Методические указания по выполнению лабораторных работ по дисциплине «Скриптовые языки программирования»

Введение

Учебная дисциплина «Скриптовые языки программирования» является составной частью цикла дисциплин по информационным технологиям, изучаемым студентами специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)». Занимая важное место в общепрофессиональной подготовке студентов, учебная дисциплина «Скриптовые языки программирования» обеспечивает подготовку специалиста, владеющего фундаментальными знаниями и практическими навыками в области проектирования, программной реализации и тестирования прикладного программного обеспечения.

Лабораторные занятия предназначены для закрепления и более глубокого изучения определенных аспектов лекционного материала на практике.

Материалы к лабораторным работам располагаются в Интернете по адресу https://github.com/sergei-tsarik/polessu-js.

1. Введение в программирование

1.1. Язык программирования JavaScript

JavaScript – это язык программирования, который используют разработчики для создания интерактивных веб-страниц. Функции JavaScript могут улучшить удобство взаимодействия пользователя с веб-сайтом: от обновления ленты новостей в социальных сетях и до отображения анимации и интерактивных карт. JavaScript является языком программирования при разработки скриптов для выполнения на стороне клиента, что делает его одной из базовых технологий во всемирной сети Интернет. Например, карусель изображения, выпадающее по клику меню и динамично меняющиеся цвета элементов на веб-странице, которые вы видите во время просмотра страниц в Интернете, выполнены при помощи JavaScript.

По мере развития языка, разработчики JavaScript создали библиотеки, фреймворки и практики программирования и начали использовать его за пределами веб-браузеров. Сегодня JavaScript можно использовать для разработки как на стороне клиента, так и на стороне сервера.

Как работает JavaScript?

Все языки программирования работают путем перевода английского синтаксиса в машинный код, который затем выполняет операционная система.

JavaScript в широком смысле можно отнести к категории скриптовых или интерпретируемых языков. Код JavaScript интерпретируется, то есть непосредственно переводится в код машинного языка движком JavaScript. В других языках программирования компилятор обрабатывает весь код в машинный код на отдельном этапе.

Движок JavaScript

Движок JavaScript – это компьютерная программа, которая выполняет код JavaScript. Первые движки JavaScript были всего лишь интерпретаторами, но все современные движки используют для повышения производительности JIT-компиляцию или компиляцию во время выполнения.

JavaScript на стороне клиента

JavaScript на стороне клиента относится к тому, как JavaScript работает в вашем браузере. В этом случае движок JavaScript находится внутри кода браузера. Все основные веб-браузеры имеют свои собственные встроенные движки JavaScript.

Разработчики веб-приложений пишут код JavaScript с разными функциями, связанными с различными событиями, такими как щелчок мыши или наведение курсора. Эти функции вносят изменения в HTML и CSS.

JavaScript на стороне сервера

JavaScript на стороне сервера относится к использованию языка кодирования в логике внутреннего сервера. В этом случае движок JavaScript находится непосредственно на сервере. Функция JavaScript на стороне сервера может обращаться к базе данных, выполнять различные логические операции и реагировать на различные события, запускаемые операционной системой сервера. Основное преимущество скриптинга на стороне сервера заключается в том, что можно в значительной степени настроить ответ сайта в соответствии с вашими требованиями, правами доступа и информационными запросами сайта.

Сторона клиента и сторона сервера

Слово динамический описывает как клиентский, так и серверный JavaScript. Динамическое поведение – это способность обновлять отображение веб-страницы для генерации нового контента по мере необходимости. Разница между JavaScript на стороне клиента и на стороне сервера заключается в генерировании нового контента. Код на стороне сервера динамически генерирует новый контент, используя логику приложения и изменяя данные из базы данных. JavaScript на стороне клиента, с другой стороны, динамически

генерирует новый контент внутри браузера, используя логику пользовательского интерфейса и изменяя контент веб-страницы, которая уже находится на стороне клиента. Смысл немного отличается в этих двух контекстах, но они связаны, и оба подхода работают вместе для улучшения пользовательского опыта.

Другое различие между двумя видами использования JavaScript, помимо реализации динамических функций, заключается в ресурсах, к которым код может получить доступ. На стороне клиента браузер контролирует среду выполнения JavaScript. Код может получить доступ только к тем ресурсам, которые ему разрешает браузер. Например, он не может записать содержимое на ваш жесткий диск, если вы не нажмете на кнопку загрузки. С другой стороны, функции на стороне сервера могут получить доступ ко всем ресурсам серверной машины по мере необходимости.

1.2. NodeJS

Node или Node.js – программная платформа, основанная на движке V8 (компилирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения.

Установка NodeJS

Установочные файлы доступны на официальной странице загрузки https://nodejs.org/en/download/.

- Windows Installer (.msi). Microsoft Installer – системный компонент, обеспечивающий установку (инсталляцию), удаление и обновление программного обеспечения в операционной системе Windows. Сразу после установки вам становится доступна новая команда node.

- Windows Binary (.zip). В этом случае архив с исходным кодом нуобходимо распаковать в директорию C:\Programs\Nodejs\ и добавить в Envoronment Variables переменную среды

NODE_ENV = C:\Programs\node\

PATH = %NODE_ENV%

После этого будет доступна команда node.

Проверить установку можно в терминале командной строки (CMD): node -v.

Команду node можно использовать двумя разными способами.

Первый способ – без аргументов. Откроется интерактивная оболочка (REPL: read-eval-print-loop), где вы можете выполнять обычный JavaScript-код.

Второй способ – передать Node файл с JavaScript для выполнения. Именно так вы и будете практически всегда делать.

Node.js – это просто еще один способ выполнять код на вашем компьютере. Это просто среда для выполнения JavaScript.

Кроме встроенных и пользовательских модулей Node.js существует огромный пласт различных библиотек и фреймворков, разнообразных утилит, которые создаются сторонними производителями и которые также можно использовать в проекте. И они тоже нам доступны в рамках Node.js. Чтобы удобнее было работать со всеми сторонними решениями, они распространяются в виде пакетов. Пакет представляет набор функциональностей.

Для автоматизации установки и обновления пакетов, как правило, применяются систему управления пакетами или менеджеры. Непосредственно в Node.js для этой цели используется пакетный менеджер NPM (Node Package Manager). NPM по умолчанию устанавливается вместе с Node.js, поэтому ничего доустанавливать не требуется.

Этот инструмент предоставляет доступ к колоссальному количеству модулей, созданных сообществом. Доступны тысячи модулей, решающих практически все типичные задачи, с которыми вы, возможно, столкнетесь. Не забывайте проверить существующие модули перед тем, как изобретать велосипед.

Проверить установку nmp можно в терминале командной строки (CMD): npm -v.

Команды npm

Первая команда, которую вы будете использовать при настройке своего проекта: npm init

Команда запросит общую информацию: имя проекта, описание, версию, имя автора, ссылку на GitHub, домашнюю страницу и т. п. После этого будет сгенерирован файл package.json в корневой папке.

Всю введенную информацию можно в любое время отредактировать.

Если вы не хотите вводить эти сведения и предпочитаете оставить все по умолчанию, запустите команду в виде npm init -yes.

Все данные при этом будут взяты из ваших настроек конфигурации.

При помощи этой команды: npm install, можно установить пакет и любые пакеты, от которых он зависит. Пакет – это папка с программой, описанная в package.json или gzip-архив такой папки.

Сокращенное написание команды: npm i.

По умолчанию команда npm install установит все модули, перечисленные как зависимости в package.json.

Для установки отдельного пакета используйте команду в такой форме: npm i package-name.

Чтобы сохранить пакет в зависимостях (dependencies – пакеты, от которых зависит ваш пакет, чаще всего это библиотеки), используйте следующую команду: npm i package-name -save.

По умолчанию все пакеты устанавливаются локально, в папке проекта. Но пакет можно установить и глобально: npm i package-name –global или npm i package-name -g.

Команда для удаления пакетов: npm uninstall. Сокращенная форма: npm un.

Чтобы удалить пакет из установленных, используется следующая команда: npm un package-name.

Эта команда принимает три опциональных флага:

-g – пакет удаляется глобально,

-save – удалить пакет из списка зависимостей,

-D – удалить пакет из списка dev-зависимостей.

Установленный модуль расположится в папке node_modules внутри каталога с вашим приложением. Оттуда его можно подключать так же, как любой встроенный модуль.

1.3. Visual Studio Code

Visual Studio Code – это упрощенный, но мощный редактор исходного кода. Имеет встроенную поддержку JavaScript, TypeScript и Node.js.

Официальный сайт разработчика программы:

https://visualstudio.microsoft.com/ru/#vscode-section.

Установочные файлы доступны на официальной странице загрузки

https://code.visualstudio.com/download.

Дополнительные материалы

- Что такое JavaScript? – https://aws.amazon.com/ru/what-is/javascript/.

- Официальный сайт NMP – https://www.npmjs.com/.

- Редактор кода Visual Studio Code. Самый подробный гайд по настройке и установке плагинов для начинающих – https://habr.com/ru/post/490754/.

Задания для самостоятельной работы

Задание 1. Установка и настройка NodeJS

- Скачайте и установите Nodejs в директорию C:\Programs

- Настройте переменные среды и выполните в командной строке команды: node -v, npm -v.

Задание 2. Установка VS Code

- Скачайте и установите Visual Studio Code.

Задание 3. Запуск js-кода с помощью node

- Создайте файл app.js в директории C:\src\js-first-app\app.js.

- Введите в файл app.js следующий фрагмент кода

```
console.log("Hello JavaScript from Nodejs");
```

- Используя командную строку (CMD) выполните команду

node C:\src\js-first-app\app.js.

Задание 4. Исполнение js-кода в html-документе

- Создайте файл index.html

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="todos.css">
<script type="text/javascript">
	window.alert("Hello JS")
</script>
</head>
<body>
</body>
</html>
```

- Откройте файл index.html с помощью браузера.

Управляемая самостоятельная работа

Дополнить конспект лекций самостоятельно изученными вопросами:

1. Интерпретация программ.

2. Компиляция программ.

2. Встроенные типы и операции с ними

Пять золотых правил

1. При объявлении переменных употребляйте ключевые слова let или const, а не var.

2. Пользуйтесь строгим режимом (strict).

3. Обращайте внимание на типы и избегайте автоматического преобразования типов.

4. Для работы с классами, конструкторами и методами применяйте современный синтаксис.

5. Не используйте ключевое слово this вне конструкторов и методов.

И еще одно метаправило: избегайте «что это?!» – фрагментов странного JavaScript-кода, сопровождаемых саркастическим «Что это?!».

2.1. Примитивы

Значения в JavaScript могут иметь следующие типы:

- число,
- булево значение false или true,
- специальные значения null и undefined,
- строка,
- символ,
- объект.

Все типы, кроме объекта, собирательно называются примитивными. Чтобы узнать тип значения, следует воспользоваться оператором typeof.

Для сохранения значения в переменной служит ключевое слово let: let counter = 0.

Если переменная явно не инициализирована, то она принимает специальное значение undefined: let x. Данное предложение объявляет x и присваивает ей значение undefined.

Если значение переменной не планируется изменять, то следует объявить ее в предложении const: const PI = 3.141592653589793.

Попытка модифицировать объявленное значение как const приведет к ошибке во время выполнения. Рекомендуется объявлять каждую переменную в отдельной строке. Имя переменной должно быть выбрано с соблюдением общего синтаксиса идентификаторов. Рекомендуется «верблюжья нотация» camelCase, когда границы слов обозначаются сменой регистра.

В JavaScript все числа с плавающей точкой двойной точности.

Как и в любом языке программирования, избежать ошибок округления при операциях над числами с плавающей точкой невозможно. Например, 0.1 + 0.2 дает 0.30000000000000004. Это неизбежно, поскольку десятичные числа вроде 0.1, 0.2 или 0.3 не имеют точного двоичного представления.

Для преобразования строки в число предназначены функции parseFloat и parseInt:

```
const notQuitePi = parseFloat('3.14') // число 3.14
const evenLessPi = parseInt('3') // целое число 3
```

Метод toString преобразует число обратно в строку:

```
const notQuitePiString = notQuitePi.toString() // строка '3.14'
```

const evenLessPiString = (3).toString() // строка '3'

Результатом деления на ноль является Infinity или -Infinity. Однако 0 / 0 равно NaN – константе, обозначающей «не число». Некоторые функции, порождающие целые числа, возвращают NaN, когда на вход передано недопустимое значение. Например, parseFloat('pie') равно NaN.

Шаблонным литералом называется строка, которая может содержать выражения и занимать несколько строчек. Такие строки заключаются в обратные кавычки (`...`), например:

let destination = 'world' // обычная строка

let greeting = `Hello, ${destination.toUpperCase()}!` // шаблонный литерал

Выражения внутри ${...} вычисляются, при необходимости преобразуются в строку и подставляются в шаблон. В данном случае результатом будет строка

Hello, WORLD!

Шаблонные литералы могут быть вложенными, т. е. одна конструкция ${...} может встречаться внутри другой:

greeting = `Hello, ${firstname.length > 0 ? `${firstname[0]}. ` : '' } ${lastname}`

Все знаки новой строки внутри шаблонного литерала включаются в строку. Например, в результате вычисления

greeting = ` Hello
${destination}
`

переменной greeting присваивается строка ' Hello \n World \n', в которой каждая строчка завершается знаком перевода строки. Чтобы включить в строку знаки обратной кавычки, доллара или обратной косой черты, следует экранировать их знаком обратной косой черты:

`\`\$\\` – строка, содержащая три символа: `$\.

Тегированным шаблонным литералом называется шаблонный литерал, которому предшествует функция, например:

html`<div>Hello, ${destination}</div>`

Здесь вызывается функция html, которой передаются фрагменты шаблона ' Hello, ' и ' ', и значение выражения destination.

2.2. Операторы сравнения

Чтобы определить, что одна строка больше другой, JavaScript использует «алфавитный» или «лексикографический» порядок. Другими словами, строки сравниваются посимвольно.

Использование обычного сравнения == может вызывать проблемы. Оператор строгого равенства === проверяет равенство без приведения типов. Другими словами, если a и b имеют разные типы, то проверка a === b немедленно возвращает false без попытки их преобразования.

Можно избежать проблем, если следовать надежным правилам:

- Относитесь очень осторожно к любому сравнению с undefined/null, кроме случаев строгого равенства ===.

- Не используйте сравнения >= > < <= с переменными, которые могут принимать значения null/undefined, разве что вы полностью уверены в том, что делаете. Если переменная может принимать эти значения, то добавьте для них отдельные проверки.

2.3. Ветвления

Условное предложение в JavaScript имеет вид: if (условие) предложение.

Необязательная ветвь else выполняется, если условие не удовлетворяется, например:

if (условие) {
предложение для случая, когда условие принимает значение true
} else {
предложение для случая, когда условие принимает значение false
}

Если ветвь else содержит еще одно предложение if, то, по соглашению, используется такой формат:

if (условие1) {
предложение для случая, когда условие1 принимает значение true
} else if (условие2) {
предложение для случая, когда условие1 принимает значение false и условие2 принимает значение true
} else {
предложение для случая, когда условие1 принимает значение false и условие2 принимает значение false
}

«Условный» оператор ? : . Результатом вычисления выражения условие ?

первое : второе является первое, если условие удовлетворяется, и второе в противном случае.

let max = x > y ? x : y

Задания для самостоятельной работы

Задания 1. Conditional Statements and Data Types

Perform addition of various types (string + boolean, string + number, number + boolean).

Perform multiplication of various types (string * boolean, string * number, number * boolean).

Divide different types (string / boolean, string / number, number / Boolean).

Perform explicit conversion (number, string, boolean).

Задание 2. Arrays and Cycles

Given an array consisting of movie names, iterate over the array with the output of the names of each movie to the console.

Given an array of car manufacturers, convert the array to a string and back to an array.

Given an array of the names of your friends, add the words hello to each element of the array.

Convert numeric array to Boolean.

Sort the array [1,6,7,8,3,4,5,6] in descending order.

Filter array [1,6,7,8,3,4,5,6] by value> 3.

Implement a loop that will print the number 'a' until it is less than 10.

Implement a loop that prints prime numbers to the console.

Implement a loop that prints odd numbers to the console.

3. Функции

В JavaScript для объявления функции нужно указать:

- имя функции,
- имена параметров,
- тело функции, в котором вычисляется и возвращается ее результат.

Типы параметров и результата не задаются.

Пример описания функции:

```
function average(x, y) {
    return (x + y) / 2
}
```

Инструкция return служит для возврата вычисленного функцией значения. Чтобы вызвать эту функцию, нужно передать ей фактические аргументы:

let result = average(6, 7) // результат равен 6.5

JavaScript – функциональный язык программирования. Функции являются значениями, которые можно сохранять в переменных, передавать в качестве аргументов или возвращать из функции в качестве значений.

Функция – это действие. Поэтому имя функции обычно является глаголом. Оно должно быть кратким, точным и описывать действие функции, чтобы программист, который будет читать код, получил верное представление о том, что делает функция.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение. Обычно в командах разработчиков действуют соглашения, касающиеся значений этих префиксов.

Например, функции, начинающиеся с «show» обычно что-то показывают.

Функции, начинающиеся с…

"get…" – возвращают значение,

"calc…" – что-то вычисляют,

"create…" – что-то создают,

"check…" – что-то проверяют и возвращают логическое значение, и т.д.

Примеры таких имен:

showMessage(..) // показывает сообщение,

getAge(..) // возвращает возраст (получая его каким-то образом),

calcSum(..) // вычисляет сумму и возвращает результат,

createForm(..) // создает форму (и обычно возвращает ее),

checkPermission(..) // проверяет доступ, возвращая true/false.

Благодаря префиксам, при первом взгляде на имя функции становится понятным, что делает ее код, и какое значение она может возвращать.

Одна функция – одно действие

Функция должна делать только то, что явно подразумевается ее названием. И это должно быть одним действием.

Два независимых действия обычно подразумевают две функции, даже если предполагается, что они будут вызываться вместе (в этом случае мы можем создать третью функцию, которая будет их вызывать).

Несколько примеров, которые нарушают это правило:

getAge – будет плохим выбором, если функция будет выводить alert с возрастом (должна только возвращать его).

createForm – будет плохим выбором, если функция будет изменять

документ, добавляя форму в него (должна только создавать форму и возвращать ее).

checkPermission – будет плохим выбором, если функция будет отображать сообщение с текстом доступ разрешен/запрещен (должна только выполнять проверку и возвращать ее результат).

В этих примерах использовались общепринятые смыслы префиксов. Конечно, можно договориться о других значениях, но обычно они мало отличаются от общепринятых. В любом случае необходимо четко понимать, что значит префикс, что функция с ним может делать, а чего не может.

Синтаксис, который мы использовали до этого, называется Function Declaration (Объявление Функции). Существует ещё один синтаксис создания функций, который называется Function Expression (Функциональное Выражение). Данный синтаксис позволяет нам создавать новую функцию в середине любого выражения.

Это выглядит следующим образом:

```
let sayHi = function() {
  console.log("Привет");
};
```

Поскольку создание функции происходит в контексте выражения присваивания (с правой стороны от =), это Function Expression. Обратите внимание, что после ключевого слова function нет имени. Для Function Expression допускается его отсутствие.

Независимо от того, как создается функция – она является значением.

Дополнительные материалы

- Function Expression – https://learn.javascript.ru/function-expressions.
- Стрелочные функции –https://learn.javascript.ru/arrow-functions-basics.
- Функции – https://learn.javascript.ru/function-basics#obyavlenie-funktsii.
- Способы добавления стилей на веб-страницу – http://htmlbook.ru/samcss/sposoby-dobavleniya-stiley-na-stranitsu.
- Внешние скрипты, порядок исполнения – https://learn.javascript.ru/external-script.

Задания для лабораторной работы

Задание 1. Objects and Functions

Your name is saved in the payment terminal, write a function to define the name in the terminal (if you entered your name, then hello + name, if not, then there

is no such name).

Write a function for calculating the type of argument and type output to the console.

Write a function that determines whether a number is prime or not.

Задание 2. Создание ToDos приложения

Создайте html-документ для реализации «Списка дел To Do List»:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="todos.css">
<script>
function newElement() {
...
}
</script>
</head>
<body>
...
</body>
</html>
```

Подробное описание задания смотрите на веб-странице https://github.com/sergei-tsarik/polessu-js.

Управляемая самостоятельная работа

Дополнить конспект лекций самостоятельно изученными вопросами:

1. Функции высшего порядка – насколько они нужны?
2. Функциональные литералы – насколько они нужны?
3. Стрелочные функции – насколько они нужны?
4. Функциональная обработка массива – насколько они нужны?

4. Объектно-ориентированное программирование в JavaScript

Базовые принципы объектно-ориентированного программирования. Основные положения объектной модели, ее преимущества.

Объектная модель в JavaScript сильно отличается от языков программирования, основанных на классах.

4.1. Объекты

В JavaScript объект – это просто совокупность пар «имя»: «значение», или «свойств», например:

{ name: 'Harry Smith', age: 42, }

У такого объекта есть только открытые данные, ни о какой инкапсуляции или поведении и речи не идет. Объект не является экземпляром какого-то класса. Объектный литерал может завершаться запятой. Это упрощает

добавление новых свойств по мере эволюции кода.

Объект можно сохранить в переменной:

let harry = { name: 'Harry Smith', age: 42 }

Имея такую переменную, можно обращаться к свойствам объекта с помощью стандартной нотации с точкой:

let harrysAge = harry.age

Можно модифицировать существующие свойства и добавлять новые:

harry.age = 40

harry.salary = 90000

Для удаления свойства служит оператор delete:

delete harry.salary

Попытка доступа к несуществующему свойству дает undefined:

let boss = harry.supervisor // undefined

Именем свойства может быть только строка. Если имя не удовлетворяет правилам формирования идентификатора, заключите его в кавычки:

let harry = { name: 'Harry Smith', 'favorite beer': 'IPA' }

Для доступа к таким свойствам нельзя использовать нотацию с точкой.

Применяйте квадратные скобки:

harry['favorite beer'] = 'Lager'

4.2. Классы

Каждый объект трактуется как уникальный набор свойств, отличающийся от любого другого объекта.

В JavaScript можно определить класс объектов, которые разделяют определенный свойства. Члены, или экземпляры, класса имеют собственные свойства для хранения или определения их состояния, но они также располагают методами, которые устанавливают их поведение. Такие методы определяются классом и разделяются всеми экземплярами.

Операция new создает и инициализирует новый объект. За ключевым словом new должен следовать вызов функции. Применяемая подобным способом функция называется конструктором и предназначена для инициализации вновь созданного объекта.

С помощью ключевого слова class можно определить собственные функции конструкторов для инициализации создаваемых новых объектов.

Базовый синтаксис класса выглядит следующим образом.

```
class MyClass {
```

```
  // методы класса
  constructor() { ... }
  method1() { ... }
  method2() { ... }
  method3() { ... }
  ...
}
```

Отметим следующие моменты, касающиеся синтаксиса class:

- класс объявляется с помощью ключевого слова class, за которым следует имя класса и тело класса в фигурных скобках.

- тело класса содержит определения методов, которые применяют сокращенное определение методов.

- ключевое слово constructor используется для определения функции конструктора класса.

При объявлении класса можно перечислить имена и начальные значения полей класса. Свойство объекта (поле класса) также можно создать в конструкторе или любом другом методе, присвоив значение this.имяСвойства.

В классе можно объявлять методы чтения и записи, называемые акцессорами чтения (getter) и записи (setter). Акцессором чтения (getter) называется метод без параметров, объявленный с помощью ключевого слова get:

```
class Person {
constructor(last, first) {
this.last = last;
this.first = first
}
get fullName() { return `${this.last}, ${this.first}` }
}
```

При вызове акцессора чтения скобки не ставятся, как если бы мы обращались к значению свойства:

```
const harry = new Person('Smith', 'Harry')
const harrysName = harry.fullName // 'Smith, Harry'
```

У объекта harry нет свойства fullName, но вызывается акцессор чтения. Можно считать, что акцессор чтения – это динамически вычисляемое свойство. Можно также задать акцессор записи (setter) – метод с одним параметром:

```
class Person {
...
set fullName(value) {
```

```
const parts = value.split(/,\s*/)
this.last = parts[0]
this.first = parts[1]
}
}
```

Акцессор записи вызывается, когда производится присваивание fullName:

```
harry.fullName = 'Smith, Harold'
```

4.3. Наследование

Ключевой концепцией объектно-ориентированного программирования является наследование. Класс определяет поведение своих экземпляров. Мы можем создать подкласс данного класса (который называется суперклассом), экземпляры которого будут вести себя в каком-то отношении по-другому, но остальное поведение унаследуют от суперкласса.

Стандартный учебный пример – иерархия наследования с суперклассом Employee и подклассом Manager. Ожидается, что работники получают за свою работу зарплату, а менеджеры, помимо основной зарплаты, получают бонусы, если достигают поставленных целей.

В JavaScript для выражения связи между классами Employee и Manager служит ключевое слово extends:

```
class Employee {
constructor(name, salary) { ... }
raiseSalary(percent) { ... }
...
}
class Manager extends Employee {
getSalary() { return this.salary + this.bonus }
...
}
```

В конструкторе подкласса обязательно вызывать конструктор суперкласса. Для этого используется синтаксис super(...). Внутри скобок поместите аргументы, которые нужно передать конструктору суперкласса.

```
class Manager extends Employee {
constructor(name, salary, bonus) {
super(name, salary) // вызывать конструктор суперкласса обязательно
this.bonus = bonus // потом допустим такой код
```

```
}
...
}
```

Ссылку this разрешается использовать только после вызова super. Однако если конструктор подкласса не предоставили, то он будет сгенерирован автоматически. И этот автоматический конструктор передает все аргументы конструктору суперкласса.

4.4. Модули

По мере роста приложения, код приложения делят на много файлов, так называемых «модулей». Модуль обычно содержит класс или библиотеку с функциями.

Модуль – это просто файл. Один скрипт – это один модуль. Модули могут загружать друг друга и обмениваться функциональностью, вызывать функции одного модуля из другого.

Export

Инструкция export используется для экспорта функций, объектов или примитивов из файла (или модуля).

Именованный экспорт:

```
export { myFunction }; // экспорт ранее объявленной функции
export const foo = Math.sqrt(2); // экспорт константы
```

Дефолтный экспорт (экспорт по умолчанию) (один на скрипт):

```
export default function() {} // или 'export default class {}'
// тут не ставится точка с запятой
```

Именованная форма более применима для экспорта нескольких величин. Во время импорта, можно будет использовать одно и то же имя, чтобы обратиться к соответствующему экспортируемому значению.

Экспорт по умолчанию (default), он может быть только один для каждого отдельного модуля (файла). Экспорт по умолчанию может представлять собой функцию, класс, объект или что-то другое. Это значение следует рассматривать как «основное», так как его будет проще всего импортировать.

Использование именованного экспорта

Мы могли бы использовать следующий код в модуле. Например, файл my-module.js может содержать функцию:

```
function cube(x) {
  return x * x * x;
```

```
}
const foo = Math.PI + Math.SQRT2;
export { cube, foo };
```

В результате, в другом скрипте можно будет использовать функцию cube( ) и константу foo.

Чтобы модуль что-нибудь отдал, можно использовать альтернативный синтаксис module.exports:

```
const x = 5;
const addX = function(value) {
  return value + x;
};
module.exports.x = x;
module.exports.addX = addX;
```

Есть еще такой способ экспортирования функций и констант из модуля:

```
var User = function(name, email) {
  this.name = name;
  this.email = email;
};
module.exports = User;
```

Разница между этими подходами не велика, но важна. Как видно, в данном случае мы экспортируем функцию напрямую:

```
module.exports.User = User;
//vs
module.exports = User;
```

В последующем при использовании (импорте) будут различия в синтаксисе.

Использование export default

Если мы хотим экспортировать единственное значение или иметь резервное значение (fallback) для данного модуля, мы можем использовать export default.

```
// модуль"my-module.js"
export default function cube(x) {
  return x * x * x;
}
```

Затем в другом скрипте можно импортировать это значение по

умолчанию.

Import vs. Require

В браузере выполнение модуля JavaScript зависит от дериктив import и export. Эти дериктивы соответственно загружают и экспортируют модули ES. Это стандартный и официальный способ повторного использования модулей в JavaScript, который изначально поддерживается большинством веб-браузеров.

NodeJS по умолчанию поддерживает формат модуля CommonJS, который загружает модули с помощью функции require() и экспортирует их с расширением module.exports.

Import

Инструкция import используется для импорта ссылок на значения, экспортированные из внешнего модуля. Импортированные модули находятся в строгом режиме независимо от того, объявлены они таковые или нет. Для работы инструкции во встроенных скриптах нужно прописать у тэга <script type="module">.

Пример использования импорта:

import { sayHi } from './sayHi.js';

Директива import загружает модуль по пути ./sayHi.js относительно текущего файла и записывает экспортированную функцию sayHi в соответствующую переменную.

Import в браузере

Браузер автоматически загрузит и запустит импортированный модуль (и те, которые он импортирует, если надо), а затем запустит скрипт.

Модули не работают локально. Только через HTTP(s). Если попытаться открыть веб-страницу локально, через протокол file://, можно обнаружить, что директивы import/export не работают. Для тестирования модулей используется локальный веб-сервер, например, static-server или возможности «живого сервера» вашего редактора, например, расширение Live Server для VS Code.

Import в NodeJS

Для того чтобы воспользоваться директивой import при выполнении js-файлов в NodeJS необходимо добавить в файл package.json атрибут и значение "type": "module".

JS-файлы содержащие описание экспортируемых модулей рекомендуется так же сохранять с расширением .mjs.

Require

Node.js использует модульную систему. То есть вся встроенная функциональность разбита на отдельные пакеты или модули. Модуль представляет блок кода, который может использоваться повторно в других модулях. При необходимости мы можем подключать нужные нам модули. Какие встроенные модули есть в node.js и какую функциональность они предоставляют, можно узнать из документации.

Для загрузки модулей применяется функция require( ), в которую передается название модуля. После получения модуля мы сможем использовать весь определенный в нем функционал. В отличие от встроенных модулей для подключения своих модулей надо передать в функцию require относительный путь с именем файла (расширение файла необязательно).

Пример использования модуля в другом файле:

```
// utils.mjs
export const getFullName = (firstname, lastName) => {
   return `my fullname is ${firstname} ${lastName}`;
};

// index.js
import { getFullName } from './utils.mjs';
console.log(getFullName('John', 'Doe')); // My fullname is John Doe
```

Локальные модули можно импортировать, используя относительный путь (например ./, ./foo, ./bar/baz, , ../foo), который будет разрешен относительно текущего рабочего каталога в котором расположен файл импортирующий модуль.

Если вы захотите определить структуру модулей, разместите модули в подкаталогах. Если, например, вы хотите хранить модуль currency в папке lib/, приведите строку require к следующему виду:

Дополнительные материалы

- Инструкция import – https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/import.
- ECMAScript модули – https://nodejs.org/docs/latest-v13.x/api/esm.html#esm_enabling.
- Документация NodeJS – https://nodejs.org/api/.
- Export Module in Node.js – Export Module in Node.js.

Задания для лабораторной работы

Задание 1. Объект Car

Create a car object, add a color property to it with the value equals 'black'.

Change the color property of the car object to 'green'.

Add the power property to the car object, which is a function and displays the engine power to the console.

Задание 2. Объект Warehouse

Pears and apples are accepted to the warehouse, write a function that returns the result of adding the number of accepted pears and apples.

Задание 3. Наследование

Create 2 objects: animal and cat, add move property to animal object, cat object must inherit move property from object animal.

Задание 4. Калькулятор

Создайте html-документ, содержащий JavaScript для реализации калькулятора. Подробное описание задания смотрите на веб-странице https://github.com/sergei-tsarik/polessu-js.

Управляемая самостоятельная работа

Дополнить конспект лекций самостоятельно изученными вопросами:

1. JSON объекты.
2. XML объекты.

5. Модульное тестирование

Общие сведения из теории тестирования. Тестирование функций, методов и классов. Обработка полученной информации из файлов.

5.1. Общие идеи

Обычно, когда мы пишем функцию, мы легко можем представить, что она должна делать, и как она будет вести себя в зависимости от переданных параметров. Во время разработки мы можем проверить правильность работы функции, просто вызвав ее, например, из консоли и сравнив полученный результат с ожидаемым значением. Если функция работает не так, как мы ожидаем, то можно внести исправления в код и запустить ее еще раз. Так можно повторять до тех пор, пока функция не станет работать так, как нам нужно.

Автоматизированные тесты – программы, которые автоматизируют процесс тестирования ПО. Это значит, что тесты пишутся отдельно, в дополнение к коду. Они по-разному запускают наши функции и затем сравнивают полученный результат с ожидаемым выводом, определенным заранее.

Такая разновидность автоматизированного тестирования, когда пишется логика для тестирования отдельных частей приложения, называется модульное тестирование.

Благодаря тестам вы сможете более критично оценить архитектуру приложения и избежать ловушек на ранних этапах разработки. Тесты также придадут уверенность в том, что последние изменения не привели к появлению ошибок. Хотя написание модульных тестов требует определенного времени, можно сэкономить время в дальнейшем, поскольку не придется вручную проверять приложение после внесения изменений.

Для запуска тестов используем тестовый Фреймворк, например, Mocha речь о котором пойдет чуть позже. Пока функция не готова, будут ошибки. Пишется код реализации функции до тех пор, пока все не начнет работать так, как нужно.

5.2. Терминология

Тестовый сценарий (test case), или просто тест, – это набор специализированных команд для автоматизации тестирования определенной части программы. Тестовый набор (test suite) – это набор тестовых сценариев, объединенных по некоторому принципу и предназначенных для проверки определенной части программы.

Таким образом, лучше всего сгруппировать тесты для определенного модуля в один тестовый набор. Набор может содержать множество тестов, каждый из которых тестирует какой-то определенный аспект модуля.

Другая важная часть модульного тестирования – утверждения. Утверждения – это функции, проверяющие ожидаемые значения с полученными.

В большинстве случаев модульное тестирование выполняется с помощью модуля assert. При этом проводится проверка некоего условия, и, если условие не выполняется, генерируется ошибка. Модуль assert используется многими сторонними фреймворками тестирования.

Примеры библиотек для тестирования.

Mocha – основной фреймворк. Он предоставляет общие функции тестирования, такие как describe и it, а также функцию запуска тестов.

Chai – библиотека, предоставляющая множество функций проверки утверждений. Пока мы будем использовать только assert.equal.

5.3. Mocha

MochaJS – это JavaScript фреймворк, используемый для автоматического тестирования приложений. Он может использоваться как на стороне сервера Javascript, так и в браузере.

Подробнее о фреймворке можно узнать на официальной странице Mochajs. В данном же случае мы рассмотрим некоторые базовые стороны работы с ним.

Определим в папке проекта новый файл package.json со следующим содержимым:

```
{
  "name": "testapp",
  "version": "1.0.0"
}
```

Далее добавим в проект пакет mocha с помощью следующей команды:

```
npm install mocha --save-dev
```

Так как фреймворк Mocha необходим только для тестирования приложения, то он добавляется в файле package.json в секцию devDependencies с помощью команды --save-dev.

```
{
  "name": "testapp",
  "version": "1.0.0",
  "devDependencies": {
    "mocha": "^3.2.0"
  }
}
```

По умолчанию, Mocha ищет каталог с именем tests хотя вы можете указать путь к файлу в качестве аргумента команды во время запуска команды mocha. Чтобы все было организовано, рекомендую создать отдельный каталог с именем tests для всех тестовых скриптов.

Тестирование с использованием Mocha, как правило, использует следующие шаблоны:

```
describe([String with Test Group Name], function() {
   it([String with Test Name], function() {
      [Test Code]
   });
});
```

Функция describe( ) используется для группировки аналогичных тестов. Для Mocha не требуется запускать тесты, но их группировка упростит поддержку нашего кода теста. Рекомендуется группировать тесты таким образом, чтобы было проще обновлять аналогичные вместе.

Функция it( ) содержит наш код теста. Именно здесь мы могли бы взаимодействовать с функциями нашего модуля и использовать библиотеку assert. Многие функции it( ) могут быть определены в функции describe( ).

Функция describe( ) используется для описания того, что необходимо проверить. Функция принимает два параметра: String и callback.

```
describe('Reverse String Test', () => {
   it('Checks if the strings is reversed', () => {
      // Code
   });
});
```

Функция it( )используется для описания того, что будет протестировано в этом блоке кода. Разрешено писать вложенные описания describe( ) и it( ).

Хорошо протестированное приложение – это, как правило, хорошо документированное приложение, и тесты иногда могут быть эффективным способом документирования.

5.4. Chai

ChaiJS – это библиотека для node и, как Mocha, Chai может использоваться на стороне сервера или в браузере. Chai может быть использован совместно с любой библиотекой для тестирования.

Для установки Chai используются команды CMD:

npm install chai --save

Библиотека ChaiJS включает в себя три интерфейса: should, expect и assert.

Функции вида assert.* используются для проверки того, что тестируемая функция работает так, как мы ожидаем. Пример часто используемой функции сравнения assert.equal. Функция сравнивает переданные значения и выбрасывает ошибку, если они не равны друг другу.

Библиотека Chai содержит множество других подобных функций, например:

assert.equal(value1, value2) – проверяет равенство value1 == value2.

assert.strictEqual(value1, value2) – проверяет строгое равенство value1 === value2.

assert.notEqual, assert.notStrictEqual – проверяет неравенство и строгое неравенство соответственно.

assert.isTrue(value) – проверяет, что value === true

assert.isFalse(value) – проверяет, что value === false

С полным списком можно ознакомиться в документации на официальном веб-сайте.

Дополнительные материалы

- Автоматическое тестирование с использованием фреймворка Mocha – https://learn.javascript.ru/testing-mocha.

- Mocha – https://nodejsdev.ru/guides/metanit/mocha/.

- Chai Assertion Library – https://www.chaijs.com/.

Задания для лабораторной работы

Задание. Тестирование с использованием Mocha и Assert

Создайте js-класс Todos и выполните его тестирование. Подробное описание задания смотрите на веб-странице https://github.com/sergei-tsarik/polessu-js.

6. Применение скриптовых языков для автоматизации тестирования веб-приложений

Библиотеки и фреймворки для автоматизации действий веб-браузера. Создание и работа с экземпляром веб-браузера. Поиск и взаимодействие с веб-элементами веб-приложения. Выполнение проверок состояний и содержания веб-элементов.

6.1. Selenium WebdriverIO

Selenium – популярная библиотека автоматизации браузера на базе Java. Использовав драйвер для конкретного языка, вы сможете подключиться к серверу Selenium и провести тесты для реального браузера.

Работать с Selenium сложнее, чем с простыми тестовыми библиотеками Node, потому что вам придется установить Java и загрузить JAR-файл Selenium. Загрузите Java для своей операционной системы, зайдите на сайт загрузки Selenium (http://docs.seleniumhq.org/download/) и загрузите JAR-файл. После этого сервер Selenium можно будет запустить командой следующего вида:

Java -jar selenium-server-standalone-2.53.0.jar

В вашем случае точная версия Selenium может быть другой. Возможно, вам также придется указать путь к двоичному файлу браузера. Например, в Windows 10 с браузером Chrome, заданным в свойстве browserName, полный

путь для Chrome может быть указан следующим образом:

java -jar -Dwebdriver.chrome.driver="C:\path\to\chrome.exe" selenium-server-standalone-3.0.1.jar

Точный путь зависит от конкретного способа установки Chrome на вашей машине. За дополнительной информацией о драйвере Chrome обращайтесь к документации SeleniumHQ.

Установка WebDriverIO

В файл package.json нужно установить пакеты npm для WebdriverIO. Необходимые пакеты для WebdriverIO в реестре NPM можно найти официальном веб-сайте npmjs.com.

Последовательность команд в CMD для создания нового проекта Node и установки WebdriverIO:

mkdir -p selenium/test/specs

cd selenium

npm init -y

npm install --save-dev webdriverio

npm install --save express

В комплект поставки WebdriverIO включен удобный генератор конфигурационных файлов. Чтобы запустить его, выполните команду CMD:

npm install wdio

Ответьте на вопросы и подтвердите значения по умолчанию:

? Where do you want to execute your tests? On my local machine

? Which framework do you want to use? mocha

? Shall I install the framework adapter for you? Yes

? Where are your test specs located? ./test/specs/**/*.js

? Which reporter do you want to use?

? Do you want to add a service to your test setup?

? Level of logging verbosity verbose

? In which directory should screenshots gets saved if a command fails? ./errorShots/

? What is the base url? http://localhost:4000

Дополните файл package.json командой wdio, чтобы тесты можно было запускать командой npm test:

"scripts": {

"test": "wdio wdio.conf.js"

},

Для того чтобы сымитировать локально работу сервера в сети Интернет и выполнить его тестирование, воспользуемся базовым сервером Express и выведем HTML-документ (листинг).

Листинг файл index.js

```
const express = require('express');
const app = express();
const port = process.env.PORT || 4000;

app.get('/', (req, res) => {
    res.send(`
<html>
    <head>
        <title>My to-do list</title>
    </head>
    <body>
        <h1>Welcome to my awesome to-do list</h1>
    </body>
</html>
    `);
});

app.listen(port, () => {
    console.log('Running on port', port);
});
```

Запустите файл index.js с помощью node выполнив в терминале команду:

node index.js

В результате в окне браузера по адресу localhost:4000 отобразиться HTML-документ с текстом «Welcome to my awersome to-do list».

Далее напишем тестовый класс с тестовым методом. В листинге ниже (файл test/specs/todo-test.js) приведен простой тест, который создает клиента WebdriverIO, а затем проверяет заголовок на странице.

Листинг. Файл todo-test.js

```
describe("Test suite", () => {
    it("First test", async () => {
        await browser.url("http://localhost:4000/");
        const pageTitle = await browser.getTitle();
        expect(pageTitle).toEqual("My to-do list");
    });
});
```

Для запуска тестов используйте команду CMD:

npm run wdio

В результате выполнения теста будет открыт реальный браузер и в адресной строке будет выполнено обращение к адресу http://localhost:4000/. Поскольку был запущен Express сервер, то в окне браузера по адресу

http://localhost:4000/ отобразиться HTML-документ.

Далее в тесте запрашивается текущее состояние документа в браузере с помощью метода getTitle( ) получаем заголовок HTML-документа из тега <title>.

В завершение теста выполняем сравнение полученного текстового значение заголовка HTML-документа с ожидаемым значением “My to-do list”.

6.2. XPath

Когда мы переходим на веб-страницу, нам нужно взаимодействовать с веб-элементами, доступными на странице, например, щелкнуть ссылку/кнопку, ввести текст в поле редактирования и т. д.

Таким образом, для взаимодействия с веб-элементом необходимо его идентифицировать.

Для идентификации веб-элемента можно создать XPath для веб-элемента.

XPath (XML Path Language) – язык запросов к элементам XML-документа. Разработан для организации доступа к частям документа XML в файлах трансформации XSLT и является стандартом консорциума W3C. XPath призван реализовать навигацию по DOM в XML.

XML имеет древовидную структуру. В самостоятельном XML-документе всегда имеется один корневой элемент, в котором допустим ряд вложенных элементов, некоторые из которых тоже могут содержать вложенные элементы. Так же могут встречаться текстовые узлы, комментарии и инструкции. Можно считать, что XML-элемент содержит массив вложенных в него элементов и массив атрибутов.

У элементов дерева бывают элементы-предки и элементы-потомки. Каждый элемент дерева находится на определенном уровне вложенности. Элементы упорядочены в порядке расположения в тексте XML, и поэтому можно говорить об их предыдущих и следующих элементах. Это очень похоже на организацию каталогов в файловой системе.

Строка XPath описывает способ выбора нужных элементов из массива элементов, которые могут содержать вложенные элементы. Начинается отбор с переданного множества элементов, на каждом шаге пути отбираются элементы, соответствующие выражению шага, и в результате оказывается отобрано подмножество элементов, соответствующих данному пути.

Для примера возьмем следующий html-документ отображаемый при запуске Express сервера (листинг index.js):

```
<html>
    <head>
        <title>My to-do list</title>
    </head>
    <body>
        <h1>Welcome to my awesome to-do list</h1>
    </body>
</html>
```

XPath-путь /html/body/h1 будет соответствовать веб-элементу <h1>Welcome to my awersome to-do list</h1> в данном html-документе.

Элементы пути преимущественно пишутся в XPath в краткой форме. В нашем случае тег <h1> является единственным тегом в представленном HTML-документе, поэтому краткая запись XPath будет выглядеть в следующем виде //h1.

XPath к любому веб-элементу html -страницы открытой в браузере можно получить, например, в браузере Chrome с помощью DevTools. Chrome DevTools – это набор инструментов для веб-разработчиков, встроенных непосредственно в браузер Google Chrome.

Вкладка Elements (Элементы) и инструмент Select an element in the page to inspect it (Проверить) позволяют выбирать веб-элементы непосредственно в окне браузера и отображают положение веб-элемента в DOM структуре документа. Использование контекстного меню на веб-элементе в DOM структуре HTML-документа дает возможность получить XPath к веб-элементу с помощью команды Copy > Copy XPath или Copy > Copy full XPath.

6.3. Тестирование веб-элемента

Протокол WebDriver предоставляет несколько стратегий выбора для запроса элемента. WebdriverIO упрощает их, чтобы упростить выбор элементов. Для запроса веб-элементов расположенных в DOM структуре HTML-документа используются команды $ и $$.

В листинге (файл test/specs/todo-test.js) приведен еще один тест, который создает клиента WebdriverIO, а затем проверяет текст веб-элемента <h1>.

Листинг. Файл todo-test.js и тестовый метод “Second test”

```
it("Second test", async () => {
    await browser.url("http://localhost:4000/");
    const h1 = await $("//h1");
    expect(h1).toHaveText("Welcome to my awesome to-do list");
});
```

Для запуска тестов используйте команду:

npm run wdio

В результате выполнения теста будет открыт реальный браузер и в

адресной строке будет выполнено обращение к адресу http://localhost:4000/. Поскольку был запущен Express сервер, то в окне браузера по адресу http://localhost:4000/ отобразиться HTML-документ.

Далее в тесте командой $(“//h1”) запрашивается веб-элемент имеющий XPath “//h1”.

В завершение теста выполняется проверка того, что между открывающим тэгом <h1> и закрывающим тэгом </h1> содержится текст "Welcome to my awersome to-do list”.

6.4. WebdriverIO основные команды

Веб-элемент представляет собой, например, узел DOM структуры HTML-документа в окне браузера. Веб-элемент можно получить с помощью одной из множества команд запроса элемента, например $, custom$, react$или shadow$.

Метод $

Метод $ позволяет получить один веб-элемент HTML-документа.

Поддерживаются следующие типы селекторов:

- CSS Query Selector
- Link Text
- Partial Link Text
- Element with certain text
- Tag Name
- Name Attribute
- xPath
- ID
- и прочие.

Если не указано иное, WebdriverIO будет запрашивать веб-элементы, используя шаблон селектора CSS , например:

```
const elem = await $('nav a.navbar__brand');
```

Чтобы получить веб-элемен описанный тегом <a> и содержащий текст, запрос на поиск веб-элемента в DOM структуре HTML-документа следует начинать со знака “=”. Например, если в HTML-документе есть тег <a> содержащий текст:

```
<a href="https://webdriver.io">WebdriverIO</a>
```

Для получения доступа к данному веб-элементу можно применить следующий запрос:

```
const link = await $('=WebdriverIO');
```

Подробнее про стратегии поиска веб-элементов в DOM структуре HTML-документа читайте на сайте https://webdriver.io/docs/selectors.

Метод click( )

Метод click( ) для выбранного веб-элемента сопровождается обычно пролистыванием в окне браузера содержимого страницы до выбранного элемента, а затем щелкает на веб-элементе.

Метод setValue( )

Метод setValue( ) выполняет заполнение поля ввода строковым литералом с предварительной очисткой поля ввода.

Метод getText( )

Метод getText( ) возвращает текст содержащийся в веб-элементе.

Дополнительные материалы

- Автоматизация тестирования на Node.JS – https://jazzteam.ru/technical-articles/test-automation-on-node-js/.
- Веб-элементы и методы работы с ними. The Element Object – https://webdriver.io/docs/api/element/.
- Стандарты XPath – https://www.w3.org/TR/xpath/.
- Chrome DevTools – https://developer.chrome.com/docs/devtools/.
- Фреймворк WebdriverIO – https://www.npmjs.com/package/webdriverio.
- Документация Chrome Driver – https://github.com/SeleniumHQ/selenium/wiki/ChromeDriver.
- Официальный сайт фреймворка WebdriverIO – http://webdriver.io/.
- Официальный веб-сайт Selenium – https://www.selenium.dev/downloads/.

Задания для лабораторной работы

Задание 1. Калькулятор веса

1. Откройте страницу сайта https://svyatoslav.biz/testlab/wt/index.php.
2. Введите в поле Имя значение «Студент».
3. Введите в поле Рост значение 180.
4. Введите в поле Вес значение 80.
5. Выберите Пол значение «М».
6. Кликните на кнопке Рассчитать.
7. Проверьте, что результатом расчета является текстовое сообщение «Идеальная масса тела».

Задание 2. Automation tools – WebdriverIO

Установите WebdriverIO и создайте тест для автоматизации действий на

веб-сайте. Пример описания задания приведен на веб-странице https://github.com/sergei-tsarik/polessu-js.

Итоговый проект. Test Automation Framework

Итоговый проект должен содержать WebdriverIO для организации взаимодействия js-кода с окном браузера Chrome и элементами тестируемого веб-сайта. В итоговом проекте необходимо использовать Page Object / Page Factory для описания веб-элементов тестируемых веб-страниц веб-сайта.

Пример пошаговых инструкций для тестирования веб-сайта смотрите в Интернете по адресу https://github.com/sergei-tsarik/polessu-js.