**NOTACIÓN DE LA SINTAXIS: Extended Backus-Naur Form -EBNF-**

La sintaxis se refiere a la estructura de los elementos de un lenguaje en función de su tipo. Las notaciones Backus-Naur Form y su versión extendida son ampliamente usadas para describir la gramática de un lenguaje, siendo esta un conjunto de reglas (también llamadas **reglas de producción**), haciendo referencia únicamente al conjunto de cadenas válidas de acuerdo a las reglas de su gramática.

* **Una descripción BNF/EBNF es una lista de reglas.**
* **Las reglas se utilizan para definir símbolos con ayuda de otros símbolos.**

TIPOS DE SÍMBOLOS EBNF

* **TERMINALES:** cadenas escritas entre comillas, destinados a usarse tal como son.
* **NO TERMINALES:** nombres que hacen referencia a otras cosas.

NOTACIÓN DE LAS REGLAS GRAMATICALES

* **Lado izquierdo:** describe el nombre de un no terminal para definirlo
* **=:** separa el lado izquierdo del derecho. Se lee “se define como”.
* **Lado derecho:** es la definición del no terminal.

COMPOSICIÓN DE NO TERMINALES

* Secuenciación: combinación de uno o más terminales y/o no terminales.
* Elección: el operador | indica que las partes separadas por él son opciones.
* Opcional: el operador [ ] indica que lo encerrado en él es opcional.
* Repetición: el operador { } indica que puede repetirse cero o más veces.
* Agrupación: el operador ( ) agrupa elementos y establece precedencia.

**SINTAXIS PARA PARSER-PY**

La sintaxis está inspirada en el documento de definición de la práctica 2 para el curso de Lenguajes Formales y de Programación ss 2023.

Expresiones regulares: usadas en el analizador léxico, abarcan parte de la definición gramatical para EBNF. SON EXPRESIONES REGULARES, NO USAN NOTACIÓN EBNF.

BLOQUES DE CÓDIGO: En la fase inicial de planteamiento, representan una lista dentro de un token de tipo “ sintáctico ”

En el segundo planteamiento considero que un bloque de código debe tratarse como un **contenedor** que guarda dentro de sí una lista de tokens sintácticos. Este contenedor también puede ser un token especializado con la lista. El contenedor de código es usado por elementos específicos de la sintaxis como **if, else, funciones, etc.**

Las sentencias dentro de los bloques de código deben evaluarse individualmente con autómatas finitos deterministas.

* Expresiones regulares para constantes:

dígito = 0-9 +

decimal = dígito + “.” dígito +

cad doblecomilla = “ .\* “

cad simplecomilla = ‘ .\* ‘

true = “true”

false = “false”

1. EXPRESIONES: Se considera una expresión todo aquello que retorne un valor.

expresión = variable # variable declarada?

expresión = función # void es una función

expresión = arreglo { arreglo } # arreglo n- dimensional

expresión = diccionario

expresión = constante

# variables declaradas: definir su tipo (puede ser uno de los siguientes)

constante = dígito | decimal | cad doblecomilla | cad simplecomilla | true | false

arreglo = “[“ constante | variable “]” # variable declarada?

diccionario = “{” [ par clave valor ] “}” # puede estar vacío

par clave valor = identificador “:” constante # constante es terminal

tupla = “(“ { constante } “)” # son inmutables

op log = “ and “ | “ or “

op neg = “ not ” # uso dif de “and” y “or”

op comp = “ == “ | “ != ” | “ >= “ | “ <= “ | “ < ” | “ > “

expresión condicional = [ op neg ] expresión condicional

expresión condicional = “ true “ | “ false “

expresión condicional = expresión condicional op log expresión condicional

expresión condicional = ( expresión ) op comp ( expresión )

operador ternario = expresión “if” expresión condicional “else” expresión

1. DECLARACIÓN DE VARIABLES Y ASIGNACIÓN:

Estructura: identificador operador\_asignacion expresión

variable = identificador

op asign = “ = ” | “ += ” | “ -= ” | “ \*\*= ” | “ /= ” | “ //= ” | “ %= ” | “ \*= ”

asignación = identificador { “ , ” identificador } op asign expresión { “ , “ expresión }

# comprobar -> al menos la misma cantidad de expresiones que identificadores

1. CONDICIONALES:

if = “ if ” expresión condicional “ : “ bloque de código [ elif { elif } ] [ else ]

elif = “ elif ” expresión condicional “ : ” bloque de código

else = “ else ” bloque de código

# Un operador ternario es una expresión puesto que devuelve un valor

# Una expresión condicional es una expresión puesto que devuelve un valor (v/f)

1. CICLOS:

FOR: Un bucle for en Python es una estructura de control que se utiliza para iterar sobre una secuencia.

for **variable** in **secuencia** :

las secuencias pueden ser:

* **arrays/listas** -> frutas = [‘manzana’,’pera’]
* **rangos** -> range(10)
* **cadenas de caracteres**
* **diccionarios**
* **tuplas** -> tupla = (x, 1, “texto”) -> son inmutables

for = “ for ” identificador “ in “ secuencia bloque de código [ else ]

# La sentencia **else** es opcional y se ejecuta si la ejecución del for no se interrumpe con un break

secuencia = arreglo | rango | caddoblecomilla | cadsimplecomilla | diccionario | tupla

rango = “ range “ “ ( “ dígito [ “ , ” digito ] [ “ ,“ digito ] “ ) “

# El rango se puede crear con 3 argumentos: inicio, fin y tamaño del paso

# **También validar variable numérica, aquí solo se usó dígito por comodidad**

WHILE

while = “ while” expresión condicional “ : ” bloque de código

1. FUNCIONES / MÉTODOS

Bloques de código reutilizables que realizan una tarea específica. Las sentencias pertenecientes a ellos deben tener la misma identación.

función = “def” identificador “ ( ” [ identificador { “ , ” identificador } ] “ ) ” bloque de código [ return expresión ]

# verificar el tipo de expresiones que se pueden retornar

**(REFERENCIA: Bloques de código - documento de definición de la práctica 2 para el curso de Lenguajes Formales y de Programación ss 2023)**

Dentro de cada bloque existirán las siguientes sentencias de manera opcional:

* [expresiones](https://docs.google.com/document/d/1m3W2MMxTXrsazHo_bV1MLyfMtfUxJxnmlA5XXph63W0/edit#heading=h.mtfr1nhany8m)
* instrucciones
  + condicionales
    - [if](https://docs.google.com/document/d/1m3W2MMxTXrsazHo_bV1MLyfMtfUxJxnmlA5XXph63W0/edit#heading=h.n2ere2qzo8fv)
    - [operador ternario](https://docs.google.com/document/d/1m3W2MMxTXrsazHo_bV1MLyfMtfUxJxnmlA5XXph63W0/edit#heading=h.7ar1qwytvok0)
  + ciclos
    - [for](https://docs.google.com/document/d/1m3W2MMxTXrsazHo_bV1MLyfMtfUxJxnmlA5XXph63W0/edit#heading=h.k02liuh3vcf9)
    - [for-else](https://docs.google.com/document/d/1m3W2MMxTXrsazHo_bV1MLyfMtfUxJxnmlA5XXph63W0/edit#heading=h.ezvxc6stsc7u)
    - [while](https://docs.google.com/document/d/1m3W2MMxTXrsazHo_bV1MLyfMtfUxJxnmlA5XXph63W0/edit#heading=h.j61c3luo2kdi)

En la fase inicial de planteamiento, representan una lista dentro de un token de tipo “ sintáctico ”

En el segundo planteamiento considero que un bloque de código debe tratarse como un **contenedor** que guarda dentro de sí una lista de tokens sintácticos. Este contenedor también puede ser un token especializado con la lista. El contenedor de código es usado por elementos específicos de la sintaxis como **if, else, funciones, etc. (REFERENCIA A LA PARTE INICIAL - SINTAXIS DE PARSER PY).**

**POR DEFINIR:**

función - ya definido

bloque de código: lista enlazada con los “tokens” contenidos dentro

Estos tokens deben evaluarse normalmente con los autómatas finitos deterministas.

expresión condicional: **en expresiones** - terminarla |

* No se pueden limitar los diccionarios o arrays pues se puede hacer la siguiente expresión condicional {} == {}
* aceptar funciones que retornan true false

**IDENTIFICACIÓN DEL TIPO DE AUTÓMATA A USAR**

Al inicio de algunas sentencias en el código no se espera ningún tipo de símbolo en específico, por lo que pasar todos los autómatas uno a uno hasta identificar el tipo que debe ser usado no es muy eficiente. A pesar de que este analizador sintáctico no pretende ser súper eficiente si que se puede mejorar un poco la detección de código. Yo propongo el uso de **diccionarios** los cuales puedan disparar el uso de un autómata detectando el tipo de token inicial a procesar.

**AUTÓMATAS RECURRENTES**

Autómatas que pueden ser usados en distintos casos:

* rango
* arreglo
* diccionario
* par clave / valor
* tupla
* else
* expresión condicional -> todas las expresiones en realidad

**PREGUNTAS**

* En la sección de ciclos, en el uso de for se usa una función como secuencia, ¿eso es válido?