## Лабораторная работа № 10

Тема: Промисы.

Цель: Изучить принципы работы с Promise в JavaScript, освоить создание промисов с помощью конструктора и использование методов .then(), .catch(), .finally()..

### Вариант 1

**1. Создание простого промиса.**
Напишите функцию loadData, которая создает и возвращает промис.
Промис должен имитировать загрузку данных с задержкой в 3 секунды.
Если загрузка успешна, разрешите промис с объектом: { data: "Данные загружены" }.
Если загрузка не удалась, отклоните промис с ошибкой: new Error("Ошибка загрузки").

**2. Цепочка промисов**
Создайте второй промис, который будет вызываться после успешного завершения первого. Этот промис должен обрабатывать загруженные данные, изменяя их (например, добавляя к строке " — обработано").
Используйте методы .then() и .catch() для обработки результатов и обработки ошибок в каждом промисе.

**3. Обработка случайной ошибки.**
Модифицируйте первый промис так, чтобы он случайным образом завершался ошибкой (50% вероятности).
Если произошла ошибка, обработайте её с помощью метода .catch(), выводя сообщение об ошибке в консоль.

**4. Использование метода .finally().**
Добавьте к цепочке промисов метод .finally(), который будет выводить сообщение в консоль: "Операция завершена", независимо от того, была ли ошибка или всё прошло успешно.

### Вариант 2

**1. Создание промиса с параметрами.**
Напишите функцию simulateDelay, которая принимает два параметра: delay (в миллисекундах) и result. Функция должна возвращать промис, который разрешается с result после указанного времени задержки.

Если delay меньше 0, отклоните промис с ошибкой: new Error("Некорректная задержка").

**2. Цепочка обработки данных.**
Создайте промис, который возвращает строку, например, "Исходные данные". Затем создайте цепочку промисов:
Первый промис должен добавлять к строке префикс "Обработано: ".
Второй промис должен возвращать длину новой строки.
Обработайте результаты с помощью .then().

**3. Случайная ошибка.**
В первом промисе добавьте случайный генератор ошибки (50% вероятности). Если ошибка произошла, обработайте её с помощью метода .catch().
Выведите сообщение об ошибке в консоль с указанием, что операция завершилась неудачей.

**4. Финальное сообщение.**
В конце цепочки добавьте метод .finally(), который будет выводить сообщение о том, что все операции завершены, независимо от результата. Убедитесь, что это сообщение выводится после того, как все промисы завершены.

⚡ **Отчет должен содержать (см. образец):**
- номер и тему лабораторной работы;
- фамилию, номер группы студента и вариант задания;
- скриншоты окна VSC с исходным кодом программ;
- скриншоты с результатами выполнения программ;
- пояснения, если необходимо;
- выводы.

Отчеты в формате **pdf** отправлять на email:
**colledge20education23@gmail.com**

**Шпаргалка по Промисам в JavaScript**
Promise — объект, представляющий результат асинхронной операции, который может быть в одном из трёх состояний:
Pending (ожидание)
Fulfilled (выполнен)
Rejected (отклонён)

**Создание промиса**
Для создания промиса используется конструктор new Promise(executor), где executor — функция, принимающая два аргумента: resolve и reject.

```
const myPromise = new Promise((resolve, reject) => {
    // Имитация асинхронной операции
    const success = true; // измените на false для
тестирования ошибки
    if (success) {
        resolve("Успешно выполнено!");
    } else {
        reject(new Error("Произошла ошибка!"));
    }
});
```

**Методы промиса**
**.then()**
Используется для обработки успешного выполнения промиса.

```
myPromise
    .then(result => {
        console.log(result); // "Успешно выполнено!"
    });
```

**.catch()**
Используется для обработки ошибок, произошедших в промисе.

```
myPromise
    .then(result => {
        console.log(result); // "Успешно выполнено!"
    })
    .catch(error => {
        console.error(error);
    });
```

**.finally()**
Выполняется в любом случае, независимо от результата промиса.

```
myPromise
    .then(result => {
        console.log(result); // "Успешно выполнено!"
    })
    .catch(error => {
        console.error(error);
    })
    .finally(() => {
        console.log("Операция завершена.");
    });
```

**Пример цепочки промисов**
Вы можете создавать цепочки промисов, что позволяет обрабатывать результаты последовательно.

```
myPromise
    .then(result => {
        console.log(result); // "Успешно выполнено!"
    })
    .catch(error => {
        console.error(error);
    })
    .finally(() => {
        console.log("Операция завершена.");
    });
```

**Обработка случайной ошибки**
Для реализации случайной ошибки можно использовать Math.random().

```
myPromise
    .then(result => {
        console.log(result); // "Успешно выполнено!"
    })
    .catch(error => {
        console.error(error);
    })
    .finally(() => {
        console.log("Операция завершена.");
    });
```