- République Algérienne Démocratique et Populaire

Ministère de L'enseignement Supérieure et de la Recherche Scientifique

Université Abderrahman MIRA de Bejaia

Faculté des Sciences Exactes

Département Informatique

Rapport du Projet du Module Compilation

Thème :

20.Les opérateurs ternaires en Python

Réaliser par :

- Oukachbi Chaima.

Groupe :

- B3.

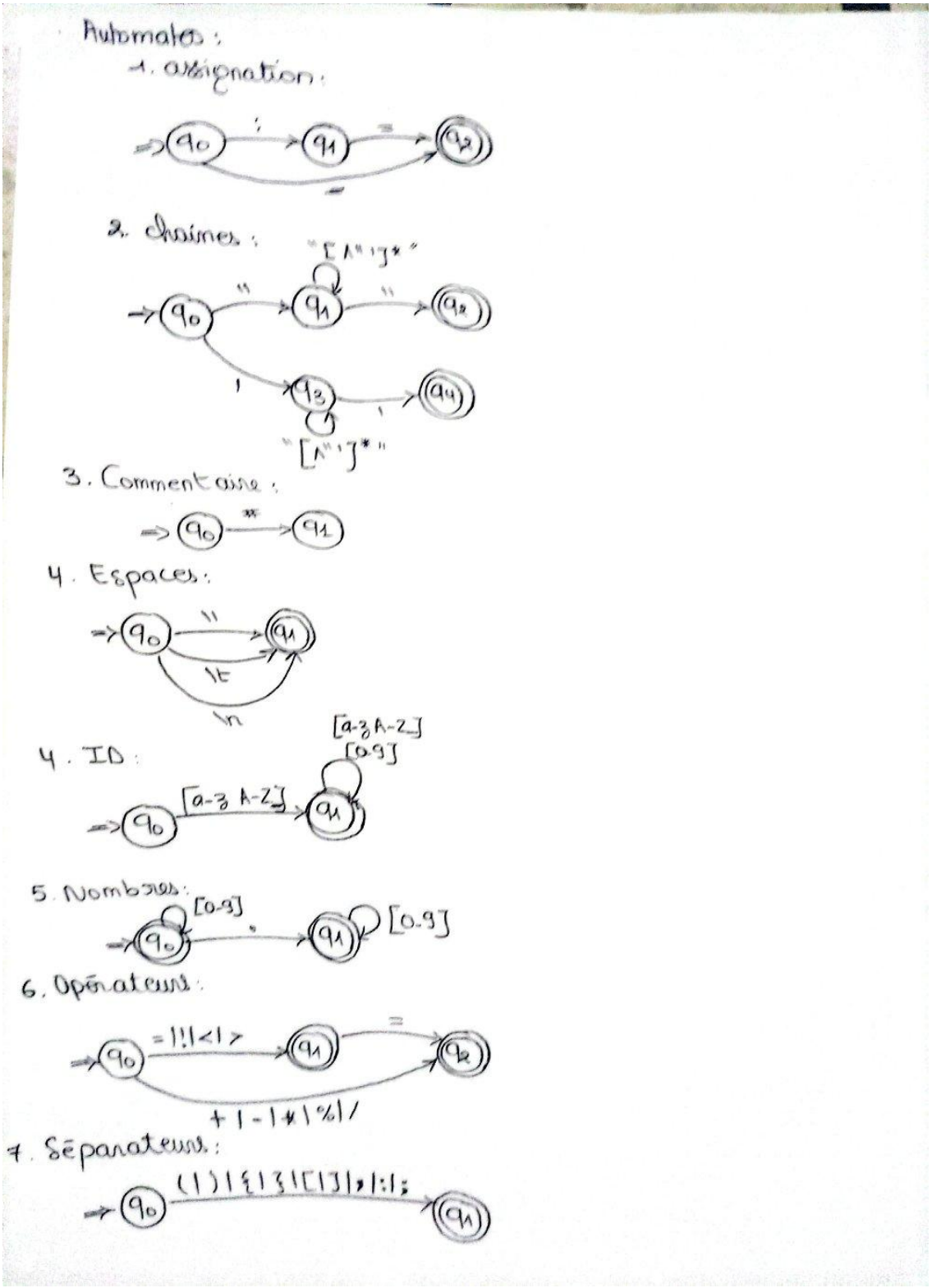
Encadré par :

- Mme Tassoult.

1.Analyseur Lexical :

- La méthode utiliser est la méthode d'automate déterministe. Voici les automates (AFD) et les matrices de transitions utiliser.

1.1. Automates (AFD) :



1.2. Matrices de transitions :

1.1.1. Assignations:

	:	=	autres
Q0	Q1	Q2	-1
Q1	-1	Q2	-1
Q2	-1	-1	-1

1.1.2. Chaines :

	«	‘	autres
Q0	Q1	Q3	-1
Q1	Q2	-1	Q1
Q2	-1	-1	-1
Q3	-1	Q4	Q3
Q4	-1	-1	-1

1.1.3. Commentaires :

	#	autres
Q0	Q1	-1
Q1	-1	-1

1.1.4. Espaces :

	‘ ‘	\n	\t	autres
Q0	Q1	Q1	Q1	-1
Q1	-1	-1	-1	-1

:

1.1.5. ID :

	chiffre	letrres	_	autres
Q0	-1	Q1	Q1	-1
Q1	Q1	Q1	Q1	-1

1.1.6. Nombres:

	Chiffres	.	autres
Q0	Q0	Q1	-1
Q1	Q1	-1	-1

1.1.7. Opérateurs:

	=	!	<	>	+	-	*	/	%
Q0	Q1	Q1	Q1	Q1	Q2	Q2	Q2	Q2	Q2
Q1	Q2	-1	-1	-1	-1	-1	-1	-1	-1
Q2	-1	-1	-1	-1	-1	-1	-1	-1	-1

1.1.8. Séparateurs:

	(	)	{	}	[	]	,	;	:
Q0	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1
Q1	-1	-1	-1	-1	-1	-1	-1	-1	-1

2.Analyseur Syntaxique :

- La méthode utiliser est la méthode de la récursivité descendante, voici la grammaire de type 2 utiliser :

2.1. Grammaire:

S ::= <statement_list>

<statement_list> ::= <statement> <statement_list> | ε

<statement> ::= <assignment> | <expression_stmt>

<assignment> ::= <identifier> "=" <expression>

<expression_stmt> ::= <expression>

<expression> ::= <conditional_expr> | <or_expr>

<conditional_expr> ::= <or_expr> "if" <or_expr> "else" <expression>

<or_expr> ::= <and_expr> ("or" <and_expr>)*

<and_expr> ::= <not_expr> ("and" <not_expr>)*

<not_expr> ::= "not" <not_expr> | <comparison>

<comparison> ::= <arith_expr> (<comp_op> <arith_expr>)*

<comp_op> ::= "==" | "!=" | "<" | ">" | "<=" | ">="

<arith_expr> ::= <term> (<add_op> <term>)*

<add_op> ::= "+" | "-"

<term> ::= <factor> (<mul_op> <factor>)*

<mul_op> ::= "*" | "/" | "%"

<factor> ::= "(" <expression> ")" | <unary_op> <factor> | <primary>

<unary_op> ::= "+" | "-"

<primary> ::= <identifier> | <number> | <string> | <boolean>

<identifier> ::= ID // Comme défini dans l'analyseur lexical (lettres, _, chiffres après)

<number> ::= NUMBER // Int ou Float comme dans checkNombre

<string> ::= CHAINE // Comme dans checkChaine

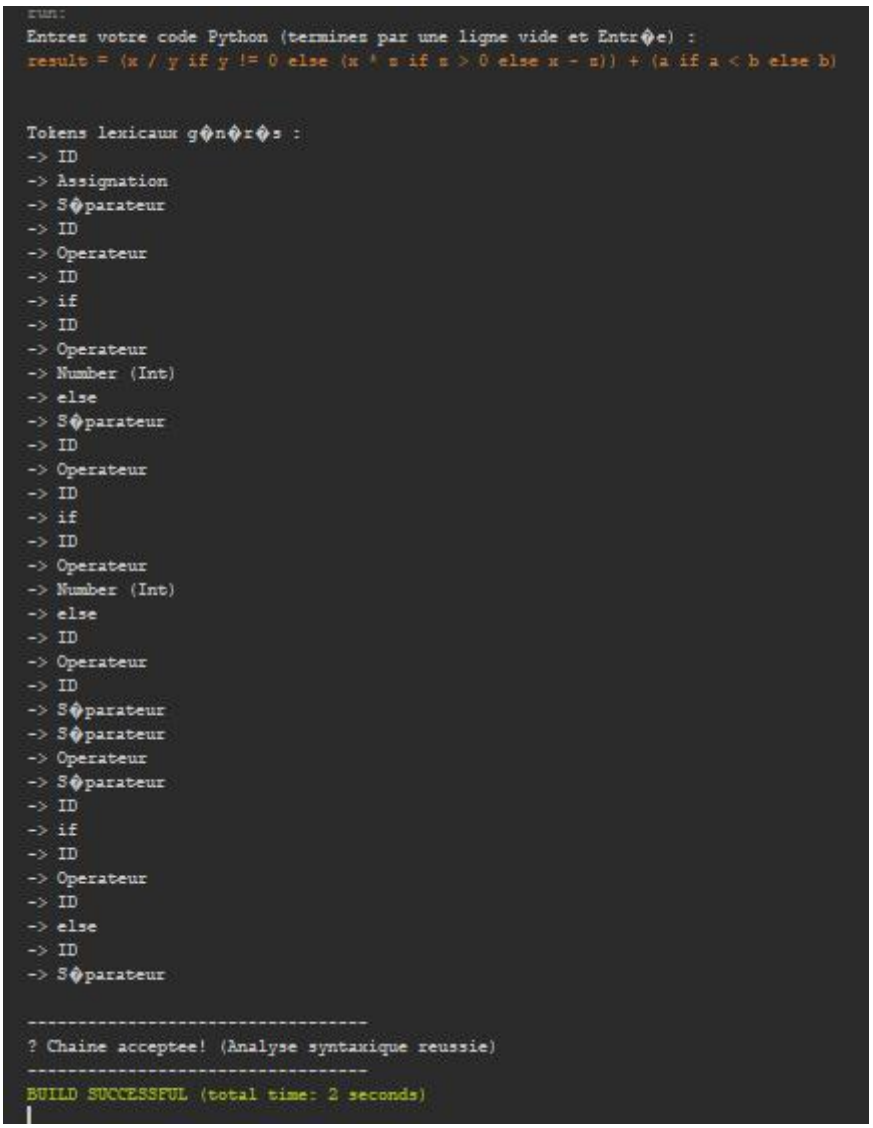
<boolean> ::= "True" | "False"

3. Structure de projet :

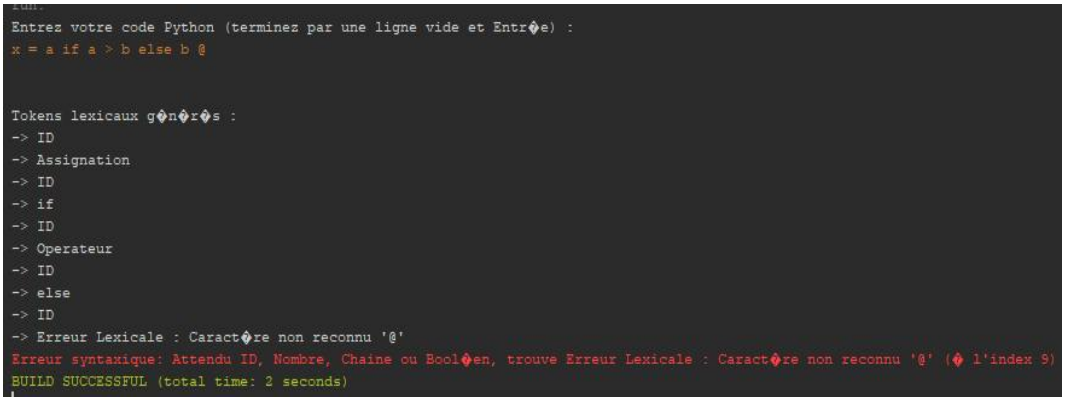
- Mon projet intitulé MiniCompilateur, a un seule package « Analyse », qui est deviser en trois classes, la classe : AnalyseLexical, où se trouve le code de l'analyseur lexical, et la classe AnalyseSyntaxique, où se trouve le code de l'analyseur syntaxique, et finalement la classe principale Main.

4. Cas du test:

- Voici un code python que j'ai tester sur mon projet :



- Voici un code python avec une erreur lexical :



- Voici un code avec une erreur syntaxique :