Nombre de participant·e·s maximal pour l'atelier	20	Nombre de sessions de 45mn que vous pouvez réaliser	3
Matériel nécessaire (vidéo-projection, grande salle, autre …)	Vidéo-projecteur, Accès à Internet, Prise électrique Ordinateur nécessaire pour les participants		

Pas à Pas dans le Code : Générer, Visualiser, Enseigner

Julien MATEESCO, CEC Emilie-Gourd, Genève, Suisse

Patrick WANG, Haute école pédagogique du canton de Vaud, Lausanne, Suisse

RÉSUMÉ *Cet atelier présente un environnement d'apprentissage de la programmation Python pour débutants (collège-lycée) combinant un générateur de code paramétrable, un éditeur Python et une traduction automatique en logigramme. L'outil est conçu pour soutenir une démarche de type Prédire-Exécuter-Investiguer-Modifier dans le navigateur. La proposition vise à illustrer comment la double représentation code-logigramme, associée à un module de défi interactif, peut soutenir la compréhension du flux d'exécution* ***et*** *l'analyse des erreurs des élèves.*

MOTS-CLÉS *• Introduction à la programmation • Python • Logigrammes.*

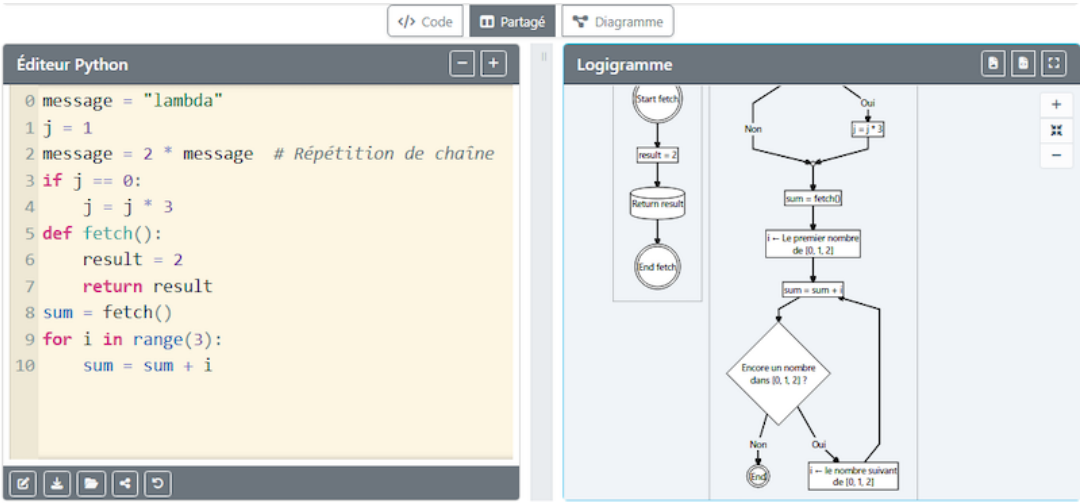

Figure 1 À gauche : un éditeur de code Python contenant un programme. À droite : le logigramme correspondant.

1. Déroulement de l'atelier

Durée	Contenu
5 min	Accueil : présentation de la démarche et de l'outil
15 min	Moment 1 : analyse croisée code-logigramme
10 min	Discussion 1 : apports pédagogiques de la double représentation
5 min	Moment 2 : une base de logs pour identifier des patterns d'erreur
10 min	Discussion 2 : perspectives pour la classe et pour la recherche

Moment 1 : Simulons la réception par l'élève d'un script syntaxiquement correct qui ne produit pas le résultat attendu. L'élève doit modifier le code pour atteindre le résultat attendu, noter sur papier le logigramme avant et après la correction apportée au processus.

Moment 2 : l'accent est mis sur les erreurs et sur la traçabilité des interactions. Chaque participant génère un exercice selon des paramètres, y répond avec ou sans logigramme. Le système de vérification automatique clôt le défi dans l'interface, mais dans le serveur, l'outil journalise anonymement les actions clés : codes chargés, valeurs prédites par les élèves, vérifications sollicitées, corrections apportées. Ces traces ouvrent des perspectives pour la classe et pour la recherche didactique : cartographier les difficultés récurrentes, les stratégies élèves, ou mesurer l'effet d'une remédiation.

Ces moments sont entrecoupés par des temps de discussion avec les participants pour recueillir leurs retours quant à l'utilité de la double représentation code-logigramme et de la récolte de traces d'interactions des élèves.

2. Description de l'outil présenté dans l'atelier

L'outil combine un générateur de code Python paramétrable, un éditeur Python et une traduction automatique du code en logigramme, suivis d'un module de "défi" interactif qui demande à l'élève de prédire la valeur stockée dans les variables présentes dans le programme en fin d'exécution de ce-dernier. L'outil est ainsi conçu pour soutenir une démarche de type Prédire–Exécuter–Investiguer–Modifier [1] dans le navigateur.

La génération de code Python est automatique, infinie, intégrant un aléa contrôlé et les éléments syntaxiques sont explicitement choisis par l'utilisateur (voir Figure 2), élève (pour renforcer l'apprentissage) ou enseignant (pour permettre l'alignement curriculaire).

La traduction sous forme de logigramme est ensuite proposée à côté du code textuel comme illustré en Figure 1. Cette juxtaposition vise à exploiter le principe de contiguïté spatiale issu de la théorie de l'apprentissage multimédia [2], facilitant ainsi la création de liens mentaux entre la syntaxe et la structure logique.

Le code Python n'est pas seulement affiché avec coloration syntaxique. La page propose un véritable éditeur, pour expérimenter avec le code, l'enregistrer ou l'exporter.

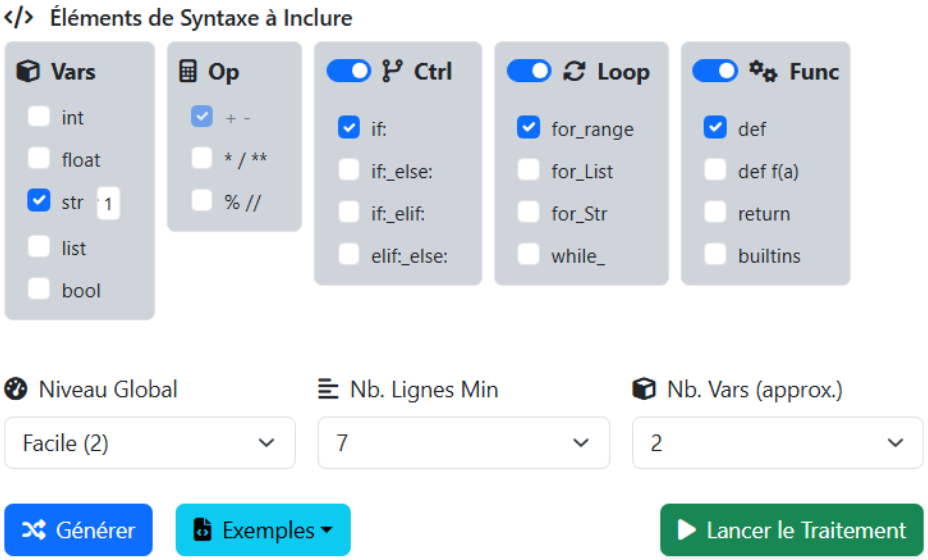

Figure 2 Panneau de configuration des concepts à inclure dans le programme à générer.

L'engagement de l'élève est assuré avec la section “Défi” qui vient tester la capacité de l'élève à tracer le code qui a été le support du logigramme. En effet, les méta-analyses sur la visualisation d'algorithmes soutiennent l’intuition enseignante que l'apprentissage nécessite un engagement actif, tel que la prédiction des résultats [3]. La page propose actuellement une rétroaction par vérification de la réponse élève (juste ou faux) et la révélation de la solution. Chaque variable dans le code est support à une question, comme illustré en Figure 3. Cet exercice de traçage vise à aider l'élève à construire une machine notionnelle [4] valide, c'est-à-dire un modèle mental correct du comportement de l'ordinateur.

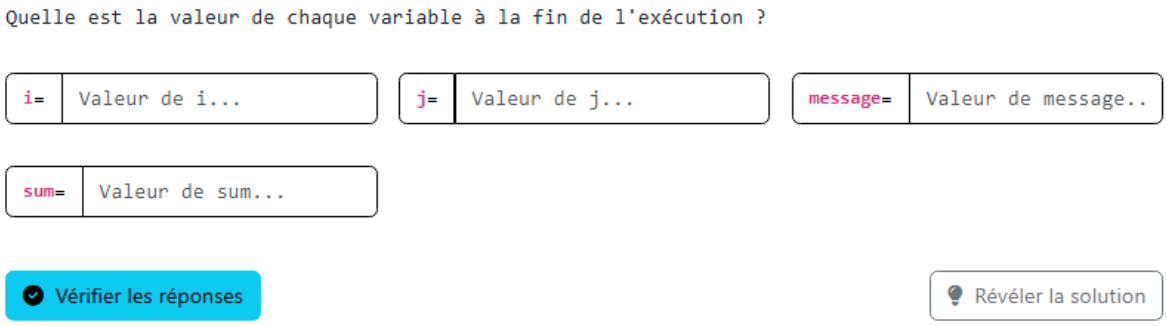

Figure 3 La section "Défi" demande de prédire la valeur stockée par les variables en fin d'exécution du programme.

En plus de "Vérifier" ou "Révéler", l'élève peut interagir avec le code grâce à une console dépliable en bas de page, et tester par exemple les effets de différents `print()`.

La journalisation sur serveur des interactions élèves (codes générés ou chargés, propositions de réponses, etc.) étend l’exerciseur automatique pour en faire un outil de gestion de classe à court terme et ouvre des perspectives de recherche à plus long terme.

3. Expérimentations envisagées

Cette plateforme a pour visée d’étudier le sujet de la lecture de code par des débutants en programmation en nous focalisant davantage sur l’effet de la présentation (simultanée ou non) d’un programme et du logigramme correspondant. Une expérimentation est dès lors envisagée avec des élèves du secondaire avec comme objectif de caractériser l’apport des logigrammes dans les tâches de lecture et d’écriture de code, notamment sous l'angle de la réduction de la charge cognitive [5] liée au décodage syntaxique. Une étude quasi-expérimentale impliquant plusieurs classes d’élève du secondaire, et avec comme variable indépendante la représentation d’un programme sous la forme de texte, de logigramme, ou de texte et logigramme est actuellement en cours de conception afin de justement répondre à la question de l’utilité de l’une ou l’autre de ces représentations dans l’acquisition de compétences de lecture et d’écriture de code.

4. Liens vers les ressources et point de contact

- Point de contact : Julien Mateesco, edu-mateescoj@eduge.ch
- Lien vers le répertoire GitHub : https://github.com/edu-mateescoj/gyminf
- Lien vers une version statique du site : https://edu-mateescoj.github.io/

5. Références

[1] S. Sentance and J. Waite, "PRIMM: Exploring pedagogical approaches for teaching text-based programming in school," in *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, 2017.

[2] R. E. Mayer, Multimedia Learning, Cambridge University Press, 2001.

[3] C. D. Hundhausen, S. A. Douglas and J. T. Stasko, "A Meta-Study of Algorithm Visualization Effectiveness," *Journal of Visual Languages & Computing,* vol. 13, no. 3, pp. 259-290, 2002.

[4] J. Sorva, "Notional machines and introductory programming education," *ACM Transactions on Computing Education,* vol. 13, no. 2, 2013.

[5] J. Sweller, P. Ayres and S. Kalyuga, Cognitive Load Theory, Springer, 2011.