Un référentiel de compétences en programmation pour identifier les difficultés des débutants et différencier les activités

Sophie Chane-Lune, Christophe Declercq et Sébastien Hoarau

Laboratoire d'Informatique et de Mathématiques, Université de La Réunion

Résumé Nous proposons un référentiel de compétences en programmation croisant compétences de la pensée informatique, et notions informatiques au programme du lycée en France, dans l'objectif de mieux identifier les difficultés des programmeurs débutants, en particulier par rapport aux compétences d'abstraction et de généralisation mises en œuvre par l'usage de fonctions informatiques.

Nous proposons une série d'activités élémentaires alignées avec les compétences définies et en déduisons un test destiné à identifier ces difficultés chez les programmeurs débutants.

Nous présentons ensuite les résultats de deux cohortes d'élèves et d'étudiants à ce test et discutons ces résultats. Nous en déduisons des nécessités de remédiation et proposons des activités diversifiées permettant de travailler les compétences non maitrisées par les élèves.

1 Introduction

L'introduction de l'informatique en tant que discipline au lycée en France en 2019 s'est effectuée avec d'une part la mise en place de programmes et d'horaires dédiés et d'autre part la formation continue puis initiale d'enseignants pour la discipline. Les moyens d'enseignement ont suivi avec des manuels, une activité intense d'élaboration entre pairs [3] et l'arrivée de sujets d'épreuves avec leur effet normalisateur sur les pratiques enseignantes.

Les praticiens ont alors redécouvert les difficultés des programmeurs débutants, pourtant largement documentées par la recherche à l'époque de l'option informatique des lycées dans les années 1980 [11,10], à savoir la difficulté des notions de variable et de fonction, l'intérêt de méthodes de programmation, la complexité intrinsèque des dispositifs d'exécution, le statut particulier des variables logiques... Une revue des travaux de cette époque a été effectuée plus récemment par Lagrange et Rogalski [6]. Les enseignants ont aussi pu découvrir, dans le contexte nouveau créé par l'introduction de Scratch au collège en 2016, la délicate transition d'un langage de programmation par blocs à un langage textuel (Python).

Les chercheurs ont rapidement mis en évidence la faible diversité des activités proposées aux élèves dans les moyens d'enseignement - voir en particulier l'analyse de [5] concernant les tâches figurant dans les activités des manuels ou dans

les EIAH. Des formes d'activités originales, proposant d'analyser les programmes avant d'en écrire, ont déjà été proposées par Goletti et Mens [4].

Les praticiens regrettaient au même moment la normalisation en partie provoquée par la forme imposée des sujets d'épreuves pratiques au baccalauréat à savoir l'activité de type feuille blanche (écrire un programme qui...) ou le programme à trous.

C'est dans ce contexte que notre recherche a débuté en tentant d'explorer d'autres formes d'activités pour pallier aux difficultés constatées des élèves. Lors d'un premier travail, nous avons commencé à utiliser l'approche par compétences [1], popularisée en informatique par les travaux de Wing [13], pour analyser les compétences en jeu dans différentes situations.

2 Vers un référentiel de compétences en programmation adapté aux programmes du lycée en France

Nous avons initialement fondé notre réflexion sur la définition opérationnelle de la pensée informatique proposée par Selby et Woolard [12] à savoir la capacité à penser en termes d'abstraction, de décomposition, à penser « algorithmiquement », et à penser en termes d'évaluation et de généralisation. Nous avions déjà proposé de formuler ces compétences en français par des verbes [2] et de préciser la « pensée algorithmique » en utilisant le verbe *anticiper*. Anticiper est en effet le principal obstacle didactique rencontré par les programmeurs débutants : « Une propriété difficile à intégrer [...] est le caractère différé d'une exécution du programme » [9]. Anticiper les étapes successives d'un traitement et les différents cas à envisager, c'est tout l'art de programmer. Nous avions alors cinq compétences génériques pour décrire la pensée informatique : évaluer, anticiper, décomposer, généraliser et abstraire.

Nous avons ensuite proposé d'ajouter une sixième compétence générique aux précédentes : la compétence *modéliser*. Cette compétence est largement partagée avec d'autres sciences dont en particulier les mathématiques, mais elle revêt une signification particulière en informatique quand il s'agit de choisir l'information à représenter comme donnée d'un problème ou résultat. Cette compétence avait déjà été énoncée par Wing : « It is choosing an appropriate representation for a problem or *modeling* the relevant aspects of a problem to make it tractable ». La compétence *modéliser* avait cependant été écartée dans la définition de Selby et Woolard au prétexte que les modèles sont utilisés par les compétences de la pensée informatique mais ne la définissent pas. Nous pensons au contraire qu'il est utile de distinguer la compétence *modéliser* de la compétence *abstraire* à laquelle elle pourrait être rattachée, parce que nous avons pu observer que des élèves maîtrisent l'une et pas l'autre.

Prenons le problème du calcul de la méridienne avec un sextant et une montre qui permet à un marin de calculer sa longitude par mesure de l'heure de culmination du soleil. Modéliser, consiste à choisir les variables pour représenter les données du problème, à savoir le temps de la montre en heures, minutes et secondes. Il y a bien sûr des abstractions le plus souvent implicites : dans ce calcul,

on peut faire abstraction de l'âge du capitaine. On pourrait aussi proposer une abstraction de données, consistant à représenter un temps par une structure de données abstraite. Pour des débutants en programmation, savoir *modéliser* des données par quelques variables élémentaires est une compétence assez accessible. La compétence *abstraire* demande quant à elle un apprentissage spécifique et beaucoup plus long, d'où l'intérêt de distinguer ces deux compétences.

Les compétences *évaluer* et *anticiper* permettent déjà de diversifier les activités. *Evaluer* un programme, c'est la capacité à lui attribuer mentalement une valeur (résultat, type...). L'activité part donc d'un programme fourni. *Anticiper*, c'est décrire à l'avance ce que devra faire la machine, c'est-à-dire prévoir par un programme, l'enchaînement séquentiel, répétitif ou conditionnel des instructions, avant même le début de son exécution. L'activité part donc d'une feuille blanche. Nos premières expériences [1] ont montré qu'il était plus facile pour les élèves d'apprendre à lire un programme (compétence *évaluer*) que d'apprendre à en écrire (compétence *anticiper*).

La maîtrise de la compétence *modéliser* est un préalable à toute activité d'écriture de programme. En l'explicitant, on déduira des activités préliminaires où la tâche de l'élève consiste à choisir des variables où des structures de données pour représenter l'information disponible ou à calculer.

Les compétences *décomposer*, *généraliser* et *abstraire* sont des compétences cognitives de plus haut niveau. Ce sont des compétences méthodologiques qui vont être sollicitées lors de la réécriture d'un programme vers une version plus générale ou plus abstraite, ou lors de l'analyse descendante d'un problème (compétence *décomposer*).

Le préambule commun aux programmes d'informatique au lycée en France [8] fait explicitement référence à ces compétences :

- *analyser et modéliser un problème en termes de flux et de traitement d'informations ;*
- *décomposer un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions ;*
- *concevoir des solutions algorithmiques ;*
- ...
- *développer des capacités d'abstraction et de généralisation.*

Cependant la suite des programmes n'y fait plus référence et est organisée par contenus, dans lesquels sont notées des capacités attendues.

La difficulté, pour l'enseignant, à mettre en œuvre une approche par compétences est alors de distinguer pour chaque compétence, les différents contextes où elle peut intervenir. Par exemple, la compétence *généraliser* est invoquée dans une activité consistant à passer d'une instance à une classe de problèmes. Au niveau élémentaire cela peut se faire en remplaçant une constante par une variable. Cette compétence est aussi mise en oeuvre lors du remplacement d'un fragment de code par une fonction ou plus tard en utilisant la notion de classe en programmation objet. Il n'est donc pas réaliste de dire qu'un élève sait *généraliser*, sans préciser le contexte dans lequel intervient cette généralisation.

2.1 Construction d'un référentiel croisant compétences et notions au programme

Notre proposition est de croiser une approche basée sur les compétences génériques de la pensée informatique avec une approche basée sur les compétences *programmer avec des notions particulières*. Notre méthode a consisté à réécrire l'ensemble des capacités attendues des programmes d'informatique au lycée relatives au domaine de la programmation, en utilisant uniquement les six verbes choisis. Cela nous a amené à déboubler en plusieurs compétences élémentaires quand une capacité attendue au programme masquait plusieurs compétences, ou parfois à regrouper plusieurs capacités en une seule compétence élémentaire. Ce travail a été mené de mars à octobre 2023 lors de séances de travail régulières entre les co-auteurs (8 séances de 3h) avec recherche de consensus pour tous les choix effectués.

La figure 1 montre cette catégorisation pour les notions de programmation au programme de la classe de seconde en sciences numériques et technologie (SNT).

Compétence	Évaluer	Modéliser	Anticiper	Décomposer	Généraliser	Abstraire
Programmer avec des expressions	Évaluer la valeur et le type (entier, chaîne ou booléen) d'une expression comportant variables, constantes et opérateurs	Modéliser les informations disponibles/à calculer par des variables de type élémentaire (entier, chaîne et booléen)	Anticiper l'écriture d'une expression comportant variables, constantes et opérateurs	Décomposer une expression complexe en identifiant une ou des variables intermédiaires	Généraliser une expression en remplaçant une constante par une variable	
Programmer avec des affectations et des séquences	Évaluer les valeurs des variables, à l'exécution d'un programme comportant une séquence d'affectations	Modéliser les informations à calculer par une ou des variables intermédiaires	Anticiper l'écriture d'un traitement séquentiel pour obtenir un résultat spécifié, la méthode étant donnée	Décomposer un traitement complexe en identifiant une ou des variables intermédiaires		
Programmer avec des constructions répétitives	Évaluer les valeurs des variables, à l'exécution d'un programme comportant une construction répétitive	Modéliser par une ou des variables les informations à modifier à chaque répétition	Anticiper l'écriture d'un traitement répétitif pour obtenir un résultat spécifié, la méthode étant donnée	Décomposer un traitement complexe en identifiant une répétition	Généraliser un traitement séquentiel en le remplaçant par un traitement répétitif	
Programmer avec des constructions alternatives ou conditionnelles	Évaluer les valeurs des variables, à l'exécution d'un programme comportant une ou des constructions alternatives ou conditionnelles	Modéliser par une variable booléenne la condition d'une alternative ou d'une conditionnelle	Anticiper l'écriture d'un traitement alternatif ou conditionnel pour obtenir un résultat spécifié, la méthode étant donnée	Décomposer un traitement complexe en plusieurs cas en utilisant des constructions alternatives ou conditionnelles		
Programmer avec des fonctions	Évaluer le résultat de l'appel d'une fonction avec des paramètres donnés	Modéliser un traitement par une fonction en spécifiant ses paramètres	Anticiper l'écriture du corps d'une fonction	Décomposer un traitement complexe en utilisant la composition de fonctions	Généraliser un traitement, par la définition d'une fonction avec paramètres ou par l'ajout d'un paramètre à une fonction	Abstraire un traitement par une définition de fonction avec paramètres et commentaire

Figure 1. Référentiel pour la classe de seconde

Au croisement d'une colonne et d'une ligne, figure une capacité ou compétence élémentaire correspondant à la compétence générique de la colonne, contextualisée par la ou les notions en jeu dans la ligne courante.

On retrouve ainsi en particulier au croisement de *généraliser* et de *programmer avec des fonctions*, la compétence élémentaire *généraliser un traitement par la définition d'une fonction avec paramètres*. Les cases restant grisées correspondent à des croisements auxquels nous n'avons pas trouvé d'intérêt spécifique. Les compétences *généraliser* et *abstraire* ne sont pas systématiquement sollicitées pour toutes les notions de programmation car elles sont intrinsèquement liées aux notions en jeu. L'abstraction regroupe l'abstraction de données qui permet d'encapsuler des informations, l'abstraction par des fonctions qui permet d'encapsuler des traitements, et l'abstraction par les classes en programmation objet qui conjugue abstraction de données et abstraction par des fonctions. La généralisation regroupe la généralisation par l'ajout de variables ou paramètres et

la généralisation de motifs qui consiste à inférer un traitement répétitif à partir d'un motif séquentiel répété.

Ce travail est en cours pour l'ensemble des programmes du lycée de la seconde à la terminale dans le cadre du travail doctoral de S. Chane-Lune. Il consiste à identifier pour chaque contexte de programmation lié à l'introduction de nouvelles notions, les possibles croisements avec les compétences génériques de la pensée informatique.

2.2 Aligner activités et compétences

Une de nos hypothèses de recherche est de postuler que l'on peut pour chaque compétence élémentaire identifier une activité élémentaire permettant de tester en contexte cette compétence.

La figure 2 regroupe pour les compétences citées en figure 1, des propositions d'activités permettant de les tester.

Compétence	Évaluer	Modéliser	Anticiper	Décomposer	Généraliser	Abstraire
Programmer avec des expressions	Quel est le type et la valeur de l'expression suivante ?	Choisir une ou des variables pour représenter …	Ecrire une expression dépendant de ... permettant de calculer …	Réécrire l'expression suivante en nommant ... la partie entre parenthèses	Réécrire l'expression suivante en remplaçant la valeur ... par une notation plus générale	
Programmer avec des affectations et des séquences	Quelle est la valeur de la variable ... à la fin de l'exécution du programme Python suivant ?	Choisir une ou des variables pour mémoriser les résultats intermédiaires utiles pour calculer …	Ecrire un programme Python permettant de calculer... / Ecrire un programme Python permettant d'afficher …	Décomposer le problème suivant en utilisant des variables intermédiaires		
Programmer avec des constructions répétitives	Quels sont les messages affichés à l'exécution du programme Python suivant ? / Quelle est la valeur de la variable … à la fin de l'exécution du programme Python suivant ?	Choisir une ou des variables à modifier itérativement pour calculer…	Ecrire un programme Python permettant de calculer … par la méthode suivante … / Ecrire un programme Python permettant d'afficher …	Décomposer le problème suivant en identifiant un traitement pouvant être répété plusieurs fois	Modifier le programme Python suivant en utilisant une répétition, tout en conservant les mêmes affichages à l'exécution / le même résultat	
Programmer avec des constructions alternatives ou conditionnelles	Quels sont les messages affichés à l'exécution du programme Python suivant ? / Quelle est la valeur de la variable … à la fin de l'exécution du programme Python suivant ?	Choisir et nommer une variable pour mémoriser le respect de la condition ... dans le problème suivant …	Ecrire un programme Python permettant de calculer … par la méthode suivante … / Ecrire un programme Python permettant d'afficher …	Décomposer le problème suivant en distinguant les cas nécessitant un traitement particulier		
Programmer avec des fonctions	Quelle est la valeur à l'évaluation de l'expression ... avec la définition de fonction Python suivante ?	Spécifier le nom, les paramètres d'entrée et le type de résultat à renvoyer pour une fonction destinée à calculer…	Ecrire une fonction Python … , qui étant donnés … , renvoie …	Décomposer le problème suivant en identifiant deux parties pouvant chacune être programmées par une fonction	Dans le programme Python suivant, remplacer les instructions écrites plusieurs fois, en définissant et utilisant une fonction	Décrire le traitement effectué par la fonction suivante en faisant abstraction de la méthode utilisée

Figure 2. Activités génériques par compétences

Il s'agit, dans cette proposition, d'énoncés génériques d'activités devant être complétés par un programme ou un problème particulier. Par exemple l'activité 1 ci-dessous est une instance possible d'activité pour la compétence élémentaire : *évaluer les valeurs des variables, à l'exécution d'un programme comportant une séquence d'affectations.* On appelera dans la suite question, toute activité qui sera incluse ultérieurement dans un test.

Question 1. Quelles sont les valeurs des variables `a` et `b` à la fin de l'exécution du programme Python suivant ?

```
a = 42
b = 56
a = b - a
b = b - a
a = a + b
```

L'activité suivante est une variante concernant la même compétence.

Question 2. Quelle est la valeur de la variable `bissextile` à la fin de l'exécution du programme Python suivant ?

```
par4 = 2024 % 4 == 0
par100 = 2024 % 100 == 0
par1000 = 2024 % 1000 == 0
bissextile = par4 and not par100 or par1000
```

Concernant les compétences de programmation au programme de la classe de seconde, nous avons ainsi pu aligner notre référentiel de compétences avec une proposition d'activités élémentaires ciblant chacune l'apprentissage d'une compétence particulière. La réponse est moins évidente concernant les compétences en jeu dans la suite des programmes du lycée où des interdépendances apparaissent, ce qui rend plus difficile la conception d'activités élémentaires spécifiques.

2.3 Proposition d'un test de compétences en programmation

Notre proposition d'un test de compétences en programmation a pour objectif d'identifier les difficultés et de proposer des activités de remédiation. Il ne s'agit pas d'évaluer les élèves, ce qui nécessiterait un travail préliminaire d'alignement entre les objectifs de l'enseignement, les activités proposés et les modalités d'évaluation.

Notre proposition d'évaluation diagnostique a pour seule ambition de repérer les difficultés récurrentes des élèves et de proposer des activités de remédiation adaptées.

Nous avons ainsi pu fonder une proposition de test de compétences en programmation sur la base des 24 énoncés génériques du tableau de la figure 2. Nous avons aussi souhaité diversifier pour chaque activité élémentaire, le type d'informations manipulées et la nature du programme (interactif ou calculatoire). Nous avons alors obtenu un test comportant plus de cinquante questions couvrant l'ensemble des compétences.

Dans la pratique, il n'a pas été jugé possible par les enseignants expérimentateurs de mettre en œuvre le test complet, ce qui aurait pris un temps supérieur à deux heures pour les élèves. Dans l'attente du développement d'une méthodologie de choix de questions permettant d'assurer une couverture suffisante à la fois des compétences et des types de données et de problèmes, tout en conservant un nombre de questions raisonnable, il a été convenu de laisser les expérimentateurs choisir leurs questions pour la première expérimentation.

Les résultats qui suivent portent donc sur des extraits de ce test, adaptés aux contextes d'expérimentation. La première expérimentation a été conduite auprès d'une classe de première NSI, dont les élèves avaient tous suivis l'enseignement de seconde l'année précédente. La seconde expérimentation a été menée avec une promotion de licence première année d'informatique et de mathématiques, qui avaient eux aussi tous suivi l'enseignement de seconde trois ans auparavant.

3 Expérimentation et premiers résultats

3.1 Expérimentation avec une classe de première spécialité NSI

Une version allégée du test a été proposée aux 49 élèves de première NSI du lycée Mémona Hintermann-Afféjee (Saint-Denis de La Réunion), en début d'année scolaire 2023. Ce test a été découpé en deux parties et chacune d'elle a été réalisée à une semaine d'intervalle. La première partie du test comportait 11 questions et portait sur les compétences *évaluer* (5 questions dont la question 3), *anticiper* (4 questions), *modéliser* (une question) et *généraliser* (question 4) dans le contexte de la programmation élémentaire (sans fonctions).

Question 3. Quelle est la valeur de l'expression Python suivante ?

```
2 + 2 == 4 or 2 + 2 == 5
```

Question 4. Modifier le programme Python suivant en utilisant une répétition, tout en conservant les mêmes affichages à l'exécution.

```
print("1 ...")
print("2 ...")
print("3 ...")
print("Partez !")
```

La deuxième partie du test portait sur le contexte des fonctions et comptait 5 questions, chacune d'elle portant sur l'une des compétences suivantes : *évaluer des fonctions*, *modéliser une fonction*, *anticiper l'écriture d'une fonction*, *généraliser* et *abstraire avec des fonctions*.

Les résultats sont médiocres en moyenne. Les seules questions avec un taux de réussite supérieur à 50 % sont les questions d'évaluation d'expressions de type entier ou chaîne. Toutes les autres questions ont un taux de réussite inférieur à 25 %.

La figure 3 montre la répartition des scores obtenus sur la cohorte de première pour les deux parties du test.

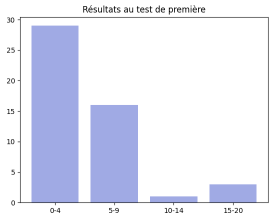

Figure 3. Répartition des scores pour les élèves de première

3.2 Expérimentation avec une promotion de licence première année

Le test qui a été administré aux étudiants de l'université une semaine après le test au lycée, avait un objectif différent : mesurer ce qu'il reste des enseignements d'informatique en seconde pour des primo-entrants à l'université n'ayant pour moitié pas poursuivi la spécialité NSI au lycée jusqu'en terminale. La variable déterminante dans ce contexte est le fait d'avoir ou pas suivi l'enseignement NSI en classe de première et terminale.

Les contraintes matérielles de passation du test en amphi le jour de la réunion de rentrée ont contraint à choisir un questionnaire à réponse numérique, réalisé sur papier et lisible par lecture optique. Ce choix permettait de réduire le temps de correction et d'obtenir rapidement de premières analyses. Cela a aussi limité les possibilités de tester différentes compétences et a amené à tester exclusivement la compétence *évaluer*. Au vu des résultats de la première expérimentation avec la classe de première, où seules ces questions ont donné lieu à des réponses majoritairement correctes, cela nous a semblé cependant utile pour obtenir une première photographie partielle des compétences acquises par cette promotion.

La promotion de première année de licence de l'université de La Réunion comportait 118 étudiants d'informatique et 43 étudiants de mathématiques. Les questions avec réponses numériques (entre 1 et 31) ont été encodées par les étudiants sur la feuille réponse en noircissant les cases correspondant au code en base 2 de la réponse. Le tableau de codage était donné sur l'énoncé. Les résultats des étudiants d'informatique ont été différenciés selon la variable : *a suivi NSI en première et en terminale*, catégorie qui comporte 51 étudiants sur 118.

Les 20 questions proposées portaient toutes sur la compétence *évaluer*, dont 5 sur *évaluer un programme avec des affectations et des séquences*, 6 sur *évaluer un programme avec des constructions répétitives*, 4 sur *évaluer un programme avec des constructions alternatives ou conditionnelles* et 5 sur *évaluer un programme avec des fonctions*. La figure 4 montre la répartition des scores pour ce test. Une analyse d'indépendance par la méthode du χ^2 confirme que la répartition des scores n'est pas indépendante de la variable *a suivi NSI en première et en terminale* avec une erreur de l'ordre de 10^{-6}.

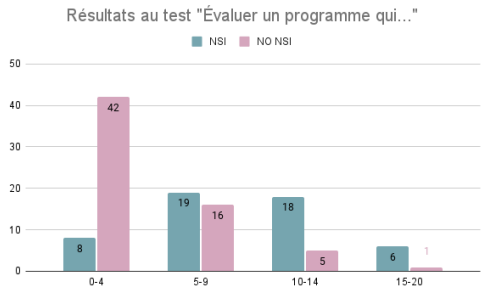

Figure 4. Répartition des scores pour les étudiants d'informatique

3.3 Analyse des résultats

On s'intéresse maintenant à l'analyse en détail de quelques questions caractéristiques de difficultés rencontrées à la fois chez les lycéens et les étudiants.

Les questions d'évaluation dont les scores moyens sont les moins bons sont ceux qui portent sur des booléens, ce qui confirme des résultats connus [6] : « les difficultés liées aux conditions (valeurs logiques de type booléen) sont sous-estimées ». Une interprétation possible du très fort taux d'erreur à la question 3, tient au conflit entre les pratiques des élèves en mathématiques et en informatique. En mathématiques les lycéens apprennent à n'écrire que des propositions vraies. Ceci peut entrer en conflit avec l'écriture en informatique d'expressions booléennes valant potentiellement faux. Cette interprétation est uniquement basée sur les verbatims d'élèves : « Madame, il y a une erreur dans la question 3 » et mériterait d'être approfondie par une étude qualitative des représentations des élèves.

Concernant la question 1, la mauvaise réponse `28` très fréquente illustre une conception erronée de la sémantique de l'affection au sein d'une séquence d'instructions. L'expression `b - a` semble être calculée une fois pour toutes avec les valeurs initiales (`42` et `56`) pour produire la valeur `14`. Cette valeur est utilisée ensuite dans les deux nouvelles affectations de `a` et `b`, expliquant la réponse `28` pour la valeur de la variable `a`. Ce qui est étonnant ici, c'est que les valeurs initiales de `a` et `b` soient utilisées dans les deux premières affectations, puis que leurs nouvelles valeurs soient enfin prises en compte dans la dernière. On peut supposer l'influence réductrice d'un savoir en calcul algébrique, le principe de substitution des variables, qui n'est pas transposable en programmation impérative, et qui est utilisé ici de manière erratique.

Les questions portant sur l'évaluation d'un programme comportant une répétition révèlent à nouveau un usage abusif du principe de substitution au moment de modifier une variable de type accumulateur. Des réponses d'élèves et d'étudiants, semblent montrer qu'ils substituent partout le nom d'une variable par sa valeur initiale avant de commencer à simuler l'exécution de la boucle.

Les questions portant sur la généralisation et l'abstraction par les fonctions n'ont été traitées par aucun élève de première. Ceci peut interroger sachant que l'écriture des fonctions informatiques figure bien à la fois au programme de mathématiques et de SNT en classe de seconde. Une piste d'explication pourrait venir des moyens d'enseignement utilisés : la quasi-totalité des activités portant sur les fonctions au sens informatique, dans les manuels de mathématiques de seconde, sont de la forme : *écrire une fonction Python ..., qui étant donnés ..., renvoie ...*. Ce type d'activité sollicite la compétence *anticiper l'écriture du corps d'une fonction*. Les élèves n'ont donc pas eu l'occasion de s'exercer à prendre l'initiative de définir une fonction pour *généraliser* ou *abstraire* un traitement. Ces compétences étaient, de fait, prises en charge par leur enseignant. L'explication peut venir aussi du curriculum de seconde, qui à la différence de celui de première et de terminale, ne précise pas l'objectif de développer les capacités d'abstraction et de généralisation.

4 Conclusion

L'élaboration d'un référentiel de compétences en programmation et d'un catalogue d'activités alignées sur ce référentiel nous a permis d'analyser finement quelques difficultés rencontrées par des programmeurs débutants. Cela nous permet de recommander la diversification d'activités sur la base des énoncés génériques proposés pour développer les compétences les moins souvent travaillées. En particulier les activités permettant de travailler la compétence *évaluer* nous semblent un préalable aux activités d'écriture de code. Nous proposons aussi de développer des activités de conception permettant de développer d'abord la compétence *modéliser* puis la compétence *décomposer* ainsi que des activités de réécriture permettant de développer les compétences *généraliser* et *abstraire*.

Les mauvais résultats globaux à nos tests sont à nuancer par la taille de nos échantillons et l'analyse des contextes d'enseignement. Le rapport du conseil supérieur des programmes [7] a en effet souligné pour l'enseignement de SNT que « la partie scientifique et technique relative aux fondements du numérique et aux activités de programmation est souvent délaissée », ce qui pourrait expliquer les faibles compétences en programmation à l'entrée en première. Le constat que ces compétences restent fragiles à l'entrée à l'université y compris pour certains élèves ayant suivi la spécialité NSI reste à investiguer à plus grande échelle. Il convient aussi d'être prudent concernant l'interprétation des résultats d'un test focalisé sur des compétences élémentaires, alors que les programmes d'enseignement préconisent un apprentissage par projet et proposent des situations de programmation parfois très complexes : programmation dynamique, algorithme de Boyer-Moore...On peut cependant s'interroger sur la possibilité d'enseigner de telles situations complexes, quand les compétences élémentaires sur les notions de base ne sont pas maîtrisées par les élèves.

Les prolongements de ce travail se déclinent sur plusieurs axes : la consolidation du référentiel, son usage pour élaborer des tests et son usage pour catégoriser des activités. Concernant la phase de consolidation, seules les cinq premières lignes du référentiel, concernant les compétences abordées en classe de seconde, ont été présentées ici et analysées. Notre référentiel global pour le lycée en comporte vingt et est en cours de finalisation. Chaque compétence élémentaire doit encore être confrontée à la possibilité de lui associer une activité spécifique. Ces activités doivent ensuite être testées en classe, soit dans le cadre de tests, soit dans le cadre d'activités d'apprentissage ou de remédiation.

Le développement de tests spécifiques de compétences pour la classe de première et de terminale, adossés à ce référentiel, est pour nous une perspective logique pour continuer à investiguer les difficultés rencontrées par les élèves et proposer des activités diversifiées et des remédiations adaptées à chaque niveau.

L'usage de notre référentiel pour catégoriser les activités de programmation complexes proposées aux élèves, lors des épreuves pratiques du baccalauréat ou dans les espaces collaboratifs de partage d'activités entre enseignants, constitue aussi une perspective de recherche intéressante, dans l'objectif de tenter de corréler les difficultés rencontrées par les élèves dans ces activités complexes avec celles identifiées lors de nos tests de compétences.

Références

1. Chane-Lune, S., Declercq, C., Hoarau, S. : Expérimentation du pair-programming en spécialité NSI en classe de première pour l'acquisition de compétences en programmation. In : Broisin, J., Declercq, C., Fluckiger, C., Parmentier, Y., Peter, Y., Secq, Y. (eds.) Atelier " Apprendre la Pensée Informatique de la Maternelle à l'Université ", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). pp. 25–32. Brest, France (2023), https ://hal.science/hal-04144211
2. Declercq, C. : Didactique de l'informatique : une formation nécessaire. Sticef **28**(3) (Apr 2022), http ://sticef.org/num/vol2021/28.3.8.declercq/28.3.8.declercq.htm
3. Djeballah, S., Comte, M.H., Fourny, M., Hoarau, S., Juton, A., Khaneboubi, M., Lagarrigue, A., Massart, T., Poulmaire, C., Prince, V., Renouf, S., Viéville, T., Vincent, J.M. : Un espace de formation francophone des enseignants, dédié à l'apprentissage de l'informatique, dans le secondaire. https ://adjectif.net (Sep 2023)
4. Goletti, O., Pierpont, F.d., Mens, K. : Création d'exemples résolus avec objectifs étiquetés pour l'apprentissage de la programmation avec Python (May 2022), https ://hal.archives-ouvertes.fr/hal-03697932
5. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Description et analyse de trois EIAH d'apprentissage de Python. In : Broisin, J., Declercq, C., Fluckiger, C., Parmentier, Y., Peter, Y., Secq, Y. (eds.) Atelier " Apprendre la Pensée Informatique de la Maternelle à l'Université ", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). pp. 9–16. Brest, France (2023), https ://hal.science/hal-04144212
6. Lagrange, J.B., Rogalski, J. : Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique. Annales de Didactiques et de Sciences Cognitives (Jul 2017), https ://hal.archives-ouvertes.fr/hal-01740442, publisher : Institut de Recherche sur l'Enseignement des Mathématiques de Paris (IREM)
7. Ministère de l'Education Nationale et de la Jeunesse : Avis du conseil supérieur des programmes sur la contribution du numérique à la transmission des savoirs et à l'amélioration des pratiques pédagogiques (juin 2022)
8. Ministère de l'Éducation nationale de la Jeunesse et des Sports : Programmes et ressources en numérique et sciences informatiques - voie G (2019), https ://eduscol.education.fr/2068/programmes-et-ressources-en-numerique-et-sciences-informatiques-voie-g
9. Rogalski, J., Samurçay, R. : Les problèmes cognitifs rencontrés par des elèves de l'enseignement secondaire dans l'apprentissage de l'informatique. European Journal of Psychology of Education (1986)
10. Rogalski, J., Samurçay, R., Hoc, J. : L'apprentissage des méthodes de programmation comme méthodes de résolution de problème. Le Travail Humain **51** (1988)
11. Samurçay, R. : Signification et fonctionnement du concept de variable informatique chez des eleves debutants. Educational Studies in Mathematics (1985)
12. Selby, C., Woollard, J. : Computational thinking : the developing definition. In : SIGCSE (2014)
13. Wing, J.M. : Computational Thinking. Communications of the ACM **49** (2006)