ENSEIGNER LA PROGRAMMATION : QUOI ?, POURQUOI ?, COMMENT ?

Jean-Pierre PEYRIN
Professeur à l'Université Joseph Fourier - Grenoble 1
LGI - IMAG ; BP 53 - F 38041 Grenoble cedex 9

1. Introduction :

L'enseignement de la programmation a déjà été largement discuté lors des trois précédentes rencontres. A Paris (1988), on comparait des manières d'enseigner des méthodes de programmation, et on s'appuyait essentiellement sur une programmation basée sur des actions[[1]]. A Namur (1990), on comparait la pertinence des différents styles de programmation, et on défendait une programmation basée sur des fonctions [[2]]. A Sion (1992), on étudiait la possibilité de faire programmer à l'aide d'autres outils que les langages, et on abordait une programmation basée sur des objets [[3]].

Lors de ces trois rencontres, le fait de devoir enseigner la programmation n'a jamais été contesté. La programmation était considérée comme le domaine le plus représentatif de la science informatique et semblait donc incontournable. Cette quatrième rencontre doit aborder la question de la raison et de la nécessité d'un enseignement de la programmation dans la formation générale. Ce texte prétend plus poser des questions, argumentées, qu'apporter des réponses. Un résumé des questions est contenu dans le titre. Qu'est-ce que la programmation ? Quel serait son apport à la formation ? Quelle serait sa pédagogie ?

Un exemple :

Observons un extrait, reformulé, d'une épreuve optionnelle d'informatique du baccalauréat français, en 1993 [[4]] :

> *On considère les deux procédures récursives X1 et X2 suivantes, ayant toutes les deux le paramètre C de type chaîne de caractères :*
>
> ```
> procédure X1(C)                 procédure X2(C)
>     si non Vide(C)                  si non Vide(C)
>     alors                           alors
>         afficher(Premier(C))            X2(SaufPremier(C))
>         X1(SaufPremier(C))              afficher(Premier(C))
> ```
>
> *Question 1 : Expliquez en détail ce que réalise l'appel de la procédure X1('SALUT').*
> *Question 2 : Expliquez en détail ce que réalise l'appel de la procédure X2('SALUT').*

Pour comprendre, profondément, qu'il faut répondre 'SALUT' à la première question et 'TULAS' à la deuxième, il faut, en fait, comprendre le sens de chaque procédure. Pour comprendre le sens d'une procédure récursive non spécifiée, il faut faire une hypothèse (de récurrence) sur ce sens. C'est un exercice très intéressant, mais particulièrement difficile. Les trois années d'enseignement optionnel d'informatique, à raison de deux heures trente par semaine, ne sont pas un luxe pour dominer une telle difficulté.
Mais on doit admettre les réticences de ceux qui ne souhaitent pas faire 240 heures d'études pour être capable d'écrire une procédure qui réécrit un mot tel quel, et une autre qui l'écrit à l'envers !

Un contre-exemple :

Observons les principaux sujets de philosophie des séries scientifiques du baccalauréat français en 1993 [5] :

Est-il souhaitable de réaliser tout ce qui est techniquement possible ?
L'expérience instruit-elle ?
Le travail n'est-il pour l'homme qu'un moyen de subvenir à ses besoins ?
Faut-il attendre de la science qu'elle nous rassure ?
Le vrai est-il toujours vraisemblable ?
L'Etat est-il l'ennemi de la liberté ?
Faut-il vivre avec son temps ?
La difficulté de comprendre les autres fausse-t-elle tout rapport avec eux ?
La discussion n'a-t-elle pour but que l'accord avec autrui ?
Tout homme a-t-il droit au respect ?

Les thèmes abordés sont profonds et touchent à de nombreux problèmes de la vie. Les élèves peuvent les traiter bien qu'ils n'aient étudié la philosophie que pendant une année à raison de trois heures par semaine !

Certes, ils les traitent avec leur âge et leurs moyens, mais ils sauraient les traiter de nouveau, plus tard, avec une maturité différente. Les moyens d'expression évoluent plus que les sujets, alors que c'est le contraire en informatique.

Un autre exemple :

Nous utilisons, depuis de nombreuses années, une série d'exercices types pour introduire une méthode de construction systématique d'une itération en DEUG ou en licence [6,7]. Ces exemples traitent des textes considérés comme des séquences de caractères. Cette série est la suivante :

- *Longueur d'un texte* : il s'agit de construire une itération simple de parcours de la séquence des caractères.
- *Nombre de 'A'* : Il s'agit de construire une itération comportant une conditionnelle. Une autre solution consiste à calculer la longueur (exercice précédent) de la séquence des 'A' extraite de la séquence donnée.

- *Nombre de 'LE'* : Il s'agit de construire une itération comportant une conditionnelle et la mémorisation du caractère précédent. Une autre solution consiste à compter le nombre de 'L' (exercice précédent) dans la séquence des lettres qui précèdent les 'E'. Une troisième solution consiste à calculer la longueur (premier exercice) de la séquence des 'E' précédés par un 'L' .
- *Nombre de mots* : Il s'agit de se ramener au comptage des couples (exercice précédent) formés d'un séparateur et d'une lettre. Une autre solution consiste à calculer la longueur (premier exercice) de la séquence des premières lettres des mots.
- *Longueur moyenne des mots d'un texte* : Il s'agit de se ramener au comptage du nombre de lettres (deuxième exercice) et du nombre de mots (quatrième exercice), éventuellement avec fusion des deux itérations. Une autre solution consiste à définir l'algorithme de la moyenne (somme et longueur) d'une séquence de nombre et de l'appliquer à la séquence des longueurs des mots.
- *etc.*

Ces exercices sont justifiés lorsqu'ils forment le début d'un enseignement de la programmation pour des futurs informaticiens. Mais ils peuvent être prodigieusement rébarbatifs et inacceptables pour des élèves qui doivent simplement comprendre, en peu de temps, ce qu'est l'informatique. Accepterions nous, comme première et seule découverte de l'art de la musique, de n'étudier que les premiers exercices de solfège et de ne pratiquer que des gammes ?

La programmation est manifestement une science difficile. Il faut beaucoup de temps d'apprentissage pour faire faire des petits dessins à l'aide de LOGO, ou pour faire gérer une petite collection d'informations à l'aide de PASCAL, ou pour écrire une petite procédure récursive. La complexité de la construction d'un programme est sans commune mesure avec la simplicité des problèmes traités. Cela peut être troublant pour les élèves et peut contribuer à donner de l'informatique une image contestable, voire négative.

Dans le vie professionnelle courante, l'utilisation de l'ordinateur n'exige plus de savoir programmer. Enseigner la programmation n'a donc pas un but pratique.

2. Ebauche de définition :

Programmer, c'est [8] :

- faire faire en différé, et de manière répétitive (en ce sens, faire le "programme" d'un colloque n'est pas de la programmation).

- organiser :
  . ce qu'il faut faire faire (le traitement)
  . ce sur quoi il faut faire faire (les informations)

- maîtriser un système :
  . prédire son comportement
  . faire exécuter sans faille
  . avoir conscience de ce qui se passe

. ne pas être pris au dépourvu

- être capable d'abstraction :
  . repérer ce qui est répétitif
  . paramètrer
  . exprimer cette abstraction

Il s'agit donc de savoir faire faire quelque chose à un ordinateur pour obtenir un résultat juste et prévisible, en un temps fini, en tenant compte des possibilités et des limites du matériel.

La programmation, au sens traditionnel, c'est :
- la programmation par actions (BASIC, LSE, LOGO, PASCAL, …)
- la programmation par fonctions (LISP, SCHEME, …)
- la programmation par relations logiques (PROLOG, …)
- la programmation par objets (SMALLTALK, PASCAL OBJET, C++, …)

En fait, la plupart des utilisations de l'ordinateur consistent à le "programmer" ou bien comportent des éléments de "programmation". Il pourrait donc ne pas être nécessaire de pratiquer un "langage" pour comprendre et pratiquer la programmation.

3. Pratique d'un environnement Hypermédia :

On réduit généralement la programmation à la construction d'algorithmes ; il est vrai que les langages usuels l'imposent. On oublie que la programmation est aussi la science de l'organisation (en modules). L'approche "objets" permet d'aborder l'aspect "organisation" avant l'aspect "algorithmique" et c'est sans doute une approche plus générale (décomposition d'un problème en objets et définition de leurs comportements et de leurs relations). La principale difficulté est que la programmation par objets nécessite également l'usage de langages dont la rigueur syntaxique peut inhiber la créativité.

Une solution pourrait être l'usage d'un environnement tel qu'HyperCard ou ToolBook. Il est possible de construire en relativement peu de temps, sans connaître les concepts complexes de la programmation par objets, des applications intéressantes et motivantes dont l'algorithmique usuelle est quasiment absente.

Un exemple de TP [9] :

Un excellent TP consiste à simuler une calculette. La "présentation" est un simple dessin. Les dix boutons de désignation des dix chiffres sont les "objets" d'une même "classe" ; les boutons de désignation des quatre opérateurs sont les "objets" d'une autre "classe" ; etc.

Ce TP ne comporte aucune reflexion algorithmique. Le travail essentiel est la définition de ce qui relie les différents boutons, l'affichage et un accumulateur (invisible).

La conception d'une application est proche de la programmation par objets : on construit les

différents objets à l'aide de classes prédéfinies, on les caractérise en précisant leurs attributs ; mais on ne peut pas définir directement ses propres classes et les relations sont entre les objets (hiérarchie), pas entre les classes (héritage).

On doit définir les méthodes à l'aide d'un langage (trop) classique, et c'est un point faible de ces environnements (dans le cadre du propos de ce texte) : Pascal n'a rien à envier à HyperTalk ou à OpenScript ! L'extrait suivant d'un script de la simulation de la calculette n'est pas plus alléchant pour un non-informaticien que les procédures récursives du premier exemple introductif :

```
on MUchiffre
  global Op, étatFenêtre
  if étatFenêtre = "affichage" then
    put "entrée" into étatFenêtre
    put the short name of the target into card field "fenêtre"
  else
    put card field "fenêtre" into n
    put n * 10 + the short name of the target into n
    put n into card field "fenêtre"
  end if
end MUchiffre

on MUopBin
  global Op, étatFenêtre
  put card field "acc" into g
  put card field "fenêtre" into d
  send "opération g, d" to card button Op
  put "affichage" into étatFenêtre
  put the short name of the target into Op
end MUopBin
```

4. Situations de programmation :

De nombreux exemples de situations de programmation se rencontrent dans la pratique des logiciels usuels :

- la définition des indications de mise en page dans un traitement de texte est une façon de "programmer" l'aspect (futur) du texte.
- la définition de formules dans un tableur est une façon de programmer les résultats (futurs) du tableau.
- la définition de liens entre des paramètres dans un texte et des rubriques d'une base de données est un réel travail de programmation. Le publipostage est un excellent TP de programmation qui demande réflexion, organisation et rigueur sans nécessiter le moindre usage d'un langage textuel. Le TP consiste en la définition d'un modèle de données et en la définition d'un modèle de texte paramétré en termes des composants du modèle de données ; la base de données est alors créée pour permettre la production de tous les textes correspondants.

Des situations de programmations se rencontrent également dans certains jeux. On peut, par exemple, définir une activité de programmation à l'aide du jeu "Lode Runner" [10]. Ce jeu

met en scène un voleur devant ramasser des sacs d'or sans se faire capturer par des policiers.

Un premier intérêt du jeu est la recherche de stratégies de résolution. C'est sans doute une activité formatrice, entre autres pour l'informatique.

Mais le principal intérêt du jeu, ici, est que l'on peut construire soi-même le décor dans lequel évoluent les personnages.

En somme, on "programme" le jeu avant de jouer. La motivation est généralement plus forte que pour le parcours d'un tableau ! Il faut noter que cette "programmation" est "par objets" : les éléments du décor sont les "objets" dont le rôle et le comportement sont prédéfinis ; les personnages sont les "messages" qui interviendront lors du jeu ("l'éxécution"). La principale difficulté est qu'un tel exercice, ludique, s'insère mal dans un système fondé plus sur les disciplines (les lobbies ?) que sur les activités.

D'autres jeux offrent de telles situations de programmation. Par exemple MacGolf [11], une simulation du jeu de golf, ou le joueur dispose d'outils pour "définir" le coup avant de le jouer (swing) : il observe, ensuite, l'exécution (ce que fait la balle). Un autre exemple est le jeu de flipper [12], où l'on dispose d'une boite à outils de construction de son propre jeu.

5. Conclusion

Il est impensable d'enseigner la science informatique sans faire comprendre ce qu'est la programmation (le fameux "faire faire"). Mais, programmer, c'est bien plus que d'écrire un programme en LOGO, en PASCAL, ou en SCHEME… La programmation est sans doute une activité intellectuelle riche et formatrice, mais elle est exigente et il faut comprendre sa fonction pédagogique. Nous devons préciser ce que nous en attendons pour définir les modalités de son éventuelle introduction dans la formation générale.

---

[1] Actes du premier colloque francophone sur la didactique de l'informatique. Publiés par l'E.P.I., Paris, 1988.

[2] Actes du deuxième colloque francophone sur la didactique de l'informatique. Publiés par les Presses Universitaires de Namur, 1990.

[3] Actes de la troisième rencontre francophone de didactique de l'informatique. Publiés par l'E.P.I., l'A.F.D.I. et l'Institut Universitaire Kurt Bösch, Sion, 1992.

[4] Bulletin de l'EPI n° 72 (décembre 1993)

[5] Millésimés Bordas. BAC 94. Philosophie, toutes séries, sujets 93, corrigés.

[6] M. Lucas, J-P. Peyrin, P-C. Scholl : algorithmique et représentation des données. Volume 1 : files, automates d'états finis. Masson, 1983

[7] P-C. Scholl, J-P. Peyrin : schémas algorithmiques fondamentaux.

Séquences et itération. Masson, 1988.

[8] M. Lucas, J-M. Bérard, J-C. Boussard, D. Monasse, C. Patoux, J-P. Peyrin, R. Raynaud: Enseignement d'un ensemble de notions et savoir faire informatiques à l'école primaire, au collège et au lycée. Problèmatique générale. Note n°16 du GTD Informatique (juin 1993).

[9] Exemple emprunté à Isabelle Borne (utilisé dans son atelier à Sion en juillet 1992).

[10] Doug Smith & Glenn Axworthy : Lode Runner. © 1983 : Bröderbund Software.

[11] Robert Pappas : MacGolf. © 1985 : Practical Computer Applications, Inc.

[12] Bill Budge & Bob Upshaw : Pinball Construction Set.