Давайте напишем немного кода программирования Javascript, чтобы практиковать то, что мы узнали! Ура! :-)

Время выполнения: около 30 минут.

Резюме: в этом задании вы собираетесь выполнить цикл над массивом имен и распечатать либо привет, либо прощание с этим именем в консоли браузера. Если имя начинается с буквы j или J, вы должны распечатать Goodbye JSomeName. Если имя начинается с любой другой буквы, вы должны распечатать Hello SomeName.

Однако для этого вам придется использовать 2 внешние библиотеки, код которых не готов к использованию на 100%. Используя то, что мы узнали в этом модуле, ваша задача будет заключаться в исправлении кода в этих библиотеках.

Вы получите некоторый стартовый код для работы, где все шаги того, что вам нужно сделать, четко прописаны для вас.

Вот что вам нужно будет сделать для того, чтобы выполнить это задание:

4. Вам нужно будет загрузить стартовые файлы для этого проекта и скопировать их в папку контейнера решения (например, в module4-solution). Поскольку задания и стартовый код время от времени обновляются, не думайте, что у вас уже есть последняя версия в вашей системе. Лучший способ убедиться, что вы работаете с самым последним стартовым кодом - это либо клонировать Git репозиторий fullstack-course4 в новый каталог, либо, если вы уже сделали "клонирование git" ранее, вы можете просто открыть свою командную строку (cmd на Windows или терминал на Mac), перейти в папку, в которую ранее был клонирован репозиторий fullstack-course4, и сделать: git pull.

Это позволит обновить Вашу локальную копию репозитория с учетом всех изменений, внесенных с момента последнего обновления.

Напомним, что полный URL-адрес репозитория выглядит следующим образом: https://github.com/jhu-ep-coursera/fullstack-course4

5. Как только локальный репозиторий в вашей системе будет обновлен, у вас будет выбор! Если вы хотите немного более сложное задание, используйте код в папке "сложнее" в качестве отправной точки. Если вы хотите немного менее сложное задание, используйте код в папке "проще" в качестве отправной точки. Разница между этими двумя начальными точками заключается в том, что в "более легкой" начальной точке есть несколько шагов, которые уже завершены для вас.

Сложнее: если вы хотите немного более сложное задание, скопируйте все содержимое папки fullstack-course4/assignments/assignment4/assignment4-solution-starter/harder в недавно созданную папку контейнера решений для этого задания, например "module4-solution".

Проще: если вы хотите получить немного менее сложное задание, скопируйте все содержимое папки fullstack-course4/assignments/assignment4/assignment4-solution-starter/easier в недавно созданную папку контейнера решений для этого задания, например "module4-solution".

Примечание: предоставленный стартовый код не будет работать. Это зависит от вас, чтобы следовать инструкциям, чтобы заставить его работать. После того как вы скопировали выбранный вами исходный код в папку решения, откройте папку решения в редакторе кода. Откройте сценарий.JS-файл и выполните следующие действия.

6. Когда вы постоянно работаете над заданием, используйте функцию синхронизации браузера и держите Chrome открытым на вкладке консоли инструментов разработчика Chrome. Скорее всего, вы увидите там ошибки для начала. Следуйте инструкциям, описанным в стартовом коде, и эти ошибки должны исчезнуть к тому времени, когда вы закончите последний шаг. Если вы все еще видите ошибки в этот момент или вы не видите результат, который вы должны видеть, вы, вероятно, где-то ошиблись, поэтому посмотрите на это и исследуйте.

**// STEP 1**: (NOTHING TO DO. ALREADY DONE FOR YOU)

// Wrap the entire contents of script.js inside of an IIFE

// See Lecture 52, part 2

// (Note, Step 2 will be done in the SpeakHello.js file.)

// **STEP 2**: Wrap the entire contents of SpeakHello.js inside of an IIFE

// See Lecture 52, part 2

// **STEP 3**: Create an object, called 'helloSpeaker' to which you will attach

// the "speak" method and which you will expose to the global context

// See Lecture 52, part 1

// DO NOT attach the speakWord variable to the 'helloSpeaker' object.

var speakWord = "Hello";

// **STEP 4**: Rewrite the 'speak' function such that it is attached to the

// helloSpeaker object instead of being a standalone function.

// See Lecture 52, part 2

**// STEP 5**: Expose the 'helloSpeaker' object to the global scope. Name it

// 'helloSpeaker' on the global scope as well.

// See Lecture 52, part 2

// (Note, Step 6 will be done in the SpeakGoodBye.js file.)

// xxxx.xxxx = helloSpeaker;

// **STEP 6**: Wrap the entire contents of SpeakGoodBye.js inside of an IIFE

// See Lecture 52, part 2

**// STEP 7:** Create an object, called 'byeSpeaker' to which you will attach

// the "speak" method and which you will expose to the global context

// See Lecture 52, part 1

// var byeSpeaker =

// DO NOT attach the speakWord variable to the 'byeSpeaker' object.

var speakWord = "Good Bye";

**// STEP 8:** Rewrite the 'speak' function such that it is attached to the

// byeSpeaker object instead of being a standalone function.

// See Lecture 52, part 2

// **STEP 9**: Expose the 'byeSpeaker' object to the global scope. Name it

// 'byeSpeaker' on the global scope as well.

// xxxx.xxxx = byeSpeaker;

// **STEP 10**: (NOTHING TO DO. ALREADY DONE FOR YOU)

// Loop over the names array and say either 'Hello' or "Good Bye"

// using the 'speak' method or either helloSpeaker's or byeSpeaker's

// 'speak' method.

// See Lecture 50, part 1

// **STEP 11**: (NOTHING TO DO. ALREADY DONE FOR YOU)

// Retrieve the first letter of the current name in the loop.

// Use the string object's 'charAt' function. Since we are looking for

// names that start with either upper case or lower case 'J'/'j', call

// string object's 'toLowerCase' method on the result so we can compare

// to lower case character 'j' afterwards.

// Look up these methods on Mozilla Developer Network web site if needed.

// **STEP 12**: (NOTHING TO DO. ALREADY DONE FOR YOU)

// Compare the 'firstLetter' retrieved in STEP 11 to lower case

// 'j'. If the same, call byeSpeaker's 'speak' method with the current name

// in the loop. Otherwise, call helloSpeaker's 'speak' method with the current

// name in the loop.

JavaScript имеет семь встроенных типов, шесть примитивов и один тип объекта:

object type - это не что иное, как набор пар имя/значение(name/value pairs)

primitive type - Неизменяемый означает, что как только он установлен, как только это значение установлено, оно не может быть изменено. Это означает, что после его установки значение переменной становится доступным только для чтения. Вы, конечно, можете создать другое значение на основе существующего значения a, но пространство памяти, выделенное для первого значения, не изменяется. Вместо этого вы создаете новое пространство памяти для нового значения.

Поэтому позвольте мне пробежаться по одному из всех примитивных типов, которые определяет JavaScript. Во-первых, булево. Ну, логическое значение может иметь только два значения, либо true, либо false. А true и false-это зарезервированные ключевые слова в языке JavaScript

Другой тип называется неопределенным. Неопределенный тип данных означает, что для этой конкретной переменной данного типа никогда не устанавливалось никакого значения. Это значение, которое получает каждая переменная, когда движок JavaScript устанавливает переменную в памяти, когда она определена, но еще никогда не присваивалось какое-либо значение. И это может иметь только одно значение. И это зарезервированное ключевое слово undefined

Еще одно значение, которое определяет язык JavaScript, - это null. Null означает отсутствие значения.

Другой тип-это number -это фактически единственный числовой тип в JavaScript, и он всегда представлен под капотом в виде 64-битной плавающей запятой двойной точности.

Другой тип данных-strings, а string в JavaScript очень похож на string type в других языках в том смысле, что это последовательность символов, используемых для представления текста, и вы можете определить строки либо с помощью одинарных, либо двойных кавычек, либо один из них является 100% законным.

Последний примитивный тип, который я упомяну, - это symbol. Теперь мы действительно не будем рассматривать этот тип данных в этом классе. И причина в том, что он является новым для ES6

Сколько раз слово "Привет" будет напечатано на консоли?

var x = 10;

if ( (null) || (console.log("Hello")) || x > 5 ) {

console.log("Hello");

}

Поскольку мы используем оператор OR, оператор ' if ' перестанет вычислять условия после того, как он столкнется с самым первым предложением, которое вычисляется как true.

null-это ложь, поэтому мы продолжаем

приставка.log ("Hello"), выводит "Hello" на консоль и ничего не возвращает, так что это "undefined", а "undefined" - ложь, поэтому мы продолжаем

x > 5 является истинным, поэтому мы выполняем тело оператора 'if' и печатаем еще одно "Hello".

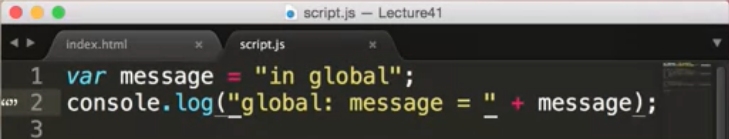
Итак, ответ заключается в том, что "Привет" будет напечатано на консоли 2 раза.

Объектный литерал - это просто фигурные скобки, и мы можем поставить точку с запятой в конце, потому что это оператор, или на самом деле это выражение, которое будет иметь свое значение внутри переменной

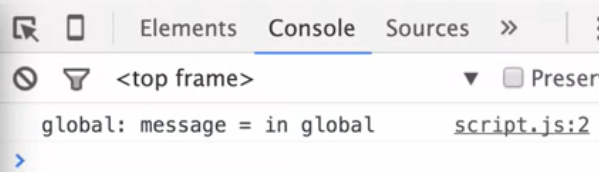
**Области видимости!!!!!!!!!!**

**1.**

Запрос:



Результат:



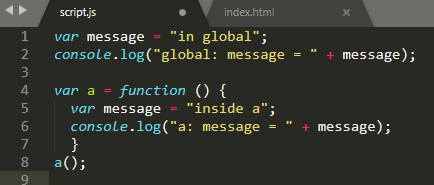
**var** message = "in global";

console.log("global: message = " + message);

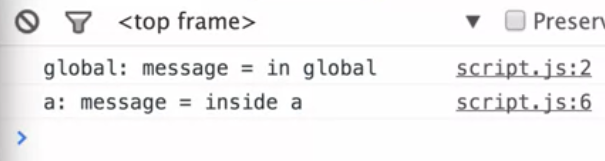
**=> global: message = in global**

**2.**

Запрос



Результат



var message = "in global";

console.log("global: message = " + message); **=> global: message = in global**

**var** **a** = function () **{**

**var** **message** = "inside a"; - задана внутри функции и использует саму себя

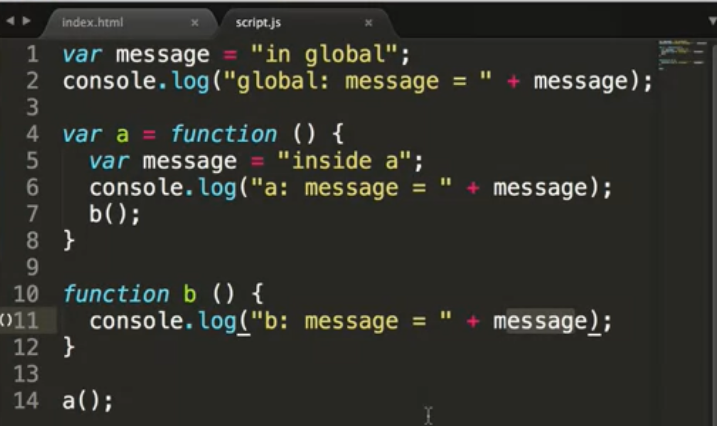
console.log("a: message = **" + message**);

**}**

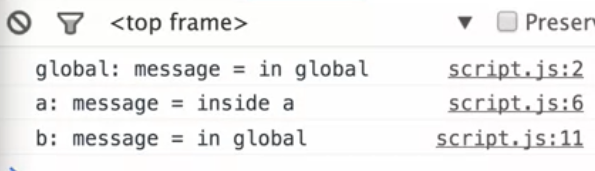
**a(); => a: message = inside a**

**3.**

Запрос

****

Результат



**var message = "in global";**

console.log("global: message = " + message); **=> global: message = in global**

var a = function () **{** **=> a: message = inside a**

**var** **message** = "inside a";

console.log("a: message = " + **message**);

**b()**

**}**

**function b ()** **{**

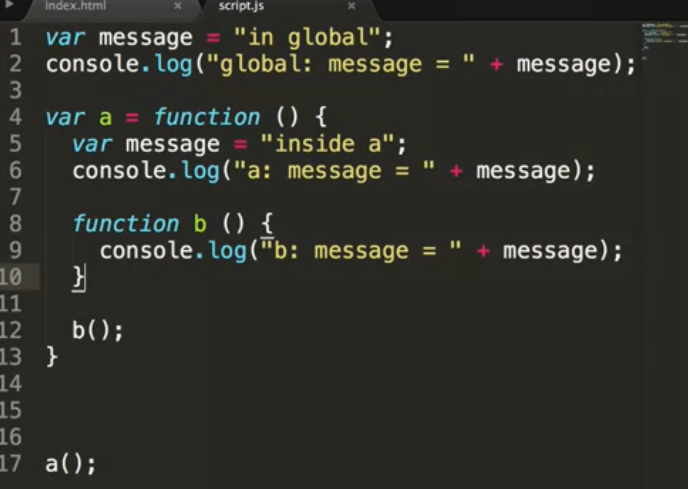
console.log("b: message = " + **message**);

**}** **=> b: message = in global**

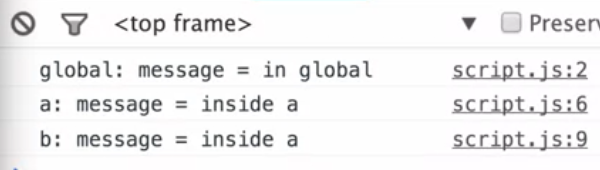
**a();**

**4.**

Запрос



Результат



var message = "in global";

console.log("global: message = " + message); **=> global: message = in global**

var a = function () **{**

var message = "inside a";

console.log("a: message = " + message); **=> a: message = inside a**

function b () {

console.log("b: message = " + message); **=> b: message = inside a**

}

**b()**

**}**

**a();**