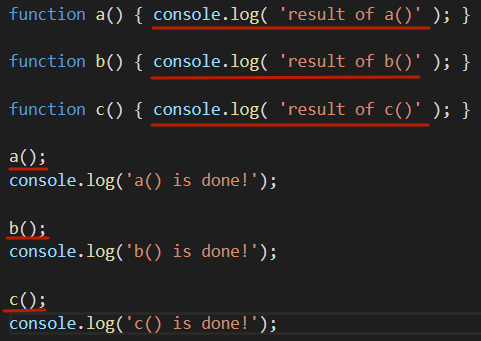
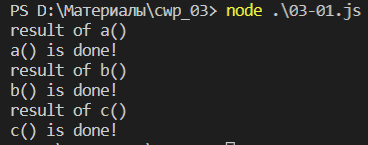
Лекция 03а

ПСКП, ПОИТ-3

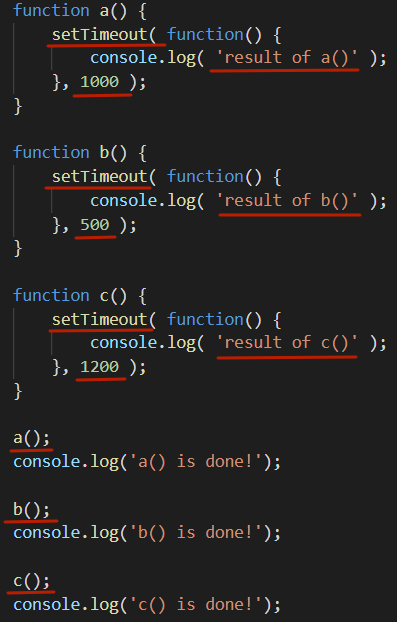
**СИНХРОННОЕ И АСИНХРОННОЕ ПРОГРАММИРОВАНИЕ**

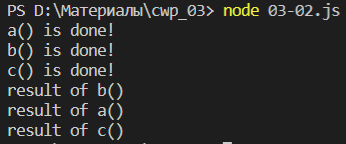
1. **JavaScript**: **синхронный однопоточный** язык программирования.
2. **Синхронное программирование:** последовательное выполнение инструкций с синхронными системными вызовами, которые ***полностью блокируют поток выполнения***, пока системная операция (чтение с диска) не завершится.
3. **Синхронное программирование:** последовательное выполнение





1. **Асинхронное программирование:** выполнение процесса в неблокирующем режиме системного вызова, что ***позволяет потоку программы продолжить обработку***.
2. **Понятие асинхронности**: операция называется асинхронной, если ее выполнение осуществляется в 2 фазы: 1) заявка на исполнение; 2) получение результата; при этом участвуют два механизма: A-механизм, формирующий заявку и потом получающий результат; B-механизм, получающий заявку от A, исполняющий операцию и отправляющий результат A; продолжительность исполнения операции B-механизмом, как правило, непредсказуемо; в то время пока B-механизм исполняет операцию, А-**механизм выполняет** собственную работу.
3. **Асинхронное программирование:** **web API** – это API, которые расширяют функциональность JavaScript для выполнения асинхронных задач (DOM, AJAX, setTimeout и т.д.) + **среда неблокирующего ввода-вывода**.
4. **Асинхронное программирование:** функция **setTimeout(callback, delay) –** глобальная функция, котораяпринимает функцию обратного вызова и временно сохраняет ее. По истечению времени, заданного в миллисекундах, функция обратного вызова помещается в **очередь коллбэков (callback queue)**. Затем event loop перемещает эту функцию в **стек вызовов (call stack)**, когда стек пуст. После этого осуществляется выполнение функции обратного вызова. В основном так работают все web API.





***Как сделать так, чтобы результат был таким?***

result of b()

b() is done!

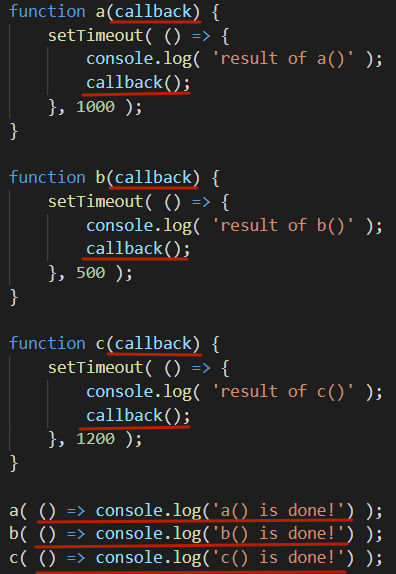
result of a()

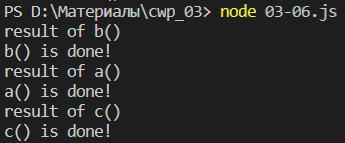
a() is done!

result of c()

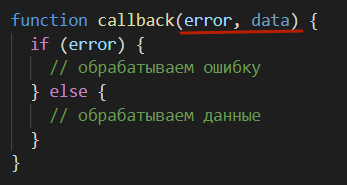
c() is done!

1. **Callback:** callback-функция (функция обратного вызова) — функция, которая передается в качестве параметра другой функции и которая **будет вызвана** асинхронно обработчиком событий **после завершения задачи**.
2. **Callback:**

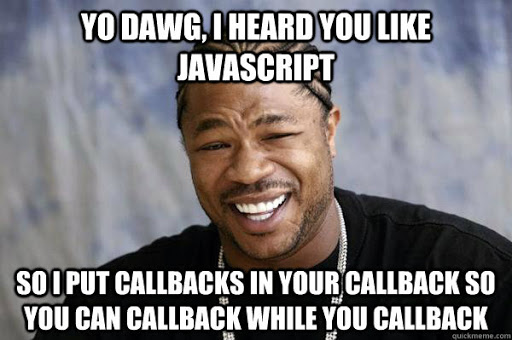
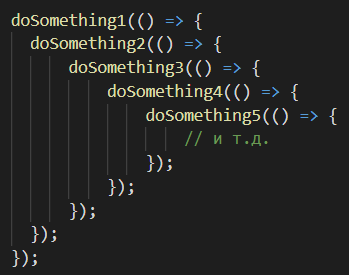




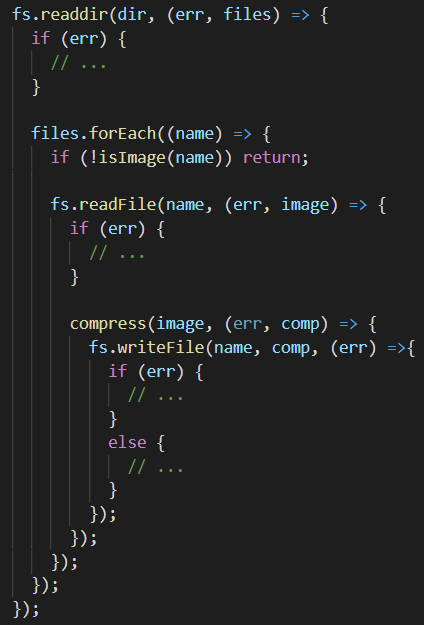
1. **Callback: Node.js style:** callback принимает **два параметра** – информацию об ошибке и данные.

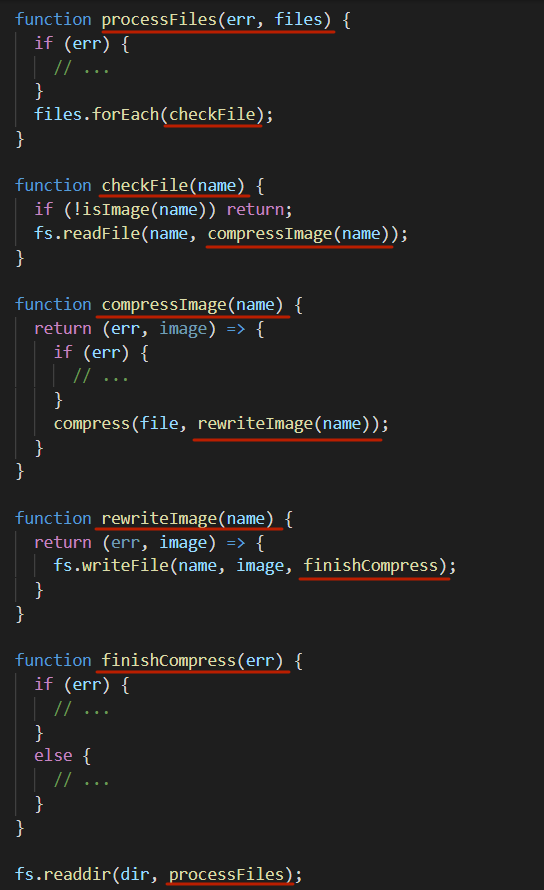


1. **Callback: плюсы:** повторное использование кода, изменение функциональности без изменения метода, создание цепочек вызовов; **минусы:** падение производительности, ухудшение читаемости.
2. **Callback: callback hell, pyramid of doom**



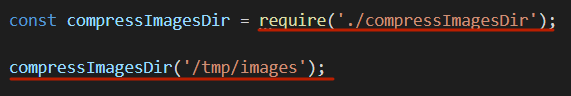
1. **Callback hell, pyramid of doom:** решение: 1) разбивать на функции; 2) разбивать на модули
2. **Callback hell, pyramid of doom:** разбитие на функции.



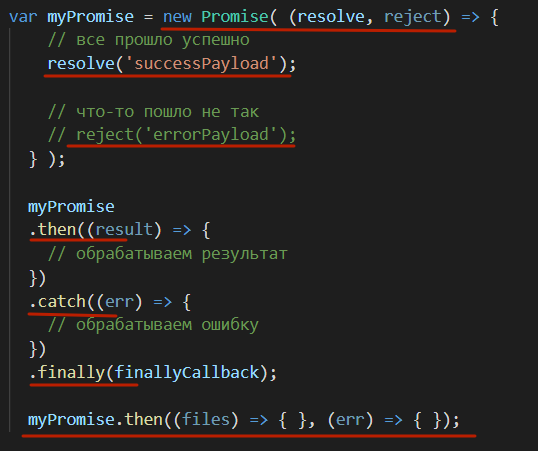


1. **Callback hell, pyramid of doom:** разбитие на модули.

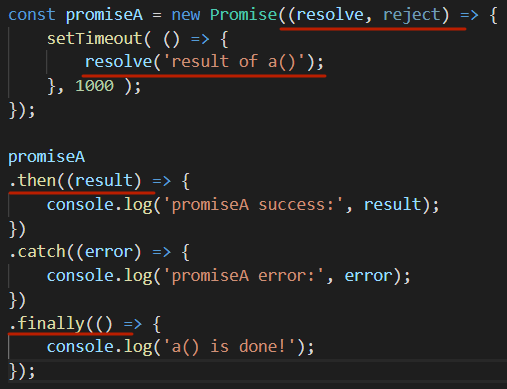


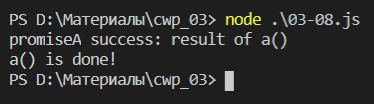


1. **Асинхронное программирование:** ES6/ES2015 – Promise, ES8/ES2017 – конструкция async/await.
2. **Promise (обещание):** объект, используемый для выполнения отложенных и асинхронных вычислений. Представляет собой операцию, которая еще не завершена, но ожидается в будущем.
3. **Promise:** создание, методы resolve и reject, обработчики then, catch, finally.

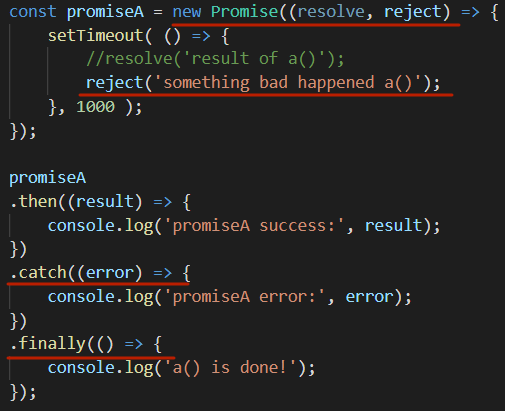


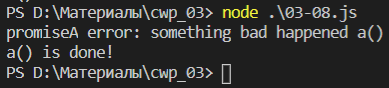
1. **Promise:** resolve, then, finally



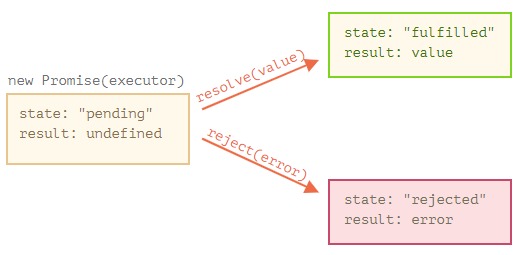


1. **Promise:** reject, catch, finally

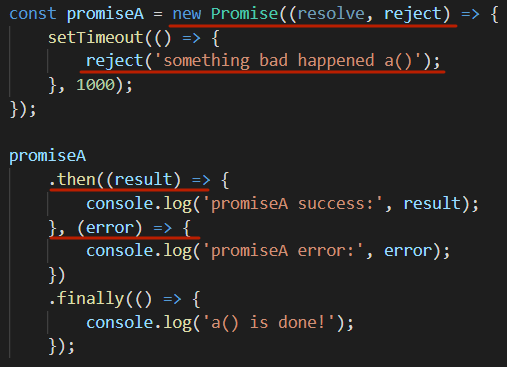


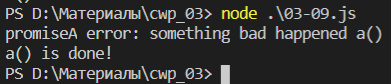


1. **Promise: Cвойствo state:** **pending** (ожидание), **fulfilled** (выполнено) при вызове resolve, **rejected** (отклонено) при вызове reject. **Свойство result**: вначале **undefined**, далее изменяется на **value** при вызове resolve(value) или на **error** при вызове reject(error).

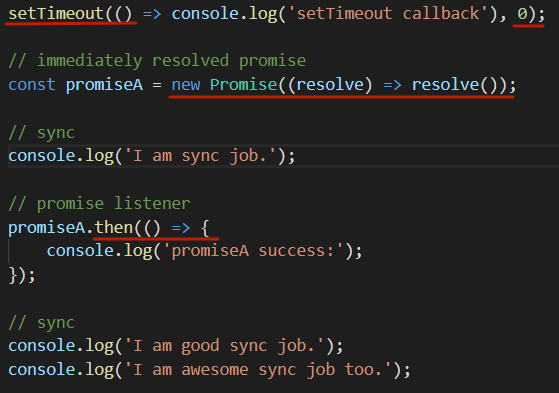


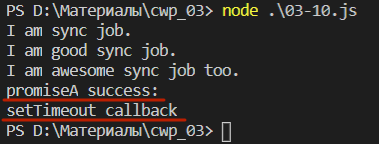
1. **Promise:** обработка отклоненного промиса без catch



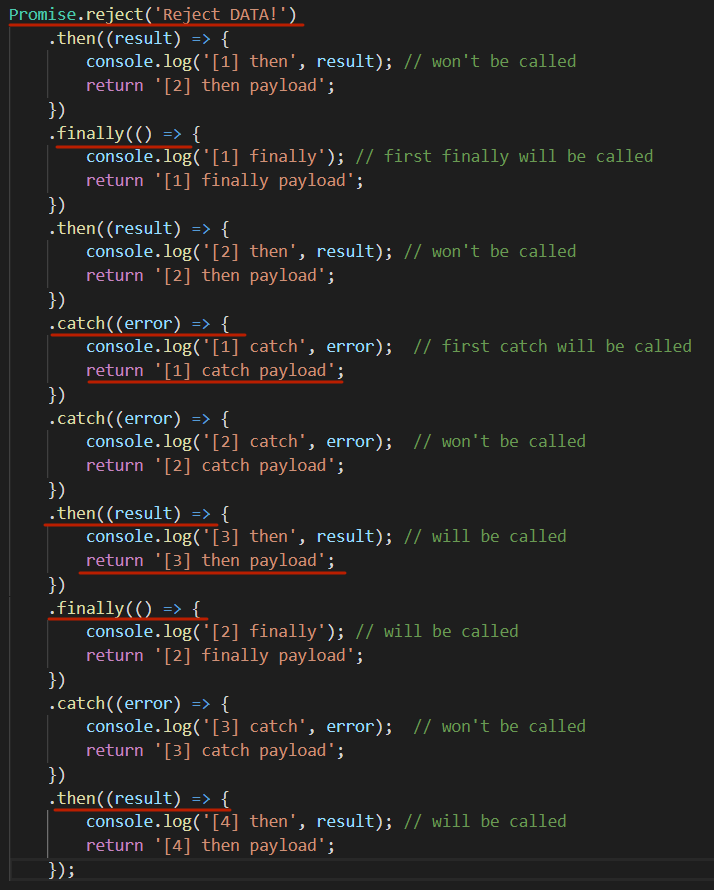


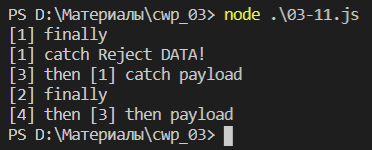
1. **Promise:** обратные вызовы промисов помещаются в очередь **микрозадач (microtask queue)**



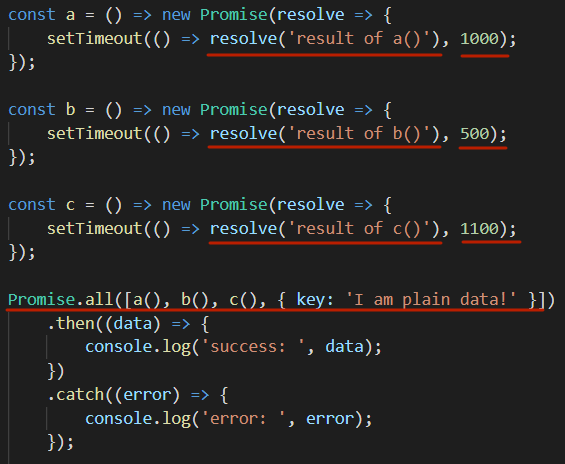


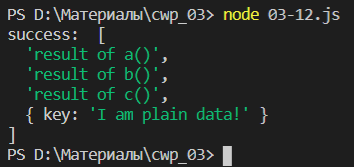
1. **Promise:** цепочка промисов. Обработчик промиса возвращает новый промис с неопределенным result (можно его задать с помощью оператора return).



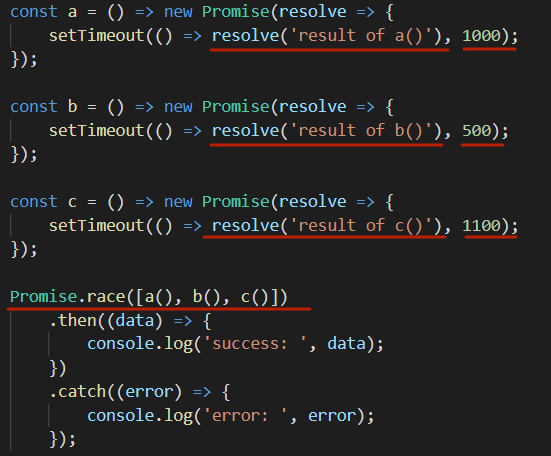


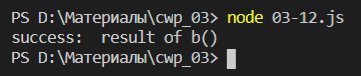
1. **Promise:** метод **Promise**.**all(iterable)** возвращает промис, который выполнится **после выполнения всех обещаний** в передаваемом итерируемом аргументе.



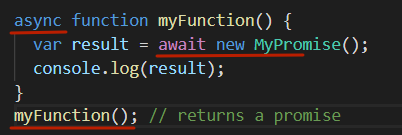


1. **Promise:** метод **Promise**.**race(iterable)**возвращает промис, который будет выполнен или отклонен с **результатом** исполнения **первого выполненного или отклонённого** итерируемого промиса.





1. **Async/await:** синтаксис для обработки нескольких промисов в режиме синхронного кода.
2. **Async/await: async** - перед объявлением функции, возвращает промис; **await** - блокирует код до тех пор, пока промис не будет разрешен или отклонен.



1. **Async/await:** пример

