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>

                     | print <methodcall>

                     | read <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>       ->   [‘ <expr> ‘]’ ‘=’ <expr>

                     | ‘=’ <expr>

<method_call> ->     ‘read’ <lvalue>

                      | ‘print’ (<expr>(‘,’<expr>)*)?
```

```
<method_call2>  ->      ‘(‘(<expr>(‘,’<expr>)*)? ‘)
<lvalue>              ->      ID
                             |  ID ‘[‘ <expr>’]’


<expr>            ->      Logical
                         |  ‘ – ‘ <expr>
                         | ‘~’ <expr>
                         | ‘[‘ <expr> ( ‘ , ‘ <expr>)* ‘ ]’
<exprP>        ->      ’[‘ <expr> ‘]’
                        | ‘(‘  <methodcall2>
<Logical>        ->      Relacional (‘or’ Relacional)*
                        | Relacional (‘and’ Relacional)*


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


< AritmeticoSumaResta >   ->      Produccion (‘+’ Produccion) *
                                 | Produccion (‘-‘ Produccion) *


<Produccion>     ->        Shift (‘/’ Shift) *
                            |Shift( ‘*’ Shift)*
                            | Shift (‘%’ Shift)*
```

```
<Shift>            ->   Term ('>>' Term) *
                       |Term ('<<' Term)*


<Term>             ->     <constant>
                          |'(' <expr> ')'
                          | ID exprP
                          | '[' <expr> (<expr>)* ']'
<inicioBloque>     ->       INDENT
<finBloque>        ->       DEDENT
<range>            ->       <expr> '…' <expr>
<constant>         ->       'NUMBER' | 'CHARCONSTANT' | <bool_const>
<bool_const>       ->       'TRUE' | 'FALSE'
```