

TC3048 Diseño de Compiladores

**Omedetou**

30 de marzo del 2022

|  |  |
| --- | --- |
| Sergio Guasso Idalias | Alejandro Cedillo Gamez |

# Visión

# Con nuestro nuevo lenguaje queremos poner a prueba los conocimientos adquiridos durante nuestra formación académica universitaria, implementando los estándares definidos para el desarrollo de compiladores, y añadiendo funcionalidades que creamos necesarias para complementar las librerías preestablecidas y aumentar la gama de problemas que es capaz de resolver.

# Objetivo principal

# Crear un lenguaje que cumpla con todas las reglas gramaticales necesarias para su funcionamiento básico, el cuál será orientado a objetos. Se tendrá que documentar toda la gramática, las estructuras utilizadas y los diagramas que describen la lógica del programa.

# Requerimientos de lenguaje

# Que soporte la creación de objetos, así como acceso a sus funciones y atributos.

## Elementos básicos (Tokens)

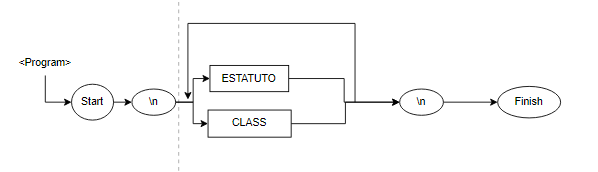
| Nombre | Signo |
| --- | --- |
| START | start |
| FINISH | finish |
| VAR | var |
| WHILE | while |
| IF | if |
| ELSE | else |
| DEF | def |
| RETURN | return |
| CLASS | class |
| EXTENDS | extends |
| PRINT | print |
| PRIVATE | Private |
| PUBLIC | Public |
| PROTECTED | Protected |
| LEFT\_CURRLY\_BRACES | { |
| RIGHT\_CURRLY\_BRACES | } |
| LEFT\_PARENTHESIS | ( |
| RIGHT\_PARENTHESIS | ) |
| COMMA | , |
| COLON | : |
| NEW\_THAN | \n |
| GREATER\_THAN | > |
| LESS\_THAN | < |
| LESS\_EQUAL | <= |
| GREATER\_EQUAL | >= |
| NOT\_EQUAL | != |
| EQUAL\_COMPARATION | == |
| EQUAL | = |
| PLUS | + |
| MINUS | - |
| ASTERISK | \* |
| SLASH | / |

## Diagramas de sintaxis

Ver siguiente link:

<https://drive.google.com/file/d/1mg31Oh5NAC9qVDS2xzRFxlYAqREs92jk/view>

### <Program>

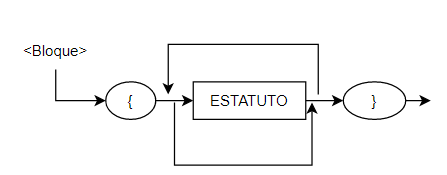


program -> Start \n program1 \n Finish

program2 - > ESTATUTO program3| CLASS program3

program3 -> program2 | ε

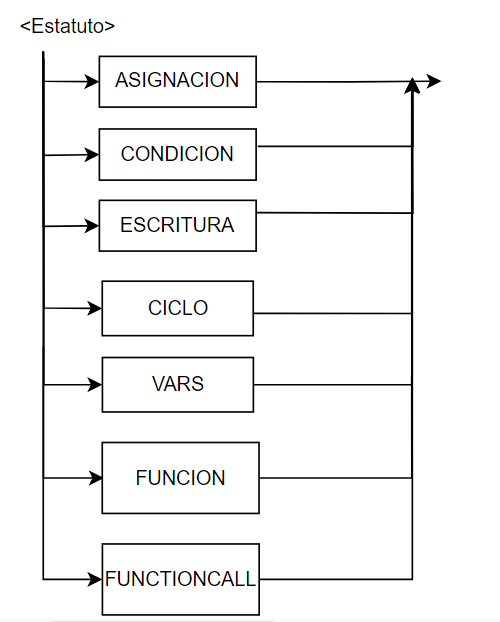
### <Bloque>



bloque global -> { B1 }

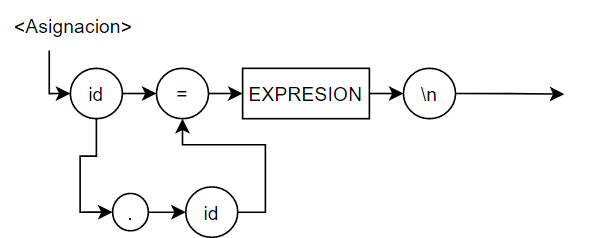
B1-> ε | ESTATUTO B1

### <Estatuto>



Estatuto-> ASIGNACION | CONDICION | ESCRITURA | CICLO | VARS | FUNCION | FUNCTIONCALL

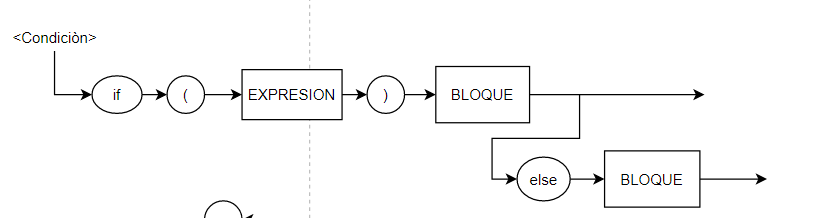
### <Asignacion>



Asignacion -> id asignacion2 = EXPRESION \n

asignacion2-> ε | . id

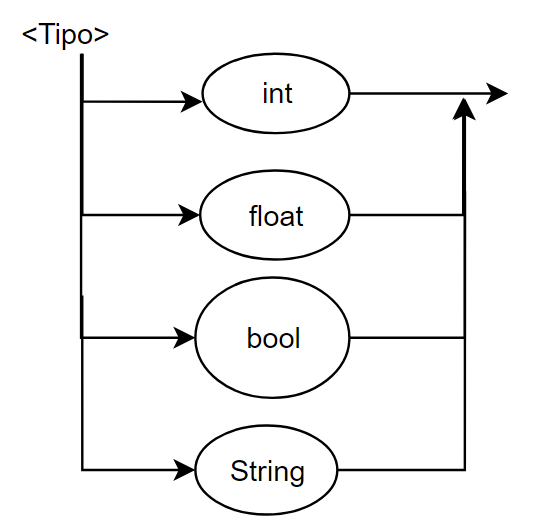
### <Condicion>



Condicion -> if ( EXPRESION ) BLOQUE C2

C2 -> else BLOQUE | ε

### <Tipo>

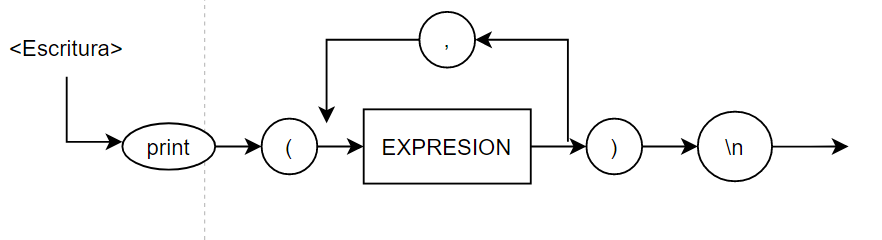


Tipo-> int | float | bool | string

### <Ciclo>

Ciclo-> while ( EXPRESION ) BLOQUE

### <Escritura>



Escritura -> print ( EXPRESION ESC2 ) \n

ESC2 -> , EXPRESION ESC2 | ε

### <Vars>

## 

VARS-> var var2

vars2->id id vars6 \n | TIPO vars3

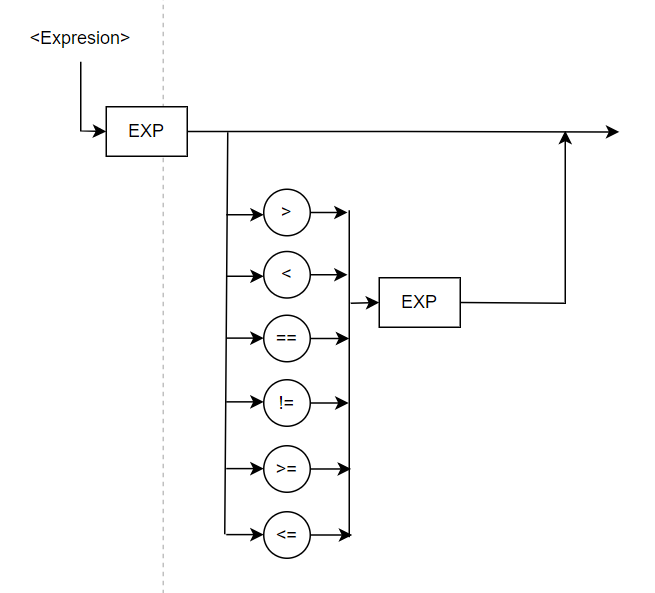
vars3-> ASIGNACION | id vars4 \n

vars4-> [ cte int ] vars5 | ε

vars5 -> [ cte int ] | ε

vars6-> = EXPRESION | ε

### <Expresion>

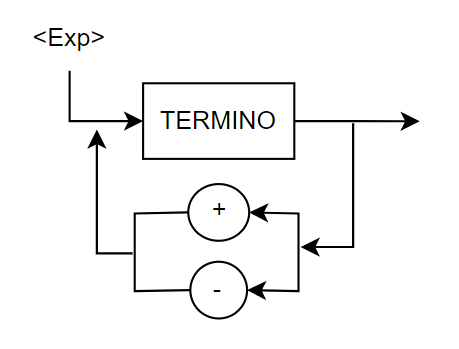


Expresion-> EXP E1

E1 -> ε | E2 EXP

E2 -> > | < | == | != | >= | <=

### <Exp>

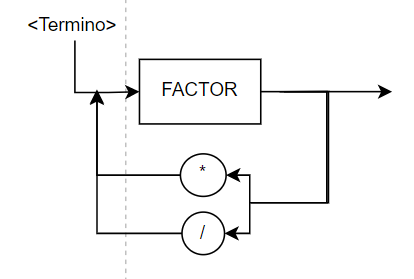


Exp-> Termino EX1

EX1-> ε | EX2 Exp

EX2-> + | -

### <Termino>

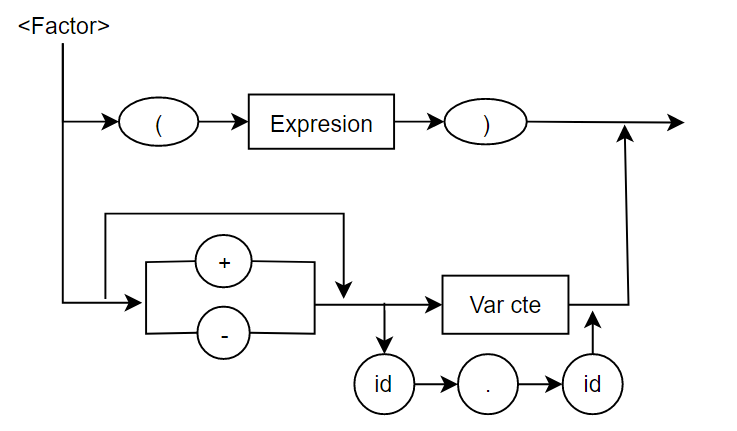


Termino-> FACTOR T1

T1 -> ε | T2 Termino

T2-> \* | /

### <Factor>



Factor-> F1 | F2

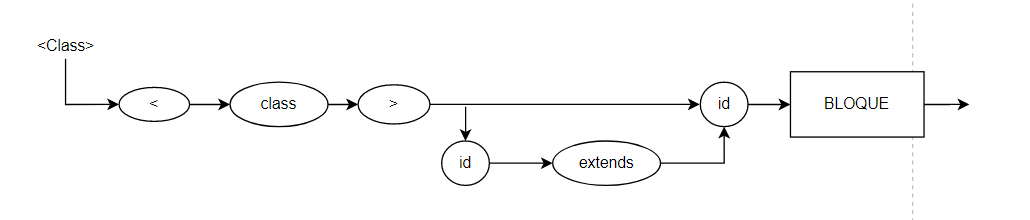
F1-> ( EXPRESION )

F2-> F3 F4

F3-> + | - | ε

F4-> Var cte | id . id

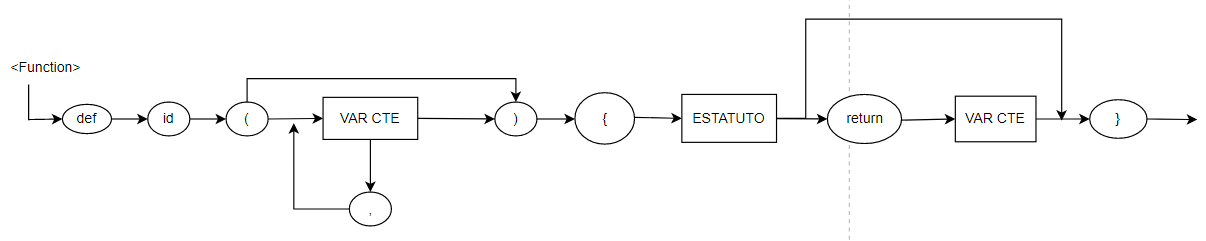
### <Class>



Class-> < class > class2 id BLOQUE

class2-> id extends | ε

### <Function>



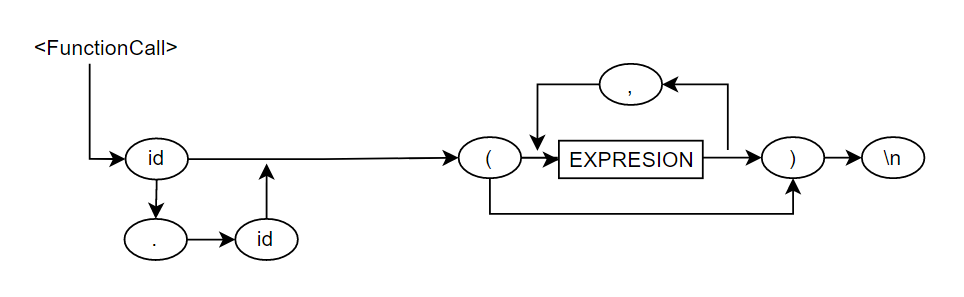
Function-> def id ( F1 ) { ESTATUTO F2 }

F1-> ε | VARCTE F3

F2-> ε | return VARCTE

F3 -> , VARCTE F3 | ε

### <FunctionCall>



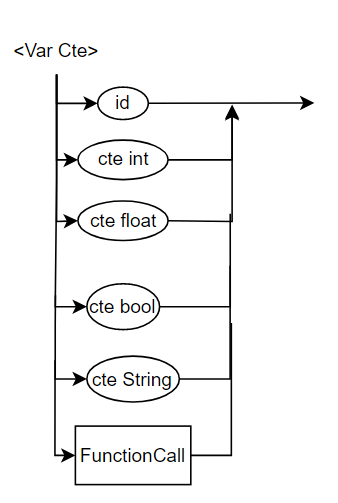
FunctionCall-> id fc1 ( fc2 ) \n

fc1-> . id | ε

fc2-> EXPRESION fc3 | ε

fc3-> , EXPRESION fc3 | ε

### <Var Cte>



VarCte -> id | cte int | cte float | cte bool | FunctionCall

## Características principales de semántica

* Soporte de programación orientada a objetos
* Seguir PEMDAS

## Descripciones específicas

Ejemplo de un código en lenguaje Omedetou:

Start

<class> Person{

var num : int

var spreak : String

num = 4

speak = "hola mundo"

}

var bob: Person

while ( bob.num > 0) {

print(bob.speak)

print("\n")

bob.num = bob.num - 1

}

Finish

## Tipos de datos

* Int
* Float
* String
* Bool

# Language and OS

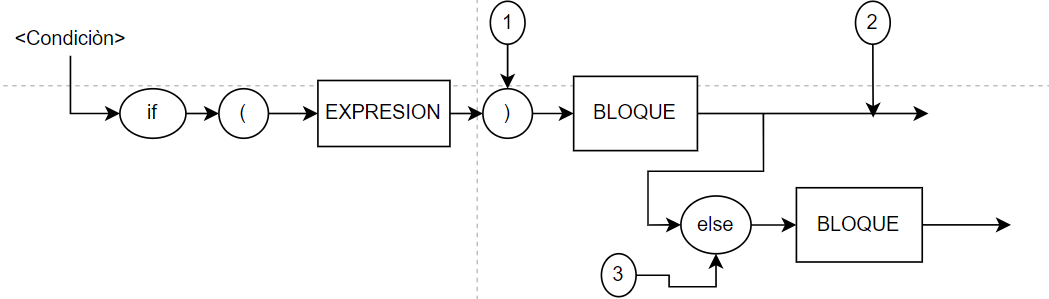
Language : Python

OS : Windows

# Bibliografía

# Directorio de procedimientos

## Condición



1-

resIF = pop ( Popper )

if (type(resIF) != bool)

Type Mismatch Error

Else

Quad: GotoF, res, ,

psaltos.push(cont-1)

2-

salida = pop(Psaltos)

completeSalto(salida,cont)

3-

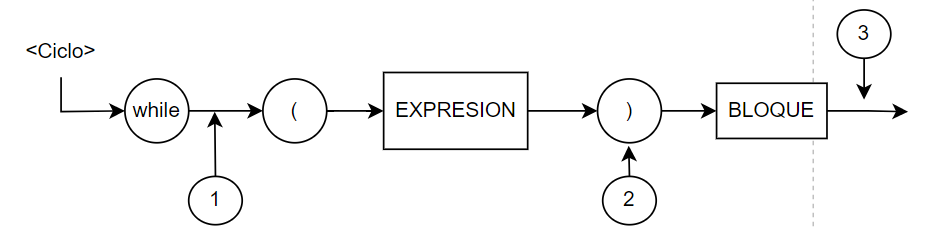
Quad: Goto\_

Falso = pop(Psaltos)

P.saltos push(cont-1)

completeSalto(falso, cont)

## Ciclos



1-

Psaltos.push(cont)

2-

resWhile= pop({Popper)

if (type(resWhile) != bool)

Error Type Mismatch

else:

Quad: gotoF, cond, \_

Psaltos.push(cont-1)

3-

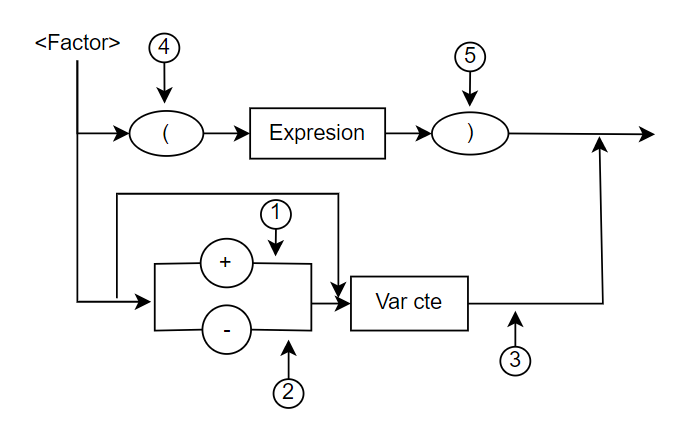
Falso = pop (Psaltos)

return = pop(Psaltos)

Quad: goto return

completeSalto(falso,cont)

## Factor



1-  
pilaO.push(‘+‘)

2-

pilaO.push(‘-’)

3-

popper.push(VarCte)

4-

# Inicio fondo falso

popper.push(‘(’)

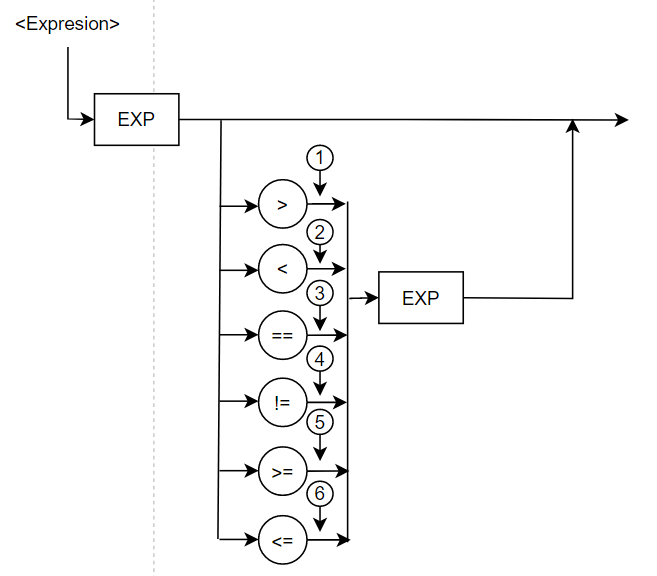
5-

# Fin fondo falso

# no se mente ‘)’ , solo se termina

popper.pop()

## Expresión



#Meter ‘>’ a PilaO

1- PilaO.push(‘>’)

2- PilaO.push(‘<’)

3- PilaO.push(‘==’)

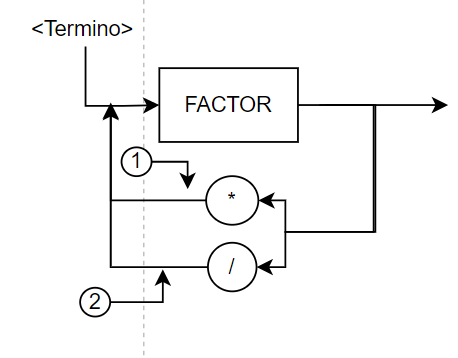
4- PilaO.push(‘!=’)

5- PilaO.push(‘>=’)

6- PilaO.push(‘<=’)

# To Do revisar manejo de errores

## Término



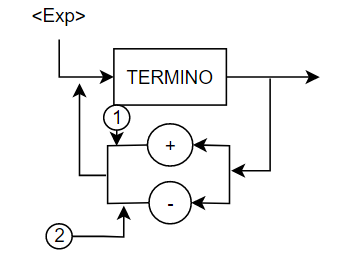
1-

pilaO.push(‘\*‘)

2-

pilaO.push(‘/‘)

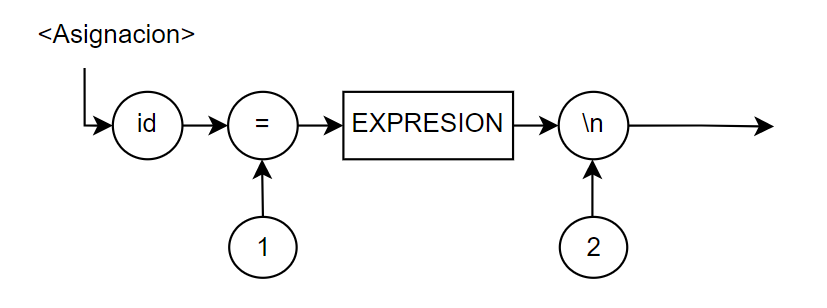
## Exp



1-pilaO.push(‘+‘)

2-pilaO.push(‘-‘)

## Asignación

****

1-

PilaO.push(‘=’)

2-

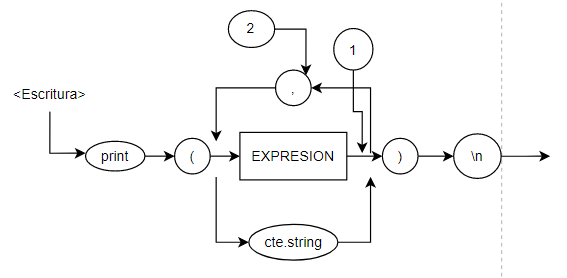
Valor\_a\_igualar = Popper.pop()

Valor\_donde\_se\_guarda = Popper.pop()

Igual\_sign = PilaO.pop()

Quad: Igual\_sign, Valor\_a\_igualar , , Valor\_donde\_se\_guarda

## Escritura



1-

Tinta = Popper.pop()

Quad: print, , , Tinta

2- Repetir paso 1