

⇒ Grammaire de base :

<file> ::= NEWLINE? <def>* <stmt>+ EOF

<def> ::= def <ident> (<ident>*,) : <suite>

<suite> ::= <simple_stmt> NEWLINE

| NEWLINE BEGIN <stmt>+ END

<simple_stmt> ::= return <expr>

| <ident> = <expr>

| <expr> [<expr>] = <expr>

| print (<expr>)

| <expr>

<stmt> ::= <simple_stmt> NEWLINE

| if <expr> : <suite>

| if <expr> : <suite> else : <suite>

| for <ident> in <expr> : <suite>

<expr> ::= <const> | <ident> | <expr> [<expr>] | - <expr> |
not <expr> | <expr> <binop> <expr> | <ident> (<expr>*,) |
[<expr>*,] (<expr>)

<binop> ::= + | - | * | // | % | <= | >= | > | < | != | == | and | or

<const> ::= <integer> | <string> | True | False | None

⇒ Introduction des priorités des opérateurs

<expr> := or_expr | <expr> [<expr>]

<or_expr> := <and_expr>*_{or}

<and_expr> := <not_expr>*_{and}

<not_expr> := not? <comp_expr>

<comp_expr> := <add_expr> (<binop_comp> add_expr) ?

<add_expr> := <mut_expr>*_{<binop_add>}

<mut_expr> := <minus_expr>*_{<binop_mut>}

<minus_expr> := - ? <terminal_expr>

<binop_comp> := <= | >= | > | < | != | ==

<binop_add> := + | -

<binop_mut> := * | // | %

<terminal_expr> := <const> | <ident>

| <ident> (<expr>*,)

| [<expr>*,] (<expr>)

⇒ Explicitation des règles * + et ? avec ou sans symbole :

<exemple>+	<exemple_plus> := <exemple> <exemple_plus_rest> <exemple_plus_rest> := ε <exemple_plus>
<exemple>*	<exemple_etoile> -> ε <exemple> <exemple_etoile>
<exemple>* _{symbole}	<exemple_plus_symbole> := <exemple> <exemple_plus_symbole_rest> <exemple_plus_symbole_rest> := ε symbole <exemple_plus_symbole>
<exemple>* _{symbole}	<exemple_etoile_symbole> := ε <exemple_plus_symbole>
exemple?	<exemple> := ε exemple

⇒ **Elimination de la récursivité à gauche :**

$\langle \text{expr} \rangle ::= \langle \text{or_expr} \rangle \mid \langle \text{expr} \rangle [\langle \text{expr} \rangle]$	$\langle \text{expr} \rangle ::= \langle \text{or_expr} \rangle \langle \text{expr_crochet_etoile} \rangle$ $\langle \text{expr_crochet_etoile} \rangle ::= \varepsilon \mid [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle$
--	--

⇒ **Factorisation des règles :**

$\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle : \langle \text{suite} \rangle$ $\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle : \langle \text{suite} \rangle \text{ else } : \langle \text{suite} \rangle$	$\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle : \langle \text{suite} \rangle \langle \text{stmt_else} \rangle$ $\langle \text{stmt_else} \rangle ::= \varepsilon \mid \text{else } : \langle \text{suite} \rangle$
$\langle \text{terminal_expr} \rangle ::= \dots \mid \langle \text{ident} \rangle \mid \langle \text{ident} \rangle (\langle \text{expr} \rangle^*,) \mid \dots$	$\langle \text{terminal_expr} \rangle ::= \langle \text{ident} \rangle \langle \text{ident_fact} \rangle$ $\langle \text{ident_fact} \rangle ::= \varepsilon \mid (\langle \text{expr} \rangle^*,)$

⇒ **Ambiguïté avec $\langle \text{simple_stmt} \rangle$:**

On a :

$\langle \text{simple_stmt} \rangle ::= \dots \mid \langle \text{expr} \rangle [\langle \text{expr} \rangle] = \langle \text{expr} \rangle$
 $\mid \langle \text{expr} \rangle \mid \dots$

L'ambiguïté apparaît quand on lit le caractère « [» :

- Doit-on utiliser la deuxième règle et ensuite utiliser la règle $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle [\langle \text{expr} \rangle]$
- Doit-on utiliser la première règle

Pour lever cette ambiguïté on fait remonter les membres droits de la règle $\langle \text{expr} \rangle$:

$\langle \text{simple_stmt} \rangle ::= \dots \mid \langle \text{or_expr} \rangle$
 $\mid \langle \text{or_expr} \rangle [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle \mid \dots$

On modifie aussi la première des règles de $\langle \text{simple_stmt} \rangle$ pour pouvoir factoriser le tout

$\langle \text{simple_stmt} \rangle ::= \dots \mid \langle \text{or_expr} \rangle [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle = \langle \text{expr} \rangle$
 $\mid \langle \text{or_expr} \rangle$
 $\mid \langle \text{or_expr} \rangle [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle \mid \dots$

On factorise

$\langle \text{simple_stmt} \rangle ::= \langle \text{or_expr} \rangle \langle \text{simple_stmt_fact} \rangle$

$\langle \text{simple_stmt_fact} \rangle ::= [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle \mid [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle = \langle \text{expr} \rangle$

On refactorise

$\langle \text{simple_stmt} \rangle ::= \langle \text{or_expr} \rangle \langle \text{simple_stmt_fact} \rangle$

$\langle \text{simple_stmt_fact} \rangle ::= [\langle \text{expr} \rangle] \langle \text{expr_crochet_etoile} \rangle \langle \text{simple_stmt_fact_fact} \rangle$

$\langle \text{simple_stmt_fact_fact} \rangle ::= \varepsilon \mid = \langle \text{expr} \rangle$

Il reste tout de même une ambiguïté avec la règle $\langle \text{simple_stmt} \rangle$ qui rend la grammaire ponctuellement LL(2) :

$\langle \text{simple_stmt} \rangle ::= \langle \text{or_expr} \rangle \langle \text{simple_stmt_fact} \rangle$
 $\langle \text{simple_stmt} \rangle ::= \langle \text{ident} \rangle = \langle \text{expr} \rangle$

Sachant que :
 $\langle \text{or_expr} \rangle ::= \langle \text{ident} \rangle$

⇒ **Grammaire finale (syntaxe gramophone) :**

file -> opt_newline def_etoile stmt_plus EOF .

opt_newline -> .

opt_newline -> NEWLINE .

def_etoile -> .

def_etoile -> def def_etoile .

stmt_plus -> stmt stmt_plus_rest .

stmt_plus_rest -> .

stmt_plus_rest -> stmt_plus .

def -> def ident "(" ident_etoile_virgule ")" ":" suite .

ident_etoile_virgule -> .

ident_etoile_virgule -> ident_plus_virgule .

ident_plus_virgule -> ident ident_plus_virgule_rest .

ident_plus_virgule_rest -> .

ident_plus_virgule_rest -> ";" ident_plus_virgule .

suite -> simple_stmt NEWLINE .

suite -> NEWLINE BEGIN stmt_plus END .

simple_stmt -> return expr .

simple_stmt -> print "(" expr ")" .

simple_stmt -> ident "=" expr .

simple_stmt -> or_expr simple_stmt_fact .

simple_stmt_fact -> simple_stmt_fact_fact .

simple_stmt_fact -> "[" expr "]" expr_crochet_etoile .

simple_stmt_fact_fact -> .

simple_stmt_fact_fact -> "=" expr .

stmt -> simple_stmt NEWLINE .

stmt -> for ident in expr ":" suite .

stmt -> if expr ":" suite stmt_else .

stmt_else -> .

stmt_else -> else ":" suite .

expr -> or_expr expr_crochet_etoile .

expr_crochet_etoile -> .

expr_crochet_etoile -> "[" expr "]" expr_crochet_etoile .

or_expr -> and_expr or_expr_rest .

or_expr_rest -> .

or_expr_rest -> binop_or or_expr .

and_expr -> not_expr and_expr_rest .

and_expr_rest -> .

and_expr_rest -> binop_and and_expr .

not_expr -> comp_expr .

not_expr -> not comp_expr .

comp_expr -> add_expr comp_expr_rest .

comp_expr_rest -> .

comp_expr_rest -> binop_comp add_expr .

add_expr -> mut_expr add_expr_rest .

add_expr_rest -> .

add_expr_rest -> binop_add add_expr .

mut_expr -> terminal_expr mut_expr_rest .

mut_expr_rest -> .

mut_expr_rest -> binop_mut mut_expr .

terminal_expr -> "(" expr ")" .

terminal_expr -> const .

terminal_expr -> ident expr_rest_ident .

terminal_expr -> "[" expr_etoile_virgule "]" .

expr_rest_ident -> "(" expr_etoile_virgule ")" .

expr_rest_ident -> .

expr_etoile_virgule -> .

expr_etoile_virgule -> expr_plus_virgule .

expr_plus_virgule -> expr expr_plus_virgule_rest .

expr_plus_virgule_rest -> .

expr_plus_virgule_rest -> ";" expr_plus_virgule .

binop_add -> "+" .

binop_add -> "-" .

binop_mut -> "*" .

binop_mut -> "/" .

binop_mut -> "%" .

binop_comp -> "<=" .

binop_comp -> ">=" .

binop_comp -> ">" .

binop_comp -> "<" .

binop_comp -> "!=" .

binop_comp -> "==" .

binop_and -> and .

binop_or -> or .

const -> integer .

const -> string .

const -> True .

const -> False .

const -> None .

MAIS la grammaire ainsi créée est devenue associative droite et ne respecte donc pas l'associativité gauche.