Christian Vielma

Interview Study Guide – Code Snippets

V1

Disclaimer

I (Christian Vielma) have personally created this guide for personal use, to be used as a reference for technical interviews, work, and study. Since I believe this document might provide value to others, I have decided to make it public.

Most of the content on this document have been added from different sources, from freely available resources on the Internet, to books, with additional added content from me. I made my best effort to provide full credit to the original sources, and in no way I'm trying to take advantage of improper quotations. If you believe I have made an invalid reference to a resource, please let me know and I'll fix it.

I provide this guide as-is, with no additional guarantees. I'm also releasing it under CC-Atribution-ShareAlike License, so you are free to extend it and update it with proper attribution.

Enjoy!

Table of Contents

Programming Languages Syntax

Ruby

- Dynamic, OO.
- Variable names are "labels" each one can be used to hold references to multiple objects.
- Arrays are declared using [x,y,z] syntax.
- Methods can be called without ().
- == and equals() work opposite than in Java.
- Uses garbage collector and objects are strongly typed.
- Return is optional.

```
#no ";" required at end of line.
require 'library' #similar to import
class SubQueue < Queue #class definition. < means left side is subclass of right.
  attr_reader :ultimot #specifying a getter for each variable. attr_accesor get/set
  @@id=0 #@@ = class variable @=instance variable
    def initialize #class constructor. Cannot overload methods.
      self.var_one = method(args) #self instead of this. var_one a method being
called with argument "method(args)".
    end #closing block

@pc.each do |i| #do loop syntax
      i.espera += t
    end

if @numclie == 0 then #if-else-then syntax: no () or {} required.
    ...
    else
      ...
    end

if cliente.nil? then return cliente end  # ? returns a value

private #methods defined after these word are private
for i in 0..pcolas.size do #for loop syntax
    end

puts "Hello #{name}" #prints "Hello " concat with the variable name.

case a
when 1..5
  puts "It's between 1 and 5"
when String
  puts "You passed a string"
else
  puts "You gave me #{a} -- I have no idea what to do with that."
end
```

JavaScript

- Dynamic, prototype-based (classes not present), dynamic typing, first-class functions. Multi-paradigm: oo, imperative and functional.
- There are moments where it seems the code is asynchronous, this happens when some functions wait (background executing, outside JS sandbox, remote calls, etc). The mechanism is to solve this is through the use of callback functions.
- JavaScript is single-threaded.
- http://stackoverflow.com/questions/2190850/create-a-custom-callback-in-javascript
- Syntax is similar to C and Java.
- Updating the prototype without overriding previous (see http://www.sitepoint.com/5-typical-javascript-interview-exercises/)
  `String.prototype.repeatify = String.prototype.repeatify || function(times) {`
-

From: http://stackoverflow.com/questions/13455134/javascript-doesnt-seem-to-wait-for-return-values

```
// Added a callback arg ---v
function geocode(geocoder, callback) {
    //do geocoding here...
    geocoder.geocode({ 'address': address }, function (results, status) {
        if (status == google.maps.GeocoderStatus.OK) { callback(results);}  // Call the
callback
         else { callback(null);} //Call the callback with a flag value there was an error
   });
}
//Then, instead of using it like this:
var results = geocode(someArgHere);
if (results) { doSomething(results); }
else { doSomethingElse(); }

//You call it like this:
geocode(someArgHere, function() {
    if (results) { doSomething(results); }
    else { doSomethingElse(); }
});
```

Recommended to use like this:

```
if (callback && typeof(callback) === "function") { callback(); }
```

Class definition in JS (from: http://www.phpied.com/3-ways-to-define-a-javascript-class/)

```
function Apple (type) { //option 1
    this.type = type;
    this.color = "red";
}
Apple.prototype.getInfo = function() {
    return this.color + ' ' + this.type + ' apple';
};

var apple = { //option 2
    type: "macintosh",
    color: "red",
    getInfo: function () {
        return this.color + ' ' + this.type + ' apple';
    }
}
```

How to define a variable as a function:

```
 var functionOne = function() {}
```

AJAX

http://www.w3schools.com/ajax/

The main element is XMLHttpRequest() (different for IE <6)
var xmlhttp = new XMLHttpRequest();
xmlhttp.open(“GET”, “ajax-info.php”, true); //method (GET or POST), url, async.
// async = true => onreadystatechange listener function callback
// async = false => next instructions below call
xmlhttp.send();

ReadyState can be:
0: request not initialized
1: server connection established
2: request received
3: processing request
4: request finished and response is ready
StatusResponse:
200: "OK"
404: Page not found

Complete:
```
function loadXMLDoc()
{
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
}
```

PHP

- Dynamic.
- '' are simple string, whereas “” are string that can have variables or escaped characters. “.” concats strings
- As in Ruby, arrays can have string ids: $_POST['msg']
- == compares equality. === compares equality and type.
- && is AND and can be written “and”. Same to “or” or ||.
- The require() function is identical to include(), except that it handles errors differently. If an error occurs, the include() function generates a warning, but the script will continue execution. The require() generates a fatal error, and the script will stop.
- The require_once() statement is identical to require() except PHP will check if the file has already been included, and if so, not include (require) it again.
- Weakly typed (1+”1” == 2, 1 . “1” = “11”).

```
<?php //code starts with <?php and ends with ?>
      $name = “Neo”; //”;” required at the end of each line (unless last line).
                     //$ defines a variable.
      echo “Hello $name”;  //echo prints variables
      $name1=$name2=$name3= “MyName”; //Multiple assigns possible
      if ($age >= 21) { //if syntax
      }
      elseif ($age >=18) {
      }
      else {}
      switch($day){ //switch case syntax
            case 1: … break;
            case 2: … break;
            default: … break;
      }
      foreach ($artists as $a) {} //foreach syntax

      //while, do-while, for, identical to java.

      isset($_POST['submit']); //checks if variable was assigned.

      $pizzaToppings = array('onion', 'tomato', 'cheese', 'anchovies', 'ham', 
'pepperoni'); //array definition
      $fruits = array('red' => 'apple', 'yellow' => 'banana', 'purple' => 'plum', 
'green' => 'grape');
      print_r($pizzaToppings);

      array_push($pasta, 'tagliatelle'); //pushes an element to the end
      array_pop($pasta); //removes an element from the end

      $colors = explode(', ', $str); //if  $str= red,blue,yellow; creates an array 
with the elements.
      $str = implode(' and ', $colors); //and the other way.
```

```
?>
```

Prolog

General purpose logic programming language. Untyped.

Rules:

```
   Head :- Body.  %Head is true if Body is true.
```

Rules call to predicates (goals): , denotes conjunction goals and ; disjunction.

Facts:

`cat(tom).` **%are written in lowercase and ended with ".". is the same as cat(tom) :-true.**

```
      ?- cat(tom)  %Is tom a cat? True
      ?- cat(x) %What things are cats? Tom
```

```
mother_child(trude, sally).
 father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
?- sibling(sally, erica).
Yes

?- write('Hello World!'),nl. %nl means new line.
Hello world!
true.

%Quicksort
partition([], _, [], []).
partition([X|Xs], Pivot, Smalls, Bigs) :-
    (   X @< Pivot ->
        Smalls = [X|Rest],
        partition(Xs, Pivot, Rest, Bigs)
    ;   Bigs = [X|Rest],
        partition(Xs, Pivot, Smalls, Rest)
    ).

quicksort([])     --> [].
quicksort([X|Xs]) -->
    { partition(Xs, X, Smaller, Bigger) },
    quicksort(Smaller), [X], quicksort(Bigger).
```

%other

`nth0(?Index, ?List, ?Elem)` **% true when Elem is the Index'th element of List. Starting at 0.**

Haskell

General purpose purely functional language, non-strict semantics (lazy evaluation) and strong static typing.

```
--Hello World
module Main where
 main :: IO ()
main = putStrLn "Hello, World!"

---Different factorial implementations
-- Type annotation (optional)
factorial :: Integer -> Integer
-- Using recursion
factorial 0 = 1
factorial n | n > 0 = n * factorial (n - 1)
-- Using recursion but written without pattern matching
factorial n = if n > 0 then n * factorial (n-1) else 1
-- Using a list
factorial n = product [1..n]
-- Using fold (implements product)
factorial n = foldl1 (*) [1..n] -- foldl1 Type: (a -> a -> a) -> [a] -> a   .takes
the first 2 items of the list and applies the function to them, then feeds the
function with this result and the third argument and so on.
f(g(x)) = (f.g)(x) --math to haskell

--Quicksort
quick_sort :: Ord a => [a] -> [a] --The Ord class is used for totally ordered
datatypes.
quick_sort []     = []
quick_sort (x:xs) = (quick_sort [a | a <- xs, a < x]) ++ [x] ++ (quick_sort [b | b
<- xs, b >= x])

--Example using let assigning variables
calcBmis :: (RealFloat a) => [(a, a)] -> [a]
calcBmis xs = [bmi | (w, h) <- xs, let bmi = w / h ^ 2, bmi >= 25.0]
```

C/C++

- Operators can be overloaded.
- Memory management (allocation): Static (at compile time, word *static*), Automatic (declared with class name), Dynamic(using *new*/*malloc()* and *delete/free()*), Garbage Collection (with specific libraries).
- Templates are equivalent to Java generics.
- Encapsulation (public, protected, private, can provide access to "friend" classes).
- Multiple inheritance.
- Java Interfaces = abstract base class (class with only virtual functions).
- C++ is a statically typed, compiled, general purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- Types: bool, char, int, float, double, void, wchar_t. Modifiers: signed, unsigned, long, short.
- Syntax similar to Java (while, for, if, case (expression has to be integral or enumerated)).

```
#include <iostream>
using namespace std;
// Base class
class Shape
{
   public:
      void setWidth(int w)
      {  width = w;
      }
      void setHeight(int h)
      { height = h;
      }
   protected:
      int width;
      int height;
};
// Derived class
class Rectangle: public Shape
{
   public:
      int getArea()
      {
         return (width * height);
      }
};
int main(void)
{
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;
   return 0;
}
//Only C from here. From: http://www.cprogramming.com
#include <stdio.h>
int main()
{
    int this_is_a_number;
   printf( "Please enter a number: " ); //print to console
    scanf( "%d", &this_is_a_number ); //read from console
    printf( "You entered %d", this_is_a_number );
    return 0;
}

//Defines a function that returns an int
int mult (int x, int y)
{ return x * y; }

/* one pointer, one regular int */
int *pointer1, nonpointer1;
/* two pointers */
int *pointer1, *pointer2;
//Pointers example
int main()
{
    int x;            /* A normal integer*/
    int *p;           /* A pointer to an integer ("*p" is an integer, so p
                       must be a pointer to an integer) */

    p = &x;           /* Read it, "assign the address of x to p" */
    scanf( "%d", &x );          /* Put a value in x, we could also use p here */
    printf( "%d\n", *p ); /* Note the use of the * to get the value */
}

//Assign/free memory
int *ptr = malloc( sizeof(*ptr) );
free( ptr );

ptr = &structure; /* Yes, you need the & when dealing with
                           structures and using pointers to them*/
    printf( "%d\n", ptr->x );  /* The -> acts somewhat like the * when
                                   it is used with pointers
                                    It says, get whatever is at that memory
                                   address Not "get what that memory address
                                   is"*/
```