# Синхронное и асинхронное выполнение кода. Промисы и асинхронные функции в JS.

Известно, что JS-программа исполняется путем последовательного выполнения операций. То есть, следующая операция не выполнится, пока не будет выполнена текущая операция. Примитивные операции, такие как присваивание, инициализация переменных, итерирование по циклу, объявления функций и т. д. выполняются достаточно быстро, поскольку в первом приближении не зависят от системных вызовов операционной системы (степень загруженности системных вызовов, приоритет системных вызовов и т. д.), от загруженности сетевого трафика (при отправке запросов к серверу по сети), от драйверов периферических устройств (вывод на экран, взаимодействие с мультимедиа и т. д.), а выполняются в пределах выделенного программе процесса.

Но веб-приложения, обязательно обмениваются данными с удаленными веб-серверами, ответы от которых могут быть достаточно долгими. Ожидание ответа не должно блокировать выполнение программы после момента отправки запроса. Для этого стандарт ECMAScript приписывает для движка языка реализовать механизм асинхронной обработки вызовов. Подробнее о такой реализации через механизм цикла событий (Event loop) можно узнать из статьи [1] и из следующей темы «Асинхронность на уровне движка JS. Цикл событий»

В следующих статьях подробно рассказывается об асинхронности и методах работы с ней в JS. Рекомендуется внимательно разобраться в примерах, приведенных там: [2, 3, 4, 5]. Статьи [6] и [7] опциональны (для тех, кто хочет углубленно разобраться в промисах).

Также хотелось бы отметить, что благодаря внедрению промисов в стандарт ECMAScript 2015, мы можем теперь самостоятельно превратить некоторый синхронный код в асинхронный с помощью конструктора new Promise(executor). Например, если у нас есть некоторая функция, выполняющая сложные долгие вычисления, то пока она не посчитает, программа не будет продолжать выполнение синхронного кода дальше:

```
function getComputedValues() {
    let result = 0;

    // некоторый сложный алгоритм вычислений

    return result;
}

// … some synchronous code…

getComputedValues();

// ждем, пока не посчитает
// … some synchronous code…
```

В такой ситуации мы можем обернуть сложные вычисления в промис и повесить обработчик (коллбэк) на возвращенное промисом «обещание» (в данном примере это getComputedValues):

```
const getComputedValues = new Promise((resolve) => {
    let result = 0;

    // некоторый сложный алгоритм вычислений

    resolve(result);
})


// … some synchronous code…

getComputedValues.then((result) => {
    // что-то сделать с результатом result, когда он будет готов
})

// Программа выполняется дальше, так как промис вернул обещание
// и мы повесили на него обработчик.

// … some synchronous code…
```

Понимание этого пригодится при выполнении заданий этой темы.

Вопросы и задания

1. Приведите пример асинхронного обработчика события взаимодействия с DOM. Вы можете взять любой подходящий на выбор из Browser API из категории DOM API.

2. Как называется ситуация, изображенная на картинке? Предложите варианты, как можно избавиться от вложенности и улучшить читабельность кода.

```
asyncFunc('something', function (err, data) {
    asyncFunc('something', function (err, data) {
        asyncFunc('something', function (err, data) {
            asyncFunc('something', function (err, data) {
                asyncFunc('something', function (err, data) {
                    asyncFunc('something', function (err, data) {
                        asyncFunc('something', function (err, data) {
                            asyncFunc('something', function (err, data) {
                                asyncFunc('something', function (err, data) {
                                    // do the final action
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});
```

3. В CodePen по ссылке [8] представлен фрагмент кода, исполняемый при загрузке страницы приложения с некоторым тестом. Функция init() стартует загрузку данных теста. Если тест успешно загружен, то стартует загрузка заданий теста. Если они успешно загружены, то стартует загрузка рекомендаций на основе загруженных заданий. Если какой-то из запросов выдал ошибку, то дальнейшие вызовы запросов обрываются и выбрасывается ошибка текущего запроса в консоль. URL запросов здесь выдуманные. Как мы видим, вложенность коллбэков друг в друга в функции init() представляет собой ситуацию, представленную в задании №2.

   Предложите свой вариант оптимизации данного фрагмента кода с точки зрения улучшения читабельности без использования промисов. Пришлите свой ответ ссылкой на CodePen с изменением данного кода.

4. Перечислите преимущества промисов в JS? Приведите примеры использования.

5. Выполните задание №3, но с использованием промисов и их методов. Требования остаются теми же: каждый запрос отправляется только когда предыдущий запрос был успешен. Если какой-либо из запросов выдал ошибку, то в консоль должна упасть только эта ошибка.

6. Иногда в работе бывает необходимо вести разработку в отсутствии готового API запросов к серверу, либо при отсутствии соединения с Интернетом. Для этого можно временно создать имитацию запроса.

   Воспользуйтесь конструктором new Promise() для создания имитации асинхронной функции запроса к серверу. Пусть это будет функция fetchUsers(), имитирующая GET-запрос для получения списка пользователей users вида:

   users = [ {id: 1, name: “Alex”}, {id: 2, name: ‘Andry’}, … ].

   Задержки таймерами можно не делать. Главное, чтобы функция возвращала промис. Реализацию пришлите ссылкой на файл в CodePen.

## Ссылки


[1]. https://habr.com/ru/articles/651037/.

[2]. https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Async_JS/Introducing

[3]. https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Async_JS/Promises

[4]. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promise

[5]. http://callbackhell.ru/

[6]. https://pouchdb.com/2015/05/18/we-have-a-problem-with-promises.html

[7]. https://habr.com/ru/companies/ruvds/articles/358808/

[8]. https://codepen.io/diana_kondakova/pen/myeWRgv?editors=1111