Description et analyse de trois EIAH d'apprentissage de Python

Sébastien Jolivet, Eva Dechaux, Anne-Claire Gobard, Patrick Wang

Actes de l'atelier "Apprendre la Pensée Informatique de la Maternelle à l'Université"
Brest, France, June 13, 2023

Description et analyse de trois EIAH d'apprentissage de Python

Sébastien Jolivet[1, 2] Eva Dechaux[1] Anne-Claire Gobard[3] Patrick Wang[4]
(1) IUFE, Pavillon Mail, 40 bd du Pont-d'Arve, CH-1205 Genève, Suisse
(2) TECFA, villa Battelle, 7 route de Drize, CH-1227 Carouge, Suisse
(3) Lycée Kastler, 26 avenue de la Palette, 95011 Cergy Pontoise, France
(4) Haute école pédagogique du canton de Vaud, Avenue de Cour 33, CH-1014 Lausanne, Suisse
sebastien.jolivet@unige.ch, eva.dechaux@unige.ch,
anne-clair.gobard@ac-versailles.fr, patrick.wang@hepl.ch

RÉSUMÉ

Dans cet article nous proposons une analyse de trois EIAH d'apprentissage de Python : AlgoPython, Citizen Code Python et Pyrates. Après avoir présenté le fonctionnement de chaque EIAH nous revenons sur les savoirs travaillés dans ces environnements et l'organisation de ce travail. Puis nous étudions les aides et rétroactions présentes ainsi que les possibilités d'orchestration proposées. Ce travail permet de mettre en évidence les points communs et différences entre environnements et soulève diverses questions sur les apports en ce qui concerne les apprentissages. Nous proposons en conclusion une synthèse de ces analyses et diverses exploitations possibles de ce travail.

ABSTRACT

Description and analysis of three Python learning environments

In this article we propose an analysis of three Python learning environments : AlgoPython, Citizen Code Python and Pyrates. First, we present the functioning of each EIAH, the knowledge worked in these environments and the organization of this work. Then we study the hints and feedbacks proposed, and the orchestration possibilities. This work allows us to highlight the similarities and differences between these environments and raises various questions about their contributions in terms of learning. In conclusion, we synthesize these analyses and several possible uses of this work.

MOTS-CLÉS : Environnements d'apprentissage de Python ; analyse d'EIAH ; EIAH.

KEYWORDS: Python learning environments ; learning environment analysis ; TEL.

1 Introduction

Dans cet article nous analysons trois EIAH d'apprentissage de Python. Nous présentons les caractéristiques principales de ces environnements, puis nous détaillons l'étude des savoirs travaillés. Cette étude a été débutée dans (Jolivet et al., 2023). Elle est basée sur un référentiel de types de tâches relatif au travail global de la programmation et au travail spécifique sur les variables, boucles et instructions conditionnelles. Puis nous enrichissons cette première analyse par l'étude des choix didactiques réalisés dans ces environnements, des aides et rétroactions proposées aux apprenants, et des outils d'orchestration mis à disposition des enseignants. La conclusion nous permet de mettre en perspective ces différents éléments pour esquisser des propositions relatives à la conception d'EIAH.

2 Présentation des trois EIAH

Les EIAH étudiés, AlgoPython [1], Citizen Code Python [2] et Pyrates [3], visent aux premiers apprentissages de Python (ils sont nommés AP, CCP et Py dans la suite du texte). Ils proposent tous une version gratuite complète. AP propose un système de licence, à destination des établissements scolaires, offrant des possibilités d'administration de la plateforme et de suivi du travail des élèves. Ils sont tous accessibles en ligne et aucun ne propose d'installation locale. Dans les sections suivantes, nous présentons le fonctionnement de chacun de ces environnements.

2.1 AlgoPython

AP est développé par un professeur de mathématiques et distribué par Génération 5. Dans sa présentation, il est signalé qu'il couvre l'intégralité du programme de lycée en matière d'apprentissage de Python, pour la part intégrée aux mathématiques. Les contenus sont organisés autour de deux grands domaines : l'apprentissage de Python et l'utilisation de Python dans le cadre d'exercices de mathématiques.Celui relatif à l'apprentissage de Python est structuré en huit séries : les instructions, les boucles `for`, les variables, les fonctions, les conditions, les boucles `while`, les listes. Chaque série contient un ensemble de 7 à 12 exercices, complétés par 3 à 12 exercices "bonus". Pour chaque exercice il y a trois zones affichées en permanence : 1) la première contient un énoncé textuel qui indique la tâche à réaliser et contient parfois certains éléments de "cours" ; 2) zone de saisie du code Python ; 3) la console Python qui présente le résultat de l'exécution du code et d'éventuels messages d'erreur. L'accès aux exercices "obligatoires" d'une section est soumis à la réussite du précédent. Enfin, AP propose une console permettant de l'utiliser comme un IDE.

2.2 Pyrates

Cet EIAH présente la particularité d'avoir été développé comme outil d'étude dans le cadre d'un travail de thèse [4] (Branthôme, 2021), il est donc normal qu'il propose ni le même volume de ressources ni la même couverture du domaine que les deux précédents. Cependant, son utilisation auprès d'un nombre important d'élèves et le travail didactique qui a fondé sa conception, motivent son intégration dans les EIAH analysés. Il se positionne dans le moment particulier de la transition entre le collège et le lycée français avec le passage de Scratch (ou autre environnement de programmation par blocs) à Python. L'environnement est organisé sous forme d'un jeu à huit niveaux. À chaque niveau le pirate se déplace dans un environnement, avec lequel il peut interagir, pour atteindre et ouvrir un coffre.

Dans Pyrates, quatre zones sont affichées en permanence : 1) une zone textuelle qui contient *l'objectif* qui définit la tâche à réaliser, des *contraintes* à respecter dans le niveau, et la définition des nouvelles *fonctions de contrôle* utiles pour le niveau ; 2) un tableau qui définit la situation initiale du niveau (le pirate, le coffre et les différents éléments du type clé, les obstacles). L'exécution du programme

1. `https://www.algopython.fr`
2. `https://www.futureengineer.fr/`
3. `https://py-rates.fr`
4. Nous analysons dans cette contribution la version disponible en ligne. D'autres fonctionnalités ont été développées par le concepteur de l'environnement à des fins de recherche et seront peut-être disponibles ultérieurement, mais ce n'est pas le cas à la date de cette analyse.

permet de mettre en action les éléments de cette zone ; 3) une zone de saisie du code Python ; 4) une zone nommée *Mémo Programmation Python* qui est un portail d'accès vers différentes aides.

2.3 Citizen Code Python

Cet EIAH a été développé par la société Tralalère et IOI France avec le soutien d'Amazon Future Engineer. Il propose une « initiation à la programmation pour tous ». Cet environnement présente la spécificité de proposer, pour tous les exercices, de les réaliser en Python ou en Blockly. Il est structuré en 11 *saisons*, chaque saison étant constituée d'un ensemble de 3 à 10 exercices. Tous les exercices sont basés sur la même mécanique : un "robot pince" saisit des blocs situés en dessous et peut les déplacer vers la droite ou la gauche pour reconstituer un modèle ; parfois il est possible de déposer les blocs dans un broyeur qui détruit des éléments.

Trois zones sont affichées en permanence : 1) une zone qui contient l'énoncé textuel qui permet de définir l'objectif à atteindre en spécifiant la position attendue des briques ; 2) une zone avec le "robot pince" et les briques à déplacer, elle participe aussi à la définition de la tâche à réaliser puisque les objets à déplacer y sont dans la configuration initiale ; 3) une zone de saisie du code Python.

Suite à cette présentation globale, nous abordons les savoirs travaillés dans les trois environnements et les choix didactiques pour organiser ce travail.

3 Savoirs travaillés dans ces EIAH et modalités de travail

Du point de vue des thématiques travaillées, AP est le plus complet avec l'ensemble de thèmes suivant : les instructions, les boucles `for`, les variables, les fonctions, les conditions, les boucles `while`, les listes. CPP n'aborde pas les listes tandis que Py n'aborde ni les listes ni les fonctions.

Au-delà de l'identification des thèmes abordés, nous nous intéressons maintenant à la manière dont ils sont travaillés. Pour cela nous reprenons le travail présenté dans (Jolivet et al., 2023) qui identifie un ensemble de types de tâches relatifs à la programmation [5] d'une manière globale et aux différents thèmes communs aux trois EIAH, à savoir les boucles, les variables et les instructions conditionnelles.

3.1 Les types de tâches présents dans les EIAH

Nous ne reprenons pas ici en détail le processus d'identification et de structuration des types de tâches. Nous précisons simplement qu'ils ont été identifiés dans divers moyens d'enseignement de l'informatique du niveau collège et lycée (français et suisse) et qu'il s'agit donc de types de tâches "réels" et non pas "inventés". Nous reprenons simplement, dans les tableaux 1 et 2, la répartition de ces types de tâches dans les trois EIAH. Ces tableaux ont été obtenus en procédant à la résolution systématique de tous les exercices concernés par les trois notions étudiées, puis à l'analyse des solutions. Nous nous sommes limités aux types de tâches prescrits et n'avons pas intégré ceux qui peuvent ou doivent être mobilisés pour la réalisation de la tâche. Par exemple, pour produire une

5. Non pas la programmation dans toute sa généralité, mais dans les limites de ce qui est travaillé lors des premiers apprentissages en milieu scolaire.

TABLE 1 – Effectif des types de tâches de *Programmation* par EIAH

Types de tâches		AP	CCP	Py
	Analyser un code existant	0	0	0
Feuille blanche	Copier et exécuter	0	6	0
	Concevoir un programme	26	43	8
Code existant : Modifier un programme P en P' qui réalise une tâche t	P est erroné	1	0	0
	P réalise t' proche de t	15	0	0
	P contient des trous	3	0	0
Nombre d'exercices décrits		45	49	8

boucle `while` il va notamment être nécessaire *d'affecter une valeur à une variable, initialisation*, cependant nous ne comptabilisons pas ce deuxième type de tâches si la tâche porte sur le premier.

Nous explicitons certains de ces types de tâches en même temps que nous commentons ces tableaux. Les détails sont disponibles dans (Jolivet et al., 2023).

Ce qui est tout d'abord notable dans ce tableau 1, c'est la très faible variété des types de tâches proposés. Ainsi CCP et Py proposent exclusivement [6] le type de tâches *Concevoir un programme* dans une situation de type *feuille blanche*. C'est-à-dire que l'élève ne dispose que de l'énoncé et d'une zone de saisie vide pour réaliser la tâche demandée. AP propose un peu plus de variété puisqu'un peu plus de la moitié des exercices proposés sont aussi de ce type mais qu'un tiers sont du type *Modifier un programme qui réalise une tâche proche ou une partie de la tâche souhaitée*, exercices pour lesquels il est proposé dans la zone énoncée un code presque identique à celui à produire. Enfin, dans AP il y a aussi, mais de manière anecdotique, un exercice où il faut corriger un code donné erroné et trois autres où il faut compléter un code donné qui contient des trous.

Il n'y a par contre, tous environnements confondus, aucun exercice visant l'analyse d'un code existant.

TABLE 2 – Effectif des types de tâches relatifs aux *boucles, variables et instructions conditionnelles*

Type de tâches	AP	CCP	Py
Produire une boucle bornée	21	105	10
Utiliser la valeur d'une expression	5	0	0
Affecter une valeur à une variable, initialisation	9	24	3
Affecter une valeur à une variable, modification	0	13	0
Utiliser une variable dans une expression	6	0	3
Ajouter une ou des instructions à une boucle existante	2	0	0
Concevoir une IC	12	16	2
Déterminer les instructions à exécuter à chaque itération	1	2	0
Modifier les instructions qui font évoluer la condition d'arrêt	1	0	0
Produire une boucle non bornée	6	12	2

Le tableau 2 nous permet de dresser divers constats. Tout d'abord, ce sont essentiellement les types de tâches "principaux" qui sont mobilisés : *Produire une boucle bornée*, *Concevoir une IC*, *Produire une boucle non bornée*. Les éléments qui sont nécessaires à la réalisation de tels types de tâches, par exemple *Déterminer les instructions à exécuter à chaque itération* ou *Utiliser la valeur d'une*

6. CCP propose pour chaque premier exercice d'une série un type de tâches un peu particulier qui est de recopier le code solution du problème qui est donné *in extenso.*

expression, sont évidemment travaillés lors de ces tâches mais ne font que très peu l'objet d'exercices spécifiques. On peut donc estimer que ces environnements permettent de s'entraîner à manipuler les notions fondamentales mais pas réellement d'apprendre le travail avec ces notions puisque les bases permettant l'apprentissage ne font pas l'objet d'exercices spécifiques.

On peut aussi noter la sur-représentation du type de tâches *Produire une boucle bornée* dans CCP. Ceci est la conséquence du choix des auteurs de l'environnement de ne pas autoriser de paramètre pour les fonctions *gauche()* et *droite()* qui permettent de déplacer la pince. Chaque déplacement de plus de deux mouvements doit donc être réalisé à l'aide d'une boucle *for* pour respecter le nombre limite de lignes de code autorisé.

À propos des fonctions il est notable que, avant même d'en faire un objet de travail, elles sont largement utilisées. En particulier dans CCP et Py qui sont basés sur des déplacements, et donc utilisant massivement des fonctions pour gérer ces déplacements. Ainsi dans CCP (*resp.* Py) nous pouvons comptabiliser, sur les exercices étudiés, 399 (*resp.* 27) appels de fonctions sans paramètre et 27 (*resp.* 6) appels de fonctions en passant une valeur en paramètre.

Si cette utilisation des fonctions est rendue nécessaire par le contexte de l'environnement, il est notable que cette utilisation précoce des fonctions n'est nullement exploitée lorsque les fonctions deviennent l'objet du travail dans CCP.

3.2 Choix et artifices didactiques

Dans les trois environnements une fois une série traitée, il est possible que la notion abordée soit remobilisée dans les séries suivantes. Il y a donc *de facto* un choix d'organisation des apprentissages porté par l'environnement. Concernant les choix réalisés dans Pyrates, ils sont largement documentés par son concepteur et nous renvoyons donc à (Branthôme, 2021) pour la présentation de ceux-ci.

Dans le cas de AP et CCP on peut noter comme similitude que les boucles `while` sont toutes les deux traitées à la suite des autres notions, seules les listes pour AP et les booléens pour CCP sont traités après. Dans AP nous pouvons noter la chronologie "boucle `for` ; variables ; fonctions ; conditions" alors que dans CCP la chronologie proposée est "boucle `for` ; instruction conditionnelle ; variables ; fonctions". Dans aucun de ces deux environnements les auteurs ne motivent ces choix, et il n'y a pas à notre connaissance, de travaux de didactique de l'informatique qui permettraient de valider ou invalider l'un plutôt que l'autre.

Au sein des exercices, l'enjeu principal des artifices didactiques utilisés, est de contraindre l'utilisation d'une notion plutôt qu'une autre (par exemple le `for` plutôt que la répétition d'une même séquence d'instructions ou une boucle `while` plutôt qu'une boucle `for`). AP et CCP utilisent à la fois la limitation du nombre de lignes de code et parfois le blocage de l'utilisation d'un mot (par exemple le `for`). Si Py utilise aussi la limitation du nombre autorisé de lignes de code on peut noter l'utilisation intéressante d'un artifice supplémentaire. Dans le cadre de l'utilisation du `while`, une variable aléatoire modifiée à chaque exécution du code est introduite dans la définition de la mission à remplir par le pirate. En l'occurrence la solidité d'un tonneau sur lequel il faut taper pour le détruire afin d'avancer, il devient ainsi nécessaire de *taper tant que le tonneau est présent*.

3.3 Conclusion sur les savoirs travaillés

Nous pouvons constater après analyse des exercices de ces trois EIAH, que ces derniers demandent à l'apprenant, principalement, de concevoir un programme, majoritairement à partir d'une feuille blanche. Il n'y a par ailleurs aucun exercice qui relève de l'analyse d'un code existant. Ces choix peuvent apparaître comme surprenant, car cette dernière activité n'est pas particulièrement difficile à implémenter par exemple sous forme d'exercices d'appariements entre code et problème. Par ailleurs, l'analyse d'un code existant est nécessaire, que ce soit pour adapter un code existant à un nouvel objectif ou pour corriger un code déjà produit.

Par ailleurs, les notions sont le plus souvent travaillées de manière globale sans que les différents types de tâches nécessaires pour leur mise en œuvre ne soient travaillés de manière spécifique. Par exemple, dans le travail sur les boucles, il n'y a aucun exercice qui permet de "simplement" déterminer le nombre d'itérations ou l'expression booléenne à définir.

D'autre part, hormis dans AP, il n'y a pas d'exercices à trous qui permettent pourtant d'amener l'apprenant à travailler spécifiquement une notion, ou une partie de notion, sans être bloqué par le fait de gérer tout ce qui va autour (déclaration des variables, etc.). Il est notable que dans AP, pour les séries sur les fonctions et les listes, il y a une part égale d'exercices de type "feuille blanche" et "d'exercices à compléter avec le début du code donné". Toujours dans AP, l'auteur change très significativement de stratégie dans les séries d'exercices relatifs aux mathématiques, puisque, s'il n'y a toujours pas d'analyse de code, ce sont exclusivement des exercices pour lesquels le début du code est donné (souvent le nommage et le choix des paramètres de la fonction à produire) ou tout le code est donné avec des trous à compléter.

CCP et Py, qui sont des environnements de jeux graphiques, mobilisent de manière significative le type de tâche "appeler une fonction". AP semble mobiliser des tâches plus variées, ce qui peut être expliqué par la présence à la fois d'exercices graphiques avec un robot et d'exercices faisant écrire des mots dans la console.

4 Aides, rétroactions et outils d'orchestration

En complément de l'analyse des savoirs travaillés réalisée dans la section précédente, nous présentons maintenant les apports respectifs des différents environnements relativement aux *rétroactions et aides* et aux possibilités *d'orchestration et de suivi du travail des élèves* disponibles pour un enseignant dans le cadre d'un usage scolaire de l'environnement.

4.1 Aides et rétroactions

Dans cette section, nous nous intéressons aux moyens mis à disposition de l'apprenant pour l'aider dans la résolution de sa tâche. Ceux-ci sont de trois natures : moyens techniques proposés par l'environnement du type exécution pas à pas ; ressources de type "cours" et enfin rétroactions.

Lors de l'exécution du code, tous les EIAH proposent une modalité qui permet d'agir sur la vitesse d'exécution du code. CCP propose trois modes d'exécution : un mode *lecture* durant lequel l'instruction exécutée est en surbrillance dans le code ; un mode *pas à pas* qui permet de visualiser l'instruction en cours d'exécution et son effet sur la scène et un mode qui permet d'afficher directement le résultat

suite à l'exécution du code. Py dispose d'un mode identique au mode *lecture* avec une vitesse d'exécution réglable. AP ne propose qu'un curseur permettant de régler la vitesse d'exécution du code pour quelques exercices (dessin et déplacement d'un robot).

Concernant les aides à disposition de l'apprenant, dans AP elles sont situées dans les énoncés des exercices qui servent aussi à définir les notions et à présenter des exemples associés. Tout est donc à disposition de l'apprenant en permanence, mais rien n'est proposé pour attirer son attention sur tel ou tel élément particulier. Dans Py la zone *Mémo programmation Python* permet d'accéder à différents éléments de cours qui contiennent à la fois des définitions et des exemples. Il y a la particularité, conformément au positionnement de Py, de mettre en parallèle des morceaux de code en Scratch et le code équivalent en Python. Enfin, dans CCP, deux boutons donnent accès l'un à une ressource appelée *Documentation* qui contient des informations générales sur ce qu'est programmer et des informations spécifiques en lien avec le thème travaillé. L'autre bouton donne accès à des indices qui sont liés à la tâche à réaliser : deux à trois indices, progressifs, sont accessibles à la demande de l'apprenant.

Concernant les rétroactions on notera qu'elles sont essentiellement relatives aux aspects syntaxiques. Dans CCP certaines erreurs (oubli de parenthèses lors de l'appel d'une fonction par exemple) sont signalées dès l'écriture du code alors qu'il faut attendre l'exécution dans AP ou Py. Dans Py ou CCP il y aussi des rétroactions contextuelles à l'environnement (déplacement ou action impossible). D'une manière générale on peut noter l'absence de rétroactions épistémiques et le renvoi aux éléments d'aide présentés précédemment en fonction de l'activité de l'apprenant.

4.2 Dispositifs d'orchestration et de suivi pour l'enseignant

CCP est visiblement conçu pour un usage personnel avec la délivrance d'*open badges* en fonction du travail réalisé. Il n'y a aucune fonctionnalité pensée pour l'utilisation dans le cadre scolaire. Le suivi de la progression d'un élève, de ses erreurs, etc., serait donc totalement à la charge de l'enseignant sans dispositif dédié. Un dispositif d'inscription et d'identification permet de conserver, individuellement, la trace du travail réalisé.

Py ne propose pas non plus de dispositif d'inscription de classe ou de suivi global du travail des élèves. Il n'est pas non plus possible de s'inscrire, mais un code est attribué à chaque partie commencée, ce qui permet de reprendre le travail à partir du dernier niveau validé.

Dans AP, sous réserve de souscrire une licence, un enseignant peut créer des classes et ajouter des élèves en leur créant des comptes. Les possibilités d'orchestration sont limitées au déblocage global d'un ou plusieurs thèmes qui peut être décidé au niveau de la classe entière ou élève par élève. Il n'y a pas de gestion au niveau des exercices. Enfin, une interface de suivi de l'activité des élèves est disponible. Même si son ergonomie peut être améliorée, elle permet d'avoir une vision globale de la classe avec le nombre d'exercices terminés par thème pour chaque élève. Un accès au suivi individuel de chaque élève permet d'obtenir quelques précisions sur l'activité de celui-ci.

5 Choix, questions, manques et perspectives

En conclusion nous proposons deux grandes orientations pour prolonger ce travail d'analyse.

Sur l'apprentissage de la programmation

On constate tout d'abord que ces environnements proposent des choix spécifiques, notamment sur la chronologie d'étude des notions. Ils ne semblent pas relever de contraintes internes aux environnements. On peut donc supposer qu'ils sont l'expression des conceptions des auteurs de ces environnements sur l'apprentissage de la programmation. L'impact de ces choix didactiques est un questionnement fondamental à aborder. Pour cela, on pourra s'appuyer sur ces EIAH, par exemple en comparant les apprentissages entre des utilisateurs de CCP et de AP comme un moyen d'étudier l'impact de l'ordre de rencontre avec les différentes notions sur les apprenants.

Un deuxième questionnement, relatif à l'apprentissage de la programmation, porte sur la ou les exploitations possibles de ces environnements en situation de classe. Au delà de la présence de fonctionnalités d'orchestration et de suivi des élèves, ces environnements semblent plutôt destinés à des apprenants qui disposent déjà de certaines bases, a minima d'algorithmique, et une familiarité minimale avec ce que signifie programmer. Ceci soulève la question des moments de l'apprentissage (Chevallard, 1999) pour lesquels ils peuvent s'avérer efficients, rejoignant ici le questionnement posé dans (Jolivet and Grugeon-Allys, 2022). Ils sont sans doute plus adaptés à un travail d'entraînement qu'à une exploitation pour une première rencontre avec les concepts ou l'institutionnalisation par exemple. Ceci est encore renforcé par l'absence de rétroactions épistémiques.

Sur la conception d'EIAH d'apprentissage de la programmation

Dans (Feitelson, 2023) il est signalé la part très significative de l'analyse de codes dans l'activité des développeurs, ceci amène au moins à s'interroger sur la place que devrait avoir une telle activité dans l'apprentissage de la programmation. Or on a constaté le faible nombre de types de tâches différents proposés, et en particulier l'absence d'exercices de lecture / analyse de code. Une diversification des types de tâches proposés, par exemple des activités de lecture de code ou des exercices de type *Parsons Problems* (Ericson et al., 2018), est sans doute une orientation pour développer des EIAH d'apprentissage de la programmation.

Références

Branthôme, M. (2021). Apprentissage de la programmation informatique à la transition collège-lycée. *Revue STICEF*, 28(3). Medium : pdf Publisher : STICEF Version Number : 1.

Chevallard, Y. (1999). L'analyse des pratiques enseignantes en théorie anthropologique du didactique. *Recherches en Didactique des Mathématiques*, 19(2) :221–265.

Ericson, B. J., Foley, J. D., and Rick, J. (2018). Evaluating the efficiency and effectiveness of adaptive parsons problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 60–68.

Feitelson, D. G. (2023). From Code Complexity Metrics to Program Comprehension. *Communications of the ACM*, 66(5) :52–61.

Jolivet, S., Dechaux, E., Gobard, A.-C., and Wang, P. (2023). Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. In *Actes du colloque EIAH2023 : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*, Brest.

Jolivet, S. and Grugeon-Allys, B. (2022). Modélisation de parcours d'apprentissage adaptés à l'apprenant dans un EIAH. In Florensa, I. and Ruiz Munzón, N., editors, *Pré-actes de la 7e conférence internationale sur la théorie anthropologique du didactique (CITAD7)*, pages 92–106, Barcelone.