2.1 What is the order of logging? Please explain why.

(function() { console.log(1);

setTimeout(function(){console.log(2)}, 1000);

setTimeout(function(){console.log(3)}, 0);

console.log(4); })();

1,4,3,2

У setTimeout есть два аргумента: первый аргумент - это вызываемая функция, а второй аргумент - это интервал времени в миллисекундах. Задача setTimeout немедленно установить таймер, который истекает через заданный интервал времени. Когда этот таймер истекает, выполняется код, который находится в функции обратного вызова первого аргумента, переданного в setTimeout.

Из-за своей однопоточной природы JavaScript может когда-либо выполнять только один фрагмент кода за раз, это означает, что когда происходит асинхронное событие (срабатывание таймера), оно ставится в очередь для выполнения в следующий доступный момент.

console.log (1) отправляется в стек вызовов и выполняется, после того, как процесс завершится, он выкидывается из стека.

setTimeout(function(){console.log(2)}, 1000); и setTimeout(function(){console.log(3)}, 0); помещаются в очередь событий и не выполняются, пока не будет выполнен весь встроенный код.

console.log(4); отправляется в стек, выполняется и тут же выкидывается из него.

setTimeout(function(){console.log(2)}, 1000); и setTimeout(function(){console.log(3)}, 0); обнаруживаются в стеке, так как задержка равна 0 для function(){console.log(3)} она происходит раньше. Потом выполняется function(){console.log(2)} после 2-х секунд задержки.

2.2 Write a function that would allow you to do this:

var addSix = createBase(6);

addSix(10); // returns 16

addSix(21); // returns 27

function createBase(a){

return function(b){return (a+b)};

}