Création d'exemples résolus avec objectifs étiquetés pour l'apprentissage de la programmation avec Python

Olivier Goletti[1,3][0000−0002−1610−4985], Florian De Pierpont[2] et Kim Mens[1][0000−0003−0303−1630]

[1] ICTEAM/INGI, UCLouvain, Louvain-la-Neuve, Belgique
firstname.lastname@uclouvain.be
[2] EPL, UCLouvain, Louvain-la-Neuve, Belgique
florian.depierpont@student.uclouvain.be
[3] LIACS, Leiden University, Leiden, Pays-Bas

Résumé L'enseignement de la programmation peut profiter de l'utilisation de stratégies d'instruction explicites pour diminuer la charge cognitive et favoriser le transfert des apprentissages. Une de ces stratégies est l'utilisation d'exemples résolus avec objectifs étiquetés. Nous avons utilisé une méthodologie d'analyse de tâche pour extraire à des experts en programmation les connaissances nécessaires à la création de tels exemples résolus. Cet article présente la méthodologie utilisée et propose un document de formation réutilisable par d'autres membres de la communauté enseignante d'informatique.

Mots clés : enseignement de l'informatique · exemples résolus · objectifs étiquetés · stratégie d'instruction explicite

1 Introduction

Comme l'enseignement de l'informatique prend de l'ampleur, la recherche en didactique également. On enseigne maintenant l'informatique de plus en plus tôt et dans de plus en plus de pays [20]. Une méthodologie souvent utilisée est l'apprentissage par problèmes (en anglais : problem based learning - PBL) [12].

Notre analyse d'un cours d'introduction à la programmation utilisant PBL a mis en évidence l'intérêt d'utiliser des stratégies d'instruction explicite pour diminuer la charge cognitive des étudiants et promouvoir le transfert des apprentissages au sein du cours de programmation. Un besoin de techniques d'instructions explicites a été identifié [6,8] . En effet, les étudiants profitent de plus d'accompagnement [18] en particulier dans le cadre de méthodologies d'enseignement moins guidées [10].

Plusieurs stratégies d'instructions explicites ont déjà été testées récemment pour enseigner la programmation [5,11,15,22]. En particulier, l'utilisation d'exemples résolus avec objectifs étiquetés (en anglais : Subgoal Labeled Worked Examples - SLWE) a fait l'objet de plusieurs études par Margulieux et al. [13,15,16]. Les SLWEs utilisés en informatique tels qu'introduits par Margulieux et al. ont été

notamment testés en Java ou pour des langages de programmation visuels (par blocs) seulement [13,15].

Dans le cadre de sa thèse, le premier auteur a identifié des stratégies d'instruction explicite et a formé des tuteurs d'un cours d'informatique de première année de bachelier en ingénieur civil à les utiliser en séance de travaux pratiques, notamment les SLWEs [8,7] . Afin de poursuivre avec la stratégie des SLWEs et dans le but de rédiger un document (appelé document de formation) pour favoriser son utilisation par les tuteurs, nous présentons dans cet article le travail de création de SLWEs qui a été réalisé.

Pour créer les SLWEs, nous avons analysé des résolutions d'exercices faites par des experts. La méthodologie pour cette analyse est l'analyse de tâche (Task Analysis) [3], une méthode efficace pour extraire des procédures implicites que des experts d'un domaine utilisent de façon implicite pour résoudre des tâches complexes. En particulier, nous avons adapté la méthodologie d'analyse de tâche par résolution de problème (Task analysis by problem solving - TAPS) de Catrambone [2] utilisée et recommandée par Margulieux et al. [15].

Cet article présente donc le contexte théorique qui justifie l'utilisation de stratégies d'instruction explicite pour l'enseignement de l'informatique. Nous présentons notre adaptation d'une méthodologie de création de SLWEs ainsi que les SLWEs que nous avons créés.

2 Contexte théorique

Nous présentons d'abord dans cette section deux théories de l'éducation qui sous-tendent le choix de stratégies d'instructions explicites et en particulier d'exemples résolus avec objectifs étiquetés (SLWE) comme stratégie d'instruction. Nous donnons ensuite plus de détails sur cette stratégie.

2.1 Cadres conceptuels

Théorie de la charge cognitive La théorie de la charge cognitive (Cognitive Load Theory - CLT) [21] est une théorie de l'instruction basée sur l'architecture cognitive humaine. La CLT considère que l'humain n'a qu'une mémoire de travail limitée. L'impact sur l'instruction de la CLT est qu'il faut tenter de réduire la charge sur la mémoire de travail quand on enseigne. Trop de contexte ou d'informations superflues augmentent la charge cognitive de l'apprenant. Il faut minimiser cette charge cognitive et soutenir l'application de compétences génériques [17], par exemple en utilisant des exemples résolus [9].

Transfert des apprentissages Le transfert des apprentissage est la remobilisation de connaissances apprises dans un nouveau contexte [4,19]. Ce transfert est un processus actif qui peut être favorisé, notamment par des incitations ou des mises en évidence [4]. L'impact sur l'instruction du transfert des apprentissages est l'utilisation de stratégies qui invitent l'apprenant à se rappeler ses connaissances acquises et à les réappliquer dans d'autres contextes, par exemple

en mettant en évidence les étapes génériques d'une solution avec des objectifs identifiés et étiquetés.

2.2 Exemples résolus avec objectifs étiquetés (SLWEs)

Exemples résolus L'utilisation d'exemples résolus permet d'accélérer l'apprentissage d'une procédure de résolution en illustrant les différentes étapes abstraites d'une procédure de résolution d'un problème dans un exemple concret [15]. Cette méthode a par ailleurs déjà été utilisée dans le cadre de l'apprentissage de l'informatique. Cependant, pour donner un exemple, on prend toujours un contexte spécifique. Un novice amalgame parfois les détails du contexte avec le concept enseigné dans un tel exemple. Une façon d'éviter cet effet est de montrer plusieurs exemples différents et d'identifier ce qui est générique dans la procédure de résolution entre ces exemples.

Apprentissages par objectifs étiquetés L'apprentissage par objectifs identifiés et étiquetés aide l'apprenant à distinguer la procédure générique de résolution d'un problème [1]. Les apprenants sont ainsi guidés dans le transfert de leurs apprentissages lorsqu'ils sont amenés à résoudre des exercices similaires. Les objectifs étiquetés doivent cependant être identifiés avec soin.

Exemples résolus avec objectifs étiquetés (SLWEs) La combinaison de ces deux techniques pour enseigner l'informatique a été proposée par Margulieux et al. [15] en étiquetant les objectifs identifiés dans les exemples résolus directement. Ces objectifs étiquetés permettent d'identifier les étapes génériques de la résolution en les distinguant du contexte spécifique des exemples résolus. Ces étapes peuvent être réutilisées lors de la lecture ou l'écriture de code similaire. Cette stratégie favorise le transfert des apprentissages et diminue la charge cognitive pour les apprenants. Dans leur article, ils proposent une méthodologie pour créer des SLWEs et donnent également des exemples de SLWEs pour la lecture et l'écriture de concepts de base de programmation en Java.

L'utilisation de ces exemples résolus avec objectifs étiquetés (Subgoal Labeled Worked Examples - SLWE) a été montrée comme efficace pour l'enseignement de la programmation [13,15,16] ainsi que dans des supports en ligne de formation d'enseignants [14]. L'objectif du présent article n'est pas d'en discuter l'efficacité mais bien d'adpapter la stratégie et de créer de nouveaux SLWEs pour l'apprentissage du Python.

3 Méthodologie

Nous discutons ici la méthodologie utilisée pour extraire les connaissances des experts à partir de résolutions de problèmes. Ceci permettra de documenter les procédures de résolutions de problèmes, d'identifier les objectifs étiquetés et de rédiger un document de formation avec des exemples résolus. Ce document servira à former des tuteurs à utiliser les SLWEs dans un cours d'informatique.

3.1 Analyse de tâche par résolution de problème

L'objectif de l'analyse de tâche est de faire extraire à un expert d'un domaine ses connaissances implicites et sa façon de résoudre un problème [3]. Plusieurs techniques existent pour ce faire, basées notamment sur l'observation de résolution de problèmes par un expert, la prise de note, la rédaction d'une procédure, etc. En effet, les experts résolvent les problèmes différemment que les novices et leurs explications sont typiquement incomplètes et insuffisantes pour un novice. Même des enseignants qui s'y essaient n'y arrivent pas spécialement [1].

La méthodologie utilisée et recommandée par Margulieux et al. est la méthodologie d'analyse de tâche par résolution de problème (TAPS) de Catambone [2]. Dans TAPS, l'expert du sujet est appelé SME (Subject-Matter Expert), on l'appellera "l'expert" dans la suite de cet article. Quant à l'expert en extraction de connaissance (Knowedlge Extractor Expert - KEE), on l'appellera "l'analyste". L'expert doit identifier des tâches qui se rapportent aux concepts à traiter et les résoudre devant l'analyste. L'analyste est de préférence novice dans le sujet à analyser et doit prendre des notes sur la technique de résolution de l'expert ou des experts.

Les étapes recommandées par TAPS sont les suivantes :

— L'expert identifie les exercices qu'un apprenant devrait pourvoir résoudre s'il maîtrise la matière.
— L'expert résout un de ces exercices.
— L'analyste prend des notes détaillées sur la raison de chaque étape de résolution et demande à l'expert de justifier chacune de ces étapes, ce que l'expert a parfois du mal à verbaliser.
— L'analyste reprend ses notes, les réorganise, extrait les procédures et justifications.
— L'expert résoud un nouveau problème.
— L'analyste complète ses notes pour combler les lacunes et résoudre les incohérences.
— L'analyste tente de résoudre un problème similaire sur base de ses notes.
— L'analyste réinterroge l'expert jusqu'à pouvoir résoudre les exercices par lui-même.

Finalement, des échanges entre les experts et l'analyste aboutissent à la rédaction d'un document. Ce document décrit pour chaque concept abordé la procédure de résolution, les objectifs étiquetés permettant de structurer les éléments génériques et des exemples résolus annotés illustrant la procédure de résolution.

3.2 TAPS adapté

La méthodologie utilisée dans notre travail est très similaire à TAPS. Les experts étaient au nombre de quatre : le troisième auteur, professeur d'informatique depuis plus de vingt ans et co-titulaire du cours d'introduction à la programmation depuis quatre ans ; le premier auteur, assistant pour le cours d'introduction à la programmation depuis six ans et doctorant en didactique de l'informatique ; ainsi que deux doctorants en informatique, aussi assistants du

cours. L'analyste quant à lui est le second auteur et est étudiant de master en informatique, il n'était donc pas novice en programmation comme recommandé par TAPS, cependant il n'avait jamais suivi de cours en Python.

Les experts ont à chaque fois identifié les exercices à maîtriser pour les différents concepts abordés dans le cours (voir Table 1). L'analyste invitait alors un ou plusieurs experts si nécessaire en visioconférence pour enregistrer sa résolution des exercices sélectionnés. Les experts travaillaient dans un IDE simple avec leur écran partagé. L'analyste posait des questions de justification des différentes étapes aux experts. L'analyste prenait des notes et pouvait éventuellement consulter l'enregistrement après coup. Si nécessaire cependant il pouvait faire appel à un autre expert pour le même concept afin de combler les lacunes ou résoudre les incohérences de ses notes.

Comme l'analyste est un étudiant de master en informatique, nous avons décidé de ne pas faire les deux dernières étapes de TAPS qui attendent du KEE qu'il tente de résoudre les problèmes par lui-même sur base de ses notes. Nous sommes conscient que cette expertise en programmation peut introduire un biais dans les SLWEs. Cependant, nous estimons que la richesse des différents profils de l'équipe de recherche (chercheur en didactique de l'informatique et assistant, enseignant, étudiant) a enrichi le regard et l'attention portés aux choix des SLWEs et constitue un avantage pour ce travail.

Concept	**Lecture de code**	**Écriture de code**	**Adaptation Python de [15]**
Affectation	x	x	x
Condition	x	x	x
Boucle	x	x	x
Fonction	x	x	x
Parcours de chaîne ou de liste		x	
Lecture de fichier		x	
Écriture de fichier		x	
Création et mise à jour de dictionnaire		x	
Création de classe		x	
Ajout de noeud dans une liste chaînée		x	
Retrait de noeud dans une liste chaînée		x	

Table 1. Concepts pour lesquels des SLWEs ont été créés en Python

4 Résultats

Nous avons créé des SLWEs pour les concepts donnés dans la Table 1. L'ensemble du document de formation est disponible en ligne [4]. Afin d'illustrer la

4. https://dial.uclouvain.be/pr/boreal/object/boreal:260190

production réalisée, voici les objectifs étiquetés pour écrire un programme qui lit un fichier en Python et l'exemple résolu associé (voir Fig. 1) :

1. **Ouvrir** le fichier
   (a) Identifier le nom et le chemin du fichier (typiquement dans `filename`)
   (b) Choisir le mode `"r"`
   (c) Ouvrir le fichier avec :
      — Soit : `with open(filename , mode) as f :`
      — Soit : `f = open(filename , mode)`
2. **Traitement** du fichier en fonction du format
   (a) Parcours des lignes
      — ligne par ligne avec `f.readline()`
      — en itérant sur les lignes avec `for line in f:`
   (b) Traitement des lignes
      — Retrait des blancs en début et fin de ligne avec `line.strip()`
      — Séparer les éléments en fonction du format avec `line.split()`
      — Convertir les éléments en fonction du type attendu
   (c) Traiter les erreurs de formatage du fichier (ignorer ligne, `raise ValueError`)
3. **Fermeture** du fichier
   — Avec un `with`, il n'y a rien à faire
   — Sinon, avec un `f.close()`
4. Gérer les **exceptions** susceptibles de se produire durant le traitement du fichier (typiquement `IOError`)
   (a) Mettre le code dans un `try : ... except :`
   (b) Traiter les exceptions par le suite avec des `except error_type: ...`

Le document de formation[4] qui contient tous les SLWEs ainsi qu'une description de leur utilisation a déjà servi à la formation de tuteurs au premier semestre de l'année académique 2021-2022. Septs tuteurs ont ainsi utilisé ces SLWEs. La manière dont ils ont utilisé cette stratégie pendant leur séance d'encadrement fera l'objet d'une publication future.

5 Conclusion

Cet article a pour objectif de contribuer à la didactique de l'informatique en proposant des exemples résolus avec objectifs étiquetés (SLWEs) pour l'enseignement de la programmation en Python. L'utilisation de ces SLWEs est justifié dans le cadre des implications en terme d'instruction des théories de la charge cognitive et du transfert des apprentissages. La création de SLWEs était nécessaire car il n'en existait pas encore pour les concepts étudiés en Python. Générer les SLWEs pour les concepts sélectionnés a été possible en utilisant une méthodologie d'analyse de tâche documentée par ailleurs dans la littérature. L'analyse de tâche TAPS a été adaptée pour extraire les connaissances d'un expert en

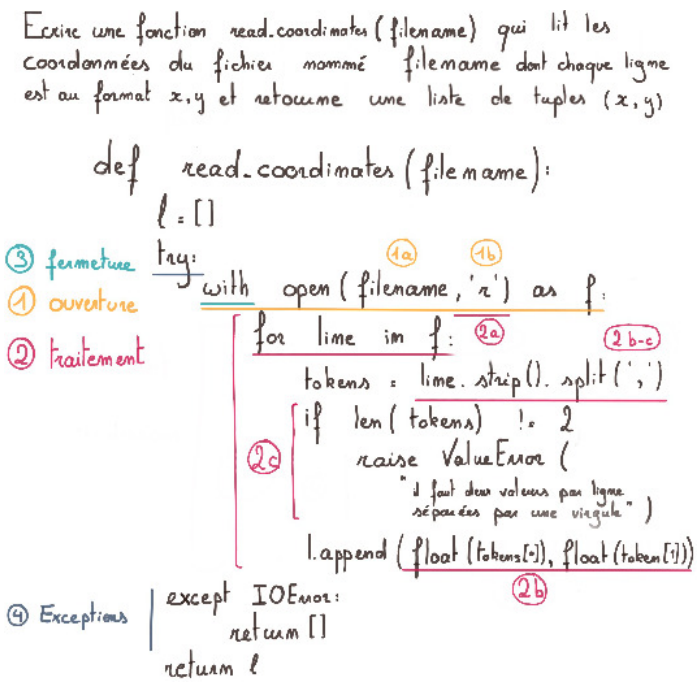

Figure 1. Exemple résolu de lecture de fichier en Python annoté d'objectifs étiquetés

programmation. Une liste de concepts vu dans le cadre d'un premier cours d'informatique de niveau universitaire a été sélectionnée, les enregistrements des résolutions d'exercices pour chaque concept par plusieurs experts ont été analysés par notre analyste, le KEE. Il en a tiré pour chaque concept une procédure de résolution de problèmes et a identifié des objectifs étiquetés pour chaque procédure. Ces procédures et objectifs étiquetés ont été discutés et validés par les auteurs. Un document de formation a ensuite été rédigé pour former les tuteurs du cours qui participaient à l'expérience. Dans ce document, les procédures pour chaque concept sont détaillées, les objectifs étiquetés sont fournis et un ou plusieurs exemples résolus annotés sont fournis. Ce document a été utilisé pour former sept tuteurs à l'utilisation des SLWEs dans leur encadrement d'un cours d'introduction à la programmation en Python. Ce document est publiquement disponible[4] et réutilisable par d'autres membres de la communauté enseignante d'informatique.

Références

1. Catrambone, R. : The Subgoal Learning Model : Creating Better Examples So That Students Can Solve Novel Problems p. 22
2. Catrambone, R. : Task analysis by problem solving (TAPS) : Uncovering expert knowledge to develop high-quality instructional materials and training. In : Learning and Technology Symposium, Columbus, GA (2011)
3. Clark, R., Feldon, D., Van Merrienboer, J.J.G., Yates, K., Early, S. : Cognitive task analysis. In : Handbook of Research on Educational Communications and Technology, chap. 43, pp. 577–593 (Jan 2008)
4. Council, N.R. : How People Learn : Brain, Mind, Experience, and School : Expanded Edition. National Academies Press (2000)
5. Ericson, B.J., Margulieux, L.E., Rick, J. : Solving Parsons Problems Versus Fixing and Writing Code. In : Proceedings of the 17th Koli Calling International Conference on Computing Education Research. pp. 20–29. Koli Calling '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3141880.3141895
6. Goletti, O. : En quoi le dispositif mis en œuvre dans le cours d'introduction à l'informatique en BAC1 ingénieur civil basé sur l'apprentissage par problèmes soutient les processus du transfert des apprentissages : l'encodage et l'accessibilité aux connaissances ? Tech. rep., UCLouvain (2019), `http://hdl.handle.net/2078.1/245579`
7. Goletti, O. : Promoting Learning Transfer in Computer Science Education by Training Teachers to use Explicit Programming Strategies. In : ICER '17. pp. 411–412. ACM, Virtual Event USA (Aug 2021). https://doi.org/10.1145/3446871.3469776
8. Goletti, O., Mens, K., Hermans, F. : Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course. In : ITiCSE '21. p. 7. ACM Press, Virtual Event, Germany (Jun 2021). https://doi.org/10.1145/3430665.3456348
9. Kirschner, P.A. : Cognitive load theory : Implications of cognitive load theory on the design of learning. Learning and Instruction **12**(1), 1–10 (Feb 2002). https://doi.org/10.1016/S0959-4752(01)00014-7
10. Kirschner, P.A., Sweller, J., Clark, R.E. : Why Minimal Guidance During Instruction Does Not Work : An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. Educational Psychologist **41**(2), 75–86 (Jun 2006). https://doi.org/10.1207/s15326985ep4102$_1$
11. Loksa, D., Ko, A.J., Jernigan, W., Oleson, A., Mendez, C.J., Burnett, M.M. : Programming, problem solving, and self-awareness : Effects of explicit guidance. In : Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. pp. 1449–1461. ACM (2016)
12. Luxton-Reilly, A., Simon, Albluwi, I., Becker, B.A., Giannakos, M., Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J., Szabo, C. : Introductory programming : A systematic literature review. In : ITiCSE '23. pp. 55–106. ITiCSE 2018 Companion, ACM, New York, NY, USA (Jul 2018). https://doi.org/10.1145/3293881.3295779
13. Margulieux, L., Catrambone, R., Guzdial, M. : Subgoal Labeled Worked Examples Improve K-12 Teacher Performance in Computer Programming Training. In : Proceedings of the Annual Meeting of the Cognitive Science Society. vol. 35 (2013)

14. Margulieux, L.E., Catrambone, R., Guzdial, M. : Employing subgoals in computer programming education. Computer Science Education **26**(1), 44–67 (Jan 2016). https://doi.org/10.1080/08993408.2016.1144429, `https://www.tandfonline.com/doi/citedby/10.1080/08993408.2016.1144429`, publisher : Routledge

15. Margulieux, L.E., Morrison, B.B., Decker, A. : Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. In : ITiCSE '19. pp. 548–554. ACM Press, Aberdeen, Scotland Uk (2019). https://doi.org/10.1145/3304221.3319756

16. Morrison, B.B., Margulieux, L.E., Guzdial, M. : Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In : ICER '11. pp. 21–29. ACM Press (2015). https://doi.org/10.1145/2787622.2787733

17. Sweller, J., Ayres, P., Kalyuga, S. : Cognitive Load Theory, Volume 1 of Explorations in the Learning Sciences, Instructional Systems and Performance Technologies. Springer, New York (2011)

18. Sweller, J., van Merriënboer, J.J., Paas, F. : Cognitive architecture and instructional design : 20 years later. Educational Psychology Review pp. 1–32 (2019)

19. Tardif, J. : Le Transfert Des Apprentissages. Editions logiques (1999)

20. Vahrenhold, J., Caspersen, M., Berry, G., Gal-Ezer, J., Kölling, M., McGettrick, A., Nardelli, E., Pereira, C., Westermeier, M. : Informatics Education in Europe : Are We All In The Same Boat ? (2017)

21. van Merriënboer, J.J.G., Sweller, J. : Cognitive Load Theory and Complex Learning : Recent Developments and Future Directions. Educational Psychology Review **17**(2), 147–177 (Jun 2005). https://doi.org/10.1007/s10648-005-3951-0

22. Xie, B., Nelson, G.L., Ko, A.J. : An explicit strategy to scaffold novice program tracing. In : SIGCSE TS '49. pp. 344–349. ACM (2018)