Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique

Jean-Baptiste Lagrange, Janine Rogalski

Article paru dans ANNALES de DIDACTIQUE et de SCIENCES COGNITIVES, Revue internationale de didactique des mathématiques, Volume 22 (2017)

Citer LAGRANGE, J.B., ROGALSKI, J. (2017) Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique, in ANNALES de DIDACTIQUE et de SCIENCES COGNITIVES, 22, 119-158.

JEAN-BAPTISTE LAGRANGE ET JANINE ROGALSKI

SAVOIRS, CONCEPTS ET SITUATIONS DANS LES PREMIERS APPRENTISSAGES EN PROGRAMMATION ET EN ALGORITHMIQUE

Abstract. In several countries including France, there is a growing interest for the teaching and learning of algorithmics and programming at school and college level. It is then necessary to question the objectives of this teaching and learning, and to propose controlled implementations. This article, written by a researcher in cognitive ergonomics and a researcher in didactics, aims at assessing some research results in this field, on the basis of research work conducted sporadically since thirty years. It first attempts to show the permanence of questions related to beginners' conceptual difficulties, and then tackles the issue of learning situations. Then it takes stock of results obtained in psychology of programming, focusing on a conceptual field precisely identified around the concept of computer variable. The conclusion gives evidence of a broad field of research now open.

Résumé. Dans de nombreux pays, dont la France, l'intérêt se porte actuellement sur l'enseignement de l'algorithmique et de la programmation aux niveaux scolaire et pré-universitaire, et la nécessité se fait sentir de travaux questionnant les objectifs d'un tel enseignement et proposant des mises en œuvre et des évaluations. Cet article, écrit par un chercheur en psychologie cognitive et un didacticien des mathématiques, vise à un premier bilan sur la recherche dans ce domaine en s'appuyant sur des travaux menés de façon sporadique depuis une trentaine d'années. Il s'attache d'abord à montrer la permanence de questions posées par les difficultés conceptuelles des élèves débutants, puis étudie la question des situations d'apprentissages. Ensuite il fait le point sur les acquis de la psychologie de la programmation, en se centrant sur un champ conceptuel précisément identifié, celui de « la variable informatique ». La conclusion montre qu'un champ de recherche très large est ouvert.

Mots-clés. Savoirs, concepts et situations en algorithmique, difficultés conceptuelles, élèves débutants, situations didactiques, variable informatique, psychologie de la programmation.

Introduction

En France, l'enseignement de l'algorithmique et de la programmation est revenu récemment dans le curriculum scolaire des mathématiques[1]. Précisons d'emblée que le curriculum de mathématiques de 2009 pour la classe de Seconde (élèves de 15 ans) nous semble illustrer la difficulté à opérer une distinction nette entre algorithmique et programmation : « Ce qui est proposé [pour la démarche algorithmique] est une formalisation en langage naturel propre à donner lieu à traduction sur une calculatrice ou à l'aide d'un logiciel. ». L'indifférenciation entre algorithmique et programmation s'exprime aussi dans les capacités attendues des élèves : « écrire une formule permettant un calcul », « écrire un programme calculant et donnant la valeur d'une fonction, ainsi que les instructions d'entrée et sortie nécessaires au traitement » ; « programmer un calcul itératif » ; « programmer une instruction conditionnelle ». L'incertitude quant à la dualité algorithmique / programmation se retrouve dans les représentations exprimées par les enseignants et dans leur pratique (Couderette, 2016 ; Haspékian & Nijimbéré, 2016)[2]. Notre postulat dans cet article est que l'on peut porter un regard didactique et psychologique sur les savoirs, concepts et situations en jeu, sans poser l'élucidation de cette dualité comme un préalable.

Dans d'autres pays, l'intérêt se porte aussi sur ce type d'enseignement au niveau scolaire (par exemple en Uruguay : daRosa, 2015) ou dans les premières années d'université (par exemple au Canada : Broley, 2016). Ces deux exemples illustrent que, dans ces pays aussi, algorithmique et programmation sont peu distinguées. Les exemples montrent aussi que le développement de ce type d'enseignement peut être soutenu par des travaux en didactique questionnant ses objectifs et proposant des mises en œuvre et des évaluations. Cependant, comme nous l'avons noté (Lagrange, 2014) la recherche en didactique sur l'enseignement de l'algorithmique et de la programmation reste encore très peu développée ; en effet, de tels

---

[1] L'informatique comme science ne se réduit certes pas à l'algorithmique et la programmation, mais c'est en revanche le cas dans les programmes de mathématiques de lycée, le niveau essentiellement considéré dans cet article. Nous n'avons pas d'éléments pour anticiper les changements que pourraient produire les introductions de l'informatique dans les niveaux antérieurs (primaire et collège).

[2] Une étude de la dualité algorithmique/programmation dans les ouvrages universitaires de référence dépasserait le cadre de notre article. Notons seulement que Knuth (2011), contraste algorithme et programme sur la dimension abstrait / concret : *Les algorithmes sont des procédures de calcul abstraites pour transformer des informations ; les programmes en sont leurs incarnations concrètes* (p. 317).

enseignements ont existé depuis plus de trente ans dans différents pays et ont donné lieu à diverses recherches, mais leur caractère sporadique n'a pas permis pas de capitaliser des acquis, ce à quoi cet article souhaite remédier.

Cet article est écrit par deux chercheurs travaillant dans le même laboratoire, mais appartenant à des champs différents, la didactique des mathématiques et la psychologie cognitive. Dans de nombreux domaines de l'enseignement et de l'apprentissage, un double point de vue impliquant ces deux champs[3] s'est montré pertinent et fructueux. Du point de vue de la didactique, nous considérons ici les apprentissages en programmation et en algorithmique dans un contexte scolaire que nous allons préciser. Du point de vue de la psychologie cognitive, ces apprentissages concernent l'informatique « objet », à la fois champ scientifique et domaine professionnel, et l'informatique « outil » comme technologie. La programmation y est considérée dans un sens large, incluant différents aspects du développement logiciel. L'existence depuis bientôt 30 ans, dans la psychologie cognitive, d'un champ de recherche sur la programmation rend incontournable la prise en compte de ses acquis dans toute recherche y compris sur des débutants.

Les premières parties de cet article se basent sur des recherches menées dans le contexte français à deux époques différentes. Nous rappelons d'abord dans quelles conditions des apprentissages en programmation et en algorithmique ont pu exister antérieurement dans le curriculum scolaire français. Dans une première partie nous recherchons s'il existe une continuité dans les phénomènes concernant les apprentissages en programmation et en algorithmique, en mettant en correspondance des observations sur le comportement des élèves dans le contexte curriculaire récent, avec des analyses dans un contexte plus ancien. De manière similaire nous recherchons comment la question des situations d'apprentissage s'est posée dans différents contextes. Ces deux parties s'appuient sur l'expérience du premier auteur et des travaux d'étudiants en didactique des mathématiques. La troisième partie répond à la nécessité d'élargir le propos au-delà des expériences isolées qui alimentent les deux premières parties, et se place du point de vue du développement de la conceptualisation chez les élèves. Nous recherchons quels acquis de la psychologie de la programmation sont utiles dans le contexte actuel, à travers une présentation de ses acteurs et de son évolution. Nous particularisons sur un champ conceptuel précisément identifié, celui de « variable informatique » un terme que nous allons préciser dans la suite. Pour conclure, nous joignons les deux points de vue pour établir un bilan et souligner les questions qui nous semblent

[3] La Double Approche pour les mathématiques a été développée par Robert et Rogalski (2004). Voir aussi Vandebrouck (2008, 2013).

prioritaires.

D'un contexte à l'autre

Au début des années 1980, l'ordinateur est devenu un objet assez accessible, mais sur les machines grand public ou pour l'enseignement, les usages possibles sont restés limités à la programmation. « *Elementarmathematik vom algorithmischen Standpunkt* » est paru en 1977 et a été traduit en français au début des années 1980 (Engel, 1980). Il proposait que les mathématiques scolaires mettent l'accent sur les algorithmes, leur construction et leur mise à l'épreuve. Un autre livre, « *Mindstorm* » sur les apports de la programmation Logo est paru aussi en français à la même époque (Papert, 1981). Ces ouvrages témoignent de ce que les activités de programmation ont été vues par beaucoup d'innovateurs comme pouvant contribuer aux apprentissages dans les domaines scientifiques existants, les mathématiques en premier lieu. Cela s'est traduit par quelques pratiques et groupes de recherche dans les IREM, sans toutefois qu'il y ait un impact sur le curriculum, au moins en France. En revanche, l'institution a mis en place un enseignement centré sur l'informatique comme champ scientifique, l'option informatique des lycées. Il s'agissait d'offrir aux élèves de lycée un nouveau champ d'étude, en soulignant les liens qu'il entretient avec les autres champs scolaires, avec une forte dimension programmation (Baron, 1989)[4].

Au tournant des années 1990, les activités de programmation ont disparu du curriculum avec l'arrêt de l'option informatique[5]. Un retournement de point de vue a eu lieu : de l'informatique comme discipline « objet » on passe à son utilisation comme « outil ». En mathématiques, le développement de progiciels comme les tableurs, ainsi que des applications dédiées aux mathématiques comme le calcul formel et la géométrie dynamique y ont contribué. Ainsi, Ruthven (1996) écrit :

> Building expertise and fluency in programming requires a

[4] Le volume horaire est alors de 2,5 h par semaine, avec le thème central de construction de programmes. La récursivité est objet d'enseignement en terminale. La dimension sociale de l'informatique y avait également une place explicite. Il est prévu une épreuve d'option au baccalauréat, avec question de cours sur l'informatique, traitement d'un « thème de société », et élaboration d'un programme. Des enseignants formés en « stages lourds » y enseignent, de diverses disciplines – souvent mais pas seulement scientifiques. La dimension sociale de l'informatique y avait également une place explicite.

[5] Le travail avec Logo, engagé hors curriculum en primaire, s'arrête pratiquement aussi.

> considerable investment of time and effort, and imposes significant intellectual demands in its own right. In purely instrumental terms, newer tools, such as the spreadsheet, offer comparable power and versatility with lower overheads of technique and greater interactive flexibility. (p.458)

La résurgence actuelle des activités de programmation en relation avec l'algorithmique peut s'interpréter comme une prise de conscience, 20 ans après, de deux réalités. D'une part les activités avec des progiciels ou des applications dédiées ne sont pas non plus faciles à mettre en œuvre et ne conduisent pas nécessairement à la compréhension des principes essentiels qui sous-tendent leur conception. D'autre part, le besoin de compétences pour la programmation professionnelle appelle des étudiants motivés et suffisamment « débrouillés » en informatique pour suivre les cursus nécessaires. D'où une demande dans de nombreux pays d'un enseignement dès le secondaire « long ».

Observations dans le contexte actuel et dans l'option informatique des années 1980

L'expérience des années 1980 telle qu'évoquée par Ruthven, contredit l'idée d'un accès aisé des élèves à une expertise suffisante pour que les activités en programmation et algorithmique puissent contribuer à des apprentissages. Nous recherchons, à travers des exemples pris dans des travaux de recherche, comment des questions liées la construction de cette expertise se posent dans les deux contextes, et comment elles renvoient à des savoirs et aux situations qu'il est nécessaire de mettre en place pour que les élèves se confrontent à ces savoirs. Nous considérons dans cette première partie deux exemples de difficultés résistantes, déjà observées dans les travaux de l'époque de l'option informatique et retrouvées dans l'introduction de l'algorithmique et de la programmation au lycée dans le curriculum de mathématiques depuis 2009. Le premier exemple concerne les spécificités des langages informatiques, et le second porte plus particulièrement sur le statut donné aux « conditions » dans les langages informatiques.

Comprendre les spécificités des langages informatiques

Les langages informatiques se définissent comme des langages formels utilisés pour l'exploitation d'un système. Nous désignerons par « dispositif », tout système commandé par un tel langage. Un texte syntaxiquement correct écrit avec le langage peut être interprété comme organisant un « traitement », c'est-à-dire une suite de modifications apportées à l'état du dispositif. Dans une large classe de langages, l'état du dispositif à une étape du traitement se décrit comme l'ensemble des valeurs prises par des « variables ». Ces variables sont caractérisées par leur « type ». Nous considérerons ici des types « simples » définis par un ensemble de

valeurs et un jeu d'opérateurs. Nous considérons : 1°) le type « nombre » avec comme ensemble de valeurs les décimaux de taille et de précision donnée et comme opérateurs les opérations arithmétiques (+, -, *, /) ; 2°) le type « chaîne », dont les valeurs sont des assemblages séquentiels de caractères reconnus par le dispositif et les opérateurs des « fonctions » que nous préciserons plus loin ; 3°) le type « booléen » avec deux valeurs (Vrai et Faux) et les connecteurs logiques (négation, conjonction, disjonction) qui opèrent sur ce type de variables.

Nous nous intéressons ici aux langages impératifs, qui sont les langages généralement proposés aux débutants. Ils permettent d'exprimer le traitement à l'aide d'instructions organisées sous forme séquentielle, alternative (`SI <condition> ALORS <instructions> SINON <instructions>`) ou itérative (par exemple `TANT QUE <condition> FAIRE <instructions>`). Dans ces langages, les types nombre et chaine disposent d'instructions de lecture et d'affichage généralement utilisées avec les débutants pour marquer respectivement les données initiales et le ou les résultats[6]. Ces caractéristiques font que les algorithmes ou programmes écrits en langage impératif « ressemblent » à l'expression de traitements « familiers ». Notre hypothèse est que, pour des débutants, cette similarité peut être initialement une aide, mais qu'elle peut aussi constituer un obstacle à la compréhension de propriétés générales des langages informatiques, et de l'expression appropriée des traitements.

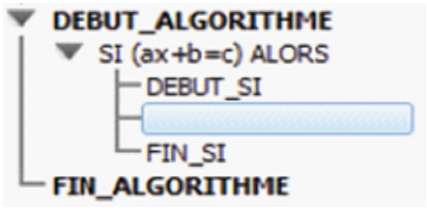
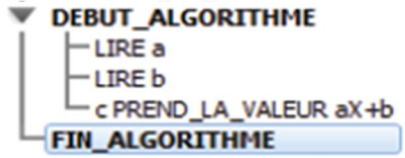
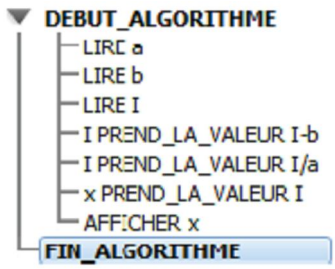
Nous prenons un premier exemple dans la thèse de Briant (2013). L'auteure présente une situation dont l'objectif est de montrer aux élèves que la résolution de certaines équations peut se faire de manière algorithmisée. Une tâche consiste pour les élèves à réaliser un traitement prenant en entrée trois réels `a`, `b` et `c`, et donnant en sortie la solution, quand elle existe et est unique, de l'équation `ax+b=c`, ou un message adéquat dans les autres cas[7]. L'auteure donne une variété de solutions d'élèves, dont certaines sont présentées dans le tableau 1.

---

[6] Dans les exemples qui sont donnés plus loin, les écritures sont produites par les élèves dans différents environnements dont les différences ne sont pas significatives pour notre propos ; les lecteurs reconnaitront l'environnement Algobox (affection notée « prend la valeur », Tableau 1) et les langages LSE (affection notée ←, Tableau 2) et Pascal (affection notée :=, Tableau 3).

[7] Nous ne discutons pas ici la pertinence de la tâche en regard de l'objectif de l'auteure. Nous nous intéressons seulement à l'écart entre la solution attendue et les réalisations des élèves, en fonction de notre hypothèse.

La solution attendue est proche de la solution « élève » n°1 du tableau 1[8]. Certains élèves produisent cette solution. Elle témoigne d'une prise de conscience de la capacité du dispositif à traiter une formule écrite dans une syntaxe très proche de l'algèbre habituelle, en instanciant les trois paramètres selon les valeurs données en entrée. D'autres élèves produisent des solutions du type de celles numérotées 2 et 3 dans le tableau : ils essaient de traduire l'équation avec les éléments syntaxiques du langage, soit par une affectation (Solution « élève » n°2), soit par une condition (Solution « élève » n°3) puis « délèguent » au dispositif la résolution et l'expression des solutions. Les solutions du type de celle numérotée 4 dans le tableau sont aussi très souvent rencontrées : la variable I prend les valeurs successives du « second membre » quand on résout l'équation en papier/crayon. Même si la solution est syntaxiquement correcte, Briant relève que les élèves sont réticents à une formulation comme celle de la solution n°1, ce qu'elle analyse comme une résistance à adopter un traitement qui ne coderait pas les actions successives.

<table>
<tr>
<td>Solution « élève » n°1
 
</td>
<td>Solution « élève » n°3
 
</td>
</tr>
<tr>
<td>Solution « élève » n°2
 
</td>
<td>Solution « élève » n°4
 
</td>
</tr>
</table>

[8] Pour que cette solution soit complète, il faudrait aussi que l'élève prenne en compte les différents cas où n'existe pas une solution unique. Cette prise en compte est importante à la fois d'un point de vue informatique et d'un point de vue mathématique, elle sera analysée dans la troisième partie.

Tableau 1: quatre solutions pour une équation du 1er degré. D'après Briant (2013, p. 414)

Des difficultés elles-aussi relatives aux spécificités des langages informatiques, avaient été antérieurement observées et analysées dans le contexte de l'option informatique (Lagrange, 1991). Il s'agissait d'une séquence d'apprentissage sur le type « chaîne » en classe de Seconde, dont l'objectif général était la prise de conscience par les élèves du caractère calculable des objets liés aux variables de ce type. Notre hypothèse était que, à la différence du type « nombre » dans l'exemple précédent, les écritures impliquant des chaînes ne seraient pas associées par les élèves à des expressions manipulées en algèbre et que donc les difficultés observées renverraient plus directement à des spécificités des langages informatiques.

Outre l'affectation, la lecture et l'affichage, ce type offre une fonction `longueur` rendant le nombre de caractères d'une chaîne et une fonction `souschaine` rendant une chaîne d'une longueur donnée (troisième paramètre) constituée de caractères consécutifs d'une chaîne donnée (premier paramètre) à partir d'un rang donné (second paramètre). Nous analysons ici une tâche typique (question 2, tableau 2), donnée à des élèves ayant eu une quarantaine d'heures en informatique avec une initiation au type chaîne de caractères (sans l'opération de concaténation).

On donne le programme :
`Lire X`
`A←souschaine(X,1,1)`
`L←longueur(X)`
`B←souschaine(X,L,1)`
`afficher A;B`
1. Indique ce que l'ordinateur affiche pour l'entrée de « BONJOUR », de « B ».
Réponse attendue : BR, BB
	2. Ecris un programme pour que l'utilisateur ayant entré une chaîne, l'ordinateur affiche la chaine en passant la première lettre à la fin. (BONJOUR → ONJOURB)
	Réponse attendue :
	`Lire X`
	`L←longueur(X)`
	`A←souschaine(X,2,L-1)`
	`B←souschaine(X,1,1)`
	`afficher A;B`

Tableau 2 : Une tâche « classique » sur les chaînes

La première question demandait d'interpréter un petit programme : les variables `A` et `B` prennent respectivement comme valeur le premier caractère et le dernier caractère d'où les réponses attendues. Une réponse correcte à la question 2 consiste à affecter à une variable `L` la longueur de la chaine entrée `X` comme dans la question 1, puis à une variable `A` la sous-chaine de la chaine entrée `X` commençant

à 2 et de longueur L-1, à une variable B le premier caractère comme dans la question 1, et à afficher la chaîne A, suivie de la chaîne B.

Pour la première question, les résultats avaient été donnés correctement (sauf par un élève qui donnait B au lieu de BB pour l'entrée de la lettre B). Pour la seconde question, environ la moitié des élèves ne donnaient pas de réponse syntaxiquement correcte. Une réponse typique d'élève est la suivante :

```
A←souschaine(X,2,L+souschaine(1,1))
```

L'élève imbrique deux appels à la fonction sous-chaine. Les deux premiers arguments sont corrects, mais le troisième est syntaxiquement incorrect et d'interprétation peu aisée : l'élève « additionne » une variable représentant probablement la longueur (mais non affectée préalablement) et un appel à `souschaine` incorrect puisqu'avec deux arguments seulement.

L'épreuve a été suivie d'un entretien devant l'ordinateur avec des élèves ayant donné ce type de réponse. Dans l'exemple qui suit, l'observateur (**O**) a d'abord tenté une correction syntaxique, soulignant l'absence d'affectation à L, l'addition d'un nombre et d'une chaîne et l'absence d'une instruction de sortie du résultat. Il a constaté que les corrections apportées ne faisaient pas sens pour l'élève (**E**). Revenant à la réponse de l'élève, il a demandé ce que cette réponse signifiait pour elle :

> **O**. Tu avais mis un « + » ici, que signifie-t-il ?
>
> **E**. Ben, là… j'avais écrit `BONJOUR` donc `X,` à partir de la deuxième, on prend la longueur totale, plus la première lettre, on en prend une.
>
> **O**. C'est quoi souschaine(1,1)… ?
>
> **E**. La longueur de `X` et `1 et 1.`

Les réponses montrent que cette élève se servait des fonctions sur les chaînes pour coder des actions, sans exprimer ces actions comme la modification de valeurs des variables. Pour cette élève, `L` n'est pas la valeur d'une variable mais l'action de « prendre la longueur totale » ; `souschaine(1,1)` est l'action de « prendre la première lettre » et cette action s'exerce sur l'objet en jeu (qu'il n'est alors pour cette élève pas nécessaire de renommer). L'affectation répond à une simple nécessité syntaxique[9].

---

[9] Ceci est attesté par le fait suivant : dans un exercice traité ensuite, l'élève produit deux affectations sur la même variable A dont la valeur n'est pas réutilisée (Lagrange, 1991).

Dans un langage impératif, le traitement s'exprime comme une succession d'affectations de variables qui modifient l'état du dispositif. Dans les deux exemples, nous voyons qu'à des titres divers, les élèves observés ne saisissent pas ce fonctionnement. Les solutions 2 et 3 du premier exemple ignorent la nécessité d'exprimer le traitement pour que le dispositif puisse l'opérer. Dans le cas des chaînes nous voyons que les objets en jeu ne sont pas clairement identifiés comme des variables, que les fonctions sur les chaînes sont conçues comme des actions qui modifient ces objets et que l'affectation n'a pas de signification fonctionnelle. Ceci nous permet d'interpréter la solution 4 du tableau 1 : bien qu'objectivement l'évolution de la variable I soit correctement organisée, les réticences des élèves à adopter la solution 1 montrent que subjectivement ils restent attachés à un traitement congruent à la succession d'actions dans la résolution papier/crayon : « soustraire b au second membre », puis « le diviser par a ».

Le traitement des booléens qui intervient nécessairement dans l'expression des conditions des alternatives nous fournit un autre exemple du caractère récurrent des difficultés de débutants.

Booléens et « conditions »

Dans les pratiques actuelles courantes au lycée, le type booléen dont nous avons donné plus haut les caractéristiques, n'est pas explicitement rencontré. Les expressions dans la partie condition des alternatives ou structures itératives sont cependant de nature booléenne, c'est-à-dire que dans une écriture comme `SI (A=0) ALORS…`, `A` étant une variable de type nombre, l'expression `(A=0)` peut comme variable de type booléen prendre deux valeurs (`VRAI` ou `FAUX`) et entrer dans la formation d'expressions plus complexes par le biais de connecteurs booléens. Comme dans la partie précédente, nous commençons par un exemple tiré du contexte actuel de l'algorithmique au lycée en France, que nous mettons en relation avec les données d'une recherche menée antérieurement dans le cadre de l'option informatique.

L'extrait suivant est tiré d'un épisode observé dans le cadre d'un mémoire de master (Guy, 2013) mettant en place une ingénierie sur laquelle nous reviendrons en deuxième partie de cet article. Il s'agit d'une phase de mise en commun après la construction d'un traitement comportant une boucle dont la condition de sortie est qu'une variable `A` prenne une des deux valeurs 495 ou 0. Il s'agit d'une bonne classe de Terminale Scientifique Option Math qui fait de l'algorithmique depuis la seconde[10]. Le langage choisi impose une structure TANT QUE, c'est-à-dire avec une

[10] Cette option comporte un enseignement d'arithmétique (théorie élémentaire des

condition en début de boucle.

Voici un extrait d'une discussion collective sur l'écriture de la condition de continuation sous la forme `(A différent de 495) ou (A différent de 0)` par un des groupes (« les filles devant »). Ce groupe ne comprend pas pourquoi l'exécution ne termine pas. Interviennent l'enseignante (P), différents élèves (E_i) et le professeur habituel de la classe (O).

> **P.** …vous les filles devant, vous m'avez dit que le test qu'on fait là c'est A différent de 495 **ou** A différent de 0, au fond, tu m'as dit…
>
> **E_3.** A différent de 0 **et** A différent de 495.
>
> **P.** Alors, qu'est-ce qui est correct, qu'est-ce qui ne l'est pas ? … Oui ?
>
> **E_7.** Ben moi ça me parait bizarre qu'un nombre soit égal à deux nombres différents.
>
> **P.** Voilà, donc, quand tu dis le et, tu vois, un nombre qui est différent de deux nombres, c'est vrai, tu vas trouver des nombres qui sont différents de 495 et différents de 0. … Mais tu vas jamais satisfaire la condition être égal à 495 et à 0. … Finalement, ton algorithme, il s'arrêtera quand tu obtiendras un nombre égal à 495 et à 0, d'accord ? Et ça tu vas pas pouvoir le trouver.
>
> **E_3.** On l'a fait fonctionner et ça a marché.
>
> **P.** Tu l'as fait tourner et ça marche…
>
> **O.** Tu l'as fait… Ça devrait pas marcher.
>
> **P.** Oui, mais, AlgoBox il a des conditions d'évaluation des…
>
> …
>
> **P.** Bon, on va y réfléchir, c'est une question à laquelle il faut réfléchir ...

L'enseignante (P) essaie de provoquer un débat. Un élève (E_3) corrige sans plus d'explication. Une élève (E_7) fait une intervention peu explicite, mais on imagine qu'elle pense à ce qui ferait sortir de la boucle, c'est-à-dire la négation de la condition `(A différent de 495) ou (A différent de 0)` qu'elle

---

nombres). Comme on le voit dans l'extrait qui suit, l'environnement dans lequel travaillent les élèves est AlgoBox, mais cette donnée n'est pas nécessaire à la compréhension de notre analyse.

conçoit comme `(A=495) et (A=0)`. L'enseignante reprend cette idée, mais de façon assez confuse, puisqu'elle considère en même temps une condition de continuation correcte « *des nombres qui sont différents de 495 et différents de 0* » et une condition d'arrêt toujours fausse « *être égal à 495 et à 0* ». La discussion tourne court, quand l'élève E_3 qui a corrigé propose une validation pragmatique, reprise par le professeur. Le débat est relancé par (O) qui semble très étonné que la condition avec le « ET » fonctionne. L'enseignante semble prise au dépourvu et se réfère au logiciel plutôt qu'à la logique, et, après une phase confuse, clôt le débat.

Pourquoi l'enseignante ne relève-t-elle pas que la condition `(A différent de 495) ou (A différent de 0)` est toujours vraie ? Pourquoi ne reprend-elle pas la remarque de E_7, en l'exprimant en termes de condition d'arrêt, et en soulignant que la condition dans le TANT QUE est une condition de continuation, négation de la condition d'arrêt ? Ceci suggère que les difficultés liées aux conditions (valeurs logiques de type booléen) sont sous-estimées. La ressemblance entre ces conditions et celles que l'on exprime dans le langage « ordinaire » est trompeuse dès le problème impose l'emploi de connecteurs logiques.

Comme annoncé, nous revenons à cette hypothèse à partir d'une recherche menée dans le cadre de l'option informatique. Lagrange (1991) présente une ingénierie sur le type « booléen » expérimentée avec des élèves d'une classe scientifique dans leur deuxième année d'option, après qu'ils aient été initiés à ce type. Le tableau 3 donne l'énoncé du premier problème de cette ingénierie, une solution attendue et une solution-élève typique. L'énoncé est conçu pour confronter les élèves aux calculs sur les booléens via les connecteurs logiques de façon à les différencier des expressions du langage courant. Il introduit ainsi une rupture entre d'une part la première et la deuxième condition qui peuvent s'exprimer sous une forme proche du langage courant, et d'autre part la troisième qui implique d'articuler trois variables dans une formule non triviale, notamment parce qu'elle met en jeu une implication.

La solution attendue exprime les trois conditions et les affecte séparément à trois variables booléennes. Une alternative finale conditionnée par la conjonction des trois variables permet la sortie de la réponse. La solution « élève » typique privilégie la structure alternative, en emboîtant plusieurs instances. Ceci permet une expression congruente à l'énoncé pour les deux premières conditions et évite le recours à la négation. Les solutions recueillies ne vont pas plus loin, car les élèves échouent à exprimer la troisième condition de cette manière.

Enoncé

J'ai 5 amis : Marie, Marc, Jean, Janine et Jean. Quand je veux les inviter à dîner, il me faut tenir compte des contraintes suivantes:

Marie et Jean ne s'entendent pas.

Marc et Marie ne viendront pas l'un sans l'autre.

Si j'invite Janine, il faut que j'invite Luc ou Jean, mais on ne peut pas les inviter tous les trois ensemble.

Après avoir déclaré 5 variables booléennes, chacune prenant la valeur vraie si la personne correspondante est invitée, écrire un programme qui permet de savoir si une invitation est possible.

Solution attendue[11]

```
C1:=Not (Marie and Jean)
C2:=(Marc=Marie)
C3:=(Luc≠Jean)or not Janine
If C1 and C2 and C3 then
   display 'Possible'
 else
   display 'Impossible'
```

Une solution « élève » typique

```
If (Marie and Jean) then
 display 'Impossible'
else
  if (Marc<>Marie) then
     display 'Impossible'
  else…
```

Tableau 3 : un problème sur les booléens

Notre interprétation est que les élèves conçoivent les expressions booléennes comme des « conditions » telles qu'elles s'expriment dans le langage usuel, plutôt que comme un calcul sur des valeurs logiques. À la différence des élèves observés sur le type chaîne, ceux-ci sont d'un bon niveau scientifique et maîtrisent les éléments de base du langage. Nous observons cependant que confrontés à un type non numérique et malgré l'enseignement reçu, ils se montrent très peu capables d'exploiter les possibilités d'expression qu'il offre. Lagrange (1991) montre dans la suite de l'ingénierie une difficile progression. Les élèves restent attachés aux

[11] La troisième condition peut s'exprimer de façon plus proche de l'énoncé, par exemple en séparant les deux sous-conditions :

```
C3a:=Luc or Jean or not Janine
C3b:=not (Luc and Jean and Janine)
C3:= C3a and C3b
```

alternatives et très réticents à l'emploi de variables booléennes. Une progression souvent constatée est l'emploi d'un type connu pour « contourner » le problème. Par exemple les élèves déclarent une chaîne à laquelle ils affectent « OUI » ou « NON », ou une variable numérique à laquelle ils affectent 0 ou 1 ; dans ce dernier cas, les conditions peuvent être calculées – ainsi (`AxB=1`) exprime la conjonction – ce qui constitue une étape vers la compréhension des booléens.

Quels savoirs sous-jacents ?

Nous avons commencé cette première partie en soulignant que l'expérience des années 1980 contredisait l'idée d'un accès aisé des élèves à une expertise suffisante pour que les activités en programmation et algorithmique puissent contribuer aux apprentissages visés par les différents curricula en informatique et en mathématiques. Nous avons montré à l'aide d'exemples que les difficultés demeurent dans le contexte actuel de l'algorithmique au lycée. Faisant le lien avec des exemples similaires étudiés dans le contexte de l'option informatique, nous avons analysé quelques aspects de cette question, que l'on peut résumer par une difficulté à comprendre un programme ou un algorithme comme organisant un traitement pour un dispositif, à percevoir ce traitement comme l'évolution des valeurs d'un ensemble de variables, et à concevoir ces variables comme des objets calculables. Les exemples présentés montrent que ces difficultés ne sont ni anecdotiques ni passagères. C'est pour nous l'indice qu'il existe des savoirs conceptuels sous-jacents.

Comment ces savoirs se situent-ils dans les enseignements de mathématiques et d'informatique ? Nous esquissons quelques repères à ce propos en conclusion de cette première partie.

Concernant les mathématiques, il est possible de faire le lien d'une part avec l'algèbre et d'autre part avec la logique. Concevoir les variables informatiques comme représentant des objets familiers mais avec un fonctionnement différent, présente en effet une certaine similarité avec la compréhension du symbolisme algébrique comme outil de modélisation du réel, une dimension importante de l'enseignement de l'algèbre soulignée par exemple par Chevallard (1989). Cependant Duval et Pluvinage (2016) montrent que les apprentissages algébriques restent peu assurés au moment où commencent les apprentissages en algorithmique et programmation. Nous observons quant à nous que ces apprentissages ne se transfèrent pas à des types non numériques, et que même lorsqu'il s'agit de nombres, les variables des langages informatiques prennent une signification différente pour les élèves.

De manière assez semblable, les problèmes sur les booléens paraissent relever de l'articulation entre « logique de sens commun » et « logique formelle » identifiée par Durand-Guerrier (1996) et largement étudiée depuis. Cependant cette

articulation s'opère en mathématiques dans des activités de raisonnement impliquant des énoncés souvent quantifiés de façon implicite. La conception d'un traitement à implémenter sur un dispositif informatique fait apparaître d'autres obstacles sur la représentation et le traitement d'expressions booléennes.

Les travaux menés dans le contexte de l'option des années 1980 indiquent que les savoirs relatifs aux spécificités des langages informatiques étaient considérés comme des prérequis par les enseignants et les ressources pour la classe, ce qui semble être une des causes d'un fort taux d'abandon en fin de première année (Lagrange 1991). Plus récemment, au lycée, nos observations suggèrent que les difficultés conceptuelles ne sont pas davantage prises en compte. En particulier la complexité propre de la notion de variable dans le contexte informatique et la rupture qui s'opère ainsi avec la notion de variable mathématique sont sous-estimées. Ce sera l'objet plus spécifiquement de la troisième partie.

2. Quelles situations pour les premiers apprentissages ?

Dans la première partie, nous avons souligné la similarité des difficultés rencontrées par les élèves dans les deux contextes de l'option informatique (années 80) et celui, actuel, de l'algorithmique au lycée. Ce constat montre la nécessité, après avoir commencé à cerner des savoirs relatifs aux spécificités des langages informatiques de questionner les situations d'apprentissages. Nous faisons pour cela un détour par une synthèse de Crahay (1987) sur l'impact réel des activités de programmation en LOGO. Ces activités ont été présentées notamment par Papert (1981) comme devant favoriser l'accès à la « pensée procédurale », c'est-à-dire à la capacité de penser une série d'actions comme une procédure paramétrable et réutilisable dans un traitement plus complexe. Crahay fait une revue des évaluations et conclut qu'elles ont toutes montré des résultats décevants. Il les attribue à la conception et à la conduite des situations didactiques par les enseignants : donnant des tâches ouvertes et laissant aux élèves une grande liberté dans une approche « constructiviste naïve », ils étaient conduits ensuite à intervenir individuellement auprès des élèves en finissant par écrire la solution à leur place[12].

Nous souhaitons montrer dans cette partie comment la théorie des situations didactiques (Brousseau, 1988) a pu, dans certaines recherches plus récentes, poser les conditions d'expérimentations mieux contrôlées rompant avec le

[12] Généralement les productions obtenues se limitaient à une seule procédure et étaient en cela proches d'écritures en langage impératif, malgré le caractère « fonctionnel » de LOGO.

constructivisme naïf dénoncé par Crahay.

D'une étude épistémologique du développement des machines mathématiques à une situation d'apprentissage de l'itération

La première recherche considérée ici est la thèse de Nguyen (2005) menée dans le contexte du curriculum de mathématiques en classe de Première[13] dans les années 2000. Nguyen s'intéresse à l'itération et montre que c'est une structure qui émerge dans l'évolution des machines mathématiques de la machine à calculer simple à la machine de Babbage. Cette dernière était conçue pour tabuler des fonctions de façon automatique, grâce à deux innovations : 1°) un emplacement mémoire correspondant à une donnée au cours d'un calcul répétitif peut prendre successivement des valeurs différentes ; 2°) le contrôle de l'exécution est fait non par l'opérateur, mais par la machine en fonction de l'état de la mémoire, ce qui rend possible l'itération. La situation d'apprentissage de l'itération proposée par Nguyen est basée sur cette étude.

Elle conduit tout d'abord à considérer un dispositif (au sens où nous avons défini ce terme en première partie) proche d'une machine concrète comportant un ensemble de mémoires A, B, C pouvant stocker un nombre, résultat d'un calcul dont les opérations sont activées par des touches[14]. L'étude épistémologique conduit à confronter les élèves à la tabulation de fonctions dans des situations où il devient avantageux d'abord d'utiliser des mémoires, puis d'organiser l'activation des calculs et des mémoires dans une structure se répétant et finalement d'imaginer une instruction de répétition. Ce choix d'une situation « a-didactique » est différent d'autres choix où les possibilités du dispositif sont présentées aux apprenants par « monstration ».

Énoncé : Soit f une fonction. Calculer les images par cette fonction de nombres espacés d'un pas donné dans un intervalle donné. $f(x) = x^2+1$, intervalle [–3 ; 2], pas 0,2 puis 0,03. (Calculatrice avec mémoires STO)

[13] Deuxième année de Lycée (grade 11, élèves de 16 ans).

[14] Ce choix est différent de celui adopté dans les enseignements où nous avons fait les observations de la première partie. Ce choix reste compatible avec notre analyse des spécificités des langages informatiques : les suites de touches constituent un langage avec sa syntaxe propre, et le dispositif évolue par affectation des variables-mémoires (touche STO). Nous n'entrons donc pas dans l'analyse que fait l'auteur des différentes « stratégies » relativement au dispositif considéré.

T1 Écrire la suite des touches nécessaire pour la faire exécuter par un robot	-3 Sto A AxA+1= -2.8 Sto A AxA+1= -2.6 Sto A AxA+1= ... 2 Sto A AxA+1=
T2 Écrire un groupe de touches à faire répéter par le robot	**Groupe de touches à répéter :** · A + 0,03 → STO B ·. B x B + 1 = ·. B STO A
T3 Ecrire un message le plus court possible pour que le robot fasse tout le calcul de lui-même	-3 STO A Répéter 166 fois A x A + 1 = A + 0,03 STO A

Tableau3. Une situation d'apprentissage de l'itération : l'énoncé, les trois tâches (à gauche), et des réponses d'élèves (à droite). D'après Nguyen et Bessot (2010).

La première tâche (T1, Tableau 3) implique l'écriture fastidieuse d'une série de touches. Pour un pas de 0,2, il y a 26 séries de touches, chaque série commençant par l'entrée d'une valeur de la variable qui est stockée dans la mémoire A. Pour un pas de 0,03, il faudrait 166 séries de touches. Les élèves voient donc l'intérêt d'un bloc d'instructions qui pourrait être répété, ce qui est l'objet de la tâche T2. Mais cela implique une rupture, puisque dans ce bloc chaque valeur de la variable ne peut plus être entrée individuellement. Dans la tâche T3, les élèves doivent imaginer une structure dans laquelle insérer ce bloc d'instructions. Cette structure comprend une instruction spéciale de répétition quantifiée et une initialisation de la mémoire utilisée.

Cette ingénierie met en valeur le fonctionnement des variables dans l'itération. Selon Samurçay (1985) les relations entre ces deux éléments de l'itération sont particulièrement délicates, et même après une première initiation avec un enseignant analysant un programme itératif simple, cette complexité entraîne des difficultés persistantes chez les débutants. Un des points forts, lié à la construction d'une structure en réponse à un problème sur des objets mathématiques bien identifiés avec un contrôle direct de l'implémentation sur le dispositif invoqué, est que les élèves distinguent bien les opérations d'initialisation de celles constituant le

corps de boucle (voir Nguyen, 2005, notamment p. 225).

La planification dans la construction d'un traitement par des élèves

À la différence de l'exemple précédent qui partait d'une étude épistémologique, nous nous intéressons ici à une problématique issue de la psychologie de la programmation. Comment des débutants peuvent-ils construire un algorithme ou un programme alors qu'ils ne maîtrisent pas les contraintes d'expression du dispositif auquel ce traitement est destiné ? Rogalski et Samurçay (1990) ont introduit cette problématique en distinguant les positions d'expert et de novice dans l'activité de « planification », c'est-à-dire de conception des étapes pour un traitement. Elles introduisent deux autres notions : les « schémas » qui sont les structures utilisées dans le traitement des informations pour atteindre des objectifs à petite échelle, et les « plans » qui sont des ensembles organisés de schémas. Les experts, c'est-à-dire les personnes qui maîtrisent les contraintes d'expression du dispositif, combinent généralement une approche descendante (top-down) où le plan est d'abord pensé, puis organisé en schémas élémentaires et une approche ascendante (bottom-up) qui part de schémas connus pour les organiser en un plan. De façon générale si l'expert voit bien où il va, il construit un plan et ensuite s'intéresse à des sous-objectifs et aux structures correspondantes. Quand il voit moins bien comment commencer, il peut partir de schémas qu'il connait. Le problème des débutants est qu'ils ne disposent pas d'un répertoire de schémas et que les plans auxquels ils pensent sont directement transposés d'un traitement manuel ce qui rend difficile la spécification des sous-objectifs pour un dispositif informatique. Ils ne peuvent par conséquent adopter ni une approche descendante ni une approche ascendante.

Nous avons abordé cette problématique dans le contexte actuel de l'algorithmique au lycée ; cette étude s'intéresse donc aussi aux interactions mathématiques-algorithmique. Un travail autour de la « recette » de Kaprekar, présentée dans le tableau 4, a été retenu de façon à mettre en jeu à côté des savoirs mathématiques, des savoirs relatifs à la planification. Les savoirs mathématiques concernent la représentation des nombres, et plus généralement l'arithmétique. Ceci est discuté plus complètement par Lagrange et Guy (2015). La situation s'adresse à des élèves de Terminale Scientifique (élèves de 17 ans, grade 12).

1. Choisir un nombre entier de trois chiffres.
2. Former le nombre obtenu en arrangeant les chiffres du nombre choisi en 1. dans l'ordre croissant.
3. Former le nombre obtenu en arrangeant les chiffres du nombre choisi en 1. dans l'ordre décroissant.
4. Calculer la différence des nombres obtenus en 2. et 3.
5. Recommencer (à partir de l'instruction 2.) avec le résultat obtenu en 4, jusqu'à obtenir un nombre déjà obtenu.

Tableau 4 : La « recette » de Kaprekar

Nous avons mis en place une situation didactique reposant sur le fait que dans cette recette, le nombre courant est considéré sous deux représentations. Pour le tri (rangement dans l'ordre croissant ou décroissant), c'est un triplet de chiffres et pour la soustraction c'est un nombre au sens habituel du terme. À la main, la différence de représentation n'apparait pas : un opérateur humain (du niveau lycée) n'a pas nécessairement conscience d'isoler des chiffres puis de les ordonner. Le traitement par un dispositif « arithmétique », c'est-à-dire un dispositif dont le langage de commande comporte les 4 opérations dans les entiers et des instructions de comparaison, impose de considérer le nombre courant sous ses deux représentations et donc de construire des traitements partiels de conversion du nombre ordinaire en triplet de chiffres et inversement.

Après que les élèves aient constaté sur quelques exemples que le traitement termine soit sur 0 soit sur 495, il leur est demandé de concevoir un traitement dans leur environnement de programmation habituel de façon à examiner la généralité de ce constat[15]. La conception de la situation repose sur les hypothèses suivantes : 1°) spontanément, les élèves vont s'engager vers un plan calqué directement sur la recette, car ils ne disposent pas des schémas de décomposition et de recomposition d'un nombre ; 2°) par confrontation au dispositif « arithmétique » il peut se produire une prise de conscience de la nécessité d'aménager le plan du traitement, et plus généralement de ce qu'un plan convenant à un traitement manuel doit être adapté en fonction des capacités du dispositif destiné à l'exécuter.

---

[15] Nous ne considérons pas ici la variable de situation « langage utilisé », pourvu que ce langage puisse commander un dispositif arithmétique au sens que nous venons de préciser. L'expérimentation est celle qui a déjà été rapportée en partie 1, avec des élèves de Terminale Scientifique (grade 12) utilisant Algobox.

Précisons le milieu. Il est constitué d'objets mathématiques avec leurs propriétés arithmétiques qui exercent certaines rétroactions. Ce sont aussi des objets à codifier pour un traitement destiné à être exécuté par un dispositif « arithmétique ». Les rétroactions du langage résultent des possibilités d'expression et des contraintes perçues de façon interne au langage, (cette écriture n'est pas « conforme ») ou en référence au dispositif (l'ordinateur ne va pas comprendre) et souvent les deux à la fois[16].

Un autre choix essentiel est de mettre l'accent sur la représentation des données (choix des variables) pour engager une dynamique de planification productive : pensant à un plan calqué sur l'exécution manuelle, les élèves vont d'abord se contenter d'une variable pour le nombre courant et éventuellement de variables pour les nombres obtenus par rangement croissant et décroissant des chiffres ; il est possible que certains élèves perçoivent que, pour un traitement par un dispositif « non humain », il est nécessaire de prévoir les variables sur lesquelles opérer le rangement des chiffres du nombre courant ; si ce n'est pas le cas, l'enseignant peut enclencher cette réflexion en demandant aux élèves si leur choix permettrait de traiter par exemple un nombre à 10 chiffres. La prise de conscience de la nécessité de variables pour les chiffres devra ensuite engendrer le besoin de traitements partiels correspondant à des schémas de décomposition et de recomposition d'un nombre, s'ajoutant au tri (rangement dans l'ordre croissant) des chiffres. Ainsi les élèves rencontreront la nécessité de penser l'écriture d'un traitement en traitements partiels[17] conçus de façon autonome (par exemple par des concepteurs différents) mais qui devront être coordonnées : c'est ce que nous désignons par « modularité ».

Les 4 phases résultent de ces choix. Dans une première phase, il est demandé aux élèves quelles variables sont à utiliser, et on s'attend, comme nous venons de le dire, à ce que se produise une prise de conscience de la nécessité d'introduire des variables pour les chiffres du nombre courant. Ensuite on peut penser que cette prise de conscience va être partagée dans la classe et qu'un plan incluant des schémas de conversion va être identifié. Dans une seconde phase, les élèves ont à concevoir les trois traitements partiels de la figure 1. L'écriture de chacun des trois traitements partiels est confiée à des groupes différents. La troisième phase consiste à rédiger le traitement complet en organisant les trois traitements partiels dans une

---

[16] Bien que la situation ait été expérimentée avec un langage « évolué", cette conception du milieu est compatible avec celle mise en place par Nguyen. Lagrange & Guy (2015) détaillent ce milieu et les choix relatifs aux variables didactiques.

[17] Désignés par le professeur par le terme « étape » dans l'ingénierie (Cf. figure 1).

itération. Il ne s'agit d'une simple concaténation puisque les trois traitements partiels sont rédigés avec des variables spécifiques qui doivent être harmonisées pour un traitement itératif : la modularité n'est pas un simple découpage. En quatrième phase, les élèves ont à construire l'itération avec une condition adéquate (épisode discuté en partie 1).

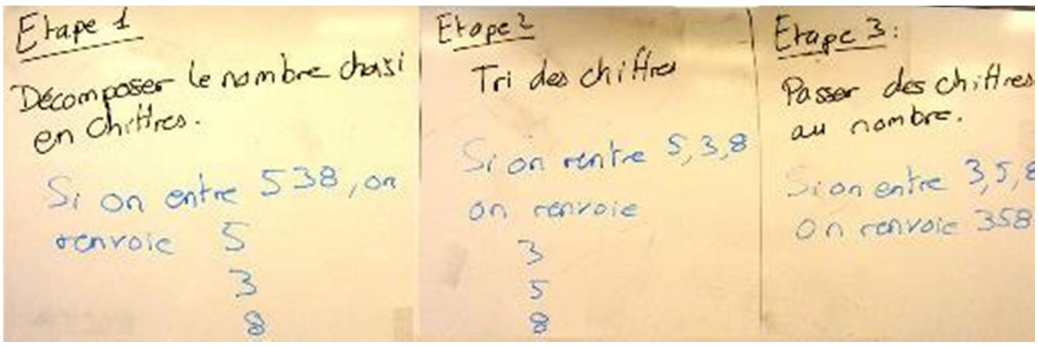


Figure 1 : Trois traitements partiels tels que spécifiés à la fin de la première phase (d'après Guy 2013) [18]

Voici quelques éléments d'analyse *a posteriori* dans chacune des phases. Dans la première phase, nous assistons bien à une prise de conscience de la nécessité de deux représentations sous l'influence du dispositif. Pour l'illustrer, voici deux prises de parole successives par deux élèves, l'un fait référence au traitement manuel « on va prendre » et présente un choix naïf des variables, dans un schéma itératif. Le second élève fait lui référence au dispositif (« il ») et à ses capacités et identifie la nécessité de variables pour les chiffres.

> **E1. On va prendre** A en nombre initial, **on va prendre** B arrangé en ordre décroissant, C en ordre croissant, **on va faire** la différence de, euh, C moins B. O**n va le remettre** dans A, et **on va refaire** avec B et C.
>
> **E2.** Ben si A c'est une variable où c'est un nombre, pour qu**'il** puisse le ranger dans l'ordre croissant et décroissant, faut d'abord séparer les chiffres des dizaines, des unités et des centaines. Donc il faut trois variables…

Comme prévu aussi, les élèves, dans une phase collective, identifient les étapes et, par groupe sur ordinateur, conçoivent les traitements partiels. Les solutions sont

[18] L'étape 3, pour être complète, doit se terminer par la donnée de la différence des nombres obtenus en arrangeant les chiffres dans l'ordre décroissant (resp. croissant).

variées, et là aussi on constate à nouveau que le dispositif joue un grand rôle dans la réflexion des élèves. Dans la dernière phase, on rencontre la difficulté attendue à assembler les traitements partiels avec des variables cohérentes entre les traitements partiels, mais aussi en entrée et en sortie du corps de boucle. Certains élèves ont construit des solutions pour plusieurs traitements partiels à la seconde phase et rencontrent moins de difficulté à les assembler, ce qui indique que la modularité ne va pas soi.

Un des points forts repérés par Lagrange et Guy (2015) est que les élèves ont progressé seuls sur les points critiques : prise de conscience de la nécessité d'étapes de conversion, conception et assemblage des traitements partiels, et plus généralement prise de conscience de la nécessité de penser un plan, non en fonction d'un traitement manuel, mais pour l'expression pour un dispositif spécifique. Une phase ultérieure visant à l'étude de la terminaison a été mise en place dans le cadre d'un travail de thèse. L'analyse de cette phase par Laval (2015) montre que ce travail en a-didacticité donne aux élèves une conception des objets en jeu adéquate pour cette phase centrée sur la preuve.

L'apport de la théorie des situations didactiques

La conception et l'analyse dans les deux exemples qui viennent d'être rapportés brièvement sont inspirées de la « théorie des situations didactiques » de Brousseau. L'apport théorique est une meilleure identification des savoirs en jeu, grâce à la conception du milieu pour une interaction a-didactique. Pour l'itération, ce travail de conception permet de répondre à des questions telles que : Quels problèmes font ressentir la nécessité d'une structure itérative ? Quelles étapes sont à franchir ? Comment le langage évolue-t-il avec ces étapes ? Le milieu pour la planification comprend quant à lui un objet sous deux représentations pour les nécessités du traitement par un dispositif spécifique. Les savoirs sous-jacents sont la capacité à concevoir ces deux représentations, à déterminer des variables adéquates pour cela et à assurer une gestion cohérente de ces variables dans l'assemblage de traitements partiels.

Les acquis de la recherche en psychologie de la programmation

L'enseignement de l'algorithmique et de la programmation est revenu en France dans le curriculum du lycée, dont on rappelle la *visée générale* : « à l'occasion de l'écriture d'algorithmes et de petits programmes, il convient de donner aux élèves de bonnes habitudes de rigueur », et les *objectifs* : « écrire une formule permettant un calcul ; écrire un programme calculant et donnant la valeur d'une fonction… », sans expliciter ce qu'est un algorithme, un programme ni leur différence.

Ce retour se fait sans référence à l'expérience acquise antérieurement, en

particulier à propos de l'option informatique des lycées (en gros, une décennie entre 1981 et 1992), et sans exploitation du questionnement déjà ouvert alors sur les apports potentiels de l'enseignement de l'informatique (essentiellement au niveau du lycée). Les collègues moins anciens que nous ignorent sans doute les recherches qui se sont parallèlement développées en didactique de l'informatique et en psychologie de la programmation (en particulier à propos des débuts des apprentissages). Ce sont les acquis de ces recherches sur lesquels nous voulons attirer l'attention, en faisant un point rapide sur les mouvements qui ont eu lieu dans ce domaine, en soulignant un certain nombre d'acquis stables sur l'activité de programmation et sur les concepts et obstacles épistémologiques et en relevant l'existence de deux questions récurrentes : celle des précurseurs potentiels de la programmation et celle des impacts mutuels entre informatique et mathématique. Une attention sera portée à la notion de « variable » dans le contexte de la programmation informatique (notion que nous avons condensée sous le terme de « variable informatique »).

L'évolution des objets de recherches en psychologie de la programmation et en didactique de l'informatique

L'évolution de ce domaine a été présentée en détail lors du Congrès de 2013 de didactique de l'informatique (Rogalski, 2015). Nous en donnons ici un bref résumé. Dans les années 1970, des théoriciens de l'informatique ont recherché des concepts puissants de programmation, d'abord du point de vue du développement du domaine scientifique (et ensuite technique). La référence centrale est l'ouvrage initiateur de Knuth (1968). Ensuite est venue une « seconde génération », celle d'études expérimentales d'informaticiens largement tournées vers les propriétés souhaitables des langages informatiques. La décennie des années 1980 a vu le déploiement des études en psychologie de la programmation (portant particulièrement – mais pas seulement – sur des débutants) et l'émergence d'une didactique de l'informatique (sous ce nom dans la communauté francophone européenne) pour une « alphabétisation informatique » scolaire dans ce champ (qui allait au-delà de la programmation). On observe ensuite, dans une « troisième génération », un « rétrécissement » des recherches sur l'informatique scolaire avec en contrepoint une ouverture des recherches de psychologie de la programmation vers les initiations du niveau universitaire ou professionnel. En conséquence l'intérêt des recherches s'est porté sur d'autres langages que les langages « *impératifs* » qui dominaient dans les recherches sur le secondaire. Il faut cependant relever une continuité de recherches en psychologie de la programmation via le groupe « *The Psychology of Programming Interest Group* » (PPIG) constitué à Cambridge (UK) en 1987, et qui depuis poursuit son activité dans le but de réunir des acteurs de diverses communautés pour explorer des intérêts partagés dans les aspects psychologiques de la programmation et dans les

aspects « computationnels » de la psychologie.

La quatrième génération (après les années 2000) s'est orientée vers une didactique pour l'informatique universitaire et professionnelle – au sens où les questions traitées sont d'une part celles des premiers enseignements universitaires, *i.e.* les initiations à la programmation, et d'autre part, celles des acquisitions de concepts avancés, en relation en particulier avec le développement de la programmation « parallèle ». Deux ordres de raisons semblent à l'origine de ces nouvelles recherches, essentiellement conduites par les informaticiens enseignant à des niveaux post-secondaire : le manque menaçant d'informaticiens dans les pays occidentaux et le besoin d'avoir des professionnels du niveau exigé pour la sécurité et la fiabilité des programmes à concevoir dans nombre de domaines sensibles (dont ceux touchant les bases de données), alors que les étudiants se dirigent moins vers les études scientifiques en général et l'informatique en particulier. Une triple contrainte semble apparaître : il faut attirer des étudiants en initiation à l'informatique, les garder pour la poursuite d'études dans ce champ – ce qui suppose qu'ils réussissent « suffisamment », et les préparer à être des informaticiens répondant aux attentes de la qualité de la programmation – ce qui suppose qu'ils atteignent un niveau « suffisant » dans leurs acquisitions. Le retour de l'informatique dans la scolarité du secondaire apparaît un moyen de préparer un « vivier » d'étudiants qui permette de répondre à ces contraintes, comme l'exprime l'introduction d'un papier récent :

> While needs of high school students in the US are being prioritized through courses such as Exploring Computer Science (ECS) (…), there is a growing belief that experiences with computing must start at an earlier age. (Grover, Pea & Cooper, 2016).

Le questionnement proprement didactique (élaboration et conduite de situations appropriées aux acquisitions visées, pour dire vite) est toutefois resté limité par rapport à celui orienté vers ces acquisitions, y compris en France. Un article (da Rosa, 2015), d'orientation piagétienne et se référant à la Théorie des Situations Didactiques, regrette d'ailleurs, lors du congrès PPIG2015, que cette dimension « didactique » reste marginale dans les recherches au niveau international. Pourtant, comme l'illustrent les exemples développés dans la première partie de cet article, non seulement on observe une persistance des questions pertinentes soulevée par la seconde génération de recherches, mais on relève aussi la validité de résultats de la recherche des années 1980, dont nous allons rappeler les grandes lignes.

Des acteurs dans la communauté française

Il est utile de citer d'abord quelques acteurs qui ont marqué ce domaine dans la seconde génération en France (travaillant dans le contexte de l'introduction de

l'informatique au lycée) et dont les travaux ont été repris comme références (lorsqu'ils étaient publiés en anglais), car ils offrent des références toujours pertinentes (et qu'ils ont concerné le contexte scolaire institutionnel en France). Pour la didactique de l'informatique, on peut retenir André Rouchier et Renan Samurçay. Le texte de Rouchier : « *Objets de savoir de nature informatique dans l'enseignement secondaire* » situe l'informatique dans un champ d'interrogation didactique, puis analyse les objets informatiques en jeu dans l'écriture de programmes élémentaires, objets d'un domaine de savoir qui peut constituer un « habitat » nouveau pour des savoirs mathématiques (Rouchier, 1990). La disparition d'un enseignement de la programmation au niveau scolaire n'a pas permis à ce travail conceptuel de didactique d'avoir l'impact qu'il aurait mérité[19].

En revanche, des acquis sur la psychologie de l'activité de programmation portant sur les concepts informatiques et les activités cognitives en jeu dans la programmation par des débutants sont restés « vivants ». Leur caractère conceptuel a certainement joué. Il en est ainsi de la recherche de Renan Samurçay sur le concept de variable informatique sur lequel on revient plus loin qui est une référence toujours citée (Samurçay, 1989). Dans la même ligne, un ouvrage de synthèse « *Psychology of programming* » (Hoc, Green, Samurçay & Gilmore, 1990) d'un groupe européen de recherche sur la programmation a été mis en ligne pour les étudiants en informatique de Cambridge comme référence sur la dimension humaine de la programmation.

Des acquis sur l'activité de programmation

Après les acteurs, faisons un point sur les acquis des recherches de la décennie 1980. Ces acquis sont de nature épistémologique et cognitive. Ils sont relatifs aux spécificités de la discipline[20] : a) l'informatique est à la fois science et pratique (prenant en compte la dimension technologique, on parle de génie informatique, comme on parle par exemple de génie civil pour l'ingénierie du bâtiment) ; à ce titre, elle pose un double problème de transposition dans l'enseignement ; b) l'activité cognitive de programmation présente des spécificités ; c) l'informatique comporte des concepts-clés qui n'ont pas de précurseurs[21] ou ont des précurseurs

19 En fait, l'approche didactique, en tant que telle, a été essentiellement engagée en France et en Allemagne (Laborde, 1988).

20 Nous nous centrons toujours ici sur l'informatique « objet » (comme « discipline ») et non sur l'informatique comme technique ou technologie, « outil » pour d'autres disciplines ou d'autres domaines d'action.

21 La notion de récursivité non terminale (Rouchier, 1987) introduite à travers une

qui peuvent jouer un double rôle : à la fois producteur (facilitateur) et réducteur (éloignant du concept à acquérir) ; enfin, d) des connaissances opérationnelles particulières sont en jeu pour la programmation, en particulier des méthodes de programmation.

Ces spécificités ont des implications didactiques, dans la conception de situations didactiques et dans la prise en compte des difficultés cognitives des élèves face aux tâches proposées.

Tout d'abord, le double problème de transposition de la pratique de la programmation et du savoir informatique comme savoir scientifique se manifeste dans la conception de situations didactiques. On peut le caractériser par la question suivante : comment articuler l'élaboration de situations didactiques centrées sur l'acquisition de concepts déterminés (variables, boucles, conditionnelles) avec une activité de programmation partant de problèmes consistants, plus ouverts ? Dans le premier cas, les tâches des élèves sont des problèmes bien circonscrits – comme dans les exemples des première et deuxième parties de cet article ; dans le second cas, des situations de « projet » transposent la programmation professionnelle (et les concepts apparaissent « à la demande » du problème.

Concernant ces situations de « projet », le constat général issu des études récentes comme de celles conduites autrefois – en particulier avec LOGO – est le fait que l'action didactique de l'enseignant est cruciale pour que l'engagement des élèves ou étudiants se traduise par l'acquisition de concepts de programmation : le projet doit être « porteur » et sa réalisation étayée par l'enseignant pour en faire apparaître l'utilisation des concepts visés[22].

La nature de cette action enseignante reste à notre point de vue une question didactique ouverte. Dans les recherches des années de l'option informatique, la centration était clairement sur la conception de situations didactiques plutôt que sur l'intervention didactique des enseignants conduits à gérer ces situations. Il en était alors de même en didactique des mathématiques. Les recherches sur les pratiques des enseignants acteurs individuels et sur leurs actions didactiques en classe se sont développées seulement à partir de la fin des années 90. Nous faisons l'hypothèse que les concepts et méthodes qui y sont développées seraient productives pour étudier l'action enseignante dans le champ de l'algorithmique en relation avec la

---

procédure ou une fonction est cognitivement éloignée de la récurrence en mathématique (et celle-ci présente un obstacle épistémologiques aux élèves même scientifiques).

[22] Il sera intéressant de mettre en regard l'expérience LOGO des années 1980 avec les usages de SCRATCH préconisés par le curriculum du collège.

programmation informatique.

En second lieu, l'analyse des spécificités de l'activité cognitive lors de la programmation informatique a conduit à souligner plusieurs points :

- en programmation il s'agit de passer de « faire » à « faire faire » (Samurçay & Rouchier, 1985) : l'exécution est déléguée à une « machine », avec perte de contrôle de celui qui a conçu le programme ; cela implique de se constituer une représentation opérationnelle de la machine ou plus précisément du « dispositif informatique »[23] (Rogalski, 1988 ; Rogalski & al., 1989) ou de la *notional machine* (Du Boulay, 1986).
- il faut effectuer un changement de niveau entre réalisation d'une occurrence particulière d'une procédure et élaboration d'une procédure générique[24] : cela suppose de passer d'une connaissance « en acte » à une analyse des objets et des actions en jeu ; il s'agit aussi de se questionner sur les entrées qui peuvent être pertinentes et sur la validité de la procédure selon le domaine sur lequel elle va « tourner » ;
- l'analyse des objets et des actions possibles dans le dispositif informatique appelle des acquisitions sur le « système de représentation et de traitement » des données informatiques (SRTI), dont Lagrange (1991) a montré combien c'était un processus difficile, en particulier pour les booléens, en jeu notamment dans les alternatives, l'itération et la récursivité.
- le fait que les élèves « contrôlent » un algorithme par son exécution constitue par ailleurs un obstacle à l'accès à la dimension fonctionnelle de la récursivité[25] ; il en est de même quand l'approche de la récursivité commence par une situation où l'appel récursif est en fin de traitement (récursivité « *terminale* »), ce qui fait que les élèves peuvent interpréter le traitement comme une itération (Pirolli, 1986 ; Kurland & Pea, 1986 ; Rinderknecht, 2014) ;
- il y a besoin d'un contrôle logique, et donc syntactique des structures

---

[23] La stratégie (implicite) d'enseignement (Ngyuen, 2006) est ici l'écriture d'un message à une machine ordinateur de Von Neumann pour obtenir l'exécution d'un algorithme.

[24] Ce passage « spécifique » / « générique » se pose évidemment pour la conception d'un algorithme, hors de tout contexte de programmation.

[25] Les environnent de programmation utilisés actuellement en lycée en France ne proposent pas une écriture « naturelle » de fonctions, et n'offrent guère d'opportunité pour introduire la récursivité.

conditionnelles – et non plus du contrôle sémantique, qui est d'ailleurs souvent présupposé et non explicité dans les exécutions « à la main ». Ainsi, dans une alternative, le dispositif informatique est censé « comprendre » que : 1°) lorsque le traitement d'une condition A (si A alors faire) est terminée, la suite du programme concerne la condition non A (sinon...), alors qu'il n'y a pas nécessairement explicitation du « sinon » ; et 2°) lorsque les deux cas d'une alternative (A ou non A) ont été traités, on passe à la suite de la procédure (en sortant de la conditionnelle) (Rogalski & al., 1989) ;

- dans la conception d'un programme interactif destiné à un utilisateur (qui peut-être le concepteur lui-même, mais plus tard), la représentation du triplet < « je conçois », « un dispositif informatique exécute », « un humain utilise » > appelle une articulation de la procédure « noyau » du programme (l'algorithme exprimé dans le langage de programmation utilisé) avec des opérations d'entrées de données par une saisie d'écriture par l'utilisateur (une « lecture » pour le dispositif) et de sortie à destination de l'utilisateur sous forme d'affichage (une « écriture » par le dispositif) que va lire l'utilisateur, opérations dont la formulation ne va pas de soi lors de l'alphabétisation[26].

En troisième lieu, des difficultés, voire des obstacles, se présentent régulièrement dans l'acquisition des deux grands concepts clés lors de l'enseignement des bases de la programmation (ce qu'on a appelé l'alphabétisation) : 1°) la variable informatique, qui est une fonction de l'exécution (elle peut changer à chaque pas d'un programme), avec des rôles différenciables dans le programme, nous y revenons ; 2°) la notion d'invariant de boucle (au cœur des écritures récursives), une clé pour la validité des programmes : elle est difficile à dégager – même en acte – pour l'écriture de boucles, avec la nécessité de mettre en relation la valeur des données dans l'entrée dans la boucle, l'exécution du corps de boucle et la valeur des données à la sortie de la boucle. Le contrôle de la fin de l'itération ou de la procédure récursive est une difficulté particulière pour les débutants (voire au-delà…). Dans l'étude récente citée plus haut d'un enseignement des bases de la programmation au niveau collège (« middle school » US) les auteurs soulignent :

> Serial execution was the easiest to learn, as expected. Between conditionals and loops, learners found loops harder to tackle. Most of the assessment questions concerning loops required manipulation of variables as well, which seemed to be the hardest topic for students to

[26] Cela dépend évidemment des spécificités des langages. Les « entrées » et « sorties » d'un programme peuvent être respectivement des prises d'information sur le « monde » et l'exécution matérielle d'actions (programmation de robots par exemple).

> grasp. [...] Both these aspects have been known to be particularly difficult for novice programmers (…) Despite our conscious efforts, students struggled with these topics. (Grover, Pea & Cooper, 2016).

Ces constantes ont été relevées dans des introductions faites le plus souvent dans une approche utilisant des rédactions d'algorithmes exprimés en « pseudo-code », c'est-à-dire ni dans un langage de programmation particulier, ni dans un format syntaxiquement contraint, mais dans une forme proche du langage naturel et supposée comprise de tous.

D'autres difficultés/obstacles spécifiques ont été relevés pour la programmation en langage orienté objet (LOO), dont les relations entre classes et les processus d'héritage entre classes et objets. Mais l'écriture des « méthodes » des objets fait rencontrer aux élèves les mêmes notions clés de variables, alternatives et itérations. Il faut souligner l'existence de multiples interactions, d'une part entre les concepts informatiques eux-mêmes, et d'autre part entre les concepts et les activités cognitives.

Ainsi, une des difficultés de la conception de programmes utilisant la récursivité est la nécessité de faire des « raisonnements sous hypothèse », dont le schéma est alors le suivant lors de l'écriture d'un module récursif ou itératif (pour simplifier, récursivité sur n entier) : je suppose que j'ai traité mon problème jusqu'à n, le problème actuel est de trouver comment passer de n à n+1 [c'est justement l'invariant de boucle], puis de chercher quelle valeur de n me permet d'initialiser le processus. Notons que cet obstacle de la récursivité est partagé avec celui de la récurrence en mathématique. Le mode spontané des élèves (pas seulement du secondaire) est de partir de $n = 1$, de chercher à passer à 2, et de coder une forme de « etc. »[27]. Ils « rabattent » ainsi la récursivité sur l'itération, processus mis en échec lorsque l'appel récursif est situé à l'intérieur de procédure et non en la fin de celle-ci. Nous renvoyons à Samurçay et Rouchier (1990) pour une approche didactique de la récursivité lors de l'alphabétisation informatique, et à Rinderknecht (2014) pour une revue de cette question.

Les structures itératives comme récursives illustrent bien le fait que la variable informatique est une fonction de l'exécution. Ce n'est pas sa seule complexité en tant que concept et selon leur rôle dans les programmes, les débutants rencontrent plus ou moins de difficultés.

---

[27] Raisonner « sous hypothèse » à partir d'une prémisse dont la validité est inconnue ou qui est fausse s'est confirmé difficile, même pour des étudiants candidats au professorat de mathématiques (Rogalski et Rogalski, 2004).

La variable (en informatique) est un concept difficile

Nous prenons ici le point de vue de la discipline informatique. Nous avons explicité dans (Rogalski et al, 1987) en quoi il est justifié de considérer la variable informatique comme une fonction, au sens où sa valeur dépend du moment de l'exécution du programme. La variable est évidemment aussi une fonction de l'entrée : c'est un composant de la complexité du concept de « variable informatique ». Cette fonction peut être constante, si la variable ne change pas de valeur au cours de l'exécution.

Cette spécificité de la variable informatique comme « état d'un processus » est relevée par Knuth à propos des différences entre modes de pensée des mathématiciens et des informaticiens (2011, p. 46-47) : « *Un concept manquant qui semble séparer les mathématiciens des informaticiens est relatif à l'opération d'affectation :=, qui change la valeur des quantités. Je devrais dire plus précisément que le concept manquant est celui de notion dynamique d'état d'un processus... La plupart des structures de données si fondamentales en informatique dépendent très fortement de la capacité à raisonner sur la notion d'état de processus.* »

La variable informatique se différencie ainsi de la variable mathématique telle qu'elle apparaît en algèbre, qu'il s'agisse de l'écriture d'expressions, de la résolution d'équations ou des variables dans les fonctions.

La notion mathématique de variable mathématique, telle que les élèves la rencontrent dans le secondaire, peut cependant servir de précurseur : elle a ainsi un rôle producteur, via la familiarité possible de l'élève de manipulation formelle d'expressions algébriques ; mais elle joue aussi un rôle réducteur, par son caractère de permanence au cours d'un traitement (dans la résolution de l'équation $ax + b = c$, x est toujours « le même » et cette invariance est encore plus importante pour résoudre un système d'équations ; qui plus est, il s'agit d'une variable mathématique particulière : l'inconnue, qui n'a pas de pendant en informatique)[28].

[28] La variable mathématique n'est pas pour autant un objet simple pour les élèves (voir Duval & Pluvinage, 2016), en particulier du fait du statut du symbole d'égalité pour les élèves : une indication de résultat ($2x+3x=5x$), une marque d'équivalence entre expressions avec des variables ($(x+3)^2=x^2+6x+9$), la marque d'une inconnue dans une résolution d'équation ($x+1=7$). De plus, le fait que la variable soit un outil pour exprimer une relation (« exprimer l'aire du carré en fonction de son côté ») n'implique pas pour l'élève que le terme « en fonction » exprime une relation particulière entre deux variables.

L'appropriation de la notion de fonction (en mathématiques) peut limiter ce rôle réducteur, ce qui pourrait d'ailleurs contribuer à expliquer l'impact du niveau mathématique des élèves sur leurs acquisitions en informatique[29].

Par ailleurs, le champ conceptuel de la variable informatique fait intervenir les notions de type de variable, la représentation dans une donnée structurée, et l'existence d'un ensemble d'opérations sur la variable. Il comporte aussi des rôles fonctionnels de la variable dans le programme. Enfin, dans la mesure où un programme suppose une modélisation du problème dans les termes du langage utilisé, les relations des variables utilisées avec les composants du problème sont plus ou moins délicates (cf. les exemples traités dans les deux premières parties). Une analyse de ce champ conceptuel a été développée pour l'alphabétisation par Samurçay (1985/1989) et pour la programmation dans l'enseignement supérieur Sajaniemi (2008) et Sorva (2008), qui font référence à Samurçay (1989).

Dans le premier article Samurçay explique comment l'analyse du lien entre les trois concepts articulés de variable, boucle et affectation a conduit à concevoir une séquence didactique (10 leçons, avec des élèves de la classe de Seconde, utilisant le langage PASCAL) autour de l'organisation d'une série de problèmes de sommation de nombres, faisant rencontrer systématiquement quatre formes d'occurrence de l'affectation : valeur constante ; valeur calculée ; duplication A:=B ; accumulation.

Samurçay a distingué de manière générale quatre rôles de la variable, lors de l'alphabétisation (entre parenthèses les termes retenus plus tard par Sajaniemi) : 1°) donnée (*fixed value*) ; 2°) compteur (*stepper*) ; 3°) accumulateur (*gatherer*) ; 4°) intermédiaire pour la programmation (*temporary*). Pour la programmation plus avancée, Sajaniemi a enrichi cet herbier de rôles, en identifiant 6 autres rôles, qui avec les précédents couvrent plus de 99% des rôles rencontrés dans l'enseignement de la programmation, qu'elle soit procédurale ou orientée-objet. (Il n'y a pas de relation univoque avec les statuts différents de la variable mathématique qui

---

[29] L'importance de la notion de variable en programmation informatique a conduit des chercheurs à élaborer un test sur l'interprétation de relations impliquant l'affectation à des étudiants allant entrer dans leur premier cours d'informatique et ignorant de la programmation. Une méta-analyse (Dehnadi, Bornat & Adams, 2009) a montré qu'une bonne cohérence dans l'interprétation de l'opération d'affectation (avant toute introduction à un langage informatique) présageait un meilleur succès dans le début de l'apprentissage de la programmation (mais que leur épreuve ne pouvait pas servir valablement de test de sélection). On pourrait faire l'hypothèse ce qui est en jeu n'est pas tant le concept de variable (que les étudiants construiront plus ou moins bien au cours de leur alphabétisation informatique) que la manipulation d'un registre symbolique arbitraire.

apparaissent au cours de l'enseignement secondaire). Certains de ces rôles s'actualisent dans le traitement de types de données comme les listes, les arbres, les tableaux.

Les propriétés des variables et de leur traitement qui présentent des accès plus ou moins délicats pour les élèves sont leur statut dans la modélisation :

- une variable qui a un rôle d'intermédiaire présentant du sens dans le problème (externalité de la variable) est plus accessible cognitivement que celle nécessitée par l'existence du dispositif informatique (internalité de la variable) ;

- un compteur qui reflète les étapes d'un traitement « à la main » est plus aisé à représenter et utiliser, de même qu'un accumulateur.

L'impact des rôles interfère avec les types de données et les opérations en jeu (en particulier l'initialisation et le test – par sa nature, la structure de données retenue, sa place dans une boucle... Au-delà de l'alphabétisation, Sorva (2008) a souligné par ailleurs que :

> The behavior of a variable, that is the logic that dictates how the variable is used, is often defined at multiple distinct locations in program code [...], may be described by inconsecutive lines of code within a function or a method, located in a number of functions or even located in several program modules (p. 64).

Il a également montré que si on explicite aux étudiants les rôles des variables dans les programmes, ils font davantage de progrès en programmation.

Les élèves (et étudiants) débutants rencontrent des difficultés différentes selon les activités cognitives portant sur la variable informatique

Parmi les activités cognitives spécifiques à la programmation informatique, qui posent problème au débutant, se pose la constitution d'une représentation des différentes opérations que l'on peut effectuer sur les composants du programme, en particulier les variables auxquelles on s'intéresse ici. La déclaration explicite de variable en début de programme (pas toujours imposée par le langage) peut contribuer positivement à la construction d'une représentation du langage de programmation (à un niveau « logique ») ; elle peut contraindre aussi à rencontrer des types de variable non familiers à l'élève (caractères alphanumériques, booléens, tableaux, listes).

Nous avons montré en première partie le saut cognitif que constitue l'utilisation de booléens – et de leur combinatoire – pour exprimer des conditions.

Les registres de représentation (au sens de Duval, 1995) utilisés en mathématiques peuvent ne pas suffire à traiter informatiquement un problème, dès lors que celui-ci fait intervenir d'autres registres dont le sujet n'est pas conscient parce qu'il les a

« naturalisés ». Il en est ainsi dans la lecture d'un nombre entier à la fois comme liste de ses chiffres et comme nombre objet potentiel des « quatre opérations »[30]. L'exemple de l'algorithme de Kaprekar développé en seconde partie illustre cette problématique – en même temps qu'il constitue une situation didactique qui peut être exploitée pour donner du sens à la notion de type de variable (qu'on utilise ou non ce terme). Rôle et déclaration sont des éléments indépendants dans le champ conceptuel de la variable : « *declaring a variable, if it is explicitly done in the language at all, is a matter separate from the variable's behavior* » (Sorva, 2008). Le contrôle des valeurs dans le programme suppose de s'affranchir des présuppositions des traitements à la main (tout au moins de certains). On peut voir dans les exemples d'écriture du programme de résolution de l'équation du premier degré $ax + b = c$, l'absence de prise en compte du cas $a = 0$: le contrat didactique courant appelle à éventuellement préciser $a \neq 0$ dans l'expression du calcul $x = (c - b) / a$, sans que le statut du cas $a = 0$ soit pour autant repéré : la présupposition sous-jacente est que si on parle de résoudre l'équation du premier degré c'est que la variable x apparaît effectivement dans l'expression, ce qui n'est pas le cas si $a = 0$. Il y a certainement interaction entre la tâche mathématique du point de vue algébrique et la notion informatique.

De manière générale, les opérations sur les variables – initialisation, mise à jour, test – peuvent constituer en elles-mêmes des ruptures avec la pratique habituelle des élèves, même avec l'écriture d'un algorithme « papier-crayon », encore plus dans l'exécution d'un algorithme, lors de laquelle l'élève sait bien quelle est la valeur de la variable qu'il traite... L'organisation séquentielle des opérations sur les variables peut aussi être en rupture avec les traitements « à la main » – ce qu'on a observé dans la difficulté plus grande à utiliser des boucles conditionnelles plutôt que des itérations en nombre déterminé (pas de test explicite), la difficulté plus grande à maîtriser les boucles de forme TANT QUE (où on teste une variable avant tout traitement) par rapport aux boucles de forme « jusqu'à » (le test sur la variable est un test d'arrêt en position finale dans la boucle). On rencontre aussi la question de la relation entre le problème à modéliser et la forme plus ou moins appropriée de la modélisation. Le contrôle des valeurs prises par une variable au cours de l'exécution d'un programme est une activité cognitive qui n'est pas non plus dans la continuité des opérations « à la main » ; ici le traitement logique des variables dans le programme interagit avec la manière dont l'élève se représente l'exécution

[30] Sur cet exemple, on pourrait aussi faire l'hypothèse que la distinction nombre/chiffres dans son écriture en base 10 n'est pas construite par l'enseignement de primaire et collège alors qu'elle devrait l'être, et qu'il s'agit là d'un obstacle didactique (nous reprenons cette remarque d'un des relecteurs).

du programme (une perspective de recherche a concerné les outils logiciels qui peuvent aider). En fait, une propriété fondamentale sous-jacente à nombre de ces difficultés cognitives de représentation et de traitement de la variable informatique est que celle-ci est une fonction. On pourrait la rapprocher de ce qui est rencontré plus tard par les élèves (dans la seconde année de lycée, en France), à savoir la notion de suite numérique – dont on peut se demander quand elle a pour eux le statut de fonction[31].

4. Conclusion

Nous reprenons ici des éléments de bilan en mettant en avant les problématiques de recherche qui auraient matière à se développer en didactique des mathématiques.

Confrontant des observations dans l'enseignement de l'algorithmique et de la programmation – revenu en France dans le curriculum du lycée depuis quelques années – à l'expérience acquise antérieurement, nous avons montré la persistance de difficultés conceptuelles chez les élèves débutants, souvent mal connues des acteurs du terrain ou institutionnels. Il apparaît que l'enjeu central est, pour les élèves que nous avons observés, de comprendre la construction d'un programme ou d'un algorithme comme l'organisation d'un traitement sur un dispositif ; ils doivent percevoir ce dispositif comme un ensemble de variables, et concevoir ces variables comme des objets « calculables » et le traitement comme l'évolution de leurs valeurs. Il existe peu de travaux proprement didactiques sur la manière d'amener les élèves à cette compréhension. Ceci renvoie – avec des années de décalage – à l'époque des travaux initiaux en didactique des mathématiques et en didactique des sciences qui étaient centrés soit sur des approches purement épistémologiques, soit sur les « erreurs » ou « *misconceptions* » des élèves. Quelques études, rappelées ici, ont cependant montré comment, dans des mises en œuvre de la théorie des situations didactiques, un travail de conception d'un milieu adéquat permet au chercheur de bien caractériser les savoirs en jeu dans l'activité des élèves. Dans le contexte de la responsabilité dévolue aux enseignants de mathématiques de prendre en charge l'enseignement d'éléments d'algorithmique et de programmation, ces travaux sont des points d'appui pour la pratique enseignante et pour les développements de recherches didactiques.

Alors que la didactique des mathématiques s'est assez peu saisie de la

[31] À notre connaissance, la question d'une interaction constructive entre la conception de la variable informatique comme variable « indexée » par l'étape du traitement et la notion de suite numérique en mathématique n'a pas été étudiée.

problématique de l'enseignement de la programmation, c'est la psychologie cognitive qui a d'abord approfondi la compréhension des enjeux épistémologiques et cognitifs dans les apprentissages en programmation – notamment par la mise en évidence d'une pluralité de rôles attribués aux variables et de la nécessité de considérer des invariants dans les traitements conditionnels – ceci renvoyant aux difficultés rencontrées lors de « l'alphabétisation informatique ». La psychologie cognitive a aussi souligné le rôle de savoirs précurseurs qui peuvent jouer un rôle producteur en permettant aux élèves qui en disposent d'entrer dans les tâches d'algorithmique et de programmation, mais qui peuvent aussi jouer un rôle réducteur dont l'enseignement doit tenir compte.

Cela conduit à questionner plus avant les concepts communs[32] en mathématiques et en algorithmique, et les convergences dans l'activité souvent relevés au plan de l'épistémologie. Des compétences en mathématiques sont-elles réellement des précurseurs pour les apprentissages en algorithmique et programmation ou existe-t-il plutôt des corrélations tenant à des précurseurs communs plus généraux, comme par exemple une bonne manipulation des opérations de raisonnement, en particulier pour les tests, ou de bonnes représentations des nombres ou de l'espace ?

Par ailleurs, la psychologie cognitive montre que la notion de « variable informatique » est cruciale, mais aussi plus largement qu'algorithmique et programmation mettent en jeu des registres symboliques spécifiques. Comme ceux-ci s'articulent (plus ou moins aisément pour les élèves) avec les registres mathématiques, les travaux sur la dimension sémiotique dans l'activité mathématique pourraient donc fournir des appuis, avec en particulier la notion de registres de représentation (Duval, 1995), pour un meilleur contrôle didactique des situations proposées en classe. Quelles activités de programmation peuvent permettre effectivement à ces registres, avec les traitements et les conversions qui leur sont attachés (op. cit.), d'être des outils disponibles chez les élèves, conduisant à des effets en retour de l'apprentissage de la programmation sur les compétences algébriques des élèves ?

Une autre perspective de convergence concerne la modélisation. Alors que l'enseignement de l'informatique dans des options spécifiques au lycée, et à l'université, proposent des situations de « projet » partant problèmes

---

[32]Les concepts qu'on pourrait identifier comme communs au niveau universitaire ou dans l'informatique théorique, ne sont pas ceux que nous avons considérés dans cet article, centré sur le lycée et des élèves débutants. Une analyse de ces concepts en termes de paradigmes pourrait être menée, comme celle qui a été faite pour la géométrie par Houdement et Kuzniak (2006).

« consistants » en termes de conception d'un modèle de la situation permettant son traitement informatique, les travaux que nous avons repérés dans les premières parties de cet article concernent des situations « simplifiées » visant spécifiquement à faire rencontrer certains concepts clés de l'algorithmique. Un courant de recherche sur la modélisation s'est développé en didactique des mathématiques : par exemple Barquero, Bosch et Gascón (2013) soulignent le rôle de la conception de parcours de recherche et de leur mise en œuvre par l'enseignant pour que l'engagement des élèves se traduise par l'acquisition de concepts mathématiques. La question est ouverte de la contribution de ce courant à l'étude de la modélisation dans le contexte de l'algorithmique et programmation. L'intégration de ce domaine dans le champ de l'enseignement mathématique introduit également la question spécifique de la modélisation d'objets mathématiques (concepts et traitements) dans le cadre de traitements exprimés en vue de leur exécution par un dispositif informatique (Lagrange et Guy 2014, Briand 2015).

Un thème de recherche plus spécifiquement didactique concerne l'impact des enseignements introduits dans les programmes français de primaire et de collège sur l'enseignement en lycée, puis les prolongements dans l'enseignement supérieur (largement considérés maintenant dans les recherches internationales). Les interrogations sont multiples. Elles portent à la fois sur les effets de maturation conceptuelle des élèves, sur le rôle de précurseurs nouveaux potentiels et sur les phénomènes liés à la transition entre collège et lycée (à supposer que celle entre primaire et collège soit réglée par la considération de cycles couvrant les deux niveaux scolaires). Cela devrait par ailleurs conduire aussi à des recherches sur ce qu'il en est pour l'enseignement technique et professionnel, qui reste un parent pauvre des recherches de didactique des mathématiques en France.

Il nous a paru nécessaire de rappeler à la fois les acquis de travaux anciens sur les difficultés et acquisitions des élèves débutants en informatique et leur pertinence dans le contexte actuel en France au lycée, car nous les considérons comme une base pour développer la recherche sur de nouvelles problématiques. Nous les considérons aussi comme des données utiles pour les enseignants, non pour les décourager face aux difficultés des élèves, mais pour que la rencontre de ces difficultés ne les conduise pas à des désillusions sur l'intérêt, la nécessité et la possibilité cet enseignement.

Nous partageons la visée sociale d'enseignements intégrant la programmation et ouverts aux jeunes de tous âges, de façon que, élèves puis citoyens, elles et ils prennent conscience par leur propre expérience de l'activité humaine à l'origine de toute application informatique, et de la rationalité qui préside à cette activité. Nous prenons acte de ce que, en France notamment, ces enseignements sont pour partie intégrés au curriculum de mathématiques. Ceci donne des responsabilités particulières à la didactique de cette discipline et c'est pourquoi nous avons insisté

sur les acquis pour ouvrir à de nouveaux développements de la recherche.

Bibliographie

BARON, G.-L. (1989). *L'informatique discipline scolaire*. Paris : PUF.

BARQUERO, B., BOSCH, M. GASCÓN, J. (2013). The ecological dimension in the teaching of mathematical modelling at university, *Recherches en Didactique des Mathématiques*, **33.3**, 307-338.

BRIANT, N. (2013). *Étude didactique de la reprise de l'algèbre par l'introduction de l'algorithmique au niveau de la classe de seconde du lycée français*. Thèse Université Montpellier II - Sciences et Techniques du Languedoc.

BROLEY, L. (2016) The place of computer programming in (undergraduate) mathematical practices. In Nardi & Winslow (Eds), *Proceedings of the First conference of International Network for Didactic Research in University Mathematics, 31 Mar-2 Apr 2016*, 197-206, Montpellier.
http:// sciencesconf.org:indrum2016:84523

BROUSSEAU, G. (1988) *Théorie des situations didactiques*. Grenoble : La Pensée Sauvage.

CHEVALLARD, Y. (1989). Le passage de l'arithmétique à l'algèbre dans l'enseignement des mathématiques au collège. Deuxième partie. Perspectives curriculaires : la notion de modélisation. *Petit x*, **19**, 43-72.

COUDERETTE, M. (2016). Enseignement de l'informatique en classe de seconde : une introduction curriculaire problématique. *Annales de Didactique et Sciences Cognitives*, **21**, 267-296.

CRAHAY, M. (1987). Logo, un environnement propice à la pensée procédurale ? *Revue française de pédagogie*, **80.1**, 37-56.

DA ROSA, S. (2015). The construction of knowledge of basic algorithms and data structures by novice learners. In M. Coles & G. Ollis (Eds.), *Proceedings of the Psychology of Programming Interest Group Annual Conference 2015* (pp. 37-47). http://ppig.org/sites/default/files/2015-PPIG-26th-proceedings.pdf (consulté 17/2/2016).

DEHNADI, S., BORNAT, R., & ADAMS, R. (2009). Meta-analysis of the effect of consistency on success in early learning of programming. *Proceedings of the 21th PPIG Conference*.

DROT-DELANGE, B. (2012). Enseignement de l'informatique, éducation aux technologies de l'information et de la communication en France, dans l'enseignement général du second degré. *Spirale : revue de recherches en*

éducation, Lille : Association de pédagogie et de didactique de l'École normale de Lille, **50**, 25-37.

DROT-DELANGE, B. (2014). Littératie informatique : quels ancrages théoriques pour quels apprentissages ? *Spirale : revue de recherches en éducation*, **53**, 121-132.

DU BOULAY, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, **2.1**, 57-73.

DURAND-GUERRIER,V. (1996). *Logique et raisonnement mathématique. Défense et illustration de la pertinence du calcul des prédicats pour une approche didactique des difficultés liées à l'implication*. Thèse de l'Université Lyon 1.

DUVAL, R. (1995). *Sémiosis et pensée humaine*. Bern : Peter Lang.

DUVAL, R., & PLUVINAGE, F. (2016). Apprentissages algébriques - première partie ; points de vue sur l'algèbre élémentaire et son enseignement. *Annales de Didactique et de Sciences Cognitives*, **21**, 117-152.

ENGEL, A. (1979) *Mathématique élémentaire d'un point de vue algorithmique* : (adapté par Daniel Reisz), Paris : CEDIC.

GROVER, S., PEA, R.D., & COOPER, S. (2016). Factors influencing computer science learning in middle school. *SIGCSE '16*, March 02-05, Memphis, TN.

GUY, M.-N. (2013). *Utilisation du cadre théorique de la planification pour la conception d'algorithmes complexes par des élèves de lycée*. Mémoire de master de Didactique des mathématiques. Université Paris Diderot.

HOUDEMENT C., KUZNIAK A. (2006) Paradigmes géométriques et enseignement de la géométrie. *Annales de Didactique et de Sciences Cognitives*, **11**, 175-195.

HASPEKIAN, M., & NIJIMBERE, C. (2016). Favoriser l'enseignement de l'algorithmique en mathématiques. *Éducation et Didactique*, **10.3**, 121-135.

HOC, J. M., GREEN, T. R. G., SAMURCAY, R., J. GILMORE D. J. (Eds). (1990). *Psychology of programming*. London: Academic Press.

KNUTH, D. (1968). *The Art of Computer Programming*. Addison-Wesley.

KNUTH, D. (2011). *Éléments pour une histoire de l'informatique*. (Articles choisis, traduction de P. Cégielski). Stanford CSLI Publications & SMF.

KURLAND, D. M., PEA, R. D. (1986) Children's Mental Models of Recursive Logo Programs, Journal of Educationa1 Computing Research, **1.2**, 235-243.

LABORDE, C. (Ed.) (1988). *Actes du premier colloque franco-allemand de didactique des mathématiques et de l'informatique.* Grenoble : La Pensée Sauvage.

LAGRANGE, J-B. (1991) *Des situations connues aux traitements sur des données codifiées : représentations mentales et processus d'acquisition dans les premiers apprentissages en informatique*. Thèse de Doctorat. Université Paris 7. http://jb.lagrange.free.fr/

LAGRANGE J.B. (2014). Algorithmics. In S. Lerman (ed.), *Encyclopedia of Mathematics Education*, DOI 10.1007/978-94-007-4978-8, Springer Science+Business Media Dordrecht 2014.

LAGRANGE, J.-B., & GUY, M.-N. (2015). Planification et connaissances mathématiques dans une situation d'apprentissage au lycée : l'algorithme de Kaprekar. *Petit x*, **97**, 45-70.

LAVAL, D. (2015). L'algorithmique comme objet d'apprentissage de la démarche de preuve en théorie élémentaire des nombres : l'algorithme de Kaprekar. *Actes du Quatrième symposium international Espace de Travail Mathématique* (pp. 103-116). Madrid : I.M.I.

NGUYEN, C. T. (2005). *Étude didactique de l'introduction d'éléments d'algorithmique et de programmation dans l'enseignement mathématique secondaire à l'aide de la calculatrice*. Thèse de l'université Joseph Fourier, Grenoble.

NGUYEN, C. T., & BESSOT, A (2010). Introduire des éléments d'algorithmique et de programmation dans l'enseignement secondaire ? Une ingénierie didactique. *Petit x*, **83**, 27-49.

PAPERT, S. (1981). *Jaillissement de l'esprit, ordinateurs et apprentissages* (texte français de R.M. Vassallo-Villaneau). Paris : Flammarion.

PIROLLI, P. L. (1986). A cognitive model and computer tutor for programming recursion. *Human Computer Interaction*, **2.4**, 319-355.

RINDERKNECHT, C. (2014). A Survey on Teaching and Learning Recursive Programming. *Informatics in Education*, **13.1**, 87-119.

ROGALSKI, J. (1988). Les représentations mentales du dispositif informatique dans l'alphabétisation. in C. Laborde (Ed.), *Actes du 1er colloque franco-allemand de didactique des mathématiques et de l'informatique* (pp. 237-245). Grenoble : La Pensée Sauvage.

ROGALSKI, J. (2015). Psychologie de la programmation, didactique de l'informatique : déjà une histoire… In G.-L. Baron, E. Bruillard, E., & B. Drot-Delange (Eds.), *L'information en éducation : perspectives curriculaires et didactiques* (pp. 279-305). Clermont-Ferrand : Presses Universitaires Blaise Pascal.

ROGALSKI, J., & HÉ, Y. (1989). Logic abilities and mental representations of the informatical device in acquisition of conditional structures by 15-16 year-old students. *European Journal of Psychology of Education*, **4.1**, 71-82.

ROGALSKI, M., SAMURÇAY, R. (1990) Acquisition of programming knowledge and skills. In J. M. Hoc, T. R. G. Green, R. Samurcay, D. J. Gilmore (Eds) *Psychology of programming* (pp. 157-173). London: Academic Press.

ROGALSKI, J., ROGALSKI, M. (2004). Contribution à l'étude des modes de traitement de la validité de l'implication par de futurs enseignants de mathématiques. *Annales de Didactique et de Sciences Cognitives*, **9,** 175-203.

ROUCHIER, A. (1987). The writing and interpretation of recursive procedures in LOGO. *Psychologie Française*, **32.4**, 281-285.

ROUCHIER, A. (1990). Objets de savoir de nature informatique dans l'enseignement secondaire. *ASTER,* **11**, 29-44.

RUTHVEN, K. (1996). Calculators in the mathematics curriculum: The scope of personal computational technology. In A. Bishop, K. Clements, C. Keitel, J. Kilpatrick & C. Laborde (Eds.) *International Handbook of Mathematics Education* (pp. 435-468). Dordrecht : Kluwer.

SAJANIEMI, J. (Ed.). (2008). Special issue on Psychology of Programming. *Human Technology*, **4.1**.

SAMURÇAY, R. (1985). Signification et fonctionnement du concept de variable informatique chez des élèves débutants. *Educational Studies in Mathematics*, **16.2**, 143-161.

SAMURÇAY, R. (1989). The concept of variable in programming : Its meaning and use in problem solving by novice programmers. In E. Soloway & J.-C. Spoher (Eds.), *Studying the novice programmer* (pp. 161-178). Hillsdale, NJ : Lawrence Erlbaum Associates.

SAMURÇAY, R., & ROUCHIER, A. (1990). Apprentissage de l'écriture et de l'interprétation des procédures récursives. *Recherches en didactique des Mathématiques,* **10.2-3**, 287-327.

SORVA, J. (2008). A Roles-Based Approach to Variable-Oriented Programming. *Human Technology*, **4.1**, 62-74.

TALL, D.O., & THOMAS, M.O.J. (1986). The value of the computer in learning lgebra concepts. *Proceedings of the 10th Conference of PME* (pp. 313-318). London : University of London Institute of Education.

VANDEBROUCK, F. (Ed.). (2008). *La classe de mathématiques : activités des élèves et pratiques des enseignants*. Toulouse : Octarès.

VANDEBROUCK, F. (Ed.). (2013). *Mathematics classrooms. Students' activities and teachers' practices*. Sense Publishers.

JEAN-BAPTISTE LAGRANGE

Adresse

jb.lagrange@casyopee.eu

JANINE ROGALSKI

Adresse

rogalski.muret@gmail.com