Traductor de JSON a YAML

Integrante	Libreta	Correo
Fachal, Matías	154/15	mfachal@dc.uba.ar
Gomez, Horacio	756/13	horaciogomez.1993@gmail.com
Gonzalez, Juan	324/14	gonzalezjuan.ab@gmail.com

Introducción

El objetivo de este trabajo es crear un traductor que tome cadenas válidas en el lenguaje *JSON* y las traduzca al lenguaje YAML, teniendo en cuenta que *JSON* soporta claves repetidas dentro de los diccionarios y YAML no; en cuyo caso la traducción no será posible.

Gramática:

Tomamos como base la gramática definida en la página oficial de JSON. La misma fue modificada a medida que se fue avanzando en el trabajo por conveniencia. La resultante fue:

- BEGIN -> VALUE
- VALUE -> str
- VALUE -> number
- VALUE -> OBJECT
- VALUE -> ARRAY
- VALUE -> true
- VALUE -> false
- VALUE -> null
- OBJECT -> {}
- OBJECT -> {MEMBERS}
- MEMBERS ->PAIR_AND_SEPARATOR MEMBERS
- MEMBERS -> PAIR
- PAIR_AND_SEPARATOR -> PAIR ,
- PAIR -> KEY VALUE
- KEY -> str :
- ELEMENTS -> VALUE
- ELEMENTS -> VALUE , ELEMENTS
- ARRAY -> [ELEMENTS]
- ARRAY -> []

Notar que por claridad hemos resumido los strings bajo 'str' y los numeros bajo 'number'. Antes que nada, es importante aclarar que *PLY* es una herramienta para parsers *LALR* que además, brinda la posibilidad de arrojar una advertencia a la hora de generar las tablas de *action* y *go_to* para una gramática con conflictos. Esto nos permitió verificar que la gramática resultante fuera *LALR* ya que no se generó ningún conflicto en *PLY*.

Lexer

Un lexer es un programa que se encarga de, dado un texto como input, transformarlo en "tokens" válidos. Decimos por eso que es el encargado de "tokenizar" la entrada en los nodos terminales de nuestra gramática. Los tokens utilizados son los siguientes: * BEGIN_ARRAY, representando al [* BEGIN_OBJECT, representando al { * END_ARRAY, representa el fin de un array,] * END_OBJECT, } * NAME_SEPARATOR, representa el : * VALUE_SEPARATOR, , * QUOTATION_MARK, " * FALSE, directamente la string `false` * TRUE, `true` * NULL, `null` * DECIMAL_POINT, . * DIGITS, representando los dígitos del 0 al 9 * E, `e`, como exponente * MINUS, –, como resta o prefijo de un negativo * PLUS, `+`, similarmente * ZERO, `0` * UNESCAPED, representa cualquier caracter no escapado * ESCAPE, representa al caracter `\`, el lexer entra en el estado en que considera la string como escapada * REVERSE_SOLIDUS, en un estado escapado, representa al caracter `\` * SOLIDUS, `/`, similarmente * BACKSPACE_CHAR, `b`, ídem * FORM_FEED_CHAR, `f`, ídem * LINE_FEED_CHAR, `n`, ídem * CARRIAGE_RETURN_CHAR, `r`, ídem * TAB_CHAR, `t`, ídem * UNICODE_HEX, en un estado escapado representa a las strings del tipo `uXXXX`, las cuales actualmente no se interpretan especialmente.

Parser

Una vez que el lexer genera la cadena de *tokens*, solo resta hacer la traducción al lenguaje *YAML* por medio del parser, ya que este nos permite entender la estructura de la cadena. Para este fin lo que hacemos es sintetizar una lista de strings de manera que cada string se corresponda con una línea del *YAML* resultante. En las producciones que anidan, debemos incrementar la indentación de todos los strings involucrados ya que así garantizamos que al final cada línea quede indentada como corresponde. En las producciones que permiten agregar elementos de diccionarios o listas en un mismo nivel debemos chequear si el elemento que que se está agregando debería anidarse (como sería en los casos de estar agregando un diccionaroi o una lista), ya que en esas situaciones debemos mostrar el elemento en una línea nueva, no al mismo nivel que su "padre". También tuvimos que chequear en la producción que permite generar un diccionario, que las claves provenientes de los pares no se repitieran, debido a que esto no está soportado en *YAML*. Al encontrar dos claves iguales lanzamos un error.

Ejemplos

*prueba es un json mas grande, con varios niveles de anidamiento y arreglos dentro *prueba2 es un json normal con varios niveles de anidamiento *prueba3 prueba los arreglos en distintos niveles *prueba4 es la prueba brindada por la cátedra *prueba5 pone a prueba _doble quotes_ para los valores que poseen '\n'

Requisitos de software

- Python 3.5+
- Ply 3.11+

La cadena que se desee traducir deberá estar almacenada en un archivo. Para traducirla se deberá

ejecutar el siguiente comando: `python3 json.py nombredelarchivo`