

PythonSubset.g4

Archivo copiado desde [grammar/PythonSubset.g4](#).

```
grammar PythonSubset;

// Entrada principal: una o más sentencias seguida de EOF
prog
: stmt+ EOF
;

// Sentencias: asignaciones, expresiones y ciclos for
stmt
: simple_stmt NEWLINE
| compound_stmt
| NEWLINE // Líneas vacías
;

// Sentencias simples (una línea)
simple_stmt
: assign_stmt
| expr_stmt
;

// Sentencias compuestas (con bloques)
compound_stmt
: for_stmt
| while_stmt
| if_stmt
;

// Asignación: variable = expresión
assign_stmt
: IDENTIFIER '=' expr
;

// Expresión como sentencia
expr_stmt
: expr
;

// Ciclo for con indentación Python
for_stmt
: FOR IDENTIFIER IN iterable ':' NEWLINE INDENT stmt+ DEDENT
;

// Ciclo while con indentación Python
while_stmt
: WHILE expr ':' NEWLINE INDENT stmt+ DEDENT
;
```

```
// Sentencia if con elif y else opcionales
if_stmt
    : IF expr ':' NEWLINE INDENT stmt+ DEDENT elif_clause* else_clause?
    ;

// Cláusula elif
elif_clause
    : ELIF expr ':' NEWLINE INDENT stmt+ DEDENT
    ;

// Cláusula else
else_clause
    : ELSE ':' NEWLINE INDENT stmt+ DEDENT
    ;

// Expresiones que pueden ser iteradas (por ahora solo range)
iterable
    : range_call
    | expr // Para futuras extensiones
    ;

// Llamada específica a range()
range_call
    : 'range' '(' range_args ')'
    ;

// Argumentos de range: range(stop) o range(start, stop) o range(start, stop, step)
range_args
    : expr # RangeStop
    | expr ',' expr # RangeStartStop
    | expr ',' expr ',' expr # RangeStartStopStep
    ;

// Expresiones soportadas (precedencia de menor a mayor)
expr
    : expr OR expr # LogicalOr
    | expr AND expr # LogicalAnd
    | comparison # ComparisonExpr
    ;

// Comparaciones # Comparison
comparison
    : arith_expr (comp=='==' | '!=' | '>=' | '<=' | '>' | '<') arith_expr?
    ;

// Expresiones aritméticas
arith_expr
    : arith_expr op=( '+' | '-' ) arith_expr # AddSub
    | arith_expr op=( '*' | '/' | '%' ) arith_expr # MulDivMod
    | unary_expr # ArithUnary
    ;
```

```

// Expresiones unarias y potencia (asociatividad derecha)
unary_expr
  : NOT unary_expr                                # LogicalNot
  | ('+' | '-') unary_expr                         # UnaryOp
  | power_expr                                     # PowerBase
  ;

// Potencia con asociatividad derecha
power_expr
  : atom_expr ('**' unary_expr)?                  # Power
  ;

// Expresiones atómicas (mayor precedencia)
atom_expr
  : IDENTIFIER '(' arg_list? ')'
  | '(' expr ')'
  | INT
  | STRING
  | TRUE
  | FALSE
  | IDENTIFIER
  ;
;

// Lista de argumentos para llamadas
arg_list
  : expr (',' expr)*
  ;
;

// ----- Lexer Rules -----
;

// Palabras clave (deben ir antes de IDENTIFIER)
FOR      : 'for' ;
WHILE    : 'while' ;
IF       : 'if' ;
ELIF     : 'elif' ;
ELSE     : 'else' ;
IN       : 'in' ;
AND      : 'and' ;
OR       : 'or' ;
NOT      : 'not' ;
TRUE     : 'True' ;
FALSE    : 'False' ;
;

// Tokens especiales para indentación (serán generados por el lexer customizado)
INDENT   : 'INDENT' ;
DEDENT   : 'DEDENT' ;
;

IDENTIFIER
  : [a-zA-Z_] [a-zA-Z_0-9]*
  ;
;

INT
  : [0-9]*
  ;
;

```

```
// Cadena entre comillas dobles o simples
STRING
: '"' (~["\r\n"])* '"'
| '\'' (~['\r\n'])* '\''
;

NEWLINE
: ( '\r'? '\n' )+
;

WS
: [ \t]+ -> skip
;

COMMENT
: '#' ~[\r\n]* -> skip
;
```