Modèles, langages et instruments pour l'enseignement et l'apprentissage de la programmation en Python

Expérimentations en formation d'enseignants et au lycée

Sébastien Hoarau[1,2], Christophe Declercq[1,2] et Amandine Ethève[2]

[1] Laboratoire d'Informatique et de Mathématiques, Université de La Réunion
[2] IREMI de La Réunion

Résumé Après une première expérimentation avec un groupe d'enseignants d'informatique au lycée, mettant en évidence l'hétérogénéité de leurs pratiques d'explication du comportement de programmes Python à leurs élèves, nous leur proposons deux modèles mémoire adaptés chacun à un niveau de classe spécifique : le modèle d'ardoise et le modèle de références. Nous étudions alors leur appropriation de ces modèles mémoire et recueillons leurs analyses par rapport à l'adéquation de chacun de ces modèles mémoire avec les choix des constructions du langage à enseigner ainsi qu'avec les instruments d'apprentissage pouvant être utilisés. Nous présentons également les résultats d'une expérimentation menée en classe de première pour le modèle d'ardoise et en classe de terminale pour le modèle de références. Nous concluons avec les perspectives de cette étude en proposant la définition d'instruments numériques systématisant l'usage des modèles mémoire proposés afin d'en mesurer l'impact sur l'apprentissage de la programmation en Python pour de plus grandes cohortes d'élèves.

1 Introduction

Dès le début de l'introduction de l'informatique comme discipline scolaire, de nombreux auteurs se sont penchés sur les difficultés des débutants face à l'apprentissage de la programmation. En France, dans les années 80, on peut citer les travaux de Jacques Arsac [2] et ceux de Janine Rogalski [16]. En 2013, Juha Sorva [18] propose un tour d'horizon des travaux internationaux sur le sujet. Plus récemment, Exibard *et al.* [8] rappelle que cet apprentissage repose sur deux aspects : acquérir les règles syntaxiques d'un langage en même temps que la sémantique associée. Si le premier ne pose pas de difficulté particulière - l'objet d'apprentissage est clairement défini - le second semble plus délicat à appréhender. En effet, la sémantique repose sur un modèle conceptuel précis mais souvent trop complexe pour un débutant. Ce modèle conceptuel peut être simplifié en une sorte de machine abstraite, c'est ce que Du Boulay *et al.* appelle machine notionnelle [6,5]. L'objectif de cette machine est d'aider l'apprenant à comprendre le fonctionnement d'un programme et de se faire un modèle mental

aussi juste que possible, c'est-à-dire proche du modèle conceptuel. Ces différents concepts et leurs relations ont été étudiés par plusieurs auteurs [18,10] qui proposent des états de l'art complets. Un groupe de travail conduit par Fincher [9] a recensé une grande variété de machines notionnelles dont les machines *Python Computer* de Mühling [13] auxquelles nous comparerons nos propositions.

La machine notionnelle s'accompagne souvent d'une méthode de visualisation parfois appelée diagramme et qui peut être réalisée par un outil. D'après Sorva [17], cette aide à la visualisation est importante pour le débutant qui a souvent du mal à la prendre en charge lui-même. Les auteurs rappellent aussi qu'un débogueur, par sa complexité et le détail de ce qu'il montre, ne constitue pas une aide utilisable pour un débutant. Malgré l'importance d'une visualisation adaptée au modèle, elle ne doit pas s'y substituer et laisser le modèle sous-jacent implicite (c'est le cas par exemple de l'outil Python Tutor [11]). Le modèle est primordial car plus simple à mémoriser qu'une série d'exemples de visualisation, il permet à l'apprenant de réaliser ses propres visualisations sur ses propres exemples. En parallèle, les conclusions de Naps *et al.* [14], dans un contexte légèrement différent de systèmes de visualisation d'algorithmes, montrent la nécessité pour l'apprenant d'aborder la visualisation en étant actif. Cette conclusion est reprise par Sorva [17] qui analyse une vingtaine d'outils de visualisation au regard de ce critère d'engagement de l'apprenant établissant ainsi une comparaison mettant en avant l'importance de l'engagement. Le tableau résumant cette analyse montre que sur les 24 systèmes de visualisation créés entre 1983 et 2011, quatre seulement sont considérés comme encore actifs en 2013, ce qui confirme l'intérêt de définir un modèle hors tout outil de visualisation.

C'est dans ce contexte que nous nous sommes intéressés aux explications utilisées par les enseignants, utilisant ou pas une machine notionnelle explicite, avec une expérimentation décrite en première partie. Nous proposons ensuite deux modèles mémoire détaillés, le modèle d'ardoise et le modèle de références avec des instruments adaptés et des choix de langage cohérents avec chaque modèle mémoire. Nous présentons enfin deux séries d'expérimentations de ces modèles mémoire, en formation d'enseignants puis en classe de première et terminale, dans les classes d'une des auteurs. Toutes les données détaillées concernant nos modèles mémoire, les expérimentations incluant les pré-tests et post-tests sont disponibles en ligne avec la version complète de cette contribution sur le site :
`https://iremi.univ-reunion.fr/?p=1670`

2 Des explications hétérogènes chez les enseignants

Suite à une première recherche exploratoire menée dans l'académie de La Réunion [12], une expérimentation systématique a été menée auprès de cinquante professeurs de Numérique et Science Informatique (NSI) de l'Agence pour l'Enseignement Français à l'Etranger (AEFE) à Rabat en juin 2025, à Athènes en décembre 2025 et à Tananarive en janvier 2026. Les expérimentations ont été réalisées en deux parties avec une consigne unique : « Reproduire le tableau de la classe après explication aux élèves de l'exécution du programme suivant ».

2.1 Des explications situées dans des cadres variés

La première partie de l'expérimentation porte sur l'explication d'un programme manipulant des variables simples et comportant une définition et un appel de fonction (voir programme de la figure 2). Les explications proposées par les enseignants ont été classées en trois groupes en fonction de la présence de signes appartenant à différents registres [7] liés à des cadres différents [4]. La présence de signes mathématiques (=, Σ, ...) réfère au cadre algébrique. Les manipulations basées sur la sémantique du langage réfèrent à un cadre sémantique. Les tableaux d'évolution des valeurs des variables réfèrent à un cadre de machine dont le modèle mémoire est un registre de représentation.

Les résultats de la classification montrent vingt explications faites dans un cadre sémantique, dix-neuf basées sur des tableaux d'évolution des variables (cadre machine) et onze explications situées dans un cadre algébrique. Neuf explications classées cadre sémantique ou machine ont aussi quelques signes du registre algébrique. Parmi les explications utilisant des tableaux, neuf utilisent des tableaux distincts pour les variables locales et les variables globales, cinq utilisent des tableaux mêlant variables locales et globales, deux utilisent un tableau uniquement pour les variables globales et trois explications utilisent un tableau uniquement pour les variables locales. Les explications effectuées dans un cadre sémantique utilisent principalement la substitution d'une expression par sa valeur (quatorze explications) et le dépliage de boucle (douze explications). Huit explications détaillent principalement le passage de paramètres. Parmi les onze explications du cadre algébrique, on repère les signes suivants : neuf construisent une formulation algébrique du résultat dont une utilise la notation Σ, quatre utilisent une notation géométrique du résultat et cinq utilisent le signe = dans son sens en mathématiques - terme à gauche non réduit à une variable, ou suite de signes = dans une même proposition.

La première conclusion que l'on peut donner de cette expérimentation est que les explications du fonctionnement d'un programme simple chez les enseignants de lycée sont très hétérogènes, d'une part par leur situation dans différents cadres, et d'autre part par l'absence de représentation partagée au sein d'un même cadre. L'usage de différents cadres peut être justifié selon les élèves suivant ou pas un enseignement de mathématiques ; il convient cependant de s'interroger sur la possible appropriation par les élèves des explications utilisées par leur enseignant. Le cadre algébrique est à manier avec précaution en particulier à cause de la différence de signification du signe = en algèbre et en programmation. De plus, il peut poser des difficultés aux élèves peu habitués à la manipulation d'expressions algébriques. L'explication dans le cadre sémantique est aussi à utiliser prudemment car elle exige de maîtriser les règles d'application du principe de substitution pour une variable dont la valeur peut changer. Le cadre machine nous semble le plus élémentaire à utiliser et à transmettre aux élèves. Nos propositions de modèles mémoire se situeront dans ce cadre et notre hypothèse de recherche est que l'usage d'un cadre machine par les enseignants peut induire cet usage par leurs élèves et leur mettre ainsi à disposition une méthode d'explication de l'exécution de leurs programmes.

2.2 Des explications informelles pour des programmes complexes

La deuxième partie de l'expérimentation porte sur deux programmes comportant un tableau et l'appel à une fonction supposée modifier en place le tableau passé en argument. Le premier programme réalise un parcours du tableau par les valeurs, l'autre par les indices. Parmi les 44 réponses exploitables, 31 proposent des explications que l'on a classées selon les cadres sémantique (16 explications) ou machine (15 explications). Les autres proposent une démarche pédagogique pour obtenir collectivement une explication, dont 8 utilisent un outil de visualisation. Python Tutor [11] est l'outil le plus cité.

Parmi les explications du cadre sémantique, on repère les mêmes signes que pour la première partie, substitutions (7), dépliage de boucles (7) et passage des paramètres (3). Parmi celles du cadre machine, cinq utilisent un tableau d'évolution, cinq autres utilisent une représentation avec des flèches, dont une seule permet de visualiser le passage de tableau en paramètre. Pour l'ensemble de ces explications, le constat est que l'enseignant formule des assertions concernant le comportement du programme avec des phrases comme : « elt ne modifie pas la liste », « il travaille sur une copie », « le tableau t n'a pas été modifié », « un tableau est de type mutable ».

Pour cette deuxième partie, nous arrivons à la même conclusion par rapport à l'hétérogénéité des cadres utilisés pour expliquer un programme comportant des fonctions avec un tableau en paramètre. De plus, le remplacement d'une explication attendue par une assertion ou une observation renforce la difficulté déjà observée [12] pour les élèves à s'approprier l'explication de l'enseignant pour analyser leurs propres programmes.

3 Propositions de deux modèles mémoire

On propose successivement deux modèles mémoire : le premier peut expliquer, au niveau initiation, le comportement de programmes simples ne comportant que des données de taille fixe et des fonctions avec des paramètres de types simples ; le second, à un niveau avancé, peut expliquer le comportement de l'ensemble des programmes Python incluant les mutables et les objets.

3.1 Un modèle ardoise / tableau d'évolution au niveau initiation

Dans les travaux de référence sur les machines notionnelles [6,5,9,8], ce modèle mémoire est souvent nommé « modèle de boites ». On préfèrera la métaphore de l'ardoise effaçable, le comportement d'une boite que l'on renverse dans une autre ne permettant pas de rendre compte justement de l'affectation. Une ardoise comporte un ensemble de cases - les variables - nommées chacune par un nom. Une ardoise permet ainsi de représenter à un instant donné l'état de la machine, déterminé par les valeurs des variables du programme en cours d'exécution (voir figure 1). Ce modèle mémoire est suffisant pour représenter l'état d'exécution d'un programme comportant tout type de variable simple (entier, flottant, chaîne

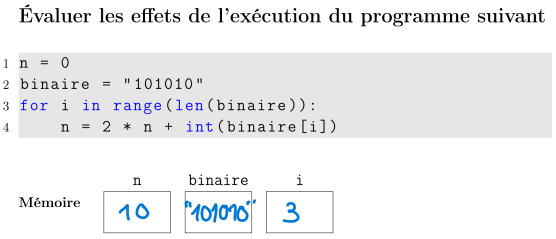

Figure 1. Ardoise pour un programme simple : état en cours d'exécution

ou booléen) ainsi que des tableaux à condition qu'ils soient de taille statique. L'usage en classe de ces ardoises consiste à en distribuer une à chaque élève avec la consigne d'exécuter le programme pas à pas en modifiant au fur et à mesure les valeurs des variables. On généralise le modèle d'ardoise à celui de multi-ardoises pour pouvoir interpréter des appels de fonctions. On dispose alors d'une ardoise par fonction, avec pour chacune la visualisation de ses variables locales et paramètres, et d'une ardoise pour le programme principal. L'activité en classe peut alors être organisée en groupes en distribuant une ardoise à chaque élève.

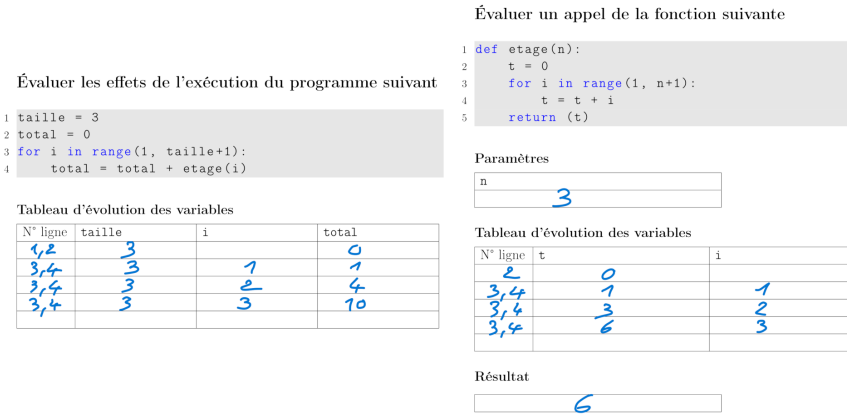

Figure 2. Tableau d'évolution des variables pour programme et fonction

Une variante utile du modèle d'ardoise est le tableau d'évolution des variables qui permet de représenter l'historique des valeurs des variables. Au lieu d'effacer une valeur pour la remplacer par la nouvelle, l'élève doit alors inscrire la nouvelle valeur sur une nouvelle ligne du tableau (voir figure 2). Le tableau d'évolution a sur l'ardoise l'avantage de pouvoir être vérifié plus facilement par l'enseignant et de constituer une trace écrite du travail d'interprétation d'un programme.

3.2 Un modèle de références au niveau avancé

Le modèle de références, appelé aussi modèle de flèches, repose sur l'existence d'une mémoire dans laquelle sont stockées à la fois les variables et les valeurs.

On appelle entité la représentation dans le modèle de références d'un objet Python : `int`, `float`, `bool`, `str`, `tuple`, `list`, `dict` et les objets au sens de la programmation orientée objet.

Comme pour le modèle d'ardoise, une variable est une case avec un nom. La différence se situe au niveau du contenu : dans le modèle de références la case contient une référence (flèche) vers une entité.

La sémantique opérationnelle du langage s'énonce alors par les actions effectuées sur la mémoire par la machine notionnelle. Les actions possibles consistent à évaluer une expression, à créer une valeur en mémoire, à créer une variable en mémoire et à créer ou modifier une référence depuis une variable vers une entité.

Cette sémantique permet de visualiser l'état de la mémoire lors de l'exécution d'un programme Python, notamment lors de l'utilisation de types mutables ou de l'appel de fonctions avec paramètres. La visualisation de l'exécution d'un programme montre d'une part les variables, d'autre part les valeurs (non modifiables) et les références des variables vers les différentes entités.

Les objets de type simple (`int`, `float`, `bool` et `str`) ainsi que les `tuple` sont modélisés par leurs valeurs. Les structures mutables que sont les `list`, `dict` et autres objets au sens de la POO sont des collections de cases étiquetées par des entiers dans le cas des `list` et par des noms dans le cas des autres structures.

La figure 3 montre le modèle de références sur différents objets de Python.

```
date = (2024, 1, 31)

position = [-21, 51]

sommet = {"alt": 3070,
  "nom": "piton des neiges"}
```

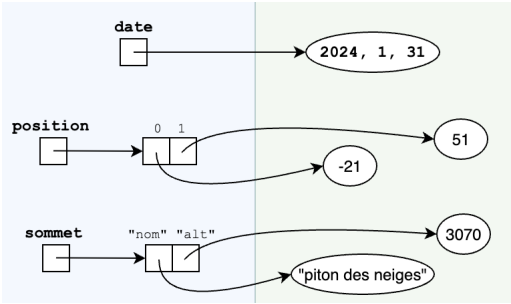

Figure 3. Modèle de références pour `tuple`, `list`, `dict`

Toute référence est nommable et modifiable. Par exemple `position[0]` est la référence vers la valeur `-21` et `sommet` est une référence vers l'enregistrement.

La sémantique de l'affectation (`variable = expression`) consiste à évaluer l'expression, à créer la valeur obtenue en mémoire, puis à créer ou modifier la référence associant la variable à cette valeur. Lorsque l'expression est réduite à

une variable, la référence se fait sur la valeur déjà référencée, il s'agit du concept d'*aliasing*.

```
x = 10
y = 10
alias = x

t = [10, 20, 30]
t2 = t
t2[2] = -1
```

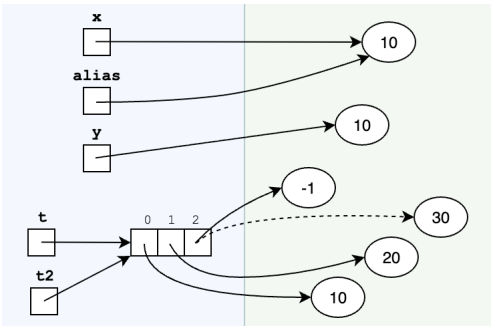

Figure 4. Modèle de références pour l'affectation

La figure 4 donne une visualisation du modèle de références pour une séquence d'affectations. Les trois premières concernent des entiers ; la variable `alias` référence la même valeur que la variable `x`. La variable `y` ne référence pas la même valeur, même si mathématiquement les entiers sont effectivement égaux.

L'affectation de la variable `t` crée une référence vers le tableau `[10, 20, 30]`, celle de `t2` crée un référence vers ce même tableau. La dernière affectation modifie la référence de la dernière case du tableau, l'ancienne référence est visualisée en pointillé. La modification est visible depuis les deux références `t` et `t2`.

De la même manière, une boucle `for` effectue une succession d'affectations de la variable de boucle vers une entité.

```
def maxi(a, b):
    if b > a:
        a = b
    return a

x = 10
y = maxi(x, 30)
```

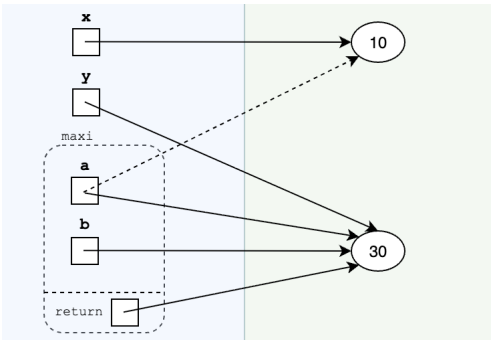

Figure 5. Modèle de références pour l'appel de fonction

La sémantique d'un appel de fonction consiste à créer une zone locale contenant une variable pour chacun des paramètres de la fonction, à créer des références entre chacune des variables et les entités modélisant les paramètres effectifs et à créer une variable supplémentaire référençant l'entité à renvoyer lors de l'instruction `return`.

La figure 5 montre l'état de la mémoire à la fin du programme. Lors de l'appel `maxi(x, 30)`, les variables `a` et `b` référencent respectivement la valeur `10` référencée par `x` et une nouvelle valeur `30`. L'affectation `a = b` remplace l'ancienne référence en pointillé par une référence vers la valeur `30`. L'instruction `return` renvoie cette même référence. L'affectation à `y` crée un alias de cette référence. La zone locale est ensuite détruite.

On note que ce modèle mémoire permet de donner une sémantique simple à l'affectation qui correspond toujours à créer ou modifier une référence. Il permet ainsi de comprendre les passages de paramètres lors des appels de fonction y compris avec des listes comme nous le verrons dans l'expérimentation du modèle avec les élèves de terminale. On remarque aussi que la visualisation séparée dans le modèle, des variables et des valeurs, permet d'expliciter la différence fondamentale entre un tuple, considéré comme une collection de valeurs, et une liste, considérée comme une collection de variables. Cette différence, si elle n'est pas visible comme par exemple dans Python Tutor, complique la compréhension de la différence entre une construction mutable et une qui ne l'est pas.

4 Expérimentations des modèles mémoire

Nos deux modèles mémoire ont été expérimentés par les cinquante enseignants de l'AEFE en formation entre juin 2025 et janvier 2026, et dans deux classes NSI en lycée à La Réunion entre septembre et novembre 2025.

4.1 Expérimentation du modèle d'ardoises en formation d'enseignants et au lycée

Les objectifs de l'expérimentation auprès des enseignants étaient de tester leur appropriation du modèle d'ardoise / tableau d'évolution en utilisant les instruments débranchés (voir figures 1 et 2), puis d'élaborer avec eux les limites d'usages de ce modèle mémoire en terme de langage et enfin d'étudier l'adéquation des environnements de programmation utilisés (*IDE*) à ce modèle mémoire. Quatre groupes d'enseignants (deux à Rabat, un à Athènes et un à Tananarive) ont suivi le même protocole avec deux ateliers successifs, le premier consacré à l'appropriation du modèle mémoire et à ses limites au niveau langage, le second consacré à l'étude des IDE en tant qu'instruments au sens de Rabardel [15].

Des fiches avec un programme et une ardoise ou un tableau d'évolution leur ont été distribuées. Parmi ces fiches, cinq portaient sur des programmes simples comportant des boucles, cinq sur des tableaux, et deux sur un programme principal et une fonction (voir figure 2). Tous les enseignants des quatre groupes se sont appropriés avec succès les fiches proposées pour évaluer correctement

l'effet de l'exécution des programmes. Concernant les limites d'usage du modèle en terme de langage, les quatre groupes ont convergé vers les limitations suivantes : les tableaux Python doivent être de taille statique pour pouvoir représenter chaque composante comme une variable élémentaire et ne doivent pas être passés en paramètre d'une fonction. Par rapport au programme de première NSI, seul le comportement des dictionnaires ne peut pas être modélisé avec le modèle d'ardoise ou tableau d'évolution.

Les ateliers sur les environnements d'apprentissage ont permis d'étudier les retours instrumentaux [15] de cinq environnements : l'éditeur Mu, l'éditeur Thonny [1], les notebook Jupyter, l'environnement Capytale et Visual Studio. Pour les quatres ateliers, c'est l'environnement Thonny qui a été distingué pour sa capacité à fournir un retour instrumental conforme au modèle de multi-ardoises, chaque appel de fonction en mode pas à pas provoquant l'ouverture d'une nouvelle fenêtre (ardoise) avec son environnement de variables locales.

L'expérimentation en lycée s'est déroulée sur une séance de deux heures avec une classe de première NSI de 24 élèves début septembre lors des révisions du programme de seconde portant sur les affectations, séquences, conditionnelles et les boucles bornées on non. La séance a été consacrée à l'expérimentation des ardoises pendant un peu plus d'une heure avec une liste de programmes à évaluer en les exécutant pas à pas à la main grâce à l'ardoise. La séance a débuté par un pré-test de vingt minutes sur les connaissances et compétences en programmation et a été conclue par un post-test différent mais de difficulté analogue au pré-test. Les deux tests comprenaient chacun cinq questions à choix unique portant sur la valeur d'une des variables à l'issue de l'exécution d'un programme donné.

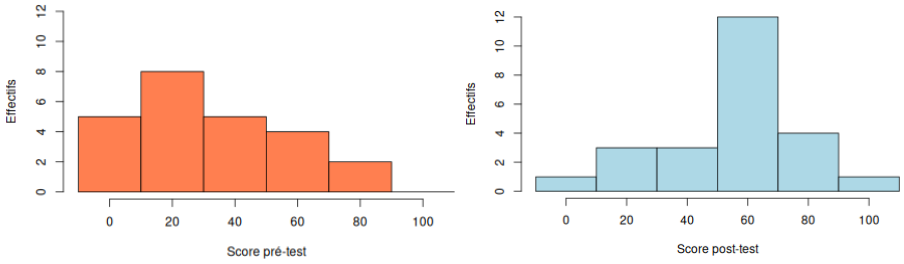

Figure 6. Distribution des scores des 24 élèves au pré-test et au post-test

Le score moyen au post-test (55) est supérieur à celui du pré-test (32). La taille d'effet (d de Cohen) est de 0.68 avec le test t de Student appairés (p-value 0.001) ce qui semble attester d'un effet positif de la séance d'apprentissage. Il convient cependant d'analyser ce résultat avec prudence à cause d'un possible effet retest, au vu des deux passations de tests différents mais analogues à un intervalle de temps très court.

4.2 Expérimentation du modèle de références en formation d'enseignants et au lycée

Les objectifs de l'expérimentation auprès des enseignants étaient de tester leur appropriation du modèle de références et notamment de leur faire découvrir la levée des limitations rencontrées avec le modèle d'ardoises. Au cours de cette phase, les enseignants ont essayé de mettre le modèle de références en défaut. De nombreux petits programmes Python d'un niveau terminale NSI et au-delà ont été imaginés et évalués avec succès à l'aide du modèle. On relève deux apports majeurs par rapport aux limitations sur le langage Python avec le modèle d'ardoise : le caractère dynamique des tableaux (`list` en Python) est rendu possible par la création dynamique de cases dans la zone des variables ; les objets au sens de la programmation orientée objet peuvent être modélisés par une collection de cases étiquetées par les noms des attributs.

L'expérimentation en lycée s'est déroulée sur trois séances. La première séance en forme de pré-test a permis de constater la difficulté des élèves à évaluer un programme Python comportant des tableaux et des appels de fonction avec un tableau en paramètre. Une séance a été consacrée à l'apprentissage des tableaux dynamiques de Python en utilisant le modèle de références comme instrument pour l'explication de la sémantique. Enfin, un post-test différent mais de difficulté analogue au pré-test a été proposé lors d'une dernière séance.

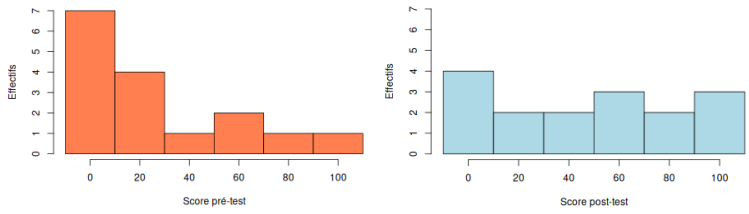

Figure 7. Distribution des scores des 16 élèves au pré-test et au post-test

Le score moyen au post-test (47.5) est supérieur à celui du pré-test (26.25) (voir la figure 7). La taille d'effet (d de Cohen) est de 0.58 avec le test t de Student appairés (p-value 0.018) ce qui semble attester d'un effet positif des séances d'apprentissage. D'un point de vue qualitatif, l'accueil à la fois de l'enseignante et des élèves est encourageant quant à l'utilité du modèle de références pour expliquer un programme manipulant des listes de Python.

Une analyse du post-test par questions pourrait montrer que le gain le plus important est sur sa question 4 : en effet, le modèle de références permet de visualiser que lors d'un parcours de tableau par les éléments, on utilise une nouvelle référence, qui, si elle est modifiée localement pendant la boucle n'a aucun impact sur les références du tableau. La figure 8 montre l'effet de l'exécution du programme de la question 4 du post-test avec le modèle de références.

```
def bonus(notes):
    for e in notes:
        e = e + 1

notes_nsi = [15, 11, 19]
bonus(notes_nsi)
```

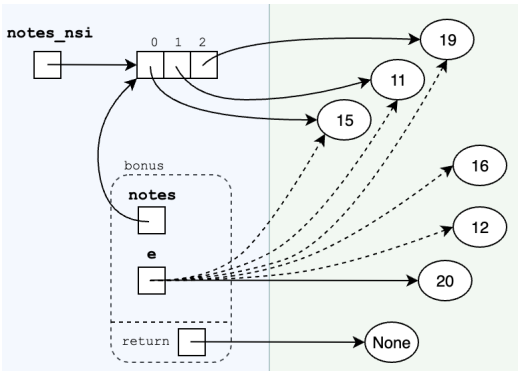

Figure 8. Modèle de références pour l'appel de fonction avec tableau en paramètre

Remarquons que la visualisation du modèle de références sur un support non effaçable surcharge un peu le dessin : les références anciennes de la variable `e` sont nombreuses et toutes gardées (en pointillés sur la figure). En effet la variable référence respectivement, et dans cet ordre, les valeurs `15`, `16` (obtenue par l'évaluation de `e+1`), `11`, `12`, `19` et enfin `20`. On retrouve cette difficulté dans la visualisation des diagrammes de Dickson et Dragon [3].

4.3 Conclusions et perspectives

L'inspiration initiale de notre réflexion était de confronter à l'expérimentation le travail épistémologique de Exibard *et al.* [8] sur les modèles mémoire, en observant des pratiques enseignantes en première et terminale NSI au lycée en France. L'utilisation du modèle de boites y a été observé principalement sous la forme de tableau d'évolution de variables, beaucoup moins le modèle de références. Le modèle d'ardoise que nous proposons est proche du modèle de boites et du modèle *Python Computer Beginning* de Mühling [13], qui propose en plus la visualisation du compteur ordinal. Notre contribution a été d'étendre ce modèle au modèle de « multi-ardoise » afin d'expliquer l'appel de fonction sans besoin d'introduire la notion de pile d'exécution (hors programme NSI), ou l'adresse de retour comme dans le modèle *Python Computer Intermediate* de Mühling.

Les expérimentations en formation d'enseignants et au lycée en classe de première montrent une possible appropriation de ce modèle mémoire. Le modèle a été opérationnalisé sous forme d'activité débranchée avec des ardoises plastifiées, qui ont été laissées aux lycéens, pour leur utilisation. Une enquête en fin d'année scolaire devra tenter de mesurer les pratiques effectives dans la durée. Au vu de l'analyse effectuée avec les enseignants en formation, nous postulons que ce modèle mémoire peut permettre à l'enseignant d'expliquer à ses élèves toutes les notions au programme de la spécialité NSI en classe de première. Cela reste à vérifier a posteriori avec les enseignants qui auront utilisé le modèle toute l'année scolaire.

La principale contribution de notre travail est la proposition d'un modèle de références original pour le langage Python, permettant grâce à la distinction entre la zone des variables et la zone des valeurs, d'expliquer toute la sémantique du langage sans besoin de présenter ni les notions de pile et de tas, ni même la notion de mutable, qui n'est pas au programme du lycée en France. Le modèle de références, en définissant la représentation de chaque structure de données par des entités précises, et en définissant la sémantique de chaque construction du langage, constitue ainsi une machine notionnelle plus abstraite que le modèle de références usuel [8]. En particulier notre modèle de références permet de distinguer simplement les tuples des listes de Python, en considérant les premiers comme des collections de valeurs (donc non modifiables) et les secondes comme des collections de variables. Ce modèle mémoire a aussi l'avantage de conserver la définition de la variable en tant que nom associé à une case, qui contient maintenant toujours une référence. Cela a pour effet de donner une sémantique simple pour l'affectation, qui correspond toujours à la création ou à la modification d'une référence. Comparé au modèle de références usuel, notre modèle de références permet aussi d'expliquer l'aliasing, concept fondamental notamment pour le passage de listes ou dictionnaires en paramètres de fonctions, tout en faisant abstraction de la gestion mémoire sur la pile et dans le tas. Nous proposons ainsi une transposition didactique originale de la programmation avancée en Python pour le lycée.

La principale perspective de ce nouveau modèle de références est de concevoir un instrument de visualisation conforme au modèle mémoire, permettant de donner à l'élève un retour instrumental pertinent sur sa programmation. L'outil Python Tutor, malgré son intérêt, présente deux points faibles : ne pas distinguer systématiquement affectations sur des valeurs semblables et aliasing ; ne pas suffisamment différencier tuples et listes, quel que soit le choix des options de visualisation (avec ou sans références). Un instrument de visualisation conforme à notre modèle permettrait à la fois la diffusion de l'usage du modèle, et le recueil de données sur son usage pour envisager une étude à plus grande échelle de l'effet de cet usage sur l'apprentissage de la programmation au lycée.

Références

1. Annamaa, A. : Thonny, : a python ide for learning programming. In : Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education. p. 343. ITiCSE '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2729094.2754849
2. Arsac, J. : Premières leçons de programmation. Cedic (1980)
3. Dickson, P.E., Dragon, T. : A memory diagram for all seasons. In : Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. pp. 150–156. ITiCSE '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3430665.3456317
4. Douady, R. : Jeux de cadres et dialectique outil-objet. Recherches En Didactique Des Mathématiques **7**(2), 5–31 (1986), https ://revue-rdm.com/1986/jeux-de-cadres-et-dialectique/

5. Du Boulay, B. : Some difficulties of learning to program. Journal of Educational Computing Research **2**(1), 57–73 (1986). https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9
6. Du Boulay, B., O'Shea, T., Monk, J. : The black box inside the glass box : presenting computing concepts to novices. International Journal of man-machine studies **14**(3), 237–249 (1981)
7. Duval, R. : Registres de représentation sémiotique et fonctionnement cognitif de la pensée. Annales de Didactique et de Sciences Cognitives, IREM de Strasbourg **5** (1993)
8. Exibard, L., Francis, N., Meyer, A., van den Bogaard, M. : Modèles de mémoire pour l'enseignement de la programmation. In : Mens, K., Goletti, O. (eds.) Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC. Volume 1. pp. 33–43. Louvain-La-Neuve, Belgium (2024), https ://hal.science/hal-04482121
9. Fincher, S., Jeuring, J., Miller, C.S., Donaldson, P., du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J.L., Petersen, A. : Notional machines in computing education : The education of attention. In : Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education. pp. 21–50. ITiCSE-WGR '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3437800.3439202
10. Greca, I.M., Moreira, M.A. : Mental models, conceptual models, and modelling. International Journal of Science Education **22**(1), 1–11 (2000). https://doi.org/10.1080/095006900289976
11. Guo, P.J. : Online python tutor : embeddable web-based program visualization for cs education. In : Proceeding of the 44th ACM Technical Symposium on Computer Science Education. pp. 579–584. SIGCSE '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2445196.2445368
12. Hoarau, S., Declercq, C. : De l'intérêt de machines notionnelles adaptées à l'apprentissage de la programmation en Python. In : Broisin, J., Declercq, C., Fluckiger, C., Jolivet, S., Parmentier, Y., Peter, Y., Secq, Y. (eds.) Actes de l'Atelier APIMU. pp. 39–49. Villeneuve d'Ascq, France (2025), https ://hal.science/hal-05084735
13. Mühling, A. : Python computer, in : Notional machines a curated collection. https ://notionalmachines.github.io/nms/PythonComputer_1.html (2020)
14. Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J.A. : Exploring the role of visualization and engagement in computer science education. SIGCSE Bull. **35**(2), 131–152 (Jun 2002). https://doi.org/10.1145/782941.782998
15. Rabardel, P. : Les hommes et les technologies ; approche cognitive des instruments contemporains. Armand Colin (1995), https ://hal.science/hal-01017462
16. Rogalski, J., Vergnaud, G. : Didactique de l'informatique et acquisitions cognitives en programmation. Psychologie Française **32**, 267–274 (1987)
17. Sorva, J., Karavirta, V., Malmi, L. : A review of generic program visualization systems for introductory programming education. ACM Trans. Comput. Educ. **13**(4) (Nov 2013). https://doi.org/10.1145/2490822
18. Sorva, J. : Notional machines and introductory programming education. ACM Transactions on Computing Education **13**, 8 :1–8 :31 (06 2013). https://doi.org/10.1145/2483710.2483713