---|
| \0 | Ноль |
| \' | одиночная кавычка (апостроф) |
| \" | двойная кавычка |
| \\ | обратный слэш |
| \n | новая строка |
| \r | возвращение каретки |
| \v | вертикальный tab |

| | |
|---|---|
| \t | Tab |
| \b | Backspace |
| \f | смена страницы |
| \uXXXX | Unicode символ, указанный с помощью шестнадцатеричного числа XXXX |
| \xXX | символ Latin-1 |

Методы объекта String приведены в таблице 12.6.

| Метод | Описание |
|---|---|
| charAt() | Возвращает символ строки с указанным индексом (позицией). |
| charCodeAt() | Возвращает числовое значение Unicode символа, индекс которого был передан методу в качестве аргумента. |
| concat() | Возвращает строку, содержащую результат объединения двух и более предоставленных строк. |
| fromCharCode() | Возвращает строку, созданную с помощью указанной последовательности значений символов Unicode. |
| indexOf() | Возвращает позицию первого символа первого вхождения указанной подстроки в строке. |
| lastIndexOf() | Возвращает позицию последнего найденного вхождения подстроки или -1, если подстрока не найдена. |
| localeCompare() | Возвращает значение, указывающее, эквивалентны ли две строки в текущем языковом стандарте. |
| match() | Ищет строку, используя предоставленный шаблон регулярного выражения, и возвращает результат в виде массива. Если совпадений не найдено, метод возвращает значение null. |
| replace() | Ищет строку для указанного значения или регулярного выражения и возвращает новую строку, где указанные значения будут заменены. Метод не изменяет строку, для которой он вызывается. |
| search() | Возвращает позицию первого соответствия указанной подстроки или регулярного выражения в строке. |
| slice() | Позволяет извлечь подстроку из строки. Первый аргумент указывает индекс с которого нужно начать извлечение. Второй необязательный аргумент указывает позицию, на которой должно остановиться извлечение. Если второй аргумент не указан, то извлечено будет все с той позиции, которую указывает первый аргумент, и до конца строки.<br><br>В качестве аргументов можно передавать отрицательные значения, в этом случае начальная или конечная точка извлечения отсчитывается с конца строки. Последний символ строки соответствует индексу -1, второй с конца -2 и т.д.<br><br>Метод возвращает новую строку, содержащую извлеченную подстроку. |
| split() | Разбивает строку на подстроки, возвращая массив подстрок. В качестве аргумента можно передать символ разделитель (например запятую), используемый для разбора строки на подстроки. |

| | |
|---|---|
| substr() | Позволяет извлечь подстроку из строки. Первый аргумент указывает индекс с которого нужно начать извлечение. Второй аргумент указывает количество символов, которое нужно извлечь. |
| substring() | Извлекает символы из строки между двух указанных индексов, если указан только один аргумент, то извлекаются символы от первого индекса и до конца строки. |
| toLocaleLowerCase() | Преобразует символы строки в нижний регистр с учетом текущего языкового стандарта. |
| toLocaleUpperCase() | Преобразует символы строки в верхний регистр с учетом текущего языкового стандарта. |
| toLowerCase() | Конвертирует все символы строки в нижний регистр и возвращает измененную строку. |
| toString() | Возвращает строковое представление объекта. |
| toUpperCase() | Конвертирует все символы строки в верхний регистр и возвращает измененную строку. |
| trim() | Удаляет пробелы в начале и конце строки и возвращает измененную строку. |
| valueOf() | Возвращает примитивное значение объекта. |

```
var str = 'someThing Thin';
alert(str.toUpperCase()); // 'SOMETHING THIN'

var str = 'something thin';
alert(str.substr(4)); // 'thing thin'
alert(str.substr(4, 0)); // ''
alert(str.substr(-4, 4)); // 'thin'

var str = 'something thin';
alert(str.split('thin')); // 'some,g ,'
alert(str.split('')); // 's,o,m,e,t,h,i,n,g, ,t,h,i,n'
```

Регулярные выражения - это шаблоны, используемые для сопоставления последовательностей символов в строках. В JavaScript регулярные выражения также являются объектами **RegExp**. Регулярное выражение можно создать двумя способами: с помощью литерала или конструктора.

```
var first = /j$/;
var first = new RegExp("j$");
```

Общий синтаксис шаблонов регулярных выражений выглядит так:

```
var foo = /шаблон/[флаги];
var foo = new RegExp("шаблон"[,"флаги"]);
```

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

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

Слеши 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 regexp = new RegExp(`<${tag}>`); // то же, что /<h2>/  при ответе "h2" на prompt выше
```

Регулярные выражения могут иметь флаги, которые влияют на поиск. В JavaScript их всего шесть. Флаги регулярных выражений приведены в таблице 12.7.

| Flag | Description |
|---|---|
| G | Глобальный поиск: с этим флагом поиск ищет все совпадения, без него – только первое |
| I | Регистронезависимый поиск: с этим флагом поиск не зависит от регистра: нет разницы между `A` и `a`. |
| M | Многострочный поиск. |
| Y | Выполняет поиск начиная с символа, который находится на позиции свойства lastindex текущего регулярного выражения. |
| S | Включает режим «dotall», при котором точка . может соответствовать символу перевода строки \n |
| U | Включает полную поддержку юникода. |

Свойства объекта RegExp приведены в таблице 12.8.

| Свойство | Описание |
|---|---|
| constructor | Ссылается на функцию-конструктор, которая была использована при создании объекта. |
| global | Содержит булево значение, указывающее, используется ли флаг g в регулярном выражении. |
| ignoreCase | Содержит булево значение, указывающее, используется ли флаг i в регулярном выражении. |
| lastIndex | Содержит индекс, указывающий позицию в исходной строке текста, с которой начнётся поиск следующего соответствия. Первоначально индекс всегда равен 0. |
| multiline | Содержит булево значение, указывающее, используется ли флаг m в регулярном выражении. |
| prototype | Ссылается на объект, являющийся прототипом для объектов типа RegExp. Данное свойство используется интерпретатором, когда функция используется как конструктор при создании нового объекта. Любой объект, созданный с помощью конструктора, наследует все свойства объекта, на который ссылается свойство prototype. |
| source | Содержит исходный код регулярного выражения в виде строки (без открывающей и закрывающей косой черты и без флагов) – как строковой шаблон для конструктора. |

Методы объекта RegExp приведены в таблице 12.9.

| Метод | Описание |
|---|---|

| | |
|---|---|
| exec() | Выполняет поиск в строке, используя шаблон регулярного выражения и возвращает массив, содержащий результаты поиска. Если совпадений не найдено, то метод вернет null. |
| test() | Проверяет, есть ли в переданном тексте соответствия шаблону. Возвращает значение true, если совпадение найдено, или false, если совпадений нет. |
| toString() | Возвращает строковое значение регулярного выражения. |

```
let str = "Любо, братцы, любо!";

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

## 5. Порядок выполнения работы

1. Напишите сценарий, который бы в зависимости от поры года выводил сообщение о месяцах, которые относятся к данной поре. Пора года должна определяться относительно текущей даты.

2. Напишите сценарий, который по введенной дате рождения определяет возраст человека.

3. Добавьте на страницу 2 кнопки: start и end. Напишите сценарий, который выводит информацию о промежутке времени, который прошел между нажатиями кнопок.

4. Выполните задание в соответствии с вариантом, таблица 12.10

Таблица 12.1 – Варианты заданий

| Вариант | Задания |
|---|---|
| 1 | 1. Создайте массив целых чисел. Напишите сценарий, который находит максимальный элемент в массиве.<br>2. Напишите сценарий, который из переданной строки позволяет извлечь подстроку в соответствии с указанной позицией. |
| 2 | 1. Дополните объект Math методом, который находит среднее значение элементов массива.<br>2. Напишите сценарий, который запрашивает у пользователя символ и возвращает числовое значение Unicode символа. |
| 3 | 1. Напишите сценарий, который находит расстояние между двумя введенными пользователем координатами.<br>2. Напишите сценарий, который конвертирует все символы переданной строки в верхний регистр и возвращает как исходную строку, так и измененную. |
| 4 | 1. Дополните объект Math методом, который возвращает количество элементов в массиве.<br>2. Напишите сценарий, который запрашивает у пользователя число в соответствующем диапазоне и возвращает символ строки с указанным индексом. |
| 5 | 1. Напишите сценарий, который возвращает наименьший из переданных аргументов.<br>2. Напишите сценарий, который для строки выполняет поиск и возвращает позицию первого символа первого вхождения указанной подстроки в строке. |

5. Через прототип расширьте встроенный объект Number методом isOdd(), который возвращает true, если число нечетное.

6.* Напишите функцию, которая получает в качестве параметра переменную содержащую строку и с помощью регулярного выражения проверяет, начинается ли содержимое этой переменной с цифры или нет.

## 6. Форма отчета о работе

*Лабораторная работа № ___*

*Номер учебной группы* _____

*Фамилия, инициалы учащегося* _____

*Дата выполнения работы* _____

*Тема работы:* _____

*Цель работы:* _____

*Оснащение работы:* _____

*Результат выполнения работы:* _____

_____

_____

## 7. Контрольные вопросы и задания

1. Перечислите известные способы создания объектов в JavaScript?

2. Приведите примеры создания объектов при помощи стандартных объектов JavaScript.

3. На какие категории можно разделить методы объекта Date?

4. В чем отличие при методов Date. getHours() и Date. getUTCHours()?

5. Опишите алгоритм сравнения строк.

## 8. Рекомендуемая литература

1. **JAVASCRIPT.RU** [Электронный ресурс] / Современный учебник JavaScript – 2007—2020 Илья Кантор. – Режим доступа: https://learn.javascript.ru. – Дата доступа: 04.03.2020.

2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. 4-е изд. – СПб.: Питер, 2018.

3. **Симпсон, К.** ES6 и не только / К. Симпсон. – СПб.: Питер, 2017.

4. **Хавербеке, М.** Выразительный JavaScript. Современное веб-программирование / М. Хавербеке – СПб.: Питер, 2019.