Expérimentation d'un environnement pour l'apprentissage de la programmation fonctionnelle et de la récursivité en terminale NSI

Christophe DECLERCQ

Laboratoire d'informatique et de mathématiques, Université de la Réunion

Sophie CHANE-LUNE

Laboratoire d'informatique et de mathématiques, Université de la Réunion

Sébastien HOARAU

Laboratoire d'informatique et de mathématiques, Université de la Réunion

Numéro thématique 3 / 2023 - T3

RÉSUMÉ *Nous proposons un environnement de programmation par blocs destiné à initier les élèves à la programmation fonctionnelle en Python et à la récursivité. Nous justifions les choix ergonomiques effectués par rapport aux notions en jeu au programme de NSI en classe de terminale et aux difficultés rencontrées par les élèves et leurs professeurs pour aborder ces notions. L'hypothèse principale de ce travail est de postuler que l'apprentissage de la récursivité en programmation fonctionnelle peut être mené de manière élémentaire, c'est à dire sans le mélanger à de la programmation impérative ni à de la programmation objet. Nous présentons une expérimentation menée dans une classe de terminale, ce qui nous permet de discuter de la pertinence de l'approche proposée.*

MOTS-CLÉS *Lycée – Programmation - Elèves*

Introduction

Après avoir posé le cadre théorique, nous présentons l'environnement *Block2Py* pour la programmation fonctionnelle, que nous avons dérivé d'un outil pour la programmation impérative conçu pour faciliter la transition de la programmation par blocs à la programmation *Python*, outil présenté en atelier à Didapro 8 (Declercq & Nény, 2020).

Nous présentons ensuite une séquence d'activités élémentaires permettant de revisiter des programmes déjà écrits en classe de première (chiffrement, conversion binaire, parcours d'une chaîne, comptage d'éléments…) dans un cadre fonctionnel permettant d'y introduire progressivement la récursivité.

L'expérimentation en classe a été menée à partir de la séquence conçue par les chercheurs, en recueillant toutes les traces des activités des élèves dans l'environnement utilisé.

Les réussites et difficultés des élèves sont analysées et commentées, par rapport aux capacités attendues du programme de terminale dans le thème "Langages et programmation".

Cadre théorique de la recherche

Analyse curriculaire

Programmes de seconde générale et technologique

La notion de fonction est abordée en tant que mécanisme de structuration des programmes et par analogie avec la notion mathématique de fonction. En classe de seconde, l'enseignement de *Sciences Numériques et Technologie* (SNT) comporte, parmi les notions transversales de programmation, la maîtrise des définitions et appels de fonctions. En mathématiques la capacité attendue est : écrire des fonctions simples ; lire, comprendre, modifier, compléter des fonctions plus complexes. Appeler une fonction.

La difficulté didactique que constitue la différence entre une fonction mathématique et une fonction informatique est simplement évoquée à travers la formule : « En programmant, les élèves revisitent les notions de variables et de fonctions sous une forme différente ».

Les documents d'accompagnement proposent des exemples de fonctions qui sont plutôt des procédures modifiant leurs paramètres (tri d'un tableau…), ou donnant un résultat variable selon l'appel (fonctions aléatoires). Le langage *Python* ne distinguant pas fonctions et procédures, l'introduction de fonctions, pour structurer un programme impératif, n'engage pas nécessairement les élèves vers le paradigme de la programmation fonctionnelle.

Programme de la spécialité NSI en terminale

Programmation fonctionnelle et récursivité figurent au programme de terminale dans le thème *Langages et programmation* (MENJ, 2019). L'objectif affiché est de diversifier les paradigmes disponibles pour les élèves.

La récursivité est une méthode fondamentale de programmation. Son introduction permet également de diversifier les algorithmes étudiés. En classe de terminale, les élèves s'initient à différents paradigmes de programmation pour ne pas se limiter à une démarche impérative.

Aucune notion théorique liée au lambda-calcul ou à l'usage ou à l'écriture de fonctions d'ordre supérieur (*map*, *reduce*…) ne figure au programme. Il est donc raisonnable de considérer ces notions comme hors-programme.

Les capacités attendues sont les suivantes :

- Distinguer sur des exemples les paradigmes impératif, fonctionnel et objet.
- Écrire un programme récursif.
- Analyser le fonctionnement d'un programme récursif.

Il n'est pas précisé, si la récursivité doit être étudiée dans le paradigme impératif ou fonctionnel. Nous envisageons ici seulement sa présentation dans le cadre de la programmation fonctionnelle où son introduction est fondamentale. La récursivité peut aussi être enseignée dans un paradigme impératif, en particulier en utilisant le dessin récursif (fractales, flocons de Koch ou triangle de Sierpinski par exemple).

Les usages ultérieurs de la récursivité sont cités dans le programme de terminale.

- L'exemple du tri fusion permet également d'exploiter la récursivité.

- L'introduction des structures d'arbres et de graphes montre tout l'intérêt d'une approche récursive dans la résolution algorithmique de problèmes.

Ces deux thèmes sont difficiles et ne peuvent pas constituer un apprentissage de la récursivité. Nous nous limiterons donc, dans notre expérimentation, à des activités élémentaires d'apprentissage de la programmation fonctionnelle et de la récursivité.

Analyse épistémologique

Le paradigme de la programmation fonctionnelle consiste à écrire le résultat que devra rendre la fonction selon les données fournies. Ce résultat est décrit par une expression dépendant des paramètres de la fonction et est noté en *Python* après le mot clé *return*. Il n'est plus utile ici de décrire une suite d'instructions, comme en programmation impérative. Seul le résultat compte. Si l'écriture de l'expression résultat est trop complexe, il est possible de décomposer le problème et de présenter l'écriture du résultat comme la composition de plusieurs fonctions intermédiaires. Il est aussi possible de décomposer par cas grâce à l'expression conditionnelle.

Pour initier l'élève à la programmation fonctionnelle, nous pouvons supprimer toutes les instructions, l'écriture du résultat d'une fonction ne nécessitant que des expressions. Nous évitons alors en particulier l'instruction d'affectation, caractéristique de la programmation impérative.

Cela permet de revisiter les constructions élémentaires vues en classe de première dans le cadre du paradigme de la programmation impérative.

- La séquence est remplacée par la composition.
- L'instruction conditionnelle est remplacée par l'expression conditionnelle.
- Les constructions itératives sont remplacées par la récursivité.

Concernant les types élémentaires de données, nous nous limitons aux entiers, aux booléens et aux chaînes de caractères. Nous évitons bien sûr tout type de données mutables, dont l'usage sort du paradigme fonctionnel.

Les notions fondamentales en programmation fonctionnelle pour son enseignement au lycée, sont ainsi - outre les définitions et appels de fonctions : l'expression, la composition et la récursion.

Nous rejoignons ainsi les travaux fondateurs de (McCarthy, 1960) sur le langage LISP en les réadaptant en langage *Python*. Pour une analyse épistémologique plus poussée de la récursivité, le lecteur pourra se référer à (Leon et al., 2020).

Analyse instrumentale

D'un point de vue didactique, il faut souligner que l'introduction de la programmation fonctionnelle et de la récursivité nécessite un changement de conception de la machine d'exécution. En programmation impérative, le programmeur pense la machine comme capable par des instructions de modifier une mémoire. En programmation fonctionnelle, le programmeur pense la machine comme capable d'évaluer des appels de fonction à partir d'expressions de résultats préalablement définis, d'où l'intérêt de la console dans ce paradigme.

La console, dans le cas d'un langage interprété est aussi parfois nommée REPL pour *Read-Eval-Print-Loop*. Pour s'en convaincre il suffit de tester le programme *Python* suivant permettant de programmer en *Python* une console *Python* qui lit une expression, l'évalue et imprime son résultat.

```
while True:
    print(eval(input(">>>>")))
```

Chacun comprend ainsi mieux pourquoi l'utilisation de `print` et de `input` est inutile, quand on dispose d'une console, puisque c'est précisément le travail que fait la console, permettant à l'apprenti programmeur de se concentrer sur l'écriture et l'appel de fonctions.

Nous pouvons aussi en déduire l'absurdité d'une consigne qui interdirait en général en informatique l'usage du `print`, puisque cela interdirait la possibilité de programmer un interprète ou tout autre programme interactif.

Nous retiendrons que la console est l'instrument, au sens de (Rabardel, 1995), le plus naturel en programmation fonctionnelle, puisque qu'il permet, dès l'écriture d'une fonction, de la tester en l'évaluant avec différentes valeurs de ses paramètres, sans avoir besoin d'un programme principal demandant à l'utilisateur les valeurs choisies et imprimant le résultat de l'appel de la fonction. Ce programme principal - et universel - existe déjà : c'est la console.

Il semble par contre inutile, en initiation à la programmation fonctionnelle, de détailler comment une machine à instructions et à mémoire pourrait exécuter un programme récursif à l'aide d'une pile. Ce problème relève de la compilation et n'est pas au programme de la classe de terminale.

Le seul retour instrumental utile à la compréhension de l'évaluation de l'appel d'une fonction récursive, est la visualisation de la trace d'exécution montrant les paramètres d'entrée et les résultats de chacun des appels. Cette visualisation est disponible dans de nombreux environnements de programmation (*Thonny* par exemple) et ne nécessite pas d'expliciter le stockage des paramètres sur une pile. Nous la proposons aussi dans l'environnement *Block2Py.*

Proposition et méthodologie

L'environnement Block2Py version programmation fonctionnelle

Au vu des choix didactiques énoncés précédemment, l'environnement *Block2Py* a été conçu comme un éditeur de programmes par blocs - ne permettant de construire que des programmes fonctionnels - auquel nous avons adjoint une console.

Le langage a été doté de la définition et de l'appel de fonctions, et des constructions permettant d'écrire des expressions donnant des résultats entiers, booléens ou chaînes de caractères. Ceci suffit largement à donner à ce tout petit sous-ensemble de *Python*, la puissance d'une machine de Turing pour décrire des fonctions calculables (Figure 1).

Le détail des constructions disponibles est le suivant :

- Expressions arithmétiques : `if else`, constantes, opérations binaires `+ - * ** % //`, opération unaire `-`, fonctions `int`, `len`, et `ord`.

- Expressions logiques : `True`, `False`, `or`, `and`, `not`, opérateurs de comparaison `==` `!=` `<` `>` `<=` `>=`
- Expressions de chaînes : `if else`, constante, `+`, fonctions `str` et `chr`, indexation `[]`, sous-chaîne `[:]`

Pour permettre la programmation sur des structures de données récursives, il serait possible d'y ajouter la notion de tuple, pour décrire listes, piles, files, arbres ou graphes.

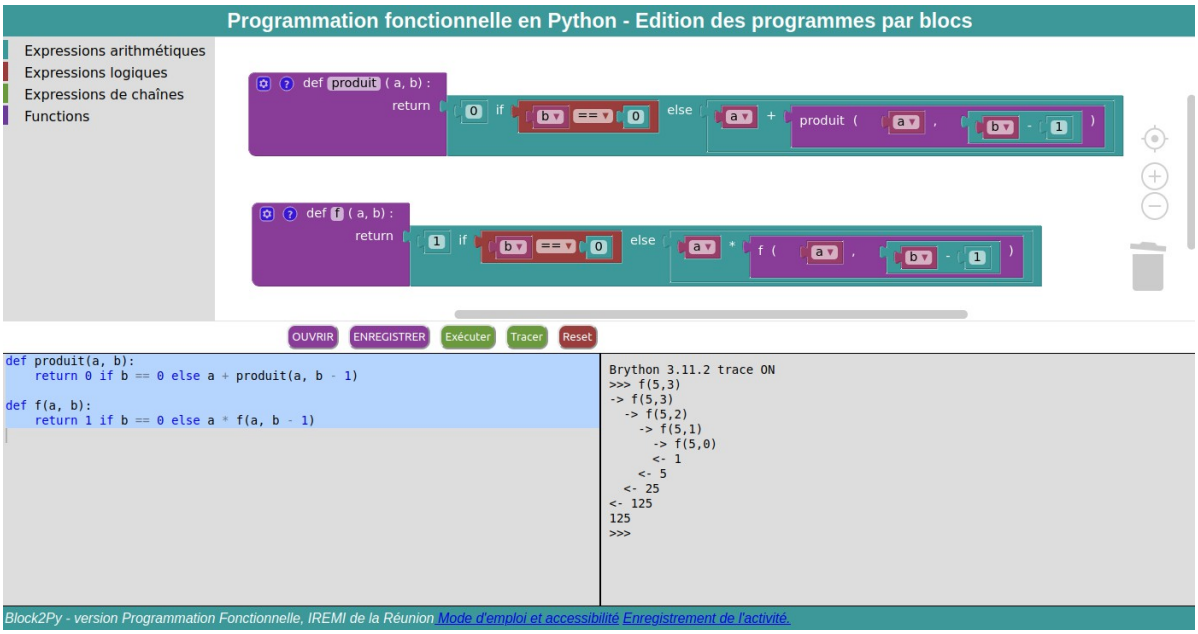

Figure 1 : Interface de *Block2py* version programmation fonctionnelle

L'interface comporte une zone de saisie de blocs en haut. Cette zone est large car les expressions ont tendance à s'étaler horizontalement, au contraire des instructions en séquence. La zone en bas à gauche contient en permanence une représentation textuelle de l'ensemble des fonctions définies par blocs. L'interface comporte enfin un interprète - initié par le bouton *Exécuter* - permettant à l'utilisateur d'appeler les fonctions définies avec les valeurs de son choix, ou par le bouton *Tracer*, s'il souhaite une trace de leur exécution. L'interprète a été réalisé grâce au système *Brython* (Quentel, Pierre, s. d.) qui fournit un interprète *Python* en *Javascript*, permettant ainsi l'utilisation de l'environnement avec un simple navigateur, sans besoin d'aucune installation.

Problématique et hypothèses de recherche

La programmation fonctionnelle et la récursivité sont introduites au programme de la spécialité NSI en terminale alors que la programmation impérative avec des fonctions est enseignée dès la classe de seconde. Nous postulons que cette confusion est à la source des difficultés des élèves au moment d'introduire la récursivité. En effet, la maîtrise de la programmation fonctionnelle nous semble un préalable à l'apprentissage de la récursivité.

L'analyse a priori montre que l'instrument proposé contraint les élèves à penser leur solution en termes d'expression du résultat plutôt qu'en termes d'instructions pour y parvenir. Au-delà

de la prise en main de l'outil, nous postulons que la console va aider les élèves dans leur apprentissage de la programmation fonctionnelle, en particulier en leur permettant d'en construire une représentation correcte.

Les hypothèses de recherche que nous souhaitons vérifier sont les suivantes :

- l'apprentissage de la récursivité en programmation fonctionnelle peut être mené de manière élémentaire, après un premier apprentissage de la programmation fonctionnelle,
- l'usage de la console contribue à construire chez l'élève une représentation correcte de la machine fonctionnelle,
- l'usage de la trace favorise l'analyse puis l'écriture de programmes récursifs.

Par ailleurs nous posons, dans le cadre de cette étude, les postulats suivants que nous ne chercherons donc pas à vérifier :

- l'apprentissage de la programmation fonctionnelle et de la récursivité est plus simple en proscrivant les données mutables,
- l'apprentissage dissocié de la programmation fonctionnelle et de la programmation impérative est un préalable avant d'envisager le mélange des deux paradigmes.

C'est en fonction du premier postulat que nous avons écarté d'emblée l'idée de traiter la récursivité sur des tableaux ou des structures de données complexes possiblement implémentées en programmation objet. L'expérimentation a été construite pour tenter de valider nos hypothèses de recherche.

Expérimentation et résultats

Conception d'une séquence d'activités pour la classe de terminale

Nous proposons une initiation à la programmation fonctionnelle, permettant de reformuler, dans le cadre du paradigme de la programmation fonctionnelle, quelques solutions à des problèmes étudiés en classe de seconde ou de première (chiffrement, conversion binaire, parcours d'une chaîne, comptage d'éléments…).

La séquence a été pensée pour aborder d'abord la programmation fonctionnelle puis la récursivité tout en développant les capacités prévues au programme. Parmi ces capacités, les deux suivantes - « distinguer les paradigmes impératif et fonctionnel », « analyser le fonctionnement d'un programme récursif » - font appel à la compétence évaluer. La capacité « écrire un programme récursif » fait appel aux compétences anticiper et décomposer (Declercq, 2022).

La première activité consiste à évaluer des programmes fournis, puis à les catégoriser en programmes impératifs et/ou fonctionnels. Cette activité permet d'introduire en particulier l'expression conditionnelle et la composition de fonctions. Elle permet aussi de tester le recours à la console par les élèves pour évaluer les programmes proposés.

La seconde activité consiste à évaluer des programmes récursifs. Elle est dédiée à l'apprentissage de la capacité « analyser le fonctionnement d'un programme récursif ».

La troisième activité consiste à faire écrire des programmes récursifs. Les problèmes abordés sont spécifiés sans faire référence à une version itérative, en précisant uniquement les résultats attendus et le cas échéant la relation de récurrence pouvant être utilisée.

Description détaillée de la séquence et de son déroulement

Les deux premières activités ont été réalisées pendant une première séance de deux heures, la dernière lors d'une seconde séance d'une heure, en terminale au lycée Le Verger à Sainte Marie (La Réunion).

Activité 1
Dans cette première activité, les élèves étaient amenés à saisir des fonctions par blocs :

- soit à partir du schéma de blocs (pour la fonction `max2`)

- soit à partir du code Python (pour la fonction `chiffrer`)

```
def chiffrer(caractere, cle):
    return chr(((ord(caractere) - ord("a")) + cle) % 26 + ord("a"))
```

puis à les exécuter et à évaluer des appels depuis la console.
Ensuite, ils devaient distinguer à partir des exemples, le paradigme impératif et le paradigme fonctionnel. Pour cela, à partir de cinq programmes donnés, il fallait identifier et repérer des constructions spécifiques à l'un ou l'autre des paradigmes, ou communes aux deux. Afin de garder une trace écrite, les élèves ont complété un document-réponse en cochant les constructions, qui selon eux, interviennent dans chacun des cinq programmes proposés. Pour conclure cette première activité, les élèves devaient nommer le paradigme de chacun de ces programmes.

Activité 2
L'objectif de la deuxième activité consistait à analyser le fonctionnement d'un programme récursif. Comme l'outil *Block2Py* pour la programmation fonctionnelle propose de visualiser la trace de l'exécution de la fonction, il a été demandé aux élèves de saisir par blocs la fonction produit, puis de comparer la trace des appels de : `produit(4, 7)` et de : `produit(7, 4)`.

Pour bien comprendre la trace de l'exécution d'une fonction, il a été également demandé aux élèves de saisir par blocs la fonction `f` (voir copie d'écran figure 1) et de comparer la trace des appels de : `f(3, 5)` et de : `f(5, 3)`.

Pour terminer cette première séance, les élèves devaient tester une fonction récursive `base2` qui convertit un nombre entier en son écriture en base 2 sous forme de chaîne de caractères.

Activité 3
Tout d'abord, l'élève devait écrire une fonction `somme` qui, étant donné un entier `n` positif, calcule la somme : `1 + 2 + ... n`. Il était proposé aux élèves de tester la fonction avec `somme(9)` qui renvoie : `45` et `somme(10)` qui renvoie : `55`.

Puis l'élève devait écrire une fonction `repete` qui, étant donnés un caractère `c` et un entier positif `n`, renvoie une chaîne de caractères répétant `n` fois le caractère `c`. Un exemple d'usage était donné avec l'appel : `repete('a', 10)` devant renvoyer : `'aaaaaaaaaa'`.

Pour terminer cette activité, les élèves devaient écrire une fonction récursive sur une chaîne de caractères calculant sa longueur.

Pour les élèves les plus rapides, était proposée une dernière fonction à écrire ou à terminer à la maison : la fonction `bin2int`, qui étant donné une chaîne de caractères composée de 0 et de 1, calcule le nombre entier correspondant à l'écriture binaire donnée.

Recueil de données

A l'issue de chacune des séances, les traces d'activités des élèves ont été collectées via l'interface de programmation par le lien "Enregistrement de l'activité". Les enregistrements des deux séances d'un même élève ont été concaténés, puis ils ont été anonymisés. Sur les 14 élèves de la classe, 13 étaient présents à la première séance et 12 à la seconde. 11 élèves étaient présents aux deux séances. Nous effectuons dans la suite une analyse essentiellement qualitative, le faible nombre ne permettant pas d'analyses quantitatives approfondies.

Les traces d'activités des élèves
Chaque enregistrement d'activité contient un horodatage de chacune des actions sur l'interface de commande de *Block2Py* (*Exécuter*, *Tracer*, *Reset*) avec, à chaque fois, l'enregistrement de l'état courant du programme et de la console.

Les programmes proposés par les élèves ont été testés par rapport aux spécifications des fonctions demandées, ce qui nous a permis de noter les heures de réussite de chaque activité, le temps passé ainsi que les fonctionnalités utilisées (*Exécuter*, *Tracer*).

Discussions

Hypothèse sur l'apprentissage élémentaire

L'examen qualitatif des enregistrements de la première activité montre que tous les élèves ont réussi à saisir les premières fonctions proposées (`max2` et `chiffrer`) et ont pu ainsi avoir une première approche de la programmation fonctionnelle. Les seules difficultés notables rencontrées concernent l'usage de la console détaillé dans la prochaine partie, et la priorité des opérateurs, cinq élèves ayant mal imbriqué les blocs de l'expression : `(... + cle) % 26 + ord ("a")`. Les rendus sur papier de cette première activité ont montré que tous les élèves ont bien distingué les constructions utilisées respectivement en programmation impérative et fonctionnelle.

La deuxième activité (évaluation de programmes récursifs) a pu se dérouler dans un contexte où tous les élèves avaient déjà pu saisir et évaluer des fonctions simples. Leur quasi-totalité ont pu évaluer la fonction `produit`, la plupart utilisant la possibilité de tracer les appels récursifs. La suite de cette activité a été diversement suivie selon le temps disponible en cette fin de première séance (6 sur 13 pour la fonction `f`, un seul élève pour la fonction `base2`).

La deuxième séance a débuté avec la troisième activité pour tous les élèves. Sur les 12 élèves présents à cette séance, 10 ont réussi à programmer correctement la fonction `somme`, dont 6 ont réussi aussi à programmer correctement la fonction `repete`.

Nous pouvons donc en déduire que cette expérience conforte notre hypothèse, qu'il est possible, en se consacrant d'abord à la programmation fonctionnelle, puis à l'évaluation de fonctions récursives simples, d'amener les élèves à écrire correctement leurs premières fonctions récursives. La validation définitive de cette hypothèse demanderait cependant une expérience de plus grande ampleur, ce que nous envisageons de faire ultérieurement.

Hypothèse sur l'usage de la console

L'usage de la console lors de la première séance a montré qu'une partie des élèves ne maîtrisaient pas initialement cet instrument.

Deux types d'erreurs ont été observés lors du test de la fonction `max2` :

- saisie des paramètres dans la console, sans le nom de la fonction : `5, 8`
- usage de la fonction `print` : `print(max2(5, 8))`

Après quelques erreurs, tous les élèves ont réussi à appeler au moins une fois une de leurs fonctions. La difficulté de compréhension de la fonction `chiffrer` a incité certains élèves à utiliser la console, par exemple pour l'élève 3 :

```
>>> ord("b")
98
>>> ord("a")
97
>>> 98-107
-9
>>> -9%(26+97)
114
```

La console a ensuite été utilisée plus systématiquement par les élèves pour tester les fonctions `produit` et `f`.

L'enregistrement suivant de l'élève 03 montre une première écriture erronée de la fonction `somme`, suivie d'un appel de la fonction depuis la console permettant de visualiser le résultat erroné, suivi immédiatement d'une deuxième version correcte et d'un appel de la fonction donnant alors un résultat correct.

```
2023-02-28 07:51:37.895000 executer
def somme(n):
    return 1 if n == 0 else n + somme(n - 1)

Brython 3.10.4 Trace OFF
>>> somme(9)
46
2023-02-28 07:52:17.802000 executer
def somme(n):
    return 0 if n == 0 else n + somme(n - 1)

Brython 3.10.4 Trace OFF
>>> somme(9)
45
```

Nous en déduisons que le retour instrumental procuré par la console permet aux élèves de comprendre progressivement, d'abord comment l'utiliser, puis de s'en servir pour tester et éventuellement corriger leurs fonctions.

Cela semble confirmer notre hypothèse selon laquelle la console est un instrument fondamental en programmation fonctionnelle, dans le sens où elle aide à faire comprendre ce que peut faire une « machine fonctionnelle ».

Hypothèse sur l'usage de la trace

L'usage de la trace d'appels d'une fonction récursive était suggéré dans l'énoncé de l'activité 2 pour l'analyse d'une fonction récursive donnée. Notre objectif était d'outiller les élèves, pour les aider à comprendre comment peut fonctionner l'appel d'une telle fonction, dans l'espoir de les voir l'utiliser ensuite spontanément lors de l'écriture de fonctions récursives.

Pendant l'activité 3, lors de l'écriture de la fonction `somme`, sur les 10 élèves qui ont finalement réussi l'activité, 8 ont utilisé la console dont 5 avec la trace. Plusieurs élèves ont utilisé la trace simplement pour confirmer l'écriture correcte de la fonction. D'autres, comme l'élève 10, s'en sont servis pour comprendre et corriger une programmation erronée.

```
2023-02-28 07:48:06.427000 tracer
def somme(n):
   return 0 if n == 0 else somme(n - 1)

Brython 3.10.4 on Netscape 5.0 (X11; Ubuntu) Trace ON
>>> somme(4)
-> somme(4)
  -> somme(3)
    -> somme(2)
      -> somme(1)
        -> somme(0)
        <- 0
```

```
    <- 0
   <- 0
  <- 0
<- 0
0
2023-02-28 07:49:40.472000 executer
def somme(n):
    return 0 if n == 0 else n + somme(n - 1)
```

Nous en déduisons, qualitativement, que l'usage de la trace d'appels d'une fonction récursive favorise l'analyse voire l'écriture de programmes récursifs. Nous ne pouvons pas conclure pour les élèves ayant écrit directement une solution correcte avant de la tester, mais au moins pour ceux ayant écrit une fonction dont l'erreur a pu être mise en évidence par la trace.

Dans notre expérience, il semblerait que les élèves qui ont le plus utilisé la console et la trace pour la fonction `somme` sont ceux qui ont le mieux réussi la fonction suivante. Notre échantillon est cependant trop réduit pour conclure à une corrélation. Notre interprétation est que les élèves vont chercher un retour instrumental destiné à les aider, à la première situation où ils ont la nécessité de dépasser un obstacle didactique. Ensuite leur meilleure compréhension et/ou instrumentation leur permettrait de continuer plus aisément leur progression. Nous ne pouvons cependant pas écarter le possible biais d'un usage plus important de la console par les élèves qui maîtrisent le mieux la programmation en général dès le début de l'activité.

Conclusions et perspectives

Cette recherche exploratoire présente à la fois un nouvel instrument pour l'apprentissage de la programmation fonctionnelle et de la récursivité en spécialité NSI en terminale, et l'analyse des premières expérimentations réalisées en classe avec cet instrument.

Les enregistrements réalisés permettent de conforter les hypothèses émises à la fois sur l'organisation de la situation d'apprentissage et sur l'intérêt des retours instrumentaux proposés par la console et la possibilité de tracer les appels récursifs.

La validation de nos hypothèses demanderait maintenant une expérience de plus grande ampleur, ce que nous envisageons de faire à la fois en répétant l'expérience dans plusieurs classes de terminale et avec une cohorte d'étudiants en première année de licence d'informatique.

Les enregistrements ont aussi incidemment permis de mettre en évidence des améliorations nécessaires de l'outil dans des cas d'erreurs d'élèves qui n'avaient pas été envisagés avant l'expérience (erreurs de type dans les expressions). Le passage à la dernière version de *Brython* a permis depuis d'améliorer la lisibilité des messages d'erreurs émis dans ces cas. L'édition par blocs ne permet en effet de contrôler les types que par rapport aux constantes et aux opérateurs, les paramètres des fonctions étant actuellement non typés.

Des travaux ultérieurs seront ensuite à mener pour mesurer l'impact de ces premiers apprentissages sur l'utilisation de la récursivité dans les thèmes les plus avancés du programme de terminale NSI à savoir l'algorithmique sur les arbres et le tri fusion. Ce dernier point s'an-

nonce le plus délicat dans la mesure où il impose l'usage de structures mutables - les tableaux *Python* - pour les trier.

Références

Declercq, C. (2022). *Didactique de l'informatique : Une formation nécessaire*. Sticef, 28(3). http://sticef.org/num/vol2021/28.3.8.declercq/28.3.8.declercq.htm

Declercq, C., & Nény, F. (2020, février). *Block2Py, un éditeur de blocs pour l'apprentissage du langage Python*. Didapro 8, L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation. https://hal.archives-ouvertes.fr/hal-02526883

Leon, N., Modeste, S., & Durand-Guerrier, V. (2020, septembre). *Récurrence et récursivité : Analyses de preuves de chercheurs dans une perspective didactique à l'interface mathématiques-informatique*. INDRUM 2020. https://hal.archives-ouvertes.fr/hal-03113854

McCarthy, J. (1960). *Recursive functions of symbolic expressions and their computation by machine, Part I.* Communications of the ACM, 3(4), 184-195. https://doi.org/10.1145/367177.367199

MENJ. (2019). *Programme de l'enseignement de spécialité de numérique et sciences informatiques de la classe terminale de la voie générale*. Ministère de l'Education Nationale et de la Jeunesse. https://www.education.gouv.fr/bo/19/Special8/MENE1921247A.htm

Quentel, Pierre. (s. d.). *Brython, Une implémentation de Python 3 pour la programmation web côté client*. Consulté 23 mars 2023, à l'adresse https://brython.info/index.html

Rabardel, P. (1995). Les hommes et les technologies; approche cognitive des instruments contemporains. Armand Colin. https://hal.archives-ouvertes.fr/hal-01017462