# Gramática MiniPython

Program -> ’class’ ID ‘:’ inicioBloque <field\_decl>\* finBloque <method\_call>\*

field\_decl -> <assign>

method\_decl -> ‘def’ ID ( ‘(‘ ID (‘,’ ID)\* ‘)’ )? ‘:’ <block>

block -> <inicioBloque><statement>\*<finBloque>

<statement> -> ID <Statementp>

| <methodcall>

| ‘if’ <expr> ‘:’ ( ‘elif’ <expr> ‘:’ <block)\* ( ‘else’ ‘:’ <block>)

| ‘while’ <expr> ‘:’ <block>

| ‘for’ ID ‘in’ <range> ‘:’ <block>

| ‘return’ <expr>

| ‘break’

<Statementp> -> ‘ ID’

| = <assignP>

| [ <assignP>

| ( methodcall2

<assign> -> <lvalue> ‘=’ <expr>

<assignP> -> ‘[‘ <expr1>‘]’ ‘=’ <expr2> { assignP.return = new AssignExpr (new

ArrayIndex(assignP.inherit.nombre,expr1.return ), expr2.return) }

| ‘=’ <expr> {assignP.return = new AssignExpr(assigP.inherit,expr.return)}

<method\_call> -> ID ‘(‘ ( (<expr> ( ‘,’ <expr>)\*)?

| ‘read’ <lvalue>

| ‘print’ (<expr>(‘,’<expr>)\*)?

<method\_call2> -> ‘(‘(<expr>(‘,’<expr>)\*)? ‘) {methodcall2.inherit.add(expr)}

<lvalue> -> ID

| ID ‘[‘ <expr>’]’

<expr> -> Relacional {expr.return = Relacional.return}

| ‘ – ‘ <expr1>{expr.return = new NegateExpr(expr1.return)}

| ‘~’ <expr1> {expr.return = new InvenrtExpr(expr1.return)}

| ‘[‘ <expr> ( ‘ , ‘ <expr>)\* ‘ ]’

<Relacional> -> | AritmeticoSumaResta ( != AritmeticoSumaResta) \*{

Expr Relacional1 =AritmeticoSumaResta1.return

While (´!=´) {

Relacional1 = new DistintoDeExpr(Relacional1,AritmeticoSumaResta2.return)

}

Relacional.return = Relacional1.return

}

| AritmeticoSumaResta ( <= AritmeticoSumaResta) \*{

Expr Relacional1 =AritmeticoSumaResta1.return

While (´<=´) {

Relacional1 = new MenorIgualExpr(Relacional1,AritmeticoSumaResta2.return)

}

Relacional.return = Relacional1.return

}

| AritmeticoSumaResta ( >= AritmeticoSumaResta) \*{

Expr Relacional1 =AritmeticoSumaResta1.return

While (´>=´) {

Relacional1 = new MayorIgualExpr(Relacional1,AritmeticoSumaResta2.return)

}

Relacional.return = Relacional1.return

}

| AritmeticoSumaResta1 ( == AritmeticoSumaResta2) \*{

Expr Relacional1 =AritmeticoSumaResta1.return

While (´==´) {

Relacional1 = new ComparacionExpr(Relacional1,AritmeticoSumaResta2.return)

}

Relacional.return = Relacional1.return

}

| AritmeticoSumaResta1 ( < AritmeticoSumaResta2) \*{

Expr Relacional1 =AritmeticoSumaResta1.return

While (´<´) {

Relacional1 = new MenorExpr(Relacional1,AritmeticoSumaResta2.return)

}

Relacional.return = Relacional1.return

}

| AritmeticoSumaResta1 ( > AritmeticoSumaResta2) \*{

Expr Relacional1 =AritmeticoSumaResta1.return

While (´>´) {

Relacional1 = new MayorExpr(Relacional1,AritmeticoSumaResta2.return)

}

Relacional.return = Relacional1.return

}

< AritmeticoSumaResta > -> Produccion1 (‘+’ Produccion2) \*{

Expr AritmeticoSumaResta1 = Produccion1.return

While (´+´) {

AritmeticoSumaResta 1 = new AddExpr(AritmeticoSumaResta1,Produccion2.return)

}

AritmeticoSumaResta.return = AritmeticoSumaResta1.return

}

| Produccion1 (‘-‘ Produccion2) \*{

Expr AritmeticoSumaResta1 = Produccion1.return

While (´-´) {

AritmeticoSumaResta 1 = new MinusExpr(AritmeticoSumaResta1,Produccion2.return)

}

AritmeticoSumaResta.return = AritmeticoSumaResta1.return

}

| Produccion1 (‘or’ Produccion2) \*{

Expr AritmeticoSumaResta1 = Produccion1.return

While (´or´) {

AritmeticoSumaResta 1 = new OrExpr(AritmeticoSumaResta1,Produccion2.return)

}

AritmeticoSumaResta.return = AritmeticoSumaResta1.return

}

<Produccion> -> Shift (‘/’ Shift) \*{

Expr shift1 = Shift1.return

While (´/´) {

shift1 = new DivExpr(shift1,Shift2.return)

}

Produccion.return = shift1

}

|Shift( ‘\*’ Shift)\* {

Expr shift1 = Shift1.return

While (´\*´) {

shift1 = new MulExpr(shift1,Shift2.return)

}

Produccion.return = shift1

}

| Shift (‘%’ Shift)\* {

Expr shift1 = Shift1.return

While (´%´) {

shift1 = new ModExpr(shift1,Shift2.return)

}

Produccion.return = shift1

}

| Shift1(‘ and’ Shift2)\* {

Expr shift1 = Shift1.return

While (´and´) {

shift1 = new AndExpr(shift1,Shift2.return)

}

Produccion.return = shift1

}

<Shift> -> Term1 (‘>>’ Term2) \* {

Expr term1 = Term1.return

While (´<<´) {

term1 = new ShiftRightExpr(term1, Term2.return)

}

Shift.return = term1

}

|Term1 (‘<<’ Term2)\* {

Expr term1 = Term1.return

While (´<<´) {

term1 = new ShiftLeftExpr(term1, Term2.return)

}

Shift.return = term1

}

<Term> -> <constant> {Term.return = constant.return}

|’(‘ <expr> ‘)’ {Term.return = expr.return}

| ID exprP {exprP.inherit = new ID (id.lexema)

Term.return = exprP.return}

| READ method\_call \*\*\*\*dejarlo pq nose q p2

| KW\_PRINT method\_call {Term.return = MethodCall.return }

<exprP> -> ‘=’ <assignP> { assignP.inherit = exprP.inherit

exprP.return = assignP.return}

|’[‘ <expr> ‘]’ {exprP.return = new ArrayIndex(expr.return)}

| ‘(‘ <methodcall2> {methodcall2.inherit = new MethodCall()

exprP.return = methodcall2.return }

<constant> -> ‘NUMBER’ {constant.return = new Number (number.lexema) }

| ‘CHARCONSTANT’ { constant.return = Char(charconstant.lexema) }

| <bool\_const> { constant.return = Boolean.return }

<bool\_const> -> [ ‘TRUE’ | ‘FALSE’ ] { bool\_const.return = new Boolean( true | false) }

<inicioBloque> -> INDENT

<finBloque> -> DEDENT

<range> -> <expr> ‘…’ <expr>