Open Modular Interpreter

Gramática

Fco. Javier Bohórquez Ogalla

Índice

1. Vista general

En esta sección se presenta la gramática del lenguaje, para ello se procede a una descripción de las reglas gramaticales mediante el lenguaje EBNF (Extended Backus–Naur Form) usado para expresar gramáticas libres de contexto. Además cada regla se acompaña de un diagrama sintáctico o diagrama de carril.

Una gramática libre de contexto G se define formalmente como sigue:

$$G = (V_t, V_n, P, S)$$

De forma que:

V_t es un conjunto finito de símbolos no terminales

V_n es un conjunto finito de símbolos terminales

P es un conjunto finito de reglas de producción

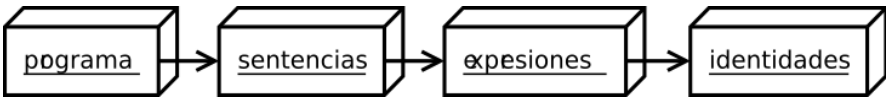
$S \ \epsilon \ V_n$ es el símbolo inicial

Las reglas de producción tiene la forma siguiente forma:

$$V_n \ \rightarrow \ (V_t \ \cup \ V_n)^*$$

La gramática libre de contexto abordada tiene como símbolo inicial el no terminal *program*. Se comienza pues describiendo las reglas de producción relicionadas con este símbolo, siguiendo con las reglas de producción derivadas de esta.

Las reglas de producción se organizan en niveles como sigue:



Estos niveles son dados a partir del nivel de abstracción del significado semántico que encierran las reglas de producción contenidas en los mismos.

Las reglas de producción correspondientes al nivel de programa son las más genéricas y se valen de las de la siguiente nivel para su definición. Estas definen el programa como

una secuencia de sentencias. La gramática descrita contempla el programa vacío, es decir, el programa que no contiene ninguna sentencia.

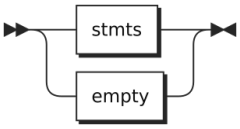
Las reglas de producción del nivel de sentencia se definen a partir de expresiones o de otras sentencias. Una sentencia contiene un significado semántico operativo de valor para el programa. Cabe decir que una expresión por si sola puede constituir una sentencia. La gramática expuesta describe la sentencia vacía, esta es una sentencia que no tienen ningún significado semántico.

Las expresiones son la unidad mínima con significado semántico atribuido por el lenguaje. La mayoría de expresiones se definen a partir de identidades, sin embargo algunos tipos de expresiones, como las funciones, pueden formarse a partir de reglas de más alto nivel de abstracción semantica. Por otro lado las reglas de producción correspondiente a las expresiones están organizadas en niveles según la prioridad atribuida en su resolución.

Las identidades son reglas de producción atómicas, componiéndose únicamente de símblos no terminales. Tienen un significado semántico asociado de forma directa. Normalmente este valor viene dado por el análisis léxico.

2. Programa

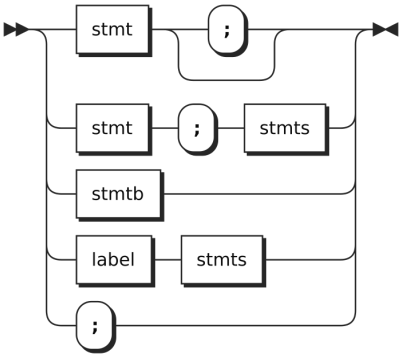
```
program  ::=  stmts
           | empty
```



3. Sentencias

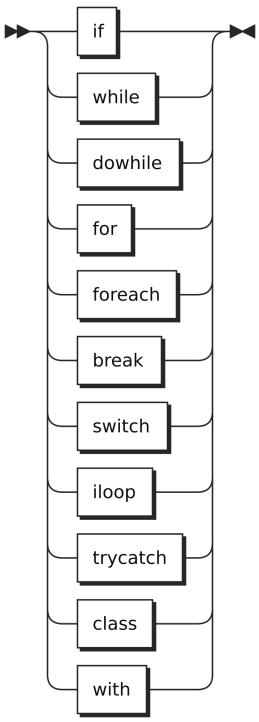
3.1. Secuecia de sentencias

```
stmts    ::=   stmt ";" stmts
           |   stmt ";"?
           |   stmtb
           |   label stmts
           |   ";"
```



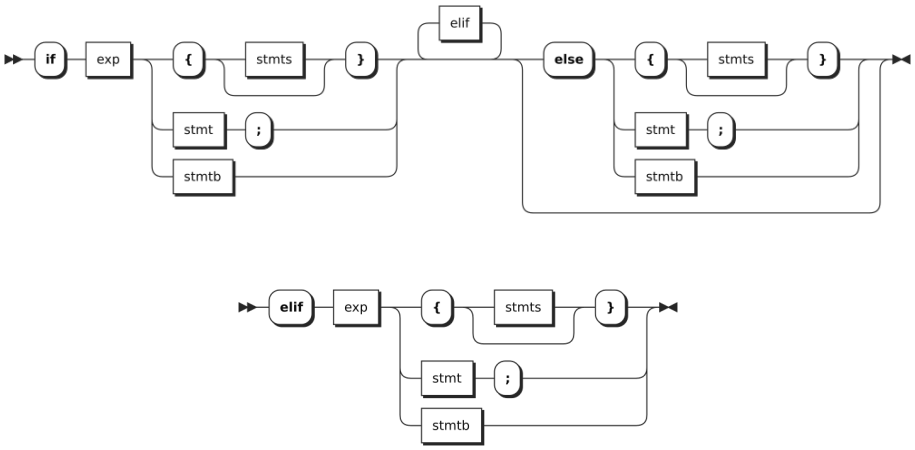
3.2. Sentencia de bloque

```
stmtb ::=   if
          | while
          | dowhile
          | for
          | foreach
          | break
          | switch
          | iloop
          | trycatch
          | class
          | with
```



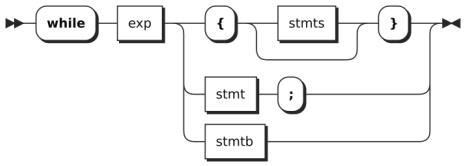
3.2.1. Sentencia de control: if

```
if ::= "if" exp ("{" stmts? "}" | stmt ";" | stmtb) elif* ("else" ("{" stmts? "}" | stmt ";" | stmtb))?
elif ::=  "elif" exp ( "{" stmts? "}" | stmt ";" | stmtb )
```



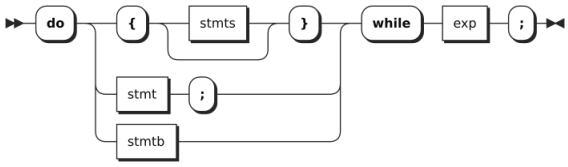
3.2.2. Sentencia de control: while

```
while ::=
   "while" exp ("{" stmts? "}" | stmt ";" | stmtb)
```



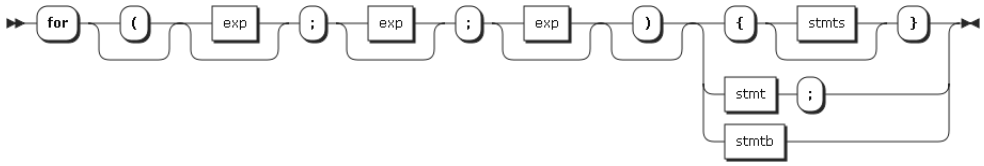
3.2.3. Sentencia de control: do...while

```
dowhile ::=  "do" ( "{" stmts? "}" | stmt ";" | stmtb ) "while" exp ";"
```



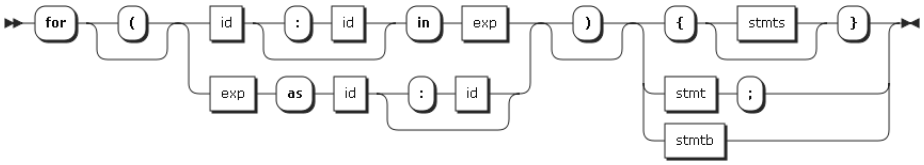
3.2.4. Sentencia de control: for

```
for ::= "for" "("? exp? ";" exp? ";" exp? ")"? ( "{" stmts? "}" | stmt ";" | stmtb )
```



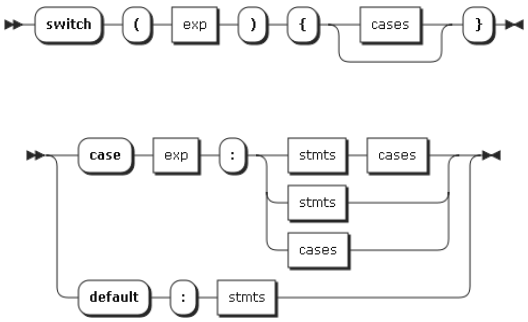
3.2.5. Sentencia de control: forearch

```
foreach ::=
   "for" "("?(id (":" id)? "in" exp | exp "as" id (":" id)?)")"? ("{" stmts? "}" | stmt ";" | stmtb)
```



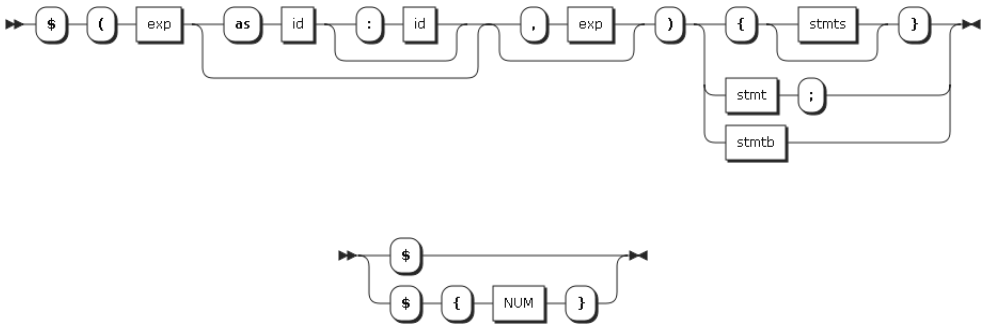
3.2.6. Sentencia de control: switch

```
switch ::= "switch" "(" exp ")" "{" cases? "}"

cases ::= "case" exp ":" ( stmts cases | stmts | cases )
   |  "default" ":" stmts
```



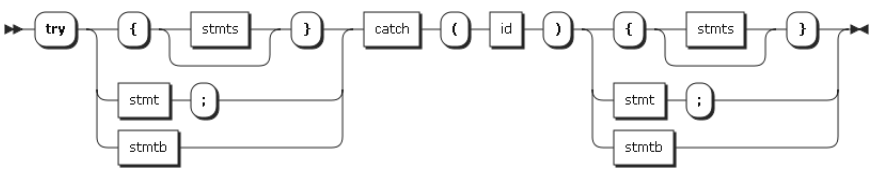
3.2.7. Sentencia de control: iloop

```
iloop ::= "$" "(" exp ( "as" id (":" id)? )? ("," exp )? ")" ( "{" stmts? "}" | stmt ";" | stmtb )

iloop_access ::=  "$"
                | "$" "{" NUM "}"
```



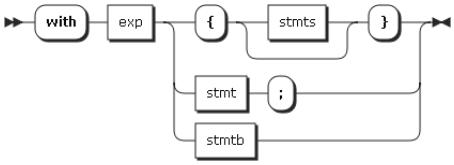
3.2.8. Sentencia de control: try...catch

```
trycatch ::=
 "try" ( "{" stmts? "}" | stmt ";" | stmtb ) catch "(" id ")" ( "{" stmts? "}" | stmt ";" |  stmtb )
```



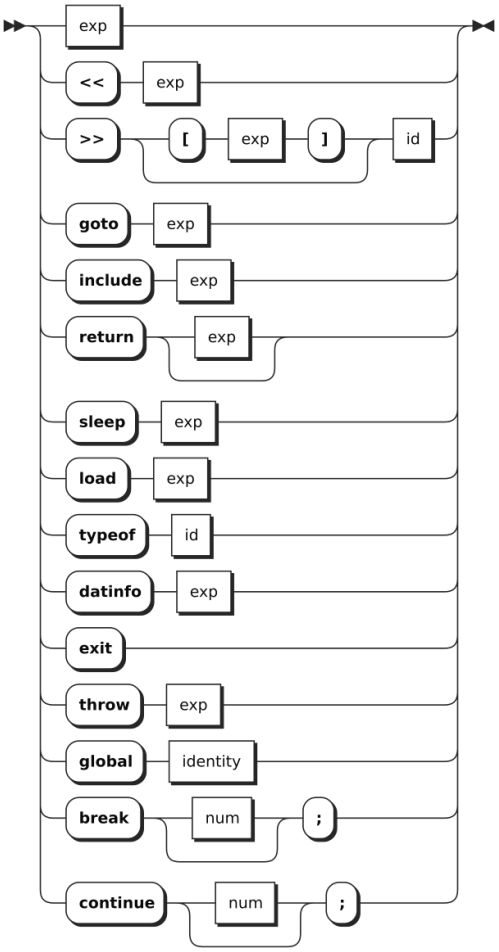
3.2.9. Sentencia de control: with

```
with ::=
 "with" exp ( "{" stmts? "}" | stmt ";" | stmtb )
```



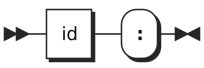
3.3. Sentencia simple

```
stmt  ::=   exp
          |  "<<" exp
          |  ">>" ("[" exp "]")? id
          |  "goto" exp
          |  "include" exp
          |  "return" exp?
          |  "sleep" exp
          |  "load" exp
          |  "typeof" id
          |  "datinfo" exp
          |  "exit"
          |  "throw" exp
          |  "global" identity
          |  "break" num? ";"
          |  "continue" num? ";"
```



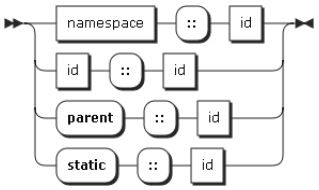
3.4. Etiquetas

```
label ::=   id ":"
```



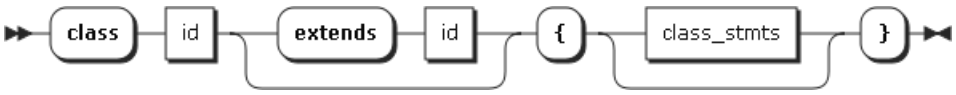
3.5. Nombres de espacios

```
namespace ::=  namespace "::" id
             | id "::" id
             | "parent" "::" id
             | "static" "::" id
```



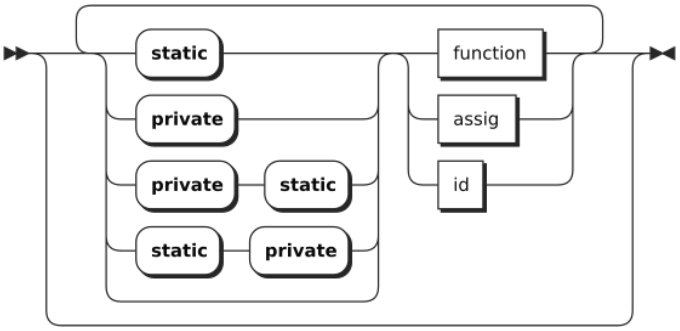
3.6. Clases

```
class ::= "class" id ("extends" id )? "{" class_stmts? "}"
```



3.6.1. Métodos y atributos

```
class_stmts ::=
   (("static"|"private"|"private" "static"|"static" "private")? (function|id|assig))*
```



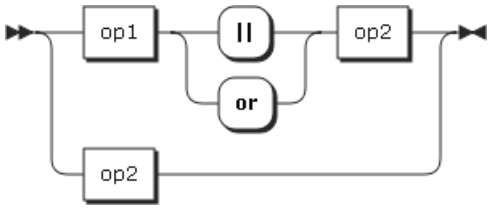
4. Expresiones

```
exp ::= op1
```

4.1. Operadores lógicos

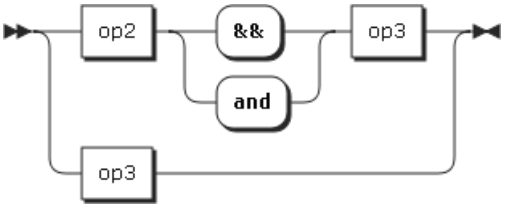
4.1.1. Or lógico

```
op1 ::=  op1 ( "||" | "or" ) op2
       |  op2
```



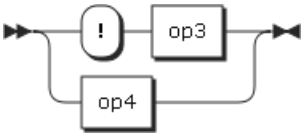
4.1.2. And lógico

```
op2 ::=  op2 ( "&&" | "and" ) op3
       |  op3
```



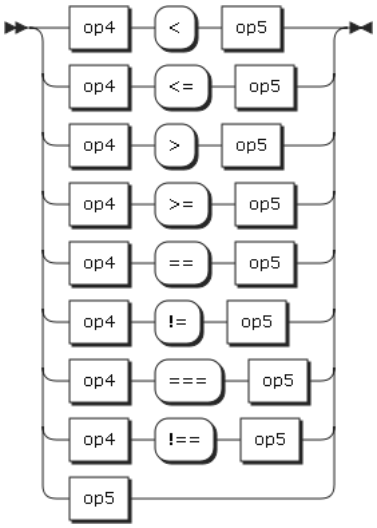
4.1.3. Negación lógica

```
op3 ::=  "!" op3
       |  op4
```



4.1.4. Comparaciones

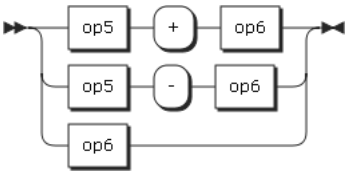
```
op4 ::=  op4 "<" op5
       |  op4 "<=" op5
       |  op4 ">" op5
       |  op4 ">=" op5
       |  op4 "==" op5
       |  op4 "!=" op5
       |  op4 "===" op5
       |  op4 "!==" op5
       |  op5
```



4.2. Operadores aritméticos

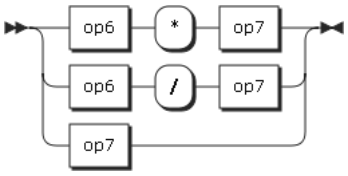
4.2.1. Suma y diferencia

```
op5 ::= op5 "+" op6
      |  op5  "-" op6
      |  op6
```



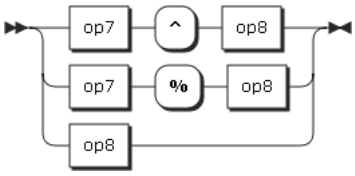
4.2.2. Producto y división

```
op6  ::= op6 "*" op7
      |  op6 "/" op7
      |  op7
```



4.2.3. Potencia y módulo

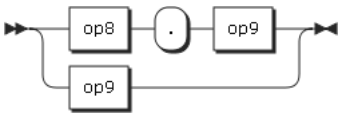
```
op7  ::= op7 "^" op8
      |  op7 "%" op8
      |  op8
```



4.3. Operadores cadenas de caracteres

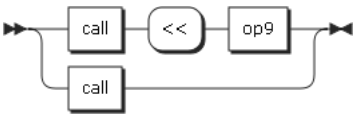
4.3.1. Concatenación

```
op8 ::=  op8 "." op9
      |  op9
```



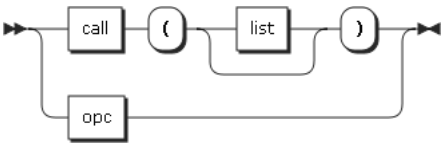
4.3.2. Flujo

```
op9  ::= call "<<" op9
      |  call
```



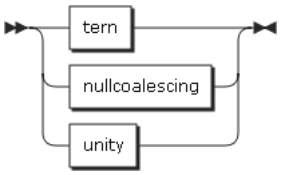
4.4. Llamadas

```
call ::= call "(" list? ")"
       |  opc
```



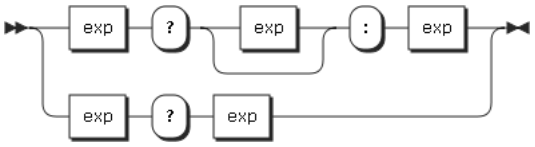
4.5. Operadores condicionales

```
opc ::=  tern
       |  nullcoalescing
       |  unity
```



4.5.1. Operador ternario

```
tern ::= exp "?" exp? ":" exp
   |  exp "?" exp
```

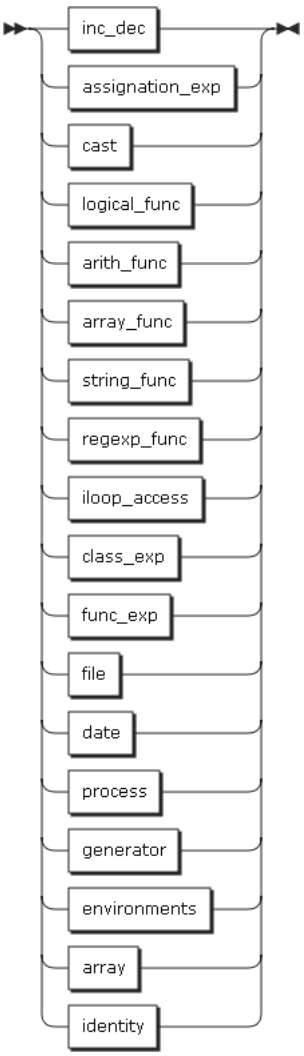


4.5.2. Null coalescing

```
nullcoalescing ::= "[[" list "]]"
```

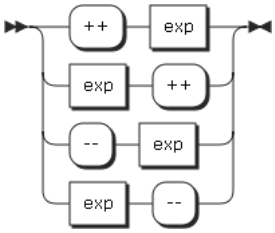
4.6. Operadores unitarios

```
unity ::=   inc_dec
          |  assignation_exp
          |  cast
          |  logical_func
          |  arith_func
          |  array_func
          |  string_func
          |  regexp_func
          |  iloop_access
          |  class_exp
          |  func_exp
          |  file
          |  date
          |  process
          |  generator
          |  environments
          |  array
          |  identity
```



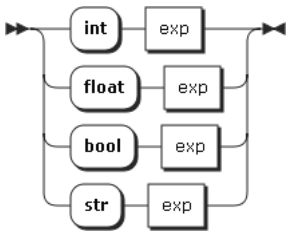
4.6.1. Incrementos y decrementos

```
inc_dec ::= "++" exp
          |  exp "++"
          |  "--" exp
          |  exp "--"
```



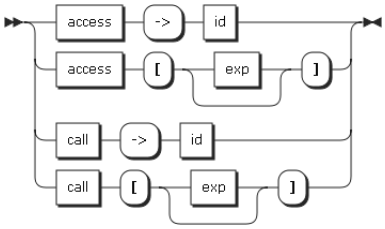
4.6.2. Conversión de tipos

```
cast ::= "int" exp
       | "float" exp
       | "bool" exp
       | "str" exp
```



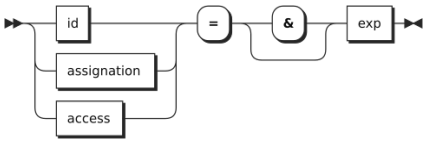
4.6.3. Accesos

```
access ::=  access "->" id
          | access "[" exp? "]"
          | call "->" id
          | call "[" exp? "]"
```



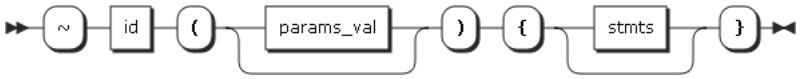
4.7. Asignaciones

```
assignation ::=
 (id | assignation | access)  "=" "&"? exp
```



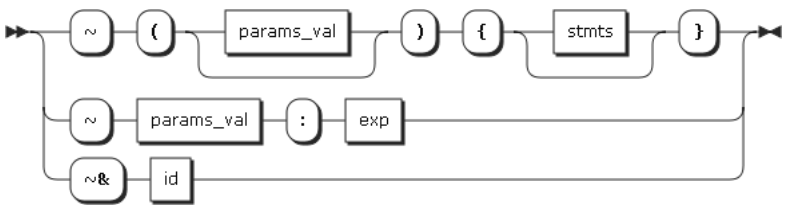
4.8. Funciones

```
function ::= "~" id "(" params_val? ")" "{" stmts? "}"
```



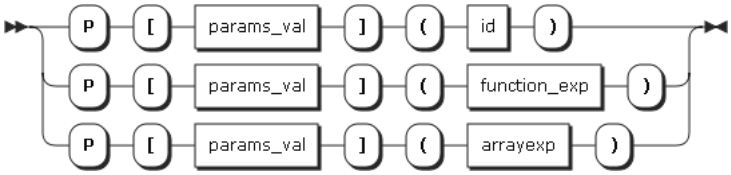
4.8.1. Función lambda

```
function_lambda ::= "~" "(" params_val? ")" "{" stmts? "}"
    | "~" params_val ":" exp
    | "~&" id
```



4.8.2. Cálculo parcial

```
function_partial ::= "P" "[" params_val "]" "(" id ")"
   |  "P" "[" params_val "]" "(" function_exp ")"
   |  "P" "[" params_val "]" "(" arrayexp ")"
```

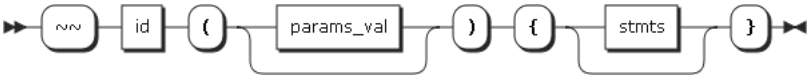


4.8.3. Función de contexto

```
function_context ::= "~>"
```

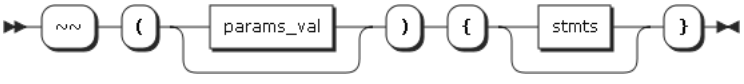
4.9. Decoradores

```
decorator ::= "~~" id "(" params_val? ")" "{" stmts? "}"
```



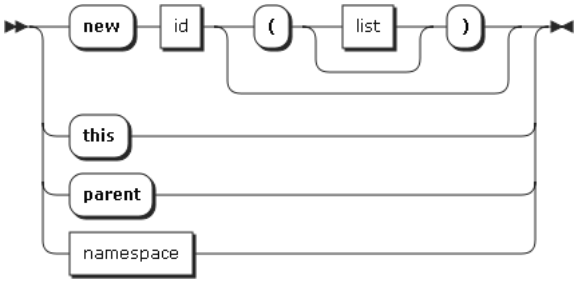
4.9.1. Decorador lambda

```
decorator_lambda ::= "~~" "(" params_val? ")" "{" stmts? "}"
```



4.10. Operadores clases y objetos

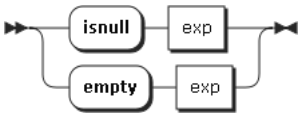
```
class_exp ::=  "new" id ("(" list? ")")?
            |  "this"
            |  "parent"
            |  namespace
```



4.11. Funciones del lenguaje

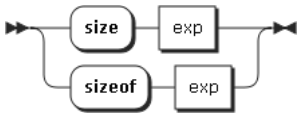
4.11.1. Funciones lógicas

```
logical_func ::=  "isnull" exp
               |  "empty" exp
```



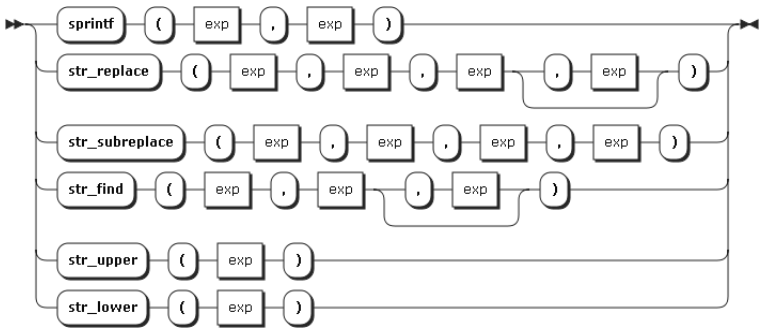
4.11.2. Funciones aritméticas

```
arith_func ::= "size" exp
            |  "sizeof" exp
```



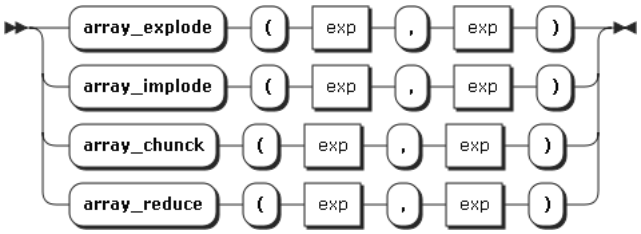
4.11.3. Funciones cadenas de caracteres

```
string_func ::=   "sprintf" "(" exp "," exp ")"
               |  "str_replace" "(" exp "," exp "," exp ("," exp)? ")"
               |  "str_subreplace" "(" exp "," exp "," exp "," exp ")"
               |  "str_find" "(" exp "," exp ("," exp)? ")"
               |  "str_upper" "(" exp ")"
               |  "str_lower" "(" exp ")"
```



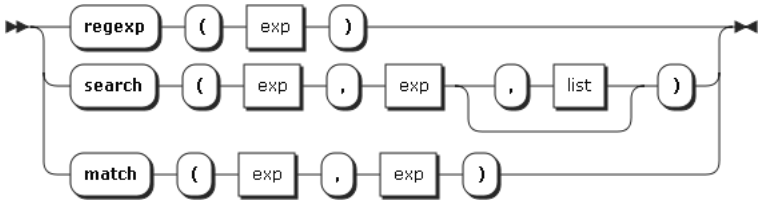
4.11.4. Funciones arrays

```
array_func ::=
 "array_explode" "(" exp "," exp ")"
 |  "array_implode" "(" exp "," exp ")"
 |  "array_chunck" "(" exp "," exp ")"
 |  "array_reduce" "(" exp "," exp ")"
```



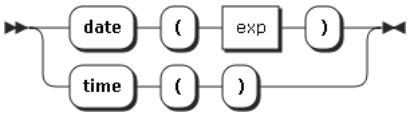
4.11.5. Funciones expresiones regulares

```
regexp_func ::=   "regexp" "(" exp ")"
                |  "search" "(" exp "," exp ("," list)? ")"
                |  "match" "(" exp "," exp ")"
```



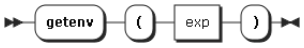
4.11.6. Funciones fechas y tiempo

```
date ::= "date" "(" exp ")"
       |  "time" ("(" ")")
```



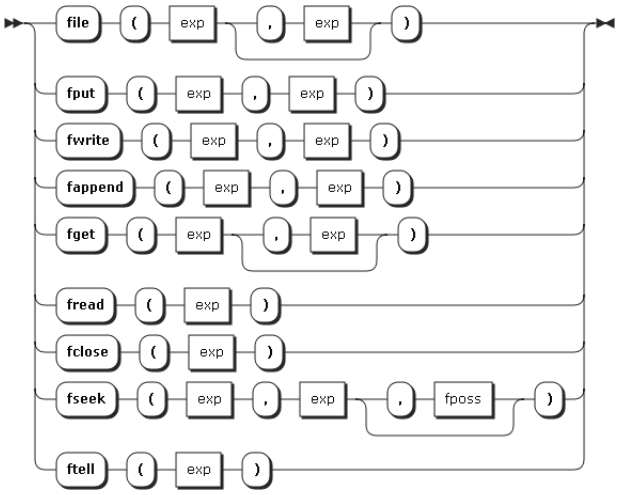
4.11.7. Funciones acceso a entorno

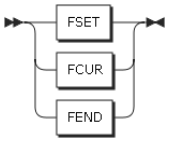
```
environment ::= "getenv" "(" exp ")"
```



4.11.8. Funciones ficheros

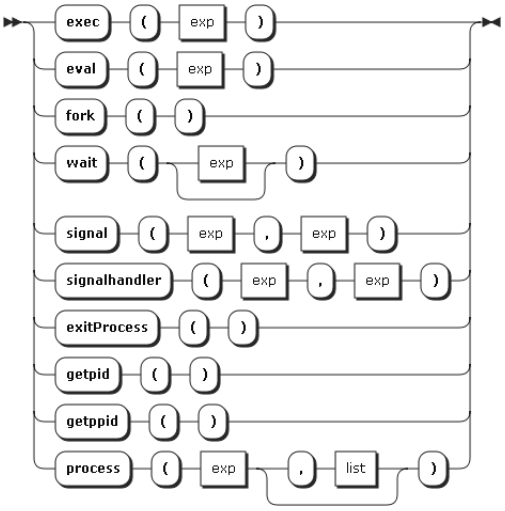
```
file ::= "file" "(" exp ("," exp)? ")"
    | "fput" "(" exp "," exp ")"
    | "fwrite" "(" exp "," exp ")"
    | "fappend" "(" exp "," exp ")"
    | "fget" "(" exp ("," exp)? ")"
    | "fread" "(" exp ")"
    | "fclose" "(" exp ")"
    | "fseek" "(" exp "," exp ("," fposs)? ")"
    | "ftell" "(" exp ")"

fpos ::= FSET
       | FCUR
       | FEND
```





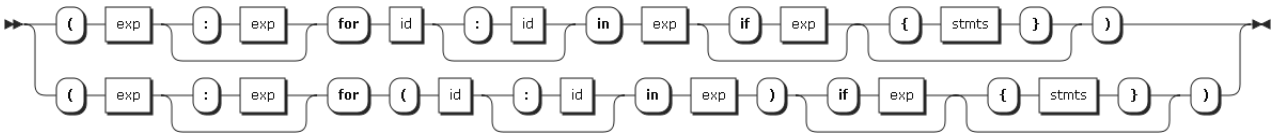
4.11.9. Funciones procesos

```
process ::= "exec" "(" exp ")"
          | "eval" "(" exp ")"
          | "fork" "(" ")"
          | "wait" "(" exp? ")"
          | "signal" "(" exp "," exp ")"
          | "signalhandler" "(" exp "," exp ")"
          | "exitProcess" "(" ")"
          | "getpid" "(" ")"
          | "getppid" "(" ")"
          | "process" "(" exp ("," list)? ")"
```



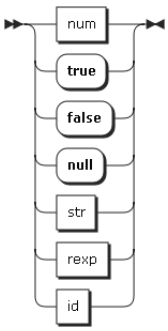
4.12. Generadores

```
generator ::= "(" exp (":" exp)? "for" id (":" id )? "in" exp ( "if" exp )? ("{" stmts "}")? ")"
    | "(" exp (":" exp)? "for" "(" id (":" id )? "in" exp ")" ( "if" exp )? ("{" stmts "}")? ")"
```



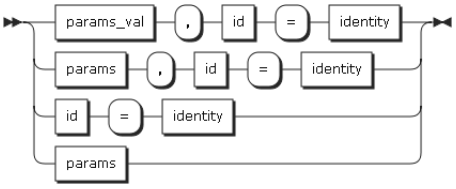
5. Identidades

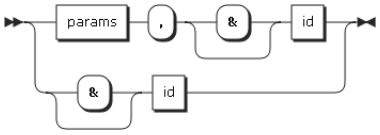
```
identity ::=   num
             |  "true"
             |  "false"
             |  "null"
             |  str
             |  rexp
             |  id
```



5.1. Parámetros

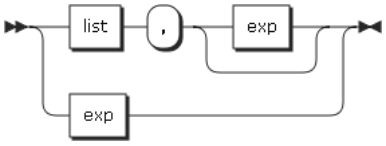
```
params_val ::= params_val "," id "=" identity
   |  params "," id "=" identity
   |  id "=" identity
   |  params

params ::= params "," "&"? id
   |  "&"? id
```



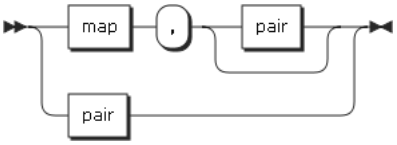


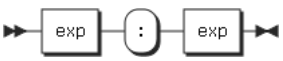
5.2. Listas y pares

```
list ::= list "," exp?
      |  exp
```



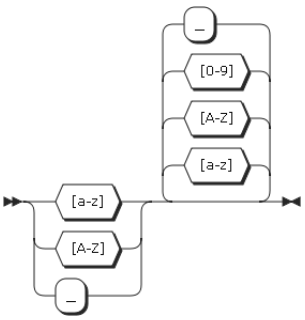

```
map ::=  map "," pair?
      |  pair

pair ::= exp ":" exp
```





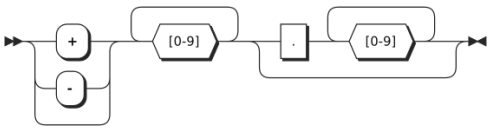
5.3. Identificador

```
id ::= [a-zA-Z_][a-zA-Z0-9_]*
```



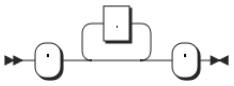
5.4. Números

```
num ::=  ("+"|"-"|) [0-9]+(.[0-9]+)?
```



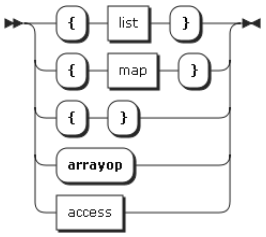
5.5. Cadenas de caracteres

```
str ::= "'".* "'"
```



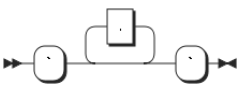
5.6. Array

```
array ::=   "{" list "}"
          |  "{" map "}"
          |  "{""}"
          |  "arrayop"
          |  access
```



5.7. Expresión regular

```
regexp ::= "`".*"`"
```



5.8. Comentarios

```
comments ::=    "/*" [^*]* "*/"
              |  "//"[^\n]
              |  "#" [^#]
```