Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Отчёт

по лабораторной работе №2

«Работа с *JavaScript*»

по дисциплине «Технологии интернет-программирования»

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| студент гр. 820601 | Н. В. Хаджинова |
| Шведов А. Р. |  |
|  |  |

Минск 2021

Содержание

[Введение 3](#_Toc86069489)

[1 Цель работы 4](#_Toc86069490)

[2 Теоретическая часть 5](#_Toc86069491)

[2.1 Объектная модель браузера 5](#_Toc86069492)

[2.2 Методы *alert*(), *confirm*() и *prompt*() объекта *window* 6](#_Toc86069493)

[2.3 Основы языка *JavaScript* 7](#_Toc86069494)

[3 Практическая часть 13](#_Toc86069495)

[3.1 Обновление структуры проекта 13](#_Toc86069496)

[3.2 Руководство пользователя 14](#_Toc86069497)

[Заключение 20](#_Toc86069498)

[Список использованных источников 21](#_Toc86069499)

Введение

Для придания веб-страницам динамизма (например, выпадающие меню, анимации) используются языки написания скриптов. Стандартным скриптовым языком во всемирной паутине является *JavaScript*. Он обычно используется как встраиваемый язык для программного доступа к объектам приложений, а наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

*JavaScript* – это легковесный, интерпретируемый или *JIT*-компилируемый, объектно-ориентированный язык с функциями первого класса. Наиболее широкое применение находит как язык сценариев веб-страниц, но также используется и во множестве других программных продуктов. Это прототипно-ориентированный, мультипарадигменный язык с динамической типизацией, который поддерживает объектно-ориентированный, императивный и декларативный стили программирования.

1. Цель работы

Ознакомиться с основами языка программирования *JavaScript* и возможностями его применения на веб-страницах. Создать новые страницы сайта, на которых продемонстрировать работу методов *alert*(), *confirm*() и *prompt*(), а также работу с переменными примитивных типов, объектами, массивами и функциями.

1. Теоретическая часть
   1. Объектная модель браузера

Веб-страницы бывают статическими и динамическими, последние отличаются тем, что в них используются сценарии (программы) на языке *JavaScript*.

Для того чтобы изменять уже нарисованный браузером экран или создавать новые окна, нужно проникнуть в иерархию объектов браузера, получить доступ к его объектам. В сценариях *JavaScript* браузер предоставляет веб-разработчику множество «готовых» объектов, с помощью которых он может взаимодействовать с элементами веб-страницы и самим браузером. В совокупности же все эти объекты составляют объектную модель браузера (*BOM* – *Browser Object Model*).

Схема объектной модели браузера приведена на рисунке 1.

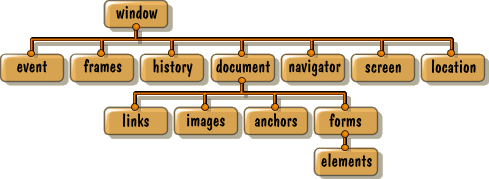


Рисунок 1 – Схема объектной модели браузера

На самом верху этой модели находится глобальный объект *window*. Он представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, *HTML*-страницей и другими объектами. Доступ к этим различным объектам окна браузера осуществляется с помощью следующих основных объектов: *navigator*, *history*, *location*, *screen*, *document* и некоторых других. Так как данные объекты являются дочерними по отношению к объекту *window*, то обращение к ним происходит как к свойствам объекта *window*.

Из всех этих объектов наибольший интерес и значимость для разработчика представляет объект *document*, который является корнем объектной модели документа (*DOM* – *Document Object Model*). Данная модель в отличие от объектной модели браузера стандартизована в спецификации и поддерживается всеми браузерами. Объект *document* представляет собой *HTML*-документ, загруженный в окно (вкладку) браузера. С помощью свойств и методов данного объекта можно получить доступ к содержимому *HTML*-документа, а также изменить его содержимое, структуру и оформление.

* 1. Методы *alert*(), *confirm*() и *prompt*() объекта *window*

Объект *window* является главным в браузере. Он является корнем иерархии всех объектов, доступных веб-разработчику в сценариях *JavaScript*, и представляет собой окно в браузере, причём каждая вкладка содержит свой собственный объект *window*.

Объект *window* кроме глобальных объектов имеет собственные свойства и методы, которые предназначены для следующих задач:

– открытия нового окна (вкладки);

– закрытия окна (вкладки) с помощью метода *close*();

– распечатывания содержимого окна (вкладки);

– передачи фокуса окну или для его перемещения на задний план (за всеми окнами);

– управления положением и размерами окна, а также для осуществления прокручивания его содержимого;

– изменения содержимого статусной строки браузера;

– взаимодействия с пользователем посредством всплывающих окон;

– выполнения определённых действий через определённые промежутки времени и некоторых других задач.

Для взаимодействия с пользователем у объекта *window* присутствуют методы *alert*(), *confirm*() и *prompt*().

Метод *alert*() предназначен для вывода в браузере предупреждающего модального диалогового окна с некоторым сообщением и кнопкой «ОК». При его появлении дальнейшее выполнение кода страницы прекращается до тех пор, пока пользователь не закроет это окно. Кроме этого, оно также блокирует возможность взаимодействия пользователя с остальной частью страницы. Метод имеет единственный аргумент (*message*) – текст сообщения, которое необходимо вывести в модальном диалоговом окне. В качестве результата *alert*() ничего не возвращает.

Метод *confirm*() применяется для вывода модального диалогового окна с сообщением и кнопками «ОК» и «Отмена». Оно обычно используется для запроса у пользователя разрешения на выполнение того или иного действия. Если пользователь нажал на кнопку «ОК», то в качестве результата возвращается *true*, в остальных случаях – *false*.

Метод *prompt*() предназначен для вывода диалогового окна с сообщением, текстовым полем для ввода данных и кнопками «ОК» и «Отмена». Это окно предназначено для запроса данных, которые пользователю нужно ввести в текстовое поле. В качестве результата возвращается значение, введённое пользователем или *null*. Если пользователь не ввёл данные (поле ввода пустое) и нажал на «ОК», то будет возвращена пустая строка. В качестве второго параметра метода можно передать строку, содержащую значение по умолчанию, отображаемую в поле ввода текста.

Синтаксис использования методов представлен ниже:

*alert(message);*

*const confirm\_result = confirm(question);*

*const prompt\_result = prompt(message, default);*

* 1. Основы языка JavaScript
     1. В *JavaScript* имеются следующие примитивные типы данных: *number*, *boolean*, *string*, *undefined*, *null*. Сразу нужно отметить, что при работе с примитивными типами данных, например, со строковыми литералами, даже не проводя явного преобразования, можно обращаться к их методам и свойствам. При попытке выполнения подобных операций литералы автоматически оснащаются соответствующей объектной обёрткой.
     2. В *JavaScript* имеется лишь один тип чисел — это числа двойной точности с плавающей запятой. Поэтому результаты вычисления некоторых выражений имеют соответствующую погрешность.

В *JavaScript* имеется объект *Number*, представляющий собой объектную обёртку для числовых значений.

Существуют глобальные функции, предназначенные для преобразования значений других типов в числовой тип. Это — *parseInt*(), *parseFloat*() и конструкция *Number*(), которая в данном случае выступает в виде обычной функции, выполняющей преобразование типов.

Если в ходе операции с числами получается нечто, не являющееся числом (в ходе неких вычислений, или при попытке преобразования чего-либо в число), *JavaScript* не выдаст ошибку, а представит результат подобной операции в виде значения *NaN* (*Not*-*a*-*Number*, не число). Для того, чтобы проверить, является ли некое значение *NaN*, можно воспользоваться функцией *isNaN*().

Арифметические операции *JavaScript* работают вполне привычным образом, но надо обратить внимание на то, что оператор «+» может выполнять и сложение чисел, и конкатенацию строк.

* + 1. Строки в JavaScript представляют собой последовательности символов *Unicode*. Строковые литералы создают, заключая текст, который нужно в них поместить, в двойные или одинарные кавычки.

Как уже было сказано, при работе со строковыми литералами можно полагаться на соответствующую объектную обёртку, в прототипе которой имеется множество полезных методов, среди которых – *substring*(), *indexOf*(), *concat*().

Строки, как и другие примитивные значения, не изменяемы (*immutable*). Например, метод *concat*() не модифицирует существующую строку, а возвращает в виде результата новую, полученную в результате конкатенации.

Для проверки соответствия строк некоторым шаблонам в языке присутствуют регулярные выражения.

* + 1. Логический тип данных в *JavaScript* представлен двумя значениями – *true* и *false*. Язык может автоматически преобразовывать различные значения к логическому типу данных. Так, ложными, помимо логического значения false, являются значения *null*, *undefined*, '' (пустая строка), 0 и *NaN*. Всё остальное, включая любые объекты, представляет собой истинные значения.

В ходе выполнения логических операций всё, что считается истинным, преобразуется к *true*, а всё, что считается ложным, преобразуется к *false*.

* + 1. Объекты – это динамические структуры, состоящие из пар ключ-значение. Значения могут иметь примитивные типы данных, могут быть объектами или функциями.

Объекты проще всего создавать, используя синтаксис объектных литералов:

*let obj = {*

*message : "A message",*

*doSomething : function() {}*

*}*

Свойства объекта можно, в любое время, читать, добавлять, редактировать и удалять.

Объекты в языке реализованы в виде хэш-таблиц. Простую хэш-таблицу можно создать, используя команду *Object.create(null)*.

Если объект нужно сделать неизменяемым, можно воспользоваться командой *Object.freeze()*.

Для перебора всех свойств объекта можно воспользоваться командой *Object.keys()*.

* + 1. В *JavaScript* переменные можно объявлять, используя ключевые слова *var*, *let* и *const*.

При использовании ключевого слова *var* можно объявить переменную, и, если надо, инициализировать её каким-то значением. Если переменная не инициализирована, её значением является *undefined*. Переменные, объявленные с использованием ключевого слова *var*, имеют функциональную область видимости.

Ключевое слово *let* очень похоже на *var*, разница заключается в том, что переменные, объявленные с ключевым словом *let*, имеют блочную область видимости.

Блочную область видимости имеют и переменные объявленные с помощью ключевого слова *const*, которые, учитывая то, что значения подобных переменных нельзя изменять, правильнее будет называть «константами». Ключевое слово *const*, которое «замораживает» значение переменной, объявленной с его использованием, можно сравнить с методом *Object.freeze*(), «замораживающим» объекты.

Если переменная объявлена за пределами какой-либо функции, её область видимости является глобальной.

* + 1. Массивы в *JavaScript* реализованы с использованием объектов. Как результат, говоря о массивах, фактически, подразумевают объекты, похожие на массивы. Работать с элементами массива можно, используя их индексы. Числовые индексы преобразуются в строки и используются как имена для доступа к значениям элементов массивов. Например, конструкция вида *arr*[1] аналогична конструкции вида *arr*['1'], обе конструкции дадут доступ к одному и тому же значению: *arr*[1] === *arr*['1']. В соответствии с вышесказанным, простой массив, объявленный командой *let arr* = ['*A*', '*B*', '*C*'], представляется в виде объекта примерно следующего вида:

*{*

*'0': 'A',*

*'1': 'B',*

*'2': 'C'*

*}*

Удаление элементов массива с использованием команды *delete* оставляет в нём «дыры». Для того чтобы избежать этой проблемы, можно использовать команду *splice*(), но работает она медленно, так как, после удаления элемента, перемещает оставшиеся элементы массива, сдвигая их влево.

Методы массивов позволяют легко реализовывать такие структуры данных, как стеки и очереди.

* + 1. Функции в *JavaScript* являются объектами. Функции можно назначать переменным, хранить в объектах или массивах, передавать в виде аргументов другим функциям и возвращать из других функций.

Существует три способа объявления функций:

– Классическое объявление функции (*Function Declaration* *или Function Statement*);

– Использование функциональных выражений (*Function Expression*), которые ещё называют функциональными литералами (*Function Literal*);

– Использование синтаксиса стрелочных функций (*Arrow Function*).

Примеры объявления функций такими способами:

*function doSomething(){}*

*let doSomething = function() {}*

*let doSomething = () = > {};*

Кроме того, функции можно вызывать различными способами: обычным, в виде метода объекта, в виде конструктора, в виде функции с использованием метода *apply*(), в виде функции с использованием метода *bind*().

Функции можно вызывать с большим или меньшим количеством аргументов, чем то количество параметров, которое было задано при их объявлении. В ходе работы функции «лишние» аргументы будут просто проигнорированы (хотя у функции будет доступ к ним), отсутствующие параметры получат значение undefined.

У функций есть два псевдо-параметра: this и arguments.

Ключевое слово this представляет собой контекст функции. Значение, на которое оно указывает, зависит от того, как была вызвана функция.

Ключевое слово arguments — это псевдопараметр, который даёт доступ ко всем аргументам, использованным при вызове функции. Он похож на массив, но массивом не является. В частности, у него нет методов массива.

* + 1. Для работы с датой и временем в языке *JavaScript* используются объекты *Date*. При создании объекта без указания параметров он инициализируется текущей датой и временем.

Объекты Date имеют множество методов для получения и установки даты и времени, их преобразования к различным единицам измерения времени, вычисления разности дат и некоторых других операций. Также присутствует механизм авто-исправления некорректных параметров конструктора для даты, например, когда 32-й день месяца преобразуется к первому дню следующего.

* + 1. Метод глобального объекта *eval*() выполняет *JavaScript*-код, представленный строкой-параметром. Метод часто используется для вычисления поступающих арифметических выражений. Однако перед его использованием всегда нужно убедиться, что в поступающей на вход строке не содержится фрагмент кода, который могут использовать злоумышленники.
    2. *JavaScript* является языком с динамической типизацией. Это означает, что конкретные значения имеют типы, а переменные — нет. Во время выполнения программы в одну и ту же переменную можно записывать значения разных типов.

Для выяснения типа данных, хранящихся в переменной, можно использовать оператор *typeof*().

* + 1. Среда выполнения *JavaScript* является однопоточной. Это, в частности, выражается в невозможности одновременного выполнения двух функций (если не учитывать возможности асинхронного выполнения кода, которые мы тут не затрагиваем). В среде выполнения имеется так называемая очередь событий (*Event Queue*), хранящая список заданий, которые нужно обработать. Как результат, для однопоточной схемы выполнения *JS* несвойственна проблема взаимных блокировок ресурсов, поэтому тут не нужен механизм блокировок. Однако, код, попадающий в очередь событий, должен выполняться быстро. Если перегрузить тяжёлой работой, в браузерном приложении, главный поток, страница приложения не будет реагировать на воздействия пользователя и браузер предложит закрыть эту страницу.
    2. В JavaScript имеется механизм для обработки исключений. Работает он по вполне обычному для подобных механизмов принципу: код, который может вызвать ошибку, оформляют с использованием конструкции *try*/*catch*. Сам код находится в блоке *try*, ошибки обрабатываются в блоке *catch*.
    3. В JavaScript функции являются объектами первого класса, язык поддерживает механизм замыканий. Это открывает путь к реализации методик функционального программирования в *JS*. В частности, речь идёт о возможности применения функций высшего порядка.

Замыкание — это внутренняя функция, у которой есть доступ к переменным, объявленным внутри родительской функции, даже после выполнения родительской функции.

Функция высшего порядка — это функция, которая способна принимать другие функции в качестве аргументов, возвращать функции, или делать и то и другое.

* + 1. Для внедрения сценариев в HTML-документ имеются различные подходы:

– помещение кода непосредственно в атрибут события *HTML*-элемента;

– помещение кода внутри тега <*script*>;

– помещение скриптов во внешний файл (с расширением .*js*), а затем связать его с документом *HTML*.

1. Практическая часть
   1. Обновление структуры проекта

Для выполнения лабораторной работы были созданы три новые страницы: *ActionsExamplePage*, *CalculatorPage*, *SignUpPage*. В соответствующих им директориях расположены файлы *index.html*, *index.js*, *styles.css*. Полученная структура проекта приведена на рисунке 2.

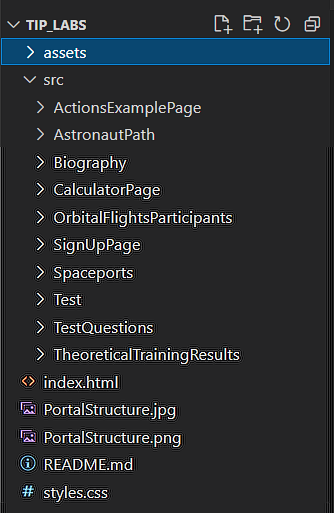


Рисунок 2 – Обновлённая структура проекта

Связи между *html*-документами и соответствующими им локальными *JavaScript*-файлами устанавливались через элемент *script* следующим образом:

*<script type="text/javascript" src="./index.js"></script>* .

* 1. Руководство пользователя
     1. После доработки сайта приведём описание новых страниц для пользователя.

На сайт были добавлены страницы со следующими названиями:

– «Примеры использования Alert, Confirm и Prompt»;

– «Калькулятор»;

– «Форма регистрации».

Осуществить переход на данные страницы можно по ссылкам из «домашней страницы».

На новых страницах сайта также присутствует панель навигации (расположена в «шапке» страниц), а для возврата на «домашнюю страницу» в «подвале» этих страниц присутствует соответствующая ссылка.

* + 1. «Домашняя страница» содержит список ссылок на все информативные страницы сайта. Это первая страница, которая отображается пользователю при открытии сайта. Для перехода на другие страницы достаточно выбрать одну из ссылок в списке навигации.

Домашняя страница сайта представлена на рисунке 3.

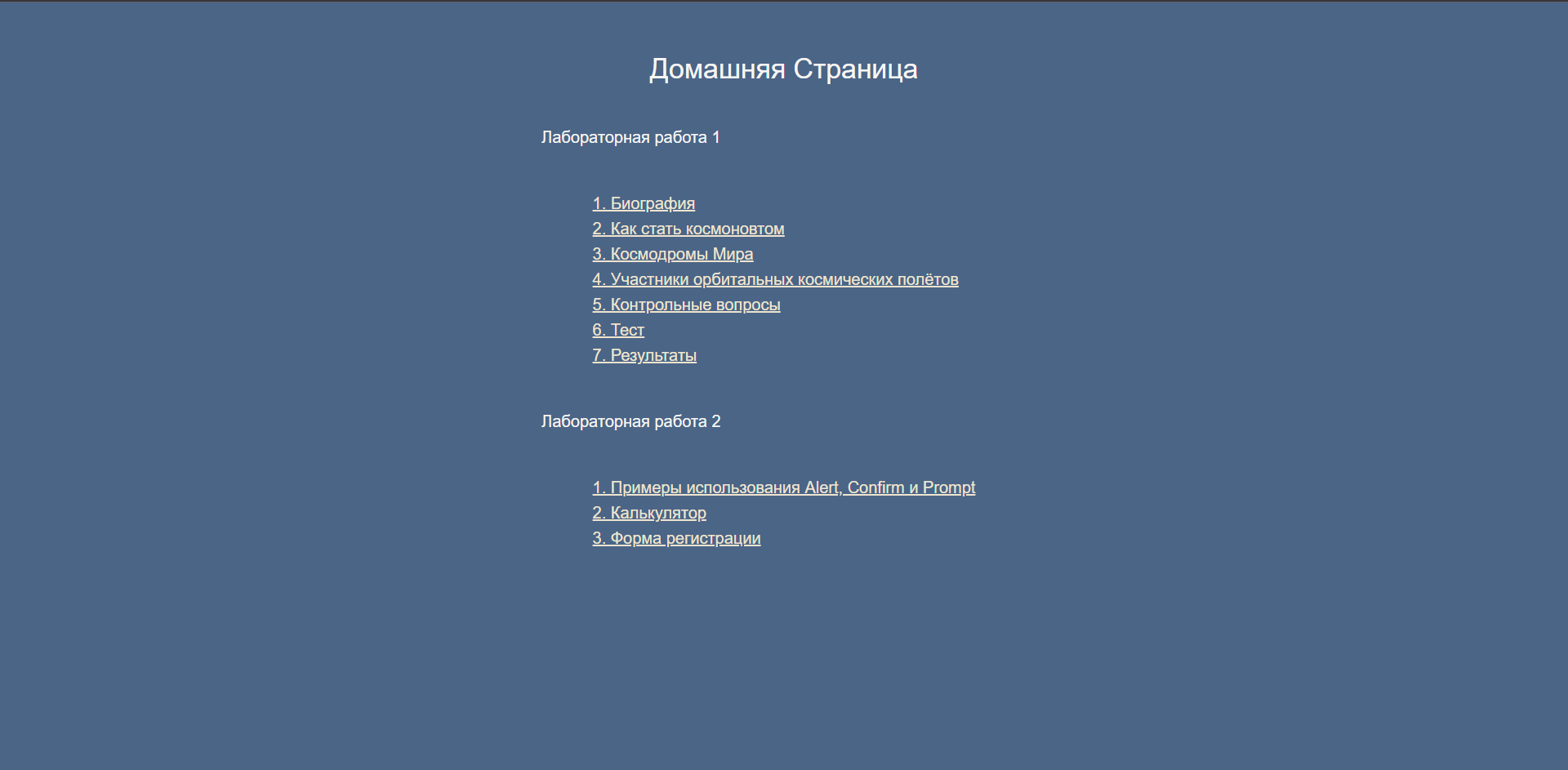


Рисунок 3 – «Домашняя страница»

* + 1. Страница «Примеры использования *Alert*, *Confirm* и *Prompt*» содержит три кнопки для демонстрации взаимодействия сайта с пользователем посредством методов *alert*(), *confirm*() и *prompt*() объекта *window*.

В «шапке» страницы расположена панель навигации для дальнейших переходов по страницам сайта. В «подвале» страницы – ссылка для возврата на «домашнюю страницу».

Страница «Примеры использования *Alert*, *Confirm* и *Prompt*» приведена на рисунке 4.

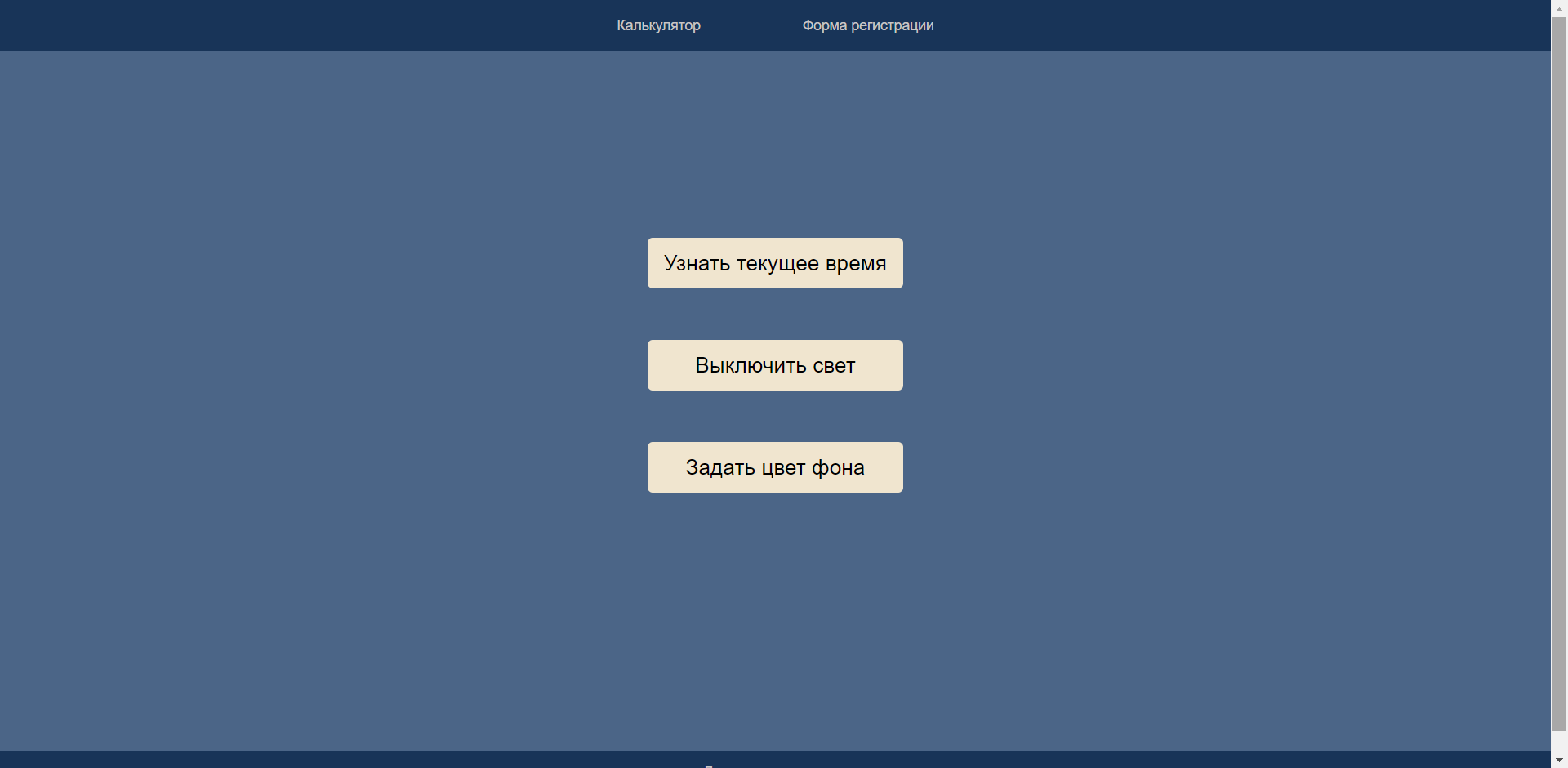


Рисунок 4 – Страница «Примеры использования Alert, Confirm и Prompt»

Взаимодействие страницы с пользователем показано на рисунках 5-7.

При нажатии на кнопку «Узнать текущее время» вверху страницы отображается модальное окно с локальным временем и кнопкой подтверждения «ОК». Результат представлен на рисунке 5.

При нажатии на кнопку «Выключить свет» вверху страницы отображается модальное окно с запросом подтверждения действия и кнопками «ОК» и «Отмена». При утвердительном ответе фон страницы изменяется на тёмный. Повторное нажатие на уже переименованную кнопку «Включить свет» отобразит новое модальное окно с запросом аналогичного подтверждения. При утвердительном ответе возвращается исходный фон страницы. Результат отображён на рисунке 6.

При нажатии на кнопку «Задать цвет фона» вверху страницы отображается модальное окно с запросом ввода нового фонового цвета в шестнадцатиричном формате (полном или сокращённом, например, #*FF0000* эквивалентен #*F00*). При вводе корректного значения цвет фона изменяется на указанный при вводе, иначе – появляется модальное окно с сообщением о некорректном вводе. Результат показан на рисунке 7.

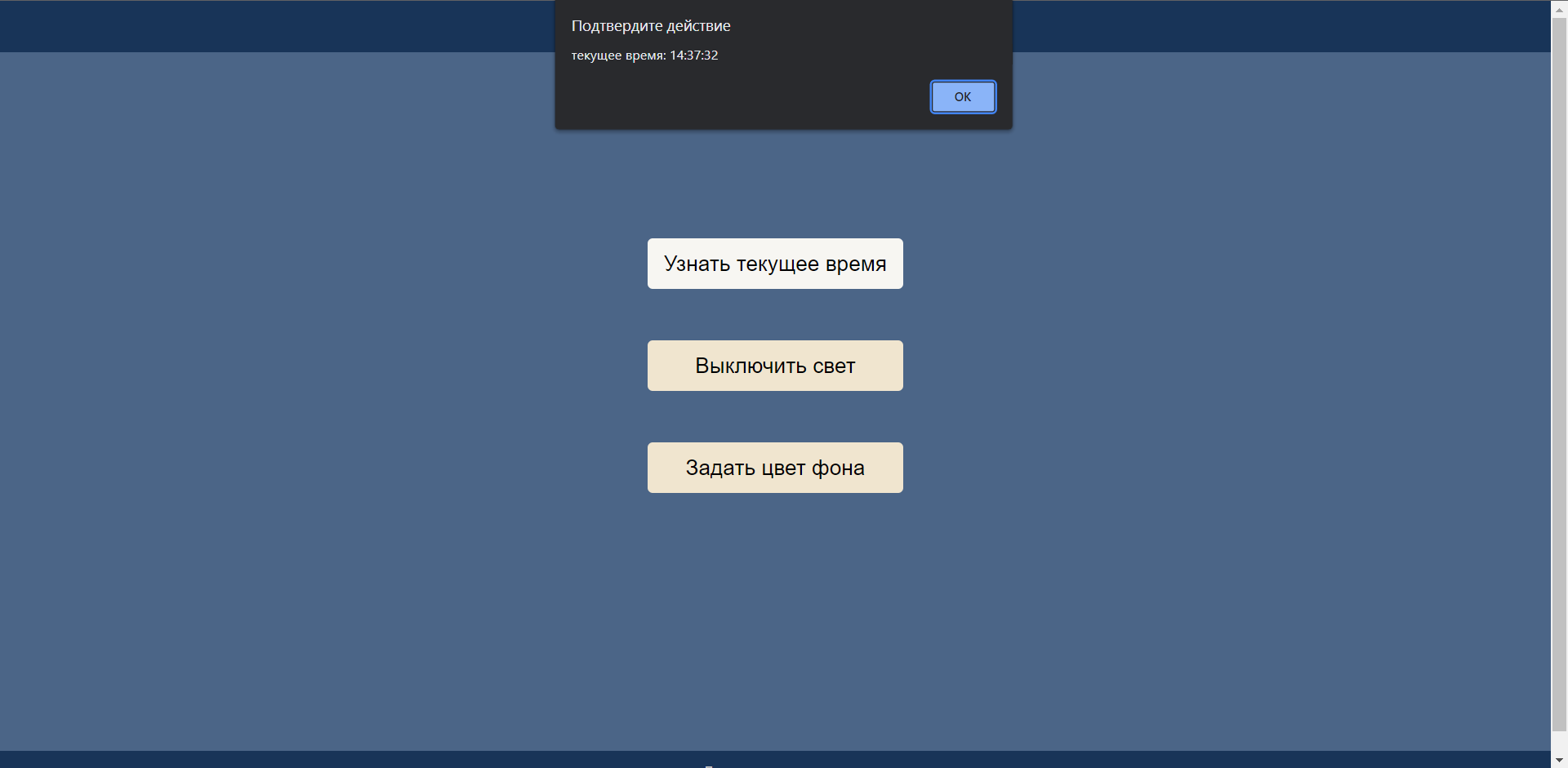


Рисунок 5 –Использование *alert*() для информирования пользователя

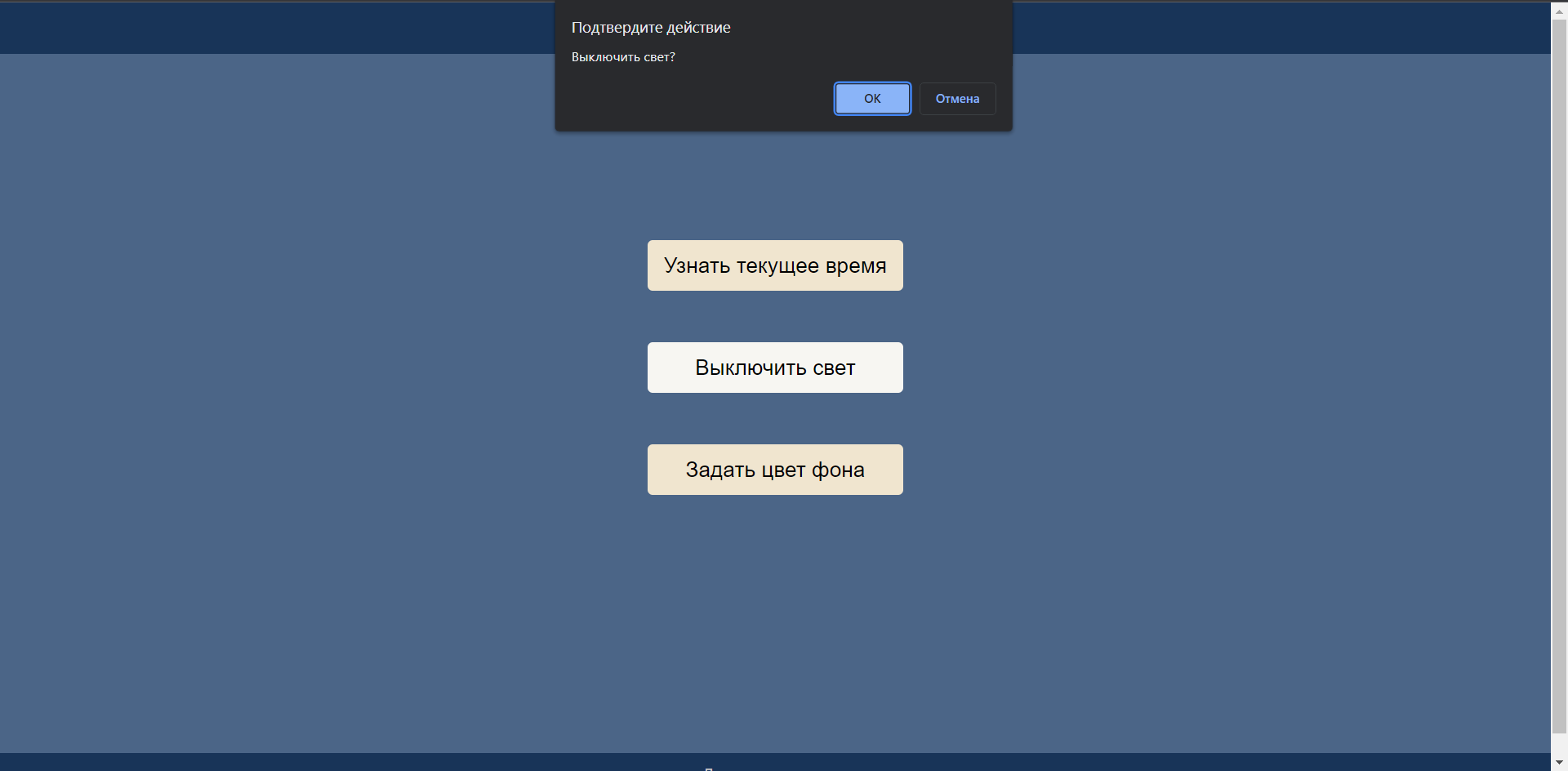


Рисунок 6 – Использование *confirm*() для запроса подтверждения действия от пользователя

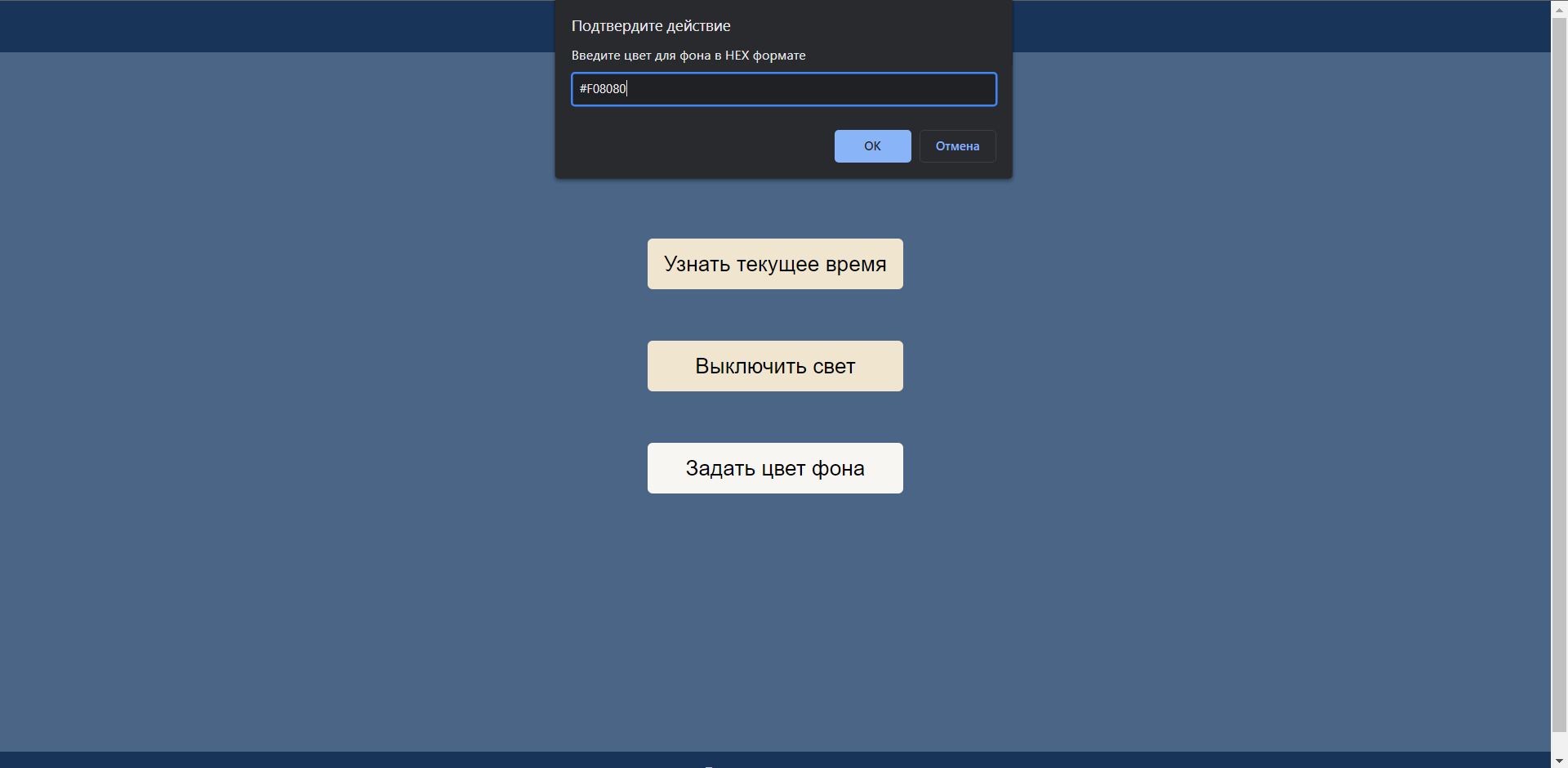


Рисунок 7 – Использование prompt() для запроса ввода данных пользователем

* + 1. Страница «Калькулятор» содержит поле для ввода строки с вычислимым математическим выражением, поле для вывода результата и кнопку «Рассчитать». В «шапке» страницы расположена панель навигации для дальнейших переходов по страницам сайта. В «подвале» страницы – ссылка для возврата на «домашнюю страницу».

Созданная страница представлена на рисунке 8.

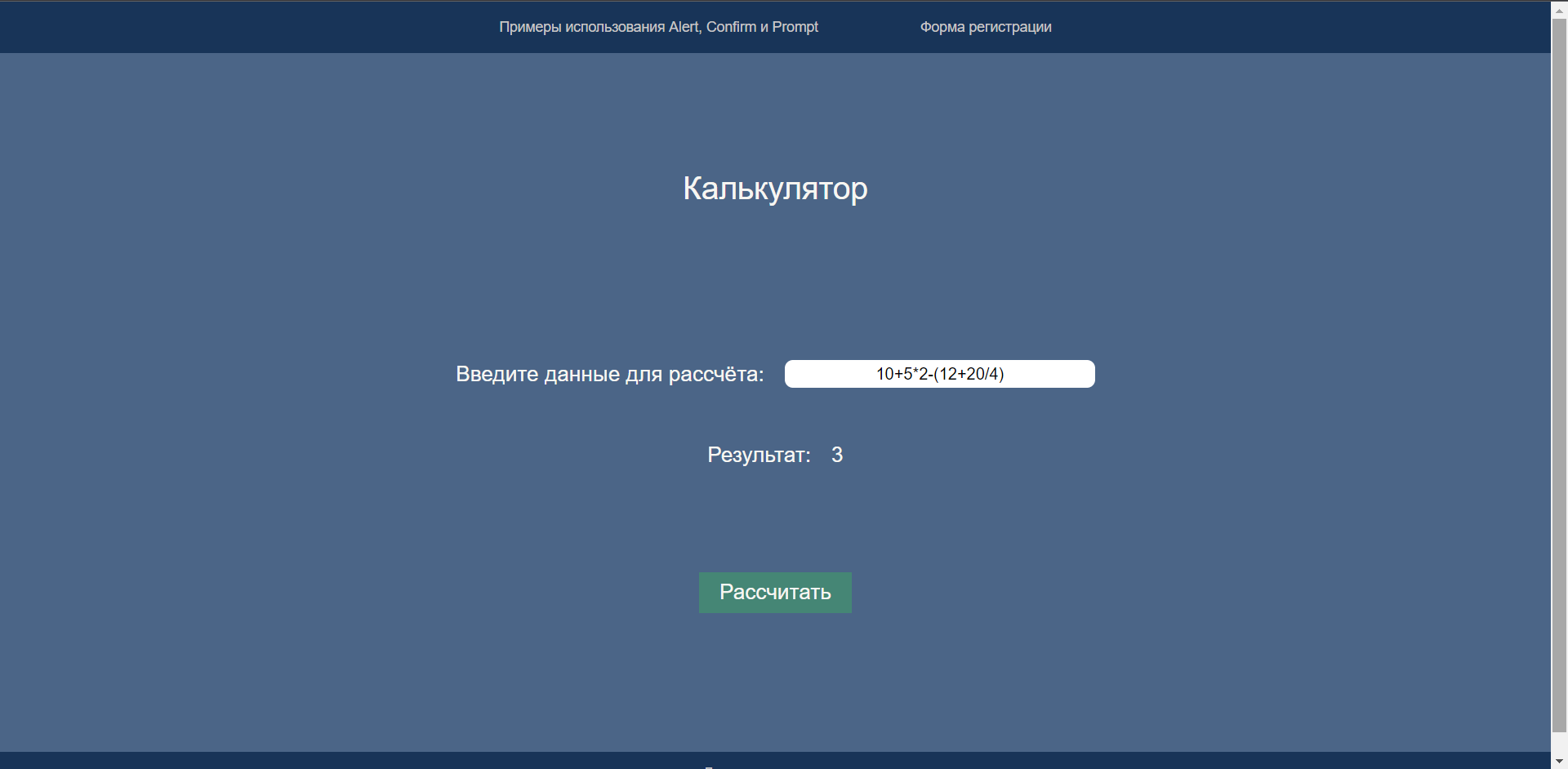


Рисунок 8 – Страница «Калькулятор»

Калькулятор поддерживает следующие математические операции: сложение («+»), вычитание («-»), умножение («\*»), деление («/»), нахождение остатка от деления («%»), постфиксные и префиксные инкременты и декременты(«++», «--»).

При вводе корректного и вычислимого математического выражения его результат будет вычислен и отображён в поле результата. При вводе некорректного выражения в поле результата будет выведено соответствующее сообщение (рисунок 9).

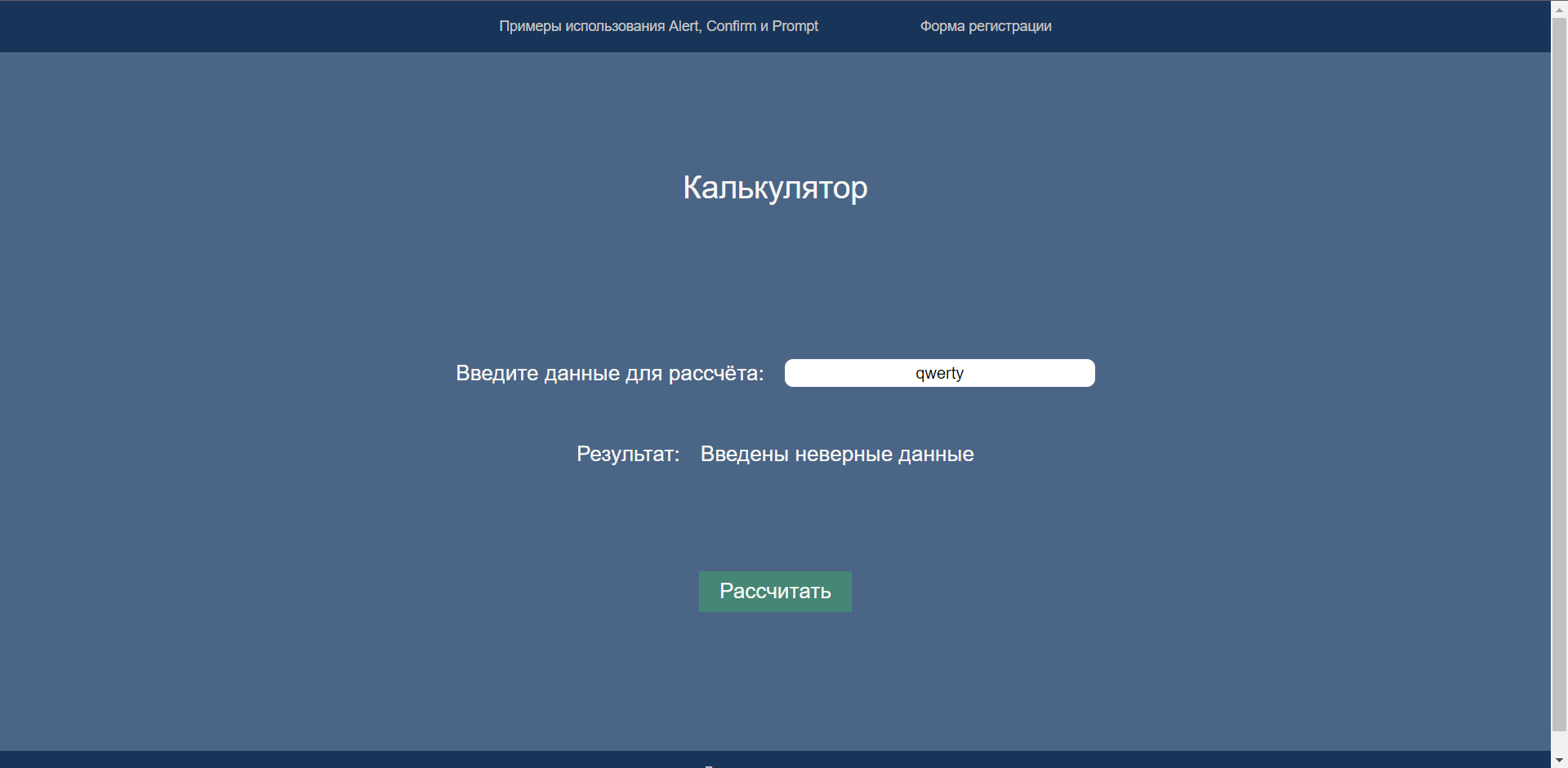


Рисунок 9 – Ввод некорректного математического выражения  
на странице «Калькулятор»

* + 1. Страница «Форма регистрации» содержит форму регистрации для пользователей, в которой необходимо указать имя, фамилию и дату рождения. Язык формы можно выбрать в выпадающем списке, расположенном в «шапке» страницы. По умолчанию выбран русский язык.

Как и на предыдущих страницах, в «шапке» страницы расположена панель навигации для дальнейших переходов по страницам сайта, а в «подвале» страницы – ссылка для возврата на «домашнюю страницу».

Полученная страница на русском и английском языках представлена на рисунке 10 и рисунке 11 соответственно.

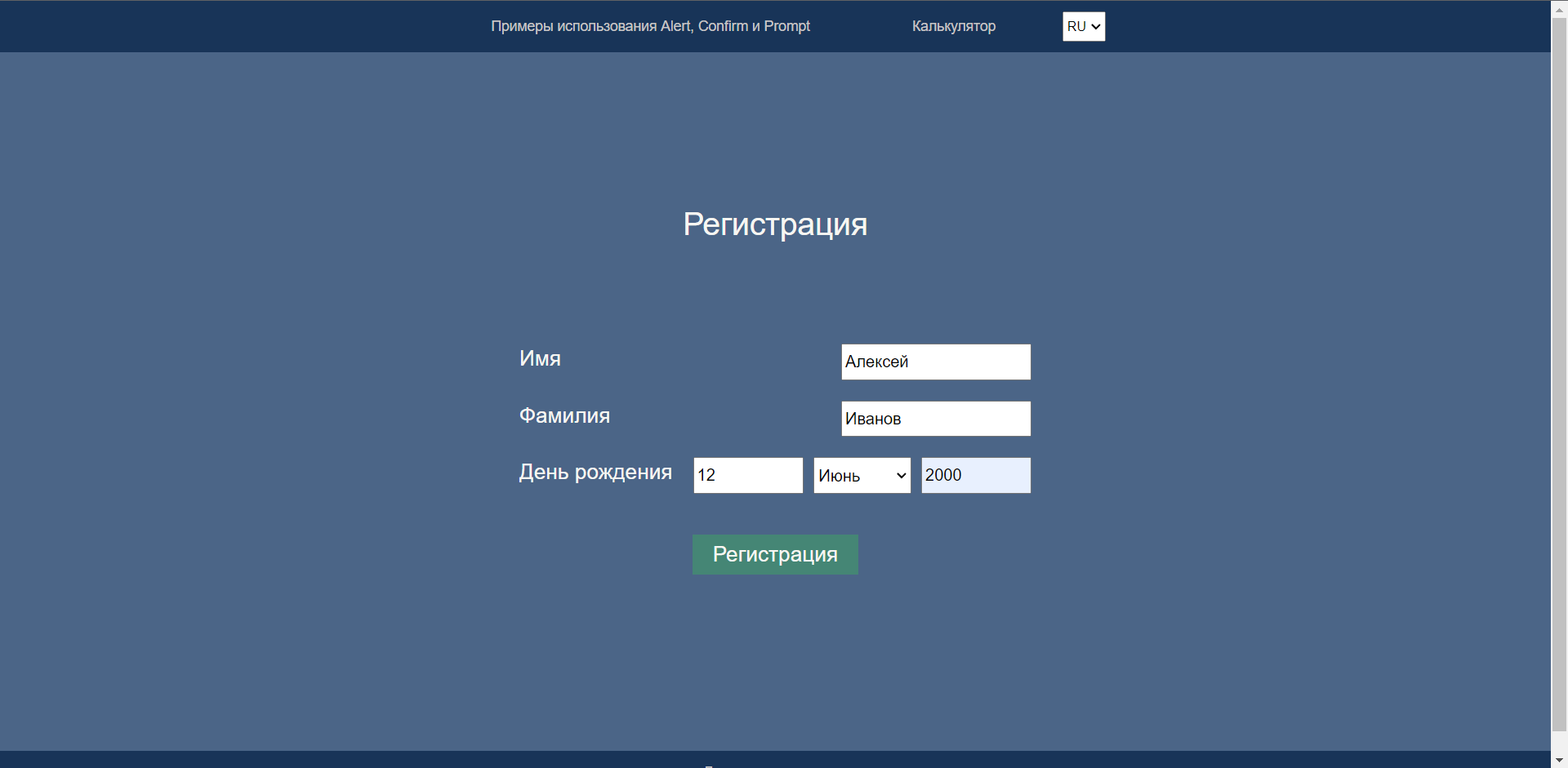


Рисунок 10 – Страница «Форма регистрации» на русском языке

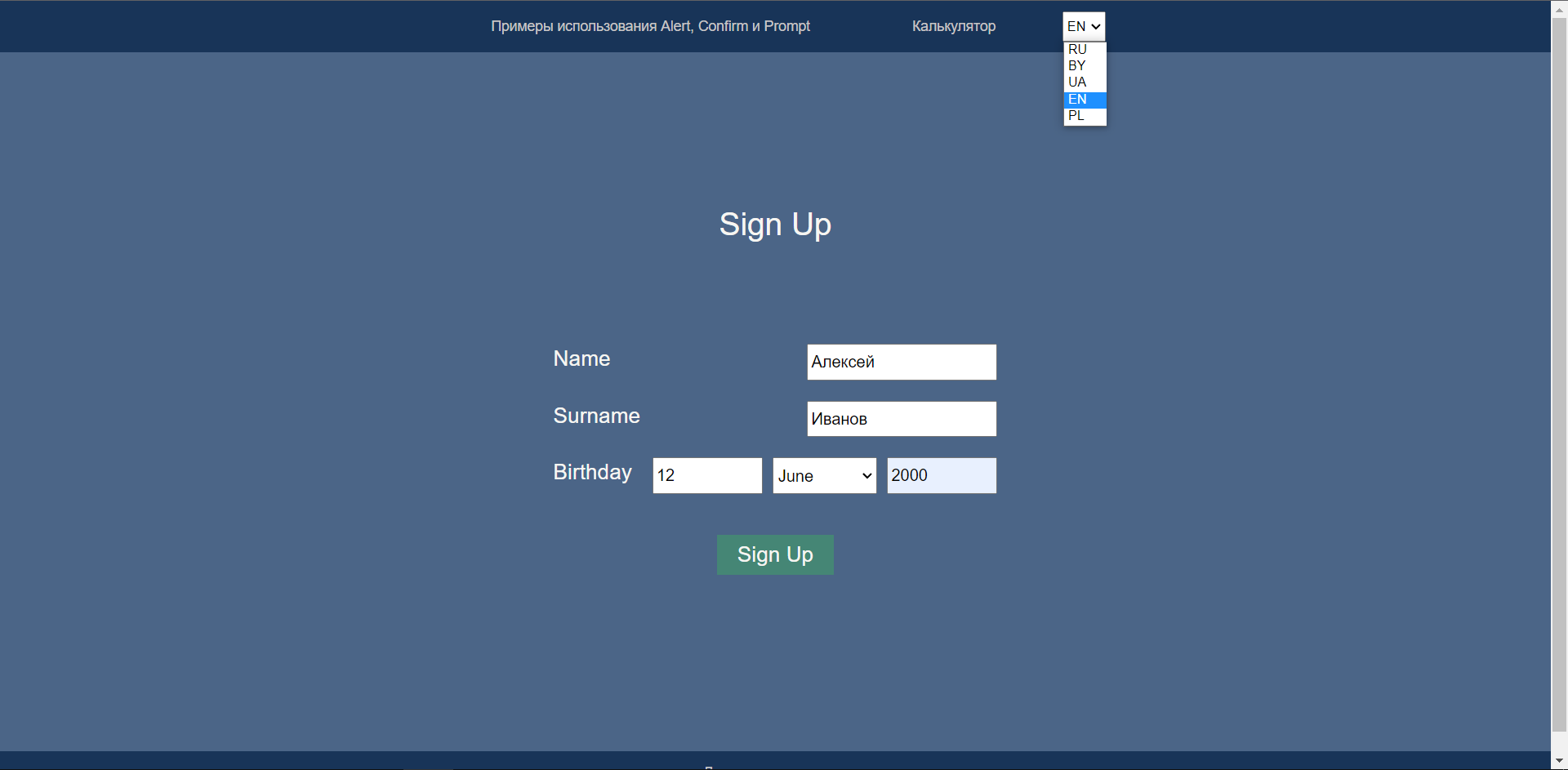


Рисунок 11 – Страница «Форма регистрации» на английском языке

Заключение

В ходе выполнения данной лабораторной работы закрепил теоретические знания основ языка программирования JavaScript, а также получил дополнительные практические навыки его применения для создания динамических веб-страниц.

Список использованных источников

[1]  Флэнаган, Дэвид *JavaScript*. Подробное руководство / Дэвид Флэнаган. – М.: Питер, 2021. – 720 c.

[2]  Крокфорд, Дуглас Как устроен *JavaScript* / Дуглас Крокфорд –   
СПб. : Питер, 2019. – 304 с.

[3]  Справочник по *HTML* и *CSS* [Электронный ресурс]. – Режим доступа : *http*://*htmlbook*.*ru*/.

[4]  *HTML Tutorial* [Электронный ресурс]. – Режим доступа : *[https](https://www.w3schools.com/html/)*[://](https://www.w3schools.com/html/)*[www](https://www.w3schools.com/html/)*[.](https://www.w3schools.com/html/)*[w3schools](https://www.w3schools.com/html/)*[.](https://www.w3schools.com/html/)*[com](https://www.w3schools.com/html/)*[/](https://www.w3schools.com/html/)*[html](https://www.w3schools.com/html/)*[/](https://www.w3schools.com/html/).

ПРИЛОЖЕНИЕ A  
 (справочное)  
Листинг Кода Страниц

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta http-equiv="X-UA-Compatible" content="IE=edge">
6. <meta name="viewport" content="width=device-width, initial-scale=1.0">
7. <link rel="stylesheet" href="./styles.css">
8. <title>Калькулятор</title>
9. </head>
10. <body>
11. <header class="header">
12. <nav>
13. <a href="../ActionsExamplePage/index.html">Примеры использования Alert, Confirm и
14. Prompt</a>
15. <a href="../SignUpPage/index.html">Форма регистрации</a>
16. </nav>
17. </header>
18. <main>
19. <section class="calculator">
20. <h1>Калькулятор</h1>
21. <div class="calculator-inputs">
22. <label for="data-to-calculate" class="data-to-calculate-label">Введите данные для рассчёта:</label>
23. <input id="data-to-calculate" class="data-to-calculate">
24. </div>
25. <div class="calculator-result">
26. <h2 class="result-title">Результат:</h2>
27. <span class="result">0</span>
28. </div>
29. <button class="calculate-button">Рассчитать</button>
30. </section>
31. </main>
32. <footer>
33. <a href="../../index.html" class="home-page-link">Домашнаяя страница<a>
34. </footer>
35. </body>
36. <script type="text/javascript" src="./index.js"></script>
37. </html>

Рисунок А.1 – Исходный код страницы CalculatorPage/index.html

1. const NUMBER\_OR\_DOT\_REGEXP = /[0-9]|\./gi
2. function setup() {
3. const [result] = document.getElementsByClassName("result");
4. const [dataToCalculateInput] = document.getElementsByClassName("data-to-calculate");
5. const [calculateButton] = document.getElementsByClassName("calculate-button");
6. calculateButton.onclick = () => {
7. const inputValue = dataToCalculateInput.value;
8. // to disable code executing in eval() except calculations
9. const safeDataToCalculate = inputValue.replace(/[^-()\d/\*+%. ]/g, '');
10. const dataWithReplacedPrefixAndPostifixOperatorsArray = [];
11. for (let i = 0; i < safeDataToCalculate.length; i++) {
12. if (safeDataToCalculate[i] === '+' || safeDataToCalculate[i] === '-') {
13. const operator = safeDataToCalculate[i] === '+' ? '+' : '-';
14. if (safeDataToCalculate[i+1] === operator) {
15. if (safeDataToCalculate[i-1]?.search(NUMBER\_OR\_DOT\_REGEXP) > -1) {
16. // postfix operator
17. let j = i-1; //last digit pos
18. let number = '';
19. while(safeDataToCalculate[j]?.search(NUMBER\_OR\_DOT\_REGEXP) > -1) {
20. number = dataWithReplacedPrefixAndPostifixOperatorsArray.pop() + number;
21. j--;
22. }
23. dataWithReplacedPrefixAndPostifixOperatorsArray.push(`(${number}${operator}1)`);
24. i += 1;
25. } else {
26. // prefix operator
27. let j = i+2; //first digit pos
28. let number = '';
29. while(safeDataToCalculate[j]?.search(NUMBER\_OR\_DOT\_REGEXP) > -1) {
30. number += safeDataToCalculate[j];
31. j++;
32. }
33. dataWithReplacedPrefixAndPostifixOperatorsArray.push(`(${number}${operator}1)`);
34. i += (number.length + 1);
35. }
36. } else {
37. dataWithReplacedPrefixAndPostifixOperatorsArray.push(safeDataToCalculate[i]);
38. }
39. } else {
40. dataWithReplacedPrefixAndPostifixOperatorsArray.push(safeDataToCalculate[i]);
41. }
42. }
43. const dataToCalculate = dataWithReplacedPrefixAndPostifixOperatorsArray.join('');
44. try {
45. const calculationResult = eval(dataToCalculate);
46. if ( typeof calculationResult !== undefined
47. && calculationResult !== null
48. && !isNaN(parseInt(calculationResult))) {
49. result.textContent = calculationResult;
50. } else {
51. result.textContent = "Введены неверные данные";
52. }
53. } catch(err) {
54. console.error(err);
55. result.textContent = "Введены неверные данные";
56. }
57. };
58. dataToCalculateInput.addEventListener("keyup", function(event) {
59. // Number 13 is the "Enter" key on the keyboard
60. if (event.keyCode === 13) {
61. calculateButton.click();
62. }
63. });
64. }
65. setup();

Рисунок А.2 – Исходный код файла CalculatorPage/index.js

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta http-equiv="X-UA-Compatible" content="IE=edge">
6. <meta name="viewport" content="width=device-width, initial-scale=1.0">
7. <link rel="stylesheet" href="./styles.css">
8. <title>Alert, Confirm, Prompt</title>
9. </head>
10. <body>
11. <header class="header">
12. <nav>
13. <a href="../CalculatorPage/index.html">Калькулятор</a>
14. <a href="../SignUpPage/index.html">Форма регистрации</a>
15. </nav>
16. </header>
17. <main>
18. <section class="actions">
19. <button class="alertButton">Узнать текущее время</button>
20. <button class="confirmButton">Выключить свет</button>
21. <button class="promptButton">Задать цвет фона</button>
22. </section>
23. </main>
24. <footer>
25. <a href="../../index.html" class="home-page-link">Домашнаяя страница<a>
26. </footer>
27. </body>
28. <script type="text/javascript" src="./index.js"></script>
29. </html>
30. </html>

Рисунок А.3 – Исходный код страницы ActionsExamplePage/index.html

1. const LIGHT\_COLOR = "#4B6587";
2. const DARK\_COLOR = "#000000";
3. function setup() {
4. const [body] = document.getElementsByTagName("body");
5. const [alertButton] = document.getElementsByClassName("alertButton");
6. alertButton.onclick = () => {
7. const currentTime = new Date();
8. alert(`текущее время: ${currentTime.toLocaleTimeString()}`);
9. };
10. const [confirmButton] = document.getElementsByClassName("confirmButton");
11. let isLightOn = true;
12. confirmButton.onclick = () => {
13. if (isLightOn) {
14. const result = confirm("Выключить свет?");
15. if (result) {
16. body.style.backgroundColor = DARK\_COLOR;
17. isLightOn = false;
18. confirmButton.textContent = "Включить свет";
19. }
20. } else {
21. const result = confirm("Включить свет?");
22. if (result) {
23. body.style.backgroundColor = LIGHT\_COLOR;
24. isLightOn = true;
25. confirmButton.textContent = "Выключить свет";
26. }
27. }
28. };
29. const [promptButton] = document.getElementsByClassName("promptButton");
30. promptButton.onclick = () => {
31. let color = prompt("Введите цвет для фона в HEX формате");
32. if (!color.includes('#')) {
33. color = `#${color}`;
34. }
35. if (/^#([0-9A-F]{3}){1,2}$/i.test(color)) {
36. body.style.backgroundColor = color;
37. } else {
38. alert('Невеный формат цвета! Введите цвет в HEX формате.');
39. }
40. };
41. };
42. setup();

Рисунок А.4 – Исходный код страницы ActionsExamplePage/index.js

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta http-equiv="X-UA-Compatible" content="IE=edge">
6. <meta name="viewport" content="width=device-width, initial-scale=1.0">
7. <link rel="stylesheet" href="./styles.css">
8. <title>Регистрация</title>
9. </head>
10. <body>
11. <header class="header">
12. <nav>
13. <a href="../ActionsExamplePage/index.html">Примеры использования Alert, Confirm и
14. Prompt</a>
15. <a href="../CalculatorPage/index.html">Калькулятор</a>
16. </nav>
17. <select class="local-select"></select>
18. </header>
19. <main>
20. <section class="sign-up">
21. <h1 class="sign-up-title"></h1>
22. <form class="sign-up-form">
23. <div class="input-container">
24. <label class="label name-label" for="name"></label>
25. <input type="text" id="name">
26. </div>
27. <div class="input-container">
28. <label class="label surname-label" for="surname"></label>
29. <input type="text" id="surname">
30. </div>
31. <div class="input-container">
32. <label class="label birth-day-label" for="birth-day-input"></label>
33. <input id="birth-day-input" type="text" pattern="^[ 0-9]+$">
34. <select class="birth-month-select"></select>
35. <input id="birth-year-input" type="text" pattern="^[ 0-9]+$">
36. </div>
37. <button class="submit-button"></button>
38. </form>
39. </section>
40. </main>
41. <footer>
42. <a href="../../index.html" class="home-page-link">Домашнаяя страница<a>
43. </footer>
44. </body>
45. <script type="text/javascript" src="./index.js"></script>
46. </html>

Рисунок А.5 – Исходный код страницы SignUpPage/in/index.html

1. function setup() {
2. const LOCAL\_STORAGE\_LOCALE = "LOCAL\_STORAGE\_LOCALE";
3. const localeValues = {
4. "RU": {
5. months: ["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь", "Июль", "Август",
6. "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"],
7. name: "Имя",
8. surname: "Фамилия",
9. birthday: "День рождения",
10. signUpTitle: "Регистрация",
11. },
12. "BY": {
13. months: ["Студзень", "Люты", "Сакавiк", "Красавiк", "Май", "Червень", "Лiпень", "Жнiвень",
14. "Верасень", "Кастрычнiк", "Лiстапад", "Снежань"],
15. name: "Імя",
16. surname: "Прозвішча",
17. birthday: "Дзень нараджэння",
18. signUpTitle: "Рэгістрацыя",
19. },
20. "UA": {
21. months: ["Січень", "Лютий", "Березень", "Квітень", "Травень", "Червень", "Липень", "Серпень",
22. "Вересень", "Жовтень", "Листопад", "Грудень"],
23. name: "Iм'я",
24. surname: "Прізвище",
25. birthday: "День народження",
26. signUpTitle: "Реєстрація",
27. },
28. "EN": {
29. months: ["January", "February", "March", "April", "May", "June", "July", "August",
30. "September", "October", "November", "December"],
31. name: "Name",
32. surname: "Surname",
33. birthday: "Birthday",
34. signUpTitle: "Sign Up",
35. },
36. "PL": {
37. months: ["Styczeń", "Luty", "Marzec", "Kwiecień", "Maj", "Czerwiec", "Lipiec", "Sierpień",
38. "Wrzesień", "Październik", "Listopad", "Grudzień"],
39. name: "Nazwa",
40. surname: "Nazwisko",
41. birthday: "Urodziny",
42. signUpTitle: "Rejestracja",
43. },
44. }
45. const [localSelect] = document.getElementsByClassName("local-select");
46. const locals = Object.keys(localeValues);
47. locals.forEach((locale) => {
48. const localOption = document.createElement("option");
49. localOption.innerText = locale;
51. localSelect.append(localOption);
52. });
53. const [signUpTitle] = document.getElementsByClassName("sign-up-title");
54. const [nameLabel] = document.getElementsByClassName("name-label");
55. const [surnameLabel] = document.getElementsByClassName("surname-label");
56. const [birthMonthSelect] = document.getElementsByClassName("birth-month-select");
57. const [birthdayLabel] = document.getElementsByClassName("birth-day-label");
58. const [submitButton] = document.getElementsByClassName(" submit-button");
59. localSelect.onchange = (event) => {
60. renderLocale(event.target.value);
61. localStorage.setItem(LOCAL\_STORAGE\_LOCALE, event.target.value);
62. };
63. const defaultLocale = localStorage.getItem(LOCAL\_STORAGE\_LOCALE) ?? locals[0];
64. localSelect.value = defaultLocale;
65. renderLocale(localSelect.value);
66. function renderLocale(locale) {
67. const values = localeValues[locale];
68. if (values) {
69. nameLabel.textContent = values.name;
70. surnameLabel.textContent = values.surname;
71. signUpTitle.textContent = values.signUpTitle;
72. submitButton.textContent = values.signUpTitle;
73. birthdayLabel.textContent = values.birthday;
74. document.title = values.signUpTitle;
75. birthMonthSelect.innerHTML = "";
76. values.months.forEach((month) => {
77. const monthOption = document.createElement("option");
78. monthOption.innerText = month;
80. birthMonthSelect.append(monthOption);
81. });
82. }
83. }
84. }
85. setup();

Рисунок А.6 – Исходный код страницы SignUpPage /index.js