Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет университетский технологический колледж
Кафедра информационных систем в экономике

Отчет защищен с оценкой_____
Преподаватель_____ *С. В. Умбетов*
«_____» _____2025 г.

Отчёт по лабораторной работе №1
по дисциплине «Производственная практика»
«Шифрование ROT13»

ЛР 09.02.07.21.000

Студент группы 1ИСП-21 Д.В.Соколов
группа и.о.,фамилия

Преподаватель *ассистент,* к. *т. н.* С.В.Умбетов
должность, ученая степень и.о.,фамилия

БАРНАУЛ 2025

Лабораторная работа №1

Шифрование ROT13

Цели и задачи работы: Вспомнить основы javascript, html и css.

Задания к работе:

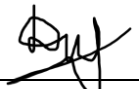
Необходимо создать html страницу и js код, страница должна быть валидной стандарту HTML5.

В коде реализовать шифрование ROT13 для четного варианта и расшифровку для нечетного

ROT13 — это простой шифр замены букв, который заменяет букву буквой, находящейся через 13 букв после неё в алфавите. Если в строку включены цифры или специальные символы, их следует вернуть в исходном виде. Смещать следует только буквы латинского/английского алфавита.

Использовать любые строковые операции например (replace, charCodeAt, fromCharCode, join, split) запрещено. Алфавит должен быть помещен в массив. Вывод алфавита и операций шифрования\расшифровки вместе с овтет должен быт на странице.

Дизайн, слева тексе до шифрования, справа зашифрованный текст

Задание принял:  Соколов Д.В.

Подпись ФИО

Ход работы.

Чтобы выполнить задание создадим локальный и удаленный Git-репозитории, а также необходимые файлы. Сделаем первый коммит.

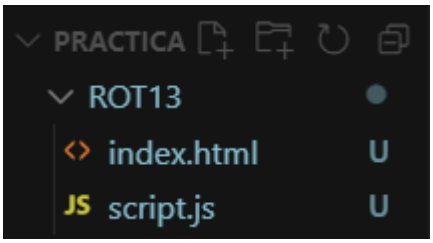


Рисунок 1 – Создали файлы.

```
PS C:\Users\diman\Desktop\practica> git add .
PS C:\Users\diman\Desktop\practica> git commit -m "first commit"
[master (root-commit) 8a53c76] first commit
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ROT13/index.html
 create mode 100644 ROT13/script.js
PS C:\Users\diman\Desktop\practica> git branch -M main
PS C:\Users\diman\Desktop\practica> git remote add origin https://github.com/dimasoko/lov_practika.git
PS C:\Users\diman\Desktop\practica> git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 271 bytes | 67.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/dimasoko/lov_practika.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\diman\Desktop\practica>
```

Рисунок 1 – Закоммитили.

Добавим файл со стилями для соответствия заданному преподавателем дизайну и напишем html-код(рис.3), который в последствии не нужно будет менять.

ROT13 > index.html > html

```
<!DOCTYPE html>
<html>
    <head>
        <title>ROT13</title>
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
        <div class="container">
            <div class="header-controls">
                <h1>ROT13 - расшифровка - вариант 21</h1>
            </div>
            <div class="text-areas">
                <textarea id="inputText" placeholder="Введите текст"></textarea>
                <textarea id="outputText" placeholder="Результат" readonly></textarea>
            </div>
        </div>
        <script src="script.js"></script>
    </body>
</html>
```

Рисунок 3 – index.html

Здесь всё, что необходимо: заголовок и два textarea: один для ввода текста, второй с выводом, доступный только для чтения.

Чтобы написать script.js мы полезли на сайт самого rot13.com. В панели разработчика нам удалось найти лишь код, который не соответствует нашему заданию: использует fromCharCode и charCodeAt. После ознакомлением с данным веб-ресурсом было обнаружено, что существуют разные ROT'ы: от 1 до 25. Взяли на заметку что вывод обрабатывается после любого изменения поля ввода.

Также путем тяжелой умственной работы обнаружено, что ROT13 – симметричный шифр, то есть его шифровка и расшифровка проходит одинаково, а смысл наполняет лишь человек, шифрующий или расшифрующий это всё. Получается, задание преподавателя «Реализовать шифровку» или «Реализовать расшифровку» имеет сугубо декоративный характер, что мы уже отчасти реализовали в index.html.

Рисунок 4 – script на rot13.com.

Далее прошла еще одна тяжелая умственная работа: мы поняли как должен работать код поэтапно:

При любом изменении поля inputText запускается работа скрипта, потом данная строка передается в функцию, которая последовательно обрабатывает каждый символ строки. Для каждого символа определяется, принадлежит ли он к латинскому алфавиту (в строчном или прописном регистре) посредством вызовов другой функции и, далее, если символ является буквой, его код смещается на 13 позиций в алфавитном порядке с учетом цикличности. Символы, не входящие в латинский

алфавит, остаются без изменений что реализуем в третьей функции. Параллельно с преобразованием символов формируется детализированный лог, включающий отображение используемого алфавита и запись о трансформации каждого релевантного символа или о его сохранении без изменений. Итоговый преобразованный текст с логом операций динамически отображается в поле outputText.

Напишем некий код script.js

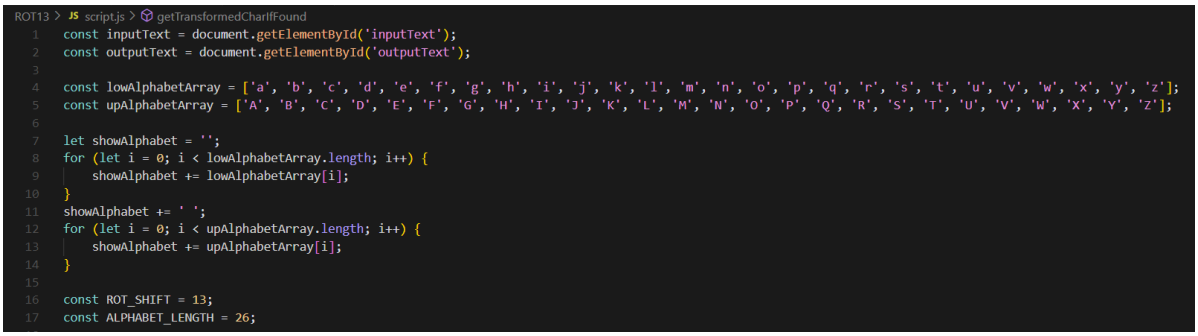


Рисунок 5 – Первая часть кода.

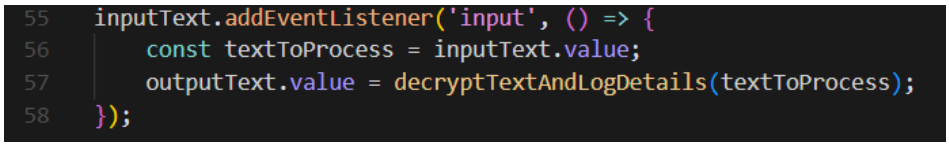
В первой части кода мы берем значения из html, создаем массивы-алфавиты(маленькие и заглавные буквы), создаем из этих массивов строки для вывода (согласно заданию) путем цикла for, который перебирает каждый элемент массива, формируя строку. Далее задаем базовые настройки: длина алфавита равна 26, сдвиг алфавиту по заданию – 13.

Далее – создаем матрешку функций согласно результату нашего брейнштorма: по ходу кода – от внутренней куколки к внешней.

```
function getTransformedCharIfFound(charToTransform, alphabetArray) {
    for (let i = 0; i < alphabetArray.length; i++) {
        if (charToTransform === alphabetArray[i]) {
            const newIndex = (i + ROT_SHIFT) % ALPHABET_LENGTH; // а расшифровка и шифровка то одинаковые!
            const transformed = alphabetArray[newIndex];
            const logEntry = `${charToTransform} - 13 = ${transformed}`;
            return { char: transformed, log: logEntry };
        }
    }
    return null;
}

function transformCharWithROT13(char) {
    const result = getTransformedCharIfFound(char, lowAlphabetArray) ||
                   getTransformedCharIfFound(char, upAlphabetArray);

    return result ?
        { char: result.char, log: result.log } :
        { char: char, log: `'${char}' (не расшифровывается)` };
}

function decryptTextAndLogDetails(text) {
    let resultText = '';
    let logDetails = `\nалфавит: ${showAlphabet}\nпроцесс:\n`;

    for (let i = 0; i < text.length; i++) {
        const originalChar = text[i];
        const decryption = transformCharWithROT13(originalChar);
        resultText += decryption.char;
        if (decryption.log) {
            logDetails += decryption.log + '\n';
        }
    }
    return resultText + logDetails;
}
```

Рисунок 6 – Вторая часть кода: функции.

Во второй части кода всё работает так, как описывалось: getTransformedCharIfFound перебирает каждый элемент массива алфавита (маленьких или заглавных), ищет совпадение для одной единственного символа. В первую функцию из transformCharWithROT13 передается сам символ и, если из первой функции возвращается null для символа, он в логе помечается как «не расшифровывается». Третья функция, decryptTextAndLogDetails, подготавливает по одному символу для второй функции (соответственно дальше – первой) и формирует шаблон для вывода.



```
inputText.addEventListener('input', () => {
    const textToProcess = inputText.value;
    outputText.value = decryptTextAndLogDetails(textToProcess);
});
```

Рисунок 7 – Последняя часть кода: обработчик

При каждом изменении содержимого данного поля ввода функция-обработчик получает текущее значение поля и вызывает decryptTextAndLogDetails для его обработки, после чего результат присваивается свойству value элемента outputText, обеспечивая немедленное отображение преобразованных данных.

Таже были написаны стили для html, но они не особо значительные, поэтому не будем даже вставлять картинку с ними.

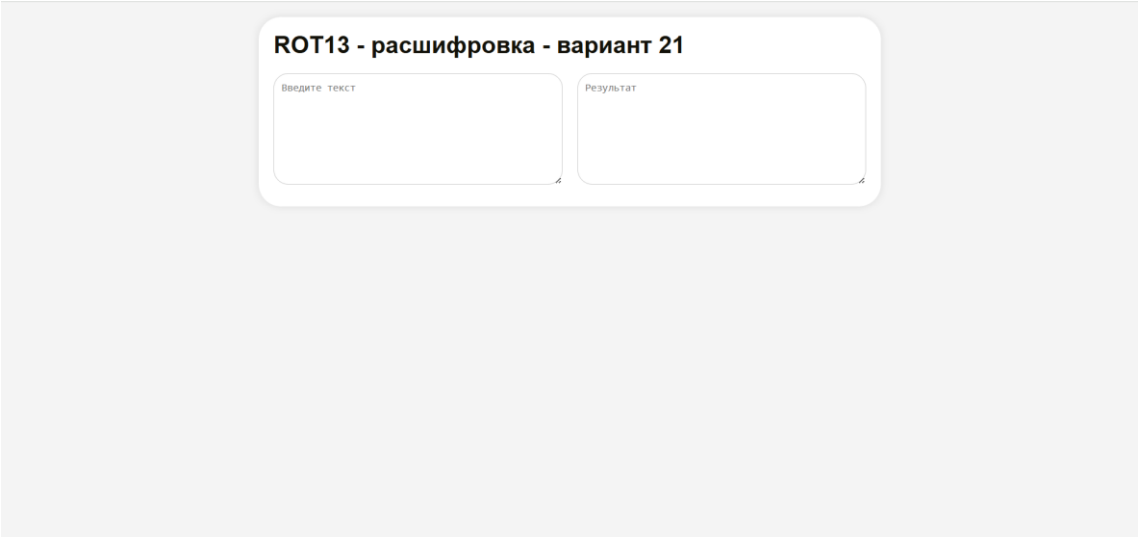


Рисунок 8 – Итоговая страница расшифровщика

Займемся проверкой нашего творения: Достоверные данные будем брать с rot13.com. Данные тестирования занесем в таблицу 1.

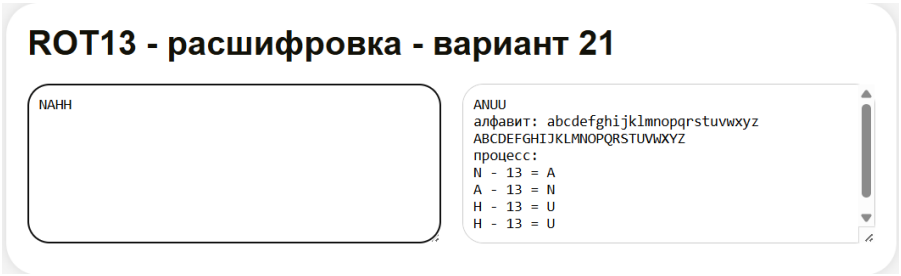


Рисунок 9 – Тест 1.

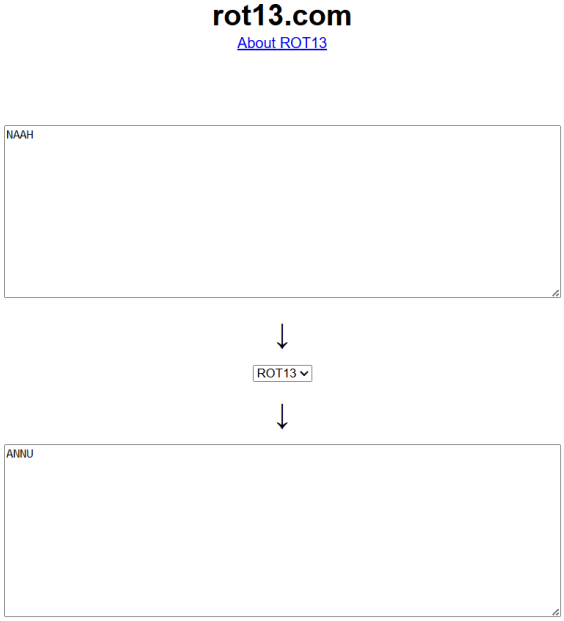


Рисунок 10 – Тест 1: rot13.com.

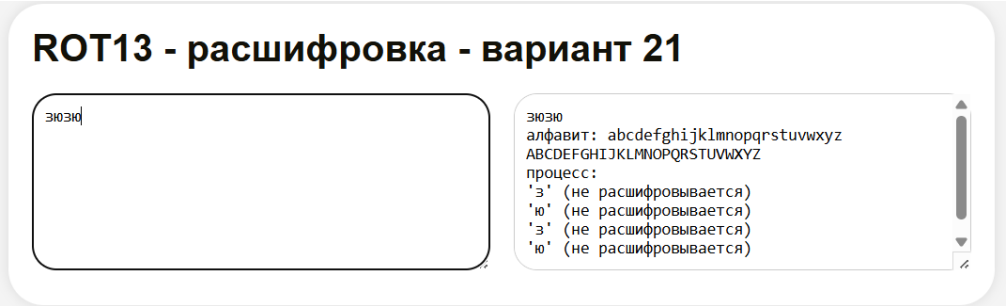


Рисунок 11 – Тест 2

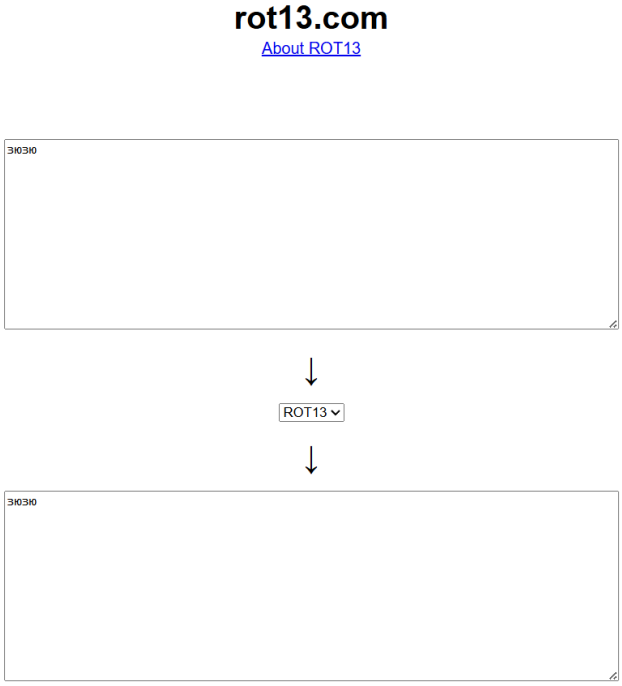


Рисунок 12 – Тест 2: rot13.com.

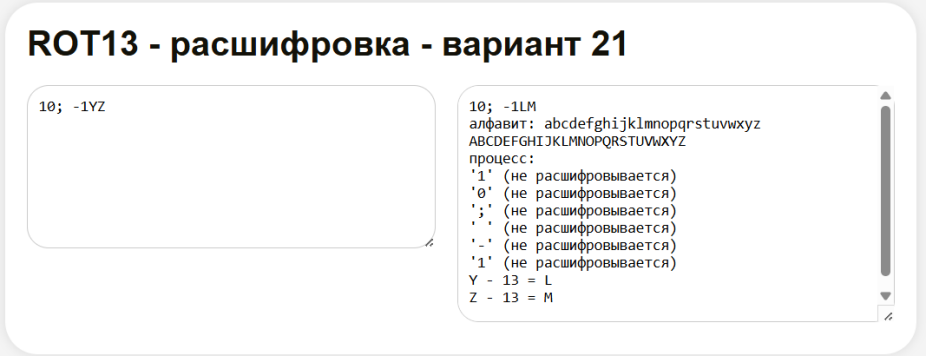


Рисунок 13 – Тест 3.

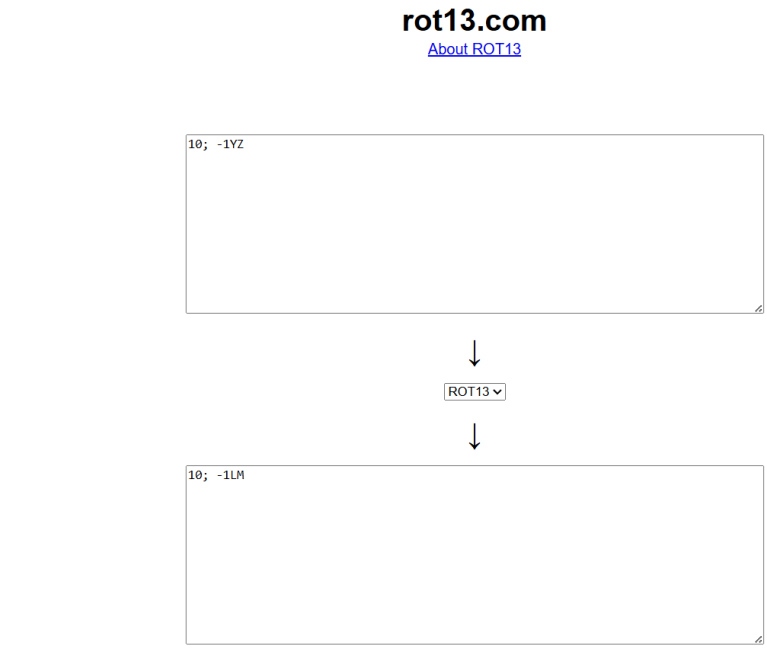


Рисунок 14 – Тест 3: rot13.com.

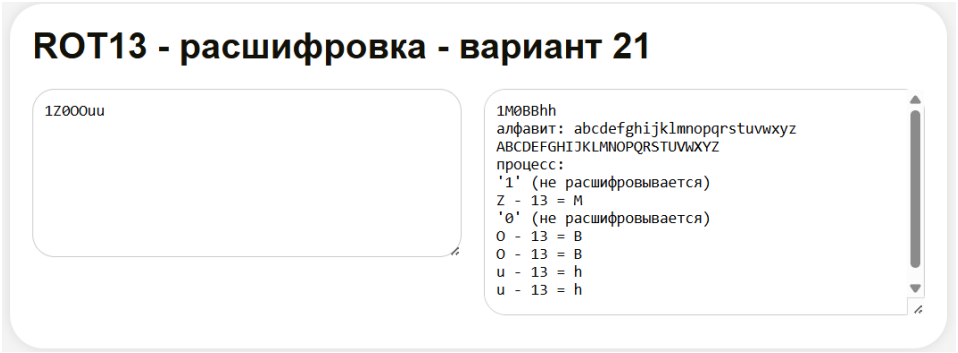


Рисунок 15 – Тест 4.

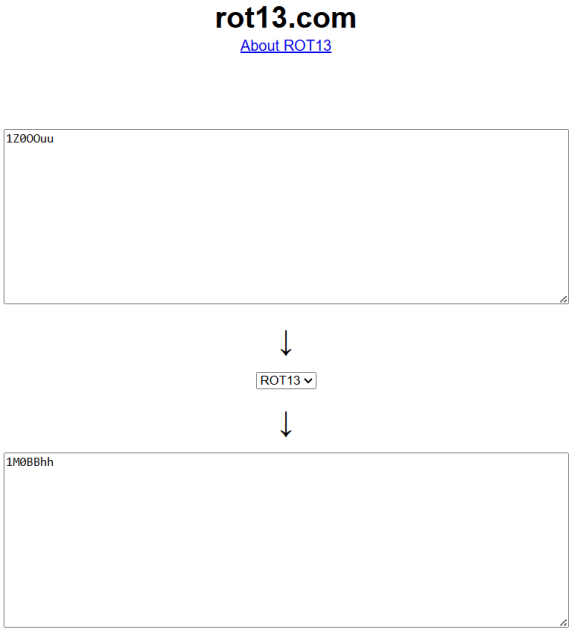


Рисунок 16 – Тест 4: rot13.com.

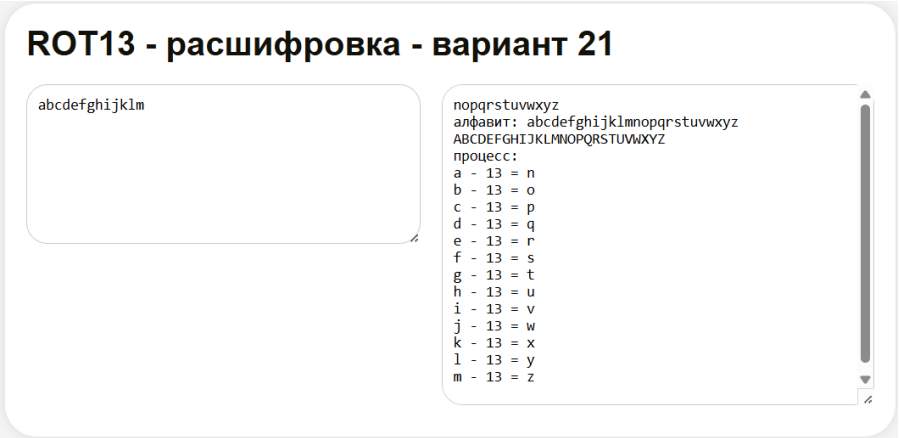


Рисунок 17 – Тест 5.

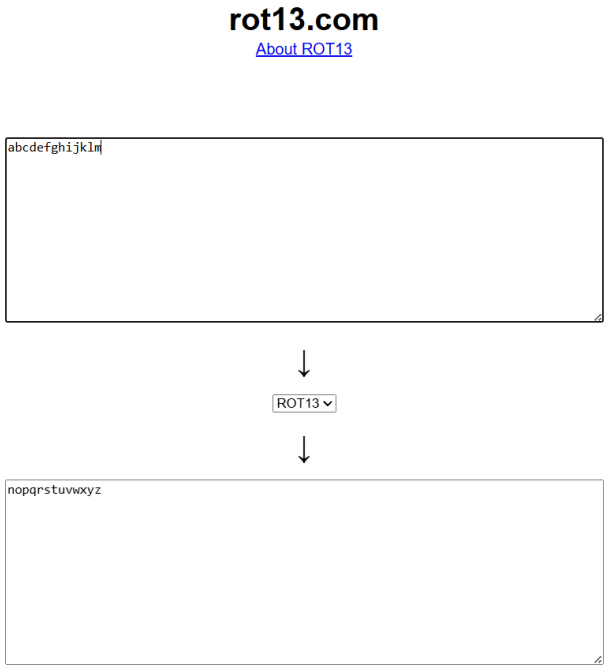


Рисунок 18 – Тест 5: rot13.com.

Таблица 1 – Тестирование программы для расшифровки ROT13.

Номер проверки	Вводные данные	Выходные данные	Достоверные данные
1	NAHH	ANUU	ANUU
2	зюзю	зюзю	зюзю
3	10; -1YZ	10; -1LM	10; -1LM
4	1Z0OOuu	1M0BBhh	1M0BBhh
5	abcdefghijklm	nopqrstuvwxyz	nopqrstuvwxyz

Программа работает корректно.

Версии отслеживается по адресу: https://github.com/dimasoko/lov_practika

Вывод

В ходе выполнения данной практической работы я посмеялся от отличий заданий по вариантам. Было нелегко построить в голове логику работы программы. Реализовывать эту логику было не то чтобы тяжело, это было интересно и бесяче, поскольку во-первых, я запутался что должно передаваться из функции в функци и во-вторых, в целом, вспоминать как писать на js было испытанием. В остальном было всё плавно и хорошо.

Все пункты задания, от дизайна до требований к написанному скрипту реализованы и работают корректно, что подтвердили тесты.