Лабораторная работа № 11

Тема: Цепочки промисов.

Цель:

Вариант 1

1. Цепочка промисов (последовательные вычисления)
Создать промис, который возвращает число 5.
В первом .then() умножить число на 2.
Во втором прибавить 3.
В третьем вывести результат.

2. Цепочка с асинхронными задержками
Написать функцию delay(ms, value), которая возвращает промис с результатом value через ms мс.
Использовать её, чтобы последовательно вывести три сообщения: «Шаг 1», «Шаг 2», «Шаг 3».
3. Promise.all (параллельные операции)
Создать три промиса с разными задержками (1, 2 и 3 секунды).
Использовать Promise.all, чтобы дождаться их завершения и вывести массив результатов.

4. Ошибка в цепочке
Создать цепочку промисов, где на втором шаге вызывается throw new Error("Ошибка").
Добавить .catch(), чтобы вывести сообщение об ошибке и продолжить выполнение с запасным значением.

5. Promise.race
Создать два промиса:
первый завершится через 1 секунду с результатом "Быстрый",
второй через 3 секунды с результатом "Медленный".
Использовать Promise.race и вывести результат.

Вариант 2.

1. Цепочка промисов (передача результатов)
Создать промис, который возвращает строку "Hello".
В цепочке:
прибавить " World",
перевести строку в верхний регистр,
вывести результат.

2. Promise.allSettled

Создать три промиса:
один успешный,
второй с ошибкой,
третий успешный.
Использовать Promise.allSettled, чтобы получить массив статусов и вывести его в консоль.

3. Promise.any
Создать массив промисов, где два завершаются ошибкой, а один успешно.
Использовать Promise.any, чтобы вывести первый успешный результат.

4. Ошибка в Promise.all
Создать три промиса, где один завершится ошибкой.
Использовать Promise.all и показать, что весь результат завершается с reject.
Обработать ошибку через .catch().

5. Цепочка + обработка ошибок
Создать цепочку промисов:
первый возвращает число,
второй выбрасывает ошибку,
.catch() перехватывает ошибку и возвращает запасное число,
следующий .then() продолжает вычисления.

⚡ **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

1. Цепочки промисов

Каждый .then() возвращает новый промис.

Результат из предыдущего шага передаётся в следующий.

👉 Пример:

```
  Promise.resolve(1)
  .then(x => x + 1)   // 2
  .then(x => x * 3)   // 6
  .then(x => {
    console.log("Итог:", x);
  });
```

2. Асинхронные цепочки

Если .then() возвращает новый промис, цепочка ждёт его выполнения.

👉 Пример:

```
wait(500)
  .then(() => {
    console.log("Прошло 0.5 секунды");
    return wait(500);
  })
  .then(() => {
    console.log("Прошла ещё 0.5 секунды");
  });
```

3. Promise.all

Используется для одновременного запуска нескольких операций.

Возвращает массив результатов, если все успешны.

👉 Пример:

```
const p1 = Promise.resolve("Один");
const p2 = Promise.resolve("Два");

Promise.all([p1, p2])
  .then(values => {
    console.log("Результаты:", values);
    // ["Один", "Два"]
  });
```

4. Обработка ошибок
Ошибку можно перехватить с помощью .catch().
После .catch() цепочку можно продолжать.
👉 Пример:

```
Promise.resolve("Начало")
  .then(() => {
    throw new Error("Что-то пошло не так");
  })
  .catch(err => {
    console.error("Перехвачено:", err.message);
    return "Запасной результат";
  })
  .then(result => {
    console.log("После ошибки:", result);
  });
```

5. Promise.race
Возвращает результат первого завершившегося промиса (успех или ошибка).
👉 Пример:

```
const fast = new Promise(resolve => setTimeout(() =>
resolve("Быстро"), 300));
const slow = new Promise(resolve => setTimeout(() =>
resolve("Медленно"), 1000));

Promise.race([fast, slow]).then(console.log);
// "Быстро"
```

6. Promise.allSettled
Ждёт выполнения всех промисов (и успешных, и с ошибками).
Результат — массив объектов с status.

👉 Пример:

```
const ok = Promise.resolve(42);
const fail = Promise.reject("Ошибка");

Promise.allSettled([ok, fail]).then(results => {
  console.log(results);
  /*
  [
    { status: "fulfilled", value: 42 },
```

```
    { status: "rejected", reason: "Ошибка" }
  ]
  */
});
```

7. Promise.any
Ждёт первый успешный промис.
Ошибки игнорируются, пока не упадут все.

👉 Пример:

```
const fail1 = Promise.reject("Ошибка 1");
const success = Promise.resolve("Успех");
const fail2 = Promise.reject("Ошибка 2");

Promise.any([fail1, success, fail2])
  .then(result => console.log("Первый успех:", result))
  .catch(err => console.error("Все промисы упали:", err));
```

⚡
Цепочки: значения передаются из одного .then() в другой.
Promise.all: ждём все промисы → массив результатов.
.catch(): ловит ошибки в цепочке.
Promise.race: ждём первый завершившийся промис.
Promise.allSettled: ждём все промисы, получаем статусы.
Promise.any: ждём первый успех, ошибки игнорируем.