Requirements should come first.

Does not keep keys in order.

So I have done this solution.

I need to do the string solution.

Personas:

Interviewer

Product Owner

Me the Software Engineer

Interviewer: I want you to make a JSON prettifier

/\*

JSON Types:

String - red

Number - blue

Array - normal

Object - normal

Boolean - purple

Key - A subset of string

How to style:

Key - Is always inlined and on a newline

Values - Is always inlined and after a key

Object/Array - First bracket inlined and last bracket newline close

Color the String, Number, Boolean, and Key

Requirement: Take a JSON string and return html that makes the JSON string pretty

Solution: Compare the types make a element and color it. Use stylesheet to set the styles.

Edge Case - Is the exception to the rule. You need extra code to fix something that won’t normally occur.

Recursion Base Case - All keys are exhausted

Edge Cases - If Array use [, if object use {, if array ignore key

Array instanceOf Object and Array instanceOf Array return true

Brute Force (Recursion Required):

Try JSON.parse to turn a string in a JavaScript object

If the JavaScript object is not a JSON object throw error

Else Do the processing

ProcessObject

End If

processObject(curObject, resultArr)

Var start = “”

Var isObject = true

If curObject instanceOf Array

Start = “<span class=”bracket”>[</span><br/>”

End = “]”

Else If curObject instanceOf Object

Start = “{<br/>”

End = “}”

Else isObject = false

resultArr.push(start);

// if value then <span class= value type> with break <br/>

// otherwise do a for in loop through the object

if start is an array bracket then omit key otherwise print key then call process to get correct value because the value could be an object

resultArr.push(End)

\*/

function getPrettyJson(json) {

// Just let the parse throw to the calling point

json = JSON.parse(json);

var html = [];

Var tablHtml = ‘’;

processObject(curObject, html, tabHtml);

return html.join(‘’);

}

function processObject(curObject, html, tabHtml){

var start = ‘’;

var end = ‘’;

var isObject = true;

var isArray = false;

if(curObject instanceOf Array){

start = ‘<span class=”squareBracket”>[</span><br/>’;

end = ‘<span class=”squareBracket”>]</span>’;

isArray = true;

}

else if(curObject instanceOf Object) {

start = ‘<span class=”curlyBrace”>{</span><br/>’;

end = ‘<span class=”curlyBrace”>}</span>’;

}

else {

isObject = false;

}

if(start)

html.push(start);

// If primitive

if(!object){

html.push(tabHtml)

switch(typeof curObject){

case ‘string’:

html.push(‘<span class=”string”>’);

break;

case ‘boolean’:

html.push(‘<span class=”boolean”>’);

break;

case ‘number’:

html.push(‘<span class=”number”>’);

break;

}

html.push(curObject + ‘</span><br/>’);

}

else { // Otherwise object loop and repeat process

// Tab over for all key value pairs since all objects have a nesting nature

tabHtml += ‘&nbsp’;

for(var key in curObject){

// Add the key if the object is not an array

if(i!sArray){

html.push(tabHtml + ‘<span class=”key”>’ + key + ‘:</span><br/>’);

}

// Rerun the processObject to account for values that can be objects/arrays

processObject(curObject[key], html, tabHtml);

}

}

if(end)

html.push(end);

}