Análisis Sintáctico (Diseño). (2,5 puntos) En este ejercicio se quiere diseñar un analizador sintáctico para reconocer programas escritos en un lenguaje llamado PY. Este lenguaje se ha diseñado para expresar programas secuenciales imperativos con tipado dinámico, listas polimórficas y asignaciones paralelas. A continuación, se describen las capacidades sintácticas de PY.

El programa PY es una secuencia de instrucciones. Las instrucciones en PY son simples: (a) asignaciones, (b) eliminación de variable e (c) impresión de expresión por pantalla o compuestas: (d) condicional y (e) iteración.

PY define 4 tipos: entero, booleano, lista y un tipo especial llamado no-tipo. Las constantes enteras son las usuales (ej: ...`-1`, `0`, `1`...), las booleanas son `True` y `False`, la única constante de no-tipo es `None` (representa ningún valor) y las constantes listas son secuencias de constantes separadas por comas y delimitadas por corchetes externos (ej. `[]`, `[True,2,[3,4,5]]`).

La expresión entera PY, además de variables y constantes enteras, también incluye suma (ej. `x+1`), resta (ej. `x-1`), multiplicación (ej. `x*1`) y división (ej. `2/3`) de expresiones enteras además del uso una función predefinida `len` (longitud de una lista) (ej. `len(l)`).

La expresión booleana PY, además de variables y constantes booleanas, también incluye conjunción (ej. `a & b`), disyunción (ej. `a | b`) y negación (ej. `not b`) de expresiones booleanas. También se incluyen operaciones relacionales entre expresiones enteras: mayor (ej. `x > 5`), menor (ej. `x < 5`), mayor o igual (ej. `x >= 5`), menor o igual (ej. `x <= 5`) y las relaciones de igualdad (ej. `s == [5]`) y desigualdad (ej. `x != False`) para cualquier tipo de expresión.

La expresión lista, además de variables y constantes de tipo lista, también incluye concatenación de dos listas (ej. `[1,2]+[3,4,5]`), concatenación de una lista n veces (n entero y n >1) (ej. `[1,2]*3`), inserción de un elemento en una posición de la lista (ej. `s.insert(x+2,1)`), inserción de un elemento al final de la lista (ej. `s.append(x+2)`), eliminación del último elemento de la lista (ej. `s.pop()`), eliminación del primer elemento de la lista (ej. `s.popleft()`), copia de una lista (ej. `s.copy()`), inversión de los elementos de la lista (ej. `s.reverse()`) y conteo de las veces que ocurre un determinado valor en una lista (ej. `s.count(x+2)`). Las listas también se pueden expresar en PY por comprensión. El patrón de lista por comprensión es: `[expresion for variable que itera elementos in lista if expresion_booleana]` (ej. `[[x+1] for x in [1,2,3,4,5] if x>1]` expresa por comprensión la lista `[[3],[4],[5],[6]]`

La expresión entre paréntesis y las componentes de una lista también se incluyen entre las expresiones válidas de PY (ej. `(x+2)*5`, `s[i+5]`).
A continuación, se muestran algunos ejemplos de expresiones pertenecientes al lenguaje PY (expresiones correctas sintácticamente):

```
[3 for x in m if True], m+[1], [x,1,x+2], []*(4+1), x, m[i+1], [[3],[4],None], [[[3]]],
s.count(x*2),([1]+[9])*2, b & c | d, 8==x, x!=y | x>z, s[j] & not b[2], True, None, 10,
len(s.popleft())/7, -1, not not x, [ls.append(4),lt.count(1+3), len(x)], True & b,
x<=2, x[1[2]], [x for x in [8,5] if x<6], --10, [[x for x in [8,5] if x<6],7], x!=y==z,
x!=(True & b)==z, s.count(x)+2
```

A continuación, se muestran algunos ejemplos de expresiones no pertenecientes al lenguaje PY (expresiones erróneas sintácticamente):

```
[3 for x[i] in s if True], True+[1], [x 1 x+2], []*[1], x + None, s[], s[[3]], s[None],
s[True], [1,2].count(x*2), [1,2].append(), 1.append().append(), 2 & c, True>2, x>z<y,
len(3), len(True), len(None), None | True, len(len(s)), -True, not 5, [3 for x in s if
2], (True & b)>=z, s.insert(x,True), s.count(x)+[2]
```

No hay en PY ninguna sección dedicada a la declaración de variables. La variable se declara implícitamente en su primera asignación. La asignación en PY puede ser simple o múltiple. La simple asigna una sola variable (ej.`x = 2*5`), la múltiple asigna más de una variable en paralelo (ej.`x,s[y+1] = 2,5`). En PY se puede asignar valor `None` a una variable para representar ausencia de valor (ej. `x = None`). La componente de una lista puede actuar como variable (ej.`s[y+1] = 5`).
En PY, se puede eliminar una variable de la memoria del programa haciendo uso de la instrucción `del` (ej. `del x`).
La instrucción `print` permite imprimir por pantalla los valores de una secuencia de expresiones (ej. `print(x+1,y)`).
Las instrucciones compuestas en PY son: condicional e iteración. Ambas usan bloques de instrucciones. Un bloque es una secuencia de instrucciones delimitadas por el símbolo `:` y la palabra clave `end`
(ej. `: l = [] print(l) end`).

A continuación, se muestra un ejemplo de condicional PY:

```
if x > 3:
   print(x)
   end
elif y < 2:
   print(y)
   end
else:
   print(l)
   end
```

Las secciones `elif` y `else` son opcionales. La sección `elif` se puede repetir más de una vez. La sección `else` ocurre como máximo una vez. Si hay alguna sección `elif` entonces el condicional debe finalizar con la sección `else`.

A continuación, se muestra un ejemplo de iteración PY:

```
while i<5:
   print(x[i])
   i=i+1
   end
else:
   print(l)
   end
```

La sección `else` es opcional.

Mostramos finalmente un ejemplo de programa PY:

```
l,y,x=[1,2,3,4,5,6,7,8,9,10], 3, 2
if x > 3:
   print(x)    end
elif y < 2:
   print(y)    end
else:
   print(l)    end
i=0
x=[e for e in l if e/2==1]
while i<5:
   print(x[i])    i=i+1    end
```

A continuación, se propone un diseño **parcial** del analizador sintáctico (gramática independiente de contexto):

```
programa: (instrucciones)? ;

instrucciones: instruccion instrucciones
   | instrucción
   ;

instruccion: instruccion_simple
   | instruccion_compuesta
   ;
```

```
instruccion_simple: asignacion
   | eliminacion
   | impresión
   ;

instruccion_compuesta: condicional
   | iteración
   ;

asignacion: variables ASIG expresiones ;

variables: variable COMA variables
   | variable
   ;

variable:


expresiones: expresion COMA expresiones
   | expresión
   ;

eliminacion: DEL variable ;

impresion: PRINT PARENTESISABIERTO expresiones PARENTESISCERRADO ;

bloque: DOSPUNTOS instrucciones END ;

condicional:


iteracion:


expresion: expresion_entera
   | expresion_booleana
   | expresion_lista
   | NONE
   ;


expresion_entera:


expresion_booleana:


expresion_lista:
```

SE PIDE: Complete el diseño escribiendo el cuerpo de las reglas `variable`, `condicional`, `iteracion`, `expresion_entera`, `expresion_booleana` y `expresion_lista`. Trabaje de forma tentativa en folios aparte y, cuando concluya, complete el diseño en la hoja dada a tal fin. **Sea claro y legible. Evite tachaduras.**

Análisis Semántico (Diseño). (2,5 puntos) En este ejercicio se quiere diseñar un analizador semántico para reconocer expresiones con tipo erróneo en un *sublenguaje* de PY. Este sublenguaje sigue considerando los 4 tipos del lenguaje original: entero, booleano, lista y no-tipo.

A continuación, se enumeran las restricciones semánticas que el analizador debe considerar:

(Restricción 1) Una expresión con alguna de las siguientes formas: `-E`, `E1-E2`, `E1/E2`, `E1>E2`, `E1>=E2`, `E1<E2`, `E1<=E2` es errónea si `E` no es entero y `E1` o `E2` no es entero.

Ejemplos:
```
x<=2 siendo el tipo de x distinto a entero,
m-1 siendo el tipo de m distinto a entero,
2/x[3] siendo el tipo de x[3] distinto a entero.
```

(Restricción 2) Una expresión con alguna de las siguientes formas: `E1&E2`, `E1|E2`, `not E` es errónea si `E` no es booleano y `E1` o `E2` no es booleano.

Ejemplos:
```
b & c | d siendo el tipo de b, c o d distinto a booleano,
not not x siendo el tipo de x distinto a booleano,
True & b siendo el tipo de b distinto a booleano,
s[j] & not b[2] siendo el tipo de s[j] o de b[2] distinto a booleano.
```

(Restricción 3) Una expresión con alguna de las siguientes formas: `len(E)`, `E.count(D)` es errónea si `E` no es de tipo lista.

Ejemplos:
```
len(s) siendo el tipo de s distinto a lista,
s.count(x*2) siendo el tipo de s distinto a lista.
```

(Restricción 4) Una expresión de la forma: `E1+E2` es errónea a menos que `E1` y `E2` sean de tipo entero o de tipo lista.

Ejemplos:
```
m +[1] siendo el tipo de m distinto a lista,
m+1 siendo el tipo de m distinto a entero.
```

(Restricción 5) Una expresión de la forma: `E1*E2` es errónea a menos que `E1` y `E2` sean de tipo entero o `E1` de tipo lista y `E2` de tipo entero.

Ejemplos:
```
m*n siendo m de tipo lista y n de tipo lista,
1*m siendo el tipo de m distinto a entero.
m*n siendo m de tipo booleano y n de tipo lista,
```

(Restricción 6) Una expresión de la forma `V[E]` es errónea si `V` no es de tipo lista o `E` no es de tipo entero o la componente `V[E]` no existe. El índice del primer elemento de la lista es 1 (no 0 como en otros lenguajes).

Ejemplos:
```
m[2] siendo m de tipo distinto a lista.
m[i] siendo el tipo de i distinto a entero
m[i+7] siendo m de tipo lista e i de tipo entero pero
       inexistente la componente referida.
```

Para resolver el problema planteado, se suponen la siguiente gramática y 4 memorias globales conteniendo información sobre las variables que ocurren en la expresión analizada: (1) `mem_lista` almacena las variables de tipo lista que ocurren en la expresión junto a los valores (constantes) y tipos de sus componentes, (2) `mem_entero` almacena las variables de tipo entero con sus valores, (3) `mem_booleano` almacena las variables de tipo booleano con sus valores y. (4) `mem_notipo` almacena las variables de tipo no-tipo.

(Gramática)

```
expresion: expresion MAS expresion
         | expresion POR expresion
```

```
| expresion MENOS expresion
| expresion DIVISION expresion
| expresion Y expresion
| expresion O expresion
| expresion MAYOR expresion
| expresion MENOR expresion
| expresion MAYORIGUAL expresion
| expresion MENORIGUAL expresion
| PARENTESISABIERTO expresion PARENTESISCERRADO
| MENOS expresion
| NOT expresion
| LEN PARENTESISABIERTO expresion PARENTESISCERRADO
| IDENTIFICADOR PUNTO COUNT PARENTESISABIERTO expresion PARENTESISCERRADO
| IDENTIFICADOR CORCHETEABIERTO expresion CORCHETECERRADO
| IDENTIFICADOR
| TRUE
| FALSE
| NUMERO
| NONE
```

Por ejemplo, dada la expresión `s[x+1]+3` y las memorias
`mem_lista = {s, [(True,booleano), (3,entero), (2,entero)]},`
`mem_entero = {x, 0}, mem_booleano = {}` y `mem_notipo = {}`
el analizador semántico determina que `s[x+1]+3` es una expresión errónea porque:
`s[x+1]` es la componente `s[1]=(True,booleano)`, es decir, una subexpresión de tipo booleano,
`3` es una subexpresión de tipo entero y
la restricción 4 no se cumple.

Otro ejemplo, con la misma expresión `s[x+1]+3` pero con las siguientes memorias
`mem_lista = {s, [(3,entero), (True,booleano), (2,entero)]},`
`mem_entero = {x, 0}, mem_booleano = {}` y `mem_notipo = {}`
produce el resultado contrario (la expresión no es errónea) porque:
`s[x+1]` es la componente `s[1]= (3,entero)`, es decir, una subexpresión de tipo entero,
`3` es una subexpresión de tipo entero y
ninguna restricción se incumple, en particular, la restricción 4.

SE PIDE:
Diseño de un analizador semántico para detectar expresión errónea en los términos especificados (si alguna restricción no se cumple entonces la expresión es errónea). Atribuya para ello la gramática previamente dada suponiendo las memorias globales. Trabaje en folios la versión no definitiva y, cuando termine, redacte la versión definitiva en la hoja dada a tal fin. **Sea claro y legible. Evite tachaduras.**

Nombre: _____ Apellidos: _____

Compilador (Diseño). (2,5 puntos) Se pretende diseñar un compilador que traduzca a Java programas escritos en un lenguaje llamado PY-Lite. Antes de comenzar el diseño de una gramática atribuida, se deben tomar algunas decisiones relevantes. Este ejercicio trata de tales decisiones.

PY-Lite es un sublenguaje de PY que suprime de éste las instrucciones compuestas, las componentes de listas, el producto de lista por número, las listas por comprensión, los operadores de igualdad y desigualdad y todas las funciones aplicables a listas salvo la longitud de la lista y el conteo de valores que ocurren en ésta. La siguiente gramática muestra PY-Lite a nivel sintáctico:

```
programa : (instrucciones)? ;

instrucciones:
     instruccion instrucciones
   | instruccion
   ;
instruccion:
     asignacion  | eliminacion  | impresion ;
asignacion:
   variables ASIG expresiones ;

variables: IDENTIFICADOR COMA variables
   | IDENTIFICADOR
   ;
expresiones:
     expresion COMA expresiones
   | expresion
   ;
eliminacion:  DEL IDENTIFICADOR   ;

impresion: PRINT PARENTESISABIERTO expresiones PARENTESISCERRADO ;

op_bool_binario: Y | O ;

op_rel_orden: MAYOR | MENOR | MAYORIGUAL | MENORIGUAL;

expresion: expresion MAS expresion
         | expresion POR expresion
         | expresion MENOS expresion
         | expresion DIVISION expresion
         | expresion op_bool_binario expresion
         | expresion op_rel_orden expresion
         | PARENTESISABIERTO expresion PARENTESISCERRADO
         | MENOS expresion
         | NOT expresion
         | LEN PARENTESISABIERTO expresion PARENTESISCERRADO
         | IDENTIFICADOR PUNTO COUNT PARENTESISABIERTO expresion PARENTESISCERRADO
         | IDENTIFICADOR
         | TRUE
         | FALSE
         | NUMERO
         | NONE
         | CORCHETEABIERTO expresiones CORCHETECERRADO
         | CORCHETEABIERTO  CORCHETECERRADO
         ;
```

Para resolver el diseño es necesario tomar las siguientes decisiones relevantes:

(Decisión 1) Las variables PY-Lite no se declaran explícitamente. La declaración se produce implícitamente al asignarle un primer valor. Sin embargo, Java obliga a declarar antes de usar. ¿Cómo puede saber el compilador cuándo debe o no declarar una variable y evitar así declararla más de una vez?
El siguiente ejemplo muestra código PY-Lite:

```
i=[]
j=True
i=1
```

(Decisión 2) La asignación múltiple (o paralela) de PY-Lite no existe en Java. ¿Cómo puede simular en Java esta característica? A continuación, se muestra un código PY-Lite. con ejemplos de asignación paralela:

```
x,y=1,6
x,y=y,x
```

(Decisión 3) PY-Lite tiene un tipado dinámico: las variables pueden cambiar de tipo a lo largo del programa. Sin embargo, Java tiene un tipado estático: las variables no pueden cambiar de tipo. ¿Cómo puede simular el tipado dinámico de PY-Lite en Java? El siguiente ejemplo muestra un código PY-Lite con una variable que cambia de tipo.

```
i=[]
i=1+len(i)
```

(Decisión 4) La lista por extensión no existe en Java. ¿Cómo puede simular esta capacidad de PY-Lite en Java? El siguiente ejemplo muestra una lista por extensión. ¿Qué código Java propone para traducirla?

```
[8,[1,5],3]
```

(Decisión 5) Haciendo uso de todas las decisiones anteriores ¿Cómo traduciría a Java el siguiente programa PY-Lite? Recuerde que PY-Lite no es orientado a objeto y Java sí.

```
x,y=[8,[1,5],3],6
x,y=y,x
print(x,y)
```

SE PIDE: Redacte las decisiones tomadas. Trabaje de forma tentativa en folios aparte y, cuando concluya, escriba el resultado en la hoja dada a tal fin. **Sea claro y legible. Evite tachaduras.**

Intérprete (Implementación). (2,5 puntos) En este ejercicio se quiere implementar en Antlr/Java parte del diseño de un intérprete de un sublenguaje de PY. La parte del diseño de interés se muestra a continuación:

(Decisión 1) Para permitir el tipado dinámico, se definen 4 memorias para almacenar las variables (y sus valores) durante la interpretación del programa: (a) mem_lista (variables de tipo lista: se almacenan los valores y tipo de cada componente), (b) mem_entero (variables de tipo entero), (c) mem_boolean (variables de tipo booleano) y (d) mem_notipo (variables de tipo no-tipo). Las variables pueden cambiar de memoria.

(Decision 2) Para interpretar una expresión se necesitará el valor y el tipo de ésta. Por ejemplo, la expresión x + y puede referirse tanto a una suma de enteros como a una concatenación de listas. El tipo de las variables permitirá desambiguar entre las posibles interpretaciones facilitando el cálculo del valor.

(Decisión 3) Las variables asignadas pueden ser componentes de listas o variables convencionales (enteras, booleanas, no-tipo). Para cubrir ambos tipos de variables se utilizarán un doble parámetro de salida en las reglas correspondientes: uno para identificar la variable (nombre_var) y otro para indicar el componente (componente), si lo hubiere.

```
(GLOBAL) mem_lista, mem_entero, mem_booleano, mem_notipo

programa : (instrucciones)?

instrucciones:   instruccion instrucciones  | instruccion

instruccion:  asignacion  | eliminacion  | impresion

asignacion:
   vars=variables ASIG exps=expresiones { actualizarMemorias(vars,exps) }

variables dev vars:
     nombre_var,componente=variable COMA vars=variables
     { añadir el elemento (nombre_var,componente) al principio de la lista vars}
   | nombre_var,componente=variable
     { la lista vars se inicializa con un primer elemento (nombre_var,componente) }

variable dev nombre_var,componente:
   IDENTIFICADOR CORCHETEABIERTO valor,tipo=expresion CORCHETECERRADO
   { nombre_var=IDENTIFICADOR y componente=valor }
  | IDENTIFICADOR { nombre_var=IDENTIFICADOR y no tiene componente }

expresiones dev exps:
     valor,tipo=expresion COMA exps=expresiones
     { añadir el elemento (valor,tipo) al principio de la lista exps }
   | valor,tipo=expresion
     { la lista exps se inicializa con un primer elemento (valor,tipo) }

eliminacion :
   DEL nombre_var,componente=variable { eliminarMemoria(nombre_var,componente) }

impresion:
   PRINT PARENTESISABIERTO exps=expresiones PARENTESISCERRADO { imprimir(exps) }
```

SE PIDE:
Implementar en Antlr/Java el diseño propuesto. No es necesario implementar ni las memorias globales ni las operaciones `actualizarMemorias(vars,exps)`, `eliminarMemoria(nombre_var,componente)` e `imprimir(exps)`, solamente su uso en la implementación pedida. Trabaje en folios la versión no definitiva y, cuando termine, redacte la versión definitiva en la hoja dada a tal fin. **Sea claro y legible. Evite tachaduras.**