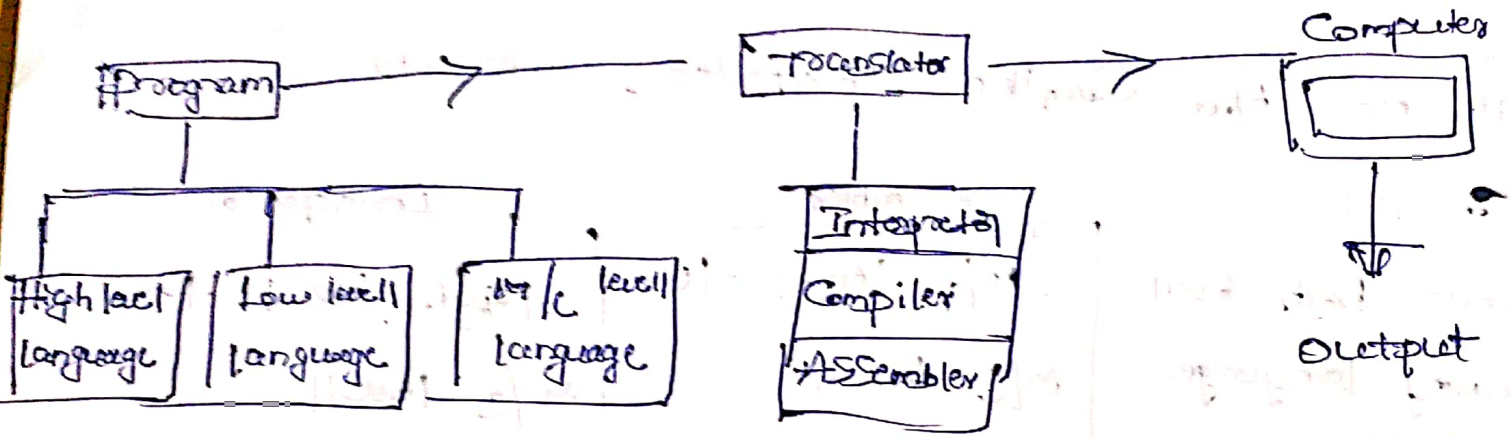
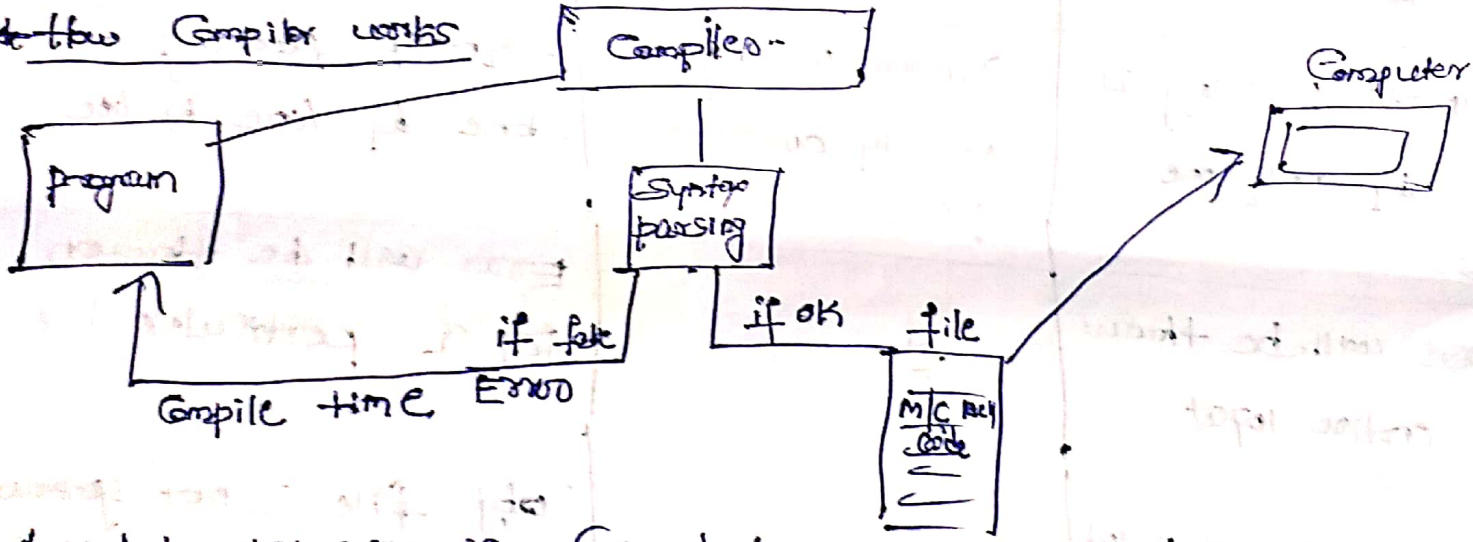
1 JS First look

1) program :- set of instructions that we are going to give for a Computer to perform a particular tasks.



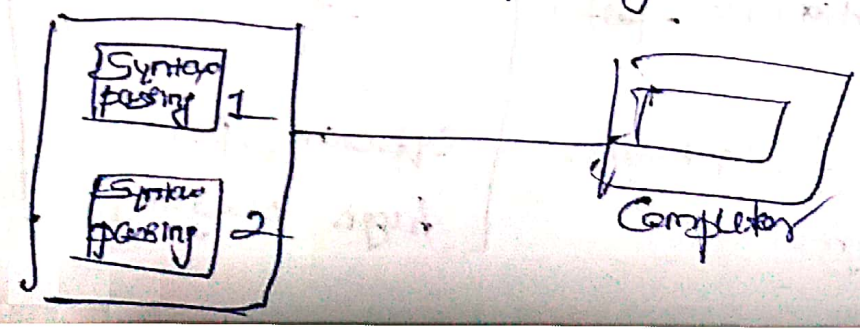
* How Compiler works



* whole program is Compiled at one shot

* How Assembler works

III[ler] to Compilers but Syntax passing works in 2 diff steps



* How interpreter works

Like Compiler but Syntax passing done by line by line

No file will generate directly translated Code will sent to Computer

Difference b/w Compiler, Interpreter, Assembler

Compiler	Assembler	Interpreter
Converts high level programming language to M/C level	low M/C level to M/C level	High level to M/C level
Syntax parsing is done by only one pass	Syntax parsing is done by one or 2 pass	Syntax passing is done by line by line
Errors will be throw for entire input		Error will be thrown for a particular line
obj file is generated after Compilation of Compilation		obj file is not generated instead the translated Code will directly fed to Computer
Execution is faster		Slower than Compiler
Efficient for huge pgm	efficient for huge program	for small piece of Code

(2)

J.S.F

* We can execute our JS code in 2 platform
  1) Browser 2) out of browser i.e Node JS

1) Browser bcz it has a interpreter Cold JS Engine
   for e.g Chrome browser has V8 JS engine
   Mozilla has spider monkey
   ~~I.E Safari~~ has JS core
   I.E has Chakra

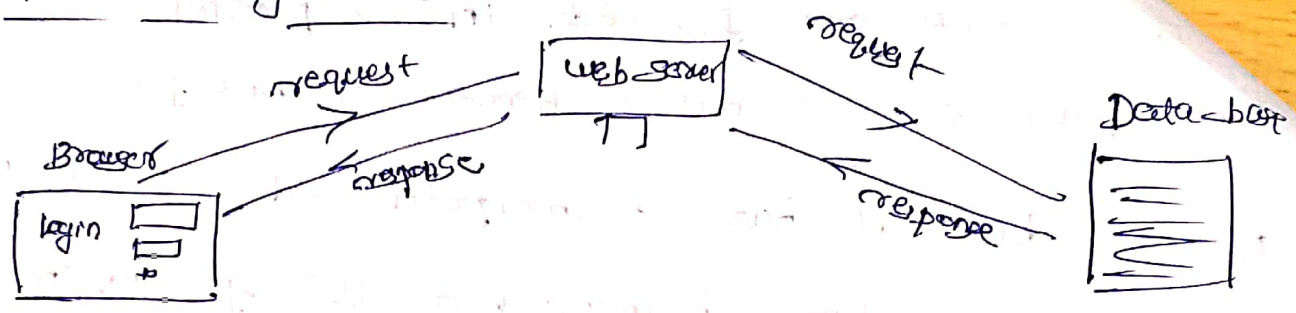
2) Node.JS platform

3) We can embed JS in HTML by using
   <Script> </Script> tags
   <Script src=" "> </Script> tags

* Java Script
* Its a Scripting language
* Using JS one can Create dynamic web pages / interactive web pages
* Developed to perform Client-Side validation
* Interpretation & execution happens at run-time So called Scripting language
* Uses interpreter for Conversions So called interpreted language

* Client Side validation :-

Take a login Scenario



* we are programming in browser end itself to Check basic validation)

* History of J.S.

* J.S was developed at Netscape (old name of firefox) by brenden Eich at 1995

* Brendan Eich already had a Code called mocha and he re-write the Code & named it as Spider Monkey

* Initial official name was livescript and later it was named as javascript

Java vs J-S

Compiler is used for execution	Interpreter is used
used mainly for Server Side validation	Used for Client Side validation

* Libraries built by using J.S

Libraries :- Libraries are used to simplify a complex task. i.e J.S performs task in bulk code, the same task can be performed with simpler codes size by adding functionality to it is called Libraries

Libraries built by using J.S

1) JQuery 2) _JS 3) Loadash 4) Bootstrap

* **Framewoks** :- Collection of libraries which improves the functionality of J.S

e.g 1) J.S along with node.JS is used for server side programming

2) react native is used to develop mob application in android and ios

3) React JS is used to develop web apln

4) Electon.JS is used to develop Standalone appln
e.g Calculator, paint etc

5) Angular JS used to develop Single page application
e.g. Gmail, Maps

6) React VR (vertual reality)

7) TensorFlow used to develop M/c learning algos

* Fullstack JS

* For Client Side or browser Side we can used Core JS
  Core JS is also called vanilaJS
* For web Server Side we can used Node JS
* For data base Server we can used Mongo DB and CouchDB
* Applications that uses JS
  * Ebay * Youtube * paypal * Smart watches
  * google Maps * Gmail

* Tokens :- They are the smallest unit of program
There are 5 types of tokens
1) Keywords 2) Identifiers 3) Literals 4) operators 5) Separa

1) Keywords :- words with predefined meaning in it. developers developing the language they defined it
  e.g var, if, else, for etc.
* There are about 34 keywords
* we should use keywords in lower case only

2) Identifies :- programmers defined words are called
  e.g. variable name, fun name

* Rules
  * keywords can not be used as identifiers
  * No special characters are allowed except $ and _
  * can be Alphanumeric but we can not start with number

* Literals :- values used while coding are called literals
e.g numeric literals, String lit's, Bool lits

* operators :- used to perform mathematical operations (+, -, *, /, %)

* Sp. operators :- (;, {}, (), [] ... etc)

* variable :- named memory location where we store the data

3 Steps to Create variable

declaration → var a
initialization → a = 10
utilization → Console.log(a)

* Arrays :- Continuous memory allocation

* Declaration syntax

var varname = []; e.g var cars = ['BMW' 'Audi' 'Tata']

* Initialization of array

Syntax:
varname [index] = value;
index starts from 0

e.g for array

var Cars = [ ]
Cars [0] = 'BMW'
Cars [1] = 'Benz'
Cars [2] = 'VW'

* utilization syntax

Console.log (arr [index]);

declaration & initialization together

var cars = ['BMW', 'Benz', 'VW']

* if else

Syntax 1

```
if (Condition ){
    [ ]
} else {
    [ ]
}
```

Syntax 2 : when there is only one statement

```
if ( ) {
    [ ]
}
```

```
if (condition)
    [ ]
else
    [ ]
```

* Switch Case :-

Syntax

```
switch (expression) {
    case value1 :
        [ ]
        break;
    case values :
        [ ]
        break
    default :
        [ ]
}
```

* if v/s Switch

* For loop
* while loop
* Do while loop
* Break & Continue.

* if v/s switch

* both works fine for single values to compare
* Switch is not a good approach to go with the range

program execution in memory :-

* whenever we run our code a global execution context will be created
* JS engine will add a default global window object in context.
* Along with window object, default variable got created called this variable
* And at the global level window ≅ this
* JS when you run a code in JS both interpretation and execution occurs at run time
* It will create creation phase and execution phase

Creation phase

* Memory allocation for all the variables
* Memory allocation for functions
* Memory allocation for the code

In creation phase variable will create in memory with default value undefined

Execution phase

Line by line Execution

Assignment of values to variables

Executing functions

e.g.
```
var a;
a = 10
function fun(){
  console.log("Hello")
}
console.log(a)
```

* Type cast
  - → internal
  - → external

* Internal type cast

Boolean ("")
Boolean (null)
Boolean (undefined)
} False

e.g.
```
function msg (name) {
    name = name || "default name"
    Console.log ("Greeting the day" + name)
}
msg ( )
```

---

* Function :-

Advantages
1. Reusability

Syntax
```
function functionName ( ) {
    [          ]
}
functionName ( )
```

* Function Execution in memory

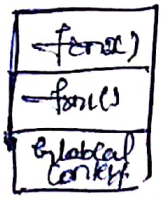
It is like to the Global Execution Context

* Scope

Functions only have an access to Global Execution Context not for another function Context

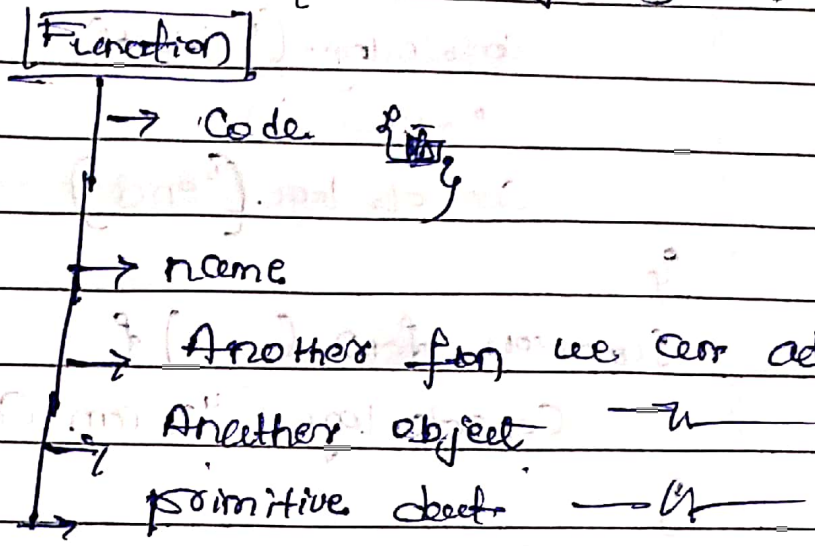
e.g.
```
var a = 10
function func ( ) {
    console.log (a)
}
```

Call point to stack



* Function is an object

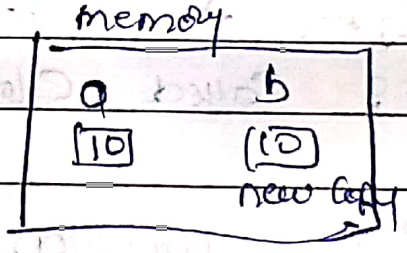
below properties will say that fun is an object



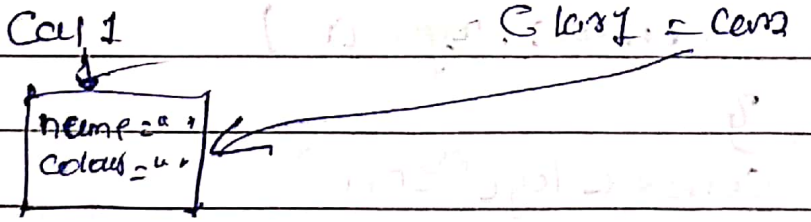
* Call by value Call by reference

* All primitive data type will follow : Call by value

```
a = 10
b = a
```



* All objects follow Call by reference



* Call back functions :-

* we can pass one fun as an input to another fun

* Calling received fun is called Callback fun

e.g.

```
function fn1(fn2){
    console.log("Start")
    fn2name()
    console.log("end")
}
function fn2(){
    console.log("I am Callback")
}
fn1(fn2)
```

* **Closures** :- The inner fun still having an access on the variable that declared in outer fun even after the execution of outer fun, this property is called Closure

e.g.

```
function outer(a){
    console.log("eg")
    function inner(){
        console.log(a)
    }
    console.log('end')
    return inner
}
inner = outer(a)
inner()
```

(7)

Functions :

Syntax - function functionname ( ) {

[statements]

}

functionname ( )

Type of fun

1) Function declaration / General fun

e.g. above

2) function expression :

var sample = function ( ) {

console.log (" msg ")

}

sample ( )

3) Anonymus fun

function ( ) {

Console.log ("msg")

}

4) Immidiatly invocable function expressions

(function ( ) {

console.log ("msg")

})( )

classmate
Date
Page

* Objects :- Collection of properties or can certify which containg primitive property, for property, and other object

* Creating and accessing object

1) By using a keyword new object()
e.g var car = new object() => object got created
To add properties to object
car.name = "BMW" } dot notation
car.colour = "Blue"
or
car["name"] = "TOYOTA" } bracket notation
car["colour"] = "white"
Create object inside the object

2) By using object literals { }
var car = { name : 'BMW",
colour : "Blue",
Model : "M5"
}

* - Objects, for and -this
* In Global Context this = window
e.g console.log(this).

* Inside the for also value of this = Global

(8) e.g.

```
function Sample(){
    console.log(this)
}
Sample()
```

* value of this = object when it is used inside object

e.g.

```
var Car = { name: "VW"
            colour: "white"
            fxn = function(){
                console.log(this)
            }
```

* Supper there 2 (outer and inner fxn) in object

```
var Car = { name: "VW"
            Colour: "Red"
            fxn: function(){
                Console.log(this);
                function fun2(){
                    console.log(this)
                }
                fun2()
            }
```

→ In above code the value of this in inner fxn is window

* To overcome the above situation, we have to initialise a variable and assing the value this to it

e.g. var car = {name: "VW"
colour: "colours"
fun: function() {
var self = this
console.log(self
function fun2() {
console.log(self)
}, fun2()
}

* call(), apply(), bind()

* every function in JS has 3 defined methods call(), bind() and apply()

1) bind() = It will create a new copy of fun and set the value for this, the value of this is equal to object that we pass as argument to bind(object) and finally it will return the function

we need to call fun explicitly for bind() bcz it is returning fun

e.g. for Call(), bind() and apply()

```
var person = { fname : 'Pavan'
               Lname: 'Deshpande'
               fullname : function () {
                   return this.fname + " " + this.lname
               }

function fun1 ( ) {
    console.log (this.fullname ())
}

fun = fun1.bind (person)  or  fun = fun1.bind (person, 'Hi', 'Hello')
fun2()

fun1.call (person)  or  fun1.call (person, 'Hi', 'Hello')

fun1.apply (person)  or  fun1.apply (person, ['Hi', 'Hello'])
```

* function borrowing by using Call() and apply ()

```
e.g var person = { fname :
                   lname :
                   fullname :
                 }
var p2 = { fname :
           lname :
         }
```

p2 object has no fullname for, we can borrow it from person

person.fullname.call(p2)

or

person.fullname.apply(p2)

* function Currying

function mul(a, b){
    log(a*b)
}

x45 = mul.bind(this, 5)

x45(4)

x45(5)

* Inheritance :— Copying properties and functions of one object to another object is called inheritance

* JS uses prototypal inheritance

* Every object in JS will have a property called __proto__ and every object, for, Array, is inherited from base Object

e.g p2.__proto__ = person

For in loop :- used to get properties of an object

e.g var person = { fname: "", lname: "", fullname: function() { console.log(this.fname + " " + this.lname) } };

var p2 = { getaddress: function() { return "India" } };

p2.__proto__ = person;

for (var prop in p2) { console.log(prop + " " + p2[prop]); }

* Object.Create() :- using this we can Creat an empty object with prototype object

e.g var p2 = Object.Create(person)

* Function Constructors :- They are the way using which we can Create an object

* We can pass args while Creating objects using fun Constr

e.g. P.T.O

e.g. function (fname, lname)

function person (fname, lname) {

this.fname = fname

this.lname = lname

}

var p1 = new person ('ABC', 'xyz')

var p2 = new person ('PQR', 'UVW')

* Prototype property :-

- All fns in JS have property called prototype
- Prototype is accessable only for objects that are created by function Constructor
- we can modify i.e. add the properties by using prototype property

e.g person.prototype.getfullname = function () {

return this.fname + this.lname

}

* Built in function Constructor

- Number
- String
- Date

e.g. var a = new Number (3)

var b = new String ("Hello")

* we can also add new fns to built in fns Constructor

* JSON :- JavaScript object Notation

* Standard means of data exchange.
* more efficient than the xml.

To Convert JS object → JSON

JSON = JSON.Stringify (Person)

To convert JSON → JS object

JSobj = JSON.Parse (JSON)

* Thread :-

1) Thread is an execution instance which will have its own Process time and memory

2) Any application opened in desktop / mobile will work as a thread

3) xty threading means executing xle threads together

4) Core JS is a single threaded & Synchronous

5) But browser is xty threaded

J.S Advanced book

Name of the Experiment :..... Date : 03/01/2021

Experiment No. :..... Experiment Result :.....

Predefined objects

String object

String methods

1) charAt(index) => returns the character at specified index

e.g var str = "Hello"

str.charAt(0) => H

str.charAt(90) => empty

str.charAt() => H

2) charCodeAt(index) => returns the ascii value of specified index

e.g str.charCodeAt(0) =>

str.charCodeAt(90) => NaN

3) Concat(value) => Joins the value with string

e.g str.Concat("hi") => Hellohai

str.Concat(true) => Hellotrue

str.Concat(Hi) => Reference Error

4) indexof(str) => Returns the first occurence index value of specified str

str.indexof('l') => 2

str.indexof('z') => -1 negetive index means no character

str.indep('l',3) => 3

str.lastindexof('l') => 3

str.lastindexof('l',3) => 2

5) localeCompare(String) => compare 2 strings if 2 strs are same it will return 0 if 1st str is greater than second negetive will be returned, and vice versa 2 string greater

6) SubString (startindex, endindex) ⇒ return subString endindex is exclusive

Str.SubString() ⇒ Hello
Str.SubString(0) ⇒ Hello
Str.SubString(0,5) ⇒ Hello

7) Slice(startindex, endindex)

8) Split() ⇒ returns array
Str.Split()

9) SubStr(start, length) ⇒ Same as SubString

10) toUpperCase() 11) toLowerCase 12) toString() 13) valueOf()

Experiment No. : ..... Experiment Result : .....

ES6 features

1) var, let, const 2) Arrow function 3) Array destructuring

* Array destructuring

* let arr = [1, 2, 3, 4, 5]
  let alpha = [a, b, c, d, e]

  var [a, b, c] = [1, 2, 3] arr

* Skipping elements in blw

  var [d, , e] = arr
  [10, 30]

* Using rest parameters (...rest)

  var [f, g, ...rest] = arr ⇒ f = 1, g = 2, rest = 3, 4, 5
  var [h, i, ...xyz] = arr

* Merging 2 arrays

  let newarray = [...arr, ...alpha]

* Object destructuring

var Car1 = { name : "BMW"
colour : "Blue" }

var Car2 = { name : "Benz"
Colour : "white" }

let { name, Colour } = Car2

let { name : Carname , Color : Carcolor } = Car2

* Template String [Backtic String] `${ }`

e.g "Hello procdeep
How are you"

* without using \n and + we can achive Concatination and next line

e.g var name = 'procdeep'
var fullname = "My name is ${name}";

e.g var a = 10
var b = 20
The Sum of ${a} and ${b} is ${a+b}

* Class

* Earlier to Create object we were using object literal syntax like
var person = {name : 'Mohan'};

* As most of the languages use Class Concept it was added to JS as well

* Syntax: Class ClassName
{
Statement
}

* Inside Class no need to write main method

* No need to use function keyword inside the Class while writing fn

* Inside a Class we can write only fns and Constructors

* We cannot Create variable inside a Class

* We cannot Creat object inside Class

* We can Create Static and non-Static fns inside Class

* To access non-Static fns we used objects and to access Static fns we used Class name with dot operator

* No Concept of function overloading.

Name of the Experiment :..... Date : | | | | | | | | |

Experiment No. :..... Experiment Result :.....

Page No. : 03

eg. Class Sample {
func() {
Console.log("Msg")
}
Static func2() {
console.log("Msg2")
}
var obj_sample = Sample new Sample();
obj_Sample.func()

Sample.func()

* how Create a Constructor in Class

eg Class Demo {
Constructor() {
Console.log("Msg")
}

* Promis :-

Syntax var varname = new Promise (callback fun)
Callback fun resolve, reject

eg var Promise = new Promise (function (resolve, reject) {
let x = 2
if (x == 2) {
resolve("success")
} else {
reject("failed")

promise.then (function(msg) {
log(msg)
} catch (function(msg) { log("msg") })

Inspire

* promise.all ([array of promise]). then (function() { })

checks all the promise fulfilled

* promise.race([ ]). then ( )