UCLouvain

epl

École polytechnique de Louvain

Visualisation des données en Python

Auteurs: **Jean de Briey, Etienne Rubbers**
Promoteur: **Charles Pecheur**
Lecteurs: **Kim Mens, Olivier Goletti**
Année académique 2023–2024
Master [120] en sciences informatiques et Master [120] en ingénieur civil en informatique

Résumé

Thonny est l'un des environnement de développement intégré en python spécialisé pour les programmeurs débutants. Cependant, il ne dispose pas d'outil de visualisation de programme. C'est pour combler ce manque que nous avons décidé d'apporter à Thonny un outil de visualisation adapté à son design épuré et à sa volonté première de cibler les programmeurs débutants.
Cet outil, que nous avons conçu dans le cadre de notre mémoire, permet d'observer les liens entre les variables. Il est une extension de Thonny et offre trois vues sur les objets et structures d'un programme en Python. La première partie du mémoire montre les différentes variables, globales et locales, du programme de façon simple et intuitive. La seconde propose une vue hiérarchique de toutes ces variables et de leurs attributs. La troisième permet de visualiser graphiquement, à l'aide de boîtes et de flèches, les liens entre toutes ces variables et leurs attributs.
Bien que plusieurs outils de visualisations proposent déjà ce genre d'aide, notre extension offre l'avantage principal de proposer ces trois vues dans un même environnement de développement intégré (IDE) spécialisé pour les débutants tel que Thonny. A présent un apprenti programmeur peut coder en Python, déboguer son code et le visualiser dans un même endroit, et ce sans autre contraintes que les limitations de l'IDE.
Dans ce mémoire, nous présenterons les différentes étapes clés qui ont mené à la réalisation de notre extension. Nous évaluerons ensuite les différents choix d'implémentation qui ont été réalisés afin d'adapter au mieux notre outil à un programmeur débutant. Nous terminerons par un exemple d'installation et une démonstration du fonctionnement des différentes vues.
Le lien vers le répertoire Github de notre outil est en annexe.

Remerciements

Tout d'abord, nous tenons à exprimer notre gratitude envers notre promoteur, le Docteur Professeur Charles Pecheur. Nous le remercions sincèrement pour son soutien et ses conseils tout au long de notre travail. Il nous a aidés à bien structurer notre travail et nous a aiguillés dans nos recherches et dans le développement de l'extension que nous proposons.

Nous remercions également nos familles et amis respectifs de leur précieux soutien tout au long de notre parcours académique.

Nous témoignons notre gratitude aux personnes qui ont pris le temps de tester et d'évaluer notre produit et à celles qui, de près ou de loin, ont permis l'aboutissement de ce travail.

Enfin, nous remercions l'Université Catholique de Louvain de nous avoir formés, de nous avoir donné accès aux programmes utiles à notre apprentissage ainsi qu'à notre mémoire et de nous avoir appris notre futur métier. Ce parcours nous a enrichis.

Liste des abréviations

PV	Visualisation de Programme (Program Visualization)
OOP	Programmation Orientée Objet (Object-oriented programming)
GUI	Interface graphique (Graphical user interface)
IDE	Environnement de développement intégré (Integrated development environment)
UML	Langage de Modélisation Unifié (Unified Modeling Language)
JIVE	Java Interactive Visualization Environment
GPC	Graphiques de profil de complexité
PyPI	Python Package Index

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Depuis deux décennies au moins, l'informatique occupe une place de plus en plus importante dans la vie quotidienne d'un nombre croissant de personnes. Cette science commence à devenir un sujet d'apprentissage incontournable. Il est actuellement devenu très rare de réaliser un parcours d'études scientifiques sans avoir approché de près ou de loin un cours d'informatique. Au cours des dernières années, de nombreux pays ont même ajouté l'informatique aux matières de l'enseignement de base [1].

Parmi les éléments fondamentaux qui composent le domaine de l'informatique, se trouve la programmation. Elle consiste en "l'ensemble des activités liées à la définition, l'écriture, la mise au point et l'exécution de programmes informatiques ; séquence des ordres auxquels doit obéir un dispositif", selon le dictionnaire Larousse [2]. Il y a beaucoup de langages de programmation [3], par exemple JavaScript, Java, C#, C ou encore Kotlin. Il y a aussi, entre autres, Python qui est devenu l'un des dix langages informatiques les plus populaires et les plus utilisés [3]. De par sa simplicité d'utilisation, Python est également, pour de nombreux développeurs, le langage d'apprentissage de référence [4, 5].

Poussé par la volonté d'apprendre à coder en Python de façon simple, Aivar Annamaa - un chercheur et assistant à l'université de Tartu en Estonie -, a conçu Thonny [6], en 2013. En effet, cet environnement de développement intégré (IDE), open source, est destiné aux programmeurs qui code en Python et plus particulièrement au débutant en informatique. Thonny offre ainsi une interface épurée, intuitive et simple d'utilisation.

Au-delà d'un environnement spécialisé, de nombreuses études ont démontré l'efficacité de l'apprentissage de la programmation à l'aide d'outils de visualisation [7, 8]. Certaines ont également montré à quel point il était important de réaliser un outil de visualisation de programme (PV) adapté pour les débutants qui peuvent présenter de nombreuses difficultés pendant leur parcours d'apprentissage de la programmation [9].

C'est dans ce contexte que nous avons souhaité améliorer l'environnement de développement qu'est Thonny en y ajoutant un outil de visualisation. Nous espérons ainsi aider les étudiants qui l'utilisent à visualiser l'organisation des structures de données et les mécanismes générés par leur code ; et ce, afin de faciliter l'apprentissage.

1.2 Problème

Comme le confirment plusieurs études, réaliser un outil de PV est un challenge qui motive aujourd'hui encore de nombreuses recherches [10, 11]. L'objectif principalement visé est de réduire la difficulté qu'ont les étudiants à appréhender la programmation. Il a été montré que l'utilisation d'un outil de visualisation adapté était la clé d'un bon apprentissage de la programmation [12, 13].

Or, Thonny, un environnement de développement ciblé pour les débutants, ne dispose pas encore d'outil qui permettrait à ses utilisateurs de visualiser ce qu'ils sont en train de programmer. C'est pour combler ce manque que nous avons décidé d'apporter à Thonny un outil de visualisation adapté à son design épuré et à sa volonté première de cibler les programmeurs débutants.

1.3 Motivation

La motivation principale qui a poussé à la réalisation de ce mémoire est de permettre aux étudiants et programmeurs débutants qui utilisent Thonny d'apprendre plus facilement à coder en Python et, notamment, de comprendre les principes liés à la programmation orientée objet (OOP). Plus particulièrement, nous souhaitons soutenir les étudiants en première année à la faculté polytechnique de Louvain-la-Neuve. Ils suivent en effet leur premier cours universitaire d'informatique en utilisant Thonny. Etant donné que cet outil a l'avantage d'être open source, le service offert par notre travail pourra également s'étendre au-delà de ce public spécifique.

1.4 Objectifs

Le but général de ce mémoire est la réalisation d'un outil de PV dans Thonny. Cet objectif principal peut-être développé en plusieurs sous-objectifs :

1. Offrir à chacun la possibilité de visualiser ses programmes dans un environnement de développement intégré open source ;
2. Axer les fonctionnalités de nos vues pour qu'elles soient particulièrement adaptées à des programmeurs débutants ;
3. Rendre l'outil le plus intuitif, simple et facile d'utilisation possible ;
4. Intégrer cet outil de visualisation de la façon la plus harmonieuse possible à l'environnement de développement Thonny.

1.5 Approche

Afin d'atteindre efficacement les objectifs souhaités, nous avons méthodiquement découpé la réalisation de l'outil de visualisation en plusieurs parties.

Tout d'abord, nous avons cherché à mettre en évidence les différentes variables, globales et locales, créées par l'utilisateur dans Thonny. Ainsi il peut les identifier et comprendre ce qu'il est en train de programmer (figure 1.1).

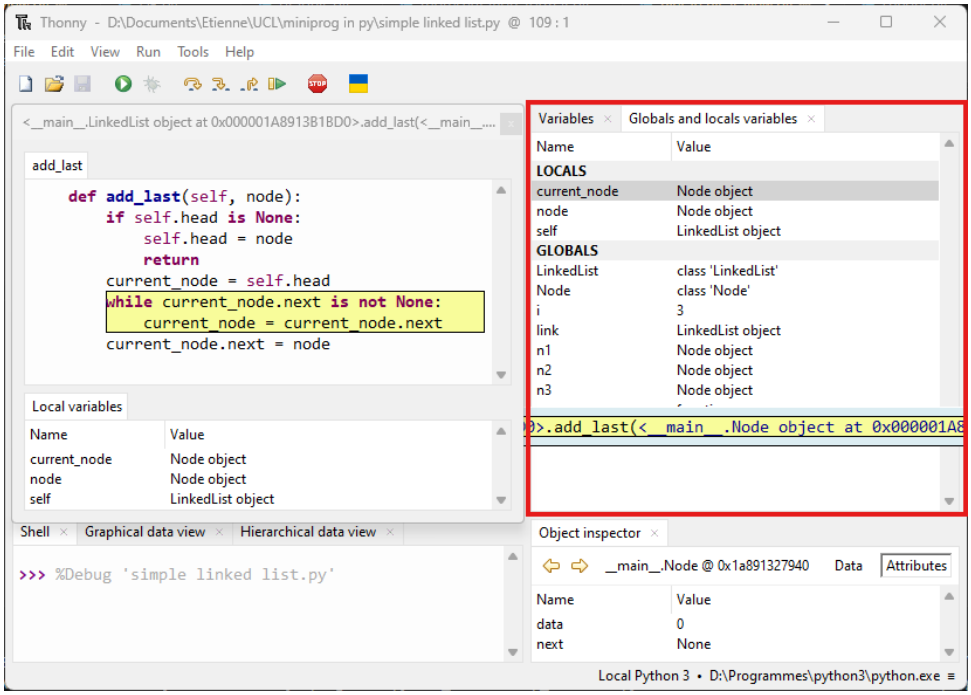


FIGURE 1.1 – Vue des variables globales et locales

Ensuite, dans le souci de visualiser les liens entre ces différentes variables, nous avons implémenté dans Thonny une vue hiérarchique liant chaque variable à celles auxquelles elle fait référence (voir figure 1.2a). Cette vue hiérarchique a l'avantage de pouvoir déjà offrir une compréhension avancée des liens entre les variables et une visualisation complète des données à l'exécution du programme codé par l'utilisateur.

Enfin, pour que notre outil soit complet, nous lui avons apporté une vue graphique, constituée de blocs et de flèches, représentant les différents objets et variables du programme de l'utilisateur et ce qui les lie (figure 1.2b). Cette dernière vue est l'aboutissement de notre outil de visualisation. En effet, par sa forme simple et ses liens clairs, elle doit permettre à un utilisateur débutant de comprendre, de façon visuelle et intuitive, ce qui se passe au sein de son programme.

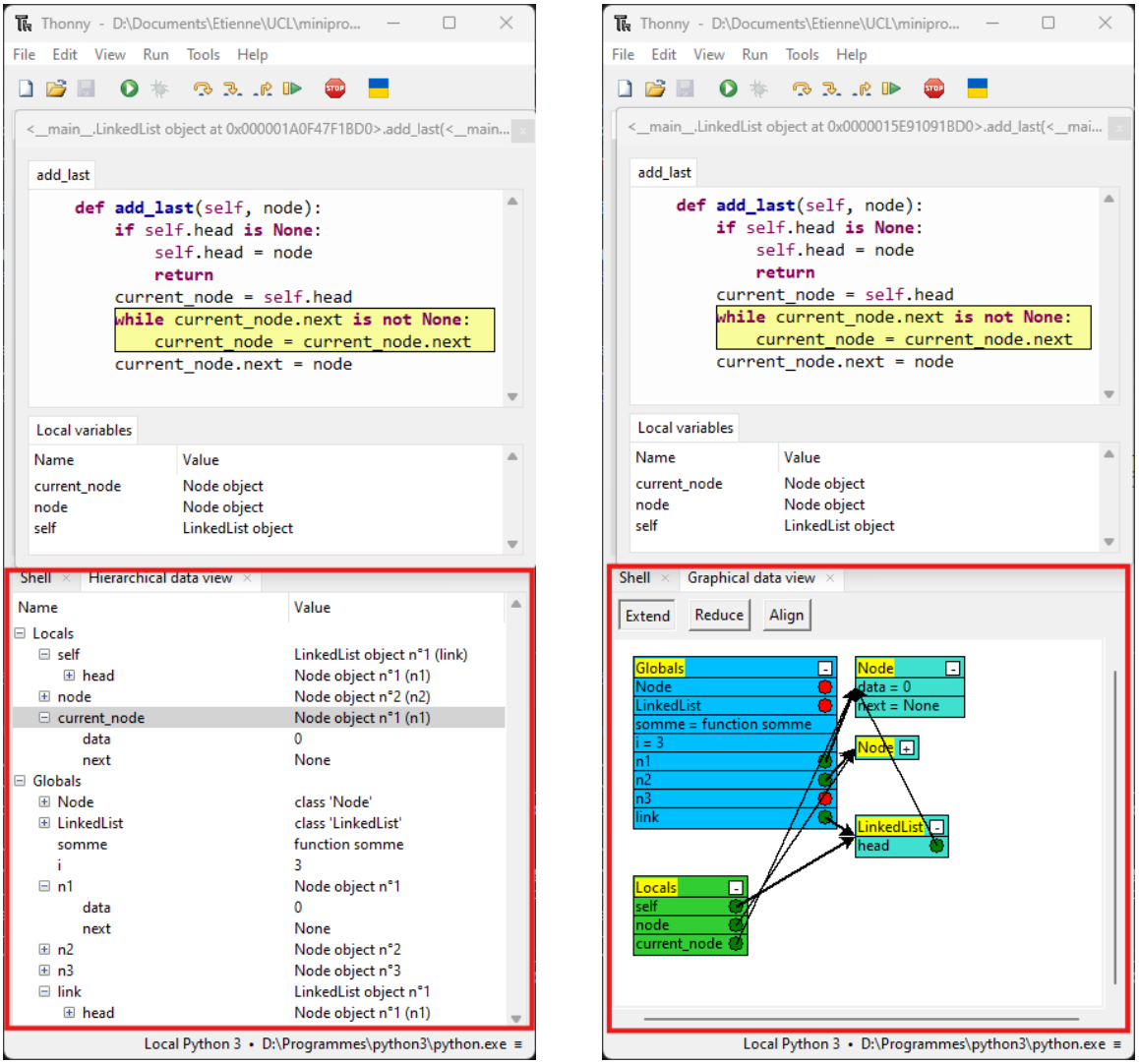
1.6 Contributions

Notre travail apporte plusieurs nouveautés :

— L'ajout à Thonny d'un outil de visualisation qui permet aux débutants de comprendre plus facilement leur programme ;

— L'accessibilité, pour tout programmeur, à un environnement de développement intégré open source proposant un outil de visualisation en son sein ;

— Un nouvel outil de PV Python qui cible les programmeurs débutants et qui possède, dans cette perspective, une apparence épurée. En outre, il offre une utilisation intuitive et interactive.



(a) Vue hiérarchique

(b) Vue graphique

FIGURE 1.2 – Vues hiérarchique et graphique

Le lien vers le répertoire Github de notre outil est en annexe.

1.7 Aperçu

La suite de ce document est structurée de la façon suivante : le prochain chapitre va présenter succinctement les bases théoriques et les autres outils de visualisation de référence à connaître. Ensuite, nous discuterons des limitations que présentent ces outils. Ceci nous amènera à définir le besoin auquel nous souhaitons répondre. De là, notre solution sera exposée au travers des différentes vues qui la constituent. Nous discutons par après la validité de notre outil et présentons les changements que nous y avons apporté suite aux retours de nos testeurs. La section qui suit propose un exemple d'utilisation présentant visuellement notre solution. Pour terminer, nous aborderons quelques pistes de travail pour l'avenir et conclurons en présentant les principales contributions de cette recherche.

Chapitre 2

État de l'art

2.1 Contexte théorique

2.1.1 Python

Python [14] est un langage de programmation inventé par Guido van Rossum en 1989. Il est largement utilisé dans le monde pour diverses raisons. Voici des points forts qu'il est intéressant de noter :

— Python est connu pour sa simplicité et sa lisibilité. Il a une syntaxe simple et concise ; ce qui permet un meilleur apprentissage, comme le prouve cette étude comparative sur les syntaxes de différents langages informatiques [15]. En effet, Python est un langage de haut niveau typé dynamiquement [16] ; ce qui signifie que les variables n'ont pas de types de données déclarés explicitement par le programmeur ; leurs types sont déterminés par l'interpréteur Python, au moment de l'exécution. Cette caractéristique offre une plus grande flexibilité et réduit la complexité du code, ce qui a pour effet de rendre le langage plus digeste pour des débutants en informatique.

— Il est gratuit, open source et libre d'utilisation [17].

— Python a une grande communauté active qui permet l'accès à des forums, des tutoriels et une documentation abondante [3].

— Ce langage a une bibliothèque standard très riche [18]. Grâce à sa large communauté, il existe un référentiel en ligne de paquets de logiciels pour Python, appelé Python Package Index (PyPI) [19]. Cet outil permet de télécharger et partager des paquets qui étendent les fonctionnalités de Python. PyPI est l'un de plus anciens et plus large référentiel de logiciels. La bibliothèque standard de Python et PyPI fournit à un utilisateur Python des outils pour réaliser efficacement et simplement une énorme quantité de tâches variées telles que le traitement de texte, des calculs mathématiques, la gestion réseau et encore bien d'autres.

— Python est un langage multiparadigme [20] qui n'oblige pas ses utilisateurs à programmer d'une manière particulière. Il permet de coder dans un style procédural, orienté objet, impératif, réflexif, concurrent (même si Python ne crée pas de véritables applications multitâches), fonctionnel et dans n'importe quel mélange

de styles. Le développeur peut programmer avec le paradigme qui lui plaît ; ce qui est attractif notamment pour des enseignants car cela leur permet d'enseigner plusieurs manières de coder sans avoir besoin d'introduire un nouveau langage à chaque fois. Comme il s'agit d'un langage orienté objet, cela nous intéresse particulièrement pour ce mémoire car cela permet aux utilisateurs de créer des codes complexes qui suivent de près les entités physiques ou mathématiques qu'ils modélisent. De plus, ces codes sont modulables et réutilisables [21]. Python intègre la plupart des concepts de la programmation orientée objet (OOP) comme l'héritage, l'héritage multiple et le polymorphisme. Par contre, il ne supporte pas l'encapsulation : Python n'a pas de membres de classe privés ou protégés, ils sont tous publics.

Ses qualités en font un langage parfait pour du développement rapide, autant pour ceux travaillant sur des projets complexes que pour les novices dans le développement informatique ; un public qui nous intéressent ici plus particulièrement.

Notons toutefois que, si c'est intéressant que Python soit un langage haut niveau, cela a malgré tout ces défauts. Le typage dynamique, notamment, rend le code moins performant [22]. Il empêche le langage de réaliser de nombreuses optimisations lors de la compilation car les types ne sont pas connus à l'avance. Ainsi, les vérifications et conversions de types doivent être effectuées durant l'exécution, ce qui peut ralentir le programme et le rendre moins performant par rapport à des langages à typage statique comme C++ ou Java.

Développement d'Interfaces Graphiques Utilisateur - Tkinter

Tkinter [23] est la bibliothèque standard de Python pour la création d'interfaces graphiques (GUI). Cet outil permet de développer des applications avec une interface utilisateur graphique en utilisant le langage de script Python. Tkinter est une interface vers la boîte à outils Tk, qui est un système de widgets multiplateforme. Un widget [24] est une classe qui représente un élément de l'interface utilisateur. Il peut interagir avec d'autres widgets et avec du code via des évènements.

Les points positifs de Tkinter [25] sont :

— Intégré par défaut dans Python, Tkinter ne nécessite aucune installation supplémentaire.

— Comme Python, il est gratuit et offre une licence de logiciel libre. Cela implique donc l'utilisation, la modification et la distribution libre de l'outil.

— Les applications développées à l'aide de cet outil sont multiplateformes. Cela signifie qu'elles fonctionnent sur toutes les plateformes compatibles avec Python - dont Windows, macOS et Linux - sans qu'aucune modification du code ne soit nécessaire.

— Tkinter est très complet et offre une vaste gamme de widgets standard (boutons, étiquettes, entrées de texte, menus, etc.).

— Cette interface bénéficie d'une documentation abondante et d'une communauté active qui facilitent ainsi l'apprentissage et la résolution des problèmes.

Quant à ses principaux défauts, nous en relevons ici deux. D'une part, le manque de performance pour des applications graphiques très complexes et, d'autre part, une apparence désuète en comparaison avec les standards modernes d'interface utilisateur [26].

NetworkX

NetworkX [27] [28] est une bibliothèque Python open source utilisée pour la création, la manipulation et l'étude des structures de diagrammes réseaux complexes. Cela comprend par exemple : les graphes orientés et non orientés, les graphes simples, les graphes avec des arêtes parallèles et ceux avec des arêtes qui bouclent sur un nœud. Cette bibliothèque a été créée en 2004 par Aric Hagberg, Dan Schult et Pieter Swart au Los Alamos National Laboratory.

Dans les diagrammes réseau de NetworkX, les nœuds peuvent être n'importe quels types d'objets ou de variables et les arêtes peuvent avoir des attributs, comme des poids ou des étiquettes. Cette bibliothèque a plusieurs algorithmes pour l'analyse des graphes dont : le parcours en largeur et en profondeur, les chemins les plus courts, les composants fortement connectés, et les mesures de centralité. NetworkX n'est pas une bibliothèque de visualisation graphique, il est conseillé de l'intégrer avec l'une de celle-ci.

Autres librairies de visualisation de programme

Il existe bien d'autres outils pour Python qui font de la visualisation de diagramme réseau. En voici une brève introduction des alternatives principales :

— **Matplotlib** [29] est une bibliothèque de visualisation 2D. Elle peut être utilisée avec NetworkX pour afficher des diagrammes réseau. Cette librairie n'est pas spécialisée dans les diagrammes réseau, elle permet de visualiser les graphes créés avec un autre outil, par exemple NetworkX. Les graphes seront placés en un endroit optimal pour éviter que les noeuds ou les arêtes ne se superposent. Matplotlib ne permet pas une interactivité au point d'avoir une fonctionnalité de "drag-and-drop"

— **Graphviz** [30] avec **PyGraphviz** [31, 32], Graphviz est une bibliothèque pour la visualisation de graphes et réseaux sous forme de diagrammes. Les outils fournis avec cette bibliothèque sont efficaces pour la création de diagrammes hiérarchiques et pour la disposition automatique des nœuds. Et PyGraphviz est une interface Python pour Graphviz. Elle permet de manipuler les graphes en Python et de les visualiser à l'aide de Graphviz. Une fois de plus, cette combinaison d'outils ne permet pas de décider de la disposition des nœuds. Elle fournit des fichiers immuables, tels que PDF ou PNG, ou des sorties textuelles. Les fichiers fournis ne sont pas interactifs.

— **Bokeh** [33] est une bibliothèque de visualisation interactive qui peut être utilisée pour créer des diagrammes réseau interactifs. L'interactivité qu'elle permet comprend, entre autres, le zoom, le plan, et l'affichage de détails au survol mais elle ne supporte pas le glisser-déposer. Elle est spécialisée pour afficher ses graphes dans

un navigateur web. Intégrer ces fichiers-là à Thonny est complexe et nécessite une intégration web spécifique.

— **PyVis** [34, 35] est une bibliothèque Python qui utilise la bibliothèque JavaScript *vis.js* pour créer des visualisations de graphes interactives. Avec Networkx, elle peut réaliser des diagrammes réseau interactifs ce qui permet de faire du drag-and-drop, de zoomer, de faire des sélections, etc. Mais sa visualisation se fait dans un navigateur ou un notebook Jupyter. L'intégrer à Thonny n'est pas simple.

— **Dash Cytoscape** [36, 37] est une bibliothèque utile pour créer des visualisations de graphes interactives basées sur *cytoscape.js*. Tout comme PyVis, elle permet une grande interactivité dont le drag-and-drop, mais elle est aussi principalement conçue pour le web.

Nous décririons plus tard, dans la section 4.4.3, pourquoi nous avons utilisé pour notre outil de visualisation NetworkX avec Tkinter, pourquoi nous avons jugé ces bibliothèques utiles et nécessaires pour l'implémentation de notre outil et pourquoi les autres librairies existantes ne correspondaient pas à nos besoins.

2.1.2 Thonny

Thonny [6] a été créé en 2013 par Aivar Annamaa à l'Université de Tartu en Estonie. Il s'agit d'un environnement de développement intégré (IDE) spécifique à Python. Il est open source et multiplateforme en s'adaptant à Windows, macOS et Linux. Son interface simple le rend fortement attrayant pour les débutants en programmation Python. Il leur offre notamment des feedbacks immédiats et des tutoriels intégrés.

Bien que son interface initiale soit épurée, Thonny est un outil assez complet et permet de rajouter facilement de nombreuses fonctionnalités ou autres extentions [24]. Les fonctionnalités de base qu'il faut activer sont notamment utiles pour aider à la compréhension de plusieurs concepts de programmation comme la liaison de variables, la récursion, le débogage, etc.

Un dernier élément qu'il est intéressant de signaler est le débogueur de thonny [38]. Il permet de visualiser l'exécution pas à pas du programme que l'utilisateur a créé, une instruction à la fois. Il permet aussi d'observer des appels de fonctions récursives ainsi que d'afficher les valeurs des variables en mémoire à chaque étape de l'exécution du programme. La figure 2.1 montre Thonny en train de déboguer du code avec les fonctionnalités "Variables" et "Inspecteur d'objet" ouvertes, qui permettent l'affichage des variables et des objets ainsi que leur état et leur valeur en mémoire.

2.1.3 La visualisation de données

La visualisation de programme (PV) n'est pas un domaine nouveau. La première source fiable que nous avons trouvée définit formellement le terme en 1986 [39]. Trois ans plus tard, en 1989, B. A. Myers le définit de manière plus développée [40] :

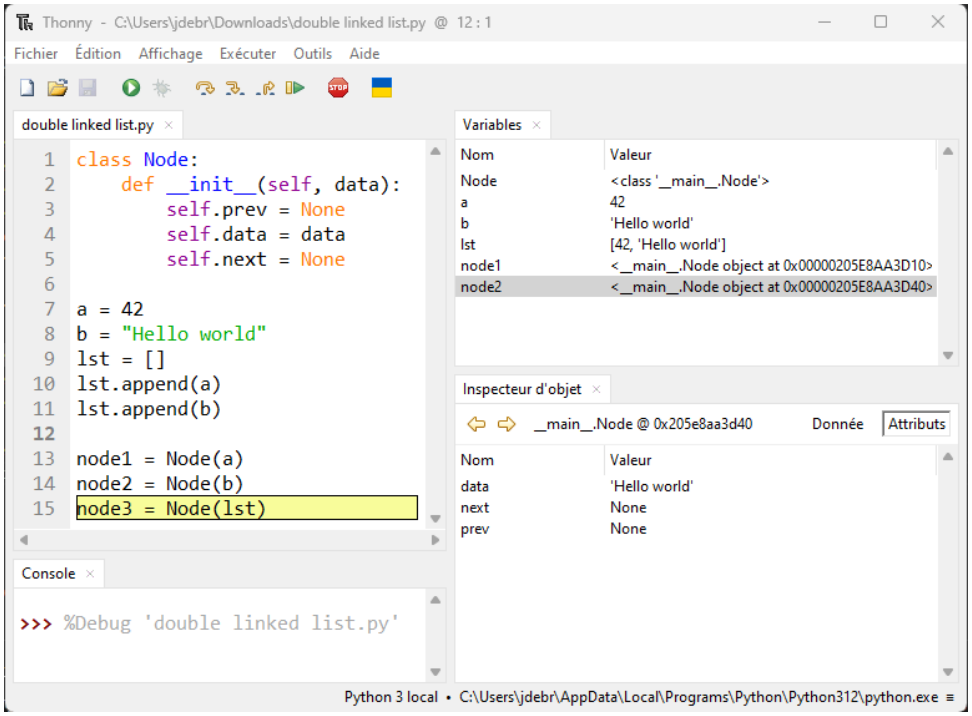


FIGURE 2.1 – L'interface graphique de Thonny avec "Variables" et "Inspecteur d'objet"

> ...dans la visualisation de programme, le programme est spécifié de manière textuelle, conventionnelle et les graphiques, bidimensionnels (ou plus), sont utilisés pour illustrer certains aspects du programme ou de son exécution au moment de l'exécution.

De plus, il précise que dans lesgraphiques bidimensionnels (ou plus), cela ne comprend pas les langages textuels conventionnels car les compilateurs ou les interprètes les traitent comme de longs flux unidimensionnels.

B. A. Myers mentionne également qu'il y a deux types de visualisations de programme : la visualisation dynamique et la visualisation statique. La première englobe les systèmes qui offrent une animation du programme en cours d'exécution. Celle dite « statique » se limite à des visualisations instantanées du programme, uniquement à certains moments.

Enfin, il déclare que la visualisation sous forme de graphique a été créée et est intéressante car "le système visuel humain et le traitement de l'information visuelle humaine sont clairement optimisés pour les données multidimensionnelles", cette affirmation a plusieurs fois été étudiée par la suite et sera développée dans le point 2.1.4.

Pour comprendre le sujet de notre mémoire il est intéressant de comprendre les autres concepts de visualisation apparentés à ce que nous faisons :

— "visualisation de données" [41] : n'a pas de véritable lien avec le programme, il s'agit de représenter des données abstraites ou théoriques pour pouvoir observer leur structure, repérer des modèles et des récurrences et, surtout, d'en tirer des

conclusions.

— "visualisation de code" [42] : il s'agit de mettre en évidence la structure du code, du texte même. Cela est donc fort dépendant du langage. Ne représente ni les variables créées en mémoire ni leur valeur.

— "visualisation d'algorithme" [43] : se concentre sur la représentation d'un algorithme et des structures de données. L'objectif est avant tout pédagogique. C'est la visualisation statique ou dynamique d'abstractions de haut niveau qui décrivent le logiciel, par opposition avec la PV qui est de bas niveau [44].

— "visualisation de software" [45] : est un terme utilisé pour décrire tous les outils qui aident à visualiser un software, et ce, quel que soit le type de visualisation utilisée et décrite ci-dessus. Si un outil ou une fonctionnalité d'un programme aide à comprendre et déboguer un software, alors cela rentre dans la définition de "visualisation de software".

La visualisation de software en général, et donc, par extension, aussi celle de programme, a un double objectif : la pédagogie et le débogage, en facilitant à la fois la compréhension humaine et l'utilisation efficace des logiciels informatiques. La visualisation a comme rôle principal de transmettre des informations. Tout le défi est alors de transmettre ces informations d'une manière compréhensible, efficace et facile à retenir [12].

2.1.4 La didactique en informatique

Très vite après l'apparition de la PV, dès 1992, le lien évident avec la didactique et l'apprentissage de l'informatique s'est établi [46]. De même, pour le lien entre la didactique et la visualisation de logiciel [47].

De nos jours, le problème ne se pose pas seulement en première année de Bachelier à l'université mais également en secondaires, voire même en primaires. Principalement, les enseignants et professeurs de l'enseignement secondaire ont des difficultés à transmettre ce que les auteurs de l'article [1] appellent la "pensée informatique". Cela comprend : la décomposition d'un programme ou d'un algorithme en composants (décomposition), la réduction de la complexité inutile (abstraction), l'identification des processus (algorithmes) et la recherche des points communs ou des modèles (généralisation). Les problèmes de décomposition et d'abstraction peuvent être résolus par une meilleure visualisation de ce que le programme fait. Il existe par ailleurs un besoin de développer la résilience des élèves pour qu'ils n'abandonnent pas la résolution d'un problème si ils prennent du temps à trouver la solution.

Naps et al. (2003) suggèrent que rendre l'abstraction inhérente des éléments de base en informatique plus concrète grâce à des représentations graphiques permet de mieux comprendre leur fonctionnement [48]. Malgré l'intérêt en apparence évident pour l'éducation en informatique, Rössling et Velázquez-Iturbide (2009) notent que l'utilisation d'outils de visualisation est inférieure aux attentes des développeurs [49]. Une étude du groupe de travail ITiCSE 2002 a identifié plusieurs obstacles tels que les difficultés d'intégration

dans les cours, le manque de temps, les problèmes techniques et le manque de preuves d'efficacité sur le plan éducatif [48]. Ben-Bassat Levy et Ben-Ari (2007) [50] expliquent cette réticence par deux raisons. D'une part, les outils pédagogiques à eux seuls ne suffisent pas ; il est donc nécessaire de les intégrer dans le curriculum, ce qui demande de réaliser des changements dans le curriculum. Or les enseignants n'ont pas le temps de les réaliser. D'autre part, les enseignants sont souvent réticents à abandonner leur position d'autorité, ce qui fait qu'il peuvent résister aux changements qui risquent de déplacer le centre d'apprentissage vers l'élève, réduisant ainsi leur rôle à celui de facilitateurs. Ils soulignent le besoin de formations pour renforcer le sentiment de contrôle des éducateurs. Shaffer et al. (2010) observent aussi que beaucoup de visualisations d'algorithmes sont de faible qualité et se concentrent sur des sujets plus faciles, ce qui fait que les enseignants ont des difficultés à trouver ce dont ils ont besoin [51].

Face à tout ces obstacles, il nous paraissait intéressant d'essayer de réaliser un outil de visualisation qui essaye de pallier tous ces problèmes. Mais, avant tout, il fallait s'assurer de l'efficacité d'un tel outil d'éducation. Or, depuis que cette inquiétude est apparue, plusieurs études sont sorties pour essayer d'y répondre :

— Une étude [12] menée en 2011 sur un échantillons de 400 étudiants avec l'outil "Jeliot3" (décrit de manière plus détaillée à la section 2.2.2) conclut que les étudiants ont de meilleurs résultats en utilisant cet outil. Les auteurs avancent aussi l'idée que "Jeliot3" fonctionne mieux sur des étudiants débutants.

— Une autre étude encore a été menée avec l'outil "Jeliot3" [52] sur 54 étudiants pendant 15 heures de cours réparties sur cinq semaines. Elle observe que le programme a clairement permis aux étudiants d'améliorer leurs performances en OOP sur du court terme. Cependant, cela est moins efficace sur du long terme. L'étude conclut que la PV est efficace mais pas suffisante.

— Cette recherche [13] étudie l'impact de l'utilisation de ViLLE (un outil décrit à la section 2.2.2) avec 22 élèves de l'enseignement secondaire. 15 d'entre-eux n'ont pas utilisé ViLLE et les sept autres élèves ont utilisé l'outil durant six semaines. Cette étude conclut que ViLLE est bénéfique à court comme à long terme pour apprendre les compétences de base en programmation. Les auteurs ajoutent que la PV de ViLLE a beaucoup aidé les élèves à comprendre comment retracer l'exécution entre le programme principal et les sous-programmes, ainsi que pour suivre l'exécution des appels de fonction.

— Cet article [53] évalue PythonTutor (décrit à la section 2.2.1). L'un de ses résultats est que la PV est plus adaptée aux étudiants habitués à apprendre de manière pratique. Il en va de même, mais dans une mesure beaucoup plus faible, pour ceux qui sont habitués à une méthode d'apprentissage plus théorique.

— Cette étude [54] un peu plus différente analyse l'effet de la PV sur la motivation des étudiants. Cette étude n'est pas parmi les plus exhaustives mais elle analyse la motivation des étudiants et les résultats obtenus montrent que la PV augmente à la fois la motivation intrinsèque et la motivation extrinsèque.
Les auteurs de l'article définissent la motivation intrinsèque comme : "the manifes-

tation of the human tendency toward learning and creativity" (la manifestation de la tendance humaine à l'apprentissage et à la créativité [notre traduction]). En d'autre mot : être motivé et avoir le désir d'apprendre ou d'être créatif parce que l'activité proprement dite nous intéresse ou nous plaît. Les auteurs définissent la motivation extrinsèque par "This refers to the implication of external aspects in the subject"(Il s'agit de l'implication d'aspects externes envers le sujet [notre traduction]).
Ce type de motivation se divise en deux sous-ensembles. Le premier sous-type est la motivation extrinsèque via une régulation identifiée. Ce type de motivation se manifeste lorsque l'individu reconnaît que la tâche ou le travail est essentiel pour atteindre ses objectifs personnels et que cela correspond à ses valeurs. Ce n'est pas le travail même qui est intéressant mais bien le résultat final ou la nouvelle capacité apprise qui est intéressante. Ensuite, le deuxième sous-type est la motivation extrinsèque via une régulation externe. Dans ce cas, les actions d'une personne sont guidées par des facteurs externes plutôt que par des motivations internes. L'individu agit principalement pour obtenir une récompense ou éviter une punition. Par exemple, en informatique, la motivation intrinsèque c'est le plaisir de coder, la motivation extrinsèque via une régulation identifiée c'est le plaisir d'avoir produit un programme utile ou d'être devenu capable de créer un programme utile. Pour finir, la motivation extrinsèque via une régulation externe c'est le plaisir d'avoir, grâce à ce travail, des bons points à l'examen ou son salaire à la fin du mois.

Même si parmi tous les articles présentés, certains se contredisent légèrement, il est quand même possible de dire qu'il y a un consensus sur le fait que les outils de visualisation sont efficaces. Bien d'autres articles arrivent à la même conclusion. Par exemple, cette étude de la littérature sur l'enseignement de l'introduction à la programmation, écrite par A. Pears, S. Seidman et cinq autres auteurs [55] ou encore cet article [53] qui sera décrit plus tard.

Stephen Wolfram [56] a écrit un jour ceci : « Les gens ne se souviennent que de 15 % de ce qu'ils entendent et 25 % de ce qu'ils voient, mais ils se souviennent de 60 % de ce avec quoi ils interagissent. ». C'est une belle citation qui doit être vérifiée. Elle l'est dans cet article, écrit par R. Pinter, D. Radosav et S. Maravić Čisar [57], qui démontre que les animations interactives peuvent contribuer de manière significative à augmenter la vitesse d'apprentissage. Cela nous a encouragés à réaliser un outil qui permette non seulement de faire de la PV mais qui soit également interactif.

2.2 Travaux apparentés

Cette section va présenter plusieurs outils déjà existants qui font, entre autres, de la PV. Nous allons commencer par présenter "PythonTutor" qui nous semble le plus proche de ce que nous faisons. Plusieurs autres outils qui font également de la PV seront présentés plus brièvement.

2.2.1 PythonTutor

PythonTutor [58] est un outil pédagogique de visualisation en ligne pour du code en Python, JavaScript, C, C++, et Java. Créé par Philip Guo, il permet de parcourir

l'exécution d'un code en entier ou un pas à la fois, tout en ayant une PV de l'état de la mémoire et de la structure de données qui se forme au même rythme que l'utilisateur parcourt le code [5]. Cela facilite la compréhension de concepts complexes tels que la récursivité, les références d'objets et la gestion de la mémoire. Les utilisateurs peuvent comprendre comment leur programme se comporte et pourquoi tel ou tel comportement se produit.

La figure 2.2 représente la visualisation offerte par PythonTutor d'une petite liste doublement chaînée de trois éléments. Grâce à ce graphique, il est possible de clairement identifier la structure de données qui a été créée en mémoire durant l'exécution de ce code. Chaque objet, qui est ici chaque noeud de la liste, a son propre carré. Leurs variables d'instance sont facilement visibles si elles ont comme valeur une valeur primitive. La valeur est notée, et si elle a comme valeur une référence à un autre objet, cette référence est visualisée à l'aide d'une flèche vers cet autre objet. Cette représentation graphique est très visuelle et aide à comprendre la structure de données stockées en mémoire ainsi que les références.

Bien que ce soit un outil assez complet et utile, il a des défauts. Cette étude sur l'efficacité d'un outil de visualisation de programme pour l'initiation à la programmation, écrite par Oscar Karnalim et Mewati Ayub [53], en développe plusieurs. Tout d'abord, elle précise que les fonctionnalités de PythonTutor sont très utiles pour apprendre la programmation en général mais pas suffisantes. Dès lors, il est préférable d'apprendre certains modules avancés via un autre outil tel que la visualisation d'algorithmes. Ensuite, l'interface utilisateur de PythonTutor devrait être plus interactive pour garder les étudiants concentrés.

Le fait d'être un outil en ligne peut être un atout car cela permet d'être facilement accessible. Cela n'est pourtant vrai que dans des pays ou des régions où internet est facile d'accès ce qui n'est pas encore le cas partout. De plus, le fait que PythonTutor soit en ligne et pas rattaché à un IDE le rend, certes, indépendant et donc à la portée de tout le monde, mais cela oblige les utilisateurs de constamment passer de l'un à l'autre. En plus d'être une perte de temps, sa situation externe affecte le processus d'apprentissage des débutants en informatique car elle leur fait perdre leur concentration.

Un autre point qu'il serait intéressant d'améliorer a été soulevé par certains des étudiants sur lesquels l'étude a été faite. Ils affirment que les changements des variables, lors du parcours de l'exécution, devraient être présentés de manière plus déclarative, par exemple en mettant en évidence la variable modifiée.

PythonTutor [58] a aussi des limites dans ce qu'il est capable de supporter : il ne supporte pas la plupart des librairies externes, ni l'interaction avec des ressources externes comme des fichiers ou des bases de données, ni les codes trop longs, ni les codes qui prennent trop de temps à s'exécuter ou ceux qui nécessitent trop de mémoire. Les créateurs argumentent ce choix d'implantation en disant qu'afficher trop d'objets ne ferait que rendre la visualisation illisible. Cela est vrai si la visualisation affiche tous les objets - ce que PythonTutor fait - au lieu de permettre à l'utilisateur d'afficher seulement ceux qu'il a décidé de voir. Les créateurs disent aussi que c'est un outil pour débutant qui n'a pas besoin de savoir fonctionner avec de la programmation très avancée.

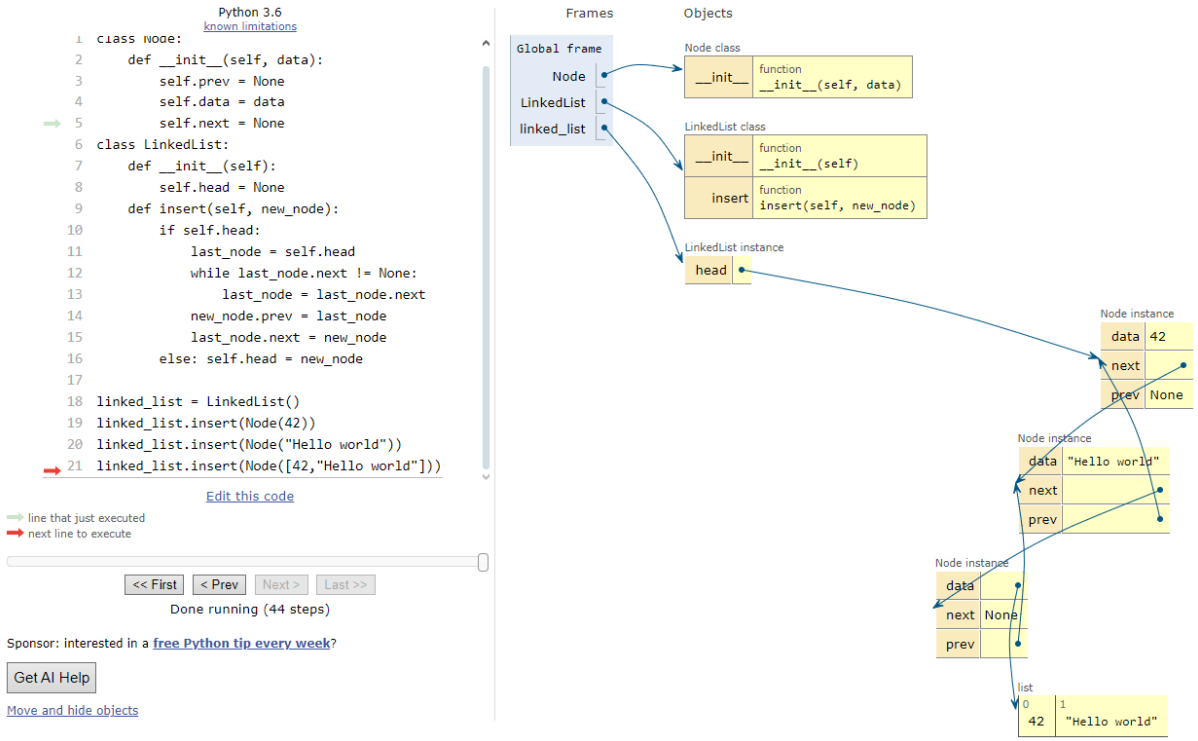


FIGURE 2.2 – L'interface graphique de PythonTutor avec une liste chainée entièrement visualisée

Nous allons décrire plus tard, en particulier dans la section "Différences avec PythonTutor" 3.1, les différences fondamentales qu'il y a entre PythonTutor et outil de visualisation que nous avons produit, quels sont les contraintes de PythonTutor que nous avons dépassées et pourquoi nous les avons gardées ou retirées.

2.2.2 Autres outils de visualisation

Il existe bien d'autres outils de visualisation, chacun avec ses propres spécificités. En voici une brève introduction. Cette présentation des différents outils existants est basée sur des articles qui les comparent, à savoir ces deux articles : [59] et [60].

— **JaguarCode** [61] est un outil en ligne interactif permettant de visualiser dynamiquemment et de déboguer du code Python, Java, JavaScript, TypeScript, Ruby, C, C++, et Kotlin. Il offre à l'utilisateur des diagrammes statiques pour les classes, une vue dynamique pour les instances d'objets et relations entre objets. Il est destiné à des étudiants ayant déjà des connaissances en informatique mais qui ont des difficultés à comprendre les objets en tant que concept. A ce jour, il est encore en cours de développement et son accès n'est pas encore ouvert au public.

— **jGRASP** [62] est un IDE léger, conçu pour améliorer la compréhension du code source en Java, Python, C, C++ et Ada grâce à ses fonctionnalités de visualisation. Il peut générer des diagrammes de structure de contrôle qui sont utilisés pour visualiser le code source et la conception d'un programme. Il réalise de la visualisation

des structures de données multi-objets (comme les listes chaînées, les tables de hachage et les arbres binaires), un peu de visualisation UML (Unified Modeling Language), des graphiques de profil de complexité (GPC) ainsi que plusieurs vues du code source, des objets de bas niveau et des visualisations de haut niveau. Ses visualisations sont statiques, il n'est donc pas très interactif. Cet IDE manque de visualisation générale des objets et n'aide pas à comprendre comment concevoir des classes, mais il est utile pour le suivi des données dans des grandes structures de données et pour des étudiants apprenant ces structures. Il ne fournit pas non plus de surlignage de code et nécessite donc une connaissance considérable de la programmation. Cet outil n'est donc probablement pas très approprié pour les novices.

— **Javalina Code** [63] est un outil en ligne, conçu pour visualiser spécifiquement du code en java. Il permet de créer des projets via un service cloud et de compiler du code. Il génère des diagrammes de classes pour chaque classe dans le projet et des graphiques de type tableau pour chaque objet dans le programme en cours d'exécution. Ses visualisations sont statiques, elles aident beaucoup à la compréhension des relations d'héritage. Javalina Code n'offre pas de diagrammes de séquence.

— **JIVE** (Java Interactive Visualization Environment) [64] est une extention pour l'IDE Eclipse. Cette extention ne fonctionne qu'avec le langage java. Elle offre comme visualisation des diagrammes de contours, des graphiques dynamiques de type tableau pour les objets, des diagrammes de séquence et des programmes en cours d'exécution. Elle est l'un des outils de visualisation qui fournit le plus de détails pour un objet individuel. L'extention ne fournit pas de diagrammes de classes statiques mais en échange JIVE se concentre pour donner le plus d'informations détaillées sur chaque objet, directement dans la vue dynamique. L'extention permet aussi d'afficher des diagrammes de séquence qui aident à la compréhension des interactions entre les objets.

— **BlueJ** [65] est un IDE spécialement conçu pour l'apprentissage de la programmation en Java. Il a été développé pour introduire progressivement les concepts de la OOP. Il permet à l'utilisateur d'interagir graphiquement avec les objets. Il utilise la représentation standard UML, constituant ainsi un très bon outil pour introduire et visualiser les concepts OOP. Parmi ceux-ci : l'abstraction et l'encapsulation des données, l'héritage et le polymorphisme, la transmission de messages et les instances individuelles d'un objet, qui sont normalement complexe à comprendre pour les étudiants. Sa visualisation manque cependant de détails quant à la représentation des objets, notamment de leur méthodes et attributs de classe. Il est l'un des plus anciens outils de visualisation fonctionnelle.

— **Jeliot3** [66] est un outil pédagogique interactif développé à l'Université de Joensuu. Conçu pour les novices, il permet de visualiser l'exécution de programmes Java de manière graphique et détaillée. Son but est de faciliter l'apprentissage et l'enseignement de la programmation procédurale et orientée objet. Il offre la possibilité de visualiser et interagir avec les graphiques générés à côtés du code écrit. Jeilot3 priorise une PV global plutôt qu'une visualisation individuelle pour chaque classe. Les graphiques produits contiennent des graphiques similaires à un diagramme de

contour avec des graphiques séparés pour les variables d'objet.
Dans l'article "Classifying Program Visualization Tools to Facilitate Informed Choices : Teaching and Learning Computer Programming" écrit par Stephen Mutua et quatre autres auteurs [60], ils écrivent que malgré le fait que Jeliot3 se limite à Java, il ne reconnaît pas certains mots-clés et fonctions Java standards. Cela pourrait être déroutant pour les étudiants débutants, rendant difficile pour eux de passer à l'utilisation d'un environnement de développement intégré Java (IDE) plus classique. Cela va à l'encontre de l'article "Various Utilizations of an Open-Source Program Visualization Tool, Jeliot3" écrit par Roman Bednarik, Andrés Moreno et Niko Myller [66]. Cet article définit Jeliot3 comme un outil conçu pour débutants car il possède une interface utilisateur simple aux visualisations complètes. Celles-ci expliquent à l'étudiant toute l'exécution du programme.

— **PandionJ** [67] est une extention pour l'IDE Eclipse, conçue pour visualiser du code en Java. PandionJ propose des visualisations interactives des objets, des classes, et des relations entre eux, ainsi qu'une représentation graphique de la pile d'appels et de l'état de la mémoire durant l'exécution du code. Ces visualisations détaillent bien l'instance d'un seul objet mais n'offrent pas de descriptions générales des classes. PandionJ visualise aussi les références nulles et fait la différence entre les objets et les types primitifs de données.

— **VILLE** [68] est un outil de PV utilisé pour créer et éditer des exemples de programmation et pour observer les événements dans les exemples, pendant leur exécution. Son objectif est de soutenir le processus d'apprentissage des programmeurs novices. Il est possible d'ajouter des exemples de programmation à VILLE, puis de visualiser leur exécution lors de cours ou sur le Web. C'est un outil bien équipé qui peut visualiser Java, Visual Basic, Python, C++ et Pseudo-code. En raison de sa prise en charge multiple des langages de programmation, il fournit une vue parallèle du code dans les langages choisis. C'est une fonctionnalité très puissante car l'utilisateur peut écrire un programme dans le langage qu'il connait le mieux et l'outil le convertit dans un autre langage de son choix.

— **Jeroo** [69] est un outil assez différent des autre outils présentés. Avec le code à gauche et l'équivalent d'un labyrinthe dessiné à droite, l'utilisateur doit, grâce à du code possédant une syntaxe limitée, aider un Jeroo (qui est une espèce d'animal proche du Kangourou) à trouver son chemin. L'exécution du code à gauche va animer le labyrinthe avec le Jeroo. Cela aide à comprendre les notions de base de l'utilisation d'objets pour résoudre des problèmes. Cela permet aussi d'écrire des méthodes qui prennent en charge une décomposition fonctionnelle de la tâche. Mais Jeroo nécessite une compréhension plus approfondie pour pouvoir l'utiliser efficacement. Cela est dû au fait qu'il est développé métaphoriquement à partir du comportement de Jeroo, l'animal.

— **Alice** [70] est un outil qui a été créé pour rendre la première expérience de programmation plus amusante. Alice se concentre sur l'introduction des concepts de programmation orientée objet en utilisant la syntaxe de Java, C++ et C#. Bien qu'il s'agisse d'un environnement intéressant, Alice est trop basique pour être utilisé dans un établissement d'enseignement supérieur à des fins d'enseignement.

En effet, il n'offre pas une transition en douceur vers un véritable environnement de programmation et peut donc être source de confusion.

— **PyCharm** [71] est un IDE très complet pour du code en Python. PyCharm ne génère aucun graphique qui pourrait aider à la visualisation. Cependant, il a une PV sous forme de vue hiérarchique. Cette vue a son importance car elle a inspiré l'un des éléments de notre travail. La vue hiérarchique de PyCharm permet de voir toute les variables créées durant l'exécution du code ainsi que les instances d'objets. Elle permet aussi d'identifier quelle variable d'instance est reliée à quel objet.
La figure 2.3 montre l'utilisation de l'outil de visualisation des structures de données de PyCharm avec une liste cyclique de deux éléments. Il est possible d'observer sur cette vue que l'élément "circular_list" a une référence vers l'élément "head". Cet élément est un objet de type "Node", qui a une valeur égale à "A" et une référence "next" vers un autre "Node" qui, lui, a une valeur "B". A son tour, il refait une référence vers le "Node" avec comme valeur "A".

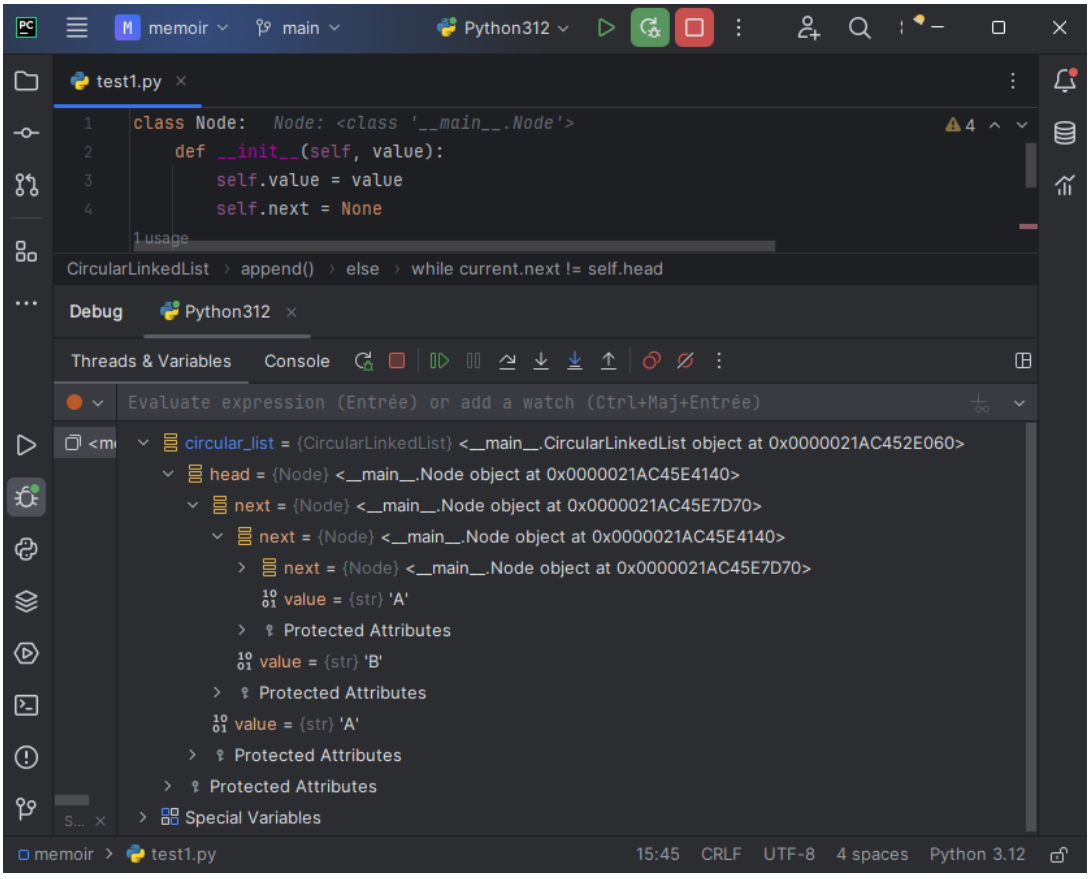


FIGURE 2.3 – L'outil de visualisation des structures de données de PyCharm avec une liste cyclique de deux éléments.

Ces différents outils de visualisation déjà existants, permettent avec leur propre caractéristique de répondre à un besoin différent. Nous allons décrire dans la section "Différences avec les autres outils" 3.2 en quoi ce qu'ils font ne répond pas aux besoins que nous essayons de répondre et quelles sont les avantages de ce que nous produisons par rapport au outils déjà existant.

Chapitre 3

Un nouvel outil de visualisation

Maintenant que le contexte théorique a été placé, nous allons présenter plus en détail la problématique à laquelle nous avons voulu répondre avec notre outil. L'idée de notre mémoire vient du constat, que les outils de visualisation sont très utiles et pratiques pour enseigner et apprendre l'informatique. Nous avons également observé que, en dépit de leur efficacité, les outils existants présentent des limites qui peuvent être défavorables à l'apprentissage des étudiants. En effet, ces contraintes peuvent entraver la progression des étudiants et poser des défis supplémentaires dans leur parcours pédagogique.

Ayant personnellement appris à coder en Python via l'environnement de développement intégré (IDE) Thonny, nous constatons qu'il s'agit d'un outil très épuré, simple d'utilisation et qu'il est en effet très pratique pour enseigner le langage Python à des débutants. Pourtant, cet outil si pédagogique manque d'une fonctionnalité de visualisation de programme (PV). La PV semble être un incontournable pour pouvoir comprendre tant la programmation orientée objet que et tout ce qui en découle, comme les références et les structures de données. Thonny dispose tout de même d'un outil d'inspecteur d'objet qui permet d'aller voir à l'intérieur des objets produits, mais cela n'a rien de visuel et n'aide pas à avoir une vue d'ensemble ou une vision des liens entre les différentes structures de données produites.

3.1 Différences avec PythonTutor

L'outil qui réalise la PV la plus proche de ce que nous essayons de produire est PythonTutor [58]. La question qu'il est dès lors légitime de se poser est la suivante : en quoi notre contribution complète-t-elle ce que cet outil propose déjà ?

Tout d'abord, PythonTutor fonctionne comme un élément externe à un IDE. Pour pouvoir observer la PV de n'importe quel élément de son code, un utilisateur se voit obligé de devoir copier de son IDE à PythonTutor ce qu'il veut visualiser, en tirer des conclusions, trouver les comportements imprévus - si il y en a -, retourner ensuite sur son IDE et ainsi de suite. Devoir utiliser plusieurs outils pour coder un même programme n'est donc pas efficace : c'est une perte de temps, d'énergie et de concentration. Or, il va sans dire que les débutants ont grandement besoin de concentration. De plus, vu que Thonny n'a pas de PV, l'idée originale de ce travail est de produire une PV à la manière de PythonTutor mais à l'intérieur même de Thonny et qui fonctionne en lien avec le débogueur de Thonny. De cette manière, le parcours de l'exécution du programme grâce au débo-

gueur produira en même temps une PV en accord avec l'état de l'exécution à ce moment-là.

Un autre problème majeur de PythonTutor que nous nous efforçons de résoudre, et qui découle de l'idée initiale, est la nécessité de disposer d'un outil fonctionnant hors ligne. Cette solution permettrait aux utilisateurs d'accéder aux fonctionnalités pédagogiques de PythonTutor sans dépendre d'une connexion internet, offrant ainsi plus de flexibilité, même dans des environnements où l'accès à internet est limité ou inexistant. La nécessité de l'accès à une connexion internet est une contrainte qui n'a pas lieu d'être pour de la PV ; il ne devrait pas y avoir besoin d'une connexion avec un serveur ou d'autres utilisateurs.

Les limites que PythonTutor s'impose, avec comme excuse d'être un produit pour débutants, nous semble trop restrictives, et peuvent, dans de très nombreux cas, limiter la progression des utilisateurs. Tout code produisant plus de 2MB de données ne peut pas être visualisé par cet outil. Certes, cela correspond à une liste chaînée de 80 éléments. Afficher une liste aussi grande peut paraître excessif pour des débutants mais il est important de noter que, pour des structures plus complexes, cela équivaut aussi à un arbre binaire d'une profondeurs maximum de 5, soit un arbre avec seulement 30 nœuds. Ce chiffre est atteint sans qu'aucune autre variable ne soit créée, qu'aucune autre fonction ne soit utilisée, ni aucun autre élément supplémentaire. Et cela ne vaut que pour un arbre binaire qui enregistre juste un nombre entier par nœud. Dans le cas de structures plus complexes - des nœuds qui gardent plus de données ou simplement un code qui a besoin de plusieurs structures de données - l'ensemble de ce qui pourra être visualisé sera encore réduit. Aussi, tout code qui a besoin d'exécuter plus de 1000 pas ne sera pas exécuté. Il suffit de quelques boucles *for* imbriquées et cela peut arriver rapidement.

En outre, PythonTutor limite la récursion de fonction à maximum 30 appels récursifs, ce qui nous parait assez peu. Un simple algorithme de tri d'un tableau de 100 éléments, s'il est récursif, empêche toute visualisation. PythonTutor limite aussi les codes trop longs ou qui prennent plus de 10 secondes à s'exécuter.

Toutes ces restrictions sont placées en prétextant qu'afficher trop d'informations en un fois va perturber la concentration de l'utilisateur. Les créateurs de PythonTutor prétendent également que, si l'utilisateur est un débutant, il ne devrait de toute façon pas atteindre les limites exprimées ci-dessus. Si le code de l'utilisateur est trop long et qu'il désire visualiser une de ses structures de données, il doit l'isoler du reste de son code pour pouvoir la voir à part.

Nous ne sommes pas d'accord avec ces justifications. Afficher trop d'informations en même temps peut en effet nuire à la visualisation et être contre productif, mais cela ne signifie pas qu'il faut empêcher celle-ci. Nous pensons qu'il est tout à fait envisageable d'offrir une PV à un utilisateur qui utilise un code long produisant beaucoup de données sans qu'il se sente perdu dans la quantité d'informations présentées. Cela se fait grâce à une visualisation interactive. Celle-ci offre la possibilité à l'utilisateur de choisir quel objet il souhaite voir ou quelle structure il veut étendre, au lieu de tout étendre et tout afficher automatiquement. Si l'utilisateur a, par exemple, un très grand graphe pour lequel il ne désire étendre que certains nœuds bien spécifiques, cela ne surchargera pas la visualisation.

Par ailleurs, en plus de ces limites de taille, PythonTutor s'impose aussi des limites

de fonctionnalité. Il ne supporte pas l'utilisation de la majorité des librairies externes. Il ne supporte pas non plus l'utilisation de fichiers externes comme des bases de données, des fichiers ou le réseau internet. En outre, il ne supporte pas l'utilisation de tout ce qu'il définit comme étant des "Advanced language features or subtleties that only experts need to know about" (Fonctionnalités ou subtilités linguistiques avancées que seuls les experts doivent connaître [notre traduction]) [58]. Encore une fois, selon les créateurs de PythonTutor, ce choix est motivé par le fait que l'outil est destiné aux débutants ; arguant qu'ils ne devraient pas avoir besoin d'utiliser ces éléments. Ajoutons à cela qu'il s'agit d'éléments externes, il n'est donc pas possible à PythonTutor de prévoir ce à quoi ils ressemblent et il est difficile de les visualiser de manière optimale. Compte tenu de ces deux raisons, il vaut mieux ne pas essayer de les visualiser. Il convient également de considérer cette contrainte : il peut être plus compliqué de le faire étant donné qu'il s'agit d'un service en ligne.

Le problème que pose l'outil en étant uniquement disponible en ligne se résout de lui-même dans notre travail. Quand à l'argument que ces éléments sont trop complexes pour être utilisés par des débutants, cela nous parait incorrect. Bien qu'il y ait beaucoup de librairies complexes à utiliser, il y en a également de très simples. Il n'est certes pas conseillé de commencer à apprendre à coder en utilisant directement des librairies ou en faisant appel à des fichiers externes. Cependant, les outils de visualisation sont utiles tout au long de l'apprentissage. L'individu qui commence à utiliser les librairies peut encore trouver la visualisation utile. Notons aussi que les visualisations ne sont pas uniquement utiles pour apprendre, mais aussi pour déboguer - ce qui est utile aussi pour un expert. Nous en voulons pour preuve la vue hiérarchique que propose PyCharm ; celui-ci n'est pas un IDE pour débutants. Avoir une visualisation d'un code qui utilise un élément externe (comme une librairie ou une base de données) peut se révéler être très intéressant.

En revanche, la question de la nécessité de visualiser précisément l'élément externe utilisé mérite une réflexion distincte. Nous pensons qu'il n'est pas correct de "pénaliser" tout un code à cause de l'utilisation à un endroit donné d'un élément plus complexe. Cependant, il est vrai que visualiser ces éléments n'est pas évident car ils ont, bien souvent, énormément de variables ou de méthodes internes. Or, beaucoup d'entre elles sont très peux intéressantes pour un débutant, voire même pour quelqu'un de déjà formé. Il y a même certains types de données pour lesquelles une visualisation graphique intéressante et compréhensible n'est pas vraiment possible. Dans un premier temps, nous avions décidé de permettre à l'utilisateur d'aller voir dans les variables de ces éléments externes. En effet, nous pensions que les débutants qui ne sont pas intéressés par ces données plus complexes peuvent simplement ne pas y regarder. Suite aux retours que nous avons eus, nous avons décidé d'offrir à l'utilisateur deux modes. D'une part, un mode pour débutants où aucun contenu de ces éléments externes n'est affiché afin de ne pas distraire avec des données qui, a priori ne les concernent pas. D'autre part, un mode où l'utilisateur peux observer les variables de ces éléments externes. Ces quelques changements sont explicités dans la section 5.3.

PythonTutor déclare encore avoir quelques limites, comme par exemple la programmation concurrente ou l'écriture dans la sortie d'erreurs ("stderr"). Cela nous semble être la conséquence logique du refus de l'importation de librairies. En tous cas, nous ne voyons pas comment faire sans librairies. De toutes façons, puisque nous n'avons pas cette

restriction, il n'y a pas de raison que l'on ne puisse pas utiliser du code asynchrone ou écrire sur le flux d'erreur standard.

Nous pouvons mentionner ici qu'avoir la visualisation dans un IDE tel que Thonny plutôt que l'outil PythonTutor permet de pouvoir utiliser des arguments de ligne de commande (par exemple "argv[]). Cela permet aussi de parcourir une ligne de code pour montrer comment les sous-expressions sont évaluées dans cette ligne car le débogueur de Thonny peut le faire. Cela ne concerne cependant pas notre travail.

Pour finir, il nous a paru utile d'implémenter une autre fonctionnalité dont PythonTutor ne dispose pas. Il s'agit de proposer un graphique interactif. Comme décrit ci-dessus, l'utilisateur peut décider quel objet afficher. Étant donné que notre visualisation peut avoir beaucoup plus d'objets affichés, nous voulons que l'utilisateur puisse déplacer lui-même l'endroit où chaque noeud individuel s'affiche. Cela permettrait à l'utilisateur d'organiser son graphique comme il l'entend : regrouper les objets, les placer comme il le souhaite, les mettre en évidence à son gré. Grâce à cela, l'utilisateur ne se perdra pas dans tous les noeuds qui pourraient être affichés en même temps.

La seule limite de PythonTutor à laquelle nous n'avons, en partie, pas touché, est celle des "Fonctionnalités ou subtilités linguistiques avancées que seuls les experts doivent connaître". En effet, nous avons remarqué que certaines fonctions ne sont pas représentables. L'outil intégré de Thonny "inspecteur d'objet" retourne une erreur lorsqu'il essaye de représenter ces fonctions. Ne voyant pas comment faire mieux que Thonny et reconnaissant que ce n'est pas d'une grande nécessité, nous n'avons pas essayé de corriger cette source d'erreurs. Les fonctions avancées que nous ne gérons pas sont, par exemple, toutes les fonctions qui vont regarder ce qu'il y a dans la mémoire sur la pile et la changer. Lorsqu'elles sont appelées, ces fonctions vont changer l'état de la mémoire. Si cela se déroule durant le parcours de l'exécution avec le débogueur, cela va générer automatiquement un message d'erreur. Il en va ainsi de la fonction qui vient de la librairie "threading".

```
1 import threading
2
3 def Hello_World():
4     print("Hello World")
5
6 thread1 = threading.Thread(target=Hello_World)
7 thread1.start()
8
9 thread1._stderr._backend._heap
```

Listing 3.1 – Exemple de Code Python

Ces fonctions sont très certainement seulement destinées aux experts vu que les attributs comme ._stderr, ._backend, et ._heap ne sont pas des attributs standards de threading, mais peuvent apparaître dans certains contextes ou implémentions spécifiques, souvent utilisés pour la gestion interne ou pour des bibliothèques tierces. Malgré tout, ces attributs peuvent être vus dans l'inspecteur d'objet de Thonny. Cependant analyser l'attribut ._heap va renvoyer un message d'erreur et empêcher de continuer l'analyse du code.

Il y a aussi, par exemple, les *frozenset* qui sont les versions immuables du type *set*. C'est un type de données très spécifique ; il n'est pas particulièrement intéressant pour

des débutants. Notre outil de visualisation va, tout comme PythonTutor, les afficher mais il ne va pas visualiser les objets qui pouraient être à l'intérieur, comme le montre l'image 3.1. Dans notre travail, nous allons utiliser une autre manière plus simple de nommer les classes mais l'idée reste la même. Même l'inspecteur d'objet de Thonny ne permet pas de visualiser les données à l'intérieur d'un *frozenset*.

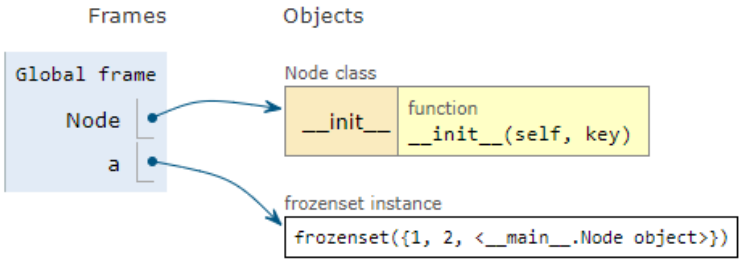


FIGURE 3.1 – Représentation d'un frozenset dans PythonTutor

3.2 Différences avec les autres outils

Maintenant que ce que nous voulons faire est clarifié, il est aussi intéressant de voir en quoi notre outil est différent des autres outils de visualisation existants. L'objectif est avant tout d'avoir une visualisation pour les néophytes et qui soit intégrée à un IDE pour débutants en Python. Cela implique que nous cherchons à avoir quelque chose d'assez épuré, clair et intuitif puisque notre IDE présente ces caractéristiques.

Nous apportons dès lors quelque chose de plus que tout les outils tels que JaguarCode, Javalina Code, Jeliot3, VILLE, Jeroo et Alice. Ceux-ci ne sont pas des IDE ou ne sont pas liés à un IDE. Nous apportons un outil qui aide à coder en Python, ce qui n'est pas le cas de JIVE, BlueJ et PandionJ qui sont tous les 3 spécifiques à du code Java.

Ceci étant dit, il nous reste, parmi les outils de visualisation que nous avons trouvés, jGRASP et PyCharm. Le premier est un IDE pour débutants qui est plus spécialisé en Java qu'en Python, ce qui lui donne des fonctionnalités dont un débutant en Python n'a pas besoin et qui risquent de semer la confusion. Cela reste un désavantage minime. Ce qui différencie cette IDE avec ce que nous faisons, c'est le type de visualisation. En effet, comme le présentent ces vidéos de tutoriel réalisées par jGRASP [72] et [73], la visualisation qu'il offre se rapproche plus de la visualisation de code et de la visualisation d'algorithme que de programme. Pour rappel, la visualisation de code met en évidence la structure du code écrit, en montrant par exemple plus clairement les boucles "for" et "while". Cela n'est pas lié à l'exécution du code - et donc à notre travail. La visualisation d'algorithme se focalise sur l'algorithme. Elle se dissocie du langage et donc également de la structure de données. Cette approche est moins efficace pour comprendre la programmation orientée objet (OOP). Par ailleurs, bien qu'elle puisse être utile pour des structures de données courantes, elle peut rencontrer des difficultés avec des structures personnalisées. Or, notre objectif est justement d'aider les débutants à comprendre la OOP et, avant tout, à comprendre leur propre structure de variables.

Il ne nous reste plus qu'à discuter de PyCharm. C'est un IDE spécifique au langage Python reconnu pour sa complétude et sa richesse en fonctionnalités. Cependant, cette abondance d'outils le rend peu optimal pour les utilisateurs débutants en programmation. En effet, son interface n'est pas épurée, même lorsqu'aucune extension n'est ajoutée. Cela peut grandement perturber et déconcentrer un débutant. En dépit de cela, PyCharm demeure un outil très intéressant et utile, notamment grâce à sa fonctionnalité de PV qui nous a inspirés pour une autre partie de notre projet.

PyCharm a une PV sous forme d'une vue hiérarchique. Cette vue montre les variables qui ont été créées, quels sont les objets qui ont été créés et quelles sont leurs variables d'instance. Certes, elle montre les mêmes information que Pythontutor mais de manière beaucoup moins visuelle, moins graphique et de façon unidimensionnelle. En outre, il n'y a pas de formes géométriques ou de flèches. Les boucles dans une structure de données sont moins évidentes. Par exemple, cette vue ne permet pas facilement de voir si une liste chaînée est cyclique ou non. Il est aussi moins évident d'identifier une structure de données parce qu'il faut la lire et la déduire ; cela n'est pas visible en un coup d'oeil. Cette vue est donc moins pédagogique et elle permet moins de comprendre la OOP. Par contre, elle permet de déboguer plus rapidement pour quelqu'un qui connaît déjà bien son code. Cette vue hiérarchique est beaucoup plus compacte et il est plus facile d'arriver rapidement à l'élément que l'utilisateur veut inspecter. C'est donc une vue qui est moyennement intéressante pour les débutants mais l'est un peu plus pour les utilisateurs expérimentés.

Cette vue nécessite de travailler sur la partie des données stockées en mémoire et celle de la connexion avec le débogueur, sans trop travailler sur la partie graphique. Nous avons pensé qu'il serait intéressant de créer deux visualisations de programme en commençant par créer une vue hiérarchique. Une fois cela fait et que nous avons compris comment faire, nous pouvons nous tourner vers une visualisation plus graphique, qui équivaut à une vue hiérarchique plus complexe. Cela permet d'offrir à la fois une vue pour débutants et une vue pour utilisateurs un peu plus expérimentés qui ont besoin de quelque chose de moins pédagogique mais de plus efficace. De plus, Thonny permet d'activer ces fonctionnalités dans l'onglet "View" (ou "Affichage" en français). Créer une ou deux vues avec une seule extension ne rend pas l'affichage par défaut plus chargé et n'impacte donc pas la concentration des débutants.

Il faut maintenant se demander ce que notre vue hiérarchique apporte de plus que celle de PyCharm. Tout d'abord, elle est sur Thonny qui n'avait pas ce type de vue avant. Cela permet de faire la transition entre un utilisateur généralement débutant de Thonny qui, en devenant plus expérimenté, pourrait aller sur PyCharm. Cette vue lui permettrait de se familiariser avec ce qui l'attendra.

Ensuite, comme montré dans la figure 2.3 dans l'état de l'art, l'adresse mémoire d'un objet permet de l'identifier. Par exemple, pour identifier une liste cyclique, il faut remarquer que le premier élément (un objet "Node") avec la valeur "A" possède la même adresse mémoire qu'un autre élément (un autre "Node") avec la valeur "A". Nous pensons que c'est très peu pratique pour reconnaître les cycles dans les structures de données et que, de toutes façons, l'adresse mémoire n'a pas besoin d'être visualisée car elle n'est pas utile pour un débutant. La manière dont nous avons décidé de rendre cela plus lisible et intéressant pour notre utilisateur sera discutée dans la section 4.3.

Chapitre 4

Première version de notre outil

Dans cette section, nous allons présenter les différentes étapes que nous avons réalisées pour atteindre notre première solution. Nous présenterons tout d'abord l'aspect structurel de Thonny avec lequel nous nous sommes accordés pour offrir un outil qui s'y intègre le mieux possible. Ensuite, nous discuterons de la première vue additionnelle que nous avons ajoutée : la vue des variables globales et locales. Par après, nous parlerons de la vue hiérarchique qui est la première forme de visualisation de programme (PV) de notre outil. Enfin, nous aborderons la réalisation de la vue graphique qui est celle qui s'apparente le plus à un outil de PV comme on l'entend communément.

4.1 Structure de Thonny

Afin que notre outil s'intègre le plus harmonieusement possible avec Thonny, nous avons commencé par analyser sa structure. Ainsi tant sur l'arrière que sur l'avant-plan, nous avons tenu à conserver le même système de fonctionnement que Thonny.

4.1.1 Avant-plan

Notre premier critère concernant l'apparence de notre extension dans Thonny est de tout afficher dans la fenêtre principale de ce dernier. En effet, bien qu'il aurait été possible d'afficher nos différentes vues dans des onglets détachés, les laisser dans la fenêtre principale permet à l'utilisateur d'interagir directement avec elles tout en programmant. Ce choix nous permet également d'assurer une visualisation dynamique et interactive avec l'outil de débogage déjà présent.

L'apparence de Thonny et, de façon générale, toute son interface avec l'utilisateur sont réalisées à l'aide de la bibliothèque standard Tkinter. Nous avons donc décidé de conserver ce choix et de réaliser nos propres interfaces à l'aide de structures issues de cette librairie. Les différentes structures que nous avons utilisées sont présentées dans les points suivants qui parlent des différentes vues composant notre travail.

Afin de nous intégrer de la meilleure façon possible à la structure de Thonny, nous avons fait en sorte que nos différentes vues soient sélectionnables dans l'onglet "Views" de l'environnement intégré. Les onglets s'ouvrant alors s'affichent aux endroits qui nous semblaient les plus appropriés : la vue des variables globales et locales dans la même

fenêtre que la vue native "Variables" et les vues hiérarchiques et graphiques dans la fenêtre principale en vis-à-vis du code (voir figure 4.1).

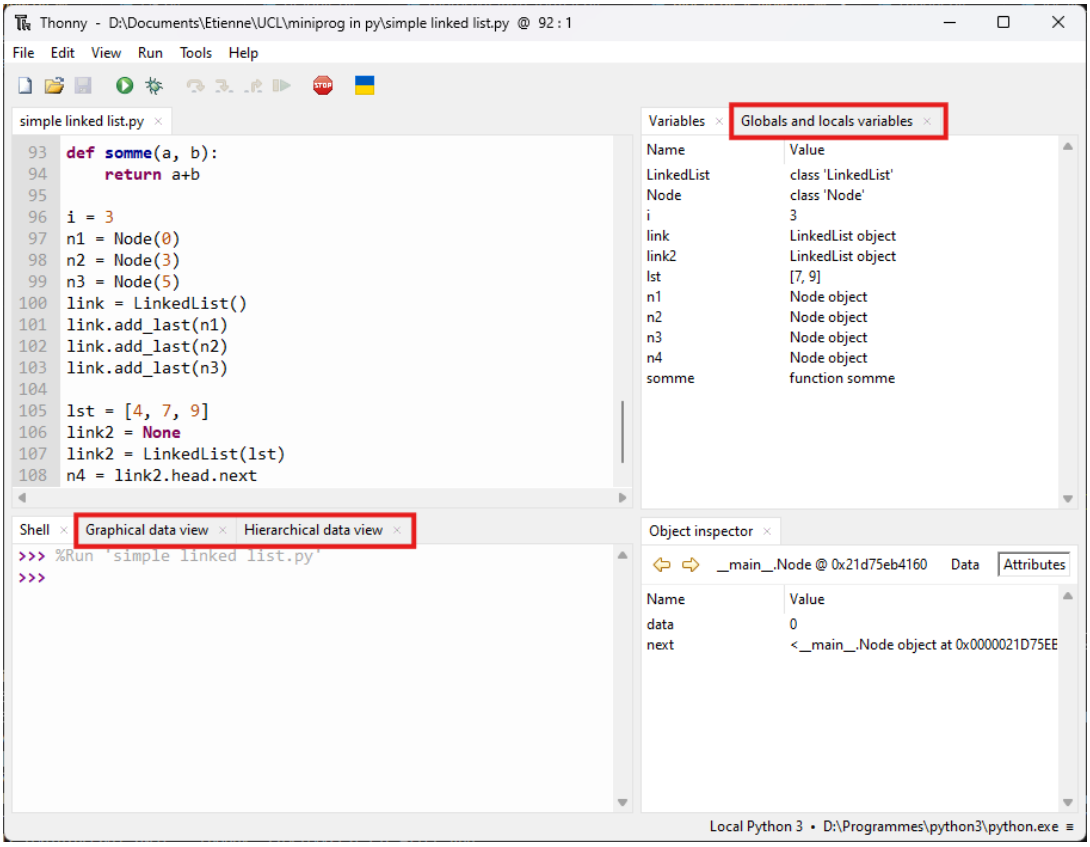


FIGURE 4.1 – Positions des différentes vues

4.1.2 Arrière-plan

Thonny a une façon précise de gérer les extensions qui lui sont liées. En effet, il possède un espace dédié - nommé "Thonnycontrib" - où sont placés tous les fichiers constituant les extensions. Au sein de cet espace, tout fichier comprenant une fonction "load_plugin" permet de charger les extensions correspondantes. Globalement, il est possible de réaliser à peu près tout ce que l'on souhaite à l'aide d'extensions. En effet, à l'aide des bons appels, il est possible d'aller chercher toutes les informations existantes dans Thonny et de les modifier.

Dans notre cas, nous interagissons principalement avec l'arrière-plan en envoyant des requêtes à la mémoire afin d'obtenir des informations sur les différents objets et variables composant le programme codé par l'utilisateur. Lorsque nous recevons ces informations, nous les traitons pour les afficher correctement et les rendre visuellement utiles. Le traitement précis de ces informations est discuté plus en profondeur dans les sections suivantes en fonction de chaque vue.

4.2 Vue des variables globales et locales

La première vue que nous avons implémentée dans notre travail est celle qui permet d'afficher les différentes variables, globales et locales, au moment précis de l'étape d'exécution du débogueur (ou après l'exécution complète du programme).

4.2.1 Fonctionnalités

Notre vue est fortement inspirée de la vue "Variables" déjà présente dans Thonny. Cette dernière affiche dans une fenêtre la liste des variables globales créées par l'utilisateur. Elle leur associe également une représentation qui peut contenir, dans le cas d'un objet plus complexe, une adresse mémoire.

Nous avons conservé cet affichage dans notre vue mais nous trouvions qu'il manquait la possibilité de visualiser les variables locales. En effet, Thonny a l'avantage de proposer un excellent débogueur qui permet de parcourir, étape après étape, l'exécution du programme codé par l'utilisateur. Cet outil offre une prise en main intuitive et une visualisation limpide de l'exécution du programme. C'est sans doute l'outil le plus facile à prendre en main pour les débutants. Ainsi, en liant notre vue au débogueur, nous offrons à l'utilisateur de pouvoir visualiser non seulement les variables globales existantes au moment de l'étape du débogage mais également les variables locales mises en lumière durant cette étape.

4.2.2 Avant-plan

Afin d'être le plus cohérents possible avec la vue "Variables" existante, nous avons placé la nôtre dans la même section de la fenêtre principale. Ainsi, notre vue présentant à la fois les variables locales et globales peut être sélectionnée pour remplacer la vue affichant simplement les variables globales.

Nous avons également conservé l'apparence et la structure de la vue native en affichant de façon identique les variables globales sous une section nommée "Globals". Nous avons rajouté la section "Locals" permettant l'affichage des variables locales de la même façon que celles globales. Le tout étant réalisé avec la librairie Tkinter, l'esthétique générale de notre vue reste cohérente avec l'apparence globale de Thonny (voir figure 4.2).

4.2.3 Arrière-plan

Pour afficher les différentes informations nécessaires dans notre vue des variables locales et globales, nous les récupérons à partir de l'étape d'exécution à laquelle nous sommes (ou de l'exécution complète). Celle-ci est partagée dans une structure appelée "event" qui contient la liste des variables globales et locales (s'il y en a) à cette étape. Cette liste associe à chaque identifiant de variable locale ou globale, sa représentation en chaîne de caractères ainsi que son adresse mémoire. Lorsque nous avons ces informations, il nous suffit de les afficher de la façon la plus claire et cohérente avec la vue "Variables" existant déjà dans Thonny.

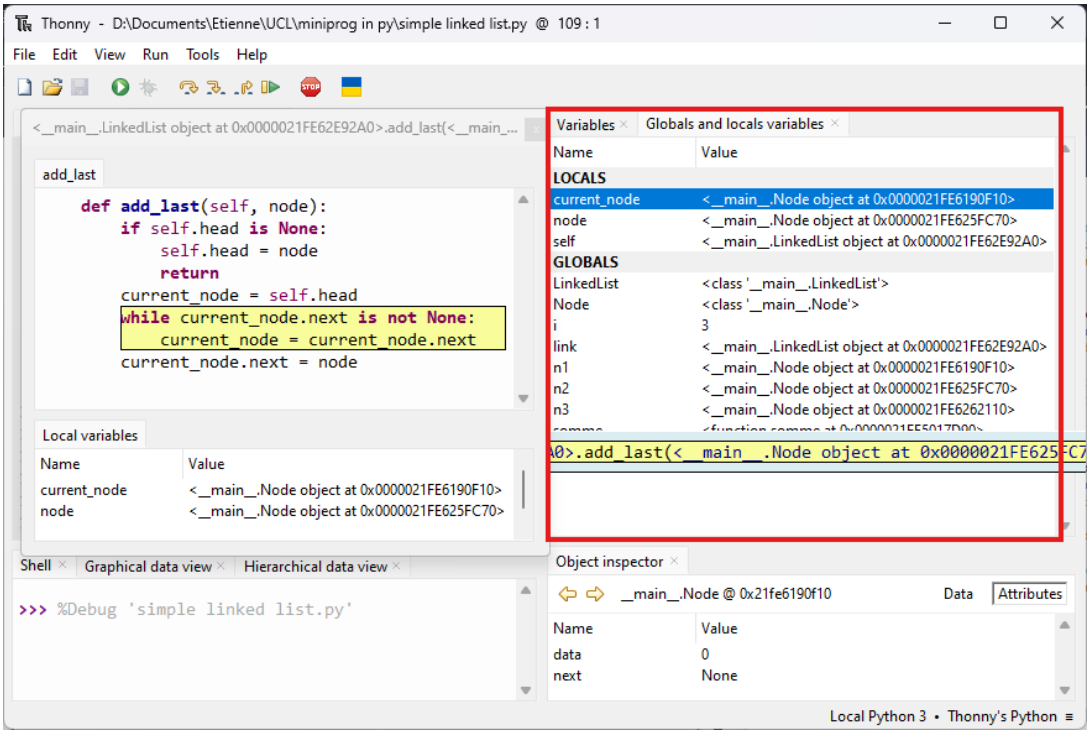


FIGURE 4.2 – Première version de la vue des variables globales et locales

4.3 Vue hiérarchique

Le premier outil de visualisation que nous proposons est une vue hiérarchique des différents objets et variables présents dans le code de l'utilisateur. Cette vue permet de visualiser textuellement la composition de ces différents objets et le lien hiérarchique existant entre eux. Avec cette vue, il devient possible de développer les objets complexes afin de visualiser leurs attributs et d'afficher si ceux-ci sont liés à des objets existants.

4.3.1 Fonctionnalités

Notre vue hiérarchique est déjà un outil de PV à part entière. L'objectif principal de cette vue est d'aider l'utilisateur à déboguer son code, en lui permettant de suivre et d'observer tout les objets créés dans le programme codé par l'utilisateur. Chaque variable locale ou globale est affichée dans sa section correspondante, identifiée par son nom dans le programme de l'utilisateur et par un identifiant simplifié qui est simplement sa représentation en chaîne de caractères modifiée dans le cas d'objets complexes.

Le choix d'utiliser la représentation d'un objet ou d'une variable pour l'identifier est fait dans les différents outils de Thonny. En effet, la vue "Variables" identifie une variable avec sa représentation. Soucieux de conserver cette cohérence, nous avons gardé ce système de représentation. Ainsi, pour identifier un objet créé par l'utilisateur et qui serait un objet d'une classe appelée "Node", sa représentation en Python se fera automatiquement sous la forme "<___main___.Node object at memory_address>" où "memory_adress" correspond à l'adresse mémoire réelle attribuée par le programme. Nous avons choisi de simplifier cette représentation en la mettant sous la forme "Node object n°x" où x correspond à un nombre que nous attribuons, afin de différencier les différents objets du même type.

Nous ne faisons cela que pour les variables explicites. Si un objet est créé dans une autre variable, alors il sera identifié comme (en reprenant l'exemple ci-dessus) "Node object unnamed".

En plus d'afficher les différentes variables globales et locales de façon simplifiée, notre vue permet de les détailler afin d'en observer les attributs. Ainsi, à l'aide de petits boutons "+" et "-", l'utilisateur peut parcourir les différents objets et variables de son programme et observer leurs attributs. Le parcours de ces objets et variables se fait de façon paresseuse (ou lazy en anglais). Ainsi, tant que l'utilisateur n'a pas décidé d'en observer les attributs, nous ne calculons pas ce qu'il va falloir représenter. Cela permet d'éviter de tourner à l'infini dans le cas où les objets à développer possèdent des références qui bouclent (comme des listes doublement chaînées).

Si certains attributs correspondent à des variables ou objets qui ont déjà été rencontrés lors de l'exploration de l'utilisateur, cela sera indiqué dans la représentation de ces attributs. En effet, à la fin de la représentation de cet attribut, une parenthèse contiendra le nom de la variable ou l'objet associé. Si ce dernier a été observé pour la première fois comme attribut d'un autre objet, alors l'intérieur de la parenthèse sera de la forme "nom_de_l'objet.nom_de_l'attribut".

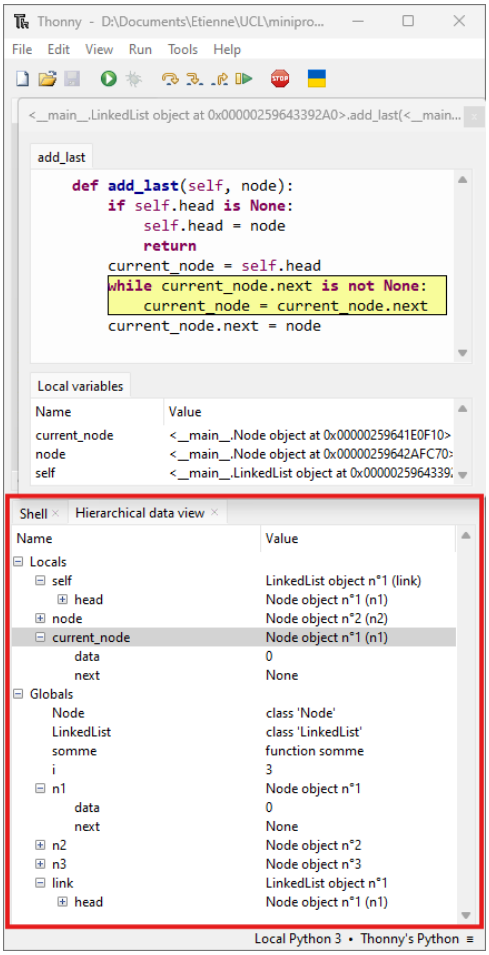


FIGURE 4.3 – Première version de la vue hiérarchique

4.3.2 Avant-plan

L'apparence de notre vue est fortement inspirée de la vue "Arbre du programme", présente dans Thonny. Dans notre souci de garder la cohérence et l'homogénéité avec l'apparence générale que propose cet environnement de développement, nous trouvions l'apparence de la vue "Programme tree" particulièrement adaptée à notre vue hiérarchique. Nous avons ainsi utilisé la même classe de Tkinter : "TreeFrame". Celle-ci nous permet de gérer facilement la présence de boutons "+" et "-" pour développer ou réduire les attributs de chaque variable et objet.

Pour son emplacement, nous avons choisi de mettre notre vue dans la fenêtre de retour principale, celle où sont affichées les sorties et où apparaît également la vue "Arbre du programme". Les raisons de ce choix sont simples : c'est la fenêtre proposant de base le plus de place, sans compter la fenêtre contenant le code. Or, elles sont justement en vis-à-vis ce qui facilite l'utilisation lors du débogage du programme (voir figure 4.3).

Pour ce qui est de la représentation plus précise des objets et variables, nous avons déjà discuté des références à la mémoire dans la section précédente. Nous avons également réalisé un choix particulier dans la représentation des dictionnaires. En effet, pour correspondre au mieux à l'encodage de ces structures en Python, nous avons décidé de les représenter en mettant en évidence des index numérotés auxquels sont rattachées une clé et une valeur. Ainsi, les paires clés-valeurs sont mises en évidence par une indentation créée par les index fictifs que nous attribuons.

4.3.3 Arrière-plan

Les différentes informations dont nous avons besoin pour l'affichage de la vue hiérarchique sont bien plus complexes à obtenir que celles pour la vue des variables locales et globales. En effet, la vue hiérarchique doit pouvoir afficher tous les attributs de chaque variable et objet et les informations de ces attributs voire, s'ils en possèdent, leurs propres attributs.

Pour aller chercher ces informations, le départ reste identique à la vue des variables : nous récupérons les informations de l'état d'exécution du programme et nous en extrayons celles qui nous intéressent sur les variables et objets locaux et globaux. Parmi ces informations, les identifiants permettant de retrouver les objets ou variables dans la mémoire et leur nom attribué par l'utilisateur dans son code vont nous permettre de récupérer le reste des informations nécessaires.

Thonny possède un système de requête à la mémoire bien spécifique. Chaque requête envoyée doit être d'un certain type. Dans notre cas, nous faisons une requête de type "get_object_info". Ce type de requête demande à la mémoire tous les détails qu'elle possède sur un certain objet spécifié par son identifiant unique. Après que la requête soit envoyée, notre programme va se mettre sur écoute pour réagir lorsque la mémoire renverra un évènement de type "object_info".

Lorsque notre outil reçoit un évènement de la mémoire signalant qu'elle souhaite transmettre des informations sur un objet, il nous faut vérifier si celui-ci est bien l'objet dont nous avons demandé les détails. De fait, d'autres fonctionnalités de Thonny utilisent également ce genre de requête. Si les informations sont bien celles que nous recherchions, alors nous avons tout ce qu'il faut pour afficher correctement la variable ou l'objet dans la vue hiérarchique et nous connaissons maintenant les attributs pour lesquels nous pouvons réitérer des requêtes mémoire pour les afficher à leur tour.

Si, par contre, les informations que nous recevons ne sont pas celles que nous voulions, il va nous falloir relancer la requête. En effet, nous avons observé que les requêtes que nous envoyons à la mémoire ne sont stockées et gérées comme dans une file que si elles sont de différents types. Or, si notre vue hiérarchique est ouverte en même temps que l'inspecteur

d'objet de Thonny, il est possible que plusieurs requêtes du même type ("get_object_info") se fassent en même temps. À ce moment-là, la dernière requête est considérée comme redondante et n'est pas prise en compte. Pour cette raison, nous sommes contraints de relancer notre requête si les informations que nous avons envoyées ne correspondent pas à ce que nous attendions.

4.4 Vue graphique

Cette vue à un objectif légèrement moins efficace que la vue hiérarchique, car la vue graphique peut perdre en lisibilité lorsqu'il y a beaucoup de variables affichées face à la vue hiérarchique. En contrepartie, elle gagne beaucoup plus en pédagogie et aide véritablement à comprendre la programmation orientée objet (OOP) est les structures de données.

Notre vue graphique est ce qui se rapproche le plus d'un outil de PV dans le sens le plus communément admis du terme. En effet, chaque type de collections - à savoir les listes, les dictionnaires, les tuples et les ensembles - ainsi que chaque objet créé par l'utilisateur au sein de son programme seront représentés par une boîte rectangulaire et les liens entre ceux-ci seront indiqués sous forme de flèches. Il y aura aussi une boîte rectangulaire "Globals" qui comprend toutes les variables globales stockées en mémoire ainsi qu'une boîte "Locals" qui comprend toutes les variables locales stockées en mémoire de l'étape que l'utilisateur est en train de parcourir.

Cette représentation, inspirée de la visualisation de PythonTutor, bien plus visuelle des objets et de leurs liens, est conçue pour offrir une meilleure compréhension à l'utilisateur du fonctionnement de son programme et plus particulièrement des liens et caractéristiques de ses objets et variables. Cela aide pour la compréhensions de tous les principes qui découlent de la OOP comme, par exemple, l'héritage ou les références. Cette visualisation est donc un diagramme réseau orienté. Par conséquent, durant cette présentation les boîtes seront appelées "nœuds" et les flèches seront appelées "arêtes".

4.4.1 Fonctionnalités

Comme tout outil de PV, la fonctionnalité principale de notre vue graphique est d'afficher les différents objets et variables du programme de façon visuelle afin d'améliorer leur compréhension. L'objectif principal était d'avoir l'affichage qui se réalise après l'exécution du code mais aussi durant le parcours de l'exécution grâce au débogueur. Il est essentiel de permettre à l'utilisateur de voir quelle étape de son code crée, par exemple, un objet, le modifie ou change la référence d'une variable vers un autre objet. Cela peut grandement aider l'utilisateur à la compréhension de son code, aux principes de la OOP utilisés dans son code, et cela l'aide aussi à trouver la position de toute erreur éventuelle et donc de la corriger. L'idée est aussi de garder le plus possible le graphique dans le même état qu'à l'étape de parcours de l'exécution précédente. Évidemment, si l'étape a changé quelque chose aux données, ce changement sera représenté. Cependant, toutes les données qui n'ont pas changé devraient être affichées de la même manière qu'avant. En effet, comme expliqué dans les paragraphes qui suivent, les données vont pouvoir être représentées de manière différente selon la volonté de l'utilisateur. Cette idée de garder en mémoire l'état du graphique durant le parcours du débogueur est importante car elle permet de garder de la continuité. Elle permet aussi à l'utilisateur de ne pas devoir restructurer l'affichage

du diagramme réseau à chaque étape.

Cette vue peut être rapidement illisible si trop de nœuds sont affichés, ce qui doit être la raison pour laquelle PythonTutor limite le nombre de données qu'il affiche. Nous avons tout de même préféré ne pas limiter le nombre de données affichées mais nous avons décidé en contrepartie de mettre plusieurs fonctionnalités en place pour aider l'utilisateur à ne pas se perdre dans le nombre de données qui peuvent être affichées.

La première grosse fonctionnalité que nous avons voulu nous imposer dans ce sens-là, c'est l'interactivité. En effet l'utilisateur peut lui-même déplacer les nœuds grâce à un "glisser-déposer" (ou drag-and-drop en anglais) et les positionner où il le veut. Cette interactivité est très importante : cela rend l'outil plus attractif et aide à la motivation et à la concentration des débutants en informatique. Surtout, cela leur permet de disposer le diagramme réseau comme ils l'entendent. Ils peuvent par exemple rassembler plusieurs nœuds, mettre des nœuds à part ou afficher des structures de données avec une disposition qui leur semble plus claire.

Ensuite, nous avons l'extension paresseuse (ou lazy en anglais) des noeuds. Notre programme va automatiquement afficher le nœud "Globals" et le nœud "Locals" s'ils contiennent des variables mais les autres nœuds ne vont s'afficher que si l'utilisateur le demande. A chaque variable qui est une référence vers un type d'ensemble ou un objet, il y aura non pas une valeur mais une petite pastille de couleur rouge sur laquelle l'utilisateur peut cliquer pour afficher le nouveau nœud (pour une compréhension plus claire dans ce rapport, cela s'appellera "ouvrir" une référence). Suite à cela la pastille va se colorer en vert, et inversement pour cacher un nœud qui a été affiché précédemment (cela s'appellera "fermer" une référence).

Le fait que cela soit un affichage paresseux (ou lazy) était au début une solution technique ; nous en discuterons dans la section "Arrière plan" 4.4.3. Cela permet aussi à l'utilisateur de n'ouvrir que la structure de données spécifiques qu'il veut voir et d'avoir un diagramme réseau qui n'est pas surchargé de nœuds, sauf si c'est sa propre volonté. L'objectif est vraiment de donner davantage le choix à l'utilisateur et de le rendre plus libre que PythonTutor ne le fait.

Toujours dans un souci de visualisation, nous avons réalisé deux états pour chaque nœud : l'état "extend" et l'état "reduce", en français : "étendu" et "réduit". L'état étendu correspond à un noeud dont toutes les variables qui lui sont rattachées sont visibles en plus du nom du nœud. Par "variable qui lui sont rattachées", nous entendons pour les nœuds "Globals" et "Locals" respectivement toutes les variables globales et locales ; pour tous les nœuds qui sont des instances d'objets, leur variable de classe et leur variable d'instance ; pour tous les nœuds qui sont la représentation de classe, leur variable de classe et leur méthode.

L'état réduit, quant à lui, correspond à un nœud dont seul le titre est visible. Ces deux états permettent à l'utilisateur de réduire la taille des nœuds afin qu'il prennent moins de place. Ainsi le graphique sera plus lisible, tout en ayant quand même accès à toutes les données intéressantes de chaque nœud, grâce à l'état "étendu". Il est possible de passer d'un état à l'autre, individuellement pour chaque nœud, en cliquant sur le bouton

"-" pour passer à l'état réduit et sur le bouton "+" pour passer à l'état étendu, qui se trouvent sur chaque boîte. Dans l'état réduit il y a aussi la possibilité pour un nœud d'ouvrir ou de fermer en un seul clic toutes les références à tous les nœuds enfants dudit nœud. Ce choix d'implémentation était assez évident à faire : devoir repasser à chaque fois dans l'état étendu pour ouvrir une nouvelle référence enfant rend le développement de l'affichage fastidieux. Cependant, étant donné que sous l'état réduit les références ne sont pas affichées individuellement et donc ne peuvent pas être sélectionnées individuellement, ouvrir une seule référence à un seul nœud enfant revient à sélectionner l'affichage de tous les nœuds enfants. Il en va de même pour le fait de vouloir cacher un seul nœud enfant. De plus, cela permet à un utilisateur qui voudrait ouvrir, pour une raison ou une autre, toutes les références de son graphique, de le faire plus rapidement.

Il y a également deux modes, "extend", "reduce" qui consistent à appliquer respectivement l'état étendu et l'état réduit à tous les nœuds du graphique. De plus, quand les modes "extend" ou "reduce" sont activés, l'affichage de tout nouveau nœud sera aussi respectivement affiché dans l'état étendu ou l'état réduit. Notons que, toujours dans l'optique de garder l'état des nœuds même quand ils ont été cachés, tous les nœuds - même ceux cachés - vont avoir leur état changé en conséquence au moment où l'utilisateur clique sur le bouton "extend" ou "reduce". Les nouveaux nœuds seront créés avec l'état qui correspond. Cependant, quel que soit le mode activé en place, quand l'état particulier d'un noeud change, s'il est par la suite caché puis ré-affiché, il gardera son état d'avant avoir été caché.

Ces deux modes ont été créés pour favoriser des actions plus rapides par la réduction ou l'extension de tous les noeuds par un seul clic. Cela évite de parcourir une boîte après l'autre. Pour l'utilisateur, moins de travail rébarbatif lui permet de rester mieux concentré. En outre, ces modes laissent l'utilisateur maître de l'extension ou non des nouveaux noeuds en fonction de son code et de l'élément qu'il souhaite analyser. Notre objectif est de laisser l'utilisateur décider par lui-même. Dans un code avec peu de variables et peu d'objets, il n'y a pas besoin de réduire l'affichage des noeuds et l'utilisateur pourrait préférer avoir tout étendu. Par contre, s'il a beaucoup de variables et d'objets, peut-être qu'il voudra n'en afficher que quelques-uns ; auquel cas il va vouloir que les nouveaux nœuds soient étendus pour choisir précisément quelle référence rendre visible. En résumé, il est question ici des préférences de chacun, raison pour laquelle l'utilisateur doit pouvoir choisir.

Le dernier outil pour aider à la visualisation est "Recenter" qui permet de repositionner les nœuds affichés de manière plus rangée. Cela permet de retrouver de l'ordre, et donc de la lisibilité, si les nœuds ont beaucoup été bougés ou ont été affichés dans un ordre aléatoire, par exemple. Après l'utilisation de notre outil pendant quelque temps, l'utilisateur peut avoir éparpillé les nœuds du graphique et les avoir distancés les uns des autres. Cela permet de tout compacter, avoir les nœuds les uns à côté des autres. Cela prend donc moins de place et permet de retrouver facilement un noeud.

4.4.2 Avant-plan

Dans l'environnement de développement intégré (IDE) de Thonny, nous avons décidé de placer l'avant-plan au même endroit que la vue hiérarchique. Nous avons choisi cet emplacement pour les mêmes raisons que pour la vue hiérarchique : c'est l'endroit avec

plus de place, à l'ouverture de Thonny, et qui se trouve en vis-à-vis avec le code. Par ailleurs, étant donné que la vue hiérarchique et la vue graphique présentent les mêmes éléments mais de manière différente, un utlisateur n'a pas besoin de visualiser les deux vues en même temps.

Comme expliqué précédemment, le graphe de notre vue forme un diagramme réseau orienté. Il est intéressant de regarder comment nous avons dessiné les nœuds et les arrêtes de ce graphe. Cette représentation peux être vue à la figure 4.4.
Les nœuds sont faits sous forme de rectangles avec, à l'intérieur, le texte lié à chaque objet auquel il correspond. Pour plus de lisibilité, chaque ligne de texte est séparée par un trait. La première ligne de texte est le "titre" du nœud, ce sera "Globals" ou "Locals" pour les nœuds généraux listant les variables globales et locales. Quant aux autres nœuds, ils auront la même identification que dans la vue hiérarchique ; cela renforce la cohérence entre nos vues.
Quand un nœud est sous forme réduite, la seule ligne de texte qui est affichée est la première ligne d'identification. Pour un nœud étendu, toutes les autres lignes de texte représentent chacune des attributs liés à l'objet auquel est dédié le nœud.

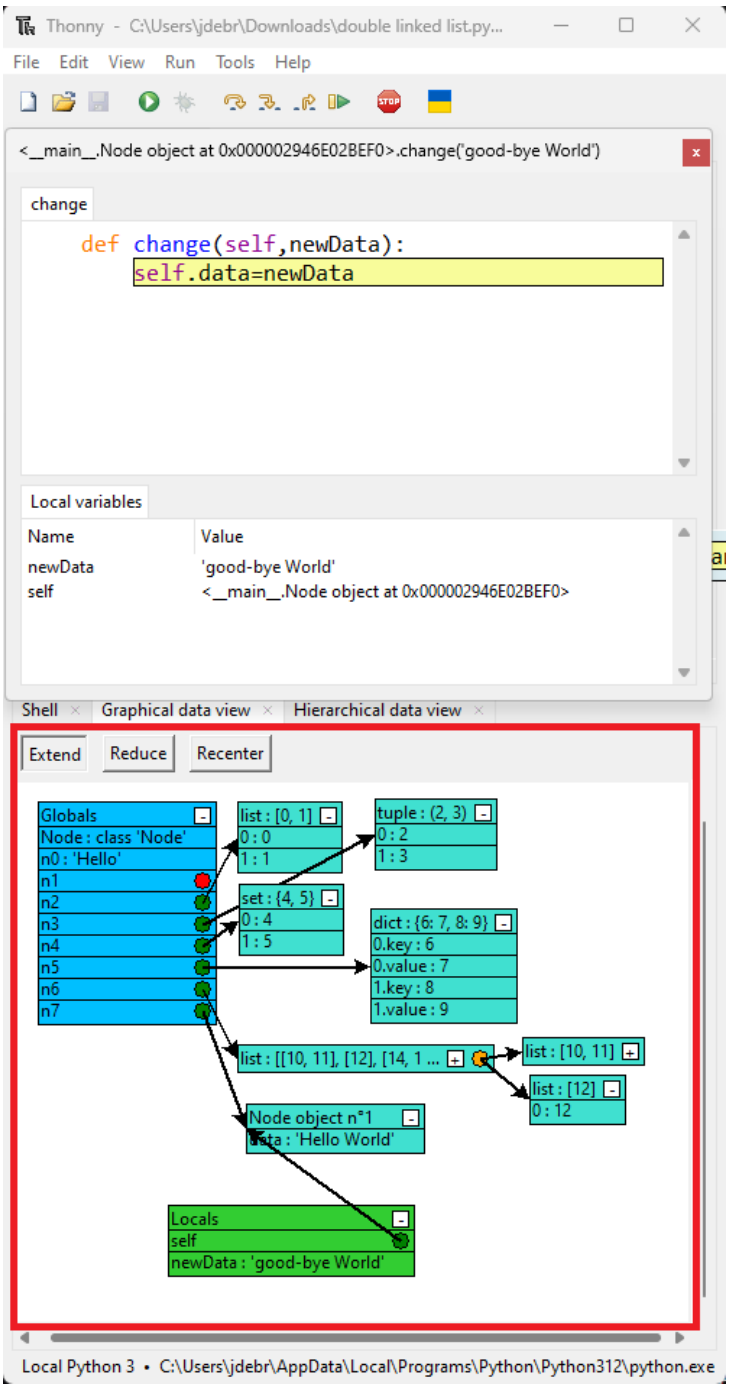


FIGURE 4.4 – Première version de la vue graphique

Lorsque les variables sont des types primitifs (comme les entiers, les chaînes de caractères ou les booléens), il y a simplement le nom de la variable, suivi d'un double point, suivi de sa valeur. Voir l'exemple "n0 :'Hello'" dans la figure 4.4
Pour les structures de données natives de Python, les noeuds correspondant ont des représentations particulières. Pour le noeud d'une liste ou d'un tuple, les différents champs

qui seront affichés en son sein indiqueront l'index, suivi d'un double point, suivi de la variable correspondante. Nous faisons la même chose pour les ensembles car, bien que dans un ensemble les variables ne devraient pas avoir d'index, Python les mémorise dans un ordre précis. Voir figure 4.4
Pour un nœud correspondant à un dictionnaire, nous représentons les paires clé-valeur ainsi : l'index de la paire, suivi d'un point est affiché sur deux lignes à l'intérieur du noeud. Sur la première est rajouté le mot "key", suivi d'un double point et de la variable servant de clé. Sur la seconde, le mot "value" suivi des deux points indiquera la variable correspondant à la valeur. Voir figure 4.4

Il y a des variables qui ne sont pas de type primitif mais une référence vers un objet ou une structure de données. Les noeuds qui concernent ces variables affichent alors une pastille de couleur contre leur bord droit plutôt qu'un double point, suivi de la valeur. Voir figure 4.4. Ces pastilles seront décrites plus tard.

IL a aussi fallu choisir la couleur des noeuds. Tout d'abord, pour que les nœuds ressortent individuellement forts, nous évitons qu'ils soient blancs ou transparents. La couleur attire le regard et permet d'accentuer la séparation entre l'intérieur de la boîte et l'extérieur.
Nous avons pensé à différencier les nœuds "Globals" et "Locals" du reste. Ces deux nœuds ont vraiment une signification différente des autres car ils ne représentent aucune structure de données mais sont la liste des variables globales et locales. Pouvoir les retrouver et les identifier rapidement est important car ils sont les points d'origine de tout le graphe. Au moins l'un des deux sera toujours présent, et c'est à partir d'eux que partent tout les autres. Ainsi, lorsqu'il y a beaucoup de nœuds, avoir un point de repère rend le graphe plus lisible. Nous avions pensé les différencier en colorant "Globals" en bleu et "Locals" en vert. Les autres noeuds seraient en turquoise, une couleur intermédiaire. Cependant, nous avons constaté que le bleu et le vert étaient trop criards et rendaient difficile la lecture du contenu des nœuds. Nous avons donc choisi quelque chose de moins tape-à-l'oeil : respectivement "deep sky blue" et "lime green", selon le diagramme des couleurs de Tkinter [74]. Voir figure 4.4.

Les pastilles rondes et de couleur, sur lesquelles l'utilisateur doit cliquer pour ouvrir ou fermer une référence, devaient garder la même logique pour les nœuds à l'état réduit et l'état étendu. Or, quand le nœud est à l'état étendu, les pastilles ont deux situations possibles : la référence est ouverte ou fermée. A l'inverse, quand le nœud est à l'état réduit cette pastille n'a pas la même signification et a trois situations possibles : soit l'ensemble des références du nœud est ouvert, soit seulement une partie des références est ouverte, soit elles sont toutes fermées. Même si la pastille du nœud étendu et celle du nœud réduit ne représentent pas exactement la même chose, cela reste fort lié. Il vaut donc mieux avoir quelque chose de standardisé pour les deux types. C'est pour cela que nous avons choisi de les représenter à l'aide des 3 couleurs les plus symboliques et universelles pour représenter 'ouvert', 'à moitié ouvert' et 'fermé'. Ces couleurs sont les couleurs des feux de circulation : respectivement vert, orange et rouge. Nous pensons que ces couleurs sont assez représentatives et compréhensibles sans explications. Voir figure 4.4.

Dans l'état réduit, il est possible que toutes les pastilles ne soit pas en même temps soit ouvertes soit fermées dans le cas où seulement certaines références sont ouvertes quand le noeud passe de l'état étendu à l'état réduit. Le cas de figure dans lequel seulement une

partie des pastilles est ouverte peut arriver aussi dans une autre situation. En effet, dans Thonny, les appels à la mémoire peuvent prendre un peu de temps. Aussi lorsque le nœud a beaucoup de références à ouvrir et qu'elles nécessitent toutes d'être ouvertes en même temps et de faire un appel à la mémoire de Thonny (car elles n'auraient pas encore été ouvertes par le passé), l'œil humain peut voir que le nœud parent n'a pas encore toutes ses références ouvertes, parce qu'elles s'ouvrent l'une après l'autre. Nous avons fait en sorte que la pastille ait le repère visuel d'entre-ouvert (c'est-à-dire l'orange). Cela a son importance ! En effet si l'utilisateur clique autre part et fait un autre appel à la mémoire, cela va écraser les appels venant du premier nœud et stopper l'ouverture de toutes ces références. Cela a pour conséquence que le nœud ne finira qu'avec une partie de ses nœuds ouverts et gardera une pastille orange.

En ce qui concerne les boutons conçus pour réduire ou étendre individuellement chaque nœud, nous avons voulu bien les différencier avec les pastilles et faire en sorte qu'il soit évident qu'il s'agit d'un autre type de boutons, conçus pour quelque chose de différent. C'est pourquoi les pastilles sont rondes tandis que le bouton "réduire & étendre" individuel à chaque nœud est carré. L'emplacement des pastilles est défini par un code couleur et la situation du bouton est définie par des symboles. Pour choisir les bons symboles, il nous fallait quelque chose de binaire et d'universel qui représente le concept d'agrandir et de réduire. Pour cela nous avons choisi respectivement "+" et "-".
Il a ensuite fallu choisir l'emplacement de ce bouton. L'endroit où il y a généralement de la place à l'état étendu est à droite du titre, au-dessus de toute éventuelle pastille. Si le nœud a au moins une pastille, cela n'agrandit pas davantage le rectangle car ce sera toujours un endroit vide. En outre, quand il y a beaucoup de noeuds affichés, il vaut mieux qu'ils ne soient pas trop grands pour garantir la lisibilité. L'absence de pastille risque d'agrandir le noeud mais standardiser l'endroit où le bouton est placé offre plus de lisibilité. Quand le nœud est à l'état réduit, il n'y a qu'une seule ligne et donc que trois endroits possibles. Pour standardiser à l'état étendu, il faut le faire après le titre. Or, si le nœud a une pastille, il faut qu'elle soit le plus à droite, car à partir d'elle peuvent partir beaucoup d'arêtes qui s'étendront généralement vers la droite (nous en expliquons plus tard la raison). Pour plus de lisibilité, il est donc nécessaire de ne pas avoir le bouton "réduire & étendre" superposé par les arêtes et donc de le placer directement à la droite du titre, avant la pastille s'il y en a une.

Pour les arêtes, nous avons choisi la simplicité : des flèches noires et droites qui représentent donc la référence d'une variable vers un objet ou vers une structure de données. Chaque flèche n'est affichée que si la référence qu'elle représente a été ouverte par l'utilisateur. Si le nœud de départ est dans l'état étendu, la flèche part du centre de la pastille de couleur liée à une variable qui a comme valeur une référence vers un objet ou une structure de données. Si le nœud est dans l'état réduit, la flèche part de la seule pastille de couleur du nœud, qui représente à elle seule toutes les références de ce nœud. Une pastille d'un nœud à l'état réduit peut donc avoir plusieurs flèches qui partent d'elle, ce qui n'est pas le cas avec un nœud étendu. Chaque flèche termine en pointant au centre du bord gauche de la boîte qui représente l'objet ou la structure de données auxquels la flèche fait référence.
Le point de départ est assez évident. Le point d'arrivée est l'endroit fixe le plus logique car le graphe s'étend toujours vers la gauche. En effet, les nœuds enfants qui sont affichés pour la première fois seront placés à droite de leurs nœuds parents. S'il faut choisir un

seul endroit fixe où arrivent les arêtes pour chaque nœud, c'est le plus évident. Nous avons pensé qu'avoir un endroit fixe pour chaque nœud, quelle que soit sa taille ou sa position, rendrait l'affichage plus standard et donc plus compréhensible. Pour notre extension, il serait contre-productif et assez complexe de faire des flèches courbes. En effet, vu que notre graphe est interactif et que les nœuds peuvent être changés de place, si les courbures des arêtes sont modifiées avec les déplacements de ces derniers, cela va devenir rapidement illisible. Si l'utilisateur a affiché beaucoup de nœuds et veut en déplacer un, les arêtes qui lui sont liées risquent fort de changer tout le temps radicalement de direction. Elles zigzagueront entre tous les nœuds affichés pendant le déplacement du nœud pour trouver le trajet le plus court sans superposer de nœuds. De plus, nous trouvons que des flèches droites resteront plus lisibles même si elles se superposent avec d'autres noeuds. En effet, il suffit alors d'observer le début ou la fin de la flèche pour savoir la direction de son autre extrémité.

Quand un nouveau nœud est affiché, notre algorithme va le positionner soit en face de son nœud parent s'il est le seul noeud enfant affiché, soit en dessous du plus bas nœud affiché qui est enfant du même parent. En d'autre termes, la position d'un nœud n'est choisie qu'en fonction de la position du nœud parent et de ces nœuds "frères". Notons que cela ne compte que pour les nouveaux nœuds. Un nœud qui aurait déjà été affiché par le passé puis caché, se ré-affichera à l'endroit où il avait été placé précédemment. S'il avait des références ouvertes au moment où il a été caché, lorsqu'il sera ré-affiché les mêmes références seront à nouveau ouvertes. Par exemple, lorsque l'utilisateur a affiché toute une liste chaînée à un endroit et qu'il cache le premier élément de la liste, toute la liste va être cachée. Lorsqu'il ré-affiche ce premier élément de la liste, toute la liste qui avait été affichée avant sera à nouveau affichée d'un seul coup et au même endroit que précédemment. Cette idée de garder les placements nous parait essentielle pour aider l'utilisateur à rester concentré en laissant le plus possible les éléments là où il les avait mis et dans l'état dans lequel il les avait laissés.

L'option "Recenter" se fait ainsi. En commençant du nœud "Globals", nous allons faire un parcours en profondeur du graphe avec tous les nœuds visibles. Lors du parcours, si l'algorithme parcourt à nouveau un nœud déjà repositionné ou un nœud qui n'a aucune référence ouverte, il revient en arrière pour explorer le chemin suivant. Et lorsque que l'algorithme parcourt un nœud, il va le positionner avec la même logique que l'affichage d'un nouveau nœud : il sera repositionné seulement en fonction de son nœud parent et de ses autres nœuds "frères" qui sont affichés et qui ont déjà été repositionnés. Une fois que tout est fait avec le nœud "Globals" et ses enfants, l'algorithme fait la même chose avec le nœud "Locals" et ses enfants juste en dessous. Cette idée de positionner ou repositionner les nœuds en ne prenant en compte que le nœud parent et ses autres nœuds enfants a le désavantage qu'il est possible que plusieurs nœuds se superposent. Cela est dû au fait qu'un autre nœud qui n'a rien à voir puisse venir d'une toute autre branche, même avec l'option "Recenter". Malgré ce désavantage, nous avons pensé que regarder la position de tous les autres nœuds affichés avant de trouver une nouvelle place pour le nœud pour qu'il n'y ait pas de superposition, rendrait l'exécution de la commande de l'utilisateur lente. De plus, ce défaut ne nous paraissait pas flagrant avec des graphiques tests de petite taille.

4.4.3 Arrière-plan

Un des choix de fonctionnement de notre extension trouve son origine dans une contrainte d'implémentation. Le fait que cela soit un affichage "lazy" - c'est-à-dire que

les nœuds ne soient pas tous affichés dès le début mais bien petit à petit en fonction des références que l'utilisateur choisit d'ouvrir -, était au début une solution technique. En effet, si, par exemple, dans le code de l'utilisateur il y a une structure de données qui crée de manière "lazy" une liste chaînée infinie, cela peut poser problème. Nous pensons notamment aux "numpy.array" qui ont comme argument "T" leur transposée qui correspond donc à une nouvelle "numpy.array" avec elle-même une transposée et ainsi de suite. Puisque c'est "lazy", cela ne crée pas de problème dans l'exécution du code. Cependant, pour un outil de visualisation, si nous affichons tous les nœuds automatiquement, cela va créer une structure de données avec un nombre infini de nœuds et donc faire crasher Thonny.

La question maintenant est de savoir comment l'affichage décrit précédemment avait été implémenté. Tout d'abord, tout le graphisme a été réalisé avec Tkinter qui est décrit dans l'état de l'art à la section 2.1.1. Cette librairie est intégrée par défaut à Python et donc à Thonny, ce qui fait que Tkinter ne nécessite pas d'installer une librairie supplémentaire. Vu que notre outil a un but pédagogique, nos utilisateurs sont en majorité des débutants pour lesquels il faut proposer une installation facile, avec peu d'étapes. Cela implique aussi que c'est gratuit. En effet, Thonny étant un outil open source et gratuit, faire payer serait incohérent. Bien que notre outil soit utile, ses fonctionnalités principales sont accessibles sur PythonTutor qui est gratuit. Si notre outil était payant, il perdrait beaucoup de son attrait par rapport à son "concurrent". Nous avons choisi Tkinter car c'est déjà l'outil utilisé par Thonny pour son graphisme. Cela permet de facilement et très bien s'intégrer à l'IDE. Thonny fonctionnant sur Windows, macOS et Linux, il nous fallait un outil pour afficher notre diagramme réseau qui soit lui aussi multiplateformes. c'est le cas de Tkinter. Il nous fallait quelque chose d'interactif qui nous permette de placer et d'afficher les nœuds là où nous voulions qu'ils soient. Nous avons trouvé plusieurs librairies comme Matplotlib, PyGraphviz ou Bokeh, spécialisées pour afficher des graphiques et, entre autres, des diagrammes réseau. Malheureusement, aucune d'entre elles ne permet une si grande interactivité. Elles affichent souvent le diagramme réseau avec les nœuds à des endroits optimaux pour avoir moins d'arêtes qui se croisent, mais cela n'est pas un affichage qui nous parait intéressant. Les quelques librairies qui permettent le niveau d'inactivité attendu, comme PyVis ou Dash Cytoscape, sont spécialisées pour afficher leur graphe dans un navigateur web. Ce ne sont pas des outils prévu pour être intégrés dans Thonny. Réussir à les intégrer est compliqué et le design dénoterait avec celui de Thonny, fait en Tkinter.

Davantage que l'absence de superposition, il nous parait plus important de comprendre les structures de données que l'utilisateur a créées. Pour cela, il est intéressant d'avoir un ordre dans l'affichage. Si l'utilisateur n'a rien bougé, au moins un des parents de n'importe quel nœud se trouvera toujours à sa gauche et à la même hauteur ou au-dessus. Cet ordre est plus important que d'afficher un graphe sans aucune superposition. En plus de cela, ces librairies ne sont pas assez interactives pour faire du "glisser-déposer" (ou drag-and-drop en anglais). Permettre à l'utilisateur d'agencer comme il l'entend son graphe est aussi une fonctionnalité importante que nous voulions réaliser. Avec ces critères nous avons compris qu'utiliser un outil de visualisation spécialisé pour les diagrammes réseaux ne serait pas possible. Il nous faut donc une librairie pour la création d'interfaces graphiques générales capables de créer des rectangles, des flèches, du texte et des boutons. Tkinter qui est un outil fort complet et qui donne déjà l'affichage de Thonny est un choix parfait. Les défauts que l'on peut noter de Tkinter sont son apparence un peu simple et sa performance qui peut aussi manquer par rapport à d'autres outils existants de nos jour. Ces défauts ne

nous concernent pas vraiment car nous cherchons justement à produire quelque chose d'épuré avec un design simple. Nous voulons en effet ne pas distraire l'utilisateur avec des éléments superflus. Nous ne dessinons que quelques rectangles, flèches, textes et boutons et il n'y a pas d'animations, de 3D ou autres éléments complexes. C'est pour cela que la performance de Tkinter est plus que suffisante.

Nous utilisons quand même un outil spécifique pour la création et la manipulation de diagrammes réseaux qu'il faut installer en plus avant d'utiliser notre extension. Cet outil est "NetworkX", il est décrit dans l'état de l'art à la section 2.1.1. Cet outil fonctionne parfaitement pour ce que nous réalisons : un diagramme réseau orienté, avec potentiellement des arêtes qui bouclent sur un nœud ou des arêtes parallèles. A partir d'un même nœud, il peut y avoir plusieurs références qui pointent vers le même objet. Bien que cet outil ne soit pas installé d'office avec Thonny, il est très utile et rend notre code plus efficace et plus rapide.

Les appels mémoire sont faits avec les mêmes fonctions que la vue hiérarchique utilisée qui se trouve dans le fichier "sender.py" de notre code source. Quant au code source de notre vue graphique, nous l'avons séparé en trois fichiers distincts pour plus de clarté. Le premier fichier "graphical_view.py" définit les données qu'il faut rechercher. C'est lui qui va faire appel aux fichiers "sender.py" et "repr_format.py" que nous avons en commun avec les autres vues pour faire les appels mémoire et créer le format dans lequel nous voulons afficher les données. Ensuite, "graphical_view.py" va faire appel au fichier "DB.py" dans le dossier "Graphical". Il s'agit du fichier qui va gérer le diagramme de réseau. Il va, par exemple, créer les lieux entre les nœuds, mémoriser l'état des nœuds, leur position, l'état de leur référence... C'est dans ce fichier que NetworkX est utilisé. Après cela, "DB.py" va faire appel au fichier "graphic.py", encore une fois dans le fichier "Graphical". C'est dans ce fichier que toute la partie visualisation graphique va être réalisée et que Tkinter sera importé et utilisé.

Chapitre 5

Validation de la première version

Dans ce chapitre, nous allons vous présenter la démarche effectuée pour tester et valider la première version de notre outil. Nous discuterons ensuite des modifications qui y ont été apportées. Tout d'abord, nous expliquerons en détails la méthode de validation que nous avons employée. Nous parlerons ensuite des observations et remarques que nous avons obtenues. Puis nous présenterons les changements apportés suite à celles-ci. Nous finirons par discuter de la validité de ces retours.

5.1 Méthode

Après avoir réalisé une première version de notre outil, nous l'avons nous-mêmes testé avec des programmes plus ou moins complexes et des structures de données de différentes tailles et origines. Ensuite, nous l'avons envoyé à une quinzaine de testeurs potentiels afin d'obtenir des retours constructifs permettant de valider notre travail. Dans le but de rendre notre outil accessible à nos testeurs, nous l'avons déposé dans un répertoire github dont le lien est en annexe. Par ailleurs, nous avons déposé un fichier compilé de notre outil dans la librairie PyPI afin de rendre l'extension facilement accessible à partir de Thonny.

Pour aiguiller nos testeurs dans leurs validations, nous leurs avons fourni un fichier Python contenant un programme dans lequel se trouvait des structures de données particulièrement adéquates à visualiser à l'aide de notre outil. Nous leurs avons aussi fourni un second fichier contenant un questionnaire pour les aider à analyser notre travail. Après plusieurs discussions sur la pertinence de certaines questions avec notre promoteur, nous avons pu rédiger et regrouper les questions retenues en 4 grandes parties. La première, plus générale, demandait si les différentes vues fonctionnaient correctement et répondaient aux attentes des testeurs. La deuxième, spécifique à la visualisation des données, questionnait la clarté des vues. La troisième ciblait l'utilisation de l'outil, son intuitivité et son bon fonctionnement et enfin, la dernière partie posait des questions ciblées sur l'aspect didactique des différentes vues et s'assurait de leur utilité. Vous trouverez ce questionnaire en détail dans la partie annexe à la section 8.
Après avoir reçu les retours de nos testeurs, nous avons modifié notre outil en fonction de ceux-ci.

5.2 Résultats

A la fin de notre démarche de validation, nous avons regroupé les remarques de nos différents testeurs. Dans cette section, nous allons vous les présenter succinctement. Afin de respecter la structure du questionnaire que nous leurs avons envoyé, nous commencerons par parler du fonctionnement général de notre outil. Ensuite, nous aborderons la qualité de la visualisation des données et nous expliciterons l'utilisation ainsi que la prise en main de l'extension. Enfin, nous traiterons de son aspect didactique.

5.2.1 Retours sur le fonctionnement général

Efficacité et clarté de l'outil

La plupart de nos testeurs ont trouvé que notre outil était particulièrement bien adapté au débogueur de Thonny. En outre, certains ont affirmé que le but de venir en aide à des débutants en informatique en première année à l'université était accompli. L'un de nos testeurs a cependant relevé le fait que nos choix de représentation des variables et objets n'étaient pas toujours cohérents ni standardisés.

Au niveau de la vue hiérarchique, plusieurs de nos testeurs y ont trouvé une utilité semblable à ce qu'ils utilisent dans d'autres environnements de développement tel que VSCode ou PyCharm. Ils trouvent ainsi que cet outil de visualisation est pratique pour comprendre de façon générale le contenu des structures présentes dans le programme. Cette vue permet, selon eux, de vérifier rapidement une valeur dans une structure et également de contrôler l'évolution du programme.

La vue graphique serait quant à elle plus utile pour visualiser facilement les objets imbriqués. D'après nos testeurs, elle met en évidence les relations entre ces objets et permet d'identifier plus rapidement les différentes références à un même objet dans le programme. Plusieurs testeurs ont par ailleurs soulevé la perte de clarté et d'efficacité de l'outil lorsque le programme présentait de trop nombreux objets imbriqués.

Pistes d'améliorations pour la clarté et la simplicité de l'outil

Une majorité de nos testeurs ont trouvé qu'il serait plus clair de distinguer les variables issues d'importation des variables créées par l'utilisateur. En effet, celles importées sont souvent plus complexes et contiennent de nombreux attributs qui ne sont pas toujours pertinents à afficher. Dans la continuité, nos testeurs nous ont également fait remarquer qu'il serait plus clair de présenter moins de méta données. De fait, celles-ci ne sont pas toujours utiles et alourdissent fortement certaines représentations d'objets ou variables.

D'après nos testeurs, il faudrait également proposer quelques explications sur les choix principaux de la représentation tels que le choix des couleurs. D'ailleurs, pour certains, il serait intéressant d'utiliser davantage de couleurs pour séparer les types de données ou mettre en valeur certaines informations telles que la première ligne de chaque boîte représentant l'objet ou la variable. L'un de nos testeurs nous a d'ailleurs fait remarquer que notre outil ne supportait pas la représentation des variables de classe.

Au niveau plus spécifique de la vue hiérarchique, un testeur nous propose d'afficher systématiquement le type de la variable native de Python (int, float, string, ...), ce qui pour le moment n'est fait que pour les objets plus complexes.

Pour ce qui est de la vue graphique, plusieurs testeurs trouvent que la navigation parmi les noeuds représentant les variables et objets manque de fluidité. Ils nous conseillent de permettre le défilement de la vue à l'aide de la molette de la souris et également d'ajouter une fonctionnalité de zoom et dézoom. Nos testeurs nous ont également fait remarquer que nous utilisons le symbole des " :" pour traduire des informations qui ne se correspondent pas toujours. En effet, il est parfois utilisé pour attribuer une valeur, parfois une représentation, parfois une référence. Pour pallier ceci, ils nous ont conseillé de représenter les structures de base de façon plus visuelles (à l'aide des indices pour une liste par exemple) et de diminuer les informations méta, comme mentionné plus haut.

Bugs et comportements anormaux de l'outil

Nos testeurs nous ont rapporté plusieurs bogues mineurs : la vue hiérarchique refuse parfois de s'afficher avant la fermeture d'une image de la librairie Matplotlib si le programme en fait apparaître. Au niveau de la vue graphique, une forte latence est observée lors de l'affichage de certains objets sur un ordinateur de faible puissance. Lors de l'extension de certains noeuds, les boites qui apparaissent se superposent à des boîtes déjà présentes. L'un de nos testeurs nous a également fait part d'un problème d'affichage des flèches représentant les relations qui apparaissent en blanc et deviennent donc difficiles à distinguer.

Complétude et fonctionnalités de l'outil

Plusieurs fonctionnalités à ajouter nous ont été suggérées par nos testeurs : tout d'abord la possibilité d'afficher les 100 éléments suivants d'une liste (pour le moment, dans le cas de grandes listes, notre outil n'affiche que les 100 premiers éléments). Ils proposent aussi l'ajout d'options de filtrage et de recherche dans les grandes structures de données pour faciliter l'exploration. Enfin, ils suggèrent la création de petits tutoriels ou de vidéos illustrant l'installation et l'utilisation de nos différentes fonctionnalités.

5.2.2 Retours spécifiques à la visualisation de données

Aide à l'apprentissage et au débogage de structures de données

De façon générale, nos testeurs s'accordent à dire que la vue graphique est plus adaptée à la compréhension des structures de données que la vue hiérarchique.

Plus spécifiquement, dans la vue hiérarchique, les testeurs ont tendance à perdre la notion de référence vers un même objet. En effet, celle-ci diminue en lisibilité si les objets contiennent beaucoup de données ; elle convient moins à l'affichage de structures complexes contenant des boucles (car il n'y a pas de retour visuel permettant de voir qu'on observe les mêmes objets). Par contre, cette vue a l'avantage d'être très pratique pour explorer les structures de données lors du débogage et, particulièrement, les dictionnaires ou les très longues listes. À nouveau, nos testeurs nous recommandent d'ajouter des commentaires et des aides pour faciliter la compréhension des structures qui y sont affichées.

Notre vue graphique offrirait donc une meilleure compréhension des structures et des liens entre les différents objets et serait donc la plus utile des deux vues. Cependant, un de nos testeurs nous confie la trouver moins claire et aurait tendance à moins l'utiliser. Plusieurs autres nous ont conseillé de simplifier grandement l'affichage et les informations qui y étaient représentées et qui pourraient embrouiller les programmeurs débutants. Ils nous suggèrent de se concentrer sur l'affichage simple des boîtes, et surtout des liens entre celles-ci.

Clarté de la visualisation des grands tableaux et dictionnaires

Une remarque nous est souvent parvenue concernant l'affichage des dictionnaires dans la vue graphique, mentionnant qu'il est difficile de distinguer la différence entre les clés et les valeurs de ceux-ci. Nos testeurs nous conseillent alors d'utiliser des couleurs, des icônes ou une représentation sur la même ligne (de type "clé : valeur") afin de mieux les distinguer.

Adéquation de l'outil à visualiser des structures de données importées

Nos testeurs nous ont fait remarquer que plusieurs données ne nous concernent pas et alourdissent l'affichage des objets importés. Pour pallier ce problème, ils nous proposent d'ajouter une option permettant de masquer ou filtrer les attributs non pertinents.

Adéquation de l'outil à visualiser des structures de données créées

D'après nos testeurs, c'est justement pour ce cas-ci que notre outil est véritablement pertinent et utile. Un testeur nous suggère encore une fois de minimaliser nos méta données pour faciliter la compréhension. Il nous propose aussi de simplifier ou de supprimer la présence de la représentation dans l'en-tête de la boîte correspondant à un objet ou à une variable.

5.2.3 Utilisation de notre outil

Clarté et simplicité de l'outil

Les testeurs ont trouvé que les deux vues étaient suffisamment instinctives et visuelles. Le fait que notre outil permette une meilleure visualisation de la structure de données permet de simplifier l'analyse et la compréhension de celles-ci. Ils nous ont tout de même conseillé d'inclure des conseils d'utilisation afin de rendre les boutons cliquables plus indicatifs et de les mettre plus en évidence.

Fonctionnalités de l'outil

Les testeurs trouvent les fonctionnalités de la vue graphique utiles et fonctionnelles. Le choix du nom du bouton permettant de réaligner les noeuds a été qualifié comme peu clair et peu explicite. En effet, les termes "allign" ou "center" sont plus évidents et plus utilisés globalement que "recenter", ce qui permet une facilité d'utilisation et de compréhension de l'outil. Un testeur trouverait avantageux de pouvoir réorganiser la vue graphique en prenant en compte les liens entre les boîtes. Il trouverait aussi utile d'ajouter une fonctionnalité pour tout développer ou réduire dans la vue hiérarchique.

Interaction entre les testeurs et l'outil

Nos testeurs nous ont conseillé de fournir avec notre outil un guide ou un tutoriel qui expliquerait les choix des différents termes et des codes couleurs afin de favoriser la clarté de l'outil. Ceci permettrait une meilleure interaction entre l'utilisateur et celui-ci.
Par ailleurs, cela pourrait être pertinent de laisser l'utilisateur choisir les couleurs en paramétrant librement le choix de celles-ci. Cette option permettrait d'avoir un code couleur propre et clair pour chaque utilisateur.

5.2.4 Aspect didactique de notre outil

Apport pour la compréhension des structures de l'outil

Les testeurs nous ont partagé que les deux vues exploitées par l'outil apportent une bonne compréhension des structures en Python. Cela pourrait être un réel atout pour l'enseignement de ce langage.

Les améliorations proposées pour aider à l'apprentissage du codage grâce à notre outil

Les testeurs nous ont conseillé de fournir, dans notre outil, des exemples de codes simples et commentés afin de faciliter et de fluidifier l'utilisation du programme. Cela peut également être pertinent de fournir des codes plus complexes. Les utilisateurs pourraient se baser sur notre "mode d'emploi" dans un premier temps pour mieux manipuler l'outil.

5.3 Analyse et discussion

Suite aux différents retours et conseils de nos testeurs, nous avons considérablement amélioré notre travail. Dans cette section, nous allons discuter plus en détails les différentes modifications que nous avons apportées et les conseils que nous avons décidé de rejeter ou de laisser comme travail futur.

5.3.1 Fonctionnalités

Tout d'abord, plusieurs fonctionnalités additionnelles nous ont été conseillées pour améliorer la prise en main de l'outil et assurer son bon fonctionnement. La première que nous avons ajoutée est la possibilité de naviguer au travers de la vue graphique à l'aide de la molette de la souris. En effet, pour le moment, il n'était possible de la parcourir qu'en utilisant la barre de défilement sur le côté.

On nous a également fait remarquer qu'en voulant simplifier la visualisation, nous ne permettions pas de développer les classes. Or, cela empêche l'utilisateur de visualiser les variables de classe. Nous les avons maintenant affichées.

La dernière fonctionnalité que nous avons ajoutée est la possibilité d'observer davantage que les 100 premiers éléments ou attributs. Auparavant, nous limitions l'affichage à 100 éléments ou attributs pour ne pas surcharger les différentes vues et pour ainsi offrir plus de clarté. Maintenant, nous limitons l'affichage à 10 éléments mais nous permettons à l'utilisateur de cliquer sur un bouton intégré à la variable ou à l'objet affiché dans l'une des

vues. Ceci lui permet de visualiser les 10 éléments ou attributs suivants et ainsi de suite. Cet ajout permet une plus grande lisibilité quand l'utilisateur veut parcourir succinctement les éléments des vues. Il permet également une meilleure lecture en profondeur lorsqu'il souhaite visualiser un attribut ou un élément précis.

5.3.2 Simplification

À de nombreuses reprises, nos testeurs nous ont fait remarquer que nous pourrions grandement simplifier nos choix de représentations pour les objets et variables, et ce tant dans la vue hiérarchique que la vue graphique. Le conseil qui nous a été le plus fréquemment donné est de supprimer les informations méta inutiles. Particulièrement pour la vue graphique, essayer de recoller le plus possible à une vue voulant simplement montrer des liens entre des variables et objets sans surcharger les débutants avec des informations additionnelles.

Pour réaliser cela, nous avons décidé de retirer dans la vue graphique les références à la mémoire, que nous avions simplifiées, ainsi que les mots clés de type "object", qui spécifiaient que la variable était bien un objet. Ainsi, là où avant nous représentions par défaut un objet "Node" créé par l'utilisateur avec une boîte dont la première ligne stipulait : "Node object n°X", à présent, la première ligne indique uniquement "Node". Nous considérons que les références mémoires sont toujours utiles pour la vue hiérarchique car elles permettent de retrouver les objets qui sont liés entre eux. Cependant, pour la vue graphique, ce rôle est joué par le flèches ; les objets ne nécessitent donc plus d'être distingués dans le contenu de leur boîte.

De la même façon, nous avons remplacé la première ligne des boîtes pour les dictionnaires, listes, tuples et sets qui montrait leur représentation et était donc redondante avec les informations que nous affichions dans la boîte. À la place, nous avons décidé d'afficher uniquement le type de la structure de données (comme nous le faisons par défaut pour les objets). Ainsi, une liste n'aura plus "[contenu]" comme première ligne mais simplement "list" et, à l'intérieur de la boîte, il sera possible d'observer le contenu.
Malgré ces simplifications, nous voulons garder la possibilité d'afficher la représentation implémentée spécifiquement pour les objets lorsque c'est le cas. Ce choix vient du fait que certaines variables importées n'ont pas toujours des attributs très clairs mais souvent une représentation par défaut assez éloquente. Ainsi, nous préférons garder cette représentation comme première ligne pour faciliter la compréhension de ces structures par un débutant.

Au sujet des variables importées, plusieurs testeurs ont souligné le fait que les attributs de celles-ci sont souvent peu utiles au programmeur. Par ailleurs, leur complexité risque d'embrouiller les débutants, d'autant plus que ces attributs ont tendance à être nombreux ; ce qui amène une surcharge des vues, tant graphique que hiérarchique. Pour toutes ces raisons, nous avons fait le choix de masquer ces attributs en suggérant à l'utilisateur qui souhaite les consulter de les observer dans l'inspecteur d'objet de Thonny, justement conçu pour cela. Dans la vue graphique, nous avons étendu cela en ajoutant un bouton permettant à l'utilisateur d'afficher ou non ces attributs des variables importées. Nous avons également profité de cette fonctionnalité pour masquer certaines informations (telles que les méthodes de classes) des objets afin de simplifier davantage la vue. Elles restent cependant accessibles grâce au nouveau bouton.

5.3.3 Clarification

Plusieurs remarques nous ont également été faites sur le manque de clarté de certains de nos choix d'implémentation. Ainsi, nous utilisions à plusieurs reprises les deux points pour signifier des choses différentes. Nous les utilisions aussi bien pour indiquer des représentations dans les en-têtes des boîtes de certains objets que pour attribuer une valeur à certaines variables, mais également pour présenter ce à quoi correspondaient certains indices. Pour résoudre ce manque de sens de l'utilisation des deux points, nous l'utilisons maintenant exclusivement pour la correspondance entre indices (ou clés et valeurs) et leur contenu. Comme expliqué dans la section précédentes, nous avons supprimé la plupart des représentations dans les en-têtes des boîtes de la vue hiérarchique. Quant aux attributions de valeurs aux variables, nous utilisons maintenant le symbole "égal" qui nous semble bien mieux convenir (par exemple i = 5).

Dans la continuité de cette amélioration, nous avons clarifié la représentation des indices et, plus particulièrement, de la distinction entre clés et valeurs des dictionnaires. Comme expliqué précédemment, tous les indices indiquent ce qu'ils contiennent à l'aide de deux points. Dans le cas des dictionnaires, il faut également mettre en évidence la paire clé-valeur, ce qui était simplement traduit par un indice mis devant. Pour davantage de clarté, nous avons indenté les paires clé-valeur à l'aide d'une flèche et nous avons gardé l'indice pour assurer l'identification du bon couple. On nous avait suggéré de placer le couple clé-valeur en une seule ligne dans la boîte correspondant au dictionnaire, mais nous trouvions que cela alourdissait la lecture de la vue graphique lorsque les deux champs pointent vers une autre boîte.

À côté de cela, pour mettre en évidence le champs le plus important de chaque boîte et faire ressortir ce qui resterait dans le mode "réduit", nous avons surligné le titre de chaque boîte de la vue graphique qui correspond à l'identification de la boîte. Dans la plupart des cas, il s'agit du type de la variable ou de l'objet contenu dans la boîte.

Par ailleurs, la gestion de l'emplacement des boîtes dans la vue graphique peut parfois manquer de clarté, notamment à cause de leur superposition. Pour résoudre ce problème, nous avons amélioré la disposition des boîtes lors de leur apparition, limitant leur superposition. Notre algorithme va simplement positionner le nœud juste à droite de son parent et s'il n'y a pas de place là, il va trouver le premier endroit avec de la place en dessous de cet endroit de départ et y placer le nœud. Nous avons fait de même lorsqu'elles sont réalignées à l'aide du bouton "Recenter". D'ailleurs, ce bouton était également une source de confusion pour certains de nos testeurs qui trouvaient que le terme "Align" serait plus correct pour une fonctionnalité réorganisant les noeuds par défaut en les alignant. Nous avons donc également renommé le bouton pour qu'il soit plus explicite.

La superposition entre nœuds et flèches peut aussi être problématique. Nous avons donc aussi changé quelques éléments avec les flèches. Nous avons gardé l'idée d'avoir des flèches droites pour les raisons expliquées dans la section solution 4. Nous avons cependant pensé à éviter que les flèches se superposent avec leur nœud de destination ; ce qui risque de cacher le texte à lire ou de la faire disparaître sous son nœud de destination. Pour cela, nous avons fait en sorte que les flèches pointent vers le bord du nœud enfant le plus proche de son parent, au lieu de pointer exactement sur le centre du bord gauche du nœud. Nous

avons créé des flèches courbes dans une seule situation très spécifique le nécessitant : dans une boucle, quand la référence d'un nœud pointe vers lui-même.
À côté de cela, nous avons remarqué que les flèches n'avaient pas leur couleur fixée et se dessinaient par défaut en noir. C'est pourquoi nous pensons que c'est à cause de cela que les flèches se dessinaient en blanc pour l'un de nos testeurs. Ce défaut a été corrigé.

5.3.4 Accompagnement à la prise en main

En dehors de notre outil, il nous a plusieurs fois été conseillé de mettre en ligne une aide expliquant clairement l'installation et l'utilisation de notre programme. Cette aide prend la forme d'un manuel d'utilisation et peut être trouvée dans le répertoire github de notre travail dont le lien est en annexe. Elle est en partie présentée dans la section 6.

5.3.5 Conseils non appliqués

Quelques suggestions nous ont été données et nous avons décidé de ne pas les mettre en place.

La première concerne la boîte contenant les variables globales dans la vue hiérarchique. Il nous a été conseillé de la diviser en plusieurs colonnes présentant systématiquement le nom, la valeur et le type de chaque variable. Dans un souci d'allègement et de clarté de notre vue graphique, nous n'avons pas pensé que cela serait judicieux. D'autant plus que les différentes informations pourraient être redondantes avec celles contenues dans les boîtes des objets.

Les deuxième et troisième remarques sont davantage liées aux limitations de Thonny qu'à celles de notre travail. En effet, on nous a signalé que l'ouverture de nouveaux noeuds pouvait prendre un certain temps sur certains supports peu rapides. Nous pensons que cela est relativement normal et lié à la façon dont les requêtes sont réalisées vers la mémoire de Thonny, ce qui à tendance à prendre du temps. On nous a également fait remarquer que nos vues n'affichent rien tant qu'une image de la librairie matplotlib est ouverte. C'est aussi normal. En effet, le programme est considéré comme tournant toujours dans Thonny tant que l'image est ouverte. Or, il faut que le programme se termine pour que nous puissions afficher l'état des différentes variables et différents objets.

Enfin, on nous a demandé de réaliser un code couleur par type de données. Nous l'avons réalisé mais avons rapidement remarqué que trop de couleur a tendance à rendre la vue davantage illisible. Nous trouvions également l'intérêt limité étant donné que notre outil veut pouvoir montrer les liens entre les objets par leurs attributs et non par leur type commun. Nous avons donc décidé d'annuler cette modification et de ne pas proposer cet ajout.

Plusieurs autres conseils intéressants nous ont été donnés mais nous n'avons pas eu l'occasion de les appliquer. Nous en discutons dans la section 7.

5.4 Freins à la validité

La démarche que nous avons appliquée pour valider notre programme possède quelques points faibles qui peuvent avoir influencé l'évaluation de notre outil.
Tout d'abord, la première étape de notre démarche constituait en une auto-évaluation durant laquelle nous avons fait notre possible pour évaluer notre travail à l'aide d'une grande variété de programmes. Cependant, nous n'avons pas pu tester toutes les structures de données importées existantes ni certains programmes complexes qui sortent du champs d'application d'un débutant en informatique. Nous n'avons pas non plus testé notre extension en présence de plusieurs autres extensions de Thonny. Pour certains programmes, certains imports ou certaines extensions, il est donc possible que notre travail présente un dysfonctionnement.

D'un autre côté, nos testeurs ont été peu nombreux à répondre à l'appel. Nous avons reçu seulement une petite dizaine de retours. Parmi ceux-ci, la plupart nous viennent d'amis ingénieurs civils qui finissaient cette année ou ont commencé un doctorat et pourraient présenter des biais positifs dans leur évaluation.

Enfin, aucun de nos testeurs n'est véritablement un débutant en informatique et nous n'avons pas eu l'occasion ni la permission d'envoyer notre travail à des étudiants en première année de bachelier. Il est donc probable que les retours et les corrections que nous avons apportées à l'outil soient basés sur une mauvaise estimation des difficultés que peut rencontrer un étudiant débutant en informatique. Malgré tout, un de nos testeur est assistant pour le cours d'informatique 1 dans le cadre de la première année de bachelier en ingénieur civil à l'UCLouvain, cours qui a justement motivé la création de cet outil.

Chapitre 6

Présentation de la version finale

Dans ce chapitre, nous présentons visuellement les différentes fonctionnalités de la version finale de notre travail. Nous commencerons par expliquer les démarches à réaliser pour installer notre extension dans Thonny. Nous partirons ensuite d'un programme Python adapté à un niveau de programmeur débutant et nous le visualiserons à l'aide des outils que nous avons créés.
Le lien vers le répertoire Github de notre travail complet est en annexe.

6.1 Installation

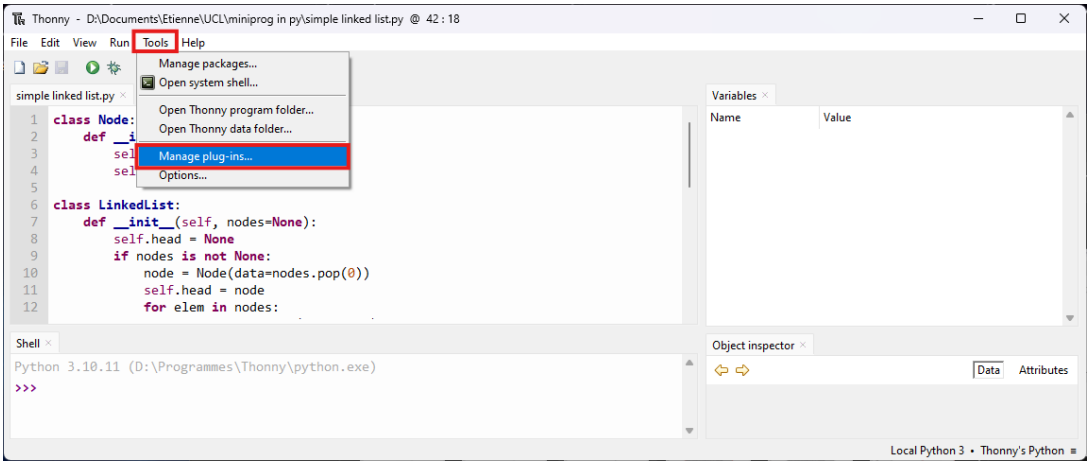


FIGURE 6.1 – Première étape d'installation

Nous allons tout d'abord vous présenter les étapes à réaliser afin d'installer correctement notre extension. Commencez par installer une version de Thonny supérieure ou égale à la version 4.1.4. Cela fait, ouvrez Thonny. Dans le coin en haut à gauche, sélectionnez l'onglet "Tools" (ou "Outils" si vous l'avez en français) et cliquez sur "Manage plug-ins..." (ou "Gérer les plugins..."). Voir la figure 6.1.

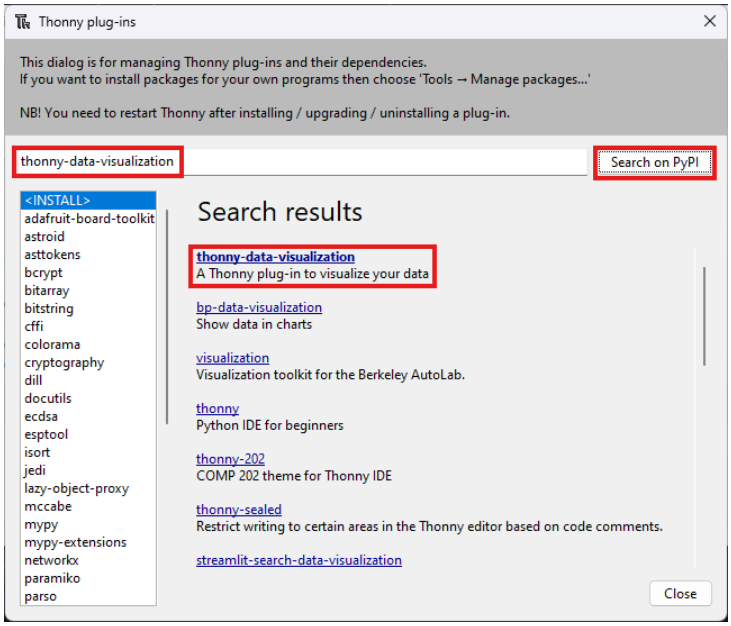


FIGURE 6.2 – Deuxième étape d'installation

A présent, une fenêtre vous propose de rechercher une extension. Tapez "Thonny-data-visualization" dans la barre de recherche et appuyez sur le bouton "Search on Pypi" (ou "Rechercher sur Pypi"). Sélectionnez la première proposition. Voir figure 6.2.

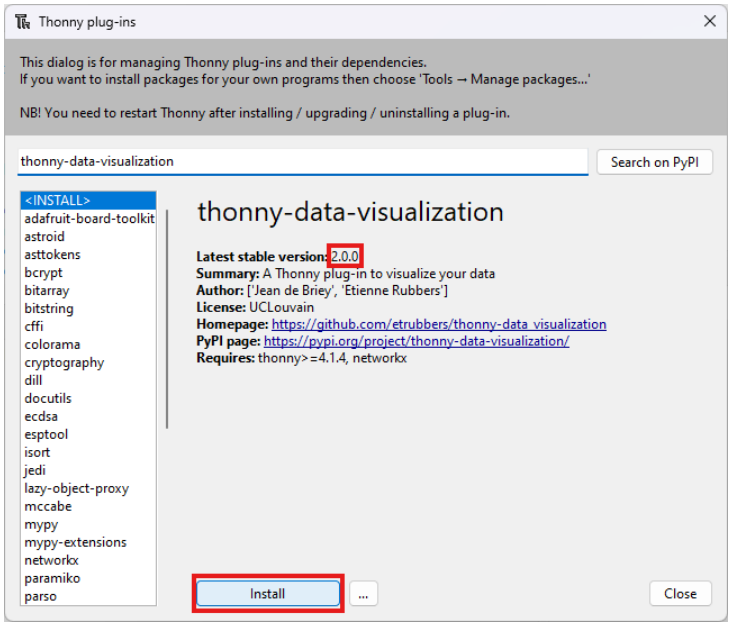


FIGURE 6.3 – Troisième étape d'installation

Vérifiez bien que la version que l'on vous propose d'installer est la plus récente (version 2.0.0 ou au-delà). Vous pouvez maintenant installer l'extension en cliquant sur le bouton "Install" (ou "Installer"). Voir figure 6.3 Afin que les fonctionnalités s'affichent, vous devez redémarrer Thonny. Pour cela, refermez-le simplement et rouvrez-le. À la fin de cette étape-ci, vous avez installé notre extension.

6.2 Vue des variables globales et locales

Maintenant que notre extension a été installée, la première vue que nous allons vous présenter permet de visualiser les variables locales et globales dans une même fenêtre.

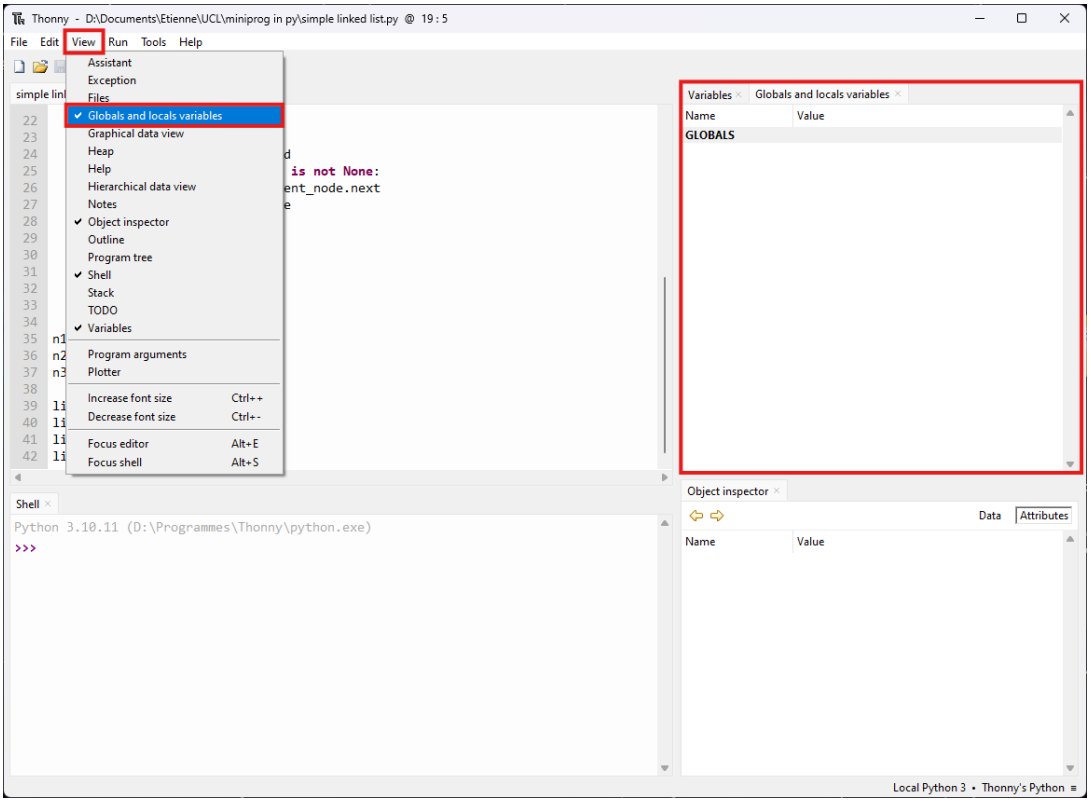


FIGURE 6.4 – Ouverture de la vue des variables globales et locales

Pour afficher la vue des variables locales et globales, allez dans l'onglet "Views" (ou "Affichage") situé en haut à gauche et sélectionnez la vue "Globals and locals variables". Une nouvelle fenêtre s'ouvre sur la droite où apparaîtront les variables globales et locales. Voir figure 6.4

À partir de maintenant, nous allons utiliser un exemple de programme codé en Python adapté à un niveau débutant. Ce programme contient deux classes. La première est une classe "Node" qui attribue à un objet de celle-ci une donnée (appelée "data") et la possibilité d'avoir un lien vers un autre objet "Node" (appelé "next"). La seconde est la classe "LinkedList" correspondant à des objets ayant un attribut "head" qui est lié à un objet "Node". Cette classe permet de lier entre eux des "Node" par leurs attributs "next" et ensuite de les parcourir. Au sein de ce programme, nous créons 3 objets "Node" (appelés "n1", "n2" et "n3") que nous lions les uns à la suite des autres dans un objet LinkedList (appelé "link").

Pour visualiser les variables globales créées dans ce programme dans la vue, il suffit d'exécuter le programme en appuyant sur le bouton vert, en haut à gauche. Les variables globales, correspondant aux objets créés dans le programme, apparaissent dans la vue à droite. Voir figure 6.5

Dans un souci de simplicité, nous avons grandement épuré ce qui est affiché dans la colonne "valeur" de notre vue. En effet, nous attribuions initialement la même valeur aux

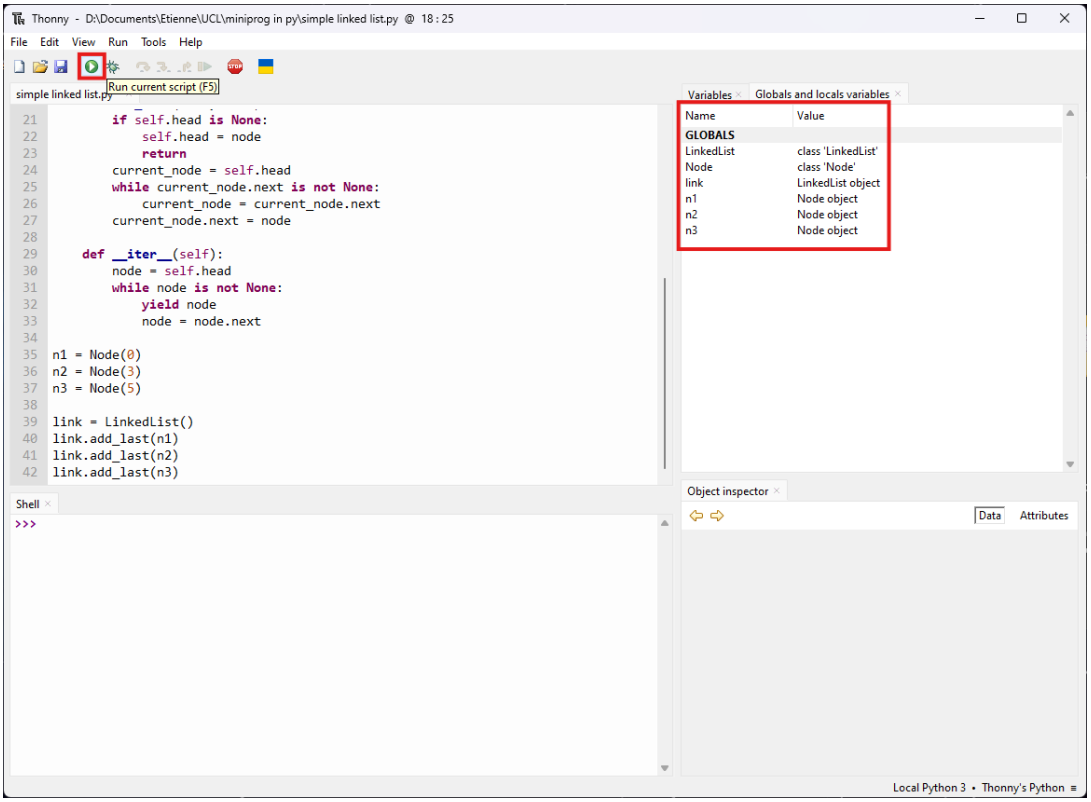


FIGURE 6.5 – Affichage des variables globales

différentes variables que celle affichée dans la fenêtre "Variables" de Thonny. Cette valeur est la représentation des variables par défaut et, donc, pour un objet "Node" réalisé par l'utilisateur : "<___main___.Node object at memory_address>" où "memory_address" correspond à l'emplacement mémoire de l'objet en question.

Cette représentation étant indigeste, et suite aux retours de nos testeurs, nous avons décidé de la simplifier et de simplement afficher "Node object" ou "LinkedList object" pour un objet complexe (ce qui les différencie des classes ou des fonctions qui seront stipulées "class" ou "function"). Pour les variables plus communes, nous affichons sa valeur si elle est simple ("1" pour l'entier 1 ou "Hello world" pour la chaîne de caractères, par exemple) ou sa représentation si elle est suffisamment explicite ("[1,2,3]" pour une liste ou "1 :"hello", 2 :"world"" pour un dictionnaire par exemple). Comme discuté dans la section 5.3, cette représentation simplifiée va rester commune, à quelques détails près dans notre extension.

Tout l'intérêt de notre outil est de travailler de concert avec le débogueur de Thonny qui est particulièrement agréable et adapté pour les débutants. Ainsi, notre vue des variables globales et locales peut être utilisée en même temps que ce dernier. Pour ce faire, appuyez sur le bouton de débogage en haut à gauche et naviguez au travers de votre programme à l'aide des trois flèches se situant à côté. En rentrant dans une instance, vous verrez apparaître des variables locales à celle-ci. Sur notre vue, elles sont distinguées des variables globales par le titre de la section correspondante. Voir figure 6.6.

Notre vue des variables globales et locales fonctionne étroitement avec l'inspecteur d'objet de Thonny. Ainsi, il est possible de cliquer sur la ligne correspondant à une variable

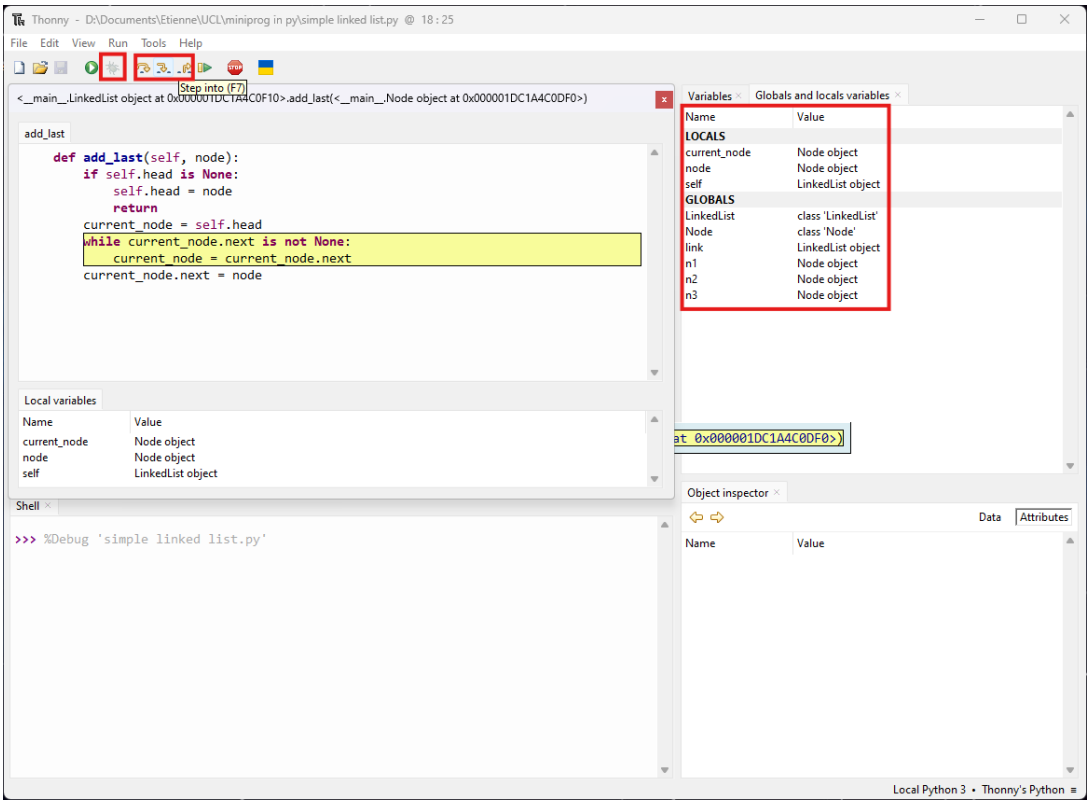


FIGURE 6.6 – Affichage des variables locales et globales

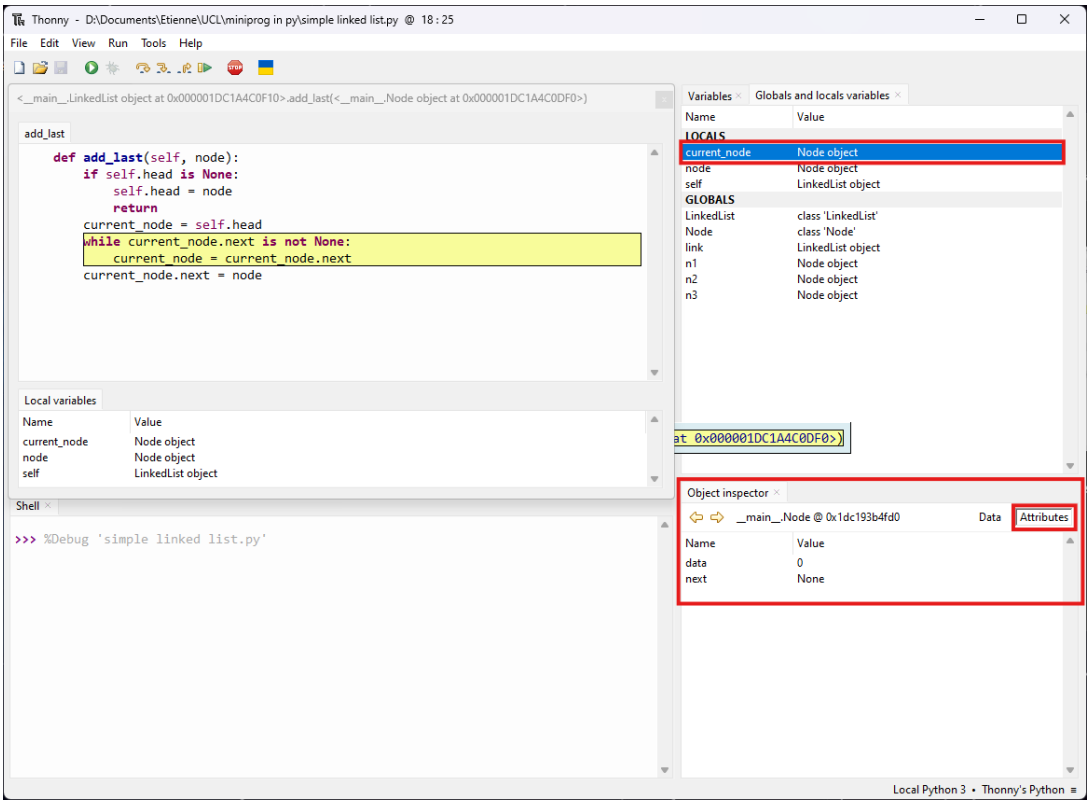


FIGURE 6.7 – Utilisation de la vue des variables locales et globales avec l'inspecteur d'objet

afin d'en distinguer les détails des attributs et l'adresse mémoire où elle est stockée. Il faut pour cela avoir la vue "Object inspector" ouverte et sélectionner la variable à explorer dans la vue des variables locales et globales. Ensuite, pour observer les attributs, il suffit de cliquer sur le bouton "Attributes", en haut à droite de l'inspecteur d'objet. Voir figure 6.7

6.3 Vue hiérarchique

La seconde vue que propose notre extension permet de faire des liens entre toutes les variables et leurs attributs. Ainsi, c'est le premier outil de visualisation à proprement parler.

Pour pouvoir l'utiliser, il vous faut, dans l'onglet "Views" (ou "Affichage") sélectionner "Hierarchical data view". La vue hiérarchique apparaîtra dans la fenêtre en bas à gauche. Voir figure 6.8

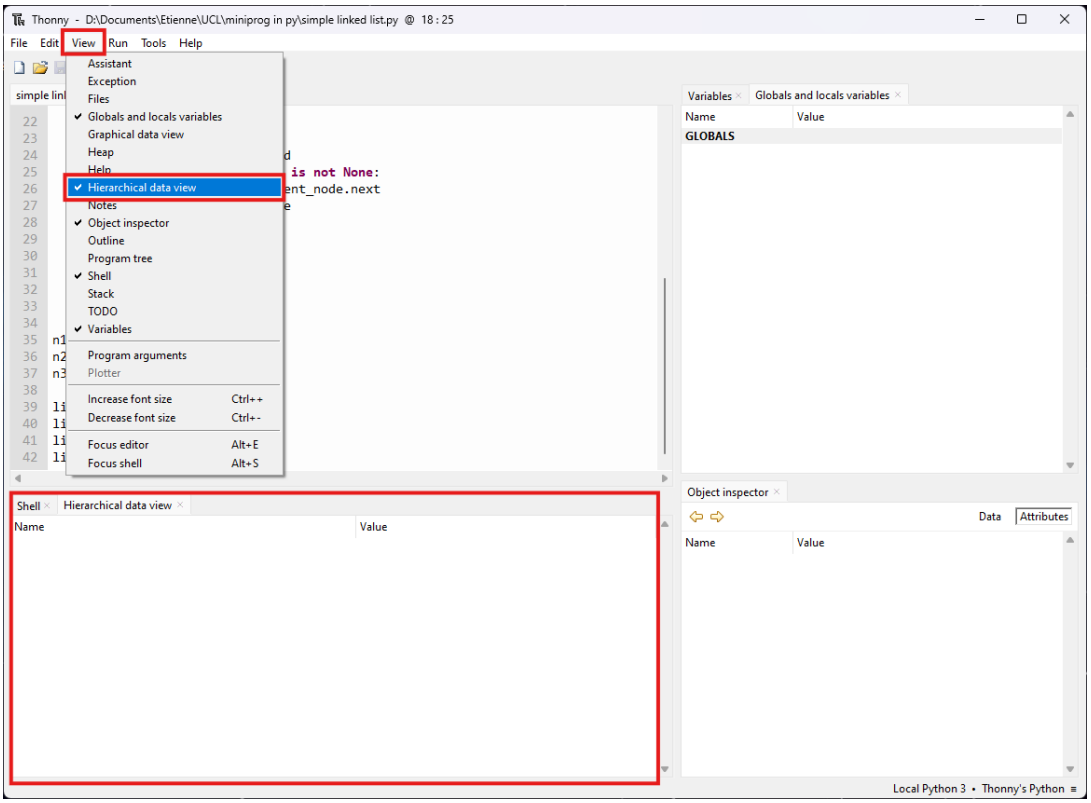


FIGURE 6.8 – Ouverture de la vue hiérarchique

Tout comme avec la vue des variables locales et globales, nous allons présenter la vue hiérarchique du programme simple créant deux objets "Node" et un objet "LinkedList" les liant l'un à l'autre.

La première façon d'utiliser notre vue hiérarchique consiste simplement à exécuter le programme voulu. Pour cela, il suffit d'appuyer sur le bouton vert en haut à gauche avec la vue hiérarchique ouverte. Voir figure 6.9

Comme présenté dans la section 4, la vue hiérarchique propose une représentation des variables simplifiée qui diffère un peu de celle de la vue des variables globales et locales. Cette représentation rajoute un numéro servant d'identifiant aux objets afin de

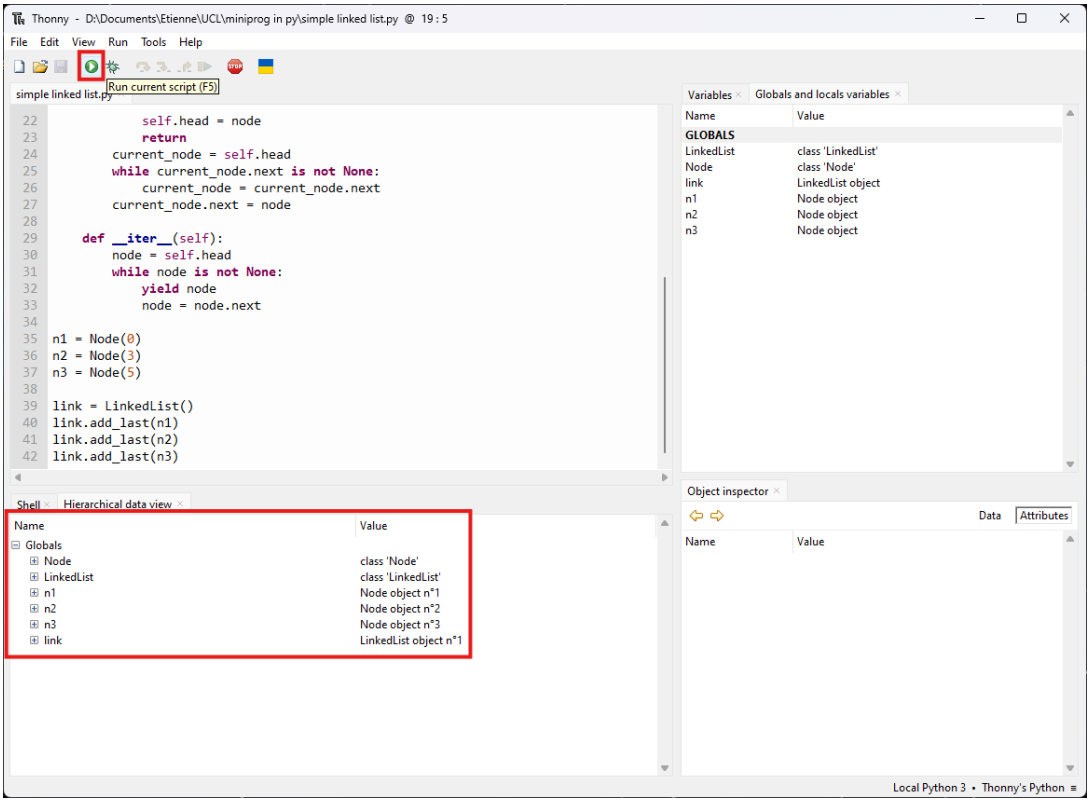


FIGURE 6.9 – Exécution complète du programme avec la vue hiérarchique

comprendre plus facilement les correspondances et les liens entre ceux-ci. En effet, grâce à ce numéro, tout utilisateur comprendra facilement quel objet fait référence à quel autre car il l'identifiera simplement par un nombre commun.

Pour donner un exemple, nous allons parcourir la vue hiérarchique en lien avec l'outil de débogage de Thonny. Pour cela, il vous suffit de cliquer sur le bouton correspondant au débogueur en haut à gauche, à côté du bouton d'exécution. Vous pouvez ensuite avancer étape par étape dans votre programme à l'aide des flèches adjacentes. Lorsque vous rentrez dans une instance du programme, la section correspondant aux variables locales de cette instance apparaîtra dans la vue. Voir figure 6.10

Tout comme dans le reste de la vue, si une variable locale (ou un attribut d'un objet) fait référence à un objet existant comme variable globale, alors cela sera rappelé par une parenthèse incluant le nom de la variable globale à laquelle il est fait référence. Dans la figure 6.10, on peut observer que la variable locale appelée "current_node" correspond actuellement à l'objet "Node" appelé "n1" qui est une variable globale. La parenthèse permet donc de mettre en évidence les références locales (ou d'attributs) aux variables globales. Pour le reste, le numéro d'identification permet de comprendre les liens entre les objets.

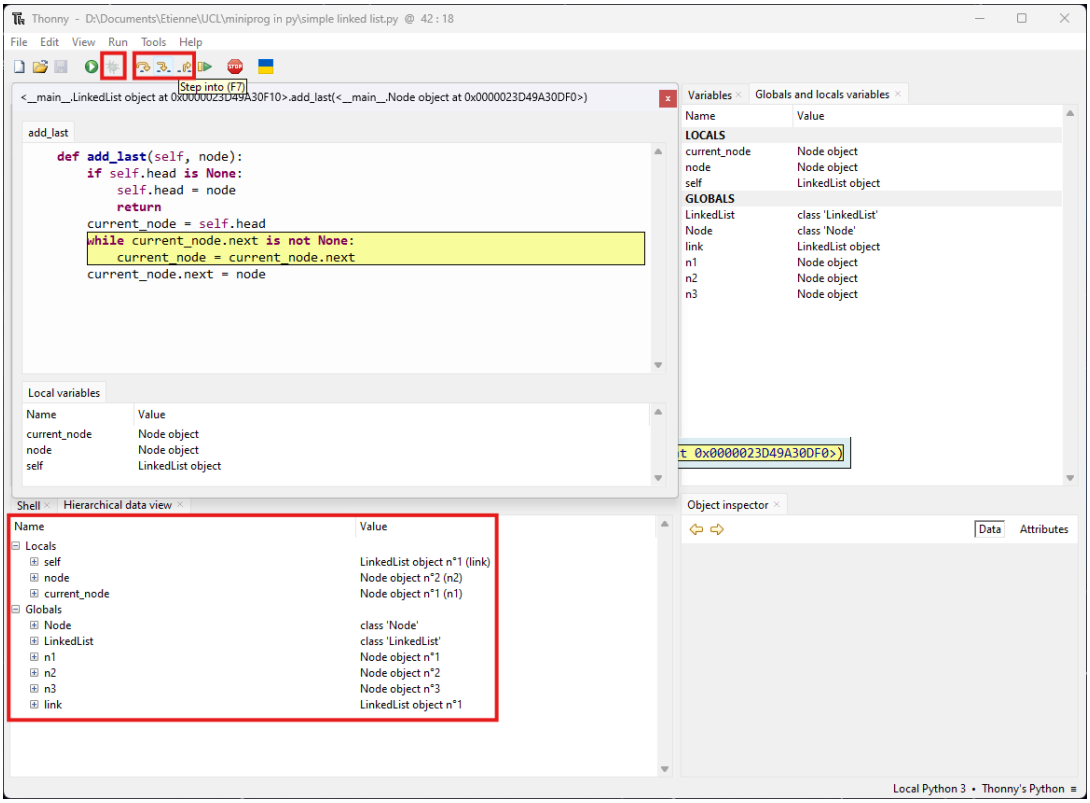


FIGURE 6.10 – Débogage du programme avec la vue hiérarchique

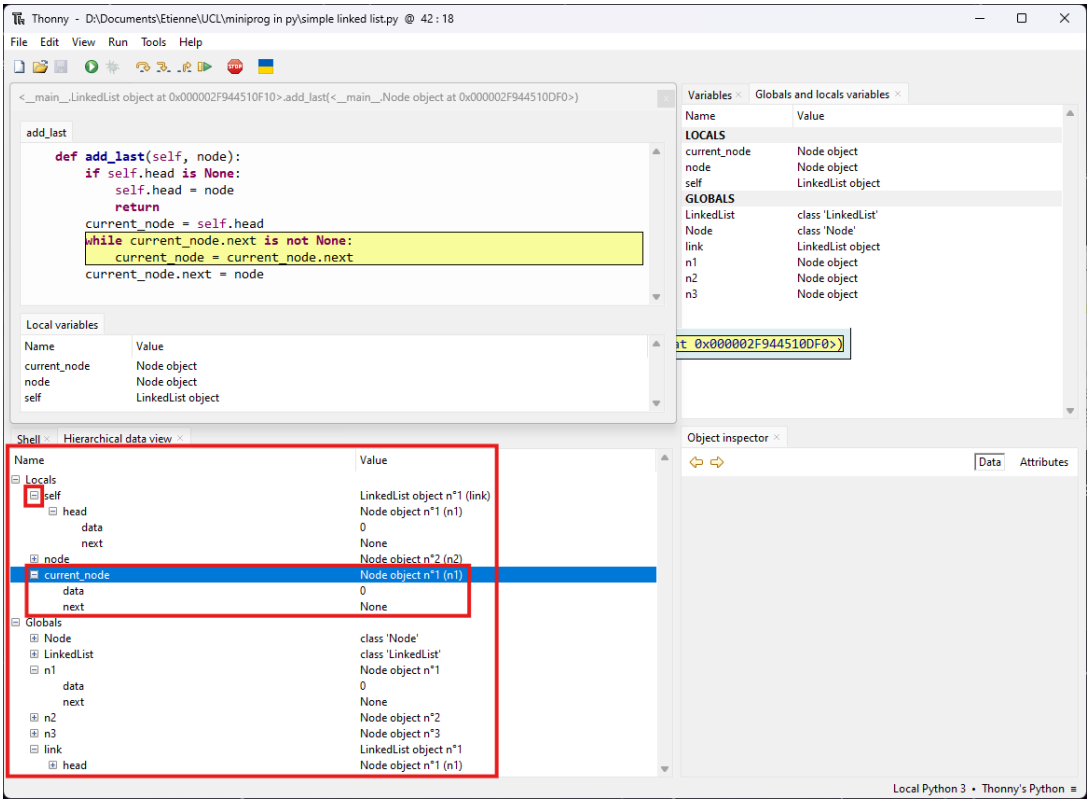


FIGURE 6.11 – Développement de la vue hiérarchique

Afin de visualiser certains objets plus en détails, il est possible de cliquer sur les petits boutons "+" et "-". En faisant cela, les différents attributs de l'objet apparaîtront. Si un de ces attributs correspond également à un objet, il est alors possible de le développer à nouveau et de visualiser les attributs de l'objet en question.
Pour bien comprendre les liens existants entre les attributs d'un objet et un autre objet, il suffit de regarder le nombre servant d'identifiant. Si le nombre correspond, c'est que la variable correspond à l'objet possédant ce nombre. Dans la figure 6.11, la variable locale "current_node" a été développée. On voit qu'elle correspond à l'objet "Node" numéro 1 qui est également la variable globale appelée "n1". On voit également que l'attribut "head" de la variable globale "link" (qui correspond d'ailleurs à la variable locale "self") est lié également à l'objet "Node" numéro 1.

Dans cas de variables importées, la vue hiérarchique propose un affichage différent. En effet, ces variables ont tendance à contenir de très nombreux attributs qui n'intéressent pas toujours l'utilisateur et à surcharger ainsi la vue hiérarchique. Pour résoudre ce problème, nous n'affichons aucun attribut et suggérons cependant à l'utilisateur de les visualiser dans l'inspecteur d'objet.

Pour illustrer nos propos, nous allons utiliser un petit programme Python qui initie simplement une liste de la librairie Numpy. Lorsque nous développons cette liste pour visualiser ses attributs, un message est écrit à la place suggérant à l'utilisateur d'en observer les détails dans l'inspecteur d'objet. Voir figure 6.12

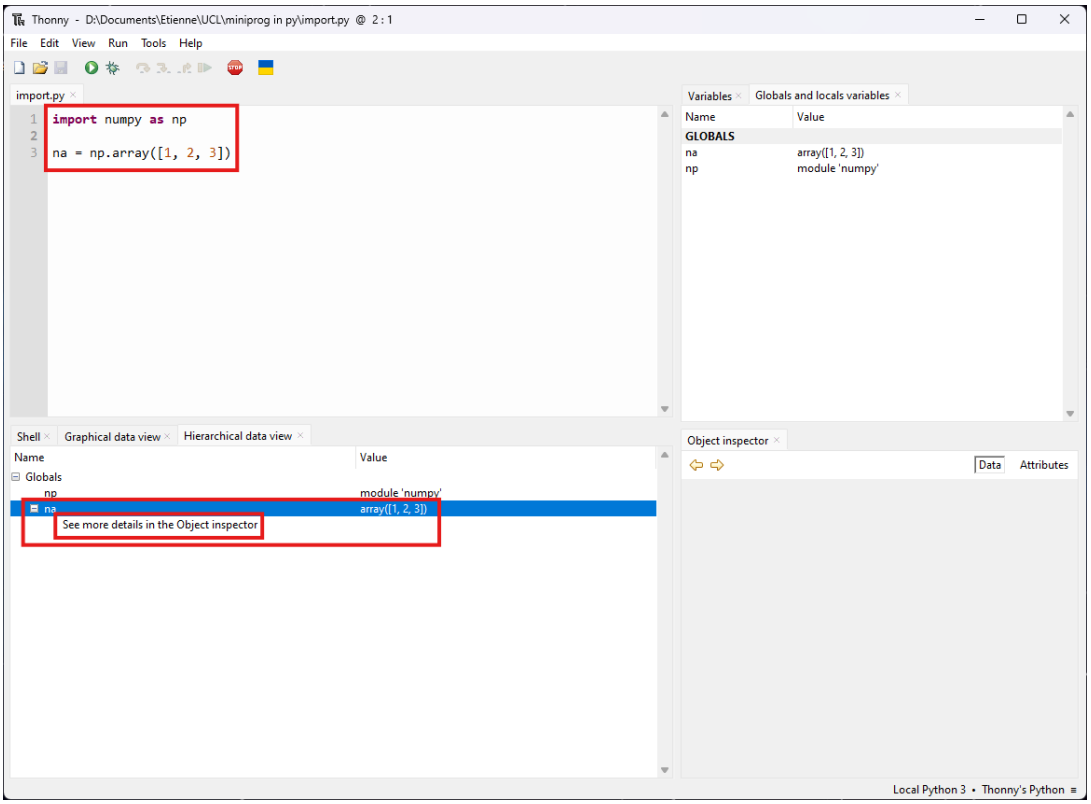


FIGURE 6.12 – Les variables importées dans la vue hiérarchique

Pour visualiser les attributs de la variable importée, il suffit de cliquer sur celle-ci dans la vue des variables locales et globales avec l'inspecteur d'objet ouvert. En faisant cela,

elle apparaîtra dans ce dernier et il suffira alors de cliquer sur le bouton "Attributes" (ou "Attributs") pour afficher l'ensemble des attributs et pouvoir les parcourir. Voir figure 6.13

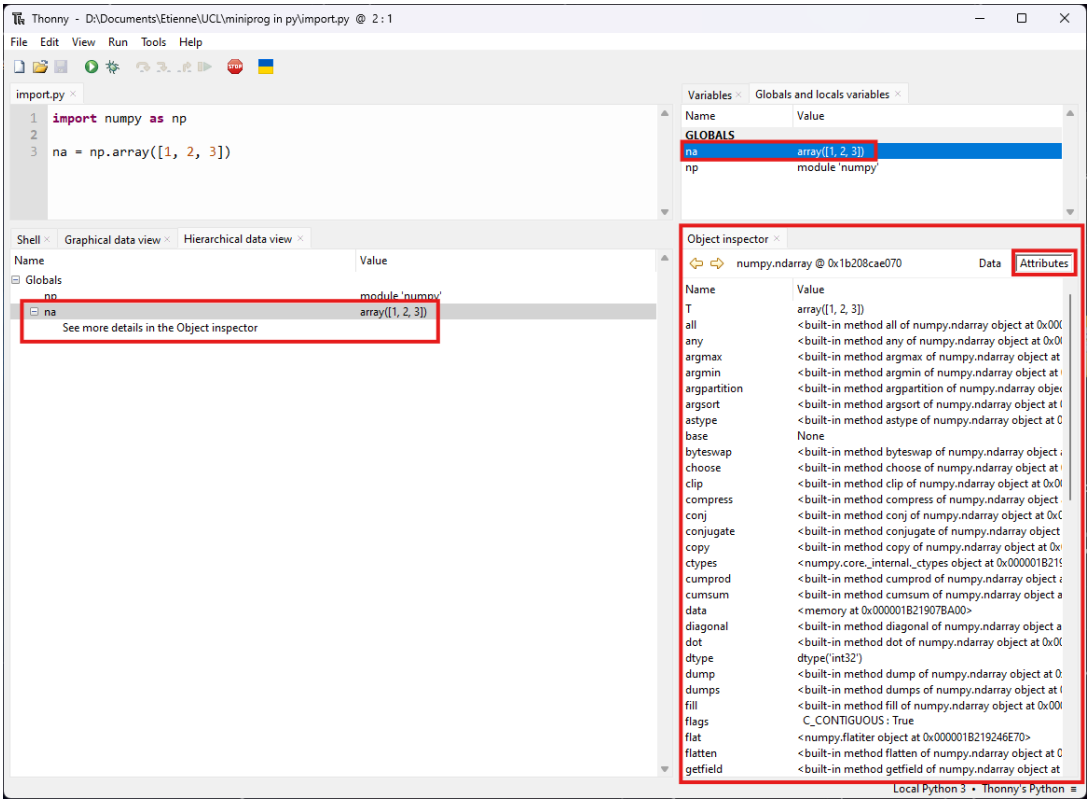


FIGURE 6.13 – Visualiser une variable importée dans l'inspecteur d'objet

6.4 Utilisation de la vue graphique

La vue graphique est l'outil de visualisation mettant le mieux en évidence les liens entre les différents objets et variables du programme. En effet, chaque objet sera représenté par une boîte contenant les différents attributs de l'objet et les liens entre ces objets seront visuellement représentés par des flèches. Il est ainsi beaucoup plus simple et visuel de distinguer les correspondances à un même objet que dans la vue hiérarchique où il faut vérifier les numéros d'identification.

Pour ouvrir la vue, il faut à nouveau aller dans l'onglet "Views" (ou "Affichage") et sélectionner "Graphical data view". La vue s'ouvre alors en bas à gauche, au même endroit que la vue hiérarchique. Voir figure 6.14

Comme pour les vues hiérarchiques et des variables globales et locales, on peut simplement afficher les variables globales en exécutant intégralement le programme à l'aide du bouton vert, en haut à droite. Voir figure 6.15

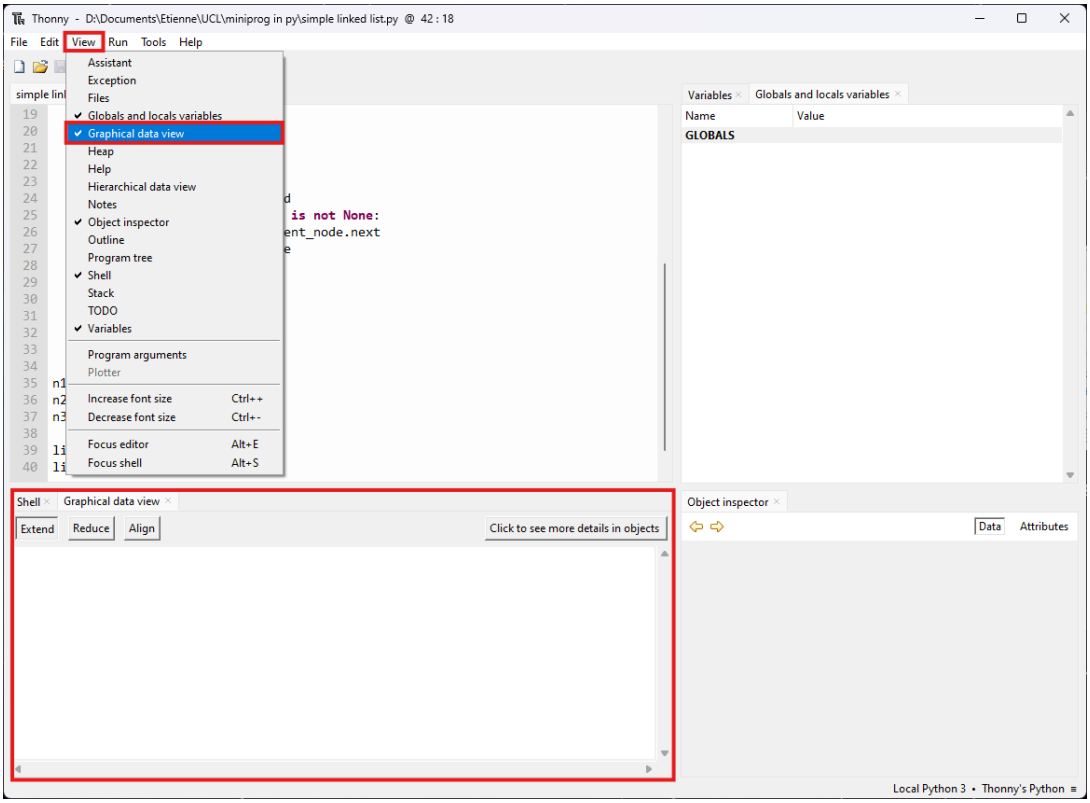


FIGURE 6.14 – Ouverture de la vue graphique

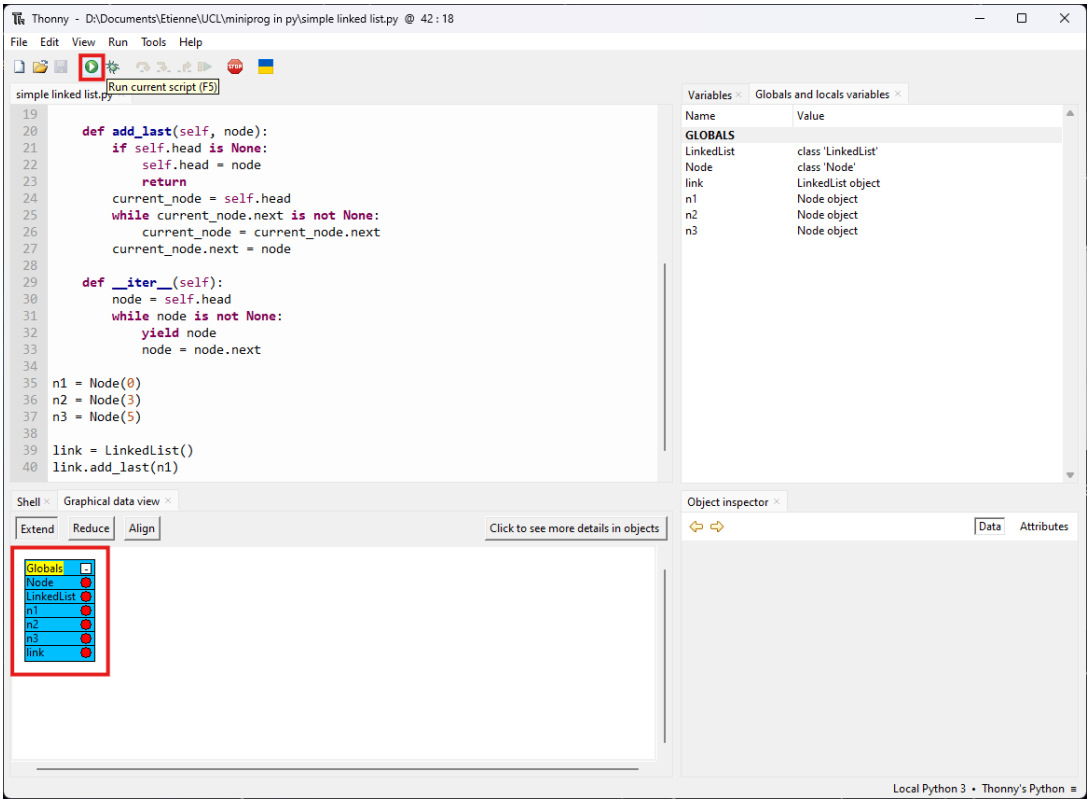


FIGURE 6.15 – Exécution complète du programme avec la vue graphique

Cependant, comme les autres vues, la vue graphique présente le plus de fonctionnalités et d'attraits lorsqu'elle est utilisée avec le débogueur de Thonny. Pour cela, il faut à nouveau appuyer sur le bouton en haut à gauche, à côté du bouton d'exécution, et naviguer à travers le programme à l'aide des flèches adjacentes. Voir figure 6.16

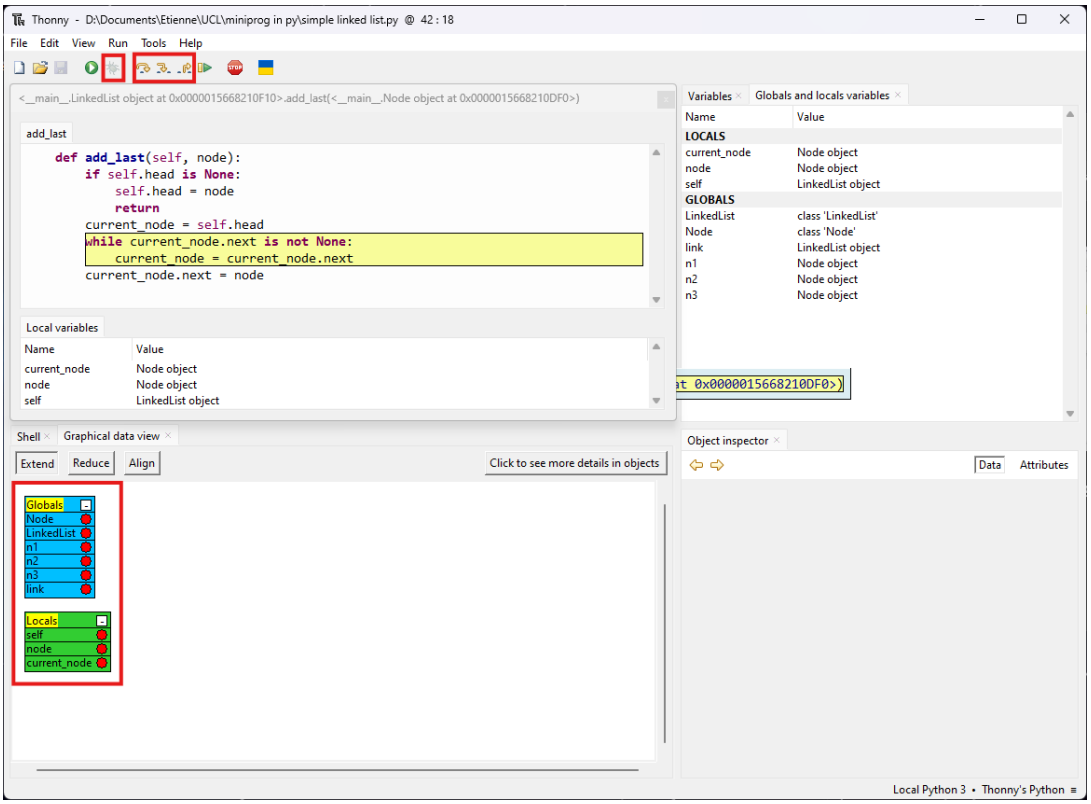


FIGURE 6.16 – Débogage du programme avec la vue graphique

Les variables globales et locales du programme apparaîtront alors dans les sections correspondantes. En bleu pour les variables globales et en vert pour les variables locales.

Afin de visualiser à quoi correspondent les différentes variables, il faut appuyer sur les petites pastilles rouges en face du nom des variables. Ces pastilles deviendront alors vertes pour indiquer que l'objet est affiché. Une flèche apparaîtra ensuite pour indiquer la correspondance entre le nom de la variable ou de l'attribut et l'objet auquel il se rapporte. Lorsque plusieurs variables ou attributs se rapportent à un même objet, les différentes flèches pointeront vers la même boîte correspondant à celui-ci.

En reprenant l'exemple du programme pour débutant avec les objets "Node" et "LinkedList", après avoir parcouru le code à l'aide du débogueur de Thonny et être entrés dans une instance locale, nous pouvons développer les différentes variables. Dans la figure 6.17, nous allons mettre en évidence les différentes variables liées à l'objet "Node" correspondant à la variable globale "n1".

À nouveau, la représentation choisie pour identifier les différents objets et variables a été simplifiée et est légèrement différente de celles employées dans les vues hiérarchiques et des variables globales et locales. Les variables sont ici uniquement identifiées par la ligne surlignée indiquant leur type dans le cas des objets (par exemple "Node" ou "LinkedList"),

et également pour les structures de données natives à Python (telles que "list", "dict" ou "set"). Les autres variables ne devant pas être développées, car n'ayant pas d'attributs ou d'éléments, ont, sur la même ligne, leur nom égalisé à leur valeur (par exemple "i = 5" ou "c = 'Hello world'"). Avec ce système, il est simple de distinguer les objets et structures des simples variables à valeur fixe. De plus, les informations sont épurées, encombrent le moins possible la vue et évitent les redondances avec le contenu des boîtes des objets.

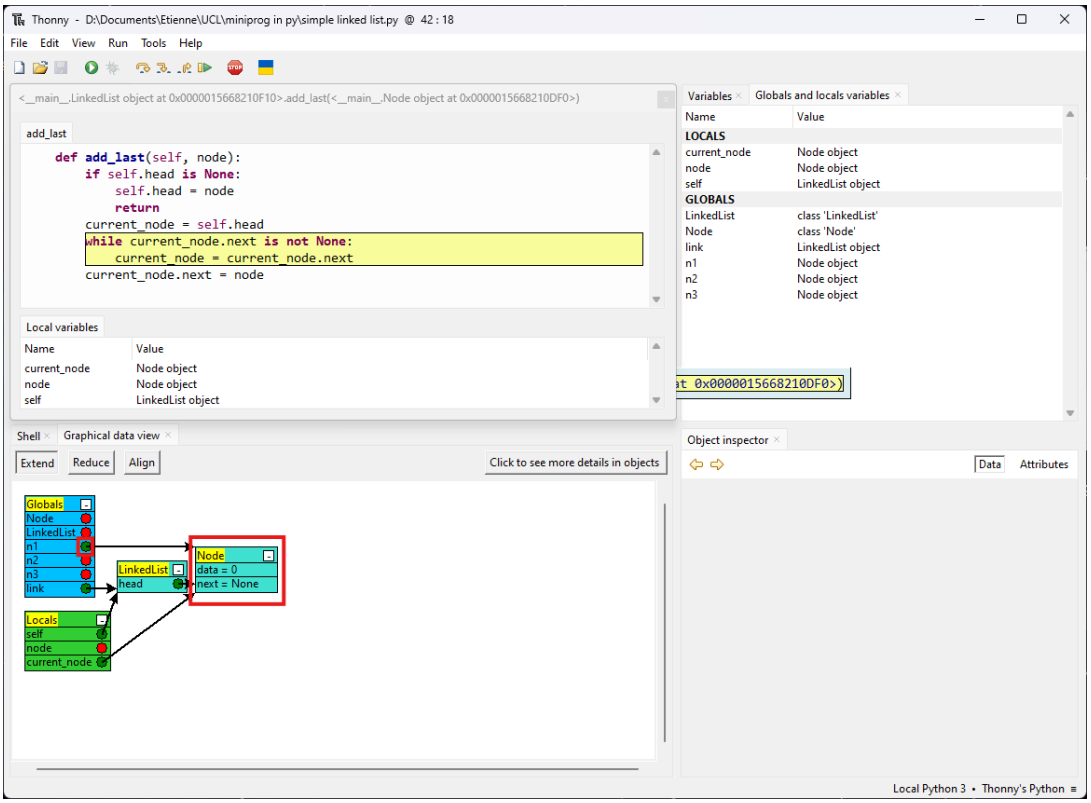


FIGURE 6.17 – Développement de la vue graphique

À l'aide des petits boutons "-", il est possible de réduire les boîtes des objets pour ne plus afficher que la ligne surlignée représentant l'objet. Cela permet de réduire la taille des boîtes et ainsi gagner en lisibilité et en espace. Un clic sur le bouton "Reduce", en haut à gauche de la vue graphique, permet de réduire la taille de toutes les boîtes en même temps. Voir figure 6.18. Il est alors possible d'étendre une boîte réduite en appuyant sur le bouton "+". Il est également possible d'étendre à nouveau toutes les boîtes d'un coup en cliquant sur le bouton "Extend".

Il arrive parfois qu'en développant des objets, en les étendant ou en les réduisant, ils se chevauchent ou que des espaces se créent. Afin de remettre toutes les boîtes alignées de façon compacte, il est possible de cliquer sur le bouton "Align". Cela aura pour effet de remettre chaque boîte dans une position par défaut relativement collée et alignée et sans chevauchement. A partir de là, il devient plus facile de replacer les boîtes en les glissant à l'aide de la souris. Voir figure 6.19

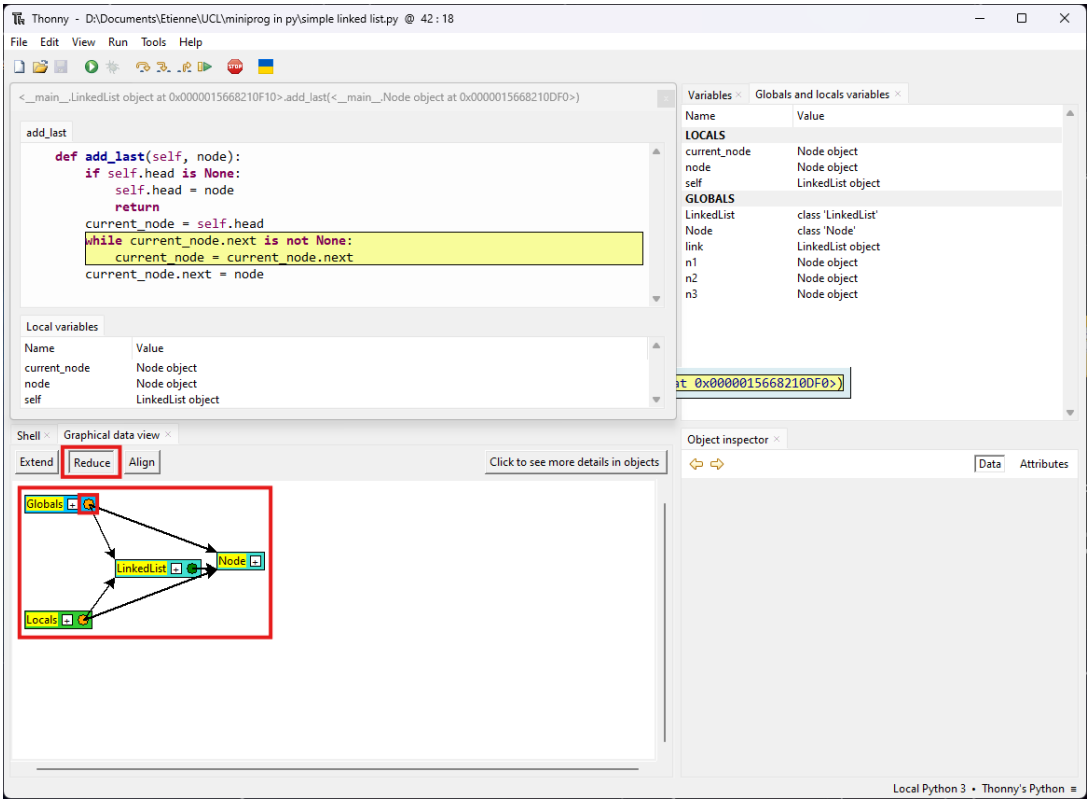


FIGURE 6.18 – Réduction des boîtes de la vue graphique

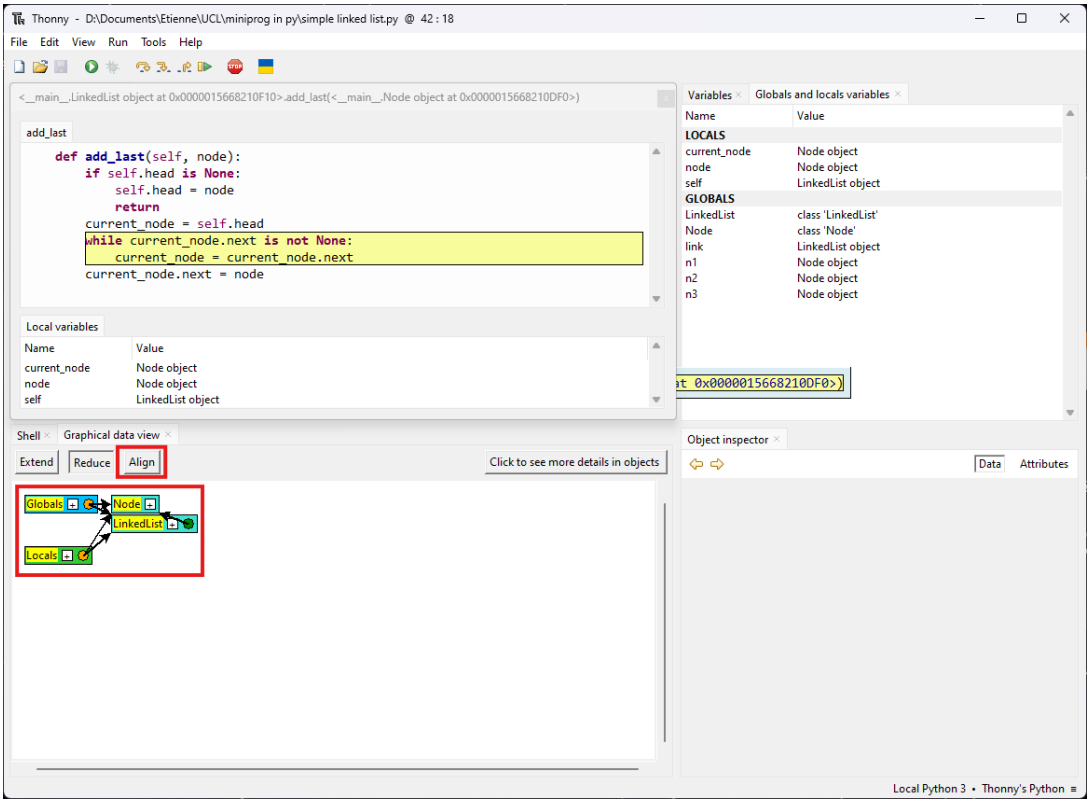


FIGURE 6.19 – Alignement des boîtes de la vue graphique

De la même façon qu'avec la vue hiérarchique, les variables importées sont traitées différemment des autres variables dans la vue graphique. En reprenant l'exemple de code utilisé avec la vue hiérarchique et important les listes de la librairie Numpy, la vue graphique affichera elle aussi un message suggérant à l'utilisateur de visualiser les attributs de la variable importée dans l'inspecteur d'objet, au lieu de les afficher directement. À nouveau, cela est fait pour simplifier la vue et pour éviter les encombrements avec de nombreux attributs qui ne sont pas toujours intéressants. Voir figure 6.20.

Tout comme dans la vue hiérarchique, il est possible d'observer les différents attributs de la variable importée en la sélectionnant dans la vue des variables locales et globales avec l'inspecteur d'objet ouvert. Ce dernier affiche alors les détails de la variable en question, dont ses attributs si l'on clique sur le bouton "Attributes" (ou "Attributs"), en haut à droite de sa fenêtre. Voir figure 6.20.

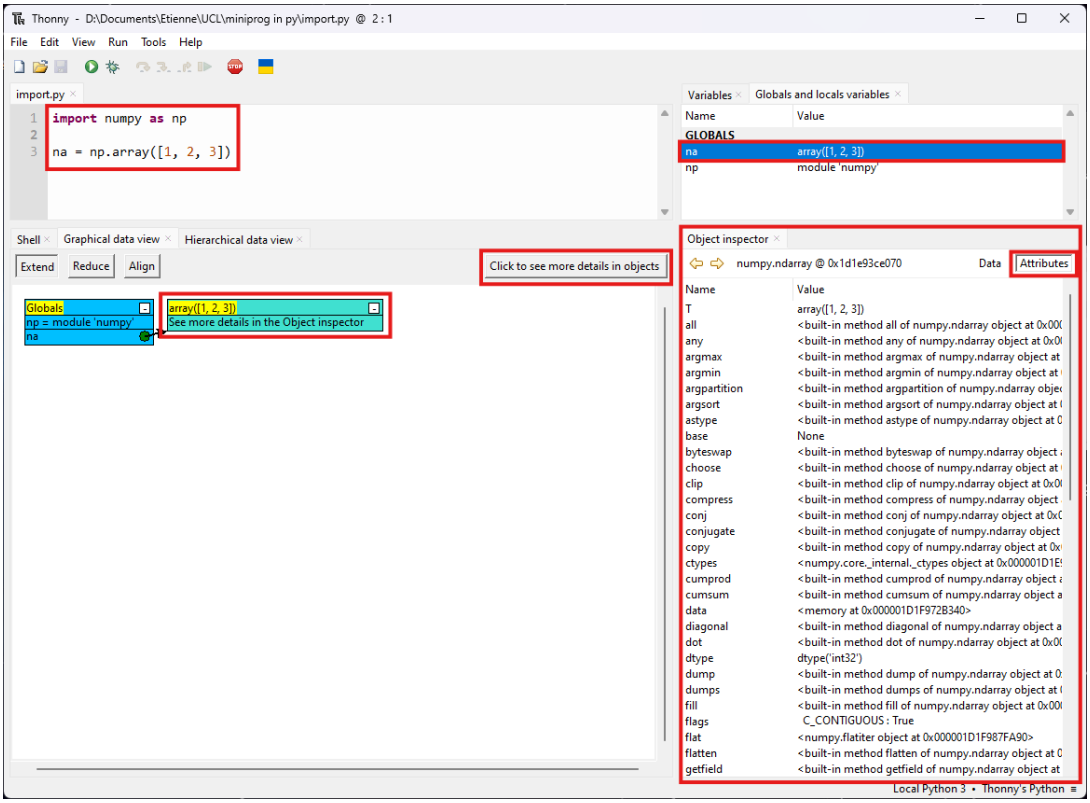


FIGURE 6.20 – Visualisation de variables importées dans la vue graphique

Contrairement à la vue hiérarchique, il est ici possible de choisir de visualiser directement dans la vue globale les différents attributs des variables importées. Pour réaliser cela, il faut tout d'abord cliquer sur le bouton "Click to see more details in objects" en haut à droite de la vue graphique (voir figure 6.20). Le message de ce dernier demandera alors d'éxécuter à nouveau le programme pour visualiser tous les objets et tous leurs attributs. Cela se fait simplement en appuyant à nouveau sur le bouton d'exécution, en haut à droite, ou en déboguant le programme à l'aide du bouton juste à côté. Voir figure 6.21.

Lorsque cela est fait, la vue graphique affichera en partie les attributs de la variable importée. En effet, nous avons fait le choix, pour éviter de surcharger la vue, de n'afficher que 10 attributs ou éléments à la fois. Il est cependant possible à l'utilisateur de cliquer

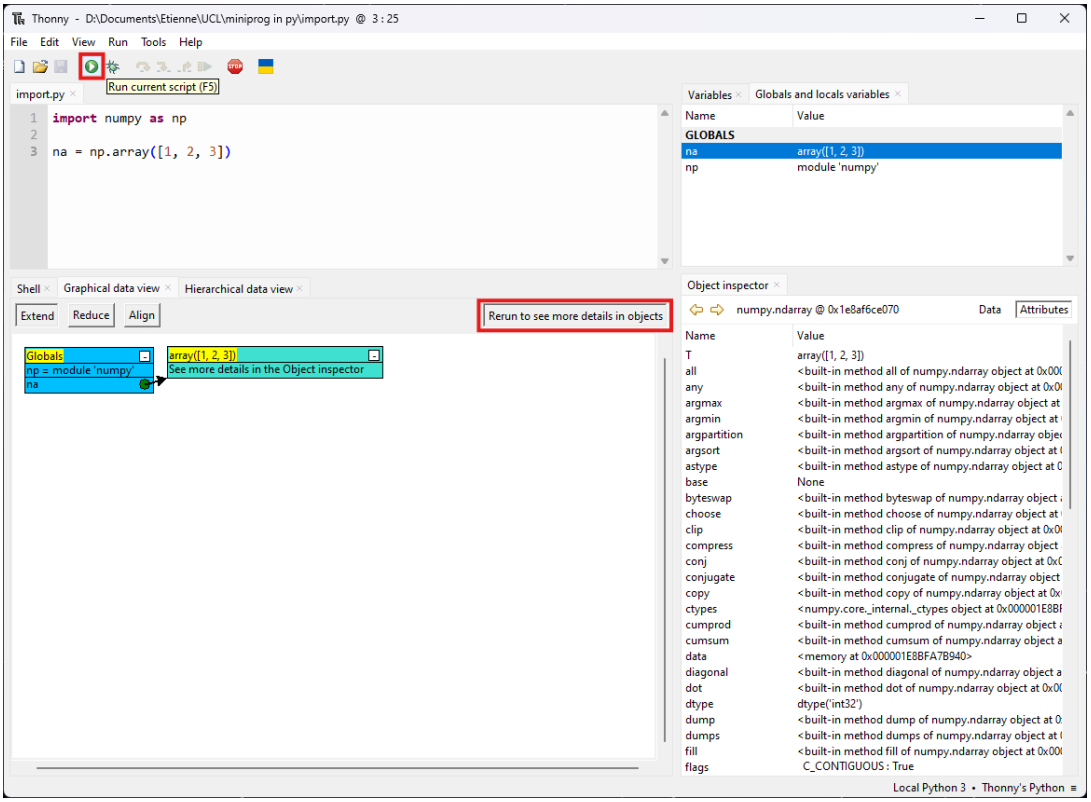


FIGURE 6.21 – Visualisation des attributs des variables importées dans la vue graphique

sur le bouton "Click here to see more" en bleu clair à la fin d'une boîte pour afficher les 10 attributs ou éléments suivants. Voir figure 6.22 et 6.23. Il est bien sûr possible de cliquer plusieurs fois sur le bouton pour visualiser les 10 attributs ou éléments suivants, etc, jusqu'à arriver aux derniers attribut ou éléments.

Il est possible également de revenir en arrière et de vouloir cacher les attributs encombrants des variables importées. Pour cela, il suffit de cliquer à nouveau sur le bouton en haut à droite de la vue graphique qui s'intitule maintenant "Click to see less details in objects". Voir figure 6.23. Il faudra à nouveau exécuter le programme ou le déboguer pour afficher les objets de façon simplifiée.

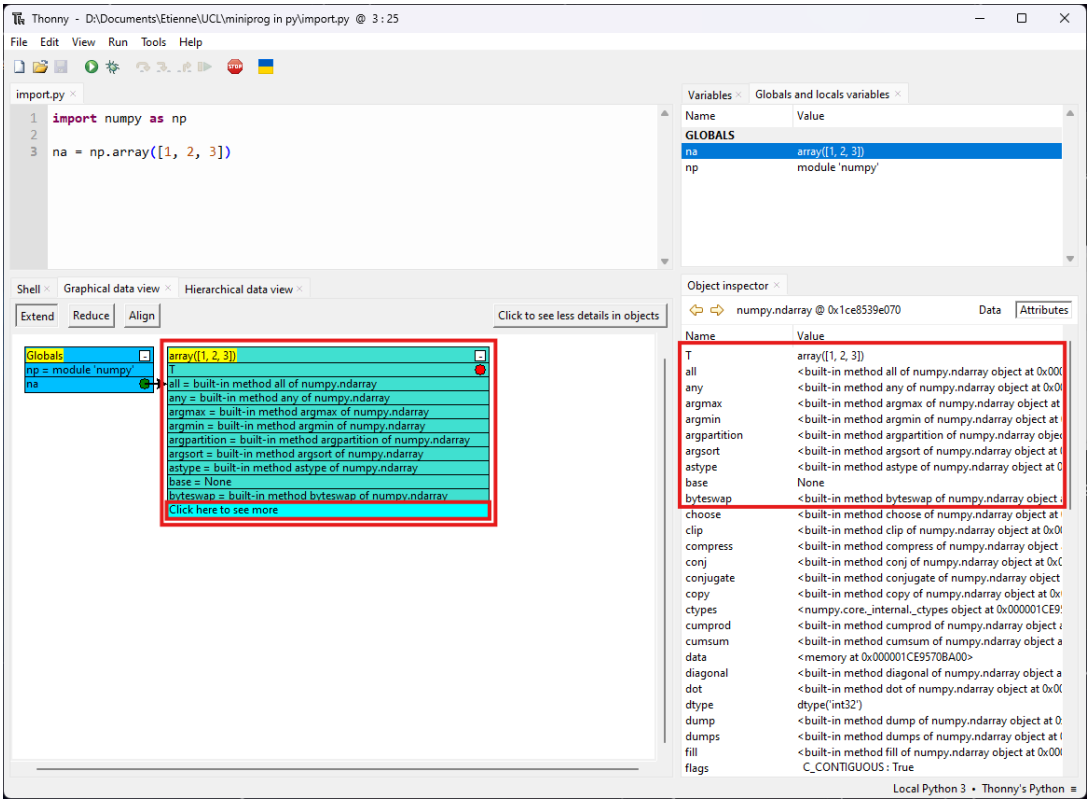


FIGURE 6.22 – Visualisation des 10 premiers attributs dans la vue graphique

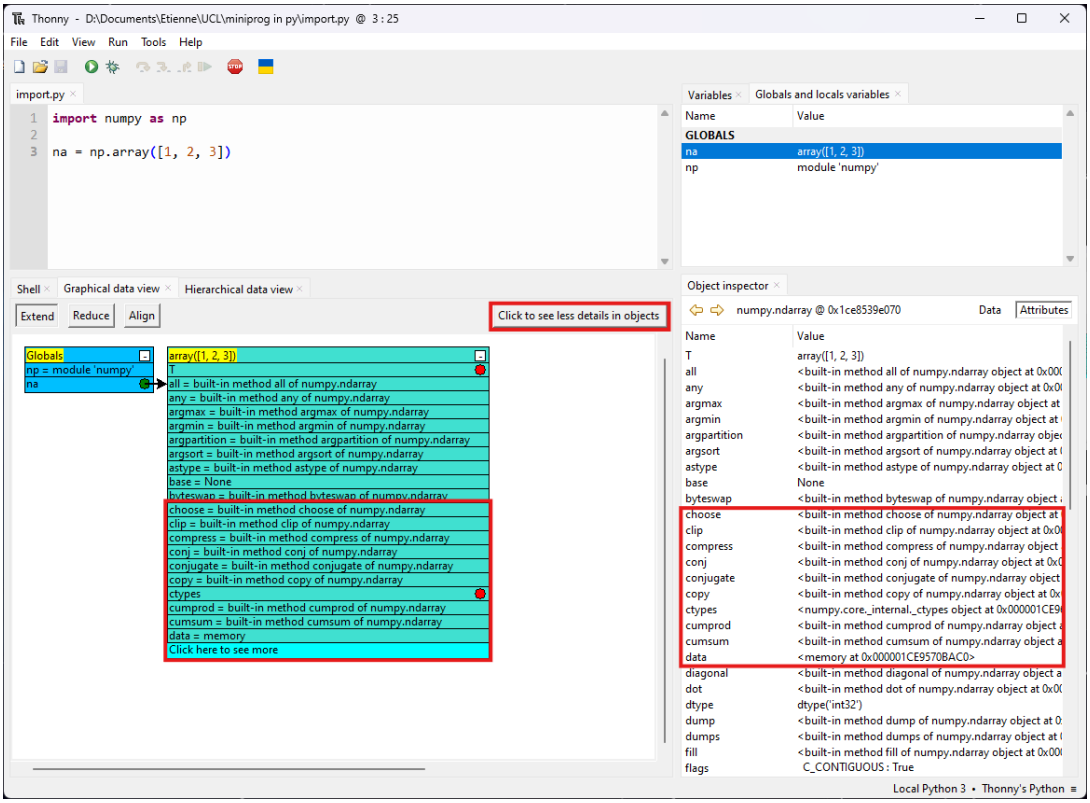


FIGURE 6.23 – Visualisation des 10 attributs suivants dans la vue graphique

Chapitre 7

Perspectives futures

L'outil de visualisation que nous avons réalisé nous parait déjà fort complet et bien construit. Il faut évidemment décider à un moment d'arrêter le développement et de rendre accessible le produit réalisé. Cependant, comme pour la majorité des outils produits, il est toujours possible de les améliorer et d'offrir des fonctionnalités supplémentaires. Nous parlons dans cette section de quelques améliorations qui nous semblent importantes et que nous n'avons pas encore réalisées car nous avons dû faire des choix.

7.1 Améliorations communes aux vues hiérarchiques et graphiques

Comme décrit dans la section "énoncé du problème" 3, il y a des limites à PythonTutor et à Thonny que nous n'avons pas réussi à corriger avec notre outil de visualisation. Il nous parait actuellement difficile de comprendre comment réussir à visualiser des outils qui vont changer la pile comme, par exemple, *thread._stderr._backend._heap* de la librairie "threading". Trouver un moyen de visualiser cela améliorerait l'outil. Détecter les fonctions avancées qui peuvent provoquer un bug et empêcher leur visualisation peut aussi être une amélioration future. Dans la même idée, nous arrivons à visualiser les structures de données *frozenset* mais nous pourrions permettre la visualisation des structures qui pourraient y être contenues.

Le code écrit dans Thonny et le contenu de la console qui y est intégrée ont tous les deux une fonctionnalité de zoom et de dézoom. Ce n'est pas le cas d'autres outils de Thonny comme, par exemple, l'inspecteur d'objet ou l'arbre du programme dont nous avons déjà parlé dans ce mémoire. C'est pour cette raison que nous n'en avons pas fait une priorité dans notre travail. Bien que ce ne soit pas prioritaire, pouvoir appliquer le zoom et le dézoom avec les outils que nous avons réalisés peut être une bonne amélioration à offrir à l'utilisateur. Après tout, il s'agit d'une fonctionnalité assez courante pour toute personne qui préfère voir sont texte en grand. De plus, le dézoom peut aider à la lisibilité en permettant à l'utilisateur de voir l'ensemble de son graphe en un coup d'œil.

Afin de permettre la visualisation des objets avec beaucoup de variables dans les vues graphique et hiérarchique, nous avons choisi une option facile en ne montrant pas immédiatement toutes les variables. Le bouton "Click here to see more" permet d'agrandir

la vue et de montrer plus de variables. Toutefois, cette manière de montrer plus de variables peut être dérangeante. Par exemple pour quelqu'un qui souhaite voir le dernier élément d'une liste de 1000 éléments, car il doit cliquer 100 fois sur le bouton et cela va créer un très grand objet sur sa vue. Il serait intéressant d'avoir des objets dont la vue ne s'agrandit pas et dans laquelle une barre de défilement permettrait de faire défiler toutes les variables d'un même objet. C'est une amélioration plus complexe où il faut éviter de confondre le défilement au sein d'un même objet et le défilement de toute la vue. De plus, pour la vue graphique, il faut trouver un moyen de visualiser les références de l'objet qui ne sont pas affichées et qui sont "cachées" par le défilement.

Une autre fonctionnalité qui pourrait permettre de retrouver ce que l'on cherche efficacement sans tout afficher serait une option de recherche, comparable aux fonctions de recherche de texte qu'il est possible de retrouver dans la plupart des navigateurs web et autres outils informatiques avec le raccourcis "ctrl+f".

Il serait utile de tester l'efficacité de notre outil de manière plus rigoureuse avec un grand échantillon d'étudiants réellement débutants. Nous pourrions observer si l'utilisation de notre outil leur a permis d'avoir de meilleurs résultats à des tests ou à leurs examens, comparé à d'autres étudiants qui ne l'auraient pas utilisé. Nous pourrions également analyser les concepts les mieux intégrés grâce à notre outil.

7.2 Amélioration spécifique à la vue hiérarchique

Pour les données et structures externes au code, comme par exemple les librairies qu'il faut importer, nous avons créé dans la vue graphique le moyen de voir ou pas leur variable interne. Cette possibilité n'est donnée que dans la vue graphique et il serait utile de la permettre aussi dans la vue hiérarchique - comme c'est déjà le cas dans PyCharm. Par exemple, il est possible de voir à la figure 7.1, que PyCharm à créé une vue spécifique pour les numpy array qui est claire et utile pour l'utilisateur qui analyserait cet objet. Cette vision offerte par PyCharm ne représente pas fidèlement les variables internes d'un numpy array qui sont enregistrées de manière beaucoup moins lisible en mémoire. Une autre option serait de faire un lien avec l'inspecteur d'objet de Thonny et lorsque l'utilisateur clique sur "See more details in the Object inspector" cela fera le lien avec l'inspecteur d'objet et affichera dans l'inspecteur d'objet l'élé-

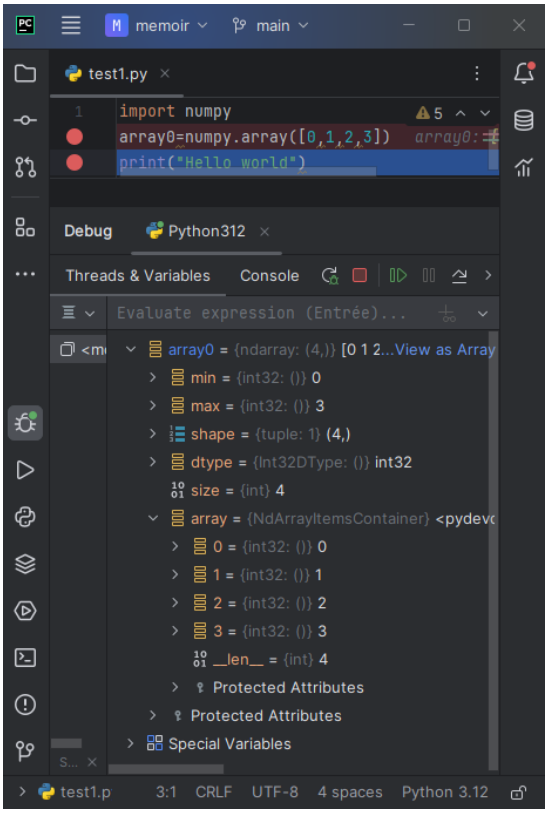


FIGURE 7.1 – Vue hiérarchique d'une numpy array par PyCharm

ment que l'utilisateur est en train d'analyser.

La vue hiérarchique manque de fonctionnalités globales comme, par exemple, l'extension ("extend") ou la réduction ("reduce") de la vue graphique. Il pourrait être intéressant de réaliser des fonctionnalités globales pour la vue hiérarchique. Celles-ci permettraient par exemple de parcourir tous les objets d'un seul coup ou de cacher la vue de tout les objets qui ont été parcourus.

7.3 Amélioration spécifique à la vue graphique

Pour les données et structures externes au code, comme les librairies qu'il faut importer, nous avons créé dans la vue graphique un mode plus simple qui ne les visualise pas. Ceci évite de distraire tout utilisateur débutant qui n'aurait pas besoin de voir ces informations. Nous avons aussi créé un mode qui affiche ces données plus avancées. Cette visualisation avancée peut permettre de comprendre et voir ce qui est contenu dans un élément externe. Cependant elle reste assez peu optimale. Réaliser une visualisation plus claire et plus pédagogique pour les librairies les plus utilisées serait une bonne amélioration, comparable à ce qui existe déjà dans PyCharm, comme expliqué précédemment. Chaque librairie fonctionne différemment et réaliser une visualisation différente pour chacune d'elles serait plus clair et pédagogique.

Nous avons réalisé une fonctionnalité "Align" qui permet de repositionner de manière claire et ordonnée tous les nœuds de la vue graphique. Ce repositionnement se fait en ne prenant en compte que les nœuds ; nous avons en effet trouvé que c'était le point le plus important et que cela offre une vue claire de la hiérarchie des nœuds. Cependant, vu que la politique de ce travail est de laisser le choix à l'utilisateur, une amélioration utile serait d'offrir à l'utilisateur la possibilité de faire un repositionnement en fonction des arêtes de façon optimale pour qu'il y ait le moins de croisements entre les flèches. Ici, permettre de faire des flèche courbes aurait du sens. Cela risque de compliquer la compréhension de la structure hiérarchique mais cette nouvelle fonctionnalité peut faciliter le suivi d'un chemin à travers le diagramme réseau, vu que les flèches se croiseront beaucoup moins.

Comme décrit dans l'un des retours des testeurs, permettre à l'utilisateur de changer les couleurs des nœuds dans la vue graphique peut être une amélioration utile à développer, même si Cette idée ne nous a pas paru essentielle. Concrètement, cela peut permettre aux daltoniens de mettre des couleurs qui leur conviennent mieux. Cela peut également servir pour réaliser un code couleur par type de données et ainsi offrir à l'utilisateur un moyen très visuel de détecter immédiatement tous les nœuds du même type. Cela peut encore offrir à l'utilisateur un repère très visuel pour retrouver rapidement un nœud en particulier qu'il a précédemment marqué. Ainsi, il y a plusieurs fonctionnalités qu'il est possible de réaliser en permettant à l'utilisateur de changer la couleur des nœuds.

Cette amélioration-ci s'écarte un peu de notre projet mais, comme nous l'avons vu, il existe plusieurs types de visualisation. La visualisation d'algorithme qui est fort liée à la pédagogie et à l'exécution du code n'existe pas encore dans Thonny. Or, elle est un

type de visualisation assez proche de la PV. Avec tous ces points réunis, il pourrait être intéressant de réaliser un outil de visualisation d'algorithme pour Thonny et de faire des liens avec la vue graphique que nous avons réalisée. En effet des liens existent, cet article [75] présente l'intérêt de lier ces deux types de visualisation et propose une approche sur la façon de procéder. Il présente l'outil DS-PITON qui fait en même temps de la PV et de la visualisation d'algorithme. Les auteurs évaluent aussi l'efficacité du produit sur des étudiants et ils arrivent à la conclusion que c'est très efficace.

Chapitre 8

Conclusion

Aux termes de ce mémoire, nous avons présenté une extension de Thonny, un environnement de développement intégré open source spécialisé pour les débutants. Notre travail apporte des capacités de visualisation des programmes Python, avec pour objectif principal de faciliter l'apprentissage de la programmation pour les débutants. Pour cela, notre outil de visualisation doit être synchronisé avec le débogueur de Thonny et être interactif.
Cette intéraction est proposée à travers trois vues distinctes : une vue des variables globales et locales, une vue hiérarchique des objets et des relations dans le programme et, enfin, une vue graphique qui représente les mêmes informations que la vue hiérarchique, mais de manière plus imagée et plus visuelle.

La création de la vue des variables globales et locales devait seulement nous aider à comprendre comment, d'une part, intégrer une vue à Thonny, d'autre part, se synchroniser au débuggeur de Thonny et, enfin, faire les appels à la mémoire de Thonny. Elle est fort identique à l'outil "Variable", déjà existant dans Thonny mais elle a la particularité de pouvoir visualiser les variable Locales, ce qui nous parait une grande amélioration par rapport à ce qui existait. C'est pourquoi nous l'avons laissée.
La vue graphique qui a été fortement inspirée de l'outil de PV PythonTutor est l'aboutissement de notre travail, cette vue permet de bien comprendre les concepts liés à la programmation orientée objet. Elle permet à tout utilisateur, en particulier les débutants, de comprendre son code et son exécution. Elle a été conçue non seulement dans un but pédagogique mais aussi pour dépasser les limites de PythonTutor. Enfin, nous voulions que ce soit une extension de Thonny, qui lui soit intégrable.
La vue hiérarchique a la même utilité que la vue graphique mais en plus épuré et en moins visuel. Elle aide l'utilisateur plus aguerri à comprendre son code et son exécution. C'est un outil moins pédagogique mais qui permet à tout utilisateur, qui sait où chercher, de trouver plus rapidement les objets qu'il veut analyser. C'est fortement inspiré de la vue hiérarchique fournie par le débogueur de PyCharm, avec l'avantage de pouvoir plus facilement détecter les boucles dans les structures de données.

À l'origine, nous voulions réaliser le même outil que PythonTutor mais en intégrant harmonieusement ses fonctionnalités dans l'environnement Thonny. Ceci rend l'analyse de l'exécution du code plus fluide, au lieu de constamment passer d'un outil à l'autre. Nos espérances ont été dépassées. Notre outil offre à nos utilisateurs plus de possibilités. Il permet, en effet, d'afficher plus de données, de visualiser des codes avec des objets

et des structures plus complexes. Il contient aussi plus de fonctionnalités permettant à la PV d'être davantage interactive. Cela offre plus de liberté à l'utilisateur et aide à la lisibilité. Les retours de nos testeurs confirment que cet outil améliore la compréhension des programmes et rend l'apprentissage plus intuitif. Par ailleurs, des changements ont été apportés suite à leurs propositions d'amélioration.

Si nos objectifs nous semblent atteints, des améliorations sont toujours possibles. Ainsi nous envisageons notamment l'enrichissement des vues hiérarchiques et graphiques, ainsi que l'ajout de fonctionnalités spécifiques à ces vues. Cela renforcerait l'accompagnement didactique offert aux utilisateurs et permettrait de mieux visualiser des structures complexes comme les librairies importées ou les *frozenset*.
Nous espérons que ce travail suscitera d'autres développements dans le domaine des environnements de développement intégrés open source ou de PV afin de rendre l'apprentissage de la programmation plus accessible et plus efficace.

Bibliographie

[1] Sue Sentance et Andrew Csizmadia. « Computing in the Curriculum : Challenges and Strategies from a Teacher's Perspective ». In : Education and Information Technologies 22 (mars 2017). doi : 10.1007/s10639-016-9482-0.

[2] Larousse. LAROUSSE : Accueil : langue française : dictionnaire : programmation. Consulté le : 08 août 2024. 2024. url : https://www.larousse.fr/dictionnaires/francais/programmation/64205.

[3] JetBrains s.r.o. État de l'écosystème des Développeurs 2023. Consulté le : 08 août 2024. 2024. url : https://www.jetbrains.com/fr-fr/lp/devecosystem-2023/.

[4] Richard J. Enbody, William F. Punch et Mark McCullen. « Python CS1 as Preparation for C++ CS2 ». In : Technical Symposium on Computer Science Education. 2009. url : https://api.semanticscholar.org/CorpusID:8508927.

[5] Philip J. Guo. « Online Python tutor : embeddable web-based program visualization for CS education ». In : Proceeding of the 44th ACM Technical Symposium on Computer Science Education. SIGCSE '13. Denver, Colorado, USA : Association for Computing Machinery, 2013, p. 579-584. isbn : 9781450318686. doi : 10.1145/2445196.2445368. url : https://doi.org/10.1145/2445196.2445368.

[6] Aivar Annamaa. Thonny - Python IDE for Beginners. 2023. url : https://thonny.org/.

[7] Christine Alvarado et al. « Performance and Use Evaluation of an Electronic Book for Introductory Python Programming ». In : 2012. url : https://api.semanticscholar.org/CorpusID:6219660.

[8] Christopher Hundhausen, Sarah Douglas et John Stasko. « A Meta-Study of Algorithm Visualization Effectiveness ». In : Journal of Visual Languages & Computing 13 (juin 2002), p. 259-290. doi : 10.1006/jvlc.2002.0237.

[9] Jeffrey Bonar et Elliot Soloway. « Preprogramming Knowledge : A Major Source of Misconceptions in Novice Programmers ». In : Hum. Comput. Interact. 1 (1985), p. 133-161. url : https://api.semanticscholar.org/CorpusID:6452139.

[10] James Fogarty. « Code and Contribution in Interactive Systems Research ». In : 2017. url : https://api.semanticscholar.org/CorpusID:32857736.

[11] Jacob O. Wobbrock et Julie A. Kientz. « Research Contributions in Human-Computer Interaction ». In : Interactions 23.3 (mai 2016), p. 38-44. issn : 1072-5520. doi : 10.1145/2907069. url : https://doi.org/10.1145/2907069.

[12] Sanja Maravić Cisar, Robert Pinter et Dragica Radosav. « Effectiveness of Program Visualization in Learning Java : a Case Study with Jeliot 3 ». In : International Journal of Computers Communications & Control 6 (2011), p. 668-680. URL : https://api.semanticscholar.org/CorpusID:58120197.

[13] Erkki Kaila et al. « Effects of Course-Long Use of a Program Visualization Tool ». In : Conferences in Research and Practice in Information Technology Series 103 (jan. 2010).

[14] Python Software Foundation. Python Language Reference, Version 3.9. 2023. URL : https://docs.python.org/3/reference/.

[15] Andreas Stefik et Susanna Siebert. « An Empirical Investigation into Programming Language Syntax ». In : ACM Trans. Comput. Educ. 13.4 (nov. 2013). DOI : 10.1145/2534973. URL : https://doi.org/10.1145/2534973.

[16] Zakaria Alomari et al. Comparative Studies of Six Programming Languages. 2015. arXiv : 1504.00693 [cs.PL]. URL : https://arxiv.org/abs/1504.00693.

[17] Python Software Foundation. Documentation Python 3.12.4 : Histoire et licence. 2024. URL : https://docs.python.org/fr/3/license.html.

[18] Python Software Foundation. The Python Standard Library. Consulté le : 08 août 2024. 2024. URL : https://docs.python.org/3/library/index.html.

[19] Ethan Bommarito et Michael James Bommarito. « An Empirical Analysis of the Python Package Index (PyPI) ». In : Software Engineering eJournal (2019). URL : https://api.semanticscholar.org/CorpusID:198899167.

[20] Mark Summerfield. A Complete Introduction to the Python Language. Addison-Wesley Professional, 2009.

[21] Fernando Pérez, Brian E. Granger et John D. Hunter. « Python : An Ecosystem for Scientific Computing ». In : Computing in Science & Engineering 13.2 (2011), p. 13-21. DOI : 10.1109/MCSE.2010.119.

[22] Alex Holkner et James Harland. « Evaluating the Dynamic Behaviour of Python Applications ». In : Proceedings of the Thirty-Second Australasian Conference on Computer Science. ACSC '09. Wellington, New Zealand : Australian Computer Society, Inc., 2009, p. 19-28. ISBN : 9781920682729.

[23] Python Software Foundation. Tkinter - Python Interface to Tcl/Tk. Available online : https://docs.python.org/3/library/tk.html. 2024.

[24] Rene Lehtma. « Visual Aids for Programming in the Thonny IDE ». University of Tartu, 2017. URL : https://core.ac.uk/download/pdf/83597620.pdf.

[25] John E. Grayson. Python and Tkinter Programming. Manning Publications, 1999.

[26] Burkhard Meier. Python GUI Programming Cookbook - Third Edition. Packt Publishing Ltd, 2019.

[27] NetworkX Developers. NetworkX : Network Analysis in Python. Accessed : 2024-06-20. 2023. URL : https://networkx.org.

[28] Aric A. Hagberg, Daniel A. Schult et Pieter J. Swart. « Exploring Network Structure, Dynamics, and Function using NetworkX ». In : Proceedings of the 7th Python in Science Conference. Sous la dir. de Gaël Varoquaux, Travis Vaught et Jarrod Millman. Pasadena, CA, USA, 2008, p. 11-15.

[29] The Matplotlib development team. Matplotlib : Visualization with Python. Consulté le : 11 août 2024. 2024. url : https://matplotlib.org.

[30] The Graphviz Authors. Graphviz home page. Consulté le : 11 août 2024. 2024. url : https://graphviz.org.

[31] PyGraphviz developers. PyGraphviz home page. Consulté le : 11 août 2024. 2024. url : https://pygraphviz.github.io.

[32] M. Anushka S. Perera, Bernt Lie et Carlos F. Pfeiffer. « Structural Observability Analysis of Large Scale Systems Using Modelica and Python ». In : Modeling, Identification and Control 36.1 (2015), p. 53-65. doi : 10.4173/mic.2015.1.4.

[33] Bokeh contributors. Bokeh home page. Consulté le : 11 août 2024. 2024. url : https://bokeh.org.

[34] West Health Institute. pyvis : Interactive network visualizations. Consulté le : 11 août 2024. 2018. url : https://pyvis.readthedocs.io/en/latest/.

[35] Giancarlo Perrone, José Unpingco et Haw-minn Lu. « Network visualizations with Pyvis and VisJS ». In : CoRR abs/2006.04951 (2020). arXiv : 2006.04951.

[36] Plotly. Dash Python : Dash Cytoscape. Consulté le : 11 août 2024. 2024. url : https://dash.plotly.com/cytoscape.

[37] Alberto Garcia-Robledo et Mahboobeh Zangiabady. « Dash Sylvereye : A Python Library for Dashboard-Driven Visualization of Large Street Networks ». In : IEEE Access 11 (2023), p. 121142-121161. doi : 10.1109/ACCESS.2023.3327008.

[38] Alar Leemet. « Omniscient Debugger for Thonny Integrated Development Environment ». University of Tartu, 2019. url : https://dspace.ut.ee/items/5865e3b6-0ce0-4908-9dae-c9f0623e3474/full.

[39] Brad A. Myers. « Visual Programming, Programming by Example, and Program Visualization : A Taxonomy ». In : Proceedings of the International Conference on Human Factors in Computing Systems. 1986. url : https://api.semanticscholar.org/CorpusID:12628167.

[40] Brad A. Myers. « Taxonomies of Visual Programming and Program Visualization ». In : Journal of Visual Languages and Computing 1 (1990), p. 97-123. url : https://api.semanticscholar.org/CorpusID:7335020.

[41] Inc. Salesforce. What Is Data Visualization ? Consulté le : 18 juillet 2024. 2024. url : https://www.tableau.com/learn/articles/data-visualization.

[42] Swimm. Visualizing Code : Top 7 Tools Compared. Consulté le : 18 juillet 2024. 2024. url : https://swimm.io/learn/code-visualization/visualizing-code-top-7-tools-compared.

[43] Steven Halim et al. « Learning Algorithms with Unified and Interactive Web-Based Visualization ». In : Olympiads in Informatics 6 (2012).

[44] Orit Hazzan, Noa Ragonis et Tami Lapidot. « Guide to Teaching Computer Science : An Activity-Based Approach ». In : Guide to Teaching Computer Science (2011). URL : https://api.semanticscholar.org/CorpusID:60723860.

[45] Kamruddin Nur et Hasan Sarwar. « Software Visualization Tools for Software Comprehension ». In : The 4th International Conference on Software, Knowledge and Applications. Août 2010.

[46] Albert N. Badre et al. « Assessing Program Visualization Systems as Instructional Aids ». In : Proceedings of the International Conference on Computers and Learning. 1992. URL : https://api.semanticscholar.org/CorpusID:7989376.

[47] Marc Eisenstadt, Blaine A. Price et J. B. Domingue. « Software Visualization as a Pedagogical Tool ». In : Instructional Science 21 (1992), p. 335-364. URL : https://api.semanticscholar.org/CorpusID:62178123.

[48] Thomas Naps et al. « Exploring the Role of Visualization and Engagement in Computer Science Education ». In : SIGCSE Bulletin 35 (mai 2003), p. 131-152.

[49] Guido Rössling et J. Ángel Velázquez-Iturbide. « Editorial : Program and Algorithm Visualization in Education ». In : TOCE 9 (juin 2009). DOI : 10.1145/1538234.1538235.

[50] Ronit Levy et Mordechai Ben-Ari. « We Work So Hard and They Don't Use It ». In : ACM SIGCSE Bulletin 39 (juin 2007), p. 246. DOI : 10.1145/1269900.1268856.

[51] Clifford A. Shaffer et al. « Algorithm Visualization : The State of the Field ». In : ACM Trans. Comput. Educ. 10.3 (août 2010). DOI : 10.1145/1821996.1821997. URL : https://doi.org/10.1145/1821996.1821997.

[52] Nuttanont Hongwarittorrn. « Effects of Program Visualization (Jeliot 3) on Students' Performance and Attitudes towards Java Programming ». In : Proceedings of the International Conference on Educational Technology. 2010. URL : https://api.semanticscholar.org/CorpusID:84840561.

[53] Oscar Karnalim et Mewati Ayub. « The Effectiveness of a Program Visualization Tool on Introductory Programming : A Case Study with PythonTutor ». In : CommIT (Communication and Information Technology) Journal 11 (déc. 2017), p. 67-76. DOI : 10.21512/commit.v11i2.3704.

[54] J. Ángel Velázquez-Iturbide, Isidoro Hernán-Losada et Maximiliano Paredes. « Evaluating the Effect of Program Visualization on Student Motivation ». In : IEEE Transactions on Education PP (jan. 2017), p. 1-8. DOI : 10.1109/TE.2017.2648781.

[55] Arnold Pears et al. « A Survey of Literature on the Teaching of Introductory Programming ». In : ACM SIGCSE Bull. 39 (2007), p. 204-223. URL : https://api.semanticscholar.org/CorpusID:1000380.

[56] D.E. Wolfgram. Creating Multimedia Presentations. Programming (Que). Que, 1994. ISBN : 9781565296671. URL : https://books.google.be/books?id=jRytCk9aJN8C.

[57] Robert Pinter, Dragica Radosav et Sanja Maravić Čisar. « Interactive Animation in Developing e-Learning Contents ». In : The 33rd International Convention MIPRO. 2010, p. 1007-1010. URL : https://api.semanticscholar.org/CorpusID:6797723.

[58] Managed by Google. Online Compiler, Visual Debugger, and AI Tutor for Python. Consulté le : 25 juillet 2024. 2024. URL : https://pythontutor.com/.

[59] Brandon Dennis et Ramyaa Ramyaa. « Comparative Analysis of Object Visualization Tools with Respect to Their Use in Education ». In : Bulletin of the Technical Committee on Learning Technology 21.4 (2021), p. 10-14.

[60] Stephen Mutua et al. « Classifying Program Visualization Tools to Facilitate Informed Choices : Teaching and Learning Computer Programming ». In : International Journal of Computer Science and Telecommunications 3 (fév. 2012), p. 42-48.

[61] Jeong Yang, Youlg Lee et Kai H. Chang. « Initial Evaluation of JaguarCode : A Web-Based Object-Oriented Programming Environment with Static and Dynamic Visualization ». In : IEEE 30th Conference on Software Engineering Education and Training. 2017, p. 152-161. DOI : 10.1109/CSEET.2017.32.

[62] Auburn University. jGRASP Home Page. Consulté le : 26 juillet 2024. 2024. URL : https://www.jgrasp.org.

[63] David R. "Chip" Kent IV. Javelina Code Coverage Tool. Consulté le : 26 juillet 2024. 2005. URL : https://javelina-cc.sourceforge.net.

[64] Demian Lessa et al. JIVE : Java Interactive Visualization Environment : About JIVE. Consulté le : 26 juillet 2024. 2022. URL : https://cse.buffalo.edu/jive/.

[65] Michael Kölling et al. BlueJ. Consulté le : 26 juillet 2024. 2024. URL : https://www.bluej.org/.

[66] Roman Bednarik, Andrés Moreno et Niko Myller. « Various Utilizations of an Open-Source Program Visualization Tool, Jeliot 3 ». In : Informatics in Education 5 (oct. 2006), p. 195-206. DOI : 10.15388/infedu.2006.15.

[67] André L. Santos. PandionJ - A Pedagogical Java Debugger for Eclipse. Consulté le : 26 juillet 2024. 2018. URL : https://pandionj.iscte-iul.pt/.

[68] Teemu Rajala et al. « VILLE : A Language-Independent Program Visualization Tool ». In : Proceedings of the 7th Koli Calling International Conference. 2007. URL : https://api.semanticscholar.org/CorpusID:58796028.

[69] Brian Dorn et al. Jeroo : A Tool for Learning Object-Oriented Programming. Consulté le : 27 juillet 2024. 2019. URL : https://www.jeroo.org.

[70] Carnegie Mellon University. Alice 3. Consulté le : 29 juillet 2024. 2020. URL : https://www.alice.org/get-alice/alice-3/.

[71] JetBrains. PyCharm : L'IDE Python pour la Science des Données. Consulté le : 27 juillet 2024. 2024. URL : https://www.jetbrains.com/fr-fr/pycharm/.

[72] jGRASP. jGRASP : Getting Started. Consulté le : 29 juillet 2024. 2010. URL : https://www.youtube.com/watch?v=DHICqIYV33k.

[73] JGRASP. jGRASP Viewer Canvas - Sorting Example. Consulté le : 29 juillet 2024. 2014. URL : https://www.youtube.com/watch?v=D-zrayZQj6w.

[74] CS111. Tkinter Color Chart. Consulté le : 07 août 2024. 2024. URL : https://cs111.wellesley.edu/archive/cs111%5C_fall14/public%5C_html/labs/lab12/tkintercolor.html.

[75] Rossevine Artha NATHASYA, Oscar KARNALIM et Mewati AYUB. « Integrating Program and Algorithm Visualisation for Learning Data Structure Implementation ». In : Egyptian Informatics Journal 20.3 (2019), p. 193-204. ISSN : 1110-8665. DOI : 10.1016/j.eij.2019.05.001. URL : https://www.sciencedirect.com/science/article/pii/S1110866518302603.

Annexes

Répertoire Github

Voici le lien vers le répertoire Github de notre outil :
`https://github.com/etrubbers/thonny-data_visualization`

Questionnaire envoyé aux testeurs

Questions sur le fonctionnement général :

En quoi trouvez-vous l'outil adapté ou non à la visualisation et au débogage des structures de données ?

Quelles sont vos idées d'améliorations à apporter à l'outil pour le rendre plus clair et plus simple d'utilisation ?

Avez-vous rencontré des bugs ou des comportements anormaux de l'outil durant son utilisation ? Si oui, lesquels ?

En quoi trouvez-vous l'outil complet ou non ? Quelles fonctionnalités ajouteriez-vous à l'outil ?

Questions spécifiques à la visualisation de données :

En quoi trouvez-vous la vue hiérarchique adaptée ou non à l'apprentissage et au débogage des structures de données ?

Comment trouvez-vous la vue graphique ?

En quoi trouvez-vous l'outil adapté ou non à visualiser de grandes listes ou des listes complexes ?

En quoi trouvez-vous l'outil adapté ou non à visualiser de grands tableaux et dictionnaires ?

En quoi trouvez-vous l'outil adapté ou non à visualiser des tuples complexes ?

En quoi trouvez-vous l'outil adapté ou non à visualiser des structures de données plus complexes issues d'imports que vous utilisez couramment ?

En quoi trouvez-vous l'outil adapté ou non à visualiser des structures de données que vous avez créées vous-même, qu'elles soient simples ou complexes ?

Questions spécifiques à l'utilisation :

En quoi trouvez-vous l'outil simple et clair d'utilisation ou trouvez-vous qu'il ne l'est pas ?

En quoi les différentes fonctionnalités (principalement d'extension, de réduction et de recentrage) vous semblent-elles utiles ou non et correctement fonctionnelles ?

En quoi trouvez-vous que les interactions avec l'outil sont de qualité ou non, qu'elles se déroulent comme souhaité et sont agréables ?

Questions spécifiques à l'aspect didactique :

En quoi trouvez-vous que l'outil apporte un plus, ou non, à la compréhension des structures de données ?

Quelles idées d'améliorations auriez-vous pour rendre l'outil plus adapté à l'apprentissage du codage ?

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**