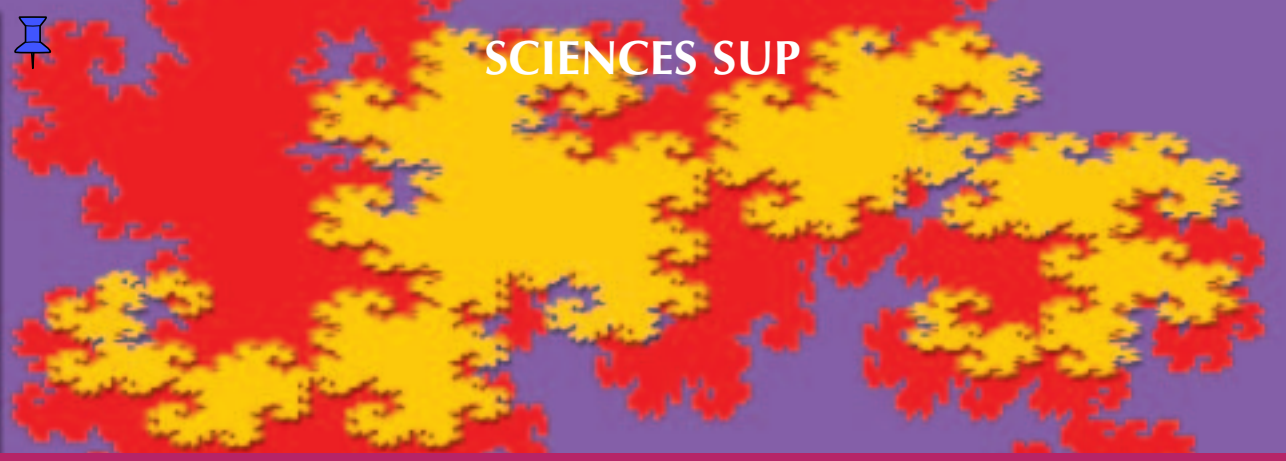


SCIENCES SUP

Cours et exercices corrigés

Licence d'informatique • Écoles d'ingénieurs

PROGRAMMATION RÉCURSIVE

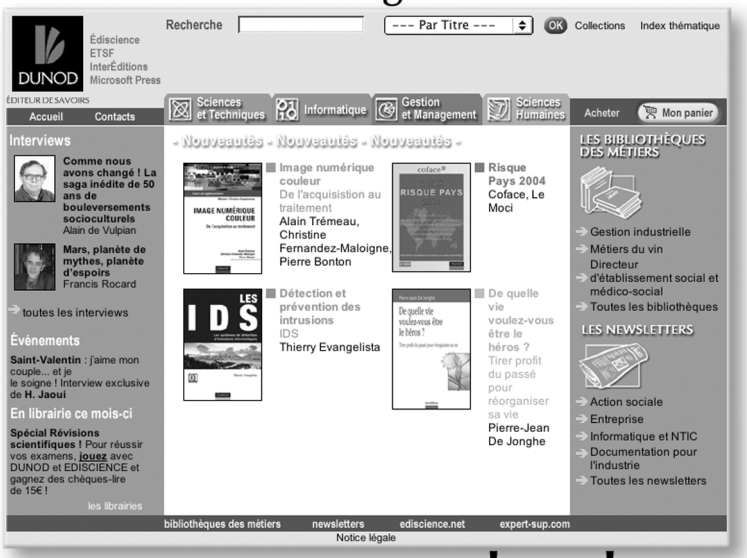
(EN SCHEME)

Anne Brygoo
Titou Durand
Maryse Pelletier
Christian Queinnec
Michèle Soria

DUNOD

PROGRAMMATION RÉCURSIVE
(EN SCHEME)

Consultez nos catalogues sur le Web



www.dunod.com

PROGRAMMATION RÉCURSIVE (EN SCHEME)

Cours et exercices corrigés

Anne Brygoo
Maître de conférences à l'UPMC

Titou Durand
Maître de conférences à l'UPMC

Maryse Pelletier
Maître de conférences à l'UPMC

Christian Queinnec
Professeur à l'UPMC

Michèle Soria
Professeur à l'UPMC

DUNOD

Illustration de couverture réalisée
à partir de l'image du dragon

ISBN 2 10 007479 2

Table des matières

Introduction

Cet ouvrage est le livre du cours intitulé « Programmation récursive et processus d'évaluation », enseigné depuis 1999 aux étudiants de première année de l'université Pierre et Marie Curie (UPMC). Il se veut une introduction à l'informatique vue comme une science, avec ses problématiques et ses idées fondamentales, mais aussi ses tours de main. Ne supposant aucune connaissance préalable en informatique, mais se fixant des objectifs ambitieux, ce livre s'adresse à différents types de public : premières années d'université scientifique, mais aussi étudiants plus avancés désirant s'initier à la programmation (écoles d'ingénieurs et écoles de commerce, licence et master d'autres disciplines...)

Le but choisi est de montrer le processus par lequel un ordinateur convertit un *texte* (le programme représentant un calcul) en une *valeur* (le résultat obtenu après avoir effectué ce calcul). Pour expliquer cette évaluation (interprétation), il est nécessaire d'utiliser un langage de programmation, de savoir s'exprimer dans ce langage (syntaxe des expressions) et de comprendre la signification (sémantique) à donner aux expressions. L'étape ultime, qui confirme la maîtrise du langage, est d'écrire un programme qui évalue les programmes, ainsi le langage s'évalue à l'aide de lui-même, *s'auto-interprète*. Peut-on espérer atteindre cette étape en ne supposant aucune connaissance préalable en informatique ? Pour prendre une image, on pourrait rêver d'apprendre la composition musicale sans passer par la maîtrise d'un instrument et de maîtriser un instrument sans passer par le solfège. Hélas, il faut commencer par faire des gammes et de même faut-il patiemment commencer par écrire des petits programmes. Cependant grâce à notre langage-instrument, Scheme, dont la syntaxe simple facilite l'apprentissage et permet de se concentrer sur les principes de la programmation récursive et l'étude des mécanismes d'évaluation, nous parvenons au but : partant des notions les plus élémentaires, nous aboutissons, à la fin du livre, à l'écriture raisonnée d'un interprète de Scheme en Scheme. Et tout au long de l'ouvrage nous aurons approfondi différents aspects fondamentaux pour la conception et l'écriture des programmes : spécification et implantation, validation et efficacité, structuration des données, etc.

1 Nos choix

Ce livre essaie de donner des fondements sur la programmation : il ne s'agit pas d'apprendre quelques règles de syntaxe pour écrire des programmes, mais d'étudier des notions générales. Le passage par un langage de programmation est incontournable, et nos programmes sont écrits en Scheme, mais cet ouvrage ne peut pas être considéré comme enseignant le langage Scheme. Il est à la fois *beaucoup moins* – nous n'exploitons qu'un tiers

des possibilités de Scheme – et *beaucoup plus* – les principes de programmation que nous avons choisis d'énoncer sont transposables dans bien des langages, et en les choisissant nous avions aussi en tête d'autres langages que Scheme.

1.1 Le langage Scheme

Le choix initial est celui du langage Scheme (dont nous n'utilisons qu'un sous-ensemble, réduit à l'application de fonction et quelques formes spéciales), avec un style de programmation purement fonctionnel. Le choix de Scheme permet d'évacuer les problèmes d'entrées/sorties et de gestion de la mémoire.

Une des grandes forces de Scheme est sa syntaxe, des plus simples et des plus régulières que l'on puisse imaginer puisqu'à base de trois marqueurs seulement – les parenthèses et l'espace –, et totalement parenthésée.

Scheme n'a quasiment aucune restriction sémantique : toutes les valeurs manipulées peuvent être passées en argument de fonction, en résultat de fonction, stockées en vecteurs, listes, etc. Le modèle d'évaluation du sous-ensemble de Scheme enseigné est le modèle par substitution. La récursivité de contrôle qu'il procure s'accorde avec la récursivité des données manipulées ce qui permet de traiter des exercices algorithmiquement intéressants sur des structures complexes.

La puissance d'expression du langage Scheme est la même que celle de tous les autres langages de programmation. Mais l'une des caractéristiques essentielles de Scheme est d'avoir éliminé tous les traits divers et variés déductibles des principes de base, ce qui permet d'en donner une description simple et courte. La norme de Scheme tient en une cinquantaine de pages, alors que les autres langages en nécessitent généralement au moins dix fois plus. Le rapport de taille est encore plus favorable à Scheme en ce qui concerne l'auto-évaluation, puisque c'est le seul langage normalisé permettant de décrire son propre interprète en une centaine de lignes.

Scheme est utilisé ici dans le contexte de DrScheme, un environnement de programmation pour Scheme développé par le *Programming Language Team* mené par Matthias Felleisen [8]. Cet environnement est interactif : on écrit son programme et l'on obtient sa valeur quasiment instantanément. Les erreurs sont clairement indiquées dans le texte. Toutes ces caractéristiques rendent rapide l'acquisition du langage.

1.2 Les points importants

Nous avons voulu écrire un livre initiatique, qui donne de bonnes habitudes et constitue une couche de base solide pour les enseignements d'informatique ultérieurs, mais qui offre aussi des à-côtés alléchants pour alimenter la réflexion des plus gourmands.

Nous insistons particulièrement sur les points suivants : la spécification des fonctions, la programmation récursive, l'abstraction par rapport à la représentation des données et des programmes, le souci d'écrire des algorithmes et des programmes corrects et efficaces, ainsi que les mécanismes de construction d'un langage.

Spécification des fonctions : Scheme est un langage non typé statiquement (le typage statique vérifie, avant l'exécution, la cohérence des programmes du point de vue de leurs types). Outre la simplification de l'écriture de l'évaluateur, l'absence de typage statique a aussi l'avantage, pour le débutant, de ne pas compliquer le modèle. Cependant l'étude de la

spécification d'une fonction, et en particulier de sa signature (type et nombre des arguments, type du résultat), est primordiale pour le programmeur, qu'il veuille utiliser une fonction ou la définir.

Nous avons donc défini une grammaire des types pour notre langage, et imposé que chaque fonction soit précédée de sa signature et de sa description (sous la forme de commentaires Scheme). Cette étape nécessaire oblige à prendre un peu de recul avant de se lancer dans la programmation et permet de corriger les programmes en vérifiant « à la main » la cohérence entre l'utilisation et la définition d'une fonction.

Récursion : le sous-langage de Scheme que nous utilisons ici est purement fonctionnel, le résultat de tout programme est obtenu par composition de fonctions. Nous insistons particulièrement sur la conception et la définition de fonctions récursives, en expliquant le principe de décomposition du traitement d'une donnée en traitements sur des données plus petites et en décrivant cette décomposition par une relation de récurrence sur laquelle apparaissent les cas de base.

Le mécanisme de récursion s'applique à tous les types de données sur lesquelles on peut faire apparaître une structure récursive : les nombres entiers mais aussi les listes, vues comme constituées d'un premier élément et d'un reste qui est aussi une liste, ainsi que les arbres constitués d'une racine et de sous-arbres. Nous essayons de montrer la ligne directrice commune dans les traitements récursifs sur toutes ces données.

Barrière d'abstraction : la structuration en couches (ou niveaux) est nécessaire dans toute organisation un tant soit peu complexe : à chaque niveau on explicite les éléments sur lesquels on s'appuie et ceux que l'on réalise. La communication entre les différentes couches se fait par le biais d'interfaces, mais les couches doivent être les plus étanches possible, au sens où les modifications apportées à un niveau se répercutent le moins possible sur les autres niveaux.

Nous insistons sur ce problème en introduisant la notion de barrière d'abstraction. La barrière d'abstraction d'une structure de données est un ensemble de fonctions permettant de manipuler les données de ce type. Cette barrière est une interface qui délimite deux niveaux : le niveau « utilisateur », où l'on manipule ces données (à l'aide des fonctions de la barrière que l'on considère alors comme des primitives) et le niveau « programmeur », où l'on s'intéresse à la représentation des données et à l'implantation des fonctions de la barrière.

Correction et efficacité : il ne suffit pas d'écrire des programmes, encore faut-il qu'ils soient corrects, c'est-à-dire que leur résultat soit conforme à leur spécification, et efficaces, c'est-à-dire que leur temps d'exécution ne soit pas prohibitif.

Étant donnée la spécification d'une fonction, on demande de fabriquer en premier lieu, avant même d'écrire la définition de la fonction, une série de tests, confrontant expressions et valeurs attendues et prenant en compte tous les cas de figure possibles, qui serviront à vérifier le comportement de la fonction. À défaut d'une preuve formelle, on aura alors la conviction que la fonction semble correcte. Et le fait de préparer ces tests à l'avance permet non seulement d'aider à la compréhension de la spécification, mais aussi d'éviter d'être influencé par le code que l'on écrit pour la définition.

Souvent nous montrons comment améliorer l'efficacité des fonctions : au niveau du code, par exemple en nommant les résultats utilisés à plusieurs reprises, pour éviter de les recalculer ; au niveau des méthodes, par exemple en mettant en évidence le gain significatif apporté par les algorithmes dichotomiques par rapport aux algorithmes linéaires.

Syntaxe et sémantique : l'objectif ultime de ce livre est de montrer les mécanismes de construction d'un langage, à partir du formalisme qui définit les expressions syntaxiquement correctes, jusqu'au programme qui interprète ces expressions pour leur donner un sens, c'est-à-dire une valeur. Au centre de ce processus se trouvent les grammaires, qui explicitent le lien entre la structure linéaire du texte du programme et la structure arborescente profonde sur laquelle se calcule la valeur.

Nous consacrons un long développement aux structures arborescentes et aux grammaires. La grammaire de Scheme et celle que nous avons définie pour les types sont utilisées dès le début du livre pour écrire les spécifications et les définitions de tous les programmes ; et nous présentons en annexe la « carte de référence », une aide précieuse qui doit toujours servir de guide dans l'écriture des programmes. Les arbres sont aussi étudiés en tant que types de données, dont les applications informatiques sont multiples, de la recherche d'information à la représentation des images, en passant par la structuration des systèmes de fichiers.

2 Comment lire ce livre

Ce livre est essentiellement composé de deux parties. Les chapitres 1 à 5 sont consacrés à l'étude du langage Scheme et de quelques principes de programmation. Les chapitres 6 à 10 abordent de nouveaux aspects comme les structures de données et les grammaires qui mènent finalement à l'interprète de Scheme en Scheme.

Chaque chapitre présente tout d'abord un développement de cours, émaillé d'exemples, présentant les notions à étudier et éventuellement des avancées, intitulées « Pour en savoir plus ». La partie cours est suivie d'un certain nombre d'exercices d'application et de problèmes faisant appel à une réflexion plus poussée. Les exercices et les problèmes sont tous corrigés. Nous nous sommes efforcés de présenter un ensemble cohérent, qui initie aux bonnes habitudes en les pratiquant ; en programmation, encore plus qu'ailleurs, c'est par imitation que l'on apprend et que l'on acquiert son propre style.

La carte de référence, placée à la fin du livre, décrit de façon exhaustive le langage de spécifications que nous avons défini et le sous-langage de Scheme que nous utilisons, ainsi que les extensions que nous y avons apportées.

Nous détaillons ci-dessous les objectifs principaux de chaque chapitre.

Le premier chapitre, intitulé « Noyau de Scheme », introduit le langage que nous utilisons, qui se restreint à l'application de fonctions et à sept formes spéciales. Ce langage est réduit mais complet puisqu'il sera ensuite étendu uniquement par l'adjonction de structures de données. Il permet d'écrire, à ce niveau, les premiers petits programmes.

Le chapitre 2, intitulé « Art et usage des spécifications », présente un certain nombre de règles à suivre pour rendre les programmes plus lisibles par des humains. Ce sont bien sûr des ordinateurs qui exécutent les programmes, mais ce sont des humains qui les écrivent, les utilisent et les corrigent. La lisibilité des programmes est donc un gage de leur fiabilité. Ces règles de programmation ne sont pas spécifiques à l'écriture de programmes en Scheme et sont transposables à tout autre type de langage de programmation.

La programmation récursive est le cœur de ce livre. Le chapitre 3, intitulé « Récursion », introduit ce mécanisme puissant, permettant d'exprimer la répétition des opérations dans un programme. On y montre comment concevoir et définir des fonctions récursives, en s'attachant à écrire des programmes corrects et efficaces. Ce chapitre traite de la récursion sur les

nombres entiers, mais les principes qu'il énonce s'appliqueront aussi à la récursion sur les listes et les arbres.

Le chapitre 4, intitulé « Notion de liste » présente la structure de liste, fondamentale en Scheme, qui permet de représenter une séquence ordonnée de valeurs. Les listes étant des structures intrinsèquement récursives, leur traitement est donc, par nature, récursif. On approfondit donc ici le mécanisme de récursion.

La puissance des fonctions est développée au chapitre 5, intitulé « Fonctionnelles ». Les fonctions sont non seulement applicables, mais aussi composables, manipulables et stockables comme les autres valeurs. L'utilisation d'itérateurs sur les listes permet d'écrire des programmes concis et génériques. La possibilité d'abstraire sur les fonctions est un concept de plus en plus important en programmation (généricité en ADA, signature de modules en CAML et interfaces en Java ou C#).

Le chapitre 6, « Modèle par substitution », s'attache à donner une sémantique au langage. L'évaluation des expressions se fait par récriture, en substituant les arguments par leur valeur. La compréhension générale de ce processus nécessite d'expliciter les notions essentielles d'environnement et de portée.

Le chapitre 7, « Structuration des données », donne une présentation des différentes structures de données de Scheme, essentiellement les vecteurs et les S-expressions. Mais il s'attache aussi à montrer comment construire ses propres structures de données, en réalisant une interface permettant de séparer la manipulation des objets de leur implantation.

Les arbres sont présentés au chapitre 8, intitulé « Structures arborescentes ». Les arbres sont non seulement une structure de données omniprésente en informatique, pour décrire tout ensemble hiérarchisé d'informations, mais ils expriment aussi l'essence même de la récursion : la récursion prend sa forme la plus générale lorsqu'elle s'applique aux structures arborescentes et inversement toute récursion est représentable par un arbre.

Le chapitre 9, « Grammaire », montre le lien entre le texte d'un programme et sa structure arborescente sous-jacente à partir de laquelle il devient manipulable, par exemple pour calculer sa valeur. Une grammaire est la spécification formelle d'un langage, et en informatique tout est langage. Cependant la grammaire ne donne qu'une vision statique du langage. Le but de tout formalisme informatique est de devenir exécutable, l'idéal étant de dériver automatiquement une implantation à partir de spécifications formelles.

Le dernier chapitre, intitulé « Évaluation », montre toutes les étapes de la construction de l'évaluateur, programme écrit en Scheme, qui permet d'évaluer les programmes Scheme, et donc de s'auto-évaluer. C'est le point d'aboutissement de l'ouvrage, une expérience que certains qualifient de « mystique ». Plutôt qu'une cerise c'est... la pastèque sur le gâteau, bien lourde mais pleine d'eau et rafraîchissante, même s'il y a quelques pépins à digérer... ou à recracher.

3 Comment utiliser ce livre

Les dépendances entre les différents chapitres suivent essentiellement l'ordre de leur présentation : la difficulté croît, du début jusqu'à la fin du livre, mais certains chapitres gagnent à être lus plusieurs fois, à différentes étapes de la progression. C'est le cas tout particulièrement pour « Art et usage des spécifications » (chapitre 2), « Modèle par substitution » (chapitre 6) et « Structuration des données » (chapitre 7).

Le dernier chapitre a un statut particulier : on peut le lire sans essayer d'en comprendre les moindres détails ou le considérer comme le corrigé d'un problème dont l'énoncé tient en une seule petite phrase : « écrire un évaluateur pour Scheme ». Travailler ce problème mobilise et consolide toutes les connaissances et compétences acquises lors des chapitres précédents.

Notre objectif, en écrivant ce livre, est de faire passer des concepts, mais aussi de communiquer des méthodes, un style de programmation et de transmettre des savoir-faire et des tours de main. Voici quelques conseils pour en tirer profit.

Il faut commencer par lire le cours et tester sa compréhension en implantant les exemples (toujours avec la carte de référence sous les yeux). Cette attitude active, apprendre en *faisant*, est illustrée par le fameux proverbe chinois que nous citons souvent à nos étudiants :

On écoute, on oublie
On écrit, on se souvient
On fait et on sait.

C'est un énorme avantage d'avoir un système pour expérimenter : la machine sanctionne le travail, elle met en évidence les erreurs et fait surgir des questions...

Ensuite il faut chercher à résoudre les exercices sans regarder la solution (les corrigés sont reportés en fin de chapitre à cette intention), implanter sa propre méthode et tester qu'elle résout bien le problème posé. Et c'est alors seulement qu'il faut aller regarder la solution proposée dans le livre, pour comparer l'algorithme choisi, l'implantation, la complexité, s'imprégner du style de programmation.

► Logiciels d'accompagnement

Dans notre enseignement à l'UPMC, Scheme est utilisé avec l'environnement de programmation DrScheme [8], auquel nous avons rajouté, au fil des ans, une bibliothèque de fonctions supplémentaires, nommée `miastools`. Pour ce qui est du graphique, cette bibliothèque s'appuie sur la bibliothèque `fungraph` de Max Hailperin, Barbara Kaiser et Karl Knight [9].

Cet environnement est disponible sur le site[1] de l'enseignement de « Programmation récursive » de la licence d'informatique de l'UPMC. Ce site est utilisé par nos étudiants et peut servir de compagnon complémentaire dans l'étude de ce livre. Il contient tout un environnement logiciel que l'on peut télécharger (et que nous distribuons chaque année à nos étudiants sous forme de cédérom) pour s'entraîner à la pratique de Scheme.

Dans cet environnement, le lecteur travaille interactivement, non seulement en testant ses propres programmes, mais aussi en répondant aux quelques centaines d'exercices que nous y proposons : près de 400 questions d'auto-évaluations sur les notions de base et une soixantaine d'exercices de programmation, totalisant près de 250 définitions de fonctions. (Précisons que les exemples et exercices de ce livre sont, en très grande majorité, différents de ceux du logiciel.)

Ce logiciel d'aide à l'apprentissage repose lui-même sur l'existence d'un interprète pour le langage étudié, qui permet de fournir des évaluations sur les réponses proposées par l'apprenant. Le lecteur intéressé par une description plus précise pourra se reporter à [4] et [6].

[1] `http://www.infop6.jussieu.fr/cederoms/li101/`

4 Comment enseigner ce livre

Ce livre est le résultat de l'enseignement dispensé depuis 1999, aux huit à neuf cents étudiants de première année de DEUG Mathématiques et Informatique Appliquées aux Sciences (MIAS) à l'UPMC ainsi qu'en seconde année du DEUG Science Pour l'Ingénieur (SPI). C'est aussi l'enseignement dispensé à partir de la rentrée 2004, en premier semestre de première année de la nouvelle licence d'informatique, mise en place dans le cadre de la réforme européenne de l'enseignement supérieur dite du LMD (Licence-Master-Doctorat).

C'est une population très hétérogène au niveau des connaissances en informatique et du goût pour cette discipline. Un tiers seulement se destine à poursuivre des études en informatique, les autres choisissant les mathématiques, la mécanique, l'électronique ou la physique. Il s'agit donc de faire une introduction à l'informatique, qui serve de culture générale et technique aux non spécialistes et constitue aussi les fondements de la formation des futurs informaticiens.

L'enseignement se déroule traditionnellement sur un semestre (50h, réparties à peu près également en heures de cours, de travaux dirigés et de travaux sur machine encadrés) ; nous avons aussi expérimenté un type d'enseignement « semi-présentiel », avec suivi des étudiants à distance et rencontre une fois par semaine [4, 6].

Depuis le début, notre idée directrice a été de présenter le processus d'évaluation et d'utiliser pour cela le langage Scheme. Les premières annéees nous consacrions la moitié des douze semaines du semestre à étudier les bases du langage et la récursion sur les nombres et les listes, puis quatre semaines sur les barrières d'abstraction, les arbres et les grammaires, et il nous restait deux semaines pour présenter l'évaluateur. Mais nous avons pris conscience que ce projet était trop ambitieux pour un laps de temps si court ; le fossé était trop grand entre la fin de la première partie et l'étude de l'évaluateur. Petit à petit la deuxième partie s'est donc concentrée sur les structures de données et la récursion sur les arbres, ne laissant plus beaucoup de place à l'évaluateur.

Globalement, pour présenter le contenu de ce livre dans un ensemble cohérent, nous pensons donc qu'il faut de l'ordre de 80 heures, en incluant un projet final. La réalisation d'un projet, comme par exemple l'écriture d'un petit évaluateur pour un mini langage, nous semble être une conclusion nécessaire pour que l'étudiant puisse mobiliser toutes ses connaissances de façon gratifiante.

5 Remerciements

Pascal Manoury nous a accompagnés pendant plusieurs années dans cette aventure, pour la réalisation du site et les expériences d'enseignement à distance, ainsi que pour la première phase de conception de ce livre. Nous regrettons que d'autres engagements l'aient empêché de se joindre à nous pour la rédaction finale : ses suggestions, ses commentaires et ses critiques nous auront beaucoup manqué.

Nous remercions tous les collègues de l'UPMC avec lesquels nous avons enseigné depuis cinq ans. C'est leur participation qui nous a permis de dispenser cet enseignement à d'impressionnantes masses d'étudiants, et c'est aussi grâce à leurs nombreuses questions et remarques que nous avons peu à peu affiné le contenu de ce livre.

Nos étudiants de ces dernières années nous ont poussés à écrire ce livre, car nous n'avions pas de réponse à donner à leur demande de bibliographie, aucun livre existant ne correspondant à notre vision pédagogique. C'est un exercice parfois difficile de les convaincre de nous suivre dans notre démarche, mais nos efforts sont récompensés lorsque nous les voyons prendre goût à la beauté de la programmation récursive, lorsque les yeux de certains s'illuminent lors de l'« expérience mystique » où l'évaluateur de Scheme en Scheme est dévoilé et lorsqu'ils nous disent, quelques années plus tard, que cet enseignement a déterminé leur vocation d'informaticien.

Nous tenons enfin à remercier toute la hiérarchie de l'UPMC, qui nous a permis de mener à bien notre enseignement et nous a toujours soutenus dans les expériences que nous avons tentées et les divers déploiements logiciels que nous avons réalisés.

Anne Brygoo
Titou Durand
Maryse Pelletier
Christian Queinnec
Michèle Soria