Découverte instrumentée de la programmation fonctionnelle et de la récursivité en terminale NSI

Christophe Declercq, INSPE de la Réunion, LIM, IREMI

2 mars 2022

Introduction

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. On postule que cette confusion est à la source des difficultés des élèves au moment d'introduire la récursivité.

On propose une initiation à la programmation fonctionnelle, permettant de reformuler, dans le cadre de ce paradigme, quelques solutions à des problèmes étudiés en classe de seconde ou de première.

On adopte le point de vue de l'analyse instrumentale, pour expliquer pourquoi le choix de l'instrument est fondamental : la console est depuis les travaux sur le langage Lisp [1] reconnue comme l'instrument adapté à l'évaluation interactive d'appels de fonctions alors que d'autres IDE sont mieux adaptés à l'exécution de programmes impératifs.

On propose, pour instrumenter l'apprentissage, un environnement d'édition de programmes par blocs, permettant aux élèves de ne saisir que des programmes fonctionnels corrects. On justifie les choix ergonomiques effectués par la nécessité pédagogique de séparer les difficultés : on évite ainsi tous les pièges qu'ils pourront rencontrer ensuite lors de l'imbrication des deux paradigmes impératif et fonctionnel dans un même programme Python.

L'environnement Block2Py version programmation fonctionnelle a été construit à partir de la version originale qui était dédiée à la programmation impérative [2]. Le nouvel environnement est disponible à l'adresse : `https://iremi974.gitlab.io/block2py`

Programmation fonctionnelle et récursivité dans les programmes officiels

En classe de terminale

Programmation fonctionnelle et récursivité figurent au programme de terminale dans le thème **Langages et programmation** [3]. 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 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 devait être étudiée dans le paradigme impératif ou fonctionnel. On envisage 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).

En classe de seconde et de première

La notion de fonction est abordée en tant que mécanisme de structuration des programmes et par analogie avec la notion mathématique de fonction. En SNT, les notions transversales de programmation incluent 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.

La programmation fonctionnelle

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.

Des activités d'initiation à la programmation fonctionnelle

Une activité élémentaire - dès la classe de seconde - peut consister en l'écriture de la fonction valeur absolue. On donne ici son expression fonctionnelle, même si en seconde on peut aussi l'écrire avec une instruction alternative et deux `return`.

```
def valeurabsolue(x):
    return x if x >= 0 else -x
```

On peut aussi proposer aux élèves d'écrire une fonction renvoyant le maximum de deux nombres, puis de quatre nombres, ce qui permet d'introduire la composition de fonctions.

```
def max2(x, y):
    return x if x >= y else y


def max4(a, b, c, d):
    return max2(max2(a, b), max2(c, d))
```

Il est utile de proposer des exemples manipulant autre chose que des nombres, pour s'écarter de la notion mathématique de *fonction d'une variable réelle*. On peut par exemple proposer une activité de chiffrement permettant de chiffrer un caractère avec un décalage donné par un entier selon le code de César.

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

Les fonctions définies par cas ou par intervalles sont des sources d'exemples d'utilisation des expressions conditionnelles.

```
def prix_timbre_en_centimes(poids):
    return 116 if poids <= 20 else (232 if poids <= 100
                                    else (400 if poids <= 250 else 600))
```

Les programmes de mathématiques sont une source naturelle d'exemples de fonctions à programmer, mais les autres disciplines peuvent être mises à contribution comme indiqué dans le programme de SNT : *Écrire et développer*

des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

La console, un instrument pour la programmation fonctionnelle

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(">>>>")))
```

On 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.

On peut 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.

On retiendra que la console est l'instrument 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.

Premiers choix didactiques

Dans le cadre de la conception d'un environnement d'édition par blocs pour le langage Python, le concepteur a la possibilité de choisir les constructions disponibles. Si l'on souhaite initier l'élève à la programmation fonctionnelle, on peut supprimer toutes les instructions, l'écriture du résultat d'une fonction ne nécessitant que des expressions.

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, on peut se limiter aux entiers, aux booléens et aux chaines de caractères. On évite bien sûr tout type de données mutables, dont l'usage sort du paradigme fonctionnel.

La récursivité

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.

Pour cela, on doit disposer de données ayant une structure récursive : c'est le cas des entiers grâce à la définition de Peano et des chaînes de caractères. En Python ces deux types de données ne sont pas mutables et s'intégrent donc parfaitement au paradigme fonctionnel.

On peut apprendre à programmer récursivement la fonction `longueur` sur des chaînes de caractères, ce qui permet d'éviter l'introduction problématique de tableaux mutables ou d'une classe spécifique pour les listes. On amène ainsi l'élève à apprendre à penser récursivement la solution à un problème sans le confronter à des obstacles complexes dus à la délicate intégration, dans le langage Python des trois paradigmes impératif, fonctionnel et objet et en particulier au problème de la mutabilité.

```
def longueur(chaine):
    return 0 if chaine == "" else 1 + longueur(chaine[1:])
```

Les activités de programmation récursive les plus accessibles sont celles où le calcul se base sur la structure des données. Pour une chaîne de caractère, il faut pouvoir rappeler la fonction sur une sous-chaine. Pour les fonctions récursives d'un entier `n`, les plus élémentaires, sont celles où l'appel récursif s'applique à `n-1`.

```
def produit(a, b):
    return 0 if b == 0 else a + produit(a, b - 1)
```

On peut aussi revisiter quelques algorithmes vues en première, comme la conversion d'un entier en binaire (noté dans une chaîne de caractères) :

```
def base2(n):
    return "1" if n == 1 else base2(n // 2) + str (n % 2)
```

ou la conversion inverse d'une représentation binaire à un entier :

```
def bin2int(chaine):
    return 0 if chaine == "0" else (1 if chaine == "1"
    else bin2int(chaine[:-1]) * 2 + bin2int(chaine[-1]))
```

Pour programmer des fonctions récursives à résultat booléen, il n'est pas utile de disposer d'une expression conditionnelle, les opérateurs booléens séquentiels font l'affaire à condition de prendre la précaution de reporter l'appel récursif au plus tard, comme on peut le voir sur la fonction `pair` qui teste la parité d'un entier naturel.

```
def pair(n):
    return n == 0 or n != 1 and pair(n - 2)
```

Dans tous ces exemples, la conception d'une solution récursive à un problème consiste à rechercher un ou des cas de base pour lesquels on peut donner directement le résultat, puis à tenter de s'en rapprocher par un ou des appels récursifs quitte à opérer sur leur résultat pour obtenir le résultat final.

On ne se préoccupe pas de distinguer récursivité terminale ou non. L'usage de la console doit permettre de convaincre l'élève que l'appel d'une fonction peut se terminer, dès lors que la fonction est définie pour la donnée correspondante, au moment où elle est appelée, et pas nécessairement au moment où l'on écrit la fonction.

Pour les plus dubitatifs, il est possible de montrer que c'est déjà le cas hors de la récursivité, comme on peut le constater sur l'exemple suivant :

```
>>> def g(x) : return f(x) + 1
>>> def f(x) : return x*x
>>> g(5)
26
```

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, on pense la machine comme capable par des instructions de modifier une mémoire. En programmation fonctionnelle, on pense la machine comme capable d'évaluer des appels de fonction à partir d'expressions résultats préalablement définis. D'où l'intérêt déjà cité de la console dans ce paradigme.

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 fonctionnel à l'aide d'une pile. Ce problème relève de la compilation et n'est pas au programme de la classe de terminale.

L'environnement Block2Py version programmation fonctionnelle

Au vu des choix didactiques énoncés précédemment, l'environnement Block2Py 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 caractère. 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.

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 simplifier la programmation sur des structures de données récursives, il est possible d'y adjoindre la seule notion de tuples, pour permettre de décrire listes, piles, files, arbres ou graphes.

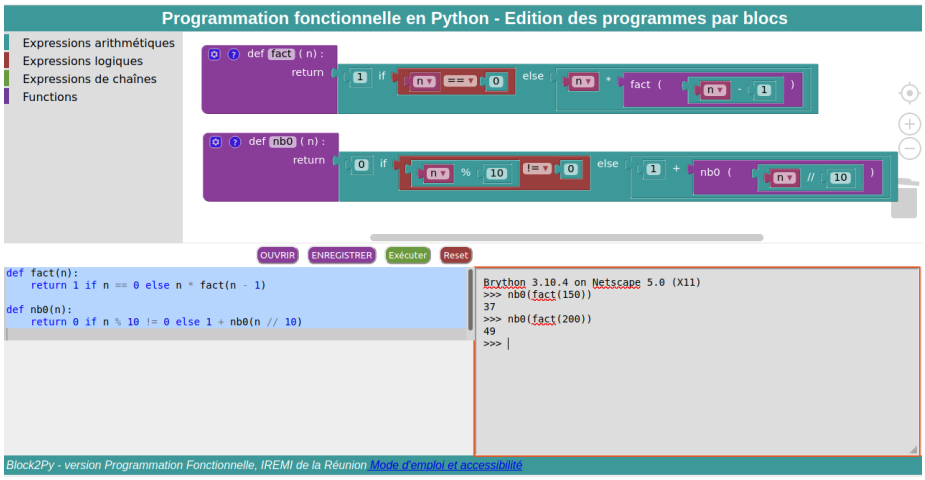

Figure 1: Interface de Block2py version programmation fonctionnelle

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.

L'interprète a été réalisé grâce au système Brython [4] qui fournit un interpréte Python en Javascript, permettant ainsi l'utilisation de l'environnement avec un simple navigateur, sans besoin d'aucune installation.

Conclusion

On a présenté un instrument pour l'apprentissage élémentaire de la programmation fonctionnelle et de la récursivité ainsi que des exemples d'usage. L'ensemble des fonctions présentées ici ont été définies avec cet environnement.

On envisage maintenant des expérimentations en classe, pour tester l'ergonomie de l'environnement, sa possible appropriation par les élèves et son intérêt pour l'apprentissage élémentaire de la programmation fonctionnelle et de la récursivité.

On pourra alors tester les hypothèses émises :

- l'apprentissage de la programmation fonctionnelle et de la récursivité est plus simple si on prend soin d'éviter d'y mêler programmation impérative et programmation objet et si on proscrit les données mutables,
- l'instrument proposé contraint les élèves à penser leur solution en terme d'expression du résultat plutôt qu'en terme d'instructions pour y parvenir,
- l'usage de la console contribue à construire chez l'élève une représentation correcte de la machine fonctionnelle,
- l'apprentissage dissocié de la programmation fonctionnelle et de la programmation impérative est un préalable avant d'envisager le mélange des deux paradigmes.

On remarquera que ces réflexions ne datent pas d'hier. Dans les années 1990, au moment de l'introduction de l'informatique dans les premiers cycles universitaires, le débat jamais tranché de savoir quel paradigme introduire d'abord, avait au moins permis de démontrer qu'il fallait éviter d'introduire les deux ensemble.

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

Références

[1] John McCarthy, Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, Massachusetts Institute of Technology, Cambridge, Mass. April 1960

[2] Christophe Declercq, Florence Nény. Block2Py, un éditeur de blocs pour l'apprentissage du langage Python. Didapro 8 – DidaSTIC, Feb 2020, Lille, France. https://hal.archives-ouvertes.fr/hal-02526883

[3] Programmes et ressources en numérique et sciences informatiques—Voie G. éduscol, Ministère de l'Éducation nationale, de la Jeunesse et des Sports - Direction générale de l'enseignement scolaire. Consulté à l'adresse https://eduscol.education.fr/2068/programmes-et-ressources-en-numerique-et-sciences-informatiques-voie-g

[4] Brython, Une implémentation de Python 3 pour la programmation web côté client.Consulté à l'adresse https://www.brython.info/