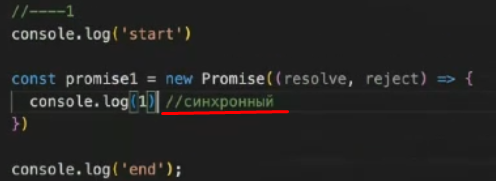
**--**

**Задача с собеса\_**

Есть IEFE которая сразу вызывается, внутри нее setTimeout(), внутри которого console.log("Привет мир!"), после самовызывающей функции бесконечный цикл, while (true). Выполнится ли консольлог внутри таймаута? Нет. Тк при бесконечном цикле у нас будет постоянно переполнятся стэк вызывов, и до очереди макрозадач просто не дойдёт.

Задача: Ответ: **start 1 end**

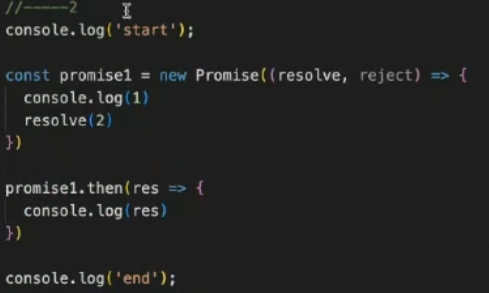
**так как**  - все что идет до **.then()** - синхронный код



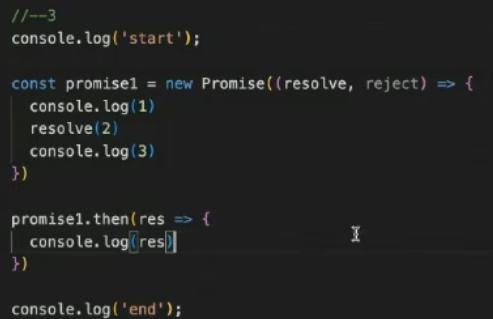
№ 2 — **Алгоритм решения задач: разделять код на синхронный и асинхронный**: сначала синхронный код — если есть await то он как синхронный отработает — микротаски (.then) — макротаски setTimeout()

**resolve** – промисс зарезолвился и результат попал в параметры в then – вызывается когда с бэка приходит асинхронный ответ — в него передается результат выполнения нашего запроса на бэк (данные помещаются), после резолва попадаем в **then** – но **«зен»** **асинхронная** **функция** и в **res** - двойка значит в консоле 2

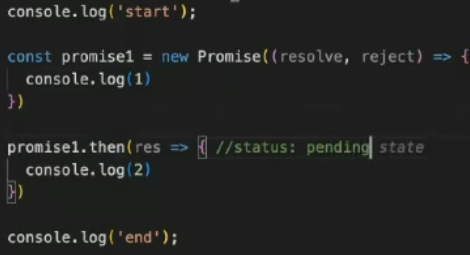
Ответ: start 1 end 2



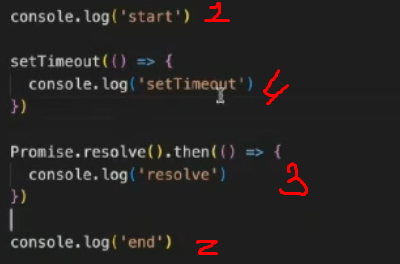
**№ 3: Ответ:** start 1 3 end 2 – делим на синхронный и нет код, все что внутри в промиссе до зен тоже синхронный, **resolve()** ничего не делает просто оповещает что пришли данные успешно с сервера, **.then** всегда асинхронный код.



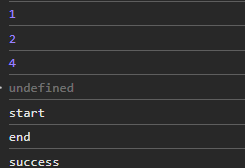
**№4** Ответ: start 1 end undefined – двоика не выполниться так как промисс в статусе pending всегда — тк промисс **не** **зарезолвился** и значит в **зен** **не** **попадем** — не отдадим в него данные



**№5** Ответ: **Алгоритм:** делим на синхронный и асинхронный — start end, далее видим асинхронный: В таких случаях Промисс имеет приоритет так как это **микротаска**(разделение очереди — микротаски это промиссы, async await и «мультиобсервер»),микротаски (then, resolve) приоритетнее чем макротаски (др асинхронный код, независимо от того что таимаут нулевой)



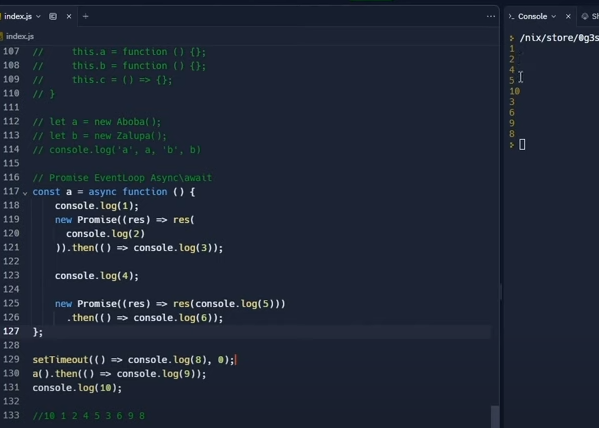
**№6:** Ответ: тк сетТаимаут внутри **синхронного промисса** он выполниться быстрее **микротаски — синхронный код быстрее отрабатывает**

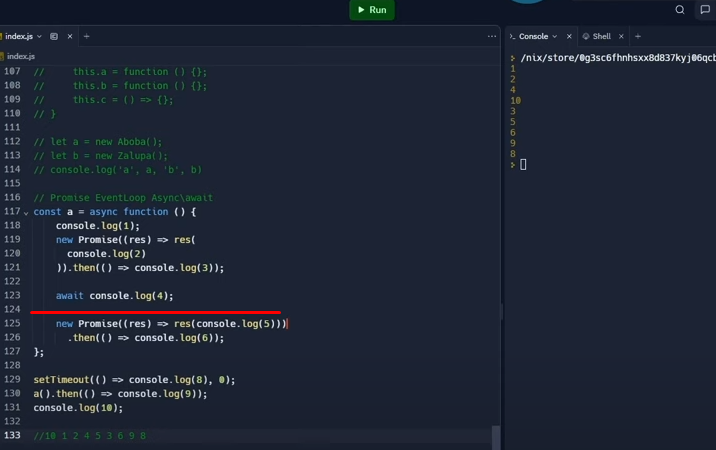




**№ 7: await** блочит выполнение кода тк он на генераторах — считать как синхронный код await при решении задач

- без await все работает стандартно:





----------

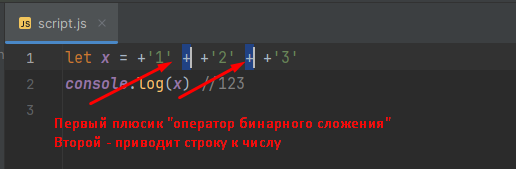
**Задачи:**

Задачи на комбинации строк:

“+” - оператор бинарного сложения и операнды сложения — **результат** сконкатинированная строка

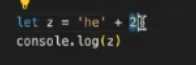


Приводим к числу: ответ: **6**



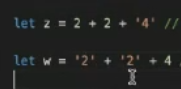
\_

**Число + строка приведет к строке — Ответ: 'he2'**  - даже если местами записи поменять:



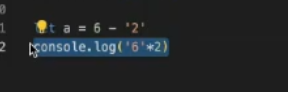
-

задача: ответ: **44 и 224** — идет слева направ



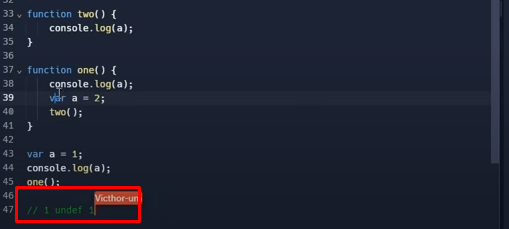
Оператор с минусом произведет вычисления «-»

Ответ: 4 и 12 — все вычисления корректо работают

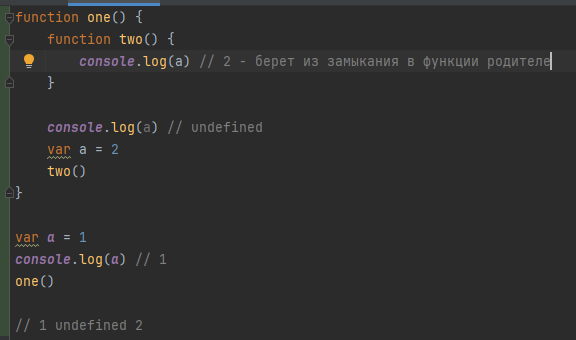


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

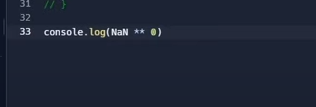
***Задача* Алгорит мысли: js** движок парсит сверху вниз — инициализирует просто сначала - потом только вызывает - мыслить парсингом и инициализацией + Сначала код вне блочной или функциональнои области отработает --- берет 2 из **ЛО = undefined – hoisting сработал — ответ —** *Сначла код вне блочной или функциональнои области отработает встречаем при парсинге первый консоль: console.log(а) - будет а == 1, вызываем one() - попадает в функцию тут хойстинг значит undefined, вызываем two() - там а берет из замыкания из глобальной (файловой области) видимости*

****

**Итак, итоговый вывод в консоль будет следующим:**



**ответ 1 — тк в нулевой степени \*\*** - степень

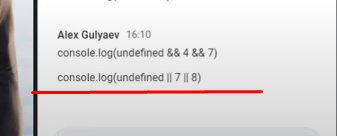
****

**Что выведится в консоль**

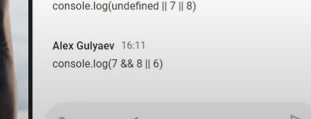


**— undefined – тк оператор && возвращает или последнии который конвертируется в true или первый который конвертируется в falsy значение(прервется на false)**

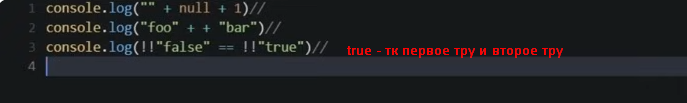
**с оператором | |** выведится число 7 так как false у первого значениям – до первого **true**



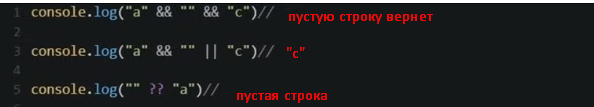
тут выведится **8** тк операция и приоритетьнее над или и тк 7 это true тогда: 8 то что спарва как в предыдущеи задаче — 7 не false значит 8

****

// приводиться к строкам (js к одному типу данных все приводит когда сравниваем разные типы данных) - вернет «» + null равно «null» и плюс 1: «null1»

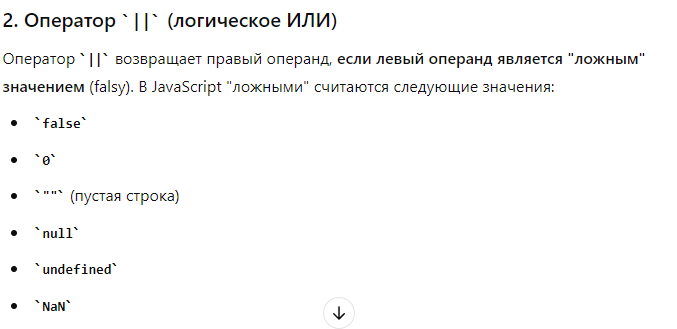
-

**а не false значит пустая строка, пустая строка false – тогда ответ пустая строка**

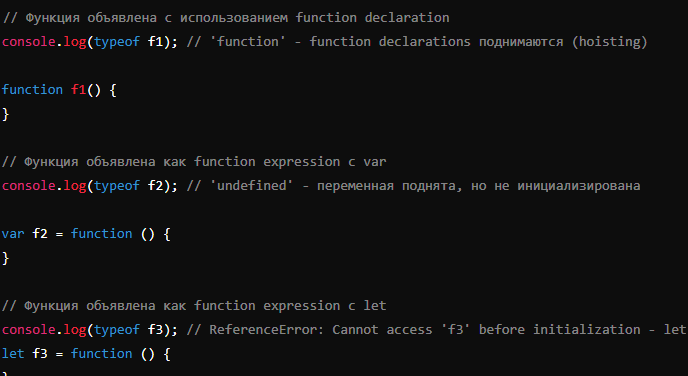
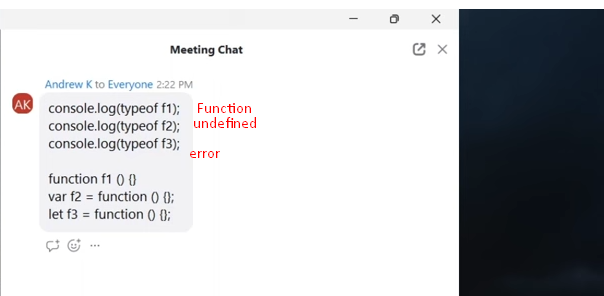


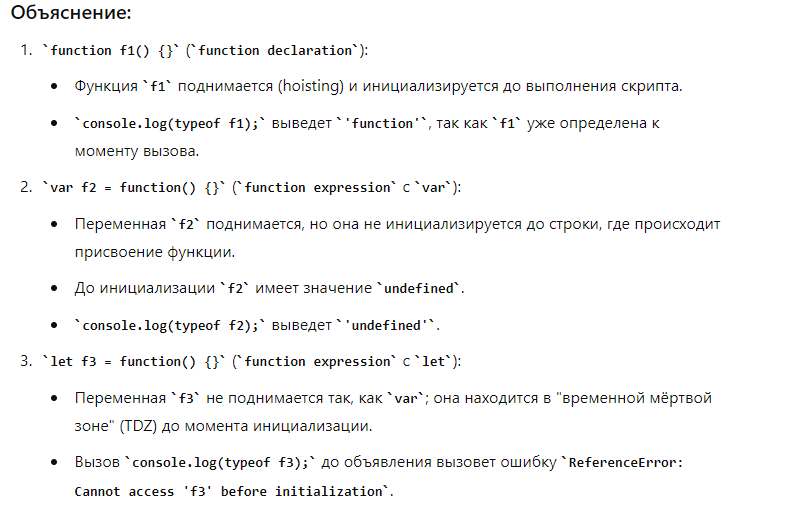
оператор **&&** возвращает или первое **false** или последнее **true –** тк „a“ не false а пустая строка не true - то идем дальше а дальше уже false значит пустая строка

**Оператор нулевого слияния ?? возвращает первый операнд, если он не является null или undefined – пустая строка не null и не undefined**

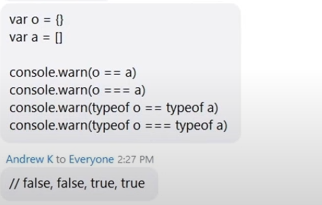
****

**Задача**: hoisting всплытие функций: **undifined** – тк хоистинг — используем выше а проинициализировали внизу

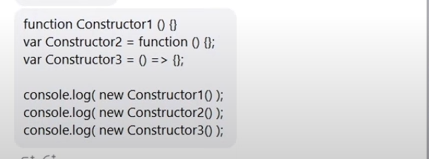




**Задача Ответ**: 2 объекта никогда не равны + true true

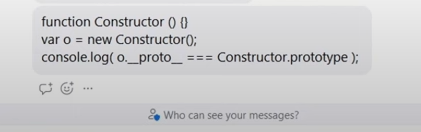


**Ответ**: 1 и 2 успешно — у нас создастся объект (функция тоже же объект), → **третья** ошибка — **Задача** тк **стрелочные функции не могут использоваться в качестве конструктора**

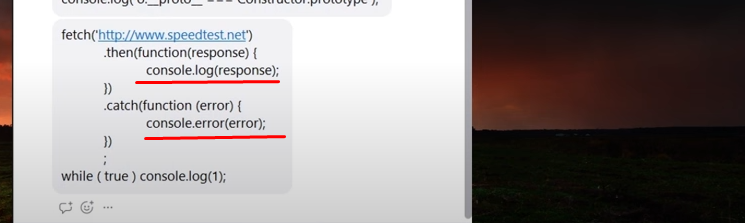


------

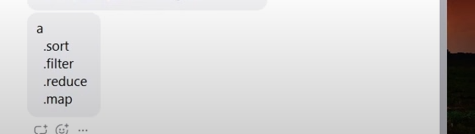
тут выведет **true** – так как **ссылка одна и та же** инстанса \_\_proto\_\_ на prototype родителя - одна и та же



**Задача** эти логи никогда не выведутся

 **тк будет выполняться бесконечный синхронный код — и сервер нас услышит?** колстек переполниться - Запрос на сервер все-таки уходит и выполняется, но его обработка (например, вывод данных) не произоидет

**Задача: Как оптимизировать это?**



Сократить кол-во циклов до 1 или меньшего кол-ва…, дальше map не может идти после reduce тк он вернет одно значение, map надо поднимать выше, filer тоже поднять выше чтобы для sort было меньшее кол-во итераций, **порядок**: filter map sort reduce

**Задача**

?? оператор «нулевого слияния» (возвращается оператор справа если значение слева: **null** или **undefined**)

**Задача Использование не строгого сравнения (==)**

Чтобы сравнить разные типы данных **js** все приводит к одному типу данных

*console.log(5 == "5"); //* ***true***

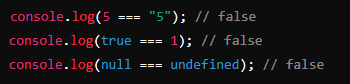
*console.log(true == 1); // true console.log(false == 0); //* ***true***

*console.log(null == undefined); //* ***true***

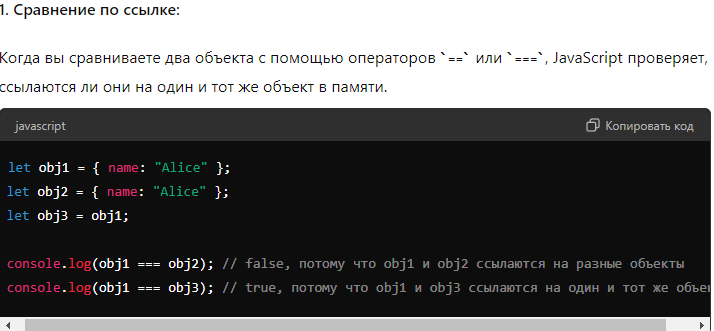
*console.log(0 == false); // true*

*console.log({} == "[object Object]"); // false*

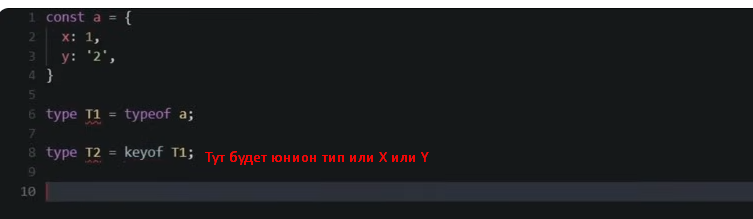
**Использование строгого сравнения (===)**



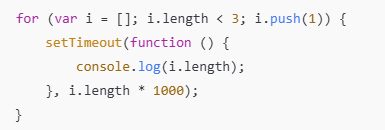
**Задача с объектом объектами на сравнение объектов**

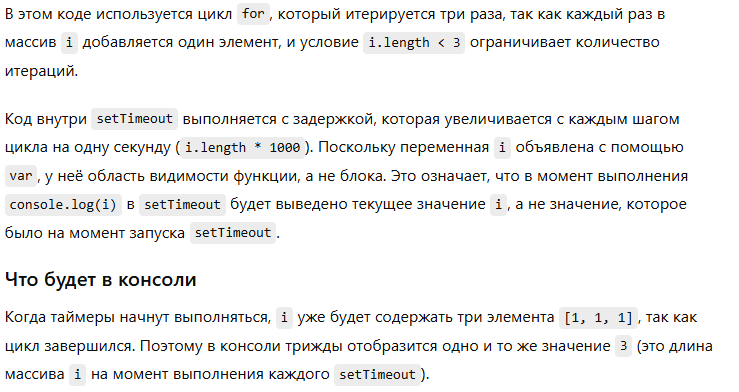
\_\_

задача:

**-**

Задача цикл var:





--

Задача: одна макротаска — потом все сихрнные и микротаски + 1 макротаска

