## **. Importaciones**

* **import ply.lex as lex y import ply.yacc as yacc** Importan las librerías PLY para crear el lexer (analizador léxico) y el parser (analizador sintáctico).
* **import re** Importa la librería de expresiones regulares, útil para validar patrones de texto (como emails, fechas, etc).
* **import tkinter as tk y otros de tkinter** Importan la librería para crear la interfaz gráfica (ventanas, botones, cuadros de texto, etc).
* **from PIL import Image, ImageTk** Permite mostrar imágenes en la interfaz.
* **import os, sys** Para manejar archivos, rutas y otras utilidades del sistema operativo.

## **2. CLAVES\_VALIDAS**

* Es una **lista de todas las claves válidas** que puede tener tu JSON.
* Se usa para validar que no haya campos inventados o mal escritos.

## **3. LEXER (Analizador Léxico)**

### **a. Diccionario reserved**

* Relaciona **palabras clave** (como "nombre\_equipo", "link", etc.) con el **nombre del token** que usará el lexer.

### **b. Lista tokens**

* Enumera **todos los tipos de tokens** que el lexer puede reconocer.
* Incluye tanto los tokens de datos (STRING, INTEGER, EMAIL, etc.) como los tokens de estructura (LLAVE\_IZQ, COMA, etc.) y los tokens de palabras reservadas.

### **c. Ignorar espacios y tabulaciones**

python

Copy Code

t\_ignore = ' \t'

* El lexer **ignora** espacios y tabulaciones.

### **d. Funciones de tokens especiales**

Cada función t\_ALGO define cómo reconocer un tipo de token: algunas de ellas descrita con regex(expresion regular)

* **t\_newline**:  
   Cuenta los saltos de línea para llevar el control de la línea actual (útil para mostrar errores con línea exacta).
* **t\_EMAIL**:  
   Reconoce emails entre comillas dobles y les quita las comillas.
* **t\_FECHA**:  
   Reconoce fechas en formato "YYYY-MM-DD" y les quita las comillas.
* **t\_URL**:  
   Reconoce URLs que empiezan con http(s):// o ftp:// y les quita las comillas.
* **t\_FLOAT**:  
   Reconoce números con decimales (positivos o negativos) y los convierte a float.
* **t\_INTEGER**:  
   Reconoce números enteros (positivos o negativos) y los convierte a int.
* **t\_BOOL**:  
   Reconoce los literales true y false y los convierte a booleanos de Python.
* **t\_NULL**:  
   Reconoce el literal null y lo convierte a None de Python.
* **Tokens de símbolos**:  
   Reconocen los símbolos de estructura JSON: comas, llaves, corchetes, dos puntos.

### **e. t\_STRING**

* Reconoce cualquier string entre comillas dobles.
* Si el valor del string está en el diccionario reserved, lo convierte en el token correspondiente (por ejemplo, "link" pasa a ser CLAVE\_LINK).
* Si no, lo deja como STRING.

### **f. Manejo de errores léxicos**

* **ultimo\_error\_lexico\_linea**:  
   Guarda la última línea donde hubo un error léxico, para no repetir el mismo error varias veces en la misma línea.
* **t\_error**:  
   Si encuentra un carácter inesperado, agrega un mensaje de error a la lista errores y salta ese carácter.

## **PARSER**

## **¿En qué consiste todo esto?**

* **Esta parte del código define cómo se lee y se interpreta cada pedacito del texto JSON**:
  + Distingue entre claves, valores, símbolos, y detecta errores de escritura.
  + Convierte los datos del texto a tipos de Python (números, booleanos, None, etc).
  + Prepara todo para que el parser pueda trabajar con los datos ya "limpios" y clasificados.

basicamente arma el autómata aceptor que dice si acepta o no lo que le llegue  
  
  
**Funciones del parser (las que empiezan con p\_)**

* **p\_json** Es la regla principal. Dice: "Un JSON empieza y termina con llaves y adentro tiene elementos".  
  + Llama a chequear\_obligatorios para ver si están los campos que sí o sí tienen que estar.
  + Devuelve el árbol de datos.
* **p\_elementos** Junta todos los pares clave-valor que hay dentro de un objeto.
* **p\_par** Analiza cada par clave:valor.  
   Acá se hacen los chequeos semánticos:  
  + ¿La clave existe y está bien escrita?
  + ¿El valor tiene sentido? (edad, cargo, estado, email, link, etc)
  + Si algo está mal, lo anota en la lista de errores.
* **p\_clave** Reconoce todas las posibles claves válidas.
* **p\_objeto** Arma un diccionario con los pares clave-valor.  
   Si es un equipo, un integrante, un proyecto o una tarea, llama a la función que genera el HTML correspondiente.
* **p\_lista** Reconoce listas (cosas entre corchetes).
* **p\_valor** Reconoce los posibles valores: string, número, booleano, null, fecha, email, url, objeto o lista.
* **p\_valores** Junta todos los valores de una lista.
* **p\_error** Si encuentra algo que no entiende, lo anota como error sintáctico.

### **para las reglas se tiene en cuenta lo siguiente una comilla simple o doble (' o "):**

* Se usa para **reglas de una sola línea**.

def p\_lista(p):

'lista : CORCHETE\_IZQ valores CORCHETE\_DER'

# código...

### **Tres comillas (simples o dobles) (''' o """):**

### **Se usan para reglas de varias líneas o para escribir varias alternativas en una sola regla.**

def p\_elementos(p):

'''elementos : par | elementos COMA par'''

# código...

si se mezcla el

### **Funciones de chequeo y utilidades**

* **chequear\_obligatorios** Se fija que estén todos los campos obligatorios en cada objeto (por ejemplo, que un equipo tenga nombre, integrantes, etc).
* **url\_valida** Se fija que los links tengan el formato correcto (por ejemplo, que no tengan barras de más).

### **Funciones de HTML**

* **html\_equipo, html\_integrante, html\_proyecto, html\_tarea** Transforman los datos del JSON en un HTML bonito para mostrar en la interfaz o guardar como archivo.

### **Funciones de ayuda**

* **imprimir\_tokens** Muestra todos los tokens que encontró el lexer (útil para depurar).
* **analizar\_sintaxis** Llama al parser para analizar el texto y devuelve el árbol de datos.
* **imprimir\_arbol** Muestra el árbol de datos de forma ordenada (útil para ver cómo quedó todo estructurado).

## **¿Cómo se ejecuta el código?**

1. **El usuario toca "Analizar".**
2. El programa toma el texto y lo pasa al **lexer** (analizador léxico).
   * El lexer separa el texto en tokens.
   * Si encuentra algo raro, lo anota como error léxico.
3. Los tokens van al **parser**.
   * El parser arma la estructura (el "árbol") y chequea que todo tenga sentido.
   * Si encuentra errores de estructura, los anota como errores sintácticos.
   * Si encuentra datos mal puestos (por ejemplo, un link mal escrito), los anota como errores semánticos.
4. Si no hay errores graves, el programa genera el **HTML** para mostrar los datos de forma visual.
5. La interfaz muestra los errores (si hay) y el HTML generado.

## **¿Qué hace el botón "Analizar"?**

* Toma el texto del JSON.
* Llama a las funciones de análisis (lexer y parser).
* Junta todos los errores encontrados.
* Si todo está bien, muestra el HTML.
* Si hay errores, los muestra para que el usuario los corrija.

## **¿Cuántas "pasadas" hace el programa?**

**Hace dos grandes pasadas:**

1. **Análisis léxico:** El texto se divide en "piezas" llamadas tokens (palabras, números, símbolos, etc).
2. **Análisis sintáctico y semántico:** Se fija si esas piezas están bien ordenadas y si tienen sentido (por ejemplo, que la edad sea un número mayor a cero, que los links sean válidos, etc).