Advanced topics

Execution Context

Lexical Environment

Scope Chain

Hoisting

Function Invocation

Function Scope vs Block Scope

Dynamic vs Lexical Scope

this - call, apply, bind

IIFE

Closures & Prototypal Inheritance

Comparing Languages

File extensions

C++ compiles to .exe

g++ your\_program.cpp -o your\_program

Java

JVM compiles to bytecode (understood by computers)

MVN compiles JAVA projects into a bundle

Is JavaScript an interpreted language? Yes & No

It depends on the JavaScript Implementation

JavaScript = ES5 interpreted language

JavaSciprt = React, Angular, Vue3, Webpack compiled languages

// High Level Language

let result = 1 + obj.x;

// V8 Engine bytecode

LdaSmi [1]

Star r0

LdaNamedProperty a0, [0], [4]

Add r0, [6]

// x86\_64 machine code (bytecode)

movl rbx, [rax+0x1b]

REX.W movq r10,0x100000000

REX.W cmpq r10,rbx

jnc 0x30d119104275 <+0x55>

REX.W movq rdx, 0x100000000

call 0x30d118e843e0 (Abort)

int3laddl rbx,0x1

...

Writing Optimized Code

Rare implementation during our work environment,

yet it sometimes helps performance tuning

* eval()
* arguments
* for in
* with (seldom seen)
* delete
* Hidden classes
* Inline caching

Before we have an idea of what Performance Optimization is 'at a

coding level'

We get to know the difference between Static Typing &

Dynamic Typing

**Hidden classes**

<https://richardartoul.github.io/jekyll/update/2015/04/26/hidden-classes.html>

Under the hood 
Javascript Hidden Classes and 
Caching in V8 
Apr 26, 2015 
Hidden Classes 
Javascript is a dynamic programming language which means that properties c 
removed from an object after its instantiation. For example, in the code snippet 
instantiated with the properties "make" and "model"; however, after the object 
created, the "year" property is dynamically added. 
var car = { 
this. make = make; 
this. model = model; 
var myCar = new ; 
myCar.year = 2005; 

In a non-dynamic language like Java, a property's location in memory can ofter 
only a single instruction whereas in Javascript several instructions are required 
from a hash table. As a result, property lookup is much slower in Javascript tha 
languages. 
Since the use of dictionaries to find the location of object properties in memory 
a different method instead: hidden classes. Hidden classes work similarly to thi 
(classes) used in languages like Java, except they are created at runtime. Whil 
post, keep in mind that V8 attaches a hidden class to each and every object, a 
hidden classes is to optimize property access time. Now, Lets take a look at wt 
function Point (x, y) { 
this.x = x; 
this. y = y; 
var obj = new Point(I,2); 
Once the new function is declared, Javascript will create hidden class CO. 
A Point object 
Class pointer 
Initial 
class CO 

A Point Object 
Class pointer 
Offset 0: x 
Hidden class Cl 
FCC X 
offset O 
Initial hidden 
class CO 
If you add 
property x, 
transition to 
class Cl 
Everyfirne a new property is added to an object, the objects old hidden class 
transition path to the new hidden class. Hdden class transitions are importan 
hidden classes to be shared among objects that are created in the sane waj 
hidden class and the same property is added to both of them, transifions ens 
receive the same new hidden class and all the opåmized code that comes w 
This process is repeated when the statement "this.y = y" is executed. A new 
created, a class transition is added to CI stating that if a property "y" is added 
already contains property "x") then the hidden class should change to C2, and 
class is updated to C2. 
A Point object 
Class pointer 
Offset O: x 
Offset 1: y 
Hidden class C2 
see 
offset O 

Note: Hidden class transitions are dependent on the order in which properties 
Take a look at the code snippet below: 
1 
2 
3 
4 
5 
7 
8 
9 
10 
11 
Point(x,y) { 
function 
this.x = x 
this. y = y 
= new Point(1,2); 
var objl 
= new POint(3,4); 
var obj2 
objl.a = 
5; 
objl.b = 
lø; 

For the objects instantiating the sample class, these objects inheriting

the same class share a common 'Hidden Class'

Until more class.property are added, each of these objects

has their own respective transition path (calculated by offset) to their 'Hidden Classes'

Up until line 9, objl and obj2 shared the same hidden class. However, since pr 
added in opposite orders, objl and obj2 end up with different hidden classes 
separate transition paths. 
Initial hidden class 
If you add property 
X, transition to 
class Cl 
Hidden class CI 
For x see offset O 
If you add property 
y. transion to 
hidden class C? 
class C2 
For x see offset O 
y See Offset 
If you prt»erty 
a, transition to 
hidden class C3a 
If you add property 
b, transition to 
hidden class C3b 
Hidden class C3a 
For x see offset O 
For y see offset 1 
For a see offset 2 
If you add property 
b, transition to 
hidden class C4a 
Hidden class C3b 
For x see Offset O 
For y See offset 1 
For b see offset 2 
If pu add property 
a. transition to 
hidden class Cab 

**V8 Engine** (base of **Node.js**) uses **Inline Caching** for optimizing

dynamically typed languages

If you've been following along closely, your first instinct might be to think that o 
different hidden classes isn't a big deal. As long as each of their hidden classe 
offsets, accessing their properties should be just as fast as if they shared a hic 
to understand why this isn't true, we need to take a look at another optimizatior 
V8 called inline caching. 
Inline Caching 
V8 takes advantage of another commonly used technique for optimizing dynar 
called "inline caching". An in-depth explanation of inline caches in Javascript 
simple terms inline caching relies upon the observation that repeated calls to t 
occur on the same type of object. 
So how does it work? V8 maintains a cache of the type of objects that were 
recent method calls, and uses that information to make an assumption about tI 
be passed as a parameter in the future. If V8 is able to make a good assumpti 
object that will be passed to a method, it can bypass the process of figuring o 
objects properties, and instead use the stored information from previous looku 
class. 
So how are the concepts of hidden classes and inline caching related? Whene 
a specific object, the V8 engine has to perform a lookup to that objects hidder 
offset for accessing a specific property. After two successful calls of the same 
hidden class, V8 omits the hidden class lookup and simply adds the offset of t 

Code Optimization JavaScript GitHub

<https://github.com/petkaantonov/bluebird/wiki/Optimization-killers#3-managing-arguments>

Inline Caching

JavaScript-Optimization > writing-optimized-code.js > JS writing 
1 
2 
3 
4 
5 
6 
7 
8 
9 
10 
11 
12 
// Inline Caching 
function findUser(user) { 
return 'found ${user. firstName} ${us• 
// ES6 here to pretend as a DB for quick 
const userData = { 
firstName: 'Johnson' , 
tastName: 'Senior' 
// Invoke callback function with userDat; 
passed—in parameter 

[Running] node "/Users/j iaqilv/Desktop. 
JavaScript—Optimization/writing—optimi. 
writing—optimized—code—I. j s" 
[Done] exited with code=Ø in e. 057 sec 
[Running] node "/Users/j iaqilv/Desktop. 

Hidden Classes demo

When we are using OOP for JavaScript

The Order/Sequence we deal with our objects affects how objects

get to their respective Hidden Classes using Transition Paths

For example, let's create a function/

Why NOT just use machine code from start?

Why we do not just use machine code from start?

Thus, we can stop using JavaScript/V8 Engine compiler

for better performance?

Because of cross-browser compatibility issue

Common Web Browsers

e.g. Microsoft Edge, Chrome, Firefox

These browser companies need to agree upon a common

JavaScript code binary executable format to allow Developers

keep on developing cross browser code

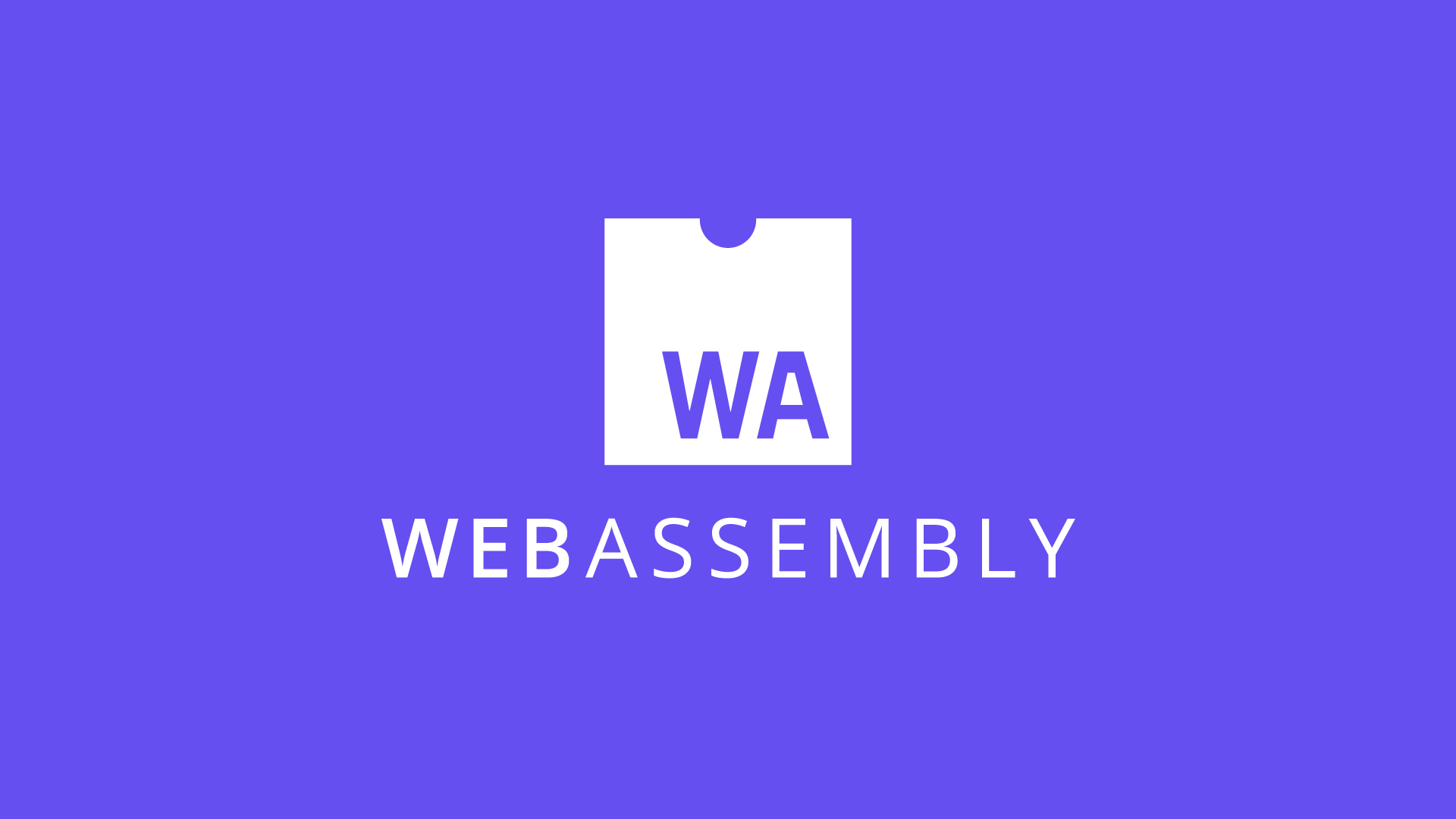
As these browsers interpret machine code (binary format) differently

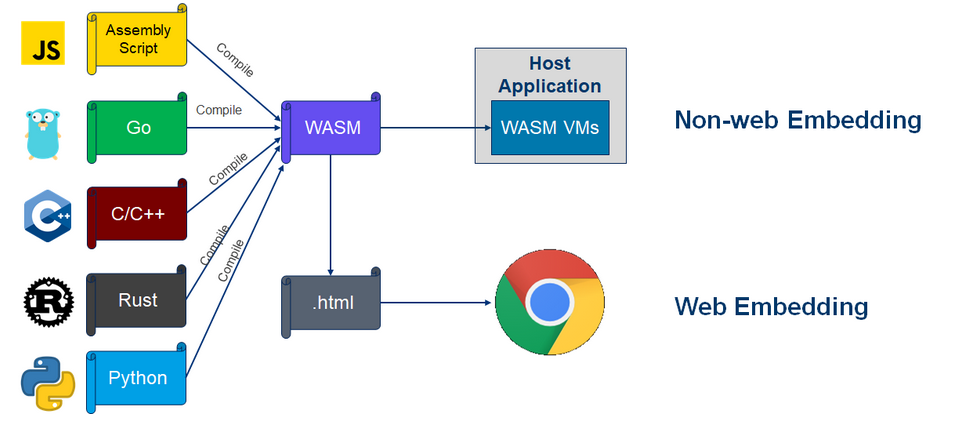
Using machine code for Development by Developers are NOT practical

Now, we've got **WebAssembly** for us as Web Developers to develop

higher performance JavaScript code across browsers

**WebAssembly will be a 'Game Changer' in the future**





Memory Heap & Call Stack Intro

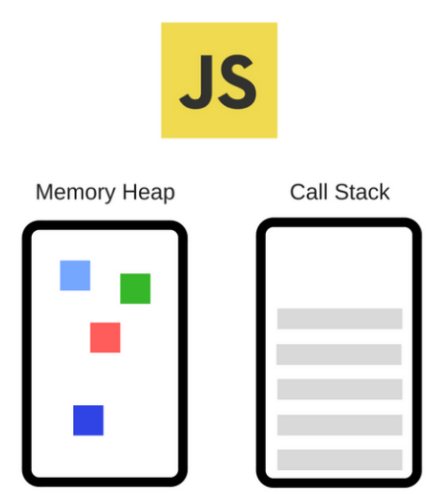
Explain Asynchronous vs Synchronous

JS is a single threaded language that can be non-blocking??

What is a Program?

-Allocate memory

-Parse and execute



Memory Heap:

Where memory allocation happens

Call Stack:

Read & Executed =

Where you're in the program

// Allocating to memory

const a = 1

const b = 10

const c = 100

Memory Leak happens when you have unused memory

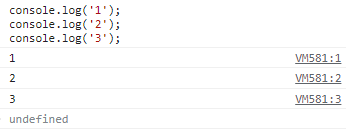
Global variables are bad that will fill up the memory

// Call stack

console.log('1');

console.log('2');

console.log('3');



run console.log('1');

returns 1 to console

removes it

run console.log('2');

returns 2 to console

removes it

run console.log('3');

returns 3 to console

removes it

// Call stack

const one = () => {

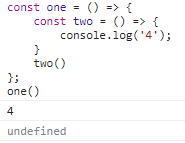
    const two = () => {

        console.log('4');

    }

    two()

}



console.log('4')

// console.log('4') runs inside of two()

two()

// two() is called when running one()

one()

// Call Stack

//1 remove console.log('4)

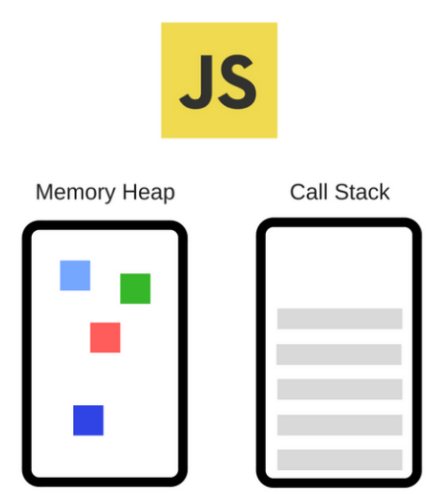
// 2 remove two()

// 3 remove one()

// Call stack is emptied

JS is a single threaded language that can be non-blocking??

JS = 1 Call Stack only = Can only do 1 thing at a time



Other languages can have multiple Call Stacks - Multi-threaded

Multi-threaded languages can have Dead Locks

Synchronous = 1 -> 2 -> 3

Stack overflow can be created

// Recursion = a func calls itself

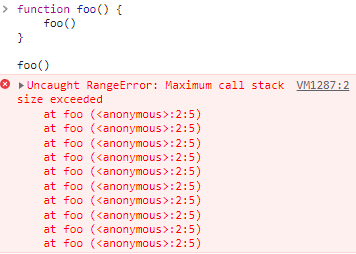
// keeps looping over & over - no end

function foo() {

    foo()

}

foo()



JS is a single threaded language that can be non-blocking by:

Asynchronous Javascript

// Asynchronous JS

// Asynchronous

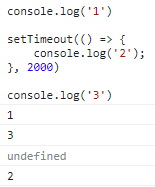
console.log('1')

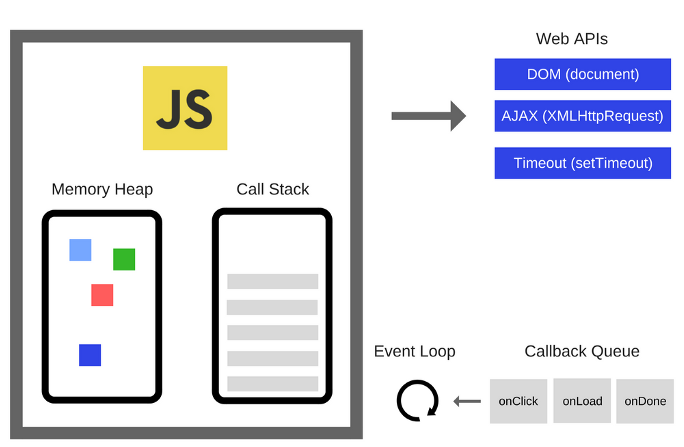
setTimeout(() => {

    console.log('2');

}, 2000)

console.log('3')





// Experiment2

console.log('1')

setTimeout(() => {

    console.log('2');

}, 0)

// although Timeout = 0

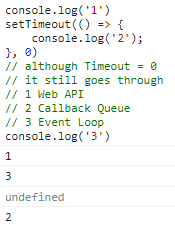
// it still goes through

// 1 Web API

// 2 Callback Queue

// 3 Event Loop

console.log('3')



// Synchronous vs Asynchronous

// Synchronous

// 1 Call teacher

// 2 Asks & waits for teacher to answer

// 3 Teacher pick ups & hopefully answers

// Asynchronous

// 1 Sends teacher a Text

// 2 When teacher has their time

// 3 Will respond & call you back

// You'll have time to do something else in the meantime

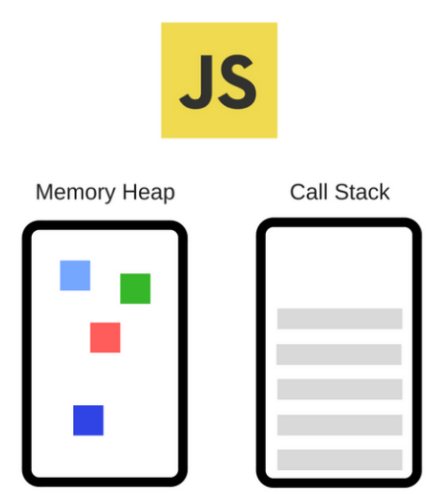
// Event Listener

Element.addEventListener('click', () => {

    console.log('click')

})

Memory Heap & Call Stack demo-1



JavaScript-Optimization > call-stack-and-memory-heap > JS de 
1 
2 
3 
4 
5 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
27 
// Call Stack + Memory Heap demo 
const number = 610; 
// allocate memory f 
const string = 'some text' // allocate 
string 
// allocate memory for an object... and 
const human = { 
first: 'User', 
last: 
'1' 
function subtractByTwo(number) { 
return number 
function calculate(numl, num2) { 
// Using Type Coersion to ensure no 
const sumTotal = +numl + +num2; 
// Invoking another callback functio 
callback function, 
// passing in 'sumTotal' as a passed 
to subtractByTwo(number) 
return subt ractByTwo( sumTotal) ; 
Using Google Chrome browser to run th 
To pause JavaScript compiler execution h 
mac OS Dev Tool Console type 
debugger; 

code

// Call Stack + Memory Heap demo

const number = 610; // allocate memory for 'number'

const string = 'some text' // allocate memory for a string

// allocate memory for an object... and it's values

const human = {

first: 'User',

last: '1'

}

function subtractByTwo(number) {

return number - 2;

}

function calculate(num1, num2) {

// Using Type Coersion to ensure no errors

const sumTotal = +num1 + +num2;

// Invoking another callback function inside this callback function,

// passing in 'sumTotal' as a passed-in parameter to subtractByTwo(number)

return subtractByTwo(sumTotal);

}

/\* Using Google Chrome browser to run this code

To pause JavaScript compiler execution here

mac OS => Dev Tool => Console => type 'allow pasting'

\*/

debugger;

/\* Whenever invoking a callback =>

Find inside Memory Heap =>

Lookup memory address of callback function object =>

Assign a new memory address for 'result1' after having referred to memory address of calculate(num1,num2)

\*/

result1 = calculate(4, 5);

console.log(`result1 = calculate(4,5 ):\n${result1}`);

mac OS => Chrome browser => Dev Tool

type 'allow pasting'

paste the code above

We'll see browser is now paused

=> (anonymous) is on the right tab

=> (anonymous) = Global Execution Context

Performance 
Paused in de... 
Q 
Online Cour„. 
Elements Console 
Page Workspace 
top 
new-tab-page 
icons 
(index) 
cart. mojom-webui.js 
chrome_cart.mojom-webuij 
discount.mojom-webui.js 
feed.mojom-webui.js 
file_suggestion.mojom-web 
google_calendar.mojom-wet 
history_clusters.mojom-web 
history_clusters_layout_ty 
history_types.mojom-webui. 
lazy_load.js 
most_relevant_tab_resumpt 
mojom -webu 
photos.mojom-webui.js 
shared,rollupjs 
strings.m.js 
Sources Network 
(E VM79 x 
Memory 
Se 
15 
16 
17 
18 
19 
2ø 
21 
22 
23 
24 
25 
26 
27 
28 
29 
3ø 
31 
32 
33 
34 
35 
36 
function calculate(numl, num2) { 
// Using Type Coersion to ensure r 
const sumTotal +num1 + +num2; 
// Invoking another callback funct 
// passing in 'sumTotat' as a pass 
return subtractByTwo(sumTota1); 
using Google Chrome browser to run 
To pause JavaScript compiler executior 
mac OS Dev Tool Console type 
Whenever invoking a callback 
Find inside Memory Heap 
Lookup memory address of callback func 
Assign a new memory address for iresul 
resuttl calculate(4, 5); 
console. log(• resultl ) 