Информационные технологии, часть 2, ИСТ (4 семестр)

Наполнение 2 части курса "Информационные технологии" – практическое задание по клиентским Web-технологиям (32 часа лабораторных работ).
Цель работы – создание web-приложения на основе выбранного Javascript-фреймворка и сайта, посвящённого этому фреймворку.
Основные требования к сайту:

- описание установки фреймворка с его официального сайта, ссылки, копии последней стабильной версии;
- описание и документирование основных возможностей фреймворка;
- описание разработанного приложения на основе фреймворка;
- непосредственная работа приложения с сайта, обязательно включающая ввод исходных данных или выбор параметров работы приложения с помощью форм HTML.

Неполный список фреймворков со ссылками на официальные страницы:

JavaScript-фреймворки и библиотеки общего назначения

1. Angular - модульный фреймворк для фронтенд-разработки на основе TypeScript
2. Aurelia JS - ответвление Angular, ставшее отдельным фреймворком
3. Backbone.js - JavaScript-фреймворк, основанный на архитектуре, подобной MVC
4. dhtmlxSuite - библиотека для создания кроссбраузерных web- и мобильных приложений
5. Dojo - модульная библиотека JavaScript с открытым кодом для разработки кроссплатформенных приложений и сайтов
6. Ember - воплощение идеи программировать на JavaScript в парадигме MVC
7. Ext JS - фреймворк для создания приложений на JavaScript
8. Express - позиционируется как минималистичный, быстрый и гибкий фреймворк, использующий все преимущества и мощность Node.js.
9. Google Web Toolkit для разработки фронтенд-приложений на JavaScript
10. jQuery - библиотека, ставшая едва ли не стандартом. Облегчает работу с объектной моделью документа (DOM), имеет огромное количество плагинов и расширений
11. Knockout - ещё один фреймворк для программирования в шаблоне "вид-модель"
12. Meteor JS - платформа для разработки модульных клиент-серверных приложений
13. Node.js - серверная платформа, позволяющая писать не только клиентские, но и серверные решения на JavaScript
14. Polymer - библиотека JavaScript с открытым исходным кодом для создания веб-приложений
15. Prototype - фреймворк на 5 функциях, упрощающий работу с Ajax
16. PureMVC - шаблоны проектирования "Модель-Вид-Контроллер" (Model, View, Controller)
17. Qooxdoo - фреймворк с объектно-ориентированной архитектурой для создания приложений на Яваскрипт
18. React - фреймворк для реактивного программирования на JS
19. Socket - событийно-ориентированная библиотека для создания web-приложений с клиентской и серверной частью
20. SpineJS - небольшой фреймворк, работающий по схеме MVC, похож на Backbone.js
21. Titanium - SDK и облачная платформа для сборки/распространения ПО
22. Vue - построенный на идеях из Angular и React фреймворк для разработки на JavaScript

Специализированные JavaScript-фреймворки и библиотеки

23. canvasjs.com - графики и диаграммы на JavaScript
24. chance - генератор случайного контента
25. chart.js - графики и диаграммы на JavaScript

26. chess.js - игра в шахматы
27. chessboard.js - шахматная доска на сайте
28. D3.js - библиотека JavaScript для создания интерактивных визуализаций и геопозиционирования с использованием веб-стандартов
29. Device - определение типа и ориентации устройства, определение браузера
30. Draggable JS - библиотека для поддержки технологии drag & drop ("перетащил и оставил")
31. FullCalendar.io - мощный календарь с поддержкой событий
32. gridstack.js - создание интерактивных информационных панелей
33. Legra - рисунки из кубиков Лего
34. MathJax - математическая символика в браузерах с использованием разметки MathML, LaTeX и ASCIIMathML
35. moment.js - манипуляции с датой и временем
36. Muuri - проектирование сточных макетов (Grid Layouts)
37. p5.js - инструмент для создания анимаций и игр на JavaScript
38. particles.js - эффекты с анимированными частицами
39. Snap.svg - позволяет создавать и анимировать векторную графику в современных браузерах, поддерживает код SVG
40. three.js - 3D-моделирование и графика на JavaScript
41. Two.js - ещё одна библиотека для 2D-рисования и моделирования
42. Underscore.js - дополнительный функционал для работы с массивами, объектами и функциями
43. Velocity.js - анимационный плагин под JQuery

Возможен выбор фреймворка **не из этого списка** или **написание собственной библиотеки**.
Обратите также внимание, что как "основной" фреймворк jQuery, так и другие фреймворки общего назначения имеют множество расширений, многие из которых сами могут служить отдельной темой.
Рекомендуемый вводный курс конкретно по jQuery (самому распространённому, но сегодня постепенно теряющему популярность фреймворку) – онлайн-учебник от metanit
https://metanit.com/web/jquery/

Как сделать собственную Javascript-библиотеку

Шаг 1. Делаем шаблон

Стараемся сделать код будущей библиотеки защищённым от "внешних воздействий" и конфликтов имён, так, чтобы он был доступен через единое "внешнее имя" объекта библиотеки.

файл lib.js

```
(function(window) { //Анонимная обёртка для всего
 'use strict'; //Строгое соблюдение синтаксиса
 function myLibrary() { //Обёртка для нашего кода
  var _myLibraryObject = {};

  //здесь будут функции библиотеки

  return _myLibraryObject;
 }

 // Чтобы библиотека была глобально доступна,
 //сохраним объект в контексте глобального объекта window
 if( typeof (window.myWindowGlobalLibraryName) === 'undefined') {
  window.myWindowGlobalLibraryName = myLibrary();
 }
})(window); //Обёртка вызывает себя, передавая себе объект window
```

файл lib_test.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Собственная JS-библиотека</title>
</head>
<body>

<script src="lib.js"></script>
<script>
 console.log(myWindowGlobalLibraryName);
</script>
<noscript>Please, switch on Javascript</noscript>

</body></html>
```

Шаг 2. Создаем функции для своей библиотеки

Создаём и прописываем функции, которые будут выполнять нужные действия над своими аргументами и возвращать результаты.

файл lib.js

```
(function(window) { //Анонимная обёртка для всего
 'use strict'; //Строгое соблюдение синтаксиса
 function myLibrary() { //Обёртка для нашего кода
  var _myLibraryObject = {};

  //Просто создаём свойства для своего объекта
  _myLibraryObject.myCustomLog = function (thingToLog) {
   console.log ("My-Custom-Log > Тип переменной : " + typeof(thingToLog));
   console.log ("My-Custom-Log > Является ли числом : " + !isNaN(thingToLog));
   console.log ("My-Custom-Log > Длина : " + (thingToLog).length);
   return console.log (thingToLog);
  };

  return _myLibraryObject;
 }

 // Чтобы библиотека была глобально доступна,
 // сохраним объект в контексте глобального объекта window
 if( typeof (window.myWindowGlobalLibraryName) === 'undefined') {
  window.myWindowGlobalLibraryName = myLibrary();
 }
})(window); //Обёртка вызывает себя, передавая себе объект window
```

файл lib_test.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Собственная JS-библиотека</title>
</head>
<body>
```

```
<script src="lib.js"></script>
<script>
 myWindowGlobalLibraryName.myCustomLog(["Моя библиотека","123"]);
</script>
<noscript>Please, switch on Javascript</noscript>

</body></html>
```

Шаг 3. Создаём приватные или общедоступные свойства объекта библиотеки

Чтобы сохранить настройки или другие свойства, не представляющие интереса для пользователя (или просто некие промежуточные данные), мы можем объявить обычную переменную settings в области действия нашей функции, и ее нельзя будет получить за пределами оболочки.
Хотя settings не может быть получена за пределами вашей оболочки, можно создать метод, который всё-таки позволяет это (см. getSettings)

файл lib.js

```
(function (window) { //Анонимная обёртка для всего
 'use strict'; //Строгое соблюдение синтаксиса
 function myLibrary() { //Обёртка для нашего кода
  var _myLibraryObject = {};

  // Эта переменная будет недоступна для пользователя,
  // она будет видна только в пределах вашей библиотеки
  var settings = {
   volume: 100,
   mute:   false
  };

  //Методы для изменения и/или получения приватных свойств:

  _myLibraryObject.setVolume = function (volume) {
   if (isNaN(volume) || !isFinite(volume)) volume = 0;
   if (volume < 0) volume = 0;    //Проверяем корректность аргумента,
   if (volume > 100) volume = 100; //при необходимости исправляя их
   settings.volume = volume;
   return volume;
  };

  _myLibraryObject.setMute = function(muteStatus) {
   if (typeof(muteStatus) === 'boolean') { //Проверяем корректность аргумента
    settings.mute = muteStatus;
   }
   else { //с выводом сообщения об ошибке в консоль
    console.error("Настройка mute принимает значения только true или false");
   }
   return settings.mute;
  };

  _myLibraryObject.getVolume = function() {
   return settings.volume;
  };

  _myLibraryObject.getMute = function() {
   return settings.mute;
  };
```

```
  //Просто создаём свойства для своего объекта
  _myLibraryObject.myCustomLog = function (thingToLog) {
   console.log ("My-Custom-Log > Тип переменной : " + typeof(thingToLog));
   console.log ("My-Custom-Log > Является ли числом : " + !isNaN(thingToLog));
   console.log ("My-Custom-Log > Длина : " + (thingToLog).length);
   return console.log (thingToLog);
  };

  //Переменная settings будет представлена этим объектом
  _myLibraryObject.getSettings = function() {
   return settings;
  };

  return _myLibraryObject;
 } //конец myLibrary()

 // Чтобы библиотека была глобально доступна,
 //сохраним объект в контексте глобального объекта window
 if (typeof (window.myWindowGlobalLibraryName) === 'undefined') {
  window.myWindowGlobalLibraryName = myLibrary();
 }
})(window); //Обёртка вызывает себя, передавая себе объект window
```

файл lib_test.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Собственная JS-библиотека</title>
</head>
<body>

<script src="lib.js"></script>
<script>
 console.log (myWindowGlobalLibraryName.setVolume(50));
 console.log (myWindowGlobalLibraryName.setMute(true));
 console.log (myWindowGlobalLibraryName.setVolume(110)); //ошибка!
 console.log (myWindowGlobalLibraryName.getSettings());
</script>
<noscript>Please, switch on Javascript</noscript>

</body></html>
```

Шаг 4. Улучшаем библиотеку и добавляем интерфейс для работы с её методами

Не очень хорошо, что метод `myCustomLog` сам выводит информацию в консоль. Лучше, если он возвращает строку, а вызывающий скрипт решает, что с этой строкой делать.

Кроме того, пользователю нашей библиотеки может понадобиться написать интерфейсный код для работы с ней. В изменённом файле `lib_test.html` данные для логирования берутся из поля ввода с идентификатором `myWindowGlobalLibraryNameData` (наше глобальное имя библиотеки очень длинное, Ваше, возможно, будет короче, но всегда лучше соблюдать единый стиль именования объектов, относящихся к одной предметной области).

файл lib.js

```
(function (window) { //Анонимная обёртка для всего
 'use strict'; //Строгое соблюдение синтаксиса
```

```
function myLibrary() { //Обёртка для нашего кода
 var _myLibraryObject = {};

 // Эта переменная будет недоступна для пользователя,
 // она будет видна только в пределах вашей библиотеки
 var settings = {
  volume: 100,
  mute:   false
 };

 //Методы для изменения и/или получения приватных свойств:

 _myLibraryObject.setVolume = function (volume) {
  if (isNaN(volume) || !isFinite(volume)) volume = 0;
  if (volume < 0) volume = 0;    //Проверяем корректность аргумента,
  if (volume > 100) volume = 100; //при необходимости исправляя их
  settings.volume = volume;
  return volume;
 };

 _myLibraryObject.setMute = function(muteStatus) {
  if (typeof(muteStatus) === 'boolean') { //Проверяем корректность аргумента
   settings.mute = muteStatus;
  }
  else { //с выводом сообщения об ошибке в консоль
   console.error("Настройка mute принимает значения только true или false");
  }
  return settings.mute;
 };

 _myLibraryObject.getVolume = function() {
  return settings.volume;
 };

 _myLibraryObject.getMute = function() {
  return settings.mute;
 };

 //Просто создаём свойства для своего объекта:

 _myLibraryObject.myCustomLog = function (thingToLog) {
  let s =
   "My-Custom-Log > Тип переменной : " + typeof(thingToLog)+"\n"+
   "My-Custom-Log > Является ли числом : " + !isNaN(thingToLog)+"\n"+
   "My-Custom-Log > Длина : " + (thingToLog).length;
  return s;
 };

 //Переменная settings будет представлена этим объектом
 _myLibraryObject.getSettings = function() {
  return settings;
 };

 return _myLibraryObject;
} //конец myLibrary()

// Чтобы библиотека была глобально доступна,
//сохраним объект в контексте глобального объекта window
if (typeof (window.myWindowGlobalLibraryName) === 'undefined') {
 window.myWindowGlobalLibraryName = myLibrary();
}
```

```
})(window); //Обёртка вызывает себя, передавая себе объект window
```

файл lib_test.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Собственная JS-библиотека</title>
</head>
<body>

<p>Введите число или строку и нажмите TAB для обновления информации:</p>
<input type="text" id="myWindowGlobalLibraryNameData" onchange="doit();">
<div id="myWindowGlobalLibraryNameResult"></div>
<script src="lib.js"></script>
<script>
function doit() {
 let val = document.getElementById('myWindowGlobalLibraryNameData').value;
 let res = myWindowGlobalLibraryName.myCustomLog (val);
 document.getElementById('myWindowGlobalLibraryNameResult').innerHTML = res;
}
</script>
<noscript>Please, switch on Javascript</noscript>

</body></html>
```