Instituto Tecnológico de Costa Rica

Escuela de Computación

Ingeniería en computación

Compiladores e Intérpretes

Grupo 2

Profesor: Ericka Marín

Análisis léxico

Adrián López Quesada, 2014081634

Josué Arrieta Salas, 2014008153

Cartago

Lunes 18 de abril

**Índice**

[Análisis de resultados 3](#_Toc448690657)

[Casos de pruebas 4](#_Toc448690658)

[Prueba 1 : Tokens de operadores 4](#_Toc448690659)

[Prueba 2 : Tokens de palabras reservadas 5](#_Toc448690660)

[Prueba 3 : Comentarios de línea y de bloque 6](#_Toc448690661)

[Prueba 4: Código exhaustivo 7](#_Toc448690662)

[Prueba 5: Strings y Chars 8](#_Toc448690663)

[Prueba 6: Números y flotantes 9](#_Toc448690664)

[Prueba numero 7: identificadores 10](#_Toc448690665)

[¿Cómo compilar y correr el Scanner? 12](#_Toc448690666)

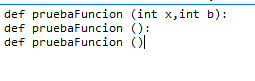
# Análisis de resultados (Josué)

Se toman en cuenta las reglas de Python para la realización de este análisis de resultados.

|  |  |
| --- | --- |
| **Objetivo** | **Porcentaje de éxito (100%)** |
| Programa recibe código fuente escrito en Python y analiza el archivo. | 100% |
| Se listan errores léxicos encontrados (por línea y el error). | 100% |
| El programa ante un error léxico ser recupera de este y no despliega errores en cascada. | 100% |
| Se listan los tokens encontrados (por tipo, línea del código fuente donde se presentan y cantidad de ocurrencias de cada token en cada línea). | 100% |
| La lista de tokens encontrados es ordenada en orden alfabético | 100% |
| Tokens de operadores son procesados correctamente. | 100% |
| Tokens de literales (strings, enteros, flotantes, char) son procesados correctamente. | 100% |
| Tokens de identificadores son procesados correctamente. | 100% |
| Tokens de palabras reservadas son procesados correctamente. | 100% |
| El programa identifica comentarios (de bloque o de línea) y omite los tokens dentro de ellos. | 100%: se ha de mencionar que si un comentario de bloque no se cierra, y se alcanza el EOF deberá notificar error. |
| Números aceptados en cualquier formato de Python (binario, hexadecimal y octal). | 100%: este objetivo es importante ya que se estará realizando un scanner lo más parecido posible al lenguaje Python real. |
| Se lograron características en un buen scanner. | 100%: este objetivo es lo que diferencia un buen scanner de uno malo. Casos como mañana, 999hola o ho+as son tomados como incorrectos. |
|  |  |
|  |  |

# Casos de pruebas

## Prueba 1 : Funciones (Adrián)



Resultado:



Ya que no se habían definido “x” cantidad de funciones

Al definirlo para “x” cantidad de funciones:



Porque falta el ultimo “:”

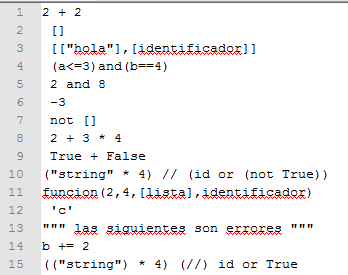
Al arreglar ese error en la prueba:



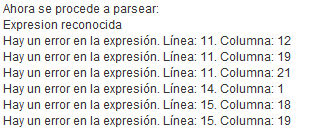
## Prueba 2 : Expresiones (Josué)

Objetivo: probar todo tipo de expresiones en el lenguaje MyPython.

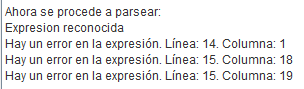
Se tiene el siguiente documento de texto de entrada:



Se espera errores en la línea 14 y en la línea 15, sin embargo las demás son expresiones válidas. Se tienen los siguientes resultados (incorrectos):

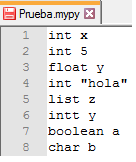


Se muestra que hay un error en la línea 11 debido a que las funciones no se contemplaron como expresiones. Se aplica un arreglo y al correr el mismo archivo se tienen los siguientes resultados (correctos):

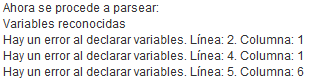


## Prueba 3 : Declaración de variables (Josué)

Objetivo: probar todo tipo de declaración de variables en el lenguaje MyPython. Se tiene el siguiente código de entrada:



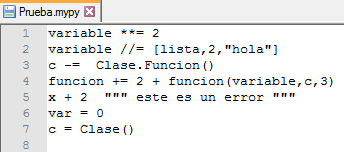
Se espera que haya errores en la línea 2, 4 y 6; y que el compilador se logre recuperar de estos y parsee hasta el final. Se obtuvieron los siguientes resultados (correctos):



Se ha de mencionar que el error de “línea 6” aparece como “línea 5 y columna 6” ya que el error fue encontrado al final de la línea 5. La línea 6 es “ignorada” ya que no hizo match. Esto ocurre para cualquier recuperación de error en donde el primer token de la línea es el incorrecto y por lo tanto reporta el error al final de la línea anterior.

## Prueba 4: Asignaciones (josué)

Objetivo: probar diferentes tipos de asignaciones posibles en el lenguaje MyPython. Se tiene el siguiente código de entrada:

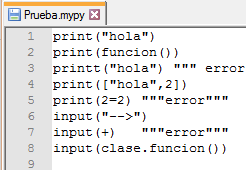


Se espera un error en la línea 5 ya que “x + 2” no es una asignación sino una expresión, sin embargo se espera que el compilador se recupere de este error. Se obtuvieron los siguientes resultados (correctos):

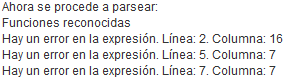


## Prueba 5: Print e Input (Josué)

Objetivo: probar la estructura de las funciones print() e input() predefinidas. Se tiene el siguiente código de entrada:



Se espera que hayan errores en las líneas: 3,5 y 7; y que además se recupere del error y siga parseando. Se obtuvieron los siguientes resultados (correctos):



## Prueba 6: While (Adrián)

## Prueba numero 7: If (Adrián)

# 

## Prueba numero 8: For (Adrián)

## Prueba numero 9: Try (Josué)

## Prueba numero 10: Break - Continue (Adrián)

## Prueba numero 11: OOP (Josué)

# ¿Cómo compilar y correr el Scanner? (Adrián)

La parte esencial para compilar el proyecto es agregar el archivo “jflex-1.6.1.jar” como JAR a las librerías que tiene acceso. De esta manera se pueden utilizar todas las funcionalidades del Jflex. En caso de duda, por favor seguir las instrucciones del video siguiente: https://www.youtube.com/watch?v=w-KfjJdRas8 cerca del minuto 10, tomando en cuenta que se importa como JAR, no como librería. Otra manera es cargar el proyecto a netbeans el cual ya esta ya tiene el JAR agregado. Además, se necesita que el archivo flex.lexer esté presente en el src y aunque no es necesario para compilar se recomienda utilizar la ubicación actual del archivo de pruebas.

Aunque no es de relevancia, el proyecto se desarrolló bajo el IDE de Netbeans. Para correr el scanner se puede utilizar el .jar “Scanner ejecutable” para hacer las pruebas con el archivo “Pruebas.MYPY” que se encuentra dentro de la carpeta ejecutable. Se debe modificar el nombre del archivo por motivos de envio por correo, solo se le debe quitar la “a” al final del nombre de la extensión .jar