Javascript Part

**\*Why was javascript created?**

When javascript was invented, it was not called “javascript”, instead, mocha, there are some reasons for the invention of the javascripts:

1. Mocha is created to m ake the web become more dynamic, simple, accessible to non web developer.
2. Java was not suited for the type of audience that would consume Mocha: scripters, amateurs, designers. Java was just too big, too enterprisy for the role.
3. Mocha was meant to be the scripting companion for Java

\*What was the problem javascript attempts to address?

A: JavaScript enables interactive [web pages](https://en.wikipedia.org/wiki/Web_page) and thus is an essential part of [web applications](https://en.wikipedia.org/wiki/Web_application). The vast majority of [websites](https://en.wikipedia.org/wiki/Website)use it.

\*Is the language a reaction to a previous language or a replacement for another language?

A: JavaScript is a reaction to language java.

\*Does the language have any particularly unique features?

Yes, There are four powerful features make javascript:

* 1. Browser support

(use javascript, you don't have to use any plugin at all. This is because all browsers have accepted javascript as a scripting language for them and provides integrated support for it)

* 1. Can be used on client side as well as on server side
  2. Functional programming language(function can be assigned to to either variables or another functions)
  3. the way javascript handles objects and inheritance is bit different from conventional object oriented programming languages like Java.

How are name spaces implemented in javascript?

A: There are two types of implementation in javascript about name spacing: Static programming, which includes the direct assignment, object literal notation, module pattern and dynamic implementation which includes 1) Supply a Namespace Argument 2) Use this as a Namespace Proxy

How is the name space used?

A: the concept of namespaces, whereas exists in other programming languagedoes not exist in JavaScript. To add insult to injury, everything you create in JavaScript is by default **global**. To solve this problem we can create a **single global object** for your app and make all functions and variables properties of that global object.

What types does the language support?

* A: It supports String, Integer, Float, Boolean, Array, Object, NULL
* Are both reference and value types supported?

A: Primitives are passed by value, Objects are passed by "copy of a reference".

Can new value types be created?

A: No

Defining a class:

A: The first way to define a class is using the class declaration, like:

Class Jeff{}

The second way to define a class is class expression, like var Jeff = class Jeff{}

\*Defining new instance:

A: class one{} var one = new one();

\*javascript initializing/constructing:

A: first way: <script type="text/javascript">

function init(){

}

</script>

Second way: <script type="text/javascript">

window.onload=function(){

}

</script>

\*Javascript deinitializing:

Class deinitializing{  
 deinit(){ }

}

\*instance reference name in data type (class)

This, function Person(data) {

data = $.extend({

name: "",

age: 0

}, data);

this.name = data.name;

this.age = data.age;

}

* Getters and setters…write your own or built in?

A: The get syntax binds an object property to a function that will be called when that property is looked up, The set syntax binds an object property to a function to be called when there is an attempt to set that property. It’s build in

* Backing variables?

I can’t find

* Computed properties?

With computed property names, instead of having to create an object and then add properties to it using bracket notation, what we can do is return a new object, and using object literal notation, put brackets around the key. Now, whatever the variable represents is going to be added as a property on the object.

function objectify (key, value) {

return {

[key] : value

}

}

Interfaces / protocols

* What does the language support?

A: JavaScript is a prototype-based language, meaning that object properties and methods can be shared through generalized objects that have the ability to be clone and extended, which is known as prototypical inheritance. It supports both mutable and immutable prototypes, whereas classical inheritance only supports immutable classes. It also inherit mutiple prototypes, java only inherits one classes.

* What abilities does it have?

A: In prototype language we only have objects instead of classes,2) Functional features can be used in conjunction with create.3) Prototypal inheritance is simple and easy to understand.

* How is it used?
* It is used like the following
* function Rectangle(width, height) {
* 2 this.height = height;
* 3 this.width = width;
* 4 }
* 5
* 6 Rectangle.prototype.area = function () {
* 7 return this.width \* this.height;
* 8 };
* 9
* 10 var rect = new Rectangle(5, 10);

console.log(rect.area());

Inheritance / extension:

A: 1) Inheriting properties: Javascript objects are dynamic “bags” of properties , they have link to the prototype object, when trying to access the property of an object, the property will not only be sought on the object but on the prototype of the object

2) Inheriting methods: JavaScript does not have "methods" in the form that class-based languages define them. In JavaScript, any function can be added to an object in the form of a property.

Reflection:

What type of reflection does this language supports?

A: It supports reflect.get, reflect.defineProperty, reflect.deleteProperty, reflect.has, reflect.getOwnProperty

How is the reflection used?

A: Traditionally, the reflection is used to load modules that list an assembly, and create an instance of it, without the reflection, this process will probably take very long time.

Memory management:

How is it handled?

A: javascript, unlike lower level language uses malloc() and free() to manage the memory,allocates strings when things are created and automatically freed when they are not used anymore.

1. How does it works?

A: It has a lifecycle with three stages: Allocation: including value initialization, function calls. Using the allocated memory, and release the allocated memory when it is not needed anymore

1. Garbage collection?

A: The main notion of the garbage collections algorithm really relys on the notion of reference. An object is said to reference other object if the former has an access to the latter one.

1. Automatic reference counting:

A: Automatic reference counting(ARC) is a garbage collection technique with a simple algorithm: every object stores a special values which is increased everytime a new reference is assigned the address of this object. In javascript, to manage reference counting special value, we use “allocate” function which

Initialize reference counting flag top zero, and also provides actual allocation of this project on the heap.

Function allocate(obj) {

var allocateaddress = heap.length;

obj.\_rc\_ = 0;

heap.push(obj);

return allocAddress;

}

Comparison of reference and value:

How are values compared?

A: When we compare the string: we use “===” to check if they are equal, if we compare value, we kust use double equal sign “==”

Null/nil references:

Which does the language support?

A: It supports null references.

Does this language have any features handling null/nil?

A: Yes, we can use “||” operator to handle null.nil“var studentname = studentname.firstname || “unknown/null”.

Error and exception handling:

A: In javascript, we use try/catch statements for error and exception handling, the try statement allows people to define a block of code to be tested for errors and while it being execute. The catch statement. There are two types of try...catch clause, the first one is called the unconditional clause and the second one is called conditional clause:

Examples of unconditional try...catch clause:

<script>

Try { addlert(“welcome guest!”); }

Catch(err) { document.getElementById(“demo”).innerHTML = err.message;}

</script>

The second one is called conditional catch clauses:

<script>

Try {

XXXXX();

}catch (e if e instanceof Typerror){ }

catch (e if e instanceof Rangerror){}

catch(e if e instanceof EvalError){}

catch(e) { logMyErrors(e); }

</script>

\*lambda expression, closure, or functions as types

A: lambda function in javascript is also called the arrow functions expression has a shorter syntax than a function expression and does not have the its own this, arguments, super, or new targets. The following shows an example of lambda expression in javascript:

var mustudent = [

‘enterfirstname’,

‘enterlastname’,

‘enterstudentd’ ]

Console.log(mystudent.map(mystudent=>mystudent.length)

* + Implementation of listener and eventhandler

To add the listener, we implements like this, element.addEventListener(event, function, userCapture),

Where the first elements is the type of the event, the second element is the function want to call when the events occurs, the third elements is the boolean value whether the events is bubbling or captured. To remove the listener, we just need to do document.removeEventListener(“mousemove”, the name of the function).

How is singleton design pattern implemented in javascript?

1. The singleton pattern is a software design pattern that restrict the instantiation of one object. In javascript, the singleton object is implemented as an anonymous function, which is executes immediately by wrapping it in brackets followed by two additional brackets followed by two additional brackets. The getinstance method is Singleton’s gatekeeper. Example:

Var instance.

Function createInstance(){ var obj = new Object(“hello”); return obj}

Return { getinstance : function(){ if(!instance){instance = createinstance();} return instance;}}

Can it be made thread safe?

A: Because javascript is single-threaded in the context of broswer, it’s relatively safe

Can javascript be lazily instantiated?

A: Yes, it can, the example I showed above is lazily instantiated

Procedural Programming:

Does javascript support procedural programming?

A: Yes

Functional Programming:

Does javascript support functional programming?

A:Yes

How is mutithread achieved in javascript?

A:

|  |
| --- |
|  |
|  |  |