# PseuToPy: Vers un langage de programmation naturel

Yassine Gader, Charles Lefever, Patrick Wang


## To cite this version:

Yassine Gader, Charles Lefever, Patrick Wang. PseuToPy: Vers un langage de programmation naturel. Atelier " Apprendre la Pensée Informatique de la Maternelle à l'Université ", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), 2021, Fribourg, Suisse. pp.87-95. hal-03241688

## HAL Id: hal-03241688
## https://hal.science/hal-03241688v1

Submitted on 28 May 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# PseuToPy: Vers un langage de programmation naturel

Yassine Gader[1,2] Charles Lefever[1] Patrick Wang[1]
[1] ISEP – Institut Supérieur d'Électronique de Paris
10 rue de Vanves, Issy les Moulineaux, 92130 France
[2] ESPRIT – École Supérieure Privée d'Ingénierie et de Technologie
1, 2 rue André Ampère – 2083 – Pôle Technologique – El Ghazala, 2083 Ariana, Tunisie
{yassine.gader, charles.lefever}@eleve.isep.fr
patrick.wang@isep.fr

## RÉSUMÉ

L'apprentissage de la programmation repose souvent sur la présentation de concepts algorithmiques puis sur leur mise en application avec un langage de programmation. Lorsqu'un langage de programmation textuel est utilisé (en opposition à un langage de programmation par blocs), l'apprentissage de la grammaire et de la syntaxe de ce langage peuvent constituer une difficulté supplémentaire pour l'apprenant. D'autre part, les langages de programmation reposent souvent sur un vocabulaire tiré de la langue anglaise. Cet aspect peut constituer un obstacle pour un public d'apprenants non anglophones puisqu'ils manipuleraient des mots et construiraient des instructions sans en comprendre leurs sens. Dans cet article, nous identifions les caractéristiques des langages de programmation qui peuvent être à l'origine de difficultés pour les apprenants. Puis, nous présentons la conception et l'implémentation de PseuToPy, un langage de programmation naturel qui permet la construction d'instructions proches de la langue naturelle de l'apprenant. Ce travail étant encore dans un état précoce, nous formulons l'hypothèse qu'un tel langage est utile dans l'apprentissage de la programmation et facilite la transition vers un langage comme Python.

## ABSTRACT

**PseuToPy : Towards a natural programming language**

Learning to program frequently relies on the introduction of algorithmic notions followed by their application using a programming language. However, such text-based programming languages (as opposed to block-based ones) can be difficult to learn because of their unnatural grammar and syntax rules. Moreover, most programming languages use English words which can represent an added difficulty for non-English speakers who wish to learn to program as they might not understand the actual meaning of the instructions they are writing. In this paper, we identify the features of programming languages that can lead to learning difficulties. Then, we introduce the design and implementation of PseuToPy, a programming language that allows for instructions that resemble natural language sentences. Because this work is still in an early stage, we hypothesize that such a language can be useful when learning to program and can also ease the transition towards another programming language such as Python.

MOTS-CLÉS : Langages de programmation, Langage naturel, Grammaire formelle, Pseudocode.

KEYWORDS: Programming language, Natural language, Formal grammar, Pseudocode.

 Attribution 4.0 International.

# 1 Introduction

L'apprentissage de la programmation présente de nombreuses difficultés. Bien souvent, cet apprentissage comporte deux composantes distinctes. D'un côté, il y a l'apprentissage des concepts importants de l'informatique permettant la conception d'algorithmes (par exemple, les variables ou les structures de contrôle). De l'autre, il y a l'apprentissage d'un langage de programmation et de ses règles de syntaxe et de sémantique afin de pouvoir exécuter ces algorithmes sur une machine.

Dans cet article, nous nous intéressons aux difficultés liées à l'apprentissage d'un langage de programmation textuel (en opposition aux langages de programmation par blocs tels que Scratch). Du Boulay (1986) avait déjà identifié cet obstacle qui oblige les apprenants à maîtriser la syntaxe et la grammaire d'un langage de programmation pour pouvoir écrire les programmes même les plus simples. La question de recherche que nous nous posons est la suivante : "Quelles sont les caractéristiques des langages de programmation *textuels* qui peuvent être à l'origine de difficultés d'apprentissage chez les débutants en programmation ?"

Dans cet article, nous cherchons à fournir des éléments de réponse à cette question de recherche avec, en Section 2, une description des caractéristiques grammaticales, syntaxiques, et sémantiques des langages de programmation et qui peuvent donc engendrer des difficultés d'apprentissage. En tenant compte de ces caractéristiques, nous travaillons sur la conception d'un langage de programmation *presque naturel*, intitulé PseuToPy, dont les objectifs sont justement de gommer un maximum ces caractéristiques. La conception et l'implémentation de ce langage de programmation (qui est toujours en cours de réalisation) est décrit en Section 3. Dans la Section 4, nous expliquons en quoi notre démarche se distingue des approches trouvées dans l'état de l'art portant sur l'apprentissage de la programmation. Enfin, nous concluons cet article en décrivant dans quelles situations nous comptons utiliser PseuToPy afin de fournir une réponse définitive à la question de recherche énoncée précédemment. Nous lançons également un appel à la communauté à contribuer à la conception de PseuToPy et de sa grammaire.

# 2 Caractéristiques d'un langage de programmation

## 2.1 Grammaire et syntaxe

Une grammaire formelle permet de spécifier les règles d'écriture des instructions d'un langage de programmation. Ces grammaires sont ensuite utilisées par un analyseur syntaxique afin d'assurer la conformité d'une instruction. Or, en spécifiant les instructions valides ou non, ces grammaires peuvent freiner l'apprentissage d'un langage de programmation de multiples façons.

Une première façon concerne l'aspect "peu naturel" des instructions écrites avec un langage de programmation. En effet, la résolution d'un problème passe souvent par une phase de conception d'un algorithme. Il semble évident qu'un apprenant va réfléchir et utiliser sa langue maternelle pour concevoir au préalable son algorithme. Une fois cette étape réalisée, il devra ensuite retranscrire cette réflexion en utilisant un langage de programmation dont les constructions et règles de grammaire sont probablement différentes de celles de sa langue naturelle. Cette étape nécessaire de retranscription peut donc être source de difficultés pour l'apprenant.

De plus, la grammaire d'un langage de programmation définit précisément la construction d'une

instruction. Par exemple, en Python, il est nécessaire d'avoir une valeur booléenne après le mot-clé `if`, quelle que soit la forme prise par cette valeur booléenne (un nom de variable booléenne, le résultat d'une comparaison, la valeur `True`, etc.). Cet apprentissage des règles de grammaire est indispensable, mais constitue donc un frein à l'élaboration de programmes, même les plus simples.

Enfin, les règles de grammaire ne laissent pas de place aux erreurs de syntaxe. En Python, l'oubli d'une indentation directement après une structure conditionnelle va générer une erreur de syntaxe. Il en va de même avec l'oubli d'un point-virgule à la fin d'une instruction en Java. Ces erreurs de syntaxe font partie des erreurs les plus communément retrouvées dans les programmes des apprenants. En effet, Denny et al. (2011) ont constaté la présence d'erreurs de syntaxe dans les rendus de près de 73% des étudiants considérés comme "faibles". De même, Altadmri and Brown (2015) ont analysé 37 millions de compilations en Java et ont constaté que l'erreur la plus fréquente, retrouvée près de 800 000 fois, correspondait à un oubli de parenthèses fermantes.

Ces caractéristiques montrent donc l'importance de mettre l'accent sur les règles de grammaire et de syntaxe d'un langage de programmation lors de son apprentissage. Les méthodes d'enseignement reposent souvent sur la présentation des concepts, mais d'autres approches mériteraient d'être étudiées. Par exemple, Portnoff (2018) suggère qu'il faudrait enseigner un langage de programmation comme une langue étrangère : en se focalisant justement sur la grammaire et l'orthographe de ce langage.

## 2.2 Sémantique

La sémantique d'un langage de programmation fait référence à la signification des mots et des symboles utilisés dans ce langage. Comme tout langage naturel, chaque langage de programmation donne aussi un sens particulier à l'ensemble des mots et symboles qu'il utilise, ce qui entraîne parfois des confusions chez l'apprenant.

Un exemple classique relevé par Qian and Lehman (2017) concerne le symbole "=". En effet, ce symbole en mathématiques est ordinairement utilisé pour représenter une égalité entre deux grandeurs. Mais le sens de ce symbole dans certains langages de programmation comme Java ou Python est complètement différent puisqu'il signifie l'affectation d'une valeur à une variable. Un autre exemple est celui des parenthèses, crochets, et accolades en Python qui permettent de construire respectivement des tuples, des listes, ou des ensembles ou dictionnaires. Un troisième exemple reprend les erreurs de syntaxe mentionnées précédemment : en Python, les indentations ont un réel sens dans la construction d'un algorithme puisqu'elles permettent de spécifier des blocs d'instructions d'un même niveau. On constate donc que ces indentations peuvent générer aussi bien des erreurs de syntaxe que des erreurs sémantiques. En plus de la syntaxe à maîtriser, l'apprenant doit donc aussi connaître le sens des différents symboles utilisés par le langage de programmation.

Une autre source de difficultés pour les apprenants résulte du fait que les langages de programmation les plus courants utilisent souvent un vocabulaire tiré de la langue anglaise. Or, en 2021, l'anglais est la langue maternelle d'environ seulement 5% de la population mondiale [1]. Ce nombre passe à 18% si l'on compte également les personnes pour qui l'anglais est une deuxième langue. Dès lors, comment pouvons-nous espérer qu'une personne non anglophone apprenne sans difficulté un langage de programmation dont elle ne comprendrait même pas le sens des mots ? D'ailleurs, une corrélation positive a été trouvée par Qian and Lehman (2016) entre le niveau d'anglais et les résultats obtenus dans un cours d'introduction à la programmation dispensé dans un collège en Chine. Pour remédier

1. https://en.wikipedia.org/wiki/List_of_languages_by_total_number_of_speakers

à ce problème, il faudrait que l'apprenant s'initie dans un premier temps à l'anglais, ou qu'on lui fournisse un langage de programmation dans sa propre langue natale.

Dans la suite de cet article, nous présentons notre approche qui consiste donc à concevoir et proposer aux apprenants un langage de programmation *naturel* et qui pourrait se décliner en plusieurs langues.

# 3 PseuToPy : un langage de programmation presque naturel

## 3.1 Conception et grammaire formelle du langage

Dans cet article, nous présentons le travail que nous menons actuellement sur PseuToPy, un langage de programmation destiné aux apprenants et présentant les caractéristiques suivantes : (1) la grammaire formelle de PseuToPy est conçue pour produire des instructions qui ressemblent à des phrases d'une langue naturelle sans pour autant être forcément grammaticalement correctes dans la langue cible, (2) plusieurs versions de la grammaire permettrait également de définir des instructions dans des langues autres que l'anglais, et (3) certaines instructions possèdent plusieurs syntaxes valides afin d'offrir plus de flexibilité à l'apprenant. Puis, PseuToPy propose aussi de convertir les instructions ainsi écrites en code Python équivalent. Avec ce langage, nous espérons faciliter l'apprentissage d'un langage de programmation en permettant aux apprenants d'écrire un algorithme dans un langage *presque* naturel, de visualiser leurs programmes en Python, et de les exécuter.

La conception de PseuToPy passe par plusieurs phases. La première consiste à identifier les instructions qu'un débutant pourrait être amené à écrire dans le cadre de son apprentissage de la programmation. Lorsque ces instructions sont identifiées, il s'agit ensuite de définir les règles de la grammaire formelle correspondante afin de spécifier le langage en cours de construction. Ce travail pour PseuToPy est encore en cours de réalisation ; en effet, nous cherchons à présent à trouver des formulations alternatives avec deux objectifs visés : le premier concerne la possibilité de construire des instructions proches du langage naturel cible, le second concerne la possibilité d'ajouter de la sémantique aux instructions en explicitant certains symboles pouvant générer de la confusion chez les apprenants. Enfin, lorsque la grammaire sera complètement spécifiée, la dernière étape de la conception de PseuToPy passe par l'implémentation de fonctions qui traduisent le code écrit en PseuToPy en du code Python équivalent.

Pour réaliser ce travail de spécification de la grammaire, et comme nous allons le présenter plus en détail dans la Section 3.2, nous avons pris la décision de nous baser sur la grammaire formelle du langage de programmation Python et de modifier ces règles en définissant de nouveaux symboles terminaux qui aideront ensuite l'apprenant à construire des instructions proches de sa langue naturelle. Ce choix est justifié par la nécessité d'écrire des instructions ayant une sémantique non ambiguë (ce qui est rendu possible grâce à la grammaire formelle) et par la possibilité de créer des instructions dont les constructions ressembleront au langage Python (puisque basées sur la même grammaire) avec l'espoir que cette ressemblance facilite la transition du langage PseuToPy vers Python.

L'approche que nous adoptons présente donc de multiples intérêts. Elle permet de désambiguïser certains symboles utilisés en Python, de générer des instructions proches du langage naturel, mais aussi de gérer de façon indépendante plusieurs langues naturelles dans un soucis d'inclusivité. Bien que PseuToPy reste un travail en cours de réalisation, nous formulons l'hypothèse que ces caractéristiques faciliteront l'apprentissage de ce langage par des apprenants débutant en programmation.

## 3.2 Implémentation du langage

Pour l'implémentation de PseuToPy, nous utilisons la bibliothèque Lark [2]. Cette bibliothèque permet de spécifier une grammaire en utilisant le formalisme *Extended Backus-Naur Form* (EBNF), et de procéder à une analyse syntaxique des instructions en respect des règles de grammaire spécifiées au préalable. Dans le cadre de PseuToPy, notre travail consiste donc à spécifier les règles de grammaire du langage que nous souhaitons créer puis de transformer l'arbre de syntaxe produit par Lark en un programme Python.

Pour ce qui est de la grammaire du langage, nous nous sommes basés sur la grammaire de Python 3.8 [3] à laquelle nous avons ajouté nos propres règles et mots-clés, et ce, dans la langue naturelle de notre choix (que ce soit par exemple en anglais ou en français). Une conséquence de cette décision est qu'il est donc possible d'analyser la syntaxe d'instructions écrites aussi bien en Python qu'en PseuToPy. Une autre conséquence est qu'il est donc possible de mélanger des instructions écrites en Python ou dans la langue naturelle cible puisqu'il n'y a qu'une seule et unique grammaire formelle.

Par exemple, la Figure 1 montre les modifications que nous avons apportées à la règle `assign` pour y ajouter une instruction en français permettant d'assigner une valeur à une variable. La première ligne indique le nom de la règle spécifiée. Les deux lignes suivantes permettent de définir cette règle en deux temps (le symbole "`|`" correspond à un "ou" logique) : soit pour spécifier une assignation en Python, soit pour spécifier une assignation utilisant des mots en français.

```
?assign:
  (testlist_star_expr ("=" (yield_expr|testlist_star_expr))*)
  | ("assigner" "à" testlist_star_expr ("la" "valeur" (yield_expr | testlist_star_expr))*)
```

FIGURE 1 – Modification apportée à la règle d'assignation de valeur à une variable.

Avec cette règle modifiée, les instructions utilisées en Figure 2 sont équivalentes comme l'illustrent les arbres de syntaxe identiques générés par Lark.

```
Instruction en PseuToPy:
assigner à ma_variable la valeur vrai
************************
Arbre de syntaxe généré:
file_input
  assign
    var ma_variable
    const_true
```

```
Instruction en Python:
ma_variable = True
************************
Arbre de syntaxe généré:
file_input
  assign
    var ma_variable
    const_true
```

FIGURE 2 – Arbres de syntaxe générés pour deux instructions équivalentes, l'une écrite en PseuToPy et l'autre en Python.

La conception des règles alternatives liées au langage naturel reste pour l'instant inachevée, car il nécessite un travail important de spécification. Par exemple, l'instruction donnée en Figure 2 utilise les mots-clés `assigner`, `à`, `la`, `valeur`, et `vrai`. Or, l'assignation d'une valeur à une variable

2. https://github.com/lark-parser/lark
3. La grammaire de Python 3.8 est disponible ici : https://docs.python.org/3.8/reference/grammar.html.

pourrait se formuler d'une autre façon, par exemple "`mettre ma_variable à vrai`". Quelle instruction devrions-nous autoriser ? Devrions-nous modifier la grammaire pour autoriser ces deux instructions ? Ces questions restent en suspens, et notre réponse initiale est de penser que, dans l'exemple ci-dessus, les instructions sont équivalentes et qu'il serait intéressant de pouvoir proposer plusieurs syntaxes pour une même règle. Cela peut d'ailleurs être fait en rajoutant simplement une autre possibilité à cette règle pour spécifier l'instruction précédente. Ainsi, la grammaire offrirait un peu plus de flexibilité dans les instructions que peut écrire un apprenant. Le travail réalisé pour la règle `assign` nécessite ensuite d'être aussi fait pour toutes les autres règles spécifiant des instructions qu'un apprenant débutant en programmation pourrait utiliser lors de son apprentissage.

Une fois ce travail d'extension de la grammaire réalisé, il restera un travail de développement illustré ci-après en Figure 3 et consistant à implémenter une fonction prenant en entrée une instruction écrite en PseuToPy et qui (1) procède à l'analyse syntaxique de cette instruction puis produit un arbre de syntaxe comme celui illustré en Figure 2 et (2) construit et retourne l'instruction Python équivalente à partir de cet arbre de syntaxe.

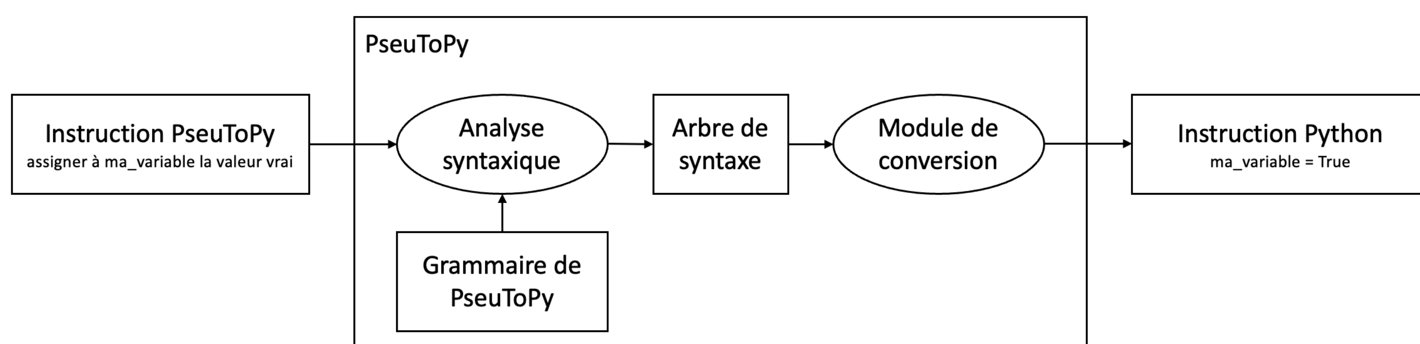


FIGURE 3 – Implémentation technique de PseuToPy.

Enfin, ce travail devra ensuite être répété pour les différentes langues naturelles que nous souhaitons proposer à la communauté d'apprenants et d'enseignants. Pour le moment, nous nous focalisons simplement sur le français et l'anglais en espérant plus tard ajouter d'autres langues. En guise d'illustration, voici les instructions de la Figure 2 écrites en arabe ou en chinois. Cette opportunité est d'autant plus intéressante qu'il est donc possible d'utiliser des caractères encodés en UTF-8 pour définir la grammaire de Lark et aussi d'utiliser des langages s'écrivant de droite à gauche.

```
Instruction en PseuToPy:
ضع x صحيح بقيمة
***********************
Arbre de syntaxe généré:
file_input
  assign
    const_true
    var x
```

```
Instruction en PseuToPy:
设置 x 等于正确
***********************
Arbre de syntaxe généré:
file_input
  assign
    var x
    const_true
```

FIGURE 4 – Exemples d'assignation de variables écrites en arabe et en chinois.

Le résultat de ce projet sera ensuite mis à disposition de la communauté sous deux formes différentes :

une bibliothèque Python qui sera disponible sur la plateforme PyPI [4], et une application Web proposant un éditeur permettant d'écrire des instructions en PseuToPy, de voir les instructions équivalentes en Python, et d'exécuter et visualiser le résultat de ces instructions.

# 4 Positionnement et état de l'art

La littérature en didactique de l'informatique présente de nombreux langages et environnements conçus spécifiquement pour l'apprentissage de la programmation. Brusilovsky et al. (1994) ont par exemple identifié trois approches utilisées pour l'enseignement de la programmation.

La première approche est qualifiée d'*incrémentale* et consiste à proposer un langage dans lequel l'apprenant construit au fur et à mesure les éléments de ce langage (par exemple DrScheme, dont le nouveau nom est désormais Racket (Findler et al., 2002; Felleisen et al., 2015)). Or, cette approche repose sur le paradigme de la programmation fonctionnelle et nécessite des connaissances préalables et n'est donc pas particulièrement adaptée à des débutants en programmation.

La seconde approche propose de simplifier un langage de programmation en n'en fournissant qu'une sous-partie adaptée aux besoins des apprenants. Par exemple, MiniJava (Roberts, 2001) propose une simplification syntaxique du langage Java, connu pour être verbeux. Cependant, le langage proposé n'étant qu'une sous-partie du langage initial, les différents symboles restent inchangés dans la grammaire, malgré qu'ils soient souvent à l'origine des erreurs les plus communément rencontrées en Java (Brown and Altadmri, 2017). Cette approche ne traite donc pas spécifiquement des difficultés d'apprentissage du langage de programmation et de ses règles de grammaire. Dans cette même optique de simplification, certains travaux cherchent à concevoir un environnement de développement adapté aux besoins des apprenants, comme par exemple BlueJ (Kölling et al., 2003) qui propose une interface permettant de construire un programme Java en définissant un diagramme UML de classe.

La dernière approche consiste à utiliser un langage construit spécifiquement pour l'initiation à la programmation. Récemment, les environnements de programmation par blocs comme Scratch (Maloney et al., 2010) ont été conçus pour focaliser l'attention de l'apprenant sur l'aspect algorithmique et limiter les erreurs de syntaxe en fournissant des blocs pré-construits. De nombreux travaux mettent ainsi en avant ce type d'environnements (Bau et al., 2015; Weintrop and Wilensky, 2017) mais très peu de langages de programmation textuels ont été conçus pour des débutants.

Toutes ces approches ne mettent pas nécessairement l'accent sur l'apprentissage des règles de syntaxe d'un langage de programmation. Dans le cas des environnements de programmation par blocs, cet aspect est même complètement mis de côté puisque des blocs prédéfinis sont utilisés pour remplacer les instructions textuelles. Bien que des travaux ont été menés pour étudier la transition entre un langage de programmation par blocs à un langage textuel (Bau et al., 2015; Weintrop and Holbert, 2017), aucune conclusion n'a pu être tirée sur l'utilité du premier pour l'apprentissage des règles de syntaxe et de grammaire du second.

PseuToPy se distingue donc de tous ces exemples en se focalisant sur les règles de grammaire des langages de programmation avec pour objectif de construire des instructions proches du langage naturel. À notre connaissance, peu de travaux ont été conduits sur cette problématique particulière. Nous pouvons par exemple mentionner MicroAlg (Gragnic, 2015) qui est un langage fonctionnel utilisant des mots de la langue française. Mais celui-ci repose sur un système de parenthèses similaire

4. Python Package Index : https://pypi.org/. Une version existe déjà, mais est en train d'être retravaillée en profondeur.

au langage Lisp, ce qui conduit à des instructions difficiles à lire pour des débutants en programmation, en plus d'être éventuellement déroutant. Nous pouvons également mentionner les travaux de Hermans ayant pour objectifs (1) de définir une *phonologie* des langages de programmation afin de déterminer précisément comment un programme peut se lire à voix haute (Hermans et al., 2018) et (2) de concevoir un langage de programmation (nommé Hedy) destinés à de jeunes enfants et dont les règles de grammaire présentent différents niveaux de complexité pour s'adapter à l'apprenant (Hermans, 2020). En particulier, les éléments syntaxiques comme les parenthèses ne sont introduits que dans les derniers niveaux pour ne pas surcharger les apprenants avec des syntaxes trop complexes dès le départ. PseuToPy se distingue de Hedy en proposant une compatibilité complète avec Python.

# 5 Discussion et travaux futurs

Dans cet article, nous avons analysé les caractéristiques syntaxiques et sémantiques des langages de programmation qui peuvent engendrer des difficultés lors de l'apprentissage de la programmation. Pour atténuer ces difficultés, nous proposons de construire un langage de programmation dont les instructions ressembleraient à des phrases issues d'une langue naturelle. Bien que nous travaillions en ce moment sur une version française, nous souhaitons proposer PseuToPy en plusieurs langages pour justement casser la barrière de la langue qui peut exister lors de l'apprentissage de la programmation.

À ce titre, la suite de nos travaux consiste en la construction de la grammaire française de PseuToPy pour laquelle nous souhaitons solliciter l'aide de la communauté francophone de chercheurs, d'éducateurs, et d'apprenants afin d'identifier les futurs mots-clés composant cette grammaire. Puis, nous étudierons l'utilisation de cet outil pour l'apprentissage de la programmation avec comme objectif de déterminer si PseuToPy facilite cet apprentissage du fait de sa ressemblance avec une langue naturelle.

# Remerciements

Les auteurs de cet article souhaitent remercier Marin Godechot, Florian Comte, et Eric Sombroek pour leurs contributions initiales à la conception de PseuToPy, ainsi que Bastien Marais, Mathieu Valentin, Sébastien Viguier, et Laurent Yu pour leurs contributions à la nouvelle version de PseuToPy et pour leurs relectures de cet article.

# Références

Altadmri, A. and Brown, N. C. (2015). 37 Million Compilations : Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 522–527, New York, NY, USA. Association for Computing Machinery.

Bau, D., Dawson, M., and Pickens, C. S. (2015). Pencil code : Block code for a text world. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 445–448, New York, NY, USA. Association for Computing Machinery.

Brown, N. C. C. and Altadmri, A. (2017). Novice Java Programming Mistakes : Large-Scale Data vs. Educator Beliefs. *ACM Transactions on Computing Education*, 17(2) :7 :1–7 :21.

Brusilovsky, P., Kouchnirenko, A., Miller, P., and Tomek, I. (1994). Teaching Programming to Novices : A Review of Approaches and Tools. In *World Conference on Educational Multimedia and Hypermedia*, Vancouver, BC.

Denny, P., Luxton-Reilly, A., Tempero, E., and Hendrickx, J. (2011). Understanding the syntax barrier for novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 208–212, New York, NY, USA. Association for Computing Machinery.

Du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, 2(1) :57–73.

Felleisen, M., Findler, R. B., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., and Tobin-Hochstadt, S. (2015). The Racket Manifesto. In Ball, T., Bodik, R., Krishnamurthi, S., Lerner, B. S., and Morrisett, G., editors, *1st Summit on Advances in Programming Languages (SNAPL 2015)*, volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113–128, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Findler, R. B., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P., and Felleisen, M. (2002). DrScheme : A programming environment for Scheme. *Journal of Functional Programming*, 12(2) :159–182.

Gragnic, C. (2015). MicroAlg, un langage de programmation pour débutants - Les nouvelles technologies pour l'enseignement des mathématiques. *MathémaTICE*, 46.

Hermans, F. (2020). Hedy : A Gradual Language for Programming Education. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ICER '20, pages 259–270, New York, NY, USA. Association for Computing Machinery.

Hermans, F., Swidan, A., and Aivaloglou, E. (2018). Code phonology : An exploration into the vocalization of code. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, pages 308–311, New York, NY, USA. Association for Computing Machinery.

Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003). The BlueJ System and its Pedagogy. *Computer Science Education*, 13(4) :249–268.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4) :16 :1–16 :15.

Portnoff, S. R. (2018). The introductory computer programming course is first and foremost a *language* course. *ACM Inroads*, 9(2) :34–52.

Qian, Y. and Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review. *ACM Transactions on Computing Education*, 18(1) :1 :1–1 :24.

Qian, Y. and Lehman, J. D. (2016). Correlates of Success in Introductory Programming : A Study with Middle School Students. *Journal of Education and Learning*, 5(2) :73–83.

Roberts, E. (2001). An overview of MiniJava. *ACM SIGCSE Bulletin*, 33(1) :1–5.

Weintrop, D. and Holbert, N. (2017). From Blocks to Text and Back : Programming Patterns in a Dual-Modality Environment. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 633–638, New York, NY, USA. Association for Computing Machinery.

Weintrop, D. and Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, 18(1) :3 :1–3 :25.