JO

Project Proposal

Programming Languages and Translators - Fall 2014

Team

```
[
   {
      "Name" : "Abhinav Bajaj",
      "UNI" : "ab3900",
      "Role" : "System Architect"
   },
   {
      "Name" : "Arpit Gupta",
      "UNI" : "ag3418",
      "Role" : "Language Guru"
   },
   {
      "Name" : "Chase Larson",
      "UNI" : "col2107",
      "Role" : "Manager"
   },
   {
      "Name" : "Sriharsha Gundapp",
      "UNI" : "sg3163",
      "Role" : "Verification & Validation"
   }
]
```

1. Introduction

1.1 Motivation

JSON or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used as lightweight data interchange format to transmit data between a server and web application. JSON is also emerging as a preferred format in "NoSQL" databases. While languages like Python and Java have libraries to handle JSON data, they are not a native aspect of the language. JSON is presently a data format, rather than something fundamental to the language, like the object of an object oriented language, or the function of a functional language. With rise of trends in Big Data, Internet of Things, No-SQL databases, we believe that our language can be provide a platform for building applications for these technologies with ease.

1.2 Language Description

JO is simple yet powerful language to handle and manipulate JSON data. The language will treat JSON object as first class citizens and provide built-in functions that operate on these objects. These basic functions can be used to define complex libraries and applications like merging JSON, finding diff in JSON, SQL like queries on JSON objects. Our language attempts to facilitate any data operati ons by handling a lot of the business logic of handling JSON and their manipulations under the hood, and allowing the programmer to use JSON in a more native and intuitive way.

2. Language Specification

2.1 Primitive Data Types

Number	Basic Numeric type (int or float) e.g 23, 34.5
String	Sequence of characters
Null	empty data type
Bool	Boolean Value - true or false

2.2 Complex Data Types

Json	Data type to store JSON e.g. {"name":"harris"}
List	Ordered data type of primitive/complex data types. e.g ["apple",45, {"name":"harris"}

2.3 Operators

+	Concatenation, works on any data type arguments, returns list Example 1. 5 + 2 = [5 , 2] 2. [5, 2] + 3 = [5, 2, 3] 3. JsonA + JsonB = [JsonA, JsonB]
-	Usage : *A* - *B*, works when *A* is a Json or List Removes attributes from A which matches with B Valid Data types - • Json - Json • Json - String • List - List • List - String • List - Json • List - Number Example

	1. { {"name": {first:chase, last:larson}}, {subject : "plt"}, marks : [2,3,4] } - { "name": {first:chase, last:larson}, marks: [2,3,4]} = { subject : "plt"} 2. { "name": {first:chase, last:larson}, subject : "plt"} - { "name": {first:abhinav, last:larson}} = { "name": {first:chase, last:larson}, subject : "plt"} 3. { {"name": {first:chase, last:larson}}, marks: [2,3]} - "name" = {marks: [2,3]} 4. ["able", "barista", "carrie"] - ["barista", "carrie"] = ["able"] 5. ["able", "barista", "carrie"] - "barista" = ["able", "carrie"]
[]	1. [ ] access values for attributes. only work on JSON objects • a = json1['Name'] , returns the value at the attribute. • json1['Name'] = 'Arpit' , stores value 'Arpit' for attribute 'Name' 2. [ ] - constructs a new list
==	Compare two same data types. Returns true if their values match else false
!=	Compare two same data types. Returns false if their values match else true
=	Assignment Operator
.	Calls function on an object
{}	Constructs a Json object

2.4 Mathematical Operators

All Mathematical Operators are only valid for Type Number.

Operator	**Description**	**Example**
++	Addition	2 ++ 2 results in 4
--	Subtraction	2 -- 2 results in 0
**	Multiplication	2 ** 2 results in 4
//	Division	2 // 2 results in 1
>	Greater Than	2 > 1 results in true
<	Less Than	2 < 1 results in false

2.5 Logical Operators

All Logical Operators only valid for Data type Bool.

Operator	**Description**	**Example**
&&	Logical And	if A and B are true, (A && B) is true
| |	Logical Or	if A or B are true, (A || B) is true
!	Logical Negation	if A is false, !A is true

2.6 Membership Operators

Operator	**Description**	**Example**
in	Results in true if variable is in given list	A in B: results in true if variable A is found in list B.
not in	Results in true if variable is not in given list	A not in B: results in true if variable A is not found in list B

2.7 Built-In Functions

Function	**Description**	**Example**
type(<arg>)	returns a String of the data type of a variable	type(5) returns "Number", type({}) returns Json

JSON.typeStruct()	returns a JSON showing the *type structure* of json.	If JSON myJSON = { "name": { "first": "chase", "last": "larson" }, "age" : 23 }
		myJSON.typeStruct() results in { String : { String: String, String: String}, String: Number }

JSON.join(<args>)	if 2 JSONs have the same *type structure*, they are joined with the values of the key being concatenated into a list	JSON A = { "name" : { "first" : "chase" } }
		JSON B = { "name" : { "first" : "arpit" } }
		JSON.join(A, B) results in:
		{ "name" : {"first": ["chase", "arpit"] }}

2.8 Control Flow

Statement	**Description**	**Usage**

if...else	Executes the if statement if the given condition is true.
	Otherwise executes the else statement.
	The condition to the if statement must be on the same line as the keyword "if".
	The code to be executed must be on a new line.
	Else statement is optional.
	The If...Else must end with the word "end"
		if <condition>
		    <statement>
		else
		    <statement>
		end

for	Iterative Expression must be of the form: <dataType> in <List>
	Iterates over the list and executes the block of code
		for <iterative expression>
		    <statement>
		end

2.9 Other Syntax

Statement	**Description**	**Usage**
func	Function Declaration. Functions require a return statement. End signifies the end of the function.	func <FUNC_NAME> (<args>) <statement> return <arg> end
/* ... */	Comments. Requires " /* " at the beginning and " */ " at the end of the commented section.	/* This is a comment */
EOL character	End of line character (\n or \r\n) is used to signify the end of the code block to be executed.	

Code Example

This example demonstrates merging two JSON objects.

Whenever there are same attributes (field) , with value not of *Json* type , then combine them to form a List, otherwise Merge the *Json* objects recursively. This should be considered as the merge equivalent of deep-copy.

JSON A =

```
{
             "Name": {
                                    "First":"Arpit",
                                    "Last":"Gupta"
                        },
            "School": "Columbia",
            "Age": 22,
            "Courses": [
                        "PLT",
                        "ML"
                        ]
}
```

JSON B =

```
{
            "Name": {
                                    "First":"Abhinav",
                                    "Last":"Bajaj"
                        },
            "School": "Columbia",
            "Age": 18,
            "Courses": [
                                    "CV",
                                    {
                                                "Audit": "Algorithms"
                                    }
                             ]
}
```

```
func Merge(A,B)
       if  A.typeStruct() != B.typeStruct()

              C = {}
              for attr in A.attrList()
                     if attr in B.attrList()
                            if type(A[attr])=="JSON" && type(B[attr]) == "JSON"
                                   C[attr] = Merge(A[attr] , B[attr])
                            else
                                   C[attr] = A[attr] + B[attr]
                            end
                     else
                            C[attr] = A[attr]
                     end
              end
              for attr in B.attrList()
                     if (attr not in A.attrList())
                            C[attr] = B[attr]
                     end
              end
              return C
       else
              return JSON.join(A,B)        /* In-built function */
       end
end
```

Output:

```
{
       "Name":{
              "First":[
                     "Arpit",
                     "Abhinav"
              ],
              "Last":[
                     "Gupta",
                     "Bajaj"
              ] },
       "School":"Columbia",
       "Age":[
              18,
              22
            ],
       "Courses":[
              "PLT",
              "ML",
              "CV",
              {
                     "Audit":"Algorithms"
              } ]
}
```