

|  |
| --- |
| Compilateur |
| Rapport JSCompilator |
| INF3 DLM-b |

|  |
| --- |
| Paysant Adrien - Monnet Joris  2020/2021 |

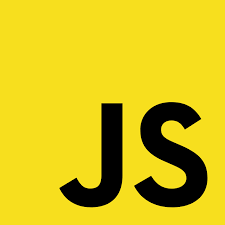
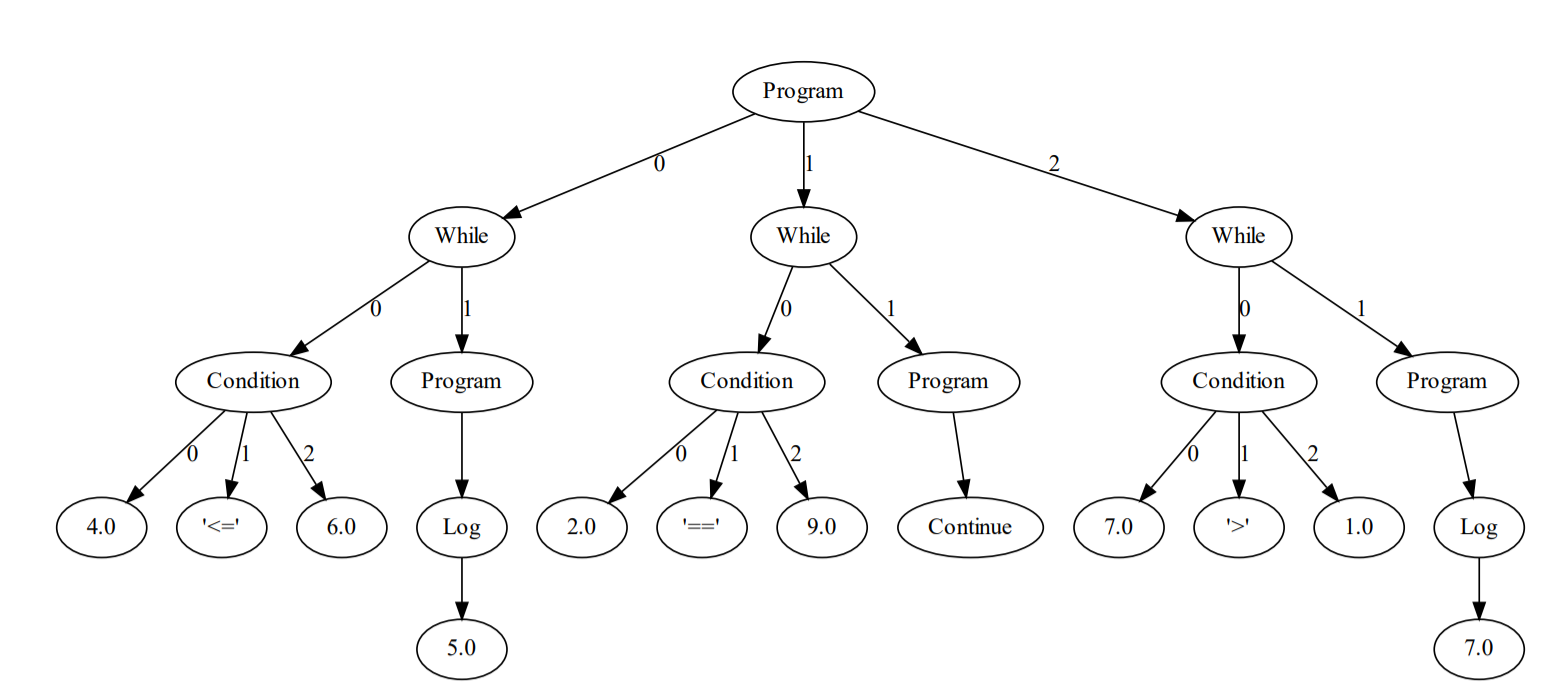


Table des matières

[Introduction 2](#_Toc61053524)

[Objectifs 2](#_Toc61053525)

[Langage 2](#_Toc61053526)

[But 2](#_Toc61053527)

[Objectifs Principaux 2](#_Toc61053528)

[Objectifs Secondaires 2](#_Toc61053529)

[Fonctionnalités Implémentées 2](#_Toc61053530)

[Règles pour les brackets 2](#_Toc61053531)

[Reprise du TP4 2](#_Toc61053532)

[Log 2](#_Toc61053533)

[While 2](#_Toc61053534)

[Raccourci d’opérations pour les expressions 2](#_Toc61053535)

[If/Else 2](#_Toc61053536)

[Opérateur ternaire 2](#_Toc61053537)

[For 2](#_Toc61053538)

[Switch 2](#_Toc61053539)

[Do while 2](#_Toc61053540)

[Var et Let 2](#_Toc61053541)

[Règles permissives pour le ; 2](#_Toc61053542)

[Break et Continue 2](#_Toc61053543)

[Tableaux de nombre et de variables 2](#_Toc61053544)

[Fonctions 2](#_Toc61053545)

[Analyse sémantique du nombre d’arguments 2](#_Toc61053546)

[Return 2](#_Toc61053547)

[Fonctionnalités non-implémentées (partie arrière) 3](#_Toc61053548)

[Pour aller plus loin 3](#_Toc61053549)

[Bugs corrigés 3](#_Toc61053550)

[Guide d’utilisation 3](#_Toc61053551)

[Analyse lexicale 3](#_Toc61053552)

[Analyse syntaxique 3](#_Toc61053553)

[Analyse sémantique 3](#_Toc61053554)

[Partie arrière 3](#_Toc61053555)

[Tests 3](#_Toc61053556)

[Retour d’expérience 3](#_Toc61053557)

[Conclusion 3](#_Toc61053558)

# Introduction

# Objectifs

## Langage

## But

## Objectifs Principaux

## Objectifs Secondaires

# Fonctionnalités Implémentées

## Règles pour les brackets

## Reprise du TP4

### Log

### While

### Raccourci d’opérations pour les expressions

## If/Else

### Opérateur ternaire

## For

## Switch

## Do while

## Var et Let

## Règles permissives pour le ;

## Break et Continue

## Tableaux de nombre et de variables

## Fonctions

## Analyse sémantique du nombre d’arguments

## Return

## Gestion des erreurs

# Fonctionnalités non-implémentées (partie arrière)

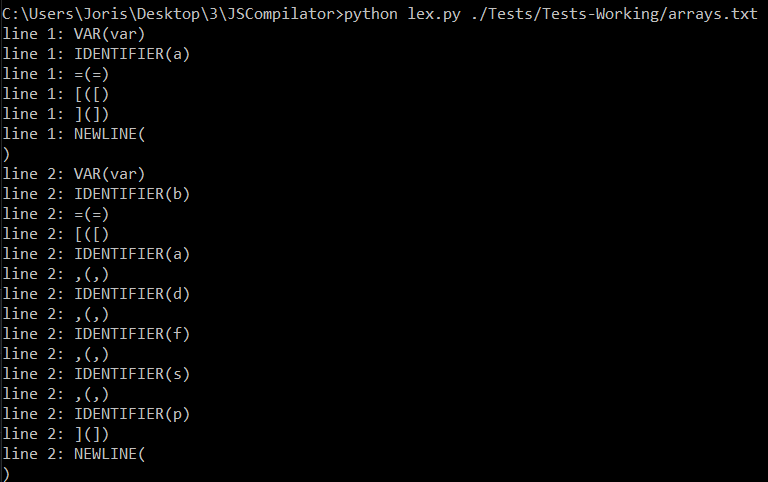
# Pour aller plus loin

# Bugs corrigés

# Guide d’utilisation

## Analyse lexicale

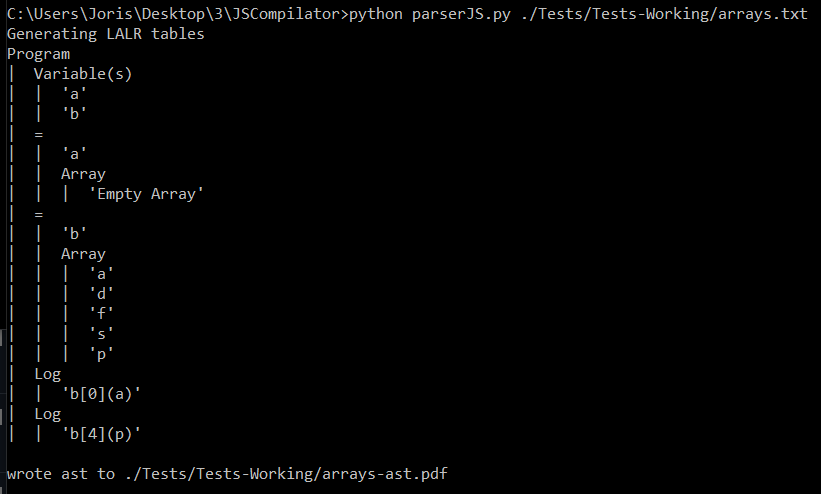
Exactement comme pour les travaux pratiques dont découlent notre projet, pour avoir accès à seulement l’analyse lexicale, lancez le fichier lex.py avec le fichier texte de votre choix en argument, ici un de nos tests préparé :



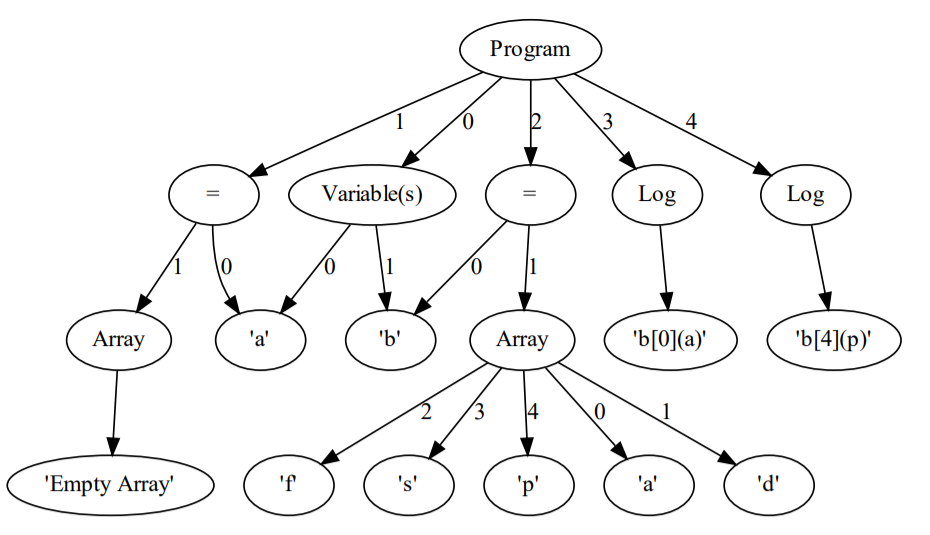
Le résultat est écrit dans la console avec pour chaque groupe de caractère, son groupe lexical si celui-ci est reconnu, sinon une erreur sera affichée à la place.

## Analyse syntaxique

De la même manière, pour lancer l’analyse syntaxique, il faut exécuter le fichier parserJS.py avec en argument un fichier texte :



Cette fois-ci, il y a deux résultats, d’une part l’arbre syntaxique affiché directement dans la console et d’autre part l’arbre syntaxique dessiné par pydot, dans le fichier PDF généré :



Dans le cas d’une erreur, celle-ci sera affichée au-dessus de l’arbre syntaxique dans la console, si ce dernier est grand et que des erreurs sont affichées, pensez à scroller au-dessus de l’arbre, là où les erreurs sont affichées.

## Analyse sémantique

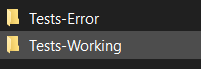
L’analyse sémantique se fait directement dans l’analyse syntaxique, en effet nous n’utilisons pas de couture dans notre projet, pour la tester, lancez donc une analyse syntaxique, notamment sur les fichiers de tests var ou les portées des variables sont testées.

## Partie arrière

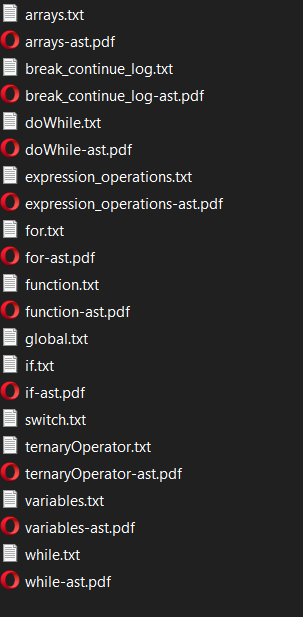
ADRIEN

## Tests

En même temps que ce rapport nous vous proposons une multitude de fichiers de tests pour chaque fonctionnalité. Ils sont rangés dans ces deux dossiers contenus dans Tests/:



Tests-Error ne comprend que des tests erronés permettant notamment de montrer notre gestion des erreurs. Au contraire Tests-Working, regroupe des tests fonctionnels montrant toutes les possibilités de syntaxe et toutes les fonctionnalités de notre compilateur. Leurs noms correspondent à une fonctionnalité testée. Dans le dossier test-Working, nous vous proposons de plus un fichier global qui permet de tester toutes les fonctionnalités en même temps (attention, arbre conséquent qui commence à devenir difficile à lire) :



De plus dans le cas ou vous voudriez tester la syntaxe javascript normale, nous avons un fichier HTML dans le dossier DebugTests/ permettant de rapidement tester une syntaxe (par exemple ;;;;; dans une ligne). Dans le même dossier se trouve un autre fichier texte nommé test.txt qui vous permettra de tester vos propres syntaxes dans notre compilateur.

# Retour d’expérience

# Conclusion