https://laurentbloch.org/BlogLB/Pourquoi-Scheme

Pourquoi Scheme ?

- Zinformatiques - Cours de bioinformatique au CNAM -

Date de mise en ligne : lundi 11 juillet 2005

Depuis le siècle dernier j'ai bien dû enseigner une demi-douzaine de langages de programmation, et jamais je n'ai échappé à une question plus ou moins pressante : « *pourquoi le langage L1, et pas plutôt L0, que tout le monde utilise ?* À cette question récurrente j'ai déjà proposé quelques éléments de réponse, mais puisqu'elle revient, j'y reviens. Le sujet abordé ici est celui de l'enseignement à des débutants : apprendre un second langage pour quelqu'un qui en connaît déjà un est un problème différent et moins difficile. On pourra aussi consulter quelques réflexions différentes sur le même sujet.

Il y aurait déjà beaucoup à dire sur l'assertion « *L0, que tout le monde utilise* », mais laissons peut-être cela pour un prochain article.

Le choix d'un langage pour enseigner la programmation exige un certain soin, parce que les progrès des étudiants et la qualité de leur apprentissage en dépendent pour une part. Il n'est pas vrai que ce choix est indifférent. Il est possible d'énumérer quelques critères à prendre en considération :

1. le langage doit être suffisamment simple pour que le débutant puisse assez vite en avoir une vue d'ensemble, et soit capable d'écrire des programmes simples de façon autonome ;
2. un élément important de cette simplicité réside dans la syntaxe du langage, dont les qualités essentielles me semblent être la sobriété, la régularité et l'expressivité [1] ; comme l'a souligné Matthias Felleisen, l'apprentissage de la syntaxe est du temps perdu, parce que cela ne présente aucun intérêt, il convient donc qu'il soit réduit au minimum ;
3. le niveau d'abstraction doit être suffisant pour que le débutant ne soit pas d'emblée contraint de comprendre le fonctionnement du système d'exploitation et du processeur sous-jacents, compréhension hors de sa portée tant qu'il n'a pas acquis quelque expérience de la programmation ;
4. mais le niveau d'abstraction ne doit pas non plus être excessif, ce qui serait un obstacle à l'acquisition des mécanismes fondamentaux : si la recherche dans une table, le tri ou la manipulation d'une table associative sont des primitives du langage, il est impossible d'apprendre à les programmer, or cet apprentissage est fondamental ;
5. du point précédent découle que l'apprentissage de la programmation avec des objets pourra compléter l'apprentissage de la programmation procédurale, mais ne pourra pas s'y substituer : la programmation procédurale et la programmation récursive sont des compétences fondamentales et indispensables à qui veut se proclamer informaticien, ou bio-informaticien [2] ;
6. c'est un détail technique, mais il a son importance : le langage doit posséder un système de gestion automatique de la mémoire, de préférence un glaneur de cellules *(garbage collector)* ; gérer l'allocation dynamique de mémoire est une complication dont il est bon de faire l'économie au début, il y a bien assez de choses à apprendre avec le reste ;
7. enfin, le langage doit pouvoir être compilé : avec les langages uniquement interprétés, le débutant a trop de mal à distinguer les objets créés par son programme de ceux qu'il trouve dans l'environnement de l'interprète, et ne peut guère se former une représentation du programme comme un objet distinct, physiquement et logiquement.

De mon expérience pratique, je n'ai rencontré que deux langages appropriés à l'apprentissage des débutants : *Turbo Pascal* et Scheme.

Ada est un fort beau langage, dont la syntaxe a bien des qualités, mais l'étendue des connaissances à posséder pour commencer à voler de ses propres ailes est telle que bien des étudiants auront perdu pied avant de pouvoir se raccrocher à l'exercice pratique de la programmation, qui permet de reprendre de l'assurance.

Le langage C est indispensable au paquetage de l'informaticien en campagne, mais de trop bas niveau, c'est-à-dire trop proche du système et du matériel, pour le débutant, qui de par son état n'est pas encore en mesure d'en

comprendre le fonctionnement. De surcroît, la syntaxe de C semble faite pour décourager l'apprentissage. Combien de biologistes n'ai-je pas vus, obligés de programmer en C sans maîtriser le langage, et de ce fait entraînés par des courants pleins de traîtrise vers des bancs de sable où ils s'échouaient sans recours ?

Aujourd'hui le leitmotiv c'est « Pourquoi pas Java ? » Ma réponse est simple : Java est un langage raisonnable pour l'enseignement initial de la programmation, il répond (avec plus ou moins de bonheur) à la liste de critères énumérés ci-dessus. Aujourd'hui, le cours d'initiation à la programmation, basé sur Scheme, comporte 30 heures de cours et 30 heures de TP, à l'issue desquelles les élèves arrivent à écrire des programmes simples. Si nous devions faire le même cours avec Java, pour atteindre le même résultat il nous faudrait le triple, soit 180 heures au total. Pourquoi ? Parce que :

– Java est un langage plus complexe que Scheme, pour l'apprendre il faut acquérir en parallèle ses aspects procéduraux et ses aspects objet, ce qui n'est pas simple pour un débutant ;

– la syntaxe de Java est assez punitive ;

– les environnements de développement comme *Eclipse*, les bibliothèques de classes toutes faites sont formidables pour les gens qui savent déjà programmer, pour des débutants il s'agit d'une masse de choses supplémentaires à maîtriser, cela ne fait que charger la barque.

Que l'apprentissage du langage demande le triple de temps a des répercussions qui ne se bornent pas à l'emploi du temps et aux réservation de salles : les efforts intellectuels devront être soutenus trois fois plus longtemps, le délai entre le moment où l'on se posera une question et celui où l'on obtiendra la réponse sera trois fois plus long, etc.

En programmation procédurale, comme en programmation récursive, il est assez facile d'exposer aux étudiants un modèle de l'environnement et un modèle de la mémoire, ce qui leur donne accès à l'intuition de l'endroit où sont les données qu'ils créent et qu'ils manipulent. Avec le modèle objet c'est beaucoup plus difficile, à la limite il faudrait leur apprendre en plus UML, mais je crains que cela ne fasse exploser le budget horaire, et il n'est d'ailleurs rien moins que certain qu'UML donne une représentation convenable de la mémoire pour un système d'objets.

Outre ces arguments négatifs contre d'autres langages, Scheme possède en sa faveur des arguments positifs. Tous ceux qui l'ont pratiqué conviennent du fait que cette expérience leur a permis d'améliorer la qualité de leur programmation, y compris en d'autres langages. Il y a à cela de bonnes raisons, qui peuvent être commodément résumées par un aphorisme de John Foderaro : Lisp (dont Scheme est une variante) est un *langage de programmation programmable*. D'ailleurs si le lecteur suit le lien ci-dessus, il y trouvera un exposé particulièrement dense et concis des caractères uniques de cette famille de langages, dont une autre idée importante, et qui rend possible la précédente : un programme Lisp est un objet du langage Lisp. Cela pourrait sembler anodin, mais évacue toute une série d'aspects arbitraires dans la syntaxe, et confère à ce que l'on écrit la dimension de la nécessité ; pour écrire en Scheme (en Lisp), il est nécessaire de savoir ce que l'on écrit, il est impossible d'entamer une boucle sans bien savoir ce que l'on va lui faire faire, juste pour commencer et « on verra bien ce que ça fait ». Il est bien sûr possible d'écrire en Scheme, comme en tout autre langage, des programmes laids, faux, ou les deux, mais il est beaucoup plus difficile d'écrire un programme incohérent.

En relisant ces dernières lignes, je m'aperçois que l'on pourrait dire la même chose d'Ada : un argument classique (et véridique) des partisans de ce langage est que s'il est souvent plus laborieux de rédiger un programme syntaxiquement correct en Ada qu'en tel ou tel autre langage, une fois ce résultat obtenu, il y a beaucoup moins d'erreurs à l'exécution. Mais justement, Ada atteint ce résultat par de tout autres moyens que Scheme, par l'empilement de couches de contrôle dont la lourdeur et la complexité font la difficulté de la programmation (et de

l'écriture de compilateurs !), là où Scheme se contente d'une idée, pourrait-on dire, celle déjà citée plus haut : un programme Lisp est un objet du langage Lisp.

---

[1] L'expressivité doit être entendue à la fois comme la puissance d'expression, la capacité à exprimer une opération complexe par un formalisme aussi simple que possible, et comme la qualité de l'expression, le fait que le texte du formalisme suggère facilement sa signification au lecteur.

[2] Sur ce sujet on lira avec profit un article de Chenglie Hu dans les Communications of the ACM, vol. 48 nÂ° 2 de février 2005, intitulé Dataless Objects Considered Harmful. Incidemment, il renvoie à un article célèbre, désormais accessible en ligne, d'Edsger Dijkstra.