⇨ **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 et associativités 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> := $<mut\_expr>^{+}_{<binop\_add>}$

<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 :**

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

⇨ **Factorisation des règles :**

<stmt> := if <expr> : <suite> <stmt> := if <expr> : <suite> else : <suite>	<stmt> := if <expr> : <suite> <stmt_else> <stmt_else> := ε | else : <suite>
<terminal_expr> := … | <ident> | <ident> (<expr>*,) | …	<terminal_expr> := <ident> <ident_fact> <ident_fact> := ε | (<expr>*,)

⇨ **Ambiguïté avec <simple_stmt> :**

On a :
<simple_stmt> ::= … | <expr> [<expr>] = <expr>

| <expr> | …

L'ambiguïté apparait quand on lit le caractère « [» :
- Doit-on utilisé la deuxième règle et ensuite utilisé la règle <expr> := <expr> [<expr>]
- Doit-on utiliser la première règle

Pour lever cette ambiguïté on fait remonter les membres droits de la règle <expr> :

<simple_stmt> ::= … | <or_expr>

| <or_expr> [<expr>] <expr_crochet_etoile> | …

On modifie aussi la première des règles de <simple_stmt> pour pouvoir factoriser le tout

<simple_stmt> ::= … | <or_expr> [<expr>] <expr_crochet_etoile> = <expr>

| <or_expr>

| <or_expr> [<expr>] <expr_crochet_etoile> | …

On factorise

<simple_stmt> := <or_expr> <simple_stmt_fact>

<simple_stmt_fact> := [<expr>] <expr_crochet_etoile> | [<expr>] <expr_crochet_etoile> = <expr>

On refactorise

<simple_stmt> := <or_expr> <simple_stmt_fact>

<simple_stmt_fact> := [<expr>] <expr_crochet_etoile> <simple_stmt_fact_fact>

<simple_stmt_fact_fact> := ε | = <expr>

Il **reste tout de même une ambiguïté avec la règle <simple_stmt> qui rend la grammaire ponctuellement LL(2) :**

<simple_stmt> := <or_expr> <simple_stmt_fact>
<simple_stmt> := <ident> = <expr>

Sachant que :
<or_expr> :=$_n$ <ident>

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

file -> opt_newline def_etoile stmt_plus EOF .

opt_newline -> .
opt_newline -> NEWLINE .

def_etoile -> .
def_etoile -> deft def_etoile .

stmt_plus -> stmt stmt_plus_rest .
stmt_plus_rest -> .
stmt_plus_rest -> stmt_plus .

deft -> 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 comp_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 .